ИПМ им.М.В.Келдыша РАН

Онлайновая библиотека

**А.В. Агеев, А.А. Богуславский,
С.О. Власов, С.М. Соколов**

Компьютерное зрение
Лабораторный практикум

Рекомендуемая форма библиографической ссылки

Агеев А.В., Богуславский А.А., Власов С.О, Соколов С.М. Компьютерное зрение. Лабораторный практикум. М.: ИПМ им.М.В.Келдыша, 2024. 60 с.
doi: **10.20948/mono-2024-ageev**
URL: **https://keldysh.ru/e-biblio/ageev/**

А.В. Агеев, А.А. Богуславский, С.О. Власов,
С.М. Соколов

Компьютерное зрение

Лабораторный практикум

О р д е н а Л е н и н а
ИНСТИТУТ ПРИКЛАДНОЙ МАТЕМАТИКИ
имени М.В. Келдыша
Р о с с и й с к о й а к а д е м и и н а у к

А.В. Агеев, А.А. Богуславский, С.О. Власов, С.М. Соколов

Компьютерное зрение
Лабораторный практикум

Москва – 2024

Агеев А.В., Богуславский А.А., Власов С.О, Соколов С.М.

Компьютерное зрение; лабораторный практикум

Методическое пособие по введению в предмет «Компьютерное зрение». Рассматриваются базовые понятия и алгоритмы с теоретической точки зрения, а также представлены практические задания по разработке программ по обработке изображений для системы технического зрения.

Ключевые слова: *компьютерное зрение, техническое зрение, изображение, обработка изображения, сегментация изображения, система технического зрения*

Aleksey Vladimirovich Ageev, Andrey Alexandrovich Boguslavsky, Sergei Olegovich Vlasov, Sergey Michailovich Sokolov

Computer vision; laboratory practice

A methodological guide for the introduction to the subject "Computer vision". The basic concepts and algorithms are considered from a theoretical point of view, as well as practical tasks for the development of image processing programs for technical vision system are presented.

Key Words: *computer vision, technical vision, image, image processing, image segmentation, technical vision system*

Оглавление

Лабораторная работа 1
«Представления изображения в памяти компьютера и алгоритм выравнивания гистограммы»

Цель: Знакомство с базовыми понятиями компьютерного зрения, с представлением изображения в компьютере и анализ его частотных характеристик.

Теоретическая часть

Компьютерное зрение – это дисциплина, которая обрабатывает данные, получаемые с системы датчиков, с целью формирования выводов относительно объектов и сцен реального мира. Для построения системы технического зрения (СТЗ) необходимо решить следующие задачи:

1. Задача восприятия - необходимо использовать систему датчиков для получения данных об наблюдаемой окружающей среде.
2. Задача кодирования/представления данных – описание структуры данных, для представления потока значений, получаемых с датчиков, с целью дальнейшего манипулирования потоком данных.
3. Задача создания алгоритмов для извлечения информации из потоков данных.

На рисунке 1 представлена схема компьютерной системы, которая выполняет 3 вышеописанных этапа. С помощью камеры выполняется задача восприятия, затем контроллер камеры и компьютер выполняют задачу кодирования, передачи информации и ее декодирование, а затем компьютер, используя алгоритмическое обеспечение, извлекает информацию об сцене, которая была зафиксирована камерой.

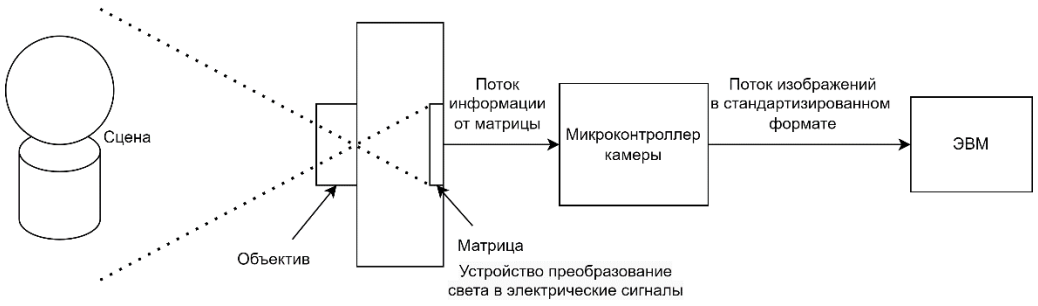

Рис. 1. Общая схема получения ЭВМ потока кадров, описывающих сцену, за которой наблюдает камера

Для создания алгоритмов обработки информации необходимо знать, в каком виде эта информация предоставляется. Для определения структуры данных изображений рассмотрим устройство камеры подробнее. Рассмотрим простейшую модель камеры: Камера-Обскура (Рис. 2). Данная камера представляет из себя коробку, у которой есть параллельные грани. У одной из параллельной грани проделано маленькое отверстие, через которое проходят

лучи света. Эти лучи падают на противоположную параллельную грань. На этой грани формируется перевернутое изображение сцены, за которой наблюдает камера. Камера обладает такой характеристикой, как фокусное расстояние – это расстояние между плоскостью изображения и отверстием, через которое проходит свет. Фокусное расстояние влияет на такие характеристики как: коэффициент увеличения наблюдаемой сцены и углы обзора. После того, как мы сформировали изображение на задней стенке камеры, необходимо его зафиксировать в некотором виде, чтобы было возможно им манипулировать.

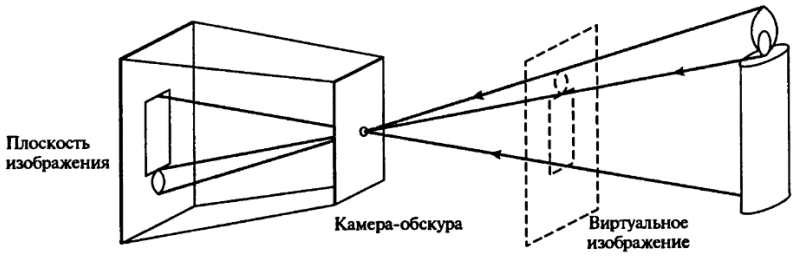

Рис. 2. Модель камеры-обскура [1 стр. 41]

Первые камеры для фиксации изображений использовали светочувствительное вещество, которое наносилось на пластину, а та вставлялась в камеру-обскуру, таким образом использовался химический способ записи изображения. Современные фотоаппараты с подобным механизмом записи информации используют объективы, которые позволяет регулировать фокусное расстояние камеры и углы обзора по мере необходимости и фотопленку как носитель информации, но задача системы технического зрения – это обработка кадров на ЭВМ, поэтому необходимы цифровые методы фиксации изображения. Для этого используется цифровая матрица, которая строится по технологии приборов с зарядовой связью (ПЗС). ПЗС-Матрица представляет из себя пластину, разбитую на ячейки. Каждая ячейка пластины – это электронный элемент, который выдает разное напряжение при попадании на него света разной интенсивности. Таким образом микроконтроллер камеры, получает информацию от каждой ячейки матрицы, формирует кадр изображения, который представляется в виде математического объекта матрица, каждый элемент которой в численном виде описывает интенсивность света попавшего на соответствующий элемент ПЗС-Матрицы. Такой элемент называется элементом изображения или пикселем (pixel – picture element).

