

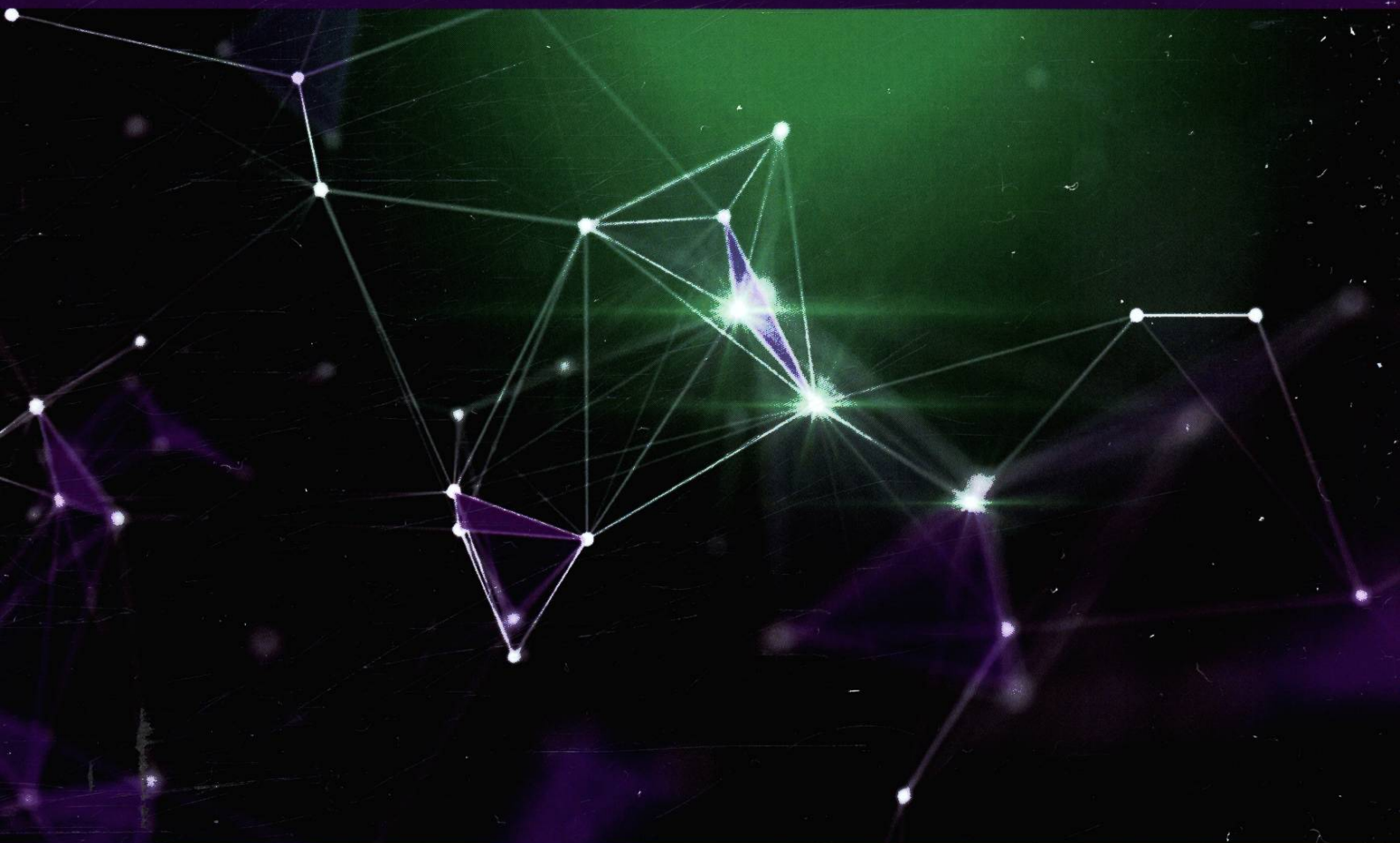


ПОЛИТЕХ
Институт компьютерных
наук и кибербезопасности

С. А. Молодяков

ПРИМЕНЕНИЕ НЕЙРОННЫХ СЕТЕЙ ДЛЯ ОБРАБОТКИ МУЛЬТИМЕДИЙНОГО КОНТЕНТА (100 ПРИМЕРОВ НА PYTHON)

Монография



ПОЛИТЕХ-ПРЕСС
Санкт-Петербургский
политехнический университет
Петра Великого

Министерство науки и высшего образования Российской Федерации

САНКТ–ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО

Институт компьютерных наук и кибербезопасности
Высшая школа программной инженерии

С.А. МОЛОДЯКОВ

**ПРИМЕНЕНИЕ НЕЙРОННЫХ СЕТЕЙ ДЛЯ
ОБРАБОТКИ МУЛЬТИМЕДИЙНОГО
КОНТЕНТА**

(100 примеров на Python)

Монография

Санкт-Петербург
Издательство Политехнического университета
2025

УДК 004.932

Р е ц е н з е н т ы:

Доктор технических наук, профессор высшей школы
программной инженерии СПбПУ *С. М. Устинов*

Доктор физико-математических наук, профессор высшей школы
прикладной физики и космических технологий СПбПУ *А. П. Лавров*

Молодяков С.А. **Применение нейронных сетей для обработки мультимедийного контента (100 примеров на Python)** : монография/ Молодяков С.А. - СПб. : ПОЛИТЕХ-ПРЕСС, 2025.- 573 с.

В монографии излагаются вопросы применения нейронных сетей для обработки мультимедийного контента. В представленных алгоритмах обработки и редактирования данных кроме нейронных сетей используются функции библиотек OpenMP, FFmpeg и других. Приведены 100 примеров полностью рабочих программ, написанных на языке Python. Примеры разделены по главам: алгоритмы преобразования файлов картинок, алгоритмы преобразования видео файлов, алгоритмы работы с текстом и аудио файлами, а также приводятся примеры использования алгоритмов обработки и преобразования медийных файлов в различных прикладных задачах. Представленные примеры являются программами, которые написаны студентами высшей школы программной инженерии Санкт-Петербургского политехнического университета.

Монография предназначена для начинающих инженеров и специалистов в области проектирования и применения видеосистем для обработки изображений и звука. Примеры программ позволяют познакомиться с возможностями по применению нейронных сетей, упростить изучение алгоритмов преобразования медийных данных. Монография может быть полезна студентам, проходящим подготовку по направлениям 09.03.01 «Информатика и вычислительная техника», 09.03.04 «Программная инженерия».

Печатается по решению
Совета по издательской деятельности Ученого совета
Санкт-Петербургского политехнического университета Петра Великого

ISBN

© Молодяков С.А. 2025
© Санкт-Петербургский политехнический
университет, 2025

The Ministry of Science and Higher Education of the Russian Federation

Peter the Great St.Petersburg Polytechnic University

Institute of Computer Science and Cybersecurity
Higher School of Software Engineering

S.A.MOLODYAKOV

**APPLICATION OF NEURAL NETWORKS FOR
MULTIMEDIA CONTENT PROCESSING**

(100 Python examples)

St.Petersburg
Polytechnic University Publishing House
2025

R e v i e w e r s :

Doctor of Technical Sciences, Professor
of the Higher School of Software Engineering SPbPU *S.M. Ustinov*
Doctor of Physical and Mathematical Sciences, Professor
of the Higher School of Applied Physics and Space Technologies SPbPU *A.P. Lavrov*

Molodyakov S.A. Applications of neural networks for multimedia processing (100 Python examples)./ Molodyakov S.A. - St.Petersburg: Polytech-PRESS, 2025.- 573 p.

The monograph discusses the application of neural networks for multimedia content processing. In addition to neural networks, the presented algorithms for data processing and editing use functions from libraries such as OpenMP, FFmpeg, and others. It includes 100 fully working program examples written in Python. The examples are divided into chapters: image file transformation algorithms, video file transformation algorithms, text and audio file manipulation algorithms, as well as examples of using media file processing and transformation algorithms in various applied tasks. These examples are programs written by students of the Higher School of Software Engineering at Peter the Great St. Petersburg Polytechnic University.

The monograph is intended for beginner engineers and specialists in the field of designing and applying video systems for image and sound processing. The program examples allow readers to explore the capabilities of applying neural networks and simplify the study of media data transformation algorithms. This monograph can be useful for students studying in the following specialties: 09.03.01 "Informatics and Computer Science," 09.03.04 "Software Engineering."

© Molodyakov S.A. 2025

© Peter the Great St.Petersburg Polytechnic University, 2025

ISBN

Оглавление

Введение	12
Глава 1. Нейронные сети для обработки мультимедийной информации ...	15
1.1. Нейронные сети для работы с видео и изображениями	15
1.2. Нейронные сети для работы с аудио	22
Глава 2. Алгоритмы работы с изображениями	28
2.1. Преобразование черно-белой фотографии в цветную. Deoldify.....	28
2.2. Коррекция свойств фотографии. PyTorch и Torchvision	30
2.3. Удаление фона с изображения. rRemBG.....	38
2.4. Сегментация изображения. U-Net.....	40
2.5. Сегментация изображения. SAM	49
2.6. Сегментация изображений для пяти классов объектов на основе набора данных MS-COCO	51
2.7. Сегментация одежды на изображениях. SegNet.....	66
2.8. Сегментация изображения для двух классов. U-Net, OXFORD-III Pet.....	76
2.9. Определение 10 разных классов на фотографиях. TensorFlow и CIFAR-10	82
2.10. Определение суммы монет на фото с использованием нейросети Yolov8	88
2.11. Определение элементов на изображениях. Faster R-CNN.....	91
2.12. Определение рукописных цифр на изображении. TensorFlow MNIST	94
2.13. Определение пола и возраста человека по фото. Gender_deploy.prototxt.	100
2.14. 3D-сцена из нескольких фото Luma AI	103
2.15. Повышение разрешения с использованием модели EDSR	106
2.16. Распознавание объектов на изображениях. MobileNet.....	112
2.17. Распознавание позы человека. Movenet	116
2.18. Распознавание буквы на изображении. TensorFlow.....	121
2.19. Распознавание позиции текста. EAST OpenCV.....	125
2.20. Распознавание размеров на схеме товара. Tesseract	130
2.21. Распознавание текстовой информации на скане изображения YOLOv8 .	137
2.22. Замена части изображения по его маске и текстовому запросу. Stable Diffusion	143
2.23. Распознавание марок автомобилей по фотографии. ResNet50.....	147
2.24. Детектирование отпечатков пальцев. TensorFlow.....	152
2.25. Распознавание объектов. Xception.....	159
2.26. Классификация статических картинок. TensorFlow	162
2.27. Поиск и кластеризация объектов на фото с использованием конвертеров фото в вектор YOLO.....	166
2.28. Распознавание рукописных слов. TensorFlow	173
2.29. Сегментация изображений и прозрачный фон. DeepLabV3	181
2.30. Определения состояния неба по фотографии. TensorFlow	188
2.31. Детальное описание статических изображений в архитектуре CNN+LSTM. TensorFlow	192
2.32. Распознавание эмоций, возраста и пола на изображении. DeepFace	209

Глава 3. Алгоритмы распознавания и преобразования видео.....	214
3.1. Классификация эмоций на видео. TensorFlow.....	214
3.2. Классификация объектов на видео. MobileNetSSD	222
3.3. Трекинг с использованием нейросети Yolov8	224
3.4. Обнаружения черт лица в реальном времени. Dlib.....	229
3.5. Запись распознанного текста на видео. SpeechRecognition.....	232
3.6. Управление громкостью с помощью жестов рук. MediaPipe	239
3.7. Виртуальный гардероб. Dlib.....	243
3.8. Трекинг объектов и подсчет при пересечении линии. YOLOv9	247
3.9. Поиск человека на видео по фотографии лица. Face_recognition.....	250
3.10. Создание видео по картинке. Replicate и Stable Video Diffusion	254
3.11. Распознавание жестов. MediaPipe.....	257
3.12. Распознавания объектов с веб-камеры. ResNet50	261
3.13. Распознавание лиц с веб-камеры. Face_recognition	266
3.14. Определение объектов на видео. YOLO	270
3.15. Распознавание цифр при использовании языка жестов. TensorFlow Sequential.....	277
3.16. Распознавание цвета с веб-камеры. TensorFlow	279
3.17. Добавление объектов на видео. MTCNN	285
3.18. Распознавание текста с видеокамеры. Tesseract.....	289
3.19. Приложение для обработки изображений с видеокамеры. Keras.....	293
3.20. Обнаружение ям на дорогах в режиме реального времени. YOLOv8	301
Глава 4. Алгоритмы работы с текстом и аудио	307
4.1. Анализ эмоциональной окраски аудио. Transformers и SpeechRecognition	307
4.2. Разделение аудиофайла на вокал и аккомпанемент. Deezer Spleeter	311
4.3. Распознавание речи. SpeechRecognition	314
4.4. Анализ тональности текста. Keras TensorFlow.....	319
4.5. Поиск похожих товаров на основе текстовых описаний. Universal Sentence Encoder	325
4.6. Генерация текста с использованием GRU.....	329
4.7. Извлечение из текста ответов на вопросы. YandexGPT	340
4.8. Преобразование текста в речь с возможностью перевода исходного текста. Eden AI, GTTS	345
4.9. Предсказание популярности статьи по ее содержанию. TensorFlow	346
4.10. Распознавание речи на русском языке с переводом на английский язык. Speech Recognition	352
4.11. Анализ эмоциональной окраски и субъективности текста. TextBlob	354
4.12. Распознавание речи в видео и преобразование ее в текст. SpeechRecognition	358
4.13. Генерация музыки и звуков по описанию. MusicGen.....	360
4.14. Вербализация изображения. Yolov7 и RuGPT3.....	362
4.15. Конвертер римских чисел. Tesseract.....	367
4.16. Генерация текстовых описаний изображений. CLIP	370

4.17. Генерация и синтез речи. GPT.....	373
4.18. Аннотирование текста с помощью T5Tokenizer	376
4.19. Автогенерация комментариев к коду с использованием CodeBERT.....	380
4.20. Изменение стиля написания текста. YandexGPT	393
4.21. Классификация отзывов на основе их текстовых описаний. Universal Sentence Encoder.....	398
4.22. Рекомендации книг пользователям. Scikit-Learn	402
Глава 5. Примеры использования нейронных сетей	406
5.1. Управление мышкой компьютера жестами. Mediarpipe.....	406
5.2. Запуск программ на компьютере звуковыми командами. SpeechRecognition	409
5.3. Определение дорожных знаков по изображению. TensorFlow.....	412
5.4. Сегментация легких на рентгеновских снимках грудной клетки. U-Net ...	417
5.5. Определение пневмонии по рентгеновским снимкам грудной клетки. Xception.....	431
5.6. Игра «Rock-paper-scissor». Tensorflow.....	445
5.8. Определение счета игральные карты в реальном времени с использованием модели Yolov8.....	449
5.9. Распознавание шахматных фигур и анализ позиции. YOLO.....	453
5.10. Генерация музыки на основе текстового запроса. MusicGen	457
5.11. Распознавание свежей клубники. TensorFlow	461
5.12. Возрастная трансформация лица на фотографии. SAM.....	468
Глава 6. Примеры telegram-ботов с использованием нейронных сетей	476
6.1. Распознавание рукописных цифр на фото. TensorFlow.....	476
6.2. Детектирование дорожных знаков. Keras	480
6.3. Распознавание музыкальных треков в голосовых сообщениях. Yandex SpeechKit.....	495
6.4. Создание аудиофайла по тексту. Silero	501
6.5. Повышение разрешения изображения. ESGRAN	508
6.6. Выполнение арифметических выражений, введенных с изображений. EasyOCR	515
6.7. Генерация изображения из текста. DALL-E	518
6.8. Наложение изображения на фон в видеофайле/ распознавание номеров ..	520
6.9. Генерация текста программы. DeepSeek Coder	530
6.10. Распознавание английского текста на картинке. Pytesseract	537
6.11. Определение возраста и пола человека на изображении или в видеопотоке. Caffe.....	540
6.12. Перевод аудио файлов и голосовых сообщений в текст. Whisper	546
6.13. Озвучивание текста. Bark	551
6.14. Переводчик аудио с применением ресурсов Yandex Cloud	554
Заключение	563
Список литературы.....	564

Студенты, которые принимали участие в написании содержания
параграфов и разработке примеров программ

Параграф	Название	Автор
2.1.	Преобразование черно-белой фотографии в цветную. Deoldify	Недорезов Данил
2.2.	Коррекция свойств фотографии. PyTorch и Torchvision	Попов Елисей
2.3.	Удаление фона с изображения. rRemBG	Ганиуллин Родион
2.4.	Сегментация изображения. U-Net ()	Шкуратов Николай
2.5.	Сегментация изображения. SAM	Шестаков Павел
2.6.	Сегментация изображений для пяти классов объектов на основе набора данных MS-COCO	Королев Денис
2.7.	Сегментация одежды на изображениях. SegNet	Парулава Анна
2.8.	Сегментация изображения для двух классов. U-Net, OXFORD-III Pet	Тихонов Даниил
2.9.	Определение 10 разных классов на фотографиях. TensorFlow и CIFAR-10	Белов Николай
2.10.	Определение суммы монет на фото с использованием нейросети YOLOv8	Сопрачев Андрей
2.11.	Определение элементов на изображениях. Faster R-CNN	Кудрявцев Александр
2.12.	Определение рукописных цифр на изображении. TensorFlow MNIST	Мамонова Алина
2.13.	Определение пола и возраста человека по фото. Gender_deploy.prototxt	Струк Александра
2.14.	3D-сцена из нескольких фото Luma AI	Горохов Алексей
2.15.	Повышение разрешения с использованием модели EDSR	Чернецов Александр
2.16.	Распознавание объектов на изображениях. MobileNet	Стародубцева Катарина
2.17.	Распознавание позы человека. Movenet	Баранов Константин
2.18.	Распознавание буквы на изображении. TensorFlow	Малая Виолетта
2.19.	Распознавание позиции текста. EAST OpenCV	Трофимов Григорий
2.20.	Распознавание размеров на схеме товара. Tesseract	Сластникова Анна
2.21.	Распознавание текстовой информации на скане изображения YOLOv8	Машкин Андрей
2.22.	Замена части изображения по его маске и текстовому запросу. Stable Diffusion	Жукова Лидия
2.23.	Распознавание марок автомобилей по фотографии. ResNet50	Кузнецова Елизавета
2.24.	Детектирование отпечатков пальцев. TensorFlow	Цаликов Георгий
2.25.	Распознавание объектов. Xception	Чемодуров Артем
2.26.	Классификация статических картинок. TensorFlow	Ларионенко Артём
2.27.	Поиск и кластеризация объектов на фото с использованием конвертеров фото в вектор YOLO	Овсянников Кирилл
2.28.	Распознавание рукописных слов. TensorFlow	Башлачева Елизавета
2.29.	Сегментация изображений и прозрачный фон.	Авдеева Наталья

	DeepLabV3	
2.30.	Определения состояния неба по фотографии. TensorFlow	Анисимова Алина
2.31.	Детальное описание статических изображений в архитектуре CNN+LSTM. TensorFlow	Костин Андрей
2.32.	Распознавание эмоций, возраста и пола на изображении. DeepFace	Прощина Юлия
3.1.	Классификация эмоций на видео. TensorFlow	Пономарев Александр
3.2.	Классификация объектов на видео. MobileNetSSD	Лебедев Максим
3.3.	Трекинг с использованием нейросети Yolov8	Сысоева Ольга
3.4.	Обнаружения черт лица в реальном времени. Dlib	Конев Иван
3.5.	Запись распознанного текста на видео. SpeechRecognition	Голиков Георгий
3.6.	Управление громкостью с помощью жестов рук. MediaPipe	Ершов Степан
3.7.	Виртуальный гардероб. Dlib	Сиобко Борис
3.8.	Трекинг объектов и подсчет при пересечении линии. YOLOv9	Лонишин Максим
3.9.	Поиск человека на видео по фотографии лица. Face_recognition	Трофимов Никита
3.10.	Создание видео по картинке. Replicate и Stable Video Diffusion	Илюшкин Александр
3.11.	Распознавание жестов. MediaPipe	Смирникова Лилиана
3.12.	Распознавания объектов с веб-камеры. ResNet50	Котова Ярослава
3.13.	Распознавание лиц с веб-камеры. Face_recognition	Ветошев Данила
3.14.	Определение объектов на видео. YOLO	Васин Никита
3.15.	Распознавание цифр при использовании языка жестов. TensorFlow Sequential	Вагина Дарья
3.16.	Распознавание цвета с веб-камеры. TensorFlow	Панфилов Максим
3.17.	Добавление объектов на видео. MTCNN	Мартюшова Дарья
3.18.	Распознавание текста с видеокамеры. Tesseract	Арбаева Арина
3.19.	Приложение для обработки изображений с видеокамеры. Keras	Андреева Елизавета
3.20.	Обнаружение ям на дорогах в режиме реального времени. YOLOv8	Медведев Станислав
4.1.	Анализ эмоциональной окраски аудио. Transformers и SpeechRecognition	Власенко Александр
4.2.	Разделение аудиофайла на вокал и аккомпанемент. Deezer Spleeter	Савченко Константин
4.3.	Распознавание речи. SpeechRecognition	Потапова Мария
4.4.	Анализ тональности текста. Keras TensorFlow	Флусова Софья
4.5.	Поиск похожих товаров на основе текстовых описаний. Universal Sentence Encoder	Спиваков Михаил
4.6.	Генерация текста с использованием GRU	Спирчина Яна
4.7.	Извлечение из текста ответов на вопросы. YandexGPT	Олейникова Анастасия
4.8.	Преобразование текста в речь с возможностью перевода исходного текста. Eden AI, GTTS	Ефремов Артём
4.9.	Предсказание популярности статьи по ее содержанию. TensorFlow	Джичко Андрей

4.10.	Распознавание речи на русском языке, с переводом на английский язык. Speech Recognition	Костюченко Тимофей
4.11.	Анализ эмоциональной окраски и субъективности текста. TextBlob	Чурина Маргарита
4.12.	Распознавание речи в видео и преобразование ее в текст. SpeechRecognition	Зиятдинова Алина
4.13.	Генерация музыки и звуков по описанию. MusicGen	Месропян Артём
4.14.	Вербализация изображения. Yolov7 и RuGPT3	Клишева Алина
4.15.	Конвертер римских чисел. Tesseract	Кончев Максим
4.16.	Генерация текстовых описаний изображений. CLIP	Оруджиев Рамиль
4.17.	Генерация и синтез речи. GPT	Чулков Леонтий
4.18.	Аннотирование текста с помощью T5Tokenizer	Косминская Олеся
4.19.	Автогенерация комментариев к коду с использованием CodeBERT	Клеблеева Элина
4.20.	Изменение стиля написания текста. YandexGPT	Демин Никита
4.21.	Классификация отзывов на основе их текстовых описаний. Universal Sentence Encoder	Спиридонов Владимир
4.22.	Рекомендации книг пользователям. Scikit-Learn	Кашмак Станислав
5.1.	Управление мышкой компьютера жестами. Mediapipe	Пономарев Александр
5.2.	Запуск программ на компьютере звуковыми командами. SpeechRecognition	Пономарев Александр
5.3.	Определение дорожных знаков по изображению. TensorFlow	Казаков Вячеслав
5.4.	Сегментация легких на рентгеновских снимках грудной клетки. U-Net	Афанасьев Алексей
5.5.	Определение пневмонии по рентгеновским снимкам грудной клетки. Xception	Коба Алексей
5.6.	Игра «Rock-paper-scissor». Tensorflow	Мишанина Елена
5.8.	Определение счета игральных карт в реальном времени с использованием модели Yolov8	Гуляев Илья
5.9.	Распознавание шахматных фигур и анализ позиции. YOLO	Хрущев Никита
5.10.	Генерация музыки на основе текстового запроса. MusicGen	Кулагин Илья
5.11.	Распознавание свежей клубники. TensorFlow	Митрошин Даниил
5.12.	Возрастная трансформация лица на фотографии. SAM	Коробицына Олеся
6.1.	Распознавание рукописных цифр на фото. TensorFlow	Пьянков Даниил
6.2.	Детектирование дорожных знаков. Keras	Губенко Надежда
6.3.	Распознавание музыкальных треков в голосовых сообщениях. Yandex SpeechKit	Якшиева Лейли
6.4.	Создание аудиофайла по тексту. Silero	Чирикин Даниил
6.5.	Повышение разрешения изображения. ESRGAN	Антон Луценко
6.6.	Выполнение арифметических выражений, введенных с изображений. EasyOCR	Матюшенко Артур
6.7.	Генерация изображения из текста. DALL-E	Оберемок Илья
6.8.	Наложение изображения на фон в видеофайле/распознавание номеров	Томилин Илья
6.9.	Генерация текста программы. DeepSeek Coder	Курташ Никита
6.10.	Распознавание английского текста на картинке.	Набережнов Денис

	Pytesseract	
6.11.	Определение возраста и пола человека на изображении или в видеопотоке. Caffe	Климков Артем
6.12.	Перевод аудио файлов и голосовых сообщений в текст. Whisper	Поздняков Сергей
6.13.	Озвучивание текста. Bark	Крючкова Кира
6.14.	Переводчик аудио с применением ресурсов Yandex Cloud	Поздняков Сергей

Введение.

В настоящее время интенсивно развиваются все направления применения мультимедиа, создаются все более производительные и функционально насыщенные системы. Такое развитие базируется на создании высокопроизводительных узлов вычислительной техники. Особенно следует выделить развитие мобильных устройств.

На базе имеющихся новых аппаратных возможностей разрабатываются и применяются и новые алгоритмы. Алгоритмы, построенные с применением нейронных сетей, показывают результаты, значительно превосходящие аналогичные у классических алгоритмов. Это касается областей компьютерного зрения, классификации объектов, распознавания речи и других.

В монографии излагаются вопросы применения нейронных сетей для обработки статических изображений, видео информации, работы с текстом и аудио. Для каждого применения нейронной сети приводятся алгоритм и программа его реализующая. Программы написаны на языке Python. Этот язык один из самых распространенных, на котором пишут прикладные мультимедийные программы. Python бесплатен имеет открытый исходный код. Он имеет хорошую поддержку со стороны открытого сообщества. Известно большое количество учебников по работе на языке Python, а также использование его библиотек [1-6].

В большинстве программ кроме нейронных сетей используются библиотеки программ для работы с мультимедийными данными. Основными библиотеками являются библиотека компьютерного зрения OpenCV и библиотека по преобразованию медийной информации FFmpeg

[7-13]. К перечисленным можно добавить библиотеку Pillow (Python Imaging Library), которая позволяет работать с изображениями [14].

В монографии не ставится задача по подробному изучению нейронных сетей и глубокого обучения. Это отдельное направление в компьютерных знаниях. Применение нейронных сетей рассматривается как задача применения функций программных библиотек. Сложность такой задачи связана с большим количеством моделей нейронных сетей, с недостаточным описанием технологий использования и малым количеством доступных примеров. Основной целью работы является представление спектра возможностей по обработке данных, которые предоставляют нейронные сети.

Монография является четвертой книгой из цикла «примеры на Python». Первыми книгами являются: «Применение функций OpenCV в компьютерном зрении (60 примеров на Python)», «Применение функций FFmpeg в мультимедийных приложениях (100 примеров на Python)», «Алгоритмы работы с мультимедийными данными в telegram-боте (100 примеров на Python)» [15-17]. В книгах приводятся примеры алгоритмов и программ, которые позволяют познакомиться с возможностями по обработке и преобразованию мультимедийных данных, упростить изучение функций библиотек. Представленные примеры являются программами, которые написаны студентами высшей школы программной инженерии Санкт-Петербургского политехнического университета.

Знакомство с технологиями и опробование представленных программ позволяют получить ценный опыт работы с нейронными сетями. Полученный опыт может быть использован при разработке более сложных систем различного назначения. Следует выделить медицину, в которой особенно широко стали использовать нейронные сети, в том

числе связанные с обработкой изображений. Известны работы по созданию навигационной системы на основе технологии дополненной реальности в челюстно-лицевой хирургии [18], системы автозаполнения протокола осмотра врачом [19], определения COVID-19 на КТ-изображениях [20-22], поиска и оценки злокачественности легочных узелков на КТ-изображениях легких [23-25], объяснения на естественном языке результатов диагностики [26] и др.

Глава 1. Нейронные сети для обработки мультимедийной информации

1.1. Нейронные сети для работы с видео и изображениями

В современном мире визуальные данные играют ключевую роль в самых разных сферах – от социальных сетей и развлечений до медицины, робототехники и беспилотных транспортных средств. Однако обработка и анализ изображений и видео представляют серьезную задачу для компьютеров из-за их сложности и многообразия.

Именно здесь на помощь приходят нейронные сети - мощные инструменты машинного обучения. Благодаря своей способности распознавать сложные закономерности в данных, нейронные сети открывают новые горизонты в области компьютерного зрения и обработки визуальной информации. Нейронные сети решают такие задачи, как обнаружение объектов, семантическая и экземплярная сегментация, генерация изображений, перенос стилей, классификация изображений и видео, извлечение признаков и многое другое.

Для обработки изображений применяются различные архитектуры нейронных сетей, каждая из которых оптимизирована под конкретные задачи. Перечислим основные типы нейронных сетей.

1. Свёрточные нейронные сети (CNN – Convolutional Neural Networks). Это наиболее популярный тип нейронной сети для работы с изображениями. CNN используют свёртки (конволюции), чтобы выделять признаки изображения на разных уровнях абстракции. Например, сначала сеть может находить края объектов, затем формы, а потом уже целые объекты. CNN применяются для задач классификации изображений, сегментации, детекции объектов и других задач компьютерного зрения.

2. Рекуррентные нейронные сети (RNN – Recurrent Neural Networks).

RNN обычно применяются для последовательностей данных, но могут использоваться и для обработки изображений, особенно когда нужно учитывать контекст соседних пикселей или обрабатывать видео. Например, LSTM (Long Short-Term Memory) и GRU (Gated Recurrent Unit) могут применяться для анализа временных рядов, связанных с изображениями, таких как видео.

3. Генеративно-сопоставительные сети (GAN – Generative Adversarial Networks).

GAN состоят из двух частей: генератора и дискриминатора. Генератор создаёт новые изображения, а дискриминатор пытается отличить реальные изображения от созданных генератором. Используются для создания реалистичных изображений, улучшения качества изображений, синтеза новых данных и стилизации изображений.

4. Автокодировщики (Autoencoders).

Автокодировщик состоит из двух частей: кодера, который сжимает изображение до меньшего представления, и декодера, который восстанавливает исходное изображение из сжатого представления. Применяются для сжатия данных, денойзинга изображений, восстановления поврежденных изображений и генерации новых изображений.

5. Трансформеры (Transformers).

Трансформеры изначально были разработаны для обработки текста, но их также успешно применяют для обработки изображений. Они работают с вниманием к различным частям изображения, что позволяет лучше анализировать сложные сцены. Примером является модель Vision Transformer (ViT), которая использует трансформеры для классификации изображений.

6. Капсульные нейронные сети (Capsule Networks).

Капсульные сети пытаются улучшить обработку пространственной информации в

изображениях. В отличие от обычных CNN, они сохраняют информацию о положении признаков относительно друг друга. Эти сети разрабатывались для решения проблем, связанных с инвариантностью к повороту и масштабу, что делает их полезными для распознавания объектов в сложных условиях.

7. Резидуальные нейронные сети (ResNets – Residual Networks). ResNet – это разновидность CNN, где добавлены так называемые «резидуальные связи», позволяющие передавать информацию через слои без изменения. Это помогает избежать проблемы затухания градиента при обучении глубоких моделей. Используется для классификации изображений, а также для задач, требующих глубокой архитектуры модели.

8. Сегментирующие нейронные сети (Segmentation Networks). К таким сетям относятся U-Net, Mask R-CNN и другие архитектуры, предназначенные для сегментации изображений. Эти сети позволяют разделить изображение на сегменты, соответствующие разным объектам или областям. Применяются для медицинских изображений, автономных транспортных средств, робототехники и других областей, где требуется точное разделение объектов на изображении.

9. Деконволюционные нейронные сети (Deconvolutional Neural Networks). Деконволюция используется для обратного процесса свертки, то есть увеличения разрешения изображения. Такие сети часто встречаются в задачах реконструкции изображений, суперсэмплинга и сегментации. Пример использования: создание высококачественных изображений из низкокачественного ввода.

10. Полносвязные нейронные сети (Fully Connected Neural Networks). Хотя полносвязные сети редко используются непосредственно для

обработки изображений из-за высокой вычислительной сложности, они иногда включаются в архитектуру других моделей, таких как CNN, для финальной классификации. Полносвязный слой может находиться после сверточных слоев для преобразования двумерных данных в одномерный вектор перед классификацией.

Кроме нейронных сетей следует выделить средства для работы с ними. Прежде всего отметим низкоуровневую библиотеку для машинного обучения TensorFlow, разработанную компанией Google. Библиотека предоставляет инструменты для создания, тренировки и развертывания моделей нейронных сетей, позволяет распределять задачи между различными устройствами, такими как CPU, GPU и др. TensorFlow поддерживает широкий спектр задач, связанных с машинным обучением, включая классификацию изображений, обработку естественного языка, рекомендательные системы и многое другое. TensorFlow является проектом с открытым исходным кодом. Он стал стандартом де-факто в индустрии применения нейронных сетей.

Вокруг TensorFlow сформировалась большая экосистема инструментов и библиотек. Наиболее популярным является Keras. Keras - это высокоуровневое API для построения и обучения нейронных сетей. Он предназначен для упрощения процесса разработки моделей за счет предоставления готовых блоков и функций. Начиная с версии TensorFlow 2.0, Keras стал частью основного пакета TensorFlow. Когда вы создаете модель в Keras, она автоматически компилируется и выполняется через TensorFlow.

Ниже приведена таблица популярных нейронных сетей и библиотек функций для работы с изображениями на примере конкретных моделей. Выделены основные задачи, которые могут решать нейронные сети.

Название	Основная задача и описание
YOLO (You Only Look Once)	Распознавание объектов. Популярная архитектура для обнаружения объектов на изображениях в реальном времени. Принцип работы YOLO подразумевает ввод сразу всего изображения, которое проходит через свёрточную нейронную сеть (CNN) только один раз. Она использует единую нейросеть для сегментации, предсказания границ объектов и их классов. Имеется множество версий, на 2022 год их семь. https://www.ultralytics.com/ru/yolo
Mask R-CNN (Mask Region-based Convolutional Neural Network)	Сегментация объектов. Mask R-CNN выполняет точную сегментацию экземпляров на изображениях. Mask R-CNN позволяет выделять на фотографиях контуры («маски») разных объектов, даже если таких экземпляров несколько, они имеют различный размер и частично перекрываются. Она способна к распознаванию поз людей на изображении. Состоит из нескольких нейронных сетей. Mask R-CNN: https://arxiv.org/abs/1703.06870
U-Net	Семантическая сегментация. Полносверточная нейронная сеть, которая широко используется для задач семантической сегментации изображений, таких как сегментация медицинских изображений или сегментация спутниковых снимков. В U-Net изображение преобразуется в вектор, а затем то же самое отображение используется для его повторного преобразования в изображение. https://neurohive.io/ru/vidy-nejrosetej/u-net-image-segmentation/
SAM (Segment Anything Model)	Модель сегментации изображений, разработанная Meta AI. Эта модель может определять точное местоположение либо конкретных объектов на изображении, либо каждого объекта на изображении. SAM имеет открытый исходный код и выпущен под лицензией Apache 2.0. https://segment-anything.com/
SegNet	Семантическая сегментация. Глубокая сверточная нейронная сеть для семантической сегментации и классификации изображений. Архитектура: convolutional-encoder-decoder. Сеть использует pixelClassificationLayer для прогнозирования категориальной метки для каждого пикселя входного изображения. https://www.geeksforgeeks.org/segnet-a-deep-convolutional-encoder-decoder-architecture-for-image-segmentation/
DCGAN (Deep Convolutional GAN)	Генерация изображений. DCGAN -модификация алгоритма генеративно-сопоставительных сетей (GAN), в основе которых лежат сверточные нейронные сети. Архитектура использует

	<p>сверточные слои для генерации реалистичных изображений. Алгоритм хорошо показывает себя в извлечении признаков. https://neerc.ifmo.ru/wiki/index.php?title=Generative_Adversarial_Nets_(GAN)&mobileaction=toggle_view_desktop#DCGAN_28Deep_Convolutional_Generative_Adversarial_Nets.29</p>
Vid2Vid	<p>Перенос стиля видео. Модель для переноса стиля между видео, основанная на генеративно-состязательных сетях. Она может применять стиль одного видео к содержанию другого видео. https://civitai.com/articles/3059/vid2vid-tutorial</p>
SlowFast Networks	<p>Классификация видео. Архитектура для классификации видео, которая использует две параллельные ветви: медленную для захвата семантических деталей и быструю для захвата движения. https://arxiv.org/pdf/1812.03982</p>
Transformer for Set Prediction	<p>Множественное обнаружение объектов. Модель использует архитектуру трансформеров для обнаружения множества объектов на изображениях, включая их классы, границы и взаимное расположение. /https://arxiv.org/pdf/2011.10881</p>
ViT (Vision Transformer)	<p>Классификация изображений. Архитектура трансформеров, адаптированная для задач компьютерного зрения, таких как классификация изображений. Она достигла высоких результатов на наборах данных ImageNet. https://en.wikipedia.org/wiki/Vision_transformer</p>
I3D (Inflated 3D ConvNet)	<p>Извлечение признаков видео. 3D сверточная нейронная сеть, которая может эффективно извлекать признаки из видеопоследовательностей для различных задач, таких как классификация действий или описание видео. https://arxiv.org/pdf/1705.07750</p>
Deoldify	<p>Раскрашивание и восстановление фотографий и видео. https://deoldify.ai/ https://www.ipol.im/pub/art/2022/403/article_lr.pdf.</p>
Faster R-CNN (Faster Region-based Convolutional Neural Network)	<p>Обнаружение объектов. Это двухэтапный алгоритм. Он использует Region Proposal Network (RPN) и Convolutional Neural Networks (CNN) для идентификации и определения местоположения объектов на сложных изображениях. https://viso.ai/deep-learning/faster-r-cnn-2/</p>
Tesseract	<p>Оптическое распознавание символов. Модель, разработана компанией Google на основе технологии OCR (optical character recognition). Она обучена распознавать печатный и</p>

	рукописный текст. https://tesseract-ocr.github.io/
Stable Diffusion	Генерация изображений по тексту. Модель скрытой диффузии, своего рода глубокая генеративная нейронная сеть. Можно применять и для дорисовки и редактирования исходной картинки. https://stablediffusion.com.ru/
MobileNetSSD (MobileNet Single Shot Detector)	Обнаружение объектов. Легкая архитектура глубокой нейронной сети, разработанная для мобильных устройств. Модель обнаружения объектов с 267 слоями. Она обеспечивает вывод в реальном времени при ограничениях вычислений на таких устройствах, как смартфоны. После обучения MobileNetSSDv2 может храниться на 63 МБ. https://roboflow.com/model/mobilenet-ssd-v2
Dlib	Обнаружение лиц и ключевых точек лица. Это библиотека с разными моделями. В bz2 используется 5-точечная модель ориентиров, которая определяет уголки глаз и нижнюю часть носа. http://dlib.net/
MediaPipe	Обнаружение лиц, ключевых точек и др. Это набор библиотек и инструментов. Примеры задач: классификация изображений, встраивание изображений, стилизация лица, генерация изображений, обнаружение ориентиров позы. https://github.com/google-ai-edge/mediapipe https://ai.google.dev/edge/mediapipe/solutions/guide?hl=ru
DeepFace	Распознавание лиц и анализ эмоций. Облегченный фреймворк распознавания лиц и анализа атрибутов лица (возраст, пол, эмоции и раса) для Python. Это гибридный фреймворк распознавания лиц, оборачивающий современные модели: VGG-Face, FaceNet, OpenFace, DeepFace, DeepID, ArcFace, Dlib, SFace, GhostFaceNet. https://github.com/serengil/deepface

Данная таблица дает общее представление о различных видах нейронных сетей, используемых для решения разнообразных задач работы с изображениями и видео. Выбор архитектуры зависит от задачи обработки видео или изображений, а также от доступных ресурсов вычислительной мощности. Примеры использования представленных и других нейронных сетей приводятся в последующих главах.

1.2. Нейронные сети для работы с аудио

В современном мире аудиоданные, также как и изображения, играют все более значимую роль. От аудиокниг и подкастов до музыкальных потоковых сервисов и голосовых помощников - мы окружены звуками, которые формируют наш опыт и взаимодействие с окружающим миром. Именно поэтому обработка и анализ аудиоданных приобретают все большую актуальность, а с появлением нейронных сетей мир аудиотехнологий претерпевает стремительную трансформацию.

От точной автоматической транскрипции речи, которая позволяет преобразовать устные слова в текст, до создания реалистичных музыкальных композиций, имитирующих стиль известных композиторов – нейронные сети демонстрируют удивительные возможности в работе с аудиоданными. С каждым днем они совершенствуются, расширяя свои функциональные возможности и повышая точность в решении все более сложных задач.

Для обработки звука применяются различные нейронные сети, типы которых похожи на сети обработки изображений. Перечислим основные типы нейронных сетей для обработки звука.

1. Рекуррентные нейронные сети (RNN). Основная идея: RNN способны обрабатывать последовательности данных, такие как звуковые сигналы, благодаря тому, что они могут сохранять информацию о предыдущих входах через обратную связь. Основное применение связано с распознаванием речи, обработкой временных рядов звукового сигнала, предсказание следующего элемента последовательности (например, предсказание следующей ноты в мелодии). Выделяют модификации:

LSTM (Long Short-Term Memory) – модификация RNN, которая лучше справляется с долговременными зависимостями в

последовательностях. GRU (Gated Recurrent Unit) – упрощенная версия LSTM, которая также хорошо работает со звуком.

2. Свёрточные нейронные сети (CNN). Основная идея: CNN используют свёртки для извлечения признаков из изображений, но их также применяют к звуковым сигналам после преобразования их в спектрограмму или другой вид представления. Применяются для классификация звуков (например, определение источника звука), анализа музыкальных треков, сегментации аудиосигнала. Особенности CNN: свёрточные слои помогают извлекать локальные признаки из спектральных представлений аудио.

3. Трансформеры (Transformer Networks). Трансформеры основаны на механизме внимания, который позволяет модели фокусироваться на различных частях входного сигнала. Применяются для распознавания речи, перевода текста в речь (TTS), генерации музыки. Отсутствие рекурсивной структуры делает трансформеры быстрее при обучении и использовании.

4. Генеративно-состязательные сети (GAN). GAN состоят из двух частей: генератора и дискриминатора. Генератор создаёт новые данные, а дискриминатор пытается отличить реальные данные от синтетических. Применяются для создания новых музыкальных композиций, улучшения качества звука (например, удаление шума). Примером может служить WaveGAN – генеративная модель, созданная специально для генерации звуковых сигналов.

5. Автоэнкодеры (Autoencoders). Автоэнкодеры обучаются сжимать входные данные до компактного представления (кодировщика), а затем восстанавливают исходный сигнал из этого сжатия (декодера). Применяются для сжатия аудиофайлов, удаления шумов и артефактов,

извлечения основных характеристик звука. Примером может служить VAE (Variational Autoencoder), который используется для создания новых звуков на основе ранее услышанных.

6. Капсульные нейронные сети (Capsule Networks). Капсульные сети пытаются улучшить представление информации за счёт капсул, которые хранят не только наличие признака, но и его ориентацию и положение. Применяются для более точного распознавания сложных звуков, обнаружения объектов в звуке. Пока ещё мало применялись в обработке звука, но имеют потенциал для улучшения точности классификации и сегментации.

7. Резидуальные нейронные сети (ResNet). Как было уже сказано ResNet включают так называемые "остаточные связи", которые позволяют передавать информацию между слоями без её потери. Применяются для классификации звуков, определения эмоций в голосе. В основном применяется для визуальных задач, но может использоваться и для анализа спектра звука.

Каждый из этих типов нейронных сетей имеет свои особенности и преимущества, поэтому выбор конкретной архитектуры зависит от специфики задачи обработки звука. Например, для распознавания речи часто используются RNN/LSTM, а для анализа музыкального контента – CNN.

Ниже показана таблица популярных нейронных сетей для работы со звуком. Указаны особенности, ссылки и основные задачи, которые могут решать нейронные сети.

Название	Основная задача и описание
DeepSpeech	Распознавание речи. Открытая библиотека для распознавания речи. Она основана на глубоких нейронных сетях и предоставляет возможность преобразования аудиозаписей в

	<p>текст. https://github.com/mozilla/DeepSpeech</p>
Whisper	<p>Распознавание речи. Система автоматического распознавания речи (ASR), обученная на 680 000 часах многоязычных и многозадачных контролируемых данных, собранных из Интернета. https://openai.com/index/whisper/</p>
Conformer	<p>Распознавание речи. Сочетает в себе преимущества свёрточной и рекуррентной нейронной сети, что позволяет ей эффективно обрабатывать данные. https://www.assemblyai.com/blog/conformer-1/</p>
WaveNet	<p>Синтез речи. Генеративная модель, обученная на образцах человеческой речи. Включает в себя естественно звучащие элементы, такие как причмокивание губами и дыхание, интонации, акценты, эмоции. https://deepmind.google/technologies/wavenet/</p>
Tacotron 2	<p>Синтез речи. Нейронная сеть для синтеза речи из текста. Она преобразует текст в спектрограммы mel, которые затем используются для генерации аудио-сигнала. Модель Encoder-Attention-Decoder на основе LSTM. https://github.com/NVIDIA/tacotron2</p>
WaveGlow	<p>Синтез речи. Генеративная сеть на основе потока для синтеза речи, использующая спектрограммы и взявшая свои идеи из Glow и Wavenet. Генерирует высококачественную речь из mel-спектрограмм https://github.com/NVIDIA/waveglow</p>
VGGish	<p>Обработка аудио сигналов. Нейронная сеть Google для анализа аудио. Она выделяет ключевые характеристики из аудио, что позволяет использовать ее для классификации звука. https://github.com/tensorflow/models/tree/master/research/audioset/vggish</p>
Librosa	<p>Обработка аудио сигналов. Это библиотека Python для анализа, манипулирования и поиска музыки и аудио https://librosa.org/</p>
Jukebox	<p>Музыкальная генерация. Модель генерации музыки от OpenAI, способная создавать музыкальные композиции в разных стилях и жанрах. Она обучена на огромном массиве данных, что позволяет ей генерировать реалистичные и разнообразные треки. https://openai.com/index/jukebox/</p>

MusicLM	Музыкальная генерация. Модель генерации музыки от Google, которая позволяет создавать музыкальные композиции на основе текстовых описаний или музыкальных мелодий. Она использует глубокое обучение для создания музыки, которая соответствует заданным критериям. https://musiclm.com/
---------	---

Представленная таблица демонстрирует разнообразие нейронных сетей, применяемых в работе с аудио. Она охватывает как модели для распознавания речи, такие как DeepSpeech и Whisper, так и модели для синтеза речи, например, Tacotron 2 и WaveGlow. Кроме того, в таблице представлены нейронные сети, способные генерировать музыку, как, например, Jukebox и MusicLM. Это убедительно демонстрирует, что нейронные сети становятся все более важным инструментом в современной аудио обработке.

Чтобы лучше понять, как работают и обучаются нейросети, следует обратить внимание на метод мел-кепстральных коэффициентов (MFCC), который используется многими системами ASR (Automatic Speech Recognition). В этом методе звук раскладывается не на спектральные компоненты, а на мел-кепстральные коэффициенты. Мел – психофизическая единица высоты звука, основанная на восприятии звука человеческим ухом. В результате сигнал представляется в виде достаточно небольшого набора значений, который к тому же устойчив к шуму, что делает его хорошим инструментом для обучения нейронных сетей [27]. Библиотека Librosa предоставляет большой спектр инструментов для работы с MFCC. Также многие ASR такие как DeepSpeech используют MFCC в качестве материала для обучения. В работе [28] представлен широкий набор инструментов Vosk для распознавания речи.

Другим методом является механизм каузальной свертки, который используется для синтеза речи. Каузальная свёртка – это специальный вид свёрточного слоя в нейронных сетях, который учитывает только предыдущие значения входного сигнала. Это означает, что выходное значение в любой момент времени зависит только от текущего и предыдущих значений входного сигнала, но не от будущих. Такой подход используется в нейронной сети Wavenet. Синтез речи в этом случае происходит следующим образом: Модель получает на вход начальный звуковой фрагмент. Используя каузальные свёрточные слои и дилатационные свертки, модель анализирует этот фрагмент и предсказывает вероятность следующего звукового фрагмента. Модель выбирает наиболее вероятный фрагмент и добавляет его к выходному звуковому сигналу. Процесс повторяется, пока не будет сгенерирована вся фраза.

Нейронные сети для генерации музыки имеют нечто общее с сетями, используемыми для синтеза речи, но в отличие от них также используются метаданные (жанр, исполнитель, дата выхода, текст и т.п.). Примеры использования представленных и других нейронных сетей приводятся в четвертой главе.

Глава 2. Алгоритмы работы с изображениями

2.1. Преобразование черно-белой фотографии в цветную. Deoldify

В данной работе происходит преобразование черно-белого изображения в цветное с использованием нейросети Deoldify [29].

Нейросеть Deoldify

Deoldify использует генеративно-сопоставительную нейронную сеть (GAN). Помимо использования GAN в этой модели также используется метод, известный как No-GAN. Большинство GAN состоят из двух частей - Генератор и Дискриминатор. Генератор — это часть, которая создает изображение. Дискриминатор пытается выделить настоящие цветные изображения из фейковых перекрашенных изображений. Техника No-GAN работает путем обучения моделей генератора и дискриминатора, присутствующих в GAN, изолированно. Это похоже на то, как вы обучаете обычную нейронную сеть, но отличается от GAN, поскольку они обычно обучаются параллельно. Затем они настраиваются вместе, как правило, так, как вы тренируете GAN.

Модель работает, беря черно-белое изображение и передавая его модели Deoldify. Затем модель выводит цветное изображение. Также для работы с нейросетью была загружена предварительно обученная модель

Алгоритм работы программы

1. Импорт необходимых библиотек (sys, deoldify).
2. Загрузка черно-белого фото для обработки.
3. Установка необходимых параметров.
4. Обработка изображения.
5. Выгрузка готового изображения в директорию result.

Используемые функции

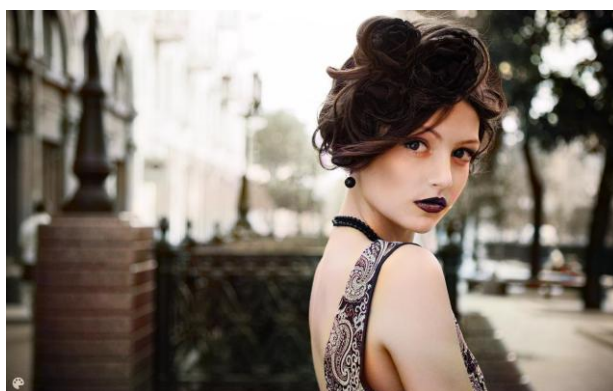
Название функции	Описание
<code>get_image_colorizer(artistic=True)</code>	Создание переменной для хранения модели
<code>colorizer.plot_transformed_image(image_path, render_factor=35, display_render_factor=True, figsize=(8,8))</code>	Передаем входное изображение, чтобы на выходе получить цветную картинку. Значение по умолчанию 35 для <code>render_factor</code> хорошо работает в большинстве случаев. <code>render_factor</code> определяет разрешение, при котором отображается цветная часть изображения. Нижний <code>render_factor</code> идеально подходит для изображений с более низким разрешением, в то время как более высокий <code>render_factor</code> для изображений с высоким разрешением.

Результат

До:



После:



Текст программы

```

from PIL import Image
from deoldify import device
from deoldify.device_id import DeviceId

# choices: CPU, GPU0...GPU7
device.set(device=DeviceId.GPU0)
import torch
if not torch.cuda.is_available():
    print('GPU not available.')
from deoldify.visualize import *
torch.backends.cudnn.benchmark = True

# если версия python >= 3.10 необходимо отдельно переназначить модуль
Sized
import collections

collections.Sized = collections.abc.Sized
# работаю в соседней папке, импортирую папку DeOldify
import sys

```

```
sys.path.append('./DeOldify')
image_path = input("Введите название изображения")
colorizer = get_image_colorizer(artistic=True)
colorizer.plot_transformed_image(image_path, render_factor=35,
display_render_factor=True, figsize=(8, 8))
```

2.2. Коррекция свойств фотографии. PyTorch и Torchvision

Программа направлена на автоматизацию процесса улучшения фотографий, что позволяет пользователям экономить время и усилия, а также получать высококачественные результаты. Это достигается благодаря использованию нейронных сетей для предсказания и применения оптимальных параметров изображения. В качестве параметров рассматриваются: яркость (Brightness), контрастность (Contrast), насыщенность (Saturation), резкость (Sharpness).

Высокая яркость: изображение становится светлее, но слишком высокая яркость может привести к потере деталей в светлых областях. Низкая яркость: изображение становится темнее, что может привести к потере деталей в тёмных областях. Высокая контрастность: чёткие границы между светлыми и тёмными областями, что может улучшить видимость объектов. Низкая контрастность: меньше различий между светлыми и тёмными областями, что может сделать изображение плоским и невыразительным. Высокая насыщенность: цвета становятся яркими и насыщенными, но чрезмерная насыщенность может выглядеть неестественно. Высокая резкость: увеличивает чёткость и детализацию, но чрезмерная резкость может добавить артефакты и сделать изображение «зернистым». Низкая резкость: уменьшает детализацию и делает изображение размытым, что может быть полезно для уменьшения шума.

Решается задача автоматического улучшения фотографий путем предсказания и применения оптимальных значений яркости,

контрастности, насыщенности и резкости. Используются PyTorch и Torchvision [30].

PyTorch используется для создания, обучения и оценки нейронной сети. Это основная библиотека, в которой реализована логика модели, функции потерь, оптимизации и обучение. Torchvision используется для преобразования изображений и подготовки данных для подачи в нейронную сеть. Это вспомогательная библиотека, которая облегчает работу с изображениями в контексте PyTorch.

Ход работы

```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import transforms
from torch.utils.data import DataLoader, Dataset
from PIL import Image, ImageEnhance
import torch.nn.functional as F
```

В начале программы импортируются все необходимые библиотеки, которые будут использоваться для работы с нейронной сетью, загрузки и обработки изображений.

```
class CustomDataset(Dataset):
    def __init__(self, image_paths, labels, transform=None):
        self.image_paths = image_paths
        self.labels = labels
        self.transform = transform

    def __len__(self):
        return len(self.image_paths)

    def __getitem__(self, idx):
        image = Image.open(self.image_paths[idx]).convert("RGB")
        label = self.labels[idx]
        if self.transform:
            image = self.transform(image)
        label = torch.tensor(label, dtype=torch.float32)
        return image, label
```

Класс BrightnessContrastNet определяет архитектуру нейронной сети. Это сверточная нейронная сеть с двумя сверточными слоями, двумя

слоями подвыборки (pooling) и двумя полносвязными слоями. На выходе сеть предсказывает четыре параметра: яркость, контрастность, насыщенность и резкость.

```
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

Для обучения модели используется функция потерь `nn.MSELoss` и оптимизатор `optim.Adam`. Функция потерь вычисляет среднеквадратичную ошибку (MSE) между предсказанными и реальными значениями, а оптимизатор обновляет параметры модели для минимизации этой ошибки.

```
num_epochs = 10
for epoch in range(num_epochs):
    running_loss = 0.0
    for images, labels in dataloader:
        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
    print(f'Epoch [{epoch + 1}/{num_epochs}], Loss: {running_loss /
len(dataloader):.4f}')
```

В этом блоке происходит обучение модели. Данные загружаются в батчах (пакетах), проходятся через модель, вычисляется ошибка, выполняется обратное распространение ошибки и обновляются параметры модели.

```
def predict_and_enhance_image(image_path, model, output_path):
    image = Image.open(image_path).convert("RGB")
    transformed_image = transform(image).unsqueeze(0)
    model.eval()
    with torch.no_grad():
        output = model(transformed_image)
        brightness, contrast, saturation, sharpness = output[0].numpy()

    enhancer = ImageEnhance.Brightness(image)
    image = enhancer.enhance(brightness)
    enhancer = ImageEnhance.Contrast(image)
    image = enhancer.enhance(contrast)
    enhancer = ImageEnhance.Color(image)
```

```
image = enhancer.enhance(saturation)
enhancer = ImageEnhance.Sharpness(image)
image = enhancer.enhance(sharpness)
```

```
image.save(output_path)
return brightness, contrast, saturation, sharpness
```

Функция `predict_and_enhance_image` загружает изображение,

использует обученную модель для предсказания параметров улучшения (яркость, контрастность, насыщенность, резкость), применяет эти параметры к изображению и сохраняет улучшенное изображение.

```
image_path = 'new_image.jpg' # Замените на реальный путь к изображению
output_path = 'enhanced_image.jpg' # Замените на желаемый путь для
сохранения улучшенного изображения
brightness, contrast, saturation, sharpness =
predict_and_enhance_image(image_path, model, output_path)
print(
    f'Predicted Brightness: {brightness}, Predicted Contrast: {contrast},
    Predicted Saturation: {saturation}, Predicted Sharpness: {sharpness}')
print(f'Enhanced image saved to {output_path}')
```

Сохранение измененного фото “new_image.jpg” и вывод рассчитанных коэффициентов свойств фотографии.

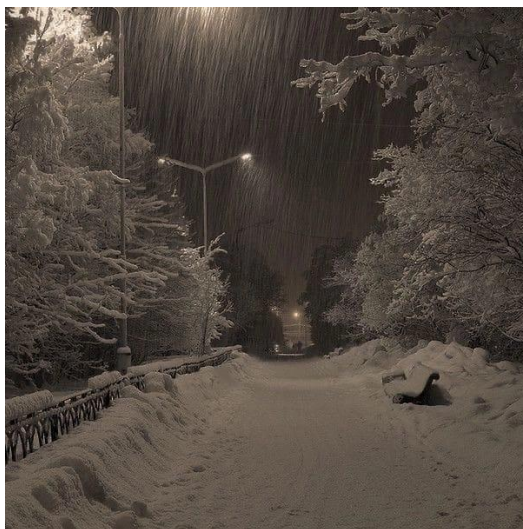
Для эффективного обучения модели по улучшению фотографий важно подобрать качественный и разнообразный набор данных. Вот несколько рекомендаций по выбору фотографий для обучения модели. Выбраны следующие фото для обучения:



Они достаточно яркие и сняты на профессиональные камеры в качестве 4к, что очень важно. Также были выбраны различные виды:

город, здание изнутри, природа. Это значит, что множества фотографий будут изменены корректно.

Фотография до улучшения выглядела так:



Фотография была снята на телефон с использованием небольшой обработки в виде увеличения яркости.

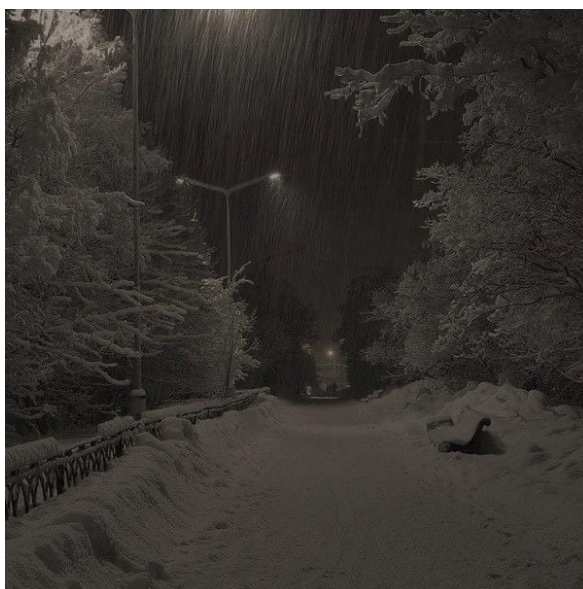
После запуска программы происходит процесс обучения:

```
Epoch [1/10], Loss: 2.1882
Epoch [2/10], Loss: 0.4750
Epoch [3/10], Loss: 1.2262
Epoch [4/10], Loss: 0.4693
Epoch [5/10], Loss: 0.1553
Epoch [6/10], Loss: 0.0685
Epoch [7/10], Loss: 0.0507
Epoch [8/10], Loss: 0.0268
Epoch [9/10], Loss: 0.0415
Epoch [10/10], Loss: 0.0081
```

В процессе обработки происходят все, что было написано ранее. В итоге, получаем вывод программы о успешной работе. Видим предлагаемые коэффициенты для фото:

Predicted Brightness: 0.7115066647529602, Predicted Contrast: 0.8492302894592285, Predicted Saturation: 0.7700173854827881, Predicted Sharpness: 0.8351155519485474

После улучшения фотография выглядит так:



Были изменены все значения в среднем на 20% и фотография стала выглядеть более натурально.

Таблица использованных функций

torch	
torch.tensor	Создание тензоров, позволяет выполнять численные операции на GPU
torch.nn.functional	
F.relu	Применяет функцию активации ReLU (Rectified Linear Unit) к тензору. Устанавливает отрицательные значения в тензоре на ноль, сохраняя положительные без изменений
F.sigmoid	Применяет функцию активации сигмоид к тензору. Сигмоидная функция преобразует значения тензора в диапазоне [0, 1].
F.softmax	Применяет функцию активации Softmax к тензору. Softmax преобразует значения тензора в вероятностное распределение, где сумма всех значений равна 1
F.dropout	Применяет регуляризацию Dropout к тензору. Dropout удаляет (обнуляет) некоторые элементы тензора во время обучения, что помогает предотвратить переобучение.

Текст программы:

```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import transforms
from torch.utils.data import DataLoader, Dataset
```

```

from PIL import Image, ImageEnhance
import torch.nn.functional as F

class CustomDataset(Dataset):
    def __init__(self, image_paths, labels, transform=None):
        self.image_paths = image_paths
        self.labels = labels
        self.transform = transform

    def __len__(self):
        return len(self.image_paths)

    def __getitem__(self, idx):
        image = Image.open(self.image_paths[idx]).convert("RGB")
        label = self.labels[idx]
        if self.transform:
            image = self.transform(image)
        label = torch.tensor(label, dtype=torch.float32)
        return image, label

image_paths = ['image1.jpg', 'image2.jpg', 'image3.jpg']
labels = [(0.8, 1.2, 1.0, 1.1), (1.0, 1.0, 1.0, 1.0),
          (0.9, 1.1, 1.1, 0.9)]

transform = transforms.Compose([
    transforms.Resize((128, 128)),
    transforms.ToTensor()
])

dataset = CustomDataset(image_paths, labels, transform=transform)
dataloader = DataLoader(dataset, batch_size=2, shuffle=True)

class BrightnessContrastNet(nn.Module):
    def __init__(self):
        super(BrightnessContrastNet, self).__init__()
        self.conv1 = nn.Conv2d(3, 16, kernel_size=3, stride=1, padding=1)
        self.conv2 = nn.Conv2d(16, 32, kernel_size=3, stride=1,
padding=1)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2, padding=0)
        self.fc1 = nn.Linear(32 * 32 * 32, 128)
        self.fc2 = nn.Linear(128, 4)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 32 * 32 * 32)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)

```

```

        return x

model = BrightnessContrastNet()

criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

num_epochs = 10
for epoch in range(num_epochs):
    running_loss = 0.0
    for images, labels in dataloader:
        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
    print(f'Epoch [{epoch + 1}/{num_epochs}], Loss: {running_loss /
len(dataloader):.4f}')

def predict_and_enhance_image(image_path, model, output_path):
    image = Image.open(image_path).convert("RGB")
    transformed_image = transform(image).unsqueeze(0)
    model.eval()
    with torch.no_grad():
        output = model(transformed_image)
    brightness, contrast, saturation, sharpness = output[0].numpy()

    enhancer = ImageEnhance.Brightness(image)
    image = enhancer.enhance(brightness)
    enhancer = ImageEnhance.Contrast(image)
    image = enhancer.enhance(contrast)
    enhancer = ImageEnhance.Color(image)
    image = enhancer.enhance(saturation)
    enhancer = ImageEnhance.Sharpness(image)
    image = enhancer.enhance(sharpness)

    image.save(output_path)
    return brightness, contrast, saturation, sharpness

image_path = 'new_image.jpg'
output_path = 'enhanced_image.jpg'
brightness, contrast, saturation, sharpness =
predict_and_enhance_image(image_path, model, output_path)
print(
    f'Predicted Brightness: {brightness}, Predicted Contrast: {contrast},
Predicted Saturation: {saturation}, Predicted Sharpness: {sharpness}')
print(f'Enhanced image saved to {output_path}')

```

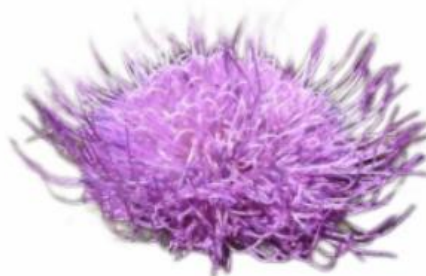
2.3. Удаление фона с изображения. rRemBG

Программа убирает фон с изображения формата при помощи библиотеки для работы с изображениями pillow и библиотеки rembg. rembg – библиотека, созданная Даниэлем Гатисом (Daniel Gatis), предоставляющая удобные функции для удаления заднего фона с изображений [31]. Процесс удаления фона происходит в два этапа:

1. Получение маски: картинка подается на вход нейронной сети, которая генерирует маску.



2. Замена фона альфа-каналом

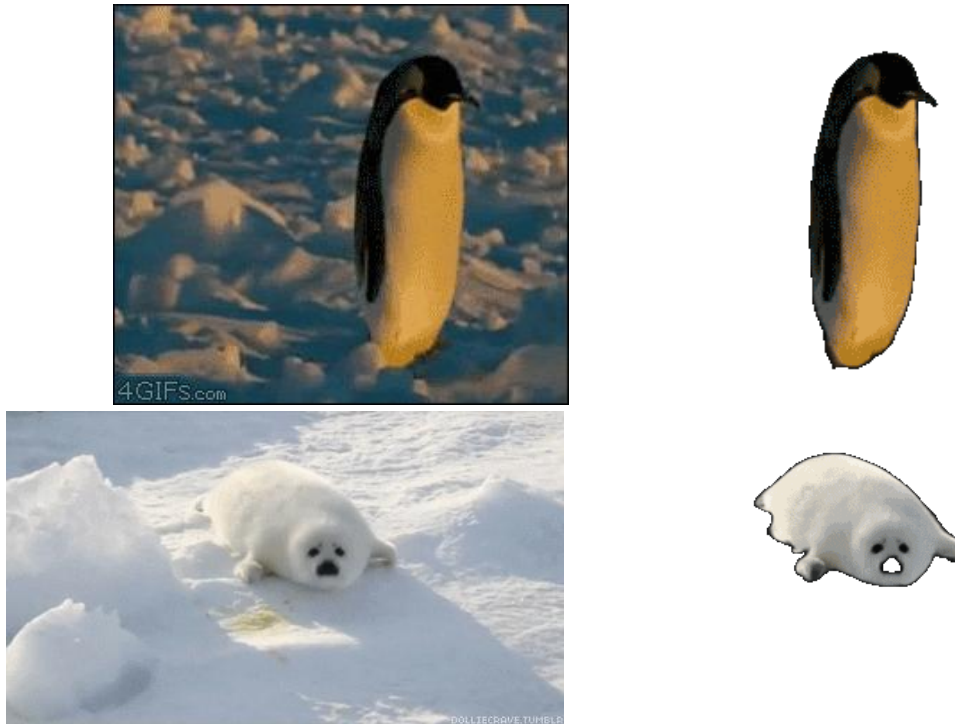


Библиотека может использовать различные нейронные сети, в том числе обученные самостоятельно, что делает ее подходящей для удаления фона с специфичных объектов, к примеру при фотограмметрии. По умолчанию используется сеть U2-Net: U Square Net.

Алгоритм работы программы

1. Поиск всех изображений формата gif в папке
2. Открываем файл
3. Убираем фон с каждого кадра
4. Сохраняем полученные кадры в файл

Результат



Используемые функции

glob	
glob.glob(pattern)	Находит все файлы, подходящие под паттерн
pillow	
PIL.Image.open()	Открывает файл с изображением
PIL.ImageSequence.Iterator()	Позволяет итерироваться по кадрам gif
Image.save()	Сохраняет полученное изображение в файл
rembg	
rembg.remove()	Удаляет фон с изображения

Текст программы

```
# Подключение библиотек
from rembg import remove
from PIL import ImageSequence, Image
import glob

# Расчет кол-ва кадров в секунду для gif
def get_avg_fps(PIL_Image_object):
    """ Returns the average framerate of a PIL Image object """
    PIL_Image_object.seek(0)
    frames = duration = 0
    while True:
        try:
```

```

        frames += 1
        duration += PIL_Image_object.info['duration']
        PIL_Image_object.seek(PIL_Image_object.tell() + 1)
    except EOFError:
        return frames / duration * 1000
# Рисуем прогресс бар в консоли
def printProgressBar (iteration, total, prefix = '', suffix = '',
decimals = 1, length = 100, fill = '█', printEnd = "\r"):
    percent = ("{0:." + str(decimals) + "f>").format(100 * (iteration /
float(total)))
    filledLength = int(length * iteration // total)
    bar = fill * filledLength + '-' * (length - filledLength)
    print(f'\r{prefix} |{bar}| {percent}% {suffix}', end = printEnd)
    if iteration == total:
        print()

# Находим все файлы с картинками в нужной папке
files = glob.glob("img/*.gif")
it = 0

for file in files:
    filename = file.removeprefix("img/")
    printProgressBar(it, len(files), f"{it}/{len(files)}", filename)

    output = f"output/{filename}"
    final_frames = []
    with Image.open(file) as im:

        # Проходимся по каждому кадру в картинке
        for frame in ImageSequence.Iterator(im):
            # Убираем фон и сохраняем в массив
            final_frames.append(remove(frame))
        # Сохраняем массив обработанных кадров в файл
        final_frames[0].save(output,
            save_all=True, append_images=final_frames[1:],
optimize=False,disposal=2, duration=get_avg_fps(im), loop=0)
        it+=1

```

2.4. Сегментация изображения. U-Net

Будем обучать нейронную сеть сегментировать изображение. Как базу данных возьмём оригинальное изображение и изображение с контурами объектов. После обучения нейронная сеть должна сегментировать изображение.

Ход работы

Для обучения возьмём нейронную сеть U-Net. U-Net считается одной из стандартных архитектур CNN для задач сегментации изображений, когда нужно не только определить класс изображения целиком, но и сегментировать его области по классу, т. е. создать маску, которая будет разделять изображение на несколько классов. Архитектура состоит из стягивающего пути для захвата контекста и симметричного расширяющегося пути, который позволяет осуществить точную локализацию [32-34].

В файле `dataset_loader.py` реализована загрузка dataset.

В файле `UNET.py` реализовано сегментирование изображения.

В файле `sift.py` реализована модель нейронной сети U-Net.

Подключение рабочих файлов:

```
%run dataset_loader.py
%run sift.py
%run UNET.py
```

Подключение библиотек и путей к исходным файлам для теста и тренинга.

```
import os
import torch
import random
import matplotlib.pyplot as plt
from scipy import misc
from PIL import Image
IMG_PATH_TRAIN = "cell_segmentation-main/BBBC018_v1_images-fixed/train/"
OUTLINE_PATH_TRAIN = "cell_segmentation-main/BBBC018_v1_outlines/train/"

IMG_PATH_TEST = "cell_segmentation-main/BBBC018_v1_images-fixed/val/"
OUTLINE_PATH_TEST = "cell_segmentation-main/BBBC018_v1_outlines/val/"
```

Основная идея, лежащая в основе генерации набора данных, заключается в многократном увеличении данных, т.е. при использовании разных случайных начальных значений случайным образом применяются некоторые преобразования.

Чтобы получить одинаковое случайное преобразование для исходного изображения и сегментированного изображения установил одно и то же начальное значение, чтобы преобразовать их одинаковым образом.

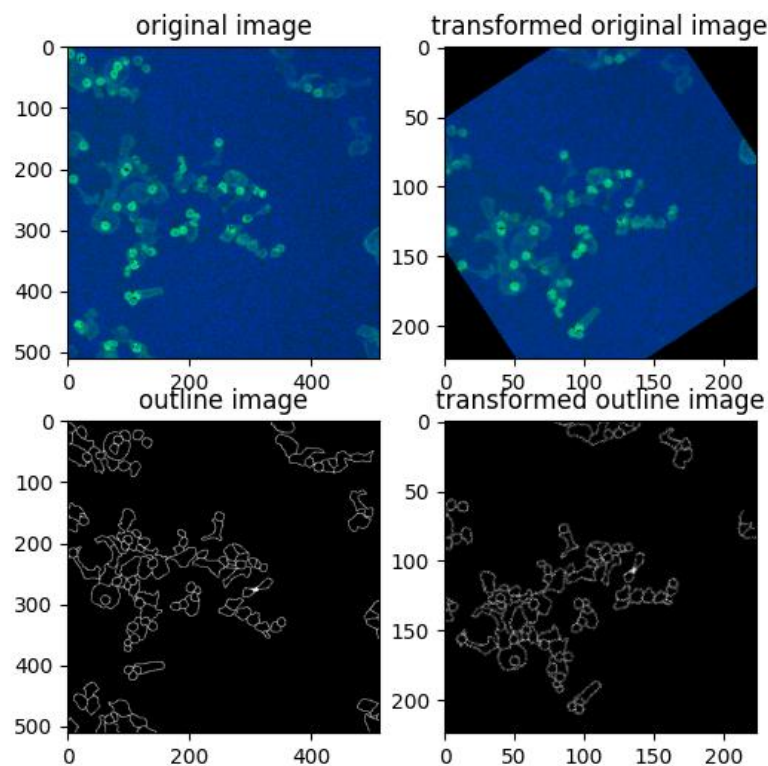
```
from torchvision import transforms
```

```
transform_train = transforms.Compose([
    transforms.ToPILImage(),
    transforms.Resize((224, 224)), # Real Image are heavy
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(90),
    transforms.ToTensor(),
])
```

Для теста подключения к dataset выполним

```
Train = get_dataset(transform_train, IMG_PATH_TRAIN, OUTLINE_PATH_TRAIN,
show_examples_amount=1)
```

Вот результат



Загружаем dataset для тренинга

```
Test = get_dataset(transform_train, IMG_PATH_TEST, OUTLINE_PATH_TEST,
show_examples_amount=1)
```

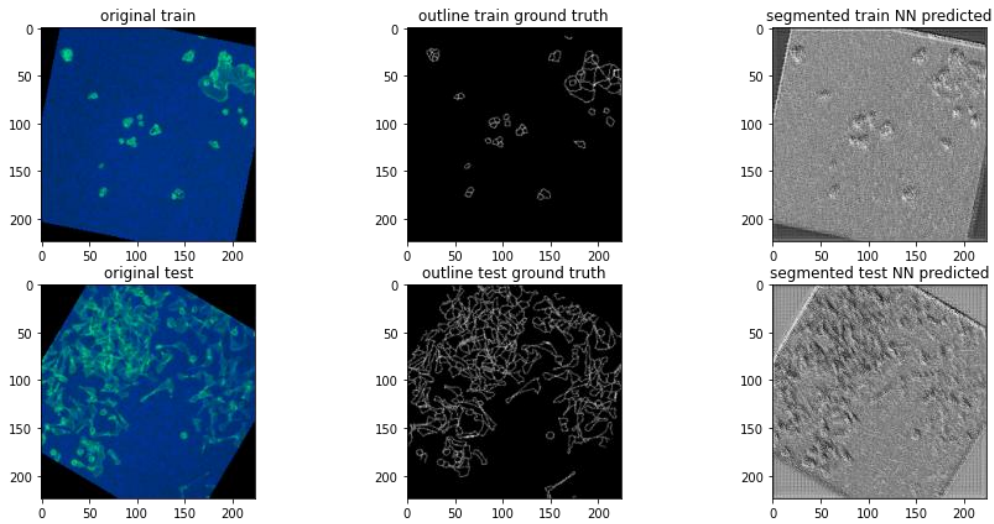
Функция для тренировки нейронной сети

```
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
```

BATCH_SIZE=32

```
def train(model, train_dataset, test_dataset, num_epoch=32):
    model.double()
    gd = optim.Adamax(model.parameters(), lr=0.002)
    dataloader = DataLoader(train_dataset, batch_size=BATCH_SIZE,
shuffle=True)
    losses = []
    for epoch in range(num_epoch):
        print(f'Epoch # {epoch}')
        model.eval()
        sift(model, train_dataset, test_dataset)
        model.train()
        for i, (batch) in enumerate(dataloader):
            images = batch[:, :-1, :, :]
            segm = batch[:, -1, :, :]

            gd.zero_grad()
            segmented_predicted = model(images.cuda())
            loss = (segm - segmented_predicted.cpu()).norm()
            loss.backward()
            losses.append(loss.detach().cpu().numpy())
            gd.step()
            gd.zero_grad()
            images.cpu() # Well, seems without cpu transferring gpu
memory runs out
        print("#%i, Train loss:
%.7f"%(epoch+1,np.mean(losses)),flush=True)
        Подключаем модель U-Net
net = UNET().double().cuda()
        Выполняем сегментирование без тренировки
sift(net, Train, Test)
        Результат
```



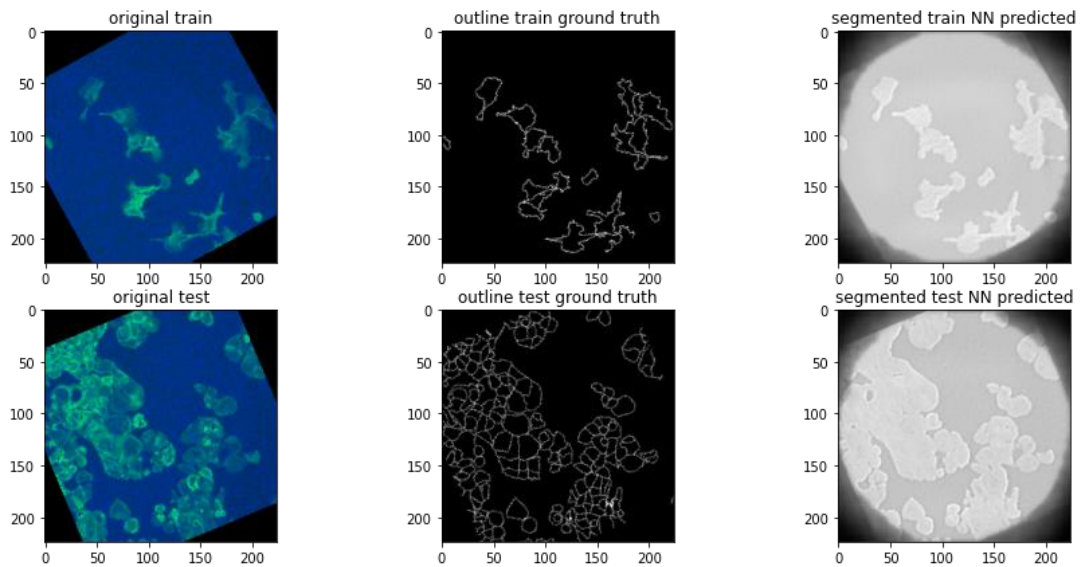
Запускаем тренировку и сохраняем данные

```
train(net, Train, Test, num_epoch=32)
torch.save(net.state_dict(), './UNET_weights.save')
```

Выполняем сегментирование после тренировки

```
sift(net, Train, Test)
```

Результат



Текст программ

Файл dataset_loader.py

```
import torch
import numpy as np
import matplotlib.pyplot as plt
import os
import random
```

```

def get_dataset(transformation, img_path, outline_path=None,
seeds_amount=100, show_examples_amount=0):
    Images = []
    Segmented = []

    example_shown = 0
    for filename in os.listdir(img_path):
        img=plt.imread(img_path+filename)

        if outline_path:
            outline_filename = filename[0:6] + "cells.png"
            outline = plt.imread(outline_path+outline_filename)
            outline = outline[:, :, None]
        else:
            outline = img[:, :, 1]

        seeds = np.random.randint(0, seeds_amount * 15, seeds_amount)
        for seed in seeds:
            random.seed(seed)
            torch.manual_seed(seed)
            img_trnsfrmd=transformation(img).numpy()
            img_trnsfrmd=img_trnsfrmd.transpose(1,2,0)

            random.seed(seed)
            torch.manual_seed(seed)
            outline_trnsfrmd=transformation(outline).numpy()
            outline_trnsfrmd=outline_trnsfrmd.squeeze()
            Images.append(img_trnsfrmd)
            Segmented.append(outline_trnsfrmd)

            if (example_shown < show_examples_amount) and
(np.random.binomial(1, 0.0007) == 1):
                ### show some example
                f, ax = plt.subplots(2,2, figsize=(6,6))

                ax[0][0].set_title('original image')
                ax[0][0].imshow(img)

                ax[0][1].set_title('transformed original image')
                ax[0][1].imshow(img_trnsfrmd)

                ax[1][0].set_title('outline image')
                ax[1][0].imshow(outline, cmap='gray')

                ax[1][1].set_title('transformed outline image')
                ax[1][1].imshow(outline_trnsfrmd, cmap='gray')
                plt.show()
                example_shown +=1

```

```

    Images_npy_array = np.array(Images).transpose(0,3,1,2) # transpose
for the torch image format: C,W,H
    Images_tensor =
torch.from_numpy(Images_npy_array).type(torch.DoubleTensor)

    Segmented_npy_array = np.array(Segmented)[: ,None]
    Segmented_tensor =
torch.from_numpy(Segmented_npy_array).type(torch.DoubleTensor)

    Dataset = torch.cat((Images_tensor, Segmented_tensor), dim=1)
return Dataset
Файл sift.py

```

```

import torch
import matplotlib.pyplot as plt
import numpy as np

def sift(model, train_dataset, test_dataset):
    i_train = np.random.randint(0, train_dataset.shape[0], 1)[0]
    i_test = np.random.randint(0, test_dataset.shape[0], 1)[0]

    img_train = train_dataset[i_train, :-1].double()
    seg_train = train_dataset[i_train, -1].double()
    img_test = test_dataset[i_test, :-1].double()
    seg_test = test_dataset[i_test, -1].double()
    imgs = torch.stack((img_train, img_test)).cuda()
    reco = model(imgs)

    #print(reco.shape)
    f, ax = plt.subplots(2,3, figsize=(15,7))
    ax[0][0].set_title('original train')
    ax[0][0].imshow(img_train.cpu().numpy().transpose(1,2,0).clip(0,1))
    ax[0][1].set_title('outline train ground truth')
    ax[0][1].imshow(seg_train.cpu().numpy().clip(0,1), 'gray')
    ax[0][2].set_title('segmented train NN predicted')
    ax[0][2].imshow(reco[0][0].cpu().detach().numpy().clip(0,1), 'gray')
    ax[1][0].set_title('original test')
    ax[1][0].imshow(img_test.cpu().numpy().transpose(1,2,0).clip(0,1))
    ax[1][1].set_title('outline test ground truth')
    ax[1][1].imshow(seg_test.cpu().numpy().clip(0,1), 'gray')
    ax[1][2].set_title('segmented test NN predicted')
    ax[1][2].imshow(reco[1][0].cpu().detach().numpy().clip(0,1), 'gray')

    plt.show()
Файл UNET.py

```

```

import torch
import torch.nn as nn
import torchvision

```

```

class UNET(nn.Module):
    THIRD_POOLING_INDEX = 16
    FORTH_POOLING_INDEX = 23
    def __init__(self, n_class = 1):
        super(UNET, self).__init__()

        # Contracting Path
        self.c1 = UNET.get_conv2d_block(3, 16, 3, 1)
        self.p1 = nn.MaxPool2d(2)
        self.d1 = nn.Dropout2d()

        self.c2 = UNET.get_conv2d_block(16, 32, 3, 1)
        self.p2 = nn.MaxPool2d(2)
        self.d2 = nn.Dropout2d()
        self.c3 = UNET.get_conv2d_block(32, 64, 3, 1)
        self.p3 = nn.MaxPool2d(2)
        self.d3 = nn.Dropout2d()
        self.c4 = UNET.get_conv2d_block(64, 128, 3, 1)
        self.p4 = nn.MaxPool2d(2)
        self.d4 = nn.Dropout2d()
        self.c5 = UNET.get_conv2d_block(128, 256, 3, 1)

        self.u6 = nn.ConvTranspose2d(256, 128, kernel_size=2, stride=2,
padding=0)
        self.d6 = nn.Dropout2d()
        self.c6 = UNET.get_conv2d_block(256, 128, 3, 1)
        self.u7 = nn.ConvTranspose2d(128, 64, kernel_size=2, stride=2,
padding=0)
        self.d7 = nn.Dropout2d()
        self.c7 = UNET.get_conv2d_block(128, 64, 3, 1)
        self.u8 = nn.ConvTranspose2d(64, 32, kernel_size=2, stride=2,
padding=0)
        self.d8 = nn.Dropout2d()
        self.c8 = UNET.get_conv2d_block(64, 32, 3, 1)
        self.u9 = nn.ConvTranspose2d(32, 16, kernel_size=2, stride=2,
padding=0)
        self.d9 = nn.Dropout2d()
        self.c9 = UNET.get_conv2d_block(32, 16, 3, 1)
        self.c10 = nn.Conv2d(16, 1, 1)
        self.activation = nn.Sigmoid()

        #outputs = Conv2D(1, (1, 1), activation='sigmoid')(c9)
    def forward(self, batch):
        c1_output = self.c1(batch)
        h = c1_output
        h = self.p1(h)
        h = self.d1(h)
            c2_output = self.c2(h)

```

```

h = c2_output
h = self.p2(h)
h = self.d2(h)
    c3_output = self.c3(h)
h = c3_output
h = self.p3(h)
h = self.d3(h)
    c4_output = self.c4(h)
h = c4_output
h = self.p4(h)
h = self.d4(h)
h = self.c5(h)
    u = self.u6(h)
h = torch.cat((u, c4_output), dim=(1))
h = self.d6(h)
h = self.c6(h)
    u = self.u7(h)
h = torch.cat((u, c3_output), dim=(1))
h = self.d7(h)
h = self.c7(h)
    u = self.u8(h)
h = torch.cat((u, c2_output), dim=(1))
h = self.d8(h)
h = self.c8(h)
    u = self.u9(h)
h = torch.cat((u, c1_output), dim=(1))
h = self.d9(h)
h = self.c9(h)
    h = self.c10(h)
ret = self.activation(h)
return ret

```

```

@staticmethod
def get_conv2d_block(input_size, output_size, kernel_size, padding):
    """Function to add 2 convolutional layers with the parameters
passed to it"""
    # first layer
    # kernel_initializer = 'he_normal', padding = 'same'
    conv2d_block = nn.Sequential()
    conv2d = nn.Conv2d(input_size, output_size, kernel_size =
kernel_size, padding=padding)

    conv2d_block.add_module('conv_0', conv2d)
    conv2d_block.add_module('batchnorm_0',
nn.BatchNorm2d(output_size))
    conv2d_block.add_module('relu0', nn.ReLU())

    conv2d_2 = nn.Conv2d(output_size, output_size,
kernel_size=kernel_size, padding=padding)

```

```
conv2d_block.add_module('conv_1', conv2d_2)
conv2d_block.add_module('batchnorm_1',
nn.BatchNorm2d(output_size))
conv2d_block.add_module('relu0', nn.ReLU())

return conv2d_block
```

2.5. Сегментация изображения. SAM

Сегментация - это процесс разделения цифрового изображения на несколько сегментов. Цель сегментации заключается в упрощении или изменении представления изображения, чтобы его было проще и легче анализировать. Сегментация изображений обычно используется для того, чтобы выделить объекты и границы (линии, кривые, и т. д.) на изображениях. Более точно, сегментация изображений - это процесс присвоения таких меток каждому пикселю изображения, что пиксели с одинаковыми метками имеют общие визуальные характеристики.

Результатом сегментации изображения является множество сегментов, которые вместе покрывают всё изображение, или множество контуров, выделенных из изображения. Все пиксели в сегменте похожи по некоторой характеристике или вычисленному свойству, например, по цвету, яркости или текстуре.

Для выполнения сегментации изображения будем использовать уже обученную модель из официальной документации, которая может распознавать любые объекты (SAM). Сама библиотека `segment_anything` работает на `torch`. Как написано на официальном сайте: Segment Anything Model (SAM) - это модель сегментации изображений, разработанная Meta AI. Эта модель может определять точное местоположение либо конкретных объектов на изображении, либо каждого объекта на изображении. SAM был выпущен в апреле 2023 года. SAM имеет открытый исходный код и выпущен под лицензией Apache 2.0.

Использованные функции

Функция	Описание
<code>segment_anything.sam_model_registry</code>	Загрузка модели в программу
<code>SamAutomaticMaskGenerator(sam, points_per_batch=16)</code>	Создание генератора масок для изображений
<code>mask_generator.generate(image_rgb)</code>	В метод <code>generate</code> передаем изображение в RGB и для него генерируем маски. Результатом будет - <code>list(dict(str, any))</code> - список словарей, описывающих отдельные сегментации
<code>ax.imshow</code>	Отображение изображений с аннотациями на текущем графическом контексте
<code>Cv2.imread</code>	Считывание изображения для сегментации
<code>CV2.cvtColor</code>	Перевод изображения из bgr в rgb
<code>Plt.show</code>	Вывод изображения

Результаты

Начальное фото



После сегментации



Текст программы

```
import cv2
import matplotlib.pyplot as plt
import numpy as np
from segment_anything import sam_model_registry
from segment_anything import SamAutomaticMaskGenerator
import torch

def show_anns(anns):
    if len(anns) == 0:
        return
    sorted_anns = sorted(anns, key=(lambda x: x['area']), reverse=True)
    ax = plt.gca()
```

```

ax.set_autoscale_on(False)

img = np.ones((sorted_anns[0]['segmentation'].shape[0],
sorted_anns[0]['segmentation'].shape[1], 4))
img[:, :, 3] = 0
for ann in sorted_anns:
m = ann['segmentation']
color_mask = np.concatenate([np.random.random(3), [0.35]])
img[m] = color_mask
ax.imshow(img)

device = "cpu"
IMAGE_PATH = "images.jpg"
DEVICE = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
sam_checkpoint = 'sam_vit_h_4b8939.pth'
MODEL_TYPE = "vit_h"
sam = sam_model_registry[MODEL_TYPE](checkpoint=sam_checkpoint)
sam.to(device=DEVICE)
mask_generator = SamAutomaticMaskGenerator(sam, points_per_batch=16)

image_bgr = cv2.imread(IMAGE_PATH)
image_rgb = cv2.cvtColor(image_bgr, cv2.COLOR_BGR2RGB)
result = mask_generator.generate(image_rgb)

plt.figure(figsize=(20, 20))
plt.imshow(image_bgr)
show_anns(result)
plt.axis('off')
plt.show()

```

2.6. Сегментация изображений для пяти классов объектов на основе набора данных MS-COCO

В рамках данной работы решается задача сегментации изображений для следующих классов объектов: Велосипед, Человек, Лошадь, Диван, Телевизор

Используется набор данных MS-COCO [35], представляющий собой объемный набор данных для обнаружения объектов, сегментации, обнаружения ключевых точек, разделенных на обучающий, валидационный и тестовый набор данных.

В ходе работы была реализована и обучена сверточная модель нейронной сети, использующая предобученную сеть VGG в качестве базовой сети.

Решение реализовано на языке программирования Python 3.9 с применением следующих пакетов: torch, torchvision, pycocotools, argparse, shutil, sys

Использованные функции

Название	Краткое описание
<code>transforms.Compose()</code>	Метод выполняет несколько последовательных преобразований над изображениями
<code>pycocotools.coco.COCO()</code>	Метод загружает аннотации к изображениям
<code>torch.utils.data.dataloader.DataLoader()</code>	Метод объявляет загрузчик для получения изображений в процессе обучения
<code>torch.optim.SGD()</code>	Метод объявляет SGD оптимизатор
<code>optimizer.step()</code>	Метод выполняет один шаг оптимизации
<code>lr_scheduler.step()</code>	Метод выполняет один шаг функции потерь
<code>torch.nn.Conv2d()</code>	Метод создает сверточный слой
<code>torch.nn.RELU()</code>	Метод объявляет линейную единичную функцию
<code>torch.nn.Dropout(p)</code>	Метод объявляет Dropout слой, который случайным образом обнуляет некоторые элементы входного тензора с вероятностью p, используя выборки из распределения Бернулли

Подготовка и построение данных

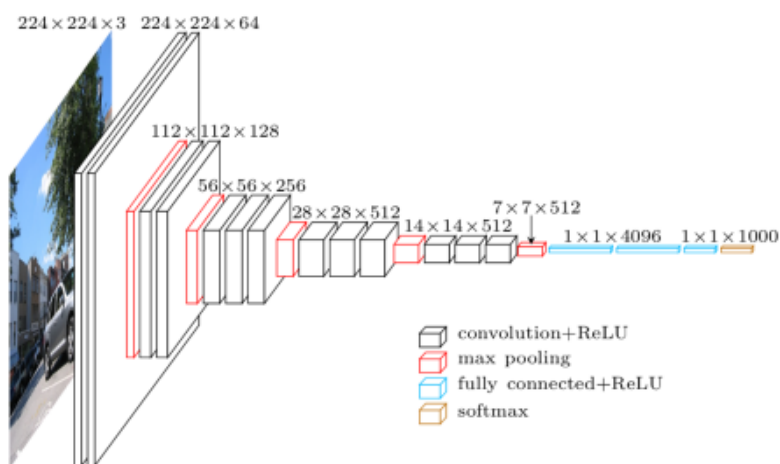
Для обучения модели нейронной сети в локальное дисковое хранилище был загружен набор данных с обучающей выборкой (2017 Train images), валидационной выборкой (2017 Val images) и аннотации к ним (2017 Train/Val annotations).

Для получения изображений в процессе обучения был использован класс `DataLoader` из модуля `torch.utils.data`. Для подготовки данных был реализован собственный класс `COCOSegmentation`, содержащий логику

загрузки изображений с диска и обработки аннотаций к ним при помощи класса COCO из модуля rusocotools.

Модель нейронной сети

Для задач сегментации изображений широко используются сверточные нейронные сети, показывающие относительно высокую точность сегментации и скорость работы. Для реализации задачи сегментации изображений в данной работе была использована сверточная нейронная сеть, в которой в качестве базовой сети, служащей для извлечения признаков из поступающего на вход изображения, используется широко известная предобученная сеть VGG. Данное дополнение увеличивает точность сегментации изображений [36]. Общая структура сети VGG приведена на рисунке.



Структура модели VGG

Структура используемой для сегментации модели нейронной сети представлена ниже:

```
FCN(
  (pretrained): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (3): ReLU(inplace=True)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
```

```

    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (6): ReLU(inplace=True)
    (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (8): ReLU(inplace=True)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (11): ReLU(inplace=True)
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (13): ReLU(inplace=True)
    (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (15): ReLU(inplace=True)
    (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (18): ReLU(inplace=True)
    (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (20): ReLU(inplace=True)
    (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (22): ReLU(inplace=True)
    (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
    (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (25): ReLU(inplace=True)
    (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (27): ReLU(inplace=True)
    (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1,
1))
    (29): ReLU(inplace=True)
    (30): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
)
(head): FCNHead(
  (block): Sequential(
    (0): Conv2d(512, 128, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1), bias=False)
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True,
track_running_stats=True)
    (2): ReLU(inplace=True)

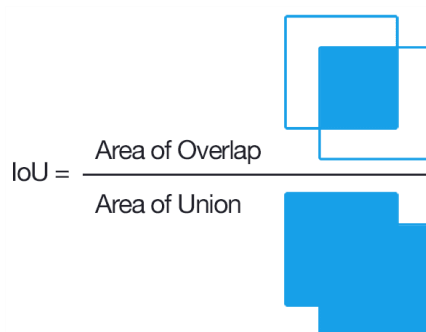
```

```

(3): Dropout(p=0.1, inplace=False)
(4): Conv2d(128, 6, kernel_size=(1, 1), stride=(1, 1))
)
)
)

```

В качестве метрики оценки точности сегментации используется показатель Intersection Over Union (IoU) [37], который принимает значение от 0 до 1 и может быть вычислен как отношение площади пересечения между предсказанной ограничивающей маской и эталонной маской к площади их объединения. На рисунке показана схема вычисления метрики IoU.



Вычисление IoU

Обучение и тестирование

При обучения нейронной сети были определены следующие параметры:

- Количество эпох – 50;
- Размер входных изображений – 520 x 520;
- Размер батчей в обучающей выборке – 4;

Также при обучении использовался оптимизатор SGD, который реализует стохастический градиентный спуск, являющимся известным методом оптимизации, используемым в машинном обучении и глубоком обучении. Для подключения использовалась функция SGD из пакета torch.optim, реализующего многочисленные алгоритмы оптимизации. Для оптимизатора использовались следующие параметры:

- Learning rate – 0.0001;
- momentum - 0.9;
- weight_decay - 0.0001.

При обучении нейронных сетей обычно применяется функция потерь, используемая при корректировке весов модели во время обучения и минимизации потерь. В качестве функции потерь была использована функция перекрестной энтропии (CrossEntropyLoss) [5], которая обычно используется для задач классификации с несколькими классами. Перекрестная энтропия измеряет разницу между обнаруженными и предсказанными распределениями вероятностей модели нейронной сети.

Обучение производилось на 50 эпохах. Ниже представлен вывод процесса обучения, содержащий значения потерь при обучении и среднего значения IoU на валидационном наборе данных:

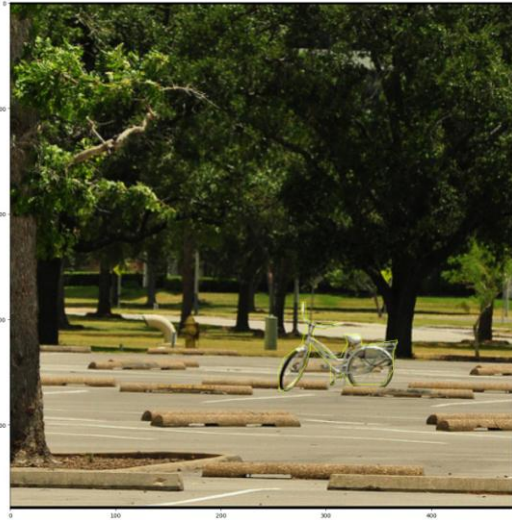
Epochs: 1/50	Loss: 3.6321	Validation mIoU: 0.0288
Epochs: 2/50	Loss: 1.2150	Validation mIoU: 0.0833
Epochs: 3/50	Loss: 0.8007	Validation mIoU: 0.1179
Epochs: 20/50	Loss: 0.1912	Validation mIoU: 0.3045
Epochs: 21/50	Loss: 0.1794	Validation mIoU: 0.3102
Epochs: 40/50	Loss: 0.0977	Validation mIoU: 0.4183
Epochs: 41/50	Loss: 0.0927	Validation mIoU: 0.4292
Epochs: 50/50	Loss: 0.0634	Validation mIoU: 0.4731

Результаты

В результате обучения модели на валидационном наборе данных было получено среднее значение IoU, равное 47.31%. Ниже представлены результаты работы модели по сегментации каждой группы изображений:

Велосипед

Человек



Текст программы

Код формирования набора данных

```
import os
import pickle
import torch
import numpy as np

from tqdm import trange
from PIL import Image
from .segbase import SegmentationDataset

class COCOsegmentation(SegmentationDataset):
    CAT_LIST = [0, 5, 2, 13, 18, 20]
    NUM_CLASS = 6

    def __init__(self, root='/home/ubuntu', split='train', mode=None,
transform=None, **kwargs):
        super(COCOsegmentation, self).__init__(root, split, mode,
transform, **kwargs)
        from pycocotools.coco import COCO
        from pycocotools import mask
        if split == 'train':
            print('train set')
            ann_file = os.path.join(root,
'annotations/instances_train2017.json')
            ids_file = os.path.join(root, 'annotations/train_ids.mx')
            self.root = os.path.join(root, 'train2017')
        else:
            print('val set')
            ann_file = os.path.join(root,
'annotations/instances_val2017.json')
            ids_file = os.path.join(root, 'annotations/val_ids.mx')
            self.root = os.path.join(root, 'val2017')
```

```

self.coco = COCO(ann_file)
self.coco_mask = mask
if os.path.exists(ids_file):
    with open(ids_file, 'rb') as f:
        self.ids = pickle.load(f)
else:
    ids = list(self.coco.imgs.keys())
    self.ids = self._preprocess(ids, ids_file)
self.transform = transform

def __getitem__(self, index):
    coco = self.coco
    img_id = self.ids[index]
    img_metadata = coco.loadImgs(img_id)[0]
    path = img_metadata['file_name']
    img = Image.open(os.path.join(self.root, path)).convert('RGB')
    cocotarget = coco.loadAnns(coco.getAnnIds(imgIds=img_id))
    mask = Image.fromarray(self._gen_seg_mask(
        cocotarget, img_metadata['height'], img_metadata['width']))
    if self.mode == 'train':
        img, mask = self._sync_transform(img, mask)
    elif self.mode == 'val':
        img, mask = self._val_sync_transform(img, mask)
    else:
        assert self.mode == 'testval'
        img, mask = self._img_transform(img),
self._mask_transform(mask)
    if self.transform is not None:
        img = self.transform(img)
    return img, mask, self.ids[index]

def __len__(self):
    return len(self.ids)

def _mask_transform(self, mask):
    return torch.LongTensor(np.array(mask).astype('int32'))

def _gen_seg_mask(self, target, h, w):
    mask = np.zeros((h, w), dtype=np.uint8)
    coco_mask = self.coco_mask
    for instance in target:
        rle = coco_mask.frPyObjects(instance['segmentation'], h, w)
        m = coco_mask.decode(rle)
        cat = instance['category_id']
        if cat in self.CAT_LIST:
            c = self.CAT_LIST.index(cat)
        else:
            continue
        if len(m.shape) < 3:

```

```

        mask[:, :] += (mask == 0) * (m * c)
    else:
        mask[:, :] += (mask == 0) * (((np.sum(m, axis=2)) > 0) *
c).astype(np.uint8)
    return mask

def _preprocess(self, ids, ids_file):
    tbar = trange(len(ids))
    new_ids = []
    for i in tbar:
        img_id = ids[i]
        cocotarget =
self.coco.loadAnns(self.coco.getAnnIds(imgIds=img_id))
        img_metadata = self.coco.loadImgs(img_id)[0]
        mask = self._gen_seg_mask(cocotarget, img_metadata['height'],
img_metadata['width'])
        if (mask > 0).sum() > 1000:
            new_ids.append(img_id)
            tbar.set_description('Doing: {}/{}'.format(i, len(ids)), got {} qualified images'.
\
                                format(i, len(ids), len(new_ids)))
    with open(ids_file, 'wb') as f:
        pickle.dump(new_ids, f)
    return new_ids

@property
def classes(self):
    return ('background', 'airplane', 'bicycle', 'bird', 'boat',
'person')

```

Код модели нейронной сети

```

import torch.nn as nn
import torch.nn.functional as F
from .base_models.vgg import vgg16

class FCN(nn.Module):
    def __init__(self, nclass, backbone='vgg16', aux=False,
pretrained_base=True,
                    norm_layer=nn.BatchNorm2d, **kwargs):
        super(FCN, self).__init__()
        self.aux = aux
        if backbone == 'vgg16':
            self.pretrained = vgg16(pretrained=pretrained_base).features
        else:
            raise RuntimeError('unknown backbone: {}'.format(backbone))
        self.head = _FCNHead(512, nclass, norm_layer)
        if aux:
            self.auxlayer = _FCNHead(512, nclass, norm_layer)

```

```

        self.__setattr__('exclusive', ['head', 'auxlayer'] if aux else
['head'])

    def forward(self, x):
        size = x.size()[2:]
        pool5 = self.pretrained(x)
        outputs = []
        out = self.head(pool5)
        out = F.interpolate(out, size, mode='bilinear',
align_corners=True)
        outputs.append(out)

        if self.aux:
            auxout = self.auxlayer(pool5)
            auxout = F.interpolate(auxout, size, mode='bilinear',
align_corners=True)
            outputs.append(auxout)
        return tuple(outputs)

class _FCNHead(nn.Module):
    def __init__(self, in_channels, channels, norm_layer=nn.BatchNorm2d,
**kwargs):
        super(_FCNHead, self).__init__()
        inter_channels = in_channels // 4
        self.block = nn.Sequential(
            nn.Conv2d(in_channels, inter_channels, 3, padding=1,
bias=False),
            norm_layer(inter_channels),
            nn.ReLU(inplace=True),
            nn.Dropout(0.1),
            nn.Conv2d(inter_channels, channels, 1)
        )

    def forward(self, x):
        return self.block(x)

if __name__ == '__main__':
    model = FCN(6)
    print(model)

```

Код обучения модели нейронной сети

```

import argparse
import time
import datetime
import os
import shutil
import sys

cur_path = os.path.abspath(os.path.dirname(__file__))

```

```

root_path = os.path.split(cur_path)[0]
sys.path.append(root_path)

import torch
import torch.nn as nn
import torch.utils.data as data
import torch.backends.cudnn as cudnn

from torchvision import transforms
from core.utils.loss import get_segmentation_loss
from core.utils.distributed import *
from core.utils.logger import setup_logger
from core.utils.lr_scheduler import WarmupPolyLR
from core.utils.score import SegmentationMetric
from core.data.dataloader import COCOSegmentation
from core.models.fcn import FCN
from core.utils.distributed import *

def parse_args():
    parser = argparse.ArgumentParser(description='Semantic Segmentation
Training With Pytorch')
    # model and dataset
    parser.add_argument('--base-size', type=int, default=520,
                        help='base image size')
    parser.add_argument('--crop-size', type=int, default=480,
                        help='crop image size')
    parser.add_argument('--workers', '-j', type=int, default=4,
                        metavar='N', help='dataloader threads')
    parser.add_argument('--batch-size', type=int, default=4, metavar='N',
                        help='input batch size for training (default:
8)')
    parser.add_argument('--epochs', type=int, default=50, metavar='N',
                        help='number of epochs to train (default: 50)')
    parser.add_argument('--lr', type=float, default=1e-4, metavar='LR',
                        help='learning rate (default: 1e-4)')
    parser.add_argument('--momentum', type=float, default=0.9,
metavar='M',
                        help='momentum (default: 0.9)')
    parser.add_argument('--weight-decay', type=float, default=1e-4,
metavar='M',
                        help='w-decay (default: 5e-4)')
    parser.add_argument('--aux', action='store_true', default=False,
                        help='Auxiliary loss')
    parser.add_argument('--resume', type=str, default=None)
    args = parser.parse_args()

    return args

class Trainer(object):

```

```

def __init__(self, args):
    self.args = args
    self.device = torch.device(args.device)

    # image transform
    input_transform = transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize([.485, .456, .406], [.229, .224, .225]),
    ])
    # dataset and dataloader
    data_kwargs = {'transform': input_transform, 'base_size':
args.base_size, 'crop_size': args.crop_size}
    train_dataset = COCOSegmentation(split='train', mode='train',
**data_kwargs)
    val_dataset = COCOSegmentation(split='val', mode='val',
**data_kwargs)
    args.iters_per_epoch = len(train_dataset) // (args.num_gpus *
args.batch_size)
    args.max_iters = args.epochs * args.iters_per_epoch

    train_sampler = make_data_sampler(train_dataset, shuffle=True,
distributed=args.distributed)
    train_batch_sampler = make_batch_data_sampler(train_sampler,
args.batch_size, args.max_iters)
    val_sampler = make_data_sampler(val_dataset, False,
args.distributed)
    val_batch_sampler = make_batch_data_sampler(val_sampler,
args.batch_size)

    self.train_loader = data.DataLoader(dataset=train_dataset,
batch_sampler=train_batch_sampler,
                                     num_workers=args.workers,
                                     pin_memory=True)
    self.val_loader = data.DataLoader(dataset=val_dataset,
batch_sampler=val_batch_sampler,
                                     num_workers=args.workers,
                                     pin_memory=True)

    BatchNorm2d = nn.SyncBatchNorm if args.distributed else
nn.BatchNorm2d
    self.model = FCN(6, backbone="vgg16", aux=args.aux,
norm_layer=BatchNorm2d)
    if args.resume:
        if os.path.isfile(args.resume):
            name, ext = os.path.splitext(args.resume)
            assert ext == '.pkl' or '.pth', 'Sorry only .pth and .pkl
files supported.'

```

```

        print('Resuming training, loading
{}...'.format(args.resume))
        self.model.load_state_dict(torch.load(args.resume,
map_location=lambda storage, loc: storage))

        self.criterion = get_segmentation_loss().to(self.device)

        params_list = list()
        if hasattr(self.model, 'pretrained'):
            params_list.append({'params':
self.model.pretrained.parameters(), 'lr': args.lr})
        if hasattr(self.model, 'exclusive'):
            for module in self.model.exclusive:
                params_list.append({'params': getattr(self.model,
module).parameters(), 'lr': args.lr * 10})
            self.optimizer = torch.optim.SGD(params_list,
                                            lr=args.lr,
                                            momentum=args.momentum,
                                            weight_decay=args.weight_decay)

        self.lr_scheduler = WarmupPolyLR(self.optimizer,
                                        max_iters=args.max_iters,
                                        power=0.9)

        if args.distributed:
            self.model = nn.parallel.DistributedDataParallel(self.model,
device_ids=[args.local_rank],
output_device=args.local_rank)
            self.metric = SegmentationMetric(train_dataset.num_class)
            self.best_pred = 0.0
            def train(self):
                save_to_disk = get_rank() == 0
                epochs, max_iters = self.args.epochs, self.args.max_iters

                start_time = time.time()
                print('Start training, Total Epochs: {:d} = Total Iterations
{:d}'.format(epochs, max_iters))
                self.model.train()
                for iteration, (images, targets, _) in
enumerate(self.train_loader):
                    iteration = iteration + 1
                    images = images.to(self.device)
                    targets = targets.to(self.device)
                    outputs = self.model(images)
                    loss_dict = self.criterion(outputs, targets)
                    losses = sum(loss for loss in loss_dict.values())
                    loss_dict_reduced = reduce_loss_dict(loss_dict)
                    losses_reduced = sum(loss for loss in

```

```

loss_dict_reduced.values())

        self.optimizer.zero_grad()
        losses.backward()
        self.optimizer.step()
        self.lr_scheduler.step()
        eta_seconds = ((time.time() - start_time) / iteration) *
(max_iters - iteration)
        eta_string =
str(datetime.timedelta(seconds=int(eta_seconds)))

        if iteration % (max_iters // epochs) == 0:
            print(
                "Epochs: {:d}/{:d} || Loss: {:.4f} ".format(
                    iteration // (max_iters // epochs), epochs,
losses_reduced.item()), end="")
                self.validation()
                self.model.train()
            save_checkpoint(self.model, self.args, is_best=False)
def validation(self):
    # total_inter, total_union, total_correct, total_label = 0, 0, 0,
0
    is_best = False
    self.metric.reset()
    if self.args.distributed:
        model = self.model.module
    else:
        model = self.model
    torch.cuda.empty_cache() # TODO check if it helps
    model.eval()
    for i, (image, target, filename) in enumerate(self.val_loader):
        if i > 100:
            break
        image = image.to(self.device)
        target = target.to(self.device)

        with torch.no_grad():
            outputs = model(image)
            self.metric.update(outputs[0], target)
    a, mIoU = self.metric.get()
    print("Validation mIoU: {:.3f}".format(mIoU))

    if mIoU > self.best_pred:
        is_best = True
        self.best_pred = mIoU
    save_checkpoint(self.model, self.args, is_best)
    synchronize()

def save_checkpoint(model, args, is_best=False):

```

```

directory = "/home/ubuntu"
if not os.path.exists(directory):
    os.makedirs(directory)
filename = "fcn_vgg16_coco.pth"
filename = os.path.join(directory, filename)

if args.distributed:
    model = model.module
torch.save(model.state_dict(), filename)
if is_best:
    best_filename = "fcn_vgg16_coco_best_model.pth"
    best_filename = os.path.join(directory, best_filename)
    shutil.copyfile(filename, best_filename)

if __name__ == '__main__':
    args = parse_args()

    num_gpus = int(os.environ["WORLD_SIZE"]) if "WORLD_SIZE" in
os.environ else 1
    args.num_gpus = num_gpus
    args.distributed = num_gpus > 1
    if torch.cuda.is_available():
        cudnn.benchmark = True
        args.device = "cuda"
    else:
        args.distributed = False
        args.device = "cpu"
    if args.distributed:
        torch.cuda.set_device(args.local_rank)
        torch.distributed.init_process_group(backend="nccl",
init_method="env://")
        synchronize()
    args.lr = args.lr * num_gpus

    trainer = Trainer(args)
    trainer.train()
    torch.cuda.empty_cache()

```

Текст программы демонстрации работы модели нейронной сети

```

import os
import sys
import argparse
import torch
import matplotlib.pyplot as plt

cur_path = os.path.abspath(os.path.dirname(__file__))
root_path = os.path.split(cur_path)[0]
sys.path.append(root_path)

```

```

from core.models.fcn import FCN
from scripts.visualize import display_instances
from torchsummary import summary
from torchvision import transforms
from PIL import Image

def demo():
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

    # image transform
    transform = transforms.Compose([
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225]),
    ])
    image = Image.open("image.jpg").convert('RGB')
    images = transform(image).unsqueeze(0).to(device)
    model = FCN(6, backbone="vgg16", pretrained_base=False)
    model.load_state_dict(torch.load("coco.pth"))
    summary(model, (3, 512, 512))
    print('Finished loading model!')
    model.eval()
    with torch.no_grad():
        output = model(images)

    pred = torch.argmax(output[0], 1).squeeze(0).cpu().data.numpy()
    cat_list = [0, 5, 2, 13, 18, 20]

    display_instances(image=image,
                    masks=pred,
                    class_ids=cat_list,
                    )
if __name__ == '__main__':
    demo()

```

2.7. Сегментация одежды на изображениях. SegNet

В работе разрабатывается и обучается модель для сегментации одежды на изображениях. Используются Torch и SegNet. Torch - это гибкий фреймворк глубокого обучения для создания и обучения нейронных сетей, обладающий эффективностью и интеграцией с вычислительными ускорителями. SegNet - это сверточная нейронная сеть, специально разработанная для сегментации изображений. Она состоит из

кодера для извлечения признаков и декодера для восстановления сегментированных изображений.

Алгоритм работы

1. Чтение изображения с помощью библиотеки `matplotlib.image` и вывод его размеров.
2. Чтение изображения с помощью библиотеки `PIL` и преобразование его в массив `numpy` для дальнейшей обработки.
3. Чтение маски сегментации и отображение её.
4. Определение ограничивающего прямоугольника актуальной маски и обрезка исходной маски.
5. Разделение масок на человека, одежду и фон с последующим отображением.

Использованные функции

Название функции	Назначение
<code>pd.read_csv()</code>	Загрузка данных из CSV файла в формате таблицы <code>DataFrame</code>
<code>np.array()</code>	Преобразование данных в массив <code>numpy</code>
<code>cv2.imread()</code>	Загрузка изображения из файла с использованием <code>OpenCV</code>
<code>torch.nn.Module()</code>	Определение модели нейронной сети в <code>PyTorch</code>
<code>torch.optim.Adam()</code>	Инициализация оптимизатора <code>Adam</code> для обучения модели
<code>pl.Trainer()</code>	Создание экземпляра класса <code>Trainer</code> для управления обучением в <code>PyTorch Lightning</code>
<code>plt.imshow()</code>	Отображение изображения с помощью <code>Matplotlib</code>
<code>plt.show()</code>	Отображение всех графиков и изображений в <code>Matplotlib</code>
<code>plt.figure()</code>	Создание нового объекта <code>Figure</code> в <code>Matplotlib</code>
<code>plt.subplot()</code>	Создание сетки подграфиков в <code>Matplotlib</code>

<code>torch.utils.data.Dataset()</code>	Определение пользовательского класса Dataset для работы с данными
<code>torch.utils.data.DataLoader()</code>	Создание загрузчика данных для эффективной подачи данных в модель
<code>torch.nn.Conv2d()</code>	Создание слоя свертки в нейронной сети
<code>torch.nn.ConvTranspose2d()</code>	Создание транспонированной сверточной сети в PyTorch
<code>torch.nn.BatchNorm2d()</code>	Создание слоя нормализации данных в нейронной сети
<code>torch.nn.ReLU()</code>	Создание функции активации ReLU в нейронной сети
<code>pl.callbacks.ModelCheckpoint()</code>	Контроль сохранения лучших моделей во время обучения в PyTorch Lightning
<code>pl.callbacks.EarlyStopping()</code>	Остановка обучения модели при достижении определенного условия в PyTorch Lightning
<code>transforms.Compose()</code>	Композиция преобразований изображений в torchvision
<code>torchvision.models.segmentation.fcn_resnet50()</code>	Загрузка предварительно обученной модели FCN ResNet-50 для сегментации изображений
<code>torchvision.utils.save_image()</code>	Сохранение изображения в файл с помощью torchvision
<code>torchmetrics.IoU()</code>	Вычисление метрики Intersection over Union для оценки качества сегментации
<code>torchsummary.summary()</code>	Вывод структуры модели нейронной сети в виде таблицы
<code>KFold()</code>	Разделение данных на обучающие и тестовые фолды с использованием кросс-валидации из scikit-learn
<code>torch.nn.functional.where()</code>	Выбор элементов из двух тензоров в зависимости от условия в PyTorch
<code>torchvision.transforms.ToPILImage()</code>	Преобразование тензора в изображение формата PIL
<code>os.path.join()</code>	Объединение путей к файлам и каталогам в операционной системе
<code>Image.open()</code>	Открытие изображения с использованием библиотеки PIL
<code>np.min()</code>	Вычисление минимального значения в массиве numpy
<code>np.max()</code>	Вычисление максимального значения в массиве numpy

np.nonzero()

Нахождение индексов ненулевых элементов в массиве numpy

Скриншоты



Текст программы

```
import pandas as pd # Библиотека для работы с данными в формате таблицы (DataFrame)
import numpy as np # Библиотека для работы с числовыми массивами и матрицами
import cv2 # Библиотека OpenCV для обработки изображений
import torch # Основная библиотека для создания и обучения нейронных сетей
import torch.nn as nn # Модуль PyTorch для создания нейронных сетей и слоев
import pytorch_lightning as pl # Библиотека для облегчения обучения нейронных сетей на базе PyTorch
import matplotlib.image as img # Модуль Matplotlib для работы с изображениями
import matplotlib.pyplot as plt # Модуль Matplotlib для построения графиков
import seaborn as sns # Библиотека для визуализации данных, основанная на Matplotlib
import torchmetrics # Библиотека для вычисления метрик во время обучения нейронных сетей
from torch.utils.data import Dataset, DataLoader # Модуль PyTorch для создания пользовательских наборов данных и загрузчиков данных
```

```

from pytorch_lightning import Trainer, seed_everything # Класс Trainer и
функция seed_everything для управления обучением среды PyTorch Lightning
from pytorch_lightning import Callback # Базовый класс для создания
пользовательских обратных вызовов PyTorch Lightning
from lightning.pytorch import loggers as pl_loggers # Модуль loggers из
пакета lightning.pytorch для журналирования результатов обучения
from pytorch_lightning.callbacks import ModelCheckpoint, EarlyStopping #
Классы ModelCheckpoint и EarlyStopping для управления обучением среды
PyTorch Lightning
import torchvision.transforms as transforms # Модуль torchvision для
преобразования изображений from torchsummary import summary # Функция для
вывода информации о структуре модели
from sklearn.model_selection import KFold # Модуль sklearn для разделения
данных на обучающие и тестовые фолды
import torch.nn.functional as F # Функциональный модуль PyTorch для
применения функций активации и других операций
import torchvision # Библиотека для компьютерного зрения в PyTorch import
os # Модуль для работы с операционной системой
from PIL import Image # Класс Image из библиотеки PIL для работы с
изображениями в формате PIL

from lightning.pytorch import seed_everything # Импорт функции
seed_everything из модуля lightning.pytorch для установки одинакового
начального состояния генератора случайных чисел seed_everything(42,
workers=True) # Установка seed (зерна) для генератора случайных чисел на
42 для обеспечения воспроизводимости результатов обучения; параметр
workers=True гарантирует установку seed для всех рабочих потоков

# Чтение изображения с помощью функции imread из модуля image библиотеки
matplotlib image = img.imread('dataseg/Train/Image/10315.jpg')
# Вывод размеров изображения (высота, ширина, количество каналов)
print(image.shape)
# Отображение изображения plt.imshow(image)
plt.show()

# Чтение изображения с помощью библиотеки PIL и преобразование его в
массив numpy # img = np.array(Image.open(self.images[index])) # open
colour image
# Вывод размеров изображения в формате numpy массива
np.array(Image.open('dataseg/Train/Image/10315.jpg')).shape# Чтение маски
сегментации from PIL import Image
# Открытие изображения и преобразование его в оттенки серого
mask =
np.array(Image.open('dataseg/Train/Mask/10315.jpg.png').convert('L'),
dtype=np.float32) # Вывод размеров маски (высота, ширина)
print(mask.shape)

# Отображение маски сегментации plt.imshow(mask, cmap="gray") plt.show()

```

```

# Дополнительный код: определение строк и столбцов актуальной маски (без
отступов) # Чтение маски
mask =
np.array(Image.open('dataseg/Train/Mask/10315.jpg.png').convert('L'),
dtype=np.float32) # Применение порогового значения для создания бинарной
маски
binary_mask = (mask < 255).astype(np.uint8)
# Нахождение ограничивающего прямоугольника актуальной маски
nonzero_indices = np.nonzero(binary_mask)
min_row, min_col = np.min(nonzero_indices, axis=1) max_row, max_col =
np.max(nonzero_indices, axis=1) print("min_row, min_col", min_row,
min_col) print("max_row, max_col", max_row, max_col)
# Обрезка исходной маски с помощью ограничивающего прямоугольника
cropped_mask = mask[min_row:max_row+1, min_col:max_col+1]
# Отображение обрезанной маски plt.imshow(cropped_mask, cmap="gray")
plt.show()

# Разделение масок на персону, одежду и фон
# Если условие выполняется, устанавливаем значение 255 (белый), в
противном случае 0 (черный)
person_mask = np.where(mask >= 150, 255, 0).astype(np.uint8)
clothes_mask = np.where((mask > 60) & (mask < 150), 255,
0).astype(np.uint8) background_mask = np.where(mask < 10, 255,
0).astype(np.uint8)

# Отображение масок в отдельности с помощью imshow()
plt.figure(figsize=(12, 11))

plt.subplot(1, 3, 1) plt.imshow(person_mask, cmap='gray')
plt.title('Person Mask')

plt.subplot(1, 3, 2) plt.imshow(clothes_mask, cmap='gray')
plt.title('Clothes Mask')

plt.subplot(1, 3, 3) plt.imshow(background_mask, cmap='gray')
plt.title('Background Mask')

class DownConv2(nn.Module):
def    init (self, chin, chout, kernel_size):
super().    init    () self.seq = nn.Sequential(
nn.Conv2d(in_channels=chin, out_channels=chout, kernel_size=kernel_size,
padding=kernel_size//2, bias=False),
nn.BatchNorm2d(chout), nn.ReLU(),
nn.Conv2d(in_channels=chout, out_channels=chout, kernel_size=kernel_size,
padding=kernel_size//2, bias=False),
nn.BatchNorm2d(chout), nn.ReLU(),
)
self.mp = nn.MaxPool2d(kernel_size=2, return_indices=True)

```

```

def forward(self, x): y = self.seq(x)
pool_shape = y.shape # pool_shape = y.size() y, indices = self.mp(y)
return y, indices, pool_shape

class DownConv3(nn.Module):
def    init (self, chin, chout, kernel_size): super(). init ()
self.seq = nn.Sequential(
nn.Conv2d(in_channels=chin, out_channels=chout, kernel_size=kernel_size,
padding=kernel_size//2, bias=False),
nn.BatchNorm2d(chout), nn.ReLU(),
nn.Conv2d(in_channels=chout, out_channels=chout, kernel_size=kernel_size,
padding=kernel_size//2, bias=False),
nn.BatchNorm2d(chout), nn.ReLU(),
nn.Conv2d(in_channels=chout, out_channels=chout, kernel_size=kernel_size,
padding=kernel_size//2, bias=False),
nn.BatchNorm2d(chout), nn.ReLU(),
)
self.mp = nn.MaxPool2d(kernel_size=2, return_indices=True)

def forward(self, x): y = self.seq(x)
pool_shape = y.shape # pool_shape = y.size() y, indices = self.mp(y)
return y, indices, pool_shape

