ВЫСШЕЕ ОБРАЗОВАНИЕ

ЦИФРОВАЯ ОБРАБОТКА ИЗОБРАЖЕНИЙ В OPENCV

Практикум

А. И. Матвеев

ЛАНЬ

E.LANBOOK.COM

УДК 004.932
ББК 32.973.26-018.2я73

М 33 **Матвеев А. И.** Цифровая обработка изображений в OpenCv. Практикум : учебное пособие для вузов / А. И. Матвеев. — Санкт-Петербург : Лань, 2022. — 104 с. : ил. — Текст : непосредственный.

ISBN 978-5-507-44739-8

В учебном пособии даны задания, предназначенные для закрепления теоретических знаний по цифровой обработке изображений в OpenCv. В первых разделах даны задания, посвященные основным операциям в OpenCv, таким как считывание и вывод изображения на экран, запись этого изображения в файл, изменение размера изображения. Многие задания, такие как дискретизация и квантование изображения, предназначены для демонстрации теоретических основ цифровой обработки изображений. Основное внимание в пособии уделено цифровой обработке бинарных изображений, которые наиболее часто встречаются в технических приложениях. Приводятся задания, предназначенные для получения геометрических характеристик бинарных изображений, для геометрического преобразования бинарных изображений. Рассматриваются различные способы получения дополнительных характерных признаков бинарных изображений. В заданиях рассмотрено применение методов Собеля, Превитта и Робертса, а также дискретных производных первого и второго порядка для выделения границ, обнаружение перепадов. В пособии есть задания для овладения пространственными методами обработки изображений, для приобретения навыков пороговой обработки изображений, нахождения и обработки контуров, выделенных на этих изображениях. Рассмотрены практические применения морфологических преобразований.

УДК 004.932
ББК 32.973.26-018.2я73

Рецензент:
Г. В. КУПОВЫХ — доктор физико-математических наук, профессор, зав. кафедрой высшей математики Института компьютерных технологий и информационной безопасности Южного федерального университета.

Александр Иванович Матвеев — доктор физико-математических наук, профессор кафедры электротехники и мехатроники Южного федерального университета

Обложка
П. И. ПОЛЯКОВА

Оглавление

Введение

Компьютерное зрение находит все большее практическое применение в различных сферах деятельности человека. Дисциплина изучает алгоритмы цифровой обработки изображений, занимается реализацией на практике предложенных решений. В методическом пособии предлагаются задания, с помощью которых можно овладеть практическими навыками работы с библиотекой компьютерного зрения и машинного обучения с открытым исходным кодом OpenCv на языке Python. Учебное пособие предназначено для студентов бакалавров и магистров по таким специальностям, как мехатроника и робототехника, вычислительная техника, управление в технических системах, автоматизация технологических процессов в производстве.

В учебном пособии даны задания, предназначенные для закрепления теоретических знаний по цифровой обработке изображений в OpenCv. В первых разделах даны задания, посвященные основным операциям в OpenCv, таким как считывание и вывод изображения на экран, запись этого изображения в файл, изменение размера изображения. Многие задания, такие как дискретизация и квантование изображения, предназначены для демонстрации теоретических основ цифровой обработки изображений. Основное внимание в пособии уделено цифровой обработке бинарных изображений, которое наиболее часто встречается в технических приложениях. Приводятся задания, предназначенные для получения геометрических характеристик бинарных изображений, для геометрического преобразования бинарных изображений. Рассматриваются различные способы получения дополнительных характерных признаков бинарных изображений. В заданиях рассмотрено применение методов Собеля, Превитта и Робертса, а также дискретных производных первого и второго порядка для выделения границ, обнаружения перепадов. В пособии есть задания для овладения пространственными методами обработки изображений, для приобретения навыков пороговой обработки изображений, нахождения и обработки контуров, выделенных на этих изображениях. Рассмотрены практические применения морфологических преобразований. Пособие заканчивается проектом, предназначенным для комплексного применения освоенного материала. После выделения характерных признаков каждого объекта из заданного множества объектов для них формируется таблица признаков. На ее основе с помощью нейронной сети проводится распознавание объектов.

1. Основы цифровой обработки изображений в OpenCv

Цель: изучение типов изображений, способов их формирования. Изучение основных функций OpenCv, применяемых для цифровой обработки изображений.

1. Считывание изображения и вывод его на экран, запись изображения в файл.
2. Вывод свойств изображения и сформированной матрицы на экран.
3. Доступ к изображению для изменения значений цвета пикселей.
4. Создание бинарного изображения и его негатива.
5. Применение библиотеки matplotlib для вывода нескольких изображений в общем окне.
6. Выделение и взятие в рамку определенного региона изображения.
7. Уменьшение размера изображения и вывод матрицы на экран.
8. Знакомство с процессом дискретизации и квантования изображения.
9. Приобретение практических навыков использования этих функций.

1.1. Краткие теоретические сведения

1.1.1. Типы изображений и их представление в цифровой форме

По способу хранения описания изображения оно может быть:

- векторным, если изображение создается набором графических примитивов (отрезок прямой, угол, многоугольник, окружность, дуга и т. д.), из которых и формируется изображение;
- растровым, если изображение кодируется двумерным массивом, элементами которого являются интенсивности серого цвета, либо одного из цветов (красного, зеленого, синего).

Существуют следующие типы изображений:

- ***бинарные*** – изображения, пиксели которого принимают только два значения: 0 и 1, что соответствует черному или белому цвету;
- ***полутоновые*** (серые или изображения в градациях серого) – диапазон значений интенсивности пикселов в формате uint8 [0, 255] или в формате double [0,1] (для языка python вещественные числа float);

- ***палитровые*** – каждому пикселу сопоставляется номер ячейки карты цветов, в карте цветов содержится описание цвета пиксела в некоторой цветовой системе (палитре);
- ***цветные*** (RGB) – пикселы непосредственно хранят информацию об интенсивностях цветного изображения, например, об интенсивности красного, зеленого, синего цвета.

Интенсивность (яркость) изображения является функцией $f(x,y)$ двух пространственных переменных x и y на ограниченной прямоугольной области. Ее называют непрерывной функцией интенсивности. Перейдем от непрерывной функции интенсивности к **дискретной функции интенсивности**.

Для этого нужно провести дискретизацию изображения. Все поле изображения разбивается на маленькие клетки с шагом T_1 по горизонтали и шагом T_2 по вертикали (см. рис. 1). Причем координаты, по которым проводится отчет шагов, направлены слева направо, сверху вниз, что соответствует нумерации столбцов и строк в матрице. Полученные после дискретизации мелкие квадраты (ячейки) называются пикселями. Интенсивность изображения, записанная в них, если ввести обозначения $x = n_1T_1$, $y = n_2T_2$, имеет вид:

79	83	89	…	95	87	235
65	69	75	…	87	78	254
79	84	91	…	107	95	98
…	…	…	…	…	…	…
…	…	…	…	…	…	…
70	73	77	…	54	54	54
67	67	67	…	51	43	64
66	68	71	…	11	43	64

$x = n_1T$

$y = n_2T_2$ Рис. 1

$$f(x,y) = f(n_1T_1, n_2T_2).$$

Эту формулу удобно переобозначить, опустив T_1, T_2, тогда дискретная функция интенсивности принимает вид: $f(n_1, n_2)$.

Интенсивность изображения квантуется, т. е. максимальное его значение разбивается на кванты. В формате uint8 величина кванта равна 1, а максимальное значение интенсивности 255. После такого представления изображения легко перейти к его кодированию в виде матрицы. В первую строку матрицы записывается интенсивность первого ряда ячеек, во вторую строку – второго ряда и так далее, как это изображено на рисунке 2. С помощью одной матрицы можно закодировать изображение полутонового (серого) цвета или изображение одного из цветов (красного, зеленого, синего).

Для того чтобы закодировать цветное изображение, нужно три матрицы R, G, B для каждого цвета (красного, зеленого, синего).

```
[[ 79  83  89 ...  95  87  235]
 [ 65  69  75 ...  87  78  254]
 [ 79  84  91 ... 107  95 98]
 ...
 [ 70  73  77 ...  54  50  66]
 [ 67  69  72 ...  51  43  64]
 [ 66  68  71 ...  1  43  64]]
```

Рис. 2. Матрица изображения

1.2. Считывание изображения и вывод его на экран

Функция чтения изображения из файла **imread(,)**. Первый аргумент в скобках указывает путь к считываемому файлу, второй – флаг.

Синтаксис: **img = cv2.imread('<имя файла>', 1)**. Функция считает изображение из файла <имя файла> и помещает его в массив img. В штрихованных кавычках указан путь к файлу, например, 'C:\izo\cat.jpg'. Второй аргумент в скобках после запятой – это флаг, который определяет, какого типа изображение будет на выходе. Если второй аргумент равен 0, то цветное изображение в jpg-файле трансформируется в полутоновое (серое) изображение и по нему формируется матрица полутонового изображения, если записано 1, то формируется матрица цветного изображения, если –1, то изображение загружает изображение как таковое. Проще всего считать изображение, если файл изображения поместить в одной папке с питоновским файлом программы (в рабочей папке), тогда не нужно указывать путь к файлу, достаточно назвать сам файл: **cv2.imread('cat.jpg')**.

Флагу также можно присвоить одно из следующих значений:

- **cv2. imread color**: загрузка цветного изображения;
- **cv2. imread grayscale**: загрузка изображения в режиме градаций серого;

cv2.imread unchanged: загрузка изображения как такового, включая альфа-канал.

Значение по умолчанию – **cv2. imread color.**

Способы создания массивов для цветного и полутонового изображений [3]:

```
img = cv2.imread ('avto.jpg', 1) # Формируем матрицу цветного изображения после загрузки файла с этим изображением;
img = cv2.imread ('avto.jpg', cv2.IMREAD_COLOR) # Та же операция другим способом;
img = cv2.imread ('avto.jpg', 0) # Формируем матрицу полутонового (серого) изображения после загрузки файла с этим изображением;
img = cv2.imread ('avto.jpg', cv2.IMREAD_GRAYSCALE) # Та же операция другим способом.
```

Функция вывода изображения на экран **imshow.**

Синтаксис: **cv2.imshow('image', img)** – вывод изображения на экран с именем 'image'. Можно выводить несколько изображений, но у каждого должно быть свое имя. Первый аргумент в скобках – это имя окна, вторым аргументом является массив, из которого информация выводится на экран.

Любой код можно закончить командами:

cv.waitKey (0)

cv.destroyAllWindows ()

Первая функция позволяет задерживать изображение после его вывода на экран. В скобках указывается время в миллисекундах. Изображение остается на экране пока не сработает клавиатура. Если нажать какую-либо клавишу, программа продолжится. Если задан **0**, то работа программы продолжится после нажатия клавиши. Вторая функция уничтожает все окна, которые мы создали. Если нужно уничтожить конкретное окно, то в скобках указывается имя окна.

Задание 1.1. Считать файл полноцветного изображения `cat.jpg`, создать для него матрицу изображения, затем вывести сначала полутоновое, затем цветное изображение на экран. Перед выполнением задания получить согласно номеру в списке группы свой файл с изображением.

Импортируем модуль cv2:

```
import cv2,
```

чтобы в дальнейшем использовать функции библиотеки OpenCV, необходимые для цифровой обработки изображений.

Файл изображения `cat.jpg` загружаем в папку вместе с питоновским файлом (в рабочую папку). Чтобы прочитать исходное изображение и создать его матрицу, нужно вызвать функцию **imread** из ранее импортированного модуля cv2. В первом аргументе этой функции, если файл изображения `cat.jpg` не находится в одной папке вместе с питоновским файлом, указываем путь к файлу изображения, например:

```
image = cv2.imread('D:/Izo/Giv/cat.jpg', 0).
```

После этого выводим с помощью функции **imshow** изображение на экран:

```
cv2.imshow('image', img).
```

Задержка изображения на экране до момента нажатия клавиши:

```
cv2.waitKey(0).
```

Уничтожение ранее созданного окна:

```
cv2.destroyAllWindows().
```

Все команды соберем в один скрипт:

```
import cv2 # Загружаем библиотеку opencv;
img = cv2.imread('cat.jpg', 0) # трансформируем цветное изображение в полутоновое и создаем матрицу img, в которую записаны интенсивности каждого пикселя полутонового изображения;
cv2.imshow('image', img) # Вывод изображения на экран;
cv2.waitKey(0) # Задержка, чтобы изображение не закрылось сразу.
```

Выполнить этот же код, заменив в функции **cv2.imread('cat.jpg', 0)** флаг **0** на флаг **cv2.IMREAD_GRAYSCALE**.

После выполнения кода получим на экране изображение:

Задание 1.2. Используя код задания 1.1, в функции **cv2.imread(,)** присвоить флагу значение 1, затем вывести изображение на экран. Выполнить этот же код, заменив в функции **cv2.imread('cat.jpg', 1)** флаг **1** на флаг **cv2.IMREAD_COLOR**.

1.3. Запись изображения в файл

Для создания изображения из его матрицы в виде файла используется функция **cv.imwrite (,)**.

Синтаксис: **imwrite(<имя файла>.<расширение>, img)** – первый аргумент в скобках – это имя сохраняемого файла, второй аргумент – это название матрицы изображения, с помощью которой создаем файл с выбранным расширением. Функция **imwrite** записывает матрицу бинарного, полутонового или полноцветного изображения на диск и сохраняет изображение в файле с именем <имя файла>.

Пример: **`cv.imwrite('img.jpg', img)`**. Из матрица `img` получим файл `img.jpg` в формате jpg и сохраним его в рабочей папке, т. е. рядом с питоновским файлом.

Задание 1.3. Сформировать матрицу изображения, записать ее в файл с расширением png. Изображение, записанное в этом файле, вывести на экран.

Скрипт:

```
import cv2
img = cv2.imread('cat.jpg')
# запись изображения из матрицы в файл
cv2.imwrite( 'img.png', img)
img = cv2.imread( 'img.png')
cv2.imshow('image', img)
cv2.waitKey(0)
```

После выполнения кода проверить, появился ли файл img.png в папке с исполненным питоновским файлом.

1.4. Вывод сформированной матрицы на экран

Задание 1.4. Сформировать матрицу, у которой выше диагонали единицы, а ниже – нули, записать ее в файл, затем считать файл и вывести на экран. Строим массив 28×28:

```
import cv2
import numpy as np # модуль для математических вычислений
n = 28
a = np.ones([28,28]) # Строим массив 28x28, заполненный нулями.
```

```
for i in range(n):     # Заполняем диагональ массива
    a[i][i] = 1

for i in range(n):   # Заполняем массив выше диагонали
    for j in range(0, i):
        a[i][j] = 0
cv2.imwrite ('ris.png', a) # записываем изображение в
файл
img = cv2.imread('ris.png', 0) # считываем изображение
с файла
cv2.imshow('image',img) #вывод записанного изображения
на экран
print(img)  # печатаем матрицу
cv2.waitKey(0)
```

На выходе программы распечатанная матрица img:

```
[[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
 [0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
 [0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
 [0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
 [0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
 [0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
 [0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
 [0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
 [0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
 [0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
 [0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
 [0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
 [0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1]
 [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1]]
```

Изображение, закодированное матрицей img, выведенное на экран:

1.5. Вывод основных свойств матрицы изображения на экран

В функциях **type(img)**, **imgshape**, **img.size**, **img.dtype** хранятся соответственно следующие сведения о матрице изображения: тип и класс данных изображения, число строк, столбцов и каналов RGB матрицы изображения, количество пикселей, формат матрицы изображения.

Задание 1.5. Вывести свойства матрицы изображения на экран.

Скрипт [3]:

```
import cv2
import numpy as np
img = cv2.imread('cat.jpg', 0)
cv2.imshow ('image', img)
print(type(img))#Класс: <class 'numpy.ndarray'>
print (img.shape)# Кортеж числа строк и столбцов (раз-
решение), и каналов RGB:(427, 500, 3)
print (img.size)# Общее количество пикселей : 640500
print (img.dtype)# Тип данных изображения : uint8
```

На выходе программы получаем число строк и столбцов, общее количество пикселей, формат матрицы изображения:

<class 'numpy.ndarray'>, (427, 500, 3), 640500, uint8

1.6. Доступ к цифровому изображению для изменения значений пикселей

После обращения к матрице цветного изображения `img[100, 150]` с указанием координат пикселя на выходе получим массив из трех значений: интенсивности синего, интенсивности зеленого и интенсивности красного цветов. Отметим, что цвета в матрице записаны в следующем порядке: B, G, R. При обращении к матрице изображения в градациях серого возвращается только значение интенсивности серого. Чтобы менять интенсивности изображения, установим библиотеку «numpy» с помощью команды «pip install numpy» в командном окне Widows.

Задание 1.6. Определить с помощью функции **print(img.shape)** максимальное число пикселей по ширине и высоте изображения. Выбрать координаты так, чтобы они не выходили за пределы размеров изображения. Задать координату по горизонтали равной сумме номе-

ра по списку группы плюс 70, по вертикали равной сумме номера по списку группы плюс 50.

Для получения доступа к значению пикселя по его координатам строк и столбцов загрузим установленные библиотеки opencv, numpy, затем цветное изображение [3]:

```
import cv2
import numpy as np
img = cv2.imread('cat.jpg')
print(img.shape)#определение размера матрицы
waitKey(0)#задержка на экране
px = img[100, 150] # доступ к пикселю цветного
#изображения с коодинатами 100, 150.
print(px)cv2
waitKey(0)
```

На выходе: `[95 129 175]`

```
   blue = img[100, 150, 0] # доступ только к синему
пикселю с координатами (100, 150)
   print(blue)
```

На выходе: `95`

Изменить значения пикселей, интенсивности B, G, R цветов, взяв интенсивности первого упражнения `[95 129 175]` и прибавив к ним номер по списку группы.

```
   img [100, 150] = [105, 139, 185]
   print(img [100, 150])
```

На выходе программы: [105 139 185]

Приведенный доступ к значениям каждого пикселя занимает много времени. С помощью функций array.item () и array.itemset () доступ будет более быстрым, но они работают только для серого изображения. Для получение доступа ко всем значениям B, G, R, нужно вызывать array.item ().

```
   # доступ к красному пикселю. Первые две цифры в
скобках – координаты пикселя 100, 150 (взять из зада-
ния 1.5), третья цифра – флаг красного цвета.
```

```
img.item(100,150,2)
```
На выходе: `59`
```
img.itemset((100,150,2),100)
```
На выходе программы: `100`

1.7. Создание бинарного изображения и его негатива, вывод нескольких изображений в общем окне

Бинарное изображение можно получить из полутонового изображения, если провести его пороговую обработку. Алгоритм бинаризации полутонового изображения таков: если значение пикселя больше порогового значения, то ему присваивается 1, если меньше, то 0. Для вывода нескольких изображений в одном окне нужно установить библиотеку «matplotlib» с помощью команды «pip install matplotlib» в командном окне windows. Затем вызвать в программе функцию **pit.subplot(211)**. С помощью этой функции можно вывести несколько изображений в одном окне. Первая цифра в этой функции – число мест в длину, вторая – число мест в высоту, третья – номер изображения. В конце программы у matplotlib должна быть функция **plt.show()** для вывода на экран всех изображений. Matplotlib по умолчанию строит оси X, Y, поэтому их нужно отключить функцией **plt.axis("off").**

Задание 1.7. Считать файл полноцветного изображения cat.jpg, создать для него два места в окне в ширину и два места в высоту. Преобразовать матрицу цветного изображения в полутоновое, из него, используя функцию **cv2.threshold**, получить бинарное монохромное изображение. Из бинарного монохромного изображения получить его негатив. Функция

cv2.threshold(gray,128,255,cv2.THRESH_BINARY)

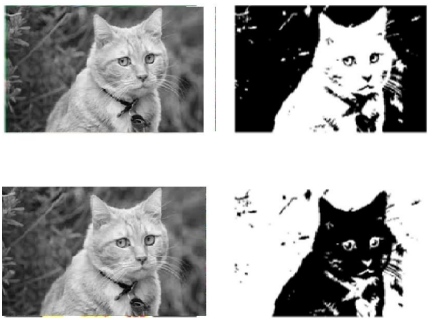
предназначена для получения монохромного изображения, в матрице которого записано либо 0, либо 255. В скобках gray – исходное изображение; 128 – пороговое значение; 255 – значение, которое придаем пикселю, если его значение больше порогового. В первое окно записать исходное изображение, во второе – цветное монохромное изображение, в третье – бинарное монохромное изображение, в четвертое – инвертированное изображение.