Таким образом формируется изображения в оттенках градациях серого – когда каждый пиксель описывается одним числом, и это число означает интенсивность яркости света попавшего на элемент ПЗС-Матрицы. Кодирование такого монохромного изображения можно выполнять по-разному, например, можно сказать, что 0% — это черный, а 100% — это белый цвет, так как на элементе ПЗС-Матрицы в первом случае интенсивность цвета будет минимальна, а во втором максимальна, но закодировать можно и наоборот,

например с точки зрения принтера 0% — это белый цвет, так как не надо использовать краску, а

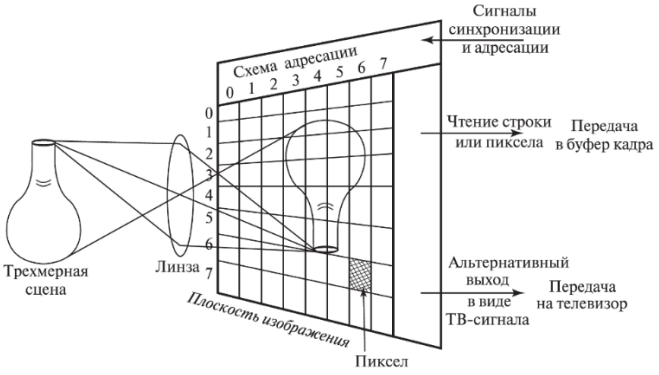

Рис. 3. Получение изображения вазы с помощью ПЗС-камеры. Дискретные ячейки преобразуют световую энергию в электрические заряды, которые при выводе в компьютер представлены числами [2 стр. 39]

100% — это черный цвет, так как необходимо использовать максимальный объем краски для закрашивания области листа (пикселя).

Кодирование цвета определяется удобством представления для решения задачи и это представление называется цветовой моделью. Естественной цветовой моделью является модель RGB. Это аддитивная модель, которая кодирует цвет тремя величинами: интенсивность красного цвета, зеленого и синего. Модель выполняет сложение тройки $\langle r, g, b \rangle$ интенсивности цвета с черным цветом $\langle 0, 0, 0 \rangle$ для получения результирующего цвета, который является линейной комбинацией трех компонент. Модель является естественной для человека, поскольку так устроены его глаза (рис. 4)

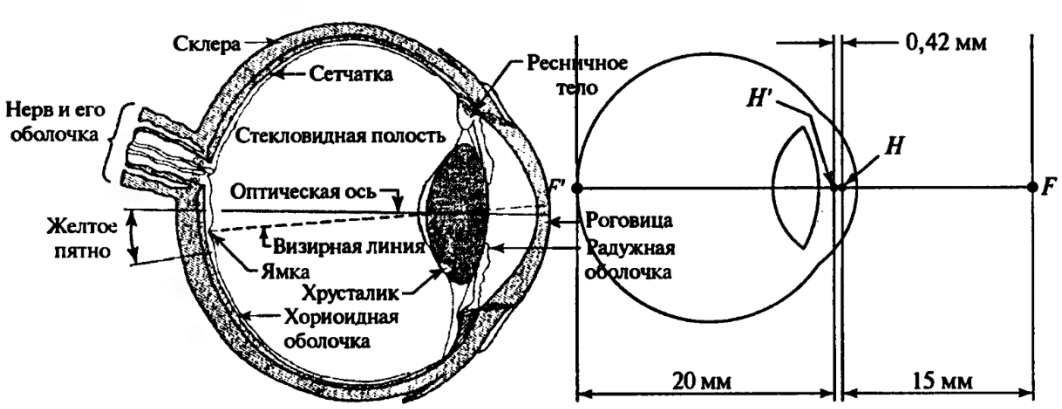

Рис. 4. Устройство человеческого глаза. Слева: основные части человеческого глаза. Справа: схема глаза [1 стр. 53]

Человеческий глаз – это оптическая система, которая обладает сенсором света в виде сетчатки, хрусталика (линза), форму, которую глаз меняет для изменения фокусного расстояния, и радужки глаза – это диафрагма, которая

регулирует радиус зрачка для изменения поступления интенсивности света на сетчатку. Сетчатка представляет из себя тонкую мембрану, состоящую из фоторецепторов – колбочек и палочек. Палочки сетчатки имеют высокую чувствительность, но воспринимают только интенсивность света. Колбочки бывают трех типов, которые воспринимают красный зеленый и синий цвет. На рисунке 5 представлена модель, которая описывает интенсивность реакции трех типов колбочек на длину волны света, попадающего в глаз. В соответствии с этой моделью можно ввести преобразование из цветового пространства RGB (которое воспринимает человек) в цветовое пространство GrayScale. Это преобразование задается следующей формулой [3]:

$$GraySacle[i,j] = 0.299 \cdot R[i,j] + 0.587 \cdot G[i,j] + 0.114 \cdot B[i,j] \quad (1)$$

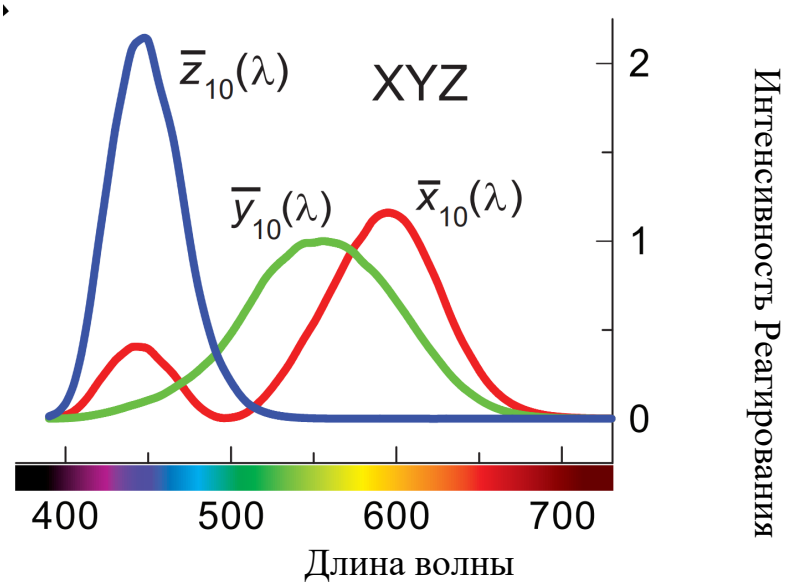

Рис. 5. Модель описывающая, как колбочки трех типов, отвечающие за три основных цвета, реагируют на разную длину волны видимого спектра света [3]

Таким образом изображение в цветовом пространстве RGB представляет из себя матрицу, каждый элемент которой является тройкой $\langle r, g, b \rangle$. Каждый элемент из тройки $\langle r, g, b \rangle$ описывает интенсивность красного, зеленого и синего цвета соответственно. По формуле (1) можно перейти в цветовое пространство GrayScale, в котором каждый элемент матрицы уже описывается одним числом – яркостью света.