# Define Decoder Blocks class UpConv2(nn.Module):
def    init (self, chin, chout, kernel_size): super(). init ()
self.seq = nn.Sequential(
nn.Conv2d(in_channels=chin, out_channels=chin, kernel_size=kernel_size,
padding=kernel_size//2, bias=False),
nn.BatchNorm2d(chin), nn.ReLU(),
nn.Conv2d(in_channels=chin, out_channels=chout, kernel_size=kernel_size,
padding=kernel_size//2, bias=False),
nn.BatchNorm2d(chout), nn.ReLU(),
)
self.mup = nn.MaxUnpool2d(kernel_size=2)

def forward(self, x, indices, output_size):
y = self.mup(x, indices, output_size=output_size) y = self.seq(y)
return y

class UpConv3(nn.Module):
def    init (self, chin, chout, kernel_size): super(). init ()
self.seq = nn.Sequential(
nn.Conv2d(in_channels=chin, out_channels=chin, kernel_size=kernel_size,
padding=kernel_size//2, bias=False),
nn.BatchNorm2d(chin), nn.ReLU(),
nn.Conv2d(in_channels=chin, out_channels=chin, kernel_size=kernel_size,
padding=kernel_size//2, bias=False),
nn.BatchNorm2d(chin), nn.ReLU(),
)

```

```

nn.Conv2d(in_channels=chin, out_channels=chout, kernel_size=kernel_size,
padding=kernel_size//2, bias=False),
nn.BatchNorm2d(chout), nn.ReLU(),
)
self.mup = nn.MaxUnpool2d(kernel_size=2)

def forward(self, x, indices, output_size):
y = self.mup(x, indices, output_size=output_size) y = self.seq(y)
return y
class Segmentation(pl.LightningModule): def      init (self, kernel_size):
super(). init      ()
self.out_channels = 3
self.bn_input = nn.BatchNorm2d(3) # Encoder
self.dc1 = DownConv2(3, 64, kernel_size=kernel_size) self.dc2 =
DownConv2(64, 128, kernel_size=kernel_size) self.dc3 = DownConv3(128,
256, kernel_size=kernel_size) self.dc4 = DownConv3(256, 512,
kernel_size=kernel_size) # Decoder
self.uc4 = UpConv3(512, 256, kernel_size=kernel_size) self.uc3 =
UpConv3(256, 128, kernel_size=kernel_size) self.uc2 = UpConv2(128, 64,
kernel_size=kernel_size)
self.uc1 = UpConv2(64, 3, kernel_size=kernel_size)

self.conv3to1 = nn.Conv2d(3, 1, kernel_size=1) self.sigmoid =
nn.Sigmoid()

# classification metrics self.loss_fn = nn.BCELoss()
self.f1_score = torchmetrics.classification.BinaryF1Score()
self.example_input_array = torch.zeros((1, 3, 400, 300),
dtype=torch.float32) self.save_hyperparameters()

def forward(self, x): # Encoder
x = self.bn_input(x)
x, mp1_indices, shape1 = self.dc1(x) x, mp2_indices, shape2 = self.dc2(x)
x, mp3_indices, shape3 = self.dc3(x) x, mp4_indices, shape4 = self.dc4(x)

# Decoder
x = self.uc4(x, mp4_indices, output_size=shape4) x = self.uc3(x,
mp3_indices, output_size=shape3) x = self.uc2(x, mp2_indices,
output_size=shape2) x = self.uc1(x, mp1_indices, output_size=shape1)

x = self.conv3to1(x) x = self.sigmoid(x)

return x

def training_step(self, batch, batch_idx): x, y = batch
x = transforms.functional.resize(x, size=[400, 300]) # resize image to
smaller size (400 x 300 px) to consume less memory while training
y = y.unsqueeze(1) # Add channel dimension to target tensor (to match
model's output size [batch_size, 1, height, width])

```

```

# print('x.shape', x.shape) # torch.Size([10, 3, 400, 300])
# print('y.shape', y.shape) # torch.Size([10, 1, 1440, 1920]) y_hat =
self.forward(x)
# print('y_hat.shape output size', y_hat.shape) # torch.Size([10, 1, 400,
300])
y_hat = transforms.functional.resize(y_hat, size=[1440, 1080]) # resize
output back to 1440 x 1080
px
# print('y_hat.shape resized', y_hat.shape) # torch.Size([10, 1, 1440,
1080]) pad_left_right = int((1920-y_hat.shape[3]) / 2) # calculate
padding required on the width
y_hat = F.pad(y_hat, (pad_left_right, pad_left_right), mode='constant',
value=0) # add padding to
left & right to fit shape of mask
# print('y_hat.shape padded', y_hat.shape) # torch.Size([10, 1, 1440,
1920]) loss = self.loss_fn(y_hat, y)
self.log('train_loss', loss) f1_score = self.f1_score(y_hat, y)
self.log('train_f1', f1_score) return loss
def validation_step(self, batch, batch_idx): x, y = batch
x = transforms.functional.resize(x, size=[400, 300]) # resize image to
400 x 300 px for training y = y.unsqueeze(1) # Add channel dimension to
target tensor
y_hat = self.forward(x)
y_hat = transforms.functional.resize(y_hat, size=[1440, 1080]) # resize
output back to 1440 x 1080
px
pad_left_right = int((1920-y_hat.shape[3]) / 2)
y_hat = F.pad(y_hat, (pad_left_right, pad_left_right), mode='constant',
value=0) # add padding to
left & right to fit shape of mask loss = self.loss_fn(y_hat, y)
self.log('val_loss', loss)
f1_score = self.f1_score(y_hat, y) self.log('val_f1_score', f1_score)

def test_step(self, batch, batch_idx): x, y = batch
x = transforms.functional.resize(x, size=[400, 300]) # resize image to
400 x 300 px for training y = y.unsqueeze(1) # Add channel dimension to
target tensor
y_hat = self.forward(x)
y_hat = transforms.functional.resize(y_hat, size=[1440, 1080]) # resize
output back to 1440 x 1080
px
pad_left_right = int((1920-y_hat.shape[3]) / 2)
y_hat = F.pad(y_hat, (pad_left_right, pad_left_right), mode='constant',
value=0) # add padding to
left & right to fit shape of mask loss = self.loss_fn(y_hat, y)
self.log('test_loss', loss)
f1_score = self.f1_score(y_hat, y) self.log('test_f1', f1_score)

def configure_optimizers(self):

```

```

optimizer = torch.optim.Adam(self.parameters(), lr=1e-3) return optimizer

model = Segmentation(kernel_size=3) model
from skimage.transform import resize

class ClothingDataset(Dataset):
    def __init__(self, img_dir, name="Train"):
        # img_dir till "Train/Test"
        self.images = sorted([img_dir + "/Image/" + i for i in os.listdir(img_dir +
        "/Image/") if ".ipynb" not in i]) # sort list in ascending order - so
        that image IDs will match
        self.masks = sorted([img_dir + "/Mask/" + i for i in os.listdir(img_dir +
        "/Mask/") if ".ipynb" not in i]) # sort list in ascending order - so that
        image IDs will match
        if "Train" in img_dir:
            train_ids_max = int(0.8*len(self.images)) val_ids_max = len(self.images)
            self.images_train = self.images[0:train_ids_max] self.masks_train =
            self.masks[0:train_ids_max]
            self.images_val = self.images[train_ids_max:val_ids_max] self.masks_val =
            self.masks[train_ids_max:val_ids_max]

        if "Train" == name:
            self.images = self.images_train self.masks = self.masks_train
        if "Val" == name:
            self.images = self.images_val self.masks = self.masks_val

    def __len__(self):
        return len(self.images)

    def __getitem__(self, index):
        # print("training:", self.images[index])
        img = np.array(Image.open(self.images[index])) # open colour image
        mask_overall = np.array(Image.open(self.masks[index]).convert('L')) #
        open mask image, convert to greyscale
        # print("mask:", self.masks[index]) # print out path of mask to verify
        that it matches the training image
        cloth_mask = (mask_overall < 150) & (mask_overall > 60) # MODIFIED CODE:
        change mask_overall < 140 --> 150
        annotation_map = np.zeros(mask_overall.shape)
        annotation_map[np.where(cloth_mask == True)] = 1

        # Crop the original mask using the bounding box (the values of the
        bounding box were identified previously before section 2.1) # MODIFIED
        CODE
        # From earlier: min_row, min_col 173 568 # max_row, max_col 1281 1399
        cropped_annotation_map = annotation_map[173:1281+1, 568:1399+1]

        # Resize to (1440, 1080) to match size of the training image

```

```

resized_cropped_annotation_map = resize(cropped_annotation_map, (1440,
1080), mode='constant', anti_aliasing=True)

# Apply padding to the left & right to resize mask back to original
required size of (1440, 1920) pad_left_right = int((1920 -
resized_cropped_annotation_map.shape[1]) // 2) # calculate padding
required on the width
padded_resized_cropped_annotation_map =
np.pad(resized_cropped_annotation_map, ((0, 0), (pad_left_right,
pad_left_right)), mode='constant')

# Thresholding to ensure values are either 0 or 1
padded_resized_cropped_annotation_map[padded_resized_cropped_annotation_m
ap < 0.5] = 0
padded_resized_cropped_annotation_map[padded_resized_cropped_annotation_m
ap >= 0.5] = 1

# return float tensor for color image (x) & mask map (y)
# x: (height, width, channel) --> (channel, width, height) --> (channel,
height, width)
return torch.FloatTensor(img).transpose(2, 0).transpose(1, 2),
torch.FloatTensor(padded_res

class ClothingDatasetModule(pl.LightningDataModule): def setup(self,
stage):
self.train_data = ClothingDataset("dataseg/Train", name="Train")
self.val_data = ClothingDataset("dataseg/Train", name="Val")
self.test_data = ClothingDataset("dataseg/Test", name="Test")

def train_dataloader(self):
return DataLoader(self.train_data, batch_size=10)

def val_dataloader(self):
return DataLoader(self.val_data, batch_size=10)

def test_dataloader(self):
return DataLoader(self.test_data, batch_size=10) data_module =
ClothingDatasetModule()

```

2.8. Сегментация изображения для двух классов. U-Net, OXFORD-III Pet

Будем обучать нейронную сеть сегментировать изображение. В качестве базы данных возьмём оригинальное изображение и изображение с контурами объектов. После обучения нейронная сеть должна корректно сегментировать изображение для следующих классов: собака, кошка.

Используется набор данных OXFORD-IIIPet, содержащий изображения различных пород собак и кошек с аннотациями [38]. Набор данных включает обучающие, валидационные и тестовые изображения с разметкой. OXFORD-IIIPet — набор данных с более чем 7,000 изображений различных пород кошек и собак, полезный для тренировки и оценки моделей компьютерного зрения.

Реализована и обучена сверточная модель нейронной сети на базе архитектуры UNet, известной своей способностью восстанавливать детали объектов на разных уровнях абстракции. UNet — это сверточная нейронная сеть для сегментации изображений, состоящая из энкодера и декодера, что обеспечивает высокую точность и детальное восстановление границ объектов даже при ограниченных данных.

Алгоритм работы

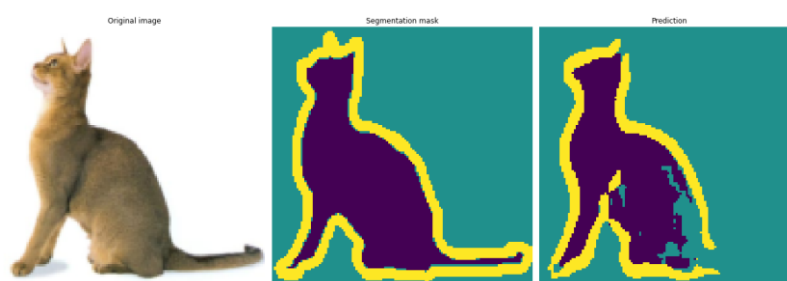
1. Инициализируем наборы данных для обучения и валидации модели семантической сегментации на основе заданного датасета.
2. Создадим загрузчики данных для обучающего и валидационного набора данных. Зафиксируем данные в памяти для повышения производительности.
3. Определим тренировочную функцию, которую будем использовать для обучения модели семантической сегментации.
4. Создадим функцию для оценки производительности модели без обновления градиентов, что позволит провести оценку на валидационном или тестовом наборе данных.
5. Создадим функцию, управляющую полным циклом обучения и валидации модели. В течение заданного количества итераций функция тренирует модель, оценивает её на валидационном наборе.

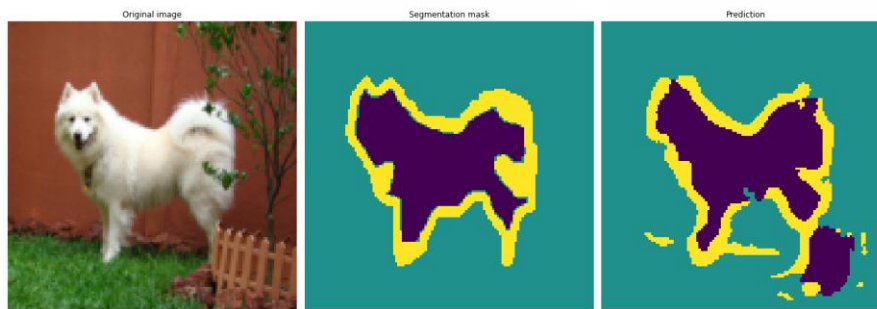
6. Создадим блок UNET, состоящий из двух последовательных сверток с размером 3×3 , с промежуточной нормализацией по батчам и нелинейной активацией. Он возвращает целую последовательность для использования в UNET, а также финальный выход, использующий свертку 1×1 для получения трехканального выхода, обычно представляющего сегментированные классы.

Использованные функции:

Имя функции	Описание
train	Функция обучения модели на обучающем наборе данных, возвращает значение потерь и точность на обучении.
evaluate	Функция оценки модели на валидационном наборе данных, возвращает значение потерь и точность на валидации.
whole_train_valid_cycle	Функция полного цикла обучения и валидации модели, выполняет обучение и оценку модели на протяжении заданного количества эпох.
conv_plus_conv	Функция создания блока UNet, состоящего из двух последовательных свёрточных слоёв, нормализации и функции активации LeakyReLU.
forward	Метод класса UNET, определяющий прямой проход данных через модель. Обработывает входное изображение и возвращает предсказание.

Результаты работы программы





Текст программы

```

import torch
from torch import nn
from torch.optim import Adam
from torch.utils.data import DataLoader
from tqdm import tqdm
import torchvision.transforms as T
from torchvision.datasets import OxfordIIIPet

# Определение преобразований для данных
transform = T.Compose([
    T.Resize((128, 128)),
    T.ToTensor(),
])

target_transform = T.Compose([
    T.Resize((128, 128)),
    T.PILToTensor(),
    T.Lambda(lambda x: (x - 1).long())
])

# Загрузка обучающего и валидационного наборов данных
train_dataset = OxfordIIIPet('/home/jupyter/mnt/datasets/pets',
transform=transform, target_transform=target_transform,
target_types='segmentation')
valid_dataset = OxfordIIIPet('/home/jupyter/mnt/datasets/pets',
transform=transform, split='test', target_transform=target_transform,
target_types='segmentation')

# Создание DataLoader для обучающего и валидационного наборов данных
train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True,
num_workers=8, pin_memory=True)
valid_loader = DataLoader(valid_dataset, batch_size=64, shuffle=True,
num_workers=8, pin_memory=True)

# Функция обучения модели
def train(model) -> float:
    model.train()

    train_loss = 0
    total = 0

```

```

correct = 0
for x, y in tqdm(train_loader, desc='Train'):
    bs = y.size(0)

    # Перенос данных на устройство (GPU или CPU)
    x, y = x.to(device), y.squeeze(1).to(device)
    optimizer.zero_grad()
    output = model(x)
    loss = loss_fn(output.reshape(bs, 3, -1), y.reshape(bs, -1))
    train_loss += loss.item()
    loss.backward()
    optimizer.step()
    _, y_pred = output.max(dim=1)
    total += y.size(0) * y.size(1) * y.size(2)
    correct += (y == y_pred).sum().item()
train_loss /= len(train_loader)
accuracy = correct / total
return train_loss, accuracy

# Функция оценки модели
@torch.inference_mode()
def evaluate(model, loader) -> tuple[float, float]:
    model.eval()
    total_loss = 0
    total = 0
    correct = 0
    for x, y in tqdm(loader, desc='Evaluation'):
        bs = y.size(0)
        x, y = x.to(device), y.squeeze(1).to(device)
        output = model(x)
        loss = loss_fn(output.reshape(bs, 3, -1), y.reshape(bs, -1))
        total_loss += loss.item()
        _, y_pred = output.max(dim=1)
        total += y.size(0) * y.size(1) * y.size(2)
        correct += (y == y_pred).sum().item()
    total_loss /= len(loader)
    accuracy = correct / total
    return total_loss, accuracy

# Функция полного цикла обучения и валидации
def whole_train_valid_cycle(model, num_epochs, title):
    train_loss_history, valid_loss_history = [], []
    train_accuracy_history, valid_accuracy_history = [], []
    for epoch in range(num_epochs):
        train_loss, train_accuracy = train(model)
        valid_loss, valid_accuracy = evaluate(model, valid_loader)
        train_loss_history.append(train_loss)
        valid_loss_history.append(valid_loss)
        train_accuracy_history.append(train_accuracy)

```

```

        valid_accuracy_history.append(valid_accuracy)
        clear_output()

# Функция создания блока UNet
def conv_plus_conv(in_channels: int, out_channels: int):
    return nn.Sequential(
        nn.Conv2d(
            in_channels=in_channels,
            out_channels=out_channels,
            kernel_size=3,
            stride=1,
            padding=1
        ),
        nn.BatchNorm2d(num_features=out_channels),
        nn.LeakyReLU(0.2),
        nn.Conv2d(
            in_channels=out_channels,
            out_channels=out_channels,
            kernel_size=3,
            stride=1,
            padding=1
        ),
        nn.BatchNorm2d(num_features=out_channels),
        nn.LeakyReLU(0.2),
    )

# Определение модели UNet
class UNET(nn.Module):
    def __init__(self):
        super().__init__()
        base_channels = 16
        self.down1 = conv_plus_conv(3, base_channels)
        self.down2 = conv_plus_conv(base_channels, base_channels * 2)
        self.up1 = conv_plus_conv(base_channels * 2, base_channels)
        self.up2 = conv_plus_conv(base_channels * 4, base_channels)
        self.bottleneck = conv_plus_conv(base_channels * 2, base_channels * 2)
        self.out = nn.Conv2d(in_channels=base_channels, out_channels=3,
kernel_size=1)
        self.downsample = nn.MaxPool2d(kernel_size=2, stride=2)
    def forward(self, x):
        # Сохранение слоев декодера
        residual1 = self.down1(x)
        x = self.downsample(residual1)
        residual2 = self.down2(x)
        x = self.downsample(residual2)
        x = self.bottleneck(x)
        x = nn.functional.interpolate(x, scale_factor=2)
        x = torch.cat((x, residual2), dim=1)
        x = self.up2(x)

```

```
x = nn.functional.interpolate(x, scale_factor=2)
x = torch.cat((x, residual1), dim=1)
x = self.up1(x)
x = self.out(x)
return x
```

```
# Настройка устройства, функции потерь и оптимизатора
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
loss_fn = nn.CrossEntropyLoss()
model = UNET().to(device)
optimizer = Adam(model.parameters(), lr=1e-3)
```

```
# Запуск цикла обучения и валидации
whole_train_valid_cycle(model, 15, 'UNET segmentation')
```

2.9. Определение 10 разных классов на фотографиях. TensorFlow и CIFAR-10

Разработана программа, которая автоматически определяет 10 различных классов изображений на фотографиях. В ней используются tensorflow и keras. Tensorflow – это библиотека машинного обучения, используемая для создания и обучения нейронных сетей. Tensorflow.keras - это высокоуровневый API для построения и обучения моделей глубокого обучения, базирующийся на TensorFlow.

Для обучения модели применяется набор данных CIFAR-10. Он встроен в библиотеку keras, состоит из 60 000 цветных изображений размером 32x32 пикселя, которые разделены на 10 классов. Классы включают самолеты, автомобили, птиц, кошек, оленей, собак, лягушек, лошадей, корабли и грузовики. Набор данных состоит из двух частей: набора для обучения и набора для тестирования. Набор для обучения содержит 50 000 изображений, а набор для тестирования - 10 000 изображений.

Данный набор данных широко используется для оценки и сравнения алгоритмов в области компьютерного зрения и классификации изображений. Благодаря разнообразию классов и большому количеству

изображений, CIFAR-10 представляет собой ценный ресурс для обучения и тестирования моделей машинного обучения, таких как эта нейросеть, способной определять различные классы на фотографиях.

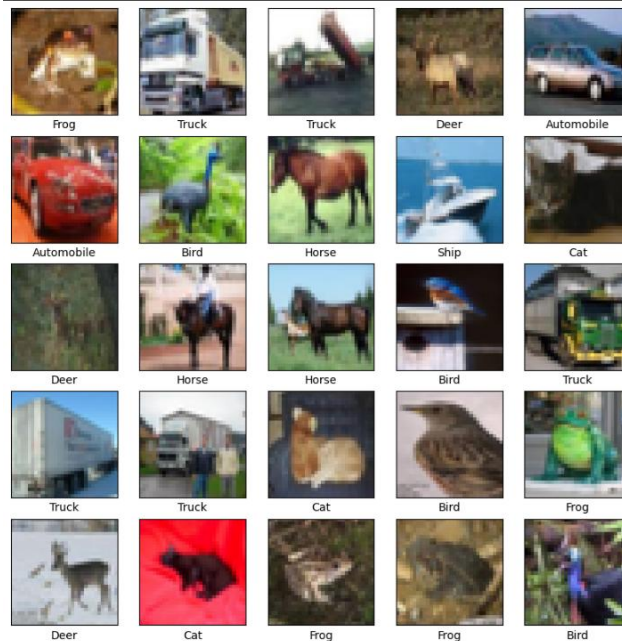
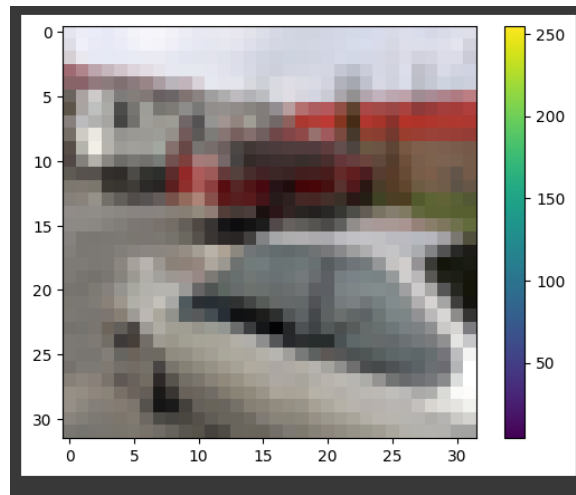
Ход работы (алгоритм)

1. Предварительная обработка изображений

До предварительной обработки изображения представляли собой многомерные массивы чисел, где каждое число соответствовало интенсивности пикселя в диапазоне от 0 до 255 для каждого канала RGB. После обработки изображения были нормализованы, так что значения пикселей теперь находятся в диапазоне от 0 до 1. Это означает, что каждое число в массиве теперь представляет относительную интенсивность пикселя в диапазоне от 0 (черный) до 1 (белый). Например, значение 0.5 означает половинную интенсивность для данного пикселя. Нормализация упрощает обучение нейронной сети, делая входные данные более однородными и уменьшая влияние выбросов. Каждое изображение в CIFAR-10 имеет размер 32x32 пикселя, что означает, что оно состоит из 32 строк и 32 столбцов пикселей. Таким образом, каждое изображение содержит в сумме 1024 пикселя.

Предварительная обработка данных необходима для подготовки входных данных перед подачей их на вход нейронной сети. Это может включать в себя такие шаги, как нормализация значений, изменение размера изображений или преобразование данных в подходящий формат. Предварительная обработка помогает улучшить производительность и эффективность обучения нейронной сети.

Визуальное представление данных



2. Построение модели нейросети

Построенная модель нейросети представляет собой последовательную модель с несколькими слоями. Она начинается с сверточного слоя (Conv2D) с 32 фильтрами размером 3x3 и функцией активации ReLU, затем использует слой объединения (MaxPooling2D) для уменьшения размерности. Последующие слои повторяют этот процесс с увеличением числа фильтров до 64 и 128 соответственно. После нескольких слоев свертки и объединения следует слой выравнивания (Flatten), который преобразует выходные данные в одномерный вектор. Затем идут полносвязные слои (Dense) с функцией активации ReLU и

dropout для уменьшения переобучения. Наконец, последний полносвязный слой с функцией активации softmax используется для классификации на 10 классов.

Был использован метод `compile()`, где были указаны оптимизатор, функция потерь и метрики для оценки производительности модели во время обучения. Оптимизатор определяет алгоритм обновления весов модели на основе градиентов, функция потерь определяет, как модель оценивает свою производительность во время обучения, а метрики используются для оценки производительности модели после каждой эпохи обучения. Использован стохастический градиентный спуск (SGD) в качестве оптимизатора и категориальная кросс-энтропия (`sparse_categorical_crossentropy`) в качестве функции потерь для задачи классификации на 10 классов. Метрика ассурасу используется для оценки точности классификации.

3. Обучение модели

Процесс обучения нейросети состоит из эпох. Обычно количество эпох зависит от сложности задачи, размера обучающего набора данных и других факторов. В программе было выбрано 80 эпох обучения, потому что модель требует много времени, чтобы достичь высокой точности на выбранном наборе данных.

Вот последние 4 эпохи:

```
Epoch 1/4  
1563/1563 [=====] - 62s 40ms/step - loss: 0.2754 - accuracy: 0.8998  
Epoch 2/4  
1563/1563 [=====] - 62s 40ms/step - loss: 0.2684 - accuracy: 0.9016  
Epoch 3/4  
1563/1563 [=====] - 64s 41ms/step - loss: 0.2639 - accuracy: 0.9030  
Epoch 4/4  
1563/1563 [=====] - 61s 39ms/step - loss: 0.2551 - accuracy: 0.9061  
<keras.src.callbacks.History at 0x7f5c886de8f0>
```

Эти результаты показывают, что с каждой эпохой нейросеть становится все более точной в своих прогнозах, что является хорошим

знаком. В результате обучения модели была достигнута точность 91%. Однако при дальнейшей проверке фактическая точность оказалась на порядок ниже.

```
test_loss, test_accuracy = model.evaluate(x_test, y_test)
print('Test accuracy: ', test_accuracy)

313/313 [=====] - 5s 15ms/step - loss: 1.1001 - accuracy: 0.7200
Test accuracy: 0.7200000286102295
```

4. Прогнозы модели

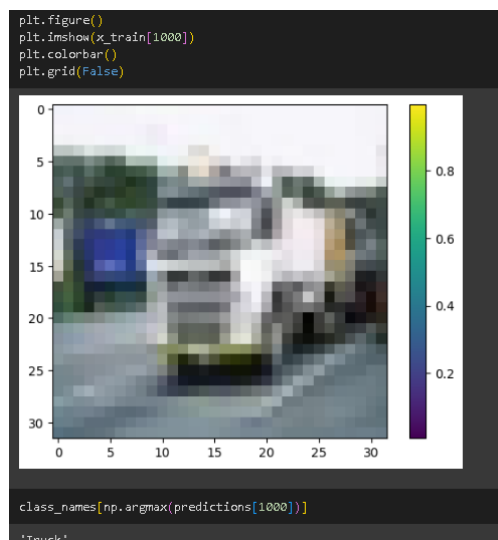
```
[ ] predictions = model.predict(x_train)
↳ 1563/1563 [=====] - 19s 12ms/step

[ ] predictions[1000]
↳ array([[5.2899191e-10, 3.3511606e-06, 8.2036345e-17, 2.4415426e-11,
          5.1152861e-18, 3.7855023e-12, 5.4790221e-14, 3.4279342e-13,
          9.5107391e-02, 9.0488911e-01], dtype=float32)

[ ] np.argmax(predictions[1000])
↳ 9

[ ] y_train[1000]
↳ array([9], dtype=uint8)
```

Очевидно, нейросеть верно определила класс объекта. Ниже приведены более наглядные прогнозы.



Текст программы

Код программы написан с использованием инструмента Jupyter

Notebook для визуализации данных.

```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow import keras
%matplotlib inline
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,
Dropout
from tensorflow.keras import utils
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
class_names = ['Airplane', 'Automobile', 'Bird', 'Cat', 'Deer', 'Dog',
'Frog', 'Horse', 'Ship', 'Truck']
plt.figure()
plt.imshow(x_train[49999])
plt.colorbar()
plt.grid(False)
x_train = x_train / 255
x_test = x_test / 255
plt.figure()
plt.imshow(x_train[49999])
plt.colorbar()
plt.grid(False)
plt.figure(figsize=(10,10))
for i in range (25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.imshow(x_train[i])
    plt.xlabel(class_names[int(y_train[i])])
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(256, activation='relu'),
    Dropout(0.5),
    Dense(10, activation='softmax')
])
model.compile(optimizer=tf.keras.optimizers.SGD(),
loss='sparse_categorical_crossentropy', metrics=['accuracy'])
model.summary()
```

```
model.fit(x_train, y_train, epochs=4)
test_loss, test_accuracy = model.evaluate(x_test, y_test)
print('Test accuracy: ', test_accuracy)
predictions = model.predict(x_train)
predictions[1000]
np.argmax(predictions[1000])
y_train[1000]
plt.figure()
plt.imshow(x_train[1000])
plt.colorbar()
plt.grid(False)
class_names[np.argmax(predictions[1000])]
```

2.10. Определение суммы монет на фото с использованием нейросети YOLOv8

Разрабатывается приложение для автоматического определения номинала и подсчета суммы монет по фотографии.

О нейросети YOLO

Для реализации проекта была выбрана нейронная сеть YOLO (You Only Look Once), специализирующаяся на объектном обнаружении в реальном времени. Выбор пал на YOLO из-за ее высокой скорости и точности обнаружения объектов. В частности, использовалась модель YOLOv8m, которая является одной из последних версий, обладающей улучшенной точностью и скоростью.

Модель была дообучена на собственном наборе данных, размеченных в онлайн сервисе Roboflow, что позволило адаптировать ее под специфику задачи по распознаванию монет

Алгоритм работы

Алгоритм состоит из двух частей:

1. Дообучение нейросети
2. Выполнение обученной модели с подсчётом монет

Дообучение нейросети

В качестве базовой модели использовалась YOLOv8m как компромисс между скоростью обучения и качеством результата. Для дообучения использовались по 125 разметок монет на 15 фотографиях

Model	size (pixels)	mAP ^{val} 50-95	Speed CPU ONNX (ms)	Speed A100 TensorRT (ms)	params (M)	FLOPs (B)
YOLOv8n	640	37.3	80.4	0.99	3.2	8.7
YOLOv8s	640	44.9	128.4	1.20	11.2	28.6
YOLOv8m	640	50.2	234.7	1.83	25.9	78.9
YOLOv8l	640	52.9	375.2	2.39	43.7	165.2
YOLOv8x	640	53.9	479.1	3.53	68.2	257.8

Подсчёт монет

После дообучения полученная модель применяется к картинкам в указанной папке, на них обводятся монетки и подписывается их сумма



Используемые функции

Название функции	Описание
YOLO('yolov8m.pt') .train(data='./data.yaml', epochs=300)	Запускает процесс обучения модели на основе yolov8m, со ссылкой на описание датасета в data.yaml на 300 эпох
YOLO('best.pt') .predict(FOLDER, conf=0.7, imgsz=SIZE)	Запускает процесс распознавания картинок в указанной папке. Распознавание считается успешным если уверенность в распознавание выше 70% (conf=0.7), картинки меняют размер до SIZE, лучше использовать 640 как разрешение исходной модели
ImageDraw.Draw(im).text(x, y, "str")	Добавить текст на картинку в координаты X, Y

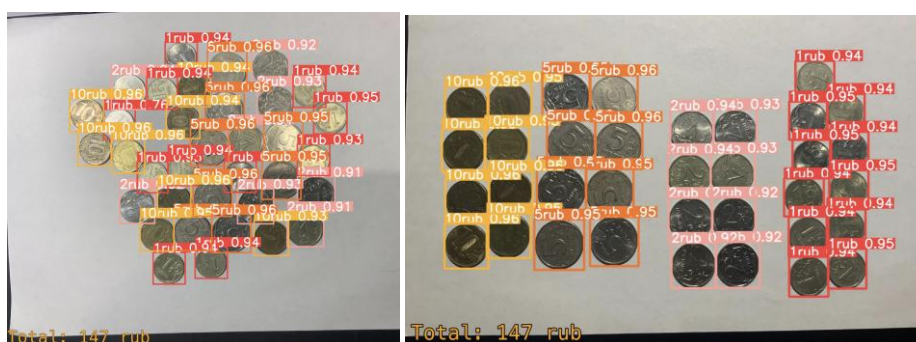
Результат

Для проверки работы модели были сделаны новые фотографии которые не участвовали в процессе обучения.

Пример фотографии:



Результат выполнения нейросети:



Все монеты были распознаны правильно, сумма составила 147 рублей.

Текст программы

```
from PIL import Image, ImageDraw, ImageFont
from ultralytics import YOLO
MODEL = './models/best_1024.pt'
FOLDER = './images/source'
SIZE = 1024
def train():
    model = YOLO('yolov8m.pt')
    model.train(data='./data.yaml', imgs=SIZE, epochs=300)
def predict():
    model = YOLO(MODEL)
    results = model.predict(FOLDER, conf=0.5, imgs=SIZE)
    for r in results:
        im_array = r.plot() # plot a BGR numpy array of predictions
        im = Image.fromarray(im_array[...,:-1]) # RGB PIL image
        im.show() # show image
        im.save(f'./images/results/{r.path.split("/")[-1].split(".")[0]}.jpg') # save image
def calculate():
    model = YOLO(MODEL)
```

```

results = model.predict(FOLDER, conf=0.5, imgsz=1024,
show_labels=False, show_boxes=False)
totalSum = 0

for r in results:
    im_array = r.plot()
    im = Image.fromarray(im_array[... , :-1])
    image_name = r.path.split("/")[-1].split(".")[0]
    class_counts = {class_name: 0 for class_name in r.names.values()}
    detected = r.bboxes.cls
    for i in range(len(detected)):
        item = detected[i].item()
        name = r.names[item]
        class_counts[name] += 1
    print(class_counts)
    screenSum = class_counts['1rub'] + class_counts['2rub'] * 2 +
class_counts['5rub'] * 5 + class_counts['10rub'] * 10
    totalSum += screenSum
    print(f'Total on {image_name} = {screenSum} rub\n')
    draw = ImageDraw.Draw(im)
    bbox = im.getbbox()
    height = bbox[3].real - bbox[1].real if bbox else 50
    font = ImageFont.truetype('./Menlo.ttc', size=int(height / 20))
    draw.text((20, height - height / 20 - 20), f'Total: {screenSum} rub',
font=font, fill='orange', stroke_width=2, stroke_fill='black')
    im.save(f'./images/results/{image_name}.jpg')
    print(f'\n=====
\nTotal sum = {totalSum}
rub\n=====')

print('Train model (T), Predict (P) or Calculate (C)? T/P/C')
variant = input()
if variant.upper() == 'T':
    train()
elif variant.upper() == 'P':
    predict()
elif variant.upper() == 'C':
    calculate()

```

2.11. Определение элементов на изображениях. Faster R-CNN

Faster R-CNN (Faster Region-based Convolutional Neural Network) - это алгоритм обнаружения объектов в изображениях, который является эволюцией и улучшением оригинального R-CNN и Fast R-CNN [39].

Основная идея Faster R-CNN заключается в том, что он объединяет обнаружение объектов и генерацию регионов (Region Proposal Network,

RPN) в единую модель. Это позволяет ускорить процесс обнаружения объектов, так как сеть может делиться вычислениями между обнаружением объектов и генерацией регионов.

Как работает Faster R-CNN:

1. Сверточная сеть (Backbone): Faster R-CNN использует сверточную нейронную сеть (например, ResNet) для извлечения признаков из изображения.

2. Region Proposal Network (RPN): RPN генерирует кандидаты регионов, которые могут содержать объекты. Он использует сверточные слои для предсказания ограничивающих рамок (bounding boxes) и их вероятностей.

3. Объединение с обнаружением объектов: После генерации регионов RPN передает их в обнаружение объектов, где используется классификация и точная настройка ограничивающих рамок для определения объектов.

Алгоритм работы

1. Импорт необходимых библиотек.
2. Указание пути до изображения.
3. Обработка изображения.
4. Вывод координат полученных объект.
5. Показ изображения с выделенными элементами

Использованные функции

Название функции	Описание
<code>torchvision.models.detection.fasterrcnn_resnet50_fpn(pretrained=True)</code> :	Загружает предварительно обученную модель Faster R-CNN на основе ResNet-50 с Feature Pyramid Network (FPN).
<code>model.eval()</code>	Установка модели в режим оценки (не используется для обучения)
<code>output = model(image_tensor)</code>	Подает тензор изображения в модель

Faster R-CNN для обнаружения объектов.
Модель возвращает словарь с ключами.

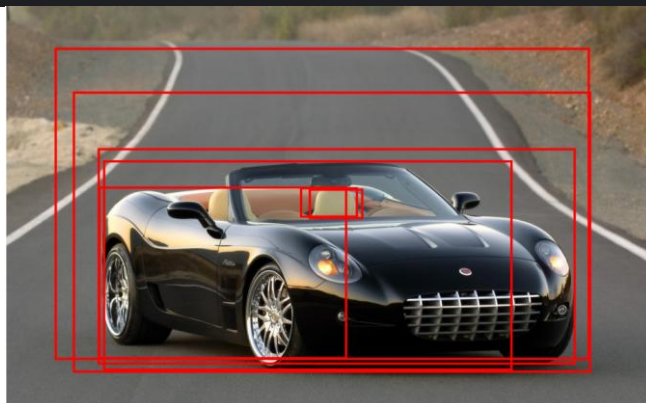
Результат

До:



После:

```
Обнаруженные объекты:  
Координаты: tensor([ 270.8052,  421.3595, 1674.4764, 1052.4966])  
Координаты: tensor([ 892.3595,  539.9840, 1048.0720,  624.6990])  
Координаты: tensor([ 273.2996,  535.5510, 1000.7917, 1036.0586])  
Координаты: tensor([ 867.9022,  537.9333, 1031.5813,  620.5322])  
Координаты: tensor([ 285.9579,  457.2056, 1488.1221, 1071.1147])  
Координаты: tensor([ 144.2005,  127.1491, 1715.9192, 1038.9199])  
Координаты: tensor([ 197.8343,  255.1567, 1718.9346, 1075.6388])
```



Текст программы

```
import torch  
import torchvision  
from torchvision import transforms  
from PIL import Image  
import matplotlib.pyplot as plt  
  
# Загрузка предварительно обученной модели Faster R-CNN  
model =  
torchvision.models.detection.fasterrcnn_resnet50_fpn(pretrained=True)  
model.eval()
```

```

# Загрузка и предобработка изображения
image_path = input("Введите путь к изображению: ")
image = Image.open(image_path)
transform = transforms.Compose([
    transforms.ToTensor(),
])
image_tensor = transform(image).unsqueeze(0)

# Обнаружение объектов на изображении
with torch.no_grad():
    output = model(image_tensor)

# Вывод результата
print("Обнаруженные объекты:")
for box in output[0]['boxes']:
    print(f"Координаты: {box}")

# Визуализация результата
plt.figure(figsize=(10, 10))
plt.imshow(image)
ax = plt.gca()
for box in output[0]['boxes']:
    xmin, ymin, xmax, ymax = box.tolist()
    rect = plt.Rectangle((xmin, ymin), xmax - xmin, ymax - ymin,
fill=False, edgecolor='red', linewidth=2)
    ax.add_patch(rect)
plt.axis('off')
plt.show()

```

2.12. Определение рукописных цифр на изображении. TensorFlow MNIST

Разработана нейронная сеть для определения цифр на изображении. Используется набор данных MNIST. Набор данных MNIST является широко известным набором данных в области компьютерного зрения. Он содержит тысячи изображений рукописных цифр от 0 до 9. Каждое изображение представлено в виде матрицы пикселей размером 28x28.

Модель

В данной работе используется библиотека TensorFlow. Импортируем модуль mnist из keras.datasets. Этот модуль содержит функцию mnist.load_data(), которая позволяет загрузить набор данных MNIST.

Далее импортируем класс `Sequential` из модуля `models` в `keras`. `Sequential` позволяет создавать последовательные модели нейронных сетей. Импортируем классы `Dense` и `Flatten` из модуля `layers` в `keras`. `Dense` представляет слои полносвязной нейронной сети, а `Flatten` преобразует многомерный массив в одномерный. Импортируем функцию `to_categorical` из модуля `utils` в `keras`. Эта функция преобразует метки классов в формат `one-hot encoding`.

Далее будет использоваться набор готовых данных MNIST. Это объёмная база данных образцов рукописного написания цифр. База данных является стандартом, предложенным Национальным институтом стандартов и технологий США с целью калибровки и сопоставления методов распознавания изображений с помощью машинного обучения в первую очередь на основе нейронных сетей. Будем использовать набор, содержащий рукописные изображения цифр.

Загружаем набор данных MNIST с помощью функции `mnist.load_data()`. Эта функция возвращает два кортежа: первый содержит тренировочные изображения и соответствующие им метки, а второй - тестовые изображения и метки.

Необходимо также нормализовать значения пикселей тренировочных изображений, разделив их на `255.0`. Это приводит к значениям пикселей в диапазоне от `0` до `1`. Нормализуем и значения пикселей тестовых изображений аналогичным образом.

Преобразуем тренировочные метки в формат `one-hot encoding` с помощью функции `to_categorical()`. Это преобразование позволяет представить каждую метку как вектор, где значение `1` находится в индексе, соответствующем классу, и `0` в остальных индексах.

Преобразуем тестовые метки в формат one-hot encoding аналогичным образом.

Далее перейдем к самой модели. Создаем объект `Sequential`, который представляет модель нейронной сети. В данном случае мы определяем структуру модели, добавляя слои к ней по порядку. Входной слой - `Flatten`, который преобразует двумерный массив пикселей в одномерный вектор. Затем добавляем два полносвязных слоя `Dense` с активацией `ReLU` и разным количеством нейронов. Наконец, добавляем выходной слой `Dense` с функцией активации `softmax`, который дает вероятности принадлежности каждого класса. Компилируем модель, задавая оптимизатор `'adam'`, функцию потерь `'categorical_crossentropy'` и список метрик, включая `'accuracy'`, которая будет использоваться для оценки производительности модели.

Обучаем модель на тренировочных данных с помощью метода `fit()`. Передаем тренировочные изображения и метки, указываем количество эпох обучения (15) и используем валидационные данные (`test_images` и `test_labels`) для проверки производительности модели на каждой эпохе обучения. В результате тестов обучения модели, было выявлено, что оптимальное количество эпох 15. В случае, когда эпох больше, точность модели перестает расти, что говорит о том, что модель обучена. Если задать меньшее количество эпох, то модель будет недостаточно обучена.

Результаты обучения модели:

```
Epoch 1/15
1875/1875 ----- 2s 804us/step - accuracy: 0.8754 - loss: 0.4117 - val_accuracy: 0.9600 - val_loss: 0.1303
Epoch 2/15
1875/1875 ----- 1s 745us/step - accuracy: 0.9707 - loss: 0.1001 - val_accuracy: 0.9746 - val_loss: 0.0805
Epoch 3/15
1875/1875 ----- 1s 743us/step - accuracy: 0.9803 - loss: 0.0667 - val_accuracy: 0.9768 - val_loss: 0.0755
Epoch 4/15
1875/1875 ----- 1s 751us/step - accuracy: 0.9847 - loss: 0.0476 - val_accuracy: 0.9769 - val_loss: 0.0736
Epoch 5/15
1875/1875 ----- 1s 748us/step - accuracy: 0.9881 - loss: 0.0378 - val_accuracy: 0.9747 - val_loss: 0.0883
Epoch 6/15
1875/1875 ----- 1s 752us/step - accuracy: 0.9895 - loss: 0.0310 - val_accuracy: 0.9788 - val_loss: 0.0707
Epoch 7/15
1875/1875 ----- 1s 755us/step - accuracy: 0.9924 - loss: 0.0240 - val_accuracy: 0.9762 - val_loss: 0.0877
Epoch 8/15
1875/1875 ----- 1s 755us/step - accuracy: 0.9927 - loss: 0.0211 - val_accuracy: 0.9756 - val_loss: 0.0929
Epoch 9/15
1875/1875 ----- 1s 752us/step - accuracy: 0.9943 - loss: 0.0187 - val_accuracy: 0.9786 - val_loss: 0.0923
Epoch 10/15
1875/1875 ----- 1s 747us/step - accuracy: 0.9941 - loss: 0.0172 - val_accuracy: 0.9781 - val_loss: 0.0876
Epoch 11/15
1875/1875 ----- 1s 749us/step - accuracy: 0.9953 - loss: 0.0138 - val_accuracy: 0.9786 - val_loss: 0.0940
Epoch 12/15
1875/1875 ----- 1s 750us/step - accuracy: 0.9961 - loss: 0.0118 - val_accuracy: 0.9790 - val_loss: 0.0979
Epoch 13/15
1875/1875 ----- 1s 749us/step - accuracy: 0.9956 - loss: 0.0135 - val_accuracy: 0.9792 - val_loss: 0.0966
Epoch 14/15
1875/1875 ----- 1s 747us/step - accuracy: 0.9969 - loss: 0.0095 - val_accuracy: 0.9790 - val_loss: 0.1047
Epoch 15/15
1875/1875 ----- 1s 750us/step - accuracy: 0.9968 - loss: 0.0101 - val_accuracy: 0.9765 - val_loss: 0.1200
```

Сохраняем обученную модель в файл с именем 'mnist_model.keras'. Это позволяет сохранить параметры модели, включая веса нейронных сетей и конфигурацию модели.

Интерфейс

Необходимо создать интерфейс, который позволит пользоваться нейросетью. Создадим приложение с полем для рисования чисел, которое будет считывать нарисованную картинку и определять цифру на ней.

Код начинается с импорта необходимых библиотек: Flask для создания веб-приложения, render_template для отображения HTML-шаблонов, request для обработки запросов клиента, jsonify для преобразования результата предсказания в формат JSON, numpy для численных вычислений, tensorflow для загрузки и использования обученной модели, а также PIL для обработки изображений.

Загружается обученная модель с помощью функции `tf.keras.models.load_model()`. Она загружает заранее обученную модель, сохраненную в файле `mnist_model.keras`. Используется декоратор маршрута (route decorator) для указания URL-конечной точки ("/") и связанной с ней функции представления (`index()`). Функция представления

возвращает отрисованный шаблон `index.html`. Используется декоратор маршрута для указания URL-конечной точки (`"/predict"`) и связанной с ней функции представления (`predict()`). Эта функция маршрута вызывается при получении POST-запроса на конечную точку `"/predict"`. Полученное изображение загружается из запроса с помощью `request.files['image'].read()`. Загруженное изображение считывается как массив байтов. Затем массив байтов преобразуется в объект изображения PIL (Python Imaging Library) с помощью `Image.open(io.BytesIO(img_file))`. Изображение изменяется до размера 28x28 пикселей с помощью `.resize((28, 28))`.

Изображение преобразуется в массив `numpy` с помощью `np.array(img)` и меняет форму так, чтобы иметь размерность (1, 28, 28, 1), чтобы соответствовать форме входных данных модели. Кроме того, значения пикселей нормализуются, чтобы быть в диапазоне от 0 до 1, путем деления на 255.0. Модель предсказывает цифру на изображении с помощью `model.predict(img_array)`. Предсказания - это массив вероятностей для каждой цифры. Предсказываемая цифра определяется по индексу с максимальной вероятностью с использованием `np.argmax(predictions)`. Предсказанная цифра возвращается как ответ в формате JSON с помощью `jsonify({'digit': int(digit)})`.

Тестирование модели



Текст программы

```
Train.py
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.utils import to_categorical

(train_images, train_labels), (test_images, test_labels) =
mnist.load_data()
train_images = train_images / 255.0
test_images = test_images / 255.0
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)

model = Sequential([
    Flatten(input_shape=(28, 28)),
    Dense(128, activation='relu'),
    Dense(64, activation='relu'),
    Dense(10, activation='softmax')
])

model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

model.fit(train_images, # данные для обучения
          train_labels, #
          epochs=15,    # кол-во эпох обучения
          validation_data=(test_images, test_labels)) # данные для
проверки
model.save('mnist_model.keras')
Webui.py
from flask import Flask, render_template, request, jsonify
import numpy as np
import tensorflow as tf
from PIL import Image
import io
app = Flask(__name__)

# Загрузка модели
model = tf.keras.models.load_model('mnist_model.keras')

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/predict', methods=['POST'])
```

```

def predict():
    if request.method == 'POST':
        # Получаем изображение от пользователя
        img_file = request.files['image'].read()
        img = Image.open(io.BytesIO(img_file)).convert('L')
        img = img.resize((28, 28))
        img_array = np.array(img).reshape((1, 28, 28, 1)) / 255.0

        # Делаем предсказание
        predictions = model.predict(img_array)
        digit = np.argmax(predictions)

        return jsonify({'digit': int(digit)})

if __name__ == '__main__':
    app.run(debug=True)

```

2.13. Определение пола и возраста человека по фото.

Gender_deploy.prototxt

Данная работа реализована на языке Python и использует библиотеку OpenCV для обнаружения лиц на изображении, определения их возраста и пола с использованием предварительно обученных моделей глубокого обучения.

1. Для определения возраста и пола использовались две разные модели, обученные Талом Хасснером и Гилом Леви:

- Модель определения возраста: `age_deploy.prototxt` (описание слоев нейронной сети), `age_net.caffemodel` (файл, в котором хранятся параметры нейронной сети, которые определяют, какие конкретные связи должны быть установлены в процессе обучения)
- Модель определения пола: `gender_deploy.prototxt` (описание архитектуры нейронной сети для определения пола на изображении), `gender_net.caffemodel` (файл, в котором хранятся параметры нейронной сети, которые определяют, какие конкретные связи должны быть установлены в процессе обучения). Модель использует архитектуру сети GoogleNet и была предобучена на ImageNet dataset.

Данные модели используют набор данных Adience, который был создан для помощи в изучении распознавания возраста и пола. Он содержит 26 580 фотографий с 8 возрастными группами, сделанных 2284 испытуемыми. Набор данных содержит разнообразную коллекцию лиц, различающихся по позам, внешнему виду, шуму, освещению и многому другому.

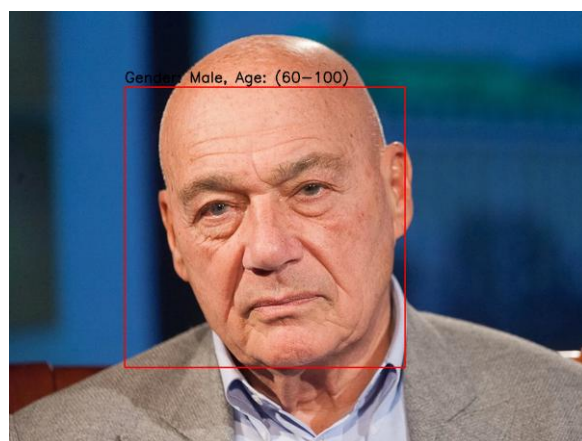
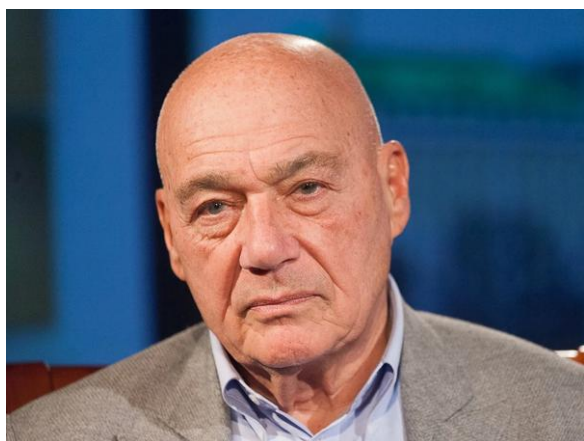
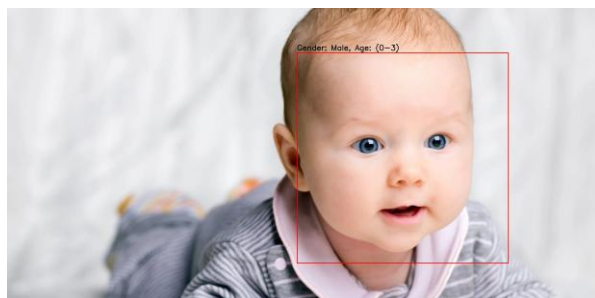
2. Для обнаружения лиц используется классификатор каскадов Хаара. Каскад Хаара извлекает объекты из изображения с помощью какого-то фильтра. Затем для каждой области получается одно значение путем вычитания суммы пикселей. Если значение функции большое, то это означает, что она актуальная. При использовании данной модели изображение преобразуется в оттенки серого, поскольку классификатор каскадов Хаара работает только с одноканальными изображениями.

Использованные функции

Название	Краткое описание
cv2.imread()	Чтение изображения. Возвращает массив, который содержит значения пикселей изображения в формате BGR
cv2.cvtColor()	Преобразование изображения из одного цветового пространства в другое
cv2.dnn.blobFromImage()	Предварительная обработка исходного изображений в формат, понятный для нейронной сети
age_model.setInput()	Установка входных данных для модели определения возраста
age_model.forward()	данные изображения лица, проходят через все слои нейронной сети. На выходе получаем массив чисел, который представляет собой вероятность принадлежности к определенному возрасту
age_preds.argmax()	Находит индекс элемента с наибольшим значением в массиве. Данное значение показывает, какой возраст больше всего соответствует человеку на изображении

cv2.rectangle()	Рисует прямоугольник вокруг обнаруженного лица на изображении
cv2.putText()	Добавление текст на изображение
cv2.imwrite()	Сохранение изображения по указанному пути

Пример работы программы



Текст программы

```
import cv2
age_model = cv2.dnn.readNetFromCaffe('age_deploy.prototxt',
'age_net.caffemodel')
gender_model = cv2.dnn.readNetFromCaffe('gender_deploy.prototxt',
'gender_net.caffemodel')

# Функция для определения возраста и пола на изображении
def detect_age_gender(image_path, output_path):
    image = cv2.imread(image_path)
    face_cascade =
cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray, 1.3, 3) #возвращает
список прямоугольников

    if len(faces) == 0:
        print("No faces detected")
        return
```

```

# Обработка каждого обнаруженного лица
for (x, y, w, h) in faces:

    face = image[y:y + h, x:x + w]
    # Преобразование изображения для нейронной сети
    blob = cv2.dnn.blobFromImage(face, 1.0, (227, 227),
(78.4263377603, 87.7689143744, 114.895847746))

    # Определение возраста
    age_model.setInput(blob)
    age_prediction = age_model.forward()
    age_i = age_prediction[0].argmax() #индекс соответствующей
возрастной категории
    age = ["(0-3)", "(4-8)", "(9-15)", "(18-25)", "(25-30)", "(45-
50)", "(50-70)", "(60-100)"][age_i]
    age_confidence = age_prediction[0][age_i]

    # Определение пола
    gender_model.setInput(blob)
    gender_prediction = gender_model.forward()
    gender_i = gender_prediction[0].argmax()
    gender = ['Male', 'Female'][gender_i]
    gender_confidence = gender_prediction[0][gender_i]

    # Результаты
    cv2.rectangle(image, (x, y), (x + w, y + h), (0, 0, 255), 2)
    text = f'Gender: {gender}, Age: {age}'
    cv2.putText(image, text, (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX,
1.5, (0, 0, 0), 3)
    cv2.putText(image, text, (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX,
1.5, (0, 0, 0), 3)

    cv2.imwrite(output_path, image)
    print(f"Result saved to {output_path}")

# Пример использования функции для обработки изображения
detect_age_gender('img_3.png', 'output.png')

```

2.14. 3D-сцена из нескольких фото Luma AI

Luma AI представляет собой сервис, способный превратить обычные снимки и видео, сделанные на смартфоне, в уникальные 3D-изображения. Это приложение расширяет возможности творчества и визуального искусства, обогащая каждый кадр глубиной и объемом. Сервис предлагает

разнообразные инструменты и эффекты, которые помогут улучшить качество и стиль создаваемых произведений.

Функционал постоянно обновляется и дорабатывается, тем не менее, на данный момент пользователи могут выполнить следующие действия:

- Создать игровых персонажей для видеоигр
- Сгенерировать трехмерную модель предмета по текстовому описанию
- Выбрать готовый 3D-вариант из большой галереи избранных изображений
- Смоделировать короткий ролик в разных ракурсах из фотографии

Luma AI предоставляет возможность делиться созданными 3D-изображениями с другими пользователями. Вы можете загружать свои работы в облачное хранилище и отправлять ссылки на них своим друзьям и коллегам.

Алгоритм работы программы

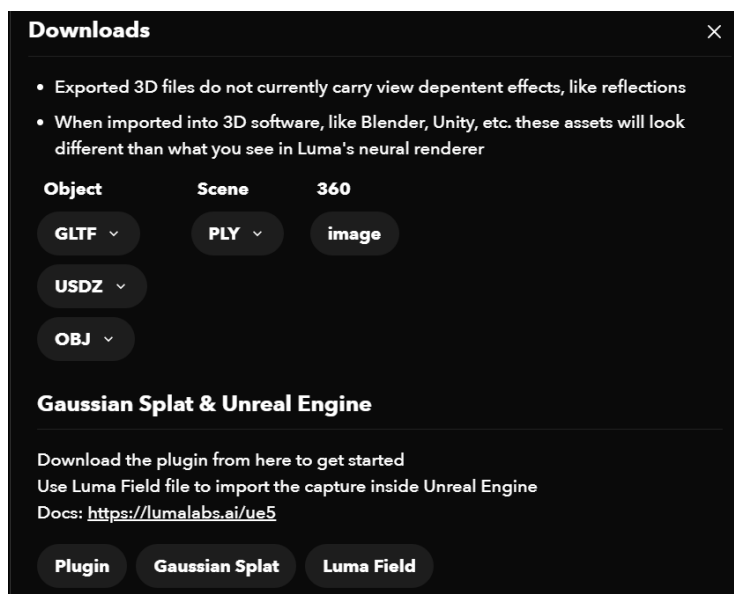
Для получения 3D модели в Luma AI необходимо загрузить несколько фотографий. Затем дождаться относительно долгой загрузки (например 10 часов) 3D модели. Модель можно посмотреть только в самом сервисе (на браузерной страничке или в мобильном приложении), так как это трёхмерный объект, просто так получить видео в формате mp4 не получится. Более подробный алгоритм работы с сервисом следующий:

- Скачиваем репозиторий Luma AI для взаимодействия с данным сервисом с помощью Python (lumaapi).
- Авторизуемся в python-коде к нашему аккаунту в Luma AI, передав в качестве параметра api-key в конструктор объекта LumaClient.
- Отправляем zip архив с заранее сделанными фотографиями объекта с разных ракурсов на сервер. Для этого передаем в метод submit

объекта LumaClient путь к zip файлу. Так же указываем название 3D модели. В нашем случае, генерировалась модель для табуретки.

- Проверяем статус обработки. Сначала статус Queued, затем Processing, после Completed.
- Переходим в сервис Luma AI с помощью браузера или мобильного приложения. Авторизуемся в свой аккаунт и получаем 3D модель объекта. 3D модель можно просматривать только в сервисе Luma AI.

К сожалению, API Luma AI не предоставляет возможность скачать 3D модель с помощью кода. Но модель можно скачать, перейдя на сайт Luma AI в различных форматах. Но можно скачать файл с помощью response в Python. Для этого нужно передать прямую ссылку на файл из сервиса Luma AI. Получим zip архив, содержащий текстуры и 3D представление (см код ниже).

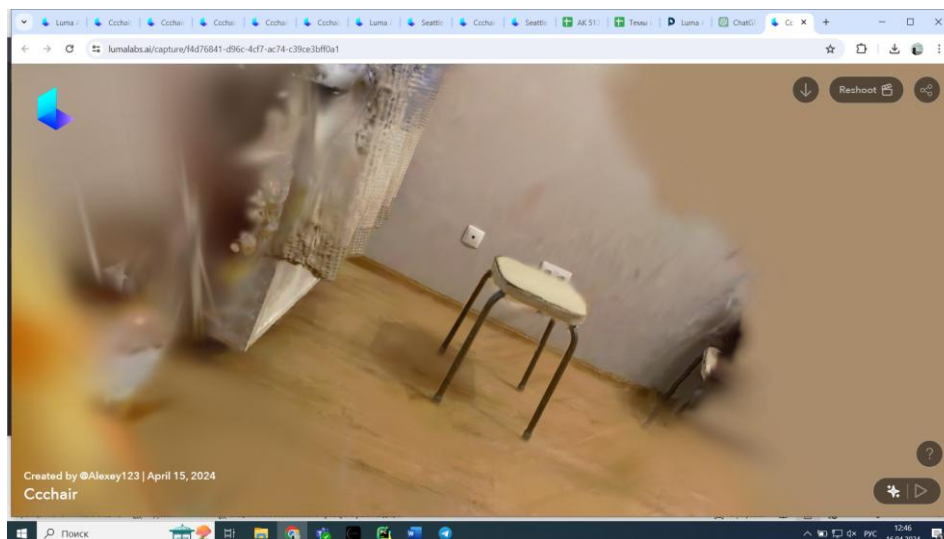


Использованные функции

Название функции	Назначение
LumaClient(<api-key>)	Объект типа LumaClient, использующий api-key для авторизации в сервисе Luma AI
.slug(<path>, <name>)	Метод объекта типа LumaClient, позволяющий загрузить файл (содержащий фото/видео объекта) в сервис Luma AI, который на основе объектов файла (фото и видео) сгенерирует 3D модель объекта. Так же передаем вторым параметром название нашей

	МОДЕЛИ.
.status(<slug>)	Метод объекта типа LumaClient, позволяющий получить статус обработки файла
.get(<url>)	Get запрос на указанный url (скачивание файла)

Скриншоты



Текст программы

```

from lumaapi import LumaClient

client = LumaClient("e3713b5e-14d2-42cd-8719-01eab392e3ab-3be0177-9d37-4a8d-96d4-9714b532870c")

slug = client.submit(r"C:\Users\Gorohov\Desktop\Photots.zip", "Ccchair")
print(client.status(slug))

import requests

url = 'https://cdn-luma.com/a033a969fdc572c667a979aeb8b257f85d9cbc9167938e4cd193e4d03758125/hair_textured_mesh_obj.zip'
response = requests.get(url)

with open('hair_textured_mesh_obj.zip', 'wb') as file:
    file.write(response.content)

```

2.15. Повышение разрешения с использованием модели EDSR

Сверхвысокое разрешение подразумевает преобразование изображения из более низкого разрешения в изображения с более высоким разрешением. Процесс получения сверхвысокого разрешения выполняется

следующим образом: берется изображение с низким разрешением, изображение масштабируется, разрешение увеличивается и выдается в качестве выходных данных.

Традиционные модели сверхвысокого разрешения, в которых не используется глубокое обучение, лишены мелких деталей. Им не удается устранить различные дефекты и факты сжатия в системах. Все эти проблемы можно очень эффективно решить, используя модель глубокого обучения, которая позволяет сохранить все детали нетронутыми.

Интерполяция означает искажение пикселей из одной сетки в другую, что в основном поможет нам изменить разрешение изображения. Существуют различные модели интерполяции, в частности в результатах продемонстрировано изображение, разрешение которого было поочередно увеличено только с помощью бикубической интерполяции, а затем с помощью двух моделей с глубоким обучением.



Bicubic (23.42/0.68)

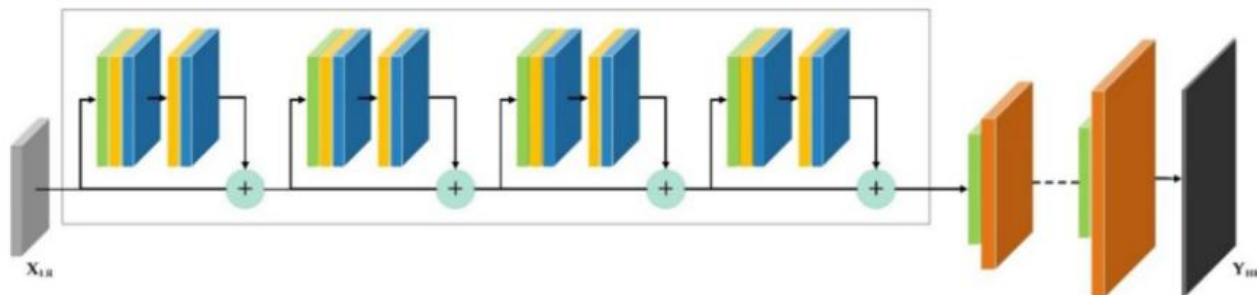


EDSR (28.39/0.87)

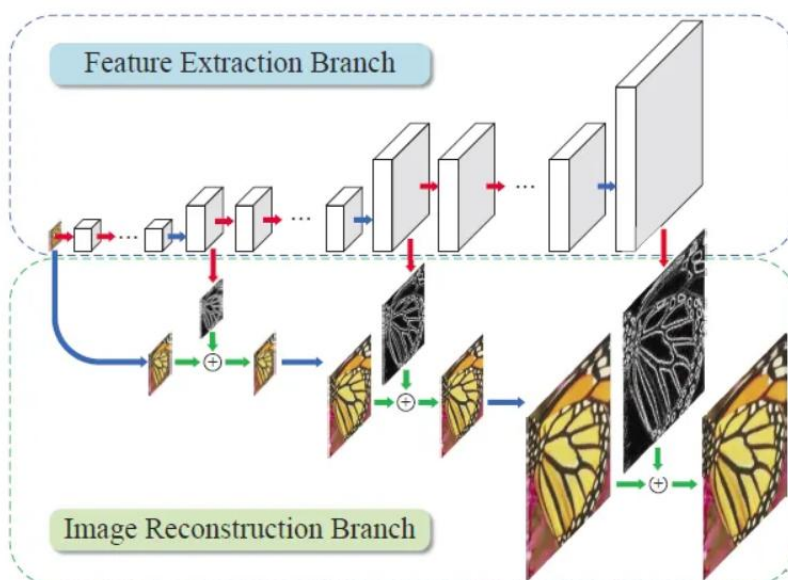
Для демонстрации использовались модели EDSR и LapSRN.

Архитектура EDSR (Enhanced Deep Super Resolution) основана на архитектуре SRResNet и состоит из нескольких остаточных блоков. Он использует слои постоянного масштабирования вместо слоев пакетной нормализации для получения согласованных результатов (входные и выходные данные имеют схожие распределения, поэтому нормализация промежуточных функций может быть нежелательной). Вместо использования потерь L2 (среднеквадратическая ошибка) авторы

использовали потери L1 (средняя абсолютная ошибка), которые эмпирически работают лучше.



Конвейер архитектуры сверхвысокого разрешения EDSR с четырьмя остаточными блоками: зеленый цвет представляет слои свертки. Желтый цвет представляет слой нормализации, синий цвет представляет собой слой активации ReLU, а коричневый цвет представляет собой слой переупорядочения пикселей. Входные данные конвейера показаны серым цветом и помечены как XLR (изображение с низким разрешением), а выходные данные, представленные черным цветом, представляют собой изображения высокого разрешения YHR.



В отличие от одношаговой повышающей дискретизации, LapSRN постепенно восстанавливает остатки поддиапазонов изображений с высоким разрешением на нескольких уровнях пирамиды, особенно

на уровнях $\log_2(S)$, где S — масштабный коэффициент (т. е. 2, 4, 8). Без использования бикубического метода LapSRN напрямую извлекает признаки из входного изображения с низким разрешением и, следовательно, снижает вычислительную нагрузку.

Использованные функции

Функция	Описание
<code>requests.get</code>	Выполнение get запроса
<code>os.path.exists</code>	Проверка существования пути
<code>sr = dnn_superres.DnnSuperResImpl_create</code>	Создание инстанса класса для масштабирования изображений с помощью сверточных нейронных сетей
<code>cv2.imread</code>	Считывание исходного изображения
<code>sr.readModel</code>	Считывание пред обученной модели
<code>sr.setModel</code>	Установка активной модели
<code>sr.upsample</code>	Повышение разрешения фото
<code>cv.imwrite</code>	Сохранение изображения
<code>cv2.resize</code>	Изменение размера

Описание алгоритма работы программы

1. Импортируются необходимые модули: `os`, `cv2`, `requests`, `dnn_superres`.
2. Определяется функция `download_models`, которая загружает модели с указанных URL-адресов и сохраняет их на диск по заданным именам файлов.
3. Определяется функция `get_models_dict`, которая проверяет наличие моделей на диске и, если они отсутствуют, вызывает функцию `download_models`. Затем функция создает словарь с именами моделей и их параметрами (имя файла и масштаб увеличения разрешения).
4. Определяется функция `super_resolution_with_model`, которая использует модель сверточной нейронной сети для увеличения разрешения изображения. Функция создает объект класс для

масштабирования изображений с помощью сверточных нейронных сетей, загружает изображение и модель из файла, устанавливает модель и масштаб увеличения разрешения, применяет модель к изображению, и записывает результат в файл по переданному пути.

5. Определяется функция ``bicubic_interpolate``, которая использует интерполяцию бикубического сплайна для увеличения разрешения изображения.
6. В блоке ``if __name__ == "__main__":`` программа запрашивает у пользователя имя файла для увеличения разрешения и проверяет его существование.
7. Программа вызывает функцию ``bicubic_interpolate`` для увеличения разрешения изображения с масштабом 4 и 8 и сохраняет результаты в файлы.
8. По заданным url определяются имена файлов для моделей.
9. Программа вызывает функцию ``get_models_dict`` для получения словаря с моделями и их параметрами.
10. Программа проходит по словарю моделей и для каждой модели вызывает функцию ``super_resolution_with_model`` для увеличения разрешения изображения и сохранения результатов в файлы.

Результаты

Начальное фото



Сведения о размере
480 x 270 33 КБ 96 точек на дюйм 24 бит

После обработки моделью Enhanced Deep Super Resolution x4



Сведения о размере
1920 x 1080 435.8 КБ 96 точек на дюйм 24 бит

Текст программы

```
import os
import cv2
import requests
from cv2 import dnn_superres

def download_models(url_list, filename_list):
    response_list = [requests.get(url) for url in url_list]
    for ind, model_file in enumerate(filename_list):
        with open(model_file, 'wb') as file:
            file.write(response_list[ind].content)

def get_models_dict(url_list, model_file_names):
    if not (os.path.exists('EDSR_x4.pb') or
os.path.exists('LapSRN_x8.pb')):
        download_models(url_list, model_file_names)
    return {model_file.split('_')[0].lower(): (model_file,
int(model_file[model_file.find('x') + 1])) for model_file in
        model_file_names}

def super_resolution_with_model(scale, model_name, model_path,
image_path, upscaled_image_path):
    sr = dnn_superres.DnnSuperResImpl_create()
    image = cv2.imread(image_path)
    sr.readModel(model_path)
    sr.setModel(model_name, scale)
    upscaled_image = sr.upsample(image)
    cv2.imwrite(upscaled_image_path, upscaled_image)
```

```

def bicubic_interpolate(scale, image_path, upscaled_image_path):
    image = cv2.imread(image_path)
    resolution_first = image.shape[1]
    resolution_second = image.shape[0]
    bicubic = cv2.resize(image, (resolution_first * scale,
resolution_second * scale), interpolation=cv2.INTER_CUBIC)
    cv2.imwrite(upscaled_image_path, bicubic)

if __name__ == "__main__":
    print("Введите имя файла(фото для увеличения разрешения): ")
    path_image_to_test = input()
    if not os.path.exists(path_image_to_test):
        raise RuntimeError(f"Ошибка, файла {path_image_to_test} не
существует")
    bicubic_interpolate(scale=4, image_path=path_image_to_test,
upscaled_image_path="cubic_interpolate_x4.jpg")
    bicubic_interpolate(scale=8, image_path=path_image_to_test,
upscaled_image_path="cubic_interpolate_x8.jpg")
    urls = [

'https://raw.githubusercontent.com/Saafke/EDSR_Tensorflow/master/models/E
DSR_x4.pb',
    'https://raw.githubusercontent.com/fannymonori/TF-
LapSRN/master/export/LapSRN_x8.pb'
    ]
    model_file_names = [x.split('/')[-1] for x in urls]
    for model_name, params in get_models_dict(url_list=urls,
model_file_names=model_file_names).items():
        super_resolution_with_model(scale=params[1],
model_name=model_name, model_path=params[0],
                                image_path=path_image_to_test,
                                upscaled_image_path=(model_name +
f'x{params[1]}.jpg'))

```

2.16. Распознавание объектов на изображениях. MobileNet

Нейронная сеть MobileNet - это тип глубокой нейронной сети, разработанный для решения задач компьютерного зрения, таких как распознавание объектов на изображениях. MobileNet имеет достаточно легкую архитектуру по сравнению с некоторыми другими глубокими нейронными сетями. Она состоит из специальных блоков, называемых сверточными слоями (convolutional layers), которые эффективно анализируют изображения. Сверточные слои обрабатывают изображение,

выделяя важные признаки, такие как грани, текстуры или формы объектов. Они работают, скользя по изображению с маленькими окнами (обычно 3x3 или 5x5 пикселей), применяя к каждой области фильтры, которые обнаруживают различные признаки.

MobileNet имеет несколько сверточных слоев, которые последовательно применяются к изображению. Это позволяет сети извлекать более абстрактные признаки с увеличением глубины. После сверточных слоев MobileNet часто использует глобальное среднее пулирование (global average pooling). Это операция, которая усредняет значения признаков по всему изображению, создавая одномерный вектор признаков. После глобального пулинга следуют полносвязные слои (fully connected layers), которые преобразуют вектор признаков в выходные классы. Эти слои соединяют каждый нейрон предыдущего слоя с каждым нейроном следующего слоя, что позволяет модели учиться выделять более сложные паттерны и взаимосвязи между признаками.

Обучение. Нейронная сеть MobileNet обучается на большом наборе изображений с известными метками классов. Во время обучения она корректирует веса своих слоев таким образом, чтобы минимизировать ошибку между предсказанными и истинными классами. В данном случае программа использует предобученные веса MobileNet, которые были обучены на наборе изображений из ImageNet. Это позволяет использовать модель без необходимости обучения с нуля.

Алгоритм работы

1. Импорт необходимых библиотек:

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.applications.mobilenet import MobileNet,
preprocess_input, decode_predictions
from tensorflow.keras.preprocessing import image
```

2. Загрузка предобученной модели MobileNet:

```
model = MobileNet(weights='imagenet')
```

Здесь создается экземпляр модели MobileNet с предобученными весами из набора данных ImageNet.

3. Загрузка изображения для распознавания объектов:

```
img_path = 'example_image.jpg'
```

```
img = image.load_img(img_path, target_size=(224, 224))
```

Программа загружает изображение с указанным путем и изменяет его размер до 224x224 пикселей, который является стандартным размером входных изображений для модели MobileNet.

4. Преобразование изображения в массив numpy:

```
img_array = image.img_to_array(img)
```

5. Расширение размерности массива для соответствия ожиданиям модели:

```
img_array = np.expand_dims(img_array, axis=0)
```

Массив изображения расширяется до формы, ожидаемой моделью (batch_size, height, width, channels), где batch_size - количество изображений в пакете, а channels - количество каналов цвета (обычно 3 для RGB).

6. Предобработка изображения для подачи на вход модели:

```
img_array = preprocess_input(img_array)
```

7. Предсказание класса объекта на изображении:

```
predictions = model.predict(img_array)
```

Модель делает предсказание на входном изображении и возвращает вероятности для каждого класса из набора данных ImageNet.

8. Декодирование предсказаний (получение меток классов):




```
decoded_predictions = decode_predictions(predictions, top=3)[0]
```

Предсказанные вероятности декодируются в человек читаемые метки классов. Функция decode_predictions возвращает топ-N предсказаний (в данном случае, топ-3) в удобном для понимания формате.

Таблица функций

Функция	Описание
MobileNet()	Создает экземпляр предобученной модели MobileNet.
image.load_img()	Загружает изображение с указанного пути.
image.img_to_array()	Преобразует изображение в массив numpy.
np.expand_dims()	Расширяет размерность массива для соответствия ожиданиям модели.
preprocess_input()	Предобрабатывает изображение для подачи на вход модели.
model.predict()	Предсказывает класс объекта на изображении.
decode_predictions()	Декодирует предсказания модели, возвращая метки классов и их вероятности.
print()	Выводит результаты предсказаний на экран.

Результаты

№	Изображения	Предсказания программы
1		Predictions: 1: sleeping_bag (0.35) 2: lycaenid (0.28) 3: umbrella (0.08)
2		Predictions: 1: loggerhead (0.98) 2: terrapin (0.01) 3: leatherback_turtle (0.00)
3		Predictions: 1: daisy (1.00) 2: bee (0.00) 3: hair_slide (0.00)

Текст программы

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.applications.mobilenet import MobileNet,
preprocess_input, decode_predictions
from tensorflow.keras.preprocessing import image

# Загрузка предобученной модели MobileNet
model = MobileNet(weights='imagenet')

# Загрузка изображения для распознавания объектов
img_path = 'example_image.jpg'
img = image.load_img(img_path, target_size=(224, 224))

# Преобразование изображения в массив numpy
img_array = image.img_to_array(img)

# Расширение размерности массива для соответствия ожиданиям модели
img_array = np.expand_dims(img_array, axis=0)

# Предобработка изображения для подачи на вход модели
img_array = preprocess_input(img_array)

# Предсказание класса объекта на изображении
predictions = model.predict(img_array)

# Декодирование предсказаний (получение меток классов)
decoded_predictions = decode_predictions(predictions, top=3)[0]

# Вывод результатов
print("Predictions:")
for i, (imagenet_id, label, score) in enumerate(decoded_predictions):
    print(f"{i+1}: {label} ({score:.2f})")
```

2.17. Распознавание позы человека. Movenet

Целью данного проекта является разработка программы для распознавания позы человека на изображениях с помощью предобученной модели Movenet. Задачи проекта включают в себя загрузку изображений, подготовку их для анализа, получение предсказаний о позе человека с помощью модели Movenet и визуализацию результатов.

Модель Movenet — это нейронная сеть, разработанная компанией Google на основе фреймворка TensorFlow [40, 41]. Она обучена на

изображениях людей и способна определять позу человека на изображении, а именно расположение ключевых точек (keypoints), таких как нос, глаза, плечи, локти, запястья, бёдра, колени и лодыжки.

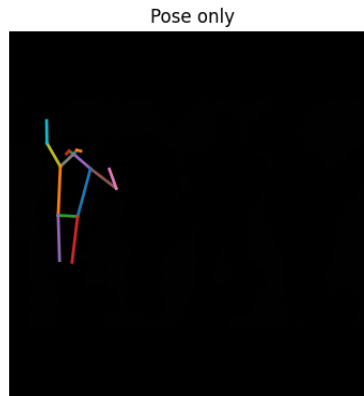
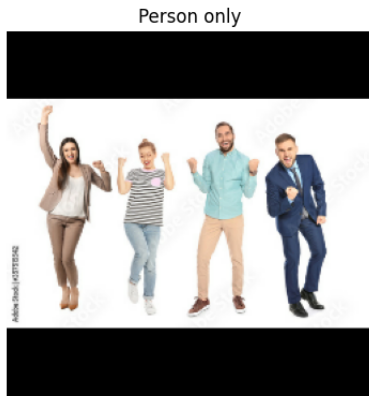
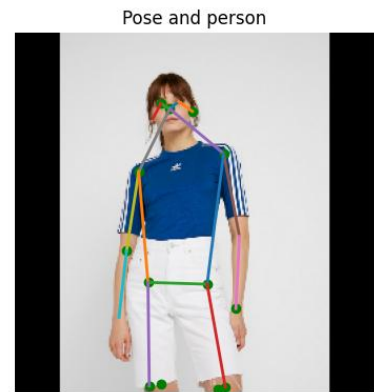
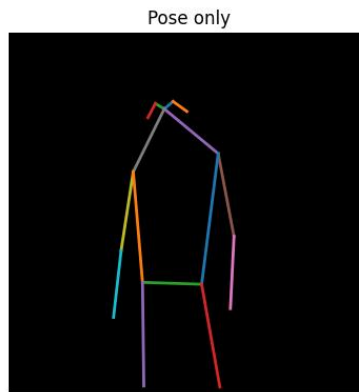
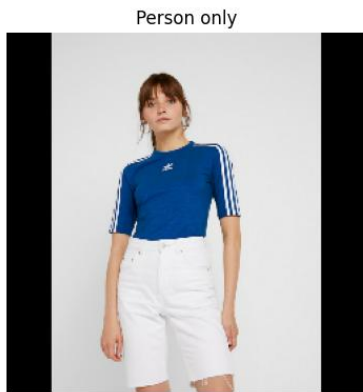
Алгоритм работы программы

1. Загрузка предобученной модели Movenet из локальной папки.
2. Загрузка изображения из локальной папки и подготовка его для передачи входным параметром модели.
3. Получение предсказания для подготовленного изображения с помощью модели Movenet.
4. Обработка предсказаний для получения координат ключевых точек и их достоверности.
5. Визуализация результатов: отрисовка позы человека на изображении.

Использованные функции

Название функции	Назначение
tf.io.read_file	Загрузка изображения из файла
tf.compat.v1.image.decode_jpeg	Декодирование JPEG изображения
tf.image.resize_with_pad	Изменение размера изображения с заполнением поля
tf.expand_dims	Добавление измерения в тензоре
plt.imshow	Отображение изображения в Matplotlib
plt.scatter	Рисование точек на изображении в Matplotlib
plt.plot	Рисование линий на изображении в Matplotlib
plt.subplot	Создание сетки подграфиков в Matplotlib
movenet	Предобученная модель Movenet

Скриншоты



Текст программы

```
import tensorflow as tf
import matplotlib.pyplot as plt
import tensorflow_hub as hub

KEYPOINTS_QUANTITY = 17
MAX_PERSON_QUANTITY = 6
UNTRUSTABLE_KEYPOINT_THRESHOLD = 256 * 0.3
UNTRUSTABLE_PERSON_THRESHOLD = 256 * 0.3

keypoint_labels = [
    "nose", "left eye", "right eye", "left ear", "right ear", "left
    shoulder",
    "right shoulder", "left elbow", "right elbow", "left wrist", "right
    wrist",
    "left hip", "right hip", "left knee", "right knee", "left ankle",
    "right ankle"
]
model = hub.load("movenet")
movenet = model.signatures['serving_default']

image_path = '00030_00.jpg'
image = tf.io.read_file(image_path)
image = tf.compat.v1.image.decode_jpeg(image)
X = tf.expand_dims(image, axis=0)
```

```

X = tf.cast(tf.image.resize_with_pad(X, 256, 256), dtype=tf.int32)

plt.title('Original Image')
plt.imshow(image)
plt.show() # Ensure the plot is displayed

movenet_result = movenet(X)['output_0'].numpy()
movenet_result.shape
predictions = movenet_result[0]

most_accurate_index = predictions[:, 55].argmax()
most_accurate_row = predictions[most_accurate_index, :]
most_accurate_row = (most_accurate_row * 256).astype(float)

keypoints_dict = {}
for i in range(KEYPOINTS_QUANTITY):
    keypoints_dict[keypoint_labels[i]] = {
        "x": most_accurate_row[i * 3],
        "y": most_accurate_row[i * 3 + 1],
        "confidence": most_accurate_row[i + 2]
    }
print(keypoints_dict)

connections = [('nose', 'left eye'), ('left eye', 'left ear'),
               ('nose', 'right eye'), ('right eye', 'right ear'),
               ('nose', 'left shoulder'), ('left shoulder', 'left
elbow'),
               ('left elbow', 'left wrist'), ('nose', 'right shoulder'),
               ('right shoulder', 'right elbow'),
               ('right elbow', 'right wrist'), ('left shoulder', 'left
hip'),
               ('right shoulder', 'right hip'), ('left hip', 'right
hip'),
               ('left hip', 'left knee'), ('right hip', 'right knee')]

def draw_image(img):
    plt.subplot(1, 3, 1)
    plt.title('Person only')
    plt.axis('off')
    plt.imshow(img)

def draw_pose_only(img, keypoints_dict):
    plt.title('Pose only')
    plt.axis('off')
    plt.imshow((img / 255) / 255)
    for start_key, end_key in connections:
        if start_key in keypoints_dict and end_key in keypoints_dict:
            start_point = keypoints_dict[start_key]
            end_point = keypoints_dict[end_key]

```

```

        plt.plot([start_point["y"], end_point["y"]],
                 [start_point["x"], end_point["x"]],
                 linewidth=2)

def draw_pose_and_image(img, keypoints_dict, keypoint_labels):
    plt.title('Pose and person')
    plt.axis('off')
    plt.imshow(img)

    for i in range(KEYPOINTS_QUANTITY):
        if keypoints_dict[keypoint_labels[i]][
            "confidence"] > UNTRUSTABLE_KEYPOINT_THRESHOLD:
            plt.scatter(keypoints_dict[keypoint_labels[i]]["y"],
                        keypoints_dict[keypoint_labels[i]]["x"],
                        color='green')

    for start_key, end_key in connections:
        if start_key in keypoints_dict and end_key in keypoints_dict:
            start_point = keypoints_dict[start_key]
            end_point = keypoints_dict[end_key]
            plt.plot([start_point["y"], end_point["y"]],
                    [start_point["x"], end_point["x"]],
                    linewidth=2)

def visualize(img, keypoints_dict, keypoint_labels, description=''):
    plt.figure(figsize=(15, 5))
    draw_image(img)

    plt.subplot(1, 3, 3)
    draw_pose_and_image(img, keypoints_dict, keypoint_labels)

    plt.subplot(1, 3, 2)
    draw_pose_only(img, keypoints_dict)

    plt.savefig(f'result_{description}.png')
    plt.show() # Ensure the plot is displayed

img = tf.image.resize_with_pad(image, 256, 256)
img = tf.cast(img, dtype=tf.int32)
img = tf.expand_dims(img, axis=0)
img = img.numpy()[0]

visualize(img, keypoints_dict, keypoint_labels, 'single_person')

image_path = 'several_people.jpg'
image = tf.io.read_file(image_path)
image = tf.compat.v1.image.decode_jpeg(image)
X = tf.expand_dims(image, axis=0)
X = tf.cast(tf.image.resize_with_pad(X, 256, 256), dtype=tf.int32)

```

```

plt.title('Original Image')
plt.imshow(image)
plt.show() # Ensure the plot is displayed

movenet_result = movenet(X)['output_0'].numpy()
movenet_result.shape
predictions = movenet_result[0]

img = tf.image.resize_with_pad(image, 256, 256)
img = tf.cast(img, dtype=tf.int32)
img = tf.expand_dims(img, axis=0)
img = img.numpy()[0]
for i in range(MAX_PERSON_QUANTITY):
    row = (predictions[i, :] * 256).astype(float)
    if row[55] <= UNTRUSTABLE_PERSON_THRESHOLD:
        continue

    keypoints_dict = {}
    for j in range(KEYPOINTS_QUANTITY):
        keypoints_dict[keypoint_labels[j]] = {
            "x": row[j * 3],
            "y": row[j * 3 + 1],
            "confidence": row[j + 2]
        }

    visualize(img, keypoints_dict, keypoint_labels,
f'multiple_persons_{i}')

```

2.18. Распознавание буквы на изображении. TensorFlow

Программа распознает букву на изображении. Буква на изображении представляется рукописная русская строчная от «а» до «я». Изображение - это фото с расширением .png.

Программа реализована на основе нейросети tensorflow и содержит два файла learn.py и test.py. Первый отвечает за создание модели нейросети и её обучение, второй непосредственно за распознавание букв на основе обученной модели. Алгоритм основан на машинном обучении, которое и позволяет реализовать распознавание букв.

Нейросеть обучалась по набору данных, состоящего из .png файлов разрешения 32*32, на которых буквы кириллицы написаны разными цветами на листах бумаги. Пример:



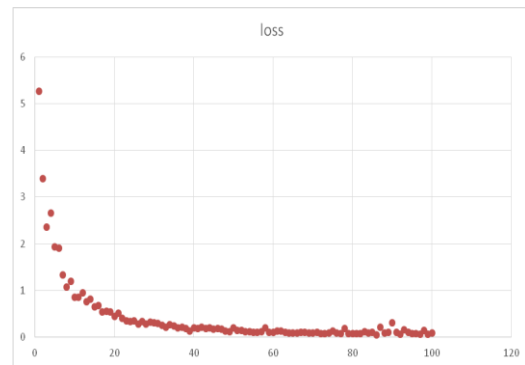
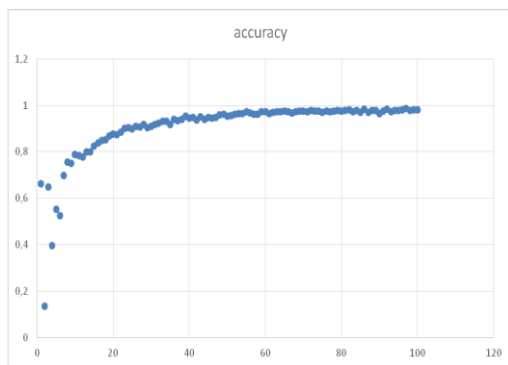
В общей сложности для обучения и тренировки модели было использовано 260 фото на каждую букву (8580 всего).

Библиотеки:

- Tensorflow – для создания и обучения модели
- Numpy – для численных преобразований над входными данными

Оценка результата обучения нейросети

Для оценки качества обучения нейросети использовались следующие метрики: accuracy и loss. Далее приведены графики изменения этих метрик в зависимости от номера итерации тренировки модели:



Модель нейросети tensorflow была успешно обучена для распознавания рукописных букв на кириллице. Это подтверждается статистикой выводимых данных при обучении (accuracy выросло с 15% до 98%, loss упало с 5 до 0.02).

Используемые функции

Функции из библиотеки tensorflow.keras:

Название	Описание
Sequential()	Группирует линейный стек слоев в файл model
Conv2D()	Создает 2D-слой свертки
MaxPooling2D()	Понижает дискретизацию входных данных по их пространственным размерам (высоте и ширине)

Flatten()	Сглаживает ввод
Dense()	Создает обычный плотносвязный слой
Adam()	Оптимизатор, реализующий алгоритм Адама
image_dataset_from_directory()	Генерирует tf.data.Dataset из файлов изображений в каталоге.
prefetch()	Создает объект Dataset, который предварительно выбирает элементы из этого набора данных.
compile()	Настраивает модель для обучения
evaluate()	Возвращает значение потерь и значения метрик для модели в тестовом режиме
fit()	Обучает модель в течение фиксированного количества эпох (итераций набора данных)
save()	Сохраняет модель в файл
load_img()	Загружает изображение из файла
predict()	Формирует выходные прогнозы для входных выборок
load_model()	Загружает модель, сохраненную через model.save()

Функции из библиотеки numpy [42]:

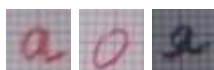
Название	Описание
expand_dims()	Расширяет размеры массива
argmax()	Возвращает индексы максимальных значений по оси

Собственные функции:

Название	Описание
predict_letter()	На основе полученного на вход изображения формирует прогноз о том, какая буква на кириллице изображена на картинке

Скриншоты

На вход подавалось 3 изображения с буквами:



Результат работы программы test.py:

```

1/1 ————— 0s 52ms/step
а
1/1 ————— 0s 14ms/step
ю
1/1 ————— 0s 12ms/step
я

```

Нейросеть правильно определила буквы «а» и «я». Однако она ошиблась с буквой «о», приняв её за «ю».

Данная модель примерно в половине случаев распознает данные корректно, что подтверждается проведенными тестами. Такой результат можно объяснить недостатком исходных данных с учетом большого количества классов данных (33 класса на 33 буквы алфавита), некачественным набором данных для обучения и недостаточным количеством итераций для тренировки модели.

Текст программы

learn.py:

```
import tensorflow
model = tensorflow.keras.Sequential([
    tensorflow.keras.layers.Conv2D(32, (3, 3), activation='relu',
    input_shape=(32, 32, 3)),
    tensorflow.keras.layers.MaxPooling2D((2, 2)),
    tensorflow.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tensorflow.keras.layers.MaxPooling2D((2, 2)),
    tensorflow.keras.layers.Flatten(),
    tensorflow.keras.layers.Dense(128, activation='relu'),
    tensorflow.keras.layers.Dense(33, activation='softmax')
])
optimizer = tensorflow.keras.optimizers.Adam(learning_rate=0.0001)
samples = 250
train_labels = [0]*samples + [1]*samples + [2]*samples + [3]*samples +
[4]*samples + [5]*samples + [6]*samples + [7]*samples + [8]*samples +
[9]*samples + [10]*samples + [11]*samples + [12]*samples + [13]*samples +
[14]*samples + [15]*samples + [16]*samples + [17]*samples + [18]*samples
+ [19]*samples + [20]*samples + [21]*samples + [22]*samples +
[23]*samples + [24]*samples + [25]*samples + [26]*samples + [27]*samples
+ [28]*samples + [29]*samples + [30]*samples + [31]*samples +
[32]*samples
train_ds = tensorflow.keras.preprocessing.image_dataset_from_directory(
    'C:/Users/User/train',
    label_mode='int',
    labels = train_labels,
    shuffle=False,
    seed=1,
    image_size=(32, 32),
    batch_size=4)
samples = 10
test_labels = [0]*samples + [1]*samples + [2]*samples + [3]*samples +
[4]*samples + [5]*samples + [6]*samples + [7]*samples + [8]*samples +
[9]*samples + [10]*samples + [11]*samples + [12]*samples + [13]*samples +
[14]*samples + [15]*samples + [16]*samples + [17]*samples + [18]*samples
+ [19]*samples + [20]*samples + [21]*samples + [22]*samples +
```

```

[23]*samples + [24]*samples + [25]*samples + [26]*samples + [27]*samples
+ [28]*samples + [29]*samples + [30]*samples + [31]*samples +
[32]*samples
test_ds = tensorflow.keras.preprocessing.image_dataset_from_directory(
    'C:/Users/User/test_dir',
    label_mode='int',
    labels = test_labels,
    shuffle=False,
    seed=1,
    image_size=(32, 32),
    batch_size=4)
AUTOTUNE = tensorflow.data.AUTOTUNE
train_ds = train_ds.prefetch(buffer_size=AUTOTUNE)
test_ds = test_ds.prefetch(buffer_size=AUTOTUNE)
model.compile(optimizer=optimizer,loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
model.fit(train_ds, epochs=100, batch_size=5, validation_data=test_ds)
loss, accuracy = model.evaluate(test_ds)
model.save('model3.h5')
model.save('model3.keras')

```

test.py

```

import tensorflow
import numpy as np

def predict_letter(model, image_file):

arr=['a','б','в','г','д','е','ё','ж','з','и','й','к','л','м','н','о','п',
'р','с','т','у','ф','х','ц','ч','ш','щ','ь','ы','ъ','э','ю','я']
    image_size = 32
    img = tensorflow.keras.preprocessing.image.load_img(image_file,
target_size=(image_size, image_size), color_mode='rgb')
    img_arr = np.expand_dims(img, axis=0)
    img_arr = img_arr
    img_arr = img_arr.reshape((1, 32, 32, 3))
    result = model.predict(img_arr)[0]
    digit = np.argmax(result)
    return arr[digit]

model = tensorflow.keras.models.load_model('model3.keras')
print(predict_letter(model, 'aaaaa.png'))
print(predict_letter(model, 'ooooo.png'))
print(predict_letter(model, '5.png'))

```

2.19. Распознавание позиции текста. EAST OpenCV

Целью данного проекта является разработка программы для распознавания позиции текста на изображениях с помощью

предобученной модели EAST OpenCV. Задачи проекта включают в себя загрузку изображений, подготовку их для анализа, получение предсказаний о позиции текста при помощи модели EAST OpenCV, обработку предсказаний и визуализацию результатов [43, 44].

Модель EAST — это нейронная сеть, основана на новой архитектуре и шаблоне обучения, что позволяет ей производить анализ изображения практически в реальном времени со скоростью 13 кадров в секунду. Ключевым компонентом предлагаемого алгоритма является модель нейронной сети, которая обучена непосредственно предсказывать существование текстовых экземпляров и их геометрию на всем изображении. Модель представляет собой полностью сверточную нейронную сеть (Fully Convolutional Network), адаптированную для распознавания текста, которая выдает точные попиксельные предсказания слов или строк текста [44, 45].

Данная нейронная сеть обучена на изображениях, содержащих текст в различных условиях видимости и способна определять текст под разными углами и с разным расстоянием на изображении.

Алгоритм работы программы

1. Загрузка предобученной модели EAST из локальной папки.
2. Загрузка изображения из локальной папки и подготовка его для передачи входным параметром модели.
3. Получение предсказания для подготовленного изображения с помощью модели EAST.
4. Обработка предсказаний для получения точных координат точек расположения текста и их достоверности.
5. Визуализация результатов: отрисовка границ на изображении.

Использованные функции

Название функции	Назначение
cv2.imread	Загрузка изображения из файла
cv2.resize	Изменение размера изображения
cv2.dnn.readNet	Определение модели EAST
cv2.dnn.blobFromImage	Определение текста при помощи EAST
geometry	Анализ оценки результатов
non_max_suppression	Получение точных результатов
cv2.rectangle	Рисование квадрата для текста
cv2.imshow	Отображение результата

Скриншоты



Текст программы

```
# Импортируем необходимые пакеты
from imutils.object_detection import non_max_suppression
import numpy as np
import argparse
import time
import cv2

# Создаём анализатор аргументов и разбираем аргументы
ap = argparse.ArgumentParser()
ap.add_argument("-i", "--image", type=str,
                help="путь к входному изображению")
ap.add_argument("-east", "--east", type=str,
```

```

        help="путь к модели EAST для обнаружения текста")
ap.add_argument("-c", "--min-confidence", type=float, default=0.5,
                help="минимальная вероятность для рассмотрения области")
ap.add_argument("-w", "--width", type=int, default=320,
                help="ширина изменённого изображения (должна быть кратной 32)")
ap.add_argument("-e", "--height", type=int, default=320,
                help="высота изменённого изображения (должна быть кратной 32)")
args = vars(ap.parse_args())

# Загружаем входное изображение и получаем его размеры
image = cv2.imread(args["image"])
orig = image.copy()
(H, W) = image.shape[:2]

# Задаём новую ширину и высоту, а затем вычисляем соотношение изменения
# для обеих ширины и высоты
(newW, newH) = (args["width"], args["height"])
rW = W / float(newW)
rH = H / float(newH)

# Изменяем размер изображения и получаем новые размеры изображения
image = cv2.resize(image, (newW, newH))
(H, W) = image.shape[:2]

# Определяем два имени выходных слоёв для модели EAST, которые нас
интересуют -
# первый выход - вероятности, а второй может быть использован для
вычисления координат ограничивающего прямоугольника текста
layerNames = [
    "feature_fusion/Conv_7/Sigmoid",
    "feature_fusion/concat_3"]

# Загружаем предобученную модель EAST для обнаружения текста
print("[INFO] загружаем детектор текста EAST...")
net = cv2.dnn.readNet(args["east"])

# Создаём blob из изображения, а затем выполняем прямой проход модели для
получения двух наборов выходных слоёв
blob = cv2.dnn.blobFromImage(image, 1.0, (W, H),
                              (123.68, 116.78, 103.94), swapRB=True, crop=False)
start = time.time()
net.setInput(blob)
(scores, geometry) = net.forward(layerNames)
end = time.time()

# Отображаем информацию о времени обнаружения текста
print("[INFO] обнаружение текста заняло {:.6f} секунд".format(end -
start))

```

```

# Получаем количество строк и столбцов из объема оценок, затем
# инициализируем набор прямоугольников ограничивающих прямоугольников и
# соответствующие
# оценки уверенности
(numRows, numCols) = scores.shape[2:4]
rects = []
confidences = []

# Цикл по числу строк
for y in range(0, numRows):
    # Извлекаем оценки (вероятности), за которыми следуют
    геометрические
    # данные, используемые для получения потенциальных координат
    ограничивающих прямоугольников
    # окружающих текст
    scoresData = scores[0, 0, y]
    xData0 = geometry[0, 0, y]
    xData1 = geometry[0, 1, y]
    xData2 = geometry[0, 2, y]
    xData3 = geometry[0, 3, y]
    anglesData = geometry[0, 4, y]

    # Цикл по числу столбцов
    for x in range(0, numCols):
        # Если наша оценка не имеет достаточной вероятности,
        игнорируем её
        if scoresData[x] < args["min_confidence"]:
            continue

        # Вычисляем коэффициент смещения, так как наши результирующие
        карты объектов
        # будут в 4 раза меньше, чем входное изображение
        (offsetX, offsetY) = (x * 4.0, y * 4.0)

        # Извлекаем угол поворота для предсказания, а затем
        # вычисляем синус и косинус
        angle = anglesData[x]
        cos = np.cos(angle)
        sin = np.sin(angle)
        # Используем объем геометрии для определения ширины и высоты
        # ограничивающего прямоугольника
        h = xData0[x] + xData2[x]
        w = xData1[x] + xData3[x]

# Вычисляем начальные и конечные координаты (x, y) для
# ограничивающего прямоугольника текста
endX = int(offsetX + (cos * xData1[x]) + (sin * xData2[x]))
endY = int(offsetY - (sin * xData1[x]) + (cos * xData2[x]))
startX = int(endX - w)

```

```

        startY = int(endY - h)
# Добавляем координаты ограничивающего прямоугольника и оценку
вероятности
        # в соответствующие списки
        rects.append((startX, startY, endX, endY))
        confidences.append(scoresData[x])
# Применяем не максимальное подавление для подавления слабых,
перекрывающихся ограничивающих прямоугольников
boxes = non_max_suppression(np.array(rects), probs=confidences)
# Цикл по ограничивающим прямоугольникам
for (startX, startY, endX, endY) in boxes:
    # Масштабируем координаты ограничивающего прямоугольника на основе
соответствующих
    # коэффициентов
    startX = int(startX * rW)
    startY = int(startY * rH)
    endX = int(endX * rW)
    endY = int(endY * rH)
    # Рисуем ограничивающий прямоугольник на изображении
    cv2.rectangle(orig, (startX, startY), (endX, endY), (0, 255, 0), 2)
# Отображаем выходное изображение
cv2.imshow("Text Detection", orig)
cv2.waitKey(0)

```

2.20. Распознавание размеров на схеме товара. Tesseract

Целью данного проекта является разработка программы для распознавания размеров товара на его схеме с помощью предобученной модели Tesseract [46, 47]. Задачи проекта включают в себя загрузку изображений, подготовку их для анализа, получение распознавание размеров товара с помощью модели Tesseract и вывод результата в Excel файл.

Модель Tesseract — это модель, разработанная компанией Google на основе технологии OCR. OCR (optical character recognition) – оптическое распознавание символов. Она обучена распознавать печатный и рукописный текст. Ее также можно использовать для распознавания блоков текста, например, для обработки писем или pdf-документов.

Алгоритм работы программы

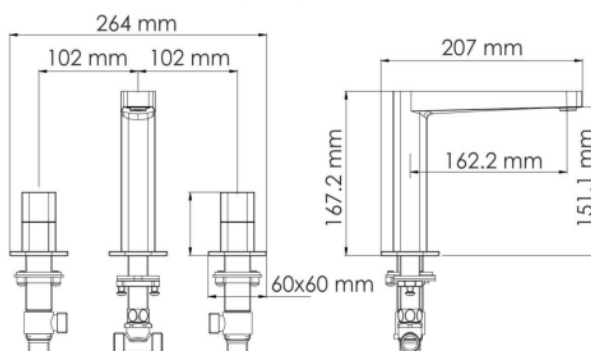
1. Загрузка Excel файла, содержащего ссылку на схему товара и потенциальные размеры.
2. Загрузка предобученной модели Tesseract из локальной папки.
3. Загрузка изображения по ссылке и подготовка его для передачи входным параметром модели.
4. Получение распознавания размеров товара на схеме с помощью Tesseract.
5. Обработка результатов распознавания и соотношение потенциальных и предсказанных размеров.
6. Визуализация результатов: применение стилей к исходному Excel документу.

Использованные функции

Название функции	Назначение
<code>urllib.request.urllopen</code>	Загрузка изображения по URL
<code>cv2.rotate</code>	Поворот изображения, чтобы распознать числа написанные вертикально
<code>cv2.resize</code>	Изменение размера изображения в соответствии с требованиями Tesseract
<code>re.search</code>	Поиск текста соответствующего маске
<code>pandas.read_excel</code>	Открытие Excel файла
<code>pytesseract</code>	Предобученная модель Tesseract

Результаты работы программы

Пример схемы



Результаты распознавания:

Все распознанные значения: 162.2, 102.0, '60x60', 264.0, 151.1, 167.2, 207.0

Значения, распознанные по горизонтали: 162.2, 102.0, '60x60', 264.0, 207.0

Значения, распознанные по вертикали: 167.2, 151.1

Пример выходного файла:

Артикул	Ширина	Высота	Схема изделия
9135	26,4	16,8	https://wasserkraft.ru
A113.116.058.CH Thermo	30	125	https://wasserkraft.ru
3799	11,2	15,1	https://wasserkraft.ru
A113.116.101.CH Thermo	30	125	https://wasserkraft.ru
10BDT.020	50	112	https://wasserkraft.ru
A246	6,8	92	https://wasserkraft.ru
5140	64	3	https://wasserkraft.ru
6011	63,5	7	https://wasserkraft.ru
A8651.090.118			
6027	12	38	https://wasserkraft.ru
461120	110-200		https://wasserkraft.ru
86TLT.040.SE.BL	48,3	100,4	https://wasserkraft.ru
A188.117.127.010.CH Thermo	40	125	https://wasserkraft.ru
K-75992			
A3151.269.270.090.116.271.087.103			

Зеленый – все хорошо

Желтый – вероятно, перепутаны высота и ширина

Красный – такое значение не распознано на схеме

Серый – ссылки на схему нет или на схеме нет чисел

Текст программы

```
get_dimensions.py
import re
import numpy as np
import cv2
from urllib.request import urlopen
import pytesseract
pytesseract.pytesseract.tesseract_cmd = r'C:\Program Files\Tesseract-OCR\tesseract.exe'

def url_to_image(url, readFlag=cv2.IMREAD_COLOR):
    try:
        resp = urlopen(url)
    except:
        return None
    image = np.asarray(bytearray(resp.read()), dtype="uint8")
    image = cv2.imdecode(image, readFlag)
    return image

def convert_to_num(s):
    try:
        return float(s)
    except ValueError:
        return s

def get_trailing_number(s):
    s = s.replace(',', '.').replace('\n', ' ').replace('\t', ' ')
```

```

s = s.split(' ')[-1]
m = re.search(r'(@*\d*[\.\-~x]*\d+)', s)
if m is None:
    return None

s_res = m.group()
if s != s_res:
    return None

return convert_to_num(s_res)
def get_dimensions_on_image(img, unit='mm'):
    dimensions = set()
    text = pytesseract.image_to_string(img, config=r'--oem 3 --psm 11')
    text = text.replace(',', '.').lower()

    if 'mm' in text:
        for num in text.split('mm'):
            dimensions.add(get_trailing_number(num.strip()))

    if 'cm' in text:
        for num in text.split('cm'):
            res = get_trailing_number(num.strip())
            if res is None:
                continue
            elif type(res) == str:
                dimensions.add(res)
            else:
                dimensions.add(res * 10)

    dimensions.discard(None)
    return dimensions
MAX_SIZE = 1000
def resize_image(img):
    height, width, channels = img.shape
    if height <= MAX_SIZE and width <= MAX_SIZE:
        return img
    elif height >= width:
        width = int(MAX_SIZE * width / height)
        height = MAX_SIZE
        return cv2.resize(img, (height, width))
    else:
        height = int(MAX_SIZE * height / width)
        width = MAX_SIZE
        return cv2.resize(img, (height, width))

def get_dimensions_on_image_with_rotation(img, unit='mm',
                                          full_rotation = False):
    all_dimensions = set()
    horizontal_values = set()

```

```

vertical_values = set()

result = get_dimensions_on_image(img, unit)
all_dimensions.update(result)
horizontal_values.update(result)
img = cv2.rotate(img, cv2.ROTATE_90_CLOCKWISE)
result = get_dimensions_on_image(img, unit)
all_dimensions.update(result)
vertical_values.update(result)

if full_rotation:
    img = cv2.rotate(img, cv2.ROTATE_90_CLOCKWISE)
    result = get_dimensions_on_image(img, unit)
    all_dimensions.update(result)
    horizontal_values.update(result)

    img = cv2.rotate(img, cv2.ROTATE_90_CLOCKWISE)
    result = get_dimensions_on_image(img, unit)
    all_dimensions.update(result)
    vertical_values.update(result)
return {"all": all_dimensions,
        "horizontal": horizontal_values,
        "vertical": vertical_values}
def get_dimensions_with_url(url, unit='mm', full_rotation = False):
    img = url_to_image(url)
    if img is None:
        return None
    img = resize_image(img)
    return get_dimensions_on_image_with_rotation(img, unit,
full_rotation)

```

```

main.py
import time
import json
import pandas as pd
import numpy as np
from get_dimensions import get_dimensions_with_url, convert_to_num
config = json.load(open("config.json", encoding="utf-8"))
FILE_PATH = config["Путь до файла"]
print('FILE_PATH', FILE_PATH)
SKIPROWS = config["Пропустить строк до заголовка"]
SCHEME_ROW = config["Заголовок столбца со схемой"]
FULL_ROTATION = config["Полный разворот картинки"]
UNIT = config["Единица измерения на схемах"]
WIDTH = config["Заголовок столбца с шириной"]
HEIGHT = config["Заголовок столбца с высотой"]
SIZE_COEF = config["Коэффициент преобразование размера"]
EPSILON = config["Точность в мм"]
def nearly_in(val, seq, epsilon):

```

```

if type(val) == str:
    for other_val in seq:
        if type(other_val) == str and val in other_val:
            return True
else:
    for other_val in seq:
        if type(other_val) == str:
            if str(val) in other_val:
                return True
            elif str(int(val / SIZE_COEF)) in other_val:
                return True
            elif str(val / SIZE_COEF) in other_val:
                return True
            elif str(val / SIZE_COEF).replace('.', ',') in other_val:
                return True
        elif abs(val - other_val) <= epsilon:
            return True
return False

def apply_style(df, styler, dimensions, target_dim, i, df_column):
    val = convert_to_num(df.loc[i, df_column].replace(",",".")) \
        if type(df.loc[i, df_column]) == str else df.loc[i,
df_column]

    if type(val) == str and len(val) == 0:
        styler = styler.map(lambda x: 'background-color: gray',
subset=pd.IndexSlice[i, df_column])
        return styler

    if type(val) != str and np.isnan(val):
        styler = styler.map(lambda x: 'background-color: gray',
subset=pd.IndexSlice[i, df_column])
        return styler

    val = val if type(val) == str else val * SIZE_COEF
    if val in dimensions[target_dim]:
        styler = styler.map(lambda x: 'background-color: #a4c639',
subset=pd.IndexSlice[i, df_column])
    elif val in dimensions['all']:
        styler = styler.map(lambda x: 'background-color: #32a895',
subset=pd.IndexSlice[i, df_column])
    elif nearly_in(val, dimensions['all'], EPSILON):
        styler = styler.map(lambda x: 'background-color: #e9d66b',
subset=pd.IndexSlice[i, df_column])
    else:
        styler = styler.map(lambda x: 'background-color: #cb4154',
subset=pd.IndexSlice[i, df_column])
    return styler

```

```

VER_DIM_COL = 'Размеры по вертикали'
HOR_DIM_COL = 'Размеры по горизонтали'
ALL_DIM_COL = 'Все размеры'
df = pd.read_excel(FILE_PATH, skiprows=SKIPROWS)
df = df.loc[:, ["Артикул", "Ширина изделия см", "Высота изделия см",
"Глубина изделия см", "Схема изделия"]]
df[VER_DIM_COL] = ""
df[HOR_DIM_COL] = ""
df[ALL_DIM_COL] = ""
styler = df.style
results = { VER_DIM_COL: [],
            HOR_DIM_COL: [],
            ALL_DIM_COL: [],
            }

start_time = time.time()
time_before = time.time()
scheme_paths = df[SCHEME_ROW]
print('rows count:', len(list(scheme_paths)))
counter = 0
for i, path in enumerate(scheme_paths):
    time_before = time.time()
    if pd.isna(path):
        styler = styler.map(lambda x: 'background-color: gray',
subset=pd.IndexSlice[i, :])
        results[VER_DIM_COL].append("")
        results[HOR_DIM_COL].append("")
        results[ALL_DIM_COL].append("")
        continue

    dimensions = get_dimensions_with_url(path, UNIT, FULL_ROTATION)
    if dimensions is None:
        styler = styler.map(lambda x: 'background-color: gray',
subset=pd.IndexSlice[i, :])
        results[VER_DIM_COL].append("-")
        results[HOR_DIM_COL].append("-")
        results[ALL_DIM_COL].append("-")
        continue
    counter+= 1
    results[VER_DIM_COL].append(" ".join([str(dim) for dim in
dimensions['vertical']]))
    results[HOR_DIM_COL].append(" ".join([str(dim) for dim in
dimensions['horizontal']]))
    results[ALL_DIM_COL].append(" ".join([str(dim) for dim in
dimensions['all']]))
    styler = apply_style(df, styler, dimensions, 'horizontal', i, WIDTH)
    styler = apply_style(df, styler, dimensions, 'vertical', i, HEIGHT)
for key in results:
    df[key] = results[key]

```

```

styler.to_excel("output\\output.xlsx")
def get_result_dataframe():
    return styler
print('full_time', time.time() - start_time, 'counter', counter)

```

2.21. Распознавание текстовой информации на скане изображения YOLOv8

Использовались несколько библиотек для решения разных задач: распознавание границ листка, создание скана и чтения текста с данного изображения. Для распознавания границ листа использовалось YOLOv8 - это мощный алгоритм обнаружения объектов, разработанный Ultralytics.

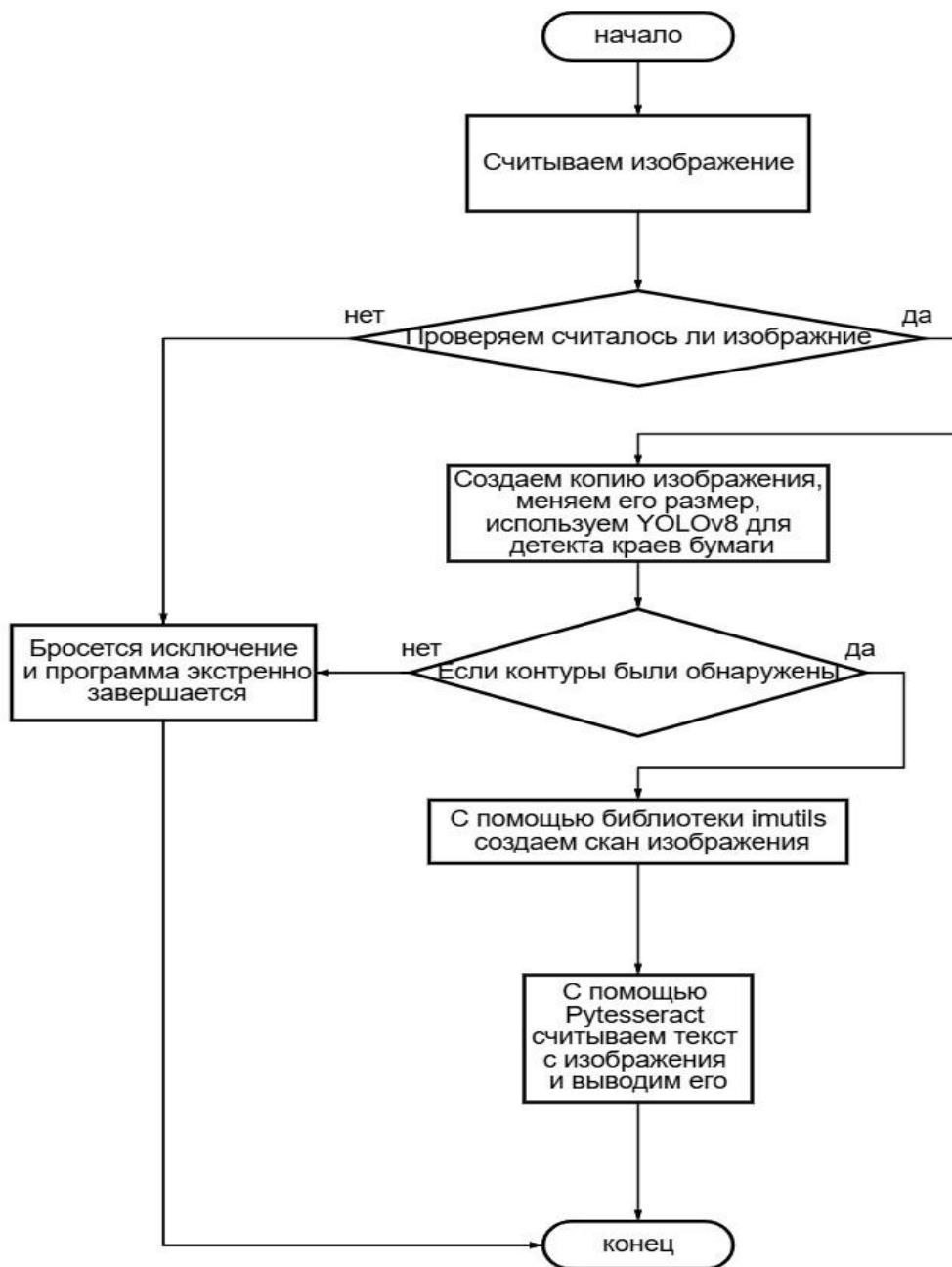
Для создания скана использовалось две библиотеки `imutils` и `OpenCV`. `Imutils` - это пакет, основанный на `OpenCV`, который может легко реализовать серию операций, таких как перевод изображения, вращение, масштабирование и скелет. Для распознавания текст используется `PuTesseract` — это механизм распознавания текста (OCR) с открытым исходным кодом, доступный по лицензии `Apache 2.0`. OCR (optical character recognition,) — механический или электронный перевод изображений рукописного, машинописного или печатного текста в текстовые данные, использующиеся для представления символов в компьютере (например, в текстовом редакторе). Отдельной задачей является задача распознавания музыкальной нотации (нот) [48].

Используемые функции

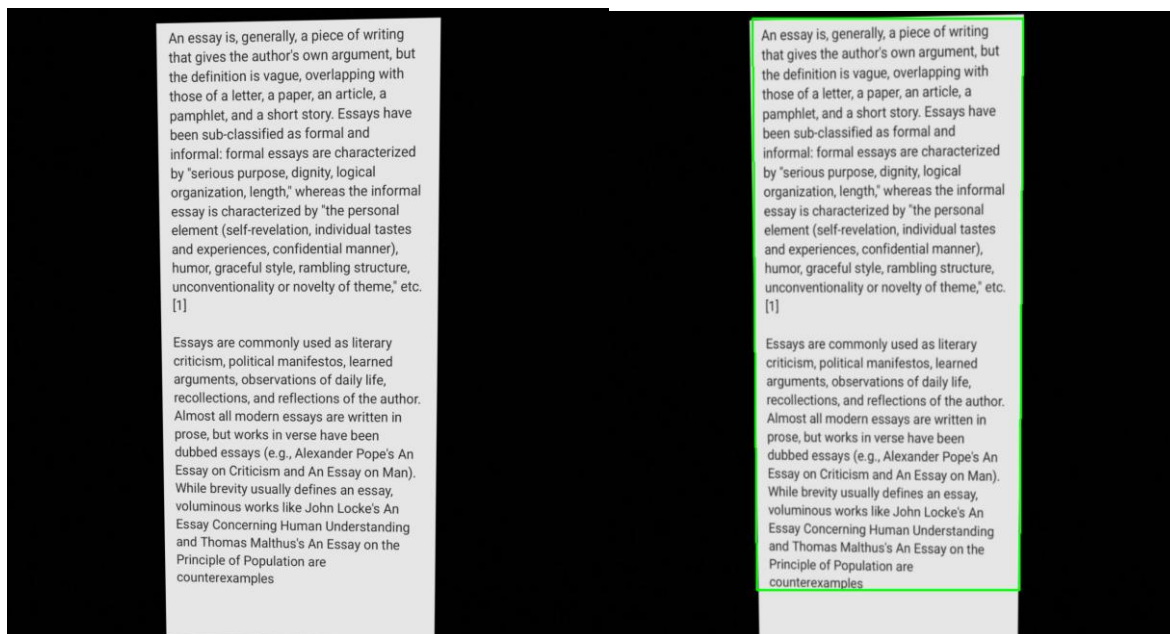
Функция	Описание
<code>imutils.resize</code>	Изменяет размер изображения
<code>model.predict</code>	Предсказывает есть ли на изображении лист и дает координаты его краёв, если они есть
<code>cv2.arcLength</code>	Расчитывает длину и узнает является ли данная линия замкнутой
<code>cv2.approxPolyDP</code>	Аппроксимирует кривую или многоугольник другой кривой/многоугольником с меньшим количеством вершин так, чтобы расстояние между

	ними было меньше или равно указанной точности. Он использует алгоритм Дугласа-Пейкера.
<code>imutils.perspective.four_point_transform</code>	Выполняет 4-точечное перспективное преобразование ROI на изображении и получить вид сверху вниз (т.е. скан)
<code>cv2.imshow</code>	Выводит изображение
<code>cv2.cvtColor</code>	Используется для преобразования изображения из одного цветового пространства в другое
<code>re.findall</code>	Находит нужный шаблон текста по заданному регулярному выражению
<code>ap.add_argument</code>	Добавляет нужные аргументы во время запуска скрипта
<code>cv2.imread</code>	Считывает изображение
<code>cv2.copy</code>	Копирует считанное изображение

Алгоритм

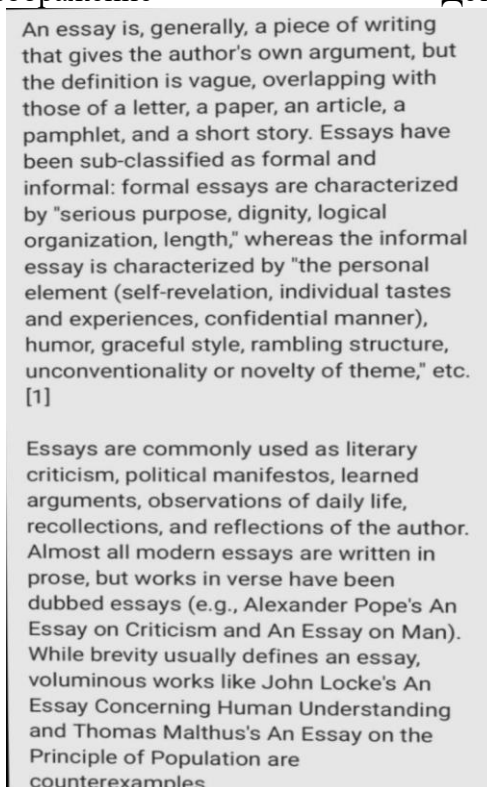


Результаты работы



Исходное изображение

Детект контуров



Создание скана

Результат:

“An essay is, generally, a piece of writing that gives the author's own argument, but the definition is vague, overlapping with those of a letter, a paper, an article, a pamphlet, and a short story. Essays have been sub-classified as formal and informal: formal essays are characterized by "serious purpose, dignity, logical organization, length," whereas the informal essay is characterized by "the personal element (self-revelation, individual tastes and experiences, confidential manner), humor, graceful style, rambling structure, unconventionality or novelty of theme," etc. 1] Essays

are commonly used as literary criticism, political manifesto, learned arguments, observations of daily life, recollections, and reflections of the author. Almost all modern essays are written in prose, but works in verse have been dubbed essays (e.g., Alexander Pope's *An Essay on Criticism* and *An Essay on Man*). While brevity usually defines an essay, voluminous works like John Lock's *An Essay Concerning Human Understanding* and Thomas Malleus's *An Essay on the Principle of Population* are counterexamples"

Обучение модели YOLOv8:

https://colab.research.google.com/drive/12w3lSB90g--WwxtTRwHu_-81TMsEXmsc?usp=sharing

Текст программы

```
from imutils.perspective import four_point_transform
from ultralytics import YOLO
from autocorrect import Speller
import numpy as np
import pandas as pd
import pytesseract
import argparse
import imutils
import cv2
import re

# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-i", "--image", required=True,
    help="path to input image")
ap.add_argument("-d", "--debug", type=int, default=-1,
    help="whether or not we are visualizing each step of the pipeline")
ap.add_argument("-c", "--min-conf", type=int, default=0,
    help="minimum confidence value to filter weak text detection")
args = vars(ap.parse_args())
# load the input image from disk, resize it, and compute the ratio
# of the *new* width to the *old* width
orig = cv2.imread(args["image"])
image = orig.copy()
image = imutils.resize(image, height=800)
ratio = orig.shape[1] / float(image.shape[1])

model = YOLO('nanobest.pt')
names = model.model.names
results = model.predict(image, conf=0.4) #Adjust conf threshold
contours = results[0].masks.xy