Скрипт [3]:

```
import cv2
from matplotlib import pyplot as plt
```

```
img = cv2.imread('cat.jpg')
imag = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
# Оригинальное изображение выводится в первое окно:
plt.subplot(221)
plt.imshow(imag)
plt.axis("off")
gray_img = cv2.imread('cat.jpg',0) # полутоновое изображение
im_bw = cv2.threshold(gray_img, 128, 255,
cv2.THRESH_BINARY)[1] # Функция для получения бинарного изображения, в скобках gray_img – исходное изображение; 128 – пороговое значение; 255 – значение, которое придаем пикселю, если его значение больше порогового; [1] – ожидаем один ответ (по умолчанию 2)
plt.subplot(222) # Выведем бинарное монохромное изображение во второе окно
plt.imshow(im_bw,'gray')
plt.axis("off")
im_bwa = cv2.threshold(imag, 128, 255,
cv2.THRESH_BINARY)[1]
plt.subplot(223) # Выводим монохромное цветное изображение в третье окно
plt.imshow(im_bwa)
plt.axis("off")
im_bwb = cv2.threshold(gray_img, 128, 255,
cv2.THRESH_BINARY_INV)[1] # Функция
cv2.THRESH_BINARY_INV – инвертирует монохромное изображение
plt.subplot(224)# Выведем инвертированное изображение
plt.imshow(im_bwb, 'gray')
plt.axis("off")
plt.show()
cv2.waitKey(0)
```

На выходе программы:



1.8. Выделение и взятие в рамку определенного региона изображения, ROI изображения

Иногда нужно выделить определенный регион изображения. Чтобы обнаружить номер автомобиля, удобно выделить сначала весь автомобиль, взяв его в рамку. Затем по характерным признакам найти сам номер, выделить его и ограничить изображение номера рамкой. Это повышает точность, потому что номер нужно искать на изображении автомобиля, и производительность, потому что мы ищем его в ограниченной области. При работе с алгоритмом ROI подключим библиотеку Numpy. Для визуального определения границ номера авто рисуем на оригинальном изображении прямоугольник, в котором заключен этот номер. Функция **cv2.rectangle(img, (340,330), (470,430), (0,0,255), 2)** рисования прямоугольника имеет следующие параметры: (340,330), (470,430) – координаты левого верхнего и правого нижнего угла, (0,0,255) – красный цвет, 2 – толщина линии. Зная координаты углов прямоугольника, вырезаем с помощью библиотеки Numpy интересующую нас область, т. е. номер авто. Затем переносим эту область в другое место, сохраняя размер прямоугольника.

Задание 1.8. На заданном изображении выделить его характерный участок.

Скрипт:

```
import cv2
img=cv2.imread('avto.jpg')
image=cv2.rectangle(img,(280,340),(330,390),(0,0,255),
2) # Рисуем прямоугольник, (340,330), (470,430) – ко-
```

ординаты левого верхнего и правого нижнего угла; `(0,0,255)` *– красный цвет,* `2` *– толщина линии.*
```
cv2.imshow('selected', img)
```

На выходе программы изображение с выделенной в рамку областью:

1.9. Уменьшение размера изображения, вывод матрицы изображения на экран после уменьшения ее размеров

Обычно размер матрицы изображения очень велик. Чтобы вывести матрицу на экран, нужно уменьшить размер изображения.

Задание 1.9. Уменьшить заданное изображение и вывести на печать матрицу уменьшенного изображения. Нам надо сохранить соотношение сторон, чтобы изображение не исказилось при уменьшении. Для этого необходимо вычислить коэффициент уменьшения стороны. Скрипт [3]:

```
import cv2
img=cv2.imread('avto.jpg')
final_wide = 200
r=float(final_wide)/img.shape[1]
dim=(final_wide, int(img.shape[0]*r))#уменьшаем изображение до подготовленных размеров
resized=cv2.resize(img,dim,interpolation=
cv2.INTER_AREA)
cv2.imshow("Resize image", resized)
print(resized.shape)
print(resized)
img=cv2.imread('avto.jpg', 0)
cv2.imshow('image', img)
print(img)
```

На выходе программы:


```
[[238 237 236 ...  11  10  21]
[239 238 237 ...  15  14  15]
[240 239 239 ...  20  14  13]
...
[ 80  57  68 ... 104 101  98]
[ 65  95  85 ... 101  94  86]
[ 84  93  69 ... 104  96  98]]
```

Задание 1.10. Считать цветное изображение, конвертировать его в полутоновое, затем получить негатив полутонового изображения.

Скрипт:

```
import cv2
img = cv2.imread("auto.jpg", 0)
img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
# Превратим в серый
img = cv2.bitwise_not(img) # Функция инвертирования изображения
cv2.imshow('Img',img)
cv2.waitKey(0)
```

На выходе программы:
изображение в негативе.

2. Процессы дискретизации и квантования изображения

2.1. Дискретизация изображения

Цель: изучение функций, использующихся для моделирования процессов квантования и дискретизации изображения на языке Python.

1. Краткие теоретические сведения.

Интенсивность изображения $f(x, y)$ является функцией двух пространственных переменных x и y на ограниченной прямоугольной области. Алгоритм дискретизации: разбиваем три матрицы цветного изображения на отдельные блоки с шагом дискретизации K. В каждом блоке вычисляем среднее значение по каждому цвету в отдельности и полагаем, что внутри блока интенсивность равна вычисленному среднему значению. Кроме того, добавим функцию автоматического сохранения оригинального размера изображения, так как размер изображения нужно изменить, чтобы он был кратен размеру шага.

Задание 2.1. Выбрать значение шага дискретизации в пределах от 5 до 15. Продискретизировать с этим шагом дискретизации изображение и вывести его на экран.

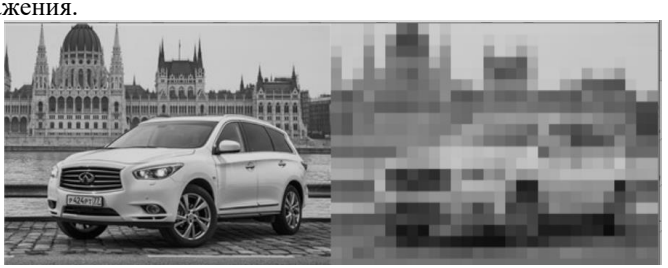
Скрипт:

```
import cv2
import numpy as np
image=cv2.imread('cat.jpg')# Загрузим
и сохраним изображение
img=image.copy() # Создадим копию изображения,
над которым и выполним дискретизацию
K= 10 # Зададим размер шага изображения (количество
пикселей в этом изображении будет KxK)
s=img.shape # Получаем размер исходного изображения
и его тональность.
```

```
h1,w1 = s[0],s[1]  # Запоминаем отдельно высоту
и ширину исходного изображения
h = (s[0] - s[0]%K)  # Делим высоту на шаг с выделением
остатка и вычитаем это из начальной высоты.
Например : (641 - 641 % 10) = 641 - 1 = 640. Это
позволяет за указанное количество шагов пройти
всё изображение точно без остатка.
w=(s[1]-s[1]%K)  # То же самое для высоты
img=cv2.resize(img,(w,h))  # Меняем размер
изображения на новые высоту и ширину
for y in range(0,h-1,K):  # Пробегаем всё изображение
по высоте (сверху вниз) с шагом K
  for x in range(0,w-1,K):  # Для каждого из пикселей
- пробегаем всё изображение по ширине (слева направо)
с шагом K
if len(s)>2:
# Проверяем : изображение полутоновое или цветное?
В цветном строка Len вернет число 3 (так как у цветного
3 параметра - высота, ширина и число каналов - rgb),
в полутоновом строка вернет число 2 (высота, ширина)
        s=np.average(img[y:(y+K), x:(x+K)], axis=0)
# Функция np.average вычисляет средневзвешанное
значение оси (здесь это - средний цвет изображения).
Здесь img[y:(y + K), x:(x + K)] - вырезаем фрагмент
изображения от текущего значения высоты y до K: (y:(y
+ K)), то же самое и для ширины x
        img[y:(y+K), x:(x+K)]=np.average
        (s, axis=0)
# Повторяем действие вычисления среднего значения цве-
та от предыдущего значения s (нужно для цветного изоб-
ражения) и записываем это значение во весь блок изоб-
ражения img[y:(y + K), x:(x + K)] (красим текущий блок
изображения его средним значением)
```

```
        else:
            s=img[y:(y+K), x:(x+K)]
            img[y:(y+K), x:(x+K)]=np.average(s)
# Если же строка возвращает значение меньше 3 (2 – ширина и высота), значит это полутоновое изображение, находим среднее значение цвета блока и красим его в этот цвет (как в предыдущем шаге)
img=cv2.resize(img, (w1,h1))
# Возвращаем размер изображения к размеру исходного изображения
res=np.hstack((image,img))
# Объединим исходное и дискретизированное изображение в одно окно для сравнения
cv2.imshow("Img",res) # И выведем это окно на экран
cv2.waitKey(0)
```

На выходе программы: оригинальное и дискретизированное изображения.



2.2. Квантование изображения

Процесс разбиения непрерывного динамического диапазона значений яркости на ряд дискретных уровней называется **квантованием**. Число уровней квантования равно

$$K = [A/\Delta A],$$

где A определяет диапазон значений яркостей функции $f(x, y)$, ΔA – величина кванта, для удобства полагаем, что ее значение равно единице. В примере сократим количество градаций, используемых в изображении, до 4. С их уменьшением уменьшается также качество изображения.

Задание 2.2. Проквантовать изображение, сократив число градаций до 4

Скрипт:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
plt.subplot(121)
img = cv2.imread('avto.jpg')
plt.imshow(cv2.cvtColor(img,cv2.COLOR_BGR2RGB))#
Меняем формат BGR на RGB
plt.axis("off")
plt.subplot(122)
Z=img.reshape((-1,3)) # Изменяем размер массива изображения, где (-1,3) – задаем число строк и столбцов, в которые помещаем прежний массив. В данном случае 3 – означает в 3 столбца, а -1 это исключение, означает что мы даем python самому понять во сколько строк это можно уместить
Z = np.float32(Z) # преобразуем массив к float (числа с плавающей точкой)
crt=(cv2.TERM_CRITERIA_EPS+cv2.TERM_CRITERIA_MAX_ITER, 10, 1.0) # Критерии для остановки квантования. Здесь останавливаем квантование, если достигли 10 итераций или если достигли точности в 1.0)
K = 4 # Количество градаций изображения
ret,label,center=cv2.kmeans(Z,K,None,crt,10,cv2.KMEANS_RANDOM_CENTERS) # Функция квантования
# Преобразуем изображение к начальному виду:
center = np.uint8(center) # Преобразование матрицы к формату uint8 (от 0 до 255)
res = center[label.flatten()] # Сворачиваем массив нескольких матриц в одну
res2 = res.reshape((img.shape))
# Выводим изображение на экран:
cv2.imshow("Img", res2)
cv2.waitKey(0)
```

На выходе программы: оригинальное изображение и квантованное изображения.

Выводы: при квантовании изображения уменьшается число градаций в сером изображении. Качество изображения становится хуже.

3. Бинарные изображения, основные характеристики бинарных изображений

Цель: изучение методов цифровой обработки бинарных изображений, геометрических характеристик этих изображений, способов получения дополнительных параметров бинарных изображений. Изучение основных функций OpenCv, применяемых для цифровой обработки бинарных изображений.

Краткие теоретические сведения

Элементы матрицы бинарного изображения имеют значения: 0 или 1, что соответствует черному или белому цвету. Бинарное изображение проще обрабатывать и анализировать различными алгоритмами, поэтому если есть возможность, то их применяют в первую очередь для анализа полутоновых и цветных изображений. Основное внимание акцентируется на геометрических характеристиках бинарного изображения – площади, объеме, положении в пространстве и ориентации. Для описания бинарного изображения используют характеристическую функцию $b(x, y)$. Она равна единице для пикселей бинарного изображения и нулю для пикселей фона.

Бинарное изображение можно получить после пороговой обработки полутонового изображения

$$b(x,\ y) = \begin{cases} 1, \text{если } f(x, y) \geq a, \\ 0, \text{если } f(x, y) < a \end{cases},$$

где a – порог, $f(x, y)$ – интенсивность полутонового изображения. Этот процесс называется пороговой бинаризацией.

Площадь бинарного изображения равна количеству единиц, содержащихся в нем: $S = \sum_{x,y} b(x, y)$. Моменты порядка pq участка S изображения I вычисляются следующим образом:

$$m_{p,q}(S) = \sum_{x,y \in S} x^p x^p b(x, y) \Big/ \sum_{x,y \in S} b(x, y).$$

Центр тяжести изображения определятся формулами:

$$x_c = \sum_{x,y \in S} xb(x, y) \Big/ \sum_{x,y \in S} b(x, y), \quad y_c = \sum_{x,y \in S} yb(x, y) \Big/ \sum_{x,y \in S} b(x, y).$$

3.1. Геометрические характеристики бинарных изображений

Геометрические характеристики бинарных изображений можно рассматривать как их характерные признаки. Признак – это количественное описание того или иного свойства объекта. Составив таблицу признаков для множества объектов, можно провести их классификацию. В первую очередь у бинарного изображения вычисляются следующие геометрические характеристики: площадь *s*, периметр *p*, ширина *w*, высота *h*, отношение ширины к высоте: *w*/*h*, отношение площади изображения к площади описывающего прямоугольника: s/(*wh*), эквивалентный диаметр – это удвоенный корень квадратный из площади изображения, деленной на π: $d = 2\sqrt{s / \pi}$, моменты *m*00, *m*01, *m*10, *m*11, определяющие площадь, центр масс объекта, и другие моменты более высокого порядка.

Для вычисления перечисленных характеристик импортируем сначала модули `cv2, numpy` и загружаем изображение. Затем с помощью функции

contours, hierarchy = cv2.findContours(thresh, 5, 5)

найдем на изображении все контуры. Здесь у функции два возвращаемых значения: первое – контур, а второе – топологическая структура (иерархия). Контур (первое возвращаемое значение) – это список, в котором хранятся все контуры изображения. Каждый контур представляет собой массив numpy, содержащий координаты точек границы объекта (*x*, *y*). Строка **cnt = contours[0]** с аргументом 0 выделяет

один внешний контур. Окончательно запишем код для формирования массива `cnt` точек этого контура:

```
import cv2 # Загружаем библиотеку opencv
import numpy as np
img = cv2. imread('bin3.jpg',0)
ret,thresh = cv2.threshold(img,0,255,0)
contours, hierarchy = cv2.findContours(thresh, 5, 5)
cnt = contours[0] # создание массива точек контура
```

Зная координаты точек контура, вычислим основные геометрические характеристики бинарных изображений.

3.1.1. Площадь, ограниченная контуром

Площадь, ограниченная контуром, определяется функцией **cv2.contourArea ()** или моментом '**m00**'.

3.1.2. Длина контурного периметра

Длина контурного периметра (длина кривой) определяется функцией cv2.arcLength (,).

Второй аргумент в скобках указывает, является ли граница бинарного изображения замкнутым контуром (указано True) или просто кривой.

3.1.3. Моменты

Вычислив моменты объектов, можно использовать их в качестве характерных признаков для классификации этих объектов. Функция **cv2.moments ()** дает список всех вычисленных значений моментов.

3.1.4. Отношение ширины к высоте ограничивающего прямоугольника

Отношение ширины к высоте ограничивающего прямоугольника найдем с помощью функции

```
x, y, w, h = cv2.boundingRect(cnt).
```

Она возвращает ширину – `w` и высоту – `h` бинарного изображения. Используя их, найдем отношение `float(w)/h`.

3.1.5. Отношение площади контура к площади ограничивающего прямоугольника

Используя параметры w,h, вычисляем площади ограничивающего прямоугольника и контура:

```
arr=w*h, ar=cv2.contourArea(cnt).
```

Затем находим отношение площади контура к площади ограничивающего прямоугольника:

```
extent=float(area)/rect_area.
```

3.1.6. Эквивалентный диаметр

Эквивалентный диаметр – это диаметр круга, площадь которого совпадает с площадью контура. Вычисляем по формулам:

```
ar=cv2.contourArea(cnt), eqdiam=np.sqrt(4*ar/np.pi)
```

Задание 3.1. Вычислить площадь *s*, периметр *p*, ширину *w*, высоту *h*, отношение ширины к высоте *w/h*, отношение площади изображения к площади описывающего прямоугольника s/(*wh*), эквивалентный диаметр, центр масс, моменты бинарного изображения.

Скрипт [4]:

```
import cv2 # Загружаем библиотеку opencv
import numpy as np
img = cv2. imread('bin3.jpg',0)
imag = cv2. imread('bin3.jpg',0)
ret,thresh = cv2.threshold(img,0,255,0)
contours, hierarchy = cv2.findContours(thresh, 5, 5)
cnt = contours[0] # создание контура
ar = cv2.contourArea(cnt)
print(ar) # вычисление площади
prm = cv2.arcLength(cnt,True)
print(prm) # вычисление периметра
M = cv2.moments(cnt) # вычисление моментов
print(M)
x,y,w,h = cv2.boundingRect(cnt)
print(x,y,w,h)
imag = cv2.rectangle(imag,(x,y),(x+w,y+h),(0,255,0),2)
cv2.imshow('Rectan', imag)
asprat = float(w) / h # соотношение сторон
rectar = w * h
```

```
extent = float(ar)/rectar
eqdiam = np.sqrt(4*ar/np.pi)
print(asprat, extent)
print(eqdiam)
cv2.waitKey(0)
```

На выходе программы:

площадь: 151704.0;

периметр: 1558.0.

Моменты: {'m00': 151704.0, 'm10': 29354724.0, 'm01': 29733984.0, 'm20': 7573518792.0, 'm11': 5753525904.0, 'm02': 7770481152.0, 'm30': 2198213829378.0, 'm21': 1484409683232.0, 'm12': 1503588102912.0, 'm03': 2284521458688.0, 'mu20': 1893379698.0, 'mu11': 0.0, 'mu02': 1942620288.0, 'mu30': 0.0, 'mu21': 0.0, 'mu12': 0.0, 'mu03': 0.0, 'nu20': 0.0822704081632653, 'nu11': 0.0, 'nu02': 0.08440999138673556, 'nu30': 0.0, 'nu21': 0.0, 'nu12': 0.0, 'nu03': 0.0}.

x, y, w, h: 0 0 388 393.

Ширина: 388, высота: 393.

Отношение ширины к высоте: 0.9872773536895675.

Отношение s/(*wh*): 0.9948847092153931.

Эквивалентный диаметр: 439.4946323841773.

3.2. Характерные параметры бинарных изображений

Кроме геометрических характеристик, у бинарных изображений можно выделить и другие характерные признаки.

Для выделения характерных признаков сформируем сначала массив всех точек изображения (маску изображения).

3.2.1. Маска и пиксельные точки

Маску изображения создадим, используя функцию `cv2.drawContours()`:

```
import cv2
import numpy as np
img = cv2. imread('bin3.jpg',0)
mask = np.zeros(img.shape,np.uint8)
cv2.drawContours(mask,[cnt],0,255,-1)
pixpoints = np.transpose(np.nonzero(mask))
pixpoints = cv2.findNonZero(mask)
```

Далее изображение маски используется для получения характерных особенностей бинарных изображений.

3.2.2. Максимальное и минимальное значения и их координаты

С помощью изображения маски найдем эти параметры.