В данном лабораторном практикуме для работы с изображениями мы будем использовать язык программирования python и библиотеку компьютерного зрения OpenCV (Open Computer Vision). Для того чтобы загрузить матрицу изображения из файла, необходимо воспользоваться методом cv.imread (листинг 1)

Листинг 1. Загрузка изображения, перевод его в оттенки градации серого и вывод исходного и преобразованного изображения

```
import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt

# cv.cvtColor(img, cv.COLOR_BGR2GRAY) – аналог из библиотеки
# OpenCV
# для перевода изображения из RGB в GrayScale
def rgbToGrayScale(img):
    output = np.zeros((img.shape[0], img.shape[1]),
                      dtype=np.uint8)
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            output[i, j] = 0.114*img[i, j, 0]
                           + 0.587*img[i, j, 1]
                           + 0.299*img[i, j, 2]
    return output


img = cv.imread('1.jpg')
gray = rgbToGrayScale(img)
fig, ax = plt.subplots(1, 2)
ax[0].imshow(cv.cvtColor(img, cv.COLOR_BGR2RGB))
ax[1].imshow(gray, cmap='gray')
plt.show()
```

Результат работы программы представлен на рисунке 6. Функция rgbToGrayScale выполняет переход из цветовой модели RGB в модель GrayScale в соответствии с формулой (1). Особенностью библиотеки OpenCv является то, что по умолчанию используется цветовая модель BGR, а не RGB, по этому для вывода изображения с помощью библиотеки рисования графиков, сначала необходимо выполнить конвертацию из BGR в RGB. Мы это делаем с помощью встроенной функции OpenCv `cvtColor`.

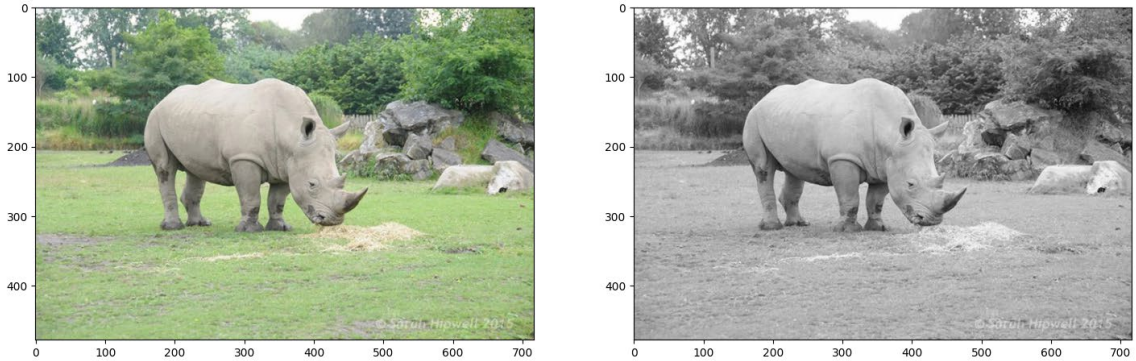

Рис. 6. Вывод программы из листинга 1

Изображения необходимо как-то описывать, для их сравнения и анализа. Одним из способов описания характеристик изображения является использование частотных характеристик. Для описания частотных характеристик используется гистограмма изображения.

Гистограмма h полутонового изображения I – это столбчатая диаграмма распределения яркостей изображения. Каждый столбик относится к оттенку серого. Высота столбика показывает количество пикселей с данным оттенком серого. В листинге 2 представлена программа для построения гистограммы изображения в оттенках серого.

Библиотеки numpy и OpenCV, pyplot предоставляют методы для построения гистограммы:

- `numpy.histogram` – вычисление гистограммы по массиву данных;
- `cv.calcHist` – вычисление гистограммы по изображению;
- `plt.hist` – построить график гистограммы по данным.

Листинг 2. построение гистограммы изображения

```
import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt

def histogram(img):
    h = np.zeros(256, dtype=np.int)
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            h[img[i, j]] = h[img[i, j]] + 1
    return h

img = cv.imread('1.jpg')
gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
h = histogram(gray)
bins = [i for i in range(0, 256)]
plt.bar(bins, h)
plt.show()
```

Результат построения гистограммы изображения программой 2 представлен на рисунке 7.

Используя гистограмму изображения возможно делать некоторые выводы об изображении. Например, по гистограмме на рисунке 7 можно сказать, что изображение имеет преимущественно светлые оттенки, так как большая часть пикселей расположена в правой части гистограммы (0 – черный цвет; 255 – белый цвет). Но при этом изображение использует практически весь цветовой диапазон. Но бывают результаты съемки, при которых изображение слишком светлое или темное и при этом использует узкий диапазон оттенков. Это можно

понять на основе гистограммы и принять решение о том, что нам необходимо скорректировать параметры камеры для съемки более качественного изображения. Но что значит «качественное» изображение/съемка?