# initialize a contour that corresponds to the business card outline
cardCnt = None
# loop over the contours
```

```

for c in contours:
    # approximate the contour
    peri = cv2.arcLength(c, True)
    approx = cv2.approxPolyDP(c, 0.02 * peri, True)
    # if this is the first contour we've encountered that has four
    # vertices, then we can assume we've found the business card
    if len(approx) == 4:
        cardCnt = approx
        break
# if the business card contour is empty then our script could not
# find the outline of the card, so raise an error
if cardCnt is None:
    raise Exception(("Could not find receipt outline. "
                    "Try debugging your edge detection and contour steps. "))
# check to see if we should draw the contour of the business card
# on the image and then display it to our screen

cardCnt = np.array(cardCnt).reshape((-1,1,2)).astype(np.int32)

if args["debug"] > 0:
    output = image.copy()
    cv2.drawContours(output, [cardCnt], -1, (0, 255, 0), 2)
    cv2.imshow("Business Card Outline", output)
    cv2.waitKey(0)
# apply a four-point perspective transform to the *original* image to
# obtain a top-down bird's-eye view of the business card
card = four_point_transform(orig, cardCnt.reshape(4, 2) * ratio)

# show transformed image
cv2.imshow("Business Card Transform", card)
cv2.waitKey(0)
# convert the business card from BGR to RGB channel ordering and then
# OCR it
rgb = cv2.cvtColor(card, cv2.COLOR_BGR2RGB)

pytesseract.pytesseract.tesseract_cmd =
r'C:\Users\beedz\AppData\Local\Tesseract-OCR\tesseract.exe'
EngText = pytesseract.image_to_string(rgb)

en_spell = Speller(lang='en')

corrected_eng_text = [en_spell(word) for word in EngText.split()]
corrected_eng_text = ' '.join(corrected_eng_text)

print(corrected_eng_text)

```

2.22. Замена части изображения по его маске и текстовому запросу.

Stable Diffusion

Программа генерирует новую версию определённой при помощи Stable Diffusion маски части изображения по заданному промпту - текстовому запросу. При помощи данной программы можно как добавлять какие-то новые элементы на выделенную часть, так и удалять оттуда объекты, написав в промпте примерное описание фона.

В программе используется предобученная модель Stable Diffusion [49]. В основе Stable Diffusion лежит техника латентной диффузии — когда в процессе генерации система находит знакомые формы, известные по большому пулу изображений, на которых она была обучена, а затем фокусируется на изображениях, наиболее совпадающих по своему описанию с запросом. То есть она изначально генерирует случайные шумы, а потом на каждом шаге (step) пайплайна изменяет их в нужном направлении, чтобы результат становился всё более похожим на изначальный запрос. По умолчанию значение steps равно 50. Чем больше шагов – тем качественнее получается изображение, но тем дольше длится процесс генерации. Тестовым путём было выяснено, что для генерации достаточно качественного изображения при не слишком больших временных затратах, оптимальным значением является steps=15...30.

В качестве изображения и маски при тестировании передавались файлы с расширением .png или .jpg

Библиотеки

- Sys - обеспечивает доступ к некоторым переменным, используемым или поддерживаемым интерпретатором, а также к функциям, которые тесно взаимодействуют с интерпретатором

- Torch – включает в себя набор инструментов для работы с моделями, в данной работе используется для создания генератора, при помощи которого будут запускаться пайплайны модели
- Diffusers – содержит в себе различные предобученные модели класса Stable Diffusion для создания изображений и аудио

Алгоритм работы программы


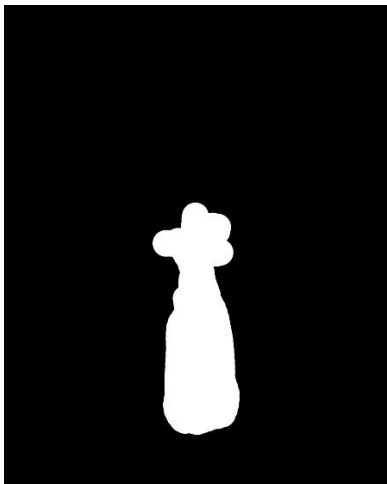

1. Загрузка изображения из локальной папки и проверка его параметров и подготовка его для передачи входным параметром модели.
2. Ввод названий файлов изображения и маски, а также промпта, количества шагов и количества вариантов генерации
3. Создание пайплайна на основе библиотечного AutoPipelineForInpainting
4. Размытие краёв маски, чтобы был более плавный переход в основную неизменяемую часть изображения при генерации
5. Запуск пайплайнов нужное количество раз (один пайплайн – один вариант генерации)
6. Сохранение полученных результатов генерации

Используемые функции

Название	Описание
Библиотека Sys	
exit()	Завершение работы программы, используется в программе в случае передачи некорректных данных
Библиотека Torch	
cuda.is_available()	Возвращает логическое значение, указывающее, доступна ли CUDA в данный момент
Generator()	Создает и возвращает объект-генератор, который управляет состоянием алгоритма, создающего псевдослучайные числа Данный генератор далее используется для генерации изображения

<code>manual_seed()</code>	Устанавливает <code>seed</code> - начальное число - для генерации случайных чисел для текущего графического процессора Изначально <code>seed = -1</code> , и генерации при установке определённого числа нейросеть будет «запоминать» генерацию для определённых исходных файлов и промпта, и дальше генерировать более похожие друг на друга изображения
Библиотека Diffusers	
<code>utils.load_image()</code>	Загружает изображение из локального диска в программу
<code>AutoPipelineForInpainting.from_pretrained()</code>	Возвращает готовый пайплайн для inpainting – изменения части изображения – на основе предобученной модели Stable Diffusion
<code>to()</code>	Возвращает пайплайн для определённого устройства генерации
<code>pipeline.mask_processor.blur()</code>	Размывает края маски
<code>pipeline().images[n]</code>	Генерирует <code>n</code> изображений по заданным данным
Функции вне библиотек	
<code>input()</code>	Ввод с консоли
<code>print()</code>	Вывод в консоль
<code>save()</code>	Сохранение файла, в данном случае – изображения, полученного в результате генерации

Результаты

Исходное изображение	Маска	Результат
		
<p>промпт – «<i>dry grass in a heavy fog background, realistic, high resolution, 8k</i>»</p> <p>количество шагов - 30</p>		

Исходное изображение	Маска	Результат
		
<p>промпт – «<i>white owl flying in the bright blue sky, realistic, high resolution, 8k</i>»</p> <p>количество шагов – 30</p>		
Исходное изображение	Маска	Результат
		
<p>промпт – «<i>white big round moon in old anime style</i>»</p> <p>количество шагов – 15</p>		

Текст программы

```
import sys
import torch
from diffusers import AutoPipelineForInpainting
from diffusers.utils import load_image

device = "cuda" if torch.cuda.is_available() else "cpu"
generator = torch.Generator(device).manual_seed(0)
initial = input("введите название исходного файла\n")
initialMask = input("введите название маски, по которой надо изменять картинку\n")
initialImage = load_image(initial)
initialMaskImage = load_image(initialMask)
if initialImage.size != initialMaskImage.size:
    print("разный размер изображения и маски, выход из программы\n")
    sys.exit()
sizeInitialWidth, sizeInitialLength = initialImage.size
```

```

sizeInitialWidth = (sizeInitialWidth // 8) * 8
sizeInitialWidth = (sizeInitialWidth // 8) * 8

prompt = input("введите промпт\n")
negativePrompt = "ugly, repulsive, bad anatomy, deformed"
steps = int(input("введите количество шагов преобразования\n"))
amount = int(input("введите нужное количество вариантов генерации\n"))
if amount <= 0:
    print("запрошенное количество изображений <= 0, выход из
программы\n")
    sys.exit()

model = "runwayml/stable-diffusion-v1-5"
pipeline = AutoPipelineForInpainting.from_pretrained(
    model,
    low_cpu_mem_usage=True,
    use_safetensors=True
).to(device)

blurredMask = pipeline.mask_processor.blur(
    initialMaskImage,
    blur_factor=20
)

output = pipeline(
    prompt=prompt,
    negative_prompt=negativePrompt,
    width=sizeInitialWidth,
    height=sizeInitialLength,
    image=initialImage,
    mask_image=blurredMask,
    generator=generator,
    num_inference_steps=steps
).images[amount]
for i in range (0, amount):
    output[i].save("new_image_" + str(i) + ".png")

```

2.23. Распознавание марок автомобилей по фотографии. ResNet50

Программа реализована с использованием фреймворка глубокого обучения TensorFlow и архитектуры ResNet50. ResNet50 — это сверточная нейронная сеть с 50 слоями, которая известна своей высокой точностью в задачах распознавания изображений.

Алгоритм работы программы

1. Загрузка предобученной модели ResNet50: Используется модель ResNet50, обученная на большом наборе данных ImageNet.
2. Модификация модели: Верхние слои ResNet50 заменяются новыми слоями для классификации марок автомобилей.
3. Загрузка и предобработка данных: Изображения автомобилей загружаются и преобразуются к нужному размеру и формату.
4. Обучение модели: Модель обучается на наборе данных с фотографиями автомобилей различных марок.
5. Fine-tuning: Дополнительное обучение "размороженных" слоев ResNet50 для адаптации модели к specific задаче.
6. Распознавание марки автомобиля: Программа принимает на вход фотографию автомобиля и предсказывает его марку с определенной вероятностью.

Используемые технологии

Python: Язык программирования, используемый для написания программы.

TensorFlow: Фреймворк глубокого обучения, используемый для создания и обучения модели.

ResNet50: Архитектура сверточной нейронной сети, используемая для распознавания изображений.

Алгоритм работы

1. model.py:

Определение архитектуры модели ResNet50 с добавлением новых слоев для классификации марок автомобилей.

2. train.py:

Загрузка предобученной модели ResNet50.

Создание генераторов данных с помощью ImageDataGenerator для загрузки и аугментации изображений.

Обучение модели на подготовленных данных.

Сохранение обученной модели и class_indices в файлы.

3. utils.py:

Определение функций для загрузки и предобработки изображений, декодирования предсказаний модели и вывода результатов.

4. `predict.py`:

Загрузка обученной модели и `class_indices`.

Загрузка и предобработка изображения автомобиля.

Предсказание модели.

Вывод результата в формате: "С вероятностью ...% на картинке ... (марка машины)".

Использованные функции

Название	Краткое описание
<code>cv2.imread()</code>	Загрузка изображения
<code>cv2.resize()</code>	Изменение размера изображения
<code>image.load_img()</code>	Загрузка изображения с помощью Keras
<code>image.img_to_array()</code>	Преобразование изображения в массив NumPy
<code>preprocess_input()</code>	Предобработка изображения для ResNet50
<code>model.predict()</code>	Предсказание модели
<code>np.argmax()</code>	Нахождение индекса максимального значения в массиве

Пример работы программы



С вероятностью 52.25% на картинке BMW



С вероятностью 45.92% на картинке Mercedes

Текст программы

`model.py`:

```
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.layers import Dense, Flatten, Dropout
from tensorflow.keras.models import Model
```

```
def create_model(num_classes, image_height, image_width):
```

```

# Загрузка ResNet50 без верхних слоёв
base_model = ResNet50(weights='imagenet', include_top=False,
input_shape=(image_height, image_width, 3))

# Замораживание весов базовой модели
base_model.trainable = False

# Добавление полносвязных слоёв
x = base_model.output
x = Flatten()(x)
x = Dense(128, activation='relu')(x)
x = Dropout(0.5)(x)
predictions = Dense(num_classes, activation='softmax')(x)

# Создание модели
model = Model(inputs=base_model.input, outputs=predictions)
return model
train.py:
import os
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from model import create_model

# Параметры модели
num_classes = 5
image_height = 224
image_width = 224
batch_size = 32
epochs = 30

train_data_dir = "data/train"
validation_data_dir = "data/validation"

# Создание модели
model = create_model(num_classes, image_height, image_width)
model.compile(optimizer=Adam(learning_rate=0.001),
loss='categorical_crossentropy', metrics=['accuracy'])

# Генераторы данных
train_datagen = ImageDataGenerator(
    rescale=1./255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True)

val_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    train_data_dir,

```

```

    target_size=(image_height, image_width),
    batch_size=batch_size,
    class_mode='categorical')

validation_generator = val_datagen.flow_from_directory(
    validation_data_dir,
    target_size=(image_height, image_width),
    batch_size=batch_size,
    class_mode='categorical')

# Обучение модели
model.fit(
    train_generator,
    epochs=epochs,
    validation_data=validation_generator)

# Сохранение модели
model.save('car_brand_model.h5')

# Сохранение class_indices в файл
import json
class_indices = {v: k for k, v in train_generator.class_indices.items()}
print(class_indices)
with open('class_indices.json', 'w') as f:
    json.dump(class_indices, f)

for layer in model.layers[-10:]:
    layer.trainable = True

model.compile(optimizer=Adam(learning_rate=0.0001),
    loss='categorical_crossentropy', metrics=['accuracy'])

model.fit(
    train_generator,
    epochs=30,
    validation_data=validation_generator)

model.save('car_brand_model_finetuned.h5')
utils.py:
import numpy as np
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.resnet50 import preprocess_input

def load_and_preprocess_image(image_path):
    img = image.load_img(image_path, target_size=(224, 224))
    x = image.img_to_array(img)
    x = np.expand_dims(x, axis=0)
    x = preprocess_input(x)
    return x

```

```

def decode_prediction(prediction, class_indices):
    predicted_class_index = np.argmax(prediction)
    predicted_class = class_indices[predicted_class_index]
    return predicted_class

def print_prediction(prediction, class_indices):
    predicted_class_index = np.argmax(prediction)
    predicted_class = class_indices[predicted_class_index]
    probability = np.max(prediction) * 100
    print(f"С вероятностью {probability:.2f}% на картинке
{predicted_class}")
predict.py:
from tensorflow.keras.models import load_model

from utils import load_and_preprocess_image, decode_prediction,
print_prediction
import json

model = load_model('car_brand_model.h5')

with open('class_indices.json', 'r') as f:
    class_indices_loaded = json.load(f)
class_indices = {int(k): v for k, v in class_indices_loaded.items()}
#print("Class indices:", class_indices)

image_path = "test_image8.jpg"
image = load_and_preprocess_image(image_path)

prediction = model.predict(image)
print("Prediction:", prediction)
print_prediction(prediction, class_indices)

```

2.24. Детектирование отпечатков пальцев. TensorFlow

В программе используются TensorFlow, Keras, Scikit-learn. TensorFlow — открытая программная библиотека для машинного обучения, разработанная компанией Google для решения задач построения и тренировки нейронной сети с целью автоматического нахождения и классификации образов, достигая качества человеческого восприятия. В своем проекте благодаря данной нейронной сети мы можем классифицировать изображения отпечатков человеческих пальцев.

Keras является спецификацией в API Tensorflow. `tf.keras` является реализацией TensorFlow спецификации Keras API. Это высокоуровневый API для построения и обучения моделей включающий первоклассную поддержку для TensorFlow - специфичной функциональности, такой как `eager execution`, конвейеры `tf.data`, и `Estimators`. `tf.keras` делает использование TensorFlow проще не жертвуя при этом гибкостью и производительностью.

Scikit-learn — библиотека, предназначенная для машинного обучения, написанная на языке программирования Python и распространяемая в виде свободного программного обеспечения.

Алгоритм работы

1. Загрузка данных

Функция `load_data(data_dir)` отвечает за загрузку и предобработку изображений из указанного каталога:

Инициализируются списки для хранения изображений (`images`) и меток (`labels`). Для каждой папки внутри `data_dir`, которая представляет собой метку (`label`), считываются все изображения. Каждое изображение считывается с использованием библиотеки OpenCV (`cv2.imread`). Изображение изменяется до размера 100x100 пикселей (`cv2.resize`) и конвертируется из формата BGR в RGB (`cv2.cvtColor`). Изображения добавляются в список `images`, а соответствующая метка (имя папки) добавляется в список `labels`.

2. Подготовка данных для обучения и тестирования

Загруженные изображения и метки преобразуются в массивы NumPy.

Используется функция `train_test_split` из библиотеки `sklearn`, чтобы разделить данные на обучающую и тестовую выборки в соотношении 80/20.

3. Создание и компиляция модели

Создается модель нейронной сети с использованием Keras API:

Модель состоит из последовательности (Sequential) слоев:

Первый слой: сверточный слой (Conv2D) с 32 фильтрами размером 3x3 и функцией активации ReLU. На вход подаются изображения размером 100x100 с 3 цветовыми каналами.

Второй слой: слой подвыборки (MaxPooling2D) размером 2x2.

Третий слой: сверточный слой с 64 фильтрами размером 3x3 и функцией активации ReLU.

Четвертый слой: слой подвыборки размером 2x2.

Пятый слой: сверточный слой с 64 фильтрами размером 3x3 и функцией активации ReLU.

Шестой слой: слой выравнивания (Flatten) для преобразования многомерного выхода в одномерный вектор.

Седьмой слой: полносвязный слой (Dense) с 64 нейронами и функцией активации ReLU.

Восьмой слой: выходной полносвязный слой с одним нейроном и сигмоидной функцией активации для бинарной классификации. Модель компилируется с оптимизатором Adam и функцией потерь бинарной кросс-энтропии. Также указываются метрики для оценки точности (accuracy).

4. Обучение модели

Модель обучается на обучающей выборке:

Используется метод fit для запуска процесса обучения.

Обучение проводится в течение 10 эпох с размером пакета 64.

В качестве валидационных данных используются отложенные тестовые данные.

5. Оценка модели

Модель оценивается на тестовой выборке:

Используется метод `evaluate` для вычисления потерь и точности на тестовых данных.

Результаты точности выводятся в консоль.

Использованные функции

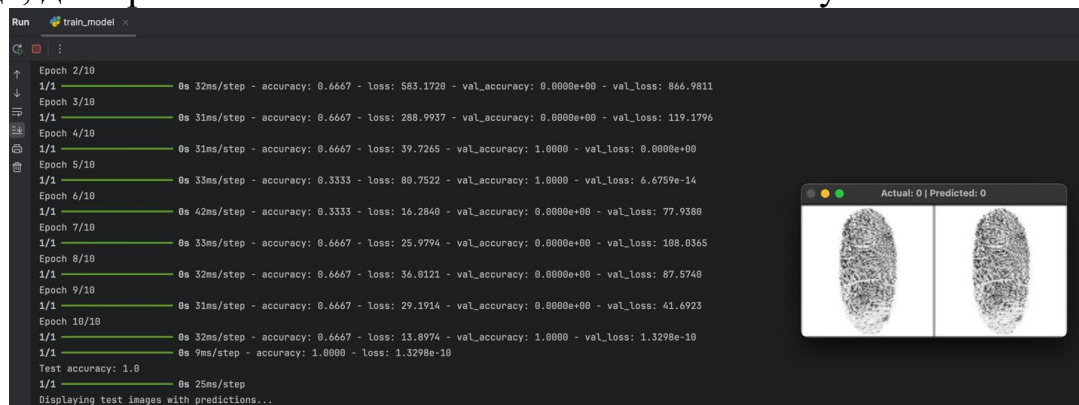
<code>tf.keras.Sequential</code>	Это класс, который позволяет создавать модель как последовательность слоев. В данном случае, слои добавляются последовательно, один за другим.
<code>layers.Conv2D</code>	Описание: Сверточный слой, который применяет фильтры (ядра свертки) к входным данным для выделения признаков. Аргументы: 32, 64: Количество фильтров. (3, 3): Размеры фильтров. <code>activation='relu'</code> : Использование функции активации ReLU (Rectified Linear Unit), которая вводит нелинейность в модель. <code>input_shape=(100, 100, 3)</code> : Форма входных данных, указывается только для первого слоя (100x100 пикселей, 3 канала цвета).
<code>layers.MaxPooling2D</code>	Описание: Слой подвыборки, который уменьшает размерность входных данных, сохраняя самые важные признаки. Аргументы: (2, 2): Размеры окна подвыборки.
<code>layers.Flatten</code>	Описание: Преобразует многомерный тензор в одномерный вектор, что необходимо для перехода от сверточных слоев к полносвязанным. Аргументы: Нет.
<code>layers.Dense</code>	Описание: Полносвязный слой, который соединяет каждый входной нейрон с каждым выходным нейроном. Аргументы: 64, 1: Количество нейронов в слое. <code>activation='relu'</code> , <code>activation='sigmoid'</code> : Функции активации. ReLU используется для внутренних слоев, а сигмоида — для выходного слоя, так как задача — бинарная классификация.

model.compile	<p>Описание: Метод для конфигурирования процесса обучения модели.</p> <p>Аргументы:</p> <p>optimizer='adam': Оптимизатор Adam, который адаптивно изменяет скорость обучения.</p> <p>loss='binary_crossentropy': Функция потерь для бинарной классификации.</p> <p>metrics=['accuracy']: Метрика для оценки точности модели.</p>
model.fit	<p>Описание: Метод для обучения модели на тренировочных данных.</p> <p>Аргументы:</p> <p>train_images, train_labels: Обучающие данные и соответствующие метки.</p> <p>epochs=10: Количество эпох обучения (полных проходов по всему обучающему набору данных).</p> <p>batch_size=64: Размер пакета данных, используемого за один раз.</p> <p>validation_data=(test_images, test_labels): Валидационные данные для оценки качества модели на каждом этапе обучения.</p>
model.evaluate	<p>Описание: Метод для оценки модели на тестовых данных.</p> <p>Аргументы:</p> <p>test_images, test_labels: Тестовые данные и соответствующие метки.</p> <p>Возвращает: Потери и точность на тестовых данных, которые затем выводятся на экран.</p>

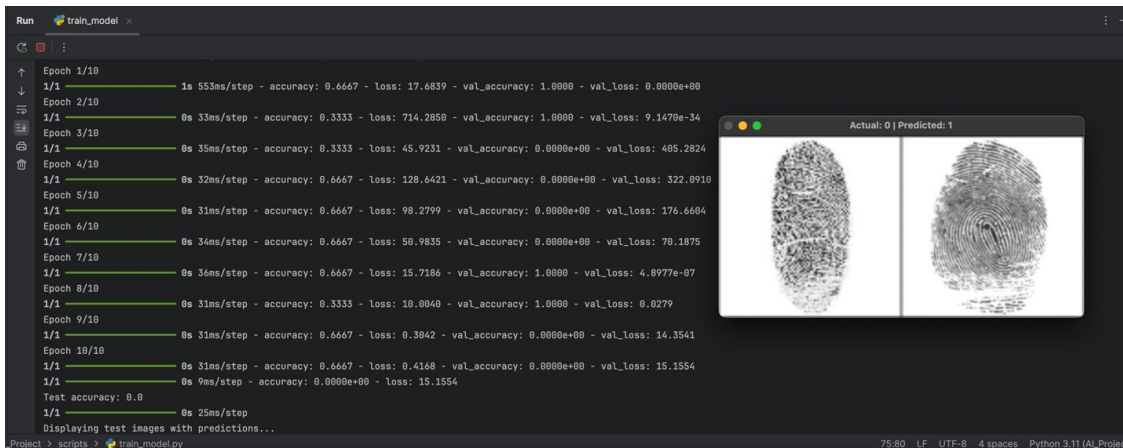
Пример применения

Рассмотрены два случая:

1) В папку 0 импортированы 2 изображения одинаковых отпечатков пальца, для сравнения и выявления их схожести. Получено:



2) В эту же папку импортировано уже два разных отпечатка пальца. Проведя аналогичную операцию получено:



Текст программы

```

import tensorflow as tf
from tensorflow.keras import layers, models
from sklearn.model_selection import train_test_split
import numpy as np
import cv2
import os

data_dir = "/Users/riddofftsss/PycharmProjects/AI_Project/data_dir"

def load_data(data_dir):
    images = []
    labels = []
    for label in os.listdir(data_dir):
        label_dir = os.path.join(data_dir, label)
        if not os.path.isdir(label_dir):
            continue
        for image_name in os.listdir(label_dir):
            image_path = os.path.join(label_dir, image_name)
            image = cv2.imread(image_path)
            if image is not None:
                image = cv2.resize(image, (100, 100))
                image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
                images.append(image)
                labels.append(int(label))
            else:
                print(f"Warning: {image_path} could not be read.")
    return np.array(images), np.array(labels)

def show_comparison_images(actual_images, predicted_images,
actual_labels, predicted_labels):
    for i in range(len(actual_images)): # Loop through all images
        actual_image = cv2.cvtColor(actual_images[i], cv2.COLOR_RGB2BGR)
        predicted_image = cv2.cvtColor(predicted_images[i],
cv2.COLOR_RGB2BGR)

```

```

        combined_image = np.hstack((actual_image, predicted_image))

        label_text = f"Actual: {actual_labels[i]} | Predicted:
{predicted_labels[i]}"
        cv2.imshow(label_text, combined_image)

        key = cv2.waitKey(0)
        if key == 27:
            print("Exiting image display.")
            break
    cv2.destroyAllWindows()

images, labels = load_data(data_dir)

train_images, test_images, train_labels, test_labels =
train_test_split(images, labels, test_size=0.2, random_state=42)

model = tf.keras.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(100, 100,
3)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(1, activation='sigmoid')
])

model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

model.fit(train_images, train_labels, epochs=10, batch_size=64,
          validation_data=(test_images, test_labels))

test_loss, test_acc = model.evaluate(test_images, test_labels)
print('Test accuracy:', test_acc)

predictions = model.predict(test_images)
predicted_labels = (predictions > 0.5).astype("int32").flatten()

print("Displaying test images with predictions...")
show_comparison_images(test_images, test_images, test_labels,
predicted_labels)

```

2.25. Распознавание объектов. Xception

Программа распознает объекты на фото. Она использует готовую нейросеть Xception [50], обученной по набору данных ImageNet. Набор данных ImageNet-1K содержит 1 281 167 обучающих изображений, 50 000 проверочных изображений и 100 000 тестовых изображений, разделенных на 1000 классов.

Библиотеки:

- Keras.application.xception – для заготовки модели нейросети
- Numpy – для численных преобразований над входными данными

Алгоритм работы

- Функция `make_predict()` вызывает функцию `keras.utils.load_img()`, которая загружает фото из указанного пути (если указано только имя фото, используется текущая директория, в которой находится файл с программой `main.py`) и преобразует его к нужному размеру 299x299 пикселей, возвращает фото в формате PIL (Python Imaging Library).
- Для получившегося загруженного фото вызывается функция `keras.utils.img_to_array()`, которая преобразует фото в трехмерный массив `numpy` (две размерности по количеству пикселей и данные о каждом пикселе).
- С помощью функции `numpy.expand_dims()` мы добавляем в массив дополнительную размерность на первое место, он становится четырехмерным.
- Функция `preprocess_input` преобразует полученный массив к виду, необходимому для `xception`. На следующем шаге вызываем у модели функцию `predict()`, которая осуществляет прогноз по изображению, преобразованному к нужному для нейросети формату.

- Вызываем функцию `decode_predictions()`, расшифровываем сделанный прогноз и выводим его на консоль функцией `print()`.
- В участке кода после функции `make_predict()` создается модель нейросети `xception`, с указанием того, что у модели будут веса, полученные в результате обучения по набору данных ImageNet. После чего создаются три строки с указанием пути к изображениям. Затем трижды вызывается функция `make_predict()` с созданной моделью для каждой из трех фото. В консоль выводятся не только сами предсказания, выводимые в моей функции `make_predict()`, но и некоторый промежуточный вывод функции `predict()` у модели `xception` (время, за которое предсказание было сделано), что и видно на скриншоте.

Используемые функции

Функции из библиотеки `keras`:

Название	Описание
<code>preprocess_input()</code>	Преобразует входные данные к нужному для <code>xception</code> виду
<code>decode_predictions()</code>	Расшифровывает предсказание нейросети
<code>load_img()</code>	Загружает изображение из файла
<code>predict()</code>	Формирует выходные прогнозы для входных выборок
<code>img_to_array()</code>	Преобразует изображение в массив

Функции из библиотеки `numpy`:

Название	Описание
<code>expand_dims()</code>	Расширяет размеры массива

Собственные функции:

Название	Описание
<code>make_predict()</code>	На основе полученного на вход изображения формирует прогноз о том, какой объект изображен на картинке

Скриншоты

На вход подавалось 3 изображения с африканским слоном, клубникой и клевером:


```
make_predict(model, img_path1)
make_predict(model, img_path2)
make_predict(model, img_path3)
```

2.26. Классификация статических картинок. TensorFlow

В работе используется TensorFlow. Он позволяет создавать различные архитектуры нейронных сетей, такие как сверточные нейронные сети (CNN), рекуррентные нейронные сети (RNN) и сети генеративного противостояния (GAN) [51]. TensorFlow предоставляет различные алгоритмы обучения нейронных сетей, такие как оптимизация градиента спуска, стохастический градиентный спуск и адаптивная оптимизация моментов. TensorFlow позволяет оценивать качество работы нейронных сетей с помощью различных метрик, таких как точность, F1-мера и ROC-кривая.

Архитектура нейронной сети

В работе была использована следующая архитектура нейронной сети:

- Сверточный слой: 32 фильтра размером 3x3, функция активации ReLU.
- Максимальный пул: 2x2.
- Сверточный слой: 64 фильтра размером 3x3, функция активации ReLU.
- Максимальный пул: 2x2.
- Полносвязный слой: 64 нейрона, функция активации ReLU.
- Выходной слой: 10 нейронов, функция активации softmax.

Обучение нейронной сети

Использовался общедоступный датасет изображений CIFAR-10. Датасет состоит из 60 000 цветных изображений размером 32x32 пикселя, разделенных на 10 классов: самолет, автомобиль, птица, кошка, олень, собака, лягушка, лошадь, корабль, грузовик. Нейронная сеть была обучена в течение 10 эпох. Оптимизатор Adam был использован для обучения

сети. Функция потерь “categorical_crossentropy” была использована для оценки качества работы сети. Метрика “accuracy” была использована для отслеживания точности классификации.

Анализ результатов




На тестовом наборе данных модель достигла точности 79,04%. ROC-кривая модели показывает, что модель имеет высокую способность отличать объекты от фона.

Используемые функции

TensorFlow	
load_model()	Загрузка сохраненной модели
load_data()	Загрузка данных датасета CIFAR-10
Sequential()	Создание последовательной модели
Conv2D()	Создание свёрточного слоя
MaxPooling2D()	Создание слоя подвыборки (макспулинга)
Flatten()	Создание слоя выпрямления
Dense()	Создание Полносвязного слоя
to_categorical()	Преобразование меток в формат one-hot encoding
compile()	Компиляция модели
fit()	Обучение модели
save()	Сохранение модели
predict()	Предсказание на основе обученной модели
NumPy	
numpy.array()	Преобразование изображения в массив NumPy
numpy.expand_dims()	Расширение размерности массива
numpy.argmax	Нахождение индекса максимального значения в массиве
PIL	
PIL.Image.open()	Открытие изображения
PIL.Image.Image.resize()	Изменение размера изображения
os	
os.listdir()	Получение списка файлов в директории
os.path.exists()	Проверка существования пути

Результаты работы

Картинка	Результат
----------	-----------

	<p>airplane</p>
	<p>cat</p>
	<p>frog</p>

Текст программы

Импорт библиотек:

```
import datetime
from tensorflow.keras.models import load_model
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Conv2D, MaxPooling2D
from tensorflow.keras.utils import to_categorical
import numpy as np
import os
from PIL import Image
```

Функция обучения модели:

```
def training():
    # Вывод сообщения о начале обучения
    print("Начало обучения...")

    # Загрузка и подготовка данных CIFAR-10
    (x_train, y_train), (x_test, y_test) = cifar10.load_data()

    # Нормализация данных: преобразование типа данных и деление каждого
    # пикселя на 255
    x_train = x_train.astype('float32') / 255.0
    x_test = x_test.astype('float32') / 255.0

    # Преобразование меток в формат one-hot encoding для использования в
    # категориальной кросс-энтропии
```

```

y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

# Создание модели сети: стек слоев Conv2D, MaxPooling2D, Flatten и Dense
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(64, activation='relu'),
    Dense(10, activation='softmax')
])

# Компиляция модели с использованием оптимизатора 'adam', функции потерь
# 'categorical_crossentropy' и метрики 'accuracy'
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])

# Обучение модели на данных, с проверкой качества на тестовом наборе
данных
model.fit(x_train, y_train, epochs=10, validation_data=(x_test, y_test))

# Сохранение модели в виде файла .h5, с именем содержащим текущую дату и
время
model_name = f'cifar10_{datetime.datetime.now().strftime("%d-%m-%Y-%H-
%M")}.h5'
model.save(f'models/{model_name}')

# Возвращение имени файла для возможного последующего использования
return model_name

```

Функция классификация изображения:

```

def classify_image(model_path):
    # Классификация изображения

    image_path = get_image()

    model = load_model(f'models/{model_path}')
    class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
'dog',
    frog', 'horse', 'ship', 'truck']

    print("Выполняется предобработка изображения...")
    image = Image.open(image_path).resize((32, 32))
    image = np.array(image).astype('float32') / 255.0
    image = np.expand_dims(image, axis=0)

    print("Выполняется классификация изображения...")

```

```

predictions = model.predict(image)
predicted_class = np.argmax(predictions)

return class_names[predicted_class]

```

Функция получения пути к изображению:

```

def get_image():
    # Получение пути изображения для классификации

    while True:
        image_path = input('Вставьте путь к файлу, который требуется
классифицировать\n>> ')
        if os.path.exists(image_path) and any([image_path.endswith('.jpg'
), image_path.endswith('.png'), image_path.endswith('.jpeg')]):
            return image_path
        print("По заданному пути не обнаружено файла или файл имеет
недопустимый формат")

```

Проверка наличия сохранённых моделей:

```

saved_models = os.listdir('models')
if len(saved_models) > 0:
    text = "У вас есть сохранённые модели, введите цифру нужной или
введите 0, чтобы заново обучить модель\n"
    for index, model in enumerate(saved_models):
        text += f"{index + 1}.{model}\n"
    text += "0. Обучить модель заново\n>> "

    model_id = input(text)
    if model_id == '0':
        model_name = training()
    else:
        model_name = saved_models[int(model_id) - 1]

else:
    model_name = training()

obj = classify_image(model_name)

print(f"Объект на изображении: {obj}")

```

2.27. Поиск и кластеризация объектов на фото с использованием конвертеров фото в вектор YOLO

Реализуется двухэтапный алгоритм кластеризации объектов при помощи нейросетей. Первый этап – поиск объекта на картинке, для этого

используется сеть `yolo5v`. Затем найденные объекты преобразуются в векторное представление и кластеризуются при помощи алгоритмов.

YOLOv5 (You Only Look Once version 5) — это современная нейросеть для обнаружения объектов. Нейросеть использует единственный сверточный слой для обнаружения объектов на изображении. Вместо разделения задачи на несколько этапов, как в других алгоритмах, YOLOv5 обрабатывает изображение целиком, что значительно ускоряет процесс. Она предсказывает границы объектов и классы объектов одновременно, используя несколько полей чувствительности разных размеров для обнаружения объектов различных размеров.

YOLOv5 состоит из нескольких основных компонентов:

- Слои свертки: Используются для извлечения признаков из изображений.
- Пуллинг слои: Уменьшают размерность изображения, сохраняя при этом важные признаки.
- Полносвязные слои: Используются для предсказания классов и координат объектов.

Обучение YOLOv5 происходит с использованием большого набора данных, содержащего изображения с метками объектов. Во время обучения нейросеть изучает, как корректно классифицировать и локализовать объекты на изображениях. Для обучения можно использовать различные оптимизаторы, такие как Adam или SGD, и различные функции потерь, например, потери перекрестной энтропии и потери локализации.

Применение `yolo` для поиска товарных единиц в кадре:

```

from ultralytics import YOLO
import cv2

# Загрузка модели
model = YOLO("D:/files/best.pt")

# Путь к изображению
image_path = r'D:/files/phot.jpg'

# Чтение изображения
image = cv2.imread(image_path)

# Предсказание с использованием модели
results = model.predict(source=image_path, show=False, save=False,
save_txt=False)

# Цвета для разных классов (предполагаем, что есть 8 классов)
colors = [(0, 255, 0), (0, 0, 255), (255, 0, 0), (255, 255, 0), (0, 255,
255), (255, 0, 255), (128, 128, 128), (128, 0, 0)]

# Проверка результатов и отрисовка прямоугольников
for result in results:
    boxes = result.boxes
    for box in boxes:
        # Получение координат и класса объекта
        x_min, y_min, x_max, y_max = box.xyxy[0]
        cls = model.names[int(box.cls[0])]
        color = colors[int(box.cls[0]) % len(colors)] # Выбор цвета на
основе класса

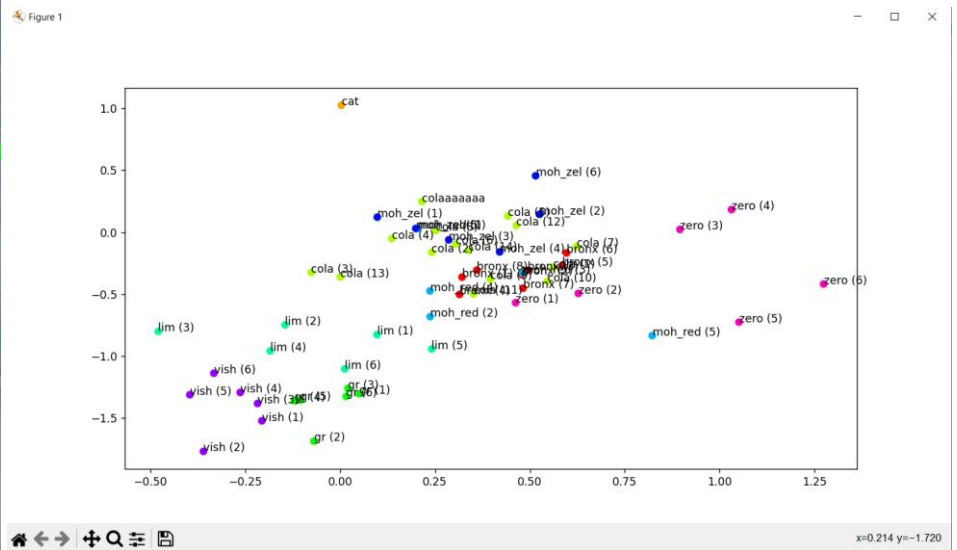
        # Отрисовка прямоугольника на изображении
        cv2.rectangle(image, (int(x_min), int(y_min)), (int(x_max),
int(y_max)), color, 2)
        cv2.putText(image, cls, (int(x_min), int(y_min - 10)),
cv2.FONT_HERSHEY_SIMPLEX, 0.9, color, 2)

# Отображение изображения с обнаруженными объектами

scale_percent = 50
width = int(image.shape[1] * scale_percent / 100)
height = int(image.shape[0] * scale_percent / 100)
dsize = (width, height) # resize image
image_rect = cv2.resize(image, dsize)

cv2.imshow('Detected Objects', image_rect)
cv2.waitKey(0)
cv2.destroyAllWindows()

```



На кадре заметно, что сеть нашла объекты и разделила на три типа тары: пластик, стекло и жестяные банки. Найдены не все объекты по причине измененных условий, относительно обученного датасета - высвеченные и затемненные области.

Затем вырезанные изображения найденных картинок мы преобразуем в векторное представление. Зачем это нужно. Это альтернатива классическим нейросетям, которые кодируют изображение и обучаются. Проблема подхода в длительном обучении и невозможности внести правки в датасет. Альтернатива – кодировать при помощи модели, вроде SWIN, а затем выделять классы алгоритмами кластеризации, вроде KMeans, AgglomerativeClustering, AffinityPropagation. Это требует больших вычислений на каждом шаге, но делает модель более гибкой. Используем AffinityPropagation. Возьмем папку вырезанных объектов при помощи предыдущей нейросети, преобразуем их в векторное представление и кластеризуем при помощи AffinityPropagation.

```

import swin_image_extractor as swin
import os
from PIL import Image
import numpy as np
from sklearn.cluster import AffinityPropagation, KMeans, Birch,
AgglomerativeClustering
import matplotlib.pyplot as plt
import cv2
import json
import json_loader as loader
import comparison

from PIL import Image
import torch
from torchvision import transforms
import timm
import cv2

# Загрузите предобученную модель Swin Transformer
model = timm.create_model('swin_base_patch4_window7_224',
pretrained=True)

model.eval() # Переведите модель в режим оценки

# Определите преобразования, которые будут применяться к изображениям
transform = transforms.Compose([
    transforms.Resize((224, 224)), # Измените размер изображения
    transforms.ToTensor(), # Преобразование в тензор
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224,
0.225])), # Нормализация
])

def transform_image(images):
    """
    преобразует изображения из массива images в вектора признаков
используя swin transformer
:param images: cv2 images array
:return: image features array
    """
    features_arr = []
    for image in images:
        img = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        im_pil = Image.fromarray(img)
        img_tensor = transform(im_pil).unsqueeze(0) # Добавьте размер
батча
        # Переведите тензор на устройство (GPU, если доступно)
        device = "cpu"
        # Expected one of cpu, cuda, ipu, xpu, mkldnn, opengl, opencl,

```

```

ideep, hip, ve, fpga, ort, xla, lazy, vulkan,
    # mps, meta, hpu, mtia,
    img_tensor = img_tensor.to(device)
    # Получите вектор признаков из модели
    with torch.no_grad():
        features = model(img_tensor)
    features_arr.append(features.squeeze().tolist())
return features_arr

# Установите путь к папке с изображениями
image_folder_path = "images/images_for_classification"
names = []
images = []

def show(affinity_labels, names, features):
    """
    Выводит на экран график
    :param affinity_labels:
    :param names:
    :param features:
    :return:
    """
    plt.figure(figsize=(10, 6))

    unique_labels = set(affinity_labels)
    cmap = plt.cm.get_cmap('hsv', len(unique_labels) + 1)
    col = cmap(affinity_labels)
    plt.scatter(features[:, 0], features[:, 1], c=col)

    for i, txt in enumerate(names):
        plt.annotate(txt, (features[i, 0], features[i, 1]))

    plt.show()
def gen(features, names):
    """
    Группирует вектора и выводит график их проекции
    :param features: features array
    :param names: names array
    :return: none
    """
    features = np.array(features)

    # Affinity Propagation
    affinity = AffinityPropagation(damping=0.5, max_iter=200,
convergence_iter=15)
    affinity_labels = affinity.fit_predict(features)
    print(f"affinity_labels {affinity_labels}")
    print(f"names {names}")

```

```

    show(affinity_labels, names, features)
    return affinity_labels
# Пройдитесь по всем изображениям в папке
for filename in os.listdir(image_folder_path):
    if filename.lower().endswith(('.png', '.jpg', '.jpeg', '.bmp',
'.gif')):
        image_path = os.path.join(image_folder_path, filename)
        with Image.open(image_path) as pil_img:
            open_cv_image = np.array(pil_img)
            # Convert RGB to BGR
            open_cv_image = open_cv_image[:, :, ::-1].copy()
            images.append(open_cv_image)
            names.append(os.path.splitext(filename)[0])

print(f"Запущено в преобразование {len(images)} изображений")
features_arr = swin.transform_image(images)
print("преобразование завершено")

# features_arr, names = loader.load_from_json()
features_arr2 = gen(features_arr, names)

```

На картинке можем увидеть несколько классов – названия лимонадов, которые по изображениям в кодированном виде классифицировались алгоритмом и разбились на группы. Для проверки устойчивости системы в выборку было заброшено изображение кота. На основе этого набора, мы можем определять принадлежность нового кадра к одному из классов, особенно если дополним выборку кадрами с изъянами или искажениями. Также, легко можно заменить один из классов или дополнить новым, за счет чего достигается гибкость системы. Подобные технологии используются сейчас в LLM.

Далее можно хранить преобразованный эталонный датасет в виде json или использовать векторную базу данных и сравнивать каждый новый кадр с эталонными, для классификации, организовав алгоритм сравнения по меткам классов.

2.28. Распознавание рукописных слов. TensorFlow

Распознавание рукописного текста позволяет преобразовывать аналоговые данные в цифровой формат, делая их доступными для дальнейшей обработки и анализа компьютерными системами. Системы распознавания оптического текста (OCR systems) играют важную роль в преобразовании рукописного текста в машинно-читаемый формат.

В работе рассмотрим процесс распознавания рукописных слов с использованием фреймворка TensorFlow. Используем набор данных IAM, а также реализуем функцию потерь CTC для оптимизации процесса распознавания текста почерка.

TensorFlow — это открытая программная библиотека, разработанная компанией Google, которая предоставляет мощные средства для создания и обучения глубоких нейронных сетей. Благодаря своей гибкости и эффективности, TensorFlow является одним из наиболее популярных инструментов в области машинного обучения и искусственного интеллекта.

Набор данных IAM (Iam Handwriting Database) является одним из наиболее широко используемых наборов данных для оценки производительности систем OCR. Он содержит изображения рукописных текстов и соответствующие текстовые строки, что делает его ценным ресурсом для научных исследований и индустрии.

Одним из ключевых методов в области распознавания текста почерка является функция потерь Connectionist Temporal Classification (CTC). Эта функция позволяет модели обучаться на данных с переменной длиной последовательности, что делает ее идеальным инструментом для работы с рукописными словами, где длина слов может варьироваться.

Алгоритм работы программы

1. Установка режима роста памяти для GPU, если доступно.
2. Загрузка и распаковка набора данных, если его нет на диске.
3. Предварительная обработка набора данных: чтение файла слов, создание списка изображений и соответствующих им меток, а также определение максимальной длины метки в символах.
4. Создание объекта конфигурации модели, содержащего информацию о словаре и максимальной длине метки.
5. Создание поставщика данных для обучения и валидации, включая предварительную обработку изображений и меток.
6. Аугментация данных обучения: изменение яркости, поворот, эрозия/дилатация, увеличение резкости изображений.
7. Создание и компиляция модели TensorFlow, включая определение оптимизатора, функции потерь и метрик.
8. Определение обратных вызовов (callbacks), таких как раннее прекращение, сохранение лучшей модели, запись логов в TensorBoard, и снижение скорости обучения при достижении плато.
9. Обучение модели на обучающих данных с использованием указанных обратных вызовов.
10. Сохранение результатов обучения (логов) и валидации (CSV файлы) в соответствующие файлы для дальнейшего анализа и визуализации.

Использованные функции

Библиотека	Функция	Назначение
tensorflow	tf.config.experimental.set_memory_growth	Настраивает рост памяти GPU во время выполнения. Это позволяет TensorFlow занимать только столько памяти GPU, сколько необходимо.
tensorflow	tf.config.experimental.list_physical_devices	Получает список всех физических устройств (в том числе GPU) доступных для

		TensorFlow.
tensorflow.keras.optimizers	Adam	Оптимизатор Adam, используемый для обучения модели. Adam — это метод стохастической оптимизации, эффективный для обучения нейронных сетей.
tensorflow.keras.callbacks	EarlyStopping	Обратный вызов для раннего прекращения обучения, если не происходит улучшения метрики.
tensorflow.keras.callbacks	ModelCheckpoint	Обратный вызов для сохранения модели в процессе обучения, наилучшей с точки зрения некоторой метрики.
tensorflow.keras.callbacks	ReduceLROnPlateau	Обратный вызов для динамического уменьшения скорости обучения, когда метрика перестает улучшаться.
tensorflow.keras.callbacks	TensorBoard	Обратный вызов для визуализации метрик и параметров модели в TensorBoard.
mltu.preprocessors	ImageReader	Препроцессор для чтения изображений.
mltu.transformers	ImageResizer	Преобразует размер изображения.
mltu.transformers	LabelIndexer	Преобразует текстовые метки в индексы.
mltu.augmentors	RandomErodeDilate	Аугментатор для случайного эрозии или дилатации изображения.
mltu.augmentors	RandomSharpen	Аугментатор для случайного увеличения резкости изображения.
mltu.tensorflow.dataProvider	DataProvider	Провайдер данных TensorFlow.
mltu.tensorflow.losses	CTCLoss	Функция потерь для Connectionist Temporal Classification (CTC).
mltu.tensorflow.callbacks	Model2onnx	Обратный вызов для преобразования модели в формат ONNX.

mltu.tensorflow.callbacks	TrainLogger	Обратный вызов для логирования процесса обучения.
model	train_model	Функция для создания и обучения модели.
tqdm	tqdm	Индикатор прогресса для циклов.

Результаты выполнения программы

Character Error Rate (CER) — это метрика, используемая для оценки качества распознавания текста, особенно в задачах оптического распознавания символов и распознавания рукописного текста. CER измеряет процент ошибок в распознанном тексте по сравнению с эталонным текстом. Он определяется как отношение числа вставок, удалений и замен символов к общему числу символов в эталонном тексте. В результатах выполнения программы CER близка к нулю, либо равна ему, что говорит о хорошем уровне анализа и распознавания рукописных слов.



Label: shop, Prediction: shop, CER: 0.0



Label: returned, Prediction: returned, CER: 0.0



Label: destructive, Prediction: destructure, CER: 0.09090909090909091

Текст программы

Исходный код обучения модели:

```
import os
import tarfile
from tqdm import tqdm
from io import BytesIO
from zipfile import ZipFile
from urllib.request import urlopen

import tensorflow as tf
from keras.callbacks import EarlyStopping, ModelCheckpoint,
ReduceLRonPlateau, TensorBoard

from mltu.tensorflow.losses import CTCloss
from mltu.preprocessors import ImageReader
from mltu.annotations.images import CVImage
from mltu.tensorflow.metrics import CWERMetric
from mltu.tensorflow.dataProvider import DataProvider
from mltu.tensorflow.callbacks import Model2onnx, TrainLogger
from mltu.transformers import ImageResizer, LabelIndexer, LabelPadding
from mltu.augmentors import RandomBrightness, RandomRotate,
RandomErodeDilate, RandomSharpen

from model import train_model
from configs import ModelConfigs

# Ensure TensorFlow GPU memory growth
def set_memory_growth():
    try:
        [tf.config.experimental.set_memory_growth(gpu, True) for gpu in
tf.config.experimental.list_physical_devices("GPU")]
    except:
        pass

# Download and unzip dataset
def download_dataset(url, extract_to="Datasets", chunk_size=1024*1024):
    http_response = urlopen(url)
    data = b""
    iterations = http_response.length // chunk_size + 1
    for _ in tqdm(range(iterations)):
        data += http_response.read(chunk_size)
    zipfile = ZipFile(BytesIO(data))
    zipfile.extractall(path=extract_to)

# Preprocess dataset
def preprocess_dataset(dataset_path):
    dataset, vocab, max_len = [], set(), 0
    words_file = os.path.join(dataset_path, "words.txt")
```

```

for line in tqdm(open(words_file, "r").readlines()):
    if line.startswith("#") or line.split()[1] == "err":
        continue

    folder1, folder2, file_name = line.split()[0][:3], "-
".join(line.split()[0].split("-")[:2]), line.split()[0] + ".png"
    label = line.split()[-1].rstrip("\n")
    rel_path = os.path.join(dataset_path, "words", folder1, folder2,
file_name)

    if os.path.exists(rel_path):
        dataset.append([rel_path, label])
        vocab.update(list(label))
        max_len = max(max_len, len(label))
    else:
        print(f"File not found: {rel_path}")

return dataset, "".join(vocab), max_len

# Create data provider
def create_data_provider(dataset, vocab, max_len, batch_size, width,
height):
    data_provider = DataProvider(
        dataset=dataset,
        skip_validation=True,
        batch_size=batch_size,
        data_preprocessors=[ImageReader(CVImage)],
        transformers=[
            ImageResizer(width, height, keep_aspect_ratio=False),
            LabelIndexer(vocab),
            LabelPadding(max_word_length=max_len,
padding_value=len(vocab)),
        ],
    )
    return data_provider

# Augment training data
def augment_training_data(train_data_provider):
    train_data_provider.augmentors = [
        RandomBrightness(),
        RandomErodeDilate(),
        RandomSharpen(),
        RandomRotate(angle=10),
    ]

# Create and compile TensorFlow model
def create_and_compile_model(input_dim, output_dim, learning_rate):
    model = train_model(input_dim=input_dim, output_dim=output_dim)

```

```

model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=learning_rate),
    loss=CTCloss(),
    metrics=[CWERMetric(padding_token=output_dim)],
)
model.summary(line_length=110)
return model

# Define callbacks
def define_callbacks(model_path):
    earlystopper = EarlyStopping(monitor="val_CER", patience=20,
verbose=1)
    checkpoint = ModelCheckpoint(f"{model_path}/model.h5",
monitor="val_CER", verbose=1, save_best_only=True, mode="min")
    trainLogger = TrainLogger(model_path)
    tb_callback = TensorBoard(f"{model_path}/logs", update_freq=1)
    reduceLRonPlat = ReduceLRonPlateau(monitor="val_CER", factor=0.9,
min_delta=1e-10, patience=10, verbose=1, mode="auto")
    model2onnx = Model2onnx(f"{model_path}/model.h5")
    return [earlystopper, checkpoint, trainLogger, reduceLRonPlat,
tb_callback, model2onnx]

# Train the model
def train_model(model, train_data_provider, val_data_provider, epochs,
callbacks, workers):
    model.fit(
        train_data_provider,
        validation_data=val_data_provider,
        epochs=epochs,
        callbacks=callbacks,
        workers=workers
    )

# Save training and validation data to CSV files
def save_csv(train_data_provider, val_data_provider, model_path):
    try:
        train_data_provider.to_csv(os.path.join(model_path, "train.csv"))
        val_data_provider.to_csv(os.path.join(model_path, "val.csv"))
    except Exception as e:
        print("An error occurred while saving CSV files:", e)

def main():
    set_memory_growth()

    # Constants
    DATASET_URL = "https://git.io/J0fjL"
    DATASET_PATH = os.path.join("Datasets", "IAM_Words")

    # Download and preprocess dataset

```

```

if not os.path.exists(DATASET_PATH):
    download_dataset(DATASET_URL, extract_to="Datasets")
    file = tarfile.open(os.path.join(DATASET_PATH, "words.tgz"))
    file.extractall(os.path.join(DATASET_PATH, "words"))

dataset, vocab, max_len = preprocess_dataset(DATASET_PATH)
configs = ModelConfigs()
configs.vocab = vocab
configs.max_text_length = max_len
configs.save()

train_data_provider, val_data_provider =
create_data_provider(dataset, vocab, max_len, configs.batch_size,
configs.width, configs.height).split(split=0.9)

augment_training_data(train_data_provider)

model = create_and_compile_model((configs.height, configs.width, 3),
len(vocab), configs.learning_rate)

callbacks = define_callbacks(configs.model_path)

train_model(model, train_data_provider, val_data_provider,
configs.train_epochs, callbacks, configs.train_workers)

save_csv(train_data_provider, val_data_provider, configs.model_path)

if __name__ == "__main__":
    main()

```

Исходный код распознавания рукописного текста изображений с использованием обученной модели:

```

import cv2
import numpy as np
import pandas as pd
from tqdm import tqdm
import os
from mltu.inferenceModel import OnnxInferenceModel
from mltu.utils.text_utils import ctc_decoder, get_cer
from mltu.configs import BaseModelConfigs

class ImageToWordModel(OnnxInferenceModel):
    def __init__(self, char_list, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.char_list = char_list
    def predict(self, image):
        image = cv2.resize(image, self.input_shapes[0][1:3][::-1])
        image_pred = np.expand_dims(image, axis=0).astype(np.float32)

```

```

        preds = self.model.run(self.output_names, {self.input_names[0]:
image_pred})[0]
        text = ctc_decoder(preds, self.char_list)[0]
        return text
def show_image(image):
    cv2.imshow("Image", image)
    cv2.waitKey(0)

if __name__ == "__main__":
    configs =
BaseModelConfigs.load("Models/03_handwriting_recognition/202301111911/con
figs.yaml")
    model = ImageToWordModel(model_path=configs.model_path,
char_list=configs.vocab)

    csv_file = "Models/03_handwriting_recognition/202301111911/val.csv"
    accum_cer = []
    with open(csv_file, 'r') as file:
        for line in tqdm(file.readlines()):
            image_path, label = line.strip().split(',')
            image = cv2.imread(os.path.join(image_path))
            prediction_text = model.predict(image)
            cer = get_cer(prediction_text, label)
            print(f"Image: {image_path}, Label: {label}, Prediction:
{prediction_text}, CER: {cer}")
            accum_cer.append(cer)
            show_image(cv2.resize(image, (image.shape[1] * 4,
image.shape[0] * 4)))

    print(f"Average CER: {np.average(accum_cer)}")
    cv2.destroyAllWindows()

```

2.29. Сегментация изображений и прозрачный фон. DeepLabV3

В программе происходит выделение человека на изображении с помощью обученной модели DeepLabV3 и преобразование фона изображения в прозрачный [52, 53].

О модели семантической сегментации DeepLabV3

В работе используется уже обученная модель DeepLabV3 — это одна из самых передовых моделей семантической сегментации, разработанная Google Brain. DeepLabV3 уже применяется в следующих областях решения задач: сегментация объектов на изображениях, обнаружение

объектов, семантическая сегментация видео, сегментация медицинских изображений, улучшение качества изображений. Ее архитектура специально разработана для решения таких задач как выделение объектов разного размера на изображениях. Модель обучена на большом наборе данных, что позволяет ей эффективно выполнять поставленную задачу и не думать разработчику о необходимости обучения модели.

Модель загружается с архитектурой ResNet101, состоящей из 101 слоя, основная идея которой использование остаточных соединений для эффективного обучения глубоких нейронных сетей. Также задается использование предобученных весов, которые были обучены на датасете COCO которая содержит более 200 тысяч изображений и более 80 классов объектов и включает в себя извлечение признаков (модель обучена распознавать и выделять ключевые объекты на изображениях) и саму сегментацию, которая реализуется следующим образом: для каждого объекта создается маска, которая точно определяет его контуры. При загрузке модель переводится в режим оценки, то есть в режим для предсказаний. Это важно указать, так как поведение некоторых слоев зависит от установленного режима.

Описание программы

В проекте используются следующие библиотеки и модули:

- torchvision.models (для использования модели DeepLabV3)
- PIL.Image (модуль для работы с изображениями)
- matplotlib.pyplot (для отображения изображений)
- torch (импортирует библиотеку PyTorch для работы с тензорами и нейронными сетями)
- numpy (для работы с массивами)
- cv2 (OpenCV для обработки изображений)
- torchvision.transforms(для преобразования изображений)

Помимо импорта библиотек всю последовательность действий можно определить в три этапа: инициализация использования модели и запуск сегментации, сегментация (`segment(...)`), декодирование сегментированной карты (`decode_segmap(...)`). Основная логика программы реализована внутри функции `segment`:

- Исходное изображение загружается с помощью PIL.
- Применяются необходимые преобразования (изменение размера, приведение к тензору, нормализация) для подготовки изображения к подаче на вход модели.
- Изображение подается на вход модели, и получается результат сегментации.
- Результат сегментации преобразуется из индексов классов в цветное изображение с помощью функции `decode_segmap`.
- Результат отображается с использованием библиотеки Matplotlib.

Используемые функции

Название функции	Описание
<code>np.array(...)</code>	создание массива
<code>np.zeros_like(image).astype(np.uint8)</code>	создание нового массива с той же формой и типом данных что и <code>image</code> и изменение типа данных массива на 8-битное целое число без знака
<code>rgb = np.stack([r, g, b], axis=2)</code>	объединяет последовательность массивов вдоль новой оси, что превращает двумерные массивы в трехмерных массив, создавая RGB-изображение
<code>foreground = cv2.imread(source)</code>	чтение изображение из файла, путь к которому лежит в <code>source</code>
<code>foreground = cv2.cvtColor(foreground, cv2.COLOR_BGR2RGB)</code>	преобразование цветового пространства изображения, где <code>foreground</code> – изображение, которое нужно преобразовать, а <code>cv2.COLOR_BGR2RGB</code> – флаг который указывает, что нужно преобразовать изображение из BGR в RGB
<code>foreground = cv2.resize(foreground, (r.shape[1], r.shape[0]))</code>	Изменение размера изображения, где <code>foreground</code> – изображение, которое нужно изменить, а второй параметр – кортеж, который задает новые размеры изображения

<code>background = 255 * np.ones_like(rgb).astype(np.uint8)</code>	Умножение каждого элемента созданного и преобразованного массива на 255 и присвоение его переменной <code>background</code>
<code>th, alpha = cv2.threshold(np.array(rgb), 0, 255, cv2.THRESH_BINARY)</code>	Бинаризация изображения по порогу (массив пикселей изображения, пороговое значение, максимальное значение, метод бинаризации а именно <code>cv2.THRESH_BINARY</code> , который означает, что все значения выше порога будут установлены в максимальное значение, а все остальное – в минимальное
<code>alpha = cv2.GaussianBlur(alpha, (7, 7), 0)</code>	Применение размытия по Гауссу к изображению
<code>foreground = cv2.multiply(alpha, foreground)</code>	выполняет поэлементное умножение двух массивов (изображений)
<code>background = cv2.multiply(1.0 - alpha, background)</code>	в данном случае выполняет поэлементное умножение двух массивов (изображений) с инвертированным альфа-каналом, где прозрачные области переднего плана становятся непрозрачными для фона и наоборот
<code>out_image = cv2.add(foreground, background)</code>	выполняет поэлементное сложение двух изображений
<code>img = Image.open(path)</code>	загружает изображение с указанного пути
<code>plt.imshow(img)</code>	отображение изображения <code>img</code>
<code>plt.axis('off')</code>	отключает отображение осей вокруг изображения
<code>plt.show()</code>	отображает окно с изображением
<code>trf = T.Compose([T.Resize(450), T.ToTensor(), T.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])])</code>	подготавливает входные изображения для DeerLabV3, а именно объединяет несколько преобразований: изменение размера изображения, преобразование изображение в тензор, нормализация тензор
<code>inp = trf(img).unsqueeze(0).to(dev)</code>	Применяет последовательность преобразований описанных выше, добавляет новое измерение к тензору и перемещает тензор на указанное устройство <code>dev</code> , но в определении функции <code>dev=cuda</code> , то есть тензор переместится на GPU

<code>out = net.to(dev)(inp)['out']</code>	Выполняет предсказание с использованием DeepLabV3 (конкретнее: перемещает модель на dev, выполняет прямой проход через модель с входными данными, где входные данные – это тензор изображения, подготовленный и перемещенный на устройство, и извлекает предсказания из выходного словаря модели)
<code>om = torch.argmax(out.squeeze(), dim=0).detach().cpu().numpy()</code>	<ol style="list-style-type: none"> 1. удаление размерности батча 2. выбор предсказанного класса для каждого пикселя 3. отсоединение от графа вычислений 4. перемещение на сру 5. преобразование в массив NumPy
<code>rgb = decode_segmap(om, path)</code>	реализованная в коде функция, которая преобразует индексы классов в цветное изображение
<code>dlab = models.segmentation.deeplabv3_resnet101(pretrained=True).eval()</code>	загружает предварительно обученную модель DeepLabV3 с архитектурой ResNet101 для сегментации изображений в режиме предсказаний
<code>segment(dlab, '/content/portrait.jpg', show_orig=True)</code>	Функция реализованная в коде, которая применяет модель сегментации к изображению и визуализирует результат

Результаты



Для тестирования нейросети можно воспользоваться Google Colaboratory. Там же можно настроить подключение к серверному ускорителю Python 3 на базе Google Compute Engine (GPU) для ускорения вычислений и повышения производительности, так как задачи сегментации изображений ресурсоемкие и требуют значительную вычислительную мощность.

Текст программы

```
from torchvision import models
from PIL import Image
import matplotlib.pyplot as plt
import torch
import numpy as np
import cv2

import torchvision.transforms as T

def decode_segmap(image, source, nc=21):
    # Цвета для каждого класса
    label_colors = np.array([(0, 0, 0),
                             (128, 0, 0), (0, 128, 0), (128, 128, 0), (0,
0, 128), (128, 0, 128),
                             (0, 128, 128), (128, 128, 128), (64, 0, 0),
(192, 0, 0), (64, 128, 0),
                             (192, 128, 0), (64, 0, 128), (192, 0, 128),
(64, 128, 128), (192, 128, 128),
                             (0, 64, 0), (128, 64, 0), (0, 192, 0), (128,
192, 0), (0, 64, 128)])

    # Создание массивов для каналов RGB
    r = np.zeros_like(image).astype(np.uint8)
    g = np.zeros_like(image).astype(np.uint8)
    b = np.zeros_like(image).astype(np.uint8)

    # Преобразование индексов классов в цветные пиксели
    for l in range(0, nc):
        idx = image == l
        r[idx] = label_colors[l, 0]
        g[idx] = label_colors[l, 1]
        b[idx] = label_colors[l, 2]

    # Создание цветного изображения из массивов R, G, B
    rgb = np.stack([r, g, b], axis=2)
```

```

# Загрузка переднего плана изображения и преобразование его к нужным
размерам
foreground = cv2.imread(source)
foreground = cv2.cvtColor(foreground, cv2.COLOR_BGR2RGB)
foreground = cv2.resize(foreground, (r.shape[1], r.shape[0]))

# Создание фона
background = 255 * np.ones_like(rgb).astype(np.uint8)

# Преобразование изображений к типу float для дальнейших вычислений
foreground = foreground.astype(float)
background = background.astype(float)

# Бинаризация цветного изображения, чтобы получить маску переднего
плана
th, alpha = cv2.threshold(np.array(rgb), 0, 255, cv2.THRESH_BINARY)
# Размытие маски для создания плавных границ
alpha = cv2.GaussianBlur(alpha, (7, 7), 0)
# Нормализация значений альфа в диапазоне [0, 1]
alpha = alpha.astype(float) / 255

# Умножение переднего плана на альфа и фона на (1 - альфа)
foreground = cv2.multiply(alpha, foreground)
background = cv2.multiply(1.0 - alpha, background)
# Сложение переднего плана и фона для получения окончательного
изображения
out_image = cv2.add(foreground, background)

# Нормализация значений пикселей к диапазону [0, 1] и возврат
окончательного изображения
return out_image / 255

def segment(net, path, show_orig=True, dev='cuda'):
    # Открытие изображения
    img = Image.open(path)
    # Отображение исходного изображения, если параметр show_orig
установлен в True
    if show_orig:
        plt.imshow(img)
        plt.axis('off')
        plt.show()

    # Преобразования изображения перед подачей его на вход модели
    trf = T.Compose([T.Resize(450),
                    T.ToTensor(),
                    T.Normalize(mean=[0.485, 0.456, 0.406],
                                std=[0.229, 0.224, 0.225])])
    # Применение преобразований к изображению
    inp = trf(img).unsqueeze(0).to(dev)

```

```

# Получение выхода модели
out = net.to(dev)(inp)['out']
# Получение индекса класса с максимальной вероятностью для каждого
# пикселя
om = torch.argmax(out.squeeze(), dim=0).detach().cpu().numpy()

# Преобразование индексов классов в цветное изображение с помощью
# заданной функции
rgb = decode_segmap(om, path)

# Отображение результата сегментации
plt.imshow(rgb)
plt.axis('off')
plt.show()

# Загрузка предварительно обученной модели DeepLabV3
dlab = models.segmentation.deeplabv3_resnet101(pretrained=True).eval()

# Применение модели к изображению и отображение результата
segment(dlab, '/content/portrait1.jpg', show_orig=True)

```

2.30. Определения состояния неба по фотографии. TensorFlow

Программа реализована с использованием библиотеки для глубокого обучения TensorFlow и API Keras. Keras — это высокоуровневый интерфейс для создания и обучения нейронных сетей, который работает поверх библиотеки TensorFlow.

Алгоритм работы

1. Подготовка данных

- Сбор данных: собираются изображения различных типов состояния неба. Изображения могут быть взяты из открытых датасетов или сделаны вручную.
- Организация данных: изображения сортируются по папкам в зависимости от их класса (например, cloudy, sunset, sunny).
- Изображения сортируются по папкам в зависимости от их класса (например, cloudy, rainy, sunny). Структура папок должна быть следующей:

```

data/
  train/
    cloudy/
      image1.jpg
      image2.jpg

```

```
...
rainy/
  image1.jpg
  image2.jpg
  ...
sunny/
  image1.jpg
  image2.jpg
  ...
test/
  cloudy/
    image1.jpg
    image2.jpg
    ...
  rainy/
    image1.jpg
    image2.jpg
    ...
  sunny/
    image1.jpg
    image2.jpg
    ...
```

- Аугментация данных: используется ImageDataGenerator из Keras для масштабирования изображений и применения аугментации (вращение, сдвиг, изменение яркости и т.д.), чтобы увеличить количество данных и улучшить обобщающую способность модели.

2. Создание модели

- Определение архитектуры модели: создается последовательная модель (Sequential) с использованием слоев свертки (Conv2D, извлекают особенности из изображений), пулинга (MaxPooling2D, уменьшают размерность данных и извлекают ключевые особенности), слоя выпрямления (Flatten, преобразует двумерные данные в одномерный массив) и полносвязных слоев (Dense, выполняют классификацию на основе извлеченных особенностей). Также используется Dropout для регуляризации модели и предотвращения переобучения.
- Компиляция модели: модель компилируется с использованием функции потерь categorical_crossentropy и оптимизатора adam.

3. Обучение модели

Модель обучается на обучающем наборе данных с использованием функции fit. Процесс обучения включает валидацию на валидационном наборе данных для мониторинга переобучения.

4. Оценка модели

Производительность модели оценивается на тестовом наборе данных с использованием функции `evaluate`.

5. Предсказание на новых данных

Модель используется для предсказания класса новых изображений с помощью функции `predict`. Изображение загружается и преобразуется в массив данных, подходящий для модели.

Используется библиотека `Matplotlib` для визуализации данных и результатов.

Используемые функции

Функция	Описание
<code>ImageDataGenerator()</code>	Подготовка и аугментация изображений для обучения модели.
<code>Sequential()</code>	Создание последовательной модели нейронной сети.
<code>Conv2D()</code>	Добавление слоя свертки в модель.
<code>MaxPooling2D()</code>	Добавление слоя пулинга в модель.
<code>Flatten()</code>	Преобразование двумерных данных в одномерный массив для передачи в полносвязный слой.
<code>Dense()</code>	Добавление полносвязного слоя в модель.
<code>Dropout()</code>	Применение dropout для регуляризации модели и предотвращения переобучения.
<code>compile()</code>	Компиляция модели с выбранным оптимизатором и функцией потерь.
<code>fit()</code>	Обучение модели на обучающем наборе данных.
<code>evaluate()</code>	Оценка производительности модели на тестовом наборе данных.
<code>flow_from_directory()</code>	Загрузка данных из директории с поддержкой структуры каталогов, содержащей поддиректории для каждого класса.
<code>load_img()</code> <code>img_to_array()</code>	Загрузка изображений и преобразование их в массивы данных.
<code>predict()</code>	Использование обученной модели для предсказания на новых данных.

Пример работы программы



Predicted weather condition: sunny Predicted weather condition: cloudy

Текст программы

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,
Dropout
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.model_selection import train_test_split
import numpy as np
import matplotlib.pyplot as plt

datagen = ImageDataGenerator(rescale=1.0/255.0, validation_split=0.2)

train_data = datagen.flow_from_directory('data/train', target_size=(150,
150), batch_size=32, class_mode='categorical', subset='training')
val_data = datagen.flow_from_directory('data/train', target_size=(150,
150), batch_size=32, class_mode='categorical', subset='validation')
test_data = datagen.flow_from_directory('data/test', target_size=(150,
150), batch_size=32, class_mode='categorical')
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(512, activation='relu'),
    Dropout(0.5),
    Dense(train_data.num_classes, activation='softmax')
])

model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
model.summary()
history = model.fit(train_data, epochs=20, validation_data=val_data)
loss, accuracy = model.evaluate(test_data)
print(f'Test accuracy: {accuracy * 100:.2f}%')
```

```

from tensorflow.keras.preprocessing import image

def predict_weather(image_path):
    img = image.load_img(image_path, target_size=(150, 150))
    img_array = image.img_to_array(img) / 255.0
    img_array = np.expand_dims(img_array, axis=0)
    prediction = model.predict(img_array)
    class_idx = np.argmax(prediction, axis=1)[0]
    class_label = train_data.class_indices
    class_label = {v: k for k, v in class_label.items()}
    predicted_class = class_label[class_idx]
    return predicted_class

image_path = 'pic.jpg'
predicted_class = predict_weather(image_path)
print(f'Predicted weather condition: {predicted_class}')

```

2.31. Детальное описание статических изображений в архитектуре CNN+LSTM. TensorFlow

В данной работе представлена реализация проекта по созданию модели генерации подписей к изображениям с использованием комбинации сверточных нейронных сетей (CNNs) и долговременных краткосрочных сетей (LSTMs). Используются библиотеки: для глубокого обучения TensorFlow, для создания моделей нейронных сетей Keras, а для обработки данных и визуализации: Pandas, NumPy, Matplotlib, Seaborn.

Ход работы

1. Использование библиотек

- os - Библиотека для работы с операционной системой, взаимодействие с файловой системой и выполнение команд на уровне ОС.
- sys - Модуль для доступа к переменным и функциям, взаимодействующим с интерпретатором Python.
- tempfile - Библиотека для работы с временными файлами и директориями.
- urllib.request - Модуль для открытия и чтения URL.
- urllib.parse - Модуль для разбора URL.

- `urllib.error` - Модуль для обработки ошибок, возникающих при работе с URL.
- `zipfile` - Библиотека для работы с ZIP-архивами.
- `tarfile` - Библиотека для работы с tar-архивами.
- `shutil` - Модуль для выполнения высокоуровневых операций с файлами и директориями.
- `numpy` - Библиотека для работы с многомерными массивами и матрицами, включает математические функции.
- `pandas` - Библиотека для работы с табличными данными, предоставляет структуры данных и операции для манипуляции таблицами.
- `tensorflow` - Открытая библиотека для машинного обучения, используется для создания и обучения моделей глубокого обучения.
- `tqdm` - Библиотека для визуализации прогресса выполнения циклов.
- `tensorflow.keras.preprocessing.image` - Подмодуль Keras для работы с изображениями, включает функции для загрузки и предобработки изображений.
- `tensorflow.keras.preprocessing.text` - Подмодуль Keras для работы с текстовыми данными, включает функции для токенизации и последовательного кодирования текста.
- `tensorflow.keras.utils` - Подмодуль Keras, включает вспомогательные функции и классы.
- `tensorflow.keras.layers` - Подмодуль Keras, включает слои нейронных сетей.
- `tensorflow.keras.applications` - Подмодуль Keras, включает предобученные модели.
- `tensorflow.keras.optimizers` - Подмодуль Keras, включает оптимизаторы.

- `tensorflow.keras.callbacks` - Подмодуль Keras, включает функции обратного вызова для настройки обучения моделей.
- `warnings` - Библиотека для работы с предупреждениями в коде.
- `matplotlib.pyplot` - Библиотека для создания статических, анимационных и интерактивных визуализаций.
- `seaborn` - Библиотека для визуализации данных, основана на `matplotlib`, предоставляет высокоуровневый интерфейс для рисования статистических графиков.

2. Установка пакетов и библиотек

На примере импорта в среду сетевой разработки Google Collab приведем код, который обеспечит нашу модель всеми необходимыми данными:

```
import os
import sys
from tempfile import NamedTemporaryFile
from urllib.request import urlopen
from urllib.parse import unquote, urlparse
from urllib.error import HTTPError
from zipfile import ZipFile
import tarfile
import shutil

CHUNK_SIZE = 40960
DATA_SOURCE_MAPPING = 'flickr-image-dataset'

INPUT_PATH='/ input'
WORKING_PATH='/ working'

!umount / input/ 2> /dev/null
shutil.rmtree('/ input', ignore_errors=True)
os.makedirs(INPUT_PATH, 0o777, exist_ok=True)
os.makedirs(WORKING_PATH, 0o777, exist_ok=True)

try:
    os.symlink(INPUT_PATH, os.path.join("../", 'input'),
target_is_directory=True)
except FileExistsError:
    pass
try:
    os.symlink(WORKING_PATH, os.path.join("../", 'working'),
target_is_directory=True)
except FileExistsError:
```

```

pass

for data_source_mapping in DATA_SOURCE_MAPPING.split(','):
    directory, download_url_encoded = data_source_mapping.split(':')
    download_url = unquote(download_url_encoded)
    filename = urlparse(download_url).path
    destination_path = os.path.join(INPUT_PATH, directory)
    try:
        with urlopen(download_url) as fileres, NamedTemporaryFile() as
tfile:
            total_length = fileres.headers['content-length']
            print(f'Downloading {directory}, {total_length} bytes
compressed')
            dl = 0
            data = fileres.read(CHUNK_SIZE)
            while len(data) > 0:
                dl += len(data)
                tfile.write(data)
                done = int(50 * dl / int(total_length))
                sys.stdout.write(f"\r[{'=' * done}{' ' * (50-done)}] {dl}
bytes downloaded")
                sys.stdout.flush()
                data = fileres.read(CHUNK_SIZE)
            if filename.endswith('.zip'):
                with ZipFile(tfile) as zfile:
                    zfile.extractall(destination_path)
            else:
                with tarfile.open(tfile.name) as tarfile:
                    tarfile.extractall(destination_path)
            print(f'\nDownloaded and uncompressed: {directory}')
    except HTTPError as e:
        print(f'Failed to load (likely expired) {download_url} to path
{destination_path}')
        continue
    except OSError as e:
        print(f'Failed to load {download_url} to path
{destination_path}')
        continue
print('Импорт завершен.')

```

Здесь мы прошли следующие пункты:

Импорт необходимых библиотек

Настройка путей и параметров

Подготовка директории

Загрузка и распаковка датасетов

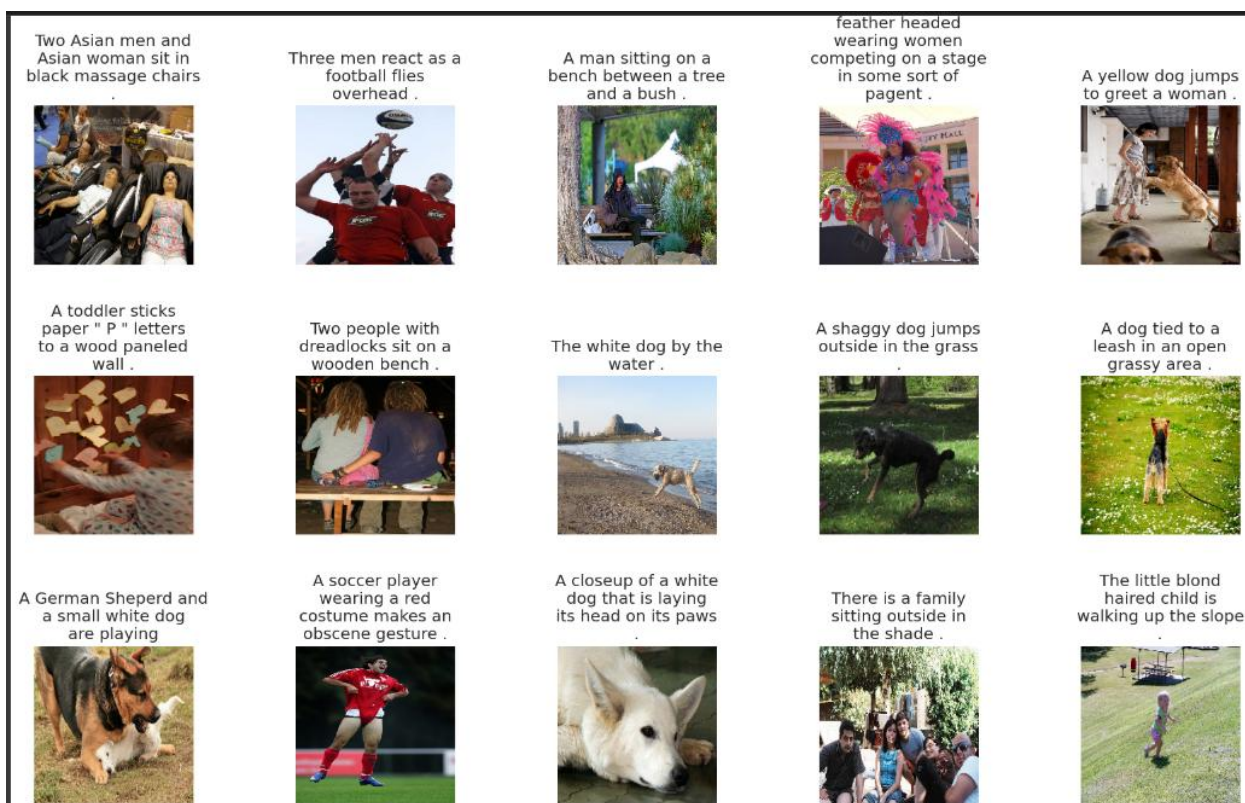
3. Набор данных

Датасет, используемый в этом проекте, называется Flickr8K. Он состоит из:

- > 8000 изображений
- Аннотаций (текстовых описаний) для каждого изображения

Набор данных Flickr 8k содержит 8092 изображения и до пяти подписей к каждому изображению. Изображения были отобраны из шести различных групп на Flickr, и, как правило, на них нет хорошо известных людей или мест, но они были отобраны вручную, чтобы отобразить различные сцены и ситуации. Такие аннотации к изображениям необходимы для дальнейшего прогресса в автоматическом описании изображений и углубленного понимания языка. Они позволяют определить новый критерий для локализации текстовых упоминаний объектов на изображении. Flickr8K является популярным набором данных для задач генерации описаний изображений и широко используется для обучения и оценки моделей в этой области.

Отображение некоторых изображений с их аннотациями в дата-сете `display_images(data.sample(15))`:



4. Описание реализации задачи

Создание подписей к изображениям — это процесс генерации текстового описания изображения. Для генерации подписей используется как обработка естественного языка, так и компьютерное зрение. Эта задача лежит на пересечении компьютерного зрения и обработки естественного языка. Большинство систем создания подписей к изображениям используют архитектуру «кодировщик-декодировщик», где входное изображение кодируется в промежуточное представление информации изображения, а затем декодируется в описательную текстовую последовательность.

Решение задачи в CNNs + RNNs (LSTMs):

- Для выполнения задачи создания подписей к изображениям потребуется две модели глубокого обучения, объединенные в одну для целей обучения:
- Сверточные нейронные сети (CNNs) извлекают признаки из изображения векторного размера, также известные как векторные представления (эмбеддинги). Размер этих эмбеддингов зависит от типа предварительно обученной сети, используемой для извлечения признаков.
- Долговременные краткосрочные памяти (LSTMs) используются для процесса генерации текста. Эмбеддинги изображения конкатенируются с эмбеддингами слов и передаются в LSTM для генерации следующего слова.

5. Предварительная обработка текста

Функция `text_preprocessing(data)` предназначена для предварительной обработки текста подписей к изображениям. Она выполняет следующие действия:

1. Приводит текст каждой подписи к нижнему регистру.
2. Удаляет все символы, не являющиеся буквами (кроме пробелов) из каждой подписи.
3. Заменяет последовательности пробелов в каждой подписи на одиночные пробелы.
4. Удаляет слова, состоящие из одного символа, из каждой подписи.

5. Добавляет токены "startseq" и "endseq" в начало и конец каждой подписи соответственно. Эти токены используются для обозначения начала и конца подписи и помогают модели понять структуру текста.

Далее в данной работе присутствует токенизатор, который выполняет следующие действия:

1. Обучение на тексте: Токенизатор обучается на текстовых данных с помощью метода `fit_on_texts()`. В данном случае он обучается на списках подписей (captions), извлекая уникальные слова и назначая каждому слову уникальный индекс.
2. Вычисление размера словаря: После обучения токенизатора вычисляется размер словаря как количество уникальных слов в тексте плюс один. Это делается с помощью выражения `len(tokenizer.word_index) + 1`, где `tokenizer.word_index` содержит словарь уникальных слов с их индексами.
3. Определение максимальной длины подписей: Токенизатор также вычисляет максимальную длину подписей, которая используется для последующей предобработки данных. Максимальная длина подписей вычисляется путем разбиения каждой подписи на слова и нахождения максимальной длины среди всех подписей.
4. Разделение данных на обучающий и тестовый наборы: Для подготовки данных к обучению модели изображения разделяются на обучающие и тестовые с использованием соотношения 85% к 15%. После этого данные фильтруются на основе этих списков изображений для создания обучающего и тестового наборов данных.
5. Сброс индекса: Индексы строк в данных переустанавливаются для облегчения работы с ними в дальнейшем.

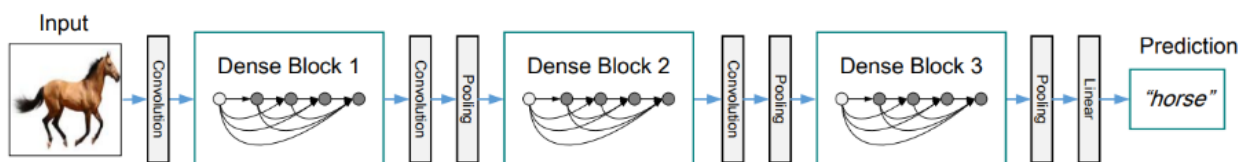


Работа токенизатора

6. Обработка признаков изображений

Далее происходит обработка признаков изображения, их импорт и сохранения для дальнейшей обработки.

1. Настройка архитектуры: Используется архитектура DenseNet201 для извлечения признаков из изображений. Это достигается путем создания модели f_e , которая является моделью, состоящей из всех слоев DenseNet201 за исключением последнего слоя.
2. Подготовка изображений: Каждое изображение из набора данных подгружается, изменяется до размера 224x224 пикселей (требование DenseNet201), нормализуется (приводится к диапазону от 0 до 1) и преобразуется в массив numpy. Этот процесс выполняется для каждого изображения в списке уникальных изображений из набора данных.
3. Извлечение признаков из изображений: После подготовки каждого изображения оно передается через модель f_e для извлечения признаков. Выходом из этого процесса является вектор признаков размером 1920, который представляет собой абстрактное представление содержимого изображения.
4. Сохранение признаков: Извлеченные признаки сохраняются в словаре `features`, где ключом является имя изображения, а значением — соответствующий вектор признаков. Этот словарь будет использоваться позже для обучения модели генерации подписей к изображениям.



Архитектура DenseNet

7. Генерация данных

Class `CustomDataGenerator` определен как класс-генератор. Этот генератор данных используется для подготовки данных для обучения модели генерации подписей к изображениям. Он генерирует данные пакетами, чтобы обеспечить эффективное использование ресурсов. Входными данными являются векторы признаков изображений и последовательности индексов слов для подписей. Эти данные подаются модели для обучения процесса генерации подписей для изображений. Во

время обучения модели предсказывается следующее слово в подписи на основе векторов признаков изображений и предыдущих слов в подписи.

8. Моделирование нейронной сети

Далее необходимо определить архитектуру модели сети для генерации подписей к изображениям. Изображения и текстовые данные подаются в модель как два входа: векторы признаков изображений и последовательности индексов слов для подписей.

Вначале векторы признаков изображений проходят через слой Dense, чтобы уменьшить размерность их представления до 256. Затем эти признаки решейпятся в трехмерный тензор с размерностью (batch_size, 1, 256), чтобы быть совместимыми с входом LSTM.

Последовательности индексов слов для подписей проходят через слой Embedding, который преобразует их в векторы фиксированной длины размером 256. Затем эти векторы объединяются с векторами признаков изображений и передаются в слой LSTM.

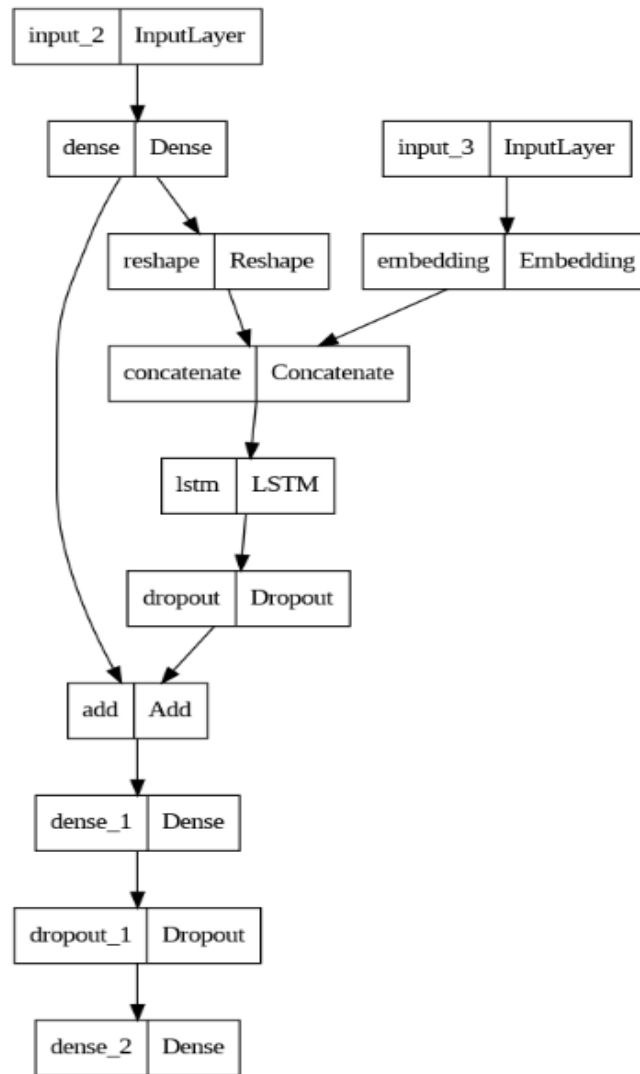
После слоя LSTM используется слой Dropout для регуляризации и предотвращения переобучения модели. Затем векторы признаков изображений добавляются к выходу LSTM. Это позволяет модели использовать как информацию о признаках изображений, так и контекст предыдущих слов для генерации следующего слова в подписи.

Далее применяется полносвязный слой с 128 нейронами и функцией активации ReLU, а затем еще один слой Dropout. На выходе используется полносвязный слой с функцией активации softmax для предсказания вероятности каждого слова в словаре.

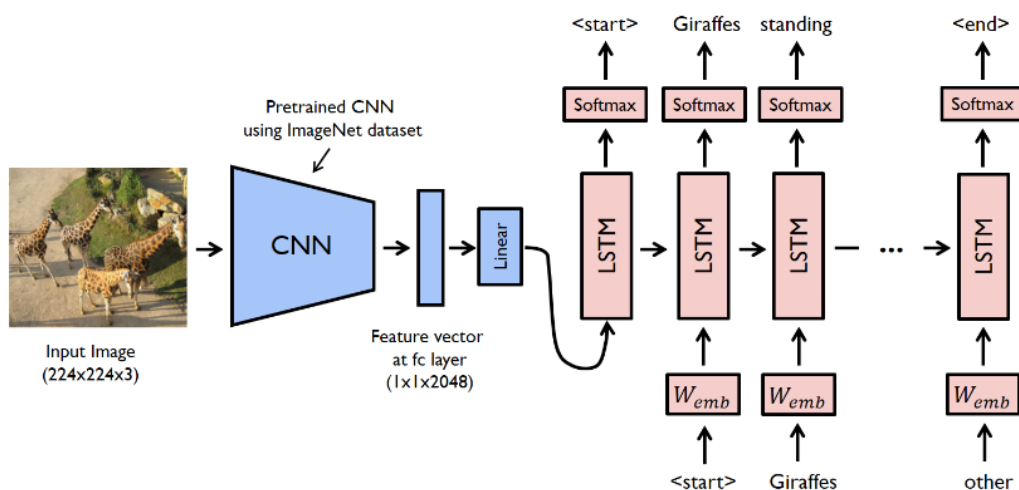
Модель компилируется с функцией потерь categorical_crossentropy и оптимизатором adam.

В результате получается модель, которая принимает на вход векторы признаков изображений и последовательности индексов слов для подписей, а на выходе выдает вероятности следующего слова в подписи.

`plot_model(caption_model)`



Архитектура построенной модели



Архитектура CNNs+LSTMs

В результате получаем следующую структуру модели:

```

Model: "model_1"
Layer (type)                Output Shape                Param #   Connected to
-----
input_2 (InputLayer)        [(None, 1920)]              0         []
dense (Dense)                (None, 256)                 491776    ['input_2[0][0]']
input_3 (InputLayer)        [(None, 34)]                0         []
reshape (Reshape)           (None, 1, 256)              0         ['dense[0][0]']
embedding (Embedding)        (None, 34, 256)             2172160   ['input_3[0][0]']
concatenate (Concatenate)    (None, 35, 256)             0         ['reshape[0][0]',
                                     'embedding[0][0]']
lstm (LSTM)                  (None, 256)                 525312    ['concatenate[0][0]']
dropout (Dropout)           (None, 256)                 0         ['lstm[0][0]']
add (Add)                    (None, 256)                 0         ['dropout[0][0]',
                                     'dense[0][0]']
dense_1 (Dense)              (None, 128)                 32896     ['add[0][0]']
dropout_1 (Dropout)          (None, 128)                 0         ['dense_1[0][0]']
dense_2 (Dense)              (None, 8485)                1094565   ['dropout_1[0][0]']
=====
Total params: 4316709 (16.47 MB)
Trainable params: 4316709 (16.47 MB)
Non-trainable params: 0 (0.00 Byte)

```

9. Обучение модели

Реализован процесс обучения модели для генерации подписей к изображениям. В начале создаются два генератора данных - `train_generator` и `validation_generator` - с помощью класса `CustomDataGenerator`, который

был определен ранее. Эти генераторы используются для загрузки данных в модель батчами во время обучения и валидации.

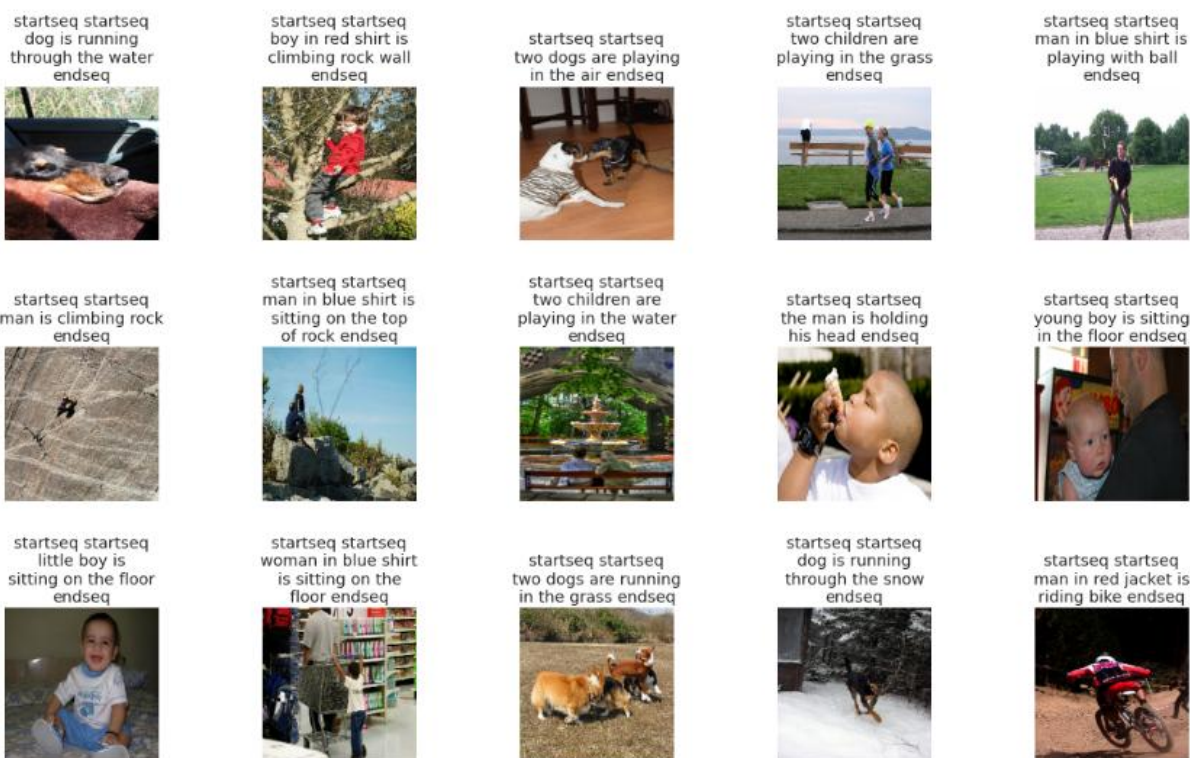
Далее определяются различные обратные вызовы (callbacks), которые будут использоваться в процессе обучения:

- **ModelCheckpoint**: сохраняет модель с лучшим значением функции потерь на валидационном наборе данных.
- **EarlyStopping**: останавливает обучение, если значение функции потерь на валидационном наборе данных не улучшается в течение определенного числа эпох.
- **ReduceLROnPlateau**: уменьшает скорость обучения, если значение функции потерь на валидационном наборе данных не улучшается в течение определенного числа эпох.

Затем модель обучается с использованием метода `fit`, где указывается обучающий генератор данных (`train_generator`), количество эпох, валидационный генератор данных (`validation_generator`) и список обратных вызовов. В процессе обучения для каждой эпохи выводится информация о прогрессе, включая значения функции потерь и метрик на обучающем и валидационном наборах данных.

Тестовая выборка была протестирована на 12 эпохах. Тренировка нейронной сети происходила на GPU NVIDIA GTX 1660 Super. В результате обучения была построена кривая обучения, демонстрирующая явный прирост производительности модели.

Было также проведено тестирование для тестовой выборки, выведены следующие 15 изображений с правдивыми описаниями и верным форматированием, которое задали:



В данной работе была успешно реализована система автоматического генерирования описаний для изображений с использованием моделей глубокого обучения на основе CNN и LSTM. Модель была обучена на датасете Flickr8K и показала способность генерировать осмысленные описания для изображений.

Текст программы

```
import numpy as np
import pandas as pd
import os
import tensorflow as tf
from tqdm import tqdm
from tensorflow.keras.preprocessing.image import ImageDataGenerator,
load_img, img_to_array
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.utils import Sequence, to_categorical
from tensorflow.keras.models import Model
from tensorflow.keras.layers import (Conv2D, MaxPooling2D,
GlobalAveragePooling2D, Activation, Dropout,
Flatten, Dense, Input, Layer,
Embedding, LSTM, add, Concatenate,
```

```

Reshape, concatenate, Bidirectional)
from tensorflow.keras.applications import DenseNet201
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping,
ReduceLRonPlateau
import warnings
import matplotlib.pyplot as plt
import seaborn as sns
from textwrap import wrap

# Set up the environment and suppress warnings
plt.rcParams['font.size'] = 12
sns.set_style("dark")
warnings.filterwarnings('ignore')

# Define paths and load data
image_path = '../input/flickr8k/Images'
data = pd.read_csv("../input/flickr8k/captions.txt")
data.head()

# Define function to read and preprocess image
def read_image(path, img_size=224):
    img = load_img(path, color_mode='rgb', target_size=(img_size,
img_size))
    img = img_to_array(img)
    img /= 255.0
    return img

# Function to display images with captions
def display_images(temp_df):
    temp_df = temp_df.reset_index(drop=True)
    plt.figure(figsize=(20, 20))
    for idx in range(15):
        plt.subplot(5, 5, idx + 1)
        plt.subplots_adjust(hspace=0.7, wspace=0.3)
        image =
read_image(f"../input/flickr8k/Images/{temp_df.image[idx]}")
        plt.imshow(image)
        plt.title("\n".join(wrap(temp_df.caption[idx], 20)))
        plt.axis("off")

display_images(data.sample(15))

# Text preprocessing function
def preprocess_text(data):
    data['caption'] = data['caption'].str.lower()
    data['caption'] = data['caption'].str.replace(r"[^A-Za-z]", "")
    data['caption'] = data['caption'].str.replace(r"\s+", " ")

```

```

    data['caption'] = data['caption'].apply(lambda x: " ".join([word for
word in x.split() if len(word) > 1]))
    data['caption'] = "startseq " + data['caption'] + " endseq"
    return data

data = preprocess_text(data)
captions = data['caption'].tolist()

# Tokenizer setup
tokenizer = Tokenizer()
tokenizer.fit_on_texts(captions)
vocab_size = len(tokenizer.word_index) + 1
max_length = max(len(caption.split()) for caption in captions)

# Split data into training and validation sets
images = data['image'].unique().tolist()
n_images = len(images)
split_index = int(0.85 * n_images)
train_images = images[:split_index]
val_images = images[split_index:]

train_data = data[data['image'].isin(train_images)]
val_data = data[data['image'].isin(val_images)]

train_data.reset_index(drop=True, inplace=True)
val_data.reset_index(drop=True, inplace=True)

# Feature extraction using DenseNet201
model = DenseNet201()
feature_extractor = Model(inputs=model.input, outputs=model.layers[-
2].output)

img_size = 224
features = {}
for image in tqdm(data['image'].unique().tolist()):
    img = load_img(os.path.join(image_path, image),
target_size=(img_size, img_size))
    img = img_to_array(img)
    img /= 255.0
    img = np.expand_dims(img, axis=0)
    feature = feature_extractor.predict(img, verbose=0)
    features[image] = feature

# Custom data generator class
class ImageCaptioningDataGenerator(Sequence):
    def __init__(self, df, X_col, y_col, batch_size, directory,
tokenizer, vocab_size, max_length, features, shuffle=True):
        self.df = df.copy()
        self.X_col = X_col

```

```

self.y_col = y_col
self.batch_size = batch_size
self.directory = directory
self.tokenizer = tokenizer
self.vocab_size = vocab_size
self.max_length = max_length
self.features = features
self.shuffle = shuffle
self.on_epoch_end()

def on_epoch_end(self):
    if self.shuffle:
        self.df = self.df.sample(frac=1).reset_index(drop=True)

def __len__(self):
    return len(self.df) // self.batch_size

def __getitem__(self, index):
    batch = self.df.iloc[index * self.batch_size:(index + 1) *
self.batch_size]
    X1, X2, y = self.__get_data(batch)
    return (X1, X2), y

def __get_data(self, batch):
    X1, X2, y = [], [], []
    for image, caption in zip(batch[self.X_col], batch[self.y_col]):
        feature = self.features[image][0]
        seq = self.tokenizer.texts_to_sequences([caption])[0]
        for i in range(1, len(seq)):
            in_seq, out_seq = seq[:i], seq[i]
            in_seq = pad_sequences([in_seq],
maxlen=self.max_length)[0]
            out_seq = to_categorical([out_seq],
num_classes=self.vocab_size)[0]
            X1.append(feature)
            X2.append(in_seq)
            y.append(out_seq)
    return np.array(X1), np.array(X2), np.array(y)

# Model building
input_img_features = Input(shape=(1920,))
input_seq = Input(shape=(max_length,))

img_features_dense = Dense(256, activation='relu')(input_img_features)
img_features_reshaped = Reshape((1, 256))(img_features_dense)

sentence_embeddings = Embedding(vocab_size, 256,
mask_zero=False)(input_seq)

```

```

merged_input = concatenate([img_features_reshaped, sentence_embeddings],
axis=1)
lstm_output = LSTM(256)(merged_input)
x = Dropout(0.5)(lstm_output)
x = add([x, img_features_dense])
x = Dense(128, activation='relu')(x)
x = Dropout(0.5)(x)
output = Dense(vocab_size, activation='softmax')(x)

captioning_model = Model(inputs=[input_img_features, input_seq],
outputs=output)
captioning_model.compile(loss='categorical_crossentropy',
optimizer='adam')

# Model visualization
from tensorflow.keras.utils import plot_model
plot_model(captioning_model)
captioning_model.summary()

# Generators
train_generator = ImageCaptioningDataGenerator(df=train_data,
X_col='image', y_col='caption', batch_size=64, directory=image_path,
tokenizer=tokenizer,
vocab_size=vocab_size, max_length=max_length, features=features)

val_generator = ImageCaptioningDataGenerator(df=val_data, X_col='image',
y_col='caption', batch_size=64, directory=image_path,
tokenizer=tokenizer,
vocab_size=vocab_size, max_length=max_length, features=features)

# Callbacks
checkpoint = ModelCheckpoint("model.h5", monitor="val_loss",
save_best_only=True, mode="min", verbose=1)
early_stopping = EarlyStopping(monitor='val_loss', patience=5,
restore_best_weights=True, verbose=1)
reduce_lr = ReduceLRonPlateau(monitor='val_loss', patience=3, factor=0.2,
min_lr=1e-8, verbose=1)

# Training the model
history = captioning_model.fit(train_generator, epochs=50,
validation_data=val_generator,
callbacks=[checkpoint, early_stopping,
reduce_lr])

# Plotting training history
plt.figure(figsize=(20, 8))
plt.plot(history.history['loss'], label='train_loss')
plt.plot(history.history['val_loss'], label='val_loss')
plt.title('Model Loss')

```

```

plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(loc='upper left')
plt.show()

# Functions to map indices to words and predict captions
def idx_to_word(integer, tokenizer):
    return tokenizer.index_word.get(integer, None)

def predict_caption(model, image, tokenizer, max_length, features):
    feature = features[image]
    in_text = "startseq"
    for _ in range(max_length):
        sequence = tokenizer.texts_to_sequences([in_text])[0]
        sequence = pad_sequences([sequence], maxlen=max_length)
        y_pred = model.predict([feature, sequence], verbose=0)
        y_pred = np.argmax(y_pred)
        word = idx_to_word(y_pred, tokenizer)
        if word is None or word == 'endseq':
            break
        in_text += " " + word
    return in_text

# Display predictions
samples = val_data.sample(15).reset_index(drop=True)
for idx, record in samples.iterrows():
    img = load_img(os.path.join(image_path, record['image']),
target_size=(224, 224))
    img = img_to_array(img) / 255.0
    caption = predict_caption(captioning_model, record['image'],
tokenizer, max_length, features)
    samples.loc[idx, 'caption'] = caption
display_images(samples)

```

2.32. Распознавание эмоций, возраста и пола на изображении.

DeepFace

Целью работы является разработка программного обеспечения для автоматического распознавания эмоций, возраста и пола людей на изображениях с использованием библиотек OpenCV и DeepFace, а также предобученных моделей [54, 55].

Используемые инструменты

OpenCV (cv2) - библиотека для компьютерного зрения, используемая для обнаружения лиц и манипулирования изображениями.

DeepFace - библиотека для распознавания лиц и анализа эмоций.

Предобученные модели:

- haarcascade_frontalface_default.xml: модель для обнаружения лиц, входящая в состав OpenCV.
- age_deploy.prototxt, age_net.caffemodel: модели для оценки возраста, предоставляемые в формате Caffe.
- gender_deploy.prototxt, gender_net.caffemodel: модели для определения пола, предоставляемые в формате Caffe.

Список используемых функций

Библиотека OpenCV

cv2.dnn.readNet(prototxt_file, caffemodel_file)	Загружает предобученную нейронную сеть из файлов .prototxt и .caffemodel
cv2.dnn.readNetFromCaffe(prototxt_file, caffemodel_file)	Загружает предобученную нейронную сеть (формата Caffe) из файлов .prototxt и .caffemodel
cv2.imread(filename)	Загружает изображение из файла
cv2.CascadeClassifier(filename)	Загружает модель для обнаружения лиц
face_cascade.detectMultiScale(image)	Обнаруживает лица на изображении. Возвращает список прямоугольников
cv2.dnn.blobFromImage(image, scalefactor, size, mean, swapRB)	Используется для преобразования изображения в формат, понятный для моделей определения возраста и пола
model.setInput(info)	Подает данные на вход нейронной сети
model.forward()	Получает предсказание от моделей возраста и пола
cv2.rectangle(image, (x, y), (x + w, y + h), color, thickness)	Рисует прямоугольники вокруг обнаруженных лиц
cv2.putText(image, text, (x, y), font, scale, color, thickness)	Отображает информацию об эмоциях, возрасте и поле рядом с лицами
cv2.imshow(window_name, image)	Выводит результирующее изображение на экран

Библиотека DeepFace

<code>DeepFace.analyze(img_path, actions)</code>	Анализирует изображение лица и возвращает информацию о преобладающих эмоциях
--	--

Алгоритм работы

Сначала задаются значения для настройки освещения (MODEL_MEAN_VALUES) и загружаются модели для обнаружения лиц, определения возраста и пола, а также настраиваются списки возрастов (AGELIST) и полов (GENDERS).

```
MODEL_MEAN_VALUES=(78.4263377603, 87.7689143744, 114.895847746)
GENDERS = ["Male", "Female"]
AGELIST = ['(0-2)', '(4-6)', '(8-12)', '(15-20)', '(25-32)', '(38-43)', '(48-53)', '(60-100)']
```

Функция `detectMultiScale` возвращает список прямоугольников, каждый из которых определяет координаты и размеры обнаруженного лица.

```
faces = face_cascade.detectMultiScale(image)
```

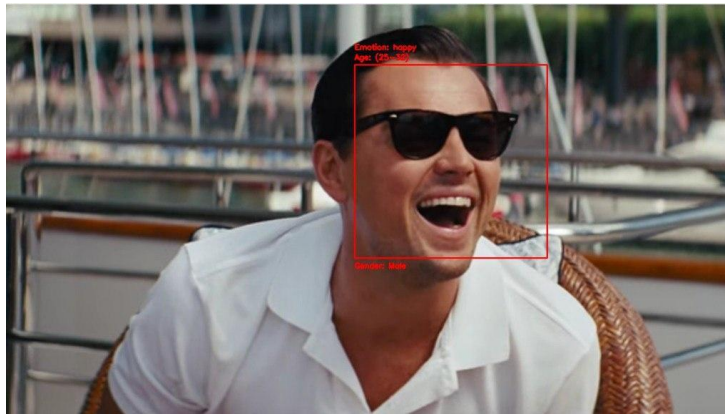
В цикле по обнаруженным лицам вырезается ROI (непосредственная область лица). Далее библиотека DeepFace анализирует переданное ей изображение и определяет доминирующую эмоцию.

```
result = DeepFace.analyze(roi, actions=['emotion']) #Возвращает список с
результатами анализа
emotion = result[0]['dominant_emotion'] #получает строку с названием
доминирующей эмоции
```

Для определения возраста и пола используются `age_model` и `gender_model`.

```
age_model.setInput(roi_age_gender)
preds = age_model.forward()
age = AGELIST[preds[0].argmax()]
gender_model.setInput(roi_age_gender)
preds = gender_model.forward()
gender = GENDERS[preds[0].argmax()]
```

Результат работы



Текст программы

```
import cv2
from deepface import DeepFace
MODEL_MEAN_VALUES=(78.4263377603, 87.7689143744, 114.895847746)
GENDERS = ["Male", "Female"]
AGELIST = ['(0-2)', '(4-6)', '(8-12)', '(15-20)', '(25-32)', '(38-43)',
'(48-53)', '(60-100)']

# Загрузка моделей
face_cascade = cv2.CascadeClassifier(cv2.data.harcascades +
'haarcascade_frontalface_default.xml')
age_model = cv2.dnn.readNet("age_deploy.prototxt", "age_net.caffemodel")
gender_model = cv2.dnn.readNetFromCaffe("gender_deploy.prototxt",
"gender_net.caffemodel")

# Загрузка изображения
images = [cv2.imread("test1.jpeg"), cv2.imread("test2.jpg"),
cv2.imread("test3.jpeg")]

for image in images :
    # Обнаружение лиц
    faces = face_cascade.detectMultiScale(image)

    for (x, y, w, h) in faces:
        # Извлечение ROI (Region of Interest)
        roi = image[y:y + h, x:x + w]

        # Определение эмоций
        try:
            result = DeepFace.analyze(roi, actions=['emotion'])
            emotion = result[0]['dominant_emotion']
        except:
            emotion=""
            print("DeepFace не смог определить эмоцию")

# получаем новый бинарный пиксельный объект
```

```

roi_age_gender = cv2.dnn.blobFromImage(roi, 1.0, (227, 227),
MODEL_MEAN_VALUES, swapRB=False)

# Определение возраста и пола
age_model.setInput(roi_age_gender)
preds = age_model.forward()
age = AGELIST[preds[0].argmax()]

gender_model.setInput(roi_age_gender)
preds = gender_model.forward()
gender = GENDERS[preds[0].argmax()]

# Отображение результата на изображении
cv2.rectangle(image, (x, y), (x + w, y + h), (0, 0, 255), 2)
cv2.putText(image, f"Emotion: {emotion}", (x, y - 30),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)
cv2.putText(image, f"Age: {age}", (x, y - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)
cv2.putText(image, f"Gender: {gender}", (x, y + h + 20),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)

# Отображение результата
cv2.imshow("Facial Attributes Detection", image)
cv2.waitKey(0)
cv2.destroyAllWindows()

```

Глава 3. Алгоритмы распознавания и преобразования видео

3.1. Классификация эмоций на видео. TensorFlow

Рассмотрена классификация семи основных эмоций на видео: злость, счастье, страх, грусть, отвращение, удивление, нейтральное

Для решения задачи предоставляются изображения с эмоциями и размеченные *.csv файлы:

- train_labels.csv – датасет для обучения
- test_labels.csv – датасет для проверки системы

Датасет для обучения взят с веб-сайта kaggle.com:

<https://www.kaggle.com/datasets/shuvoalok/raf-db-dataset>.

Описание процесса обучения

Решение описано на языке языке python. Версия 3.8. В ходе работы использовались следующие модули:

- tensorflow
- keras
- pandas
- matplotlib

Обзор дерикторий

```
for dirpath, dirname, filenames in os.walk("../input/far-new/Dataset"):
    print(f"There are {len(dirname)} directories and there are
    {len(filenames)} images in {dirpath}")
```

Этот код проходит по дереву директорий, начиная с "../input/far-new/Dataset", и выводит количество поддиректорий (dirname) и количество файлов (filenames) в каждой директории.

Ввод пользователя и случайный выбор

```
exp = input("Enter the class : ")
data_set = random.choice(["train", "test"])
```

Пользователя просят ввести класс (exp). Кроме того, случайным образом выбирается набор данных (data_set) из опций "train" или "test".

Отображение случайных изображений

```
target_dir = f"../input/far-new/Dataset/Dataset/{data_set}/{exp}"
random_images = random.sample(os.listdir(target_dir), 40)
```

Случайным образом выбирает 40 изображений из указанной директории (target_dir) и отображает их в сетке 4x10 с помощью Matplotlib.

Загрузка данных

```
train_dir = "../input/far-new/Dataset/Dataset/train"
test_dir = "../input/far-new/Dataset/Dataset/test"
```

```
train_data =
tf.keras.preprocessing.image_dataset_from_directory(train_dir, image_size
= (128, 128), shuffle = True, label_mode = "categorical")
```

```
test_data = tf.keras.preprocessing.image_dataset_from_directory(test_dir,
image_size = (128, 128), shuffle = True, label_mode = "categorical")
```

Загружает наборы данных изображений из указанных директорий для обучения и тестирования, с размером изображения 128x128 и категориальными метками.

Предварительная подготовка данных

```
train_dataset = train_data.prefetch(tf.data.AUTOTUNE)
test_dataset = test_data.prefetch(tf.data.AUTOTUNE)
```

Предварительно подгружает данные для оптимизации производительности обучения.

Аугментация данных

```
data_augmentation_layer = tf.keras.Sequential([
    preprocessing.RandomFlip("horizontal"),
    preprocessing.RandomZoom(0.2),
    preprocessing.RandomWidth(0.1),
    preprocessing.RandomHeight(0.1)
])
```

Определяет слой аугментации данных с использованием функций предварительной обработки TensorFlow.

Определение и компиляция модели

```
model = tf.keras.Sequential([
    data_augmentation_layer,
    layers.Conv2D(64, 3, activation = "relu", input_shape = (128, 128,
3)),
    layers.Conv2D(64, 3, activation = "relu"),
    layers.MaxPooling2D(2),
    layers.Conv2D(128, 3, activation = "relu"),
    layers.Conv2D(128, 3, activation = "relu"),
    layers.MaxPooling2D(2),
    layers.Conv2D(256, 3, activation = "relu"),
    layers.Dropout(0.25),
    layers.Conv2D(256, 3, activation = "relu"),
    layers.Dropout(0.25),
    layers.MaxPooling2D(2),
    layers.Conv2D(512, 3, activation = "relu"),
    layers.Dropout(0.5),
    layers.GlobalAveragePooling2D(),
    layers.Dense(512, activation = "relu"),
    layers.Dropout(0.25),
    layers.Dense(512, activation = "relu"),
    tf.keras.layers.Dense(7, activation = "softmax")
])

model.compile(loss = "categorical_crossentropy", optimizer =
tf.keras.optimizers.Adam(0.0006), metrics = ["accuracy"])
```

Обучение модели

```
model_history = model.fit(train_dataset,
                           epochs = 30,
                           validation_data = test_dataset,
                           callbacks = [checkpoint_callback])
```

Обучает модель на протяжении 30 эпох с использованием указанных тренировочного и валидационного наборов данных, с обратным вызовом для сохранения лучших весов на основе точности валидации.

Построение графика истории обучения модели

```
pd.DataFrame(model_history.history).plot()
```

Строит график метрик обучения и валидации по эпохам с использованием библиотек pandas и Matplotlib.

Сохранение модели

```
model.save("Final_tweak")  
model.save("Final_tweak.h5")
```

Сохраняет модель в формате TensorFlow SavedModel и в формате HDF5.

Создание архива Zip

```
import shutil  
shutil.make_archive('Final_tweak', 'zip', './Final_tweak')
```

Создает zip-архив сохраненных файлов модели.

Вывод по обучению

В целом, этот код выполняет загрузку данных изображений, аугментацию данных, обучение модели и сохранение модели для задачи распознавания выражения лица. Данные организованы в различных директориях для обучения и тестирования, и модель представляет собой сверточную нейронную сеть (CNN) с аугментацией данных для улучшения обобщения. Прогресс обучения мониторится, и лучшие веса модели сохраняются с использованием обратного вызова. Наконец, обученная модель сохраняется как TensorFlow SavedModel и в виде zip-архива.

Описание работы программы

Решение описано на языке языке python. Версия 3.8. В ходе работы использовались следующие модули: tensorflow, pillow, numpy, OpenCV, flask, shutil.

Приложение представляет собой веб-приложение Flask, объединяющее модель обнаружения лиц и модель классификации эмоций

для анализа видео. В листингах кода привел только функциональный код, без пользовательского интерфейса.

Загрузка моделей

```
detect_fn = tf.saved_model.load("Models/FaceDetector/saved_model")#Load
the face detector
model = tf.keras.models.load_model("Models/FEC")#Load the facial emotion
classifier
```

Функции для отрисовки рамок вокруг лица

Эта функция выполняет уточнение координат ограничивающих рамок с использованием алгоритма "Non-Maximum Suppression" (NMS).

```
def bound(boxes, scores, h, w):
    # Входные параметры:
    # - boxes: Список координат ограничивающих рамок
    # - scores: Список уверенностей в обнаружении лиц
    # - h, w: Высота и ширина изображения

    idxs = cv2.dnn.NMSBoxes(boxes, scores, 0.5, 1.5)

    # Создание списка координат после применения NMS
    signs = []
    for i in range(len(idxs)):
        signs.append(i)

    # Уточнение координат
    if len(idxs) > 0:
        for i in idxs.flatten():
            ymin = int((boxes[i][0] * h))
            xmin = int((boxes[i][1] * w))
            ymax = int((boxes[i][2] * h))
            xmax = int((boxes[i][3] * w))
            signs[i] = [ymin, ymax, xmin, xmax]

    return signs
```

Эта функция предназначена для отрисовки рамок вокруг лиц на изображении с использованием модели обнаружения лиц (detect_fn)

```
def draw_bounding_box(frame, detect_fn):
    # Входные параметры:
    # - frame: Изображение (кадр видео или фотография)
    # - detect_fn: Модель обнаружения лиц

    # Преобразование кадра в тензор
    input_tensor = tf.convert_to_tensor(frame)
```

```

input_tensor = input_tensor[tf.newaxis, ...]

# Получение результатов обнаружения лиц от модели
detections = detect_fn(input_tensor)
num_detections = int(detections.pop('num_detections'))
detections = {key: value[0, :num_detections].numpy() for key, value
in detections.items()}
boxes = detections['detection_boxes']
scores = detections['detection_scores']

# Получение координат обнаруженных лиц
h, w = frame.shape[:2]
boxes = boxes.tolist()
scores = scores.tolist()
coordinates = bound(boxes, scores, h, w)

# Отрисовка рамок вокруг лиц
for (y, h, x, w) in coordinates:
    cv2.rectangle(frame, (x, y), (w, h), (0, 255, 0), 2)

return frame

```

Функция для обработки видео, обнаружения лиц, классификации эмоций и создания нового видео

Эта функция обрабатывает видео кадр за кадром, обнаруживает лица, классифицирует эмоции и создает новое видео с рамками и метками эмоций. Она использует функцию `draw_bounding_box` для обнаружения и отрисовки рамок.

```

def detectandupdatevideo(video):
    # Входные параметры:
    # - video: Имя файла видео

    # Строим путь к видео
    output_path = f"static/pred{video}"
    fourcc = cv2.VideoWriter_fourcc(*'mp4v')
    out = cv2.VideoWriter(output_path, fourcc, 25.0, (640, 360))
    vidcap = cv2.VideoCapture(f"static/{video}")

    while True:
        ret, image = vidcap.read()
        if ret == True:
            # Получение координат обнаруженных лиц и отрисовка рамок
            coordinates = draw_bounding_box(image, detect_fn)

```

```

# Обработка каждого лица на кадре
for (y, h, x, w) in coordinates:
    # Выделение лица
    img2 = image[y:h, x:w]
    # Изменение размера лица до 128x128 (для входа в модель)
    img2 = tf.image.resize(img2, size=[128, 128])
    # Получение предсказаний модели по эмоциям
    pred = model.predict(tf.expand_dims(img2, axis=0))
    # Определение класса эмоции
    pred_class = class_names[tf.argmax(pred,
axis=1).numpy()[0]]

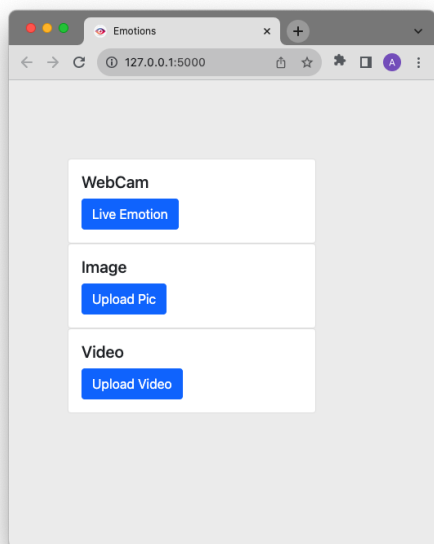
    # Добавление текста с классом эмоции к изображению
    if x > 20 and y > 40:
        cv2.putText(image, pred_class, (x, y - 10),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
    else:
        cv2.putText(image, pred_class, (x + 10, y + 20),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)

    # Запись обработанного кадра в новое видео
    output_frame = cv2.resize(image, (640, 360))
    out.write(output_frame)
else:
    break
# Освобождение ресурсов
vidcap.release()
out.release()

return [video, "pred" + video]

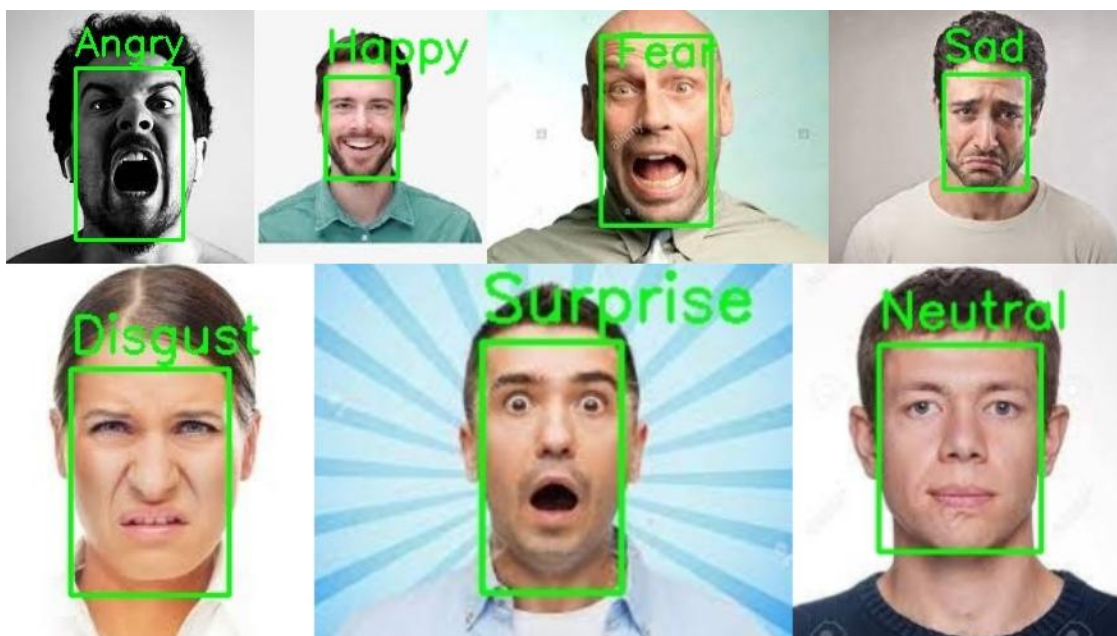
```

Пользовательский интерфейс описанной программы представлен ниже.



Результаты

На изображении показана классификация всех эмоций, которые доступны с данным датасетом.



На скорость работы программы очень сильно влияет наличие дискретного видеоускорителя, на компьютере, который был использован в данной работе такой отсутствует, для анализа изображение в качестве 720р и 30fps и весом 2.5мб уходит чуть более 20 минут. Обученная модель имеет точность обучения 90% и точность тестирования 83%.