```
minval,maxval,minloc,maxloc=
cv2.minMaxLoc(imgray,mask =mask)
```

3.2.3. Крайние точки

Крайние точки означают крайнюю верхнюю, крайнюю нижнюю, крайнюю правую и крайнюю левую точки изображения. Все точки находятся с помощью функций:

```
leftmost = tuple(cnt[cnt[:,:,0].argmin()][0])
rightmost = tuple(cnt[cnt[:,:,0].argmax()][0])
topmost = tuple(cnt[cnt[:,:,1].argmin()][0])
bottommost = tuple(cnt[cnt[:,:,1].argmax()][0])
```

3.2.4. Средняя интенсивность

Используя ту же маску, вычислим среднюю интенсивность изображения в градациях серого:

```
mean_val = cv2.mean(im,mask = mask)
```

3.2.5. Ориентация

Ориентация – это угол, под которым направлено выделенное направление изображения. Угол вычисляется с помощью функции `cv2.fitEllipse()`.

```
import cv2
import numpy as np
img = cv2. imread('bin3.jpg',0)
ret,thresh = cv2.threshold(img,0,255,0)
contours, hierarchy = cv2.findContours(thresh, 5, 5)
cnt = contours[0] # создание контура
(x,y),(MA,ma),ang=cv2.fitEllipse(cnt)
print(ang)
```

Задание 3.2. Используя изображение маски определить крайние точки, минимальное и максимальное значения и их координаты для бинарного изображения. Найти среднюю интенсивность изображения

в градациях серого, ориентацию бинарного изображения с выделенной осью.

Скрипт [4]:

```
import cv2 # Загружаем библиотеку opencv
import numpy as np
img = cv2. imread('bin3.jpg',0)
ret,thresh = cv2.threshold(img,0,255,0)
contours, hierarchy = cv2.findContours(thresh, 5, 5)
cnt = contours[0] # создание контура
mask = np.zeros(img.shape, np.uint8)
cv2.drawContours(mask, [cnt], 0, 255, -1)
pixpoin = np.transpose(np.nonzero(mask))
minv, maxv, minl, maxl = cv2.minMaxLoc(img, mask=mask)
leftmost = tuple(cnt[cnt[:, :, 0].argmin()][0])
rightmost = tuple(cnt[cnt[:, :, 0].argmax()][0])
topmost = tuple(cnt[cnt[:, :, 1].argmin()][0])
bottommost = tuple(cnt[cnt[:, :, 1].argmax()][0])
(x,y),(MA,ma),ang=cv2.fitEllipse(cnt)
meanv = cv2.mean(img,mask = mask)
print(pixpoin)
print(minv, maxv, minl, maxl)
print(leftmost, rightmost, topmost, bottommost)
print(meanv)
print(ang)
cv2.waitKey(0)
```

На выходе программы:

пиксельные точки

```
[[  0 233]
 [  1 232]
 .................
 [158 233]
 [159 233]
 [160 233]]
```

Максимальное и минимальное значения и их координаты: 1.0 53.0 (232, 1) (233, 5).

Крайние точки: (232, 13) (233, 0) (233, 0) (233, 160).

Средняя интенсивность: (49.20359281437126, 0.0, 0.0, 0.0).

Ориентация: 180.0.

4. Цифровая обработка бинарных изображений

4.1. Геометрические преобразования изображений

Цель: изучить основные операции геометрических преобразований изображений, такие как изменение размера, сдвиг, вращение, аффинное преобразование и т. д.

4.1.1. Изменение размера изображения

Изменение размера изображения в OpenCV можно совершить функцией **cv.resize (img, dim, interpolation=…)**. Первый аргумент – матрица изображения, второй **dim** либо **width**, **height** – размер изображения, третий – метод интерполяции. Способы изменения размера следующие. 1. Размер нового изображения указывается в процентах (например: 50%): **scale_percent = 50**. 2. Размер изображения задается вручную: **width=58, height=71.** 3. Размер изображения задается с помощью коэффициента масштабирования. В процессе масштабирования используются разные методы интерполяции. Основные методы интерполяции таковы: **cv.INTER_AREA** – для сжатия, **cv.INTER_CUBIC** и **cv.INTER_LINEAR** – для масштабирования. По умолчанию используется метод интерполяции **cv.INTER_LINEAR**.

Задание 4.1. Изменить размер изображения.

Скрипт для изменения размера входного изображения:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
img = cv2.imread('cat.jpg', 0)
#Первый способ изменения размера задается в процентах
scale_percent = 50 # процент изменения
width = int(img.shape[1] * scale_percent / 100)
height = int(img.shape[0] * scale_percent / 100)
dim = (width, height)
resized = cv2.resize(img, dim, interpolation=
cv2.INTER_AREA)
print('Resized Dimensions : ', resized.shape)
cv2.imshow("Resized image", resized)
cv2.waitKey(0)
```

```
# Второй способ изменения размера задается вручную
print('Original Dimensions : ',img.shape)
width = 58
height = 71
dim1 = (width, height)
# resize image
resized1 = cv2.resize(img, dim1, interpola-
tion=cv2.INTER_AREA)
print('Resized Dimensions : ', resized1.shape)
cv2.imshow("Resized image", resized1)
cv2.waitKey(0)
#третий способ: задается коэффициентом масштабирования
res = cv2.resize(img,None,fx=2, fy=2, interpolation =
cv2.INTER_CUBIC)#OR
height, width = img.shape[:2]
res = cv2.resize(img,(2*width, 2*height), interpola-
tion = cv2.INTER_CUBIC)
cv2.imshow('img', img)
cv2.imshow('res', res)
```

На выходе программы:

4.1.2. Сдвиг, смещение местоположения объекта

Матрица смещения в плоскости (x, y) имеет вид:

$$M = \begin{pmatrix} 1\ 0\ t_x \\ 0\ 1\ t_y \end{pmatrix},$$

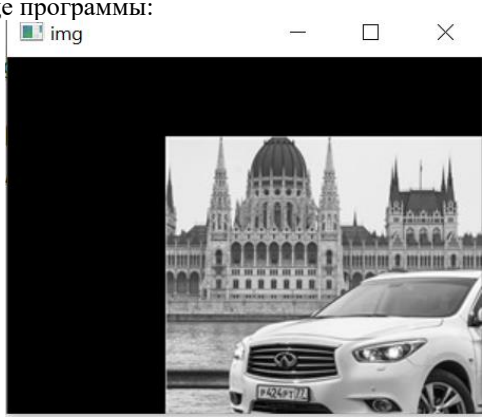
где (t_x, t_y) – вектор смещения. В программе сначала определяем количество столбцов (`rows`) – это ширина, затем количество строк (`cols`) – это высота. С помощью функции **cv.warpAffine ()** изображение сдвигается в направлении (t_x, t_y).

Задание 4.2. Определить размер изображения и сдвинуть изображение на 100 столбцов и 50 строк.

Скрипт [5]:

```
import numpy as np
img = cv2.imread('avto.jpg',0)
rows,cols = img.shape
M = np.float32([[1,0,100],[0,1,50]])
dst = cv2.warpAffine(img,M,(cols,rows))
cv2.imshow('img',dst)
cv2.waitKey(0)
```

На выходе программы:



4.1.3. Вращение изображения

Поворот изображения на некоторый угол θ достигается с помощью матрицы

$$M = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}.$$

У функции вращения **cv.getRotationMatrix2D(,)** первые два аргумента – координаты центра, третий аргумент – угол поворота. Ниже дан пример поворота изображения на 90 градусов относительно центра без изменения размера. Координаты центра равны половине высоты и ширины изображения.

Задание 4.3. Определить размер изображения, его центр и повернуть его на 90 градусов.

Скрипт [5]:

```
img = cv2.imread('messi5.jpg',0)
rows,cols = img.shape
M = cv2.getRotationMatrix2D((cols/2,rows/2),90,1)
dst = cv2.warpAffine(img,M,(cols,rows))
```

На выходе программы:

4.1.4. Аффинная трансформация изображения

При аффинном преобразовании все параллельные линии исходного изображения остаются параллельными и в выходном изображении. Чтобы найти матрицу преобразования, нам нужны три точки из входного изображения и их соответствующие местоположения в вы-

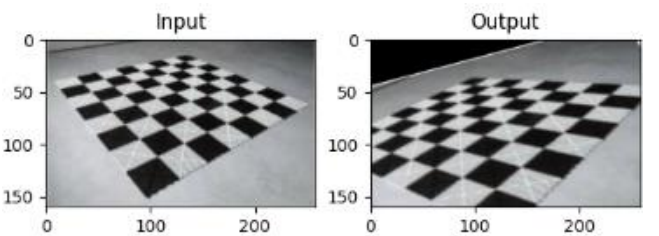
ходном изображении. Затем функция **cv.getAffineTransform** создаст матрицу 2×3, которая передается функции **cv.warpAffine**.

Задание 4.4. Определить размер изображения, задать 3 точки, изменить их координаты и провести аффинное преобразование всего изображения по этим точкам.

Скрипт [5]:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
img = cv2.imread('chax.jpg')
rows,cols,ch = img.shape
pts1 = np.float32([[50,50],[200,50],[50,200]])
pts2 = np.float32([[10,100],[200,50],[100,250]])
M = cv2.getAffineTransform(pts1,pts2)
dst = cv2.warpAffine(img,M,(cols,rows))
plt.subplot(121),plt.imshow(img),plt.title('Input')
plt.subplot(122),plt.imshow(dst),plt.title('Output')
plt.show()
cv2.waitKey(0)
```

На выходе программы:



4.2. Способы получения дополнительных характерных признаков бинарных изображений

4.2.1. Охват объекта повернутым прямоугольником

Поворот прямоугольника, который ограничивает объект, позволяет минимизировать площадь этого прямоугольника. Функция **cv2.drawContours()** возвращает структуру box, которая содержит

следующие аргументы: верхний левый угол (*x*, *y*), ширину, высоту, угол поворота. Чтобы нарисовать прямоугольник, нужны 4 угла прямоугольника, которые задаются функцией **cv2.boxPoints ()**.

Перед выполнением заданий, используя вставку в Microsoft office «создать фигуры правильной формы», создайте фигуру бинарного изображения, скопируйте в Paint и сохраните в файле с расширением jpg или png. Созданное бинарное изображение должно быть инвертированным, т. е. эти фигуры должно быть темными, тогда прямоугольник, круг, эллипс, которые охватывают эти фигуры, будут хорошо видны.

Задание 4.5. Провести охват изображения в прямоугольник, повернутый так, чтобы площадь этого прямоугольника была минимальной.

Скрипт [6]:

```
import numpy as np
img = cv2.imread('bin.jpg', 0)
ret, thresh = cv2.threshold(img, 127, 255,
cv2.THRESH_BINARY)
contours, hierarchy = cv2.findContours(thresh, 1, 1)
cnt = contours[0]
rect = cv2.minAreaRect(cnt)
box = cv2.boxPoints(rect)
box = np.int0(box)
imp = cv2.drawContours(img, [box], 0, (0, 0, 255), 2)
cv2.imshow('прямоугольник', imp)
cv2.waitKey(0)
```

4.2.2. Заключение изображения в круг с минимальной площадью

Окружность с минимальной площадью, охватывающей объект, нарисуем с помощью функции **cv2.minEnclosingCircle ()**.

Задание 4.6. Провести охват изображения в круг.

Скрипт [6]:

```
(x,y),radius = cv2.minEnclosingCircle(cnt)
center = (int(x),int(y))
radius = int(radius)
```

```
img = cv2.circle(img,center,radius,(0,255,0),2)
cv2.imshow('круг', img)
cv2.waitKey(0)
```

4.2.3. Заключение изображения в эллипс с минимальной площадью

Используя функцию **cv2.ellipse()**, можно вписать изображение в эллипс с минимальной площадью.

Задание 4.7. Провести охват изображения в эллипс, повернутый так, чтобы площадь этого эллипса была минимальной.

Скрипт [6]:

```
ellipse = cv2.fitEllipse(cnt)
imag = cv2.ellipse(img,ellipse,(0,255,0),2
cv2.imshow('эллипс', imag)
```

4.2.4. Установка прямой линии в направлении оси симметрии

Если изображение имеет ось симметрии, то ее можно выделить прямой линией.

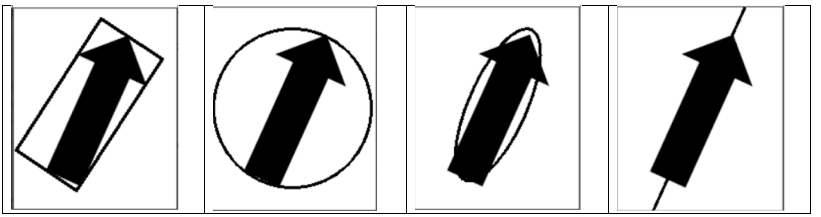
Задание 4.8. Провести прямую линию вдоль оси симметрии изображения.

Скрипт [6]:

```
#Установка прямой линии
import cv2
import numpy as np
img = cv2.imread('bin.jpg', 0)
ret, thresh = cv2.threshold(img, 127, 255,
cv2.THRESH_BINARY)
inv = cv2.bitwise_not(thresh)
cv2.imshow('image', img)
contours, hierarchy = cv2.findContours(inv, 1, 1)
cnt = contours[0]
rows,cols = inv.shape[:2]
[vx,vy,x,y] = cv2.fitLine(cnt,
cv2.DIST_L2,0,0.01,0.01)
lefty = int((-x*vy/vx) + y)
righty = int(((cols-x)*vy/vx)+y)
img = cv2.line(img,(cols-1,righty),(0,lefty),
```

```
(0,255,0),2)
cv2.imshow(' прямая линия', img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

На выходе программы:



4.2.5. Создание выпуклой оболочки вокруг контура

Чтобы нарисовать выпуклую оболочку вокруг контура некоторого изображения, выделяем все его крайние точки и соединяем их ломанной прямой линией. Ни одна точка изображения не должна выходить за пределы выпуклой оболочки.

Импортируем цветное изображение и трансформируем его в полутоновое изображение. Функция Canny выделяет контуры, а с помощью функции **cv2.findContours()** создаем иерархию контуров. Выделяем только внешние контуры изображения. Затем, используя цикл **for**, проходим по каждому из контуров изображения. С помощью переменной **hull** создаем выпуклую оболочку сначала для первого контура, затем для каждого другого контура. В результате получим контур, охватывающий изображение.

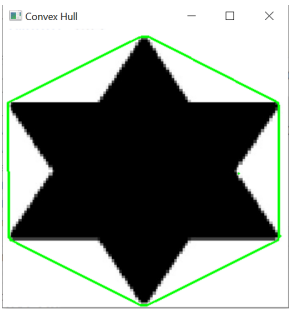
Задание 4.9. Нарисовать контур, охватывающий изображение, толщиной 2, вывести полученное изображение на экран.

Скрипт [6]:

```
import cv2
image= cv2.imread('bin.jpg')
original_image= image
gray= cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
edges= cv2.Canny(gray, 50,200)
contours, hierarchy= cv2.findContours(edges.copy(),
cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)
```

```
for cnt in contours:
    hull= cv2.convexHull(cnt)
    cv2.drawContours(image, [hull],0,(0,255,0),2)
  cv2.imshow('оболочка', image)
cv2.waitKey(0)
```

На выходе программы:



4.2.6. Аппроксимация контура

Функция **cv2.approxPolyDP(cnt,epsilon,True)** позволяет аппроксимировать контур. Первый аргумент cnt = contours [i] – массив с координатами пикселей контура, аргумент epsilon задается в процентах, с уменьшением epsilon максимальное расстояние между ломаной прямой, аппроксимирующей контур, и самим контуром также уменьшается. Значение этого аргумента вычисляется функцией **epsilon = 0.1*cv2.arcLength(cnt,True)**.

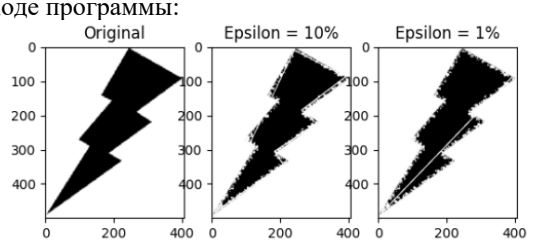
Задание 4.10. Выполнить аппроксимацию контура, полагая `epsilon` =1%, `epsilon`=5% и `epsilon`=10%.