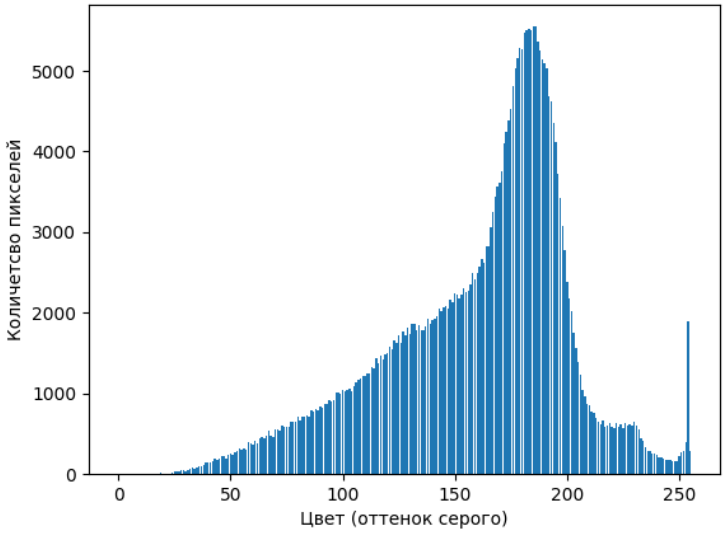

Рис. 7. Гистограмма изображения в оттенках серого, представленная на рисунке 6

В задачах систем технического зрения одним из критериев качества можно рассматривать ширину и равномерность использования диапазона оттенков изображением. Бывает не всегда возможно подобрать параметры оборудования, для получения идеальной гистограммы (в данном случае под идеальностью понимается получения изображения, гистограмма которого совпадает с гистограммой равномерного распределения). Для достижения необходимого результата существует метод, который пытается выполнить растяжение исходной гистограммы на весь диапазон оттенков – это **алгоритм выравнивания (эквализации) гистограммы**. Суть которого заключается в том, что мы рассматриваем распределение оттенков на изображении, как некоторую случайную величину, для которой возможно построить эмпирическую (кумулятивную) функцию распределения оттенков. Метод выполняет нормализацию этой функции распределения:

$$cdf'(v) = \frac{cdf(v) - cdf_{min}}{cdf_{max} - cdf_{min}} \cdot L \qquad (2)$$

$cdf(v)$ – эмпирическая функция распределения оттенков изображения; cdf_{min} – минимальное значение функции распределения $cdf(v)$ отличное от нуля. cdf_{max} – максимальное значение функции распределения $cdf(v)$. L – максимальное значение яркости пикселя на изображении, которое мы хотим получить после нормализации, поскольку мы хотим растянуть гистограмму на весь диапазон то для восьмибитных изображений $L = 2^8 = 256$.

Для того чтобы получить эмпирическую функцию распределения, нам необходимо выполнить интегрирование функции плотности вероятности распределения оттенков пикселей:

1. Вычисляем гистограмму h изображения (листинг 2).
2. Вычисляем интеграл по гистограмме:

$$cdf(v) = \sum_{i=0}^{v} h(i) \quad (3)$$

Функция 2 описывает правила преобразования оттенка v исходного изображения в оттенок изображения с нормализированной функцией распределения (эквализированной гистограммой). В листинге 3 представлена реализация метода эквализации гистограммы, а на рисунке 8 результаты работы программы.

OpenCV предоставляет функцию для эквализации гистограммы: `cv.equalizeHist.`

Листинг 3. Реализация алгоритма эквализации гистограммы

```
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt

def makeCdf(histogram):
    integral = np.zeros_like(histogram)
    integral[0] = histogram[0]
    for i in range(1, integral.shape[0]):
        integral[i] = integral[i-1] + histogram[i]
    return integral

def histEqualization(image):
    h = histogram(image)
    numPixels = image.shape[0]*image.shape[1]

    cdf = makeCdf(h)

    # normalization
    firstNoZeroElement = 0
    for el in h:
        if el != 0:
            firstNoZeroElement = el
            break

    sourceRange = numPixels
    destinationRange = 256
```

```
    lut = np.zeros_like(h)
    for i in range(destinationRange):
        offset = cdf[i] - firstNoZeroElement
        if offset < 0:
            offset = 0
        lut[i] = (offset/sourceRange)*(destinationRange)

    return cv.LUT(image, lut).astype(np.uint8)


img = cv.imread('1.jpg')
gray = rgbToGrayScale(img)
equalized = histEqualization(gray)

h = histogram(gray)
cdf = makeCdf(h)
h2 = histogram(equalized)
cdf2 = makeCdf(h2)

bins = [i for i in range(0, 256)]
fig, ax = plt.subplots(2, 2)
ax[0, 0].imshow(gray, cmap="gray")
ax[0, 0].set_title("Исходное изображение")

ax[1, 0].imshow(equalized, cmap="gray")
ax[1, 0].set_title("Эквализированное изображение")

ax[0, 1].bar(bins, h)
ax[0,   1].set_title("Плотность   распределения   и   функция
вероятности
                    оттенков\nисходного изображения")
ax[0, 1].set_xlabel("Оттенки (градации серого)")
ax[0,   1].set_ylabel("Количество   пикселей\nс   конкрентным
оттенком")

ax[1, 1].bar(bins, h2)
ax[1,   1].set_title("Плотность   распределения   и   функция
вероятности
                    оттенков\nэквализированного изображения")
ax[1, 1].set_xlabel("Оттенки (градации серого)")
ax[1,   1].set_ylabel("Количество   пикселей\nс   конкрентным
оттенком")
```

```
ax01_twin = ax[0, 1].twinx()
ax11_twin = ax[1, 1].twinx()

ax01_twin.plot(bins, cdf, "r")
ax01_twin.set_ylabel("Количество пикселей,\nкоторые имееют
                      заданное\nили меншее значение оттенка")
ax11_twin.plot(bins, cdf2, "r")
ax11_twin.set_ylabel("Количество пикселей,\nкоторые имееют
                      заданное\nили меншее значение оттенка")
plt.show()
```