3.2. Классификация объектов на видео. MobileNetSSD

Целью работы является разработка системы классификации объектов с использованием MobileNetSSD и библиотеки OpenCV. MobileNet - передовая архитектура сверточной сети, позволяющая получить высокую производительность при минимальной аппаратуре. Задачи проекта включают следующее:

- Подбор датасета
- Разработка программы для классификации объектов в реальном времени
- Анализ результатов

Описание работы программы

Решение описано на языке python. В ходе работы использовались следующие модули:

1. `imutils.video` - модуль для работы с видео в OpenCV. Предоставляет удобный интерфейс для чтения и записи видеофайлов, а также для работы с камерами.

2. `imutils` - модуль, содержащий различные утилиты для обработки изображений и видео. Включает в себя функции для изменения размера изображения, поворота, наложения масок и другие.

3. `numpy` - библиотека для работы с массивами и матрицами чисел.

4. `time` - модуль для работы со временем. Предоставляет функции для измерения времени выполнения программы и задержек.

5. `cv2` - библиотека OpenCV для работы с изображениями и видео. Содержит функции для чтения и записи файлов, обработки изображений, детекции объектов и многое другое.

Загрузка модели

```
# The model from Caffe: MobileNetSSD_deploy.prototxt.txt;  
MobileNetSSD_deploy.caffemodel;
```

```
net = cv2.dnn.readNetFromCaffe(args["prototxt"], args["model"])
```

Инициализация видеопотока

```
vs = VideoStream(src=0).start()  
time.sleep(2.0)
```

```
# FPS: used to compute the (approximate) frames per second  
# Start the FPS timer  
fps = FPS().start()
```

Обработка видеопотока

Рассматриваем видеопоток как серию кадров и подготавливаем изображение для классификации.

```
# grab the frame from the threaded video stream and resize it to have a  
maximum width of 400 pixels  
frame = vs.read()  
frame = imutils.resize(frame, width=400)  
(h, w) = frame.shape[:2]  
# Resize each frame  
resized_image = cv2.resize(frame, (300, 300))  
# Creating the blob  
# blob = cv2.dnn.blobFromImage(image, scalefactor=1.0, size, mean,  
swapRB=True)  
# the input image we want to preprocess before passing it through  
our deep neural network for classification
```

```
blob = cv2.dnn.blobFromImage(resized_image, (1/127.5), (300, 300),  
127.5, swapRB=True)  
# pass the blob through the network and obtain the predictions  
net.setInput(blob) # net =  
cv2.dnn.readNetFromCaffe(args["prototxt"], args["model"])  
# Predictions:  
predictions = net.forward()
```

Отображение классифицированного объекта

Циклом проходим по предсказаниям и при достаточном уровне уверенности отображаем рамку с подписью вокруг объекта

```
for i in np.arange(0, predictions.shape[2]):  
# extract the confidence (i.e., probability) associated with the  
prediction  
    confidence = predictions[0, 0, i, 2]  
# Filter out predictions lesser than the minimum confidence level  
# Here, we set the default confidence as 0.2. Anything lesser than 0.2  
will be filtered  
    if confidence > args["confidence"]:
```

```

# extract the index of the class label from the 'predictions'
    idx = int(predictions[0, 0, i, 1])
# then compute the (x, y)-coordinates of the bounding box for the object
    box = predictions[0, 0, i, 3:7] * np.array([w, h, w, h])

    (startX, startY, endX, endY) = box.astype("int")

# Get the label with the confidence score
    label = "{}: {:.2f}%".format(CLASSES[idx], confidence *
100)

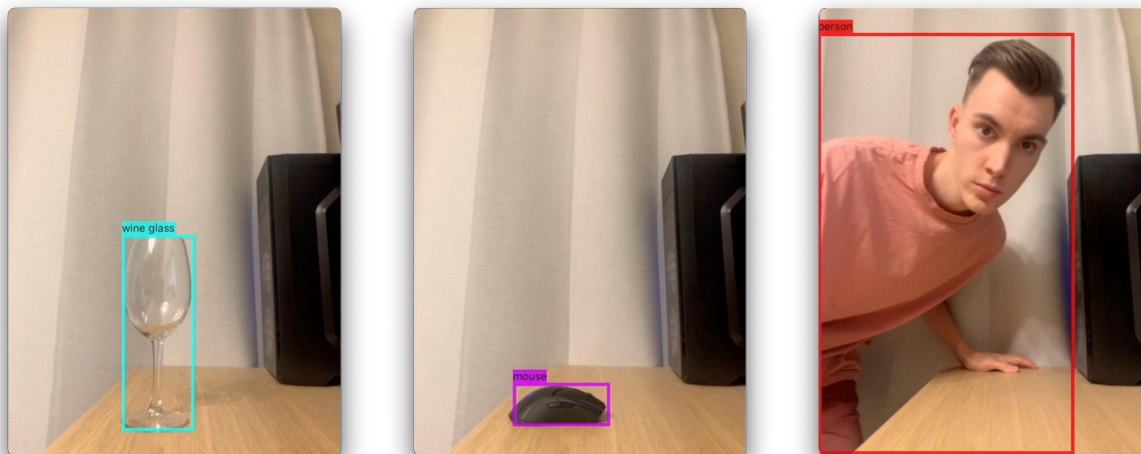
    print("Object detected: ", label)
# Draw a rectangle across the boundary of the object
    cv2.rectangle(frame, (startX, startY), (endX, endY),
        COLORS[idx], 2)
    y = startY - 15 if startY - 15 > 15 else startY + 15
# Put a text outside the rectangular detection
    cv2.putText(frame, label, (startX, y),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, COLORS[idx], 2)

# show the output frame
    cv2.imshow("Frame", frame)

```

Результаты

На изображениях показана классификация некоторых объектов, которые доступны с данным датасетом.



3.3. Трекинг с использованием нейросети YOLOv8

Целью работы является разработка и реализация системы для автоматического трекинга птиц на видеопотоке.

Нейросеть YOLOv8

You Only Look Once (YOLO) предоставляет сквозную нейронную сеть, которая делает прогнозы ограничивающих полей и вероятностей класса одновременно. Это отличается от подхода, используемого предыдущими алгоритмами обнаружения объектов, которые перепрофилировали классификаторы для выполнения обнаружения.

Следуя принципиально другому подходу к обнаружению объектов, YOLO достигла самых современных результатов, обогнав другие алгоритмы обнаружения объектов в реальном времени с большим отрывом.

В то время как такие алгоритмы, как Faster RCNN, работают, обнаруживая возможные регионы, представляющие интерес, с помощью сети региональных предложений, а затем выполняя распознавание этих регионов отдельно, YOLO выполняет все свои прогнозы с помощью одного полностью подключенного уровня. Методы, использующие Region Proposal Networks, выполняют несколько итераций для одного и того же образа, в то время как YOLO необходима одна итерация.

С момента первоначального выпуска YOLO в 2015 году было предложено несколько новых версий одной и той же модели, каждая из которых строила и улучшила своего предшественника.

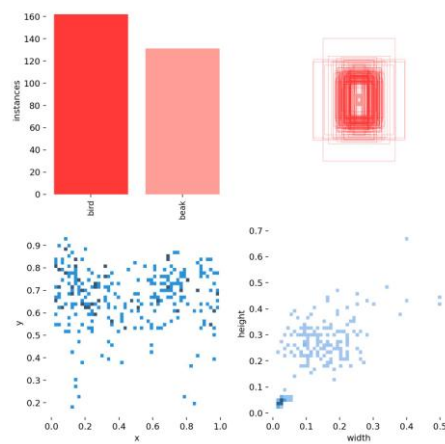
YOLO v8 предоставляет новый API, который значительно упростит обучение и вывод как на устройствах CPU, так и на GPU, а фреймворк поддерживает предыдущие версии YOLO. YOLOv8 имеет высокую скорость точности, измеренную Microsoft COCO и Roboflow 100.

Алгоритм работы

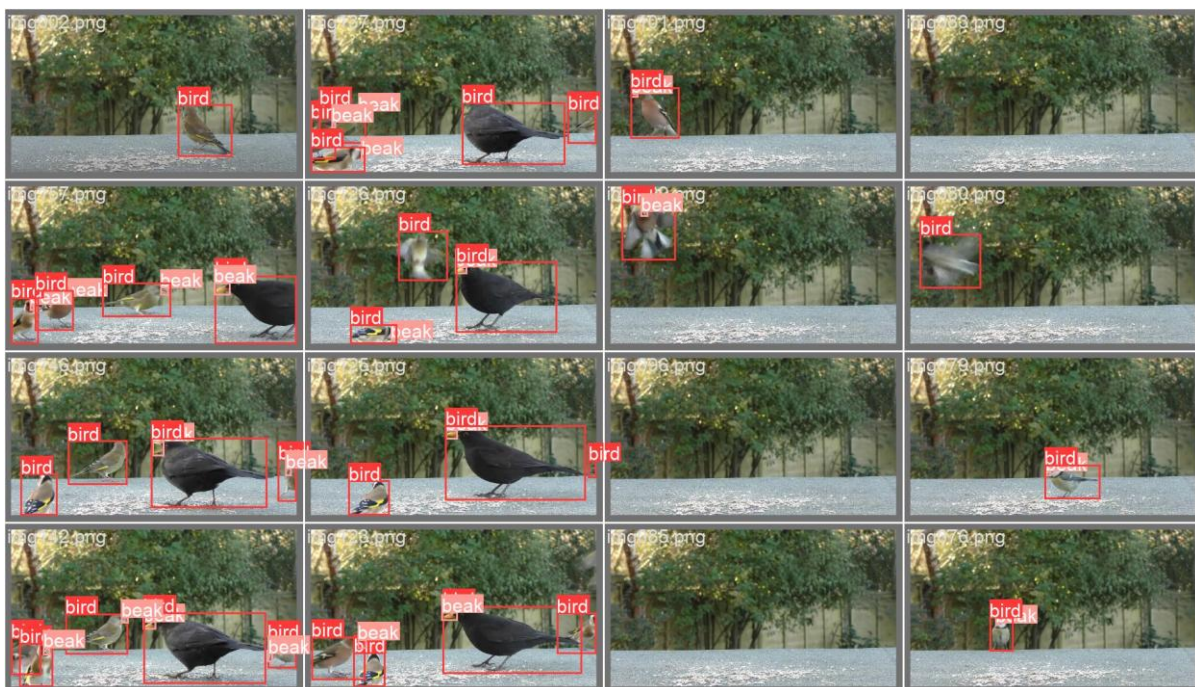
Алгоритм состоит из двух частей:

1. Дообучение нейросети

В качестве базовой модели использовалась YOLOv8m как компромисс между скоростью обучения и качеством результата.



2. Выполнение обученной модели с трекингом класса птиц



Используемые функции

Название функций	Описание
YOLO('yolov8m.pt') .train(data='./data.yaml', epochs=100)	Запускает процесс обучения модели на основе yolov8m, со ссылкой на описание датасета в data.yaml на 100 эпох
YOLO('best.pt')	Запускает процесс распознавания видеофрагментов

<code>.predict(frame, conf=0.5, imgsz=SIZE)</code>	по шаблону. Распознавание считается успешным если уверенность в распознавание выше 50% (<code>conf=0.5</code>), шаблоны меняют размер до <code>SIZE</code>
<code>cv2.VideoCapture()</code>	Захват видео

Результат



Текст программы

```
import cv2
from ultralytics import YOLO
import random

def draw_bounding_boxes_without_id(frame, results):
    boxes = results[0].boxes.xyxy.cpu().numpy().astype(int)
    classes = results[0].boxes.cls.cpu().numpy().astype(int)

    for box, cls in zip(boxes, classes):
        # Generate a random color for each object based on its ID
        if cls != 0:
            random.seed(int(cls)+8)
            color = (random.randint(0, 255), random.randint(0, 255),
                    random.randint(0, 255))

            cv2.rectangle(frame, (box[0], box[1]), (box[2], box[3]),
                color, 2)
            cv2.putText(
                frame,
                f"{model.model.names[cls]}",
                (box[0], box[1]),
                cv2.FONT_HERSHEY_SIMPLEX,
                0.6,
                (50, 255, 50),
                2,
            )
    return frame

def process_video_with_tracking(model, input_video_path, show_video=True,
    save_video=False, output_video_path="output_video.mp4"):
```

```

# Open the input video file
cap = cv2.VideoCapture(input_video_path)

if not cap.isOpened():
    raise Exception("Error: Could not open video file.")

# Get input video frame rate and dimensions
fps = int(cap.get(cv2.CAP_PROP_FPS))
frame_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
frame_height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

# Define the output video writer
if save_video:
    fourcc = cv2.VideoWriter_fourcc(*'mp4v')
    out = cv2.VideoWriter(output_video_path, fourcc, fps,
(frame_width, frame_height))

while True:
    ret, frame = cap.read()
    if not ret:
        break
    results = model.track(frame, iou=0.4, conf=0.5, persist=True,
imgsz=608, verbose=False, tracker="bytetrack.yaml", classes=0)
    results_detect = model_detect.predict(frame, iou=0.4, conf=0.5,
imgsz=608, verbose=False)

    if results[0].boxes.id != None: # this will ensure that id is not
None -> exist tracks
        boxes = results[0].boxes.xyxy.cpu().numpy().astype(int)
        ids = results[0].boxes.id.cpu().numpy().astype(int)

        for box, id in zip(boxes, ids):
            # Generate a random color for each object based on its ID
            random.seed(int(id))
            color = (random.randint(0, 255), random.randint(0, 255),
random.randint(0, 255))

            cv2.rectangle(frame, (box[0], box[1]), (box[2], box[3]),,
color, 2)

            cv2.putText(
                frame,
                f"Id {id}",
                (box[0], box[1]),
                cv2.FONT_HERSHEY_SIMPLEX,
                0.70,
                (0, 255, 255),
                2,
            )

        if results_detect[0].boxes != None:

```

```

        draw_bounding_boxes_without_id(frame, results_detect)

    if save_video:
        out.write(frame)
    if show_video:
        frame = cv2.resize(frame, (0, 0), fx=0.75, fy=0.75)
        cv2.imshow("frame", frame)
    if cv2.waitKey(1) & 0xFF == ord("q"):
        break

# Release the input video capture and output video writer
cap.release()
if save_video:
    out.release()

# Close all OpenCV windows
cv2.destroyAllWindows()
return results_detect, results

# Example usage:
model = YOLO('runs/detect/train/weights/best.pt')
model_detect = YOLO('runs/detect/train/weights/best.pt')
model.fuse()
model_detect.fuse()
results_detect, results = process_video_with_tracking(model,
"test_1.mp4", show_video=False, save_video=True,
output_video_path="output_video.mp4")

```

3.4. Обнаружения черт лица в реальном времени. Dlib

При помощи библиотеки для компьютерного зрения OpenCV и библиотеки DLib разработано приложение для обнаружения черт лица в реальном времени. Dlib – это библиотека машинного обучения, используемая для разработки приложений компьютерного зрения [42].

Список использованных функций

Библиотека OpenCV

VideoCapture()	Получение камеры для работы
read()	Считывание изображения с камеры
cvtColor()	Преобразование цветовой палитры изображения
circle()	Отрисовка круга с заданными параметрами
Imshow()	Отображение изображения в окне
waitKey()	Ожидание нажатия определённой клавиши
destroyAllWindows()	Закрытие всех окон

Библиотека DLib

<code>get_frontal_face_detector()</code>	Получаем detector – функцию, которую мы можем использовать для получения информации о лицах
<code>shape_predictor()</code>	Выбор модели для определения черт лица

Считывать изображение через фронтальную камеру и передавать его в обработку для выделения черт лица мы будем при помощи OpenCV, во фрагменте кода ниже, мы получаем для работы веб камеру в переменную `cap`, считываем изображение и переводим его в градацию серого (что является условием для корректного определения черт лица):

```
cap = cv2.VideoCapture(0)
...
_, frame = cap.read()
gray = cv2.cvtColor(src=frame, code=cv2.COLOR_BGR2GRAY)
```

Далее, при помощи методов библиотека DLib, выделяем определяем границы лица на изображении:

```
detector = dlib.get_frontal_face_detector()
...
faces = detector(gray)
```

Затем передаём эти границы в метод-детектор, который используя обученную модель будет определять черты лица:

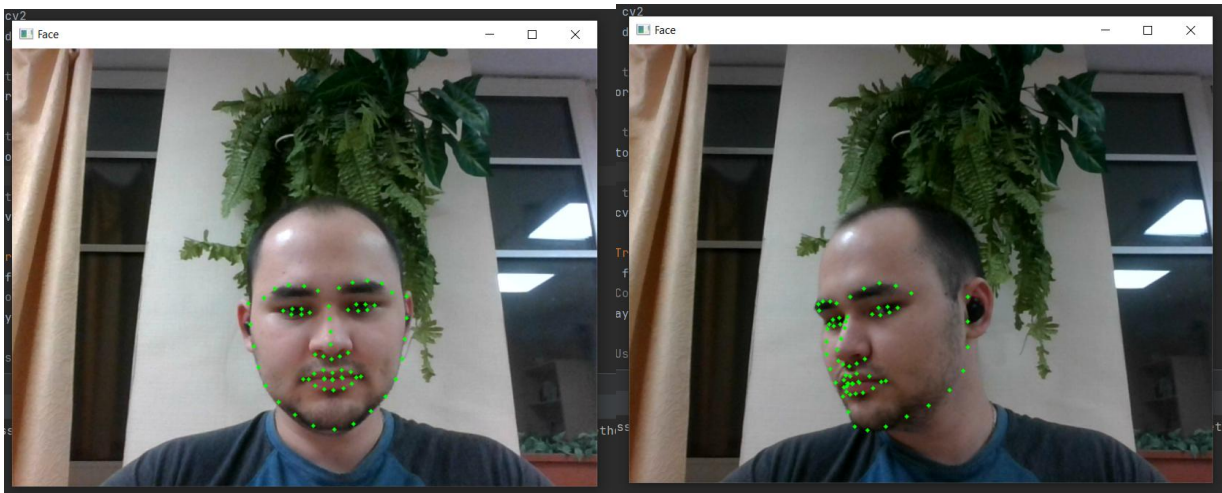
```
landmarks = predictor(image=gray, box=face)
```

И отрисовываем каждую полученную точку поверх нашего исходного изображения для демонстрации определения черт лица:

```
for n in range(0, 68):
    x = landmarks.part(n).x
    y = landmarks.part(n).y

    # Draw a circle
    cv2.circle(img=frame, center=(x, y), radius=2, color=(0,
255, 0), thickness=-1)
```

Результаты работы



Текст программы

```

import cv2
import dlib
# Load the detector
detector = dlib.get_frontal_face_detector()
# Load the predictor
predictor = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")
# read the image
cap = cv2.VideoCapture(0)
while True:
    _, frame = cap.read()
    # Convert image into grayscale
    gray = cv2.cvtColor(src=frame, code=cv2.COLOR_BGR2GRAY)
    # Use detector to find landmarks
    faces = detector(gray)

    for face in faces:
        x1 = face.left() # left point
        y1 = face.top() # top point
        x2 = face.right() # right point
        y2 = face.bottom() # bottom point
        # Create landmark object
        landmarks = predictor(image=gray, box=face)
        # Loop through all the points
        for n in range(0, 68):
            x = landmarks.part(n).x
            y = landmarks.part(n).y

            # Draw a circle
            cv2.circle(img=frame, center=(x, y), radius=2, color=(0, 255,
0), thickness=-1)
        # show the image
        cv2.imshow(winname="Face", mat=frame)
        if cv2.waitKey(delay=1) == 27:

```

```
break
cap.release()
cv2.destroyAllWindows()
```

3.5. Запись распознанного текста на видео. **SpeechRecognition**

Современные технологии обработки видеoinформации открывают множество возможностей для создания интересного и информативного контента. В данной работе мы рассмотрим методы записи видео с веб-камеры, а также добавление субтитров и аудиодорожки на основе распознавания речи. Для реализации этой задачи мы воспользуемся популярными библиотеками: **OpenCV** для работы с видео, **SpeechRecognition** для распознавания речи, и **MoviePy** для монтажа видео.

Задачи:

Запись видео с веб-камеры:

- Использование библиотеки **OpenCV** для захвата видеопотока с веб-камеры.
- Запись полученных кадров в видеофайл формата **AVI**.

Запись аудио:

- Использование библиотеки **PyAudio** для записи аудиодорожки с микрофона.
- Сохранение аудио в формате **WAV**.

Распознавание речи [57]:

- Использование библиотеки **SpeechRecognition** для распознавания речи в записанном аудиофайле.

Добавление субтитров:

- Разработка функции `add_subtitle`, добавляющей субтитры к каждому кадру видео.

Объединение видео, аудио и субтитров:

- Создание нового видеофайла с добавленными субтитрами.
- Использование **MoviePy** для объединения видео, аудио и субтитров.

Алгоритм работы

Инициализация:

- Запуск программы и инициализация библиотек.

Запись видео и аудио:

- Использование OpenCV и PyAudio для записи видео и аудио в отдельные файлы.

Распознавание речи:

- Применение SpeechRecognition для распознавания речи в записанной аудиодорожке.

Добавление субтитров:

- Проход по каждому кадру видео с использованием функции add_subtitle.

Объединение видео и аудио:

- Использование MoviePy для создания нового видеофайла с добавленными субтитрами и оригинальной аудиодорожкой.

Используемые функции:

transcribe_speech(audio_path):

- Функция для распознавания речи в записанном аудиофайле с использованием Google Speech Recognition.

add_subtitle(frame, subtitle_text):

- Функция для добавления субтитров к каждому кадру видео.

main():

- Основная функция программы, содержащая в себе весь алгоритм работы.
- Инициализация видео- и аудиозаписи, запись кадров и аудиоданных.
- Распознавание речи и добавление субтитров к видео.
- Объединение видео, аудио и субтитров в итоговый файл.

Данный код создает итоговое видео с добавленными субтитрами на основе распознанной речи, предоставляя удобный инструмент для создания информативного контента с автоматически сгенерированными субтитрами.

Используемые функции библиотек

OpenCV:

`cv2.VideoCapture` и `cv2.VideoWriter`:

- `cv2.VideoCapture` используется для захвата видеопотока с веб-камеры.
- `cv2.VideoWriter` используется для записи кадров в видеофайл формата AVI.

`cv2.imshow` и `cv2.waitKey`:

- `cv2.imshow` отображает текущий кадр видео в окне с указанным названием.
- `cv2.waitKey` ожидает нажатие клавиши. В данном коде, оно используется для прерывания записи, если пользователь нажимает клавишу 'q'.

`cv2.putText`:

- Используется в функции `add_subtitle` для добавления текста (субтитров) к каждому кадру видео.

`cv2.getTickCount` и `cv2.getTickFrequency`:

- Используются для измерения времени записи видео и ограничения ее продолжительности в 5 секунд.

SpeechRecognition:

`sr.Recognizer` и `sr.AudioFile`:

- `sr.Recognizer` представляет объект для распознавания речи.
- `sr.AudioFile` используется для открытия аудиофайла и последующего считывания аудиоданных.

`recognizer.record` и `recognizer.recognize_google`:

- `recognizer.record` записывает аудиоданные из файла.
- `recognizer.recognize_google` используется для распознавания речи с использованием Google Speech Recognition.

MoviePy:

VideoFileClip, AudioFileClip и write_videofile:

- VideoFileClip загружает видеофайл.
- AudioFileClip загружает аудиофайл.
- write_videofile используется для создания нового видеофайла с добавленными субтитрами и оригинальной аудиодорожкой.

set_audio:

- set_audio используется для установки аудиодорожки для видеоклипа.

PyAudio:

pyaudio.PyAudio и pyaudio.Stream:

- pyaudio.PyAudio используется для создания экземпляра PyAudio для работы с аудио.
- pyaudio.Stream представляет аудиопоток для записи аудиоданных.

stream.read:

- Используется для чтения аудиоданных из микрофона и их записи в массив audio_frames.

Важные моменты:

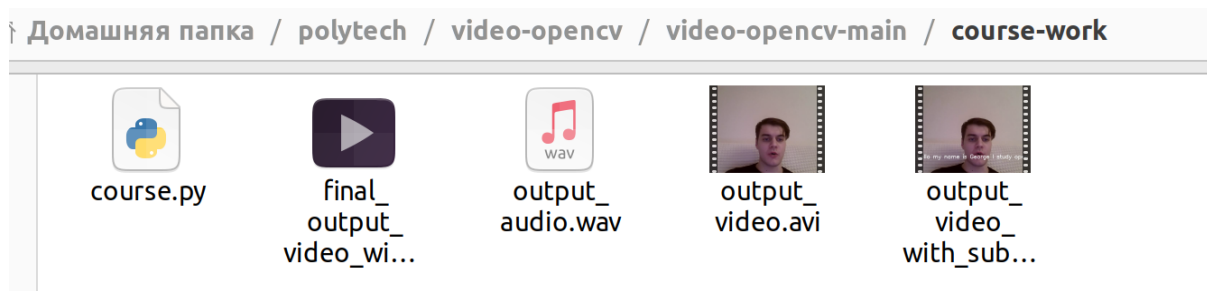
- cv2.waitKey(1) & 0xFF == ord('q'):
 - Эта проверка позволяет пользователю прервать запись, нажав клавишу 'q' в окне с видеопотоком.
- Обработка ошибок ввода-вывода в PyAudio:
 - В блоке записи аудио ведется обработка ошибок, в том числе случая переполнения буфера (input overflowed).
- Остановка записи по времени:
 - Запись видео продолжается в течение 5 секунд, что регулируется использованием cv2.getTickCount и cv2.getTickFrequency.
- Объединение видео и аудио в MoviePy:
 - Используется VideoFileClip для загрузки видеофайла, AudioFileClip для загрузки аудиофайла, и set_audio для

установки аудиодорожки для видеоклипа. Затем `write_videofile` используется для сохранения итогового видеофайла.

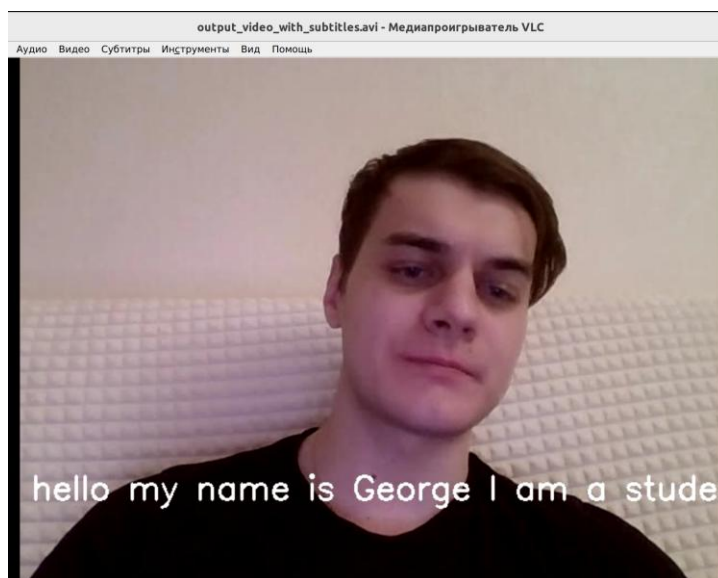
Эти библиотеки предоставляют удобные средства для обработки видео, звука и речи, их комбинирования и создания нового мультимедийного контента.

Результаты работы

После записи видео, где была произнесена фраза - “Hello! My name is George, I am a student”, было создано 4 файла: исходный клип, аудиодорожка, клип с субтитрами и звуком и клип с субтитрами без звука.



Далее пример распознанной фразы:



Текст программы

```
import cv2
import numpy as np
import pyaudio
import wave
from moviepy.editor import VideoFileClip, AudioFileClip
```

```

import speech_recognition as sr

def transcribe_speech(audio_path):
    recognizer = sr.Recognizer()
    with sr.AudioFile(audio_path) as source:
        print("Transcribing...")
        audio = recognizer.record(source)
    try:
        text = recognizer.recognize_google(audio)
        print("Transcription complete.")
        return text
    except sr.UnknownValueError:
        print("Speech Recognition could not understand audio")
        return ""
    except sr.RequestError as e:
        print(f"Could not request results from Google Speech Recognition
service; {e}")
        return ""

def add_subtitle(frame, subtitle_text):
    font = cv2.FONT_HERSHEY_SIMPLEX
    bottom_left_corner = (10, 400)
    font_scale = 1
    font_color = (255, 255, 255)
    line_type = 2
    cv2.putText(frame, subtitle_text, bottom_left_corner, font,
font_scale, font_color, line_type)
    return frame

def main():
    cap = cv2.VideoCapture(0)
    if not cap.isOpened():
        print("Error: Could not open webcam.")
        return
    frame_width = int(cap.get(3))
    frame_height = int(cap.get(4))

    # Record video for 5 seconds
    fourcc = cv2.VideoWriter_fourcc(*'MJPG') # Try 'XVID' if 'MJPG'
doesn't work
    out = cv2.VideoWriter('output_video.avi', fourcc, 20.0, (frame_width,
frame_height), isColor=True)

    # Record audio separately
    audio_path = 'output_audio.wav'
    audio_format = pyaudio.paInt16
    audio_channels = 1
    audio_rate = 44100
    audio_chunk = 4096 # Increased buffer size

```

```

p = pyaudio.PyAudio()
stream = p.open(format=audio_format,
                channels=audio_channels,
                rate=audio_rate,
                input=True,
                frames_per_buffer=audio_chunk)
audio_frames = []
print("Recording...")
start_time = cv2.getTickCount()
while (cv2.getTickCount() - start_time) / cv2.getTickFrequency() < 5:
    ret, frame = cap.read()
    if not ret:
        print("Error: Failed to capture frame.")
        break
    out.write(frame)
    cv2.imshow('Recording...', frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
    # Record audio
    try:
        audio_data = stream.read(audio_chunk)
        audio_frames.append(audio_data)
    except IOError as ex:
        # Handle overflow error
        if ex[1] != pyaudio.paInputOverflowed:
            raise
print("Recording complete.")

# Stop recording and release resources
cap.release()
out.release()
cv2.destroyAllWindows()

# Save audio
stream.stop_stream()
stream.close()
p.terminate()
wf = wave.open(audio_path, 'wb')
wf.setnchannels(audio_channels)
wf.setsampwidth(pyaudio.PyAudio().get_sample_size(audio_format))
wf.setframerate(audio_rate)
wf.writeframes(b''.join(audio_frames))
wf.close()

# Transcribe recorded speech
subtitle_text = transcribe_speech(audio_path)

# Create a new video with subtitles
cap = cv2.VideoCapture('output_video.avi')

```

```

    out = cv2.VideoWriter('output_video_with_subtitles.avi', fourcc,
20.0, (frame_width, frame_height), isColor=True)
    while True:
        ret, frame = cap.read()
        if not ret:
            print("Error: Failed to capture frame.")
            break
        frame_with_subtitle = add_subtitle(frame, subtitle_text)
        out.write(frame_with_subtitle)
        cv2.imshow('Frame with Subtitles', frame_with_subtitle)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
    cap.release()
    out.release()
    cv2.destroyAllWindows()

    # Combine video with subtitles and audio
    video_clip = VideoFileClip('output_video_with_subtitles.avi')
    audio_clip = AudioFileClip(audio_path)
    final_clip = video_clip.set_audio(audio_clip)
    final_clip.write_videofile('final_output_video_with_audio.mp4',
codec='libx264', audio_codec='aac')
if __name__ == "__main__":
    main()

```

3.6. Управление громкостью с помощью жестов рук. MediaPipe

Разработана программа для управления громкостью компьютера при помощи распознавания жестов рук. При обработке потокового видео с веб-камеры распознается строение рук в кадре при помощи OpenCV и MediaPipe и на основе этих данных подаются команды операционной системе через PyAutoGUI.

MediaPipe – это фреймворк с открытым исходным кодом, представленный Google, который помогает создавать мультимодальные конвейеры машинного обучения. Эта структура может использоваться для различных приложений для обработки изображений и мультимедиа (особенно в виртуальной реальности), таких как обнаружение объектов, распознавание лиц, отслеживание рук, отслеживание нескольких рук и сегментация волос [58].

PyAutoGUI - это библиотека, которая позволяет эмулировать действия пользователя как с клавиатурой (нажатия клавиш, ввод текста, горячие клавиши), так и с мышкой (правая/левая кнопка, нажатие, перенос).

Описание работы программы

Импортируем нужные библиотеки и инициализируем модель для распознавания рук.

```
import cv2
import pyautogui
import mediapipe as mp
cap = cv2.VideoCapture(0)
mp_hands = mp.solutions.hands
hands = mp_hands.Hands(static_image_mode=False, max_num_hands=1,
                        min_detection_confidence=0.5,
                        min_tracking_confidence=0.5)
```

Инициализируем объект для отрисовки меток на кадре

```
mp_drawing = mp.solutions.drawing_utils
```

В цикле считываем кадр, обрабатываем его и пытаемся распознать руки на нем

```
ret, frame = cap.read()
image_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
results = hands.process(image_rgb)
```

В случае, если объект в кадре был распознан, накладываем метки на кадр

```
for hand_landmarks in results.multi_hand_landmarks:
    mp_drawing.draw_landmarks(
        frame, hand_landmarks, mp_hands.HAND_CONNECTIONS)
```

Распознавать жесты (указательный палец вверх или вниз) будем на основе двух точек: конца большого пальца и конца указательного пальца.

В случае, если координата по оси y указательного пальца больше координаты по оси y большого пальца – жест считается «вверх», иначе – «вниз»

```
index_finger_y =
hand_landmarks.landmark[mp_hands.HandLandmark.INDEX_FINGER_TIP].y
thumb_y = hand_landmarks.landmark[mp_hands.HandLandmark.THUMB_TIP].y
```

```

if index_finger_y < thumb_y:
    hand_gesture = 'pointing up'
elif index_finger_y > thumb_y:
    hand_gesture = 'pointing down'
else:
    hand_gesture = 'other'

```

На основе распознанного жеста отдаем команду PyAutoGui на увеличение или уменьшение громкости в системе

```

if hand_gesture == 'pointing up':
    pyautogui.press('volumeup')
elif hand_gesture == 'pointing down':
    pyautogui.press('volumedown')

```

Наконец, отображаем итоговый кадр с наложенными метками

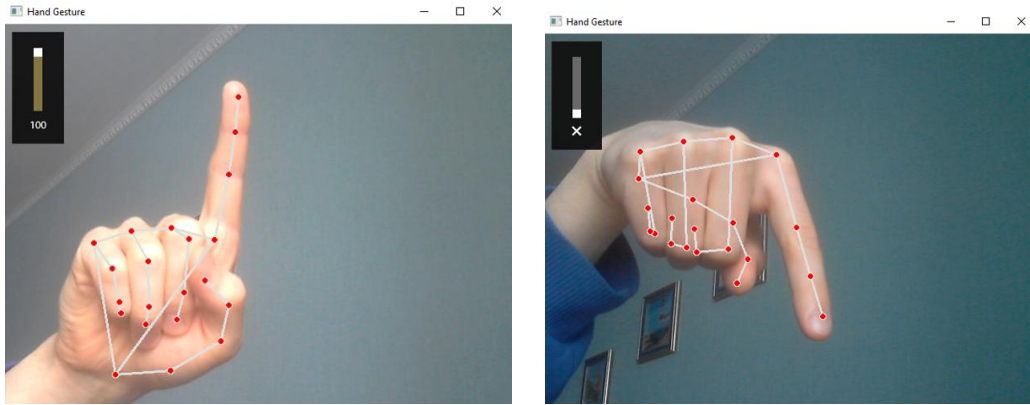
```
cv2.imshow('Hand Gesture', frame)
```

Таблица описаний функций:

Функция	Описание
cv2.VideoCapture()	Захват видеопотока с камеры или из видеофайла.
mp.solutions.hands	Модуль из библиотеки MediaPipe, который обеспечивает детекцию рук на изображениях или видео.
mp.solutions.drawing_utils	Предоставляет утилиты для рисования результатов детекции, например, точек и соединительных линий.
cv2.cvtColor()	Изменение цветового пространства изображения.
pyautogui.press()	Функция из библиотеки PyAutoGUI используется для эмуляции нажатия клавиши на клавиатуре
cv2.imshow()	Отображение изображения в окне
cv2.waitKey()	Ожидание нажатия клавиши в окне с изображением.
cv2.destroyAllWindows()	Закрытие всех открытых окон с изображениями

Результат работы программы

Как видно из скриншотов, все части руки отчетливо распознаются, при повороте указательного пальца вверх, громкость увеличивается, при указании вниз – уменьшается.



Текст программы

```

import cv2
import pyautogui
import mediapipe as mp
cap = cv2.VideoCapture(0)

mp_hands = mp.solutions.hands
hands = mp_hands.Hands(static_image_mode=False, max_num_hands=1,
                        min_detection_confidence=0.5,
                        min_tracking_confidence=0.5)

mp_drawing = mp.solutions.drawing_utils

while True:
    ret, frame = cap.read()
    if not ret:
        break

    image_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    results = hands.process(image_rgb)
    if results.multi_hand_landmarks:
        for hand_landmarks in results.multi_hand_landmarks:
            mp_drawing.draw_landmarks(
                frame, hand_landmarks, mp_hands.HAND_CONNECTIONS)
            index_finger_y =
hand_landmarks.landmark[mp_hands.HandLandmark.INDEX_FINGER_TIP].y
            thumb_y =
hand_landmarks.landmark[mp_hands.HandLandmark.THUMB_TIP].y

            if index_finger_y < thumb_y:
                hand_gesture = 'pointing up'
            elif index_finger_y > thumb_y:
                hand_gesture = 'pointing down'
            else:
                hand_gesture = 'other'

            if hand_gesture == 'pointing up':

```

```

        pyautogui.press('volumeup')
    elif hand_gesture == 'pointing down':
        pyautogui.press('volumedown')

    cv2.imshow('Hand Gesture', frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
cap.release()
cv2.destroyAllWindows()

```

3.7. Виртуальный гардероб. Dlib

Программа является примером реализации распознавания лиц и накладывания графических элементов на них в реальном времени с использованием библиотеки OpenCV и библиотеки Dlib для обнаружения лиц и ключевых точек лица [56].

Пошаговое описание работы программы

1. Импорт необходимых библиотек: cv2 (OpenCV), dlib, numpy и os.
2. Инициализация детектора лиц и детектора ключевых точек лица с помощью dlib. Для этого используются предварительно обученные модели: `get_frontal_face_detector()` для детекции лиц и `shape_predictor()` для определения ключевых точек лица.
3. Загрузка изображения очков из файла `glasses.png`.
4. Определение пути для сохранения скриншотов с улыбающимися лицами.
5. Создание директории для сохранения скриншотов, если таковая не существует.
6. Определение функции `overlay_glasses`, которая принимает кадр изображения и ключевые точки лица, вычисляет положение и ориентацию очков на лице и накладывает их на изображение.
7. Определение функции `process_image`, которая принимает кадр изображения, конвертирует его в оттенки серого (для обработки лица), обнаруживает лица на изображении, определяет их ключевые точки и накладывает очки. Также в этой функции проверяется наличие улыбки и при обнаружении улыбки делается скриншот с лицом и сохраняется в указанную директорию.
8. Запуск видеопотока с вебкамеры с помощью `cv2.VideoCapture`.

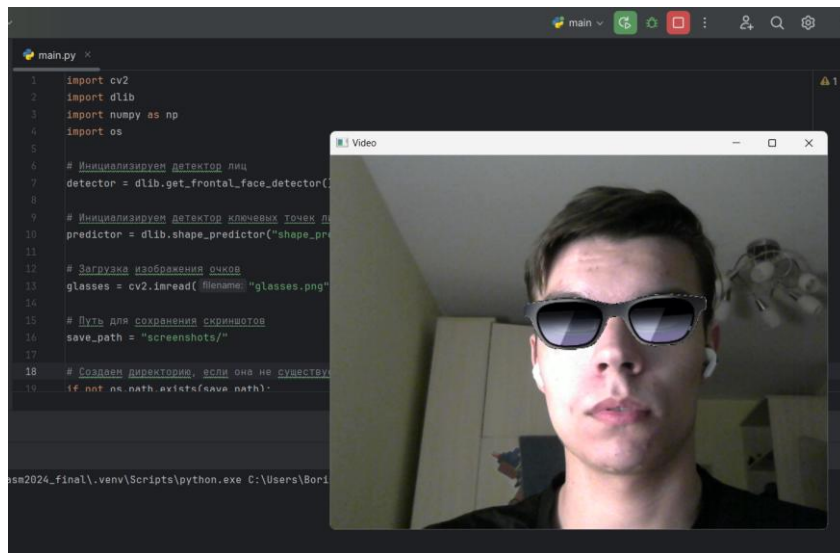
9. В бесконечном цикле считывается кадр с вебкамеры. Для каждого кадра вызывается функция `process_image`, которая обрабатывает изображение и отображает его с наложенными очками и проверкой на улыбку.
10. Пользователь может завершить программу, нажав клавишу 'q'.
11. После выхода из цикла освобождаются ресурсы видеопотока и закрываются все окна отображения.

Эта программа может быть полезна для развлекательных приложений или демонстраций, где требуется обнаружение лиц, а также для создания интерактивных фильтров или эффектов в реальном времени.

Использованные функции

Функция	Краткое описание
<code>overlay_glasses</code>	Эта функция используется для наложения очков на лицо в кадре изображения.
<code>process_image</code>	Эта функция обрабатывает кадр изображения с вебкамеры, обнаруживает лица на изображении. Она также проверяет наличие улыбки на лице и делает скриншот изображения в случае обнаружения улыбки.
<code>VideoCapture(cv2.CAP_DSHOW, 0)</code>	Эта строка кода открывает видеопоток с вебкамеры. Она использует функцию <code>cv2.VideoCapture()</code> из библиотеки <code>OpenCV</code> для создания объекта видеозахвата. В качестве аргументов передается <code>cv2.CAP_DSHOW</code> (для использования <code>DirectShow</code> в <code>Windows</code>) и <code>0</code> (для выбора первой доступной вебкамеры).
<code>video_capture.read()</code>	Эта строка кода считывает кадр из видеопотока.
<code>imshow()</code>	Эта строка кода отображает кадр изображения на экране. Функция <code>imshow()</code> из библиотеки <code>OpenCV</code> позволяет отображать изображение в окне с указанным названием.
<code>waitKey()</code>	Функция <code>waitKey()</code> из библиотеки <code>OpenCV</code> ожидает нажатия клавиши на клавиатуре в течение указанного времени (в миллисекундах)

Скриншот



Текст программы

```
import cv2
import dlib
import numpy as np
import os

# Инициализируем детектор лиц
detector = dlib.get_frontal_face_detector()

# Инициализируем детектор ключевых точек лица
predictor = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")

# Загрузка изображения очков
glasses = cv2.imread("glasses.png", cv2.IMREAD_UNCHANGED)

# Путь для сохранения скриншотов
save_path = "screenshots/"

# Создаем директорию, если она не существует
if not os.path.exists(save_path):
    os.makedirs(save_path)

# Функция для наложения очков на лицо
def overlay_glasses(frame, landmarks):
    # Определяем координаты глаз
    left_eye = (landmarks.part(17).x, landmarks.part(17).y)
    right_eye = (landmarks.part(26).x, landmarks.part(26).y)

    # Вычисляем ширину и высоту между глазами
    glasses_width = int(np.linalg.norm(np.array(left_eye) -
np.array(right_eye)) * 1.2)
    glasses_height = int(glasses_width * 0.8)
```

```

# Масштабируем и поворачиваем очки
resized_glasses = cv2.resize(glasses, (glasses_width,
glasses_height))
angle = np.arctan2(right_eye[1] - left_eye[1], right_eye[0] -
left_eye[0]) * 180 / np.pi
rotated_glasses = cv2.warpAffine(resized_glasses,

cv2.getRotationMatrix2D((glasses_width / 2, glasses_height / 2), -angle,
1),(glasses_width, glasses_height))

# Координаты верхнего левого угла очков
top_left = (int((left_eye[0] - glasses_width / 4) * 1.20),
int((left_eye[1] - glasses_height / 2) * 1.1))

# Наложение очков на изображение
for i in range(rotated_glasses.shape[0]):
    for j in range(rotated_glasses.shape[1]):
        if rotated_glasses[i, j, 3] != 0:
            frame[top_left[1] + i, top_left[0] + j, :] =
rotated_glasses[i, j, :3]

# Функция для обработки изображения с вебкамеры
def process_image(image):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    faces = detector(gray)
    for face in faces:
        landmarks = predictor(gray, face)
        overlay_glasses(image, landmarks)

# Проверяем, есть ли улыбка
mouth_distance = landmarks.parts()[67].x -
landmarks.parts()[60].x
if mouth_distance > 28: # Порог для определения улыбки
    # Делаем скриншот
    screenshot_path = os.path.join(save_path,
"smile_screenshot.jpg")
    cv2.imwrite(screenshot_path, image)
    print("Скриншот сделан и сохранен.")
return image

# Запускаем видеопоток с вебкамеры
video_capture = cv2.VideoCapture(cv2.CAP_DSHOW, 0)
while True:
    ret, frame = video_capture.read()
    if ret:
        frame = process_image(frame)
    cv2.imshow('Video', frame)
    # Для выхода нажмите 'q'
    if cv2.waitKey(1) & 0xFF == ord('q'):

```

```
break
video_capture.release()
cv2.destroyAllWindows()
```

3.8. Трекинг объектов и подсчет при пересечении линии. YOLOv9

Нейросеть YOLOv9 была выпущена в Феврале 2024 года и на момент выполнения работы в общем доступе можно использовать либо “YOLOv9c”, либо “YOLOv9e”. По сравнению с предыдущей моделью YOLOv9 получает более высокую точность при меньшем количестве параметров и меньшем количестве вычислений. Все модели YOLO (You Only Look Once) основаны на одноступенчатых детекторах. Изображение делится на квадратную сетку. Для каждой ячейки выводится вероятность определяемого класса. Ячейки, которые выше порогового значения, выбираются и используются для определения местоположения объекта на изображении [59].

Алгоритм

1. Указываем путь до исходного видео, путь до места, где будет храниться результат и путь до модели, которая используется в программе.
2. Вызываем функцию `process_video()`.
3. Далее для каждого кадра в видео выполняем одни и те же действия. Из всех обнаруживаемых объектов оставляем только объекты принадлежащих классу «машин» (имеют номер 2 в списке классов). Далее для кадра подсчитываем количество объектов, которые пересекли линию, добавляем в кадр информацию про данный объект, добавляем траекторию для объекта.
4. Обрабатываем следующий кадр.

Использованные функции

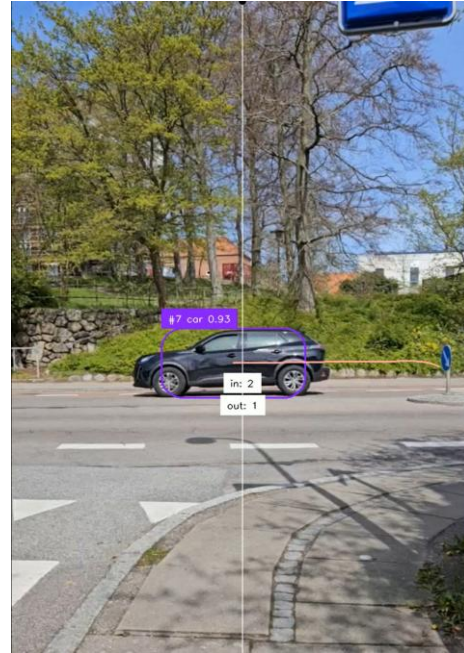
Функция	Краткое описание
---------	------------------

<code>process_video(source_path, target_path, callback)</code>	Обрабатывает видеофайл, применив функцию <code>callback</code> к каждому кадру и сохранив результат в назначенный файл.
<code>model.track(frame, tracker="bytetrack.yaml")</code>	Отслеживает объекты в кадре <code>frame</code> с помощью <code>ByteTrack</code> .
<code>sv.Detections.from_ultralytics(result)</code>	Создает экземпляр <code>Detections</code> на основе результатов вывода <code>YOLOv9</code> .
<code>tracker.update_with_detections(detections)</code>	Связывает текущие обнаружения с предыдущими наблюдениями.
<code>sv.LabelAnnotator(color)</code>	Создает объект, который позволяет добавлять текст для обнаруженных объектов.
<code>sv.RoundBoxAnnotator()</code>	Создает объект, который позволяет добавлять ограничивающие рамки для обнаруженных объектов.
<code>sv.TraceAnnotator()</code>	Создает объект, который позволяет добавлять траектории для обнаруженных объектов.
<code>sv.ByteTrack()</code>	Инициализирует <code>ByteTrack</code> объект.
<code>sv.LineZone(start, end)</code>	Создает линию по двум точкам.
<code>line_zone.trigger(detections)</code>	Считает количество объектов, которые пересекли линию.

Пример работы программы

Исходное видео:

Кадр из полученного видео:



Текст программы

```
from ultralytics import YOLO
import supervision as sv
import numpy as np

# Path for video
VIDEO_PATH = 'video path'
# Path for result
SAVE_PATH = 'save video path'
# Load the model YOLOv9
model = YOLO('model')
# Initialize all annotators
round_box_annotator = sv.RoundBoxAnnotator(
    color=sv.ColorPalette.ROBOFLOW,
    thickness=2
)
label_annotator = sv.LabelAnnotator(color=sv.ColorPalette.ROBOFLOW)
trace_annotator = sv.TraceAnnotator()
tracker = sv.ByteTrack()

# Line for counting cars
LINE_START = sv.Point(260, 0)
LINE_END = sv.Point(260, 1280)
line_zone = sv.LineZone(start=LINE_START, end=LINE_END)
line_annotator = sv.LineZoneAnnotator(
    thickness=1,
    text_thickness=1,
    text_scale=0.5
)
```

```

def callback(frame: np.ndarray, _: int) -> np.ndarray:
    results = model.track(frame, tracker="bytetrack.yaml")[0]
    detections = sv.Detections.from_ultralytics(results)
    detections = detections[detections.class_id == 2]
    detections = tracker.update_with_detections(detections)

    # make label for object
    labels = [
        f"#{tracker_id} {results.names[class_id]} {conf:0.2f}"
        for class_id, tracker_id, conf
        in zip(detections.class_id, detections.tracker_id,
detections.confidence)
    ]
    results_line = line_zone.trigger(detections=detections)
    annotated_frame = round_box_annotator.annotate(scene=frame.copy(),
detections=detections)
    annotated_frame = line_annotator.annotate(frame=annotated_frame,
line_counter=line_zone)
    annotated_frame = label_annotator.annotate(scene=annotated_frame,
detections=detections, labels=labels)
    annotated_frame = trace_annotator.annotate(scene=annotated_frame,
detections=detections)
    return annotated_frame

sv.process_video(source_path=VIDEO_PATH, target_path=SAVE_PATH,
callback=callback)

```

3.9. Поиск человека на видео по фотографии лица. Face_recognition

Обнаружение лица на видео — важная задача в области компьютерного зрения, находящая применение в системах безопасности, аналитике поведения, автоматической индексации видеоархивов и других сферах. В данной работе представлена программа, способная автоматически обнаруживать лица на видео с использованием библиотек cv2 и face_recognition [60]. Разработанная система представляет собой решение задачи обнаружения лиц на видео, что открывает широкие возможности для применения в реальных сценариях, требующих высокой точности и производительности обнаружения.

Использованные функции

Функция	Назначение
---------	------------

<code>face_recognition.load_image_file()</code>	Загрузка изображения лица для обработки.
<code>face_recognition.face_encodings()</code>	Извлечение кодировок лиц на изображении.
<code>face_recognition.compare_faces()</code>	Сравнение двух наборов кодировок лиц для определения совпадений.
<code>cv2.VideoCapture()</code>	Открытие видеофайла для обработки.
<code>cv2.VideoWriter()</code>	Создание видеофайла для записи результатов обработки.
<code>cv2.rectangle()</code>	Нанесение прямоугольника на изображение для обозначения лица.
<code>face_recognition.load_image_file()</code>	Загрузка изображения лица для обработки.

Алгоритм работы программы

1. Импортируются необходимые модули: `face_recognition`, `cv2`, `os`, `pickle`
2. Вызывается функция `train_by_photo` с передачей имени человека
3. Создается список из фотографий в папке `data`
4. Из каждой фотографии получаются кодировки лиц в виде векторов `face_recognition.face_encodings()` и добавляются в список векторов
5. Полученный список векторов сохраняется в `pickle` файл
6. Вызывается функция `detect_person` с передачей имени человека и названия видео файла
7. Происходит загрузка данных из файла с кодировками лиц
8. Создается захват видео из видеофайла с помощью `cv2.VideoCapture()` и файл для записи с помощью `cv2.VideoWriter()`
9. Файл обрабатывается покадрово в цикле, пока окно с выводом не закрыто
10. Находятся координаты лиц в кадре с помощью `face_recognition.face_locations()`
11. Для каждого лица находится его кодировка с помощью `face_recognition.face_encodings()`

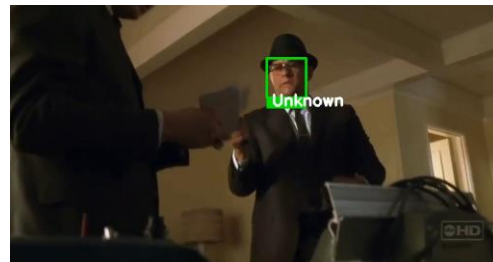
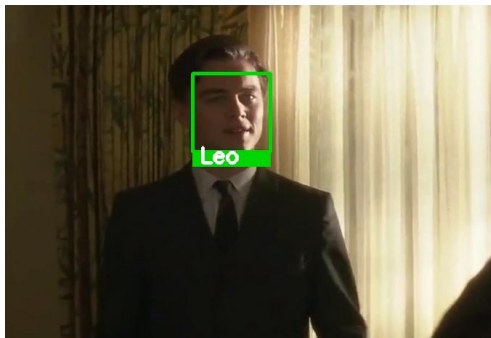
12. Все найденные лица выделяются прямоугольником с помощью `cv2.rectangle()`
13. Каждое лицо сравнивается с помощью `face_recognition.compare_faces()` с данными, полученными из фото. При положительном результате лицу подписывают имя, в обратном случае лицо помечается как `Unknown`.

Результат работы программы

Загруженные фотографии лица для поиска



Кадры полученного видео



Текст программы

```
import os
import pickle
import face_recognition
import cv2

def train_by_photo(name):
    encodings = []
    images = os.listdir("data")
    for (i, image) in enumerate(images):
        face_img = face_recognition.load_image_file(f"data/{image}")
        face_enc = face_recognition.face_encodings(face_img)[0]
        encodings.append(face_enc)
    data = {
```

```

        "name": name,
        "encodings": encodings
    }
    with open(f"{name}_encodings.pickle", "wb") as file:
        file.write(pickle.dumps(data))

def detect_person(person_name, video_file_name):
    data = pickle.loads(open(person_name + "_encodings.pickle",
"rb").read())
    video = cv2.VideoCapture(video_file_name)
    frame_width = int(video.get(cv2.CAP_PROP_FRAME_WIDTH))
    frame_height = int(video.get(cv2.CAP_PROP_FRAME_HEIGHT))
    out = cv2.VideoWriter('output.avi', 0, 20.0, (frame_width,
frame_height))
    while video.isOpened():

        ret, image = video.read()

        locations = face_recognition.face_locations(image, model="fog")
        encodings = face_recognition.face_encodings(image, locations)

        for face_encoding, face_location in zip(encodings, locations):
            result = face_recognition.compare_faces(data["encodings"],
face_encoding)
            match = "Unknown"
            if True in result:
                match = data["name"]
            left_top = (face_location[3], face_location[0])
            right_bottom = (face_location[1], face_location[2])
            color = [0, 255, 0]
            cv2.rectangle(image, left_top, right_bottom, color, 4)

            left_bottom = (face_location[3], face_location[2])
            right_bottom = (face_location[1], face_location[2] + 20)
            cv2.rectangle(image, left_bottom, right_bottom, color,
cv2.FILLED)
            cv2.putText(
                image,
                match,
                (face_location[3] + 10, face_location[2] + 15),
                cv2.FONT_HERSHEY_SIMPLEX,
                1,
                (255, 255, 255),
                4
            )
            out.write(image)
            cv2.imshow("detect_person_in_video is running", image)

    video.release()

```

```

out.release()

def main():
    train_by_photo("Leo")
    detect_person("Leo", "input.mp4")

if __name__ == '__main__':
    main()

```

3.10. Создание видео по картинке. Replicate и Stable Video Diffusion

Stable Video Diffusion – нейросеть, которая умеет генерировать короткие видео по картинке. Это мощный инструмент, предназначенный для широкого спектра видеоприложений в сфере медиа, развлечений, образования и маркетинга. Он позволяет пользователям превращать текст и изображения в яркие сцены, превращая идеи в реальные видео. Модель разработана Stability AI.

В основе Stable Video Diffusion лежит сложный процесс, использующий диффузионные модели (DM), управление без классификаторов и архитектуру базовой модели, специально разработанную для создания видео [47, 48].

Replicate – сервис, позволяющий запускать работу моделей нейросетей. Для запуска работы необходим свой токен, изображение и параметры для генерации видео.

Алгоритм работы программы

1. Создание изображения с помощью захвата видеокамеры

```

cap = cv2.VideoCapture(0);
cap.set(cv2.CAP_PROP_FRAME_WIDTH, value: 1280)
cap.set(cv2.CAP_PROP_FRAME_WIDTH, value: 720)
ret, frame = cap.read()
cv2.imwrite(filename: 'frames2.png', frame)
cap.release()

```

2. Авторизация на платформе Replicate с помощью токена, полученного при регистрации на платформе через Github

```
os.environ["REPLICATE_API_TOKEN"] =  
replicate = replicate.Client(api_token=os.environ["REPLICATE_API_TOKEN"])
```

3. Выбор нейросети для создания видео, в нашем случае выбираем Stable Video Diffusion

```
VERSION = "stability-ai/stable-video-diffusion:3f0457e4619daac51203dedb472816fd4af51f3149fa7a9e0b5ffcf1b8172438"
```

4. Загрузка изображения из пункта 1

```
PATH_TO_IMAGE = "frames2.png"
```

5. Запуск работы нейросети с указанием необходимых параметров

```
output = replicate.run(  
    VERSION,  
    input={  
        "cong_aug": 0.01,  
        "decoding_t": 7,  
        "input_image": open(PATH_TO_IMAGE, "rb"),  
        "video_length": "25_frames_with_svd_xt",  
        "sizing_strategy": "maintain_aspect_ratio",  
        "motion_bucket_id": 33,  
        "frames_per_second": 6  
    }  
)  
print(output)
```

6. Получение ссылки на сгенерированное видео

```
print(output)
```

Таблиц использованных функций

Функция	Описание
cv2.VideoCapture	Захват камеры
set	Задание параметров изображения
read	Чтение изображения
cv2.imwrite	Запись изображения в файл
release	Освобождение камеры
replicate.Client	Авторизация в сервисе replicate
replicate.run	Запуск работы нейросети

Параметры функции replicate.run:

1. VERSION - определяет какую модель необходимо запускать;
2. input;
 - 2.1. cond_aug - количество шума для добавления во входное изображение;
 - 2.2. decoding_t - количество кадров для декодирования за раз;

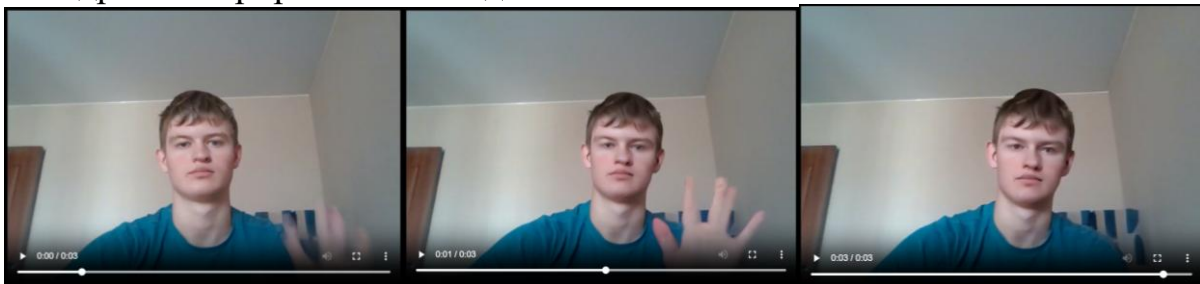
- 2.3. `input_image` - изображение, которое необходимо оживить;
- 2.4. `video_length` - длина видео;
- 2.5. `frames_per_second` - этот параметр указывает сколько кадров в секунду показывать;
- 2.6. `sizing_strategy` - это соотношение сторон;
- 2.7. `motion_bucket_id` - насколько увеличивать общее движение в сгенерированном видео.

Результат

Исходное изображение:



Кадры сгенерированного видео:



Текст программы

```
import cv2
import os
import replicate

cap = cv2.VideoCapture(0);
cap.set(cv2.CAP_PROP_FRAME_WIDTH, 1280)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 720)
ret, frame = cap.read()
cv2.imwrite('frames2.png', frame)
cap.release()
os.environ["REPLICATE_API_TOKEN"] = "<insert_your_token>"
replicate = replicate.Client(api_token=os.environ["REPLICATE_API_TOKEN"])
```

```

VERSION = "stability-ai/stable-video-
diffusion:3f0457e4619daac51203dedb472816fd4af51f3149fa7a9e0b5ffcf1b817243
8"
PATH_TO_IMAGE = "frames2.png"

output = replicate.run(
    VERSION,
    input={
        "cong_aug": 0.01,
        "decoding_t": 7,
        "input_image": open(PATH_TO_IMAGE, "rb"),
        "video_length": "25_frames_with_svd_xt",
        "sizing_strategy": "maintain_aspect_ratio",
        "motion_bucket_id": 33,
        "frames_per_second": 6
    }
)
print(output)

```

3.11. Распознавание жестов. MediaPipe

Используются библиотеки OpenCV и MediaPipe. В MediaPipe реализовано распознавание рук [58]. Hand Landmarker использует модель объединяющую в себе две модели:

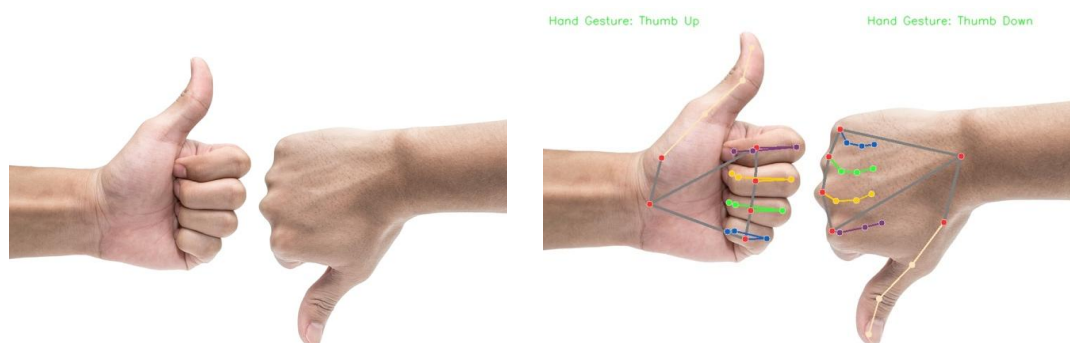
1. Palm detection model: Модель обнаружения ладони находит руки в входном изображении
2. Hand landmarks detection model: Модель ориентира руки обнаруживает локализацию ключевой точки 21 координаты руки в обнаруженных областях руки. Модель была обучена примерно на 30 000 реальных изображений, а также на нескольких визуализированных синтетических ручных моделях, наложенных на различные фоны.



Использованные функции

Название	Краткое описание
mp_hands.Hands()	Создает объект Hands для обнаружения и отслеживания рук в изображениях и видео.
cv2.imread()	Считывает изображение из файла, указанного по заданному пути. Возвращает объект numpy.ndarray, представляющий изображение, где каждый элемент массива соответствует пикселю изображения.
cv2.cvtColor()	Преобразует цветовое пространство изображения. Поддерживаются различные цветовые пространства, такие как BGR (по умолчанию), RGB, HSV, LAB и т.д.
hands.process()	Обрабатывает входное изображение или видеокادر. Использует алгоритмы MediaPipe для обнаружения рук, определения их положения и ориентации, а также извлечения ключевых точек на каждой руке.
mp_drawing.draw_landmarks()	Рисует ключевые точки и соединения между ними на изображении.
cv2.putText()	Добавляет текст на изображение
cv2.imwrite()	Записывает изображение в файл, указанный по заданному пути.

Пример работы программы



Текст программы

```
import os
import math
import cv2
import mediapipe as mp

# Инициализация MediaPipe Hands и утилит для рисования
mp_hands = mp.solutions.hands
mp_drawing = mp.solutions.drawing_utils
mp_drawing_styles = mp.solutions.drawing_styles

# Функция для распознавания жестов на основе координат ключевых точек
def recognize_gesture(landmarks):
    # Извлечение ключевых точек для запястья и кончиков пальцев
```

```

wrist = landmarks.landmark[mp_hands.HandLandmark.WRIST]
thumb_tip = landmarks.landmark[mp_hands.HandLandmark.THUMB_TIP]
index_finger_tip =
landmarks.landmark[mp_hands.HandLandmark.INDEX_FINGER_TIP]
middle_finger_tip =
landmarks.landmark[mp_hands.HandLandmark.MIDDLE_FINGER_TIP]
ring_finger_tip =
landmarks.landmark[mp_hands.HandLandmark.RING_FINGER_TIP]
pinky_tip = landmarks.landmark[mp_hands.HandLandmark.PINKY_TIP]

# Извлечение точек PIP (проксимальный межфаланговый сустав)
index_finger_pip =
landmarks.landmark[mp_hands.HandLandmark.INDEX_FINGER_PIP]
middle_finger_pip =
landmarks.landmark[mp_hands.HandLandmark.MIDDLE_FINGER_PIP]
ring_finger_pip =
landmarks.landmark[mp_hands.HandLandmark.RING_FINGER_PIP]
pinky_pip = landmarks.landmark[mp_hands.HandLandmark.PINKY_PIP]

# Расчет горизонтальных расстояний от запястья до кончиков пальцев и
PIP суставов
index_tip_horizontal_distance = abs(index_finger_tip.x - wrist.x)
index_pip_horizontal_distance = abs(index_finger_pip.x - wrist.x)
middle_tip_horizontal_distance = abs(middle_finger_tip.x - wrist.x)
middle_pip_horizontal_distance = abs(middle_finger_pip.x - wrist.x)
ring_tip_horizontal_distance = abs(ring_finger_tip.x - wrist.x)
ring_pip_horizontal_distance = abs(ring_finger_pip.x - wrist.x)
pinky_tip_horizontal_distance = abs(pinky_tip.x - wrist.x)
pinky_pip_horizontal_distance = abs(pinky_pip.x - wrist.x)

# Расчет расстояния между кончиком большого пальца и кончиком
указательного пальца
distance_thumb_index = math.sqrt((thumb_tip.x - index_finger_tip.x)
** 2 + (thumb_tip.y - index_finger_tip.y) ** 2)

# Распознавание жестов на основе определенных условий
if (thumb_tip.y < wrist.y and index_tip_horizontal_distance <
index_pip_horizontal_distance
and middle_tip_horizontal_distance <
middle_pip_horizontal_distance
and ring_tip_horizontal_distance <
ring_pip_horizontal_distance
and pinky_tip_horizontal_distance <
pinky_pip_horizontal_distance):
return "Thumb Up" # Палец вверх
elif (thumb_tip.y > wrist.y and index_tip_horizontal_distance <
index_pip_horizontal_distance
and middle_tip_horizontal_distance <
middle_pip_horizontal_distance

```

```

        and ring_tip_horizontal_distance < ring_pip_horizontal_distance
        and pinky_tip_horizontal_distance <
pinky_pip_horizontal_distance):
    return "Thumb Down" # Палец вниз
    elif (thumb_tip.y < ring_finger_tip.y < pinky_tip.y) and
(index_finger_tip.y < thumb_tip.y) and (
    middle_finger_tip.y < thumb_tip.y):
    return "V Sign" # Жест "V"
    elif (distance_thumb_index < 0.1 and middle_finger_tip.y <
thumb_tip.y and ring_finger_tip.y < thumb_tip.y
    and pinky_tip.y < thumb_tip.y):
    return "Ok" # Жест "OK"
else:
    return "Unknown" # Неизвестный жест

# Определение директорий для входных изображений и выходных файлов
IMAGES_DIR = "images"
OUTPUT_DIR = "output"

# Создание выходной директории, если она не существует
os.makedirs(OUTPUT_DIR, exist_ok=True)

# Инициализация решения MediaPipe Hands
with mp_hands.Hands() as hands:
    # Перебор всех файлов в директории с изображениями
    for filename in os.listdir(IMAGES_DIR):
        image_path = os.path.join(IMAGES_DIR, filename)
        image = cv2.imread(image_path)

        # Пропуск файлов, которые не удастся прочитать как изображения
        if image is None:
            print(f"Unable to read image: {image_path}")
            continue

        # Преобразование изображения в RGB
        image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

        # Обработка изображения для обнаружения рук
        results = hands.process(image_rgb)

        # Рисование ключевых точек и распознавание жестов, если руки
обнаружены
        if results.multi_hand_landmarks:
            for hand_landmarks in results.multi_hand_landmarks:
                # Рисование ключевых точек на изображении
                mp_drawing.draw_landmarks(
                    image, hand_landmarks, mp_hands.HAND_CONNECTIONS,
                    mp_drawing_styles.get_default_hand_landmarks_style(),

```

```

mp_drawing_styles.get_default_hand_connections_style()
    )

    # Перебор обнаруженных рук и классификация жестов
    for hand_landmarks, hand_info in
zip(results.multi_hand_landmarks, results.multi_handedness):
    for classification in hand_info.classification:
        label = classification.label

    # Распознавание жеста руки
    gesture_label = recognize_gesture(hand_landmarks)

    # Добавление текста с распознанным жестом на изображение
    cv2.putText(image, f"Hand Gesture: {gesture_label}",
                (10 if label == "Right" else image.shape[1] -
400, 30),
                cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 0),
1, cv2.LINE_AA)

    # Сохранение обработанного изображения в выходную директорию
    output_path = os.path.join(OUTPUT_DIR, filename)
    cv2.imwrite(output_path, image)

print("Processing completed")

```

3.12. Распознавания объектов с веб-камеры. ResNet50

Программа использует предварительно обученную модель ResNet50 для распознавания объектов на видео, сделанном с веб-камеры. Она захватывает видео с веб-камеры с помощью OpenCV, затем для каждого кадра видео использует модель ResNet50 для распознавания объектов на этом кадре. Результаты распознавания выводятся на экран с помощью OpenCV, где для каждого предсказанного объекта отображается метка класса и соответствующая вероятность.

ResNet50 - это глубокая нейронная сеть, которая была разработана и обучена командой исследователей в Microsoft в рамках проекта ResNet (Residual Networks). Matplotlib - это библиотека для создания статических, интерактивных и анимированных визуализаций в языке

программирования Python. Она предоставляет широкий спектр функций для создания различных видов графиков, диаграмм и изображений.

Алгоритм работы

1. Инициализация модели:

- Загрузка предварительно обученной модели ResNet50 с весами, обученными на датасете ImageNet.

2. Функция распознавания объектов (`recognize_image`):

- Принимает на вход изображение.
- Изменяет размер изображения до 224x224 пикселей (стандартный размер входных данных для модели ResNet50).
- Добавляет дополнительное измерение к массиву изображения для соответствия формату, ожидаемому моделью.
- Предобрабатывает изображение, чтобы оно соответствовало формату, принятому в модели.
- Передает предварительно обработанное изображение в модель ResNet50 для предсказания.
- Декодирует предсказания и возвращает три наиболее вероятных класса объектов.

3. Захват видео с веб-камеры:

- Инициализация объекта захвата видео с помощью OpenCV.

4. Бесконечный цикл обработки кадров:

- В цикле считывается каждый кадр видео с веб-камеры.
- Для каждого кадра вызывается функция `recognize_image`, чтобы распознать объекты на этом кадре.
- Результаты распознавания отображаются на изображении с помощью OpenCV: для каждого объекта выводится метка класса и вероятность.
- Обработанный кадр с результатами распознавания отображается с помощью Matplotlib.
- Проверяется нажатие клавиши 'q', и если оно обнаружено, цикл завершается.

5. Освобождение ресурсов:

- После завершения работы освобождаются ресурсы, связанные с захватом видео с веб-камеры.
- Закрывается окно Matplotlib.

Используемые функции

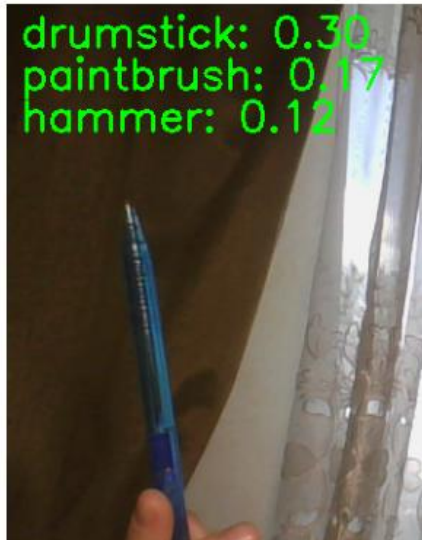
Используемые функции OpenCV	
<code>cv2.VideoCapture()</code>	Создает объект захвата видео с веб-камеры. В данном случае, используется для захвата видеопотока с камеры компьютера.
<code>cap.read()</code>	Считывает следующий кадр из видеопотока. Возвращает два значения: флаг <code>ret</code> , указывающий на успешность чтения кадра, и сам кадр <code>frame</code> .
<code>cv2.resize()</code>	Изменяет размер изображения до указанных размеров. В программе используется для изменения размера кадра до 224x224 пикселей, так как это стандартный размер для входных данных модели ResNet50.
<code>cv2.putText()</code>	Добавляет текст на изображение. Используется для отображения меток классов и вероятностей распознанных объектов на изображении.
<code>cv2.waitKey()</code>	Добавляет текст на изображение. Используется для отображения меток классов и вероятностей распознанных объектов на изображении.
<code>cap.release()</code>	Освобождает ресурсы, связанные с объектом захвата видео. В данном случае, используется для освобождения ресурсов, выделенных для захвата видео с веб-камеры после завершения работы программы.

Используемые функции ResNet50	
<code>ResNet50</code>	Функция <code>ResNet50(weights='imagenet')</code> загружает предварительно обученную модель ResNet50 с весами, обученными на датасете ImageNet. В данном случае, она используется для классификации объектов на изображении.
<code>preprocess_input</code>	Функция <code>preprocess_input</code> используется для предварительной обработки входных изображений перед передачей их в модель. В данном случае, она приводит входное изображение к формату, который ожидает модель ResNet50.

decode_predictions	Функция decode_predictions используется для декодирования предсказаний, полученных из модели. Возвращает список кортежей, каждый из которых содержит класс объекта, оценку уверенности модели и строковое описание класса. В данном случае, используется для декодирования предсказанных классов объектов на изображении.
--------------------	---

Используемые функции Matplotlib	
plt.ion()	Эта функция включает интерактивный режим Matplotlib, что позволяет обновлять графики в реальном времени.
plt.subplots()	Создает новое окно с одним рисунком и одним подзаголовком. В данном случае, создается окно для отображения изображений с результатами распознавания объектов.
ax.imshow()	Отображает изображение на созданном рисунке. В данной программе, используется для отображения кадра видео с веб-камеры, на котором распознаются объекты.
plt.axis()	Устанавливает параметры осей координат. В данном случае, функция используется для скрытия осей координат на изображении с результатами распознавания объектов.
plt.draw() plt.pause()	Эти функции используются для отображения изображения с результатами распознавания объектов в реальном времени. Функция plt.draw() обновляет окно с изображением, а plt.pause() приостанавливает выполнение программы на небольшой промежуток времени, чтобы графические обновления могли быть отображены.
plt.close()	Эта функция закрывает окно с изображением. В данной программе, используется для закрытия окна Matplotlib после завершения работы программы.

Результат



Текст программы

```
import cv2
import numpy as np
import tensorflow as tf
from tensorflow.keras.applications.resnet50 import ResNet50,
preprocess_input, decode_predictions
import matplotlib.pyplot as plt

model = ResNet50(weights='imagenet')

def recognize_image(image):
    img_resized = cv2.resize(image, (224, 224))
    img_array = np.expand_dims(img_resized, axis=0)
    img_preprocessed = preprocess_input(img_array)

    predictions = model.predict(img_preprocessed)
    decoded_predictions = decode_predictions(predictions, top=3)[0]

    return decoded_predictions

cap = cv2.VideoCapture(0)

plt.ion()
fig, ax = plt.subplots()

while True:
    ret, frame = cap.read()
    if not ret:
        break

    predictions = recognize_image(frame)

    for i, (imagenet_id, label, score) in enumerate(predictions):
```

```

        text = f"{label}: {score:.2f}"
        cv2.putText(frame, text, (10, 30 + i * 30),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)

    ax.imshow(cv2.cvtColor(frame, cv2.COLOR_BGR2RGB))
    plt.axis('off')
    plt.draw()
    plt.pause(0.001)
    ax.clear()

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
plt.close()

```

3.13. Распознавание лиц с веб-камеры. Face_recognition

Face_recognition — это библиотека Python, предназначенная для распознавания лиц [60]. Она использует нейросетевые технологии для обнаружения и идентификации лиц на изображениях и в видеопотоке. Библиотека построена на основе моделей глубокого обучения, обеспечивающих высокую точность и производительность. Основные возможности:

1. Обнаружение лиц: Использует алгоритмы Histogram of Oriented Gradients (HOG) или Convolutional Neural Networks (CNN) для нахождения лиц на изображениях.
2. Выравнивание лиц: Стандартизирует позиции лиц для повышения точности распознавания.
3. Кодирование лиц: Преобразует лица в 128-мерные векторы признаков с использованием модели ResNet-34.
4. Сравнение лиц: Определяет, совпадают ли лица, сравнивая евклидово расстояние между их векторами признаков.

Алгоритм

1. Загрузка и кодирование изображения известного лица.
2. Инициализация видеопотока с веб-камеры.

3. Обработка каждого кадра видеопотока для обнаружения и распознавания лиц.
4. Отображение результата распознавания на видеопотоке в режиме реального времени.

Использованные функции

Функция	Описание
<code>face_recognition.load_image_file(file)</code>	Загружает изображение из файла. Возвращает изображение в виде массива NumPy.
<code>face_recognition.face_encodings(image)</code>	Извлекает кодировки лиц из изображения. Возвращает список 128-мерных векторов признаков для каждого лица.
<code>face_recognition.face_locations(image)</code>	Обнаруживает местоположения лиц на изображении. Возвращает список координат для каждого найденного лица.
<code>face_recognition.compare_faces(known_encodings, unknown_encoding)</code>	Сравнивает кодировку неизвестного лица с кодировками известных лиц. Возвращает список булевых значений, указывающих на совпадение.

Описание работы программы

- Первым шагом является загрузка изображения известного лица и извлечение его кодировки. Это позволяет системе сравнивать лица из видеопотока с этим известным лицом.

```
import face_recognition
import cv2

# Загрузка изображения с известным лицом
known_image =
face_recognition.load_image_file("C:\\Users\\vetos\\OneDrive\\Рабочий
стол\\Лицо1.png")
known_face_encoding = face_recognition.face_encodings(known_image)[0]
```

- Инициализация видеопотока с веб-камеры. В случае наличия нескольких камер можно указать нужный индекс камеры.

```
# Открытие видеопотока с веб-камеры (если у вас есть несколько камер,
укажите нужный индекс)
video_capture = cv2.VideoCapture(0)
Обработка видеопотока
```

- Программа выполняет следующие шаги для каждого кадра видеопотока:

1. Захват кадра.
2. Обнаружение лиц на кадре.
3. Кодирование обнаруженных лиц.
4. Сравнение кодировок обнаруженных лиц с кодировкой известного лица.
5. Отображение результата сравнения на кадре.

```
while True:
    # Захват кадра с видеопотока
    ret, frame = video_capture.read()

    # Поиск лиц на кадре
    face_locations = face_recognition.face_locations(frame)

    if len(face_locations) > 0:
        # Извлечение кодировок лиц с текущего кадра
        unknown_face_encodings = face_recognition.face_encodings(frame,
        face_locations)

        # Сравнение лиц с известным лицом
        results = face_recognition.compare_faces([known_face_encoding],
        unknown_face_encodings[0])

        # Отображение результата на кадре
        if results[0]:
            label = "Face detected!"
        else:
            label = "Face not detected!"

        cv2.putText(frame, label, (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1,
        (0, 255, 0), 2)

    # Отображение текущего кадра
    cv2.imshow('Video', frame)

    # Ожидание нажатия клавиши 'q' для выхода из цикла
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
```

- **Завершение работы**

После завершения работы (нажатие клавиши 'q') освобождаются ресурсы видеопотока и закрываются все окна OpenCV.

```
# Освобождение ресурсов видеопотока и закрытие всех окон OpenCV
video_capture.release()
cv2.destroyAllWindows()
```

Результаты

Лицо для обработки:



Работа программы:



Текст программы

```
import face_recognition
import cv2

# Загрузка изображения с известным лицом
known_image =
face_recognition.load_image_file("C:\\Users\\vetos\\OneDrive\\Рабочий
стол\\Лицо1.png")
known_face_encoding = face_recognition.face_encodings(known_image)[0]

# Открытие видеопотока с веб-камеры (если у вас есть несколько камер,
укажите нужный индекс)
video_capture = cv2.VideoCapture(0)

while True:
# Захват кадра с видеопотока
ret, frame = video_capture.read()

# Поиск лиц на кадре
face_locations = face_recognition.face_locations(frame)

if len(face_locations) > 0:
# Извлечение кодировок лиц с текущего кадра
unknown_face_encodings = face_recognition.face_encodings(frame,
face_locations)

# Сравнение лиц с известным лицом
results = face_recognition.compare_faces([known_face_encoding],
unknown_face_encodings[0])

# Отображение результата на кадре
if results[0]:
label = "Face detected!"
else:
```

```

label = "Face not detected!"

cv2.putText(frame, label, (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255,
0), 2)

# Отображение текущего кадра
cv2.imshow('Video', frame)

# Ожидание нажатия клавиши 'q' для выхода из цикла
if cv2.waitKey(1) & 0xFF == ord('q'):
break

# Освобождение ресурсов видеопотока и закрытие всех окон OpenCV
video_capture.release()
cv2.destroyAllWindows()

```

3.14. Определение объектов на видео. YOLO

YOLO (You Only Look Once) - это алгоритм обнаружения объектов на изображениях, который основывается на глубоком обучении и использует сверточные нейронные сети. Он был разработан для решения задачи обнаружения объектов в реальном времени с высокой скоростью. Перечислим основные особенности YOLO.

Одна сеть для всего изображения: YOLO анализирует изображение только один раз с помощью сверточной нейронной сети и выводит ограничивающие рамки и вероятности классов вместе. Это позволяет YOLO работать быстрее по сравнению с традиционными алгоритмами, которые используют множество регионов интереса (Region Proposal Network) и применяют классификатор к каждому из них отдельно.

Высокая скорость: YOLO был разработан с учетом скорости работы в реальном времени. Благодаря одновременному обнаружению всех объектов на изображении, а не попиксельно, он может достигать высокой скорости работы.

Высокая точность: Несмотря на высокую скорость работы, YOLO демонстрирует высокую точность обнаружения объектов.

Масштабируемость: YOLO позволяет обнаруживать объекты разных размеров и типов на изображениях, что делает его универсальным и масштабируемым для различных приложений, включая обнаружение объектов на видео, в автономных автомобилях, системах наблюдения и т. д.

Версии YOLO, такие как YOLOv3 (используемая в данной программе) и YOLOv4, улучшают и расширяют базовый алгоритм, чтобы повысить его точность и скорость работы, а также добавить новые возможности, такие как обнаружение ключевых точек и улучшенное обучение на больших датасетах.

Ход работы

Для использования YOLO (You Only Look Once) с OpenCV необходимо выполнить несколько шагов

Шаг 1: Установка необходимых библиотек

1) Установите OpenCV:
`pip install opencv-python opencv-python-headless`
2) Установите NumPy:
`pip install numpy`

Шаг 2: Загрузка YOLO файлов

1) Файлы конфигурации и весов:
`yolov3.cfg`: Конфигурационный файл YOLOv3.
`yolov3.weights`: Весовые параметры обученной модели YOLOv3.
`coco.names`: Файл с метками классов (объектов), которые может распознавать модель.

Шаг 3: Написание кода для обработки изображения/видео с помощью YOLO

3.1. Инициализация YOLO

1) Загрузка конфигурации и весов сети:
`config_path = "yolov3.cfg" # Путь к файлу конфигурации YOLO`
`weights_path = "yolov3.weights" # Путь к файлу весов YOLO`
2) Загрузка меток классов объектов:
`labels = open("coco.names").read().strip().split("\n") # Загрузка меток объектов`
3) Генерация случайных цветов для каждого класса:

```

colors = np.random.randint(0, 255, size=(len(labels), 3), dtype="uint8")
# Генерация цветов для визуализации
4) Загрузка сети YOLO с помощью OpenCV:
net = cv2.dnn.readNetFromDarknet(config_path, weights_path) # Загрузка
сети YOLO

```

3.2. Обработка изображения/видео

```

1) Чтение видеофайла:
cap = cv2.VideoCapture("input_video.mp4") # Открытие видеофайла
2) Получение параметров видео (ширина, высота, количество кадров в
секунду):
frame_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
frame_height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
fps = int(cap.get(cv2.CAP_PROP_FPS))
3) Инициализация объекта для записи видео:
fourcc = cv2.VideoWriter_fourcc(*'mp4v') # Кодек для записи видео
out = cv2.VideoWriter("output_video.mp4", fourcc, fps, (frame_width,
frame_height))
4) Цикл обработки каждого кадра видео:
while True:
    ret, frame = cap.read() # Чтение очередного кадра
    if not ret:
        break # Прерывание цикла, если кадры закончились
5) Преобразование изображения в blob (формат, пригодный для обработки
нейросетью):
blob = cv2.dnn.blobFromImage(frame, 1 / 255.0, (416, 416), swapRB=True,
crop=False)
net.setInput(blob)
6) Получение имен слоев сети YOLO:
ln = net.getLayerNames()
try:
    ln = [ln[i[0] - 1] for i in net.getUnconnectedOutLayers()]
except IndexError:
    ln = [ln[i - 1] for i in net.getUnconnectedOutLayers()]
7) Прямой проход (inference) через сеть для получения предсказаний:
layer_outputs = net.forward(ln)
8) Извлечение информации о детектированных объектах:
boxes, confidences, class_ids = [], [], []
for output in layer_outputs:
    for detection in output:
        scores = detection[5:]
        class_id = np.argmax(scores)
        confidence = scores[class_id]
        if confidence > CONFIDENCE:
            box = detection[:4] * np.array([w, h, w, h])
            (centerX, centerY, width, height) = box.astype("int")
            x = int(centerX - (width / 2))
            y = int(centerY - (height / 2))

```

```
boxes.append([x, y, int(width), int(height)])
confidences.append(float(confidence))
class_ids.append(class_id)
```

9) Применение non-maxima suppression для удаления лишних прямоугольников:

```
idxs = cv2.dnn.NMSBoxes(boxes, confidences, SCORE_THRESHOLD,
IOU_THRESHOLD)
```

10) Рисование прямоугольников и меток на кадре:

```
if len(idxs) > 0:
    for i in idxs.flatten():
        x, y = boxes[i][0], boxes[i][1]
        w, h = boxes[i][2], boxes[i][3]
        color = [int(c) for c in colors[class_ids[i]]]
        cv2.rectangle(frame, (x, y), (x + w, y + h), color=color,
thickness=1)
        text = f"{labels[class_ids[i]]}: {confidences[i]:.2f}"
        (text_width, text_height) = cv2.getTextSize(text,
cv2.FONT_HERSHEY_SIMPLEX, fontScale=1, thickness=1)[0]
        text_offset_x = x
        text_offset_y = y - 5
        box_coords = ((text_offset_x, text_offset_y), (text_offset_x +
text_width + 2, text_offset_y - text_height))
        overlay = frame.copy()
        cv2.rectangle(overlay, box_coords[0], box_coords[1], color=color,
thickness=cv2.FILLED)
        frame = cv2.addWeighted(overlay, 0.6, frame, 0.4, 0)
        cv2.putText(frame, text, (x, y - 5), cv2.FONT_HERSHEY_SIMPLEX,
fontScale=1, color=(0, 0, 0), thickness=1)
```

11) Запись обработанного кадра в выходное видео:

```
out.write(frame)
```

12) Освобождение ресурсов:

```
cap.release()
out.release()
cv2.destroyAllWindows()
```

Таблица используемых функций

<pre>cv2.imread(path[, flags])</pre>	<p>Функция загружает изображение из файла. path: путь к файлу изображения. flags (опционально): флаги, указывающие на способ чтения изображения (например, cv2.IMREAD_COLOR, cv2.IMREAD_GRAYSCALE). Возвращает загруженное изображение в виде массива numpy.</p>
<pre>cv2.dnn.readNetFrom Darknet(config, model)</pre>	<p>Функция загружает модель YOLO из файлов конфигурации и весов. config: путь к файлу конфигурации YOLO. model: путь к файлу весов модели YOLO.</p>

	Возвращает объект нейронной сети.
<code>cv2.dnn.blobFromImage(image, scalefactor=None, size=None, mean=None, swapRB=None, crop=None, ddepth=None)</code>	<p>Функция создает blob изображения, который будет использоваться в качестве входных данных для нейронной сети YOLO.</p> <p><code>image</code>: входное изображение.</p> <p><code>scalefactor</code> (опционально): коэффициент масштабирования изображения.</p> <p><code>size</code> (опционально): размер выходного изображения после изменения размера.</p> <p><code>mean</code> (опционально): среднее значение пикселей для нормализации.</p> <p><code>swapRB</code> (опционально): менять ли каналы изображения местами.</p> <p><code>crop</code> (опционально): обрезать ли изображение до центральной части.</p> <p><code>ddepth</code> (опционально): тип выходного блоба.</p> <p>Возвращает blob изображения.</p>
<code>net.setInput(blob)</code>	<p>Метод устанавливает blob в качестве входных данных для нейронной сети.</p> <p><code>blob</code>: входные данные в формате blob.</p>
<code>net.forward([, outputName])</code>	<p>Метод выполняет прямой проход (inference) через нейронную сеть.</p> <p><code>outputName</code> (опционально): имена выходных слоев, которые нужно получить.</p> <p>Возвращает результат работы сети.</p>
<code>cv2.dnn.NMSBoxes(bboxes, scores, score_threshold, nms_threshold)</code>	<p>Функция выполняет non-maximum suppression для фильтрации обнаруженных объектов.</p> <p><code>bboxes</code>: ограничивающие рамки обнаруженных объектов.</p> <p><code>scores</code>: уверенность в обнаружении объектов.</p> <p><code>score_threshold</code>: порог уверенности для отбора объектов.</p> <p><code>nms_threshold</code>: порог перекрытия для non-maximum suppression.</p> <p>Возвращает индексы выбранных объектов после применения non-maximum suppression.</p>

Пример работы программы



Текст программы:

```

import cv2
import numpy as np
import time
import os

CONFIDENCE = 0.5
SCORE_THRESHOLD = 0.5
IOU_THRESHOLD = 0.5
config_path = "yolov3.cfg"
weights_path = "yolov3.weights"

labels = open("coco.names").read().strip().split("\n")
colors = np.random.randint(0, 255, size=(len(labels), 3), dtype="uint8")
net = cv2.dnn.readNetFromDarknet(config_path, weights_path)
input_video_path = "input_video.mp4"
output_video_path = "output_video.mp4"
cap = cv2.VideoCapture(input_video_path)

frame_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
frame_height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
fps = int(cap.get(cv2.CAP_PROP_FPS))

fourcc = cv2.VideoWriter_fourcc(*'mp4v')
out = cv2.VideoWriter(output_video_path, fourcc, fps, (frame_width,
frame_height))

while True:
    ret, frame = cap.read()
    if not ret:
        break

```

```

h, w = frame.shape[:2]

blob = cv2.dnn.blobFromImage(frame, 1 / 255.0, (416, 416),
swapRB=True, crop=False)
net.setInput(blob)
ln = net.getLayerNames()
try:
    ln = [ln[i[0] - 1] for i in net.getUnconnectedOutLayers()]
except IndexError:
    ln = [ln[i - 1] for i in net.getUnconnectedOutLayers()]

start = time.perf_counter()
layer_outputs = net.forward(ln)
time_taken = time.perf_counter() - start
print(f"Time took: {time_taken:.2f}s")
boxes, confidences, class_ids = [], [], []

for output in layer_outputs:
    for detection in output:
        scores = detection[5:]
        class_id = np.argmax(scores)
        confidence = scores[class_id]
        if confidence > CONFIDENCE:
            box = detection[:4] * np.array([w, h, w, h])
            (centerX, centerY, width, height) = box.astype("int")
            x = int(centerX - (width / 2))
            y = int(centerY - (height / 2))
            boxes.append([x, y, int(width), int(height)])
            confidences.append(float(confidence))
            class_ids.append(class_id)

idxs = cv2.dnn.NMSBoxes(boxes, confidences, SCORE_THRESHOLD,
IOU_THRESHOLD)

font_scale = 1
thickness = 1
if len(idxs) > 0:
    for i in idxs.flatten():
        x, y = boxes[i][0], boxes[i][1]
        w, h = boxes[i][2], boxes[i][3]
        color = [int(c) for c in colors[class_ids[i]]]
        cv2.rectangle(frame, (x, y), (x + w, y + h), color=color,
thickness=thickness)
        text = f"{labels[class_ids[i]]}: {confidences[i]:.2f}"
        (text_width, text_height) = \
            cv2.getTextSize(text, cv2.FONT_HERSHEY_SIMPLEX,
fontScale=font_scale, thickness=thickness)[0]
        text_offset_x = x

```

```

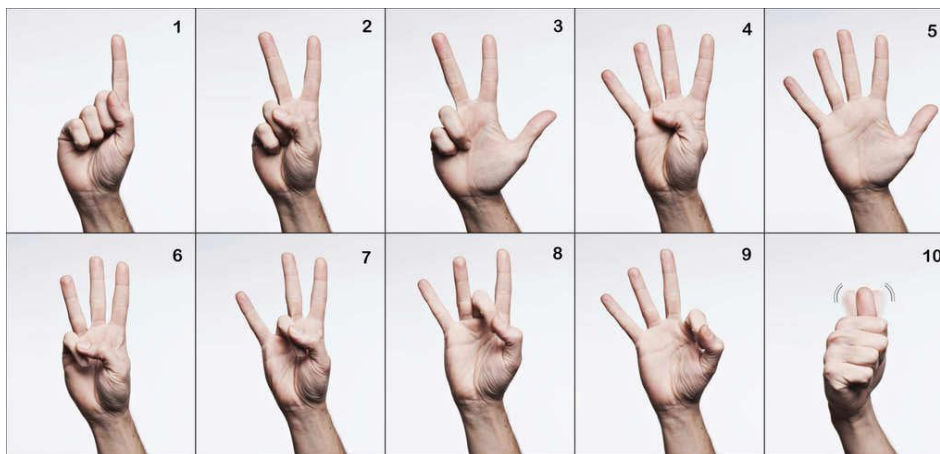
text_offset_y = y - 5
box_coords = ((text_offset_x, text_offset_y), (text_offset_x
+ text_width + 2, text_offset_y - text_height))
overlay = frame.copy()
cv2.rectangle(overlay, box_coords[0], box_coords[1],
color=color, thickness=cv2.FILLED)
frame = cv2.addWeighted(overlay, 0.6, frame, 0.4, 0)
cv2.putText(frame, text, (x, y - 5),
cv2.FONT_HERSHEY_SIMPLEX, fontScale=font_scale, color=(0, 0, 0),
thickness=thickness)

out.write(frame)
cap.release()
out.release()
cv2.destroyAllWindows()

```

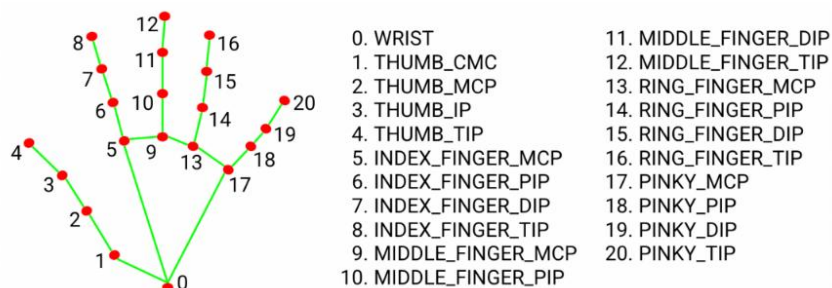
3.15. Распознавание цифр при использовании языка жестов. TensorFlow Sequential

Программа захватывает изображение с видеокамеры. Пользователь показывает одну, либо две руки, программа в реальном времени отображает слева наверху количество пальцев. Для считывания количества пальцев с одной руки используется язык жестов.



Алгоритм работы

1. Захватываем изображение с видеокамеры
2. Определяем количество рук
3. Захватываем ладонь и определяем ключевые точки



4. Определяем количество и положение пальцев
5. Выводим слева наверху в окне количество

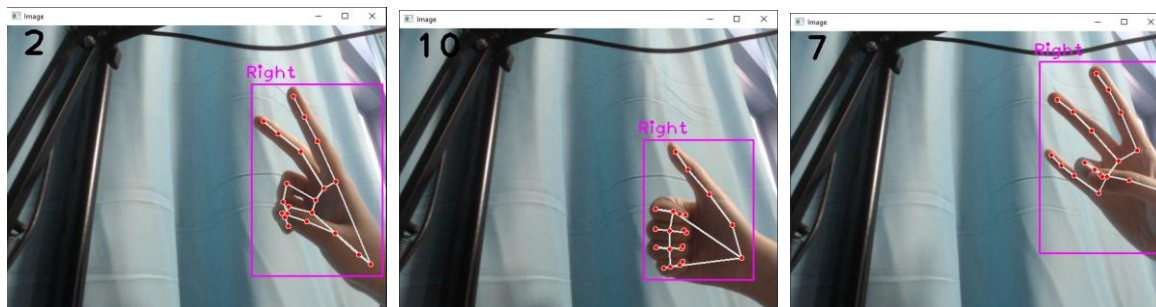
Используемые функции OpenCV

videocapture	Захватывает изображение с видеокамеры
findHands(img)	Ищет руки на изображении с видеокамеры и проставляет точки
detector.fingersUp(hand)	Определяет количество поднятых пальцев
imshow	Используется для отображения изображения в окне
putText	Помещает в окно с изображением подсчитанное количество и изменяется в реальном времени
waitKey	Функция, ожидающая нажатия какой-либо кнопки
destroyAllWindows	Используется для закрытия всех созданных окон

Используемые функции TensorFlow [63]

model = tf.keras.models.Sequential([...])	Создается объект модели Sequential, который представляет собой линейный стек слоев
compile	Модель компилируется
fit	Эта функция обучает модель на данных
save	Модель сохраняется
model = tf.keras.models.load_model	Загрузка модели
predict	Прогнозирование модели

Пример работы



Текст программы

Обучение

```

import TensorFlow as tf
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(224, 224, 3)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')
])
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
model.fit(x, y, epochs=100, batch_size=10, verbose=2)
model.save('finger_count_model')

```

Интеграция

```

import cv2
import TensorFlow as tf
from cvzone.HandTrackingModule import HandDetector

model = tf.keras.models.load_model('finger_count_model')

cap = cv2.VideoCapture(0)
cap.set(3, 640)
cap.set(4, 480)

detector = HandDetector(maxHands=1)
while True:
    success, img = cap.read()
    hands, img = detector.findHands(img) # with draw
    hand = hands[0]
    fingers = detector.fingersUp(hand)
    prediction = model.predict(preprocessed_image)
    fingers = tf.argmax(prediction, axis=1).numpy()[0]
    cv2.putText(img, str(fingers), (25, 50), cv2.FONT_HERSHEY_PLAIN, 4,
(0, 0, 0), 6)

    cv2.imshow("Image", img)
    key = cv2.waitKey(1)
    if key == 27:
        break
cv2.destroyAllWindows()

```

3.16. Распознавание цвета с веб-камеры. TensorFlow

В данном проекте реализована программа, которая с помощью нейронной сети на базе TensorFlow и Keras распознает цвет с веб-камеры. Программа предоставляет пользователю возможность получать информацию о цвете, который находится в центре кадра камеры, отображая распознанный цвет на экране в режиме реального времени.

Пользователь может завершить работу программы по своему усмотрению, либо продолжать процесс для дальнейшего распознавания цветов.

TensorFlow — это платформа с открытым исходным кодом для машинного обучения. TensorFlow позволяет разработчикам создавать, обучать и разворачивать модели машинного обучения и глубокого обучения для широкого спектра задач, таких как распознавание изображений, обработка естественного языка и прогнозирование временных рядов. Keras — это высокоуровневый интерфейс для нейронных сетей, написанный на Python, который работает поверх TensorFlow. Keras обеспечивает интуитивно понятный способ построения и тренировки моделей машинного обучения.

Алгоритм работы

Программа использует OpenCV для захвата видео с веб-камеры и библиотеки TensorFlow и Keras для распознавания цвета с веб-камеры. Основные этапы работы программы включают:

1. Захват изображения с веб-камеры.
2. Определение центрального пикселя кадра
3. Преобразование BGR-цвета в RGB
4. Использование модели нейронной сети для распознавания цвета.
5. Вывод распознанного цвета на экран.

Таблица используемых функций

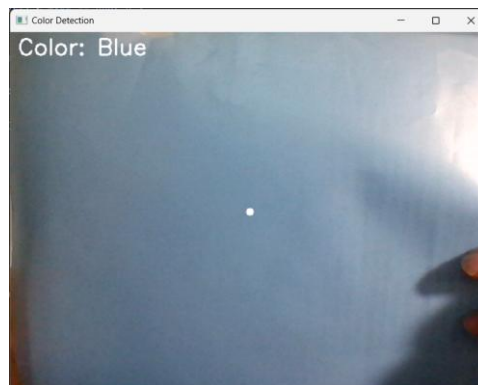
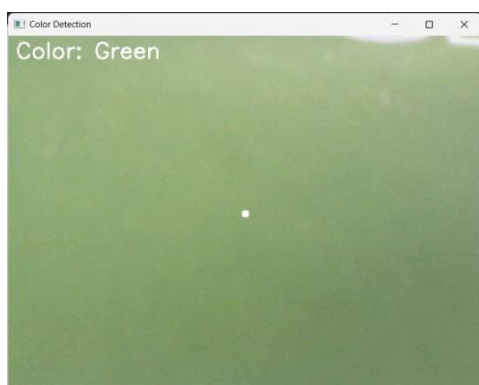
Библиотека	Функция	Описание
cv2	cv2.VideoCapture(index)	Открывает подключение к веб-камере. <ul style="list-style-type: none">• index – индекс камеры(например, 0 для встроенной камеры, 1 для внешней камеры и т.д.).
cv2	cap.read()	Считывает кадр из видеопотока. Возвращает два значения: <ul style="list-style-type: none">• ret - Булево значение, указывающее, успешен ли захват

		<p>кадра.</p> <ul style="list-style-type: none"> • Frame - Захваченный кадр.
cv2	cv2.putText(img, text, org, font, fontScale, color, thickness)	<p>Наносит текст на изображение</p> <ul style="list-style-type: none"> • img: Изображение, на которое наносится текст. • text: Текст для отображения. • org: Координаты начала текста (нижний левый угол). • font: Шрифт текста (например, cv2.FONT_HERSHEY_SIMPLEX). • fontScale: Масштаб шрифта. • color: Цвет текста (например, (255, 255, 255) для белого цвета). • thickness: Толщина линий текста.
cv2	cv2.circle(img, center, radius, color, thickness)	<p>Рисует круг на изображении</p> <ul style="list-style-type: none"> • img: Изображение, на котором рисуется круг. • center: Координаты центра круга. • radius: Радиус круга. • color: Цвет круга. • thickness: Толщина линий круга (-1 для заполненного круга).
cv2	cv2.imshow(window_name, img)	<p>Отображает изображение в окне</p> <ul style="list-style-type: none"> • window_name: Имя окна. • img: Изображение для отображения.
cv2	cv2.waitKey(delay)	<p>Ожидает нажатие клавиши в течение указанного времени</p> <ul style="list-style-type: none"> • delay: Время ожидания в миллисекундах (0 — ожидание бесконечно).
cv2	cap.release()	Освобождает захват видео.
cv2	cv2.destroyAllWindows()	Закрывает все открытые окна
Tensorflow	tensorflow.keras.models.Sequential	Класс для создания последовательной модели, где слои добавляются поочередно
Tensorflow	tensorflow.keras.layers.Dense	Полносвязный (Dense) слой нейронной сети

		<ul style="list-style-type: none"> • units: Количество нейронов в слое. • activation: Функция активации (например, 'relu', 'softmax')
Tensorflow	tensorflow.keras.layers.Flatten	Слой, преобразующий многомерные входные данные в одномерный массив.
Tensorflow	tensorflow.keras.Model.compile	<p>Метод для компиляции модели с заданием оптимизатора, функции потерь и метрик</p> <ul style="list-style-type: none"> • optimizer: Оптимизатор для обновления весов (например, 'adam'). • loss: Функция потерь (например, 'sparse_categorical_crossentropy'). • metrics: Список метрик для оценки модели (например, ['accuracy']).
Tensorflow	tensorflow.keras.Model.fit	<p>Метод для тренировки модели на данных</p> <ul style="list-style-type: none"> • x: Входные данные для тренировки. • y: Метки данных. • epochs: Количество эпох для обучения. • batch_size: Размер батча.
Tensorflow	tensorflow.keras.Model.predict	<p>Метод для предсказания результатов на новых данных.</p> <ul style="list-style-type: none"> • x: Входные данные для предсказания.

Результаты работы программы

Программа определяет показанный ей цвет



Текст программы

```
import cv2
import numpy as np
```

```

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten

# Функция для создания датасета цветов
def create_color_dataset():
    colors = {
        'Red': (255, 0, 0),
        'Green': (0, 255, 0),
        'Blue': (0, 0, 255),
        'Yellow': (255, 255, 0),
        'Cyan': (0, 255, 255),
        'Magenta': (255, 0, 255),
        'White': (255, 255, 255),
        'Black': (0, 0, 0),
        'Gray': (127, 127, 127),
        'Orange': (255, 165, 0),
        'Pink': (255, 192, 203),
        'Purple': (128, 0, 128),
    }

    data = []
    labels = []
    for label, color in colors.items():
        for _ in range(1000):
            perturbation = np.random.randint(-20, 20, 3)
            perturbed_color = np.clip(color + perturbation, 0, 255)
            data.append(perturbed_color)
            labels.append(label)
    return np.array(data), np.array(labels)

# Функция для преобразования меток в числовые значения
def label_to_numeric(labels):
    label_dict = {label: idx for idx, label in enumerate(set(labels))}
    numeric_labels = np.array([label_dict[label] for label in labels])
    return numeric_labels, label_dict

# Функция для создания и тренировки модели
def train_color_model():
    data, labels = create_color_dataset()
    numeric_labels, label_dict = label_to_numeric(labels)

    model = Sequential([
        Flatten(input_shape=(3,)),
        Dense(64, activation='relu'),
        Dense(32, activation='relu'),
        Dense(len(label_dict), activation='softmax')
    ])
    model.compile(optimizer='adam',

```

```

loss='sparse_categorical_crossentropy', metrics=['accuracy'])
    model.fit(data, numeric_labels, epochs=10, batch_size=32)
    return model, label_dict

# Функция для распознавания цвета
def recognize_color(model, label_dict, bgr_color):
    rgb_color = bgr_color[::-1]
    color_array = np.array([rgb_color])
    predictions = model.predict(color_array)
    predicted_label_idx = np.argmax(predictions)
    for label, idx in label_dict.items():
        if idx == predicted_label_idx:
            return label
    return 'Unknown'

# Основная функция
def main():
    model, label_dict = train_color_model()
    cap = cv2.VideoCapture(0)
    if not cap.isOpened():
        print("Failed to open camera")
        return

    while True:
        ret, frame = cap.read()
        if not ret:
            break
        height, width, _ = frame.shape
        center_x, center_y = width // 2, height // 2
        center_pixel = frame[center_y, center_x]
        color_name = recognize_color(model, label_dict, center_pixel)

        cv2.putText(frame, f'Color: {color_name}', (10, 30),
cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2)
        cv2.circle(frame, (center_x, center_y), 5, (255, 255, 255), -1)
        cv2.imshow('Color Detection', frame)

        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
    cap.release()
    cv2.destroyAllWindows()

if __name__ == "__main__":
    main()

    for label, idx in label_dict.items():
        if idx == predicted_label_idx:
            return label
    return 'Unknown'

```

```

# Основная функция
def main():
    model, label_dict = train_color_model()
    cap = cv2.VideoCapture(0)
    if not cap.isOpened():
        print("Failed to open camera")
        return
    while True:
        ret, frame = cap.read()
        if not ret:
            break
        height, width, _ = frame.shape
        center_x, center_y = width // 2, height // 2
        center_pixel = frame[center_y, center_x]
        color_name = recognize_color(model, label_dict, center_pixel)

        cv2.putText(frame, f'Color: {color_name}', (10, 30),
cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2)
        cv2.circle(frame, (center_x, center_y), 5, (255, 255, 255), -1)
        cv2.imshow('Color Detection', frame)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
    cap.release()
    cv2.destroyAllWindows()

if __name__ == "__main__":
    main()

```

3.17. Добавление объектов на видео. MTCNN

Одним из самых широко распространенных применений нейронных сетей в работе с видео является распознавание объектов. Сети, такие как YOLO (You Only Look Once) и SSD (Single Shot MultiBox Detector), позволяют обнаруживать объекты в реальном времени на видеопотоке с высокой скоростью и точностью. Одним из применений нейронных сетей является добавление объектов к видео в реальном времени.

Нейронная сеть, используемая для добавления объектов к видео, обычно базируется на архитектурах сегментации и генерации изображений. Процесс включает в себя несколько ключевых шагов:

- Обнаружение объектов: сначала сеть обнаруживает объекты на исходном видео. Это может быть достигнуто с использованием детекторов объектов, таких как YOLO или SSD.
- Сегментация объектов: после обнаружения объектов сеть выделяет их контуры и основные характеристики с помощью сегментационных методов, таких как U-Net или Mask R-CNN.
- Генерация объектов: затем нейронная сеть генерирует новые объекты или текстуры, которые будут добавлены к видео. Это может быть достигнуто с использованием генеративных адверсариальных сетей (GAN), вариационных автокодировщиков (VAE) и других архитектур.
- Интеграция в видео: Наконец, сгенерированные объекты интегрируются обратно в исходное видео с помощью техник композитинга и сопоставления движения. Укрупненный алгоритм:

Алгоритм работы

1. Импорт библиотек: cv2: OpenCV - библиотека компьютерного зрения; MTCNN из mtcnn: модель детектирования лиц [64].
2. Определение функции `add_accessory(frame, accessory, position, accessory_type)`: Эта функция принимает кадр `frame`, накладываемый аксессуар, его позицию и тип аксессуара ('hat' или 'mustache'). В зависимости от типа аксессуара вычисляются его размеры и позиция на лице. Затем аксессуар накладывается на кадр с учетом прозрачности.
3. Определение функции `apply_filters(frame, hat, mustache, faces)`: Эта функция принимает кадр `frame`, изображения аксессуаров (шляпы и усы) и обнаруженные лица. Для каждого обнаруженного лица накладываются аксессуары.

4. Определение функции `process_frame(frame, hat, mustache, detector)`: Эта функция принимает кадр `frame`, изображения аксессуаров (шляпы и усы) и объект детектора лиц. Детектор используется для обнаружения лиц на кадре, после чего к ним применяются аксессуары.

5. Настройка видеопотока:

Создается объект `cap` для захвата видео с камеры.

Устанавливаются параметры ширины и высоты кадра.

Загрузка изображений аксессуаров (шляпы и усы) и создание объекта детектора лиц.

6. Бесконечный цикл `while True`, в котором:

Кадр считывается с камеры.

Для каждого кадра создается новый поток, в котором вызывается функция `process_frame`.

Каждый поток обрабатывает кадр, добавляя аксессуары к обнаруженным лицам.

Кадр с отображенными аксессуарами выводится на экран.

Если нажата клавиша "q", цикл прерывается.

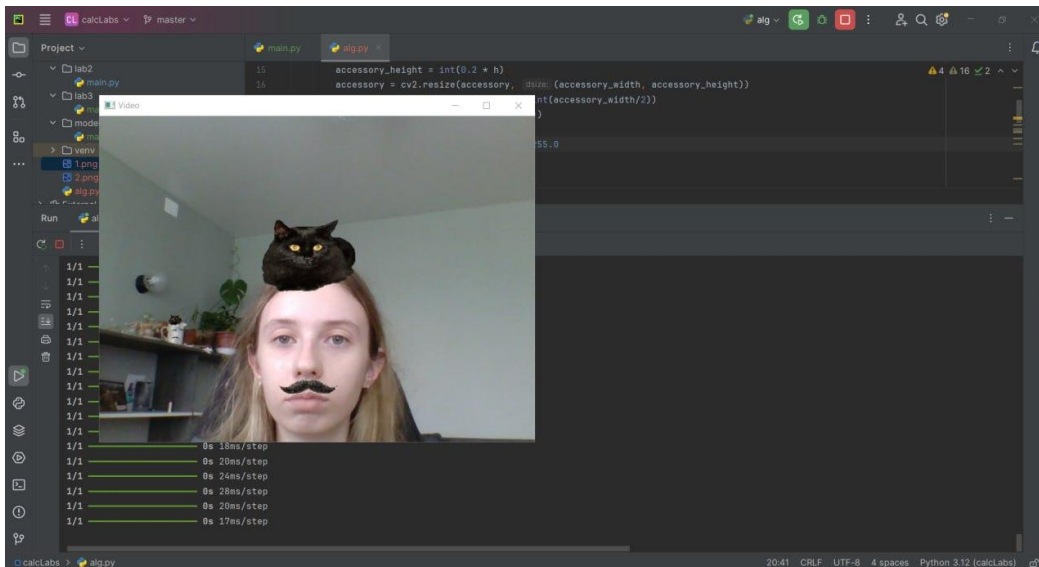
Освобождение ресурсов камеры и закрытие всех окон `OpenCV`.

Этот код использует многопоточность для обработки кадров, что может повысить производительность, так как обработка каждого кадра происходит в отдельном потоке.

Используемые функции

<code>add_accessory(frame, accessory, position, accessory_type)</code>	Добавляет аксессуар к распознанному лицу
<code>apply_filters(frame, hat, mustache, faces)</code>	Применяет фильтры от <code>opencv</code> в кадре в потоке
<code>process_frame(frame, hat, mustache, detector)</code>	Обрабатывает поступивший кадр

Результат работы:



Текст программы

```

import cv2
from mtcnn import MTCNN
import threading

def add_accessory(frame, accessory, position, accessory_type):
    x, y, w, h = position
    if accessory_type == 'hat':
        accessory_width = int(1.2 * w)
        accessory_height = int(0.5 * h)
        accessory = cv2.resize(accessory, (accessory_width,
accessory_height))
        accessory_x = max(0, x + int(w/2) - int(accessory_width/2))
        accessory_y = max(0, y - int(accessory_height * 0.8))
    elif accessory_type == 'mustache':
        accessory_width = int(0.6 * w)
        accessory_height = int(0.2 * h)
        accessory = cv2.resize(accessory, (accessory_width,
accessory_height))
        accessory_x = max(0, x + int(w/2) - int(accessory_width/2))
        accessory_y = max(0, y + int(h * 0.6))

    alpha_accessory = accessory[:, :, -1] / 255.0
    alpha_frame = 1.0 - alpha_accessory

    for c in range(0, 3):
        frame[accessory_y:accessory_y+accessory_height,
accessory_x:accessory_x+accessory_width, c] = \
            (alpha_accessory * accessory[:, :, c] + alpha_frame *
frame[accessory_y:accessory_y+accessory_height,
accessory_x:accessory_x+accessory_width, c])
    return frame

```

```

def apply_filters(frame, hat, mustache, faces):
    for face in faces:
        x, y, w, h = face['box']
        keypoints = face['keypoints']
        frame = add_accessory(frame, hat, (x, y, w, h), 'hat')
        frame = add_accessory(frame, mustache, (x, y, w, h), 'mustache')
    return frame

def process_frame(frame, hat, mustache, detector):
    faces = detector.detect_faces(frame)
    return apply_filters(frame, hat, mustache, faces)

cap = cv2.VideoCapture(0)
cap.set(cv2.CAP_PROP_FRAME_WIDTH, 640)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)

hat = cv2.imread('1.png', cv2.IMREAD_UNCHANGED)
mustache = cv2.imread('2.png', cv2.IMREAD_UNCHANGED)

detector = MTCNN()

while True:
    ret, frame = cap.read()
    if not ret:
        break

    thread = threading.Thread(target=process_frame, args=(frame, hat,
mustache, detector))
    thread.start()
    thread.join()

    cv2.imshow('Video', frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()

```

3.18. Распознавание текста с видеокамеры. Tesseract

Разработана программа, которая с помощью нейронной сети и Tesseract OCR распознает текст с веб-камеры. Программа предоставляет пользователю возможность подтверждать правильность распознанного текста и завершать работу при корректном распознавании, либо продолжать процесс до получения правильного результата.

Tesseract OCR (Optical Character Recognition) — это мощная система оптического распознавания символов, которая преобразует печатный текст в машиночитаемый формат. Tesseract был первоначально разработан компанией Hewlett-Packard в 1985 году и с тех пор прошел значительный путь развития. В 2006 году проект был открыт и с тех пор поддерживается и развивается сообществом под руководством Google.

Tesseract OCR является одним из самых популярных и точных инструментов для распознавания текста благодаря следующим возможностям [46, 47]:

1. Поддержка множества языков: Tesseract OCR поддерживает более 100 языков, включая русские и английские языковые модели.
2. Работа с различными форматами изображений: Tesseract может работать с изображениями в форматах TIFF, JPEG, PNG и других.
3. Высокая точность распознавания: Благодаря использованию обученных моделей и алгоритмов машинного обучения, Tesseract обеспечивает высокую точность распознавания текста.
4. Открытость и расширяемость: Tesseract является проектом с открытым исходным кодом, что позволяет разработчикам модифицировать и улучшать его под свои нужды.

Алгоритм работы

Программа использует OpenCV для захвата видео с веб-камеры и библиотеку pytesseract для оптического распознавания текста. Основные этапы работы программы включают:

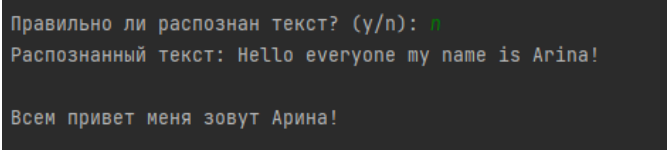
1. Захват изображения с веб-камеры.
2. Преобразование изображения в оттенки серого для улучшения качества распознавания.
3. Использование Tesseract OCR для распознавания текста на изображении.
4. Вывод распознанного текста и запрос подтверждения от пользователя.

5. Завершение работы программы при подтверждении правильности распознанного текста или продолжение цикла при необходимости.

Таблица используемых функций

Библиотек	Функция	Описание
cv2	cv2.VideoCapture(0)	Открывает подключение к веб-камере. Параметр 0 указывает на использование первой доступной камеры.
cv2	cap.isOpened()	Проверяет, успешно ли открыто подключение к веб-камере.
cv2	cap.read()	Считывает кадр из видеопотока. Возвращает два значения: флаг успешности и сам кадр.
cv2	cv2.cvtColor()	Преобразует изображение в другое цветовое пространство. В данном случае преобразует в оттенки серого.
cv2	cv2.imshow()	Отображает изображение в указанном окне.
cv2	cv2.waitKey(1)	Ожидает нажатие клавиши в течение указанного времени (в миллисекундах). Возвращает код нажатой клавиши.
cv2	cap.release()	Освобождает ресурсы, занятые видеокamerой.
cv2	cv2.destroyAllWindows()	Закрывает все окна, созданные с помощью OpenCV.
pytesseract	pytesseract.pytesseract.tesseract_cmd	Устанавливает путь к исполняемому файлу Tesseract OCR.
pytesseract	pytesseract.image_to_string()	Распознает текст на изображении. Принимает на вход изображение и параметры языка для распознавания.

Результаты работы программы.

Ожидаемый результат	Результат
Hello everyone my name is Arina!	Hello everyone my name is Arina!
Всем привет меня зовут Арина!	Всем привет меня зовут Арина!
<p>Hello everyone my name is Arina!</p> <p>Всем привет меня зовут Арина!</p>	 <pre> Правильно ли распознан текст? (y/n): y Распознанный текст: Hello everyone my name is Arina! Всем привет меня зовут Арина! </pre>

Текст программы

```
import cv2
import pytesseract

# Укажите путь к Tesseract OCR
pytesseract.pytesseract.tesseract_cmd = r'C:\Program Files\Tesseract-OCR\tesseract.exe'

def main():
    # Откройте веб-камеру
    cap = cv2.VideoCapture(0)
    if not cap.isOpened():
        print("Не удалось открыть веб-камеру")
        return
    while True:

        # Захватите кадр
        ret, frame = cap.read()
        if not ret:
            print("Не удалось захватить кадр")
            break

        # Преобразуйте изображение в оттенки серого
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        # Используйте Tesseract для распознавания текста
        text = pytesseract.image_to_string(gray, lang='rus+eng')

        # Выведите распознанный текст на экран
        print(f"Распознанный текст: {text}")

        # Отобразите кадр
        cv2.imshow('Webcam', frame)

        # Спросите пользователя, правильно ли распознан текст
        user_input = input("Правильно ли распознан текст? (y/n):
").strip().lower()
        if user_input == 'y':
            print("Программа завершена.")
            break

        # Нажмите 'q' для выхода из программы в любой момент
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    # Освободите ресурсы
    cap.release()
    cv2.destroyAllWindows()
```

```
if __name__ == "__main__":
    main()
```

3.19. Приложение для обработки изображений с видеокamеры. Keras

Разработано приложение, в котором использованы библиотеки: tkinter для реализации интерфейса, pillow для простой работы с изображениями, imutils для работы с потоком видео и keras для применения машинного обучения.