Скрипт [6]:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
img = cv2.imread('bin5.jpg',0)
ret, thresh = cv2.threshold(img, 0, 255, 0)
contours, hierarchy = cv2.findContours(thresh, 2, 3)
imag = cv2.imread('bin5.jpg')
plt.subplot(1,3,1)
plt.title('Original')
```

```
plt.imshow(img,'gray')
plt.subplot(1,3,2)
plt.title('Epsilon = 10%')
for i in range(np.size(contours)):
    cnt = contours [i]
    epsilon = 0.01 * cv2.arcLength(cnt, True)
    approx = cv2.approxPolyDP(cnt,epsilon,True)
    cv2.drawContours(imag,[approx],-1,(255,255,255),2)
plt.imshow(imag)
plt.subplot(1,3,3)
plt.title('Epsilon = 1%')
imAg = cv2.imread('bin5.jpg')
for i in range(np.size(contours)):
    cnt = contours [i]
    epsilon = 0.1 * cv2.arcLength(cnt, True)
    approx = cv2.approxPolyDP(cnt,epsilon,True)
    cv2.drawContours(imAg,[approx],-1,(255,255,255),2)
plt.imshow(imAg)
plt.show()
```

На выходе программы:



Вывод: с уменьшением `epsilon` точность аппроксимации повышается.

4.2.7. Выделение на изображении интересующей области, создание для нее отдельного изображения

Выделим на изображении интересующую нас область, заключив ее в прямоугольную рамку с помощью функции рисования **cv2.rectangle**. Фрагмент изображения, заключенный в рамке, выведем на экран. Используя функцию **.shape**, получим размер изображения и изменим его с помощью функции **cv2.resize**. Функция

cv2.getRotationMatrix2D предназначена для поворота изображения, а функция **cv2.warpAffine** – для аффинного преобразования.

Задание 4.11. Нарисовать прямоугольник в месте, где нужно вырезать фрагмент (см. рис. 3), вывести на экран фрагмент, ограниченный прямоугольником, увеличив этот фрагмент. Определить размер изображения, его центр и повернуть его на 90 градусов.

Скрипт:

```
import cv2
import numpy as np
img = cv2.imread('avto4.jpg', 0)
image = cv2.rectangle(img, (230, 160), (350, 215), (0, 0, 255), 2) # Рисуем на оригинальном изображении прямоугольник в месте, где собираемся вырезать фрагмент. Здесь (230, 160), (350, 215) координаты левого верхнего и правого нижнего угла, (0,0,255) – красный цвет. 2 – толщина линии.
cv2.imshow("Image", image)
crop = img[160:215, 230:350] # Выведем на экран участок изображения, здесь 160:215 – границы по высоте, 230:350 – границы по ширине.
piece = cv2.resize(crop, (200,100), interpolation=cv2.INTER_LINEAR) # Увеличим этот фрагмент до 200x100
cv2.imshow("Area", piece)
(h, w) = piece.shape[:2] # Получим длину и ширину изображения как w и h.
print(w,h)
center = (w / 2, h / 2) # Получим центр изображения
M = cv2.getRotationMatrix2D(center, 90, 1.0) # Повернем центр на 90 градусов с коэффициентом масштаба 1.0.
rotated = cv2.warpAffine(piece, M, (150, 200)) # Преобразовываем исходное piece изображение
```

```
cv2.imshow("Rotated", rotated)
cv2.waitKey()
```

На выходе программы:

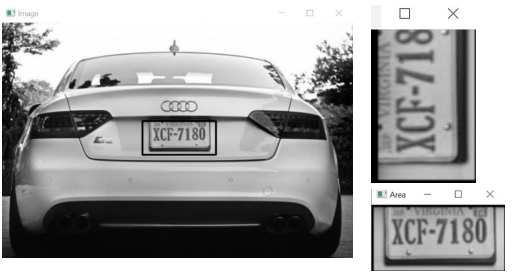


Рис. 3. Взятие интересующей области в рамку и ее поворот

5. Пороговая обработка изображений

Цель: изучение алгоритмов порогового преобразования. Рассмотрение методов адаптивного определения порога, нахождение порогового значения Оцу.

Изучение функций **cv.threshold , cv.adaptiveThreshold**.

5.1. Простой порог

В процессе пороговой обработки изображения все значения интенсивности пикселей по очереди сравниваются с пороговым значением. Старое значение интенсивности пикселя удаляется, и в зависимости от того, больше или меньше интенсивность пикселя порогового значения, этому пикселю присваивается новое значение интенсивности. Если интенсивность пикселя больше порогового значения, то новое значение интенсивности будет равно 255, если интенсивность пикселя меньше порогового значения, то новое значение интенсивности будет равно 0. В результате получим бинарное изображение. Первым аргумент в функции **cv.threshold(img,127, 255, cv.THRESH)** – это исходное изображение, которое должно быть в градациях серого. Второй аргумент – это величина порога. Третий аргумент – это значение интенсивности на выходе функции, когда зна-

чение пикселя больше порогового значения. В режиме инвертирования меньше порогового значения. Четвертым параметром задаются различные типы порогового значения.

cv.THRESH_BINARY – для формирования на выходе бинарного изображения:

$$b(x,y)=\begin{cases}255, \textit{если } f(x,y)>p;\\ 0, \textit{если } f(x,y)<p.\end{cases}$$

cv.THRESH_BINARY_INV – для формирования на выходе инвертированного бинарного изображения:

$$b(x,y)=\begin{cases}0, \textit{если } f(x,y)>p;\\ 255, \textit{если } f(x,y)<p.\end{cases}$$

cv.THRESH_TRUNC, на выходе:

$$b(x,y)=\begin{cases}threshold, \textit{если } f(x,y)>p;\\ src(x,y), \textit{если } f(x,y)<p.\end{cases}$$

cv.THRESH_TOZERO, на выходе:

$$b(x,y)=\begin{cases}src(x,y), \textit{если } f(x,y)>p;\\ 0, \textit{если } f(x,y)<p.\end{cases}$$

cv.THRESH_TOZERO_INV, на выходе:

$$b(x,y)=\begin{cases}0, \textit{если } f(x,y)>p;\\ src(x,y), \textit{если } f(x,y)<p.\end{cases}$$

Графики ниже демонстрируют работу функций простой пороговой бинаризации. На первом графике (см. рис. 4*a*) интенсивность сначала линейно растет, превышая порог p, затем также линейно уменьшается. На выходе в интервале $f(x,y)>p$ значение функции скачком увеличивается и становится равным 255 (см. рис. 4*b*). Вне этого интервала значение функции равно 0.

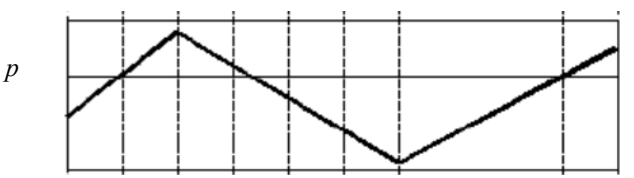


Рис. 4*a*. График зависимости интенсивности от координаты *x* на входе программы

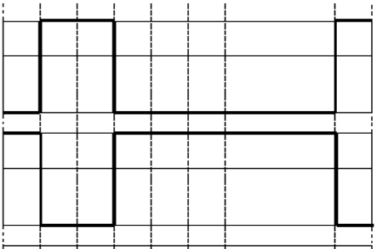


Рис. 4*b*. Графики зависимости интенсивности от координаты *x* на выходе после применения функций

cv.threshold(img,127,255,cv.THRESH_BINARY)

– верхний график,

cv.threshold(img,127,255,cv.THRESH_BINARY_INV)

– нижний график.

Работа функций

cv.threshold(img,127,255,cv.THRESH_TOZERO)
cv.threshold(img,127,255,cv.THRESH_TOZERO_INV)

поясняется графиками (см. рис. 5*a*, *b*).

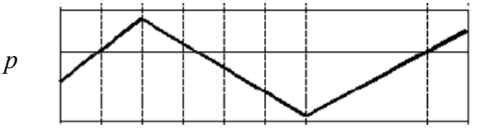


Рис. 5*a*. График зависимости интенсивности от координаты *x* на входе программы

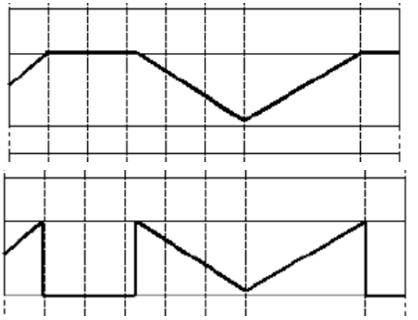


Рис. 5*b*. Графики зависимости интенсивности от координаты *x* на выходе после применения функций

cv.threshold(img,127,255,cv.THRESH_TOZERO)

– верхний график,

cv.threshold(img,127,255,cv.THRESH_TOZERO_INV)

– нижний график.

Задание 5.1. Для трех значений порога 70 + №, 140 + №, 210 + №, где № – номер по списку группы, провести пороговую обработку полутонового изображения с плавным изменением интенсивности.

Скрипт [7]:

```
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt
img = cv.imread ('Grad.png' , 0)
ret,    thresh1    =    cv.threshold    (img,    127,255,
cv.THRESH_BINARY)
ret,    thresh2    =    cv.threshold    (img,    127,255,
cv.THRESH_BINARY_INV)
ret,    thresh3    =    cv.threshold    (img,    127,255,
cv.THRESH_TRUNC)
ret,    thresh4    =    cv.threshold    (img,    127,255,
cv.THRESH_TOZERO)
ret,    thresh5    =    cv.threshold    (img,    127,255,
cv.THRESH_TOZERO_INV)
title  =  ['Original  Image',  'BINARY',  'BINARY_INV',
'TRUNC', 'TOZERO', 'TOZERO_INV']
images  =  [img,  thresh1,  thresh2,  thresh3,  thresh4,
thresh5]
for i in range(6):
    plt.subplot(2,3,i+1),plt.imshow(images[i],'gray')
    plt.title(title[i])
    plt.xticks([]),plt.yticks([])
plt.show ()
```

На выходе программы для вывода в одном окне используем функцию **plt.subplot ()**. Первое изображение (см. рис. 6) – это изображение на входе программы, с монотонным нарастанием интенсив-

ности. Второе изображение на выходе программы – результат бинаризации, полученный с помощью функции

cv.threshold(img,127,255,cv.THRESH_BINARY).

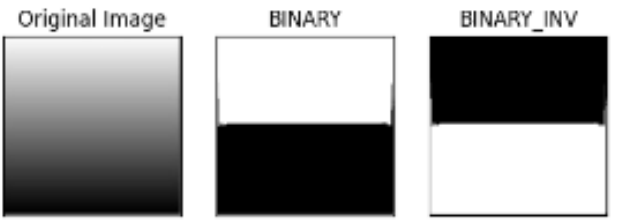
На третьем – результат бинаризации с инверсией, полученный с помощью функции

cv.threshold(img,127,255,cv.THRESH_BINARY_INV).

Рис. 6. Изображения на экране после применения функций

cv.threshold(img,127,255,cv.THRESH_BINARY),
cv.threshold(img,127,255,cv.THRESH_BINARY_INV).

Изображения на экране



после применения функций

cv.threshold(img,127,255, cv.THRESH_TOZERO),

cv.threshold(img,127,255, cv.THRESH_TOZERO_INV).

5.2. Адаптивный порог

Для выполнения этого задания выбрать изображение для обработки с тенью только на одной стороне либо с разной степенью освещенности сторон.

При простой пороговой бинаризации используется порог, значение которого в процессе обработки изображения во всех областях оставалось постоянным. Однако, если у изображения освещение в разных областях неодинаково, то этот метод не работает. В этом случае нужно использовать адаптивный порог. В процессе обработки таким порогом изображение разбивается на области, и в каждой области вычисляется свое значение порога. Он имеет три «специальных» входных параметра и только один выходной аргумент.

Рассмотрим две функции с адаптивным порогом.

cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY,11,2) – в качестве порогового значения берется среднее арифметическое всех пикселей в окрестности выделенного пикселя.

cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,cv2.THRESH_BINARY,11,2) – в качестве порогового значения берется взвешенная сумма значений окрестностей, причем весовые коэффициенты находятся с помощью функции Гаусса. Первый аргумент в этих функциях – исходное изображение, второй – значение интенсивности на выходе функции, третий указывает, какой метод используется: берется среднее арифметическое всех пикселей в окрестности выделенного пикселя или среднее по Гауссу, пятый определяет размер окрестности 11×11 выделенного пикселя, нужной для вычисления порогового значение. Последний аргумент – постоянная *C* вычитается из вычисленного среднего или взвешенного среднего, ее применение позволяет точно настроить пороговое значение.

Задание 5.2. Протестировать функции с адаптивным порогом, задавая последовательно два значения порога, примерно 1/3 и 2/3 от максимума интенсивности. Проанализировать результат пороговой обработки изображения (рис. 7).

Скрипт [7]:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
img = cv2.imread('adaptiv.jpg',0)
img = cv2.medianBlur(img,5)
cv2.imshow('orig',img)
ret1,th1 =
cv2.threshold(img,127,255,cv2.THRESH_BINARY)
th2 = cv2.adaptiveThreshold(img,255,
cv2.ADAPTIVE_THRESH_MEAN_C,cv2.THRESH_BINARY,11,2)
th3 = cv2.adaptiveThreshold(img,255,
cv2.ADAPTIVE_THRESH_GAUSSIAN_C,cv2.THRESH_BINARY,11,2)
titles = ['Original Image', 'Global Thresholding (v =
127)', 'Adaptive Mean Thresholding', 'Adaptive Gaussi-
an Thresholding']
images = [img, th1, th2, th3]
cv2.imshow('th1',th1)
cv2.imshow('th2',th2)
cv2.imshow('th3',th3)
cv2.waitKey(0)
```

На выходе программы:

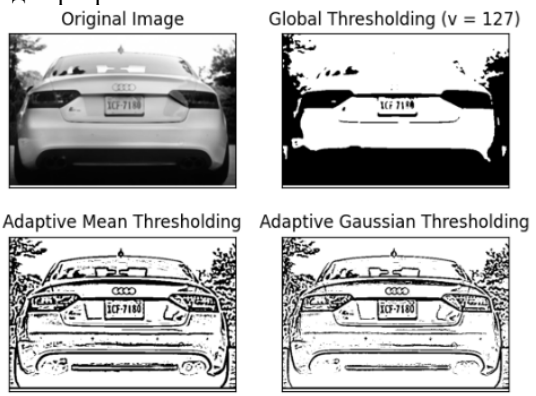


Рис. 7. Первое изображение на входе программы является исходным, все остальные на ее выходе: второе с простым порогом, два последних с адаптивным порогом.

В дальнейшем будут нужны зашумленные изображения. Поэтому создадим файл, в котором изображение испорчено шумом типа соль-перец.

Задание 5.3. Загрузить модули **cv2, random, PIL**. Создать зашумленное изображение.

Скрипт:

```
import cv2 # Подключение библиотеки
import random
from PIL import Image, ImageDraw
image = Image.open('avto.jpg') # Открываем изображение
draw = ImageDraw.Draw(image) # Создаем инструмент для
рисования
width = image.size[0]  # Определяем ширину
height = image.size[1]  # Определяем высоту
pix = image.load()  # Выгружаем значения пикселей
for i in range(width):
    for j in range(height):
        rand = random.randint(0, 200)
        a = pix[i, j][0] + rand
        b = pix[i, j][1] + rand
        c = pix[i, j][2] + rand
        if (a > 255):
            a = 255
        if (b > 255):
            b = 255
        if (c > 255):
            c = 255
        draw.point((i, j), (a, b, c))
image.save("median.png", "JPEG") # сохранить изображе-
ние
cv2.waitKey(0) # Задержка
```

На выходе программы зашумленное изображение:

5.3. Бинаризация Оцу

Если объект отличается по яркости от фона, то можно ввести порог, чтобы разделить изображение на светлый объект и темный фон. Объект – это множество пикселей, яркость которых превышает порог $I > p$, а фон – множество остальных пикселей, яркость которых ниже порога $I < p$. Метод Оцу для расчета порога использует гистограмму изображения. Гистограмма показывает, как часто встречается на данном изображении то или иное значение пикселя. Зная яркость каждого пикселя, подсчитаем сколько пикселей имеют такую яркость. Используя эти сведения, построим гистограмму: по горизонтали откладываем яркость, по вертикали число пикселей с такой яркостью. Наиболее эффективен метод Оцу для обработки бимодального изображения, у которого гистограмма имеет два пика. Такая гистограмма дает два четко разделяющихся класса «полезные пиксели» и «фоновые». В качестве порогового значения можем приблизительно принять значение в середине между этими пиками.

Для выбора способа обработки используется функция **cv2.threshold(img,127,255,cv2.THRESH_BINARY)**, но передается дополнительный флаг **cv.THRESH_OTSU**. Для порогового значения просто введите ноль. Затем алгоритм находит оптимальное пороговое значение и возвращает вас в качестве второго выхода retVal.

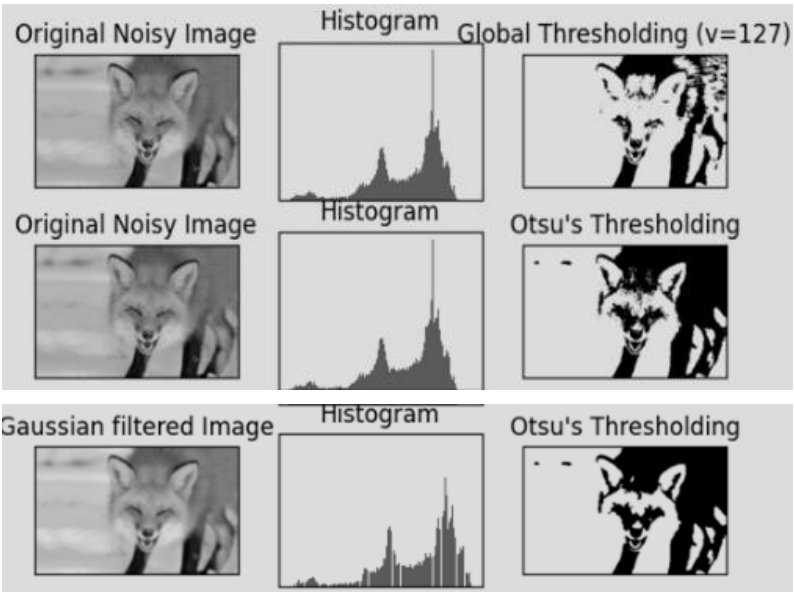
Задание 5.4. На вход программы пороговой обработки подается зашумленное изображение. Это изображение обрабатывается тремя способами. В первом случае используется глобальный порог со значением 127. Во втором случае напрямую применяется порог Оцу. В третьем случае изображение сначала удаляет шум фильтром с гауссовым ядром 5×5, затем применяется пороговая обработка Оцу. Сделать анализ того, как фильтрация шума улучшает результат.

Скрипт [7]:

```
Import cv2
import numpy as np
from matplotlib import pyplot as plt
img = cv2.imread('Lis2.jpg',0)
ret1,th1 =
cv2.threshold(img,127,255,cv2.THRESH_BINARY)# Глобаль-
ная обработка
ret2,th2 =
cv2.threshold(img,0,255,cv2.THRESH_BINARY+cv2.THRESH_O
TSU)# обработка Otsu
blur = cv2.GaussianBlur(img,(5,5),0)
ret3,th3 =
cv2.threshold(blur,0,255,cv2.THRESH_BINARY+cv2.THRESH_
OTSU) # обработка Otsu's после фильтра Гаусса
images = [img, 0, th1,
          img, 0, th2,
          blur, 0, th3]
titles = ['Original Noisy Image','Histogram','Global
Thresholding (v=127)','Original Noisy Image', 'Histo-
gram',"Otsu's Thresholding",'Gaussian filtered Image',
'Histogram',"Otsu's Thresh-olding"]
for I in range(3):
   plt.subplot(3,3,i*3+1), plt.imshow(images[i*3],
   'gray')
   plt.title(titles[i*3]), plt.xticks([]),
   plt.yticks([])
```

```
   plt.subplot(3,3,i*3+2),plt.hist(images[i*3].ravel(),
256)
   plt.title(titles[i*3+1]), plt.xticks([]),
   plt.yticks([])
   plt.subplot(3,3,i*3+3),plt.imshow(images[i*3+2],
‘gray’)
   plt.title(titles[i*3+2]), plt.xticks([]),
   plt.yticks([])
   plt.show()
```

На выходе программы:



Сделать анализ, рассмотреть, как фильтрация шума улучшает результат.