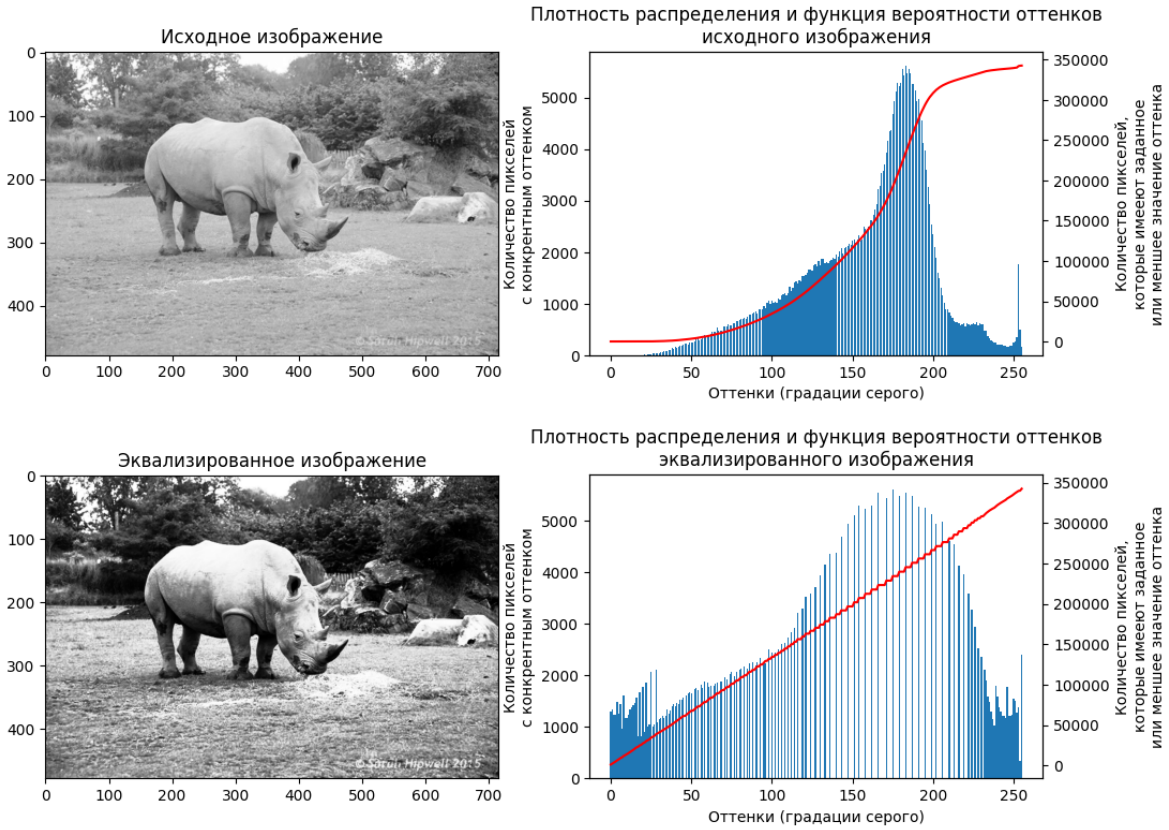

Рис. 8. Эквализация гистограммы изображения. Результат работы программы из листинга 3

Задание

1. В интернете найти 3 изображения, которые соответствуют следующим требованиям:
   a. Первое изображение должно быть недоэкспонированным;
   b. Второе изображение должно быть нормально экспонированным;
   c. Треть изображение должно быть переэкспонированным.

2. Над каждым из трех изображений необходимо выполнить операции:
    a. Загружаем изображение с помощью opencv: cv.imread;
    b. Переводим изображение из цветного пространства RGB в градацию оттенков серого: cv.cvtColor(img, cv.COLOR_BGR2GRAY);
    c. С помощью функции plt.hist() – библиотеки matplotlib построить гистограмму черно-белого изображения;
    d. К черно-белому изображению применить операцию эквализации гистограммы (OpencCv: cv.equalizeHist);
    e. Вывести эквализированное изображение и его гистограмму;
    f. Для исходного и эквализированного изображения построить эмпирическую функцию распределения.
3. Подготовить отчет, который должен содержать:
    a. Ответы на вопросы из пункта «Вопросы для самопроверки»;
    b. Описание операции эквализации гистограммы;
    c. Описание результатов ваших экспериментов;
    d. Исходный код программы на языке программирования python.
4. У каждого студента должны быть свой набор изображений.

Вопросы для самопроверки

1. Как изображение представлено в компьютере? Что такое пиксель?
2. Чем отличается представление изображения в цветовом пространстве RGB от цветового пространства Градация оттенков серого?
3. Как выполнить предобразование RGB изображение в изображение в градациях оттенка серого?
4. Что описывает гистограмма изображения, как ее построить?
5. Как работает операция эквализация гистограммы изображения?

Лабораторная работа 2 «Фильтрация и Бинаризация изображений»

Цель: Изучение подхода бинаризации и оператора свертки для извлечения информации из изображений.

Теоретическая часть

В данной лабораторной работе будет рассмотрена задача обнаружения объектов и предложен один из возможных способов ее решения. Будут использованы такие подходы как, бинаризация изображения, фильтрация изображения с помощью оператора свёртки, и поиск связанных компонент на бинарном изображении.

Бинаризация – это подход к преобразованию изображения в градациях оттенков серого в изображение, состоящее из двух цветов: черного и белого. Это преобразование заключается в классификации пикселей на два класса: Пиксель

фона и пиксель переднего плана. Фоновые пиксели принимают черный цвет, а пиксели переднего плана белый. Выполнять эту классификацию можно по-разному, в зависимости от задачи. Один из возможных способов реализовать бинаризацию – это выполнить пороговую классификацию пикселей. Если яркость пикселя меньше порога, то это фоновый пиксель, если – больше, то это пиксель переднего плана. Таким образом, выполняется попиксельное сканирование изображения, в котором каждый пиксель сравнивается с порогом и в зависимости от результата сравнения, пикселю присваивается тот или иной класс (Листинг 1).

OpenCv предоставляет функцию для ручной бинаризации: `cv.threshold`

Листинг 1. Реализация алгоритма бинаризации с ручным подбором порога

```
def manualThreshold(image, t):
    result = np.copy(image)
    for i in range(image.shape[0]):
        for j in range(image.shape[1]):
            if image[i, j] <= t:
                result[i, j] = 0
            else:
                result[i, j] = 255
    return result
```

Представленный алгоритм бинаризации простой и позволяет решать прикладные задачи, но встает вопрос, какой порог необходимо использовать в каком случае. Можно подбирать его вручную для достижения необходимого результата, а можно попытаться автоматизировать процесс. Нобуюки Отцу предложил следующий подход. Гистограмму изображения будем рассматривать как плотность вероятности распределения пикселей изображения. Так как мы разбиваем пиксели изображения на два класса то предположим, что плотность вероятности, которую мы имеем, на самом деле – это смесь плотностей двух случайных величин. Одна плотность вероятности описывает распределение фоновых пикселей, а вторая плотность, описывает распределение пикселей переднего плана. Какие пиксели относятся к фону, а какие к переднему плану – темные или светлые – зависит от решаемой задачи. Необходимо подобрать такой порог бинаризации, который бы наилучшим способом отделял эти две плотности вероятности. Метод Отцу использует критерий однородности двух групп. Однородность группы определяется через дисперсию внутри группы: Необходимо найти такое значение порога t , которое будет уменьшать суммарное (общее) значений дисперсий двух групп [2 стр. 120]:

$$\sigma_W^2(t) = q_1(t)\sigma_1^2(t) + q_2(t)\sigma_2^2(t) \to min;$$
$$q_1(t) = \sum_{i=1}^{t} p(i);\ \ q_2(t) = \sum_{i=t+1}^{I} p(i); \qquad (1)$$

$$\mu_1(t) = \frac{1}{q_1(t)} \sum_{i=1}^{t} i\, p(i)\,;\ \ \mu_2(t) = \frac{1}{q_2(t)} \sum_{i=t+1}^{I} i\, p(i)\,;$$

$$\sigma_1^2(t) = \frac{1}{q_1(t)} \sum_{i=1}^{t} \big(i - \mu_1(t)\big)^2\,;\ \ \sigma_2^2(t) = \frac{1}{q_2(t)} \sum_{i=t+1}^{I} \big(i - \mu_2(t)\big)^2\,;$$

Где:

- t – порог бинаризации;
- I – индекс белого цвета ($I = 255$);
- $p(i)$ – вероятность встретить цвет i на изображении. Это высота столбика гистограммы, поделенное на общее количество пикселей в изображении;
- $q_1(t)$ – суммарная вероятность встретить пиксель на изображении из первой группы (левой группы);
- $q_2(t)$ – суммарная вероятность встретить пиксель на изображении из второй группы (правой группы)*;*
- $\mu_1(t)$ – математическое ожидание оттенка первой группы (средний оттенок);
- $\mu_2(t)$ – математическое ожидание оттенка второй группы (средний оттенок);
- $\sigma_1^2(t)$ – дисперсия между элементами первой группы;
- $\sigma_2^2(t)$ – дисперсия между элементами второй группы;
- $\sigma_W^2(t)$ – суммарное значение внутренних дисперсий двух групп, которое необходимо минимизировать по t.