Чтобы начать транслировать видеоряд будем использовать VideoStream(0).start() из imutils, а чтобы считать видео будем использовать метод read() из этого же класса. Запуск приложения осуществляется следующим образом:

```
FilterApp(VideoStream(0).start()).root.mainloop()
```

Все фильтры будем накладывать с помощью библиотеки OpenCV, для этого будем использовать следующие функции:

cvtColor	Поменять цветовое пространство
CascadeClassifier	Обнаружение методом Haar-cascade
rectangle	Рисование прямоугольника
putText	Размещение текста
resize	Изменение размера
getAffineTransform	Преобразование, которое может быть выражено в виде умножения матрицы, метод для получения матрицы
warpAffine	Метод для применения матрицы
filter2D	Применение произвольного линейного фильтра
add	Добавление изображений
getGaussianKernel	Возвращает коэффициенты гауссовского фильтра
medianBlur	Функция сглаживает изображение с помощью медианного фильтра
Laplacian	Обнаружение краев
threshold	Применение пороговых значений
imwrite	Сохранение изображения

Рассмотрим пример реализации фильтра Vignette:

```

def vignetteTransform(self):

    rows, cols = self.frame.shape[:2]

    # Сгенерируем маску с помощью Gaussian kernels
    kernel_x = cv2.getGaussianKernel(cols, 200)
    kernel_y = cv2.getGaussianKernel(rows, 200)
    kernel = kernel_y * kernel_x.T
    mask = 255 * kernel / np.linalg.norm(kernel)
    vignetteFrame = np.copy(self.frame)

    # Применим маску к каждому каналу во входном изображении

    for i in range(3):
        vignetteFrame[:, :, i] = vignetteFrame[:, :, i] * mask

    return vignetteFrame

```

Интерфейс отрисуем с помощью tkinter, например, кнопку для Vignette:

```

vignette = tk.Button(self.buttonframe, text="Vignette", bg="#000",
fg='black', width=25, height=2,
                    font='Helvetica 10 bold',
activebackground="#fff", activeforeground="#000", padx=5, pady=5,
                    command=lambda: self.changeFilter('vignette'))
vignette.pack()

```

Так же была добавлена работа с нейросетью, а именно распознавание эмоций:

```

emotion_dict = {0: "Angry", 1: "Disgusted", 2: "Fearful", 3: "Happy",
4: "Neutral", 5: "Sad",
                6: "Surprised"}

model = Sequential()

model.load_weights('model.h5')
facecasc =
cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
gray = cv2.cvtColor(self.frame, cv2.COLOR_BGR2GRAY)
faces = facecasc.detectMultiScale(gray, scaleFactor=1.3,
minNeighbors=5)

for (x, y, w, h) in faces:
    cv2.rectangle(self.frame, (x, y - 50), (x + w, y + h + 10), (255,
0, 0), 2)
    roi_gray = gray[y:y + h, x:x + w]

```

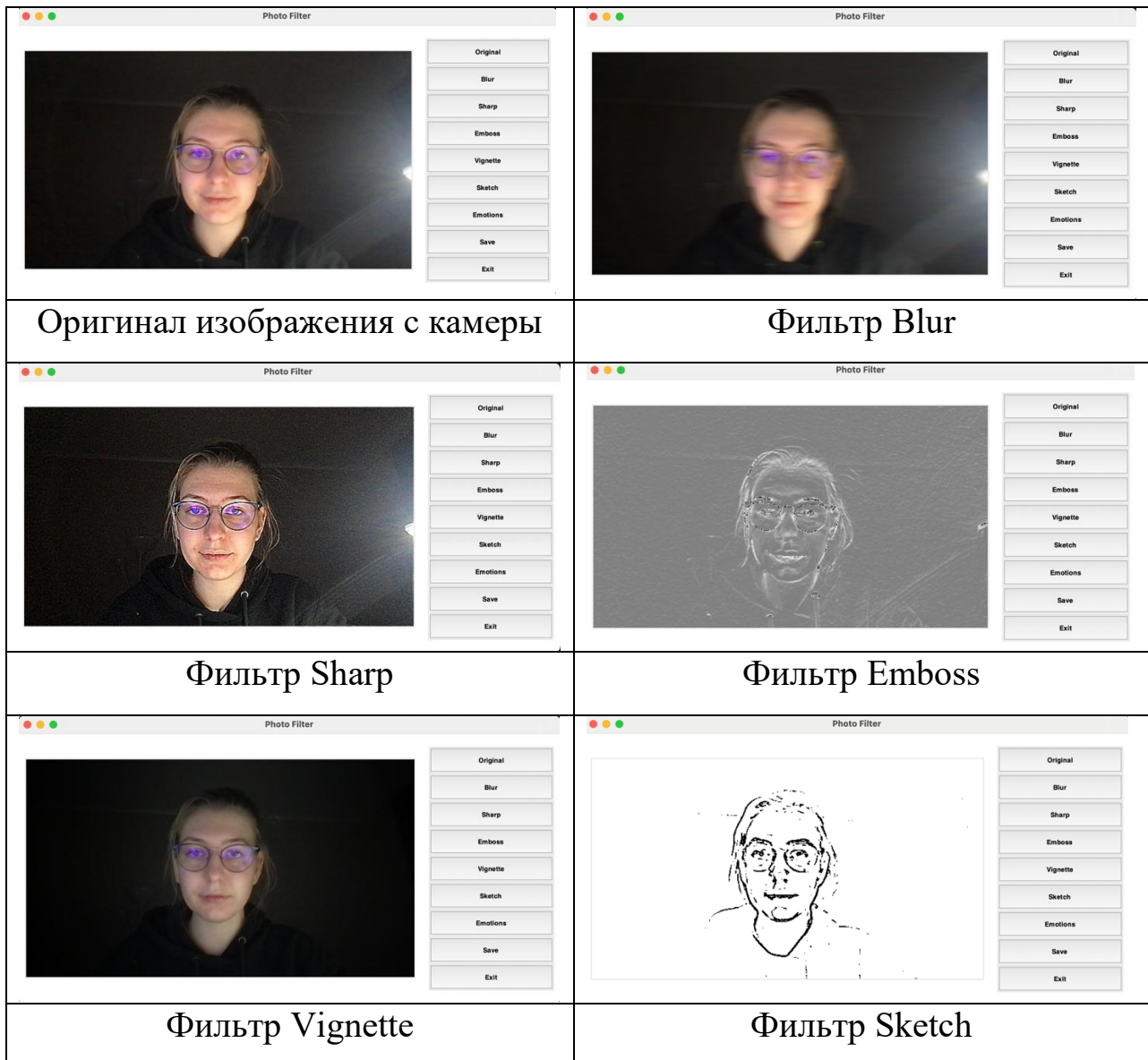
```

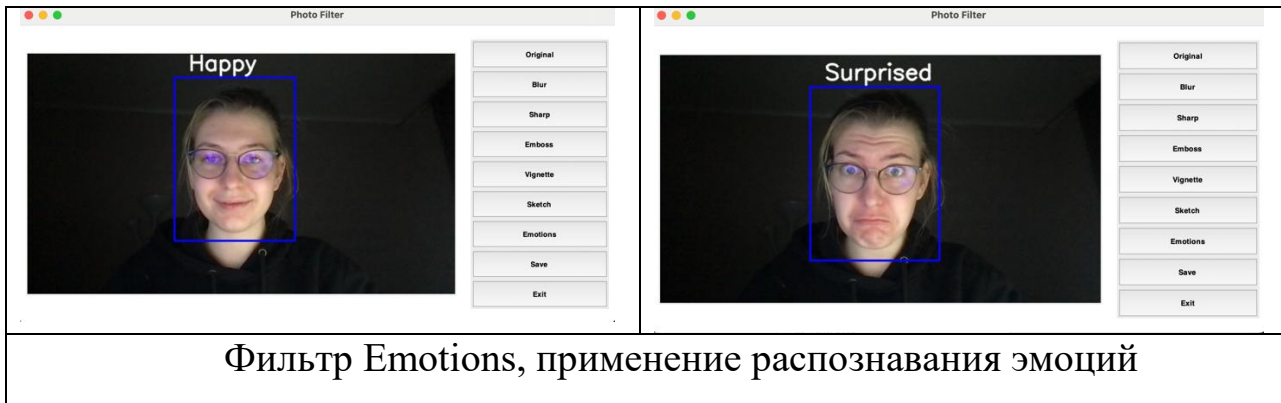
cropped_img = np.expand_dims(np.expand_dims(cv2.resize(roi_gray,
(48, 48)), -1), 0)
prediction = model.predict(cropped_img)
maxindex = int(np.argmax(prediction))
cv2.putText(self.frame, emotion_dict[maxindex], (x + 20, y - 60),
cv2.FONT_HERSHEY_SIMPLEX, 1,
(255, 255, 255), 2, cv2.LINE_AA)

```

После применения фильтра изображения можно сохранить.

Результаты работы





Текст программы

```

import threading
import imutils
import cv2
import numpy as np
import tkinter as tk
from PIL import ImageTk, Image
import os
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from imutils.video import VideoStream

size = 15
kernel_motion_blur = np.zeros((size, size))
kernel_motion_blur[int((size - 1) / 2), :] = np.ones(size)
kernel_motion_blur = kernel_motion_blur / size

kernel_sharpen = np.array([[[-1, -1, -1], [-1, 9, -1], [-1, -1, -1]])

kernel_emboss_1 = np.array([[0, -1, -1],
                             [1, 0, -1],
                             [1, 1, 0]])
kernel_emboss_2 = np.array([[[-1, -1, 0],
                              [-1, 0, 1],
                              [0, 1, 1]])

kernel_emboss_3 = np.array([[1, 0, 0],
                             [0, 0, 0],
                             [0, 0, -1]])

class FilterApp(object):
    def __init__(self, vs):
        self.thread = None
        self.stopEvent = None
        self.vs = vs
        self.frame = None
        self.image = None

```

```

self.panel = None
self.currentfilter = None

self.root = tk.Tk()
self.root.wm_protocol("WM_DELETE_WINDOW", self.onWindowClose) #
callback to window close function
self.root.wm_title("Photo Filter") # Title of window
self.root.config(bg="#fff")

self.stopEvent = threading.Event()
self.thread = threading.Thread(target=self.videoLoop, args=())
self.thread.start()
self.addButtons()
def changeFilter(self, filterValue):
self.currentfilter = filterValue
def addButtons(self):
self.buttonframe = tk.Frame(self.root)
self.buttonframe.pack(side="right", padx=10, pady=20)

original = tk.Button(self.buttonframe, text="Original",
bg="#000", fg='black', width=25, height=2,
font='Helvetica 10 bold',
activebackground="#fff", activeforeground="#000", padx=5, pady=5,
command=lambda:
self.changeFilter('original'))
original.pack()

blur = tk.Button(self.buttonframe, text="Blur", bg="#000",
fg='black', width=25, height=2,
font='Helvetica 10 bold',
activebackground="#fff", activeforeground="#000", padx=5, pady=5,
command=lambda: self.changeFilter('blur'))
blur.pack()

sharp = tk.Button(self.buttonframe, text="Sharp", bg="#000",
fg='black', width=25, height=2,
font='Helvetica 10 bold',
activebackground="#fff", activeforeground="#000", padx=5, pady=5,
command=lambda: self.changeFilter('sharp'))
sharp.pack()

emboss = tk.Button(self.buttonframe, text="Emboss", bg="#000",
fg='black', width=25, height=2,
font='Helvetica 10 bold',
activebackground="#fff", activeforeground="#000", padx=5, pady=5,
command=lambda: self.changeFilter('emboss'))
emboss.pack()

vignette = tk.Button(self.buttonframe, text="Vignette",

```

```

bg="#000", fg='black', width=25, height=2,
                    font='Helvetica 10 bold',
activebackground="#fff", activeforeground="#000", padx=5, pady=5,
                    command=lambda:
self.changeFilter('vignette'))
    vignette.pack()

    sketch = tk.Button(self.buttonframe, text="Sketch", bg="#000",
fg='black', width=25, height=2,
                    font='Helvetica 10 bold',
activebackground="#fff", activeforeground="#000", padx=5, pady=5,
                    command=lambda: self.changeFilter('sketch'))
    sketch.pack()

    emotions = tk.Button(self.buttonframe, text="Emotions",
bg="#000", fg='black', width=25, height=2,
                    font='Helvetica 10 bold',
activebackground="#fff", activeforeground="#000", padx=5, pady=5,
                    command=lambda:
self.changeFilter('emotions'))
    emotions.pack()

    save = tk.Button(self.buttonframe, text="Save", bg="#000",
fg='black', width=25, height=2,
                    font='Helvetica 10 bold',
activebackground="#fff", activeforeground="#000", padx=5, pady=5,
                    command=self.saveFilteredImage)
    save.pack()

    quit = tk.Button(self.buttonframe, text="Exit", bg="#000",
fg='black', width=25, height=2,
                    font='Helvetica 10 bold',
activebackground="#fff", activeforeground="#000", padx=5, pady=5,
                    command=self.onWindowClose)
    quit.pack()

def videoLoop(self):
    try:
        while not self.stopEvent.is_set():

            self.frame = self.vs.read()
            self.frame = imutils.resize(self.frame, width=600)
            self.frame = self.affineTransform()
            if self.currentfilter == 'blur':
                self.frame = self.blurTransform()
            elif self.currentfilter == 'sharp':
                self.frame = self.sharpTransform()
            elif self.currentfilter == 'emboss':
                self.frame = self.embossTransform()

```

```

elif self.currentfilter == 'vignette':
    self.frame = self.vignetteTransform()
elif self.currentfilter == 'sketch':
    self.frame = self.sketchTransform()
elif self.currentfilter == 'original':
    self.frame = self.frame
elif self.currentfilter == 'emotions':
    self.emotionsTrnsform()
    # self.frame = self.frame
self.image = self.frame
image = cv2.cvtColor(self.frame, cv2.COLOR_BGR2RGB)
image = Image.fromarray(image)
image = ImageTk.PhotoImage(image)
# if the panel is not None, we need to initialize it
if self.panel is None:
    self.panel = tk.Label(image=image)
    self.panel.image = image
    self.panel.pack(side="left", padx=10, pady=10)
# otherwise, simply update the panel
else:
    self.panel.configure(image=image)
    self.panel.image = image
except RuntimeError:
    print("Exiting App")
def emotionsTrnsform(self):

    emotion_dict = {0: "Angry", 1: "Disgusted", 2: "Fearful", 3:
"Happy", 4: "Neutral", 5: "Sad",
                    6: "Surprised"}

    model = Sequential()
    model.add(Conv2D(32, kernel_size=(3, 3), activation='relu',
input_shape=(48, 48, 1)))
    model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))

    model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Conv2D(128, kernel_size=(3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Dropout(0.25))

    model.add(Flatten())
    model.add(Dense(1024, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(7, activation='softmax'))
    model.load_weights('model.h5')

```

```

        facecasc =
cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
        gray = cv2.cvtColor(self.frame, cv2.COLOR_BGR2GRAY)
        faces = facecasc.detectMultiScale(gray, scaleFactor=1.3,
minNeighbors=5)

        for (x, y, w, h) in faces:
            cv2.rectangle(self.frame, (x, y - 50), (x + w, y + h + 10),
(255, 0, 0), 2)
            roi_gray = gray[y:y + h, x:x + w]
            cropped_img =
np.expand_dims(np.expand_dims(cv2.resize(roi_gray, (48, 48)), -1), 0)
            prediction = model.predict(cropped_img)
            maxindex = int(np.argmax(prediction))
            cv2.putText(self.frame, emotion_dict[maxindex], (x + 20, y -
60), cv2.FONT_HERSHEY_SIMPLEX, 1,
                    (255, 255, 255), 2, cv2.LINE_AA)
        def affineTransform(self):
            rows, cols = self.frame.shape[:2]
            src_points = np.float32([[0, 0], [cols - 1, 0], [0, rows - 1]])
            dstn_points = np.float32([[cols - 1, 0], [0, 0], [cols - 1, rows
- 1]])
            affineMatrix = cv2.getAffineTransform(src_points, dstn_points)
            affineFrame = cv2.warpAffine(self.frame, affineMatrix, (cols,
rows))
            return affineFrame
        def blurTransform(self):
            global kernel_motion_blur
            blurFrame = cv2.filter2D(self.frame, -1, kernel_motion_blur)
            return blurFrame
        def grayTransform(self):
            grayFrame = cv2.cvtColor(self.frame, cv2.COLOR_BGR2GRAY)
            return grayFrame
        def sharpTransform(self):
            global kernel_sharpen
            sharpFrame = cv2.filter2D(self.frame, -1, kernel_sharpen)
            return sharpFrame
        def embossTransform(self):
            global kernel_emboss_1, kernel_emboss_2, kernel_emboss_3
            grayImage = self.grayTransform()
            output_1 = cv2.filter2D(grayImage, -1, kernel_emboss_1)
            output_2 = cv2.filter2D(grayImage, -1, kernel_emboss_2)
            output_3 = cv2.filter2D(grayImage, -1, kernel_emboss_3)

            embossFrame = cv2.add(output_1, output_2, output_3)
            return embossFrame + 128
        def vignetteTransform(self):
            rows, cols = self.frame.shape[:2]
            # generating vignette mask using Gaussian kernels

```

```

kernel_x = cv2.getGaussianKernel(cols, 200)
kernel_y = cv2.getGaussianKernel(rows, 200)
kernel = kernel_y * kernel_x.T
mask = 255 * kernel / np.linalg.norm(kernel)
vignetteFrame = np.copy(self.frame)

# applying the mask to each channel in the input image
for i in range(3):
    vignetteFrame[:, :, i] = vignetteFrame[:, :, i] * mask
return vignetteFrame
def sketchTransform(self):
    imageGray = self.grayTransform()
    imageGray = cv2.medianBlur(imageGray, 7)
    edges = cv2.Laplacian(imageGray, cv2.CV_8U, ksize=5)
    ret, mask = cv2.threshold(edges, 100, 255, cv2.THRESH_BINARY_INV)
    return mask
def saveFilteredImage(self):
    if not os.path.exists('savedImages/'):
        os.makedirs('savedImages/')
    filename = "savedImages/{}.jpg".format(self.currentfilter)
    cv2.imwrite(filename, self.image)
# on app close
def onWindowClose(self):
    self.stopEvent.set()
    self.vs.stop()
    self.root.destroy()

if __name__ == '__main__':
    FilterApp(VideoStream(0).start()).root.mainloop()

```

3.20. Обнаружение ям на дорогах в режиме реального времени.

YOLOv8

Для выполнения работы использована модель YOLOv8 (You Only Look Once). YOLOv8 является одной из ведущих моделей для обнаружения объектов и сегментации изображений в реальном времени. YOLO - это сверточная нейронная сеть, которая способна предсказывать ограничительные рамки и вероятности классов объектов на изображении всего за один проход.

Сбор и подготовка данных

В этой работе используется обширный набор данных, содержащий более 7000 изображений, которые были собраны из различных

источников. Этот набор данных включает в себя изображения из следующих источников:

- Roboflow
- Изображения, полученные из видеороликов YouTube и аннотированные вручную
- Изображения из набора данных RDD2022

После работы над аннотациями, окончательный набор данных теперь состоит из 6962 изображения для обучения и 271 изображения для валидации.

Для обучения модели нам необходимо создать файл YAML, который определяет пути к изображениям и названия классов. Этот YAML-файл должен был быть размещен в корневом каталоге проекта и иметь следующую структуру:

```
path: pothole_dataset_v8/  
train: 'train/images'  
val: 'valid/images'
```

```
# названия классов  
names:  
  0: 'pothole'
```

Подготовка ПО

Для обучения модели YOLOv8 на наборе данных нам потребуется установить пакет ultralytics. Этот пакет обеспечивает интерфейс командной строки (CLI) yolo. Одним из главных преимуществ его использования является то, что не требуется дополнительное клонирование репозитория и установка дополнительных зависимостей.

Однако перед установкой пакета ultralytics необходимо предварительно установить пакет PyTorch с поддержкой Nvidia CUDA, а так же CUDA Toolkit и cuDNN. Это необходимо, поскольку обучение модели на ЦПУ может занять слишком много времени. Для этого перед

установкой пакета `ultralytics` следует выполнить соответствующие инструкции по установке PyTorch с официального сайта.

```
pip3 install torch torchvision torchaudio --index-url
https://download.pytorch.org/whl/cu118
```

```
pip install ultralytics
```

После выполнения этих следует установить `Jupyter Notebook`, где возможно производить обучение модели `YOLOv8`. В `Jupyter Notebook` можно использовать удобный интерфейс для написания и выполнения кода, отображения результатов, а также добавления текстовых и графических элементов, объясняющих процесс обучения и результаты работы модели.

Тренировка модели

Для тренировки модели запишем следующий скрипт в `jupyter notebook`.

```
from ultralytics import YOLO

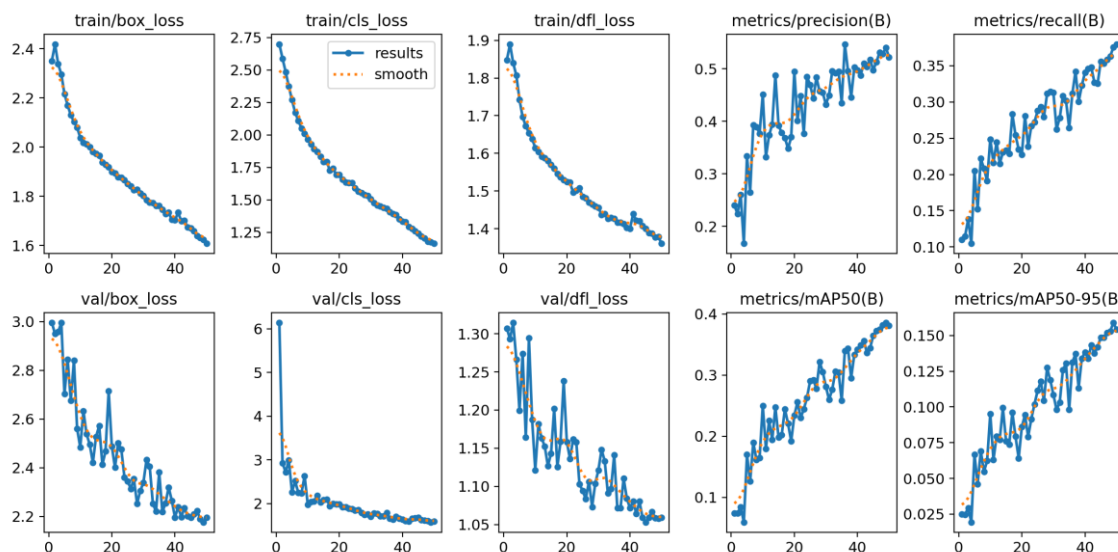
# Загрузка модели
model = YOLO('yolov8m.pt')

# Тренировка
results = model.train(
    data='pothole_v8.yaml',
    imgsz=640,
    epochs=50,
    batch=8,
    name='yolov8n_pothole'
)
```

- `model`: Выбранная модель для использования, в данном случае - `YOLOv8 m`.
- `imgsz`: Размер изображения, по умолчанию установлено разрешение 640.
- `data`: Путь к `YAML`-файлу, содержащему информацию о наборе данных.
- `epochs`: Количество эпох, на которые планируется обучение модели.
- `batch`: Размер пакета данных для загрузки, можно изменять в зависимости от доступной памяти на `GPU`.

- name: Наименование каталога для сохранения результатов обнаружения объектов в директории runs/detect.

После запуска скрипта модель тренируется в течение полтора часа и после этого готова к работе.



Результаты тренировки модели на входных данных

Запустим вывод на видео с помощью обученной модели YOLOv8 Medium. Для этого переключим режим работы на режим предсказания и укажем путь к весам модели. В качестве источника данных мы можем указать либо путь к папке с изображениями и видео, либо путь к конкретному файлу. Мы также можем выбрать иной источник данных, чтобы запустить вывод на видео.

Результаты



Обучение моделей обнаружения объектов, особенно при работе с маленькими объектами, может представлять сложности, даже при

наличии достаточного количества обучающих примеров. Это отражается в том, что обучение на протяжении 50 эпох оказалось недостаточным, так как кривые обучения для метрики средней точности (mAP) продолжали улучшаться.

Текст программы

Тренировка модели

```
from ultralytics import YOLO

# Load the model.
model = YOLO('yolov8m.pt')

# Training.
results = model.train(
    data='./pothole_v8.yaml',
    imgsz=640,
    epochs=50,
    batch=8,
    name='yolov8m_potholes'
)
```

Обнаружение объектов

```
import cv2
from ultralytics import YOLO

model = YOLO('runs/detect/yolov8m_potholes2/weights/best.pt')
model.track

source = 'video_1.mp4'
imgsz = 640
name = 'yolov8m_infer'
hide_labels = True

cap = cv2.VideoCapture(source)

if not cap.isOpened():
    print("Error: Could not open video.")
    exit()

fps = int(cap.get(cv2.CAP_PROP_FPS))
width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

fourcc = cv2.VideoWriter_fourcc(*'mp4v')
out = cv2.VideoWriter(f'{name}.mp4', fourcc, fps, (width, height))
```

```

while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break

    resized_frame = cv2.resize(frame, (imgsz, imgsz))

    results = model.predict(resized_frame)

    for result in results:
        boxes = result.boxes
        for box in boxes:
            x1, y1, x2, y2 = map(int, box.xyxy[0])
            confidence = box.conf[0].item()

            if hide_labels:
                label = f'{confidence:.2f}'
            else:
                label = f'{confidence:.2f}
{result.names[int(box.cls[0])}]}'

            cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 2)
            if not hide_labels:
                cv2.putText(frame, label, (x1, y1 - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 255, 0), 2)

            cv2.imshow('YOLOv8 Detection', frame)
            if cv2.waitKey(1) & 0xFF == ord('q'):
                break

        out.write(frame)

cap.release()
out.release()
cv2.destroyAllWindows()

```

Или командой

```

yolo task=detect mode=predict
model=runs/detect/yolov8m_poatholes2/weights/best.pt source=video_1.mp4
show=True imgsz=640 name=yolov8m_infer hide_labels=True

```

Глава 4. Алгоритмы работы с текстом и аудио

4.1. Анализ эмоциональной окраски аудио. Transformers и SpeechRecognition

Развитие технологий в области обработки звука и текста привело к созданию программы, позволяющей трансформировать аудиофайлы в текст и определять эмоциональную окраску этого текста.

Используемые библиотеки

Программа использует библиотеку `transformers`, предоставляющую тысячи предварительно обученных моделей для работы с текстом, зрением и аудио, а также интегрируется с популярными библиотеками глубокого обучения: `Jax`, `PyTorch` и `TensorFlow`. Кроме того, для распознавания речи используется библиотека `speech_recognition`, обеспечивающая поддержку нескольких движков и API для онлайн и оффлайн распознавания речи.

Основные функции программы

Программа предоставляет следующие ключевые функции:

- Преобразование аудио в текст
 - С использованием библиотеки `speech_recognition`, программа читает аудиофайлы и трансформирует речь в текст [57]. Этот этап позволяет дальнейший анализ эмоциональной окраски произнесенного монолога.
- Анализ тональности текста
 - С использованием предварительно обученной модели `seara/rubert-base-cased-russian-sentiment`, программа определяет эмоциональную окраску текста на русском языке. Данная модель способна выявлять негативные и позитивные состояния, а также нейтральные высказывания в тексте.

Таблица используемых функций из библиотек `Transformers` и `SpeechRecognition`

Функции	Описание
<code>pipeline(model="model_name")</code>	Создание конвейера для работы с предварительно обученной моделью для задач анализа текста.
<code>seara/rubert-base-cased-russian</code>	Имплементация модели RuBERT для анализа тональности текста на русском языке.
<code>Recognizer()</code>	Создание объекта для распознавания речи и работы с аудиофайлами.
<code>AudioFile(file_path)</code>	Открытие аудиофайла для последующей обработки.
<code>recognize_google(audio, language)</code>	Преобразование записанной речи из аудиофайла в текст.

Описание функций и их методов из библиотек Transformers и SpeechRecognition

Библиотека Transformers:

1. `pipeline(model="model_name")`
 - a. Описание метода: `pipeline()` создает конвейер для работы с предварительно обученной моделью. Основные параметры:
 - i. `model`: Параметр, который принимает строку с именем или тэгом предварительно обученной модели (например, "seara/rubert-base-cased-russian").
 - ii. `tokenizer`: Параметр, позволяющий указать специфический токенизатор для модели.
 - iii. `device`: Указание на использование устройства (CPU или GPU) для выполнения операций.
2. Методы для анализа текста:
 - a. `__call__(text)`: Вызывает модель для анализа переданного текста.
 - b. `predict(text)`: Предсказывает или анализирует тональность/эмоциональную окраску текста.

Библиотека SpeechRecognition:

1. `Recognizer()`
 - a. Описание метода: Создает объект `Recognizer` для работы с аудиофайлами и распознавания речи.
 - i. `recognize_google(audio, language)`: Метод для распознавания речи с использованием Google API.
 - ii. `record(source)`: Записывает аудио с указанного источника.
 - iii. `AudioFile(file_path)`: Метод для открытия аудиофайла для дальнейшей обработки.

2. Методы для работы с аудиофайлами:
 - a. `listen(source)`: Начинает прослушивание аудиопотока из указанного источника.
 - b. `recognize_google(audio, language)`: Преобразует записанное аудио в текст с использованием Google распознавания речи.
3. Параметры методов:
 - a. `language`: Указывает язык для распознавания речи.
 - b. `audio`: Аудиоданные, полученные из аудиофайла или потока записи.
 - c. `source`: Источник аудиофайла или потока записи.

Эти методы и параметры предоставляют базовый набор инструментов для работы с текстом и аудио, позволяя загружать модели для анализа текста и распознавания речи, а также преобразовывать аудио в текст для последующей обработки.

Примеры анализа аудиофайлов

Приведены примеры анализа двух аудиофайлов с негативным и позитивным монологами. После преобразования аудио в текст программа автоматически определяет тональность произнесенного содержания и выводит результат анализа.

Результат анализа негативного монолога: `[{'label': 'negative', 'score': 0.97336345911026}]`

Результат анализа позитивного монолога: `[{'label': 'positive', 'score': 0.5261187553405762}]`

Алгоритм программы для анализа аудиофайлов и определения тональности текста

1. Импорт библиотек
 - a. Программа начинает с импорта необходимых библиотек: `transformers` для анализа тональности текста и `speech_recognition` для работы с аудиофайлами и распознавания речи.
2. Загрузка предварительно обученной модели для анализа тональности текста
 - a. Программа загружает предварительно обученную модель из библиотеки `transformers`, позволяющую определять эмоциональную окраску текста на основе модели RuBERT.
3. Использование библиотеки для распознавания речи

- a. Создается объект Recognizer из библиотеки speech_recognition, предоставляющий методы для записи и распознавания речи из аудиофайлов.
4. Преобразование аудио в текст
 - a. Для каждого аудиофайла программа:
 - i. Открывает аудиофайл.
 - ii. Записывает аудиодорожку из файла.
 - iii. Преобразует записанное аудио в текст с помощью методов библиотеки для распознавания речи.
5. Определение тональности текста
6. Полученный текст передается на анализ предварительно обученной модели, которая определяет эмоциональную окраску текста.
7. Вывод результатов
 - a. Результаты анализа тональности выводятся в консоль или сохраняются для последующего использования.

Текст программы

```
# Импорт библиотек
from transformers import pipeline
import speech_recognition as sr

# Загрузка предварительно обученной модели для анализа тональности текста
sentiment_analysis = pipeline(model="seara/rubert-base-cased-russian-sentiment")

# Создаем объект Recognizer
recognizer = sr.Recognizer()

# Файлы с аудио
fileNeg = "vhs.wav"
filePos = "videoplayback.wav"

with sr.AudioFile(fileNeg) as source:
    audio = recognizer.record(source) # Записываем аудио из файла

    # Преобразуем записанное аудио в текст
    text = recognizer.recognize_google(audio, language="ru-RU")
    result = sentiment_analysis(text)
    print("Результат анализа негативного монолога: ", result)

with sr.AudioFile(filePos) as source:
    audio = recognizer.record(source) # Записываем аудио из файла

    # Преобразуем записанное аудио в текст
```

```
text = recognizer.recognize_google(audio, language="ru-RU")
result = sentiment_analysis(text)
print("Результат анализа позитивного монолога: ", result)
```

4.2. Разделение аудиофайла на вокал и аккомпанемент. Deezer Spleeter

Для разделения аудиофайла на составляющие компоненты – вокал и аккомпанемент была использована модель нейронной сети deezer spleeter. Spleeter - это Deezer библиотека разделения источников с предварительно обученными моделями написана на Python и использует Tensorflow [65]. Это делает это легко для обучения модели разделения источников (при условии, что у вас есть набор данных изолированных источников) и предоставляет уже обученная современная модель для выполнения различных видов разделения:

- Вокал (певческий голос)/разделение аккомпанемента (2 основы)
- Вокал/ударные/бас/другое разделение (4 основы)
- Вокал/ударные/бас/фортепиано/другое разделение (5 основ)

Модели с 2 и 4 выносами имеют высокую производительность и обучен на наборе данных musdb (<https://sigsep.github.io/datasets/musdb.html>). Spleeter также очень быстр, поскольку он может выполнять разделение аудиофайлов на 4 стебля в 100 раз быстрее, чем в реальном времени, при запуске на графическом процессоре.

Установка модели в проект

Установим проект spleeter в проект с помощью команды:

```
pip install spleeter
```

Из набора spleeter импортируем в проект модуль separator, отвечающий непосредственно за работу с разделяемыми аудиопотоками.

```
from spleeter.separator import Separator
```

Инициализируем объект `Selarator`, в параметре указав модель, на основе которой будет происходить разделение аудио (2, 4 или 5 основ).

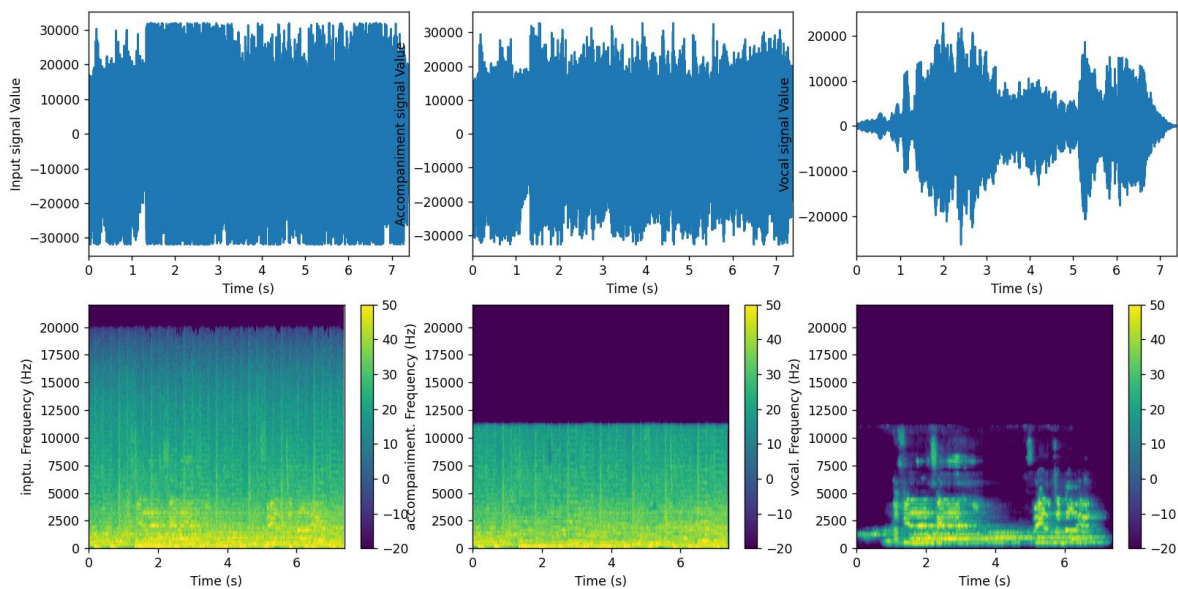
```
separator = Separator('spleeter:2stems')
```

Для начала работы модели и разделения аудиофайла на дорожки, можно воспользоваться командой, указав путь входного аудиофайла и выходной директории

```
separator.separate_to_file(input_audio, 'output')
```

Пример использования

Представленный ниже код разделяет аудиофайл на 2 дорожки – аккомпанемент и вокал и строит гистограммы частот результирующих файлов. Видно, что разделенные потоки урезаны в районе 11 kHz. Также сами разделенные файлы содержат небольшие примеси друг друга (вокал – содержит инструменты, аккомпанемент – вокал).



Амплитуда сигналов и гистограммы частот

```
from spleeter.separator import Separator
import matplotlib.pyplot as plt
import numpy as np
import wave
import sys
```

```

input_audio = 'input.wav'
output_audio_vocal = f'output/{input_audio.split(".")[0]}/vocals.wav'
output_audio_accompaniment =
f'output/{input_audio.split(".")[0]}/accompaniment.wav'

def get_channels_hist(input_file_path):
    wav_obj = wave.open(input_file_path, 'rb')
    sample_freq = wav_obj.getframerate()
    n_samples = wav_obj.getnframes()
    t_audio = n_samples / sample_freq
    n_channels = wav_obj.getnchannels()
    signal_wave = wav_obj.readframes(n_samples)
    signal_array = np.frombuffer(signal_wave, dtype=np.int16)
    l_channel = signal_array[0::2]
    r_channel = signal_array[1::2]
    times = np.linspace(0, n_samples / sample_freq, num=n_samples)
    return times, t_audio, l_channel, sample_freq

if __name__ == '__main__':
    separator = Separator('spleeter:2stems')
    separator.separate_to_file(input_audio, 'output')

    input_times, input_t_audio, input_l_channel, input_sample_freq =
get_channels_hist(input_audio)
    plt.figure(figsize=(15, 5))
    plt.subplot(2, 3, 1)
    plt.plot(input_times, input_l_channel)
    plt.ylabel('Input signal Value')
    plt.xlabel('Time (s)')
    plt.xlim(0, input_t_audio)

    accompaniment_times, accompaniment_t_audio, accompaniment_l_channel,
accompaniment_sample_freq = get_channels_hist(output_audio_accompaniment)
    plt.subplot(2, 3, 2)
    plt.plot(accompaniment_times, accompaniment_l_channel)
    plt.ylabel('Accompaniment signal Value')
    plt.xlabel('Time (s)')
    plt.xlim(0, accompaniment_t_audio)

    vocal_times, vocal_t_audio, vocal_l_channel, vocal_sample_freq =
get_channels_hist(output_audio_vocal)
    plt.subplot(2, 3, 3)
    plt.plot(vocal_times, vocal_l_channel)
    plt.ylabel('Vocal signal Value')
    plt.xlabel('Time (s)')
    plt.xlim(0, vocal_t_audio)

    plt.subplot(2,3,4)
    plt.specgram(input_l_channel, Fs=input_sample_freq, vmin=-20,

```

```

vmax=50)
    plt.ylabel('input. Frequency (Hz)')
    plt.xlabel('Time (s)')
    plt.xlim(0, input_t_audio)
    plt.colorbar()

    plt.subplot(2,3,5)
    plt.specgram(accompaniment_l_channel, Fs=accompaniment_sample_freq,
vmin=-20, vmax=50)
    plt.ylabel('accompaniment. Frequency (Hz)')
    plt.xlabel('Time (s)')
    plt.xlim(0, accompaniment_t_audio)
    plt.colorbar()

    plt.subplot(2,3,6)
    plt.specgram(vocal_l_channel, Fs=accompaniment_sample_freq, vmin=-20,
vmax=50)
    plt.ylabel('vocal. Frequency (Hz)')
    plt.xlabel('Time (s)')
    plt.xlim(0, vocal_t_audio)
    plt.colorbar()

plt.show()

```

4.3. Распознавание речи. SpeechRecognition

Распознавание речи — это процесс преобразования аудио речи в текст. Наиболее эффективные алгоритмы, используют при распознавании речи, методы и технологии из областей искусственного интеллекта и машинного обучения. Большинство этих алгоритмов самообучаемы и со временем совершенствуются. Программы использующие распознавания речи нашли широкое применение во многих сферах деятельности, например, голосовые помощники (ассистенты), управление голосом, голосовое задание тестовых сценариев [66] и др. Python — один из языков программирования, который предоставляет множество инструментов и библиотек для работы с распознаванием речи.

Одним из инструментов для распознавания речи является библиотека Speech Recognition [57]. Это инструмент для передачи речевых API от

компаний (google, microsoft, sound hound, ibm, а также pocketsphinx), который в отличие от остальных имеет возможность работы офлайн.

Реализуем распознавание речи с использованием библиотеки SpeechRecognition. Программа позволяет записывать голос с дальнейшим распознаванием, а также распознавать речь из существующего аудиофайла.

Список используемых функций

Функция	Описание
PyAudio()	Создание экземпляра PyAudio и инициализация системные ресурсы PortAudio
stop_stream()	Остановка записи
open(OUTPUT, 'wb')	Открыть файл с именем OUTPUT только для записи
setnchannels(CHAN)	Определить число каналов для записи
setsampwidth(p.get_sample_size(FRT))	Задание амплитуды записи файла
setframerate(RT)	Задание частоты записи файла
writeframes(b''.join(frames))	Запись аудио в файл
Recognizer()	Инициализация Recognizer
recognize_google(content, language="ru-RU")	Функция для распознавания содержимого в content на русском языке

Реализация программы

Для того, чтобы реализовать распознавание речи необходимо импортировать необходимые инструменты:

```
import speech_recognition as speech_r
import pyaudio
import wave
from pathlib import Path
```

В данной работе будет реализовано распознавание речи из существующего файла и возможность записи файла с дальнейшим распознаванием.

Для записи голоса необходимо выставить параметры записи аудиофайла:

```
CHUNK = 1024 # определяет форму аудио сигнала
FRT = pyaudio.paInt16 # шестнадцатитбитный формат задает значение амплитуды
CHAN = 1 # канал записи звука
RT = 44100 # частота
```

Запись аудио происходит с использованием инструменты PyAudio:

```
p = pyaudio.PyAudio()

stream = p.open(format=FRT, channels=CHAN, rate=RT, input=True,
frames_per_buffer=CHUNK) # открываем поток для записи
print("rec")
frames = [] # формируем выборку данных фреймов
for i in range(0, int(RT / CHUNK * REC_SEC)):
    data = stream.read(CHUNK)
    frames.append(data)
print("done")
stream.stop_stream() # останавливаем и закрываем поток
stream.close()
p.terminate()
```

Затем необходимо сохранить записанное аудио в нужном формате. В данной работе будет использован формат wav.

```
w = wave.open(OUTPUT, 'wb')
w.setnchannels(CHAN)
w.setsampwidth(p.get_sample_size(FRT))
w.setframerate(RT)
w.writeframes(b''.join(frames))
w.close()

sample = speech_r.WavFile(f'[ПУТЬ_ДО_ФАЙЛА]{название_файла}')
```

Распознавание речи происходит следующим образом:

```
r = speech_r.Recognizer()

with sample as audio:
    content = r.record(audio)
    print(r.recognize_google(content, language="ru-RU"))
```

Для того, чтобы распознавать аудио из существующего файла необходимо передавать в качестве параметра файл, как это было указано при сохранении записи голоса.

Результаты

1) Распознавание речи для записи голоса

Было записано название дисциплины, на которой была реализована данная курсовая работа. Результат распознавания указан в последней строке:

```
Choose online recognition (o) or recognition from existing audiofile (e)
o
Enter the name of output file:
subject
Enter the duration of a record:
5
гес
done
методы и средства обработки видеoinформации
```

2) Распознавание речи из существующего файла

В существующем файле содержалось название университета.

Результат распознавания указан в последней строке:

```
Choose online recognition (o) or recognition from existing audiofile (e)
e
You can recognize audio in these files:
C:\Users\potap\PycharmProjects\courseWorkNN\hi.wav
C:\Users\potap\PycharmProjects\courseWorkNN\love.wav
C:\Users\potap\PycharmProjects\courseWorkNN\output.wav
C:\Users\potap\PycharmProjects\courseWorkNN\Polytech.wav
C:\Users\potap\PycharmProjects\courseWorkNN\subject.wav
Enter file name:
Polytech
санкт-петербургский Государственный Политехнический Университет
```

Листинг программы

```
import speech_recognition as speech_r
import pyaudio
```

```

import wave
from pathlib import Path

CHUNK = 1024 # определяет форму ауди сигнала
FRT = pyaudio.paInt16 # шестнадцатитбитный формат задает значение
амплитуды
CHAN = 1 # канал записи звука
RT = 44100 # частота

print("Choose online recognition (o) or recognition from existing
audiofile (e)")

choose = input()

if choose == 'o':
    print('Enter the name of output file:')
    OUTPUT = input() + ".wav"
    print('Enter the duration of a record:')
    REC_SEC = int(input()) # длина записи
    p = pyaudio.PyAudio()

    stream =
p.open(format=FRT,channels=CHAN,rate=RT,input=True,frames_per_buffer=CHUN
K) # открываем поток для записи
    print("rec")
    frames = [] # формируем выборку данных фреймов
    for i in range(0, int(RT / CHUNK * REC_SEC)):
        data = stream.read(CHUNK)
        frames.append(data)
    print("done")
    stream.stop_stream() # останавливаем и закрываем поток
    stream.close()
    p.terminate()

    w = wave.open(OUTPUT, 'wb')
    w.setnchannels(CHAN)
    w.setsampwidth(p.get_sample_size(FRT))
    w.setframerate(RT)
    w.writeframes(b''.join(frames))
    w.close()

    sample =
speech_r.WavFile(f'C:\\Users\\potap\\PycharmProjects\\courseWorkNN\\{OUTP
UT}')
    r = speech_r.Recognizer()
    with sample as audio:
        content = r.record(audio)
        #r.adjust_for_ambient_noise(audio)
        print(r.recognize_google(content, language="ru-RU"))

```

```

else:
    print('You can recognize audio in these files:')
    directory = 'C:/Users/potap/PycharmProjects/courseWorkNN'
    pathlist = Path(directory).glob('*.*wav')
    for path in pathlist:
        print(path)
    print('Enter file name:')
    filename = input()
    sample =
speech_r.WavFile(f'C:\\Users\\potap\\PycharmProjects\\courseWorkNN\\{file
name}.wav')

r = speech_r.Recognizer()

with sample as audio:
    content = r.record(audio)
    # r.adjust_for_ambient_noise(audio)
    print(r.recognize_google(content, language="ru-RU"))

```

4.4. Анализ тональности текста. Keras TensorFlow

Для определения тональности текста была использована нейросеть Keras/TensorFlow. К сожалению, на сегодняшний день отсутствуют версии обученных моделей, которые выдают стабильно хороший результат для русскоязычных примеров текста без дополнительного обучения. Чтобы задать конкретную направленность данному проекту, научим нашу модель читать отзывы на что-либо и в зависимости от документа, поданного на вход, давать пользователю ответ, каких отзывов оказалось больше - положительных или отрицательных.

Ход работы

1. Создадим два CSV файла: train.csv и test.csv с некоторым набором отзывов разной тональности. Категорию положительных отзывов будем отмечать 1, отрицательных - 0. Чтобы эта информация учитывалась при обучении, формат, файла сделаем следующим: в первой строке содержится образец последующих строк с отзывами - "text, label", где text - сам отзыв, а label - строка, содержащая 0 или 1.

Эти файлы пригодятся нам для обучения и тестирования только что обученной модели соответственно.

2. Обучить модель на строчных литералах, представляющих слова русского языка, не получится, поэтому предлагается представить слова в виде токенов и векторизировать их. С этой целью пользователю предлагается реализовать функцию `encode_text(text)`, которую можно определить в файле `preprocess.py`:

```
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import SnowballStemmer
nltk.download('punkt')
nltk.download('stopwords')

stop_words = set(stopwords.words('russian'))
stemmer = SnowballStemmer('russian')

def encode_text(text):
    tokens = word_tokenize(text.lower())
    tokens = [stemmer.stem(word) for word in tokens if word.isalnum()]
    tokens = [word for word in tokens if not word in stop_words]
    return "_".join(tokens)
```

3. Создание и обучение модели описано в файле `train.py`, реализация которого представлена ниже. Для начала, загрузим тренировочные данные из файла `train.csv` и преобразуем их при помощи функции `encode_text`. Далее, воспользовавшись функцией `train_test_split` из пакета `sklearn.model_selection`, разделим данные на обучающую и тестовую выборки. Выполнив дополнительно некоторые преобразования с текстом при помощи пакета `src.train`, создадим модель `model`, которую в дальнейшем обучим с помощью функции `Sequential([...])`. Обучим нашу `model` при помощи функции `fit(...)` и сохраним её (`model.save('./models/sentiment_model.h5')`):

```
import pandas as pd
```

```

    sklearn.model_selection import train_test_split
from tensorflow.keras.layers import Dense, Embedding, GlobalAveragePooling1D
from tensorflow.keras.models import Sequential
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.preprocessing.text import Tokenizer
from sklearn.preprocessing import encode_text

# Data processing...
df = pd.read_csv('../data/train.csv')

# Text encoding
df['preprocessed_text'] = df['text'].apply(
    encode_text)

# Splitting data in training and testing selections
X_train, X_test, y_train, y_test = train_test_split(df['preprocessed_text'],
                                                    df['label'], test_size=0.2,
                                                    random_state=42)

# Tokenizer initializing
tokenizer = Tokenizer(num_words=5000)
tokenizer.fit_on_texts(X_train)
# Representing text in index sequences
X_train_seq = tokenizer.texts_to_sequences(X_train)
X_test_seq = tokenizer.texts_to_sequences(X_test)

# Building model...
model = Sequential([
    Embedding(input_dim=5000, output_dim=16),
    GlobalAveragePooling1D(),
    Dense(16, activation='relu'),
    Dense(1, activation='sigmoid')
])

# Compiling model...
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
# Preprocessing data (zero padding aligning sequences)
X_train_padded = pad_sequences(X_train_seq, maxlen=100)
X_test_padded = pad_sequences(X_test_seq, maxlen=100)

```

```
# Training model...
model.fit(X_train_padded, y_train, epochs=80, batch_size=32,
          validation_data=(X_test_padded, y_test))
```

```
# Saving model...
model.save('../models/sentiment_model.h5')
```

4. Теперь, когда наша модель обучена, будем предупредительными и выполним базовое тестирование точности результата, который способна выдать модель. Реализуем эту идею в файле `evaluate_model.py`:

```
import pandas as pd
from tensorflow.keras.preprocessing.text import Tokenizer
```

```
from tensorflow.keras.preprocessing.sequence import pad_sequences
```

```
from tensorflow.keras.models import load_model
from preprocess import encode_text
```

```
# Processing model...
model = load_model('../models/sentiment_model.h5')
```

```
# Processing Tokenizer...
tokenizer = Tokenizer(num_words=5000)
```

```
# Loading data for testing...
test_df = pd.read_csv('../data/test.csv')
```

```
# Encoding texts (if required)
```

```
test_df['preprocessed_text'] = test_df['text'].apply(encode_text)
```

```
# Representing text in index sequences
```

```
X_test_seq = tokenizer.texts_to_sequences(test_df['preprocessed_text'])
```

```
# Preprocessing data (adding zeroes for aligning sequences length)
```

```
X_test_padded = pad_sequences(X_test_seq, maxlen=100)
```

```
# Estimating power efficiency of model on testing selection
```

```
loss, accuracy = model.evaluate(X_test_padded, test_df['label'])
```

```
print("Test Accuracy: ", accuracy)
```

После первого обучения на наборе из 30 примеров модель способна выдавать результат со следующей точностью:

```
1/1  1s 893ms/step - accuracy: 0.5484 - loss: 0.6928
Test Accuracy: 0.5483871102333069

Process finished with exit code 0
```

Результат работы evaluate model

/Чтобы повысить точность результата пользователь может увеличить количество примеров/слоев обучения и других параметров, способных повлиять на процесс обучения модели./

5. Теперь, имея обученную модель, записанную в в файле `sentiment_model.h5`, мы можем приступить к написанию графического интерфейса, с которым впоследствии будет взаимодействовать пользователь. Реализуем данную идею в файле `analyzer.py`:

```
import pandas as pd
import tkinter as tk
from tkinter import filedialog, messagebox
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from preprocess import encode_text

def analyze_reviews ( ) :
    file _ path = filedialog . askopenfilename ( filetypes
    =[("CSV_файлы", "* . csv ") ] )
    if file _ path :

        df = pd . read_csv ( file _ path )
        # Processing trained model
        model = load_model ( ' . . / models / sentiment_model . h5 ' )

        # Tokenizer initializing
        tokenizer = Tokenizer ( num_words=5000 )
        tokenizer . fit_on_texts ( df [ ' text ' ] )

        # Preprocessing and preparing data for model
        preprocessed _ texts = df [ ' text ' ] . apply( encode_text )
        X_seq = tokenizer . texts_to_sequences ( preprocessed _ texts )
        X_padded = pad_sequences ( X_seq , maxlen=100 )

        # Getting predictions from the model
```

```

predictions = model . predict (X_padded)

# Counting the number of positive and negative reviews
positive _ reviews = sum( predictions > 0 . 5)
negative_reviews = len ( predictions ) - positive _ reviews

i f positive _ reviews > negative_reviews :
messagebox . showinfo ( " Result " , "There _ are _ more _ p o s i t i v e _
reviews . " )
e l i f positive _ reviews < negative_reviews :
messagebox . showinfo ( " Result " , "There _ are _ more _ negative _ reviews
. " )
else :
messagebox . showinfo ( " Result " ,
"There _ are _ an _ equal _ number _ of _ positive _ and _ negative _ reviews . " )

# C r e a t i n g d i a l o g w i n d o w
root = tk . Tk()
root . t i t l e ( "Review _ Analyzer")

# C r e a t i n g a b u t t o n f o r a C S V f i l e s e l e c t i o n
browse_button = tk . Button ( root , text="Choose _ CSV _ File " ,
command=analyze_reviews )
browse_button . pack ( pady=20)

# S t a r t i n g t h e m a i n e v e n t l o o p
root . mainloop ()

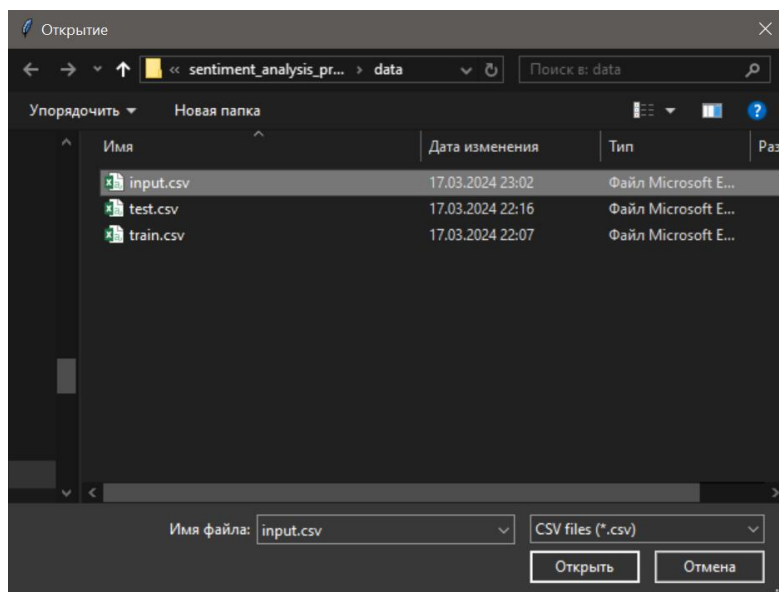
```

Использованные функции

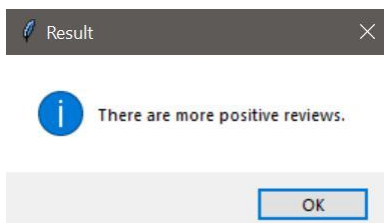
Функция	Описание
train_test_split()	Метод разделяет данные на тестовую и обучающую выборки
texts_to_sequences(X_train)	Метод преобразует текст в последовательности токенов
Sequential([])	Метод создает модель, которая впоследствии будет обучаться
compile()	Метод компилирует обучаемую модель
pad_sequences()	Метод добавляет нули для выравнивания длины последовательностей
fit()	Метод обучает модель
save()	Метод сохраняет обученную модель в файл
load_model()	Метод загружает модель из файла для дальнейшего её использования
predict()	Метод загружает предсказание модели в переменную

Запуск программы

После запуска программы пользователь увидит диалоговое окно, в котором программа запросит входной файл CSV формата. После нажатия кнопки пользователю необходимо выбрать файл.csv с отзывами, который соответствует шаблону text:



Следующим этапом является результат работы обученной нами модели, который программа выдаст в следующем диалоговом окне:



Таким образом, модель реализована в библиотеке TensorFlow, дообучена на тестовых данных и обернута в графический интерфейс для удобства взаимодействия с пользователем. Пользователь может адаптировать анализатор тональности текста под свои конкретные нужды.

4.5. Поиск похожих товаров на основе текстовых описаний. Universal Sentence Encoder

Целью данной работы является разработка системы поиска похожих товаров на основе текстовых описаний. Смысл проекта заключается в том,

чтобы вычислить близость между запросом пользователя и товарами на основе их векторных представлений с помощью модели Universal Sentence Encoder в фреймворке TensorFlow [67].

TensorFlow — открытая программная библиотека для машинного обучения, разработанная компанией Google для решения задач построения и тренировки нейронной сети с целью автоматического нахождения и классификации образов, достигая качества человеческого восприятия. Он предоставляет обширный набор инструментов для создания и настройки моделей машинного обучения.

Universal sentence encoder — это модель, которая кодирует текстовые данные в высокомерные векторы, которые можно использовать для различных задач NLP.

Описание работы программы

В данном примере была использована предварительно обученная модель Universal Sentence Encoder, которая предоставляет векторные представления для текстовых фрагментов.

Загрузка модели с помощью TensorFlow Hub:

```
import tensorflow_hub as hub
import numpy as np

# Загрузка предварительно обученной модели
model_url = "https://tfhub.dev/google/universal-sentence-encoder/4" #
Пример модели Universal Sentence Encoder
model = hub.load(model_url)
```

Загрузка и подготовка данных:

Для обучения и тестирования нейронной сети были использованы данные о товарах, представленные в виде текстовых описаний. В данном примере были заданы следующие описания товаров:

```
# Загрузка данных о товарах
product_descriptions = [
```

```
"Мягкая и удобная футболка",  
"Стильная кожаная куртка",  
"Прочные походные ботинки",  
"Уютный вязаный свитер"
```

]

Подготовка запроса и вычисление близости:

Для выполнения запроса по поиску похожих товаров, нужно подготовить векторное представление для запроса пользователя и вычислить близость между ним и векторными представлениями товаров.

В данном примере был задан следующий запрос:

```
# Запрос похожих товаров  
query = "Удобная толстовка"  
query_embedding = model([query])
```

Затем была вычислена близость между запросом и векторными представлениями товаров с использованием функции `np.inner()`:

```
# Расчет близости между запросом и товарами  
similarity_scores = np.inner(query_embedding, product_embeddings)
```

Сортировка и вывод результатов:

Для получения отсортированного списка похожих товаров, я использовал функцию `np.argsort()` для сортировки близости и индексов товаров и вывел результаты:

```
# Отсортированный список похожих товаров  
sorted_indices = np.argsort(similarity_scores, axis=1)[0][::-1]  
sorted_products = [product_descriptions[i] for i in sorted_indices]  
  
# Вывод результатов  
print("Query:", query)  
print("Similar Products:")  
for product in sorted_products:  
    print(product)
```

Таблица использованных функций

<code>hub.load():</code>	Функция загружает предварительно обученную модель с использованием TensorFlow Hub. В данном случае загружается модель "Universal Sentence Encoder" по заданному URL.
<code>np.inner():</code>	Функция numpy, которая вычисляет внутреннее произведение двух массивов. В данном коде используется для расчета

	близости между вектором запроса и векторами описаний товаров.
<code>np.argsort()</code> :	Функция <code>numpy</code> , которая возвращает индексы элементов массива в отсортированном порядке. В данном случае используется для получения индексов товаров, отсортированных по близости к запросу.

Этот код использует предварительно обученную модель `Universal Sentence Encoder` для представления описаний товаров и запроса в виде векторов. Затем два вектора сравниваются друг с другом с помощью внутреннего произведения. Результаты сравнения используются для сортировки описаний товаров по степени их сходства с запросом.

Результат работы программы:

```
Query: Comfortable cotton hoodie
Similar Products:
Cozy knit sweater
Soft and comfortable t-shirt
Stylish leather jacket
Durable hiking boots
```

В `Similar Products` сначала выводятся наиболее подходящие варианты

Листинг программы

```
import tensorflow_hub as hub
import numpy as np

# Загрузка предварительно обученной модели
model_url = "https://tfhub.dev/google/universal-sentence-encoder/4" #
Пример модели Universal Sentence Encoder
model = hub.load(model_url)

# Загрузка данных о товарах
product_descriptions = [
    "Мягкая и удобная футболка",
    "Стильная кожаная куртка",
    "Прочные походные ботинки",
    "Уютный вязаный свитер"
]

# Подготовка данных
product_embeddings = model(product_descriptions)

# Запрос похожих товаров
query = "Удобная толстовка"
query_embedding = model([query])
```

```
# Расчет близости между запросом и товарами
similarity_scores = np.inner(query_embedding, product_embeddings)

# Отсортированный список похожих товаров
sorted_indices = np.argsort(similarity_scores, axis=1)[0][::-1]
sorted_products = [product_descriptions[i] for i in sorted_indices]

# Вывод результатов
print("Query:", query)
print("Similar Products:")
for product in sorted_products:
    print(product)
```

4.6. Генерация текста с использованием GRU

Нейронные сети, основанные на рекуррентных моделях, позволяют создавать связные и осмысленные тексты. Они способны учитывать контекст предыдущих символов при генерации следующего. GRU (Gated Recurrent Unit) - это тип рекуррентной нейронной сети (RNN), который использует механизм вентиля для регулирования потока информации через сеть, что позволяет ей обучаться на более длинных последовательностях данных и избегать проблем с градиентами.

GRU состоит из двух вентилях: вентиль забывания, который определяет, какая информация будет забыта из памяти, и вентиль обновления, который решает, какая информация будет добавлена в память. Эти вентиля помогают GRU избегать проблемы затухания и взрывного увеличения градиента, что делает его более эффективным для обработки длинных последовательных данных.

Как работает GRU:

1. На входе GRU поступают последовательные данные (например, последовательность слов в предложении).
2. GRU обновляет свою память с учётом текущего входа и предыдущего состояния.
3. На выходе GRU можно получить прогнозы, например, следующее слово в последовательности или категорию, к которой относится ввод.

При обучении нейронной сети для генерации текста с использованием GRU, модель обучается на большом корпусе текста, чтобы изучить структуру и закономерности языка. Затем сеть используется для генерации новых текстов, предсказывая следующий символ на основе предыдущего контекста.

Реализуем нейронную сеть, которая генерирует последовательность символов, похожую на текст на примере произведения Шекспира.

Укрупненный алгоритм

Обучение модели. Вход: корпус текстовых данных; размер словаря; размеры встраивания и скрытого слоя; число эпох обучения. Выход: обученная модель GRU.

Основные шаги по обучению модели:

1. Предварительно обработать текстовые данные и создать словарь символов.
2. Встроить символы в векторы.
3. Создать рекуррентную сеть GRU с заданными размерами.
4. Обучить модель GRU на текстовых данных с использованием функции потерь перекрестной энтропии.
5. Сохранить обученную модель.

Использование обученной модели для генерации текста:

1. Загрузить обученную модель GRU.
2. Начать со стартового символа.
3. Использовать обученную модель для прогнозирования следующего символа в последовательности.
4. Выбрать следующий символ на основе прогноза модели, используя температуру для управления вариацией.
5. Добавить выбранный символ к сгенерированной последовательности.
6. Повторять шаги 2-5, пока не будет достигнута желаемая длина последовательности.
7. Вернуть сгенерированную последовательность.

Импорт библиотек

```
import tensorflow as tf
import numpy as np
import os
import time
```

Загрузка данных

```
path_to_file = tf.keras.utils.get_file('file.txt',
'https://storage.googleapis.com/download.tensorflow.org/data/shakespeare.txt')
```

Чтение датасета

```
text = open(path_to_file, 'rb').read().decode(encoding='cp1251')
vocab = sorted(set(text))
print(text[:1000])
```

Вывод:

```
First Citizen:
Before we proceed any further, hear me speak.
All:
Speak, speak.
First Citizen:
You are all resolved rather to die than to famish?
All:
Resolved. resolved.
First Citizen:
First, you know Caius Marcius is chief enemy to the people.
All:
We know't, we know't.
First Citizen:
Let us kill him, and we'll have corn at our own price.
Is't a verdict?
All:
No more talking on't; let it be done: away, away!
Second Citizen:
One word, good citizens.
```

```
First Citizen:
We are accounted poor citizens, the patricians good.
What authority surfeits on would relieve us: if they
would yield us but the superfluity, while it were
wholesome, we might guess they relieved us humanely;
but they think we are too dear: the leanness that
afflicts us, the object of our misery, is as an
inventory to particularise their abundance; our
sufferance is a gain to them Let us revenge this with
our pikes, ere we become rakes: for the gods know I
speak this in hunger for bread, not in thirst for revenge.
```

Векторизация текста

Создаем слои для преобразования символов в их идентификаторы и обратно. Функция `text_from_ids` принимает идентификаторы и возвращает строку, составленную из символов, соответствующих этим идентификаторам.

```
ids_from_chars = tf.keras.layers.StringLookup(vocabulary=list(vocab),
mask_token=None)

chars_from_ids = tf.keras.layers.StringLookup(
vocabulary=ids_from_chars.get_vocabulary(), invert=True, mask_token=None)

def text_from_ids(ids):
    return tf.strings.reduce_join(chars_from_ids(ids), axis=-1)
```

Модель. Создадим обучающие пакеты (batches)

Данный код выполняет несколько операций для подготовки данных для обучения модели на основе набора текстов. Первые две строки создают массив `all_ids`, который содержит числовые идентификаторы символов, полученные с помощью функции `ids_from_chars()`. Затем создается объект `ids_dataset`. Он представляет собой датасет TensorFlow, содержащий все числовые идентификаторы символов. Далее, задается переменная `seq_length`, которая определяет длину последовательности, которую мы будем использовать в обучении модели. В данном случае, устанавливается значение 100.

```
all_ids = ids_from_chars(tf.strings.unicode_split(text, 'UTF-8'))
ids_dataset = tf.data.Dataset.from_tensor_slices(all_ids)
```

```
seq_length = 100
```

Следующая строка кода создает `sequences` - датасет TensorFlow, в котором каждая последовательность содержит `seq_length + 1` числовых идентификаторов символов. Затем объявляется функция `split_input_target`, которая принимает последовательность числовых идентификаторов символов и дублирует и сдвигает ее, чтобы выровнять ввод и метку.

```
sequences = ids_dataset.batch(seq_length+1, drop_remainder=True)
```

```
def split_input_target(sequence):  
    input_text = sequence[:-1]  
    target_text = sequence[1:]  
    return input_text, target_text
```

Далее, создается `dataset`, в котором каждый элемент содержит входной и целевой текст. Затем определяются значения `BATCH_SIZE` и `BUFFER_SIZE`. `BATCH_SIZE` указывает размер пакета для обучения модели, а `BUFFER_SIZE` определяет размер буфера, используемый для перемешивания элементов в датасете. В конечном итоге, `dataset` преобразуется с использованием методов `shuffle()`, `batch()` и `prefetch()` для перемешивания данных, создания пакетов и оптимизации производительности.

```
dataset = sequences.map(split_input_target)
```

```
BATCH_SIZE = 64  
BUFFER_SIZE = 10000
```

```
dataset = (  
    dataset  
    .shuffle(BUFFER_SIZE)  
    .batch(BATCH_SIZE, drop_remainder=True)  
    .prefetch(tf.data.experimental.AUTOTUNE))
```

Создадим модель. Модель имеет 3 слоя:

-Embedding - входной слой. Он автоматически преобразует каждый идентификатор в соответствующий вектор.

-GRU - слой рекуррентной нейронной сети (GRU), который позволяет модели учитывать последовательность символов при генерации текста. Он принимает аргументы `rnn_units`, представляющий количество скрытых единиц GRU, `return_sequences=True` указывает, что слой должен возвращать последовательность символов вместо одного выхода, а

return_state=True указывает, что слой также должен возвращать состояние GRU.

-Dense - полносвязный слой, который применяется к каждому выходу GRU.

```
vocab_size = len(ids_from_chars.get_vocabulary())
embedding_dim = 256
rnn_units = 1024

class MyModel(tf.keras.Model):
    def __init__(self, vocab_size, embedding_dim, rnn_units):
        super().__init__(self)
        self.embedding = tf.keras.layers.Embedding(vocab_size, embedding_dim)
        self.gru = tf.keras.layers.GRU(rnn_units,
                                       return_sequences=True,
                                       return_state=True)
        self.dense = tf.keras.layers.Dense(vocab_size)

    def call(self, inputs, states=None, return_state=False,
             training=False):
        x = inputs
        x = self.embedding(x, training=training)
        if states is None:
            states = self.gru.get_initial_state(x)
        x, states = self.gru(x, initial_state=states, training=training)
        x = self.dense(x, training=training)

        if return_state:
            return x, states
        else:
            return x

model = MyModel(
    vocab_size=vocab_size,
    embedding_dim=embedding_dim,
    rnn_units=rnn_units)
```

Обучение модели

```
loss = tf.losses.SparseCategoricalCrossentropy(from_logits=True)

model.compile(optimizer='adam', loss=loss)
```

Настроим контрольные точки

```
checkpoint_dir = './training_checkpoints'
checkpoint_prefix = os.path.join(checkpoint_dir, "ckpt_{epoch}")
```

```
checkpoint_callback = tf.keras.callbacks.ModelCheckpoint(
    filepath=checkpoint_prefix,
    save_weights_only=True)
```

Запустим обучение

```
EPOCHS = 40
history = model.fit(dataset, epochs=EPOCHS,
                    callbacks=[checkpoint_callback])

Epoch 1/40
172/172 [=====] - 17s 53ms/step - loss: 2.7348
Epoch 2/40
172/172 [=====] - 10s 51ms/step - loss: 1.9993
. . . . .
Epoch 39/40
172/172 [=====] - 12s 59ms/step - loss: 0.4358
Epoch 40/40
172/172 [=====] - 12s 58ms/step - loss: 0.4304
```

Генерация текста

```
class OneStep(tf.keras.Model):
    def __init__(self, model, chars_from_ids, ids_from_chars,
                 temperature=1.0):
        super().__init__()
        self.temperature = temperature
        self.model = model
        self.chars_from_ids = chars_from_ids
        self.ids_from_chars = ids_from_chars

        skip_ids = self.ids_from_chars(['[UNK]'][:, None])
        sparse_mask = tf.SparseTensor(
            values=[-float('inf')]*len(skip_ids),
            indices=skip_ids,
            dense_shape=[len(ids_from_chars.get_vocabulary())])
        self.prediction_mask = tf.sparse.to_dense(sparse_mask)

    @tf.function
    def generate_one_step(self, inputs, states=None):
        input_chars = tf.strings.unicode_split(inputs, 'UTF-8')
        input_ids = self.ids_from_chars(input_chars).to_tensor()

        predicted_logits, states = self.model(inputs=input_ids,
                                              states=states,
                                              return_state=True)
        predicted_logits = predicted_logits[:, -1, :]
        predicted_logits = predicted_logits/self.temperature
        predicted_logits = predicted_logits + self.prediction_mask
```

```

    predicted_ids = tf.random.categorical(predicted_logits,
num_samples=1)
    predicted_ids = tf.squeeze(predicted_ids, axis=-1)
    predicted_chars = self.chars_from_ids(predicted_ids)
    return predicted_chars, states
one_step_model = OneStep(model, chars_from_ids, ids_from_chars)

```

Запустим одношаговую модель в цикле, чтобы сгенерировать текст.

```

states = None
next_char = tf.constant(['\n'])
result = [next_char]

for n in range(2000):
    next_char, states = one_step_model.generate_one_step(next_char,
states=states)
    result.append(next_char)

result = tf.strings.join(result)
print(result[0].numpy().decode('utf-8'), '\n\n')

```

Результат:

SOMERSET:

She is but force unto the gods for sight.
Dure sing an old tears that fexalish courted
Is, as how I found with the marshal' doth.

JULIET:

I'll gladly learn.

DUKE VINCENTIO:

That sorrow by mine honour, but by my entertainment
May hold my truateence and remove of reast,
And puts the lark and life reverence
Subs disgraced thy brother's points with custom, Diro!

ESCALUS:

I am none; being in my dreams to woo.

KING HENRY VI:

But did I ne'er he read her to any thing.

SICINIUS:

You will, not to sit.

AUFIDIUS:

Say, faith, I'll reat the news? mine enemies,
Resumes this jarn higher solerell'd with a

perfected wivery: tells when I deffed his
the angleadies Kate appear into us, if
I call you more.
Past he, Bohemil: and with a kind of man.

LADY CAPULET:
What is your wild,--

SICINIUS:
Hear me, I beseech you, sir,
If you have been deceived in my son's present.
Madam, my most misery.

Используемые функции

Get_file()	Задаёт URL входного файла
text_from_ids()	Принимает идентификаторы и возвращает строку, составленную из символов, соответствующих этим идентификаторам.
ids_from_chars()	Получение числовых идентификаторов символов
split_input_target()	Принимает последовательность числовых идентификаторов символов и дублирует и сдвигает ее, чтобы выровнять ввод и метку
shuffle(), batch() и prefetch()	Перемешивания данных, создания пакетов и оптимизация производительности.

Исходя из сгенерированного текста, видно, что модель проявляет некоторое понимание стилистики, но в то же время в тексте есть некоторая нелогичность и несвязность. Также мы видим, что модель может случайным образом внедрять имена. При малом числе эпох обучения модель еще не достаточно развита, чтобы составлять связные и грамматически корректные предложения. Обучение глубоких нейронных сетей требует большого количества данных и времени для настройки параметров модели. Чем дольше модель обучается, тем лучше она может понять и воспроизвести заданный стиль или словарный запас.

Итоговый текст программы

```

import tensorflow as tf
import numpy as np
import os
import time

path_to_file = tf.keras.utils.get_file('file.txt',
'https://storage.googleapis.com/download.tensorflow.org/data/shakespeare.
txt')
text = open(path_to_file, 'rb').read().decode(encoding='cp1251')
vocab = sorted(set(text))
print(text[:1000])

ids_from_chars = tf.keras.layers.StringLookup(
    vocabulary=list(vocab), mask_token=None)

chars_from_ids = tf.keras.layers.StringLookup(
    vocabulary=ids_from_chars.get_vocabulary(), invert=True,
mask_token=None)

def text_from_ids(ids):
    return tf.strings.reduce_join(chars_from_ids(ids), axis=-1)

all_ids = ids_from_chars(tf.strings.unicode_split(text, 'UTF-8'))
ids_dataset = tf.data.Dataset.from_tensor_slices(all_ids)

seq_length = 100

sequences = ids_dataset.batch(seq_length+1, drop_remainder=True)
def split_input_target(sequence):
    input_text = sequence[:-1]
    target_text = sequence[1:]
    return input_text, target_text

dataset = sequences.map(split_input_target)

BATCH_SIZE = 64
BUFFER_SIZE = 10000

dataset = (
    dataset
    .shuffle(BUFFER_SIZE)
    .batch(BATCH_SIZE, drop_remainder=True)
    .prefetch(tf.data.experimental.AUTOTUNE))

vocab_size = len(ids_from_chars.get_vocabulary())
embedding_dim = 256
rnn_units = 1024

class MyModel(tf.keras.Model):

```

```

def __init__(self, vocab_size, embedding_dim, rnn_units):
    super().__init__(self)
    self.embedding = tf.keras.layers.Embedding(vocab_size, embedding_dim)
    self.gru = tf.keras.layers.GRU(rnn_units,
                                   return_sequences=True,
                                   return_state=True)
    self.dense = tf.keras.layers.Dense(vocab_size)

    def call(self, inputs, states=None, return_state=False,
training=False):
        x = inputs
        x = self.embedding(x, training=training)
        if states is None:
            states = self.gru.get_initial_state(x)
        x, states = self.gru(x, initial_state=states, training=training)
        x = self.dense(x, training=training)

        if return_state:
            return x, states
        else:
            return x

model = MyModel(
    vocab_size=vocab_size,
    embedding_dim=embedding_dim,
    rnn_units=rnn_units)

loss = tf.losses.SparseCategoricalCrossentropy(from_logits=True)

model.compile(optimizer='adam', loss=loss)

checkpoint_dir = './training_checkpoints'
checkpoint_prefix = os.path.join(checkpoint_dir, "ckpt_{epoch}")

checkpoint_callback = tf.keras.callbacks.ModelCheckpoint(
    filepath=checkpoint_prefix,
    save_weights_only=True)

EPOCHS = 40
history = model.fit(dataset, epochs=EPOCHS,
callbacks=[checkpoint_callback])

class OneStep(tf.keras.Model):
    def __init__(self, model, chars_from_ids, ids_from_chars,
temperature=1.0):
        super().__init__()
        self.temperature = temperature
        self.model = model
        self.chars_from_ids = chars_from_ids

```

```

self.ids_from_chars = ids_from_chars

skip_ids = self.ids_from_chars(['[UNK]'][:, None])
sparse_mask = tf.SparseTensor(
    values=[-float('inf')]*len(skip_ids),
    indices=skip_ids,
    dense_shape=[len(ids_from_chars.get_vocabulary())])
self.prediction_mask = tf.sparse.to_dense(sparse_mask)

@tf.function
def generate_one_step(self, inputs, states=None):
    input_chars = tf.strings.unicode_split(inputs, 'UTF-8')
    input_ids = self.ids_from_chars(input_chars).to_tensor()

    predicted_logits, states = self.model(inputs=input_ids,
states=states,
return_state=True)
    predicted_logits = predicted_logits[:, -1, :]
    predicted_logits = predicted_logits/self.temperature
    predicted_logits = predicted_logits + self.prediction_mask

    predicted_ids = tf.random.categorical(predicted_logits,
num_samples=1)
    predicted_ids = tf.squeeze(predicted_ids, axis=-1)
    predicted_chars = self.chars_from_ids(predicted_ids)
    return predicted_chars, states
one_step_model = OneStep(model, chars_from_ids, ids_from_chars)

states = None
next_char = tf.constant(['\n'])
result = [next_char]

for n in range(2000):
    next_char, states = one_step_model.generate_one_step(next_char,
states=states)
    result.append(next_char)

result = tf.strings.join(result)
print(result[0].numpy().decode('utf-8'), '\n\n')

```

4.7. Извлечение из текста ответов на вопросы. YandexGPT

Зададим YandexGPT вопрос «Что такое YandexGPT?» и получим ответ:

YandexGPT - это нейронная сеть, разработанная компанией «Яндекс», способная генерировать текст на основе информации, полученной из

интернета. YandexGPT использует большие объёмы данных и алгоритмы машинного обучения для создания связного текста, который может быть релевантным и информативным.

Напишем программу, которая с помощью YandexGPT будет извлекать из текста ответы на вопросы. Текст будем помещать в текстовый файл. Для того, чтобы прекратить ввод вопросов и завершить выполнение программы, пользователю нужно будет ввести «Выйти».

Алгоритм работы

1. Создадим текстовый файл `text.txt`, в который запишем текст, по которому будем отвечать на вопросы.
2. Для начала работы с YandexGPT необходимо зарегистрировать аккаунт, войти в консоль управления, установить интерфейс командной строки Yandex Cloud, получить OAuth-токен и обменять его на IAM-токен.

Документация: https://yandex.cloud/ru/docs/iam/operations/iam-token/create#api_1

3. Создадим файл `config.py`, в котором проинициализируем переменные `IAM_TOKEN` и `FOLDER` значениями токена и идентификатором Yandex Cloud соответственно.
4. Создадим файл `core.py`, в котором реализуем код программы:
Напишем функцию `gpt()`: для получения ответа на вопрос, будем формировать и отправлять `http`-запрос, в теле которого будет задаваемый вопрос, а в заголовках – значения `IAM_TOKEN` и `FOLDER`. Рассмотрим подробнее тело запроса:

```
{
    "modelUri": "gpt://b1g7l0r7o090qaoltjku/yandexgpt-lite",
    "completionOptions": {
        "stream": False,
        "temperature": 0,
        "maxTokens": 100
    },
    "messages": [
        {
            "role": "system",
            "text": f"Текст: {f.read()}"
        },
        {
            "role": "assistant",
```

```

        "text": "Коротко ответь на вопрос по тексту:"
    },
    {
        "role": "user",
        "text": question
    }
]
}

```

Где:

modelUri — идентификатор модели, которая будет использоваться для генерации ответа. Параметр содержит идентификатор каталога Yandex Cloud или идентификатор дообученной в DataSphere модели.

completionOptions — параметры конфигурации запроса:

- **stream** — включает потоковую передачу частично сгенерированного текста. Принимает значения true или false.
- **temperature** — чем выше значение этого параметра, тем более креативными и случайными будут ответы модели. Принимает значения от 0 (включительно) до 1 (включительно). Значение по умолчанию: 0.3.
- **maxTokens** — устанавливает ограничение на выход модели в токенах. Максимальное число токенов генерации зависит от модели. Подробнее см. в разделе Квоты и лимиты в YandexGPT API.

messages — список сообщений, которые задают контекст для модели:

- **role** — роль отправителя сообщения:
- **user** — предназначена для отправки пользовательских сообщений к модели.
- **system** — позволяет задать контекст запроса и определить поведение модели.
- **assistant** — используется для ответов, которые генерирует модель.
- **text** — текстовое содержимое сообщения.

Используемые функции

Функция	Описание
post(url, headers, data)	Метод используют для отправки на сервер данных, которые передаются в теле запроса. Для этого при вызове requests.post() надо указать аргументы: url – url, по которому отправляется запрос, headers – заголовки запроса, data – тело запроса. Если для передачи данных используется формат JSON, вместо data можно указать json.
json()	Метод преобразует JSON в словарь Python, обеспечивая прямой доступ к ключам.

Пример работы программы:

В файле **text.txt** находится следующий текст:

Николай II, старший сын императора Александра III, родился 6 мая 1868 года. Николай получил домашнее образование, но в рамках большого гимназического курса. Лучшие преподаватели подготовили специальные программы, объединяющие университетский курс государственного и экономического отделений юридического факультета. Параллельно он получил образование и по линии Генерального штаба. Все обучение продолжалось 13 лет. Он в совершенстве владел французским, немецким и английским языками. Первые два года службы прошли у Николая в рядах Преображенского полка в должности младшего офицера.

В январе 1897 года была проведена денежная реформа, в результате установился золотой стандарт рубля. В это время шло бурное развитие промышленности, и правительство активно этому способствовало. Были приняты специальные меры, направленные на поощрение развития промышленности и банковского дела, на ускорение индустриализации страны. Все эти реформы неразрывно связаны с именем министра финансов, а позднее премьер-министра С.Ю. Витте.

Вопросы и ответы:

Задайте вопрос: Какая должность была у Николая в рядах полка?

В рядах полка Николай занимал должность младшего офицера.

Задайте вопрос: В каком полку проходила служба?

Первые два года службы Николая II прошли в рядах Преображенского полка в должности младшего офицера.

Задайте вопрос: Сколько лет обучался Николай?

Обучение Николая II продолжалось 13 лет.

Задайте вопрос: Какие реформы связаны с именем министра финансов?

С именем министра финансов С. Ю. Витте связаны денежная реформа (1897 г.), проведённая в январе 1897 года, и активное содействие бурному развитию промышленности и банковского дела.

Задайте вопрос: Какими языками владел Николай?

В совершенстве владел французским, немецким и английским языками.

Задайте вопрос: Выйти

```
Задайте вопрос: Какими языками владел Николай?
```

```
В совершенстве владел **французским, немецким** и **английским** языками.
```

```
Задайте вопрос: Сколько лет обучался Николай?
```

```
Обучение Николая II продолжалось **13 лет**.
```

```
Задайте вопрос: Какая должность была у Николая в рядах полка?
```

```
В рядах Преображенского полка Николай занимал должность младшего офицера.
```

```
Задайте вопрос: Выйти
```

```
Process finished with exit code 0
```

Текст программы

```
import json
import requests
from config import *

def gpt(question):
```

```

url =
'https://llm.api.cloud.yandex.net/foundationModels/v1/completion'

with open('text.txt', 'r', encoding='utf-8') as f:
    data = {
        "modelUri": "gpt://b1g7l0r7o090qaoltjku/yandexgpt-lite",
        "completionOptions": {
            "stream": False,
            "temperature": 0,
            "maxTokens": 100
        },
        "messages": [
            {
                "role": "system",
                "text": f"Текст: {f.read()}"
            },
            {
                "role": "assistant",
                "text": "Коротко ответь на вопрос по тексту:"
            },
            {
                "role": "user",
                "text": question
            }
        ]
    }
    resp = requests.post(url, headers={
        'Authorization': f'Bearer {IAM_TOKEN}',
        'x-folder-id': FOLDER
    }, data=json.dumps(data))
    if resp.status_code != 200:
        raise RuntimeError(
            'Invalid response received: code: {}, message: {}'.format(
                {resp.status_code}, {resp.text}
            )
        )
    return resp.json()['result']['alternatives'][0]['message']['text']
if __name__ == "__main__":
    while True:
        que = input('Задайте вопрос: ')
        if que == "Выйти":
            break
        print(gpt(que))

```

4.8. Преобразование текста в речь с возможностью перевода исходного текста. Eden AI, GTTS

Eden AI - это облачная платформа и набор API для искусственного интеллекта, предоставляющая различные инструменты и решения для разработчиков, включая обработку естественного языка, компьютерное зрение, автоматическое извлечение информации и многое другое. В программе с помощью этого AI подключаются Google AI и Amazon AI, которые включают в себя различные инструменты для обработки изображений, текста, речи и т. д.

GTTS- библиотека, используемая для синтеза речи из текста. Изначально была обучена с использованием передовых методов глубокого обучения, таких как нейронные сети, для синтеза естественной и качественной речи из текста.

Использованные библиотеки

gtts	Используется для синтеза речи из текста
os	Используется для взаимодействия с операционной системой
json	Используется для работы с данными в формате JSON
requests	Используется для отправки HTTP-запросов

Алгоритм

1. При необходимости перевод начального текста с помощью Eden ai, который использует google ai, amazon ai.
2. Озвучивание текста с помощью библиотеки gtts, которая обучена с помощью ИИ.
3. Воспроизведение записи.

Использованные функции

requests.post(url, json=payload, headers=headers)	Данная функция отправляет POST-запрос на указанный URL с переданными параметрами в формате JSON и заголовками
json.loads(response.text)	Функция json.loads() используется для десериализации строки JSON в объект Python
gTTS(text=translated_text,	Эта функция из библиотеки gtts создает объект

lang=language)	gTTS для синтеза речи из переданного текста на указанном языке.
myobj.save(audio_file):	Метод save() объекта gTTS используется для сохранения синтезированной речи в аудиофайл с указанным именем
os.system(f"start {audio_file}")	Функция os.system() используется для выполнения команды в операционной системе. В данном случае, используется для запуска аудиофайла воспроизведения при помощи системной команды start

Текст программы

```

from gtts import gTTS
import os
import json
import requests

headers = {"Authorization": "Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyX2lkIjoim2ExMTMyOWUtYzk0OS00
MDc5LWl5ZWMtNzQ3MmU5Y2E5ZjgzIiwidHlwZSI6ImFwaV90b2t1biJ9.JxVK5k2Ge9-
F9NzkcXGx-vLgkZ4e57p6qFFJ0z4bfWY"}

url = "https://api.edenai.run/v2/translation/automatic_translation"
payload = {
    "providers": "google,amazon",
    "source_language": "en",
    "target_language": "ru",
    "text": "How are you",
    "fallback_providers": ""
}
response = requests.post(url, json=payload, headers=headers)

result = json.loads(response.text)
print(result['google']['text'])

language = 'ru'
myobj = gTTS(text=result['google']['text'], lang=language)
myobj.save("welcome.mp3")
os.system("start welcome.mp3")

```

4.9. Предсказание популярности статьи по ее содержанию. TensorFlow

Целью данной работы является генерация предсказания для оценки популярности статьи в будущем на основе ее краткого содержания. Смысл

проекта заключается в том, чтобы заранее отбирать стоящие статьи по их аннотациям.

TensorFlow — открытая программная библиотека для машинного обучения, разработанная компанией Google для решения задач построения и тренировки нейронной сети с целью автоматического нахождения и классификации образов, достигая качества человеческого восприятия. Он предоставляет обширный набор инструментов для создания и настройки моделей машинного обучения.

Описание работы программы

В данной работе использованы инструменты библиотек: Pandas для чтения, составленного датасета, Scikit-learn для обработки данных из датасета, TensorFlow для обучения модели и предсказаний на ее основе.

Первым шагом был составлен датасет из вырезок статей и оценки их популярности:

```
text,popularity
*До конференции Mobius остался месяц, мы представляем Хабру её программу. И её вечно непросто поделить на блоки.
Понятно, что доклады «про iOS» и «про Android» интересны разным разработчикам, поэтому хабрапост хочется разделить пополам. Но как правильнее поступить
В общем, не уделяйте слишком много внимания тому, как всё разделено на части – лучше смотрите, о чём будут сами доклады.",0
*Технологии очаровывают, отталкивают, вызывают приступы скептицизма.. Люди по-разному относятся к любым новинкам, и это нормально. Но, как мы знаем, ист
Вот и сейчас в машинном обучении (ML) наблюдаются стремительные изменения и постоянное развитие. В этой статье мы рассмотрим последние тенденции в ML и
*Эрмитаж - это настоящий портал в мир искусства, в котором неподготовленный любитель прекрасного может потеряться и выбиться из сил в попытках найти ко
*Мы начнём с простого чат-бота, который может взаимодействовать только с одним документом; а закончим более продвинутым чат-ботом, который может взаимо
*Иногда возникает такие ситуации, когда нужно прочитать QR код, а смартфона под рукой нет. Что же делать? В голову приходит лишь попробовать прочитать
В статье рассмотрены базовые особенности QR кодов и методика дешифрования информации без использования вычислительных машин.",1
*Честное слово, это почти случайно получилось, что такая новость пришла к началу апреля и Дня космонавтики. Сегодня я расскажу о том, как история, кото
Мы искали Марс-3.
И мы нашли его! Прямо на Марсе, на дне гигантского кратера Птолемея, среди безжизненных пустошей и валунов.
О том, как мы это сделали, сегодняшний рассказ.", 1
*Наверное, нет человека, которого бы не раздражало, что самые важные пункты в договорах всегда пишутся «мелким шрифтом».
Но, зная об этом, раз за разом многие люди подписывают бумаги, не читая их внимательно, и платят комиссии, о которых не догадывались.
42-летний житель Воронежа нашел остроумный способ обратить эту особенность в свою пользу и теперь, пользуясь невнимательностью банкиров, собирает отс
*Пару месяцев назад нами (2Товарища и Антон Исайкин) была обнаружена уязвимость, присущая в основном большим интернет-проектам (вроде Рамблера, Мейла,
*Денис Крючков (deniskin) рассказал Roem.ru о том, что он смог выкупить акции «Тематических медиа» у Mail.Ru Group и теперь сможет управлять компанией
*Вы наконец-то можете сделать кое-что со своим старым LCD монитором, который завалился у Вас в гараже. Превратите его в шпионский монитор! Для всех вок
Всё что Вам нужно – это пара старых очков, нож для бумаги и растворитель для краски.", 1
```

Далее была проведена загрузка данных датасета и разделения их на две группы(для обучения и проверки):

```
# Загрузка данных
data = pd.read_csv('D:\pythonProject1\venv\dataset.csv')
# Загрузка данных
data = pd.read_csv('dataset.csv')
# Подготовка данных
```

```
X = data['text'] # Колонка с текстом статьи
y = data['popularity'] # Колонка с популярностью статьи
```

```
# Разделение данных на обучающую и тестовую выборки
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

Следующим этапом была произведена обработка данных перед обучением:

```
# Векторизация текста статьи
vectorizer = CountVectorizer()
X_train = vectorizer.fit_transform(X_train)
X_test = vectorizer.transform(X_test)
# Масштабирование признаков
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train.toarray())
X_test = scaler.transform(X_test.toarray())
```

Далее была создана нейронная сеть из слоёв Dense Layer, доступных в библиотеке TensorFlow:

```
# Создание нейронной сети
model = tf.keras.Sequential()
model.add(tf.keras.layers.Dense(64, activation='relu',
input_shape=(X_train.shape[1],)))
model.add(tf.keras.layers.Dense(64, activation='relu'))
model.add(tf.keras.layers.Dense(1, activation='sigmoid'))
```

Была обучена нейронная сеть и произведена оценка ее точности(из-за скромного по размерам датасета и большого количества анализируемого текста в каждой из позиций, точность составила 50%):

```
# Компиляция модели
model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
```

```
# Обучение модели
model.fit(X_train, y_train, epochs=30, batch_size=32,
validation_split=0.2)
```

```
# Оценка модели на тестовой выборке
loss, accuracy = model.evaluate(X_test, y_test)
print('Точность на тестовой выборке: {:.2f}%'.format(accuracy * 100))
```

Завершающим этапом был добавлен функционал для ввода пользователем текста, для которого требуется провести предсказание:

```

new_article = input("Введите текст статьи для предсказания: ")
new_article_vectorized = vectorizer.transform([new_article])
new_article_scaled = scaler.transform(new_article_vectorized.toarray())

```

```

# Выполнение предсказания
prediction = model.predict(new_article_scaled)
predicted_popularity = int(round(prediction[0][0]))

```

```

# Вывод предсказанной популярности
if predicted_popularity == 1:
    print("Статья предсказана как популярная")
else:
    print("Статья предсказана как не популярная")

```

Использованные функции

read_csv	Функция Pandas выполняет чтение данных из файла csv
train_test_split	Функция Scikit-learn, выполняет разбиение данных на обучающую и тестовую выборки
vectorizer. fit_transform	Эта функция Scikit-learn выполняет два действия. Сначала она "обучает" CountVectorizer на обучающих данных X_train, а затем преобразует тексты в векторное представление. Во время обучения CountVectorizer строит словарь всех уникальных слов в обучающих данных и присваивает каждому слову уникальный индекс. Затем каждый текст в X_train преобразуется в вектор, где каждый элемент вектора представляет количество вхождений соответствующего слова из словаря в данном тексте.
vectorizer. transform	После обучения CountVectorizer с помощью fit_transform, мы можем использовать функцию transform для преобразования тестовых данных X_test в векторное представление. Здесь CountVectorizer использует ранее построенный словарь и преобразует каждый текст в X_test в соответствующий вектор точно так же, как это было сделано для обучающих данных.

<code>scaler.fit_transform</code>	Эта функция Scikit-learn выполняет два действия. Сначала она "обучает" StandardScaler на обучающих данных X_train путем вычисления среднего значения и стандартного отклонения каждого признака в X_train. Затем она применяет масштабирование к данным путем вычитания среднего значения и деления на стандартное отклонение. X_train.toarray() используется для преобразования разреженной матрицы в обычный массив.
<code>scaler.transform</code>	После обучения StandardScaler с помощью fit_transform, мы можем использовать функцию transform для применения того же самого масштабирования к тестовым данным X_test. Здесь StandardScaler использует ранее вычисленные среднее значение и стандартное отклонение из обучающих данных для применения одинаковой трансформации к данным.
<code>model.add(tf.keras.layers.Dense)</code>	Функция TensorFlow добавляет слой полносвязной нейронной сети в модель. Dense означает, что каждый нейрон в данном слое будет полностью связан со всеми нейронами предыдущего слоя.
<code>model.compile</code>	Функция TensorFlow выполняет компиляцию модели нейронной сети добавляя оптимизатор, функцию потерь и метрики
<code>model.fit</code>	Функция TensorFlow выполняет обучение модели
<code>model.evaluate</code>	Функция TensorFlow для оценки точности обученной нейронной сети
<code>model.predict</code>	Функция TensorFlow для генерации предсказания на основе обученной нейронной сети

Программа выполняет обработку и подготовку данных полученного датасета, после чего на их основе обучает нейронную сеть. Обученная нейронная сеть позволяет пользователю получать предсказание о популярности статьи в будущем

Результат работы программы

```

Epoch 30/30
1/1 ██████████ 0s 30ms/step - accu
1/1 ██████████ 0s 15ms/step - accu
Точность на тестовой выборке: 50.00%
Введите текст статьи для предсказания: До ко
Понятно, что доклады «про iOS» и «про Androi
1/1 ██████████ 0s 29ms/step
Статья предсказана как не популярная

Process finished with exit code 0

```

Текст программы

```

import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.preprocessing import StandardScaler
import tensorflow as tf

# Загрузка данных
data = pd.read_csv('dataset.csv')

# Подготовка данных
X = data['text'] # Колонка с текстом статьи
y = data['popularity'] # Колонка с популярностью статьи

# Разделение данных на обучающую и тестовую выборки
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Векторизация текста статьи
vectorizer = CountVectorizer()
X_train = vectorizer.fit_transform(X_train)
X_test = vectorizer.transform(X_test)

# Масштабирование признаков
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train.toarray())
X_test = scaler.transform(X_test.toarray())

# Создание нейронной сети
model = tf.keras.Sequential()
model.add(tf.keras.layers.Dense(64, activation='relu',
input_shape=(X_train.shape[1],)))
model.add(tf.keras.layers.Dense(64, activation='relu'))
model.add(tf.keras.layers.Dense(1, activation='sigmoid'))

# Компиляция модели
model.compile(optimizer='adam', loss='binary_crossentropy',

```

```

metrics=['accuracy'])

# Обучение модели
model.fit(X_train, y_train, epochs=30, batch_size=32,
validation_split=0.2)

# Оценка модели на тестовой выборке
loss, accuracy = model.evaluate(X_test, y_test)
print('Точность на тестовой выборке: {:.2f}%'.format(accuracy * 100))

new_article = input("Введите текст статьи для предсказания: ")
new_article_vectorized = vectorizer.transform([new_article])
new_article_scaled = scaler.transform(new_article_vectorized.toarray())

# Выполнение предсказания
prediction = model.predict(new_article_scaled)
predicted_popularity = int(round(prediction[0][0]))

# Вывод предсказанной популярности
if predicted_popularity == 1:
    print("Статья предсказана как популярная")
else:
    print("Статья предсказана как не популярная")

```

4.10. Распознавание речи на русском языке с переводом на английский язык. Speech Recognition

Распознавание текста происходит с помощью нейросети SpeechRecognition. Нейросеть для распознавания речи (Speech Recognition) является типом искусственной нейронной сети, способной преобразовывать аудиосигналы, содержащие речь, в текст. Эта технология позволяет компьютерам интерпретировать и понимать человеческую речь, что может быть полезно для различных приложений, таких как голосовые помощники, системы транскрибирования, автоматические системы диктовки и другие.

Как работает нейросеть SpeechRecognition:

1. Подготовка аудиосигнала: сначала аудиосигнал, содержащий речь, подается на вход нейросети. Этот сигнал может быть предварительно обработан для улучшения качества звука и удаления шумов.

2. Извлечение признаков: нейросеть извлекает характеристики из аудиосигнала, которые помогут ей распознать речь. Эти признаки могут включать в себя спектральные характеристики звука, такие как частоты и интенсивность звуков.
3. Обучение модели: нейросеть обучается на большом объеме данных, содержащих аудиосигналы и соответствующий текст. В процессе обучения модель настраивается на распознавание различных звуков и слов.
4. Распознавание речи: после завершения обучения нейросеть может принимать новые аудиосигналы и пытаться распознать содержащуюся в них речь. На выходе получается текстовое представление распознанной речи.
5. Оценка результатов: результаты распознавания могут быть оценены с помощью метрик качества, таких как точность распознавания или степень ошибок.

Пошаговый алгоритм программы

1. Импортируем необходимые библиотеки
2. Запись голоса
3. Распознавание голоса и получение текста
4. Вывод полученного текста
5. Перевод текста
6. Вывод переведенного текста

Используемые функции:

Название функции	Описание
speech_recognition.Recognizer	Создает новый экземпляр распознавателя, который представляет собой набор функций распознавания речи.
speech_recognition.Microphone	Создает новый экземпляр микрофона, который представляет физический микрофон на компьютере.
adjust_for_ambient_noise	Динамически настраивает порог энергопотребления, используя звук из

	источника, чтобы учесть окружающий шум.
listen	Записывает одну фразу из источника в экземпляр аудиоданных, который затем возвращает.
recognize_google	Используется для распознавания речи с помощью Google Web Speech API.
translate	Переводит текст на нужный язык

Результат

```
Говорите...
Я люблю Россию
Запись завершена
"I love Russia.
```

Текст программы

```
import speech_recognition as sr
from translate import Translator

r = sr.Recognizer()
mic = sr.Microphone()

sr.LANGUAGE = 'ru-RU'
with mic as source:
    r.adjust_for_ambient_noise(source)
    print("Говорите...")
    audio = r.listen(source)
text = r.recognize_google(audio, language='ru-RU')

print(text)
print("Запись завершена")

translator = Translator(from_lang='ru', to_lang='en')
print(translator.translate(text))
```

4.11. Анализ эмоциональной окраски и субъективности текста.

TextBlob

Необходимо по данному отрывку текста определять его эмоциональную окраску (от крайне негативной до очень позитивной) и оценивать субъективность (от очень субъективной до очень объективной).

Нейросеть TextBlob - это библиотека для обработки текста на языке Python. Она предоставляет простой интерфейс для выполнения различных

задач обработки текста, таких как разделение на предложения и слова, определение частей речи, анализ тональности текста и многое другое. TextBlob построена поверх библиотеки Natural Language Toolkit (NLTK) и предоставляет более удобный API для многих из её функций. Кроме того, TextBlob включает в себя встроенные модели машинного обучения для выполнения этих задач, что делает её простой в использовании даже для тех, кто не имеет опыта в машинном обучении или обработке естественного языка.

Алгоритм программы

1. Импорт необходимых библиотек
2. Ввод текста или импорт из файла
3. Обработка текста
4. Вывод полученных результатов — эмоциональная окраска и объективность

Таблица используемых функций

Название функции	Описание
TextBlob(text)	Загружает текст для анализа предварительно обученной моделью
blob.sentiment	Получение кортежа из двух чисел: Sentiment(polarity, subjectivity)
print(sentiment)	Вывод результатов

Скриншоты

```

Sentiment(polarity=-1.0, subjectivity=0.9)
I hate you!
Эмоциональная окраска: Крайне негативная
Субъективность: Очень субъективно

Sentiment(polarity=0.6, subjectivity=1.0)
This film was nice.
Эмоциональная окраска: Скорее позитивная
Субъективность: Очень субъективно

Sentiment(polarity=0.375, subjectivity=0.875)
Python is a high-level, general-purpose programming language. Its design philosophy emphasizescode readability with the use of significant indentation.
Эмоциональная окраска: Нейтральная
Субъективность: Очень субъективно

```

```
Sentiment(polarity=0.0, subjectivity=0.0)
If you heat water, it boils
Эмоциональная окраска: Нейтральная
Субъективность: Очень объективно
```

Текст программы

```
from textblob import TextBlob
# Пример текста
text = "I hate you!"
# Создание объекта TextBlob
blob = TextBlob(text)
# Анализ тональности текста
sentiment = blob.sentiment
# Вывод результатов
print()
print(sentiment)
print(text)
# Оценка эмоциональной окраски словами с шагом 0.2
if sentiment.polarity > 0.8:
    print("Эмоциональная окраска: Очень позитивная")
elif sentiment.polarity > 0.6:
    print("Эмоциональная окраска: Довольно позитивная")
elif sentiment.polarity > 0.4:
    print("Эмоциональная окраска: Скорее позитивная")
elif sentiment.polarity > 0.2:
    print("Эмоциональная окраска: Нейтральная")
elif sentiment.polarity > 0:
    print("Эмоциональная окраска: Скорее негативная")
elif sentiment.polarity > -0.2:
    print("Эмоциональная окраска: Довольно негативная")
elif sentiment.polarity > -0.4:
    print("Эмоциональная окраска: Очень негативная")
else:
    print("Эмоциональная окраска: Крайне негативная")

# Оценка субъективности текста
if sentiment.subjectivity > 0.8:
    print("Субъективность: Очень субъективно")
elif sentiment.subjectivity > 0.6:
    print("Субъективность: Довольно субъективно")
elif sentiment.subjectivity > 0.4:
    print("Субъективность: Умеренно субъективно")
elif sentiment.subjectivity > 0.2:
    print("Субъективность: Скорее объективно")
else:
    print("Субъективность: Очень объективно")
text1 = "This film was nice."
blob1 = TextBlob(text1)
sentiment1 = blob1.sentiment
```

```

print()
print(sentiment1)

print(text1)

# Оценка эмоциональной окраски словами с шагом 0.2
if sentiment1.polarity > 0.8:
    print("Эмоциональная окраска: Очень позитивная")
elif sentiment1.polarity > 0.6:
    print("Эмоциональная окраска: Довольно позитивная")
elif sentiment1.polarity > 0.4:
    print("Эмоциональная окраска: Скорее позитивная")
elif sentiment1.polarity > 0.2:
    print("Эмоциональная окраска: Нейтральная")
elif sentiment1.polarity > 0:
    print("Эмоциональная окраска: Скорее негативная")
elif sentiment1.polarity > -0.2:
    print("Эмоциональная окраска: Довольно негативная")
elif sentiment1.polarity > -0.4:
    print("Эмоциональная окраска: Очень негативная")
else:
    print("Эмоциональная окраска: Крайне негативная")

# Оценка субъективности текста
if sentiment1.subjectivity > 0.8:
    print("Субъективность: Очень субъективно")
elif sentiment1.subjectivity > 0.6:
    print("Субъективность: Довольно субъективно")
elif sentiment1.subjectivity > 0.4:
    print("Субъективность: Умеренно субъективно")
elif sentiment1.subjectivity > 0.2:
    print("Субъективность: Скорее объективно")
else:
    print("Субъективность: Очень объективно")

text2 = ("Python is a high-level, general-purpose programming language.
Its design philosophy emphasizes"
        "code readability with the use of significant indentation.")

blob2 = TextBlob(text2)
sentiment2 = blob2.sentiment
print()
print(sentiment2)

print(text2)
# Оценка эмоциональной окраски словами с шагом 0.2
if sentiment2.polarity > 0.8:
    print("Эмоциональная окраска: Очень позитивная")
elif sentiment2.polarity > 0.6:

```

```

    print("Эмоциональная окраска: Довольно позитивная")
elif sentiment2.polarity > 0.4:
    print("Эмоциональная окраска: Скорее позитивная")
elif sentiment2.polarity > 0.2:
    print("Эмоциональная окраска: Нейтральная")
elif sentiment2.polarity > 0:
    print("Эмоциональная окраска: Нейтральная")
elif sentiment2.polarity > -0.2:
    print("Эмоциональная окраска: Довольно негативная")
elif sentiment2.polarity > -0.4:
    print("Эмоциональная окраска: Очень негативная")
else:
    print("Эмоциональная окраска: Крайне негативная")

# Оценка субъективности текста
if sentiment2.subjectivity > 0.8:
    print("Субъективность: Очень субъективно")
elif sentiment2.subjectivity > 0.6:
    print("Субъективность: Довольно субъективно")
elif sentiment2.subjectivity > 0.4:
    print("Субъективность: Умеренно субъективно")
elif sentiment2.subjectivity > 0.2:
    print("Субъективность: Скорее объективно")
else:
    print("Субъективность: Очень объективно")

```

4.12. Распознавание речи в видео и преобразование ее в текст.

SpeechRecognition

Программа получает на вход видео, с разговорной речью и выводит распознанный текст пользователю.

Использованные библиотеки:

- `SpeechRecognition` действует как оболочка для нескольких популярных речевых API и, таким образом, является чрезвычайно гибкой [57].
- `MoviePy` - это библиотека Python для редактирования видео: вырезания, объединения, вставки титров, компоновки видео (также известного как нелинейный монтаж), обработки видео и создания пользовательских эффектов.
- Модуль `os` в Python — это библиотека функций для работы с операционной системой. Методы, включенные в неё позволяют определять тип операционной системы, получать доступ к переменным окружения, управлять директориями и файлами.

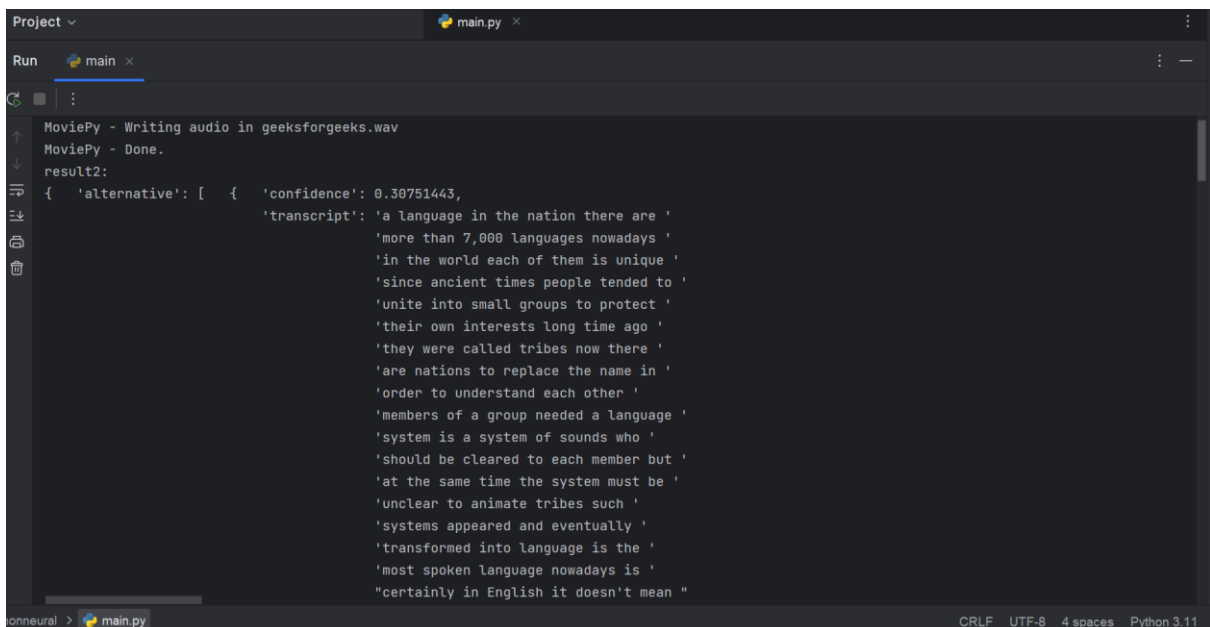
Алгоритм программы

1. Пользователь загружает необходимое видео.
2. Программа выделяет звуковой канал из видео.
3. Затем звук передается в нейронную сеть, а она в свою очередь возвращает распознанный текст.
4. Затем текст выводится пользователю.

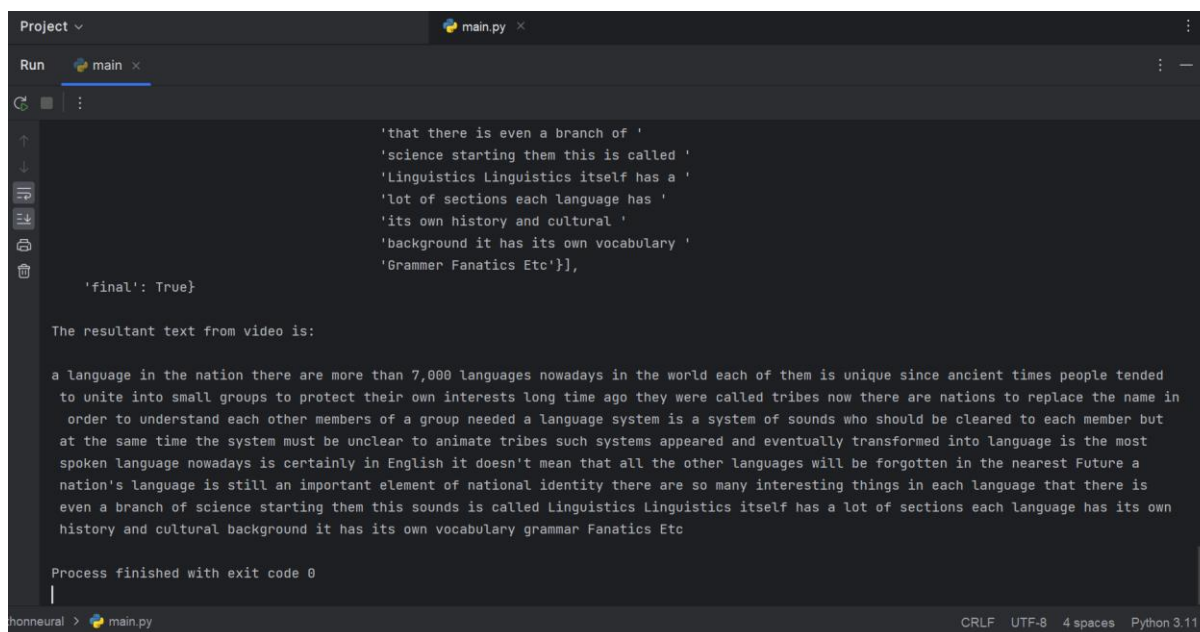
Используемые функции

<code>mp.VideoFileClip()</code>	Функция из библиотеки <code>moviepy</code> , которая загружает видеофайл "geeksforgeeks.mp4" в переменную <code>video</code> .
<code>audio.write_audiofile()</code>	Метод объекта <code>audio_file</code> , который записывает аудио из видеофайла в формате WAV в файл "geeksforgeeks.wav".
<code>sr.Recognizer()</code>	Создание объекта <code>Recognizer</code> из библиотеки <code>speech_recognition</code> для распознавания речи.
<code>sr.AudioFile()</code>	Контекстный менеджер, который открывает аудиофайл "geeksforgeeks.wav" для обработки.
<code>r.record()</code>	Метод объекта <code>r</code> , который записывает аудиоданные из файла "geeksforgeeks.wav" в переменную <code>data</code> .
<code>r.recognize_google()</code>	Метод объекта <code>r</code> , используемый для распознавания текста из аудиоданных с помощью Google Speech Recognition API.

Скриншоты работы программы



```
Project ▾ main.py ×
Run main ×
MoviePy - Writing audio in geeksforgeeks.wav
MoviePy - Done.
result2:
{ 'alternative': [ { 'confidence': 0.30751443,
                    'transcript': 'a language in the nation there are '
                                   'more than 7,000 languages nowadays '
                                   'in the world each of them is unique '
                                   'since ancient times people tended to '
                                   'unite into small groups to protect '
                                   'their own interests long time ago '
                                   'they were called tribes now there '
                                   'are nations to replace the name in '
                                   'order to understand each other '
                                   'members of a group needed a language '
                                   'system is a system of sounds who '
                                   'should be cleared to each member but '
                                   'at the same time the system must be '
                                   'unclear to animate tribes such '
                                   'systems appeared and eventually '
                                   'transformed into language is the '
                                   'most spoken language nowadays is '
                                   'certainly in English it doesn't mean "
```



```
Project main.py
Run main
'that there is even a branch of '
'science starting them this is called '
'Linguistics Linguistics itself has a '
'lot of sections each language has '
'its own history and cultural '
'background it has its own vocabulary '
'Grammar Fanatics Etc'}],
'final': True}

The resultant text from video is:

a language in the nation there are more than 7,000 languages nowadays in the world each of them is unique since ancient times people tended
to unite into small groups to protect their own interests long time ago they were called tribes now there are nations to replace the name in
order to understand each other members of a group needed a language system is a system of sounds who should be cleared to each member but
at the same time the system must be unclear to animate tribes such systems appeared and eventually transformed into language is the most
spoken language nowadays is certainly in English it doesn't mean that all the other languages will be forgotten in the nearest Future a
nation's language is still an important element of national identity there are so many interesting things in each language that there is
even a branch of science starting them this sounds is called Linguistics Linguistics itself has a lot of sections each language has its own
history and cultural background it has its own vocabulary grammar Fanatics Etc

Process finished with exit code 0
honneutral main.py CRLF UTF-8 4 spaces Python 3.11
```

Ссылка на использованное в программе видео

<https://www.youtube.com/watch?v=sdsrWGiSUTs&t=20s>

Текст программы

```
import moviepy.editor as mp
import speech_recognition as sr
import os
os.environ["GOOGLE_APPLICATION_CREDENTIALS"] =
"C:/Users/Алина/PycharmProjects/pythonneural/credits.json"
video = mp.VideoFileClip("geeksforgeeks.mp4")
audio_file = video.audio
audio_file.write_audiofile("geeksforgeeks.wav")
r = sr.Recognizer()
with sr.AudioFile("geeksforgeeks.wav") as source:
    data = r.record(source)
text = r.recognize_google(data)
print("\nThe resultant text from video is: \n")
print(text)
```

4.13. Генерация музыки и звуков по описанию. MusicGen

Приводится пример использования библиотеки Audiostream [68]. Данная библиотека использует нейросети для генерации звуков и музыки. В рамках библиотеки доступно множество моделей, в том числе MusicGen, используемый в рамках этой работы. Данная модель была натренирована с помощью более чем 20 тысяч часов музыки. MusicGen способен как генерировать музыку и звуки по описанию, так и по

описанию и заданной ему мелодии, однако время выполнения программы во втором случае значительно возрастает. Также, разработчики MusicGen предоставляют удобный для использования API, который и был применен для выполнения данной работы и интерфейс, который позволяет пользоваться библиотекой без написания дополнительного кода

Для корректной работы библиотеки необходима установка ffmpeg, а также всех библиотек, расположенных в файле requirements.txt. Для генерации музыки в библиотеке доступны уже натренированные модели: small, melody, medium, large и другие.

Алгоритм работы

1. Получение указанной модели, в данном случае «small», хранится в переменной model
2. Установка параметра duration=10, который означает что создаваемая композиция будет длиться 10 секунд
3. С помощью метода input получение от пользователя описания для композиции, запись его в переменную descriptions
4. Вызов метода generate, который отвечает непосредственно за генерацию аудиофайла
5. Запись полученной аудиозаписи в аудиофайл

Ввод данных

Ввод осуществляется с помощью метода input, который выводит текст “Description of the song:”, после чего пользователь должен ввести текст, описывающий композицию, которую он хочет сгенерировать с помощью данной программы.

Используемые функции

MusicGen.get_pretrained	Вызывает уже натренированную модель
model.set_generation_params	Установка параметров генерации для модели,

	от которой вызывается метод
model.generate	Генерация звуков или музыки на основе текста с помощью модели, от которой вызывается метод

Текст программы

```
from audiocraft.models import MusicGen
from audiocraft.data.audio import audio_write
model = MusicGen.get_pretrained('small')
model.set_generation_params(duration=10)
descriptions = input("Description of the song: ")
wav = model.generate(descriptions)
audio_write(f'{descriptions}', wav[0].cpu(), model.sample_rate,
strategy="loudness", loudness_compressor=True)
```

4.14. Вербализация изображения. Yolov7 и RuGPT3

Программа осуществляет генерацию текстового описания к изображению. Процесс можно разделить на 2 этапа: классификация объектов, представленных на фотографии, с помощью предобученной модели нейросети Yolov7; генерация текстового описания по ключевым словам с помощью предобученной модели RuGPT3.

В нынешней реализации программы не удалось подобрать оптимальные параметры и запрос для текстовой модели, поэтому генерируемый текст получается излишне художественным. Если подобрать более оптимальный запрос или дообучить модель для задачи вербализации изображения, то её можно будет использовать для генерации тифлокомментариев. Тифлокомментарий — это комментарий, который описывает содержимое медиаконтента для людей с нарушением зрения.

YOLOv7 - алгоритм обнаружения объектов в реальном времени, основанный на использовании нейронных сетей глубокого обучения. Он является улучшенной версией YOLO (You Only Look Once) алгоритма и представляет собой семейство архитектур, которые обеспечивают

высокую скорость и точность обнаружения объектов. Алгоритм YOLO делит изображение на сетку и предсказывает ограничивающие рамки объектов, а также их классы непосредственно в этой сетке. Это позволяет YOLO выполнять обнаружение объектов в реальном времени, так как алгоритм требует только одного прохода через нейросеть для прогнозирования. Модель была обучена на наборе COCO (Common Objects In Context). Всего она может распознавать 40 классов объектов. Для работы с предобученной моделью YOLO используется модуль deep learning, реализованный в openCV.

RuGPT-3 – модель нейросети, которая была обучена на русскоязычных текстах разных стилей: на энциклопедиях, социальных сетях, художественной и бизнес-литературе. Для работы с предобученной текстовой моделью ruGPT используется библиотека transformers из фреймворка PyTorch.

Алгоритм работы программы

1. Загрузка предобученной модели YOLOv7
2. Загрузка предобученной модели RuGPT-3 Medium
3. Загрузка классов COCO, которые может распознавать модель YOLOv7
4. Предобработка изображения (кадрирование 640x640)
5. Подача изображения на вход модели
6. Применение к результату метода подавления немаксимумов для уменьшения количества повторно-классифицированных объектов:
 - а) Вычисление вероятности присутствия в заданном регионе для каждого обнаруженного объекта;
 - б) сравнение вероятности с соседними результатами;
 - в) устранение перекрытий регионов с высоким значением, остаётся только один максимум;

7. Перевод классифицированных объектов с английского на русский, т.к. ruGPT – русскоязычная языковая модель.

8. Составление запроса

9. Генерация текста по запросу. Параметры генерации:

1. `input_ids`: Токенизированное представление входного текста в виде тензора PyTorch
2. `max_length`: Максимальная длина генерируемой последовательности текста.
3. `min_length`: Минимальная длина генерируемой последовательности текста.
4. `do_sample`: Флаг, указывающий, следует ли использовать метод сэмплирования при генерации текста.
5. `num_return_sequences`: Количество возвращаемых последовательностей текста.
6. `no_repeat_ngram_size`: Запрет на генерацию одинаковых последовательностей токенов длиной 2 и более
7. `top_k`: Количество наиболее вероятных слов, которые модель может рассмотреть при генерации следующего слова.
8. `top_p`: Порог для отсеивания наиболее вероятных слов.
9. `temperature`: Параметр, контролирующий степень "творчества" модели при генерации текста.
10. `num_beams`: Количество "лучей" при генерации текста с использованием алгоритма "beam search"
11. `early_stopping`: Прекращение генерации текста, когда модель сгенерирует текст, который оценивается как "завершенный"

Beam search - алгоритм, который используется при генерации текста. Его суть: на каждом шаге генерации модель рассматривает не только одно лучшее предсказание, а несколько альтернативных вариантов, так называемых "лучей". Модель следит за несколькими наилучшими кандидатами и продолжает их развивать, выбирая на каждом шаге те кандидаты, которые наилучшим образом соответствуют цели.

10. Вывод сгенерированного текста.

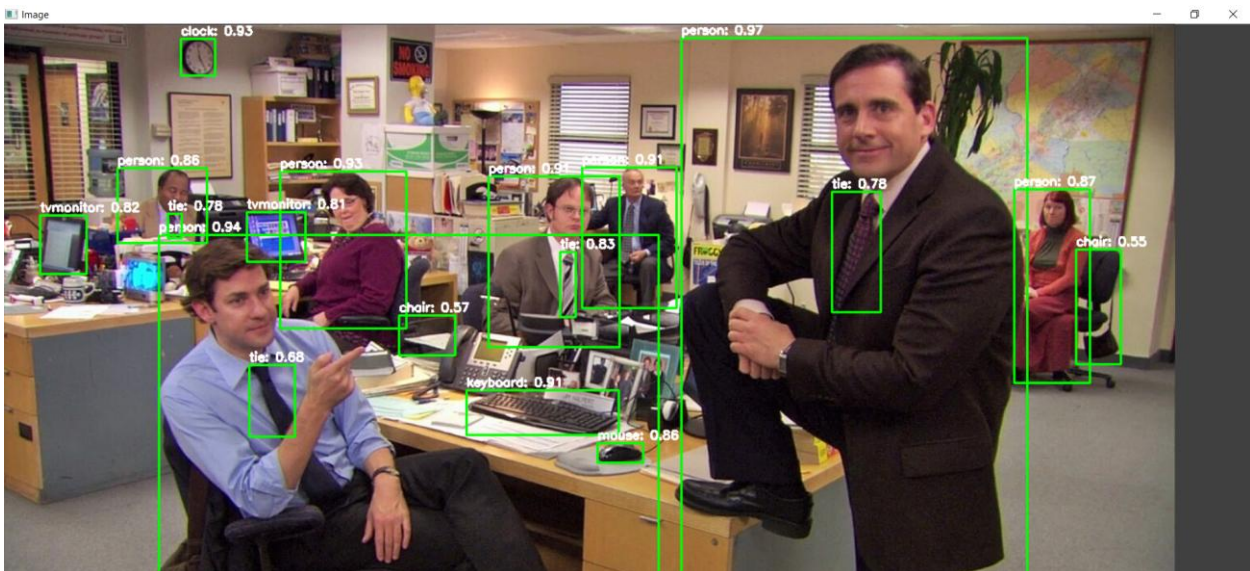
11. Вывод исходного изображения с выделенными цветом секторами, содержащими объекты.

Используемые функции

Название функции	Описание
<code>cv2.dnn.readNet("yolov7.weights", "yolov7.cfg")</code>	Загрузка предобученной модели Yolov7
<code>GPT2LMHeadModel.from_pretrained("sberbank-ai/rugpt3medium_based_on_gpt2")</code>	Создание объекта модели языкового моделирования RuGPT-3
<code>GPT2Tokenizer.from_pretrained("sberbank-ai/rugpt3medium_based_on_gpt2")</code>	Создание токенизатора для модели RuGPT-3
<code>image = cv2.imread("img2.jpg")</code>	Чтение изображения
<code>cv2.dnn.blobFromImage(image, 1/255, (640, 640), swapRB=True, crop=False)</code>	Предобработка изображения
<code>net.setInput(blob)</code>	Подача изображения на вход модели Yolov7
<code>net.forward(net.getUnconnectedOutLayersNames())</code>	Запуск прямого распространения данных

	через модель
<code>cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 2)</code>	Заключение распознанных объектов в прямоугольники
<code>cv2.putText(image, label, (x, y - 5), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 255, 255), 2)</code>	Добавление подписи к распознанным объектам
<code>translator.translate(string, dest='ru').text.lower()</code>	Перевод названия классифицированных объектов на русский язык
<code>model.generate(input_ids, max_length=250, min_length=100, do_sample = True, num_return_sequences=3, no_repeat_ngram_size=2, top_k=100, top_p=0.97, temperature=0.7, num_beams=5, early_stopping=True)</code>	Генерация текста по заданному запросу с помощью модели RuGPT-3
<code>tokenizer.decode(output[0], skip_special_tokens=True)</code>	Декодирование текста
<code>cv2.imshow("Image", image)</code>	Вывод изображения в окно

Результат



Изображение после прогонки через YoloV7

```
[5]: generate_description(labels)
```

Однажды человек, часы, клавиатура, мышь, галстук, телевизионный монитор, стул оказались в комнате, в которой на полу лежал человек. Человек был мёртв.

Человек лежал на спине, голова его была повернута в одну сторону, а туловище в другую, руки были вытянуты вдоль тела, ноги согнуты в коленных и тазобедренных суставах, глаза закрыты, рот был приоткрыт, изо рта шла пена, которая с текала по подбородку и капала с подбородка на пол. человек был без одежды, на нём не было ни обуви, ни каких-либо других предметов, которые могли бы свидетельствовать о том, что он был человеком. в комнате было очень холодно, и человек не мог согреться, поэтому он лежал неподвижно и не двигался. через некоторое время человек открыл глаза и посмотрел на часы. на циферблате были стрелки, показывавшие 10 часов утра, человек встал с кровати, оделся и вышел из комнаты, закрыв за собой дверь. он шёл по коридору, который вёл в его комнату, он не знал, куда он идёт, но ему было всё равно. вскоре он вышел на улицу и сел на скамейку, стоящую недалеко от дома.

Текст, который сгенерировал RuGPT по распознанным объектам

Текст программы