6. Пространственные методы обработки изображений

6.1. Краткое теоретическое введение. Использование скользящих масок при обработке полутоновых изображений

Для улучшения изображений, выделения характерных особенностей применяют локальные преобразования в окрестности некоторого пикселя. Для этого используют матрицу, размеры которой значительно меньше всего изображения. Примеры локального преобразования следующие. Вычисление локального среднего для пикселя с координатами *x*, *y*:

$$\mu(x,y)=\frac{1}{(2m+1)^2}\sum_{i=-m}^{m}\sum_{j=-m}^{m}I(x+i,\,y+j).$$

Другой пример – вычисление локального максимума для пикселя с координатами *x*, *y*:

$$\mu_m(x,y)=\max\{I(x+i,y+j):-k\le i\le k,-k\le j\le k\}.$$

Матрицу локального преобразования некоторого пикселя называют фильтром, маской, шаблоном или окном. Маска – это локальный оператор, определяющий новое значение любого пикселя, которое равно линейной комбинации интенсивностей в окрестности пикселей. Коэффициенты линейного преобразования называются весами. На рисунке 8 приводится маска с весами w_i и дан участок изображения под маской, z_i – значения интенсивности его пикселей. Центр маски совмещается с пикселем, который нужно пересчитать. Затем каждый вес маски умножается на интенсивность, находящуюся под этим весом.

w_1	w_2	w_3
w_4	w_5	w_6
w_1	w_2	w_9

	z_1	z_2	z_3		
	z_4	z_5	z_6		
	z_7	z_8	z_9		

Рис. 8. Маска и участок изображения, который покрывается маской

После сложения всех произведений весов и интенсивностей участка изображения получим их свертку

$$Q=w_1z_1+w_2z_2+\ldots+w_9z_9=\sum_{i=1}^{9}w_iz_i\,.$$

В общем виде такое локальное преобразование пикселя с координатами x, y имеет вид:

$$F(x,y)=\sum_{i=-n_1}^{n_1}\sum_{j=-m_1}^{m_1} w(i,j)\,f(x+i,y+j),$$

где $w(i,j)$ – веса маски размером $2n\times 2m$; i, j – координаты весов; $f(x+i,y+j)$ – интенсивности пикселей с координатами $x+i$, $y+j$. В процессе обработки с помощью маски ее центр перемещается сначала вдоль верхней строки с шагом 1, в результате все интенсивности этой строки заменяются новыми значениями. Закончив обработку верхней строки, центр маски смещается на вторую строку, после чего происходит ее обработка и так далее. Когда маска находится на граничных пикселях, то ее ячейки выходят за пределы изображения, поэтому веса этих ячеек умножаются на ноль.

6.2. Зашумление изображений

При тестировании программ иногда нужны зашумленные изображения. Здесь приводится код для зашумления изображения шумом типа соль-перец. Этот шум еще называют импульсным шумом. Такой шум возникает при случайном изменении интенсивностей пикселей, которые принимают минимальные или максимальные значения динамического диапазона пикселей. На изображении эти пиксели распределяются случайным образом. Это приводит к появлению на изображении черных и белых пикселей. В функции **sp_noise(image,0.3)** второй аргумент задает интенсивность шума. Меняя его, изменяем степень зашумленности.

Задание 6.1. Создать файл с зашумлением изображения шумом типа соль-перец.

Скрипт:

```
import cv2
import numpy as np
import random
red, green, blue = (255, 0, 0), (0, 255, 0), (0, 0,
255) # Создаем 3 возможных цвета - красный, зеленый и
синий
rgb = [red, green, blue] # Помещаем их в кортеж
def sp_noise(image,prob): # Создаем функцию с парамет-
```

```
рами (<наше изображение>, <вероятность зашумления>)
    output = np.zeros(image.shape,np.uint8) # Создаем массив нулей такого же размера и формата как исходное изображение
    thres = 1 – prob # Задаем порог
    for i in range(image.shape[0]): # Пробегаем все столбцы
        for j in range(image.shape[1]): # И для каждого из них пробегаем все строки
            rdn = random.random() # Задаем случайное число от 0 до 1
            if rdn > thres: # Если это случайное число выпало больше нашего порога
                output[i][j] = random.choice(rgb) # Задаем пикселю случайное значение из кортежа
            else:
                output[i][j] = image[i][j] # Иначе оставляем пиксель без изменения
    return output # Возвращаем получившееся изображение
image = cv2.imread('cat.jpg')
image = cv2.resize(image, (900, 600))
noise_img = sp_noise(image,0.3) # Применяем к нашему изображению image, созданную функцию sp_noise, где 0.3 - вероятность зашумления пикселя
res = np.hstack((image, noise_img)) # Объединяем оригинальное и зашумленное изображения в одно окно (для наглядности)
cv2.imshow("Img", res)
cv2.imwrite('Noise.jpg', noise_img) # Сохраняем изображение в папку с файлом, чтобы использовать в дальнейшем
cv2.waitKey(0)
```

На выходе программы:


6.3. Сглаживание изображений

Цель: сглаживание изображений с помощью различных фильтров нижних частот. Усвоение навыков применения 2D-свертки к изображениям.

Для сглаживания изображений используют усредняющий фильтр. Усреднение участка изображения проводится с помощью ядра, например, 5×5. Оно имеет вид:

$$K = \frac{1}{25}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

В процессе фильтрации центр окна 5×5 этого ядра совмещается с выбранным пикселем, затем все пиксели, попадающие в это окно, суммируются, а результат затем делится на 25. В результате находится средние значения пикселей внутри рассматриваемого окна.

В OpenCV двумерная (2D) фильтрация осуществляется функцией **cv2.filter2D ()**.

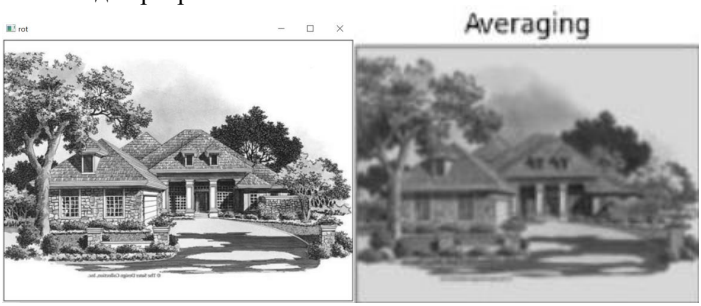
Задание 6.2. Провести сглаживание изображения с помощью функции **cv2.filter2D ()**, используя ядро 5×5.

Скрипт:

```
import  cv2
import  numpy  as  np
from  matplotlib  import  pyplot  as  plt
img  =  cv2.imread ('dom.png')
```

```
kernel = np.ones((5,5),np.float32)/25
dst = cv2.filter2D(img,-1,kernel)
plt.subplot(121),plt.imshow(img),plt.title('Original')
plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(dst),plt.title('Averaging')
plt.xticks([]), plt.yticks([])
plt.show()
```

На выходе программы:



Для сглаживания (размытия) изображения используют также свертку с маской. Параметры маски: размер маски равен сумме весов: 11 – 2 – 2 – 2 – 2 – 2 = 3. Смещение = = 0.

0	–2	0
–2	11	–2
0	–2	0

6.4. Усреднение

Кроме функции **cv2.filter2D ()** для усреднения (сглаживания) используют функции **cv2.blur ()** или **cv2.boxFilter ()**. Перед применением этих функций задается ширина и высота ядра. Например, нормализованный прямоугольный фильтр 3×3 будет таким:

$$K = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

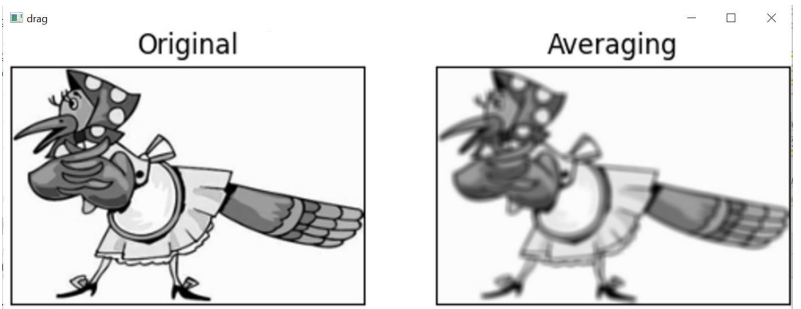
Задание 6.3. Провести усреднение изображения с помощью функции **cv2.blur ()**, используя ядро 5×5.

Скрипт для функции **cv2.blur ()**:

```
import  cv2
import  numpy  as  np
```

```
from  matplotlib  import  pyplot  as  plt
img  =  cv2.imread (‘dom.png’)
blur = cv2.blur(img,(5,5))
plt.subplot(121),plt.imshow(img),plt.title(‘Original’)
plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(blur),plt.title(‘Blurred’)
plt.xticks([]), plt.yticks([])
plt.show()
```

На выходе программы:



6.5. Гауссова фильтрация

Фильтрация с ядром Гаусса выполняется маской, веса которой находятся с помощью функции Гаусса. Рассматриваемая фильтрация осуществляется с помощью функции **cv2.GaussianBlur ()**. В скобках второй аргумент – это ширина и высота ядра, которые должны быть положительными и нечетными. Указывается также стандартное отклонение в направлениях *X* и *Y*, sigma*X* и sigma*Y* соответственно. Если указан нуль, то они рассчитываются исходя из размера ядра. Гауссова фильтрация очень эффективна при удалении гауссовского шума из изображения.

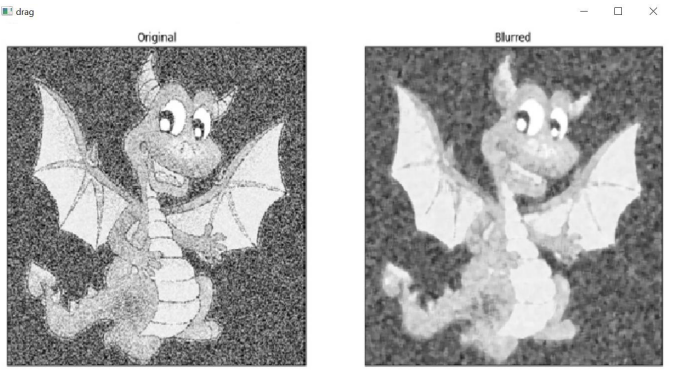
Задание 6.4. Добавить к исходному изображению 20−30% шума. Провести фильтрацию изображения по Гауссу, используя ядро 5×5.

Скрипт:

```
import  cv2
import  numpy  as  np
from  matplotlib  import  pyplot  as  plt
img  =  cv2.imread (‘dom.png’)
```

```
blur = cv2.GaussianBlur(img,(5,5), 0)
plt.subplot(121),plt.imshow(img),plt.title('Original')
plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(blur),plt.title('Blurred')
plt.xticks([]), plt.yticks([])
plt.show()
```

На выходе программы:



6.6. Медианная фильтрация

В процессе этой фильтрации функция **cv2.medianBlur ()** вычисляет медианное значение всех пикселей, окружающих центральный пиксель, и его значение заменяется медианным значением. Это очень эффективно для устранения шума соли и перца. Размер ядра должен быть положительным нечетным целым числом.

Задание 6.5. Добавить к исходному изображению 20–50% шума. Провести медианную фильтрацию изображения, используя ядро 5×5.

Скрипт:

```
import  cv2
import  numpy  as  np
from  matplotlib  import  pyplot  as  plt
img  =  cv2.imread ('median.png')
median = cv2.medianBlur(img,5)
plt.subplot(121),plt.imshow(img),plt.title('Original')
plt.xticks([]), plt.yticks([])
```

```
plt.subplot(122),plt.imshow(blur),plt.title('Blurred')
plt.xticks([]), plt.yticks([])
plt.show()
```

На выходе программы:

6.7. Градиенты изображения, обнаружение перепадов

Цель: нахождение градиентов изображения, края и т. д. Изучение функций: **cv2.Sobel ()**, **cv2.Scharr ()**, **cv2.Laplacian ()**.

Задача обнаружения края или границы некоторой связанной области важна в процессе цифровой обработки изображений. Для решения этой задачи используют методы Собела, Превитта и Робертса. Они основаны на применении первой производной, которая моделируется масками Собела, Превитта и Робертса.

6.7.1. Краткое теоретическое введение

Для обнаружения перепадов используются дискретные аналоги производных первого и второго порядка. Задача обнаружения перепада ставится следующим образом. Перепад яркости – это связанное множество пикселей, которое разделяет области большей и меньшей яркости. На рисунке 9 показаны изменения яркости для наклонных перепадов, они более близки к реальным изображениям.

Анализируя графики производных, отметим, что первая производная положительна в пределах наклонного перепада. Вне его она равна нулю.

Вторая производная положительна в точке перехода от минимального значения интенсивности к наклонному ее росту и отрицательна в точке перехода от наклонного участка к максимальному значения интенсивности. С помощью второй производной можно определить не только границы области размытости изображения, т. е. точки начала и конца роста интенсивности, но и середину этой области

размытости. Для этого достаточно провести линию, соединяющую вершины отрицательного и положительного пиков. Так, где линия проходит через ноль, находится середина размытости.

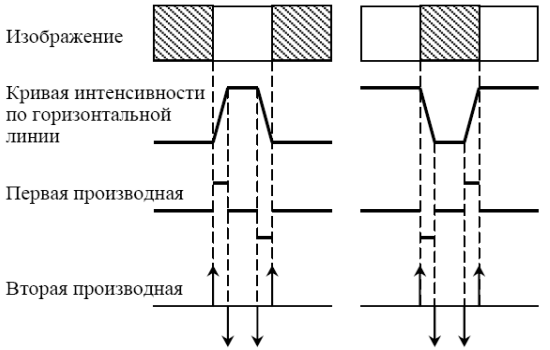


Рис. 9. Графики производных

Замкнутый протяженный перепад яркости называется контуром. Его ширина должна быть много меньше длины.

Операторы градиента. Рассмотрим конкретные методы применения первой и второй производной для определения перепадов. Градиент – это вектор

$$\nabla f = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \partial f / \partial x \\ \partial f / \partial y \end{bmatrix}.$$

Частная производная вдоль оси *x* в дискретной форме равна

$$G_x = \frac{\partial f}{\partial x} = f(x+1) - f(x).$$

–1	0	1
–1	0	1
–1	0	1

	z_1	z_2	z_3		
	z_4	z_5	z_6		
	z_7	z_8	z_9		

Рис. 10. Маска для получения дискретной производной

Маска для получения этой производной показана справа (см. рис. 10).

Если маску совместить с участком (он слева) так, чтобы ее центр совпадал с центром участка z_5, то после свертки масок для выбранного участка получим дискретную производную вдоль оси *x*:

$$G_x(z_5) = (z_3 + z_6 + z_9) - (z_1 + z_4 + z_7).$$

По аналогии создадим маску Превитта, которая моделирует дискретную производную

$$G_y(z_5) = (z_7 + z_8 + z_9) - (z_1 + z_2 + z_3)$$

вдоль оси *y*. На рисунке изображены маски Превитта и Собела для дискретных производных вдоль осей *x* и *y*.

Превитта

–1	0	1
–1	0	1
–1	0	1

Превитта

–1	–1	–1
0	0	0
1	1	1

Собела

–1	–2	–1
0	0	0
1	2	1

Собела

–1	0	1
–2	0	2
–1	0	1

Рис. 11. Маски для получения дискретной производной

Первая слева (см. рис. 11) – это маска Превитта вдоль оси *x*, легко проверить, что ее свертка с участком изображения дает дискретную производную $G_x(z_5)$. Сверка масок Собела с участком изображения дает следующие дискретные производные:

$$G_x(z_5) = (z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7),$$
$$G_y(z_5) = (z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3).$$

Операторы Собеля, Превитта предназначены для выделения краев, (скачков интенсивности), направленных горизонтально, вертикально или под 45°. Они настроены так, чтобы давать максимальный отклик на такие скачки интенсивности. Рассмотрим участок изображения на рисунке 12 ниже, у которого по линии z_2, z_5, z_8 скачок. Слева от этой линии темные пиксели, справа – светлые. Поэтому $z_3 > z_1$, $z_6 > z_4$,

–1	0	1
–2	0	2
–1	0	1

	z_1	z_2	z_3		
	z_4	z_5	z_6		
	z_7	z_8	z_9		

Рис. 12. Получение дискретной производной с помощью маски

$z_9 > z_7$ и свертка маски с пикселями участка изображения положительна

$$G_x(z_5) = (z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7) > 0.$$

То есть отклик у маски положительный. Если нет скачка $z_3 = z_1$, $z_6 = z_4$, $z_9 = z_7$, то отклик равен нулю $G_x(z_5) = 0$.

6.7.2. Обнаружение перепадов (вертикальных и горизонтальных) методом Собеля

В скобках функции Собеля **cv2.Sobel(img, cv2.CV_64F, 1, 0, ksize=5)** первый аргумент – матрица изображения, второй – параметр глубины изображения, горизонтальное направление производной задается параметром 1, 0 (производная берется только по *x*), вертикальное направление производной задается параметром 0, 1 (производная берется только по *y*), последний аргумент – размер маски. Последний аргумент – ksize = 5 – означает размер матрицы маски (тут 5×5), может быть 3, 5 или 7. Если в последнем аргументе ksize = –1, то используется фильтр 3×3 Scharr, который дает лучшие результаты, чем фильтр 3×3 Sobel.

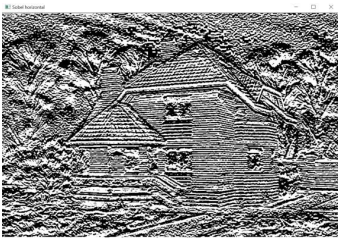
Задание 6.6. Создать файл с изображением, в котором обязательно присутствуют вертикальные и горизонтальные линии. С помощью оператора Собеля обнаружить и выделить эти линии.

Скрипт для оператора Собеля [8]:

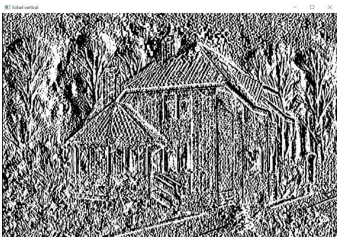
```
import cv2
img = cv2.imread('avt.jpg', 0)
img = cv2.resize(img, (900, 600))
sobel_vertical = cv2.Sobel(img, cv2.CV_64F, 1, 0, ksize=5) # Функция Собеля для вычисления вертикальных линий.
Sobel_horizontal = cv2.Sobel(img, cv2.CV_64F, 0, 1, ksize=5) # То же самое, но 0, 1 означает, что теперь берем производную по y.
Cv2.imshow('Original', img)
cv2.imshow('Sobel horizontal', sobel_horizontal)
cv2.imshow('Sobel vertical', sobel_vertical)
cv2.waitKey(0)
```

На выходе программы:

(*1*)



(*2*)



(*3*)

Исходное изображение (1), с выделенными горизонтальными (2), вертикальными (3) линиями.

Если вы хотите методом Собела обнаружить оба ребра, лучший вариант – сохранить выходной тип данных в некоторых более высоких формах, таких как **cv2.CV_16S, cv2.CV_64F**. Взять его абсолютное значение, а затем преобразовать обратно в **cv2.CV_8U**.

Задание 6.7. Сравнить оба способа для горизонтального фильтра Собела с преобразованием в cv2.CV_8U и без него.