Для решения задачи минимизации (1) мы будем использовать полный перебор всех целочисленных значений t от 0 до 255. Реализация алгоритма бинаризации методом Отцу представлена в листинге 2, а на рисунке 2 представлен пример работы реализованного алгоритма бинаризации.

Библиотека OpenCV предоставляет функцию бинаризации изображения по методу Отцу: `cv.threshold` с использованием флага `cv.THRESH_OTSU` в качестве параметра.

Листинг 2. Метод бинаризации Отцу

```
def groupWeight(h, s, e):
    res = 0.0
    for i in range(s, e):
        res = res + h[i]
    return res

def groupMean(h, q, s, e):
    res = 0.0
    for i in range(s, e):
        res = res + i * h[i]
    return res / q
```

```
def groupDispersion(h, mean, q, s, e):
    res = 0.0
    for i in range(s, e):
        res = res + ((i - mean) ** 2) * h[i]
    return res / q

def otsu(image):
    h = histogram(image)
    numPixels = np.sum(h)
    for i in range(0, h.shape[0]):
        h[i] = h[i] / numPixels

    w_res = float('inf')
    t_res = 0

    for t in range(0, 256):
        q1 = groupWeight(h, startNonZero, t)
        q2 = groupWeight(h, t + 1, endNonZero)
        if q1 == 0 or q2 == 0:
            continue

        mean1 = groupMean(h, q1, startNonZero, t)
        mean2 = groupMean(h, q2, t+1, endNonZero)

        disp1 = groupDispersion(h, mean1, q1, startNonZero, t)
        disp2 = groupDispersion(h, mean2, q2, t+1, endNonZero)
        w_tmp = disp1 + disp2
        if w_tmp < w_res:
            w_res = w_tmp
            t_res = t
    print(t_res)
    return manualThreshold(image, t_res), t_res
```

Рассматривая бинарное изображение монет (рис. 1), можно увидеть, что не все монеты полностью черные. Это связано с тем, что освещение изображения не равномерное. Свет, примерно, расположен в левом верхнем углу, а также некоторые монеты из-за угла съемки отражает свет. Таким образом изображение имеет неравномерное распределение цветов в разных областях изображения. Метод ручной бинаризации и метод Отцу являются **методами глобальной бинаризации**, который выполняют бинаризацию изображения использую один порог на все изображение. Из-за неравномерности яркости на изображении, эти методы иногда не подходят. Для бинаризации изображений с неравномерной

яркостью существуют методы **адаптивной бинаризации**. Эти методы используют скользящее окно, которым сканируют все изображение.

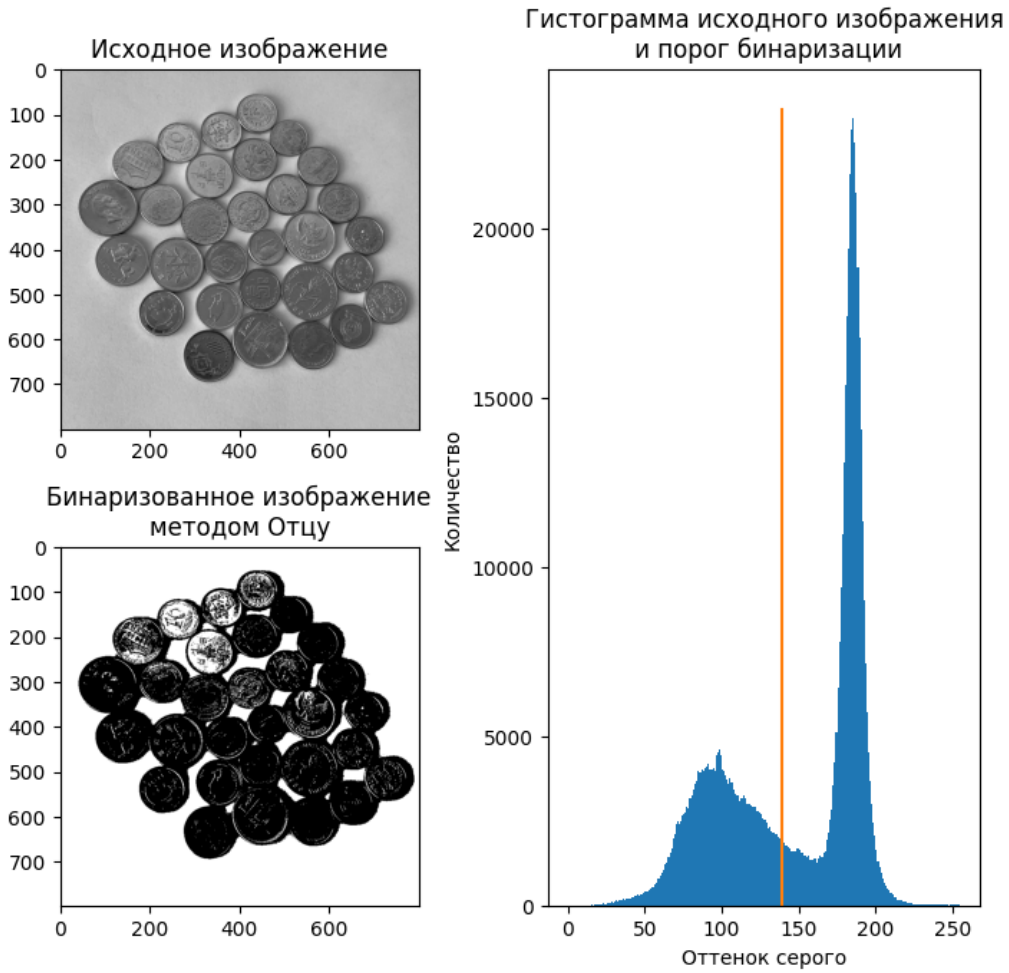

Рис. 1. Результат бинаризации изображения методом Отцу. Автоматически был выбран порог бинаризации 139. Для дальнейшей обработки изображения его необходимо инвертировать (`cv.bitwise_not`), чтобы монеты были белыми, а фон черным

Для каждой области изображения, которая попадает в окно, подбирается свой порог бинаризации, таким образом учитывается неравномерность изображения.