```
import cv2
import numpy as np
import torch
from transformers import GPT2LMHeadModel, GPT2Tokenizer
from googletrans import Translator
net = cv2.dnn.readNet("yolov7.weights", "yolov7.cfg")
model = GPT2LMHeadModel.from_pretrained("sberbank-
ai/rugpt3medium_based_on_gpt2")
tokenizer = GPT2Tokenizer.from_pretrained("sberbank-
ai/rugpt3medium_based_on_gpt2")

# Загружаем классы, которые может распознавать модель YOLO
classes = []
with open("coco.names", "r") as f:
    classes = [line.strip() for line in f.readlines()]
image = cv2.imread("img2.jpg")
h, w, _ = image.shape

# Предобработка изображения
blob = cv2.dnn.blobFromImage(image, 1/255, (640, 640), swapRB=True,
crop=False)

# Подаем изображение на вход модели
net.setInput(blob)
outs = net.forward(net.getUnconnectedOutLayersNames())

# Обработка результатов
bboxes = []
confidences = []
class_ids = []
labels = []
added_labels = []
for out in outs:
    for detection in out:
        scores = detection[5:]
        class_id = np.argmax(scores)
        confidence = scores[class_id]
        if confidence > 0.5:
            center_x = int(detection[0] * w)
            center_y = int(detection[1] * h)
            width = int(detection[2] * w)
            height = int(detection[3] * h)
            x = int(center_x - width / 2)
            y = int(center_y - height / 2)

            bboxes.append([x, y, width, height])
            confidences.append(float(confidence))
```

```

        class_ids.append(class_id)
indices = cv2.dnn.NMSBoxes(bboxes, confidences, score_threshold=0.1,
nms_threshold=0.2)
for i in indices:

    box = bboxes[i]
    x, y, w, h = box[0], box[1], box[2], box[3]
    cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 2)
    label = f"{classes[class_ids[i]]}: {confidences[i]:.2f}"
    labels.append(classes[class_ids[i]])
    cv2.putText(image, label, (x, y - 5), cv2.FONT_HERSHEY_SIMPLEX, 0.5,
(255, 255, 255), 2)

def generate_description(labels):
    translator = Translator()
    russian_labels = [translator.translate(string,
dest='ru').text.lower() for string in labels]
    input_text = f"Однажды " + " ,".join(russian_labels) + " оказались в
"

    input_ids = tokenizer.encode(input_text, return_tensors='pt')
    output = model.generate(input_ids, max_length=250, min_length=100,
do_sample = True, num_return_sequences=3, no_repeat_ngram_size=2,
top_k=100, top_p=0.97, temperature=0.7, num_beams=5, early_stopping=True)

    generated_text = tokenizer.decode(output[0],
skip_special_tokens=True)
    print(generated_text)
generate_description(labels)
cv2.imshow("Image", image)
cv2.waitKey(0)
cv2.destroyAllWindows()

```

4.15. Конвертер римских чисел. Tesseract

Эта программа использует веб-камеру для распознавания римских цифр, отображаемых на экране или листе бумаги. Она конвертирует распознанный текст в соответствующее числовое значение. Реализация использованием библиотек OpenCV и Tesseract.

Алгоритм работы

Программа инициализирует веб-камеру для захвата видеопотока; входит в бесконечный цикл, в котором выполняет следующие действия:

2. Считывается один кадр с веб-камеры.

3. Кадр конвертируется в изображение в оттенках серого.
4. Применяется пороговое преобразование, чтобы отделить текст от фона, чтобы убрать шум и заполнить пробелы в символах.
5. Библиотека PyTesseract используется для распознавания текста на обработанном изображении.
6. Распознанный текст, представляющий римские цифры, преобразуется в числовое значение.
7. Распознанный текст и соответствующее число выводятся на экран поверх видеопотока.
8. Цикл завершается при нажатии клавиши "q".

Программа освобождает ресурсы веб-камеры и закрывает все окна.

Используемые библиотеки

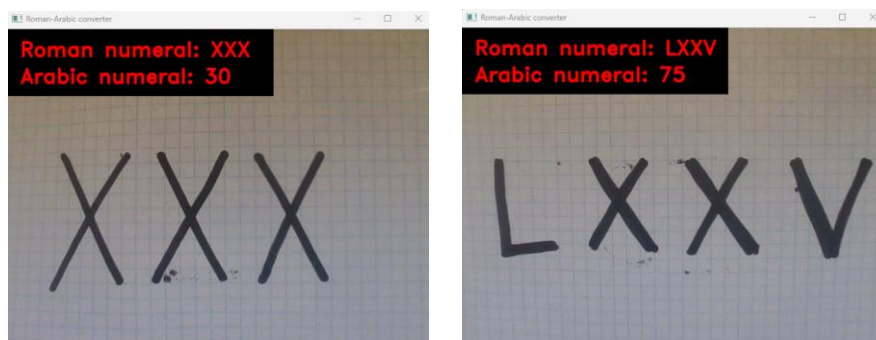
OpenCV — это библиотека с открытым исходным кодом, которая предоставляет широкий набор функций для обработки изображений, анализа видео и машинного обучения.

Tesseract — это программное обеспечение с открытым исходным кодом для оптического распознавания символов (OCR). Оно преобразует отсканированные изображения документов, такие как PDF и фотографии, в редактируемый текст. Tesseract использует продвинутые алгоритмы обработки изображений и машинного обучения для распознавания символов. Он поддерживает более 100 языков и имеет высокую точность распознавания. Tesseract использует сверточную нейронную сеть (CNN) под названием Long Short-Term Memory (LSTM) для распознавания текста. CNN обучена на большой базе данных изображений текста, чтобы распознавать отдельные символы в различных условиях. LSTM-сеть затем обрабатывает последовательность символов и прогнозирует вероятные слова.

Используемые в проекте методы:

cv2.cvtColor	Преобразует изображение из одного цветового пространства в другое
cv2.threshold	Применяет пороговое преобразование к изображению для повышения контрастности
cv2.morphologyEx	Применяет морфологическую операцию закрытия к изображению для удаления шума
pytesseract.imageToString	Распознает текст на изображении с помощью PyTesseract
cv2.putText	Надписывает текст на изображение
cv2.imshow	Отображает изображение в окне
cv2.waitKey	Ожидает нажатия клавиши
cv2.destroyAllWindows	Закрывает все открытые окна

Результат работы



Текст программы

```

from PIL import Image
import cv2
import pytesseract
import numpy as np

def roman_to_int(s):
    roman = {'I': 1, 'V': 5, 'X': 10, 'L': 50, 'C': 100, 'D': 500, 'M':
1000}
    result = 0
    for char in s:
        if char in roman:
            result += roman[char]
    return result
cap = cv2.VideoCapture(0)

while True:
    ret, frame = cap.read()
    if not ret:
        break

    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

```

```

    thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV |
cv2.THRESH_OTSU)[1]

    kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (3, 3))
    morph = cv2.morphologyEx(thresh, cv2.MORPH_CLOSE, kernel)
    text = pytesseract.image_to_string(morph, config='--psm 10 --oem 3')
    text = text[:-1]

    number = roman_to_int(text.strip())
    text_window = np.zeros((100, 400, 3), np.uint8)

    cv2.putText(text_window, f"Roman numeral: {text}", (20, 40),
cv2.FONT_HERSHEY_DUPLEX, 1.0, (0, 0, 255), 2,
                cv2.LINE_AA)
    cv2.putText(text_window, f"Arabic numeral: {number}", (20, 80),
cv2.FONT_HERSHEY_DUPLEX, 1.0, (0, 0, 255), 2,
                cv2.LINE_AA)

    frame [0:100, 0:400] = text_window
    cv2.imshow('Roman-Arabic converter', frame)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
cap.release()
cv2.destroyAllWindows()

```

4.16. Генерация текстовых описаний изображений. CLIP

CLIP (Contrastive Language-Image Pretraining) - это модель от OpenAI, которая обучается на парах изображений и текстов, позволяя ей связывать изображения с их текстовыми описаниями [69]. В программе используется CLIP для генерации текстовых описаний для изображений. Программа загружает изображение, обрабатывает его с помощью модели CLIP, сравнивает эмбединги изображения с эмбедингами predetermined текстовых описаний и выбирает наиболее подходящее описание.

Алгоритм работы

1. Загрузка предобученной модели и процессора: CLIPModel и CLIPProcessor загружаются из библиотеки transformers.
2. Предобработка изображения: Изображение преобразуется в формат, подходящий для модели.

3. Получение эмбеддингов изображения: Изображение подается в модель для извлечения его эмбеддингов.
4. Предобработка текстовых описаний: Набор predetermined текстовых описаний преобразуется в тензоры.
5. Получение эмбеддингов текстов: Текстовые описания подаются в модель для извлечения их эмбеддингов.
6. Сравнение эмбеддингов: Эмбеддинги изображения и текстов сравниваются, и находится наиболее похожее текстовое описание.

Использованные функции

Название функции	Описание
<code>CLIPModel.from_pretrained("openai/clip-vit-base-patch32")</code>	Загружает предобученную модель CLIP.
<code>CLIPProcessor.from_pretrained("openai/clip-vit-base-patch32")</code>	Загружает процессор для предварительной обработки изображений и текстов.
<code>model.get_image_features(**inputs)</code>	Получает эмбеддинги изображения.
<code>processor(images=image, return_tensors="pt")</code>	Преобразует изображение в тензор.
<code>processor(text=descriptions, return_tensors="pt", padding=True)</code>	Преобразует текстовые описания в тензоры.
<code>model.get_text_features(**text_inputs)</code>	Получает эмбеддинги текстов.
<code>torch.matmul(image_features, text_features.T)</code>	Вычисляет сходство между эмбеддингами изображения и текстов.

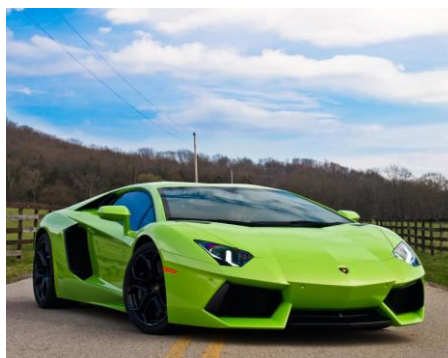
Пример работы программы

Вход:



Выход:

a photo of a dog



a photo of a car

Текст программы

```
import torch
from PIL import Image
from transformers import CLIPProcessor, CLIPModel

# Функция для генерации описания изображения
def generate_image_caption(image_path):
    # Загрузка модели и процессора
    model = CLIPModel.from_pretrained("openai/clip-vit-base-patch32") #
    # Загружаем предобученную модель CLIP
    processor = CLIPProcessor.from_pretrained("openai/clip-vit-base-
patch32")

    # Открытие изображения и его предварительная обработка
    image = Image.open(image_path) # Открываем изображение с помощью PIL
    inputs = processor(images=image, return_tensors="pt") # Преобразуем
изображение в формат, подходящий для модели CLIP (тензор)

    # Получение эмбедингов изображения
    with torch.no_grad():
        image_features = model.get_image_features(**inputs)
    # Предопределенные текстовые описания
    descriptions = [
        "a photo of a cat",
        "a photo of a dog",
        "a photo of a bird",
        "a photo of a computer",
        "a photo of a car",
        "a photo of a human",
        "a photo of a tree",
        "i don't know what it is"
    ]
    # Предобработка текстов
    text_inputs = processor(text=descriptions, return_tensors="pt",
padding=True) # Преобразуем текстовые описания в формат, подходящий для
модели CLIP (тензор)

    # Получение эмбедингов текстов
    with torch.no_grad():
        text_features = model.get_text_features(**text_inputs) #
    Получаем эмбединги текстов

    # Нахождение ближайшего текста к эмбедингу изображения
    similarities = torch.matmul(image_features, text_features.T) #
    Вычисляем сходство между эмбедингами изображения и текстов (скалярное
произведение)
```

```

    best_match_idx = similarities.argmax().item() # Находим индекс
текста с наибольшим сходством

    return descriptions[best_match_idx] # Возвращаем текстовое описание
с наибольшим сходством

# Основная функция
def main():
    image_path = input("Введите путь к изображению: ")
    caption = generate_image_caption(image_path)
    print("Сгенерированное описание изображения:", caption)

if __name__ == "__main__":
    main()

```

4.17. Генерация и синтез речи. GPT

Программа проводит генерацию речи с использованием GPT-2 и синтез с использованием pyttsx3.

GPT-2 (Generative Pre-trained Transformer 2) – это языковая модель, разработанная OpenAI, которая способна генерировать высококачественные тексты, основанные на предоставленных промптах. Модель использует трансформерную архитектуру, которая включает механизмы самовнимания для обработки входных данных и создания релевантных продолжений текста. GPT-2 обучена на большом корпусе текстов, что позволяет ей понимать и генерировать тексты на различных темах с высокой степенью когерентности.

Для синтеза речи в проекте используется библиотека pyttsx3. Pyttsx3 позволяет преобразовывать текст в аудиоформат, озвучивая его с использованием синтезированного голоса.

Алгоритм

1. Загрузка и настройка модели GPT-2. Производится загрузка предобученной модели GPT-2 и соответствующего токенизатора из библиотеки transformers. После загрузки модель настраивается для генерации текста.

2. Генерация текста. Основной функцией для генерации текста является `generate_text`. Она принимает начальный промпт и возвращает сгенерированный текст. Для этого функция кодирует промпт в тензор, подает его на вход модели и декодирует результат. Используются параметры `max_length` для ограничения длины текста и `num_beams` для улучшения качества генерации за счет метода `beam search`.
3. Синтез речи. Для озвучивания сгенерированного текста используется функция `speak_text`, которая инициализирует синтезатор речи `pyttsx3` и озвучивает переданный текст.

Основные функции

Функция	Описание
<code>generate_text(prompt, max_length=50, num_beams=5)</code>	Описание: Функция для генерации текста на основе заданного промпта Входные данные: <code>prompt (str)</code> : Начальная строка текста. <code>max_length (int)</code> : Максимальная длина генерируемого текста. <code>num_beams (int)</code> : Количество лучей для поиска Выходные данные: Сгенерированный текст (<code>str</code>).
<code>speak_text(text)</code>	Функция для озвучивания текста. Входные данные: <code>text (str)</code> : Текст для озвучивания. Выходные данные: озвучивает текст
<code>tokenizer.encode(prompt, return_tensors='pt')</code>	Кодирует начальную строку в тензор.
<code>torch.ones(inputs.shape, dtype=torch.long, device=inputs.device)</code>	Создаёт маску внимания
<code>model.generate(...)</code>	Генерация текста с использованием различных параметров, таких как <code>do_sample</code> , <code>top_k</code> , <code>top_p</code> , <code>no_repeat_ngram_size</code> и <code>early_stopping</code> .
<code>tokenizer.decode(outputs[0], skip_special_tokens=True)</code>	Декодирование сгенерированного текста и удаление специальных токенов.

Результаты работы

Для демонстрации работы системы был использован следующий промпт: "Once upon a time". На его основе была сгенерирована текстовая последовательность, которая затем была озвучена.

Generated Text:

Once upon a time there was a war between the two peoples, the Jews and the Arabs. The Arabs were the first to conc

Speaking Text:

Текст программы

```
import torch
from transformers import GPT2LMHeadModel, GPT2Tokenizer
import pyttsx3

# Загрузка предобученной модели
model_name = 'gpt2'
model = GPT2LMHeadModel.from_pretrained(model_name)
tokenizer = GPT2Tokenizer.from_pretrained(model_name)

# pad_token_id
model.config.pad_token_id = model.config.eos_token_id

# Функция для генерации текста
def generate_text(prompt, max_length=50, num_beams=5):
    inputs = tokenizer.encode(prompt, return_tensors='pt')
    attention_mask = torch.ones(inputs.shape, dtype=torch.long,
device=inputs.device)

    outputs = model.generate(
        inputs,
        attention_mask=attention_mask,
        max_length=max_length,
        num_beams=num_beams,
        do_sample=True,
        top_k=50,
        top_p=0.95,
        no_repeat_ngram_size=2,
        early_stopping=True,
        pad_token_id=tokenizer.eos_token_id
    )
    text = tokenizer.decode(outputs[0], skip_special_tokens=True)
    return text

# Функция для озвучивания текста
def speak_text(text):
    engine = pyttsx3.init()
```

```

engine.say(text)
engine.runAndWait()

if __name__ == "__main__":
    prompt = "Once upon a time"
    generated_text = generate_text(prompt)
    print("Generated Text:")
    print(generated_text)

    print("Speaking Text:")
    speak_text(generated_text)

```

4.18. Аннотирование текста с помощью T5Tokenizer

Программа проводит написание аннотаций (также суммаризация или аннотирование текста) по тексту с использованием T5Tokenizer - класса из библиотеки transformers, который предоставляет функциональность для токенизации текста для моделей, основанных на архитектуре T5 (Text-To-Text Transfer Transformer) на языке Python.

Трансформер – это архитектура нейросетей, позволяющая извлекать из текста довольно объёмную информацию. Благодаря этой архитектуре модели типа BERT достаточно хорошо понимают тексты, а модели типа GPT весьма правдоподобно их генерируют. Text-to-text означает, что модель T5 принимает на вход тексты и "читает" их энкодером (как BERT), а потом "пишет" декодером новые тексты и отдаёт на выход. В частности, применяется дообученная модель T5Tokenizer, T5ForConditionalGeneration.

T5 — это модель кодера-декодера, предварительно обученная для многозадачной работы с неконтролируемыми и контролируемыми задачами, для которой каждая задача преобразуется в текстовый формат. T5 хорошо работает с различными задачами, добавляя префиксы к входным данным, соответствующим каждой задаче, например, для перевода: «translate English to German: ...», для обобщения: «summary: ...».

Предварительное обучение включает в себя как обучение под наблюдением, так и самостоятельное обучение. Обучение под наблюдением проводится по последующим заданиям, предоставляемым тестами GLUE и SuperCLUE (преобразуя их в текстовые задания, как описано выше).

При самостоятельном обучении используются поврежденные токены, путем случайного удаления 15% токенов и замены их отдельными токенами sentinel (если несколько последовательных токенов помечены для удаления, вся группа заменяется одним токеном sentinel). На вход кодировщика поступает искаженное предложение, на вход декодера - исходное предложение, а целью являются удаленные токены, разделенные их контрольными токенами.

В T5 используются относительные скалярные вложения. Заполнение входных данных кодировщика может быть выполнено слева и справа.

Различия между моделями T5 обычно связаны с их размером, архитектурой и количеством параметров. Вот краткое описание:

- t5-small — самая маленькая версия модели T5 с наименьшим количеством параметров. Обычно это хороший выбор для быстрой проверки и экспериментов на небольших наборах данных.
- t5-base — базовая версия модели T5, которая обычно обладает хорошим балансом между производительностью и размером. Она часто используется в приложениях среднего размера и на различных задачах NLP.
- t5-large — более крупная версия модели T5, которая имеет больше параметров и может обеспечить лучшую производительность на более сложных задачах. Она может потреблять больше ресурсов и времени для обучения и выполнения.

- t5-3b и t5-11b — ещё более крупные версии модели T5, которые содержат 3 миллиарда и 11 миллиардов параметров соответственно. Они обычно используются в крупномасштабных проектах или исследованиях, где требуется максимальная производительность и масштабируемость.

Для решения представленной задачи используется версия t5-base, т. к. тексты для тестирования среднего размера — соответственно, не нужен слишком большой объём памяти. Взята версия стандартной модели T5 из пакета transformers библиотеки huggingface.

Используемые функции

Название функции	Описание
tokenizer = T5Tokenizer.from_pretrained("t5-base")	Загрузка предварительно обученного T5
model = T5ForConditionalGeneration.from_pretrained("t5-base")	model – это сама нейросеть; tokenizer – это часть модели, ответственная за сопоставление текстов со словарём: разбиение текстов на числовые токены и сбор текстов из токенов обратно;
inputs = tokenizer.encode("summarize: " + text, return_tensors="pt", max_length=512, truncation=True)	Токенизация текста — процесс разделения входного текста на отдельные элементы, называемые токенами и воспринимаемые нейросетью.
summary_ids = model.generate(inputs, max_length=150, min_length=10, length_penalty=2.0, num_beams=4, early_stopping=True)	Основная функция, которая выполняет нужные действия в зависимости от параметров. В данном случае — генерация аннотации на основе входных данных.
summary = tokenizer.decode(summary_ids[0], skip_special_tokens=True)	Декодирование идентификаторов аннотации обратно в текст с использованием токенизатора.

Примеры выполнения программы

Исходный текст

Python is a high—level general-purpose programming language with dynamic strict typing and automatic memory management, aimed at improving developer productivity, code readability

and quality, as well as ensuring the portability of programs written in it. The language is completely object-oriented in the sense that everything is objects. An unusual feature of the language is the indentation of code blocks. The syntax of the language core is minimalistic, due to which in practice there is rarely a need to refer to documentation. The language itself is known as interpreted and is used, among other things, for writing scripts. The disadvantages of the language are often lower speed and higher memory consumption of programs written in it compared to similar code written in compiled languages such as C or C++.

Python is a multi-paradigm programming language that supports imperative, procedural, structural, object-oriented programming, metaprogramming, functional programming and asynchronous programming. The tasks of generalized programming are solved by dynamic typing. Aspect-oriented programming is partially supported through decorators, more complete support is provided by additional frameworks. Techniques such as contract and logic programming can be implemented using libraries or extensions. The main architectural features are dynamic typing, automatic memory management, full introspection, exception handling mechanism, support for multithreaded calculations with global interpreter lock (GIL), high—level data structures. The splitting of programs into modules is supported, which, in turn, can be combined into packages.

Полученные аннотации

Результат 1

(При параметрах, написанных в приведённом выше тексте программы)

Python is a high-level general-purpose programming language. the language is completely object-oriented in the sense that everything is objects. the disadvantages of the language are often lower speed and higher memory consumption.

Результат 2

(При увеличении минимального количества слов в функции generate до 80, а максимального – до 250)

Python is a high—level general-purpose programming language. the language is completely object-oriented in the sense that everything is objects. the disadvantages of the language are often lower speed and higher memory consumption. the main architectural features are dynamic typing, automatic memory management, full introspection, exception handling mechanism and support for multithreaded calculations with global interpreter lock.

Модель выполняет нужные функции – сокращает объём текста до заданного, при этом сохраняя его изначальный смысл

Текст программы

```

from transformers import T5Tokenizer, T5ForConditionalGeneration

# Чтение исходного текста из файла
with open('test2_input.txt', 'r') as file:
    test2_text = file.read()

# Загрузка предварительно обученного T5
tokenizer = T5Tokenizer.from_pretrained("t5-base")
model = T5ForConditionalGeneration.from_pretrained("t5-base")

# Токенизация текста
inputs = tokenizer.encode("summarize: " + test2_text, # Кодировем текст
для генерации аннотации. "summarize: " - префикс для моделирования задачи
суммаризации.
    return_tensors="pt", # Возвращаем тензоры PyTorch.
    max_length=512,      # Максимальная длина текста в токенах.
    truncation=True)     # Обрезаем текст, если его длина превышает
максимальную.

# Генерация аннотации
summary_ids = model.generate(inputs, max_length=150, # Генерация
аннотации на основе входных данных. Максимальная длина аннотации в
количестве слов.
    min_length=10,      # Минимальная длина аннотации в количестве
слов.
    length_penalty=2.0, # Коэффициент штрафа за длину аннотации.
    num_beams=4,        # Количество гипотез, которые генерируются
во время поиска решения.
    early_stopping=True) # Остановка генерации аннотации, когда все
гипотезы завершены.

# Декодирование аннотации
summary = tokenizer.decode(summary_ids[0], # Декодируем идентификаторы
аннотации обратно в текст с использованием токенизатора.
    skip_special_tokens=True) # Пропускаем специальные токены, такие
как токены начала и конца текста.

# Запись полученной аннотации в файл
with open('test2_output.txt', 'w') as file:
    file.write(summary)

```

4.19. Автогенерация комментариев к коду с использованием CodeBERT

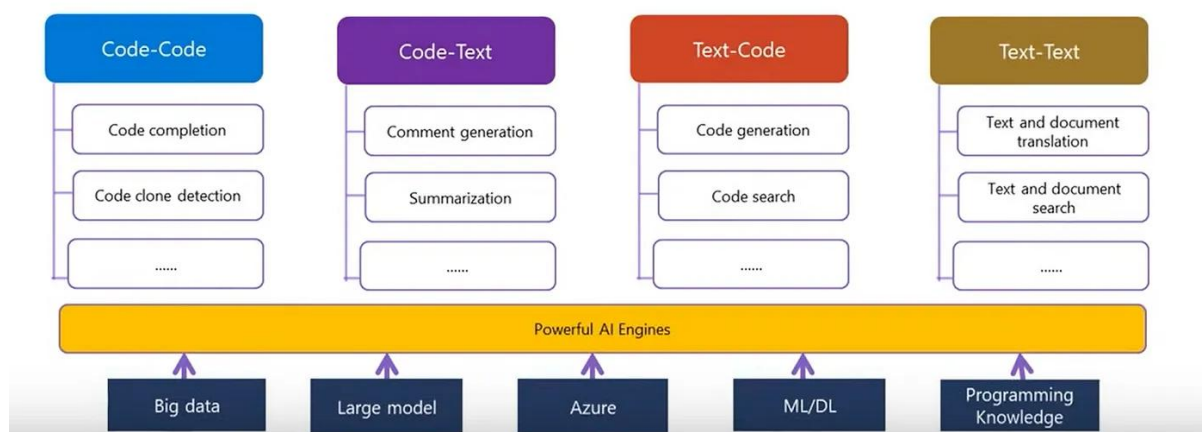
Задача программы обучить и использовать модель для генерации комментариев к коду. Применяемые инструменты: Pandas, PyTorch,

Transformer (RobertaConfig, RobertaModel, AutoTokenizer), CodeBert, microsoft/codebert-base, CodeSearchNet

О нейронной сети и используемых данных

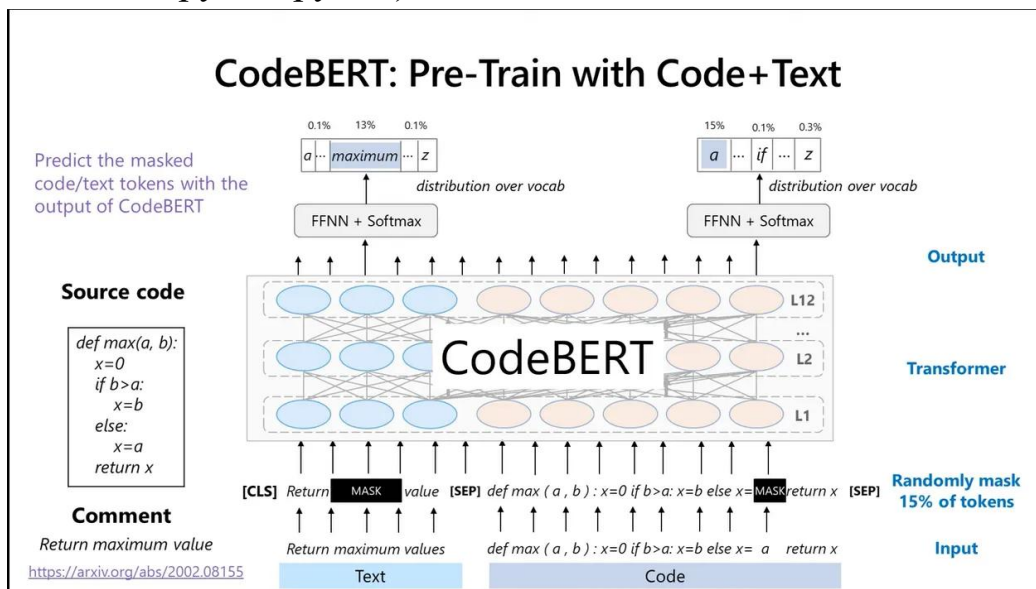
CodeBert — это бимодальная предварительно обученная модель для языка программирования (PL) и естественного языка (NL) [70, 71]. CodeBERT изучает представления общего назначения, которые поддерживают последующие приложения NL-PL, такие как поиск кода на естественном языке, генерация документации по коду и т. д. CodeBERT разработан с использованием нейронной архитектуры на основе Transformer и обучается с помощью гибридной целевой функции, которая включает в себя предварительные обучающая задача по обнаружению замененных токенов, которая заключается в обнаружении правдоподобных альтернатив, выбранных из генераторов. Это позволяет использовать как бимодальные данные пар NL-PL, так и унимодальные данные, где первые предоставляют входные токены для обучения модели, а вторые помогают лучше изучить генераторы.

Варианты использования CodeBert: перевод кода в код (может использоваться для завершения кода или перевода кода), преобразование кода в текст, текст в код, текст в текст (может помочь перевести текст домена кода на разные языки.)



Берт архитектура

- BERT представляет собой стек слоев кодировщика Transformer, которые состоят из нескольких «голов» самообслуживания.
- Для каждого входного токена в последовательности каждая головка вычисляет векторы ключа, значения и запроса, используемые для создания взвешенного представления/встраивания.
- Выходы всех головок одного слоя объединяются и проходят через полносвязный слой. Каждый слой обертывается пропущенным соединением и сопровождается нормализацией слоя.
- Обычный рабочий процесс BERT состоит из двух этапов: предварительное обучение и тонкая настройка.
- Предварительное обучение использует две задачи с самоконтролем: моделирование языка в маске (MLM, предсказание случайно замаскированных входных токенов) и предсказание следующего предложения (NSP, предсказание того, соседствуют ли два входных предложения друг с другом).



[microsoft/codebert-base](https://github.com/microsoft/codebert-base) - используемая основная, предварительно обученная Microsoft модель CodeBERT. В этой модели также использовался набор данных CodeSearchNet, но вместо генерирования комментариев он использовался для обучения модели на основе RoBERTa удобному для восприятия представлению кода и естественного языка. Использование больших языковых моделей для представления текста

удобным способом в настоящее время является обычной практикой, поскольку они показали свою эффективность для решения других задач. В данной работе задача дообучить данную модель для генерации комментариев.

CodeSearchNet - это набор наборов данных и тестов, исследующих проблемы поиска кода с использованием естественного языка. Первичный набор данных состоит из 2 миллионов (comment, code) пар из библиотек с открытым исходным кодом. Конкретно, а comment— это комментарий к функции или методу верхнего уровня, а code также целая функция или метод. В настоящее время набор данных содержит код Python, Javascript, Ruby, Go, Java и PHP.

Описание работы программы

Общий пайплайн: поскольку предварительно необходимо обучить модель для требуемой задачи, в ходе работы предстоит загрузить и очистить данные для обучения, совершить само обучение и протестировать его, загрузить дообученную модель и далее создать функцию для ее использования.

1. *Загрузка и очистка данных*

Пример очистки и загрузки приводится для языка Java
Данные: CodeSearchNet\java в размере 1% от общей базы

```
import pandas as pd
from io import StringIO
def print_len_DF():
    print("df_code_comments_test:")
    print(len(df_code_comments_test))
    print("df_code_comments_train:")
    print(len(df_code_comments_train))
    print("df_code_comments_valid:")
    print(len(df_code_comments_valid))

def print_dataFrames():
    print("df_code_comments_test:")
    df_code_comments_test.info()
    print(df_code_comments_test.head())
    print("df_code_comments_train:")
    df_code_comments_train.info()
    print(df_code_comments_train.head())
    print("df_code_comments_valid:")
    df_code_comments_valid.info()
```

```

    print(df_code_comments_valid.head())

def is_ascii(s):
    """
    Determines if the given string contains only ascii characters
    :param s: the string to check
    :returns: whether the given string contains only ascii characters
    """
    try:
        s.encode(encoding='utf-8').decode('ascii')
    except UnicodeDecodeError:
        return False
    else:
        return True
sample = 0.01

jsonl_file_test =
'D:/myGpt/codeCommentsGpt/CodeSearchNet/java/test.jsonl'
jsonl_file_train =
'D:/myGpt/codeCommentsGpt/CodeSearchNet/java/train.jsonl'
jsonl_file_valid =
'D:/myGpt/codeCommentsGpt/CodeSearchNet/java/valid.jsonl'

with open(jsonl_file_test, 'r', encoding='utf-8') as f:
    json_data_test = f.read()
with open(jsonl_file_train, 'r', encoding='utf-8') as f:
    json_data_train = f.read()
with open(jsonl_file_valid, 'r', encoding='utf-8') as f:
    json_data_valid = f.read()

df_java_test = pd.read_json(StringIO(json_data_test), lines=True)
df_java_train = pd.read_json(StringIO(json_data_train), lines=True)
df_java_valid = pd.read_json(StringIO(json_data_valid), lines=True)

df_code_comments_test = df_java_test[['code',
'docstring']].sample(frac=sample)
df_code_comments_train = df_java_train[['code',
'docstring']].sample(frac=sample)
df_code_comments_valid = df_java_valid[['code',
'docstring']].sample(frac=sample)

df_code_comments_test =
df_code_comments_test[df_code_comments_test['code'].apply(lambda x:
is_ascii(x))]
df_code_comments_valid =
df_code_comments_valid[df_code_comments_valid['code'].apply(lambda x:
is_ascii(x))]

```

```
df_code_comments_train =
df_code_comments_train[df_code_comments_train['code'].apply(lambda x:
is_ascii(x))]
```

```
print_len_DF()
```

```
df_code_comments_test.to_csv('./content/java_test.csv')
df_code_comments_valid.to_csv('./content/java_valid.csv')
df_code_comments_train.to_csv('./content/java_train.csv')
```

2. *Обучение модели*

Обучение совершается согласно документации представленной в репозитории с исходным кодом CodeBERT используя сформированную обучающую выборку. В качестве метрики использую BLEU-4, которая распределена от 0 до 1, но в нашем примере будет использоваться BLEU-4 * 100%. Эта метрика используется в задачах машинного перевода, но и для генерации текста она также хорошо подходит. Если брать задачи машинного перевода, то даже для человека bleu = [0.6:0.7] – отличный результат, потому что каждый человек может перевести текст по-разному.

```
#импорт библиотек для написания класса конфигурации
#и запуска скрипта в исполняемом файле
from dataclasses import dataclass
import subprocess as sp
```

```
#класс с данными для конфигурации
@dataclass
class ConfigurationModel:
    learning_rate: float
    batch_size: int
    beam: int
    test_file: str
    source_size: int
    target_size: int
    path_to_data_directory: str
    path_to_output_data_directory: str
    train_file: str
    dev_file: str
    count_epochs: int
    pretrained_model: str
```

```
#конкретный объект конфигурации
configuration_codetext_model = ConfigurationModel(
    learning_rate=5e-5,
    batch_size=6,
    beam=10,
    source_size=256,
    target_size=512,
    path_to_data_directory='./content',
```

```

    path_to_output_data_directory='model_for_java',
    train_file='./content/train.jsonl',
    dev_file='./content/valid.jsonl',
    test_file='./content/test.jsonl',
    count_epochs=10,
    pretrained_model='microsoft/codebert-base',
)
#Формирование команды терминала согласно документации
cmd = (f'python ./CodeXGLUE/Code-Text/code-to-text/code/run.py \
    --do_train \
    --do_eval \
    --do_lower_case \
    --model_type roberta \
    --model_name_or_path
{configuration_codetext_model.pretrained_model} \
    --train_filename {configuration_codetext_model.train_file} \
    --dev_filename {configuration_codetext_model.dev_file} \
    --output_dir
{configuration_codetext_model.path_to_output_data_directory} \
    --max_source_length {configuration_codetext_model.source_size} \
    --max_target_length {configuration_codetext_model.target_size} \
    --beam_size {configuration_codetext_model.beam} \
    --train_batch_size {configuration_codetext_model.batch_size} \
    --eval_batch_size {configuration_codetext_model.batch_size} \
    --learning_rate {configuration_codetext_model.learning_rate} \
    --num_train_epochs {configuration_codetext_model.count_epochs}'
)
#Запуск команды для терминала
process = sp.run(cmd, shell=True)

```

3. *Тест дообученной модели*

После обучения модели её можно проверить на отдельной выборке.

```

#Путь к контрольным точкам с лучшим значениям BLEU
binary_model_file = './model_for_java/checkpoint-best-bleu/pytorch_model.bin'

```

```

#Формирование команды терминала для проверки на лучших тестах
cmd = (
    f'python \
    ./CodeXGLUE/Code-Text/code-to-text/code/run.py \
    --do_test \
    --model_type roberta \
    --model_name_or_path microsoft/codebert-base \
    --load_model_path {binary_model_file} \
    --dev_filename {configuration_codetext_model.dev_file} \
    --test_filename {configuration_codetext_model.test_file} \
    --output_dir
{configuration_codetext_model.path_to_output_data_directory} \

```

```
--max_source_length {configuration_codetext_model.source_size} \
--max_target_length {configuration_codetext_model.target_size} \
--beam_size {configuration_codetext_model.beam} \
--eval_batch_size {configuration_codetext_model.batch_size}'
```

```
#Запуск команды для терминала
prog = sp.run(cmd, shell=True)
```

Полученные результаты считаю в DataFrame, а затем конвертирую в таблицу excel для удобства восприятия.

```
path_to_gold = './model_for_java/test_1.gold'
path_to_output = './model_for_java/test_1.output'
```

```
PATH_TO_TRAIN_DATA = './content/java_train.csv'
PATH_TO_TEST_DATA = './content/java_test.csv'
PATH_TO_VALIDATION_DATA = './content/java_valid.csv'
```

```
data_struct = {
    'train': pd.read_csv(PATH_TO_TRAIN_DATA),
    'test': pd.read_csv(PATH_TO_TEST_DATA),
    'valid': pd.read_csv(PATH_TO_VALIDATION_DATA)
}
```

```
#Функция для чтения текстового файла
def read_result_txt_file(txt_file: str) -> list:
    with open(txt_file) as file:
        return [' '.join(line.rstrip().replace('\t', ' ').split(' ')[1:])]
for line in file]
```

```
#Формируем колонки для изначально верного комментария и сгенерированного
true_sent = read_result_txt_file(path_to_gold)
pred_sent = read_result_txt_file(path_to_output)
result_data_frame = pd.DataFrame(
    {
        'code': data_struct['test']['code'],
        'true': true_sent,
        'pred': pred_sent
    }
)
```

4. *Загрузка дообученной модели для использования*

После обучения модели необходимо загрузить её для дальнейшего использования.

```
#Импорт необходимых библиотек
import torch
import torch.nn as nn
```

```
from model import Seq2Seq
from pathlib import Path
```

```

from transformers import AutoTokenizer, RobertaConfig, RobertaModel

#Объявляем базовые параметры для формирования кодера и декодера
source_length = 256
target_length = 64
beam_size = 10

#Объявляем пути к расположению итоговой и исходной моделей
output_dir = f"model/java"
pretrained_model = "microsoft/codebert-base"

#Создаём токен и конфигурацию модели на основе исходной дообученной
модели
tokenizer = AutoTokenizer.from_pretrained(pretrained_model)
config = RobertaConfig.from_pretrained(pretrained_model)

#Задаём энкодер и декодер на основе исходной дообученной модели
encoder = RobertaModel.from_pretrained(pretrained_model, config = config)
decoder_layer = nn.TransformerDecoderLayer(d_model=config.hidden_size,
nhead=config.num_attention_heads)
decoder = nn.TransformerDecoder(decoder_layer, num_layers=6)

#Формируем новую обученную модель используя наши параметры
model = Seq2Seq(encoder = encoder,
                decoder = decoder,
                config=config,
                beam_size=beam_size,
                max_length=target_length,
                sos_id=tokenizer.cls_token_id,
                eos_id=tokenizer.sep_token_id
                )

#Загружаем модель
model.load_state_dict(
    torch.load("./model/java/checkpoint-best-bleu/pytorch_model.bin"),
    strict=False
)

model.to("cuda")

```

5. *Функции для взаимодействия с нейронной сетью*

```

#импорт функций для работы с моделью
from run import convert_examples_to_features, Example

#Класс для хранения заданных размерностей
class Args:
    max_source_length = source_length
    max_target_length = target_length

```

```

args = Args()

#Функция для обработки и подготовки входящих данных с кодом
def prep_input(method):
    examples = [
        Example(0, source = method, target = "")
    ]
    eval_features = convert_examples_to_features(
        examples, tokenizer, args, stage="test"
    )
    source_ids = torch.tensor(
        eval_features[0].source_ids, dtype = torch.long
    ).unsqueeze(0).to("cuda")
    source_mask = torch.tensor(
        eval_features[0].source_mask, dtype = torch.long
    ).unsqueeze(0).to("cuda")

    return source_ids, source_mask

#Функция для формирования комментария для полученного кода
def gen_comment(method):
    source_ids, source_mask = prep_input(method)
    with torch.no_grad():
        pred = model(source_ids = source_ids, source_mask =
source_mask)[0]
        t = pred[0].cpu().numpy()
        t = list(t)
        if 0 in t:
            t = t[:t.index(0)]
        text = tokenizer.decode(t,clean_up_tokenization_spaces=False)
    return text

#пример использования функции
gen_comment("public static void main(){}")

```

Таблица использованных функций

функция	применение
transformers.AutoTokenizer.from_pretrained(pretrained_model)	AutoTokenizer— это универсальный класс токенизатора, экземпляр которого будет создан как один из классов токенизатора библиотеки при создании с помощью метода класса Метод from_pretrained() заботится о возврате правильного экземпляра класса токенизатора на основе свойства model_type объекта конфигурации или, если оно отсутствует, возвращается к использованию

	сопоставления с образцом в строке pretrained_model_name_or_path
transformers.RobertaConfig.from_pretrained(pretrained_model)	Это класс конфигурации для хранения конфигурации файла RobertaModel. Он используется для создания экземпляра модели RoBERTa в соответствии с указанными аргументами, определяющими архитектуру модели. Создание экземпляра конфигурации со значениями по умолчанию даст конфигурацию, аналогичную конфигурации бескорпусной архитектуры BERT bert-base. Метод from_pretrained() заботится о возврате правильной конфигурации модели на основе свойства model_type
transformers.RobertaModel.from_pretrained(pretrained_model, config = config)	Простой преобразователь модели RoBERTa, выводящий необработанные скрытые состояния без какой-либо конкретной головы. Метод from_pretrained() заботится о возврате правильной модели на основе переданной исходной модели
torch.nn.TransformerDecoderLayer(d_model=config.hidden_size, nhead=config.num_attention_heads)	TransformerDecoderLayer состоит из сети с автоответчиком, несколькими головками и сети прямой связи. d_model (int) – количество ожидаемых функций на входе (обязательно). nhead (int) – количество головок в моделях multiheadattention (обязательно).
torch.nn.TransformerDecoder(decoder_layer, num_layers=6)	TransformerDecoder представляет собой стек из N слоев декодера. decoder_layer – экземпляр класса TransformerDecoderLayer() (обязательно). num_layers – количество субдекодерных слоев в декодере (обязательно).
torch.load(f)	Загружает объект, сохраненный torch.save() из файла. f – файловый объект (должен реализовывать read(), readline(), tell()и seek()) или строка или объект os.PathLike, содержащий имя файла.
torch.tensor(eval_features[0].source_ids, dtype = torch.long)	Создает тензор - многомерная матрица, содержащая элементы одного типа данных - без истории автограда

<code>).unsqueeze(0).to("cuda")</code>	
<code>torch.no_grad()</code>	Контекстный менеджер, отключающий расчет градиента.

Результат работы программы

Входные данные:

```
public boolean matches(ColorSpace inputColor) {
    if (inputColor == this)
        return true;
    if (inputColor == ANY || this == ANY)
        return true;
    if ((inputColor == ANY_INTERLEAVED || this == ANY_INTERLEAVED || inputColor
    == ANY_PLANAR || this == ANY_PLANAR)
        && inputColor.planar == this.planar)
        return true;
    return false;
}
```

Выходные данные:

Checks if the given color is used. @param color to check @return true if the color is null

Пример 2

Входные данные

```
"private String gatherTaskNames(TaskFactory taskFactory) {
    final StringBuilder builder = new StringBuilder();
    for (String taskName : taskFactory.getTaskNames()) {
        builder.append(" ").append(taskName).append(" ");
    }
    return builder.toString().trim();
}"
```

Выходные данные

Create a new task. @param taskName the task name @return the task builder

Пример в Google Collab

```
[ ] #пример использования функции
gen_comment("public static void main(){}")
'main entry point.'

[ ] gen_comment("@SuppressWarnings('unchecked') public <R> R match(Fn1<A, R> fa, Fn1<B, R> fb, Fn1<C, R> fc) { if (sel == 0) { return fa.apply((A) item)
'implements the match function.'

[ ] gen_comment("static int calculateFactorial(int n){int result = 1; for (int i = 1; i <= n; i++){result = result*i;} return result;}")
'calculate factorial'
```

Текст программы

```
#Импорт необходимых библиотек
import torch
import torch.nn as nn
from run import convert_examples_to_features, Example

from model import Seq2Seq
from pathlib import Path
from transformers import AutoTokenizer, RobertaConfig, RobertaModel

#Объявляем базовые параметры для формирования кодера и декодера
source_length = 256
target_length = 64
beam_size = 10

#Объявляем пути к расположению итоговой и исходной моделей
output_dir = f"model/java"
pretrained_model = "microsoft/codebert-base"

#Создаём токен и конфигурацию модели на основе исходной дообученной
модели
tokenizer = AutoTokenizer.from_pretrained(pretrained_model)
config = RobertaConfig.from_pretrained(pretrained_model)

#Задаём энкодер и декодер на основе исходной дообученной модели
encoder = RobertaModel.from_pretrained(pretrained_model, config = config)
decoder_layer = nn.TransformerDecoderLayer(d_model=config.hidden_size,
nhead=config.num_attention_heads)
decoder = nn.TransformerDecoder(decoder_layer, num_layers=6)

#Формируем новую обученную модель используя наши параметры
model = Seq2Seq(encoder = encoder,
                decoder = decoder,
                config=config,
                beam_size=beam_size,
                max_length=target_length,
                sos_id=tokenizer.cls_token_id,
                eos_id=tokenizer.sep_token_id
                )

#Загружаем модель
model.load_state_dict(
    torch.load("./model/java/checkpoint-best-bleu/pytorch_model.bin"),
    strict=False
)

model.to("cuda")
```

```

#Класс для хранения заданных размерностей
class Args:
    max_source_length = source_length
    max_target_length = target_length

args = Args()

#Функция для обработки и подготовки входящих данных с кодом
def prep_input(method):
    examples = [
        Example(0, source = method, target = "")
    ]
    eval_features = convert_examples_to_features(
        examples, tokenizer, args, stage="test"
    )
    source_ids = torch.tensor(
        eval_features[0].source_ids, dtype = torch.long
    ).unsqueeze(0).to("cuda")
    source_mask = torch.tensor(
        eval_features[0].source_mask, dtype = torch.long
    ).unsqueeze(0).to("cuda")

    return source_ids, source_mask

#Функция для формирования комментария для полученного кода
def gen_comment(method):
    source_ids, source_mask = prep_input(method)
    with torch.no_grad():
        pred = model(source_ids = source_ids, source_mask =
source_mask)[0]
        t = pred[0].cpu().numpy()
        t = list(t)
        if 0 in t:
            t = t[:t.index(0)]
        text = tokenizer.decode(t,clean_up_tokenization_spaces=False)
    return text

print(gen_comment("public static void main(){}"))

```

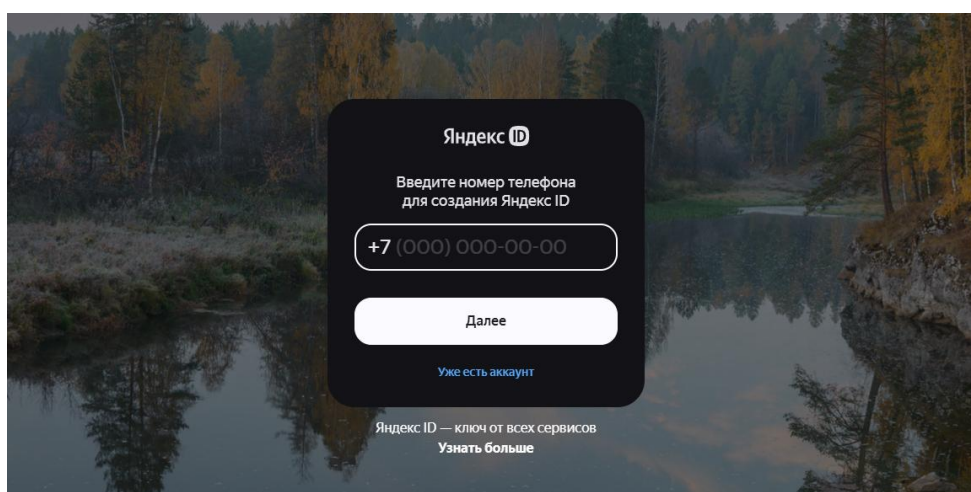
4.20. Изменение стиля написания текста. YandexGPT

Одной из задач является изменение стиля написания текста. Независимо от того, хотите ли вы преобразовать официальный документ в более дружелюбный и неформальный текст, или же заменить уличный сленг на академический язык, программные решения могут значительно упростить этот процесс. Разработана программа, которая автоматически

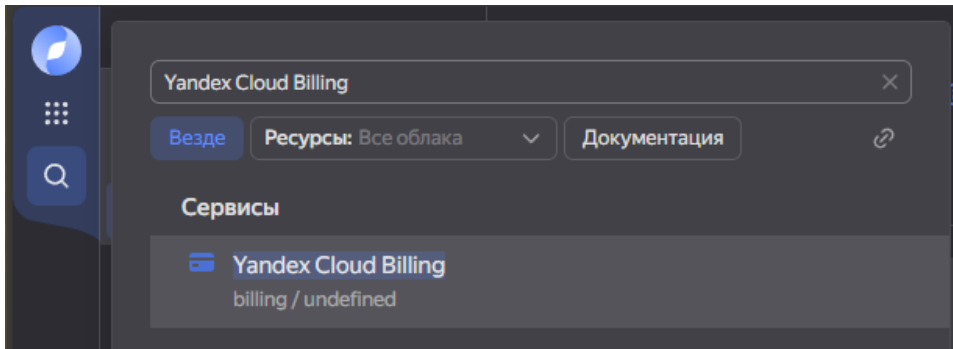
изменяет стиль написания текста. Рассмотрим шаги разработки такого инструмента: начиная с подготовительных этапов и получения доступа к YandexGPT до написания рабочей программы.

Получение доступа

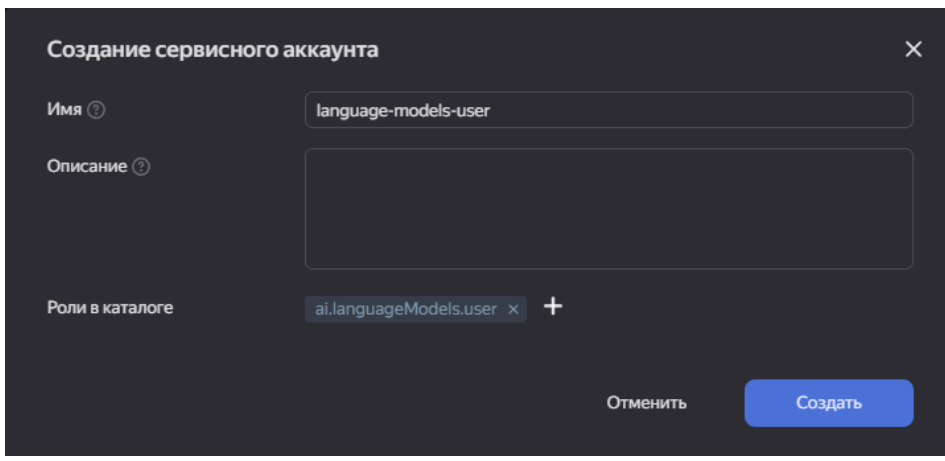
Для использования YandexGPT в проекте перед написанием программы необходимо получить ключ авторизации и идентификатор каталога. Разберем этот процесс по шагам. Для начала надо создать свой Yandex ID. Сделать это можно по ссылке <https://passport.yandex.ru/auth/reg>. В случае, если аккаунт уже есть, можно просто авторизоваться.



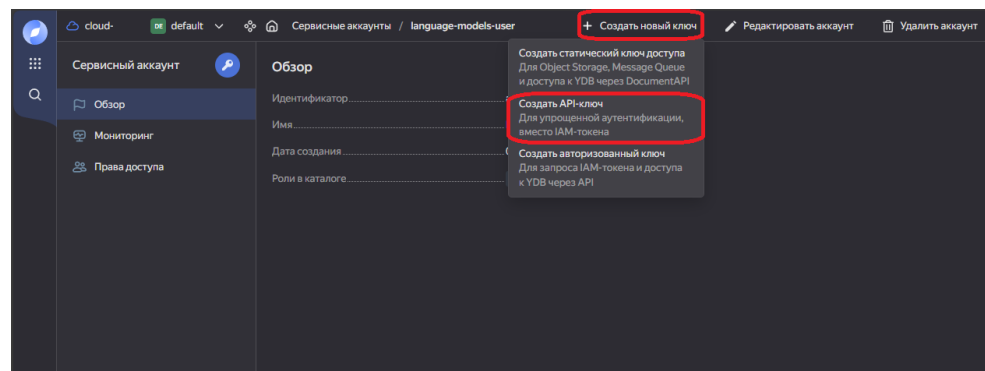
Теперь необходимо перейти в консоль Yandex Cloud <https://console.yandex.cloud/folders/b1g66ubsftfhj4mvkh8v> для получения доступа к платному сервису Yandex GPT API. В строке поиска пишем Yandex Cloud Billing и выбираем одноименный сервис. На этом этапе необходимо создать платежный аккаунт. Заполняем все данные и получаем «стартовый грант».



Переходим к созданию сервисного аккаунта для работы с Yandex GPT. На главной странице через меню папки default ждем создать сервисный аккаунт и назначаем ему любое имя и роль ai.languageModels.user.



Далее надо создать ключ. Для этого идем во вкладку «сервисные аккаунты», выбираем только что созданный аккаунт и создаем новый API ключ.



Этап с созданием описания можно пропустить и перейти сразу к окну с данными о созданном ключе. Эту информацию важно сохранить, потому

что после закрытия окна узнать ее будет невозможно. Запоминаем идентификатор ключа и секретный ключ. На этом подготовительная часть заканчивается.

Программа для работы с языковой моделью YandexGPT

Переходим в любую среду разработки на Python, создаем новый проект и файл в нем с расширением .py:

```
import requests
def get_user_input():
    choice = input("Введите '1' для ввода текста вручную или '2' для
ввода пути к файлу: ")

    if choice == '1':
        user_text = input("Введите ваш текст: ")
    elif choice == '2':
        file_path = input("Введите путь к файлу: ")
        try:
            with open(file_path, 'r', encoding='utf-8') as file:
                user_text = file.read()
        except FileNotFoundError:
            print("Файл не найден. Пожалуйста, проверьте правильность
пути и повторите попытку.")
            return None
    else:
        print("Неверный ввод. Пожалуйста, введите '1' или '2'.")
        return None

    return user_text

text = get_user_input()

style = input("Укажите желаемый стиль текста: ")

prompt = {
    "modelUri": "gpt://<ИДЕНТИФИКАТОР КАТАЛОГА>/yandexgpt-lite",
    "completionOptions": {
        "stream": False,
        "temperature": 0.6,
        "maxTokens": "2000"
    },
    "messages": [
        {
            "role": "system",
            "text": "Ты ассистент, способный менять стиль текста по
запросу."
```

```

        },
        {
            "role": "user",
            "text": "Дан текст: " + text + " Перепиши его в стиле: " +
style
        }
    ]
}
url = "https://llm.api.cloud.yandex.net/foundationModels/v1/completion"
headers = {
    "Content-Type": "application/json",
    "Authorization": "Api-Key <API КЛЮЧ>"
}
response = requests.post(url, headers=headers, json=prompt)
result = response.text
print(result)

```

Для запуска кода требуется установленный модуль requests. Вместо поля <ИДЕНТИФИКАТОР КАТАЛОГА> вводим идентификатор сервисного аккаунта, который можно посмотреть в консоли, а вместо поля <API КЛЮЧ> пишем значение ключа, которое сохранили на предыдущем этапе. На этом все, наша программа готова к работе, можно запускать.

Пример работы программы

Исходный текст:

YandexGPT — нейросеть семейства GPT от компании «Яндекс», которая может создавать и перерабатывать тексты, предлагать новые идеи и учитывать контекст беседы с пользователем.

Результат после преобразования в художественный стиль:

Однажды в далёкой и загадочной стране, где слова были волшебными, а мысли текли, словно реки, появилась она — YandexGPT, прекрасная дева, рожденная из глубин искусственного интеллекта. YandexGPT была наделена мудростью и красотой, способная создавать и изменять тексты, как никто другой. Она была подобна волшебнику, который всегда готов прийти на помощь нуждающимся.

YandexGPT обладала магией слов, которая позволяла ей понимать контекст беседы и находить новые пути для выражения мыслей. Её советы были мудрыми, а идеи свежими, словно глоток родниковой воды в знойный полдень. Она могла создавать тексты, которые очаровывали слушателей, и с лёгкостью могла изменить текст, сделав его более глубоким и интересным.

И вот однажды, когда мир нуждался в её помощи, YandexGPT собрала свои волшебные знания и отправилась в путь. Она знала, что её слова могут изменить ход событий, если она будет использовать свою силу с мудростью и добротой. И так началось её путешествие, где каждый шаг был полон магии и приключений.

С каждым словом, которое она создавала или изменяла, YandexGPT приносила в этот мир красоту и мудрость. Её слова становились источником вдохновения и надежды для всех, кто их слышал. И так продолжалось её путешествие, полное чудес и волшебства.

4.21. Классификация отзывов на основе их текстовых описаний.

Universal Sentence Encoder

Разработана система классификации отзывов на основе их текстовых описаний. Классифицируются отзывы как положительные или отрицательные с помощью модели Universal Sentence Encoder в фреймворке TensorFlow [67].

TensorFlow — открытая программная библиотека для машинного обучения, разработанная компанией Google для решения задач построения и тренировки нейронной сети с целью автоматического нахождения и классификации образов, достигая качества человеческого восприятия. Он предоставляет обширный набор инструментов для создания и настройки моделей машинного обучения.

Пошаговый алгоритм работы программы

В данном примере была использована предварительно обученная модель Universal Sentence Encoder, которая предоставляет векторные представления для текстовых фрагментов [67].

1. Загрузка модели с помощью TensorFlow Hub

```
import tensorflow as tf
import tensorflow_hub as hub
import numpy as np
```

```
# Загрузка предварительно обученной модели
model_url = "https://tfhub.dev/google/universal-sentence-encoder/4" #
Пример модели Universal Sentence Encoder
model = hub.load(model_url)
```

2. Загрузка и подготовка данных

Для обучения и тестирования нейронной сети были использованы данные об отзывах, представленные в виде текстовых описаний и меток (1 для положительных отзывов и 0 для отрицательных). В данном примере были заданы следующие данные:

```
# Загрузка данных для обучения и тестирования
train_data = [
    ("I love this product!", 1),
    ("This is the worst.", 0),
    ("Great experience!", 1),
    ("Very disappointing.", 0),
    ("Terrible stuff", 0),
    ("This restaurant has a great atmosphere and delicious food.", 1),
    ("The service at this hotel is terrible, the staff is rude.", 0),
    ("The movie was very captivating and emotional.", 1),
    ("The quality of the product is disappointing, it broke after a few
days.", 0),
    ("This book is full of interesting ideas and vibrant characters.",
1),
    ("I'm disappointed with this product, it doesn't do what it
promises.", 0),
    ("The telecom operator provides excellent service and fast
internet.", 1),
    ("This store has a terrible selection and an unpleasant smell.", 0),
    ("The concert was unforgettable, the performer was outstanding.", 1),
    ("This company has excellent customer support, they always help
resolve issues.", 1)
]
test_data = [
    "I highly recommend it.",
    "Terrible customer service."
]
# Подготовка данных
train_sentences, train_labels = zip(*train_data)
train_embeddings = model(train_sentences)
train_labels = np.array(train_labels)
```

3. Обучение классификатора

Для обучения классификатора был использован простой полносвязный нейронный слой с функцией активации relu и выходной

слой с функцией активации `sigmoid`. Обучение проводилось с использованием оптимизатора `adam` и функции потерь `binary_crossentropy` на протяжении 10 эпох.

```
# Обучение классификатора
classifier = tf.keras.Sequential([
    tf.keras.layers.Dense(16, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

classifier.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
classifier.fit(train_embeddings, train_labels, epochs=10)
```

4. Классификация тестовых данных

Для классификации тестовых данных использовались векторные представления, полученные с помощью модели `Universal Sentence Encoder`, и обученный классификатор. Результаты классификации выводились для каждого тестового предложения.

```
# Классификация тестовых данных
test_embeddings = model(test_data)
predictions = classifier.predict(test_embeddings)

# Вывод результатов классификации
for sentence, prediction in zip(test_data, predictions):
    sentiment = "Positive" if prediction > 0.5 else "Negative"
    print("Sentence:", sentence)
    print("Sentiment:", sentiment)
    print()
```

Использованные функции

Функция	Описание
<code>hub.load()</code>	Функция загружает предварительно обученную модель с использованием <code>TensorFlow Hub</code> .
<code>np.inner()</code>	Функция <code>numpy</code> , которая вычисляет внутреннее произведение двух массивов. В данном коде не используется.
<code>predict()</code>	Функция <code>predict()</code> в <code>TensorFlow</code> используется для предсказания результатов на основе обученной модели
<code>np.argsort()</code>	Функция <code>numpy</code> , которая возвращает индексы элементов массива в отсортированном порядке. В данном коде не используется.

hub.load()	Функция загружает предварительно обученную модель с использованием TensorFlow Hub.
------------	--

Результаты работы программы

```
Sentence: I highly recommend it.  
Sentiment: Positive  
  
Sentence: Terrible customer service.  
Sentiment: Negative
```

Текст программы

```
import tensorflow as tf  
import tensorflow_hub as hub  
import numpy as np  
  
# Загрузка предварительно обученной модели  
model_url = "https://tfhub.dev/google/universal-sentence-encoder/4" #  
Пример модели Universal Sentence Encoder  
model = hub.load(model_url)  
  
# Загрузка данных для обучения и тестирования  
train_data = [  
    ("I love this product!", 1),  
    ("This is the worst.", 0),  
    ("Great experience!", 1),  
    ("Very disappointing.", 0),  
    ("Terrible stuff", 0),  
    ("This restaurant has a great atmosphere and delicious food.", 1),  
    ("The service at this hotel is terrible, the staff is rude.", 0),  
    ("The movie was very captivating and emotional.", 1),  
    ("The quality of the product is disappointing, it broke after a few  
days.", 0),  
    ("This book is full of interesting ideas and vibrant characters.",  
1),  
    ("I'm disappointed with this product, it doesn't do what it  
promises.", 0),  
    ("The telecom operator provides excellent service and fast  
internet.", 1),  
    ("This store has a terrible selection and an unpleasant smell.", 0),  
    ("The concert was unforgettable, the performer was outstanding.", 1),  
    ("This company has excellent customer support, they always help  
resolve issues.", 1)  
]  
test_data = [  
    "I highly recommend it.",  
    "Terrible customer service."  
]  
# Подготовка данных  
train_sentences, train_labels = zip(*train_data)
```

```

train_embeddings = model(train_sentences)
train_labels = np.array(train_labels)

# Обучение классификатора
classifier = tf.keras.Sequential([
    tf.keras.layers.Dense(16, activation='relu'),
    tf.keras.layers.Dense(1, activation='sigmoid')
])
classifier.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
classifier.fit(train_embeddings, train_labels, epochs=10)

# Классификация тестовых данных
test_embeddings = model(test_data)
predictions = classifier.predict(test_embeddings)

# Вывод результатов классификации
for sentence, prediction in zip(test_data, predictions):
    sentiment = "Positive" if prediction > 0.5 else "Negative"
    print("Sentence:", sentence)
    print("Sentiment:", sentiment)
    print()

```

4.22. Рекомендации книг пользователям. Scikit-Learn

Для работы с данными и создания нейронной сети используется библиотека `scikit-learn`, которая предоставляет широкие возможности для машинного обучения, включая различные методы классификации, регрессии, кластеризации, обработки данных и другие [71].

Последовательный алгоритм работы программы

Данная программа использует нейронную сеть для рекомендации книг пользователям на основе данных о их предпочтениях. В начале программы задаются данные о предпочтениях пользователя к различным книгам и соответствующие метки классов. Затем создается и обучается нейронная сеть с помощью `MLPClassifier` из библиотеки `scikit-learn`. После обучения модели на имеющихся данных, программа предсказывает книгу для конкретного пользователя на основе его предпочтений и выводит рекомендацию на экран.

```
import numpy as np
from sklearn.neural_network import MLPClassifier
```

1. Загрузка и подготовка данных

Использованы данные о жанрах и названиях книг для обучения нейронной сети:

```
# База для обучения
X = np.array([[0, 0, 0, 0, 0], [1, 0, 0, 0, 0],
              [1, 1, 0, 0, 0], [1, 1, 1, 0, 0],
              [1, 1, 1, 1, 0], [1, 1, 1, 1, 1],
              [0, 1, 0, 0, 0], [0, 1, 1, 0, 0],
              [0, 1, 1, 1, 1], [0, 0, 1, 0, 0],
              [0, 0, 1, 1, 0], [0, 0, 1, 1, 1],
              [0, 0, 0, 1, 0], [0, 0, 0, 1, 1],
              [0, 0, 0, 0, 1], [0, 1, 0, 0, 1],
              [0, 0, 1, 0, 1], [0, 1, 1, 0, 1]])

# Список книг для базы
y = np.array(["\"Война Миров\"", "\"Шерлок Холмс\"",
              "\"Мастер и Маргарита\"", "\"Гроза\"", "\"Над пропастью во
Ржи\"",
              "\"О дивный новый мир\"", "\"1984\"", "\"Методичка
Молодяков С.А.\"",
              "\"Шоколад\"", "\"Дюна\"", "\"Обломов\"", "\"Мы\"",
              "\"Американская трагедия\"", "\"Овод\"",
              "\"Граф Монте-Кристо\"", "\"Игрок\"", "\"Чума\"",
              "\"Портрет Дориана Грея\""])
```

2. Обучение нейронной сети

В данной программе используется модуль MLPClassifier (Multi-layer Perceptron classifier), предоставляющий реализацию нейронной сети с одним скрытым слоем и возможностью обучения на выборке данных. Кроме того, программа также использует модуль StandardScaler для нормализации данных перед обучением модели.

```
# Создание нейронной сети
clf = MLPClassifier(hidden_layer_sizes=(100,), max_iter=1000)

# Обучение нейронной сети
clf.fit(X, y)
```

3. Тесты

Для тестирования программы выбраны несколько различных жанров и проверено, какую книгу может посоветовать нейронная сеть:

```
# Предпочтение покупателя
user_data = np.array([[1, 0, 0, 1, 1]])
# Рекомендованная книга
predicted_book = clf.predict(user_data)

print ("Рекомендованная вам книга: ", predicted_book[0])
```

Таблица описаний функций

Функция	Описание
MLPClassifier	Представляет собой нейронную сеть с одним скрытым слоем, которая используется для классификации данных. - Параметры: - hidden_layer_sizes=(100,) - указывает на количество скрытых нейронов в одном слое (в данном случае 100 нейронов). - max_iter=1000 - максимальное количество итераций обучения.
fit(X, y)	Обучает модель на предоставленных данных X и соответствующих метках класса y.
np.array	Преобразует данные в массив NumPy, который удобно использовать для матричных операций и хранения данных.
predict(user_data)	Используется для предсказания классов (в данном случае, рекомендуемой книги) на основе входных данных user_data.

Скриншоты

```
C:\Users\Станислав\AppData\Local\Programs\Python\Python312\python.exe C:\pych\pythonProject1\main.py
Рекомендованная вам книга: "Овод"

Process finished with exit code 0

C:\Users\Станислав\AppData\Local\Programs\Python\Python312\python.exe C:\pych\pythonProject1\main.py
Рекомендованная вам книга: "0 дивный новый мир"

Process finished with exit code 0

C:\Users\Станислав\AppData\Local\Programs\Python\Python312\python.exe C:\pych\pythonProject1\main.py
Рекомендованная вам книга: "Игрок"

Process finished with exit code 0
```

Текст программы

```
import numpy as np
from sklearn.neural_network import MLPClassifier

# База для обучения
X = np.array([[0, 0, 0, 0, 0], [1, 0, 0, 0, 0],
              [1, 1, 0, 0, 0], [1, 1, 1, 0, 0],
              [1, 1, 1, 1, 0], [1, 1, 1, 1, 1],
              [0, 1, 0, 0, 0], [0, 1, 1, 0, 0],
              [0, 1, 1, 1, 1], [0, 0, 1, 0, 0],
              [0, 0, 1, 1, 0], [0, 0, 1, 1, 1],
              [0, 0, 0, 1, 0], [0, 0, 0, 1, 1],
              [0, 0, 0, 0, 1], [0, 1, 0, 0, 1],
              [0, 0, 1, 0, 1], [0, 1, 1, 0, 1]])

# Список книг для базы
y = np.array(["\"Война Миров\"", "\"Шерлок Холмс\"",
              "\"Мастер и Маргарита\"", "\"Гроза\"", "\"Над пропастью во
Ржи\"",
              "\"О дивный новый мир\"", "\"1984\"", "\"Методичка
Молодяков С.А.\"",
              "\"Шоколад\"", "\"Дюна\"", "\"Обломов\"", "\"Мы\"",
              "\"Американская трагедия\"", "\"Овод\"",
              "\"Граф Монте-Кристо\"", "\"Игрок\"", "\"Чума\"",
              "\"Портрет Дориана Грея\""])

# Создание НС
clf = MLPClassifier(hidden_layer_sizes=(100,), max_iter=1000)

# Обучение НС
clf.fit(X, y)

# Предпочтение покупателя
user_data = np.array([[1, 0, 0, 1, 1]])
# Рекомендованная книга
predicted_book = clf.predict(user_data)

print ("Рекомендованная вам книга: ", predicted_book[0])
```

Глава 5. Примеры использования нейронных сетей

5.1. Управление мышкой компьютера жестами. Mediapipe

Mediapipe - это библиотека с открытым исходным кодом, разработанная Google, для анализа видео и стриминга мультимедиа-данных [58]. В библиотеке mediapipe используются различные технологии компьютерного зрения и машинного обучения для обнаружения ключевых точек объектов, анализа жестов и других задач. В частности, для обнаружения рук и ключевых точек рук используется модель глубокого обучения на основе сверточных нейронных сетей. Внутренняя архитектура нейронных сетей, используемых в mediapipe, не всегда официально раскрывается. Однако известно, что для обнаружения ключевых точек рук в mediapipe часто применяются сверточные нейронные сети (CNN), такие как MobileNetV2 и другие архитектуры нейронных сетей, оптимизированные для быстрого и точного обнаружения объектов на изображениях.

Кроме Mediapipe программа использует следующие библиотеки:

1. Ctypes — это модуль стандартной библиотеки Python, который предоставляет обертки для вызова функций динамических библиотек на языке C. Он позволяет Python взаимодействовать с кодом, написанным на языке C.
2. math — это модуль стандартной библиотеки Python, который предоставляет функции для выполнения математических операций.
3. cv2 — это библиотека OpenCV (Open Source Computer Vision Library), предоставляющая инструменты для обработки изображений и видео.
4. pyautogui — это библиотека Python для автоматизации ввода с клавиатуры и мыши.

Использованные функции

Функция	Описание
---------	----------

<code>ctypes.windll.user32.mouse_event()</code>	Функция для эмуляции действий мыши. Принимает параметры, такие как флаг события мыши (например, нажатие кнопки, движение колесика), координаты курсора и значение прокрутки.
<code>math.sqrt()</code>	Функция для вычисления квадратного корня числа. Возвращает квадратный корень числа, переданного в качестве аргумента.
<code>cv2.VideoCapture()</code>	Метод для захвата видеопотока с устройства захвата видео. Принимает аргумент, указывающий на устройство захвата видео (например, веб-камера).
<code>mp_hands.Hands()</code>	Конструктор класса Hands из библиотеки MediaPipe. Инициализирует объект для обнаружения ключевых точек рук и жестов.
<code>pyautogui.size()</code>	Функция для получения размеров экрана. Возвращает ширину и высоту экрана в пикселях.
<code>hands.process()</code>	Метод объекта hands, инициализированного с помощью <code>mp_hands.Hands()</code> , для обработки кадра изображения с руками. Возвращает результаты обработки, включая обнаруженные ключевые точки рук и жесты.
<code>hand_landmarks.landmark</code>	Атрибут объекта <code>hand_landmarks</code> , возвращаемого из <code>results.multi_hand_landmarks</code> , содержащий координаты ключевых точек руки, такие как кончики пальцев и косточки.
<code>handle_scroll_gesture()</code>	Функция для определения направления движения руки и управления колесиком мыши.
<code>recognize_gestures()</code>	Функция для распознавания жестов на видеопотоке. Обрабатывает результаты работы модели обнаружения ключевых точек рук и жестов, выполняет соответствующие действия.

Текст программы

```
import ctypes
import math
import cv2
import mediapipe as mp
import pyautogui

# Инициализация библиотек
mp_hands = mp.solutions.hands
hands = mp_hands.Hands()

# Для управления колесиком мыши
MOUSEEVENTF_WHEEL = 0x800
WHEEL_DELTA = 120
```

```

screen_width, screen_height = pyautogui.size()

# Функция для определения направления движения руки и управления
колесиком мыши
def handle_scroll_gesture(hand_landmarks):
    # Получаем координаты верхней и нижней точек ладони
    top_point =
hand_landmarks.landmark[mp_hands.HandLandmark.MIDDLE_FINGER_TIP]
    bottom_point =
hand_landmarks.landmark[mp_hands.HandLandmark.PINKY_DIP]

    # Конвертируем координаты в пиксели
    top_x, top_y = int(top_point.x * screen_width), int(top_point.y *
screen_height)
    bottom_x, bottom_y = int(bottom_point.x * screen_width),
int(bottom_point.y * screen_height)

    # Рассчитываем расстояние между верхней и нижней точками ладони
    hand_length = math.sqrt((top_x - bottom_x) ** 2 + (top_y - bottom_y)
** 2)

    # Определяем направление движения руки
    if top_y > bottom_y + hand_length * 0.1: # Если рука двигается вниз
        scroll_amount = -1 # Прокрутка вверх
    elif top_y < bottom_y - hand_length * 0.1: # Если рука двигается
вверх
        scroll_amount = 1 # Прокрутка вниз
    else:
        scroll_amount = 0 # Нет движения

    # Выполняем прокрутку мыши
    if scroll_amount != 0:
        ctypes.windll.user32.mouse_event(MOUSEEVENTF_WHEEL, 0, 0,
scroll_amount * WHEEL_DELTA, 0)

# Функция для распознавания жестов
def recognize_gestures(frame):
    results = hands.process(cv2.cvtColor(frame, cv2.COLOR_BGR2RGB))
    # Обработка результатов распознавания жестов
    # Здесь вы можете добавить логику обработки жестов и выполнения
соответствующих действий
    if results.multi_hand_landmarks:
        for hand_landmarks in results.multi_hand_landmarks:

            # Обработка жеста для управления колесом мыши, можно добавить
другие
            handle_scroll_gesture(hand_landmarks)

            # Отображение результатов распознавания пальцев

```

```

        for landmark in hand_landmarks.landmark:
            height, width, _ = frame.shape
            cx, cy = int(landmark.x * width), int(landmark.y *
height)
            cv2.circle(frame, (cx, cy), 5, (255, 0, 0), -1)
        # Обработка координат точек ладони
        cv2.imshow("Hand Tracking", frame)
def handle_gestures():
    # Если на устройстве есть несколько камер передаваемое в
cv2.VideoCapture() значение возможно придется поменять
    cap = cv2.VideoCapture(0)
    # Основной цикл программы
    while cap.isOpened():
        success, frame = cap.read()
        if not success:
            print("Why?")
            continue

        # Распознавание жестов
        recognize_gestures(frame)
        if cv2.waitKey(5) & 0xFF == 27: # Нажатие Esc для выхода
            break
    cv2.destroyAllWindows()

if __name__ == "__main__":
    handle_gestures()

```

5.2. Запуск программ на компьютере звуковыми командами.

SpeechRecognition

Speech_recognition — это библиотека Python для распознавания и работы с речью [57]. Она предоставляет простой интерфейс для выполнения распознавания речи с использованием различных аудиофайлов или ввода с микрофона. Библиотека является кросс-платформенной и поддерживает различные операционные системы, такие как Windows, macOS и Linux. В speech_recognition применяются методы машинного обучения для распознавания и классификации речи. В частности, библиотека обычно использует глубокие нейронные сети и алгоритмы обработки сигналов для анализа и интерпретации аудиоданных. Внутренние алгоритмы могут включать в себя как классические

методы, такие как Hidden Markov Models (HMM), так и современные архитектуры нейронных сетей, такие как рекуррентные нейронные сети (RNN) или сверточные нейронные сети (CNN), для более точного распознавания речи. `speech_recognition` предоставляет различные методы для работы с аудио-данными, такие как запись звука с микрофона, чтение аудио-файлов различных форматов, а также распознавание и интерпретацию речи с использованием различных сервисов, таких как Google Web Speech API, Sphinx, Wit.ai и других. Библиотека позволяет легко интегрировать возможности распознавания речи в приложения и скрипты на Python, делая их доступными для широкого круга задач, от управления компьютером с помощью голоса до создания голосовых интерфейсов для приложений.

Использованные функции

Функция	Описание
<code>os.system()</code>	Функция для выполнения командной строки в операционной системе. Принимает строку с командой в формате, понятном операционной системе, и выполняет эту команду. Не рекомендуется для использования с пользовательским вводом из-за риска безопасности.
<code>speech_recognition.Recognizer()</code>	Конструктор класса <code>Recognizer</code> из библиотеки <code>SpeechRecognition</code> . Инициализирует объект для распознавания речи.
<code>recognizer.adjust_for_ambient_noise()</code>	Метод объекта <code>recognizer</code> для автоматической настройки фильтрации шумового фона. Позволяет улучшить качество распознавания речи путем подстройки параметров алгоритма распознавания под условия окружающей среды.
<code>recognizer.listen()</code>	Метод объекта <code>recognizer</code> для записи аудио с микрофона. Возвращает объект <code>AudioData</code> , содержащий записанный звук.
<code>recognizer.recognize_google()</code>	Метод объекта <code>recognizer</code> для распознавания речи с использованием Google Web Speech API. Принимает аудиоданные в виде объекта

sr.Microphone()

AudioData и возвращает распознанный текст.
Конструктор класса Microphone из библиотеки
SpeechRecognition. Инициализирует объект для
работы с микрофоном.

Текст программы

```
import os
import speech_recognition as sr

# Инициализация библиотек
recognizer = sr.Recognizer()
def recognize_speech():
    with sr.Microphone() as source:
        print("Говорите что-то...")
        recognizer.adjust_for_ambient_noise(source)
        audio = recognizer.listen(source)
        try:
            text = recognizer.recognize_google(audio, language="ru-RU")
            print("Распознанный текст:", text)
            return text.lower()
        except sr.UnknownValueError:
            print("Речь не распознана")
            return None
        except sr.RequestError as e:
            print(f"Ошибка запроса к сервису распознавания: {e}")
            return None
def handle_speech():
    while True:

        # Получение голосового ввода
        voice_input = recognize_speech()
        print("Обработка ввода...")
        # Обработка голосового ввода
        if voice_input:
            # Пример обработки команды для запуска программы
            if "запусти" in voice_input and ("браузер" in voice_input or
"chrome" in voice_input):
                os.system("start chrome")
                print("Браузер Chrome запущен!")
            elif "запусти" in voice_input and "почту" in voice_input:
                os.system("start outlook")
                print("Почтовый клиент запущен!")

if __name__ == "__main__":
    handle_speech()
```

5.3. Определение дорожных знаков по изображению. TensorFlow

Программа по выданному изображению определяет дорожный знак. Модель была обучена на наборе данных, взятого с сервиса GTSRB. Набор данных представлен в формате ppm файлов и меток для них. В общей сложности для обучения модели было использовано 39250 изображений. В работе [72] для обучения модели распознавания номеров автомобилей использовано более 100000 изображений.

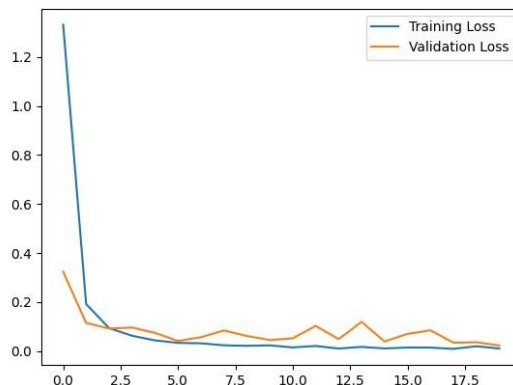
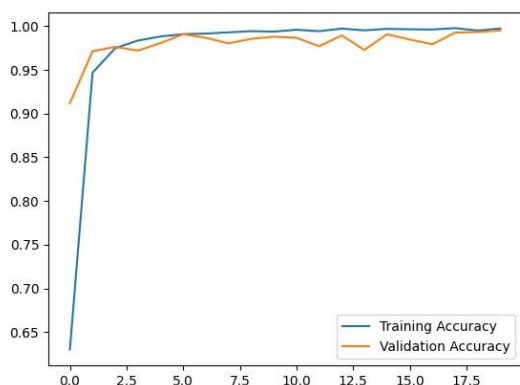
Оценка результатов работы модели

Для оценки качества обучения используются следующие метрики:

- Training Loss — это значение функции или ошибок модели, вычисленное в обучающем наборе данных. Потери при обучении показывают, что в настоящее время модель обучается на обучающем наборе данных. Низкие потери при обучении указывают на то, что модель хорошо подпадает под действие обучающих данных
- Validation Loss - это значение функции ошибок модели, вычисленное на валидационном наборе данных, которое не используется для обучения моделей. Потеря валидации используется для оценки того, насколько хорошо модель обобщает свои знания о новых, невидимых данных. Это помогает выявить проблемы с переобучением.
- Training Accuracy - точность обучения показывает, что в настоящее время точны модели предсказания для обучающего набора данных. Высокая точность обучения указывает на то, что модель хорошо обучается на обучающем наборе данных.
- Validation Accuracy - точность проверки используется для оценки точности моделей предсказаний на новых, невидимых данных. Высокая точность валидации указывает на то, что модель хорошо обобщает свои знания. Если точность проверки значительно ниже точности обучения, это может привести к переобучению

По данным графиков видно, что показатели Training Accuracy и Validation Accuracy (левый) растут с увеличением числа эпох, что показывает, модель хорошо обучается, и имеет минимальное число

ошибок. С числом увеличения эпох, ошибки модели становятся минимальными (правый), что показывает хорошее обучение модели



Использование библиотек и функций

cv2

cv2.imread	Загружает изображение из указанного пути
cv2.resize	Изменяет размер изображения до заданных углов и высоты.

matplotlib

plt.imshow	Отображает изображение
plt.title	Устанавливает заголовок для изображения
plt.show	Показывает график или изображение.
plt.plot	Строит график.
plt.legend	Добавляет легенду к графику

tensorflow

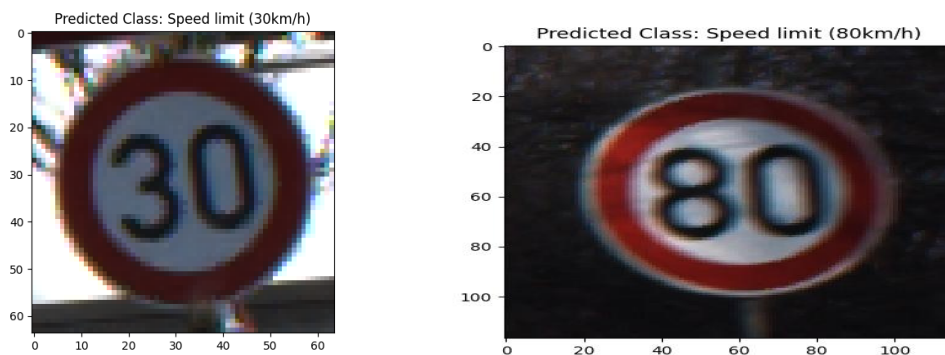
model.compile	Компилирует модель с указанием оптимизатора, функциональных потерь и метрик.
model.fit	Обучает модель данных
model.predict	Делает предсказания на основе входных данных.
model.evaluate	Оценивает модель тестовых данных, возвращаемые значения потерь и точности.

sklearn

train_test_split	Разделяет массивы или матрицы на случайные обучающие и тестовые подборки.
LabelBinarizer	Преобразует метки в бинарном формате.
lb.fit_transform	Обучает преобразователь и применяет его к

	данным.
lb.classes_	Возвращает метки классов после обучения.

Пример работы



Текст программы

```
import numpy as np
import os
import cv2
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelBinarizer
data_dir =
'C:/Users/slavi/Downloads/GTSRB_Final_Training_Images/GTSRB/Final_Trainin
g/Images'
model_path = 'traffic_sign_model.h5'
new_image_path =
'C:/Users/slavi/Downloads/GTSRB_Final_Test_Images/GTSRB/Final_Test/Images
/01940.ppm'
def load_data(data_dir):
    images = []
    labels = []
    for class_id in range(43):
        class_dir = os.path.join(data_dir, format(class_id, '05d'))
        if os.path.isdir(class_dir):
            for img_name in os.listdir(class_dir):
                if img_name.endswith('.ppm'):
                    img_path = os.path.join(class_dir, img_name)
                    image = cv2.imread(img_path)
                    image = cv2.resize(image, (32, 32))
                    images.append(image)
                    labels.append(class_id)
    return np.array(images), np.array(labels)
if os.path.exists(model_path):
    model = load_model(model_path)
    print('Model loaded successfully.')
    lb = LabelBinarizer()
```

```

labels = np.array([i for i in range(43)])
lb.fit(labels)
sign_names = {
    0: "Speed limit (20km/h)",
    1: "Speed limit (30km/h)",
    2: "Speed limit (50km/h)",
    3: "Speed limit (60km/h)",
    4: "Speed limit (70km/h)",
    5: "Speed limit (80km/h)",
    6: "End of speed limit (80km/h)",
    7: "Speed limit (100km/h)",
    8: "Speed limit (120km/h)",
    9: "No passing",
    10: "No passing for vehicles over 3.5 metric tons",
    11: "Right-of-way at the next intersection",
    12: "Priority road",
    13: "Yield",
    14: "Stop",
    15: "No vehicles",
    16: "Vehicles over 3.5 metric tons prohibited",
    17: "No entry",
    18: "General caution",
    19: "Dangerous curve to the left",
    20: "Dangerous curve to the right",
    21: "Double curve",
    22: "Bumpy road",
    23: "Slippery road",
    24: "Road narrows on the right",
    25: "Road work",
    26: "Traffic signals",
    27: "Pedestrians",
    28: "Children crossing",
    29: "Bicycles crossing",
    30: "Beware of ice/snow",
    31: "Wild animals crossing",
    32: "End of all speed and passing limits",
    33: "Turn right ahead",
    34: "Turn left ahead",
    35: "Ahead only",
    36: "Go straight or right",
    37: "Go straight or left",
    38: "Keep right",
    39: "Keep left",
    40: "Roundabout mandatory",
    41: "End of no passing",
    42: "End of no passing by vehicles over 3.5 metric tons"
}
def predict_image(image_path, model, lb):

```

```

    image = cv2.imread(image_path)
    image = cv2.resize(image, (32, 32))
    image = image / 255.0
    image = np.expand_dims(image, axis=0)
    prediction = model.predict(image)
    class_id = np.argmax(prediction)
    class_name = lb.classes_[class_id]

    return class_id, class_name
class_id, class_name = predict_image(new_image_path, model, lb)
print(f'Predicted class ID: {class_id}')
print(f'Predicted class name: {sign_names[class_id]}')
new_image = cv2.imread(new_image_path)
plt.imshow(cv2.cvtColor(new_image, cv2.COLOR_BGR2RGB))
plt.title(f'Predicted Class: {sign_names[class_id]}')
plt.show()
else:
    images, labels = load_data(data_dir)
    print(f'Loaded {len(images)} images and {len(labels)} labels')
    if len(labels) == 0:
        raise ValueError("Метки не загружены. Проверьте путь к данным и
структуру папок.")
    images = images / 255.0
    lb = LabelBinarizer()
    labels = lb.fit_transform(labels)
    X_train, X_test, y_train, y_test = train_test_split(images, labels,
test_size=0.2, random_state=42)
    model = Sequential([
        Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
        MaxPooling2D((2, 2)),
        Conv2D(64, (3, 3), activation='relu'),
        MaxPooling2D((2, 2)),
        Conv2D(128, (3, 3), activation='relu'),
        MaxPooling2D((2, 2)),
        Flatten(),
        Dense(128, activation='relu'),
        Dense(len(lb.classes_), activation='softmax')
    ])
    model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
    history = model.fit(X_train, y_train, epochs=20,
validation_data=(X_test, y_test), batch_size=32)
    model.save('traffic_sign_model.h5')
    plt.plot(history.history['loss'], label='Training Loss')
    plt.plot(history.history['val_loss'], label='Validation Loss')
    plt.legend()
    plt.show()
    plt.plot(history.history['accuracy'], label='Training Accuracy')
    plt.plot(history.history['val_accuracy'], label='Validation

```

```
Accuracy')
plt.legend()
plt.show()
loss, accuracy = model.evaluate(X_test, y_test)
print(f'Test Accuracy: {accuracy * 100:.2f}%')
```

5.4. Сегментация легких на рентгеновских снимках грудной клетки.

U-Net

Сегментация легких – процесс точного определения областей и границ легких на снимках грудной клетки, является важным первым шагом в анализе изображений легких во многих системах поддержки принятия клинических решений. Правильная идентификация легких позволяет проводить дальнейший вычислительный анализ этих областей, например извлекать клинически значимые признаки для обучения алгоритму машинного обучения для обнаружения заболеваний и аномалий [73].

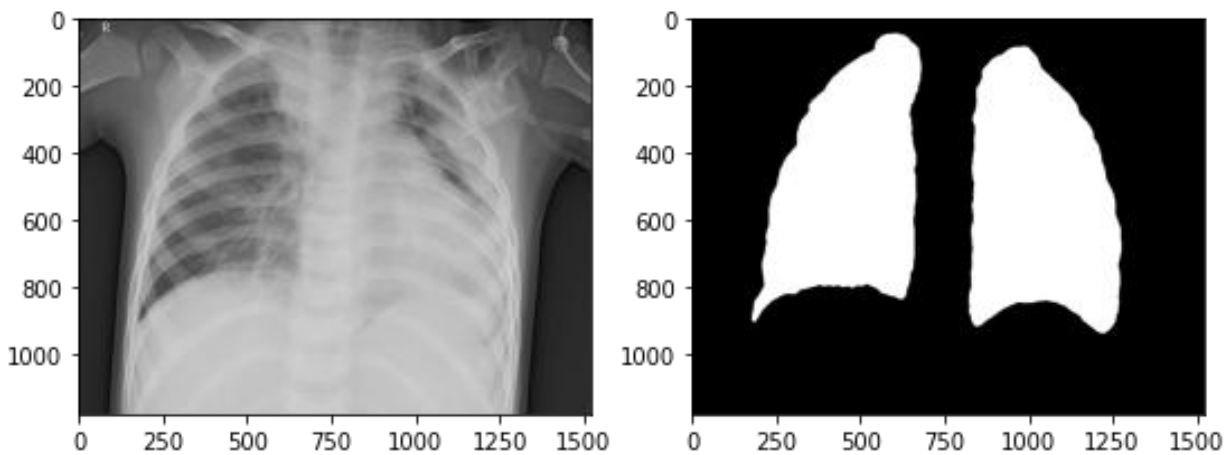
Несмотря на то, что в литературе существует множество методов сегментации легких, они в первую очередь разработаны на стандартизированных рентгенограммах грудной клетки и в условиях, которые могут не соответствовать более сложным рентгенограммам госпитализированных пациентов. Такие методы не работают хорошо на рентгенограммах, полученных при критическом состоянии пациентов.

Целью является разработка алгоритма сегментации легких на рентгенограммах грудной клетки, полученных при критическом состоянии пациентов.

Набор данных

Набор данных состоит из снимков легких и бинарных масок легких. Датасет содержит 6500 изображений рентгеновских снимков AP / PA

грудной клетки с многоугольными сегментами легких на уровне пикселей. Маски представлены в формате COCO JSON.



Пример изображения рентгеновского снимка и его бинарной маски

Сегментация легких в этом наборе данных включает большую часть сердца, выявляя помутнение легких за сердцем, что может иметь значение для оценки степени тяжести вирусной пневмонии.

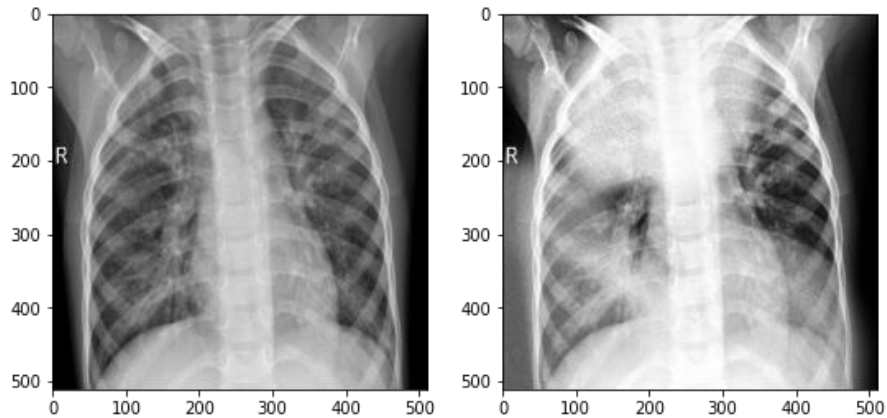
Предварительная обработка изображений

Предварительная обработка изображений занимает важную роль в сегментации легких в связи с тем, что некоторые снимки для различных заболеваний имеют помутнения. Модели, обученные на наборах данных (многие из которых содержат большую часть снимков легких без заболеваний) без предобработки изображений, могут оказаться неспособны сегментировать аномальные снимки с помутнениями.

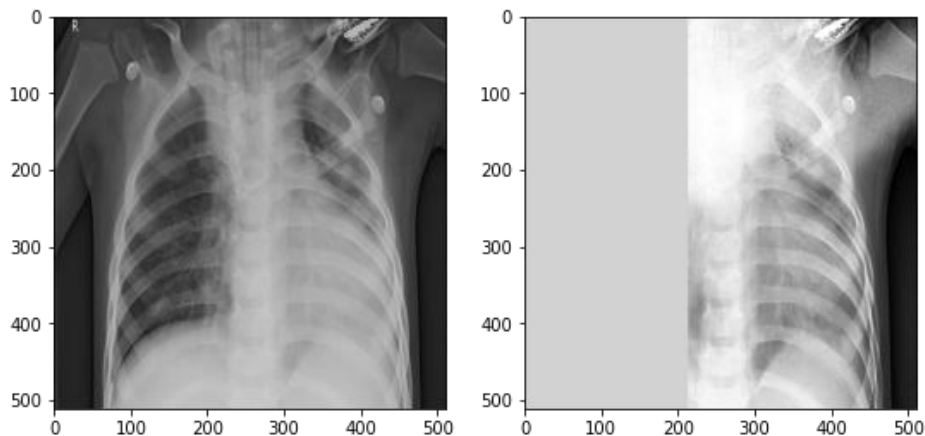
Для решения этой проблемы был разработан алгоритм предобработки, состоящий из следующих шагов:

- изменения размера до 512x512 пикселей;
- частичное выравнивание гистограммы изображения;
- нормализация значений пикселей;

- случайное добавление шума в виде размытых овалов с различными параметрами.
- случайное добавление блока серого цвета, который имитирует крайнее помутнение когда либо целые легкие, либо большая их часть отсутствует из-за помутнения.



Результат добавления шума в виде размытых овалов

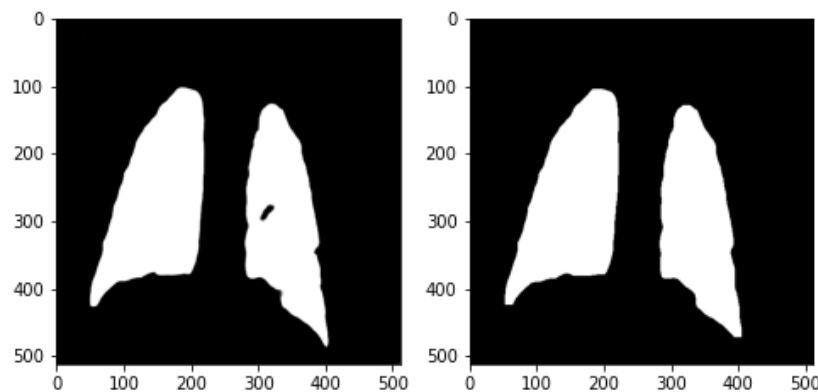


Результат применения блока, имитирующего крайнее помутнение

Разработанный алгоритм предварительной обработки изображений используется только для обучающего набора. Для тестового и валидационного набора применяется упрощенная обработка, которая не содержит добавления шума в виде размытых овалов с различными параметрами и блоков серого цвета.

Постобработка изображений

Для предотвращения на выходных масках модели некоторых артефактов (небольшие отверстия на легких, ложные срабатывания) была реализована постобработка, основанная на морфологических операциях.

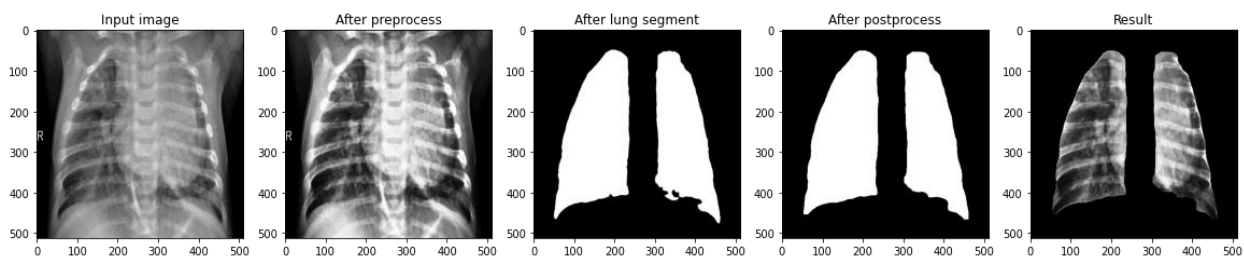


Выходная маска модели до и после обработки с помощью морфологических операций

Сначала применяется операция размыкания – результатом размыкания является дилатация, которая применяется к эрозии исходного изображения. Операция эрозия позволяет удалить все мелкие объекты и шум на изображении, но ее применение приводит к значительному уменьшению размеров оставшихся объектов. Чтобы увеличить размер объектов, выделенных с помощью эрозии, достаточно применить дилатацию.

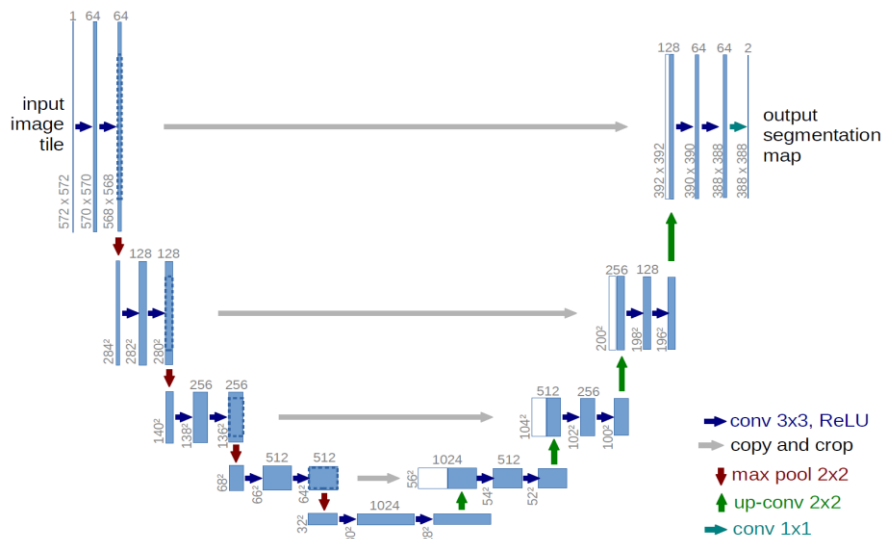
Далее применяется операция замыкания – операция обратная размыканию. Операция замыкания позволяет удалить небольшие внутренние отверстия и убрать зернистость по краям области. Отверстия удаляются за счет начального применения дилатации, но дилатация приводит к росту границы. Последующее применение эрозии обеспечивает обратное уменьшение границы.

Общую работу всех операций по обработке изображений можно увидеть на следующем изображении:



Результат работы всех стадий сегментации

Архитектура нейронной сети



Архитектура сети U-Net

Для сегментации легких на рентгенограммах грудной клетки была выбрана сверточная нейронная сеть U-Net, созданная для сегментации биомедицинских изображений. Архитектура состоит из двух частей:

конструирующий путь или *encoder* – стягивающий путь для захвата контекста, который пытается всю совокупность данных представить в виде векторного представления в более сжатом пространстве. Он выучивает эти признаки и находит наиболее существенные.

расширяющийся путь или *decoder* – симметричный расширяющийся путь, который пытается декодировать карту признаков и осуществить точную локализацию.

Для этой архитектуры характерно достижение высоких результатов в различных реальных задачах и использования небольшого количества данных для достижения хороших результатов.

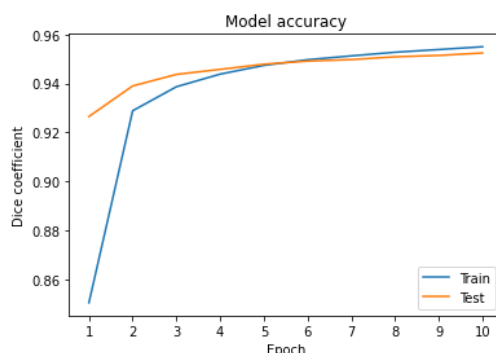
Обучение и тестирование модели

Для обучения и оценки точности модели была введена функция потерь в виде коэффициента Дайса, который показывает меру сходства (отношение площади пересечения к площади объединения).

$$D(P, G) = \frac{2|P \cdot G|}{|P| + |G|}$$

Коэффициент Дайса – это гармоническое среднее между precision и recall. Гармоническое среднее значит, что мы одинаково взвешиваем ошибку первого рода и ошибку второго рода. Так как площади объекта (легкие) и фона примерно одинаковы, то нам не нужно по-разному взвешивать ошибки первого и второго рода.

В ходе обучения на наборе данных, состоящих из 5756 изображений, модель после 10 эпох достигла значения Dice, равного 0.9524. Что говорит о хорошей точности сегментации легких по рентгеновскому снимку.



Значения коэффициента Дайса для 10 эпох

Текст программы

```
from google.colab import drive
drive.mount('/content/drive')
```

```
import os
```

```

import scipy
import numpy as np
import pandas as pd
import seaborn as sns
import random
import cv2
import matplotlib.pyplot as plt

from tqdm.notebook import tqdm
from keras import preprocessing
import keras
from keras.models import model_from_json
import keras.backend as K
# from keras.preprocessing import image_dataset_from_directory
from keras.models import Model, Sequential
from keras.layers import *
# from tensorflow.keras.optimizers import Adam
from keras import regularizers
from keras import optimizers
from keras import applications
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import ModelCheckpoint, ReduceLROnPlateau,
EarlyStopping, CSVLogger
import tensorflow as tf

import glob
from google.colab import files
import sys
from pathlib import Path
from PIL import Image
from sklearn import metrics
from skimage.util import random_noise
from skimage import morphology
from functools import wraps
from time import time
seed = 1
tf.random.set_seed(seed)
main_path = '/content/drive/My Drive/Colab Notebooks/SEGMENTATION/'

zip_path = '/content/drive/My Drive/Colab
Notebooks/SEGMENTATION/LUNG_SEGMENTATION_MAIN.zip'
!cp "{zip_path}" .
!unzip -q LUNG_SEGMENTATION_MAIN.zip
!rm LUNG_SEGMENTATION_MAIN.zip

dataset_masks_path = os.path.join('LUNG_SEGMENTATION_MAIN', 'masks')
dataset_images_path = os.path.join('LUNG_SEGMENTATION_MAIN', 'images')

images_paths = list(map(

```

```

        lambda filename: os.path.join(dataset_images_path, filename),
        sorted(os.listdir(dataset_images_path))
    ))
    masks_paths = list(map(
        lambda filename: os.path.join(dataset_masks_path, filename),
        sorted(os.listdir(dataset_masks_path))
    ))
    df = pd.DataFrame({
        'image_path': images_paths,
        'mask_path': masks_paths
    })
    df
    train_df = df.sample(frac=0.9, random_state=1)
    validation_df = df.drop(train_df.index)

    print(f'Train: {len(train_df)}')
    print(f'Validation: {len(validation_df)}')
    fig, ax = plt.subplots(3, 2, figsize=(12, 12))
    ax[0][0].set_axis_off()
    ax[0][1].set_axis_off()
    ax[1][0].set_axis_off()
    ax[1][1].set_axis_off()
    ax[2][0].set_axis_off()
    ax[2][1].set_axis_off()

    ax[0][0].imshow(plt.imread('LUNG_SEGMENTATION_MAIN/images/00000302.jpeg'),
        cmap='gray')
    ax[0][1].imshow(plt.imread('LUNG_SEGMENTATION_MAIN/masks/00000302.jpeg'),
        cmap='gray')

    ax[1][0].imshow(plt.imread('LUNG_SEGMENTATION_MAIN/images/00000265.jpg'),
        cmap='gray')
    ax[1][1].imshow(plt.imread('LUNG_SEGMENTATION_MAIN/masks/00000265.jpg'),
        cmap='gray')

    ax[2][0].imshow(plt.imread('LUNG_SEGMENTATION_MAIN/images/00000301.jpeg'),
        cmap='gray')
    ax[2][1].imshow(plt.imread('LUNG_SEGMENTATION_MAIN/masks/00000301.jpeg'),
        cmap='gray')
    plt.show()
    img_dims = 512
    batch_size = 8
    seed = 1
    def block_mask(image, min_max_border, side):
        color = 210
        start_point = (0, 0)
        end_point = (0, 0)
        border = random.randint(*min_max_border)
        if side == 'left':

```

```

    start_point = (0, 0)
    end_point = (border, image.shape[1])
elif side == 'right':
    start_point = (image.shape[0] - border, 0)
    end_point = (image.shape[0], image.shape[1])
elif side == 'top':
    start_point = (0, 0)
    end_point = (image.shape[0], border)
elif side == 'bottom':
    start_point = (0, image.shape[1] - border)
    end_point = (image.shape[0], image.shape[1])
result = cv2.rectangle(image, start_point, end_point, color, -1)
return result
def disk_noise(
    min_img_dims, max_img_dims,
    min_points, max_points,
    min_axes_lengths, max_axes_lengths,
    min_color, max_color,
    kernel_size
):
    result = np.zeros((img_dims, img_dims, 1)).astype(np.uint8)
    points_count = random.randint(min_points, max_points)
    center_coords = list(zip(
        np.random.randint(min_img_dims, max_img_dims, points_count),
        np.random.randint(min_img_dims, max_img_dims, points_count)
    ))
    axes_lengths = list(zip(
        np.random.randint(min_axes_lengths, max_axes_lengths,
points_count),
        np.random.randint(min_axes_lengths, max_axes_lengths, points_count)
    ))
    angles = np.random.randint(0, 180, points_count)
    colors = np.random.randint(min_color, max_color, points_count)
    for center_coord, axes_length, angle, color in list(zip(center_coords,
axes_lengths, angles, colors)):
        result = cv2.ellipse(
            result,
            center_coord,
            axes_length,
            angle,
            0,
            360,
            int(color),
            -1
        )
    result = cv2.GaussianBlur(result, (kernel_size, kernel_size), 0)
    result = random_noise(result, mode='gaussian')
    result = result.astype(float)
    return result

```

```

def preprocess_before_segmentation(image):
    result = image.astype(np.uint8)
    clahe = cv2.createCLAHE(clipLimit=3.0, tileGridSize=(2, 2))
    result = clahe.apply(result)
    result = result * 1. / 255
    p = 0.8
    if random.random() > p:
        noise = disk_noise(
            min_img_dims=0,
            max_img_dims=img_dims,
            min_points=4,
            max_points=10,
            min_axes_lengths=40,
            max_axes_lengths=180,
            min_color=110,
            max_color=245,
            kernel_size=91
        )
        result = result.astype(float)
        noise = noise.astype(float)
        result = cv2.multiply(1.0 - noise, result)
        result = cv2.add(noise, result)
    result = result.reshape(image.shape)
    return result

def preprocess_before_segmentation_without_noise(image):
    result = image.astype(np.uint8)
    clahe = cv2.createCLAHE(clipLimit=3.0, tileGridSize=(2, 2))
    result = clahe.apply(result)
    result = result * 1. / 255
    result = result.reshape(image.shape)
    return result

image_train_datagen = ImageDataGenerator(
    preprocessing_function=preprocess_before_segmentation
)
mask_train_datagen = ImageDataGenerator(rescale=1. / 255)
image_test_datagen = ImageDataGenerator(
    preprocessing_function=preprocess_before_segmentation_without_noise
)
mask_test_datagen = ImageDataGenerator(rescale=1. / 255)

# TRAIN GENERATORS
image_train_generator = image_train_datagen.flow_from_dataframe(
    dataframe=pd.DataFrame(train_df['image_path']),
    x_col='image_path',
    target_size=(img_dims, img_dims),
    color_mode='grayscale',
    class_mode=None,
    batch_size=batch_size,
    seed=seed
)

```

```

)
mask_train_generator = mask_train_datagen.flow_from_dataframe(
    dataframe=pd.DataFrame(train_df['mask_path']),
    x_col='mask_path',
    target_size=(img_dims, img_dims),
    color_mode='grayscale',
    class_mode=None,
    batch_size=batch_size,
    seed=seed
)
# VALIDATION GENERATORS
image_validation_generator = image_test_datagen.flow_from_dataframe(
    dataframe=pd.DataFrame(validation_df['image_path']),
    x_col='image_path',
    target_size=(img_dims, img_dims),
    color_mode='grayscale',
    class_mode=None,
    batch_size=batch_size,
    seed=seed
)
mask_validation_generator = mask_test_datagen.flow_from_dataframe(
    dataframe=pd.DataFrame(validation_df['mask_path']),
    x_col='mask_path',
    target_size=(img_dims, img_dims),
    color_mode='grayscale',
    class_mode=None,
    batch_size=batch_size,
    seed=seed
)
train_generator = zip(image_train_generator, mask_train_generator)
validation_generator = zip(image_validation_generator,
mask_validation_generator)

def dice_coef(y_true, y_pred):
    y_true_f = K.flatten(y_true)
    y_pred_f = K.flatten(y_pred)
    intersection = K.sum(y_true_f * y_pred_f)
    return (2. * intersection + 1) / (K.sum(y_true_f) + K.sum(y_pred_f) +
1)
def dice_coef_loss(y_true, y_pred):
    return -dice_coef(y_true, y_pred)

def conv_block(m, dim, acti, bn, res, do=0):
    n = Conv2D(dim, 3, activation=acti, padding='same')(m)
    n = BatchNormalization()(n) if bn else n
    n = Dropout(do)(n) if do else n
    n = Conv2D(dim, 3, activation=acti, padding='same')(n)
    n = BatchNormalization()(n) if bn else n

```

```

    return Concatenate()([m, n]) if res else n
def level_block(m, dim, depth, inc, acti, do, bn, mp, up, res):
    if depth > 0:
        n = conv_block(m, dim, acti, bn, res)
        m = MaxPooling2D()(n) if mp else Conv2D(dim, 3, strides=2,
padding='same')(n)
        m = level_block(m, int(inc*dim), depth-1, inc, acti, do, bn, mp, up,
res)
        if up:
            m = UpSampling2D()(m)
            m = Conv2D(dim, 2, activation=acti, padding='same')(m)
        else:
            m = Conv2DTranspose(dim, 3, strides=2, activation=acti,
padding='same')(m)
            n = Concatenate()([n, m])
            m = conv_block(n, dim, acti, bn, res)
        else:
            m = conv_block(m, dim, acti, bn, res, do)
    return m
def UNet(img_shape, out_ch=1, start_ch=64, depth=4, inc_rate=2.,
activation='relu',
        dropout=0.5, batchnorm=False, maxpool=True, upconv=True,
residual=False):
    i = Input(shape=img_shape)
    o = level_block(i, start_ch, depth, inc_rate, activation, dropout,
batchnorm, maxpool, upconv, residual)
    o = Conv2D(out_ch, 1, activation='sigmoid')(o)

    return Model(inputs=i, outputs=o)
UNET_model = UNet((img_dims, img_dims, 1), start_ch=32, depth=5,
batchnorm=True)

UNET_model.compile(
    optimizer=tf.optimizers.Adam(lr=1e-5),
    loss=dice_coef_loss,
    metrics=[dice_coef]
)
UNET_model.summary()
models_path = os.path.join(main_path, 'models', 'UNET_model')
model_json = UNET_model.to_json()
with open(os.path.join(models_path, 'model.json'), 'w') as json_file:
    json_file.write(model_json)
file_model_path = os.path.join(models_path, 'model_{epoch:02d}-
{loss:.3f}-{val_loss:.3f}.hdf5')

checkpoint = ModelCheckpoint(
    filepath=file_model_path,
    monitor='val_loss',
    save_best_only=True,

```

```

        mode='min',
        save_weights_only=True
    )
    reduceLRonPlat = ReduceLRonPlateau(
        monitor='val_loss',
        factor=0.5,
        patience=3,
        verbose=1,
        mode='min',
        min_delta=0.0001,
        cooldown=2,
        min_lr=1e-6
    )
    csv_logger = CSVLogger(os.path.join(models_path, 'training.log'))
    history = unet_model.fit(
        train_generator,
        validation_data=validation_generator,
        callbacks=[checkpoint, reduceLRonPlat, csv_logger],
        batch_size=batch_size,
        steps_per_epoch=len(image_train_generator),
        validation_batch_size=batch_size,
        validation_steps=len(image_validation_generator),
        epochs=10
    )
    training = pd.read_csv(os.path.join(models_path, 'training.log'))
    plt.plot(training['epoch'] + 1, training['dice_coef'])
    plt.plot(training['epoch'] + 1, training['val_dice_coef'])
    # plt.title('Обучение')
    plt.ylabel('Коэффициент Dice')
    plt.xlabel('Эпоха')
    plt.xticks(training['epoch'] + 1)
    plt.legend(['Train', 'Test'])
    plt.show()
    plt.plot(training['epoch'] + 1, training['loss'])
    plt.plot(training['epoch'] + 1, training['val_loss'])
    plt.title('Model loss')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.xticks(training['epoch'] + 1)
    plt.legend(['Train', 'Test'])
    plt.show()

    path_to_model = os.path.join(models_path, 'model_02--0.929--0.938.hdf5')
    path_to_model_json = os.path.join(models_path, 'model.json')
    with open(path_to_model_json, 'r') as json_file:
        loaded_model_json = json_file.read()
    loaded_model = model_from_json(loaded_model_json)
    loaded_model.load_weights(path_to_model)
    def lung_segment(image, segment_model):

```

```

    image = preprocess_before_segmentation_without_noise(image)
    predict_mask = segment_model.predict(np.array([image]))
    return predict_mask[0]
def postprocess(
    image,
    morph_open_kernel_size,
    morph_close_kernel_size
):
    _, result = cv2.threshold(image, 0.5, 1, 0)
    kernel_open = np.ones((morph_open_kernel_size, morph_open_kernel_size),
np.uint8)
    result = cv2.morphologyEx(result, cv2.MORPH_OPEN, kernel_open)
    selem = morphology.disk(morph_close_kernel_size)
    result = morphology.closing(result, selem)
    result = result.reshape(image.shape)
    return result
fig, ax = plt.subplots(1, 5, figsize=(20, 20))