Скрипт [8]:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
img = cv2.imread('N_avt.jpg',0)
# Output dtype = cv2.CV_8U
sobelx8u = cv2.Sobel(img,cv2.CV_8U,1,0,ksize=5)
# Output dtype = cv2.CV_64F.
sobelx64f = cv2.Sobel(img,cv2.CV_64F,1,0,ksize=5)
abs_sobel64f = np.absolute(sobelx64f)
sobel_8u = np.uint8(abs_sobel64f)
plt.subplot(1,3,1),plt.imshow(img,cmap = 'gray')
plt.title('Original'), plt.xticks([]), plt.yticks([])
```

```
plt.subplot(1,3,2),plt.imshow(sobelx8u,cmap = 'gray')
plt.title('Sobel        CV_8U'),        plt.xticks([]),
plt.yticks([])
plt.subplot(1,3,3),plt.imshow(sobel_8u,cmap = 'gray')
plt.title('Sobel     abs(CV_64F)'),     plt.xticks([]),
plt.yticks([])
plt.show()
```

На выходе программы:

6.7.3. Обнаружение перепадов (вертикальных и горизонтальных) методом Превитта

Различие между операторами Превитта и Собеля, как это видно из таблицы, для этих масок лишь в разных весовых коэффициентах.

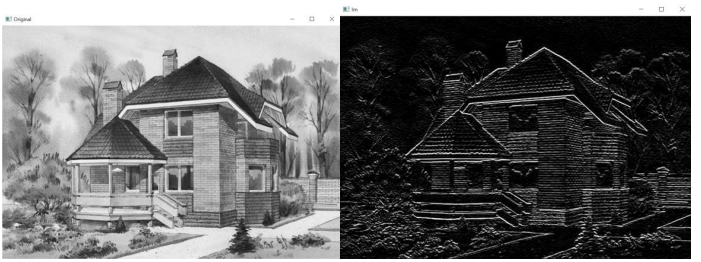
Задание 6.8. Создать файл с изображением, который обязательно содержит вертикальные и горизонтальные линии. С помощью оператора Превитта обнаружить и выделить эти линии.

Скрипт для оператора Превитта [8]:

```
import cv2
import numpy as np
img = cv2.imread('avt.jpg',0)
img = cv2.resize(img, (900, 600))
66ernel = np.array([[-1,-1,-1],[0,0,0],[1,1,1]]) # Со-
здаем ядро (маску) для x
66ernel = np.array([[-1,0,1],[-1,0,1],[-1,0,1]]) # Со-
здаем ядро (маску) для y
img_prewittx = cv2.filter2D(img, -1, 66ernel) # Функ-
ция соединения изображения с ядром, здесь -1 – это
глубина изображения (если значение отрицательное, то
глубина соответствует исходному изображению, как и
cv2.CV_64F)
img_prewitty = cv2.filter2D(img, -1, 66ernel)
```

```
cv2.imshow('Im',img_prewittx)
cv2.imshow('Img',img_prewitty)
cv2.waitKey(0)
```

На выходе программы:



Изображение (справа) после выделения горизонтальных линий методом Превитта.

6.7.4. Выделение границ методом Робертса

Оператор Робертса имеет различие с предыдущими операторами. Теперь маски имеют размер 2×2, что вызывает некоторые неудобства, так как нет четко выраженного центрального элемента. Однако это компенсируется высокой скоростью обработки изображения. Ядра оператора Робертса имеют вид:

-1	0
0	1

0	-1
1	0

Задание 6.9. Используя оператор Робертса, выделить линии на изображении.

Скрипт:

```
import cv2
import numpy as np
img = cv2.imread('Nomer1.jpg',0)
img = cv2.resize(img, (900, 600))
kernel1 = np.array([[1, 0], [0, 1]])
kernel2 = np.array ([[0, 1],[0, 1]])
img_robx = cv2.filter2D(img, -1, kernel1)
img_roby = cv2.filter2D(img, -1, kernel2)
```

```
output_image = img_robx + img_roby
cv2.imshow('output_image',output_image)
cv2.waitKey(0)
```

На выходе программы:

6.7.5. Оператор Лапласа

Чтобы получить оператор Лапласа, найдем частные производные второго порядка. Первые производные в двух точках, расстояние между которыми равно 1, таковы:

$$\frac{df(x+1)}{dx} = f(x+1) - f(x), \quad \frac{df(x-1)}{dx} = f(x) - f(x-1).$$

Используя их, найдем вторую производную:

$$\frac{df(x+1)}{dx} - \frac{df(x-1)}{dx} = f(x+1) - 2f(x) + f(x-1).$$

Вторые производные по x, y в дискретной форме находятся по формулам:

$$\frac{\partial^2 f}{\partial x^2} = f(x+1, y) + f(x-1, y) - 2f(x, y),$$

$$\frac{\partial^2 f}{\partial y^2} = f(y+1, x) + f(y-1, x) - 2f(y, x).$$

Откуда дискретный оператор Лапласа равен

$$\Delta f(x, y) = f(x+1, y) + f(x-1, y) + f(y+1, x) + f(y-1, x) - 4f(y, x).$$

Маски фильтров для реализации дискретного Лапласиана имеют вид

0	0	0
1	–4	1
0	0	0

1	1	1
1	–8	1
1	1	1

Оператор Лапласа подчеркивает разрывы, скачки яркости и ослабляет плавное изменение яркости. Для повышения резкости изображение, обработанное оператором Лапласа, накладывают на исходное изображение.

$$g(x, y) = f(x, y) + \nabla^2 f(y, x).$$

Задание 6.10. Создать файл с изображением, в котором присутствуют перепады изображения. С помощью оператора Лапласа обнаружить и выделить эти перепады.

Скрипт [9]:

```
import cv2
img = cv2.imread('avt.jpg', 0)
img = cv2.resize(img, (900, 600))
laplacian = cv2.Laplacian(img, cv2.CV_64F) # Функция Лапласиана.
Cv2.imshow('Original', img)
cv2.imshow('laplacian', laplacian)
cv2.waitKey(0)
```

На выходе программы:

7. Нахождение и обработка контуров

Цель: обнаружение и выделение контуров на изображении, анализ контуров. Изучение функций **cv2.findContours()**, **cv2.drawContours ()**

Дискретный путь бинарного изображения между двумя пикселями – это последовательность пикселей, в которой при перемещении от начального пикселя к конечному пикселю каждый пиксель последовательности имеет два соседних пикселя, принадлежащих этой последовательности: предшествующий и последующий.

Контур бинарного изображения – это любой замкнутый путь. В случае бинарного изображения контур – это граница связанной области с одинаковой интенсивностью.

Контура используют для анализа формы бинарного изображения, с их помощью проводят обнаружение и распознавание объектов. Контуры удобно выделять в бинарном изображении, поэтому перед поиском контуров применяют пороговую обработку. Функция findContours изменяет исходное изображение. Поэтому исходное изображение нужно сохранить под другим именем. В OpenCV контур выделяется у белого объекта на черном фоне, поэтому объект должен быть белым, а фон – черным.

7.1. Обнаружение контуров

Для обнаружения контуров используют функцию **cv2.findContours ()**. Функция **image, contours, hierarchy=cv.findContours(image, mode, method [,contours [,hierarchy [,offset]]])** выводит изображение, контуры и иерархию: **image, contours, hierarchy**. Параметр **contours** обозначает список Python всех контуров изображения. Каждый отдельный контур – это массив **Numpy** с координатами (*x*, *y*) граничных точек объекта. Параметр **hierarchy** выводит иерархию контуров. Следует учитывать, что в зависимости от версии OpenCV могут использоваться только два параметра: **contours, hierarchy**. Если на выходе программы ошибка: ValueError: not enough values to unpack (expected 3, got 2), то в вашей версии питон параметр `image` нужно удалить.

Массив **contours** обычно переобозначают: **cnt = contours[0]**, где 0 указывает, что берется только внешний контур. В скобках функции **cv.findContours(image,mode,method [,contours[,hierarchy[,offset]]])**

три аргумента: первый – входное изображение, второй – режим поиска контура, третий – метод аппроксимации контура.

Для рисования контуров используется функция **cv2.drawContours**. Ее первым аргументом является исходное изображение, вторым аргументом являются контуры, которые следует передать в виде списка Python, третьим аргументом является индекс контуров (полезно при рисовании отдельного контура).

Чтобы нарисовать все контуры изображения, нужно использовать функцию **img = cv2.drawContours(img, contours,-1, (0,255,0),3)**.

Чтобы нарисовать конкретный контур, скажем 4-й, добавим:

cnt = contours[4],

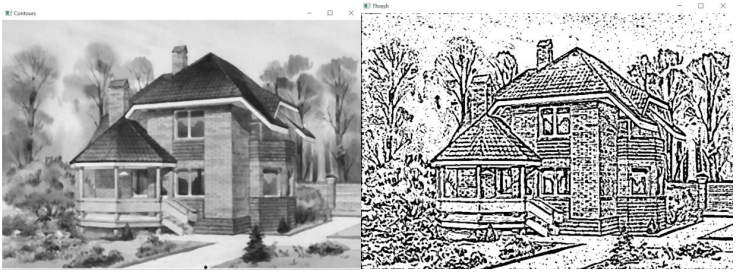
img = cv2.drawContours(img, [cnt], 0, (0,255,0), 3).

Задание 7.1. С помощью функции **cv2.findContours** найти все контуры изображения.

Скрипт:

```
import cv2
import numpy as np
img = cv2.imread('Dom.jpg', 0)
img = cv2.medianBlur(img, 5)
thresh =
cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE_THRESH_GAUS
SIAN_C,cv2.THRESH_BINARY,11,2)
cv2.imshow('Thresh', thresh)
contours, hierarchy = cv2.findContours(thresh.copy(),
cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE) # модуль кон-
туров
cnt = contours[4]
img = cv2.drawContours(img, [cnt], 0, (0,255,0), 3)
cv2.imshow('Contours', img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

На выходе программы:



7.2. Метод контурной аппроксимации

Очевидно, что хранить все координаты пикселей контура не рационально, занимает много памяти. Если контур включает в себя прямые отрезки, то достаточно задать начальную и конечную координату отрезка. В скобках функции поиска контура **cv2.findContours ()** имеется аргумент **cv2.CHAIN_APPROX_NONE**, где хранятся все граничные точки. Однако удобнее использовать аргумент **cv2.CHAIN_APPROX_SIMPLE**, с помощью которого можно удалить все лишние точки и сжать контур, сэкономив память.

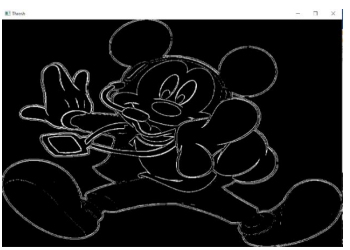
Задание 7.2. Протестировать функцию поиска контура **cv2.findContours** с аргументом **cv2.CHAIN_APPROX_SIMPLE**, который экономит память.

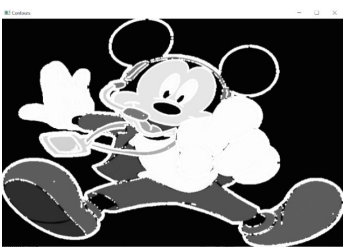
Скрипт:

```
import cv2
img = cv2.imread('avto.jpg', 0)
img = cv2.resize(img, (900, 600))
image = cv2.medianBlur(img, 5) # Функция размытия
(сглаживания) изображения медианным фильтром, где 5 –
параметр размытия (можно выбирать только нечетным и
больше 1)
thresh = cv2.adaptiveThreshold(img, 255,
cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY_INV,
3,10) # Превращаем входное изображения в бинарно-
инвертированное изображение. Здесь 255 – максимальное
значение, 3 – пороговое значение для пикселя (толщина
```

```
контура), 10 – пороговое значение для всего изображения
cv2.imshow(‘Thresh’, thresh)
contours, hierarchy = cv2.findContours(thresh.copy(),
cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE) # При исполь-
зовании cv2.CHAIN_APPROX_SIMPLE  экономим память
cv2.drawContours(image, contours, -1, (255,255,255),
3) # Функция рисует контур (на основе другого конту-
ра). Здесь «-1» – указывает, какой контур рисовать
(если -1, то рисуются все контура), 255,255,255 – цвет
контура (белый), 3 – толщина контура
cv2.imshow('Contours', image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

На выходе программы:





7.3. Выделение контуров методом Канни

Метод Канни обнаруживает границы связанной области изображения, выполняя поиск локальных максимумов градиента $f(x, y)$. Градиент вычисляется после применения фильтра Гаусса. Метод использует два порога для нахождения сильных и слабых краев. Слабые края включаются в выход, если они связаны с сильными.

Функция Канни **cv2.Canny(img, T_lower, T_upper,apertureSize = 3)** содержит в скобках следующие аргументы: img – входное изображение, T_lower: нижний порог, T_upper: верхний порог, apertureSize – размер апертуры оператора Собеля.

Алгоритм Канни.

1. Предварительное сглаживание по Гауссу для уменьшения шума.

2. Вычисление амплитуды и направления градиента изображения, обычно выбирается один из четырех возможных углов: 0º, 45°, 90° и 135°.

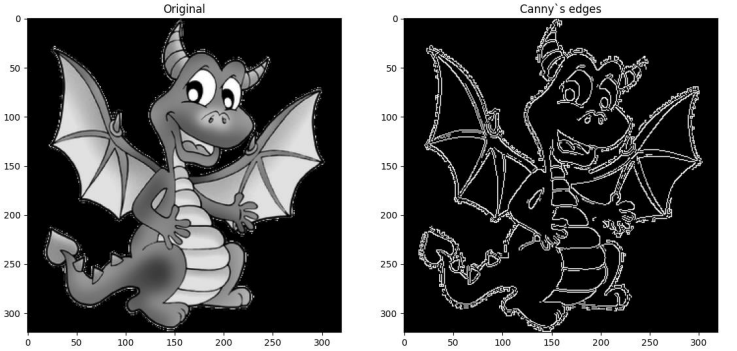
3. Не максимальное подавление: исключаются некраевые пиксели, это позволяет утончать края.

4. Использование порогового гистерезиса, для которого необходимы два порога: высокий и низкий порог. Если амплитуда градиента исследуемых пикселей больше, чем значения высокого порога, то эти пиксели резервируются как краевые пиксели. Если амплитуда положения пикселя меньше нижнего порога, то пиксель исключается. Если амплитуда интенсивности пикселя находится между двумя порогами, пиксель сохраняется только тогда, когда он подключен к пикселю выше верхнего порога.

Задание 7.3. Выделить границу методом Канни.

```
Import cv2
img = cv2.imread('avt.jpg',0)
img = cv2.resize(img, (900, 600))
edges = cv2.Canny(img,700,100,apertureSize = 3) # Функция объясняется в задании 2
cv2.imshow('Img', edges)
```

На выходе программы: оригинальное полутоновое изображение и изображение на выходе после выделения контуров методом Канни.



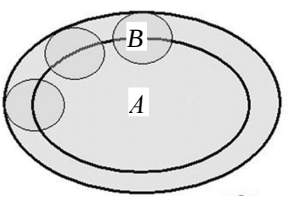
8. Морфологические преобразования

Цель: изучение различных морфологических операций, таких как эрозия, расширение, открытие, закрытие и т. д. Приобретение навыков работы с функциями: **cv2.erode ()**, **cv2.dilate ()**, **cv2.morphologyEx ()**.

Краткое теоретическое введение.

Морфологические преобразования – это преобразования множества пикселей одного изображения с помощью множества пикселей другого – структурирующего элемента или ядра. Имеются два основных морфологических оператора – это эрозия и расширение. Применение этих операторов в различных комбинациях дает новые морфологические преобразования: открытие, закрытие и т. д.

8.1. Дилатация (расширение)



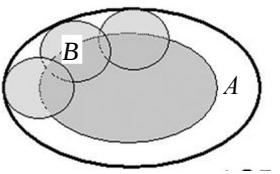
На рисунке B – это структурирующий элемент. Его центр должен лежать внутри или на границе множества A. Объединение всех множеств B дает новое множество – наращивание множества A по множеству B (на рисунке закрашено серым). Оно обозначается $A \oplus B$ и определяется как

$$A \oplus B = \bigcup_{a \in A} B_a .$$

Первый аргумент функции дилатации – это массив основного множества, второй – это ядро:

dilation = cv2.dilate(img,kernel,iterations = 1).

8.2. Эрозия



Операция эрозия – это объединение всех центральных пикселей ядра B, которое не выходит за пределы основного множества A, результат объединения на рисунке закрашен серым. Операция эрозия ужимает (утончает) изображения. Эрозия множества A по множеству B, обозначаемая $A \ominus B$, определяется как

$$A \ominus B = \{z \mid (B)_z \subset A\}.$$

Эрозия множества A по B – это множество всех таких точек $z \in A$, при сдвиге в которые центра множества B это множество целиком содержится во множестве A. Дилатация приводит к расширению изображения, эрозия к сжатию.

Первый аргумент функции эрозии – это массив основного множества, второй – это ядро:

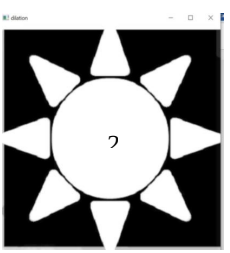
erosion = cv2.erode(img,kernel,iterations = 1).

Задание 8.1. Загрузить библиотеку **numpy**, файл `bin.jpg` и преобразовать его с помощью операций дилатация и эрозия. Выбрать ядро, размер которого равен последней цифре в номере списка группы. Здесь ядро 5×5. Выполним сначала операцию дилатации, затем и эрозии.

Скрипт [10]:

```
import cv2
import numpy as np
img = cv2.imread('bin.jpg',0)
kernel = np.ones((5,5),np.uint8)
dilation = cv2.dilate(img,kernel,iterations = 1)
erosion = cv2.erode(img, kernel,iterations = 1)
cv2.imshow(erosion)
```

На выходе программы: 1) исходное изображение, 2) после дилатации, 3) после эрозии:



Следует обратить внимание, что после применения операций дилатации и эрозии расширяются и сужаются светлые области. Так, в качестве примера рассмотрим обработку номерных знаков:

Из анализа рисунка видно, что после применения эрозии сужается светлая область, а буквы и цифры номерного знака расширяются.

8.3. Открытие, размыкание

Операция открытия, размыкания – это объединение всех множеств примитивов, находящихся внутри основного множества:

$$A \circ B = \bigcup \{(B)_z \mid (B)_z \subseteq A\}.$$

Открытие – это комбинация операций эрозии, за которой следует расширение. Размыкание сглаживает контуры объекта, обрывает узкие перешейки и ликвидирует выступы небольшой ширины. Размыкание множества A по примитиву B обозначается $A \circ B$ и определяется равенством

$$A \circ B = (A \ominus B) \oplus B.$$

Для открытия используется функция **cv2.morphologyEx ()**

opening=cv2.morphologyEx(img, cv2.MORPH_OPEN, kernel).

На рисунке 13 виден результат после применения операции открытия. Углы изображения становятся сглаженными.

Операция открытия полезна для удаления шума.

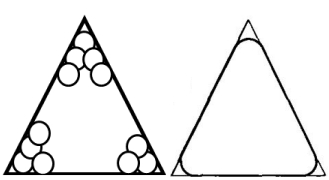


Рис. 13. Обработка изображения ядром в процессе операции открытия

Задание 8.2. Для демонстрации удаления шума создать зашумленный файл, затем к зашумленному файлу применить операцию открытия.

Скрипт:

```
import cv2 # Подключение библиотеки
import random
from PIL import Image, ImageDraw
image = Image.open('avto.jpg') # Открываем изображение
draw = ImageDraw.Draw(image) # Создаем инструмент для
рисования
width = image.size[0]  # Определяем ширину
height = image.size[1]  # Определяем высоту
pix = image.load()  # Выгружаем значения пикселей
for i in range(width):
    for j in range(height):
        rand = random.randint(0, 200)
        a = pix[i, j][0] + rand
        b = pix[i, j][1] + rand
        c = pix[i, j][2] + rand
        if (a > 255):
            a = 255
        if (b > 255):
            b = 255
        if (c > 255):
            c = 255
        draw.point((i, j), (a, b, c))
image.save("median.png", "JPEG") # сохранение
cv2.waitKey(0) # Задержка
```

На рисунке 14 показано удаление шума типа соль-перец. На выходе программы:

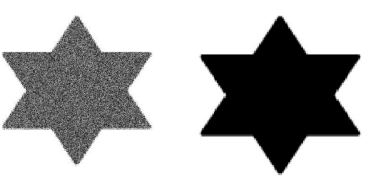


Рис. 14. Удаление шума типа соль-перец

8.4. Закрытие, замыкание

Замыкание также сглаживает участки контура, но в отличие от размыкания «заливает» узкие разрывы и углубления малой ширины, ликвидирует небольшие отверстия или маленькие черные точки на объекте. Замыкание множества A по примитиву B обозначается $A \bullet B$ и определяется как

$$A \bullet B = (A \oplus B) \ominus B.$$

Закрытие (замыкание) – это комбинация операций: сначала расширение потом эрозия.

На рисунке 15 показана последовательность создания замыкания на изображении.

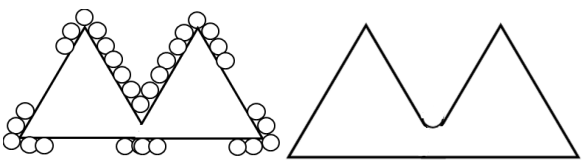


Рис. 15. Процесс создания замыкания на изображении

Задание 8.3. Трансформировать цветное изображение в полутоновое при его загрузке, к полутоновому файлу применить операцию открытия.

Скрипт [10]:

```
import cv2
import numpy as np
img = cv2.imread('bin.jpg',0)
img1 = cv2.imread('bin1.jpg',0)
kernel =
cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(30,30))
open = cv2.morphologyEx(img, cv2.MORPH_OPEN, kernel)
clos = cv2.morphologyEx(img1, cv2.MORPH_CLOSE, kernel)
cv2.imshow('imag', img)
cv2.imshow('open', open)
cv2.imshow('clos ', clos)
cv2.waitKey(0)
```

На выходе программы:

8.5. Морфологический градиент

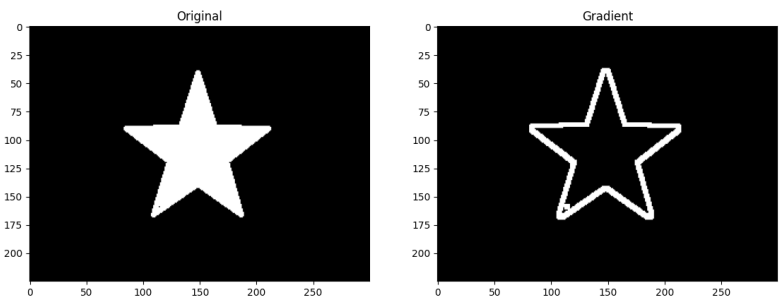
Находится разность между расширением и эрозией изображения.

Скрипт:

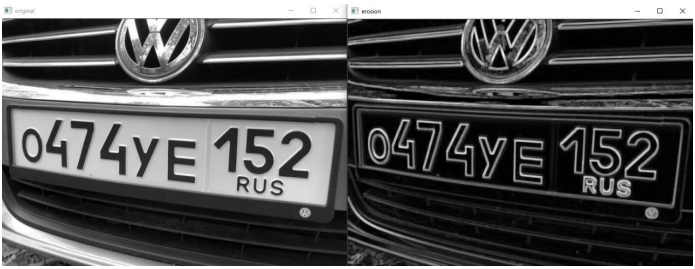
```
import cv2
import numpy as np
grad=cv2.morphologyEx(img, cv2.MORPH_GRADIENT,kernel)
```

Задание 8.4. Трансформировать цветное изображение в полутоновое при его загрузке. Скопировать полутоновое изображение. К первому изображению применить операцию расширения, ко второму эрозию. Затем вычесть из расширенного изображения изображение после эрозии. Результат похож на контур объекта.

Скрипт:



Применение морфологического градиента к номерному знаку дает следующий рисунок:



8.6. Цилиндр

Находится разность между входным изображением и открытием изображения. Ниже приведен пример для ядра 50×50.

Цилиндр множества A по примитиву B определяется равенством

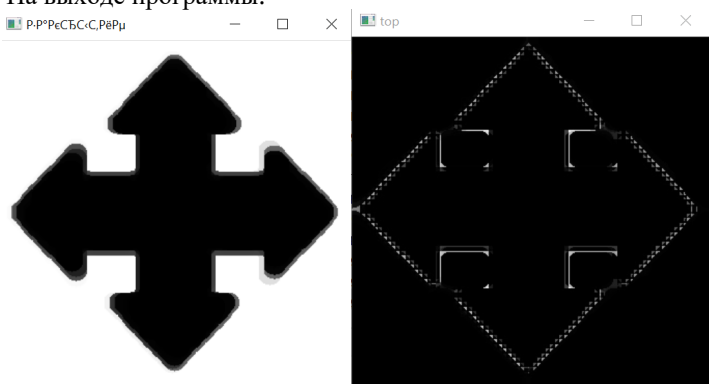
$$A - (A \ominus B) \oplus B .$$

Скрипт [10]:

```
import cv2
import numpy as np
top = cv2.morphologyEx(img, cv2.MORPH_TOPHAT, kernel).
```

Задание 8.5. Применить операцию цилиндр к изображению, размер ядра равен 40 + №, № – номер по списку группы.

На выходе программы:



8.7. Черная шляпа

Это разница между закрытием входного изображения и входным изображением.

Черная шляпа множества A по примитиву B определяется равенством

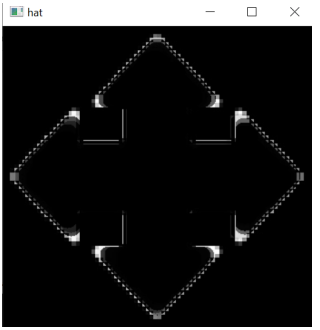
$$(A \oplus B) \ominus B - A.$$

Задание 8.6. Применить операцию черная шляпа к изображению, размер ядра равен 40 + №, № – номер по списку группы.

Скрипт [10]:

```
import cv2
import numpy as np
hat=cv2.morphologyEx(img,
cv2.MORPH_BLACKHAT,kernel).
```

На выходе программы результат применения операции черная шляпа:



8.8. Обработка изображения с помощью структурирующего элемента (ядра)

Размер ядра структурирующего элемента выбрать из ряда 3×3, 3×5, 5×3, 5×5, 5×7, 3×7, 7×3, 7×5, 5×7, 7×7, номер варианта должен быть равен номеру по списку группы.

В предыдущих примерах элементы структурирования создались вручную с помощью Numpy. Это прямоугольная форма. Но в некоторых случаях вам могут понадобиться ядра эллиптической/круглой формы. Для этой цели в OpenCV есть функция **cv2.getStructuringElement ()**. Вы просто передаете форму и размер ядра и получаете желаемое ядро. Примеры различных ядер:

```
# Rectangular Kernel
import cv2
import numpy as np
kernel =
cv2.getStructuringElement(cv2.MORPH_RECT,(5,5))
print(kernel)
```

```
array([[1, 1, 1, 1, 1],
    [1, 1, 1, 1, 1],
    [1, 1, 1, 1, 1],
    [1, 1, 1, 1, 1],
    [1, 1, 1, 1, 1]], dtype=uint8)

# Elliptical Kernel
kernel=cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(5,5))
print(kernel)
array([[0, 0, 1, 0, 0],
    [1, 1, 1, 1, 1],
    [1, 1, 1, 1, 1],
    [1, 1, 1, 1, 1],
    [0, 0, 1, 0, 0]], dtype=uint8)

# Cross-shaped Kernel
kernel=cv2.getStructuringElement(cv2.MORPH_CROSS,(5,5))
print(kernel)
array([[0, 0, 1, 0, 0],
    [0, 0, 1, 0, 0],
    [1, 1, 1, 1, 1],
    [0, 0, 1, 0, 0],
    [0, 0, 1, 0, 0]], dtype=uint8)
```

Задание 8.7. Изготовить ядро, его размер выбрать из ряда 3×3, 3×5, 5×3, 5×5, 5×7, 3×7, 7×3, 7×5, 5×7, 7×7, номер варианта должен быть равен номеру по списку группы.

Обработать изображение с помощью выбранного ядра и ядра размером 9×9. Сравнить результаты обработки изображения этими ядрами.

Скрипт [10]:

```
import cv2
import numpy as np
imge = cv2.imread('av4.jpg',0)
img = cv2.resize(imge, (900, 600))
close_size = 20
```

```
kernel = cv2.getStructuringElement(cv2.MORPH_RECT,
(close_size, close_size))
result = cv2.morphologyEx(img, cv2.MORPH_CLOSE, ker-
nel)
cv2.imshow('Img',result)
cv2.waitKey(0)
```

Результат для структурирующего элемента:

[0, 0, 1, 0, 0],
[0, 0, 1, 0, 0],
[1, 1, 1, 1, 1],
[0, 0, 1, 0, 0],
[0, 0, 1, 0, 0]

Результат для структурирующего элемента:

```
[[0 0 1 0 0]
 [1 1 1 1 1]
 [1 1 1 1 1]
 [1 1 1 1 1]
 [0 0 1 0 0]]
```

На выходе программы:

9. Проект

При изучении любой дисциплины большое значение имеет практическое применение накопленных знаний, формирование навыков их использования. На первом этапе изучаются основные понятия, простые операции. Затем ставятся более сложные задачи, которые включают в себя некоторую последовательность этих операций. Для

решения таких задач необходимо разработать алгоритм применения простых операций. В этом разделе решается задача, в которой необходимо комплексное использование функций цифровой обработки изображений, которые изучены в приведенных выше заданиях.

Постановка задачи. Выделить на фоне всего изображения интересующие нас объекты. Создать таблицу признаков для сформированного набора объектов. Провести распознавание этих объектов с помощью нейронной сети.

9.1. Алгоритм создания таблицы признаков для множества объектов

1. На первом этапе проводится предварительная обработка изображений, которая включает в себя удаление шума, повышение резкости изображений. С помощью этой обработки выделяются характерные детали, подавляется шум, повышается быстродействие, уменьшается объем информации.

2. Следующий шаг – удалении фона на изображении, для этого сканируется все пространство изображения и отсканированные пиксели с одинаковой интенсивностью обнуляются. В результате интересующие нас объекты будут более четко выделены на черном фоне.

3. Для выделения объектов используется операция сегментации изображения методом водораздела, с последующей маркировкой результата сегментации.

4. Для более точного выделения объектов их нужно разнести, увеличив расстояние между ними.

5. С помощью операции распознавания объекта по шаблону каждый объект охватывается прямоугольной рамкой.

6. Создание изображения каждого объекта отдельно.

7. Каждый объект, после его отделения от других объектов, масштабируется так, чтобы у всех объектов была одинаковая высота.

8. Для каждого объекта после масштабирования вычисляются его характерные признаки. Процесс определения характерных признаков заключается в следующем:

- получение из цветного изображения полутонового объекта;
- обработка полутонового изображения, удаление шума с помощью морфологических операций;
- бинаризация полутонового изображения пороговым методом;
- выделение контура у каждого объекта;

• создание на основе бинарного изображения объекта основных его признаков.

9. Формирование таблицы признаков множества объектов.

В проекте импортируем библиотеки и модули:

```
-*-coding:utf8  – * – # позволяет читать надписи
на русском языке.
import cv2
import numpy as np
import imutils
import random
from collections import Counter # импорт модуля контуров
from sklearn.cluster import KMeans # модуль кластерного
анализа
```

9.2. Обзор функций, которые используются в проекте

Сама программа состоит из нескольких функций, к которым она может неоднократно обращаться в процессе работы. Перед запуском всей программы необходимо отладить работу каждой функции этой программы.

Функция def shelf () предназначена для удаления фона. В процессе работы этой программы сканируется все строки, начиная с верхней. Если при сканировании интенсивности пикселей не меняются, то они обнуляются. В результате на изображении фон становится темным.

Код функции:

```
import cv2import
numpy as np
  def shelf (img, k = 120):
     x = 0
     for i in range(img.shape[1]):
       if x > ((img.shape[1]) /1.01):
          img[i-1]=[0, 0, 0]
          img[i-2]=[0, 0, 0]
```

```
      x = 0
      for j in range(img.shape[0]):
        if (img[i][0][0]-k <= img[i][j][0] <=
img[i][0][0]+k) and (img[i][0][1]-k <= img[i][j][1] <=
img[i][0][1]+k) and (img[i][0][2]-k <= img[i][j][2] <=
img[i][0][2]+k:
                    x + = 1
img = cv2.imread('prg.jpg')
img = cv2.resize(img, (800, 800))
img2 = img.copy()
img = shelf(img, k = 120)
cv2.imshow("shelf", img)
cv2.waitKey(0)
```

Функция удаления фона 2.

```
def background (imr, p = 40):
  hsv_image = cv2.cvtColor(imr, cv2.COLOR_BGR2HSV)
  dom_color = get_color(hsv_image)
  dom_color_hsv=np.full(imr.shape,[255,255,255],
dtype='uint8'
  lower = np.array([dom_color[0]-p,dom_color[1]-p,
dom_color[2]-p])
  upper= np.array([dom_color[0]+p,dom_color[1]+p,
dom_color[2]+p])
  mask=cv2.inRange(hsv_image, lower, upper)
  res=cv2.bitwise_not(dom_color_hsv,hsv_image,
mask= mask
  return cv2.cvtColor(res, cv2.COLOR_HSV2BGR)
image = cv2.imread('polk.jpg')
image = cv2.resize(image, (700, 700))
img2 = image.copy()
```

```
imag=get_color(image, k=4)
imr = background(image, p = 40)
cv2.imshow("shelf", imr)
```

На рисунке 16 приведены изображения в результате работы программы до и после удаления фона.

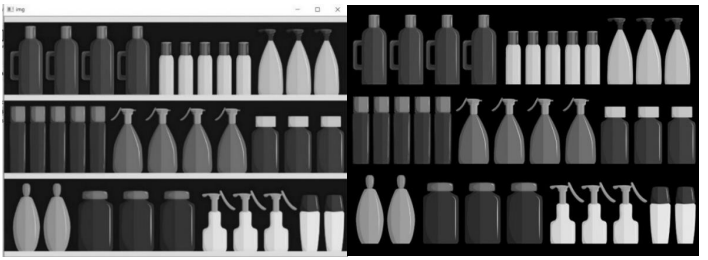


Рис. 16. Изображения до и после удаления фона

Сегментация изображения – это разбиение изображения на множество покрывающих его областей, называемых сегментами. С ее помощью можно выделить объекты, создавая границы между ними. Далее для операции сегментации используется метод водораздела. После сегментации изображения проводится маркировка всех сегментов изображения.

Описание функции **segment()**. Функция segment() предназначена для разбиения всего поля изображения на сегменты, с последующей их маркировкой:

def segment (img):

С помощью функции cv2.cvtColor цветное изображение ims трансформируется в полутоновое изображение gray:

gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY).

Используя функцию cv2.threshold, получим, задавая порог 30, бинарное изображение:

```
im_bw = cv2.threshold(gray, 30, 255, cv2.THRESH_BINARY)[1]
```

Выбираем формат uint8:

```
kernel = np.ones((4,3), np.uint8)
```

Морфологическая функция `cv2.morphologyEx( )` используется в режиме открытия MORPH_OPEN. Операция открытия – это по-

следовательность двух операций: сначала применяется операция эрозии, потом операция расширения. На рисунке видно, что с помощью этой операции хорошо удаляется шум.

Применим эту функцию к изображению imbw в режиме открытия:

opening = cv2.morphologyEx(im_bw, cv2.MORPH_OPEN, kernel, iterations = 1).

Затем полученное изображение обработаем операцией закрытия:

sure_bg = cv2.dilate(opening, kernel, iterations=1).

После бинаризации изображения и очистки его от шума увеличим расстояние между объектами. Для этого применим операцию distanceTransform:

dist_transform = cv2.distanceTransform(sure_bg,cv2.DIST_L2,3)

К полученному изображению применим пороговую обработку:

```
ret,sure_fg=cv2.threshold(dist_transform,
0.01*dist_transform.max(), 255, 0)
sure_fg = np.uint8(sure_fg)
```

На полученном изображении, используя метод Canny, выделим все контуры.

```
unknown = cv2.Canny(sure_bg,700,100,apertureSize=3)
```

Маркерная маркировка – это присвоение каждой области внутри замкнутого контура некоторого символа. Таким образом создается база промаркированных изображений, необходимая для распознавания.

```
ret,mark=cv2.connectedComponents(sure_fg)
```

Переобозначим маркеры, чтобы фон был l:

```
маркеры = маркеры + 1
    mark=mark+1
```

Области неопределенности присвоим нулевые маркеры

```
mark[unknown ==255] = 0
```

Сегментация изображения проводится методом водораздела

```
markers = cv2.watershed(img, mark)
```

Конец блока:

```
    return img
```

Полный код для проверки функции сегментации:

```
import cv2
import numpy as np
from sklearn.cluster import KMeans
from collections import Counter
import imutils
import random
def segment (img):
  gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
  im_bw = cv2.threshold(gray, 30, 255,
cv2.THRESH_BINARY)[1]
  cv2.imshow("im_bw ", im_bw)
  kernel = np.ones((4,3),np.uint8)
  opening = cv2.morphologyEx(im_bw,cv2.MORPH_OPEN,
kernel, iterations = 1)
  sure_bg = cv2.dilate(opening,kernel,iterations=1)
  dist_transform = cv2.distanceTransform(sure_bg,
cv2.DIST_L2,3)
  ret, sure_fg = cv2.threshold(dist_transform,
0.01*dist_transform.max(),255,0)
  sure_fg = np.uint8(sure_fg)
  unknown=cv2.Canny(sure_bg,700,100,apertureSize=3)
  cv2.imshow("Canny", unknown)
  ret, mark=cv2.connectedComponents(sure_fg)
  mark = mark+1
  mark[unknown==255] = 0
  markers = cv2.watershed(img,mark)
  img [markers == -1] = [255,0,0]
  return img
imgor=cv2.imread('D:\\py\\Lab5\\orig.jpg')
cv2.imshow("imgor", imgor)
```

```
img = cv2.resize(imgor, (800, 800))
img2 = img.copy()
marker=segment(img) #print(marker)
cv2.imshow("Output", marker)
cv2.waitKey(0)
```

На рисунках 17, 18 представлены изображения объектов после применения функции segment.

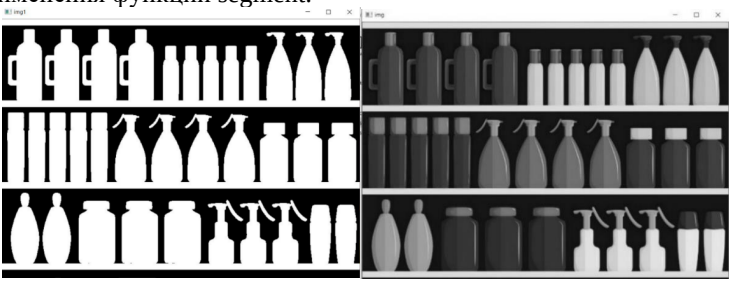


Рис. 17. Изображение на входе программы и изображение после бинаризации

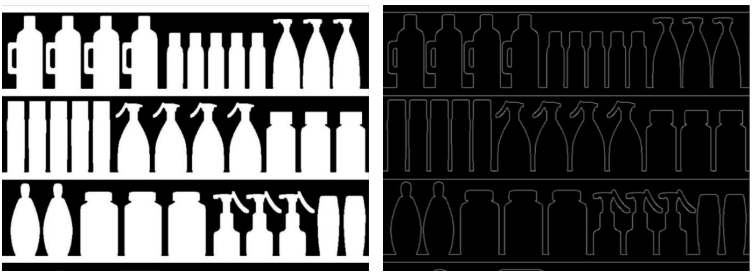


Рис. 18. Изображение после преобразования расстояния и удаления шума (слева), выделение контуров методом Канни (справа)

После удаления фона, разнесения объектов каждый объект выделяется, берется в рамочку. Дальнейшая обработка объекта проводится в области, заключенной в рамку.

Распознавание объекта по шаблону. Метод заключается в поиске объекта на большом изображении, который соответствует выбранному шаблону. Поиск проводится с помощью функции

cv2.matchTemplate(). Шаблон перемещается по исходному изображению, его пиксели сравниваются с пикселями этого изображения. На выходе получаем изображение в градациях серого. На рисунках 19, 20 демонстрируется работа функции поиска объекта по шаблонному изображению.

Рис. 19. Исходное изображение и шаблонное изображение

В случае, когда на изображении несколько одинаковых объектов, каждый выделяется отдельно (см. рис. 20).

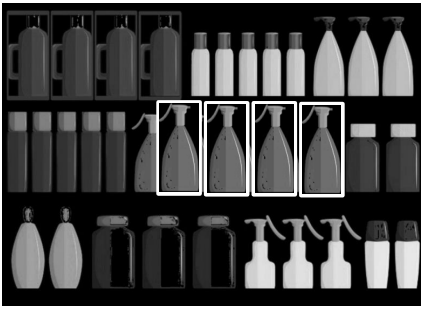


Рис. 20. Результат работы функции поиска объекта по шаблонному изображению

Скрипт для поиска по шаблону:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
img_rgb = cv2.imread('img.jpg')
```

```
img_gray = cv2.cvtColor(img_rgb, cv2.COLOR_BGR2GRAY)
template = cv2.imread('1.jpg',0)
w, h = template.shape[::-1]
res = cv2.matchTemplate(img_gray,template,
cv2.TM_CCOEFF_NORMED)
threshold = 0.8
loc = np.where(res >= threshold)
for pt in zip(*loc[::-1]):
    cv2.rectangle(img_rgb, pt, (pt[0]+w, pt[1] + h),
(0,0,255), 2)
    cv2.imwrite('res.png',img_rgb)
cv2.imshow('res.png',img_rgb)
```

Приведение объектов распознавания к единому размеру. Для правильного создания признаков объектов их нужно привести к одинаковому размеру по высоте. Для этого у каждого объекта определяем коэффициент пересчета $k = h1/h0$, где $h0$ – стандартная высота, в нашем случае $h0 = 160$, $h1$ – старая высота. Тогда новая ширина равна $w = w1* h1/h0$. После изменения размера записываем матрицу изображения в новый файл.