При обработке изображений, на изображениях часто встречаются дефекты, которые мешают получения корректных результатов. В качестве дефектов могут выступать шум на изображении или текстура, которая мешает обработке. Например, на рисунке 2 представлена фотография монеты, применяя бинаризацию, к которой мы получаем неудовлетворительный результат, так как задача стоял в выделении монеты, а на бинаризованном изображении кроме монеты так же были выделены блики подложки, что мешает корректной идентификации. Тогда необходимо выполнить фильтрацию – это предобработка, которая заключается в уменьшении детализации изображения с целью ликвидирования лишней информации, мешающей обработке.

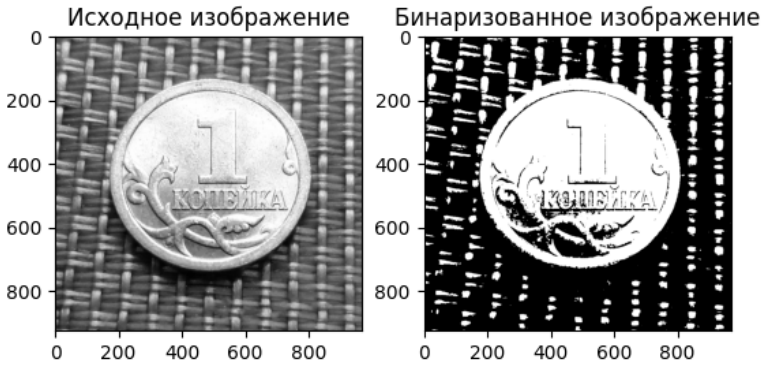

Рис. 2. Бинаризация изображения с текстурой методом Отцу.
(Автоматически выбранный порог бинаризации: 159)

Операция сверки является универсальным механизмом для фильтрации изображений, которая заключается в формировании нового изображения на основе исходного изображения и ядра. Ядро – это матрица коэффициентов. Для формирования нового изображения выполняется сканирование пиксель за пикселем исходного изображения, при котором выполняется формирование значений пикселей нового изображения путем использования линейной комбинации текущего пикселя с соседями в некоторой окрестности. Ядро свёртки задает размер этой окрестности и коэффициенты линейной комбинации, более формально:

Свертка двух функций $f(x,y)$ и $\mathrm{h}(x,y)$, определяется выражением:

$$g(x,y) = f(x,y) * h(x,y) = \int_{x'=-\infty}^{+\infty} \int_{y'=-\infty}^{+\infty} f(x',y')h(x-x',y-y')dx'dy' \quad (2)$$

Где $f(x,y)$ – исходная функция; $h(x,y)$ – ядро свертки, весовая функция, которая определяет линейную комбинацию в окрестности точки (x,y).

Поскольку мы работает с изображениями, а это конечные дискретные объекты, то интегралы переходя в следующие суммы:

$$g(x,y) = f(x,y) * h(x,y) = \sum_{i=-d}^{i=d+1} \sum_{j=-d}^{j=d+1} f(x-i,y-j)h(i+d,j+d) \quad (3)$$

Где $f(x,y)$ – матрица изображения; $h(x,y)$ – матрица ядра свертки, которая имеет размерность $[2\mathrm{d}+1; 2\mathrm{d}+1]$. Формула (3) описывает вычисление значения одного пикселя, на основе линейной комбинации его окрестности в исходном изображении $f(x,y)$. Реализация алгоритма свертки представлена в листинге 3, а результат работы программы представлен на рисунке 3.

Листинг 3. Алгоритм свертки матрицы изображения с матрицей ядра

```
def convolution2(image, kernel):
    d = int((kernel.shape[0] - 1)/2)
    output = np.zeros_like(image, dtype=np.float)
```

```
    for i in range(0, image.shape[0]):
        for j in range(0, image.shape[1]):
            kernel_i = -1
            kernel_j = -1
            for ni in range(i-d, i+d+1):
                kernel_i = kernel_i + 1
                if ni < 0 or ni >= image.shape[0]:
                    continue
                for nj in range(j-d, j+d+1):
                    kernel_j = kernel_j + 1
                    if nj < 0 or nj >= image.shape[1]:
                        continue
                    value = image[ni, nj]
                            * kernel[kernel_i, kernel_j]
                    output[i, j] = output[i, j] + value
                kernel_j = -1
            if output[i, j] < 0:
                output[i, j] = 0
            elif output[i, j] > 255:
                output[i, j] = 255

    return np.round(output)

size = 21
kernel_blur = np.ones((size, size),
                      dtype=np.float64)
                     * 1 / (size**2)

img = cv.imread("money2.png")
img = cv.resize(img, (320, 320))
gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
blure = np.round(convolution2(gray,
                              kernel_blur)).astype(np.uint8)

b, t = otsu(blure)
ax1 = plt.subplot(221)
ax1.imshow(gray, cmap="gray")
ax1.set_title("Исходное изображение")
ax2 = plt.subplot(223)
ax2.imshow(b, cmap='gray')
ax2.set_title("Бинаризованное изображение\nметодом Отцу")
ax3 = plt.subplot(122)
```

```
print(t)
ax3.imshow(blure, cmap='gray')
ax3.set_title("Размытое изображение\nc помощью операции свертки")
plt.show()
```

В листинге 3 используется матрица, все коэффициенты которой равняются дроби:

$$\frac{1}{d \cdot d} \tag{4}$$

Где d – это размерность квадратной матрицы ядра. Таким образом при выполнении свертки, значение пикселя нового изображения – это среднее значение пикселей в окрестности 21 пиксель – и это операция размытия, которая относится к **фильтру низких частот** – борьба с шумом или мелкими постоянно повторяющимися деталями – борьба с высокочастотными компонентами. Размер ядра должен быть больше, чем размер деталей, которые мы хотим убрать. В данном случае, нашей целью было избавиться от текстуры на изображении. Так же с помощью операции свертки можно реализовать и **фильтр высоких частот** – борьба с низкочастотными компонентами. Например, такого типа фильтрацией можно отнести детектор границ (Оператор Собеля, Алгоритм Cany).