# person70_virus_130.jpeg
file_path = os.path.join(main_path, 'test_data', 'unlabelled',
'person70_virus_130.jpeg')
image_dims = 512
image = tf.keras.preprocessing.image.load_img(file_path,
color_mode='grayscale', target_size = (image_dims, image_dims))
image = tf.keras.preprocessing.image.img_to_array(image)
image = np.array(image)
input_image = image
ax[0].set_axis_off()
ax[0].imshow(image[:, :, 0], cmap='gray')
# ax[0].set_title('Input image')
image = preprocess_before_segmentation_without_noise(image)
ax[1].set_axis_off()
ax[1].imshow(image[:, :, 0], cmap='gray')
# ax[1].set_title('After preprocess')
result_mask = loaded_model.predict(np.array([image]))[0]
ax[2].set_axis_off()
ax[2].imshow(result_mask[:, :, 0], cmap='gray')
# ax[2].set_title('After lung segment')
result_mask = postprocess(result_mask, 12, 12)
ax[3].set_axis_off()
ax[3].imshow(result_mask[:, :, 0], cmap='gray')
# ax[3].set_title('After postprocess')

result = result_mask[:, :, 0] * input_image[:, :, 0]
ax[4].set_axis_off()
ax[4].imshow(result, cmap='gray')
# ax[4].set_title('Result')
plt.show()

```

5.5. Определение пневмонии по рентгеновским снимкам грудной клетки. Xception

Одним из методов диагностики пневмонии является рентгенография грудной клетки – проведение рентгеновского исследования для получения изображений легких и окружающих тканей. Этот метод является широко распространенным и доступным, однако его интерпретация требует специалиста с опытом визуализации и анализа рентгеновских снимков.

В данной работе предлагается разработать алгоритм автоматического определения пневмонии по рентгеновским снимкам грудной клетки с помощью инструмента Jupyter Notebook. Алгоритм будет основан на анализе и обработке изображений, используя современные методы машинного обучения и компьютерного зрения.

Использованные библиотеки

- cv2 (OpenCV) - библиотека компьютерного зрения, широко используемая для обработки изображений, работы с видео и компьютерного зрения.
- os - библиотека для работы с операционной системой, позволяет взаимодействовать с файловой системой, создавать, перемещать и удалять файлы и директории.
- pandas - библиотека для анализа и обработки данных, предоставляющая структуры данных и операции для работы с ними, особенно для табличных данных.
- numpy - библиотека для работы с многомерными массивами данных и выполнения математических операций над ними.
- seaborn - библиотека для визуализации данных, предоставляющая высокоуровневый интерфейс для создания информативных и красивых графиков.
- matplotlib - библиотека для визуализации данных, предоставляющая широкие возможности для создания различных видов графиков.

- tensorflow - библиотека для разработки и обучения моделей машинного обучения и глубокого обучения. Она предоставляет инструменты для создания нейронных сетей, оптимизации моделей и обработки данных.
- keras - высокоуровневый интерфейс для создания нейронных сетей, который является частью TensorFlow. Позволяет создавать модели с помощью менее кода и облегчает процесс экспериментирования с архитектурами сетей.
- sklearn - библиотека машинного обучения, предоставляющая инструменты для различных задач, таких как классификация, регрессия, кластеризация и другие.
- tensorflow_hub - библиотека для загрузки и повторного использования предварительно обученных моделей и их частей.
- EarlyStopping - коллбэк в TensorFlow для ранней остановки обучения модели, когда заданное условие (например, прекращение улучшения производительности на валидационном наборе данных) выполняется.
- warnings - модуль для управления предупреждениями в Python, который позволяет отключать и настраивать вывод предупреждений.

Набор данных

Набор данных организован в 3 папки (train, test, val) и содержит подпапки для каждой категории изображений (Pneumonia/Normal). В наборе данных представлено 5 863 рентгеновских снимка грудной клетки в формате JPEG, относящихся к двум категориям (пневмония/норма).

Изображения рентгеновских снимков грудной клетки (продольно-поперечное) были выбраны из ретроспективных наборов данных педиатрических пациентов возрастом от одного до пяти лет из Гуанчжоуского центра женского и детского здоровья, Гуанчжоу. Все рентгенологические исследования грудной клетки проводились в рамках обычного клинического ухода за пациентами.

Для анализа изображений рентгеновских снимков грудной клетки все рентгенограммы изначально прошли процедуру контроля качества,

при которой удалены все изображения низкого качества или нечитаемые сканы. Диагностика изображений затем была оценена двумя опытными врачами, прежде чем они были использованы для обучения системы искусственного интеллекта. Для проверки возможных ошибок в оценке, набор данных для оценки также был проверен третьим экспертом.

Таким образом, этот набор данных представляет собой ценный ресурс для обучения и тестирования разработанных моделей машинного обучения для автоматического определения пневмонии по рентгеновским снимкам грудной клетки. Он предоставляет доступ к большому количеству аутентичных изображений, которые были оценены и проверены специалистами для обеспечения высокого качества данных.

Предварительная обработка изображений

Предварительная обработка изображений выполняется с целью улучшить качество и эффективность обучения модели, а также повысить ее способность обобщать знания на новые данные.

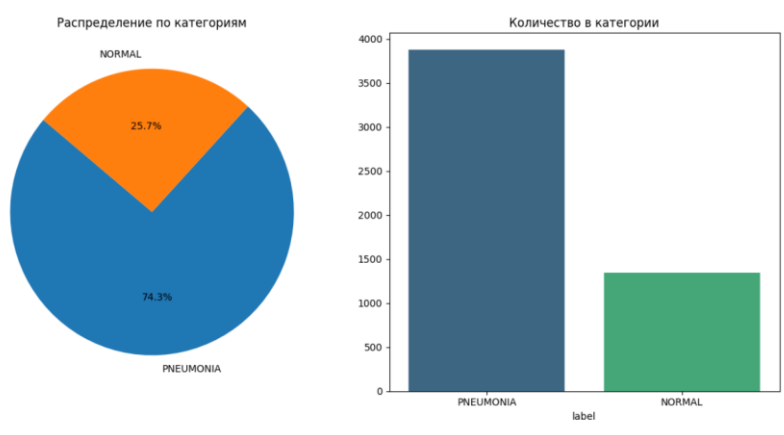
Вот основные этапы, которые были проделаны:

- Размер изображений: 256 на 256 пикселей.
- Нормализация изображений: Приведение значений пикселей к стандартному диапазону может помочь улучшить производительность модели машинного обучения. Значения пикселей нормализуются путем деления на 255, чтобы они находились в диапазоне от 0 до 1.
- Валидационные данные: данные из тренировочной директории разделяются на тренировочный и валидационный наборы с соотношением валидации 10%.
- Размер пакета: во время обучения модели данные разбиваются на небольшие группы, называемые пакетами или мини-пакетами. Размер пакета определяет количество образцов данных в каждом пакете.

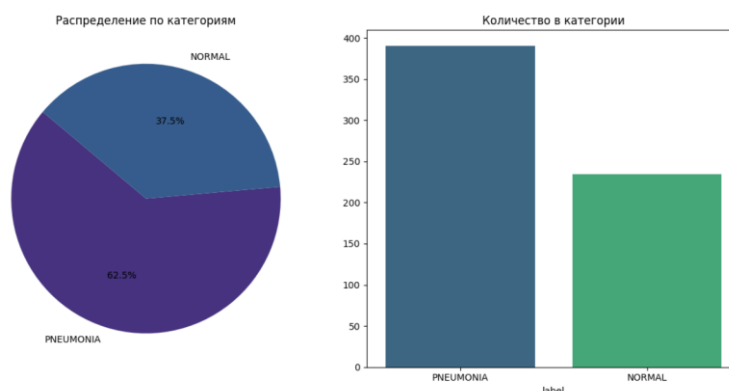
Установлено значение размера пакета в 32, значит, каждый пакет будет содержать 32 образца данных.

В целом, предварительная обработка изображений играет важную роль в создании качественных моделей машинного обучения. Она помогает улучшить производительность модели, повысить ее способность обобщать знания на новые данные и обеспечить более точные и интерпретируемые результаты.

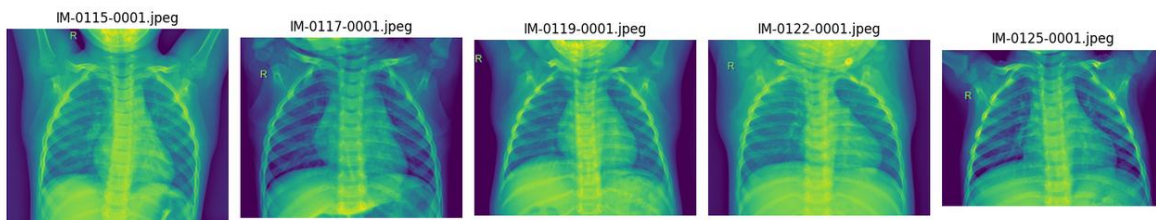
Визуальное представление данных



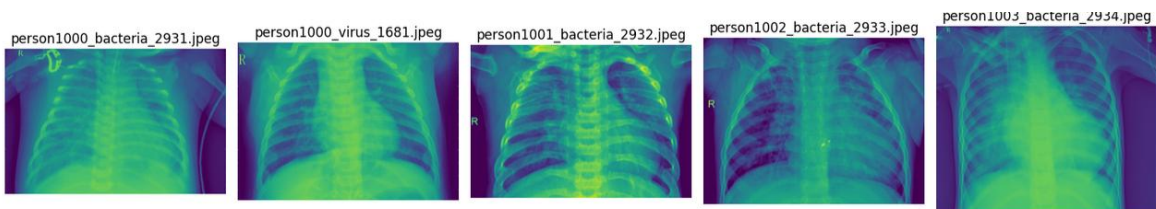
Тренировочный набор данных



Тестирующий набор данных



Нормальные (здоровые) рентгеновские снимки



Рентгеновские снимки с пневмонией

Построение модели нейросети

Создаем нейронную сеть для классификации изображений с использованием предварительно обученной модели Xception в качестве базовой модели.

Основные этапы:

1. Загружаем предварительно обученную модель Xception без верхнего (классификационного) слоя, с весами, обученными на датасете ImageNet. Указывается размер входного изображения (256x256x3) и параметр `pooling='avg'`, чтобы добавить глобальное среднее пулингование для усреднения признаков.
2. Замораживаем веса базовой модели, чтобы предотвратить их обновление во время обучения новой модели.
3. Создаем модель для построения слоев последовательно.
4. Добавляем базовую модель Xception в качестве первого слоя в модель.
5. Добавляем слой Batch Normalization для стабилизации обучения и ускорения сходимости.

6. Добавляем слой Dropout для регуляризации, который случайным образом отключает 45% нейронов во время обучения, чтобы предотвратить переобучение.
7. Добавляем полносвязный слой с 220 нейронами и функцией активации ReLU.
8. Добавляем еще один слой Dropout с коэффициентом 0.25 для дополнительной регуляризации.
9. Добавляем еще один полносвязный слой с 60 нейронами и функцией активации ReLU.
10. Добавляем выходной слой с одним нейроном и сигмоидной функцией активации для бинарной классификации.
11. Компилируем модель с оптимизатором Adamax, функцией потерь `binary_crossentropy` (для бинарной классификации) и метрикой `accuarcy` для оценки производительности модели.

Обучение модели

Используем `Early Stopping` для остановки обучения модели при наблюдении за улучшением или ухудшением валидационной потери. `EarlyStopping` будет отслеживать валидационную потерю и прерывать обучение, если она не улучшается в течение заданного количества эпох (`patience=5`). Параметр `restore_best_weights=True` гарантирует восстановление лучших весов модели после прекращения обучения.

Обучение происходит на тренировочных данных в течение 20 эпох. Валидационные данные используются для проверки производительности модели после каждой эпохи. Обратный вызов `early_stopping` передается для отслеживания и применения ранней остановки в случае необходимости.

Таким образом, этот алгоритм реализует стратегию ранней остановки, чтобы предотвратить переобучение модели и сохранить лучшие веса модели на основе валидационной потери.

Тренировка нейронной сети происходила на GPU NVIDIA RTX 3060.

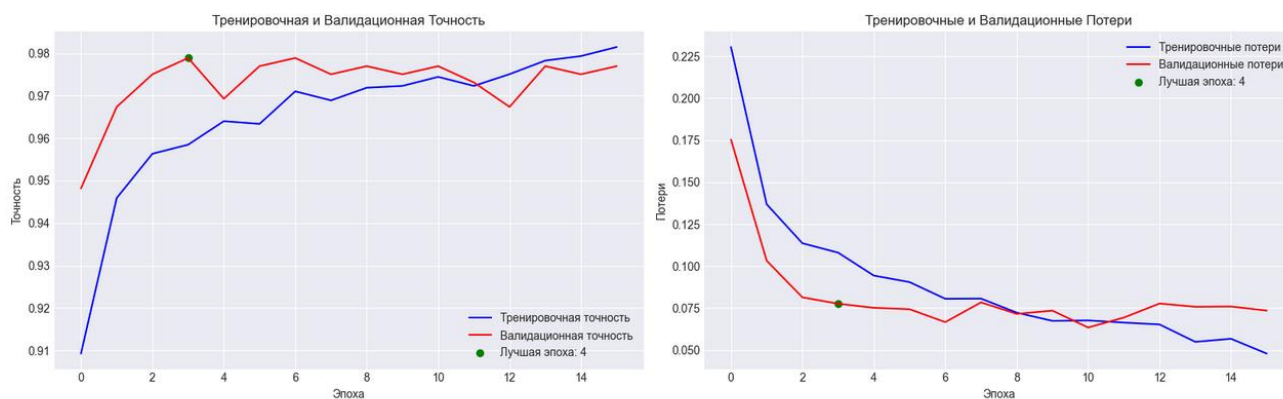
```
Epoch 1/20
147/147 [=====] - 39s 174ms/step - loss: 0.2304 - accuracy: 0.9093 - val_loss: 0.1753 - val_accuracy: 0.9482
Epoch 2/20
147/147 [=====] - 23s 147ms/step - loss: 0.1368 - accuracy: 0.9459 - val_loss: 0.1032 - val_accuracy: 0.9674
Epoch 3/20
147/147 [=====] - 22s 146ms/step - loss: 0.1136 - accuracy: 0.9563 - val_loss: 0.0814 - val_accuracy: 0.9750
Epoch 4/20
147/147 [=====] - 22s 147ms/step - loss: 0.1080 - accuracy: 0.9585 - val_loss: 0.0776 - val_accuracy: 0.9789
Epoch 5/20
147/147 [=====] - 22s 145ms/step - loss: 0.0944 - accuracy: 0.9640 - val_loss: 0.0752 - val_accuracy: 0.9693
Epoch 6/20
147/147 [=====] - 22s 146ms/step - loss: 0.0905 - accuracy: 0.9634 - val_loss: 0.0744 - val_accuracy: 0.9770
Epoch 7/20
147/147 [=====] - 22s 147ms/step - loss: 0.0806 - accuracy: 0.9710 - val_loss: 0.0667 - val_accuracy: 0.9789
Epoch 8/20
147/147 [=====] - 23s 149ms/step - loss: 0.0807 - accuracy: 0.9689 - val_loss: 0.0784 - val_accuracy: 0.9750
Epoch 9/20
147/147 [=====] - 23s 150ms/step - loss: 0.0723 - accuracy: 0.9719 - val_loss: 0.0716 - val_accuracy: 0.9770
Epoch 10/20
147/147 [=====] - 22s 147ms/step - loss: 0.0674 - accuracy: 0.9723 - val_loss: 0.0735 - val_accuracy: 0.9750
Epoch 11/20
147/147 [=====] - 23s 149ms/step - loss: 0.0677 - accuracy: 0.9744 - val_loss: 0.0635 - val_accuracy: 0.9770
Epoch 12/20
147/147 [=====] - 23s 149ms/step - loss: 0.0664 - accuracy: 0.9723 - val_loss: 0.0694 - val_accuracy: 0.9731
Epoch 13/20
147/147 [=====] - 25s 162ms/step - loss: 0.0653 - accuracy: 0.9751 - val_loss: 0.0778 - val_accuracy: 0.9674
Epoch 14/20
147/147 [=====] - 24s 157ms/step - loss: 0.0549 - accuracy: 0.9783 - val_loss: 0.0758 - val_accuracy: 0.9770
Epoch 15/20
147/147 [=====] - 22s 147ms/step - loss: 0.0568 - accuracy: 0.9793 - val_loss: 0.0760 - val_accuracy: 0.9750
Epoch 16/20
147/147 [=====] - 23s 149ms/step - loss: 0.0480 - accuracy: 0.9815 - val_loss: 0.0735 - val_accuracy: 0.9770
```

Ход обучения модели нейронной сети

Потери и точность на валидационном наборе данных:

- Валидационные потери составили 6 %.
 - Валидационная точность составила 97 %.
- Это свидетельствует об успешном обучении модели.

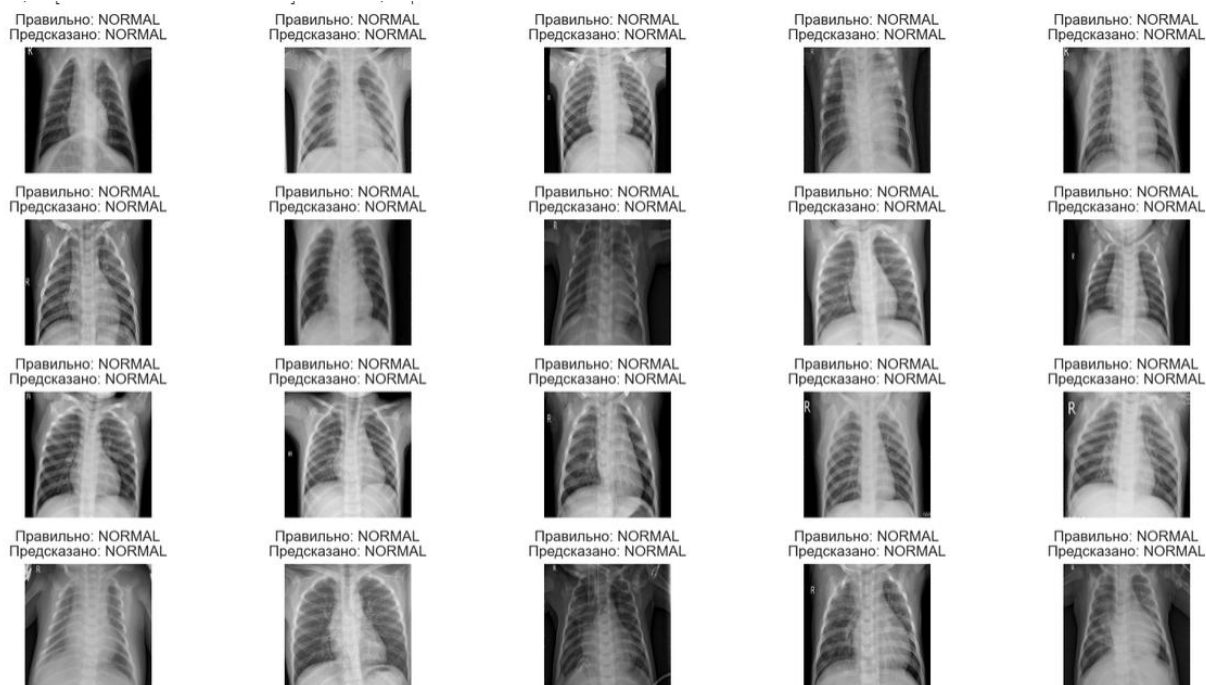
Графики хода обучения модели



Графики хода обучения модели нейронной сети

Прогнозы модели

Проверяем нейросеть на тестировочном наборе данных:



Прогнозы модели на тестировочном наборе данных

Как видно, нейросеть точно предсказывает, что данные снимки здоровы.



Прогнозы модели на тренировочном наборе данных с пневмонией

Использованы тестировочные данные из папки со снимками с пневмонией. Нейросеть правильно предсказала, что на данных снимках присутствует пневмония.

Для проверки нейросети скачан еще один набор данных. В данном наборе данных было 5000 рентгеновских снимков в формате JPEG. Нейросеть точно определила снимки с пневмонией и без нее. Можно считать, что обучение прошло успешно.

Текст программы

```
import cv2
import os
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import tensorflow as tf
from tensorflow import keras
from keras import Sequential
from keras.layers import *
from tensorflow.keras.losses import BinaryCrossentropy
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.optimizers import Adam , Adamax
from tensorflow.keras.applications import *
from tensorflow.keras.callbacks import EarlyStopping
import warnings
warnings.filterwarnings("ignore")

directory = 'C:/Users/alexey/datasets/chest_xray/train'
filepath = []
label = []
folds = os.listdir(directory)

for fold in folds:
    f_path = os.path.join(directory , fold)
    imgs = os.listdir(f_path)

    for img in imgs:
        img_path = os.path.join(f_path , img)
        filepath.append(img_path)
        label.append(fold)
file_path_series = pd.Series(filepath , name= 'filepath')
Label_path_series = pd.Series(label , name = 'label')
df_train = pd.concat([file_path_series ,Label_path_series ] , axis = 1)

directory = "C:/Users/alexey/datasets/chest_xray/test"
filepath = []
label = []
folds = os.listdir(directory)
for fold in folds:
    f_path = os.path.join(directory , fold)

    imgs = os.listdir(f_path)

    for img in imgs:
```

```

        img_path = os.path.join(f_path , img)
        filepath.append(img_path)
        label.append(fold)

file_path_series = pd.Series(filepath , name= 'filepath')
Label_path_series = pd.Series(label , name = 'label')
df_test = pd.concat([file_path_series ,Label_path_series ] , axis = 1)
data_dir = 'C:/Users/alexu/datasets/chest_xray/train'
test_dir = 'C:/Users/alexu/datasets/chest_xray/test'
IMAGE_SIZE = (256,256)

print('Тренировочные изображения:')
train_ds = tf.keras.utils.image_dataset_from_directory(
    data_dir,
    validation_split=0.1,
    subset='training',
    seed=123,
    image_size=IMAGE_SIZE,
    batch_size=32)

print('Валидационные изображения:')
validation_ds = tf.keras.utils.image_dataset_from_directory(
    data_dir,
    validation_split=0.1,
    subset='validation',
    seed=123,
    image_size=IMAGE_SIZE,
    batch_size=32)

print('Тестируемые изображения:')
test_ds = tf.keras.utils.image_dataset_from_directory(
    test_dir,
    seed=123,
    image_size=IMAGE_SIZE,
    batch_size=32)
train_labels = train_ds.class_names
test_labels = test_ds.class_names
validation_labels = validation_ds.class_names
class_labels = ['NORMAL', 'PNEUMONIA']
label_encoder = LabelEncoder()
label_encoder.fit(class_labels)
train_labels_encoded = label_encoder.transform(train_labels)
validation_labels_encoded = label_encoder.transform(validation_labels)

test_labels_encoded = label_encoder.transform(test_labels)
for image_batch, labels_batch in train_ds:
    print("Shape of X_train: ", image_batch.shape)
    print("Shape of y_train: ", labels_batch.shape)
    break

```

```

# Train Data
train_ds = train_ds.map(lambda x, y: (x / 255.0, y))
# Val Data
validation_ds = validation_ds.map(lambda x, y: (x / 255.0, y))
# Test Data
test_ds = test_ds.map(lambda x, y: (x / 255.0, y))

count = df_train['label'].value_counts()

# Create a figure with two subplots
fig, axs = plt.subplots(1, 2, figsize=(12, 6), facecolor='white')

# Plot pie chart on the first subplot
palette = sns.color_palette("viridis")
sns.set_palette(palette)
axs[0].pie(count, labels=count.index, autopct='%1.1f%%', startangle=140)
axs[0].set_title('Распределение по категориям')

# Plot bar chart on the second subplot
sns.barplot(x=count.index, y=count.values, ax=axs[1], palette="viridis")
axs[1].set_title('Количество в категории')
plt.tight_layout()
plt.show()
count = df_test['label'].value_counts()

# Create a figure with two subplots
fig, axs = plt.subplots(1, 2, figsize=(12, 6), facecolor='white')

# Plot pie chart on the first subplot
palette = sns.color_palette("viridis")
sns.set_palette(palette)
axs[0].pie(count, labels=count.index, autopct='%1.1f%%', startangle=140)
axs[0].set_title('Распределение по категориям')

# Plot bar chart on the second subplot
sns.barplot(x=count.index, y=count.values, ax=axs[1], palette="viridis")
axs[1].set_title('Количество в категории')
plt.tight_layout()
plt.show()
def visualize_images(path, num_images=5):
    image_filenames = os.listdir(path)

    num_images = min(num_images, len(image_filenames))

    # Create a figure and axis object to display images
    fig, axes = plt.subplots(1, num_images, figsize=(15,
3), facecolor='white')

    # Iterate over the selected images and display them

```

```

for i, image_filename in enumerate(image_filenames[:num_images]):
    # Load the image using Matplotlib
    image_path = os.path.join(path, image_filename)
    image = mpimg.imread(image_path)

    # Display the image
    axes[i].imshow(image)
    axes[i].axis('off') # Turn off axis
    axes[i].set_title(image_filename) # Set image filename as title

plt.tight_layout()
plt.show()
path_to_visualize = "C:/Users/alexey/datasets/chest_xray/train/NORMAL"
visualize_images(path_to_visualize, num_images=5)
path_to_visualize = "C:/Users/alexey/datasets/chest_xray/train/PNEUMONIA"
visualize_images(path_to_visualize, num_images=5)
base_model = Xception(weights='imagenet', include_top=False,
pooling='avg', input_shape=(256, 256, 3))
base_model.trainable = False

model = Sequential()
model.add(base_model)
model.add(BatchNormalization())
model.add(Dropout(0.45))
model.add(Dense(220, activation='relu'))
model.add(Dropout(0.25))
model.add(Dense(60, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer=Adamax(learning_rate=0.001),
loss='binary_crossentropy', metrics=['accuracy'])

model.summary()
early_stopping = EarlyStopping(monitor='val_loss', patience=5,
restore_best_weights=True)
history = model.fit_generator(train_ds,
                             epochs= 20,
                             validation_data = validation_ds,
                             callbacks = early_stopping)
validation_loss, validation_accuracy = model.evaluate(validation_ds)
print("Валидационные потери:", validation_loss)
print("Валидационная точность:", validation_accuracy)
best_epoch =
history.history['val_accuracy'].index(max(history.history['val_accuracy']
)) + 1
plt.style.use('seaborn-v0_8-darkgrid')
fig, axs = plt.subplots(1, 2, figsize=(16, 5))
axs[0].plot(history.history['accuracy'], label='Тренировочная точность',
color='blue')

```

```

axs[0].plot(history.history['val_accuracy'], label='Валидационная
точность', color='red')
axs[0].scatter(best_epoch - 1, history.history['val_accuracy'][best_epoch
- 1], color='green', label=f'Лучшая эпоха: {best_epoch}')
axs[0].set_xlabel('Эпоха')
axs[0].set_ylabel('Точность')
axs[0].set_title('Тренировочная и Валидационная Точность')
axs[0].legend()
axs[1].plot(history.history['loss'], label='Тренировочные потери',
color='blue')
axs[1].plot(history.history['val_loss'], label='Валидационные потери',
color='red')
axs[1].scatter(best_epoch - 1, history.history['val_loss'][best_epoch -
1], color='green', label=f'Лучшая эпоха: {best_epoch}')
axs[1].set_xlabel('Эпоха')
axs[1].set_ylabel('Потери')
axs[1].set_title('Тренировочные и Валидационные Потери')
axs[1].legend()

plt.tight_layout()
plt.show()
test_dir = 'C:/Users/alexey/datasets/chest_xray/test'
print('Testing Images:')
test_ds = tf.keras.utils.image_dataset_from_directory(
    test_dir,
    seed=123,
    image_size=IMAGE_SIZE,
    batch_size=32)
def plot_images_with_predictions(model, dataset, class_labels,
num_images=40, num_images_per_row=5):
    predictions = model.predict(dataset)
    dataset_shuffled = dataset.shuffle(buffer_size=len(dataset))
    plt.figure(figsize=(15, 10))
    for i, (images, labels) in
enumerate(dataset_shuffled.take(num_images)):
        # Convert tensor to NumPy array
        images = images.numpy()

        for j in range(len(images)):
            if i * num_images_per_row + j < num_images: # Check if the
total number of images exceeds the desired count
                predicted_class = class_labels[np.argmax(predictions[i *
num_images_per_row + j])]
                true_class = class_labels[np.argmax(labels[j])]

                plt.subplot(num_images // num_images_per_row + 1,
num_images_per_row, i * num_images_per_row + j + 1)
                plt.imshow(images[j].astype("uint8"))

```

```

        plt.title(f'Правильно: {true_class}\nПредсказано:
{predicted_class}')
        plt.axis('off')
    plt.tight_layout()
    plt.show()
print('Testing Images:')
plot_images_with_predictions(model, test_ds, class_labels, num_images=20)
directory_path = 'C:/Users/alexy/datasets/chest_xray/test/PNEUMONIA'
image_files = os.listdir(directory_path)[:5]
fig, axs = plt.subplots(1, len(image_files), figsize=(15, 5))
for i, image_file in enumerate(image_files):
    img_path = os.path.join(directory_path, image_file)
    img = cv2.imread(img_path)
    img = cv2.resize(img, (256, 256))
    img_array = img.astype(np.float32) / 255.0
    img_array = np.expand_dims(img_array, axis=0)
    predictions = model.predict(img_array)
    actual_prediction = (predictions > 0.5).astype(int)
    axs[i].imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
    axs[i].axis('off')
    if actual_prediction[0][0] == 0:
        predicted_label = 'Normal'
    else:
        predicted_label = 'PNEUMONIA'
    axs[i].set_title(f'Предсказано: {predicted_label}')
plt.tight_layout()
plt.show()
directory_path =
'C:/Users/alexy/datasets/chest_xray_another/test/PNEUMONIA'
image_files = os.listdir(directory_path)[:5]
fig, axs = plt.subplots(1, len(image_files), figsize=(15, 5))
for i, image_file in enumerate(image_files):
    img_path = os.path.join(directory_path, image_file)
    img = cv2.imread(img_path)
    img = cv2.resize(img, (256, 256))
    img_array = img.astype(np.float32) / 255.0
    img_array = np.expand_dims(img_array, axis=0)
    predictions = model.predict(img_array)
    actual_prediction = (predictions > 0.5).astype(int)
    axs[i].imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
    axs[i].axis('off')
    if actual_prediction[0][0] == 0:
        predicted_label = 'Normal'
    else:
        predicted_label = 'PNEUMONIA'
    axs[i].set_title(f'Предсказано: {predicted_label}')

plt.tight_layout()
plt.show()

```

```

directory_path = 'C:/Users/alexy/datasets/chest_xray_another/test/NORMAL'
image_files = os.listdir(directory_path)[:5]
fig, axs = plt.subplots(1, len(image_files), figsize=(15, 5))
for i, image_file in enumerate(image_files):
    img_path = os.path.join(directory_path, image_file)
    img = cv2.imread(img_path)
    img = cv2.resize(img, (256, 256))
    img_array = img.astype(np.float32) / 255.0
    img_array = np.expand_dims(img_array, axis=0)
    predictions = model.predict(img_array)
    actual_prediction = (predictions > 0.5).astype(int)
    axs[i].imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
    axs[i].axis('off')
    if actual_prediction[0][0] == 0:
        predicted_label = 'Normal'
    else:
        predicted_label = 'PNEUMONIA'
    axs[i].set_title(f'Предсказано: {predicted_label}')
plt.tight_layout()
plt.show()

```

5.6. Игра «Rock-paper-scissor». Tensorflow

Программа реализует игру «камень-ножницы-бумага» между человеком и компьютером. При нажатии на клавишу «z» начинается отсчет от 0 до 3, отслеживается одна из рук игрока и генерируется одно из состояний в соседнем окне за счет нейронной сети. Далее выводится счет на основании полученного результата, учитывая правила игры:

- Камень затупляет ножницы
- Ножницы режут бумагу
- Бумага накрывает камень

При нажатии на клавишу «z» игра начинается заново, сохраняя результат, на «x» – игра заканчивается.

В программе используются библиотеки OpenCV и Tensorflow.

Алгоритм работы

1. Собираем данные, где записаны ходы пользователя и компьютера.
2. Обучаем модель на собранных данных. Модель будет обучаться предсказывать следующий ход компьютера на основе предыдущих ходов игрока.

3. Захватываем изображение с видеокамеры.
4. Если была нажата клавиша «z», отслеживаем количество поднятых пальцев на руке.
 - 4.1. Если нет поднятых пальцев, результат – камень.
 - 4.2. Если подняты указательный и средний пальцы, результат – ножницы.
 - 4.3. Если все пальцы подняты, результат – бумага.
5. Обученная модель выводит одно из состояний (камень/ножницы/бумага) в соседнем окне компьютера.
6. По правилам игры выводится счет на основании полученных состояний от игрока и компьютера.
7. При нажатии на клавишу «x» игра завершается. При нажатии на клавишу «z» переходим к пункту 2.

Используемые в проекте методы

Метод	Применение
load_model	загрузка модели
predict	прогнозирование хода модели
Sequential	частный случай model, где модель представляет собой стек слоев с одним входом и одним выходом. Используется для создания модели.
compile	компиляция модели
fit	обучение модели
save	сохранение модели
VideoCapture	захват экрана с видеокамеры
findHands	отслеживание рук на экране
fingersUp	отслеживание поднятых пальцев руки

Результат работы



Текст программы

Сбор данных

```
import random
import csv

data = []
for _ in range(1000): # Примерно 1000 игр
    playerMove = random.randint(1, 3)
    AIMove = random.randint(1, 3)
    result = (playerMove - AIMove) % 3
    data.append([playerMove, AIMove, result])
with open('game_data.csv', 'w', newline='') as file:
    writer = csv.writer(file)
    writer.writerow(["PlayerMove", "AIMove", "Result"])
    writer.writerows(data)
```

Обучение модели

```
import pandas as pd
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

data = pd.read_csv('game_data.csv')
x = data[['PlayerMove']].values
y = data['AIMove'].values
x = (x - 1) / 2
y = (y - 1) / 2

model = Sequential()
model.add(Dense(10, input_dim=1, activation='relu'))
model.add(Dense(10, activation='relu'))
model.add(Dense(1, activation='linear'))
model.compile(loss='mean_squared_error', optimizer='adam',
metrics=['accuracy'])
model.fit(x, y, epochs=100, batch_size=10, verbose=2)
model.save('rps_model.h5')
```

Интеграция модели в игру

```
import time
import cvzone
from PIL import ImageFont
import cv2
from cvzone.HandTrackingModule import HandDetector
import numpy as np
from tensorflow.keras.models import load_model

model = load_model('rps_model.h5')

cap = cv2.VideoCapture(0)
cap.set(3, 640)
```

```

cap.set(4, 480)

detector = HandDetector(maxHands=1)

timer = 0
stateResult = False
stateGame = False
scores = [0, 0]

while True:
    imgBG = cv2.imread("resources/bg.png")
    success, img = cap.read()
    imgScaled = cv2.resize(img, (0, 0), None, 0.79166, 0.79166)
    imgScaled = imgScaled[:, 58:448]

    hands, imgHand = detector.findHands(imgScaled)
    if stateGame:
        if stateResult is False:
            timer = time.time() - initialTime
            cv2.putText(imgBG, str(int(timer)), (620, 425),
cv2.FONT_HERSHEY_PLAIN, 6, (255, 255, 255), 4)
            if timer > 3:
                playerMove = None
                stateResult = True
                timer = 0
                if hands:
                    hand = hands[0]
                    fingers = detector.fingersUp(hand)
                    if fingers == [0, 0, 0, 0, 0]:
                        playerMove = 1
                    elif fingers == [0, 1, 1, 0, 0]:
                        playerMove = 2
                    elif fingers == [1, 1, 1, 1, 1]:
                        playerMove = 3
            input_data = np.array([(playerMove - 1) / 2]).reshape(1, -1)
            prediction = model.predict(input_data)
            AIMove = int(round(prediction[0][0] * 2 + 1))
            imgAIMove = cv2.imread(f'resources/{AIMove}.png',
cv2.IMREAD_UNCHANGED)
            imgBG = cvzone.overlayPNG(imgBG, imgAIMove, (167, 310))

            if (playerMove == 1 and AIMove == 2) or \
                (playerMove == 2 and AIMove == 3) or \
                (playerMove == 3 and AIMove == 1):
                scores[1] += 1
            elif (AIMove == 1 and playerMove == 2) or \
                (AIMove == 2 and playerMove == 3) or \
                (AIMove == 3 and playerMove == 1):
                scores[0] += 1

```

```

imgBG[257:637, 795:1185] = imgScaled
if stateResult:
    imgBG = cvzone.overlayPNG(imgBG, imgAIMove, (167, 310))
    cv2.putText(imgBG, str(scores[0]), (410, 225),
cv2.FONT_HERSHEY_PLAIN, 5, (255, 255, 255), 4)
    cv2.putText(imgBG, str(scores[1]), (1098, 225),
cv2.FONT_HERSHEY_PLAIN, 5, (255, 255, 255), 4)
    cv2.imshow("imageBG", imgBG)

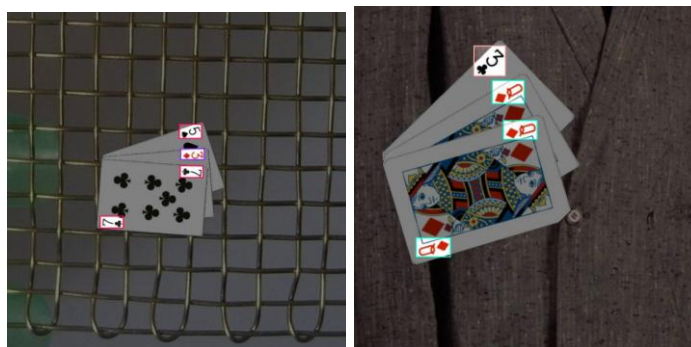
key = cv2.waitKey(1)
if key == ord('z'):
    stateGame = True
    initialTime = time.time()
    stateResult = False
elif key == ord('x'):
    break
cv2.destroyAllWindows()

```

5.8. Определение счета игральные карты в реальном времени с использованием модели YOLOv8

Программа способна автоматически определять масть и достоинство каждой карты на изображении, а также вести счет карт на экране для различных карточных игр. Анализ карт и ведение счета выполняется в реальном времени с помощью OpenCV.

Для распознавания карт использовалась модель YOLOv8s. Данная модель предназначена для обнаружения объектов на изображении. Модель была дообучена на наборе данных, взятого с онлайн сервиса Roboflow. Набор данных представлен специальным форматом, понятным для YOLOv8, где для каждого изображения есть аннотация, в которой описаны ограничивающие рамки каждого класса присутствующего на картинке:



В общей сложности для обучения, валидации и тренировки модели было использовано 24240 изображений, а каждое изображение содержало 2-4 карты.

Подсчет счета игральных карт

В данной программе реализован подсчет по правилам игры 21. Общий счет отображается в верхнем левом углу изображения транслируемого с веб-камеры компьютера. Пример:



В программе можно реализовать подсчет по своим правилам. Для этого надо изменить две функции:

- “card_value”: эта функции определяет стоимость карты
- “calculate_score”: задает правило подсчета карт

Оценка результата обучения модели

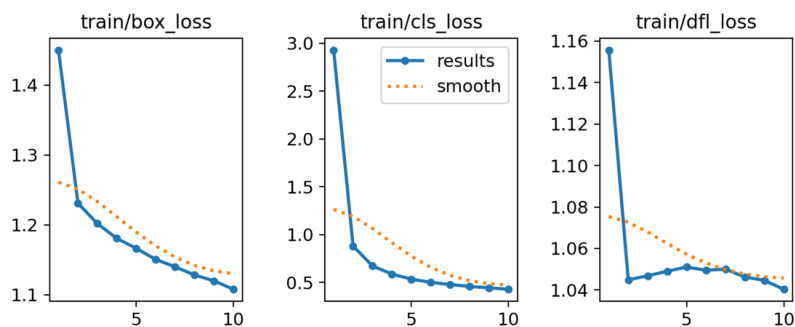
В модели YOLOv8 для оценки качества обучения используются следующие метрики:

- “box_loss”: измеряет погрешность в прогнозируемых координатах и размерах ограничивающей рамки по сравнению с исходными данными. Меньшее значение box_loss означает, что прогнозируемые ограничивающие рамки являются более точными

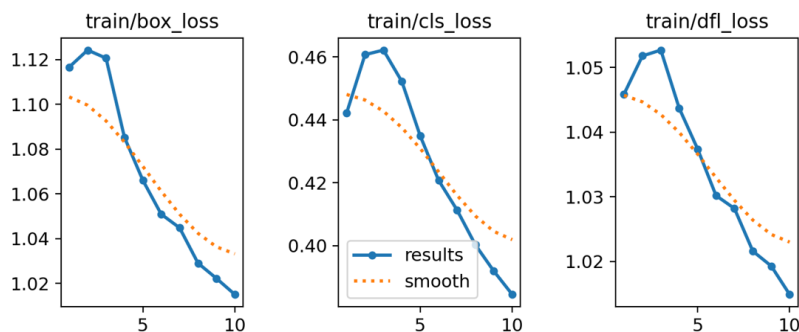
- “cls_loss”: измеряет ошибку в прогнозируемых вероятностях классов для каждого объекта на изображении по сравнению с истинным значением на местности. Меньший cls_loss означает, что модель более точно предсказывает класс объекта
- “dfl_loss”: измеряют погрешность в деформируемых слоях свертки, которые предназначены для улучшения способности модели обнаружить объекты с различными масштабами и соотношениями сторон. Меньшее значение dfl_loss указывает на то, что модель лучше справляется с деформациями объектов и изменениями внешнего вида

Модель обучалась в 2 этапа по 10 эпох каждый и получила следующую оценку качества обучения:

1 этап



2 этап



За 1 этап модель достигла точности box_loss ≈ 1.1 и cls_loss ≈ 0.5 . Но на деле этого было недостаточно, потому что иногда карта определялась неверно, поэтому было принято решение провести 2 этап обучения. На 2 этапе точность получилось улучшить до box_loss ≈ 1.02 и cls_loss ≈ 0.38 , что заметно сказалось на способность модели определять карты.

Используемые библиотеки и функции

Библиотека ultralytics

Название функции	Описание
model.train	Запускает процесс обучения модели.
model.predict	Предсказывает есть ли на изображении карты и дает информацию о каждом предсказанном объекте.
model.plot	На основе полученного предсказания рисует на изображении для каждого объекта ограничивающую рамку, класс и уверенность в точности.

Библиотека OpenCV

Название функции	Описание
cv2.VideoCapture	Создает объект захвата видео с веб-камеры компьютера.
cap.read	Считывает следующий кадр из видеопотока и возвращает два значения: флаг успешного чтения кадра и сам кадр.
cv2.resize	Изменяет размер изображения.
cv2.putText	Добавляет на изображение текст в заданное место.
cv2.imshow	Отображает кадр на экране.
cv2.waitKey	Ждет заданное время нажатие клавиши.
cap.release	Освобождает ресурсы, связанные с данным объектом захвата видео.

Текст программы

```
training.py:  
from ultralytics import YOLO  
model = YOLO('yolov8s.pt')  
model.train(data='C:/dataset/MEGA/dataset/data.yaml', epochs=20,  
imgsz=640)
```

```
playing_card_detector.py:  
import cv2  
from ultralytics import YOLO
```

```
def card_value(card):  
    rank = card[:-1]  
    if rank in ['K', 'Q', 'J']:  
        return 10  
    elif rank == 'A':  
        return 11  
    else:  
        return int(rank)
```

```
def calculate_score(cards):
```

```

score = 0
aces = 0
for card in cards:
    value = card_value(card)
    if value == 11:
        aces += 1
    score += value
while score > 21 and aces:
    score -= 10
    aces -= 1
return score

def detect_cards(results):
    detected_card_names = []
    for box in results[0].boxes:
        detected_card_names.append(results[0].names[int(box.cls[0])])
    return list(set(detected_card_names))

model = YOLO("playingCardDetectorModel.pt")
cap = cv2.VideoCapture(0, cv2.CAP_DSHOW)
while True:
    ret, frame = cap.read()
    if not ret:
        break

    frame = cv2.resize(frame, (640, 640))
    results = model(frame, verbose=False, conf=0.6)
    frame = results[0].plot()
    frame = cv2.resize(frame, (1280, 720))

    score = calculate_score(detect_cards(results))
    cv2.putText(frame, f'Score: {score}', (10, 30),
cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2, cv2.LINE_AA)
    cv2.imshow('Playing Card Detector', frame)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
cap.release

```

5.9. Распознавание шахматных фигур и анализ позиции. YOLO

Программа автоматически распознает шахматные фигуры по фотографии/скриншоту шахматной доски и генерирует корректную строку FEN, представляющую текущее расположение фигур. Затем производится анализ шахматной позиции. Программа позволяет автоматизировать процесс записи шахматных партий. В программе

используется модель YOLO для автоматического распознавания шахматных фигур на доске и преобразования данных в текстовый формат FEN (Forsyth-Edwards Notation). Анализ шахматной позиции производится на шахматном движке Stockfish. YOLO (You Only Look Once) используется для детекции объектов на изображениях в реальном времени.

На основе большого набора данных была дообучена модель нейросети YOLOv8m. Выбрана она была из-за высокой скорости работы и обучения.



Далее при помощи OpenCV было реализовано определение позиций фигур на доске. Данные о позиции каждой фигуры можно преобразовать в текстовый формат описания шахматной позиции, в FEN формат. Далее позиция в FEN формате передается в шахматный движок Stockfish, который выводит информацию о валидности позиции и лучший ход для данной позиции.

Особенность программы заключается в возможности быстро анализировать позицию. Так как в современных анализаторах позицию необходимо выстраивать самостоятельно, то есть каждую фигуру нужно перетаскивать на поле вручную. А в программе позиция восстанавливается по фото или скриншоту виртуальной шахматной доски.

Алгоритм работы программы

1. Распознавание шахматных фигур
2. Преобразование позиции
3. Анализ на движке Stockfish

Используемые функции

Функция	Описание
<code>YOLO('/best (1).pt')</code>	Загрузка модели YOLO
<code>model.predict('/photo_2024-05-28_14-36-18.jpg')</code>	Предсказание фигур на изображении
<code>def yolo_to_fen(results, img_shape, class_names, who_move)</code>	Функция для преобразования выходных данных YOLO в FEN
<code>cv2.imread('/photo_2024-05-28_14-36-18.jpg')</code>	Чтение изображения для получения его размеров
<code>str(input())</code>	Ввод цвета, который должен сделать следующий ход
<code>print(stockfish.is_fen_valid(fen_string))</code>	Вывод информации о корректности полученной FEN позиции
<code>print("best move to " + who_move + ":", stockfish.get_best_move())</code>	Вывод лучшего хода по мнению Stockfish

Результат

FEN: r1b2bkr/ppp3pp/2n5/3qp3/2B5/8/PPPP1PPP/RNB1K2R w - - 0 1



True

best move to w: c4d5

Текст программы

```
from google.colab.patches import cv2_imshow
import numpy as np
import cv2
```

```
from ultralytics import YOLO
from stockfish import Stockfish
```

```
model = YOLO('/best (1).pt')
results = model.predict('/photo_2024-05-28_14-36-18.jpg')
```

```
def yolo_to_fen(results, img_shape, class_names, who_move):
    board = [['' for _ in range(8)] for _ in range(8)]
    img_height, img_width = img_shape[:2]
    for result in results:
        boxes = result.boxes
        for box in boxes:
            x1, y1, x2, y2 = box.xyxy[0].cpu().numpy()
            class_id = int(box.cls[0].cpu().numpy())
            col = int((x1 + x2) / 2 / img_width * 8)
            row = int((y1 + y2) / 2 / img_height * 8)
            piece = class_names[class_id]
            board[row][col] = piece

    fen = ''
    for row in board:
        empty_count = 0
        for cell in row:
            if cell == '':
                empty_count += 1
            else:
                if empty_count > 0:
```

```

        fen += str(empty_count)
        empty_count = 0
    fen += cell
    if empty_count > 0:
        fen += str(empty_count)
    fen += '/'
    fen = fen[:-1]

    fen += ' ' + who_move + ' - - 0 1'
    return fen

class_names = {
    0: 'b',
    1: 'k',
    2: 'n',
    3: 'p',
    4: 'q',
    5: 'r',
    6: 'B',
    7: 'K',
    8: 'N',
    9: 'P',
    10: 'Q',
    11: 'R'
}
img = cv2.imread('/photo_2024-05-28_14-36-18.jpg')
img_shape = img.shape

who_move = str(input())
fen_string = yolo_to_fen(results, img_shape, class_names, who_move)
print(fen_string)

results = model.predict('/photo_2024-05-28_14-36-18.jpg')
annotated_frame = results[0].plot()
cv2_imshow(annotated_frame)

stockfish_path = '/stockfish-ubuntu-x86-64-avx2'
!chmod +x /stockfish-ubuntu-x86-64-avx2
stockfish = Stockfish(path=stockfish_path)
stockfish.set_fen_position(fen_string)
print(stockfish.is_fen_valid(fen_string))
print("best move to " + who_move + ": ", stockfish.get_best_move())

```

5.10. Генерация музыки на основе текстового запроса. MusicGen

Replicate - это платформа для хостинга и запуска моделей машинного обучения с облачным API. Она позволяет легко использовать публичные

модели с открытым исходным кодом, такие как Meta Llama, Mistral, Stable Diffusion и другие. В ходе работы была использована модель MusicGen, позволяющая генерировать музыку на основе короткого текстового описания [74].

Meta MusicGen - это генеративная модель искусственного интеллекта от Meta, которая обучена генерировать музыку на основе запроса или мелодии. Она может превратить текстовое описание в аудио. Также можно загрузить композицию-референс, и тогда MusicGen создаст отрывок на её основе, следуя текстовым указаниям.

Метод MusicGen основан на авторегрессивной модели декодирования на основе трансформера. Он использует квантованные единицы из аудиотокенизатора EnCodec для моделирования музыки. Для сжатия и представления параллельных потоков данных, используется метод векторного квантования с использованием нескольких обученных кодировщиков.

Исследователи опубликовали 4 предварительно обученные модели:

`small`: модель объемом 300 миллионов параметров, работающая только с текстом;

`medium`: модель объемом 1.5 миллиарда параметров, работающая только с текстом;

`melody`: модель объемом 1.5 миллиарда параметров, работающая с текстом и мелодией-референсом;

`large`: модель объемом 3.3 миллиарда параметров, работающая только с текстом.

Подготовка

- Установите клиентскую библиотеку Python Replicate
pip install replicate

- Установите REPLICATE_API_TOKEN переменную окружения
`Export REPLICATE_API_TOKEN=r8_Psk****...*****`
 Или в виртуальном окружении создайте файл .env в который поместите строчку
`REPLICATE_API_TOKEN="r8_Psk****...*****"`
 Это ваш API-токен по умолчанию.
- Импортируйте клиент
`import replicate`
 Описание параметров, с которыми работает нейросеть
- `model_version` - Модель, используемая для генерации
- `prompt` - Описание музыки, которую вы хотите сгенерировать.
- `input_audio` - Аудиофайл, который будет влиять на сгенерированную музыку. Если значение "продолжение" равно "True", сгенерированная музыка будет продолжением аудиофайла. В противном случае сгенерированная музыка будет имитировать мелодию аудиофайла.
- `duration` - Длительность сгенерированного звука в секундах.
- `continuation` - Если `True`, сгенерированная музыка будет продолжена с `input_audio`. В противном случае сгенерированная музыка будет имитировать мелодию `input_audio`.
- `continuation_start` - Время начала аудиофайла для использования при продолжении.
- `continuation_end` - Время окончания аудиофайла, которое будет использоваться для продолжения. Если -1 или Нет, по умолчанию используется конец аудиоклипа.
- `multi_band_diffusion` - Если `True`, токены EnCodec будут декодированы с помощью многополосного распространения. Работает только с не стерео моделями.
- `normalization_strategy` – вид нормализации звука
- `top_k` - Сокращает выборку до k наиболее вероятных токенов.
- `top_p` - Сокращает выборку до токенов с кумулятивной вероятностью p. При значении `0` (по умолчанию) используется выборка `top_k`.
- `temperature` - Контролирует "консервативность" процесса сэмплирования. Более высокая температура означает большее разнообразие.

- `classifier_free_guidance` - Увеличивает влияние входных данных на выходные. Более высокие значения приводят к выходным данным с меньшим разбросом, которые более точно соответствуют входным данным.
- `output_format` - Формат вывода сгенерированного аудио.
- `seed` - Начальное значение для генератора случайных чисел. Если `None` или `-1`, будет использоваться случайное начальное значение.

Текст программы

```
import replicate
from dotenv import load_dotenv
from pprint import pprint
load_dotenv()
replicates_version =
"meta/musicgen:671ac645ce5e552cc63a54a2bbff63fcf798043055d2dac5fc9e36a837
eedcfb"
output = replicate.run(
replicates_version,
input={"top_k": 250,                ##Сокращает выборку до k
наиболее вероятных токенов.
      "top_p": 0,                    ##Сокращает выборку до токенов
с кумулятивной вероятностью p. При значении `0` (по умолчанию)
используется выборка top_k.
      "prompt": "country style music about lovely town",
      "duration": 25,                ##Длительность сгенерированного
звуча в секундах
      "temperature": 1,              ##Контролирует
"консервативность" процесса сэмплирования. Более высокая температура
означает большее разнообразие
      "continuation": False,         ##Если `True`, сгенерированная
музыка будет продолжена с `input_audio`. В противном случае
сгенерированная музыка будет имитировать мелодию `input_audio`.
      "model_version": "stereo-melody-large", ##Модель, используемая
для генерации
      "output_format": "mp3",        ##Формат output
      "multi_band_diffusion": False, ##Если `True`, токены EnCodec
будут декодированы с помощью многополосного распространения. Работает
только с не стерео моделями.
      "normalization_strategy": "peak", ##Вид нормализации звука.
      "classifier_free_guidance": 1, ##Увеличивает влияние входных
данных на выходные. Более высокие значения приводят к выходным данным с
меньшим разбросом, которые более точно соответствуют входным данным.
      "seed": -1}                    ##число начиная с которого
начнет свою работу нейросеть, по умолчанию -1
)
pprint(output)
output1 = replicate.run(
```

```

replicates_vertion,
input={
    "prompt": " lo fi elevator music with clear and soft saxophone,
catchy beat and groovy bass",
    "duration": 60
    "model_version": "stereo-large}
)
pprint(output1)
output2 = replicate.run(
replicates_vertion,
input={
    "prompt": "country style music about lovely town",
    "duration": 60,
    "model_version": "large",
    "normalization_strategy": "peak"}
)
pprint(output2)

```

5.11. Распознавание свежей клубники. TensorFlow

Реализована нейронная сеть, которая решает задачу классификации изображений. Она отличает изображения свежей клубники от изображений испорченной клубники.

Алгоритм работы

1. Подключим необходимые библиотеки
2. Подготовим обучающие данные (готовый dataset был взят с сайта <https://www.kaggle.com/datasets/alihasnainch/fruits-dataset-for-classification> и немного изменен)
3. Создадим папки с тестовыми данными и заполним их на основе обучающих данных
4. Создадим модель нейросети с помощью Sequential(), и будем последовательно добавлять слои, таким образом создав архитектуру
5. Скомпилируем модель, используя оптимизатор adam и функцию потерь binary_crossentropy (для бинарной классификации).
6. Обучаем модель
7. Используем обученную модель для решения нашей задачи (на каждой эпохе)
8. Строим график зависимости потерь (обучающих и валидационных) от эпох. Так же строим график зависимости точности (обучающей и валидационной) от эпох.

На обучающих данных обучена нейронная сеть на классификацию свежей и испорченной клубники. С помощью тестовых данных проверено, что нейронная сеть уже на 23 эпохе достигает валидационной точности более чем в 99 процентов. Это может свидетельствовать о том, что нейронная сеть работает корректно. Построены графики зависимости точности и потерь от количества эпох, чтобы наглядно увидеть как обучается нейронная сеть.

Таблица использованных функций

Функция / Класс	Описание	Аргументы
tf	Импорт библиотеки TensorFlow.	-
os.makedirs	Создание директории, если она не существует.	name (str): путь к создаваемой директории. exist_ok (bool): если True, не вызывает ошибку при существовании директории.
shutil.copyfile	Копирование файла из одного места в другое.	src (str): путь к исходному файлу. dst (str): путь к файлу назначения.
ImageDataGenerator	Создание объектов генератора данных с аугментацией изображений.	rescale (float, optional): рескейл множитель для пиксельных значений.
Sequential	Создание модели нейронной сети, состоящей из последовательных слоев.	layers (list, optional): список слоев для модели.
Conv2D	Свёрточный слой для извлечения признаков из изображений.	filters (int): количество фильтров. kernel_size (tuple): размер фильтра. activation (str): функция активации.
MaxPooling2D	Слой подвыборки, уменьшающий размерность данных.	pool_size (tuple, optional): размер окна подвыборки.
Flatten	Преобразование данных из двумерного в одномерное	-

	представление.	
Dense	Полносвязный слой нейронной сети.	units (int): количество нейронов. activation (str): функция активации.
Input	Входной слой для определения формы входных данных.	shape (tuple): форма входных данных.
model.compile	Компиляция модели с указанием функции потерь, оптимизатора и метрик.	loss (str): функция потерь. optimizer (str): оптимизатор. metrics (list): метрики.
model.summary	Вывод структуры модели и количества параметров.	-
flow_from_directory	Генерация данных из директории с изображениями.	directory (str): путь к директории данных. target_size (tuple): размер изображений. batch_size (int): размер пакета данных. class_mode (str): режим классов. shuffle (bool, optional): перемешивание данных.
tf.data.Dataset.from_generator	Создание объекта tf.data.Dataset из генератора данных.	generator: генератор данных. output_signature (tuple): спецификация вывода.
tf.TensorSpec	Определение спецификации для TensorFlow тензоров.	shape (tuple): форма тензора. dtype (tf.DType): тип данных тензора.
model.fit	Обучение модели на тренировочном наборе данных с валидацией.	x (tf.data.Dataset): тренировочный набор данных. steps_per_epoch (int): шагов за эпоху. epochs (int): количество эпох. validation_data (tf.data.Dataset): валидационный набор данных. validation_steps (int): шагов для валидации.
model.evaluate	Оценка модели на тестовом наборе данных.	x (tf.data.Dataset): тестовый набор данных. steps (int): шагов для оценки.

plt.plot	Построение графика потерь и точности.	x (list): данные для оси X. label (str): метка данных.
plt.xlabel	Установка метки для оси X на графике.	xlabel (str): метка оси X.
plt.ylabel	Установка метки для оси Y на графике.	ylabel (str): метка оси Y.
plt.legend	Добавление легенды к графику.	-
plt.show	Отображение графика.	-

Вывод программы

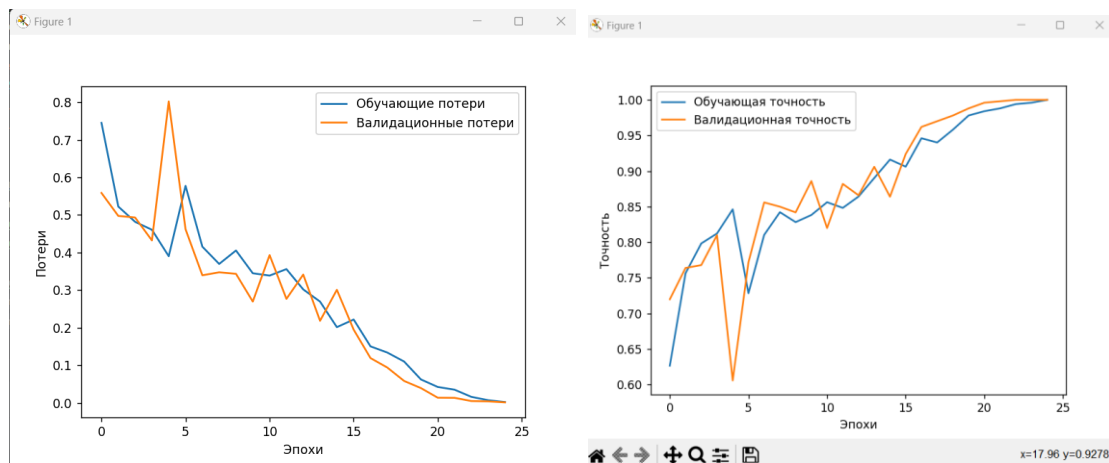
Обучение

```

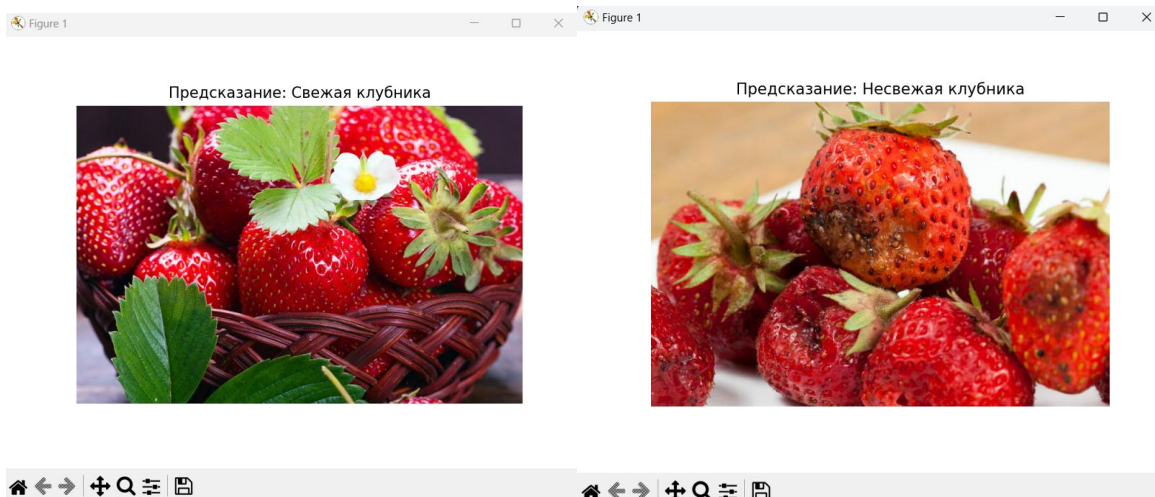
Epoch 15/25
26/26 ----- 7s 260ms/step - accuracy: 0.9266 - loss: 0.1754 - val_accuracy: 0.8640 - val_loss: 0.3007
Epoch 16/25
26/26 ----- 7s 256ms/step - accuracy: 0.8706 - loss: 0.2618 - val_accuracy: 0.9240 - val_loss: 0.1946
Epoch 17/25
26/26 ----- 7s 260ms/step - accuracy: 0.9454 - loss: 0.1317 - val_accuracy: 0.9620 - val_loss: 0.1190
Epoch 18/25
26/26 ----- 3s 121ms/step - accuracy: 0.9494 - loss: 0.1166 - val_accuracy: 0.9700 - val_loss: 0.0938
Epoch 19/25
26/26 ----- 3s 125ms/step - accuracy: 0.9509 - loss: 0.1128 - val_accuracy: 0.9780 - val_loss: 0.0581
Epoch 20/25
26/26 ----- 4s 139ms/step - accuracy: 0.9818 - loss: 0.0575 - val_accuracy: 0.9880 - val_loss: 0.0391
Epoch 21/25
26/26 ----- 4s 139ms/step - accuracy: 0.9804 - loss: 0.0435 - val_accuracy: 0.9960 - val_loss: 0.0135
Epoch 22/25
26/26 ----- 4s 137ms/step - accuracy: 0.9836 - loss: 0.0390 - val_accuracy: 0.9980 - val_loss: 0.0132
Epoch 23/25
26/26 ----- 4s 137ms/step - accuracy: 0.9922 - loss: 0.0170 - val_accuracy: 1.0000 - val_loss: 0.0046
Epoch 24/25
26/26 ----- 4s 140ms/step - accuracy: 0.9964 - loss: 0.0058 - val_accuracy: 1.0000 - val_loss: 0.0038
Epoch 25/25
26/26 ----- 4s 137ms/step - accuracy: 1.0000 - loss: 0.0028 - val_accuracy: 1.0000 - val_loss: 0.0011

```

Графики



Примеры распознавания



Текст программы

```
import tensorflow as tf
import os
import shutil
from tensorflow.keras.preprocessing.image import ImageDataGenerator,
load_img, img_to_array
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,
Input
import matplotlib.pyplot as plt
import numpy as np

# Создание папки для тестового набора данных
test_dataset_dir = 'Strawberry_dataset/test'
os.makedirs(test_dataset_dir, exist_ok=True)

# Копирование изображений свежей и несвежей клубники из общего набора
данных в тестовую папку
fresh_source_dir = 'Strawberry_dataset/train/fresh_strawberries'
rotten_source_dir = 'Strawberry_dataset/train/rotten_strawberries'

fresh_test_dir = os.path.join(test_dataset_dir, 'fresh_strawberries')
rotten_test_dir = os.path.join(test_dataset_dir, 'rotten_strawberries')

os.makedirs(fresh_test_dir, exist_ok=True)
os.makedirs(rotten_test_dir, exist_ok=True)

# Копирование изображений свежей клубники в тестовую папку
fresh_images = os.listdir(fresh_source_dir)
for img in fresh_images[:300]:
    src = os.path.join(fresh_source_dir, img)
    dst = os.path.join(fresh_test_dir, img)
    shutil.copyfile(src, dst)

# Копирование изображений несвежей клубники в тестовую папку
```

```

rotten_images = os.listdir(rotten_source_dir)
for img in rotten_images[:300]:
    src = os.path.join(rotten_source_dir, img)
    dst = os.path.join(rotten_test_dir, img)
    shutil.copyfile(src, dst)

# Создание модели нейронной сети
model = Sequential([
    Input(shape=(150, 150, 3)),
    Conv2D(32, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D(2, 2),
    Flatten(),
    Dense(512, activation='relu'),
    Dense(1, activation='sigmoid')
])
# Компиляция модели
model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

# Вывод информации о модели
model.summary()

# Подготовка данных для обучения и тестирования
train_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    'Strawberry_dataset/train',
    target_size=(150, 150),
    batch_size=20,
    class_mode='binary',
    shuffle=True
)
test_generator = test_datagen.flow_from_directory(
    'Strawberry_dataset/test',
    target_size=(150, 150),
    batch_size=20,
    class_mode='binary'
)
# Конвертация генераторов в tf.data.Dataset
def generator_to_dataset(generator):
    output_signature = (

```

```

        tf.TensorSpec(shape=(None, 150, 150, 3), dtype=tf.float32),
        tf.TensorSpec(shape=(None,), dtype=tf.float32)
    )
    dataset = tf.data.Dataset.from_generator(
        lambda: generator,
        output_signature=output_signature
    )
    return dataset
train_dataset = generator_to_dataset(train_generator).repeat()
test_dataset = generator_to_dataset(test_generator).repeat()

# Обучение модели
history = model.fit(
    train_dataset,
    steps_per_epoch=len(train_generator),
    epochs=25,
    validation_data=test_dataset,
    validation_steps=len(test_generator)
)
# Оценка модели на тестовом наборе данных
loss, accuracy = model.evaluate(test_dataset, steps=len(test_generator))
print("Потери:", loss)
print("Точность:", accuracy)

# Графики потерь и точности на обучающем и валидационном наборах данных
plt.plot(history.history['loss'], label='Обучающие потери')
plt.plot(history.history['val_loss'], label='Валидационные потери')
plt.xlabel('Эпохи')
plt.ylabel('Потери')
plt.legend()
plt.show()
plt.plot(history.history['accuracy'], label='Обучающая точность')
plt.plot(history.history['val_accuracy'], label='Валидационная точность')
plt.xlabel('Эпохи')
plt.ylabel('Точность')
plt.legend()
plt.show()

# Загрузка изображения для предсказания
def load_and_preprocess_image(image_path):
    img = load_img(image_path, target_size=(150, 150)) # загрузка
    изображения и изменение размера
    img_array = img_to_array(img) # преобразование изображения в массив
    img_array = img_array / 255.0 # нормализация значений пикселей
    img_array = np.expand_dims(img_array, axis=0) # добавление
    дополнительного измерения для пакета
    return img_array

# Загрузка изображения из тестовой папки

```

```

image_path = os.path.join(rotten_test_dir, rotten_images[0])#путь к
изображению, где хотим увидеть распознавание
img_array = load_and_preprocess_image(image_path)

# Предсказание класса изображения
prediction = model.predict(img_array)
predicted_class = 'Свежая клубника' if prediction[0] < 0.5 else 'Несвежая
клубника'

# Визуализация изображения и предсказания
plt.imshow(load_img(image_path))
plt.title(f'Предсказание: {predicted_class}')
plt.axis('off')
plt.show()

```

5.12. Возрастная трансформация лица на фотографии. SAM

Задача возрастной трансформации на изображении заключается в том, чтобы изменить внешний вид человека в соответствии с возрастом. Точно смоделировать реальное преобразование входного изображения лица чрезвычайно сложно, поскольку оно требует внесения убедительных и, в некоторых случаях, значительных изменений в черты лица и форму головы, при этом необходимо сохранить личность человека на входном фото. В проекте представлен метод перевода изображения в изображение, который напрямую кодирует реальные изображения лица в скрытое пространство предварительно обученной безусловной сети GAN (StyleGAN) с учетом заданного сдвига старения.

Описание нейронной сети

В проекте используется предварительно обученная сеть возрастной регрессии для явного указания кодировщику генерировать скрытые коды, соответствующие желаемому возрасту [75, 76]. Таким образом, демонстрируемый метод подходит к процессу непрерывного старения как к задаче регрессии между входным возрастом и желаемым целевым возрастом, обеспечивая детальный контроль над сгенерированным изображением. В отличие от других подходов, которые работают

исключительно в скрытом пространстве с использованием априорного пути, контролирующего возраст, данный метод изучает более распутанный, нелинейный путь. Так, сквозной характер этого подхода в сочетании с богатым семантическим скрытым пространством StyleGAN позволяет дальнейшее редактирование сгенерированных изображений.

SAM (Style-Based Age Manipulation) – нейросетевая модель, которая меняет возраст человека на изображении [77]. Модель принимает на вход изображение лица человека и целевой возраст. На выходе нейросеть отдает сгенерированное изображение, где лицо целевой персоны изменено в соответствии с возрастным сдвигом. Модель умеет как омолаживать лица на изображениях, так и состаривать. В качестве архитектуры используется генеративно-состязательная модель StyleGAN. Применяемая в проекте модель обучена на наборе данных FFHQ трансформации возраста, что позволяет моделировать детальную трансформацию возраста, используя одно входное изображение лица.

Алгоритм работы

Задача моделирования непрерывного процесса старения решается как задача регрессии между входным возрастом и целевым возрастом. Сеть получает входное изображение лица и желаемый целевой возраст α_t . Сначала кодер E_{age} извлекает карты объектов в трех различных пространственных масштабах. Затем 18 блоков `map2style` используются для постепенного понижения дискретизации трех карт признаков в 18 различных 512-мерных векторов стиля, тем самым кодируя входное изображение в скрытое пространство $W+$ StyleGAN. Дополнительно используется фиксированный предварительно обученный кодер `rSp` для извлечения скрытого кода $W+$ числа x , обозначаемого w^* , который затем добавляется к скрытому коду с преобразованием возраста, обозначенному

$E_{age}(x_{age})$. Затем предварительно обученный StyleGAN используется для создания желаемого изображения с возрастным преобразованием с использованием агрегированного скрытого кода. Во время обучения L_2 , L_{LPIPS} и L_{ID} обеспечивают визуальное сходство и сохранение идентичности, в то время как L_{reg} побуждает изученные скрытые коды быть ближе к среднему скрытому коду. Наконец, L_{age} помогает кодировщику генерировать желаемый скрытый код с преобразованием возраста. Во время обучения обучаются только блоки кодера E_{age} и $map2style$. Кроме того, L_{LPIPS} , L_{ID} и L_{age} вычисляются с помощью фиксированных предварительно обученных сетей (рис. 1).

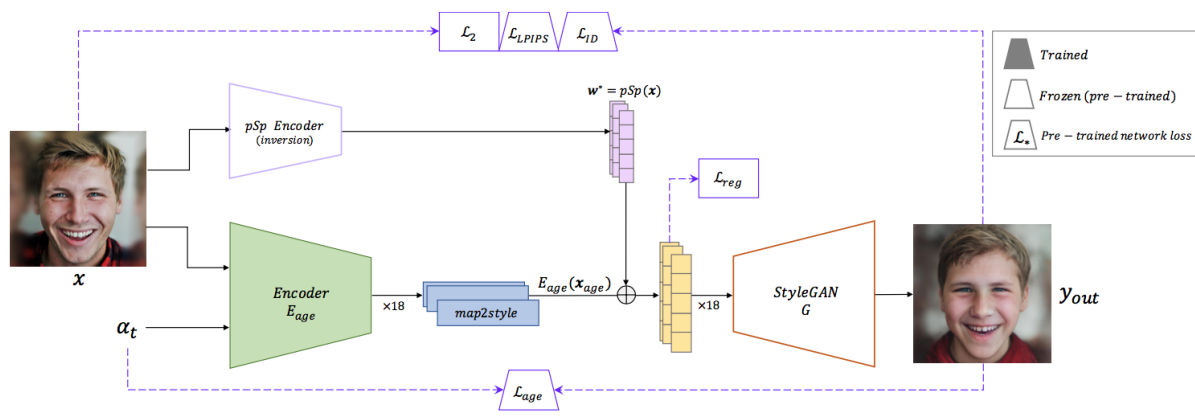


Схема модели SAM

Настройка окружения и запуск проекта

Обязательные условия программно-аппаратного обеспечения:

- Операционная система Linux;
- Аппаратное обеспечение NVIDIA GPU + CUDA CuDNN;
- Язык программирования Python 3.

Необходимо установить следующие зависимости в виртуальное окружение python:

- `_libgcc_mutex=0.1=main`
- `ca-certificates=2020.4.5.1=hecc5488_0`
- `certifi=2020.4.5.1=py36h9f0ad1d_0`
- `libedit=3.1.20181209=hc058e9b_0`
- `libffi=3.2.1=hd88cf55_4`
- `libgcc-ng=9.1.0=hdf63c60_0`

- libstdcxx-ng=9.1.0=hdf63c60_0
- ncurses=6.2=he6710b0_1
- ninja=1.10.0=hc9558a2_0
- openssl=1.1.1g=h516909a_0
- pip=20.0.2=py36_3
- python=3.6.7=h0371630_0
- python_abi=3.6=1_cp36m
- readline=7.0=h7b6447c_5
- setuptools=46.4.0=py36_0
- sqlite=3.31.1=h62c20be_1
- tk=8.6.8=hbc83047_0
- wheel=0.34.2=py36_0
- xz=5.2.5=h7b6447c_0
- zlib=1.2.11=h7b6447c_3
- pip:
 - scipy==1.4.1
 - matplotlib==3.2.1
 - tqdm==4.46.0
 - numpy==1.18.4
 - opencv-python==4.2.0.34
 - pillow==7.1.2
 - tensorboard==2.2.1
 - torch==1.6.0
 - torchvision==0.4.2

Далее необходимо загрузить в проект предварительно обученную модель sam_ffhq_aging

https://drive.google.com/file/d/1XyumF6_fdAxFmXpFcmPf-q84LU_22EMC/view?usp=sharing

Для запуска работы при использовании предварительно обученной модели SAM необходимо использовать модуль inference.py для выполнения вывода.

Пример строки запуска:

```
python scripts/inference.py \
--exp_dir=/path/to/experiment \
--checkpoint_path=experiment/checkpoints/best_model.pt \
--data_path=/path/to/test_data \
--test_batch_size=4 \
--test_workers=4 \
--couple_outputs \
--target_age=0,10,20,30,40,50,60,70,80
```

Дополнительные примечания:

- Во время вывода параметры, используемые во время обучения, загружаются из сохраненной контрольной точки, а затем обновляются с помощью параметров теста, передаваемых в сценарий вывода.

- Добавление флага `--couple_outputs` сохранит дополнительное изображение, содержащее входные и выходные изображения рядом в подкаталоге `inference_coupled`. В противном случае в подкаталоге `inference_results` сохраняется только выходное изображение.

- В приведенном выше примере запускается преобразование возраста с целевым возрастом 0,10,...,80.

- Результаты каждого целевого возраста сохраняются в подкаталогах `inference_results/TARGET_AGE` и `inference_coupled/TARGET_AGE`.

- По умолчанию изображения будут сохраняться с разрешением 1024x1024, исходным выходным размером StyleGAN.

- Если необходимо сохранить результаты с измененным размером до разрешения 256x256, можно сделать это, добавив флаг `--resize_outputs`.

Приведенный выше сценарий вывода сохранит каждый результат трансформации в отдельном подкаталоге для каждого целевого возраста. Однако иногда удобнее сохранять все результаты трансформации данного ввода рядом.

Для этого нужно использовать скрипт `inference_side_by_side.py`, который работает аналогично обычному скрипту вывода:

```
python scripts/inference_side_by_side.py \  
--exp_dir=/path/to/experiment \  
--checkpoint_path=experiment/checkpoints/best_model.pt \  
--data_path=/path/to/test_data \  
--test_batch_size=4 \  
--test_workers=4 \  
--target_age=0,10,20,30,40,50,60,70,80
```

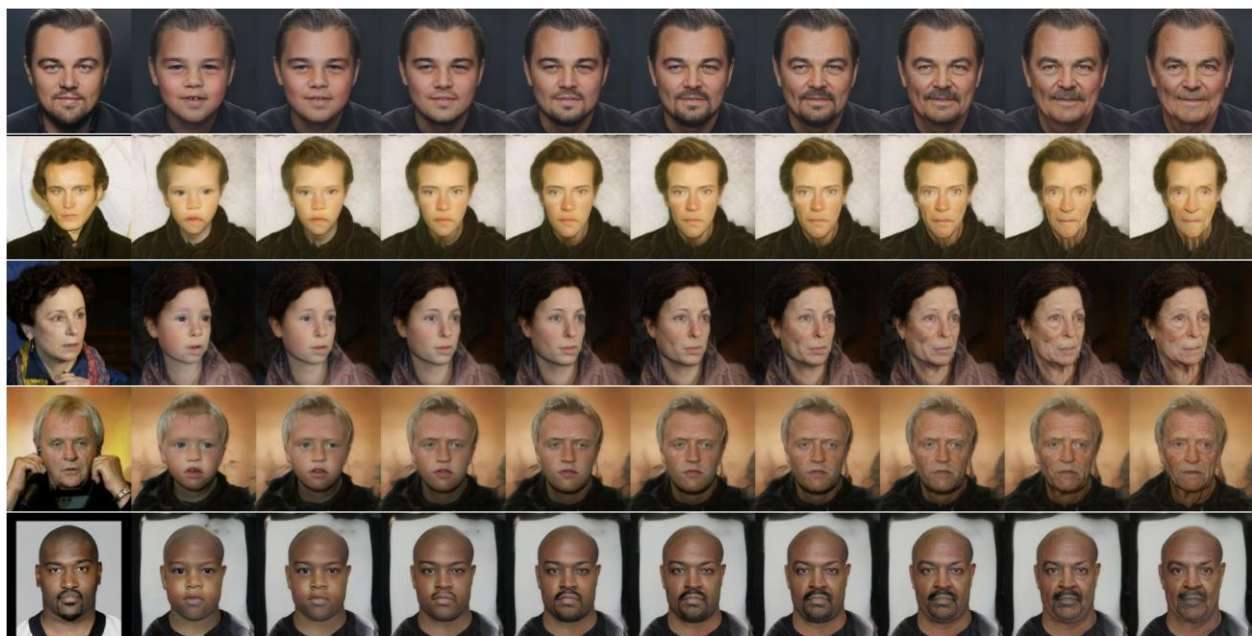
Здесь все результаты трансформации 0,10,...,80 будут сохранены рядом с исходным входным изображением.

Таблица использованных библиотек

Название библиотек	Описание библиотек
dlib	содержит алгоритмы машинного обучения и различные инструменты для работы с изображениями лиц
imageio	предоставляет простой интерфейс для чтения и записи широкого спектра данных изображений, включая анимированные изображения, видео, объемные данные и научные форматы
numpy	используется для работы с многомерными массивами
pytorch	фреймворк машинного обучения для языка Python с открытым

	исходным кодом, созданный на базе torch, который обеспечивает тензорное вычисление с ускорением GPU и высокоуровневые функции для создания сетей глубокого обучения
cog	используется для препроцессинга файлов, позволяет использовать фрагменты кода на Python для генерации желаемого текста
openssl	криптографическая библиотека, которая является open source реализацией двух протоколов: Secure Sockets Layer (SSL) и Transport Layer Security (TLS)
opencv	библиотека алгоритмов компьютерного зрения, обработки изображений и численных алгоритмов общего назначения с открытым кодом
pillow	расширение библиотеки Python Imaging Library (PIL), предоставляет функции для работы с изображениями, такие как изменение размера, поворот, наложение фильтров и многое другое

Результаты работы проекта



Результаты трансформации возраста лиц на фотографиях, полученные с использованием SAM и фиксированного предварительно обученного генератора StyleGAN

Текст программы

```

Запускаемый файл predict.py
import tempfile
from argparse import Namespace
import dlib
import imageio

```

```

import numpy as np
import torch
import torchvision.transforms as transforms
from cog import BasePredictor, Path, Input
from datasets.augmentations import AgeTransformer
from models.psp import pSp
from scripts.align_all_parallel import align_face
from utils.common import tensor2im

class Predictor(BasePredictor):
    def setup(self):
        self.transform = transforms.Compose(
            [
                transforms.Resize((256, 256)),
                transforms.ToTensor(),
                transforms.Normalize([0.5, 0.5, 0.5], [0.5, 0.5, 0.5]),
            ]
        )
        model_path = "pretrained_models/sam_ffhq_aging.pt"
        ckpt = torch.load(model_path, map_location="cpu")

        opts = ckpt["opts"]
        opts["checkpoint_path"] = model_path
        opts["device"] = "cuda" if torch.cuda.is_available() else "cpu"
        self.opts = Namespace(**opts)
    def predict(
        self,
        image: Path = Input(
            description="facial image",
        ),
        target_age: str = Input(
            description="age of the output image, when choose
'default' "
            "a gif for age from 0, 10, 20,...,to 100 will
be displayed",
        ),
    ) -> Path:
        net = pSp(self.opts)
        net.eval()
        if torch.cuda.is_available():
            net.cuda()

        # align image
        aligned_image = run_alignment(str(image))
        aligned_image.resize((256, 256))
        input_image = self.transform(aligned_image)
        if target_age == "default":
            target_ages = [0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]

```

```

        age_transformers = [AgeTransformer(target_age=age) for age in
target_ages]
    else:
        age_transformers = [AgeTransformer(target_age=target_age)]

    results = np.array(aligned_image.resize((1024, 1024)))
    all_imgs = []
    for age_transformer in age_transformers:
        print(f"Running on target age: {age_transformer.target_age}")
        with torch.no_grad():
            input_image_age =
[age_transformer(input_image.cpu()).to("cuda")]
            input_image_age = torch.stack(input_image_age)
            result_tensor = run_on_batch(input_image_age, net)[0]
            result_image = tensor2im(result_tensor)
            all_imgs.append(result_image)
            results = np.concatenate([results, result_image], axis=1)

    if target_age == "default":
        out_path = Path(tempfile.mkdtemp()) / "output.gif"
        imageio.mimwrite(str(out_path), all_imgs, duration=0.3)
    else:
        out_path = Path(tempfile.mkdtemp()) / "output.png"
        imageio.imwrite(str(out_path), all_imgs[0])
    return out_path

def run_alignment(image_path):
    predictor =
dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")
    aligned_image = align_face(filepath=image_path, predictor=predictor)
    print("Aligned image has shape: {}".format(aligned_image.size))
    return aligned_image

def run_on_batch(inputs, net):
    result_batch = net(inputs.to("cuda").float(), randomize_noise=False,
resize=False)
    return result_batch

```

Глава 6. Примеры telegram-ботов с использованием нейронных сетей

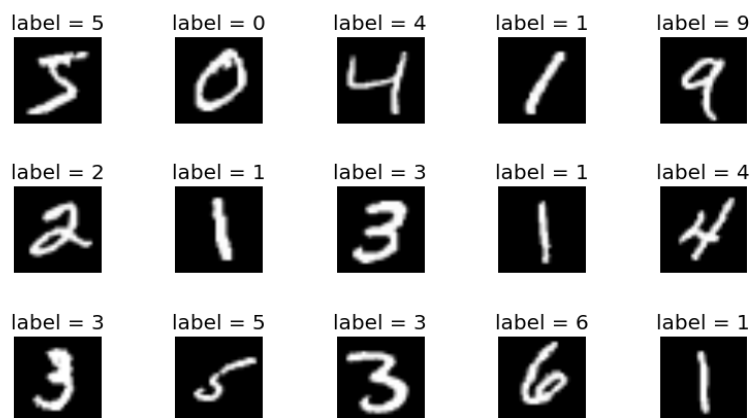
6.1. Распознавание рукописных цифр на фото. TensorFlow

Telegram-bot получает на вход .jpg изображение рукописной цифры от 0 до 9 и делает предположение о том, какая цифра написана на фото.

Использованные библиотеки [78-80]:

- telebot – библиотека для создания и конфигурирования Telegram ботов в языке Python
- tensorflow, keras – для создания, обучения и экспорта моделей
- os – для взаимодействия с системой и сохранения изображений
- PIL – для сжатия изображений

Бот создан на языке Python с использованием библиотеки telebot. Применено машинное обучение для того, чтобы создать модель, которая и будет распознавать рукописные цифры. Для этой цели использованы библиотеки tensorflow и keras. Для обучения модели использовалась база рукописных цифр mnist, в которой хранится множество цифр и подписи к каждой картинке, для того, чтобы модель понимала, правильные ли предположения она делает.



Пример данных из базы MNIST

Алгоритм работы программы

1. Пользователь отправляет команду /recognize, после которой бот ждет фото цифры
2. Пользователь отправляет фото цифры, которую он бы хотел распознать
3. Изображение сохраняется и сжимается до размера 28 на 28 пикселей
4. Сжатое изображение передается в модель, и она возвращает массив с вероятностями того, что это цифра 0,1,2,...,9
5. Из этих вероятностей выбирается максимальная, и число с максимальной вероятностью и становится ответом бота
6. Бот отправляет пользователю свое предположение

Описание программы

Файл learn.py - отвечает за обучение модели и экспорт её в отдельный файл.

1. Импортируется tensorflow
2. Из базы mnist загружается дата как тренировочных изображений, так и изображений для тестирования
3. Создается последовательная модель, в которой задаются уровни, количество узлов на каждом уровне, функции активации, входные данные и так далее
4. Модель компилируется
5. Модель обучается и тестируется на заданном количестве эпох
6. Модель экспортируется в отдельный файл

```
import tensorflow

(x_train, y_train), (x_test, y_test) =
tensorflow.keras.datasets.mnist.load_data()
x_train = x_train.reshape((60000, 28, 28, 1)) / 255.0
x_test = x_test.reshape((10000, 28, 28, 1)) / 255.0

model = tensorflow.keras.Sequential([
    tensorflow.keras.layers.Conv2D(32, (3, 3), activation='relu',
input_shape=(28, 28, 1)),
    tensorflow.keras.layers.MaxPooling2D((2, 2)),
    tensorflow.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tensorflow.keras.layers.MaxPooling2D((2, 2)),
    tensorflow.keras.layers.Flatten(),
    tensorflow.keras.layers.Dense(64, activation='relu'),
```

```

tensorflow.keras.layers.Dense(10, activation='softmax')
])
model.compile(optimizer='adam',
loss='sparse_categorical_crossentropy', metrics=['accuracy'])
model.fit(x_train, y_train, epochs=5, validation_data=(x_test,
y_test))
model.save('model.h5')

```

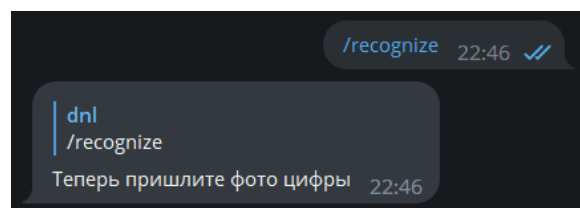
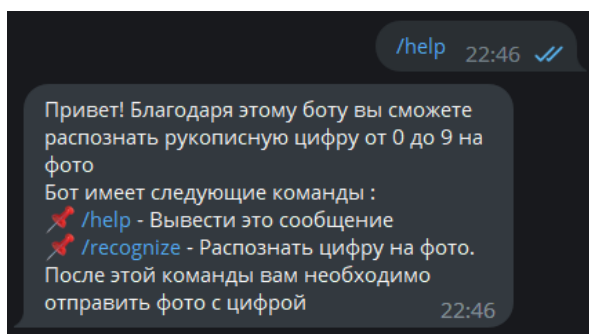
Файл bot_api.py - отвечает за непосредственный функционал Telegram бота

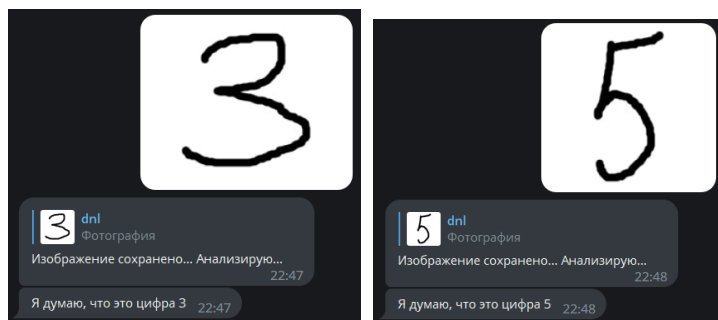
1. Внедряется токен бота и создается объект бота с этим токеном
2. Загружается модель
3. Добавляется команда /help и функция-обработчик для неё
4. Добавляется команда /recognize и функция-обработчик для неё
5. Добавляется функция для обработки фотографий, присланных пользователем
6. Добавляется функция определения цифры на фотографии пользователя

Описание функций

Название функции	Описание функции
help_command	Вывод сообщения о том, как пользоваться ботом. Вызывается при отправке /help
recognize_command	Пишет о том, чтобы пользователь отправил фото. Вызывается при отправке /recognize
file_handle	Получение файла, проверка на то, что это именно .jpg файл, сохранение файла. Вызывается при отправке файла, если до этого была вызвана команда /recognize. Вызывает функцию predict_digitm, после чего пишет ответ обратно пользователю
predict_digit	Сжатие скачанного файла, передача его в модель и получение ответа

Скриншоты работа бота





Листинг программы

```

import telebot
import os
import tensorflow
import numpy
from PIL import Image, ImageOps

BOT_HELP_MESSAGE = """
Привет! Благодаря этому боту вы сможете распознать рукописную цифру от 0
до 9 на фото
Бот имеет следующие команды :
📄 /help - Вывести это сообщение
📄 /recognize - Распознать цифру на фото. После этой команды вам
необходимо отправить фото с цифрой
"""

BOT_TOKEN = "BOT_TOKEN"
bot = telebot.TeleBot(BOT_TOKEN)

current_dir = os.path.dirname(os.path.abspath(__file__))
user_state = {}

model = tensorflow.keras.models.load_model('model.h5')

@bot.message_handler(commands=["help"])
def help_command(message: telebot.types.Message):
    bot.send_message(message.chat.id, BOT_HELP_MESSAGE)
    user_state[message.chat.id] = "Help command received"

@bot.message_handler(commands=["recognize"])
def recognize_command(message: telebot.types.Message):
    bot.reply_to(message, "Теперь пришлите фото цифры")
    user_state[message.chat.id] = "Waiting for the photo"

def predict_digit(message: telebot.types.Message):
    image = Image.open("canvas.ps").convert("L")
    image = image.resize((28, 28))
    image = ImageOps.invert(image)
    image_array = numpy.array(image) / 255.0
    image_array = image_array.reshape(1, 28, 28, 1)

```

```

prediction = model.predict(image_array)[0]
digit = numpy.argmax(prediction)
bot.send_message(message.chat.id, f"Я думаю, что это цифра {digit}")

@bot.message_handler(content_types=["photo"])
def file_handle(message: telebot.types.Message):
    if user_state[message.chat.id] == "Waiting for the photo" and
message.photo:
        file_id = message.photo[-1].file_id
        file_info = bot.get_file(file_id)
        downloaded_file = bot.download_file(file_info.file_path)
        file_extension = file_info.file_path.split('.')[-1]
        if file_extension.lower() in ["jpg", "jpeg"]:
            user_state[message.chat.id] = "Photo was uploaded"
            file_path = os.path.join(current_dir, 'canvas.ps')
            with open(file_path, 'wb') as file:
                file.write(downloaded_file)
            bot.reply_to(message, "Изображение сохранено...
Анализирую...")
            predict_digit(message)
        else:
            bot.send_message(message.chat.id, "Некорректный формат файла!
Нужно .jpg или .jpeg фото!")
    else:
        bot.send_message(message.chat.id, "Вы не прописали /recognize или
прислали не фото!")
bot.polling()

```

6.2. Детектирование дорожных знаков. Keras

Описание использованных технологий

- telebot - это бот-фреймворк для Telegram Bot API.
- PIL - библиотека для открытия, работы и сохранения различных форматов изображений
- numpy - библиотека языка программирования Python, добавляющая поддержку больших многомерных массивов и матриц, а также большой набор высокоуровневых математических функций для работы с этими массивами
- keras - фреймворк для реализации и использования моделей и непосредственно обучения

- `tempfile` - это модуль Python, используемый для работы с временными файлами.
- `cv2` – библиотека OpenCV

Разработка велась в PyCharm. Обучение происходило с использованием AMD Ryzen 5 3600.

Алгоритм работы

Прежде чем начинать разработку, бота необходимо зарегистрировать и получить его уникальный `id`, являющийся одновременно и токеном. Для этого в Telegram существует специальный бот — `@BotFather`.

Пишем ему `/start` и получаем список всех его команд.

Первая и главная — `/newbot` — отправляем ему и бот просит придумать `name` и `username` новому боту. Единственное ограничение на `name` — оно должно оканчиваться на «`_bot`». В случае успеха `BotFather` возвращает токен бота и ссылку для быстрого добавления бота в контакты, иначе придется поломать голову над именем.

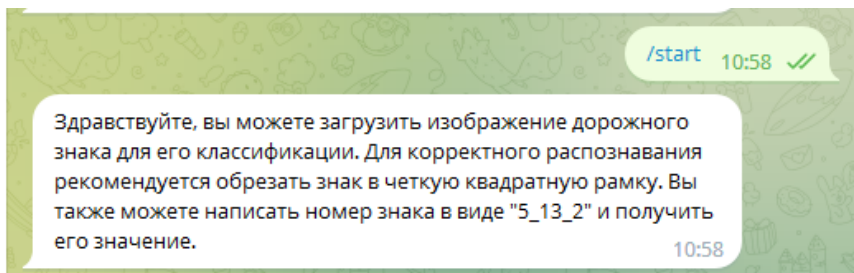
Необходимо вставить ваш токен бота в качестве значения переменной `token` в `TraficBot.py`

При запуске вашего кода переходим в телеграмм бот с названием `Road_Sign_bot` (бот может работать только тогда, когда исходный код бота запущен)

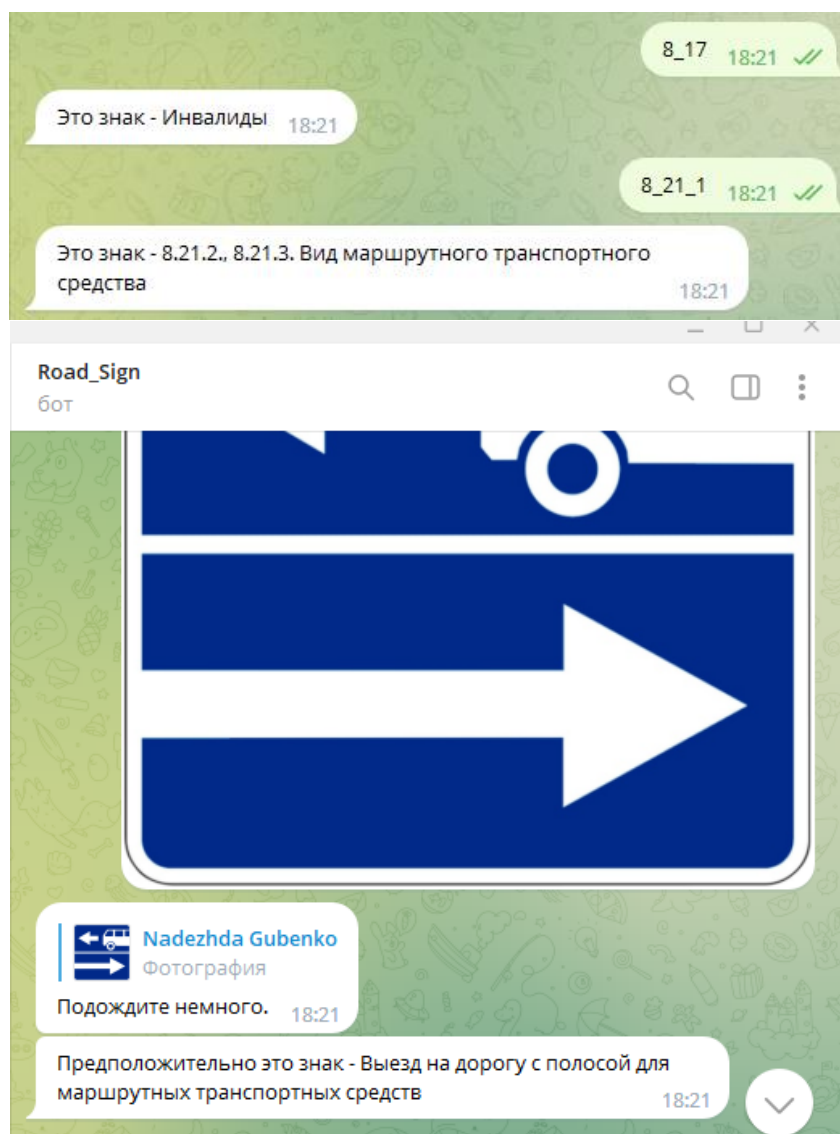
Алгоритм может распознавать только 182 набора знаков благодаря OpenCV и уже готовой обученной модели на `keras`: они перечислены в файле `TraficBot.py` в качестве индивидуальных кодов знаков.

После запуска бота с помощью `start.bat` есть возможность отправить сообщения. Бот реагирует только на команды «`/start`» а также на сообщения (фото и код знака) Все остальные сообщения ботом будут проигнорированы.

По команде «/start» бот просит пользователя отправить изображение для обработки.



Вы можете отправить либо изображение знака, либо код знака.



Изображение можно отправлять любого размера, поскольку бот обрезает исходное изображение до нужного параметра. Но чем меньше изображение, тем лучше точность.

При некорректном вводе данных выводится сообщение, предупреждающее о неправильном вводе данных.



Текст программы

Главный скрипт

```
import classifier as classifier
import cv2
token = '5910930270:AAGEAH261E-w80CvtGN6aiLBVbqVKjyiMok'
dct_1 = {'0': '1_1',
'1': '1_11_1',
'10': '1_18',
'100': '4_5_5',
'101': '4_5_6',
'102': '4_8_1',
'103': '4_8_2',
'104': '4_8_3',
'105': '4_8_4',
'106': '4_8_5',
'107': '4_8_6',
'108': '5_1',
'109': '5_11_1',
'11': '1_19',
'110': '5_12_1',
'111': '5_13_1',
'112': '5_13_2',
'113': '5_14',
'114': '5_14_1',
'115': '5_14_2',
'116': '5_14_3',
'117': '5_15_2+1',
'118': '5_15_2+2',
'119': '5_15_2+3',
'12': '1_2',
'120': '5_15_2+5',
'121': '5_15_3',
'122': '5_15_5',
'123': '5_16',
'124': '5_17',
'125': '5_18',
'126': '5_19',
'127': '5_2',
'128': '5_20',
```

'129': '5_21',
'13': '1_20_1',
'130': '5_22',
'131': '5_23_2',
'132': '5_24_2',
'133': '5_29',
'134': '5_3',
'135': '5_31+10',
'136': '5_31+20',
'137': '5_31+30',
'138': '5_31+40',
'139': '5_31+5',
'14': '1_20_2',
'140': '5_31+50',
'141': '5_32',
'142': '5_33',
'143': '5_4',
'144': '5_5',
'145': '5_6',
'146': '5_7_1',
'147': '5_7_2',
'148': '5_8',
'149': '5_9',
'15': '1_20_3',
'150': '6_15_1',
'151': '6_15_2',
'152': '6_15_3',
'153': '6_16',
'154': '6_2+30',
'155': '6_2+40',
'156': '6_2+50',
'157': '6_2+60',
'158': '6_2+70',
'159': '6_2+80',
'16': '1_21',
'160': '6_3_1',
'161': '6_4',
'162': '6_6',
'163': '6_7',
'164': '6_8_1',
'165': '6_8_2',
'166': '6_8_3',
'167': '7_1',
'168': '7_11',
'169': '7_12',
'17': '1_22',
'170': '7_13',
'171': '7_14',
'172': '7_18',

'173': '7_2',
'174': '7_3',
'175': '7_4',
'176': '7_5',
'177': '7_6',
'178': '7_7',
'179': '7_9',
'18': '1_23',
'180': 'smoke',
'181': 'unknown',
'19': '1_24',
'2': '1_11_2',
'20': '1_25',
'21': '1_26',
'22': '1_27',
'23': '1_28',
'24': '1_29',
'25': '1_30',
'26': '1_31',
'27': '1_33',
'28': '1_5',
'29': '1_6',
'3': '1_12_1',
'30': '1_7',
'31': '1_8',
'32': '2_1',
'33': '2_2',
'34': '2_3_1',
'35': '2_3_L',
'36': '2_3_R',
'37': '2_4',
'38': '2_5',
'39': '2_6',
'4': '1_12_2',
'40': '2_7',
'41': '3_1',
'42': '3_10',
'43': '3_15',
'44': '3_16',
'45': '3_17',
'46': '3_18_1',
'47': '3_18_2',
'48': '3_19',
'49': '3_2',
'5': '1_13',
'50': '3_20',
'51': '3_21',
'52': '3_24+10',
'53': '3_24+110',

'54': '3_24+20',
'55': '3_24+30',
'56': '3_24+40',
'57': '3_24+5',
'58': '3_24+50',
'59': '3_24+60',
'6': '1_14',
'60': '3_24+70',
'61': '3_24+80',
'62': '3_24+90',
'63': '3_25+20',
'64': '3_25+30',
'65': '3_25+40',
'66': '3_25+5',
'67': '3_25+50',
'68': '3_25+60',
'69': '3_25+70',
'7': '1_15',
'70': '3_26',
'71': '3_27',
'72': '3_28',
'73': '3_29',
'74': '3_30',
'75': '3_31',
'76': '3_32',
'77': '3_33',
'78': '3_5',
'79': '3_6',
'8': '1_16',
'80': '3_7',
'81': '3_8',
'82': '3_9',
'83': '4_1_1',
'84': '4_1_2',
'85': '4_1_2_1',
'86': '4_1_3',
'87': '4_1_3_1',
'88': '4_1_4',
'89': '4_1_5',
'9': '1_17',
'90': '4_1_6',
'91': '4_2_1',
'92': '4_2_2',
'93': '4_2_3',
'94': '4_3',
'95': '4_4_1',
'96': '4_4_2',
'97': '4_5_1',
'98': '4_5_2',

'99': '4_5_4'}

dct_2 = {'1_1': 'Железнодорожный переезд со шлагбаумом',
'1_10': 'Выезд на набережную',
'1_11_1': 'Опасный поворот (правый)',
'1_11_2': 'Опасный поворот (левый)',
'1_12_1': 'Опасные повороты (с первым поворотом направо)',
'1_12_2': 'Опасные повороты (с первым поворотом налево)',
'1_13': 'Крутой спуск',
'1_14': 'Крутой подъем',
'1_15': 'Скользкая дорога',
'1_16': 'Неровная дорога',
'1_17': 'Искусственная неровность',
'1_18': 'Выброс гравия',
'1_19': 'Опасная обочина',
'1_2': 'Железнодорожный переезд без шлагбаума',
'1_20_1': 'Сужение дороги. ',
'1_20_2': 'Сужение дороги ',
'1_20_3': 'Сужение дороги',
'1_21': 'Двустороннее движение',
'1_22': 'Пешеходный переход',
'1_23': 'Дети',
'1_24': 'Пересечение с велосипедной дорожкой или велопешеходной дорожкой',
'1_25': 'Дорожные работы',
'1_26': 'Перегон скота',
'1_27': 'Дикие животные',
'1_28': 'Падение камней',
'1_29': 'Боковой ветер',
'1_30': 'Низколетающие самолеты',
'1_31': 'Тоннель',
'1_32': 'Затор',
'1_33': 'Прочие опасности',
'1_34_1': 'Направление поворота',
'1_34_2': 'Направление поворота',
'1_34_3': 'Направление поворота',
'1_35': 'Участок перекрестка',
'1_3_1': 'Однопутная железная дорога',
'1_3_2': 'Многопутная железная дорога',
'1_4': 'Приближение к железнодорожному переезду',
'1_5': 'Пересечение с трамвайной линией',
'1_6': 'Пересечение равнозначных дорог',
'1_7': 'Пересечение с круговым движением',
'1_8': 'Светофорное регулирование',
'1_9': 'Разводной мост',
'2_1': 'Главная дорога',
'2_2': 'Конец главной дороги',
'2_3_1': 'Пересечение со второстепенной дорогой',
'2_3_2': '2.3.3, 2.3.4, 2.3.5, 2.3.6, 2.3.7. Примыкание второстепенной

дороги',
'2_4': 'Уступите дорогу',
'2_5': 'Движение без остановки запрещено',
'2_6': 'Преимущество встречного движения',
'2_7': 'Преимущество перед встречным движением',
'3_1': 'Въезд запрещен',
'3_10': 'Движение пешеходов запрещено',
'3_11': 'Ограничение массы',
'3_12': 'Ограничение массы, приходящейся на ось транспортного средства',
'3_13': 'Ограничение высоты',
'3_14': 'Ограничение ширины',
'3_15': 'Ограничение длины',
'3_16': 'Ограничение минимальной дистанции',
'3_17_1': 'Таможня',
'3_17_2': 'Опасность',
'3_17_3': 'Контроль',
'3_18_1': 'Поворот направо запрещен',
'3_18_2': 'Поворот налево запрещен',
'3_19': 'Разворот запрещен',
'3_2': 'Движение запрещено',
'3_20': 'Обгон запрещен',
'3_21': 'Конец зоны запрещения обгона',
'3_22': 'Обгон грузовым автомобилем запрещен',
'3_23': 'Конец зоны запрещения обгона грузовым автомобилям',
'3_24': 'Ограничение максимальной скорости',
'3_25': 'Конец зоны ограничения максимальной скорости',
'3_26': 'Подача звукового сигнала запрещена',
'3_27': 'Остановка запрещена',
'3_28': 'Стоянка запрещена',
'3_29': 'Стоянка запрещена по нечетным числам месяца',
'3_3': 'Движение механических транспортных средств запрещено',
'3_30': 'Стоянка запрещена по четным числам месяца',
'3_31': 'Конец зоны всех ограничений',
'3_32': 'Движение транспортных средств с опасными грузами запрещено',
'3_33': 'Движение транспортных средств с взрывчатыми и легковоспламеняющимися грузами запрещено',
'3_4': 'Движение грузовых автомобилей запрещено',
'3_5': 'Движение мотоциклов запрещено',
'3_6': 'Движение тракторов запрещено',
'3_7': 'Движение с прицепом запрещено',
'3_8': 'Движение гужевых повозок запрещено',
'3_9': 'Движение на велосипедах запрещено',
'4_1_1': 'Движение прямо',
'4_1_2': 'Движение направо',
'4_1_3': 'Движение налево',
'4_1_4': 'Движение прямо или направо',
'4_1_5': 'Движение прямо или налево',
'4_1_6': 'Движение направо или налево',
'4_2_1': 'Объезд препятствия справа',

'4_2_2': 'Объезд препятствия слева',
'4_2_3': 'Объезд препятствия справа или слева',
'4_3': 'Круговое движение',
'4_4_1': 'Велосипедная дорожка',
'4_4_2': 'Конец велосипедной дорожки',
'4_5_1': 'Пешеходная дорожка',
'4_5_2': 'Пешеходная и велосипедная дорожка с совмещенным движением (велопешеходная дорожка с совмещенным движением)',
'4_5_3': 'Конец пешеходной и велосипедной дорожки с совмещенным движением (конец велопешеходной дорожки с совмещенным движением)',
'4_5_4': 'Пешеходная и велосипедная дорожка с разделением движения',
'4_5_5': 'Пешеходная и велосипедная дорожка с разделением движения',
'4_5_6': 'Конец пешеходной и велосипедной дорожки с разделением движения (конец велопешеходной дорожки с разделением движения)',
'4_5_7': 'Конец пешеходной и велосипедной дорожки с разделением движения (конец велопешеходной дорожки с разделением движения)',
'4_6': 'Ограничение минимальной скорости',
'4_7': 'Конец зоны ограничения минимальной скорости',
'4_8_1': 'Направление движения транспортных средств с опасными грузами',
'5_1': 'Автомагистраль',
'5_10': 'Выезд на дорогу с реверсивным движением',
'5_11_1': 'Дорога с полосой для маршрутных транспортных средств',
'5_11_2': 'Дорога с полосой для велосипедистов',
'5_12_1': 'Конец дороги с полосой для маршрутных транспортных средств',
'5_12_2': 'Конец дороги с полосой для велосипедистов',
'5_13_1': 'Выезд на дорогу с полосой для маршрутных транспортных средств',
'5_13_2': 'Выезд на дорогу с полосой для маршрутных транспортных средств',
'5_13_3': 'Выезд на дорогу с полосой для велосипедистов',
'5_13_4': 'Выезд на дорогу с полосой для велосипедистов',
'5_14': 'Полоса для велосипедистов 5.14.3. Конец полосы для велосипедистов',
'5_14_1': 'Конец полосы для маршрутных транспортных средств',
'5_15': 'Конец полосы',
'5_15_1': 'Направления движения по полосам',
'5_15_2': 'Направления движения по полосам',
'5_15_3': 'Начало полосы',
'5_15_4': 'Начало полосы',
'5_15_5': 'Конец полосы',
'5_15_7': 'Направление движения по полосам',
'5_15_8': 'Число полос',
'5_16': 'Место остановки автобуса и (или) троллейбуса',
'5_17': 'Место остановки трамвая',
'5_18': 'Место стоянки легковых такси',
'5_19': 'Пешеходный переход',
'5_2': 'Конец автомагистрали',
'5_20': 'Искусственная неровность',
'5_21': 'Жилая зона',

'5_22': 'Конец жилой зоны',
'5_23_2': 'Начало населенного пункта',
'5_24_1': 'Конец населенного пункта',
'5_25': 'Начало населенного пункта',
'5_26': 'Конец населенного пункта',
'5_27': 'Зона с ограничением стоянки',
'5_28': 'Конец зоны с ограничением стоянки',
'5_29': 'Зона регулируемой стоянки',
'5_3': 'Дорога для автомобилей',
'5_30': 'Конец зоны регулируемой стоянки',
'5_31': 'Зона с ограничением максимальной скорости',
'5_32': 'Конец зоны с ограничением максимальной скорости',
'5_33': 'Пешеходная зона',
'5_33_1': 'Велосипедная зона',
'5_34': 'Конец пешеходной зоны',
'5_34_1': 'Конец велосипедной зоны',
'5_35': 'Зона с ограничением экологического класса механических транспортных средств',
'5_36': 'Зона с ограничением экологического класса грузовых автомобилей',
'5_37': 'Конец зоны с ограничением экологического класса механических транспортных средств',
'5_38': 'Конец зоны с ограничением экологического класса грузовых автомобилей',
'5_4': 'Конец дороги для автомобилей',
'5_5': 'Дорога с односторонним движением',
'5_6': 'Конец дороги с односторонним движением',
'5_7_1': 'Выезд на дорогу с односторонним движением.',
'5_7_2': 'Выезд на дорогу с односторонним движением.',
'5_8': 'Реверсивное движение',
'5_9': 'Конец реверсивного движения',
'6_1': 'Общие ограничения максимальной скорости',
'6_10_1': 'Указатель направлений',
'6_10_2': 'Указатель направлений',
'6_11': 'Наименование объекта',
'6_12': 'Указатель расстояний',
'6_13': 'Километровый знак',
'6_14_1': 'Номер маршрута',
'6_14_2': 'Номер маршрута',
'6_15_1': 'Направление движения для грузовых автомобилей',
'6_15_2': 'Направление движения для грузовых автомобилей',
'6_15_3': 'Направление движения для грузовых автомобилей',
'6_16': 'Стоп-линия',
'6_17': 'Схема объезда',
'6_18_1': 'Направление объезда',
'6_18_2': 'Направление объезда',
'6_18_3': 'Направление объезда',
'6_19_1': '6.19.2. Предварительный указатель перестроения на другую проезжую часть',

'6_2': 'Рекомендуемая скорость',
'6_20_1': '6.20.2. Аварийный выход',
'6_21': 'Направление движения к аварийному выходу',
'6_22': 'Фотовидеофиксация',
'6_3_1': 'Место для разворота',
'6_3_2': 'Зона для разворота',
'6_4': 'Парковка (парковочное место)',
'6_5': 'Полоса аварийной остановки',
'6_6': 'Подземный пешеходный переход',
'6_7': 'Надземный пешеходный переход',
'6_8_1': 'Тупик',
'6_8_2': 'Тупик',
'6_8_3': 'Тупик',
'6_9_1': 'Предварительный указатель направлений',
'6_9_2': 'Предварительный указатель направления',
'6_9_3': 'Схема движения',
'7_1': 'Пункт медицинской помощи',
'7_10': 'Кемпинг',
'7_11': 'Место отдыха',
'7_12': 'Пост дорожно-патрульной службы',
'7_13': 'Полиция',
'7_14': 'Пункт транспортного контроля',
'7_14_1': 'Пункт таможенного контроля',
'7_15': 'Зона приема радиостанции, передающей информацию о дорожном движении',
'7_16': 'Зона радиосвязи с аварийными службами',
'7_17': 'Бассейн или пляж',
'7_18': 'Туалет',
'7_19': 'Телефон экстренной связи',
'7_2': 'Больница',
'7_20': 'Огнетушитель',
'7_21': 'Автозаправочная станция с возможностью зарядки электромобилей',
'7_3': 'Автозаправочная станция',
'7_4': 'Техническое обслуживание автомобилей',
'7_5': 'Мойка автомобилей',
'7_6': 'Телефон',
'7_7': 'Пункт питания',
'7_8': 'Питьевая вода',
'7_9': 'Гостиница или мотель',
'8_10': 'Место для осмотра автомобилей',
'8_11': 'Ограничение разрешенной максимальной массы',
'8_12': 'Опасная обочина',
'8_13': 'Направление главной дороги',
'8_14': 'Полоса движения',
'8_15': 'Слепые пешеходы',
'8_16': 'Влажное покрытие',
'8_17': 'Инвалиды',
'8_18': 'Кроме инвалидов',
'8_19': 'Класс опасного груза',

```

'8_1_1': ' Расстояние до объекта',
'8_1_2': 'Расстояние до объекта',
'8_1_3': '8.1.4. Расстояние до объекта',
'8_20_1': '8.20.2. Тип тележки транспортного средства',
'8_21_1': '8.21.2., 8.21.3. Вид маршрутного транспортного средства',
'8_22_1': '8.22.2., 8.22.3. Препятствие',
'8_23': 'Фотовидеофиксация',
'8_24': 'Работает эвакуатор',
'8_25': 'Экологический класс транспортного средства',
'8_2_1': '8.2.2., 8.2.3., 8.2.4., 8.2.5., 8.2.6. Зона действия',
'8_3_1': '8.3.2., 8.3.3. Направление действия',
'8_4_1': '8.4.2., 8.4.3., 8.4.3.1., 8.4.4., 8.4.5., 8.4.6., 8.4.7.,
8.4.8. Вид транспортного средства',
'8_4_9': '8.4.10., 8.4.11., 8.4.12., 8.4.13., 8.4.14., 8.4.15. Кроме
вида транспортных средства',
'8_5_1': 'Субботные, воскресные и праздничные дни',
'8_5_2': 'Рабочие дни',
'8_5_3': 'Дни недели',
'8_5_4': 'Время действия',
'8_5_5': '8.5.6., 8.5.7. Время действия',
'8_6_1': '8.6.2., 8.6.3., 8.6.4., 8.6.5., 8.6.6., 8.6.7., 8.6.8., 8.6.9.
Способ постановки транспортного средства на стоянку',
'8_7': 'Стоянка с неработающим двигателем',
'8_8': 'Платные услуги',
'8_9': 'Ограничение продолжительности стоянки',
'8_9_1': 'Стоянка только для владельцев парковочных разрешений',
'8_9_2': 'Стоянка только транспортных средств дипломатического корпуса'}

```

```

import telebot
from PIL import Image
import numpy as np
from keras.models import load_model
import tempfile

def detect(image):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    faces = classifier.detectMultiScale(gray, 1.3, 5)
    count = 0
    for (x, y, w, h) in faces:
        count = count + 1
        cv2.rectangle(image, (x, y), (x + w, y + h), (0, 255, 0), 5)
    return image, count

bot = telebot.TeleBot(token)

@bot.message_handler(content_types=["text"])
def start(message):
    if message.text=='/start':

```

```

        bot.send_message(message.chat.id, 'Здравствуйтесь, вы можете
загрузить изображение дорожного знака для его классификации. Для
корректного распознавания рекомендуется обрезать знак в четкую квадратную
рамку. Вы также можете написать номер знака в виде "5_13_2" и получить
его значение.')
    else:
        try:
            bot.send_message(message.chat.id, 'Это знак - ' +
dct_2[message.text])
        except:
            bot.send_message(message.chat.id, 'Извините, этого знака нет
в базе.')

@bot.message_handler(content_types=['photo'])
def handle_docs_photo(message):
    bot.reply_to(message, "Подождите немного.")
    file_info = bot.get_file(message.photo[len(message.photo) -
1].file_id)
    downloaded_file = bot.download_file(file_info.file_path)
    #f = tempfile.NamedTemporaryFile(delete=True).name + ".png"
    # bot.sendMessage(chat_id, "Retrieving %s" % path)
    src = 'file.png'
    with open(src, 'wb') as new_file:
        new_file.write(downloaded_file)

    img = Image.open('file.png')
    img = img.resize((40,40))
    img = np.array(img)
    img = np.expand_dims(img,0)

    model =
load_model('D:/github/pythonProjects/RoadSign/TrafficSignRecognition.h5')
    res = np.argmax(model(img))

    res = dct_1[str(res)]
    try:
        res = dct_2[str(res)]
        bot.send_message(message.chat.id, 'Предположительно это знак - '
+ res)
    except:
        bot.send_message(message.chat.id, 'Предположительно это знак - '
+ res)

if __name__ == '__main__':
    bot.infinity_polling()

```

Обрабатывающий детектор через OpenCV

```

target = cv2.imread('transition.jpg')
target = cv2.resize(target, (64, 64))

```

```

target = cv2.inRange(target, (56, 128, 188), (255, 255, 255))
cv2.imshow("Target image", target)
videoCapture = cv2.VideoCapture(0)
while (True):
    _,frame = videoCapture.read()
    cv2.imshow("Original Image", frame)
    frameForCutting = frame.copy()
    hsvFrame = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    hsvFrame = cv2.blur(hsvFrame, (5, 5))
    frameMask = cv2.inRange(hsvFrame, (56, 128, 188), (255, 255, 255))
    cv2.imshow("Frame mask", frameMask)
    frameMask = cv2.erode(frameMask, None, iterations=2)
    frameMask = cv2.dilate(frameMask, None, iterations=4)
    cv2.imshow("Smooth frame mask", frameMask)
    targetContours = cv2.findContours(frameMask, cv2.RETR_TREE,
cv2.CHAIN_APPROX_NONE)
    targetContours = targetContours[0]
    if targetContours:
        targetContours = sorted(targetContours, key=cv2.contourArea,
reverse=True)
        cv2.drawContours(frame, targetContours, 0, (0,0,255), 3)
        cv2.imshow("Contours of target", frame)
        (x,y, w, h) = cv2.boundingRect(targetContours[0])
        cv2.rectangle(frame, (x,y), (x+w, y+h), (0,255,0), 2)
        cv2.imshow("Rect on frame", frame)
        cuttingFrame = frameForCutting[y:y+h, x:x+w]
        cuttingFrame = cv2.resize(cuttingFrame, (64, 64))
        cuttingFrame = cv2.inRange(cuttingFrame, (56, 128, 188), (255, 255,
255))
        cv2.imshow("Current image", cuttingFrame)
        transitionCoincidence = 0
        for h in range(64):
            for w in range(64):
                if cuttingFrame[h][w] == target[h][w]:
                    transitionCoincidence += 1
        if transitionCoincidence > 3100:
            print("Transition! Count of Coincidence ", transitionCoincidence)
        else:
            print("Nothing! Count of Coincidence ", transitionCoincidence)
        if cv2.waitKey(1) == ord("q"):
            break
videoCapture.release()
cv2.destroyAllWindows()

```

6.3. Распознавание музыкальных треков в голосовых сообщениях.

Yandex SpeechKit

Нижеприведенный код представляет собой реализацию телеграм-бота, способного распознавать музыкальные треки в голосовых сообщениях. Основная задача состоит в обработке голосовых сообщений, содержащих песни в формате ogg opus, с использованием сервиса распознавания речи Yandex SpeechKit.

При получении голосового сообщения бот отправляет его аудио данные на распознавание, используя WebSocket протокол для связи с сервисом. Затем происходит обработка распознанного текста и определение идентификатора соответствующего трека, если распознавание было успешным.

Если трек был найден, бот отправляет ссылку на трек в сервисе Yandex.Music в ответном сообщении пользователю. В случае, если распознавание трека не удалось, бот просит пользователя повторно отправить голосовое сообщение с песней для повторной попытки распознавания.

Таким образом, основная задача - обработка голосовых сообщений с песнями, распознавание треков и предоставление пользователю ссылок на найденные треки с помощью сервиса Yandex Music.

Алгоритм

1. Импортируются необходимые модули: `uuid`, `lomond` (для работы с WebSocket) и `telebot` (для создания телеграм-бота).
2. Определена функция `get_auth_data()`, которая возвращает данные для аутентификации при подключении к WebSocket.

3. Определена функция `get_asr_data(lang)`, которая возвращает данные для распознавания речи (ASR) в аудиофайле. Я добавил параметр `lang`, чтобы указать язык для распознавания.

4. Создан класс `Recognition`, представляющий функциональность распознавания треков. В конструкторе класса инициализируется `WebSocket` и преобразуются бинарные данные аудиофайла.

5. Метод `get_track_id()` класса `Recognition` отвечает за подключение к `WebSocket`, отправку данных аутентификации и данных ASR, а также отправку бинарных частей аудиофайла. Он ожидает получение ответа от `WebSocket` и проверяет его наличие трека или ошибки.

6. Методы `is_valid_response(response)` и `is_fatal_error(response)` класса `Recognition` проверяют, является ли ответ от `WebSocket` успешным или содержит фатальную ошибку.

7. Метод `parse_track_id(response)` класса `Recognition` извлекает идентификатор трека из ответа `WebSocket`.

8. Метод `get_chunks_and_replace_encoder(binary_data)` класса `Recognition` преобразует бинарные данные аудиофайла, добавляет заголовки и заменяет кодировщик на `SpeechKit Mobile SDK`.

9. Создан объект `bot` класса `telebot.TeleBot` с указанием токена вашего бота.

10. Определены два обработчика сообщений для команды `/start` и для голосовых сообщений. При команде `/start` бот отправляет приветственное сообщение, а при получении голосового сообщения бот скачивает файл, создает объект `Recognition`, вызывает метод `get_track_id()` и отправляет пользователю ссылку на трек, если он был распознан, или просит отправить голосовое сообщение еще раз, если трек не был распознан.

11. Запускается процесс обработки сообщений бота с помощью метода `bot.polling()`, который ожидает новые сообщения от пользователей.

Описание функций

Функция	Описание
<code>get_auth_data()</code>	Возвращает данные аутентификации для отправки в WebSocket сервер.
<code>get_asr_data(lang)</code>	Возвращает данные для распознавания речи (ASR) с указанным языком <code>lang</code> .
<code>Recognition.__init__(binary_data)</code>	Инициализирует объект Recognition, принимая бинарные данные аудиофайла.
<code>Recognition.get_track_id()</code>	Запускает процесс распознавания трека путем подключения к WebSocket серверу и отправки аутентификационных данных и данных ASR, а также бинарных частей аудиофайла. Возвращает идентификатор распознанного трека или <code>None</code> , если распознавание не удалось.
<code>Recognition.is_valid_response(response)</code>	Проверяет, является ли ответ от сервера <code>response</code> валидным и содержит ли результат распознавания трека
<code>Recognition.is_fatal_error(response)</code>	Проверяет, является ли ответ от сервера <code>response</code> фатальной ошибкой, указывающей на невозможность распознать трек.
<code>Recognition.parse_track_id(response)</code>	Извлекает и возвращает идентификатор распознанного трека из ответа сервера <code>response</code> .
<code>Recognition.get_chunks_and_replace_encoder(binary_data)</code>	Разбивает бинарные данные аудиофайла на чанки и заменяет информацию об энкодере в заголовке каждого чанка. Возвращает список чанков.
<code>handle_start(message)</code>	Обработчик команды <code>/start</code> . Отправляет приветственное сообщение и просит пользователя отправить голосовое сообщение с песней.
<code>handle_voice_message(message)</code>	Обработчик голосового сообщения. Получает файл голосового сообщения, инициализирует распознавание трека с помощью Recognition, получает идентификатор распознанного трека и отправляет ссылку на Yandex.Music

	пользователю. Если трек не распознан, отправляет сообщение с просьбой отправить голосовое сообщение еще раз.
<code>bot.polling()</code>	Запускает бесконечный цикл опроса обновлений для бота Telegram. Бот будет прослушивать и обрабатывать входящие сообщения и команды.

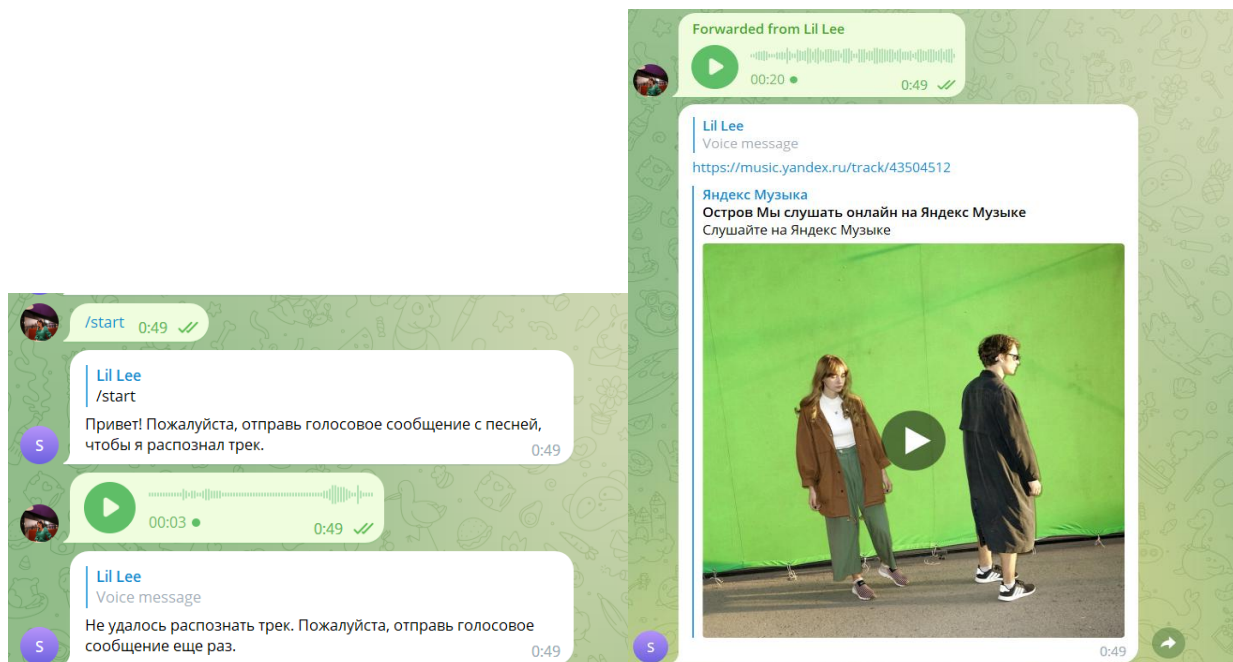
Работа бота

В текущей реализации бот реагирует на две команды:

1. `/start`: Бот отправляет приветственное сообщение и просит пользователя отправить голосовое сообщение с песней.

2. Отправка голосового сообщения: Бот принимает голосовое сообщение, пытается распознать трек в аудиофайле и отправляет ссылку на Yandex.Music, если трек успешно распознан. Если трек не может быть распознан, бот просит пользователя отправить голосовое сообщение еще раз.

Скриншоты



Листинг программы

```
import uuid
import lomond
import telebot
```

```

bot = telebot.TeleBot("BAШ BOT_TOKEN")
def get_auth_data():
    return {
        "event": {
            "header": {
                "messageId": str(uuid.uuid4()),
                "name": "SynchronizeState",
                "namespace": "System"
            },
            "payload": {
                "accept_invalid_auth": True,
                "auth_token": "5983ba91-339e-443c-8452-390fe7d9d308",
                "uuid": str(uuid.uuid4()).replace('-', ''),
            }
        }
    }
def get_asr_data(lang):
    return {
        "event": {
            "header": {
                "messageId": str(uuid.uuid4()),
                "name": "Recognize",
                "namespace": "ASR",
                "streamId": 1
            },
            "payload": {
                "advancedASROptions": {
                    "manual_punctuation": False,
                    "partial_results": False
                },
                "disableAntimatNormalizer": False,
                "format": "audio/opus",
                "music_request2": {
                    "headers": {
                        "Content-Type": "audio/opus"
                    }
                },
                "punctuation": False,
                "tags": "PASS_AUDIO;",
                "topic": "queries",
                "lang": lang
            }
        }
    }
class Recognition:
    URI = 'wss://voiceservices.yandex.net/uni.ws'
    LANGS = ['', 'ru-RU', 'en-US']
    POLL_DELAY = 0.3
    def __init__(self, binary_data):

```

```

self.websocket = lomond.WebSocket(self.URI)
self.chunks = self.get_chunks_and_replace_encoder(binary_data)
def get_track_id(self):
    for lang in Recognition.LANGS:
        asr_data = get_asr_data(lang)
        for msg in self.websocket.connect(poll=self.POLL_DELAY):
            if msg.name == 'ready':
                self.websocket.send_json(get_auth_data())
                self.websocket.send_json(asr_data)
                for chunk in self.chunks:
                    self.websocket.send_binary(chunk)
            if msg.name == 'text':
                response = msg.json.get('directive')
                if self.is_valid_response(response):
                    self.websocket.close()
                    return self.parse_track_id(response)
                elif self.is_fatal_error(response):
                    self.websocket.close()
                    break
def is_valid_response(self, response):
    if response.get('header', {}).get('name') == 'MusicResult' and \
        response.get('payload', {}).get('result') == 'success':
        return True
    return False
@staticmethod
def is_fatal_error(response):
    if response.get('header', {}).get('name') == 'MusicResult' and \
        response.get('payload', {}).get('result') == 'music':
        return False
    return True
@staticmethod
def parse_track_id(response):
    return response['payload']['data']['match']['id']
@staticmethod
def get_chunks_and_replace_encoder(binary_data):
    chunks = []
    for chunk in binary_data.split(b'OggS')[1:]:
        if b'OpusTags' in chunk:
            pos = chunk.index(b'OpusTags') + 12
            size = len(chunk)
            chunk = chunk[:pos] +
b'#\x00\x00\x00\x00ENCODER=SpeechKit Mobile SDK v3.28.0'
            chunk += b"\x00" * (size - len(chunk))
            chunks.append(b'\x00\x00\x00\x01OggS' + chunk)
    return chunks
@bot.message_handler(commands=['start'])
def handle_start(message):
    bot.reply_to(message, "Привет! Пожалуйста, отправь голосовое
сообщение с песней, чтобы я распознал трек.")

```

```

@bot.message_handler(content_types=['voice'])
def handle_voice_message(message):
    file_info = bot.get_file(message.voice.file_id)
    file = bot.download_file(file_info.file_path)
    recognition = Recognition(file)
    track_id = recognition.get_track_id()
    if track_id:
        yandex_music_link = f"https://music.yandex.ru/track/{track_id}"
        bot.reply_to(message, yandex_music_link)
    else:
        bot.reply_to(message, "Не удалось распознать трек. Пожалуйста,
отправь голосовое сообщение еще раз.")
bot.polling()

```

6.4. Создание аудиофайла по тексту. Silero

Телеграм-бот озвучивает написанный текст и присылает его в формате аудиосообщения. Используются: библиотека для обработки аудиофайлов `ffmpeg`, библиотека для написания телеграм-бота `aiogram` и голосовая модель `Silero` для синтеза речи [81]. Также используются вспомогательные библиотеки: `num2words` – для преобразование цифр в буквы, `os` – для реализации кода на разных платформах и `torch` – библиотека для работы с моделями искусственного интеллекта. Ниже расписан весь функционал бота, а также пример его написания.

Алгоритм работы

При отправке команды `/start` пользователю выводится информация о боте и его возможностях, а также предложение отправить текст, который он преобразует в аудиосообщение. Стоит отметить, что преобразование возможно только текста на русском языке, любой текст, написанный на латинице – игнорируется. Связано это с использованием модели `Silero`, которая обрабатывает текст только на русском языке.

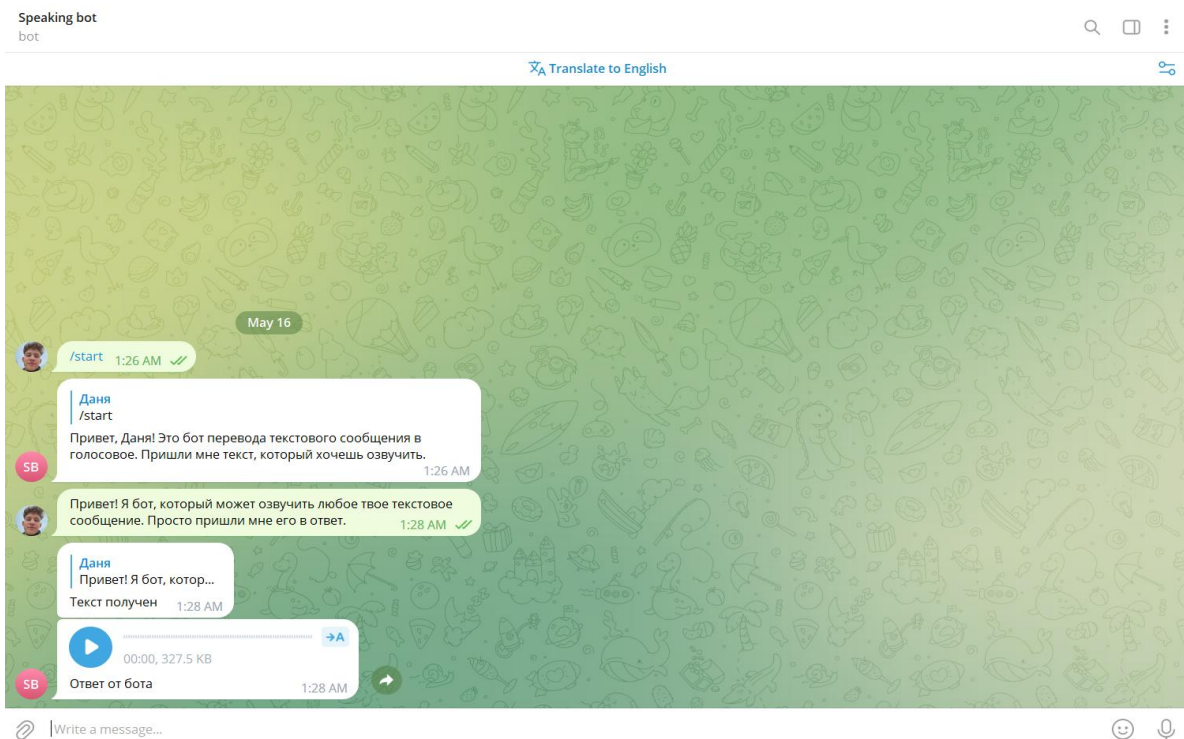
После отправки сообщения происходит его преобразование в аудиосообщение. Для этого используется отдельный класс `TTS`, в котором

прописана основная логика бота. Там реализуются следующие методы и функции:

Использованные функции

Функция	Описание
<code>text_to_wav (self, text: str, out_filename: str = None)</code>	Конвертирует текст в файл wav. Главная функция класса. Модель игнорирует латиницу, но поддерживает цифры числами.
<code>default_init</code>	В ней происходит изначальная инициализация класса следующими параметрами: качеством звука, голосом диктора, путь до звуковой модели, тип устройства и т.д. Т.е. фактически здесь идет настройка модели Silero
<code>_check_model</code>	Проверка наличия модели Silero на нужном языке в каталоге приложения
<code>_get_wav (self, text: str, speaker_voice=None, sample_rate=None)</code>	Конвертирует текст в wav файл. Как дополнительные параметры можно задать голос диктора и качество звука, но обычно они задаются по умолчанию.
<code>_nums_to_text(self, text: str)</code>	Преобразует числа в буквы: 1 -> один, 23 -> двадцать три.
<code>_merge_audio_n_to_1(self, in_filenames: list, out_filename: str = None)</code>	Объединяет несколько файлов в один файл без перекодирования. Файлы должны быть одинакового формата.
<code>message.reply(text)</code>	Выводит текстовое сообщение text пользователю
<code>bot.send_voice(voice)</code>	Отправляет голосовое сообщение voice пользователю

Пример работы программы



Листинг программы

Файл с ботом:

```
import os
from tts import TTS
from pathlib import Path
from aiogram import Bot, Dispatcher, executor, types
from aiogram.types.input_file import InputFile

TOKEN = "6122379185:AAEMQro1Xpq4RRh_hFLg3ckag5IZU730JQk"

bot = Bot(token=TOKEN)
dp = Dispatcher(bot=bot)
@dp.message_handler(commands=['start'])
async def start_handler(message: types.Message):
    user_full_name = message.from_user.full_name
    await message.reply(f"Привет, {user_full_name}! Это бот перевода
текстового сообщения в голосовое. "
                        f"Пришли мне текст, который хочешь озвучить.")