Скрипт:

```
image = cv2. imread('r1.jpg',1)
s=image.shape # Получаем размер исходного изображения
h1,w1=s[0],s[1] # Запоминаем отдельно высоту и ширину
wr=77*h1/160
img = cv2.resize(image, (wr, 160))# изменяем размер
cv2.imwrite("r1.jpg", img)
print (s)
print (w1)
```

9.3. Создание таблицы признаков

Далее для распознавания объектов используются понятия и методы теории классификации.

Признак – это количественная характеристика того или иного свойства исследуемого объекта.

Образ – это совокупность признаков, с помощью которых объект идентифицируется с большой вероятностью.

Класс образова – это совокупность образов с общими свойствами.

Класс образов – характеризует наличие «схожести» объектов.

Чтобы различать классы, каждый класс удобно обозначить идентифицирующей меткой.

Если у объекта N признаков, то для их цифровой обработки необходимо использовать N-мерное пространство признаков. Каждый признак в этом пространстве описывается вектором с компонентами (x_1, x_2, ..., x_N), которые являются значениями признаков этого объекта.

Распознавание – это процесс сравнения признаков некоторого неизвестного объекта, с набором признаков отмаркированных объектов, хранящихся в базе, в результате которого определяется принадлежность неизвестного объекта к определенному классу.

В процессе распознавания используют наиболее характерные признаки. К ним можно отнести следующие:

– площадь объекта (сумма всех пикселей внутри контура);

– высота и ширина прямоугольника, в котором заключено изображение;

– отношение ширины к высоте ограничивающего прямоугольника объекта;

– отношение площади контура к площади ограничивающего прямоугольника;

– эквивалентный диаметр;

– выделенная ось объекта – это ось с минимальным моментом инерции, проходящая через центр тяжести;

– моменты пикселов второго и более высокого порядка относительно главной оси.

Для создания таблицы признаков объекта цветное изображение трансформируется в серое, к которому, используя функцию

```
cv2.adaptiveThreshold(img, 255,
cv2.ADAPTIVE_THRESH_MEAN_C,cv2.THRESH_BINARY,11,2)
```

применим операцию бинаризации. Первый аргумент – матрица серого изображения; второй – значение интенсивности монохромного изображения в случае, когда интенсивность серого изображения больше порога, если меньше, то интенсивность монохромного изоб-

ражения равна нулю; третий задает адаптивную бинаризацию серого изображения.

На рисунке 21 приводится изображения после адаптивной пороговой бинаризации и обнаружения контура.

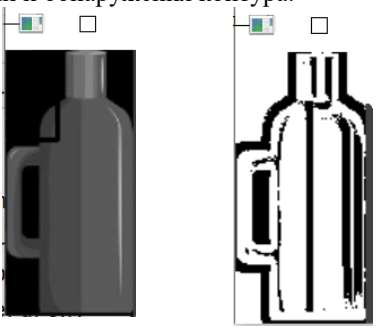


Рис. 21. Изображения после адаптивной пороговой бинаризации и обнаружения контура

На основе массива точек изображения и массива координат контура вычисляются все признаки объекта.

Скрипт для вычисления признаков:

```
import cv2 # Загружаем библиотеку opencv
import numpy as np
img = cv2. imread('1.jpg',0)
th =
cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE_THRESH_MEAN
_C,cv2.THRESH_BINARY,11,2)
contours, hierarchy = cv2.findContours(th, 5, 5)
cv2.imshow('binar', th)
cv2.waitKey(0)
cnt = contours[0] # создание контура
area = cv2.contourArea(cnt)
prm = cv2.arcLength(cnt,True)
hull = cv2.convexHull(cnt) #выпуклая оболочка
#print(hull)
x,y,w,h = cv2.boundingRect(cnt)
print(x,y,w,h)
imag = cv2.rectangle(img,(x,y),(x+w,y+h),(0,255,0),2)
cv2.imshow('Rectangle', imag)
# параметры бинарного изображения
```

```
x, y, w, h = cv2.boundingRect(cnt)
epsilon = 0.1 * cv2.arcLength(cnt, True)
aspect_ratio = float(w) / h # соотношение сторон
rect_area = w * h
extent = float(area) / rect_area
hull_area = cv2.contourArea(hull) #площадь выпуклой
оболочки
equi_diameter = np.sqrt(4*area/np.pi)
mask = np.zeros(img.shape, np.uint8)
cv2.drawContours(mask, [cnt], 0, 255, -1)
M = cv2.moments(cnt)
x,y,w,h = cv2.boundingRect(cnt)
#print(M) # вычисление моментов
pixelpoints = np.transpose(np.nonzero(mask))
min_val, max_val, min_loc, max_loc =
cv2.minMaxLoc(img, mask=mask)
leftmost = tuple(cnt[cnt[:, :, 0].argmin()][0])
rightmost = tuple(cnt[cnt[:, :, 0].argmax()][0])
topmost = tuple(cnt[cnt[:, :, 1].argmin()][0])
bottommost = tuple(cnt[cnt[:, :, 1].argmax()][0])
print(area) # вычисление площади
print(prm) # вычисление периметра
print(w,h)
print(aspect_ratio, extent)
print(equi_diameter)
print(M)
#print(pixelpoints)
#print(min_val, max_val, min_loc, max_loc)
#print(leftmost, rightmost, topmost, bottommost)
cv2.waitKey(0)
```

Данные на выходе этой программы для каждого объекта заносятся в таблицу признаков 1. В этой таблице объектов используются следующие обозначения: *s* – площадь, *p* – периметр, *w* – ширина, *h* – высота, *w*/*h* – отношение ширины к высоте, s/(*wh*), отношение, *d* – эквивалент диаметр, *m*00, *m*01, *m*10, *m*11 – моменты, определяющие площадь, центр масс объекта, и другие моменты более высокого порядка.

Таблица 1. Признаки объектов

	Объекты							
Признаки	объект 1	объект 2	объект 3	объект 4	объект 5	объект 6	объект 7	объект 8
s	1249,0	577,0	180,5	7795,5	163,0	111,0	909,5	29,5
p	157,55	141,34	90,476	1262,1	61,799	78,064	175,47	48,730
w	31	51	7	66	15	5	24	5
h	54	26	41	158	22	37	70	22
w/h	0,574	1,9615	0,1707	0,4177	0,6818	0,1351	0,3429	0,2272
s/wh	0,7461	0,4351	0,6289	0,7475	0,4939	0,6	0,5414	0,2682
d	39,878	27,105	15,160	99,627	14,406	11,888	34,029	6,1287
*m*00	1249,0	577,0	180,5	7795,5	163,0	111,0	909,5	29,
*m*10	15 994	18 479	3319,8	255 475	790,83	174,5	8059,0	40,833
*m*01	27 883	4990,2	1440,7	616 222	1220,3	2033,3	21 165	160,5
*m*11	28 880× ×10^2	67 700× ×10	7943,3	109 123× ×10^2	5623,0	372,16	10 442× ×10	87,917
Результаты	35 515	20 110	559,2	20 507× ×10^2	722,2	595,7	18 532	53,42

9.4. Распознавание объектов с помощью нейронной сети

Классификация и распознавание объектов проводится по данным таблицы признаков, после их обработки с помощью нейронной сети. Значения признаков, взятые из таблицы для каждого объекта, поступают на вход нейронной сети. Значения на выходе нейронной сети от каждого набора признаков объекта приводятся в последней строке таблицы «Результаты».

Алгоритм распознавания объектов.

1. По вычисленным признакам всех объектов проводится обучение нейросети, т. е. для каждого объекта по его признакам определяется значение на выходе нейросети и заносится в последнюю строку таблицы. Таким образом каждому объекту сопоставляется маркер.

2. Сравнение данных из базы признаков с признаками неизвестного объекта позволяет распознать этот объект и определить его маркер.

Скрипт нейронной сети:

```
import cv2
import numpy as np
```

```
s=[1249.0,577.0, 180.5, 7795.5, 163.0, 111.0, 909.5,
29.5]
p=[157.55,141.34,90.476,1262.1,61.799,78.064,175.47,48
.730]
w=[31, 51, 7, 66, 15, 5, 24, 5]
h=[54, 26, 41, 158, 22,   37, 70, 22]
w/h
=[0.574,1.9615,0.1707,0.4177,0.6818,0.1351,0.3429,0.22
72]
s/wh=[0.7461,0.4351,0.6289,0.7475,0.4939,0.6,0.5414,0.
2682]
d=[39.878,27.105,15.160,99.627,14.406,11.888,34.029,6.
1287]
m00=[1249.0,577.0,180.5,7795.5,163.0,111.0,909.5,29.5]
m10=[15994,18479,3319.8,255475,790.83,174.5,8059.0,40.
833]
m01=[27883,4990.2,1440.7,616222,1220.3,2033.3,21165,16
0.5]
m1=[307788,175877,180.5,19613900,4657.8,3300.2,153434,
152.375]

weights = [0.3, 0.5, 0.1, 0.2]
def sum(a,b):
    assert(len(a) == len(b))
    output = 0
    for i in range(len(a)):
        output+=(a[i]*b[i])
    return output
def art_neuron(input,weights):
    pred = sum(input,weights)
    return pred
#vek=[3, 2, 4, 3]
#pred_v = sum(vek,weights)
s=[1249.0,577.0, 180.5, 7795.5, 163.0, 111.0, 909.5,
29.5]
p=[157.55,141.34,90.476,1262.1,61.799,78.064,175.47,48
.730]
w=[31, 51, 7, 66, 15, 5, 24, 5]
h=[54, 26, 41, 158, 22,   37, 70, 22]
```

```
kw
=[0.574,1.9615,0.1707,0.4177,0.6818,0.1351,0.3429,0.22
72]
ks=[0.7461,0.4351,0.6289,0.7475,0.4939,0.6,0.5414,0.26
82]
d=[39.878,27.105,15.160,99.627,14.406,11.888,34.029,6.
1287]
m0=[1249.0,577.0,180.5,7795.5,163.0,111.0,909.5,29.5]
m1=[15994,18479,3319.8,255475,790.83,174.5,8059.0,40.8
33]
m2=[27883,4990.2,1440.7,616222,1220.3,2033.3,21165,160
.5]
m3=[307788,175877,180.5,19613900,4657.8,3300.2,153434,
152.375]
in0=[s[0],p[0],w[0],h[0],kw[0],ks[0],d[0],m0[0],m1[0],
m2[0], m3[0]]
in1=[s[1],p[1],w[1],h[1],kw[1],ks[1],d[1],m0[1],m1[1],
m2[1], m3[1]]
in2=[s[2],p[2],w[2],h[2],kw[2],ks[2],d[2],m0[2],m1[2],
m2[2], m3[2]]
in3=[s[3],p[3],w[3],h[3],kw[3],ks[3],d[3],m0[3],m1[3],
m2[3], m3[3]]
in4=[s[4],p[4],w[4],h[4],kw[4],ks[4],d[4],m0[4],m1[4],
m2[4], m3[4]]
in5=[s[5],p[5],w[5],h[5],kw[5],ks[5],d[5],m0[5],m1[5],
m2[5], m3[5]]
print(in0)
print(in5)
pred0 = art_neuron(in0,weights)
pred1 = art_neuron(in1,weights)
pred2 = art_neuron(in2,weights)
pred3 = art_neuron(in3,weights)
pred4 = art_neuron(in4,weights)
pred5 = art_neuron(in5,weights)
#print(pred_v)
print(pred0)
print(pred1)
print(pred2)
print(pred3)
```

```
print(pred4)
print(pred5)
```

На выходе программы:

```
35515
20110
559.19
2050730
722.17
595.72
18532.4
53.4177
```

По значениям на выходе нейронной сети можно провести распознавание неизвестного объекта из множества объектов, перечисленных в таблице. На рисунке 22 изображена схема нейронной сети, с помощью которой можно провести распознавание объектов.

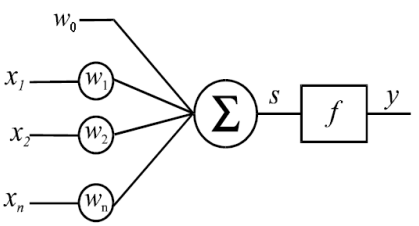


Рис. 22. Схема нейронной сети

На этой схеме обозначены: w_n – веса признаков, x_n – значения признаков, S – сигнал на выходе сумматора, $f(S)$ – активационная функция, y – выходной сигнал.

Значение на выходе нейронной сети для k объекта определяется по формуле

$$y_k = \sum_{i=1}^{n} w_i x_i^k ,$$

где x_i^k – значения признаков k объекта, y_k – сигнал на выходе нейронной сети для k объекта.

Для того чтобы провести распознавание объектов, сопоставим каждому объекту маркеры в виде числовых значений на выходе

нейронной сети. С изменением освещенности объекта значения его признаков и значение на выходе нейронной сети также меняются в небольших пределах. Поэтому для более достоверного распознавания объектов установим границы числовых значений маркеров для каждого объекта (см. табл. 2). Пусть y_k и y_{k+1} – два соседних значения на выходе нейронной сети. Разобьем диапазон всех значений на выходе нейронной сети на интервалы. Граница между интервалами находится по формуле

$$g_k = (y_{k+1} - y_k)/2.$$

Занесем вычисленные значения для границ в таблицу 2.

Таблица 2. Диапазон числовых значений базы маркеров

	Объекты	Диапазон значений базы маркеров
1	объект 1	0–306
2	объект 2	306–578
3	объект 3	578–659
4	объект 4	659–9627
5	объект 5	9627–19321
6	объект 6	19321–27816
7	объект 7	27816–1043107
8	объект 8	>1043107

С помощью этой таблицы в зависимости от того, в какой диапазон числовых значений базы маркеров попадает значение на выходе нейронной сети неизвестного объекта, проводится распознавание объекта.

Литература

1. *Клетте, Р.* Компьютерное зрение. Теория и алгоритмы / пер. с англ. А. А. Слинкин. – М. : ДМК Пресс, 2019. – 506 с.
2. *Шапиро, Л.* Компьютерное зрение / Л. Шапиро, Дж. Стокман / пер. с англ. – М. : БИНОМ. Лаборатория знаний, 2006. – 752 с.
3. OpenCV Tutorials [Электронный ресурс] // Open Source Computer Vision. – URL : https://docs.opencv.org/3.4/d9/df8/tutorial_root.html (дата обращения : 25.04.2022).
4. Contour Features [Электронный ресурс] // Open Source Computer Vision. – URL : https://docs.opencv.org/3.4/dd/d49/tutorial_py_contour_features.html (дата обращения : 25.04.2022).
5. Contour Properties [Электронный ресурс] // Open Source Computer Vision. – URL : https://docs.opencv.org/4.x/d1/d32/tutorial_py_contour_properties.html (дата обращения : 25.04.2022).
6. Contours : More Functions [Электронный ресурс] // Open Source Computer Vision. – URL : https://docs.opencv.org/3.4/d5/d45/tutorial_py_contours_more_functions.html (дата обращения : 25.04.2022).
7. Image Thresholding [Электронный ресурс] // Open Source Computer Vision. URL : https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html (дата обращения : 25.04.2022).
8. Basic Thresholding Operations [Электронный ресурс] // Open Source Computer Vision. URL : https://docs.opencv.org/3.4/db/d8e/tutorial_threshold.html (дата обращения : 25.04.2022).
9. Canny Edge Detector [Электронный ресурс] // Open Source Computer Vision. – URL : https://docs.opencv.org/3.4/da/d5c/tutorial_canny_detector.html (дата обращения : 25.04.2022).
10. Morphological Transformations [Электронный ресурс] // Open Source Computer Vision. – URL : https://docs.opencv.org/4.x/d9/d61/tutorial_py_morphological_ops.html (дата обращения : 25.04.2022).

Александр Иванович МАТВЕЕВ

ЦИФРОВАЯ ОБРАБОТКА ИЗОБРАЖЕНИЙ В OPENCV. ПРАКТИКУМ

Учебное пособие

Зав. редакцией
литературы по информационным технологиям
и системам связи *О. Е. Гайнутдинова*
Ответственный редактор *В. В. Яески*
Корректор *О. В. Федорова*
Выпускающий *В. А. Иутин*

ЛР № 065466 от 21.10.97
Гигиенический сертификат 78.01.10.953.П.1028
от 14.04.2016 г., выдан ЦГСЭН в СПб

Издательство «ЛАНЬ»
lan@lanbook.ru; www.lanbook.com
196105, Санкт-Петербург, пр. Юрия Гагарина, д. 1, лит. А

Тел./факс: (812) 336-25-09, 412-92-72
Бесплатный звонок по России: 8-800-700-40-71

Подписано в печать 08.09.22.
Бумага офсетная. Гарнитура Школьная. Формат 84×108 $^{1}/_{32}$.
Печать офсетная/цифровая. Усл. п. л. 5,46. Тираж 30 экз.

Заказ № 1274-22.

Отпечатано в полном соответствии
с качеством предоставленного оригинал-макета
в АО «Т8 Издательские Технологии».
109316, г. Москва, Волгоградский пр., д. 42, к. 5.