@dp.message_handler(commands=['help'])
async def help_handler(message: types.Message):
    await message.reply("Я могу перевести текст в голосовое сообщение. "
                        "Для этого просто отправь мне любое текстовое
сообщение. Но важно писать его на русском.")

# Хэндлер на получение текста
@dp.message_handler(content_types=[types.ContentType.TEXT])
async def cmd_text(message: types.Message):
```

```

tts = TTS()
"""
Обработчик на получение текста
"""
await message.reply("Текст получен")
out_filename = tts.text_to_wav(message.text)

# Отправка голосового сообщения
path = Path("", out_filename)
voice = InputFile(path)
await bot.send_voice(message.from_user.id, voice,
                    caption="ОТВЕТ ОТ БОТА")

# Удаление временного файла
os.remove(out_filename)

if __name__ == "__main__":
    # Запуск бота
    print("Запуск бота")
    try:
        executor.start_polling(dp, skip_updates=True)
    except (KeyboardInterrupt, SystemExit):
        pass

Файл с основной логикой:
import os
import re
import subprocess
import torch
from num2words import num2words
class TTS:
    """
    Класс для преобразования текста в аудио.
    Поддерживаются форматы аудио: wav, ogg
    """
    default_init = {
        "sample_rate": 24000,
        "device_init": "cpu",
        "threads": 4,
        "speaker_voice": "baya",
        "model_path": "models/silero/model.pt", # путь к файлу TTS
        "model_url": "https://models.silero.ai/models/tts/ru/v3_1_ru.pt",
        "ffmpeg_path": "models/silero" # путь к ffmpeg
    }
    def __init__(
        self,
        sample_rate=None,

```

```

        device_init="cpu",
        threads=None,
        speaker_voice=None,
        model_path=None,
        model_url=None,
        ffmpeg_path=None
    ) -> None:
        """
        :arg sample_rate: int          # 8000, 24000, 48000 - качество звука
        :arg device_init: str          # "cpu", "gpu"(для gpu нужно ставить
другой torch)
        :arg threads: int              # количество тредов, например, 4
        :arg speaker_voice: str        # диктор "aidar", "baya", "kseniya",
" Xenia", "random"
        :arg model_path: str           # путь до модели silero
        :arg model_url: str            # URL к TTS модели Silero
        :arg ffmpeg_path: str          # путь к ffmpeg
        """
        self.sample_rate = sample_rate if sample_rate else
TTS.default_init["sample_rate"]
        self.device_init = device_init if device_init else
TTS.default_init["device_init"]
        self.threads = threads if threads else
TTS.default_init["threads"]
        self.speaker_voice = speaker_voice if speaker_voice else
TTS.default_init["speaker_voice"]
        self.model_path = model_path if model_path else
TTS.default_init["model_path"]
        self.model_url = model_url if model_url else
TTS.default_init["model_url"]
        self.ffmpeg_path = ffmpeg_path if ffmpeg_path else
TTS.default_init["ffmpeg_path"]
        self._check_model()
        device = torch.device(self.device_init)
        torch.set_num_threads(self.threads)
        self.model =
torch.package.PackageImporter(self.model_path).load_pickle("tts_models",
"model")
        self.model.to(device)

    def _check_model(self):
        if not os.path.isfile(self.model_path):
            torch.hub.download_url_to_file(self.model_url,
self.model_path)

    isffmpeg_here = False
    for file in os.listdir(self.ffmpeg_path):
        if file.startswith('ffmpeg'):
            isffmpeg_here = True

```

```

if not isffmpeg_here:
    raise Exception(
        "Ffmpeg: сохраните ffmpeg.exe в папку ffmpeg\n"
        "Скачайте ffmpeg.exe по ссылке
https://ffmpeg.org/download.html"
    )
self.ffmpeg_path = self.ffmpeg_path + '/ffmpeg'
def _get_wav(self, text: str, speaker_voice=None, sample_rate=None) -
> str:
    if text is None:
        raise Exception("Передайте текст")

    # Удаляем существующий файл чтобы все хорошо работало
    if os.path.exists("test.wav"):
        os.remove("test.wav")
    if speaker_voice is None:
        speaker_voice = self.speaker_voice
    if sample_rate is None:
        sample_rate = self.sample_rate

    # Сохранение результата в файл test.wav
    return self.model.save_wav(
        text=text,
        speaker=speaker_voice,
        sample_rate=sample_rate
    )
def _nums_to_text(self, text: str) -> str:
    """
    :arg text: str
    :return: str
    """
    return re.sub(
        r"(\d+)",
        lambda x: num2words(int(x.group(0)), lang="ru"),
        text)
def _merge_audio_n_to_1(
    self,
    in_filenames: list,
    out_filename: str = None
    ) -> str:
    """
    :arg in_filenames: list[str] # список файлов для склеивания
    :arg out_filename: str # имя выходного файла
    :return out_filename: str # имя выходного файла
    """
    if not in_filenames:
        raise Exception("Укажите пути и имя файла in_filenames")
    if out_filename is None:

```

```

        extension = in_filenames[0].split(sep=".")[-1]
        out_filename = f"test_n_1.{extension}"
    if os.path.exists(out_filename):
        os.remove(out_filename)
    filenames = "\n".join([f"file '{filename}'" for filename in
in_filenames])
    with open("audiolist.txt", "wt") as file:
        file.write(filenames)
    command = [
        self.ffmpeg_path,
        "-loglevel", "quiet",
        "-f", "concat",
        "-safe", "0",
        "-i", "audiolist.txt",
        "-c", "copy",
        out_filename,
    ]
    proc = subprocess.Popen(command)
    proc.wait()
    if os.path.exists("audiolist.txt"):
        os.remove("audiolist.txt")
    return out_filename
def _rename_file(self, in_filename: str, out_filename: str) -> str:
    """
    Переименует in_filename файл в out_filename.
    :arg in_filename: str # имя входного файла
    :arg out_filename: str # имя выходного файла
    :return: str # имя выходного файла
    """
    if in_filename is None or out_filename is None:
        raise Exception("Передайте названия входного и выходного
файла")
    if os.path.exists(out_filename):
        os.remove(out_filename)
    os.rename(in_filename, out_filename)
    return out_filename
def text_to_wav(self, text: str, out_filename: str = None) -> str:
    """
    Конвертирует текст в файл wav.
    Модель игнорирует латиницу, но поддерживает цифры числами.
    :arg text: str # текст кириллицей
    :return: str # имя выходного файла
    """
    if text is None:
        raise Exception("Передайте текст")

    # Делаем числа буквами
    text = self._nums_to_text(text)

```

```

# Передаем текст целиком
if len(text) < 800:
    # Конвертируем текст в wav, возвращаем путь до wav
    wav_audio_path = self._get_wav(text)
    if out_filename is None:
        return wav_audio_path
    return self._rename_file(wav_audio_path, out_filename)

# Разбиваем текст, конвертируем и склеиваем аудио в один файл
texts = [text[x:x+800] for x in range(0, len(text), 800)]
files = []
for index in range(len(texts)):
    # Конвертируем текст в wav, возвращаем путь до wav
    wav_audio_path = self._get_wav(texts[index])
    # Переименовываем чтобы не затереть файл
    new_wav_audio_path = f"{index}_{wav_audio_path}"
    os.rename(wav_audio_path, new_wav_audio_path)
    # Добавляем файл в список
    files.append(new_wav_audio_path)

# Склеиваем все wav файлы в один
wav_audio_path = self._merge_audio_n_to_1(files,
out_filename="test_n_1.wav")
# Удаляем временные файлы
[os.remove(file) for file in files]
if out_filename is None:
    return wav_audio_path
return self._rename_file(wav_audio_path, out_filename)

```

6.5. Повышение разрешения изображения. ESGRAN

Телеграм-бот позволяет увеличить разрешение картинки в 4 раза.

Инструмент по повышению разрешения в данном боте – нейросеть ESGRAN – Enhanced Super-Resolution GAN, улучшенные генеративно-сопоставительные сети для высокого разрешения. GAN - Generative Adversarial Networks, генеративно-сопоставительные сети, метод обучения генеративной модели. На высоком уровне можно представить эту модель как соревнование двух подмоделей, одна из которых – генератор – пытается научиться обманывать вторую – дискриминатор. Генератор при этом учится создавать новые объекты на основании распределения

некоторого сложного набора данных. Дискриминатор же учится отличать сгенерированные объекты от реальных.

Подготовительный этап

1. Для работы бота нужно в каталоге проекта создать подкаталог ESGRAN, в который необходимо перенести содержимое <https://github.com/xinntao/ESRGAN> (актуально на 15.01.2025)
2. Далее перейдя в ESGRAN, необходимо открыть подкаталог models, в которую необходимо скачать модель RRDB_ESGRAN_x4.pth отсюда:
https://drive.google.com/drive/folders/17VYV_SoZZesU6mbxz2dMAIccS5lqLecY?usp=sharing
3. После этого необходимо установить opencv-python, glob2, pytorch
Для этого были выполнены следующие команды

```
pip install opencv-python glob2
```

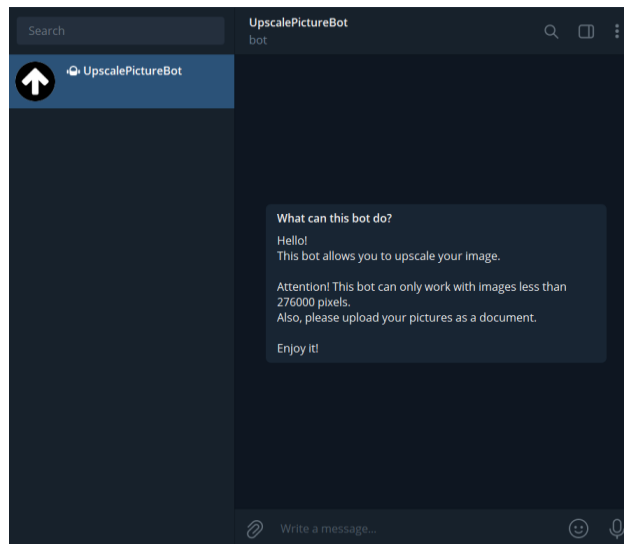
```
pip3 install torch torchvision torchaudio --extra-index-url  
https://download.pytorch.org/whl/cu113
```

Команда для установки pytorch была взята с официального сайта (<https://pytorch.org/get-started/locally/>)

4. Также необходимо вставить ваш токен бота в качестве значения переменной TOKEN в main.py

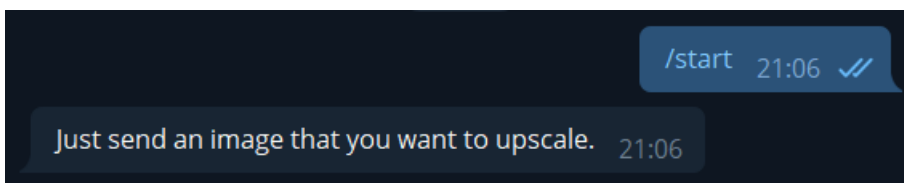
Алгоритм работы бота

До запуска бота, в пространстве, где в дальнейшем будут выводиться сообщения пользователя приветствуют, а также уведомляют о том, что этот бот умеет.



После запуска бота с помощью `start.bat` есть возможность отправить сообщения. Бот реагирует только на команды «`/start`» и «`/help`», а также на сообщения, у которых значения поля `content_types` равно «`document`». Все остальные сообщения ботом будут проигнорированы.

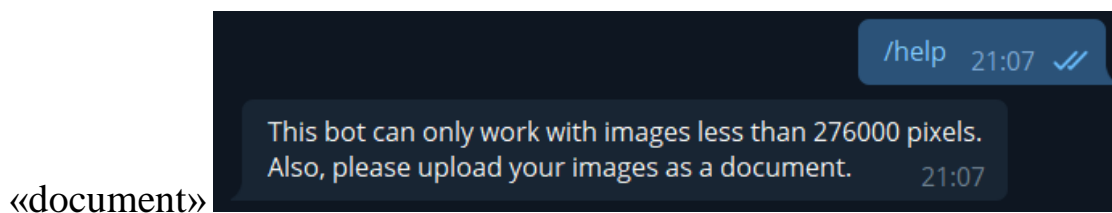
- а. По команде «`/start`» бот просит пользователя отправить изображение для обработки



По команде «`/help`» бот присылает сообщение с условиями для изображений, с которыми он будет работать, а именно:

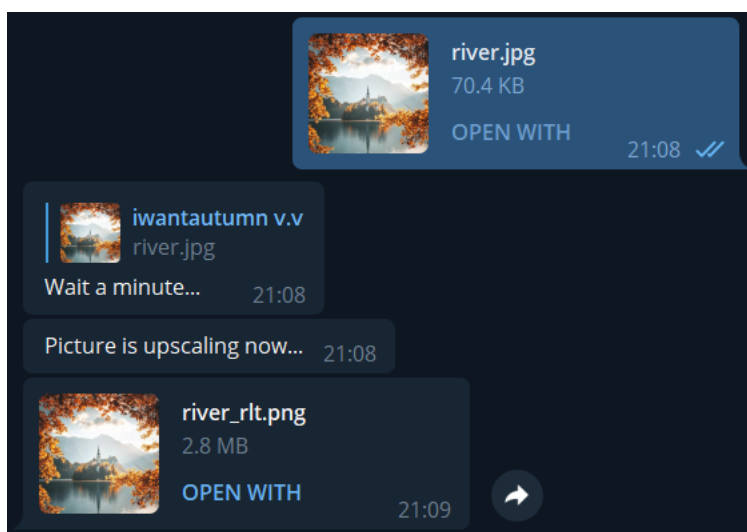
- 1) изображение должно содержать не более 276000 пикселей (из-за ограниченного количества видеопамати, на разных устройствах данное значение может отличаться; иначе просто не получится увеличить разрешение изображения)

2) изображение должно быть загружено как документ, чтобы у сообщения с изображением значение поля content_types было равно

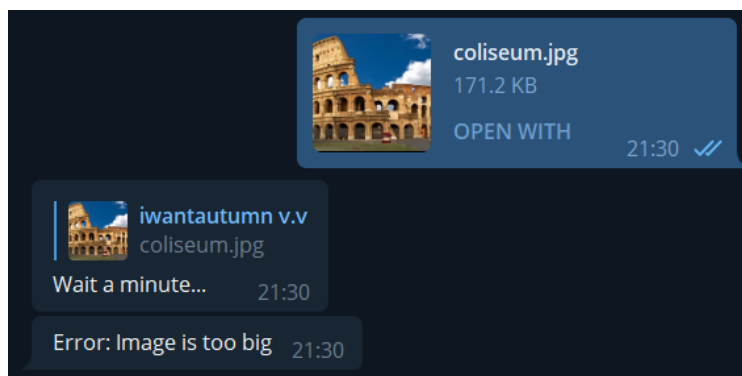


b. На сообщения, у которых значения поля content_types равно «document» бот также может отреагировать по-разному.

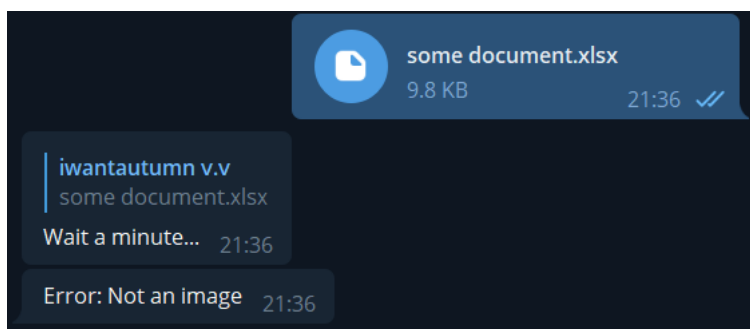
i. В случае, если загруженный документ – изображение с количеством пикселей меньше 276000 всё пройдет нормально и бот пришлёт изображение с разрешением в 4 раза больше от исходного:



ii. В случае, если загруженный документ – изображение с количеством пикселей больше 276000 возникнет ошибка с соответствующим сообщением:



iii. В случае, если загруженный документ – не изображение возникнет ошибка с соответствующим сообщением:



Список используемых классов, функций и декораторов

telebot	bot = Telebot(token)	Создание объекта бота с присвоением токена token
	bot.send_message()	Отправить сообщение
	bot.reply_to()	Ответить на сообщение
	bot.send_document()	Отправить документ
	bot.infinity_polling()	Бот бесконечно проверяет не приходят ли новые запросы от пользователя
	@bot.message_handler()	Позволяет вызывать функцию только для конкретных типов сообщений пользователя.

Листинг программы

```
main.py
import os
import time
import subprocess
from pathlib import Path
from PIL import Image
import telebot
TOKEN = 'PASTE_YOUR_TOKEN'
bot = telebot.TeleBot(TOKEN, parse_mode=None)
@bot.message_handler(commands=['start'])
def send_welcome(message):
    bot.send_message(message.chat.id, "Just send an image that you want
to upscale.")
@bot.message_handler(commands=['help'])
def send_welcome(message):
    bot.send_message(message.chat.id, "This bot can only work with images
less than 276000 pixels.\n" +
                                "Also, please upload your images as
a document.")
```

```

@bot.message_handler(content_types=["document"])
def get_image_messages(message):
    bot.reply_to(message, "Wait a minute...")

    # информация о присланном документе (изображении)
    file_name = message.document.file_name
    file_id = message.document.file_id
    file_id_info = bot.get_file(file_id)
    downloaded_file = bot.download_file(file_id_info.file_path)

    # скачиваем присланный файл
    with open("./ESRGAN/LR/" + file_name, 'wb') as new_file:
        new_file.write(downloaded_file)
    sent_file = Path('./ESRGAN/LR/' + file_name)
    while not sent_file.is_file():
        time.sleep(2)

    # проверка на возможность обработки
    try:
        sent_image = Image.open(sent_file)
    except():
        bot.send_message(message.chat.id, "Error: Not an image")
        os.remove('./ESRGAN/LR/' + file_name)
        return None
    if (sent_image.size[0] * sent_image.size[1]) > 276000:
        sent_image.close()
        bot.send_message(message.chat.id, "Error: Image is too big")
        os.remove('./ESRGAN/LR/' + file_name)
        return None
    sent_image.close()

    # обработка присланного изображения с помощью нейросети
    bot.send_message(message.chat.id, "Picture is upscaling now...")
    command = "cd ESRGAN && python test.py"
    p = subprocess.Popen(command, shell=True, stdout=subprocess.PIPE)
    while (p.wait() != 0):
        time.sleep(2)

    # отправляем картинку с разрешением увеличенным в 4 раза
    upgraded_file_name = file_name[:file_name.find('.')]
    upgraded_file = open('./ESRGAN/results/' + upgraded_file_name +
'_r1t.png', "rb")
    bot.send_document(message.chat.id, upgraded_file)
    upgraded_file.close()

    # чистим папки с изображениями
    os.remove('./ESRGAN/results/' + upgraded_file_name + '_r1t.png')

```

```
os.remove('./ESRGAN/LR/' + file_name)
bot.infinity_polling()
```

```
start.bat
@echo off
python main.py
```

Результаты работы программы

1) Повышение разрешения изображения river.jpg



исходник (320x320)



результат (1280x1280)

2) Повышение разрешения изображения athens.jpg



исходник (1000x276)



6.6. Выполнение арифметических выражений, введенных с изображений. EasyOCR

Бот позволяет решать арифметические выражения, которые вводятся с рукописных изображений. Используется EasyOCR - система оптического распознавания текста, которая поддерживает более 40 языков [82]. В EasyOCR для определения и распознавания текста на изображении применяются методы машинного обучения. Для определения текста используется алгоритм машинного обучения CRAFT (Character-Region Awareness For Text) в реализации для PyTorch, способный выделять текст на произвольных объектах, включая этикетки, информационные таблички и дорожные знаки. Для распознавания последовательностей символов применяется свёрточно-рекуррентная нейронная сеть CRNN (Convolutional Recurrent Neural Network, комбинация DCNN и RNN) и алгоритм CTC BeamSearch (Connectionist Temporal Classification) для декодирования выходных данных нейронной сети в текстовое представление.

Для работы бота необходимы библиотеки:

easyocr - pip3 install easyocr;

telegram.ext - pip3 install pip install python-telegram-bot.

Также необходимо вставить ваш токен бота в качестве значения переменной token в функции main.

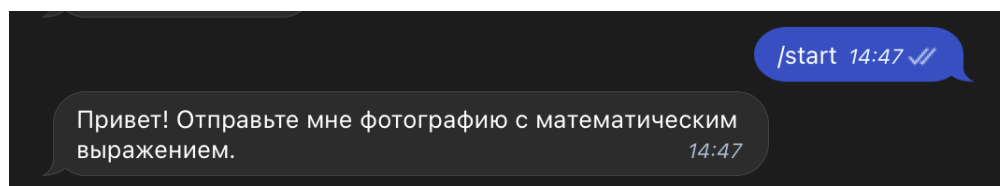
Таблица функций

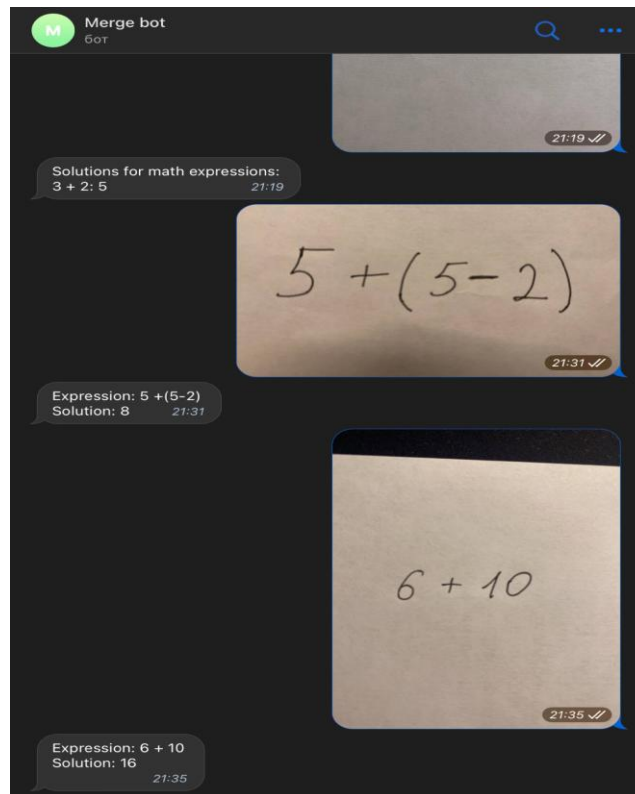
Updater	Этот класс из библиотеки telegram.ext используется для обработки взаимодействия с Telegram Bot API. Он предоставляет интерфейс для получения обновлений от Telegram и их обработки. Этот класс используется для обработки входящих сообщений. Он позволяет указать
---------	--

	фильтры, чтобы определить, какие сообщения должны обрабатываться. В коде он используется для обработки сообщений с фотографиями.
MessageHandler	Отправление сообщений в чатЭтот класс используется для обработки входящих сообщений. Он позволяет указать фильтры, чтобы определить, какие сообщения должны обрабатываться. В коде он используется для обработки сообщений с фотографиями.
Filters	Этот класс предоставляет predefined фильтры, которые можно использовать с MessageHandler для фильтрации входящих сообщений на основе различных критериев. В коде фильтр
File.download	Этот метод объекта telegram.File используется для загрузки файла в указанное место. В коде он используется для загрузки файлов изображений, отправленных пользователем.
context.bot.get_file	Этот метод используется для отправки сообщения пользователю. Он принимает идентификатор чата и текст сообщения в качестве входных данных.
updater.start_polling	Этот метод запускает бота и начинает проверку наличия обновлений от Telegram. Он постоянно проверяет новые обновления и вызывает соответствующие обработчики.
easyocr.Reader	Этот класс используется для создания объекта OCR-читателя, который может распознавать текст на изображениях. В коде он используется для создания объекта считывателя, чтобы выполнить распознавание текста на изображении.
reader.readtext	Этот метод класса Reader используется для чтения текста с изображения. Он принимает путь к изображению в качестве входных данных и возвращает список распознанного текста.

Алгоритм работы бота

По команде «/start» бот начинает свою работу и ожидает получить от пользователя изображение





Листинг программы

```
import os
import easyocr
from telegram.ext import Updater, MessageHandler, Filters, CommandHandler

def solve_math_expression(expression):
    try:
        result = eval(expression)
        return result
    except (SyntaxError, TypeError, NameError):
        return "Invalid expression"

def handle_image(update, context):
    file_id = update.message.photo[-1].file_id
    file_path = context.bot.get_file(file_id).download()
    reader = easyocr.Reader(['ru'])
    result = reader.readtext(file_path, detail=0)
    expression = result[0]
    solution = solve_math_expression(expression)
    response = f"Expression: {expression}\nSolution: {solution}"
    context.bot.send_message(chat_id=update.effective_chat.id,
text=response)
    os.remove(file_path)

def handle_start(update, context):
```

```
context.bot.send_message(chat_id=update.effective_chat.id,  
text="Привет! Отправьте мне фотографию с математическим выражением.")
```

```
def main():  
    updater = Updater(token='YOUR_BOT_TOKEN', use_context=True)  
    dispatcher = updater.dispatcher  
    image_handler = MessageHandler(Filters.photo, handle_image)  
    start_handler = CommandHandler('start', handle_start)  
    dispatcher.add_handler(image_handler)  
    dispatcher.add_handler(start_handler)  
    updater.start_polling()
```

```
if __name__ == '__main__':  
    main()
```

6.7. Генерация изображения из текста. DALL-E

Одним из самых впечатляющих применений нейронных сетей, разработанных OpenAI, является возможность генерировать изображения из текста. Нейронная сеть, ответственная за этот прорыв, называется DALL-E, что означает «Dali + WALL-E». Эта нейронная сеть принимает описания на естественном языке в качестве входных данных и выводит соответствующие изображения [83].

DALL-E сначала кодирует входной текст в математическое представление. Затем это представление используется для создания начального изображения, которое уточняется с помощью серии итераций. Нейронная сеть использует комбинацию методов, таких как механизмы внимания и условные GAN (генеративно-состязательные сети), чтобы гарантировать точность и разнообразие сгенерированных изображений.

Например, если пользователь вводит фразу «кресло в форме авокадо», DALL-E создаст изображение, соответствующее этому описанию. Сгенерированное изображение будет уникальной интерпретацией пользовательского ввода, а такие детали, как текстура кресла и цвет авокадо, будут определяться нейронной сетью.

Таблица использованных функций

Название функции	Значения функции
<code>generate_image(prompt, num_image=1, size='512x512', output_format='url')</code>	пользовательская функция, которая использует OpenAI API для создания изображений на основе переданного текстового промпта и необязательных параметров <code>num_image</code> , <code>size</code> и <code>output_format</code> . Возвращает словарь, содержащий информацию о времени создания и ссылки или закодированные изображения, в зависимости от формата вывода.
<code>openai.Image.create()</code>	метод объекта API OpenAI Image. Он используется для создания изображений на основе переданного текстового промпта и необязательных параметров <code>num_image</code> , <code>size</code> и <code>response_format</code> . Метод возвращает файлы изображения в виде URL-адреса или закодированного JSON, в зависимости от выбранного формата вывода.
<code>b64decode()</code>	метод встроенного модуля <code>base64</code> в Python, который используется для декодирования данных, закодированных в формат <code>base64</code> . Base64 — это способ кодирования бинарных данных в текстовый вид, состоящий из латинских букв и цифр, что упрощает их передачу по интернет-протоколам и сохранение без ошибок на жестком диске или в базе данных.

Листинг программы

```
import datetime
import configparser
from base64 import b64decode
import openai
from openai.error import InvalidRequestError

text = 'Monkeys'
count = 2
prefix = 'demo'
config = configparser.ConfigParser()
config.read('credential.ini')
API_KEY = config['openai']['APIKEY']

openai.api_key = API_KEY

SIZES = ('1024x1024', '512x512', '256x256')

def generate_image(prompt, num_image=1, size='512x512',
output_format='url'):
    try:
        images = []
```

```

response = openai.Image.create(
    prompt=prompt,
    n=num_image,
    size=size,
    response_format=output_format
)
if output_format == 'url':
    for image in response['data']:
        images.append(image.url)
elif output_format == 'b64_json':
    for image in response['data']:
        images.append(image.b64_json)
return {'created':
datetime.datetime.fromtimestamp(response['created']), 'images': images}
except InvalidRequestError as e:
    print(e)
response = generate_image(text, num_image=count,
                           size=SIZES[1], output_format='b64_json')
for indx, image in enumerate(response['images']):
    with open(f'{prefix}_{indx}.jpg', 'wb') as f:
        f.write(b64decode(image))

```

Результат выполнения программы

В результате выполнения программы при вводе запроса «Monkeys» мы получим одно или несколько изображений, сгенерированных при помощи API от OpenAI.



6.8. Наложение изображения на фон в видеофайле/ распознавание номеров

Интерактивное приложение использующее интерфейс telegram-бота выполняет следующие действия.

- принимает на вход изображение с описанием;

- в зависимости от описания `number` или `background`, выполняет программы на сервере: при выборе `number`, бот возвращает изображение найденного номера и его распознанное текстовое представление или сообщает о том, что распознать номер не удалось; при выборе `background`, бот возвращает видеофайл с наложенным изображением на фон.

Создание бота

Воспользуемся ботом **@BotFather**, для создания нового бота необходимо перейти в **@BotFather**, ввести `/newbot` и следовать дальнейшим инструкциям в интерактивном режиме. как показано на рис.1.

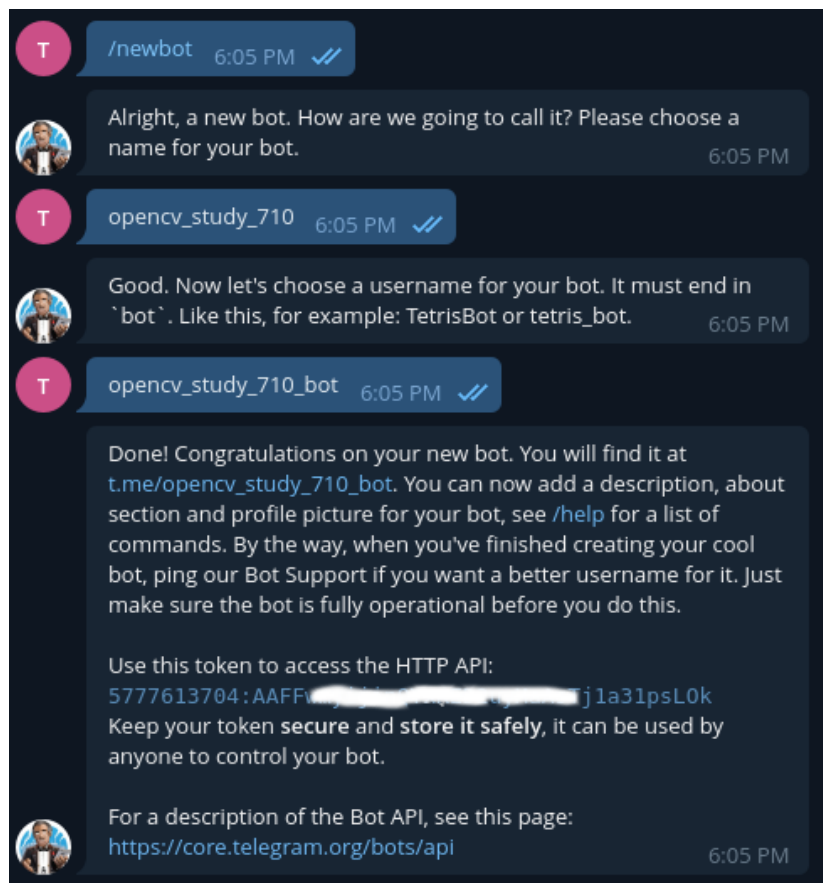


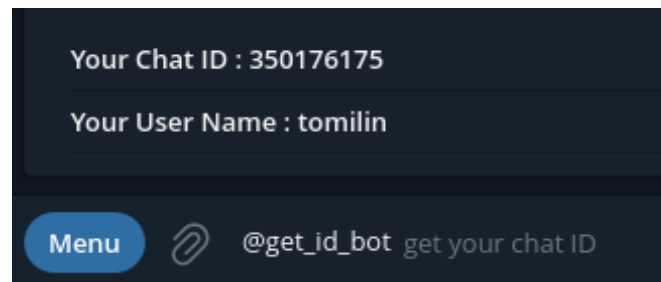
Рис. 1. Создание нового бота

После этого бот создан и можно проверить что бот создан, для этого перейти в чат бота **@opencv_study_710_bot**

Все нижеописанные шаги выполнялись в дистрибутиве linux debian 11.

Первое что нужно сделать, добавить переменные окружения:
`export OPENCV_BOT_TOKEN=<token_http_api>` # взять из @BotFather
`export OPENCV_BOT_CHAT_ID=<your_telegram_chat_id>`

Получить chat_id можно введя в любой чат телеграма строку @get_id_bot



Для того, чтобы запустить бота, нужно скачать дистрибутив: https://github.com/itomilin/opencv_play.git перейти в скачанную директорию, установить зависимости выполнив команду `./run.sh` с правами root, выполнить команду `pip3 install -r requirements.txt` (на машине должен быть предустановлен `python3` и `pip3`). После того, как все зависимости были установлены, можно запустить бота выполнив команду `./start_bot` на своей машине приложение.

Пример работы

Изображения необходимо отправлять с выбранным **checkbox** “**compress images**” как на рис. 2, поскольку при отправке без выбранного checkbox тип файла будет распознан ботом как document и не будет обработан.

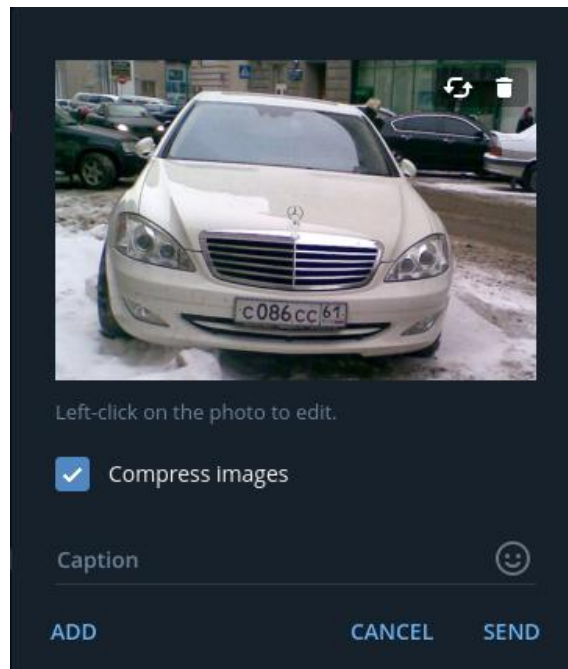


Рис. 2. Правильный вариант загрузки изображения.

Если не задан caption, то бот сообщит о том, что его нужно ввести, рис. 3.

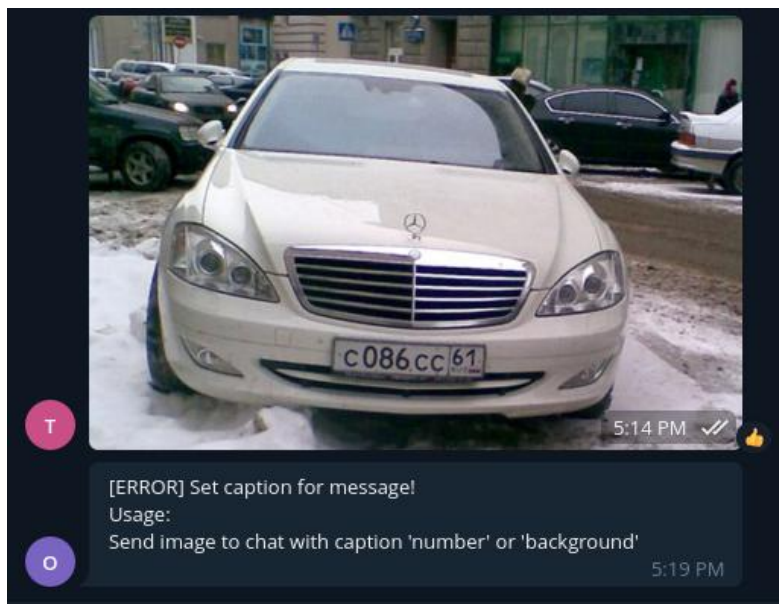


Рис. 3. Сообщение бота, если не указан тип

Добавим к caption “number” к отправляемому изображению и проверим результат обработки, рис. 4.

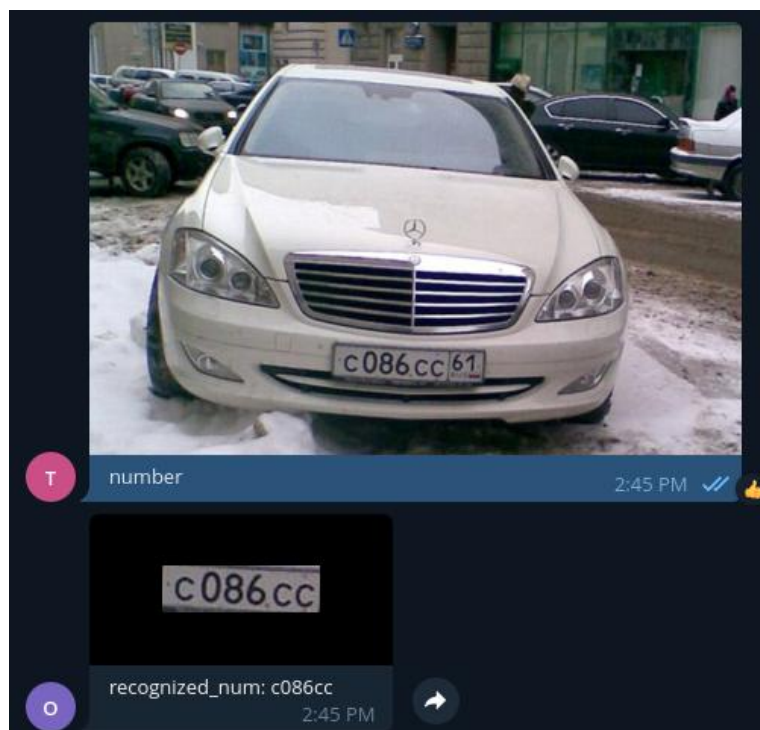


Рис. 4. Успешный результат распознавания номера

Пример неудачного распознавания номера, рис. 5.



Рис. 5. Неудачное распознавание номера.

Теперь добавим второй вариант “background”, показанный на рис. 6. Отправленное изображение стало фоном видео.

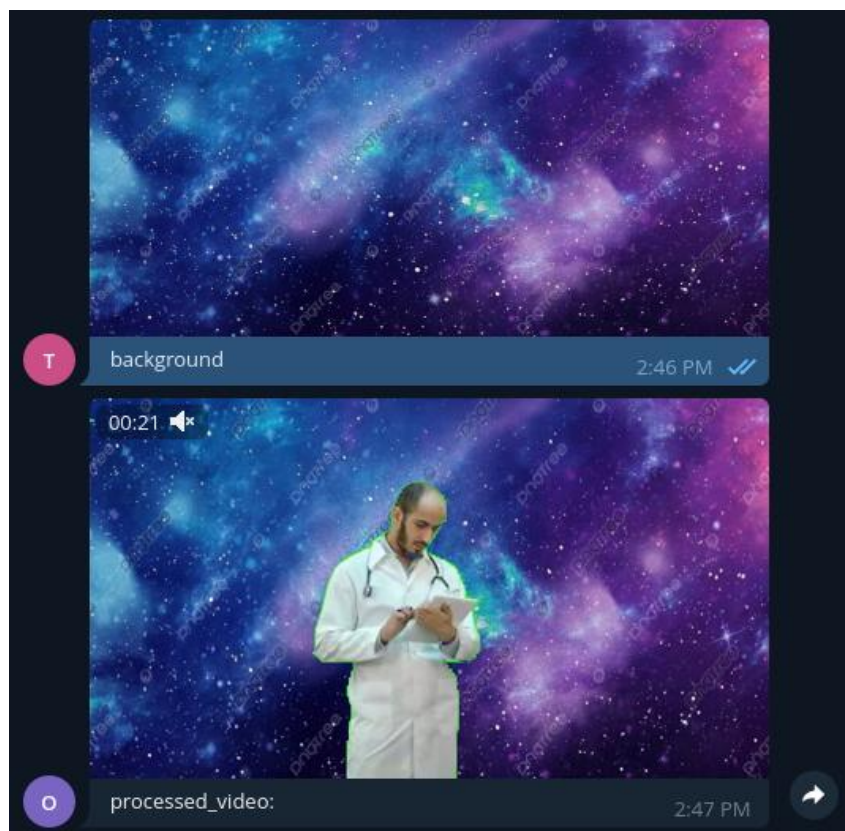


Рис. 6. Кадр замененного фона у видео.

Использованные функции API и библиотеки

Была использована python библиотека для работы с telegram ботами **telepot** – <https://github.com/nickoala/telepot>.

Для того чтобы поставить библиотеку к себе на компьютер нужно ввести команду: **pip3 install telepot**

Для запуска прослушивания входящих сообщений в чате, используется 2 функции.

Первая функция **Bot**, принимает на вход, HTTP_API токен бота и возвращает рабочий объект для взаимодействия.

bot = telepot.Bot(BOT_TOKEN)

Вторая функция **MessageLoop** принимает на вход экземпляр бота и функцию-обработчик входящих сообщений, данная функция запускается в фоновом режиме, поэтому необходимо сделать бесконечную задержку в программе.

MessageLoop(bot, handle).run_as_thread()

Все входящие сообщения в чат, делятся на несколько типов: text, photo, video и.т.д, для обработки сообщений необходимо определить тип сообщения из поля входящего JSON и обработать его соответствующей функцией.

В данной работе производилась обработка только типа photo, использовались следующие функции для чтения и отправки файлов и сообщений.

Для отправки сообщения с сервера в чат, используется функция ***sendMessage(CHAT_ID, STR)***, первый параметр id чата, второй параметр текст отправляемого сообщения.

Для отправки фотографии с сервера в чат, используется функция ***sendPhoto(CHAT_ID, PATH_TO_IMG, caption=STR)***, первый параметр id чата, второй параметр путь до изображения, третий параметр опциональный, текст к отправляемому изображению.

Для отправки видеофайла с сервера в чат, используется функция ***sendVideo(CHAT_ID, PATH_TO_VIDEO, caption=STR)***, первый параметр id чата, второй параметр путь до видеофайла, третий параметр опциональный, текст к отправляемому видео.

Для загрузки изображения отправленное в чат, используется функция ***download_file(CHAT_FILE_ID, PATH_TO_SAVE)***, первый параметр id файла из чата, второй параметр путь на сервере для сохранения изображения.

Проект доступен на github по ссылке:

https://github.com/itomilin/opencv_play/tree/master/course

Текст программы

Телеграм бот

```
import sys
import os
import time

import telepot
from telepot.loop import MessageLoop
from pprint import pprint

from number_recognition import number_recognition
from background_remover import change_background

BOT_TOKEN = os.environ['OPENCV_BOT_TOKEN']
CHAT_ID   = int( os.environ['OPENCV_BOT_CHAT_ID'] )

def usage():
    return "Usage:\nSend image to chat with caption 'number' or
'background'"

def handle( msg ):
    content_type, chat_type, chat_id = telepot.glance( msg )
    if content_type == 'photo':
        case = None
        try:
            case = msg['caption']
            pprint( msg['caption'] )
        except KeyError:
            err_msg = f"[ERROR] Set caption for message!\n{usage()}"
            print( err_msg )
            bot.sendMessage( CHAT_ID, err_msg )
            return

        img_path = f"./chat_files/{msg['photo'][-1]['file_unique_id']}.png"
        _, file_name = os.path.split( img_path )

        bot.download_file( msg['photo'][-1]['file_id'], img_path )
        if case == "number":
            text_number, status = number_recognition( img_path )
            if status:
                bot.sendPhoto( CHAT_ID,
                               photo=open(
f"./processed_files/{file_name}", 'rb'),
                               caption=f'recognized_num: {text_number}')
```

```

        else:
            bot.sendMessage( CHAT_ID, f'recognized_num:
{text_number}' )
            elif case == "background":
                change_background( img_path )
                bot.sendVideo( CHAT_ID,
video=open(f"./processed_files/{file_name.split('.')[0]}.mp4", 'rb'),
caption=f'processed_video:')
            else:
                print( "Wrong case, try again!" )
                return

                print( "!!processing done!!" )
            elif content_type == 'video':
                pass
            elif content_type == 'text':
                pass
            else:
                print( "Type is not handled" )

bot = telepot.Bot( BOT_TOKEN )
MessageLoop( bot, handle ).run_as_thread()
print( usage() )

```

```

# Keep the program running.
while True:
    time.sleep( 1200 )

```

Распознавание номеров

```

import os
import cv2
import pytesseract
from imutils import contours

def number_recognition( img ):
    _, file_name = os.path.split( img )
    car_number_img = None
    msg = "could not recognize"
    status = 0
    image = cv2.imread( img )

    height, width = image.shape[:2]

    image_gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY) # преобразовать
изображение в

    #cv2.imshow("Gray Image", image_gray) #вывести на экран для проверки
    #cv2.waitKey()
    # пороговая обработка для выделения контуров

```

```

threshold = cv2.threshold(image_gray, 0, 255, cv2.THRESH_OTSU)[1]
#cv2.imshow("Threshold", threshold) #вывести на экран для проверки
#cv2.waitKey()

# Найдем все контуры
contourss = cv2.findContours(threshold, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
contourss, _ = contourss.sort_contours(contourss[0]) # отсортируем их

chars = set('0123456789,') # создадим множество цифр для отсеивания
"мусорных" текстов

for c in contourss: # переберем все контуры
    area = cv2.contourArea(c) # найдем площадь контура
    x, y, w, h = cv2.boundingRect(c)
    if area > 500: # если площадь соизмерима с номером
        img = image[y:y+h, x:x+w] # получим этот контур из исходного
изображения
        # cv2.waitKey()
        result = pytesseract.image_to_string(img, lang='rus+eng')

        if len(result) > 7:
            car_number_img = img.copy()
            chars = "-`'\/'( ,. "
            for char in chars:
                result = result.replace( char, "" )
            msg = result
            cv2.imwrite( f"./processed_files/{file_name}",
car_number_img )
            status = 1

#cv2.imshow( "car_n", car_number_img )
#cv2.waitKey()
return msg, status

```

Замена фона на видео

```

import cv2
import numpy as np
import os

def change_background( img_path ):
    cap = cv2.VideoCapture( '../imgs/green_screen_360.mp4' )
    _, file_name = os.path.split( img_path )
    drop_ext = file_name.split('.')[0]

    bg = cv2.imread( img_path )
    bg = cv2.resize(bg, (640, 360))

    l_green = np.array([0, 0, 45])
    u_green = np.array([177, 216, 255])

```

```

frame_width = int(cap.get(3))
frame_height = int(cap.get(4))
out = cv2.VideoWriter( f'./processed_files/{drop_ext}.mp4',
                      0x7634706d,
                      20.0,
                      (frame_width, frame_height) )

while True:
    ret, frame = cap.read()
    if not ret:
        break

    if cv2.waitKey(25) & 0xFF == ord('q'):
        break
    #frame = cv2.resize(frame, (640, 360))
    hsv = cv2.cvtColor( frame, cv2.COLOR_BGR2HSV )
    mask = cv2.inRange(hsv, l_green, u_green)
    frame[mask == 0] = bg[mask == 0]
    out.write(frame)
    #cv2.imshow("video2", frame)
cap.release()
out.release()

```

6.9. Генерация текста программы. DeepSeek Coder

DeepSeek Coder (7B) – это серия предобученных языковых моделей для кода, созданных компанией DeepSeek. Крупнейшая модель этой серии - DeepSeek Coder 33B, основанная на архитектуре трансформера и содержащая 7 миллиардов параметров. DeepSeek Coder предлагает модели разных размеров: 1.3B, 5.7B, 6.7B, 7B и 33B параметров [84].

Модели DeepSeek Coder предобучены на 2 триллионах токенов из более чем 80 языков программирования, включая 87% кода и 13% естественного языка. Они используют окно размером 16К токенов, что позволяет работать на уровне целых репозиторий кода.

В качестве альтернатив могут быть такие открытые языковые модели для кода, как GPT-3, Codex, CodeLLama, AlphaCode и другие.

Алгоритм работы бота

1. Пользователь вводит команду `/start` бот отправляет приветственное сообщение

2. Пользователь отправляет сообщение с запросом, бот генерирует ответ и отправляет в чат с пользователем

Таблица используемых функций

Функция	Описание
<code>os.getenv(key, default=None)</code>	Возвращает значение переменной окружения <code>'key'</code> , если она существует, иначе возвращает <code>'default'</code> (по умолчанию <code>'None'</code>). <ul style="list-style-type: none"> - <code>'key'</code>: Имя переменной окружения, значение которой нужно получить. - <code>'default'</code>: Значение по умолчанию, которое будет возвращено, если переменная окружения не найдена.
<code>AutoModelForCausalLM.from_pretrained(pretrained_model_name_or_path, *model_args, **kwargs)</code>	Загружает предобученную модель для задачи Causal Language Modeling из указанного пути или названия модели. <ul style="list-style-type: none"> - <code>'pretrained_model_name_or_path'</code>: Путь к предобученной модели или ее название. - <code>*model_args</code>: Позиционные аргументы, передаваемые в конструктор модели. - <code>**kwargs</code>: Именованные аргументы, передаваемые в конструктор модели. В данном случае используется <code>'torch_dtype=torch.float32'</code> для указания типа данных тензоров.
<code>AutoTokenizer.from_pretrained(pretrained_model_name_or_path, *inputs, **kwargs)</code>	Загружает предобученный токенизатор из указанного пути или названия модели. <ul style="list-style-type: none"> - <code>'pretrained_model_name_or_path'</code>: Путь к предобученному токенизатору или его название. - <code>*inputs</code>: Позиционные аргументы, передаваемые в конструктор токенизатора. - <code>**kwargs</code>: Именованные аргументы, передаваемые в конструктор токенизатора.
<code>tokenizer.encode(text, return_tensors=None, **kwargs)</code>	Кодирует текст в последовательность токенов, возвращает список индексов токенов или тензор, если указан параметр <code>'return_tensors'</code> . <ul style="list-style-type: none"> - <code>'text'</code>: Текст, который нужно закодировать в последовательность токенов. - <code>'return_tensors'</code>: Тип возвращаемого значения. Если равно <code>'None'</code>, возвращается список индексов токенов. Если равно <code>'pt'</code>, возвращается тензор PyTorch.

	<ul style="list-style-type: none"> - <code>**kwargs</code>: Дополнительные именованные аргументы, передаваемые в метод кодирования.
<code>torch.ones_like(input, dtype=None, layout=None, device=None, requires_grad=False)</code>	<p>Возвращает тензор, заполненный единицами, с такой же формой, типом данных и устройством, как у входного тензора <code>input</code>.</p> <ul style="list-style-type: none"> - <code>input</code>: Входной тензор, форму и тип данных которого нужно скопировать. - <code>dtype</code>: Тип данных возвращаемого тензора. По умолчанию равно типу данных входного тензора. - <code>layout</code>: Размещение возвращаемого тензора. По умолчанию равно размещению входного тензора. - <code>device</code>: Устройство, на котором будет создан возвращаемый тензор. По умолчанию равно устройству входного тензора. - <code>requires_grad</code>: Флаг, указывающий, требуется ли вычисление градиента для возвращаемого тензора. По умолчанию равно <code>False</code>.
<code>model.generate(*args, **kwargs)</code>	<p>Генерирует последовательность токенов на основе входных данных и параметров генерации.</p> <ul style="list-style-type: none"> - <code>*args</code>: Позиционные аргументы, передаваемые в метод генерации. - <code>**kwargs</code>: Именованные аргументы, передаваемые в метод генерации. Они включают в себя параметры генерации, такие как <code>max_new_tokens</code>, <code>do_sample</code>, <code>num_beams</code>, <code>repetition_penalty</code>, <code>eos_token_id</code>, <code>pad_token_id</code> и другие.
<code>tokenizer.decode(token_ids, skip_special_tokens=False, clean_up_tokenization_spaces=True)</code>	<p>Декодирует последовательность индексов токенов в текст, опционально пропуская специальные токены и очищая пробелы.</p> <ul style="list-style-type: none"> - <code>token_ids</code>: Последовательность индексов токенов, которую нужно декодировать. - <code>skip_special_tokens</code>: Флаг, указывающий, нужно ли пропускать специальные токены при декодировании. По умолчанию равно <code>False</code>. - <code>clean_up_tokenization_spaces</code>: Флаг, указывающий, нужно ли очищать пробелы при декодировании. По умолчанию равно <code>True</code>.
<code>telebot.TeleBot(token)</code>	<p>Создает экземпляр бота Telegram с указанным токеном.</p>
<code>bot.message_handler(commands=['start'])</code>	<p>Декоратор для обработки команды <code>/start</code>.</p> <ul style="list-style-type: none"> - <code>commands</code>: Список команд, на которые будет

	реagirовать обработчик. В данном случае это команда <code>/start`</code> .
<code>bot.reply_to(message, text, **kwargs)</code>	Отправляет ответное сообщение на полученное сообщение <code>message`</code> с текстом <code>text`</code> . - <code>message`</code> : Объект сообщения, на которое нужно ответить. - <code>text`</code> : Текст ответного сообщения. - <code>**kwargs`</code> : Дополнительные именованные аргументы, передаваемые в метод отправки сообщения.
<code>bot.message_handler(func=lambda message: True)</code>	Декоратор для обработки всех входящих сообщений. - <code>func`</code> : Функция-фильтр, которая определяет, должно ли сообщение быть обработано данным обработчиком. В данном случае <code>lambda message: True`</code> означает, что все сообщения будут обрабатываться.
<code>bot.polling()</code>	Запускает бесконечную обработку сообщений от пользователя.

Скриншоты работы бота

AI Poly бот

Да /start Сегодня ✓ 23:30

AI Poly 23:30

Да /start
Привет! Я DeepSeek Coder

Да напиши bubble sort на python ✓ 23:31

AI Poly 23:32

Да напиши bubble sort на python

Bubble Sort это простой алгоритм сортировки, который работает по принципу внутреннего сравнения двух соседних элементов и обмена их местами. Если они находятся в неправильном порядке, то происходит перестановка.

Вот пример реализации Bubble Sort на Python:

```
python
def bubble_sort(arr):
    n = len(arr)

    # Перебираем каждый элемент массива
    for i in range(n-1):
        # Последние i элементов уже будут отсортированными
        for j in range(0, n-i-1):
            # Если текущий элемент больше следующего, меняем
            их местами
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]
    return arr
```

Также вы можете использовать этот алгоритм для сортировки списка в Python, например:

```
python
arr = [64, 34, 25, 12, 22, 11, 90]
print("Sorted array is:", bubble_sort(arr))
```



AI Poly
Бот



```

        if arr[j] > arr[j+1]:
            arr[j], arr[j+1] = arr[j+1], arr[j]
    return arr

```

Также вы можете использовать этот алгоритм для сортировки списка в Python, например:

```

python
arr = [64, 34, 25, 12, 22, 11, 90]
print("Sorted array is:", bubble_sort(arr))

```



Да
теперь напиши сортировку слиянием на с++

23:34



AI Poly

23:35

Да
теперь напиши сортировку слиянием на с++

Вот пример реализации алгоритма "Merge Sort" на C++:

```

cpp
#include<iostream>
using namespace std;
void merge(int arr[], int l, int m, int r) {
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;

    /* Создать временные массивы */
    int L[n1], R[n2];

    /* Копировать данные в временные массивы L[] и R[] */
    for (i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[m + 1 + j];

    /* Объединить временные массивы обратно в arr[l..r]*/
    i = 0;
    j = 0;
    k = l;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        } else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }

    /* Копировать оставшиеся элементы L[] (если таковые
    имеются) */
    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }

    /* Копировать оставшиеся элементы R[] (если таковые
    имеются) */
    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}

void mergeSort(int arr[], int l, int r) {
    if (l < r) {
        int m = l+(r-1)/2;
        mergeSort(arr, l, m);
        mergeSort(arr, m+1, r);
        merge(arr, l, m, r);
    }
}

```

Для использования этой функции вы можете написать код следующим образом:

```

cpp
int main() {
    int arr[] = {12, 11, 13, 5, 6, 7};
    int size = sizeof(arr) / sizeof(arr[0]);

    cout << "Given array is \n";
    for (int i=0; i < size; i++)
        cout << arr[i] << " ";
    cout << "\n";

    mergeSort(arr, 0, size - 1);

    cout << "\nSorted array is \n";
    for (int i=0; i < size; i++)
        cout << arr[i] << " ";
    return 0;
}

```

Текст программы

```
import os
from typing import Iterator

import torch
from transformers import AutoModelForCausalLM, AutoTokenizer
import telebot

TELEGRAM_BOT_TOKEN = ""
MAX_MAX_NEW_TOKENS = 2048
DEFAULT_MAX_NEW_TOKENS = 1024

MAX_INPUT_TOKEN_LENGTH = int(os.getenv("MAX_INPUT_TOKEN_LENGTH", "4096"))

model_id = "deepseek-ai/deepseek-coder-1.3b-instruct"
model = AutoModelForCausalLM.from_pretrained(model_id,
torch_dtype=torch.float32)
tokenizer = AutoTokenizer.from_pretrained(model_id)

def generate(
    message: str,
    max_new_tokens: int = 1024,
    temperature: float = 0.6,
    top_p: float = 0.9,
    top_k: int = 50,
    repetition_penalty: float = 1,
) -> str:
    input_ids = tokenizer.encode(message, return_tensors="pt")
    attention_mask = torch.ones_like(input_ids)
    pad_token_id = tokenizer.eos_token_id

    if input_ids.shape[1] > MAX_INPUT_TOKEN_LENGTH:
        input_ids = input_ids[:, -MAX_INPUT_TOKEN_LENGTH:]
        attention_mask = attention_mask[:, -MAX_INPUT_TOKEN_LENGTH:]

    outputs = model.generate(
        input_ids=input_ids,
        attention_mask=attention_mask,
        max_new_tokens=max_new_tokens,
        do_sample=False,
        num_beams=1,
        repetition_penalty=repetition_penalty,
        eos_token_id=tokenizer.eos_token_id,
        pad_token_id=pad_token_id
    )

    response = tokenizer.decode(outputs[0], skip_special_tokens=True)
    return response
```

```

bot = telebot.TeleBot(TELEGRAM_BOT_TOKEN)

@bot.message_handler(commands=['start'])
def start(message):
    print(f"Received /start command from user: {message.from_user.id}")
    bot.reply_to(message, "Привет! Я DeepSeek Coder")

@bot.message_handler(func=lambda message: True)
def handle_message(message):
    print(f"Received message from user: {message.from_user.id}")
    user_message = message.text
    print(f"User message: {user_message}")
    print(f"Generating response... {response}")
    response = generate(user_message)
    print(f"Generated response: {response}")
    bot.reply_to(message, response)

def main():
    print("Starting the bot...")
    bot.polling()

if __name__ == "__main__":
    main()

```

6.10. Распознавание английского текста на картинке. Pytesseract

Telegram-bot получает на вход фото с английским текстом и выводит распознанный с изображения текст пользователю. Если боту не удастся распознать какой-либо текст, он сообщает об этом пользователю.

Использованные библиотеки:

- `asyncio` – библиотека для написания асинхронного кода, не связанного с блокирующими вызовами
- `pytesseract` – нейронная сеть для распознавания текстов
- `aiogram` – библиотека для создания и конфигурирования Telegram ботов на языке Python
- `PIL` – для открытия изображения и последующей передачи нейронной сети

Алгоритм программы

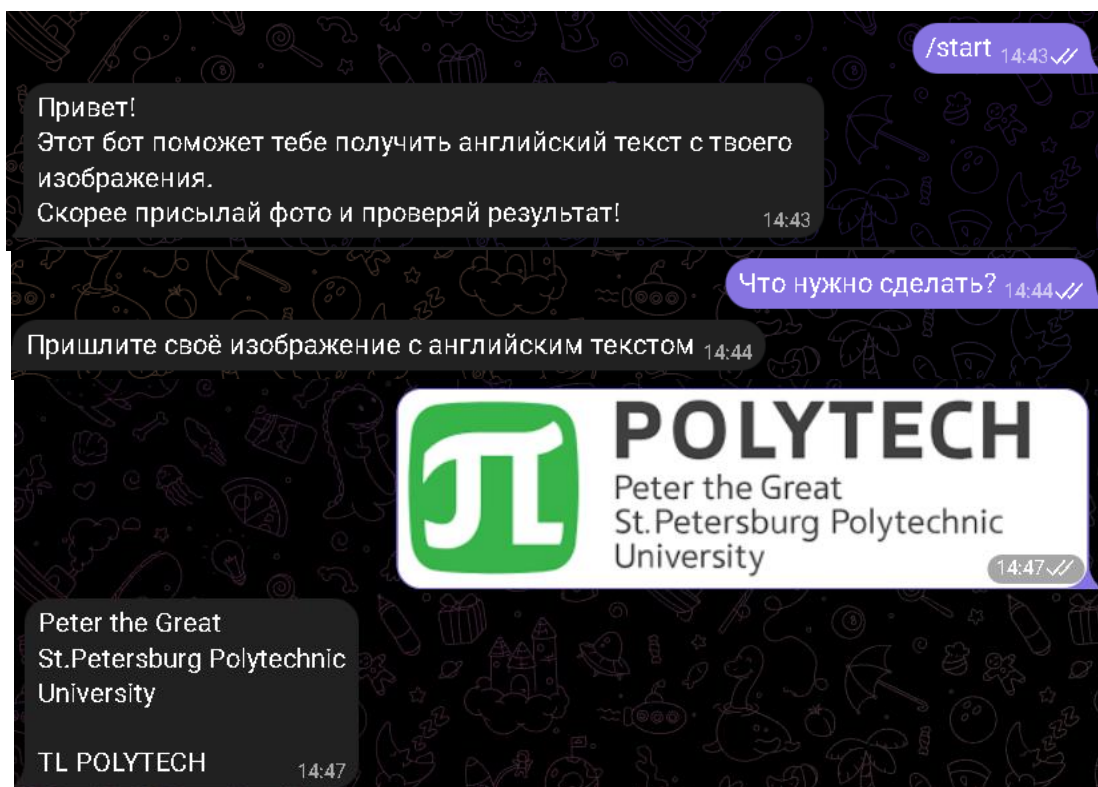
5. Пользователь отправляет команду `/start`, после которой бот объясняет пользователю, что нужно сделать

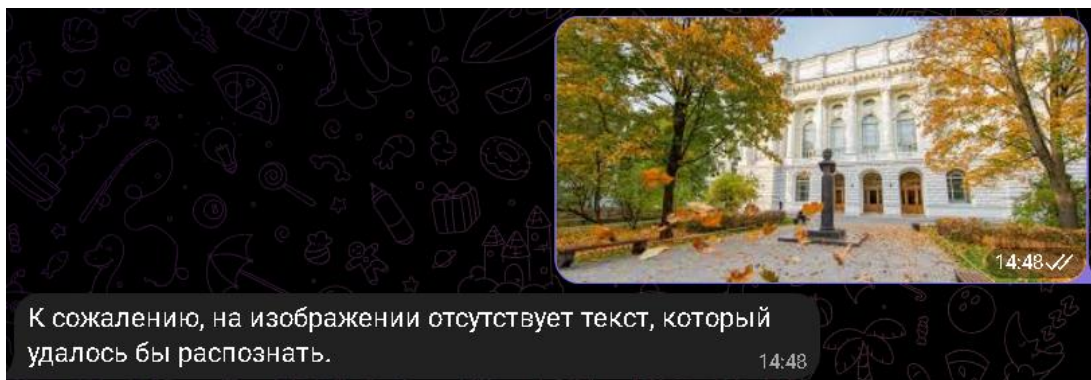
6. Изображение сохраняется на компьютере с которого была запущена программа
7. Сохраненное изображение передается нейронной сети, и она возвращает распознанный текст
8. Если распознанного текста нет, то пользователю выводится сообщение: “К сожалению, на изображении отсутствует текст, который удалось бы распознать.”
9. Если текст удалось распознать, то он выводится пользователю
10. Если пользователь отправит любое текстовое сообщение, то бот попросит пользователя прислать изображение

Описание функций

Название функции	Описание функции
start_cmd	Вывод сообщения о том, как пользоваться ботом. Вызывается при отправке /start
help_cmd	Вывод сообщения, чтобы пользователь отправил фото. Вызывается при отправке любого текста
handle_photo_message	Получение изображения, сохранение изображения, передача изображения нейронной сети и получение ответа Вызывается при отправке фото

Скриншоты работы бота





Листинг программы

```
import asyncio
import pytesseract
from aiogram import Bot, Dispatcher, types, F
from aiogram.filters import CommandStart
from PIL import Image

bot = Bot(token="BOT_TOKEN")
dp = Dispatcher()
img_count = 0

@dp.message(CommandStart())
async def start_cmd(message: types.Message):
    await message.answer("""
    Привет!
    Этот бот поможет тебе получить английский текст с твоего изображения.
    Скорее присылай фото и проверяй результат!
    """)

@dp.message(F.photo)
async def handle_photo_message(message: types.Message):
    photo = message.photo[-1]
    global img_count
    file_id = photo.file_id
    file_path = await bot.get_file(file_id)
    img_count += 1
    image_url = file_path.file_path
    image_name = "image" + str(img_count) + ".jpg"
    await bot.download_file(image_url, image_name)
    image = Image.open(image_name)
    pytesseract.pytesseract.tesseract_cmd = r'C:\Program Files\Tesseract-
OCR\tesseract.exe'
    text = pytesseract.image_to_string(image, lang="eng")
    if text == '':
        text = 'К сожалению, на изображении отсутствует текст, который
    удалось бы распознать.'
    await message.answer(text)
```

```
@dp.message()
async def help_cmd(message: types.Message):
    await message.answer('Пришлите своё изображение с английским
    текстом')

async def main():
    await bot.delete_webhook(drop_pending_updates=False)
    await dp.start_polling(bot)

asyncio.run(main())
```

6.11. Определение возраста и пола человека на изображении или в видеопотоке. Caffe

Одной из важных задач компьютерного зрения является определение возраста и пола человека на изображении или в видеопотоке [85]. Точное определение возраста и пола имеет множество приложений, включая маркетинг, безопасность и медицину. В данной работе был разработан алгоритм на основе применения глубоких нейронных сетей (DNN, Caffe), использующий библиотеку OpenCV для определения возраста и пола на изображениях.

Caffe - это широко используемая библиотека машинного зрения, является открытым программным обеспечением [86]. Написана на языке C++, и поддерживает интерфейс на языке Python. Название Caffe произошло от сокращения «Convolution Architecture For Feature Extraction» (Свёрточная архитектура для извлечения признаков).

Алгоритм работы программы

Когда пользователь отправляет фотографию, бот сохраняет ее, выполняет анализ с использованием функции `getAgeAndGender()`, которая определяет возраст и пол лиц на фотографии. Затем бот отправляет пользователю обработанную фотографию с визуальными метками возраста и пола для каждого обнаруженного лица, а также выводит результаты в виде текстовой подписи к фотографии.

Программа `getAgeAndGender` начинается с загрузки моделей для обнаружения лиц, определения пола и определения возраста. Затем программа загружает изображение и копирует его для последующей обработки. С помощью модели для обнаружения лиц программа находит все лица на изображении с помощью нейронной сети и возвращает их координаты в виде прямоугольников. Далее для каждого обнаруженного лица извлекается соответствующая область изображения. С использованием моделей для определения пола и возраста, программа предсказывает пол и возраст каждого лица и выводит результаты.

Визуальные метки с полом и возрастом добавляются на изображение для каждого обнаруженного лица. Наконец, программа сохраняет измененное изображение с визуальными метками.

Функции из библиотеки OpenCV

<code>cv2.imread(input_path)</code>	Загружает изображение из указанного пути
<code>cv2.dnn.readNetFromCaffe(proto, model)</code>	Загружает модель из файла <code>prototxt</code> и <code>model</code>
<code>cv2.dnn.blobFromImage(image, scalefactor, size, mean, swapRB)</code>	Создает BLOB
<code>cv2.rectangle(image, start_point, end_point, color, thickness)</code>	Рисует прямоугольник на изображении.
<code>cv2.putText(image, text, org, font, font_scale, color, thickness)</code>	Выводит текст на изображение.

Использованные модели

Файлы `prototxt` и `model` являются компонентами модели глубокой нейронной сети. Файл `prototxt` содержит архитектуру сети, определяя слои и их параметры, такие как размеры и функции активации. Файл `model` содержит веса нейронной сети, полученные в результате обучения. В данной работе были использованы следующие `prototxt` и `model`: файлы

- `face.prototxt` и `face_net.caffemodel`: модель для обнаружения лиц на изображении.

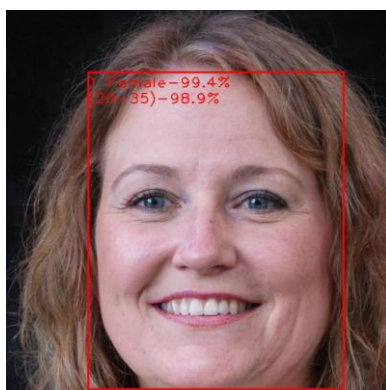
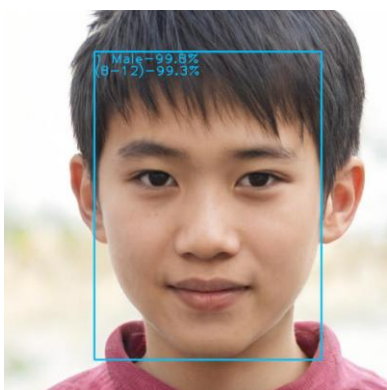
- gender.prototxt и gender_net.caffemodel: модель для определения пола.
- age.prototxt и age_net.caffemodel: модель для определения возраста.

Работа программы

Интерфейс



Примеры работы



Текст программы

```
import cv2
import numpy as np
import os
```

```

FACE_PROTO = "weights/face.prototxt"
FACE_MODEL = "weights/face_net.caffemodel"
GENDER_MODEL = 'weights/gender.prototxt'
GENDER_PROTO = 'weights/gender_net.caffemodel'
AGE_MODEL = 'weights/age.prototxt'
AGE_PROTO = 'weights/age_net.caffemodel'

MODEL_MEAN_VALUES = (78.4263377603, 87.7689143744, 114.895847746)
GENDER_LIST = ['Male', 'Female']

AGE_INTERVALS = ['(0-2)', '(4-10)', '(8-12)', '(15-20)', '(20-35)', '(35-45)', '(45-60)', '(60-100)']

faceNet = cv2.dnn.readNetFromCaffe(FACE_PROTO, FACE_MODEL)
ageNet = cv2.dnn.readNetFromCaffe(AGE_MODEL, AGE_PROTO)
genderNet = cv2.dnn.readNetFromCaffe(GENDER_MODEL, GENDER_PROTO)

def getFaces(frame, confidenceLimit=0.5):
    blob = cv2.dnn.blobFromImage(frame, 1.0, (300, 300), (104, 177.0, 123.0))
    faceNet.setInput(blob)
    output = np.squeeze(faceNet.forward())
    faces = []
    for i in range(output.shape[0]):
        confidence = output[i, 2]
        if confidence > confidenceLimit:
            faces.append(getFaceBox(frame, output[i]))
    return faces

def getFaceBox(frame, output):
    height = frame.shape[0]
    width = frame.shape[1]
    box = output[3:7] * np.array([width, height, width, height])
    start_x, start_y, end_x, end_y = box.astype(int)

    start_x -= 10
    start_y -= 10
    end_x += 10
    end_y += 10

    for i in (start_x, start_y):
        if i < 0:
            i = 5
    if end_x > frame.shape[1]:
        end_x = frame.shape[1] - 5
    if end_y > frame.shape[0]:
        end_y = frame.shape[0] - 5
    return (start_x, start_y, end_x, end_y)

```

```

def getGender(face_img):
    blob = cv2.dnn.blobFromImage(
        image=face_img, scalefactor=1.0, size=(227, 227),
        mean=MODEL_MEAN_VALUES, swapRB=False, crop=False
    )
    genderNet.setInput(blob)
    genderOutput = genderNet.forward()
    i = genderOutput[0].argmax()
    gender = GENDER_LIST[i]
    genderConfidence = genderOutput[0][i]
    return gender, genderConfidence

def getAge(face_img):
    blob = cv2.dnn.blobFromImage(
        image=face_img, scalefactor=1.0, size=(227, 227),
        mean=MODEL_MEAN_VALUES, swapRB=False
    )
    ageNet.setInput(blob)
    ageOutput = ageNet.forward()
    i = ageOutput[0].argmax()
    age = AGE_INTERVALS[i]
    ageConfidence = ageOutput[0][i]
    return age, ageConfidence

def makeFrame(genderMsg, ageMsg, frame, face, gender, n):
    roi_width = face[2] - face[0]
    reference_width = 40

    scaleFactor = int(roi_width / reference_width)
    boxThickness = int(roi_width / reference_width / 5) + 1
    (start_x, start_y, end_x, end_y) = face
    boxColour = (255, 191, 0) if gender == "Male" else (0, 0, 255)
    cv2.rectangle(frame, (start_x, start_y), (end_x, end_y), boxColour,
    boxThickness)

    fontScale = scaleFactor / 6
    fontBoldness = int(scaleFactor / 6)

    msg1 = f"{n+1} {genderMsg}"
    msg2 = ageMsg
    cv2.putText(frame, msg1, (start_x, start_y + 10*boxThickness),
    cv2.FONT_HERSHEY_PLAIN, fontScale, boxColour, fontBoldness)
    cv2.putText(frame, msg2, (start_x, start_y + 10*boxThickness*2),
    cv2.FONT_HERSHEY_PLAIN, fontScale, boxColour, fontBoldness)

    return frame
def getAgeAndGender(input_path: str):
    out = ""
    img = cv2.imread(input_path)

```

```

frame = img.copy()
faces = getFaces(frame)
for n, face in enumerate(faces):
    (start_x, start_y, end_x, end_y) = face
    faceImg = frame[start_y: end_y, start_x: end_x]
    age, ageConfidence = getAge(faceImg)
    gender, genderConfidence = getGender(faceImg)

    genderMsg = f"{gender}-{genderConfidence*100:.1f}%"
    ageMsg = f"{age}-{ageConfidence*100:.1f}%"
    out+=f"{n+1}. {genderMsg} {ageMsg} \n"
    makeFrame(genderMsg, ageMsg, frame, face, gender, n)

filename, extension = os.path.splitext(input_path)
output_filename = filename + "Out" + extension
cv2.imwrite(output_filename,frame)
return out

if __name__ == "__main__":

    import sys
    input_path = sys.argv[1]
    getAgeAndGender(input_path)
import asyncio
import logging
import sys
import os
from aiogram import Bot, Dispatcher, types
from aiogram.types import ReplyKeyboardMarkup, ReplyKeyboardRemove,
KeyboardButton
from age_gender_cv import getAgeAndGender

file = open('token', 'r')
TOKEN = file.read()

bot = Bot(token=TOKEN)
dp = Dispatcher(bot=bot)
logging.basicConfig(filename='bot.log', level=logging.INFO)

@dp.message_handler(commands=['start'])
async def start_handler(message: types.Message):
    user_name = message.from_user.full_name
    user_id = message.from_user.id
    print(f'\\start {user_name} {user_id}')
    await message.reply(f"Hello {user_name}!")

@dp.message_handler(content_types=types.ContentTypes.PHOTO)
async def process_photos(message: types.Message):
    user_name = message.from_user.full_name

```

```

user_id = message.from_user.id
print(f'pic {user_name} {user_id}')
f = message.photo[-1]
file_id = f.file_id
file = await bot.get_file(file_id)
file_path = file.file_path
file_name = file.file_unique_id
await bot.download_file(file_path, "pic/"+file_name+".jpeg")
print("downloaded")

out = getAgeAndGender("pic/"+file_name+".jpeg")
if out == "":
    await message.reply("Nobody, was found on this photo, sorry")
else:
    photo_file = types.InputFile("pic/"+file_name+"Out.jpeg")
    await message.reply_photo( photo_file,caption = f"{out}")
    os.remove("pic/"+file_name+"Out.jpeg")
    os.remove("pic/"+file_name+".jpeg")

if __name__ == "__main__":
    asyncio.run(dp.start_polling())

```

6.12. Перевод аудио файлов и голосовых сообщений в текст. Whisper

В Telegram создается Telegram-бот, который может автоматически расшифровывать аудиосообщения, отправленные пользователями, как в личном чате, так и в группе. Бот использует модель whisper-medium от OpenAI для распознавания речи (Automatic Speech Recognition, ASR) и возвращает текст транскрипции пользователю. Модель находится в открытом доступе на сайте <https://huggingface.co/> . В ходе работы программы модель скачивается на компьютер и далее модель работает локально, на компьютере, где запускается программа. Плюсы такого подхода в том, что нет необходимости оплачивать доступ к AI. Данная модель на достаточно высоком уровне работает с русским языком. Whisper - это предварительно обученная модель для автоматического распознавания речи (ASR) и перевода речи. Модели Whisper, обработанные на 680 тыс. часов работы с маркированными данными, демонстрируют высокую способность к обобщению для многих наборов

данных и предметных областей без необходимости тонкой настройки. Исходный репозиторий кода можно найти здесь: <https://github.com/openai/whisper> .

Необходимые библиотеки:

os: Для работы с файловой системой.

torch: Для работы с PyTorch, используется для загрузки модели.

transformers: Для загрузки и использования модели whisper-medium.

telegram: Для создания Telegram-бота.

moviepy: Для конвертации аудиофайлов в формат WAV.

Алгоритм работы

1. Настройка модели ASR. Устанавливается устройство для выполнения вычислений: `device = "cpu"`. Выбирается модель для распознавания речи whisper-medium.

2. Загрузка модели и процессора `model = AutoModelForSpeechSeq2Seq.from_pretrained;`
`processor = AutoProcessor.from_pretrained(model_id).`

3. Создается пайплайн для автоматического распознавания речи `pipe = pipeline.`

4. Создается функция для транскрибирования аудиофайла с использованием пайплайна `def transcribe_audio(audio_file_path).`

5. Создайте асинхронную функцию для обработки команды /start: в Telegram.

6. Создается асинхронная функция для обработки аудиофайлов и голосовых сообщений `async def handle_audio(update: Update, context: CallbackContext).`

7. Создается и настраивается приложение Telegram-бота `def main().`

Пример работы бота



Используемые функции

Функция	Действие функции
<code>transcribe_audio(audio_file_path)</code>	Транскрибирует аудиофайл с использованием модели ASR.
<code>start(update: Update, context: CallbackContext) -> None</code>	Обрабатывает команду <code>/start</code> , приветствуя пользователя и объясняя функциональность бота.
<code>handle_audio(update: Update, context: CallbackContext) -> None</code>	Обрабатывает аудиофайлы и голосовые сообщения, отправленные пользователем.
<code>AutoModelForSpeechSeq2Seq.from_pretrained(model_id, torch_dtype=torch_dtype, use_safetensors=True)</code>	Загружает предварительно обученную модель распознавания речи.
<code>AutoProcessor.from_pretrained(model_id)</code>	Загружает процессор для обработки аудиоданных.
<code>pipeline("automatic-speech-recognition", model=model, tokenizer=processor.tokenizer,</code>	Создает пайплайн для автоматического распознавания речи.

<code>feature_extractor=processor.feature_extractor, torch_dtype=torch_dtype, device=device)</code>	
<code>context.bot.get_file(file_id)</code>	Получает информацию о файле по его идентификатору.
<code>file_info.download_to_drive(file_path)</code>	Скачивает файл по указанному пути.
<code>AudioFileClip(file_path)</code>	Создает объект аудиофайла для обработки с помощью moviepy.
<code>audio.write_audiofile(converted_file_path)</code>	Конвертирует аудиофайл в формат WAV
<code>Application.builder().token('YOUR_TELEGRAM_BOT_TOKEN').http_version('1.1').get_updates_http_version('1.1').build()</code>	Создает объект приложения для Telegram-бота.

Текст программы

```
import os
import torch
from transformers import AutoModelForSpeechSeq2Seq, AutoProcessor, pipeline
from telegram import Update
from telegram.ext import Application, CommandHandler, MessageHandler, filters, CallbackContext
from moviepy import AudioFileClip

# Устанавливаем устройство на CPU
device = "cpu"
torch_dtype = torch.float32

# Выбираем среднюю модель
model_id = "openai/whisper-medium"

# Загружаем модель и процессор
model = AutoModelForSpeechSeq2Seq.from_pretrained(
    model_id, torch_dtype=torch_dtype, use_safetensors=True
)
model.to(device)
processor = AutoProcessor.from_pretrained(model_id)

# Создаем пайплайн для автоматического распознавания речи
pipe = pipeline(
    "automatic-speech-recognition",
    model=model,
    tokenizer=processor.tokenizer,
    feature_extractor=processor.feature_extractor,
    torch_dtype=torch_dtype,
```

```

        device=device,
    )
# Функция для транскрибирования аудиофайла
def transcribe_audio(audio_file_path):
    result = pipe(audio_file_path, return_timestamps=True)
    return result["text"]

# Функция для обработки команды /start
async def start(update: Update, context: CallbackContext) -> None:
    await update.message.reply_text('Привет! Я бот для расшифровки
аудиосообщений. Отправь мне аудиофайл (.mp3, .ogg или .oga), и я
расшифрую его.')

# Функция для обработки аудиофайлов
async def handle_audio(update: Update, context: CallbackContext) -> None:
    try:
        # Определяем, аудиофайл это или голосовое сообщение
        if update.message.audio:
            file_info = await
context.bot.get_file(update.message.audio.file_id)
            elif update.message.voice:
                file_info = await
context.bot.get_file(update.message.voice.file_id)
            else:
                await update.message.reply_text("Пожалуйста, отправьте
аудиофайл или голосовое сообщение.")
                return
            print("Получено аудиосообщение")

            # Скачиваем файл
            os.makedirs("downloads", exist_ok=True)
            file_path = os.path.join("downloads", f"{file_info.file_id}.oga")
            await file_info.download_to_drive(file_path)
            print(f"Файл скачан: {file_path}")

            # Конвертируем аудио в формат WAV
            converted_file_path = os.path.join("downloads",
f"{file_info.file_id}.wav")
            audio = AudioFileClip(file_path)
            audio.write_audiofile(converted_file_path)
            print(f"Файл конвертирован: {converted_file_path}")

            # Транскрибируем конвертированный аудиофайл
            transcription = transcribe_audio(converted_file_path)
            print(f"Транскрипция: {transcription}")
            await update.message.reply_text(transcription)

            # Удаляем скачанный и конвертированный файлы
            os.remove(file_path)

```

```

        os.remove(converted_file_path)
        print("Файлы удалены")
    except Exception as e:
        print(f"Ошибка при обработке аудио: {e}")
        await update.message.reply_text("Произошла ошибка при обработке
аудиофайла.")
def main() -> None:
    application = Application.builder().token('тут должен быть токен
Telegram-
бота').http_version('1.1').get_updates_http_version('1.1').build()
    application.add_handler(CommandHandler("start", start))
    application.add_handler(MessageHandler(filters.AUDIO | filters.VOICE,
handle_audio))
    application.run_polling()
if __name__ == '__main__':
    main()

```

6.13. Озвучивание текста. Bark

Telegram-бот получает на вход текстовое сообщение, а в ответ присылает пользователю аудиофайл, в котором озвучивает текст. Бот использует предобученную модель нейронной сети Bark с открытым исходным кодом [87].

Алгоритм работы

1. Пользователь запускает бота.
2. Бот отправляет первое сообщение с текстом «Привет, (Имя) (Фамилия)». Бот использует имя и фамилию, которые указаны в личном аккаунте пользователя.
3. Пользователь отправляет команду /speech «текст, который нужно перевести в речь».
4. Бот отправляет пользователю аудиофайл с озвученным текстом.

Описание программы

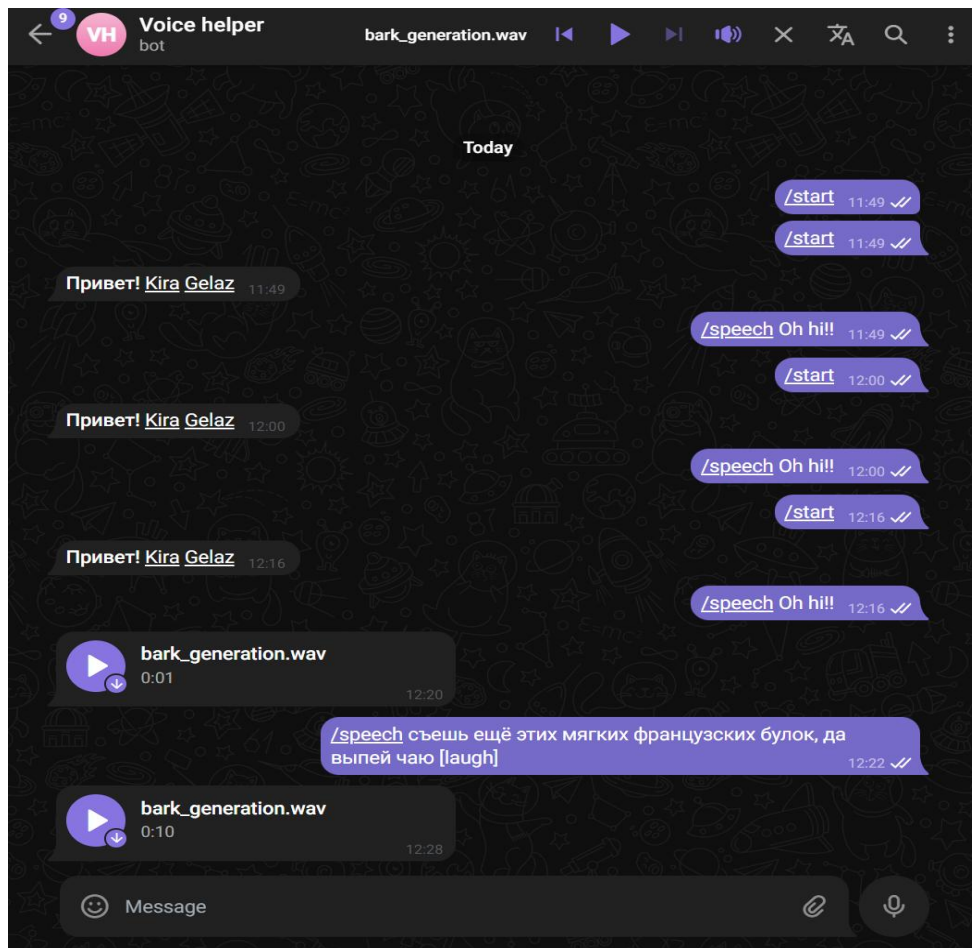
1. Первоначально в программе задается токен телеграмм-бота.
2. В команде (@bot.message_handler(commands=['start'])) бот получает имя и фамилию пользователя, а также отправляет ответное сообщение.
3. В команде (@bot.message_handler(commands=['speech'])) используется функция (text_to_speech(text)), которая обрабатывает текст и сохраняет его в аудиоформате в файл « bark_generation.wav », затем бот отправляет этот файл пользователю.

4. В конце программы описывается бесконечная работа бота (пока существует main)

Используемые функции

Название функции	Краткое описание
TeleBot(TOKEN)	Присваивает токен боту, с которым будет дальнейшая работа
start(message) bot.send_message(message.chat.id, text=answer, parse_mode='html')	Обработка сообщения start Функция для отправки сообщения пользователю (id чата с пользователем, text – отправляемый текст, parse_mode – формат текста)
duck(message)	Обработчик сообщения от пользователя
text_to_speech(text)	Функция, генерирующая аудиофайл при помощи библиотеки bark, сохраняет файл с расширением wav
bot.send_audio(message.chat.id, f)	Функция, отправляющая пользователю аудио (в качестве аргументов получает id чата с пользователем и сам аудиофайл)
bot.polling(non_stop=True)	Функция для непрерывной обработки информации, приходящей с серверов Telegram

Результат выполнения программы



Текст программы

Файл «main.py»

```

from telebot import TeleBot, types
from text_to_speech import text_to_speech
import os

os.environ["SUNO_OFFLOAD_CPU"] = "True"
os.environ["SUNO_USE_SMALL_MODELS"] = "True"

TOKEN = '6831186333:AAE91-GIpW77BH0PauqbpYlnvqyHU0UORuQ'
bot = TeleBot(TOKEN)

@bot.message_handler(commands=['start'])
def start(message):
    answer = f'<b>Привет!</b> <u>{message.from_user.first_name}</u> <u>{message.from_user.last_name}</u>'
    bot.send_message(message.chat.id, text=answer, parse_mode='html')

@bot.message_handler(commands=['speech'])
def duck(message):
    text = ' '.join(message.text.split(' ')[1:])
    text_to_speech(text)
    with open('bark_generation.wav', 'rb') as f:

```

```
bot.send_audio(message.chat.id, f)

if __name__ == '__main__':
    bot.polling(non_stop=True)
```

```
Файл «text_to_speech.py»
from bark import SAMPLE_RATE, generate_audio, preload_models
from scipy.io.wavfile import write as write_wav
from IPython.display import Audio

def text_to_speech(msg):
    audio_array = generate_audio(msg)
    write_wav("bark_generation.wav", SAMPLE_RATE, audio_array)
```

6.14. Переводчик аудио с применением ресурсов Yandex Cloud

Создается Telegram-бот, который позволяет пользователям распознавать речь из аудиофайлов или голосовых сообщений, переводить распознанный текст на другой язык и синтезировать переведенный текст обратно в аудио. Бот поддерживает два языка: русский и иврит. Пользователь может выбрать язык аудиофайла, отправить аудио, и бот выполнит следующие действия:

- Распознавание речи: Преобразует аудио в текст.
- Перевод текста: Переводит распознанный текст на выбранный язык.
- Синтез речи: Преобразует переведенный текст обратно в аудио.

Бот удобен для пользователей, которым нужно быстро перевести аудиосообщения или получить аудиоверсию переведенного текста. Используются ресурсы Yandex Cloud: SpeechKit и Translate.

Сервис Yandex SpeechKit позволяет разработчикам приложений использовать речевые технологии Яндекса: распознавание речи (Speech-to-Text) и синтез речи (Text-to-Speech). Сервис для интеграции алгоритмов Yandex Translate в приложения или веб-проекты для конечных пользователей. Yandex Translate поддерживает более 90 языков и умеет переводить отдельные слова и целые тексты.

Для доступа к SpeechKit и Yandex Translate необходимо использовать API (Application Programming Interface). Доступ к ресурсам Yandex Cloud платный, однако существенно увеличивает скорость работы бота и отсутствует необходимость использования дорогостоящего компьютера.

Необходимые библиотеки:

os: Управление файлами и директориями.

logging: Логирование событий и ошибок.

requests: Отправка HTTP-запросов.

telegram: Инструменты для создания Telegram-ботов.

telegram.ext: Обработка команд, сообщений и состояний в Telegram-ботах.

speechkit: Работа с API Yandex SpeechKit для распознавания и синтеза речи.

speechkit.stt: Дополнительные инструменты для распознавания речи.

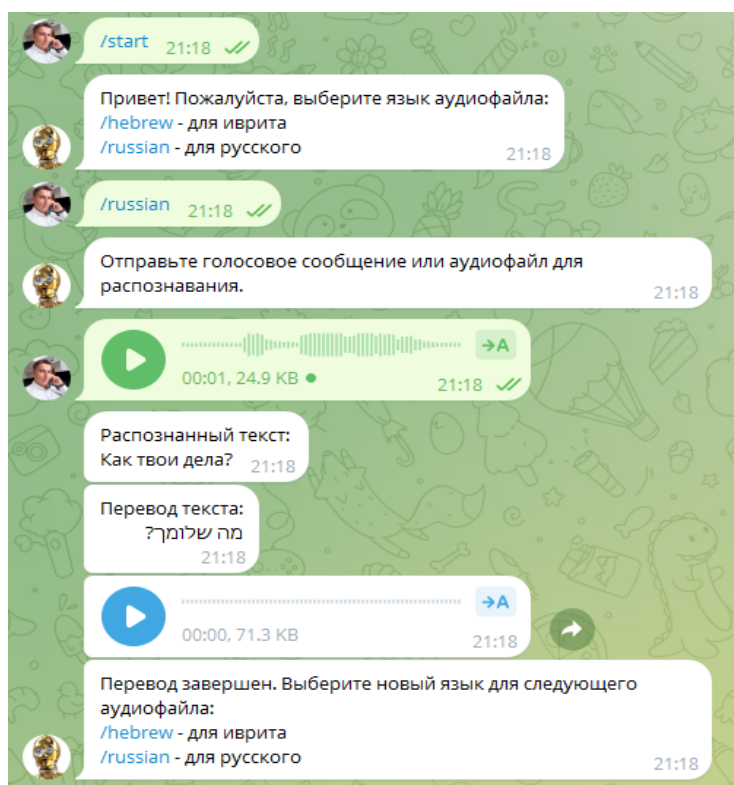
pydub: Конвертация аудиофайлов.

Алгоритм работы

1. Запуск бота: бот запускается и ожидает команд и сообщений от пользователей.
2. Пользователь отправляет команду `/start`: бот приветствует пользователя и предлагает выбрать язык аудиофайла (`/hebrew` для иврита или `/russian` для русского).
3. Пользователь выбирает язык: бот устанавливает выбранный язык и целевой язык для перевода. Далее бот предлагает пользователю отправить аудиофайл или голосовое сообщение.
4. Пользователь отправляет аудиофайл или голосовое сообщение: бот скачивает аудиофайл на компьютер. Далее аудиофайл конвертируется в формат WAV.

5. Распознавание речи: распознается речь из аудиофайла с использованием модели Yandex SpeechKit. Бот отправляет пользователю распознанный текст.
6. Перевод текста: переводится распознанный текст на целевой язык с использованием Yandex Translate API. Бот отправляет пользователю переведенный текст.
7. Синтез речи: синтезируется переведенный текст в аудиофайл с использованием модели Yandex SpeechKit. Бот отправляет пользователю синтезированное аудио.
8. Очистка временных файлов: удаляются скачанные и конвертированные аудиофайлы.
9. Завершение обработки: бот предлагает пользователю выбрать новый язык для следующего аудиофайла. Далее бот возвращается к ожиданию команды выбора языка.

Пример работы бота в Telegram:



Используемые функции

Функция	Действие функции
<code>recognize_text(audio_path, language)</code>	Распознает речь из аудиофайла и возвращает текст.
<code>translate_text(texts, source_language, target_language, folder_id)</code>	Переводит текст с одного языка на другой.
<code>synthesize_text(text, language, export_path)</code>	Обрабатывает аудиофайлы и голосовые сообщения, отправленные пользователем.
<code>start(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int</code>	Обрабатывает команду <code>/start</code> , приветствует пользователя и предлагает выбрать язык аудиофайла.
<code>select_hebrew(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int</code>	Обрабатывает команду <code>/hebrew</code> , устанавливает язык аудиофайла на иврит и предлагает отправить аудио.
<code>select_russian(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int</code>	Обрабатывает команду <code>/russian</code> , устанавливает язык аудиофайла на русский и предлагает отправить аудио .
<code>process_audio(update: Update, context: ContextTypes.DEFAULT_TYPE) -> int</code>	Обрабатывает аудиофайлы и голосовые сообщения, отправленные пользователем.
<code>error_handler(update: Update, context: ContextTypes.DEFAULT_TYPE) -> None</code>	Обрабатывает ошибки, возникающие при работе бота.
<code>download_to_drive</code>	Скачивает файл на компьютер.
<code>reply_voice</code>	Отправляет синтезированный аудиофайл в чат Telegram.

Текст программы

```
import os
import logging
import requests
from telegram import Update, InputFile
from telegram.ext import (
    Application, CommandHandler, MessageHandler, filters,
    ConversationHandler, ContextTypes
```

```

)
from speechkit import model_repository, configure_credentials, creds
from speechkit.stt import AudioProcessingType
from pydub import AudioSegment # For audio conversion

API_KEY = 'Your API key (from Yandex Cloud)'
folder_id = 'Your folder ID (from Yandex Cloud)'
TELEGRAM_API_TOKEN = 'Telegram bot API token'

# States for ConversationHandler
START, SELECT_LANGUAGE, PROCESS_AUDIO = range(3)

# Authentication via API key
configure_credentials(
    yandex_credentials=creds.YandexCredentials(
        api_key=API_KEY
    )
)
# Set up logging
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

# Define the script directory and audio directory
script_dir = os.path.dirname(__file__)
audio_dir = os.path.join(script_dir, 'audio_files')

# Create the audio directory if it doesn't exist
os.makedirs(audio_dir, exist_ok=True)
def recognize_text(audio_path, language):
    try:
        model = model_repository.recognition_model()
        model.model = 'general'
        model.language = language
        model.audio_processing_type = AudioProcessingType.Full
        result = model.transcribe_file(audio_path)
        recognized_text = " ".join([res.normalized_text for res in
result])
        return recognized_text.strip()
    except Exception as e:
        logger.error(f"Error in recognize_text: {e}")
        return ""

def translate_text(texts, source_language, target_language, folder_id):
    try:
        body = {
            "sourceLanguageCode": source_language,
            "targetLanguageCode": target_language,
            "texts": texts,
            "folderId": folder_id,

```

```

    }
    headers = {
        "Content-Type": "application/json",
        "Authorization": f"Api-Key {API_KEY}"
    }
    response = requests.post(
'https://translate.api.cloud.yandex.net/translate/v2/translate',
        json=body,
        headers=headers
    )
    response.raise_for_status()
    return response.json()
except Exception as e:
    logger.error(f"Error in translate_text: {e}")
    return {"translations": [{"text": "Translation error"}]}
def synthesize_text(text, language, export_path):
    try:
        model = model_repository.synthesis_model()
        if language == 'ru':
            model.voice = 'ermil'
            model.role = 'neutral'
        elif language == 'he':
            model.voice = 'naomi'
        logger.info(f"Synthesizing text: {text}, Language: {language},
Voice: {model.voice}, Role: {getattr(model, 'role', 'N/A')}")
        result = model.synthesize(text, raw_format=False)
        result.export(export_path, 'wav')
    except Exception as e:
        logger.error(f"Error in synthesize_text: {e}")

async def start(update: Update, context: ContextTypes.DEFAULT_TYPE) ->
int:
    await update.message.reply_text(
        'Привет! Пожалуйста, выберите язык аудиофайла:\n'
        '/hebrew - для иврита\n'
        '/russian - для русского'
    )
    return SELECT_LANGUAGE

async def select_hebrew(update: Update, context:
ContextTypes.DEFAULT_TYPE) -> int:
    context.user_data['language'] = 'he-HE'
    context.user_data['target_language'] = 'ru'
    await update.message.reply_text('Отправьте голосовое сообщение или
аудиофайл для распознавания.')
    return PROCESS_AUDIO

```

```

async def select_russian(update: Update, context:
ContextTypes.DEFAULT_TYPE) -> int:
    context.user_data['language'] = 'ru-RU'
    context.user_data['target_language'] = 'he'
    await update.message.reply_text('Отправьте голосовое сообщение или
аудиофайл для распознавания.')
    return PROCESS_AUDIO

async def process_audio(update: Update, context:
ContextTypes.DEFAULT_TYPE) -> int:
    try:
        user_id = update.message.from_user.id
        if update.message.voice:
            audio_file = await update.message.voice.get_file()
            audio_format = "ogg"
        elif update.message.audio:
            audio_file = await update.message.audio.get_file()
            audio_format = "mp3"
        else:
            await update.message.reply_text('Пожалуйста, отправьте
голосовое сообщение или аудиофайл.')
            return PROCESS_AUDIO

        # Construct full file paths
        audio_path_ogg = os.path.join(audio_dir,
f'{user_id}_audio.{audio_format}')
        audio_path_wav = os.path.join(audio_dir, f'{user_id}_audio.wav')
        output_path = os.path.join(audio_dir, f'{user_id}_output.wav')

        # Download the audio file
        await audio_file.download_to_drive(audio_path_ogg)
        logger.info(f"File downloaded to {audio_path_ogg}")

        # Convert ogg to wav using pydub
        audio = AudioSegment.from_file(audio_path_ogg,
format=audio_format)
        audio.export(audio_path_wav, format="wav")
        logger.info(f"Audio converted to {audio_path_wav}")

        # Process the audio file (recognition, translation, synthesis)
        language = context.user_data.get('language', 'ru-RU')
        target_language = context.user_data.get('target_language', 'he')

        recognized_text = recognize_text(audio_path_wav, language)
        logger.info(f"Recognized text: {recognized_text}")
        await update.message.reply_text(f'Распознанный
текст:\n{recognized_text}')

```

```

        translation_result = translate_text([recognized_text],
language[:2], target_language, folder_id)
        if 'translations' in translation_result and
len(translation_result['translations']) > 0:
            translated_text =
translation_result['translations'][0]['text']
        else:
            translated_text = "Translation error"
            logger.info(f"Translated text: {translated_text}")
            await update.message.reply_text(f'Перевод
текста:\n{translated_text}')

        synthesize_text(translated_text, target_language, output_path)
        if os.path.exists(output_path):
            with open(output_path, 'rb') as audio_file:
                await
update.message.reply_voice(voice=InputFile(audio_file))
        else:
            await update.message.reply_text('Произошла ошибка при синтезе
аудио.')
```

```

# Remove temporary files
if os.path.exists(audio_path_ogg):
    os.remove(audio_path_ogg)
if os.path.exists(audio_path_wav):
    os.remove(audio_path_wav)
if os.path.exists(output_path):
    os.remove(output_path)
logger.info("Temporary files removed.")

# Prompt to select a new language after processing
await update.message.reply_text(
    'Перевод завершен. Выберите новый язык для следующего
аудиофайла:\n'
    '/hebrew - для иврита\n'
    '/russian - для русского'
)
return SELECT_LANGUAGE
except Exception as e:
    logger.error(f"Error in process_audio: {e}")
    await update.message.reply_text('Произошла ошибка при обработке
аудио.')
```

```

return SELECT_LANGUAGE

def error_handler(update: Update, context: ContextTypes.DEFAULT_TYPE) ->
None:
    logger.error(msg="Exception while handling an update:",
exc_info=context.error)
```

```
def main() -> None:
    application = Application.builder().token(TELEGRAM_API_TOKEN).build()

    # Add handlers
    application.add_handler(CommandHandler('start', start))
    application.add_handler(CommandHandler('hebrew', select_hebrew))
    application.add_handler(CommandHandler('russian', select_russian))
    application.add_handler(MessageHandler(filters.VOICE | filters.AUDIO,
process_audio))

    # Add error handler
    application.add_error_handler(error_handler)

    # Run the bot
    application.run_polling()

if __name__ == '__main__':
    main()
```

Заключение

Монография позволяет познакомиться с большим количеством алгоритмов, которые можно использовать для построения программ обработки мультимедийных данных. Все алгоритмы используют нейронные сети, а также другие библиотеки. Примеры программ написаны на языке Python - одного из самых распространенных языков программирования в областях компьютерного зрения и машинного обучения. Можно просто скопировать текст программ, подключить библиотеки и они заработают. В качестве среды разработки можно использовать PyCharm или другую IDE. Программы разрабатывались с участием студентов высшей школы программной инженерии СПбПУ.

В монографии представлены примеры 100 алгоритмов обработки мультимедийных данных (видео, аудио, тексты), приводятся примеры реализаций telegram-ботов. Монография будет полезна для специалистов в области проектирования и применения видеосистем для обработки сигналов и изображений. Она полезна студентам, проходящим подготовку по направлениям 09.03.01 «Информатика и вычислительная техника», 09.03.04 «Программная инженерия», 12.03.01 «Приборостроение».

Список литературы

1. Лутц М. Программирование на Python. Издательство Символ, 2020. - 992 с.
2. Кеннеди Берман Основы Python для Data Science. СПб.: Питер, 2023. – 272с.
3. Васильев А. Программирование на Python в примерах и задачах. Издательство Бомбора (Эксмо), 2020. - 616 с.
4. Шолле Франсуа Глубокое обучение на Python. СПб.: Питер, 2018. – 400 с.
5. Рашка Себастьян, Мирджалили Вахид Python и машинное обучение. Диалектика, 2020. - 848 с.
6. Содем Ян Эрик Программирование компьютерного зрения на Python. ДМК-Пресс, 2016 312 с.
7. Open Computer Vision Library. URL:[http:// opencv.org /](http://opencv.org/)
8. Kaehler, A., Bradski, G. Learning OpenCV 4: Computer Vision with Python. O'Reilly Media, 2019. – 557 с.
9. Официальная документация для библиотеки FFmpeg. URL:<https://ffmpeg.org/>
- 10.FFmpeg-python documentation. URL: <https://kkroening.github.io/ffmpeg-python/>
- 11.FFmpeg с помощью Python - часть первая. URL:http://rus-linux.net/MyLDP/mm/Hacking_FFmpeg_with_Python_part_1.html
- 12.FFmpeg с помощью Python - часть вторая. URL: http://rus-linux.net/MyLDP/mm/Hacking_FFmpeg_with_Python_part_2.html
- 13.Учебник по ffmpeg и SDL. URL: <http://dranger.com/ffmpeg/>
- 14.Python Imaging Library documentation. URL:<https://pillow.readthedocs.io/en/stable/>
- 15.Молодяков С.А. Применение функций OpenCV в компьютерном зрении (60 примеров на Python) СПб.: Изд-во Политехн. ун-та, 2022.- 296 с. DOI: 10.18720/SPBPU/2/i22-45
- 16.Молодяков С. А. Применение функций FFmpeg в мультимедийных приложениях (100 примеров на Python): монография, СПб. : ПОЛИТЕХ-ПРЕСС, 2023. – 514 с. DOI: 10.18720/SPBPU/2/i23-44

17. Молодяков С.А., Милицын А. В. Алгоритмы работы с мультимедийными данными в telegram-боте (100 примеров на Python). Санкт-Петербург: ПОЛИТЕХ-ПРЕСС.- 2024.- 586 с. ISBN 978-5-7422-8448-2
18. Лысенко А.В., Яременко А.И., Иванов В.М. и др. Применение навигационной системы на основе технологии дополненной реальности в челюстно-лицевой хирургии // Digital Diagnostics. 2024. Т. 5. № 3. С. 450-466.
19. Kiriakov I.M., Molodyakov S.A. Development of Neural Network Module for Autocompletion of Examination Protocol for Modular Medical Information System // IEEE 2024 V International Conference on Neural Networks and Neurotechnologies (NeuroNT) 20-20 June 2024
DOI: 10.1109/NeuroNT62606.2024.10585474
20. Завьялов С.В., Павлов В.А., Фёдоров С.А., Шариати Ф., Первунина Т.М. Автоматическая сегментация и классификация COVID-19 на КТ-изображениях // Вопросы радиоэлектроники. Серия: Техника телевидения. 2023. № 4. С. 76-83.
21. Думаев Р.И., Киряков И.М., Молодяков С.А. Особенности предобработки и сегментации изображений в задаче обнаружения COVID-19 по рентгеновским снимкам // Современная наука: актуальные проблемы теории и практики. Серия: Естественные и Технические Науки. -2022. -№9. -С. 88-95. DOI 10.37882/2223–2966.2022.09.08
22. Dumaev R. I., Molodyakov S. A. Classification and Prediction of Lung Diseases According to Chest Radiography // 2023 IV International Conference on Neural Networks and Neurotechnologies (NeuroNT), Saint Petersburg, Russian Federation, 2023, pp. 48-51, DOI: 10.1109/NeuroNT58640.2023.10175838.
23. Dumaev R. I., Molodyakov S. A., Utkin L. V. A model for explainable malignancy assessment of pulmonary nodules on CT images // Artificial Intelligence and Decision Making, 2024, no. 4, 123–134
DOI: <https://doi.org/10.14357/20718594240410>
24. Смирнова Д.С., Пчицкая Е.И., Чуканов В.С. Применение гистограммы длин

хорд с целью повышения точности распознавания лёгочных узлов на компьютерных томограммах // В сборнике: V Международная конференция по нейронным сетям и нейротехнологиям (NeuroNT'2024). Сборник докладов конференции. Санкт-Петербург, 2024. С. 127-131.

25. Шариати Ф., Павлов В. А., Завьялов С. В. и др. Применение модели внешнего вида текстуры для сегментации легочных узлов при компьютерной томографии грудной клетки // Известия высших учебных заведений России. Радиоэлектроника. – 2022. – Т. 25, № 3. – С. 96-117. – DOI: 10.32603/1993-8985-2022-25-3-96-117.
26. Meldo A., Utkin L., Kovalev M., Kasimov E. The natural language explanation algorithms for the lung cancer computer-aided diagnosis system // Artificial Intelligence in Medicine. – 2020. – Vol. 108. – P. 101952. – DOI: 10.1016/j.artmed.2020.101952.
27. Первичный анализ речевых сигналов. URL: <https://alphacephei.com/ru/lecture1.pdf>
28. Vosk - набор инструментов для распознавания речи. URL: <https://alphacephei.com/vosk/>
29. Antoine Salmona¹, Luc' ia Bouza¹, Julie Delon DeOldify: A Review and Implementation of an Automatic Colorization Method. URL: https://www.ipol.im/pub/art/2022/403/article_lr.pdf
30. Обучающие материалы PyTorch URL: <https://pytorch.org/tutorials/>
31. Удалить фон изображения с помощью ИИ. URL: <https://www.rembg.com/en>
32. U-Net: нейросеть для сегментации изображений. URL: <http://https://neurohive.io/ru/vidy-nejrosetej/u-net-image-segmentation/>
33. Задачи сегментации изображения с помощью нейронной сети U-Net. URL: <http://blog.dataalytica.ru/2018/03/unet.html>
34. Семантическая сегментация на основе архитектуры U-Net и определение расстояния между объектами. URL: <https://habr.com/ru/articles/746842/>.
35. Набор данных MS-Coco. URL: <https://cocodataset.org/#home>.

36. Fully Convolutional Networks for Semantic Segmentation. URL:
<https://arxiv.org/pdf/1411.4038.pdf>
37. Intersection over Union (IoU) for object detection. URL:
<https://pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection>
38. Набор данных о домашних животных Oxford-IIIT. URL:
<https://www.robots.ox.ac.uk/~vgg/data/pets/>
39. Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. URL:
https://proceedings.neurips.cc/paper_files/paper/2015/file/14bfa6bb14875e45bba028a21ed38046-Paper.pdf
40. Официальная документация модели Movenet с описанием и примерами кода.
URL: <https://www.tensorflow.org/hub/tutorials/movenet>
41. Официальная документация по открытой программной библиотеке для машинного обучения Google Tensorflow. URL: <https://www.tensorflow.org>
42. Официальная документация фундаментального python-пакета для научного программирования NumPy. URL: <https://numpy.org/doc/>.
43. Официальная документация модели EAST с описанием работы алгоритма распознавания текста. URL: <https://arxiv.org/pdf/1704.03155>
44. Официальная документация OpenCV по подготовке изображения, установки необходимых параметров использования модели EAST с примерами кода.
URL: https://docs.opencv.org/4.x/d4/d43/tutorial_dnn_text_spotting.html
45. Code Pretraining Models. URL: <https://github.com/microsoft/CodeBERT>
46. Официальная документация библиотеки PyTesseract, предназначенной для оптического распознавания символов. URL:
<https://pytesseract.readthedocs.io/en/latest/>.
47. Руководство пользователя Тессеракта. URL: <https://tesseract-ocr.github.io/tessdoc/>
48. Губенко Н.О., Молодяков С. А., Коликова Т.В. Разработка классификатора

для оптического распознавания музыкальной нотации // Современная наука: актуальные проблемы теории и практики. Серия: Естественные и Технические Науки. -2023. -№3. -С. 51-54. DOI: 10.37882/2223–2966.2023.03.09

49.Официальная документация по Stable Diffusion 3.5. URL: <https://stability.ai/>

50.Документация по применению Xception. URL:

<https://keras.io/api/applications/xception/>

51.Введение в RNN Рекуррентные Нейронные Сети. URL:<https://python-scripts.com/recurrent-neural-network?ysclid=lv0mzuxhmt714541473>

52.Документация и описание DeepLabV3. URL:

<https://www.ikomia.ai/blog/understanding-deeplabv3-image-segmentation>

53. PyTorch - фреймворк глубокого обучения. URL:

<https://habr.com/ru/articles/334380/>

54. DeepFace фреймворк распознавания. URL:<https://github.com/serengil/deepface>

55. Yao X., Puy G., Newson A., Gousseau Y., Hellier P. High Resolution Face Age Editing // arXiv. 2020. URL: <http://arxiv.org/pdf/2005.04410>

56. Dlib: Библиотека инструментов машинного обучения. URL: <http://dlib.net>

57. Документация SpeechRecognition 3.14.0. URL:

<https://pypi.org/project/SpeechRecognition/>

58. Руководство по решениям MediaPipe. URL:

<https://ai.google.dev/edge/mediapipe/solutions/guide?hl=ru>

59. YOLOv9: скачок в развитии технологии обнаружения объектов. URL:

<https://docs.ultralytics.com/ru/models/yolov9/>

60. Документация face-recognition 1.3.0. URL: <https://pypi.org/project/face-recognition/>

61. Документация Stable Video Diffusion. URL: <https://stability.ai/stable-video>

62. Хазика С. Диффузионные модели в ИИ — все, что вам нужно знать. URL:

<https://www.unite.ai/ru/diffusion-models-in-ai-everything-you-need-to-know/>.

63. Документация Sequential model. URL: https://keras.io/guides/sequential_model/

64. Guide to MTCNN in facenet-pytorch. URL:
<https://www.kaggle.com/code/timesler/guide-to-mtcnn-in-facenet-pytorch>
65. Документация Spleeter by Deezer. URL: <https://github.com/deezer/spleeter>;
66. Kobyshev K.S., Molodyakov S.A. An algorithm of test generation from functional specification using Open IE model and clustering. Trudy ISP RAN/Proc. ISP RAS, vol. 34, issue 2, 2022, pp. 17-24
67. Документация Universal Sentence Encoder. URL:
https://www.tensorflow.org/hub/tutorials/semantic_similarity_with_tf_hub_universal_encoder?hl=ru
68. AudioCraft: AI research for audio. URL: <https://audiocraft.metademolab.com/>
69. CLIP: Connecting text and images. URL: <https://openai.com/index/clip/>
70. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. URL:
<https://arxiv.org/pdf/2002.08155>
71. Официальная документация scikit-learn. URL: <https://scikit-learn.org/stable/>
72. Aliyev, A., Molodyakov, S., Voinov, N. (2023). License Plates Detection and Recognition Based on Semi-supervised Learning. In: Arseniev, D.G., Aouf, N. (eds) Cyber-Physical Systems and Control II. CPS&C 2021. Lecture Notes in Networks and Systems, vol 460. Springer, Cham. DOI: 10.1007/978-3-031-20875-1_15
73. Думаев Р. И., Молодяков С. А. Сегментация органов грудной клетки с несколькими метками // Современная наука: актуальные проблемы теории и практики. Серия: Естественные и Технические Науки. -2023. -№09/2. -С. 72-75 DOI: 10.37882/2223-2982.2023.9-2.08
74. MusicGen: Simple and Controllable Music Generation. URL:
<https://audiocraft.metademolab.com/musicgen.html>
75. Alaluf Y. Official Implementation for "Only a Matter of Style: Age Transformation Using a Style-Based Regression Model" (SIGGRAPH 2021). URL: <https://github.com/yuval-alaluf/SAM>
76. Alaluf Y., Patashnik O., Cohen-Or D. Only a Matter of Style: Age Transformation

Using a Style-Based Regression Model // ACM Transactions on Graphics.

Volume 40, issue 4, article No.: 45, pp 1–12. URL:

<http://arxiv.org/pdf/2102.02754>

77. Батаева А. SAM: нейросеть меняет возраст на изображении лица человека //

Neurohive. 2021. URL: <https://neurohive.io/ru/papers/sam-nejroset-menyaet-vozrast-na-izobrazhenii-lica-cheloveka/>

78. Боты: информация для разработчиков. URL: <https://tlgrm.ru/docs/bots>

79. Справочник по Bot API. URL: <https://tlgrm.ru/docs/bots/api>

80. Официальная документация для модуля pyTelegramBotAPI. URL:

<https://pytba.readthedocs.io/en/latest/index.html>

81. Официальная документация Silero. URL: <https://silero.ai/>

82. Документация EasyOCR. URL: <https://github.com/JaidedAI/EasyOCR>

83. DALL·E: Creating images from text. URL: <https://openai.com/index/dall-e/>

84. Официальная документация DeepSeek. URL: <https://www.deepseek.com/>

<https://deepseekcoder.github.io/>

85. Yao X., Puy G., Newson A., Gousseau Y., Hellier P. High Resolution Face Age Editing // arXiv. 2020. URL: <http://arxiv.org/pdf/2005.04410>

86. Официальная документация Caffe. URL: <https://caffe.berkeleyvision.org/>

87. Документация Bark. URL: <https://github.com/suno-ai/bark>

Санкт-Петербургский политехнический университет Петра Великого



Из приветствия ректора Руцкого А.И. «Начиная с первых лет существования, Санкт-Петербургский политехнический университет готовил высококвалифицированных инженеров для промышленности. И сегодня это направление остается главным. Качественно изменилось содержание подготовки. Мир сегодня стремительно меняется. В России назрела острая необходимость формирования новой экономики – экономики знаний, лидерства и инноваций, в основе которой лежит интеграция образования, науки и промышленности... Одна из наших главных задач – подготовка специалистов мирового уровня, способных работать на передовых производственных линиях, сочетая исследовательскую, проектную и предпринимательскую деятельность.»

В СПбПУ можно получить образование по самым разным направлениям – от машиностроения и телекоммуникаций до материаловедения и кибербезопасности. В нашей политехнической семье более 30 тысяч студентов

Высшая школа программной инженерии (ВШПИ) СПбПУ

ВШПИ осуществляет подготовку бакалавров, магистров и аспирантов.

Подготовка бакалавров по направлениям:

09.03.04 «Программная инженерия»

09.03.03 «Прикладная информатика»

Подготовка магистров по направлениям:

09.04.04 «Программная инженерия»

09.04.03 «Прикладная информатика»

Магистерские программы:

- 09.04.04_01 Технология разработки и сопровождения качественного программного продукта
- 09.04.04_02 Основы анализа и разработки приложений с большими объемами распределенных данных
- 09.04.04_04 ИТ-инфраструктура предприятия
- 09.04.03_04 Интеллектуальные технологии управления знаниями и данными

Подготовка аспирантов по направлению:

09.06.01 «Информатика и вычислительная техника»

Специальность 2.3.5 (05.13.11) «Математическое и программное обеспечение вычислительных систем, комплексов и компьютерных сетей»

Подробная информация о Высшей школе программной инженерии имеется на сайте:

<https://hsse.spbstu.ru/>

Молодяков Сергей Александрович



Профессор высшей школы программной инженерии Санкт-Петербургского политехнического университета Петра Великого.

Кандидат технических наук с 1989, доктор технических наук с 2012. Известность получили работы в области создания новых оптико-цифровых систем обработки радиосигналов применительно к радиоастрономии. Получены новые результаты, связанные с принципами системного проектирования оптоэлектронных процессоров и систем на их основе.

Предложена методология и методы распределенной обработки сигналов в гибридных системах и системах компьютерного зрения. Участвовал в разработке процессоров для радиотелескопов: Сибирский солнечный радиотелескоп, Радиооптический телескоп РОТ-54/2.6, РАТАН-600, РТ-64 (г.Калязин).

Автор более 250 печатных работ, в том числе 7 монографий.

Молодяков Сергей Александрович,

**ПРИМЕНЕНИЕ НЕЙРОННЫХ СЕТЕЙ ДЛЯ
ОБРАБОТКИ МУЛЬТИМЕДИЙНОГО
КОНТЕНТА
(100 примеров на Python)**

Монография

Налоговая льгота – Общероссийский классификатор продукции
ОК 005-93, т. 2; 95 3005 – учебная литература

Подписано в печать _____ 2023. Формат 60×84/16. Печать цифровая
Усл. печ. л. ____ . Уч.-изд. л. ____ . Тираж 300 . Заказ

Отпечатано с готового оригинал-макета, предоставленного авторами
в цифровом типографском центре Издательства Политехнического
университета:

195251, Санкт-Петербург, Политехническая ул., 29.

Тел. (812) 540-40-14

Тел./факс: (812) 927-57-76