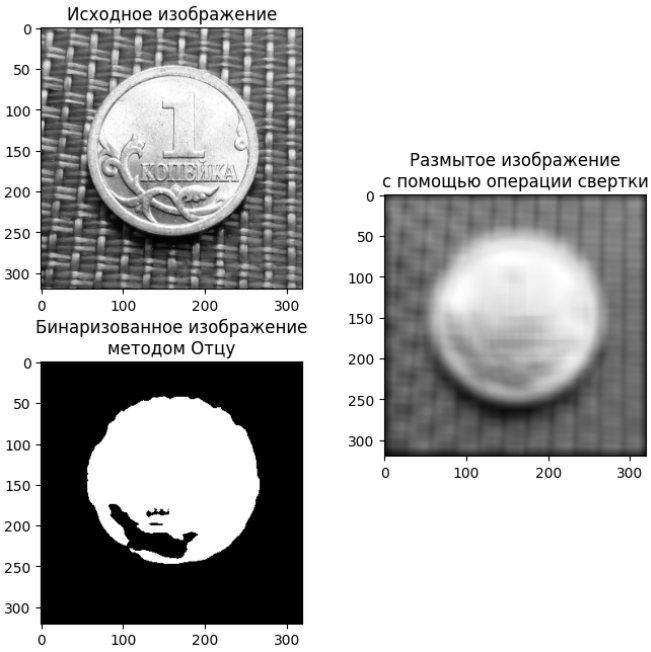

Рис. 3. Бинаризация методом Отцу предварительно размытого изображения оператором свёртки с ядром размером 21x21 (Автоматически выбранный порог бинаризации: 163)

Что такое граница на изображении между объектами? Это резкое изменение цвета соседних пикселей. Например, если мы рассматриваем изображение в оттенках градации серого, то это резкое изменения яркости между соседними пикселями. Для формирования детектора границ воспользуемся этим фактором.

В математическом анализе есть градиент – это вектор, указывающий в направления наискорейшего роста функции (5).

$$grad\, f = \nabla f = \left(\frac{\partial f}{\partial x_1}, \dots, \frac{\partial f}{\partial x_i}, \dots, \frac{\partial f}{\partial x_n}\right)$$

$$\frac{\partial f}{\partial x_i}(x_1, \dots x_i, \dots x_n) = \lim_{h \to 0} \frac{f(x_1, \dots, x_i + h, \dots x_n) - f(x_1, \dots, x_i + h, \dots x_n)}{h} \quad (5)$$

В контексте обработки изображений, мы можем вычислить градиент в каждой точке изображения, как разность яркостей соседних пикселей по двум направлениям. По направлению оси x и направлению оси y, а затем вычислить длину вектора градиента в каждой точке. Поскольку граница – это резкий перепад яркости между соседними пикселями, то и длина вектора градиента в точках резких перепадов будет большой, а в других местах длина будет маленькой. Таким образом мы сможем пометить границы на изображении. Sobel I. и Feldman G. предложили следующий градиентный оператор (6) на основе свертки:

$$G_y = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix} * A; \quad G_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} * A$$
$$G = \sqrt{G_x^2 + G_y^2} \quad (6)$$

Где A – матрица изображения; G_y – частная производная изображения по оси y в каждой точке; G_x – частная производная изображения по оси x в каждой точке; G – длина вектора градиента в каждой точке изображения.

Листинг 4. Оператор Собеля

```
import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt
def sobelsOperator(image):
    dx_kernel = np.array([[-1, 0, 1],
                          [-2, 0, 2],
                          [-1, 0, 1]])
    dy_kernel = np.array([[1, 2, 1],
                          [0, 0, 0],
                          [-1, -2, -1]])

    dx = convolution2(img, dx_kernel)
    dy = convolution2(img, dy_kernel)
    return np.sqrt(np.square(dx) + np.square(dy)), dx, dy
```

```
img = cv.imread("money2.png")
img = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
g, dx, dy = sobelsOperator(img)
fig, ax = plt.subplots(2, 2)
ax[0, 0].imshow(img, cmap="gray")
ax[0, 0].set_title("Исходное изображение")
ax[0, 1].imshow(np.round(g).astype(np.uint8), cmap="gray")
ax[0, 1].set_title("Результата применения\nоператора Собеля")
ax[1, 0].imshow(np.round(dx).astype(np.uint8), cmap="gray")
ax[1, 0].set_title("Производная по оси X")
ax[1, 1].imshow(np.round(dy).astype(np.uint8), cmap="gray")
ax[1, 1].set_title("Производная по оси Y")
plt.show()
```

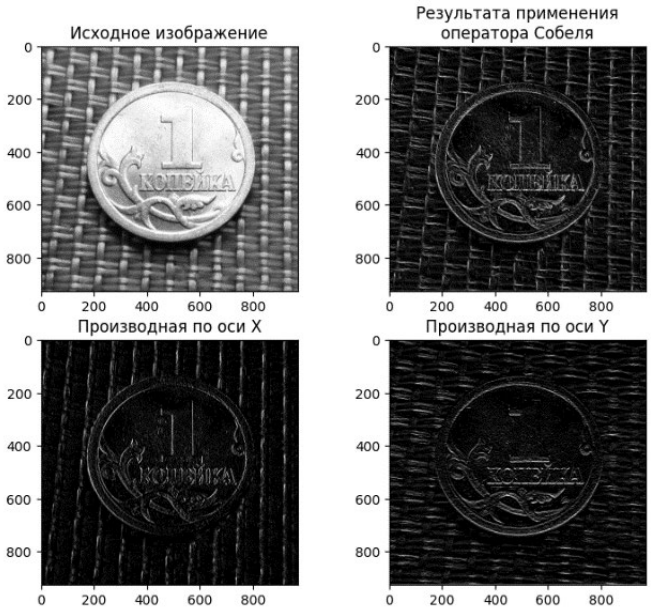

Рис. 4. Применение оператора Собеля к изображению в оттенках градации серого

Практическая часть

Необходимо реализовать программное обеспечение, которое будет определять отверстия на фотографиях деталей. Для реализации необходимо использовать алгоритмы бинаризации и метод низкочастотного фильтра (размытие изображений).

1. Загружаем изображение (cv.imread).

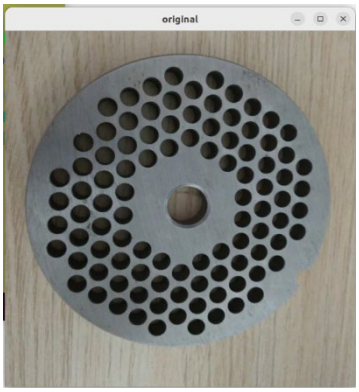

Рис. 5. Результат загрузки изображения и вывода изображения

2. Переводим изображение из цветового пространства BGR в оттенки градации серого (cv.cvtColor).

Рис. 6. Результат перевода изображения в оттенки градации серого

3. Перед бинаризацией, возможно, потребуется размыть изображение, медианным фильтром или фильтром гаусса, экспериментируйте (cv.medianBlur; cv.GaussianBlur).

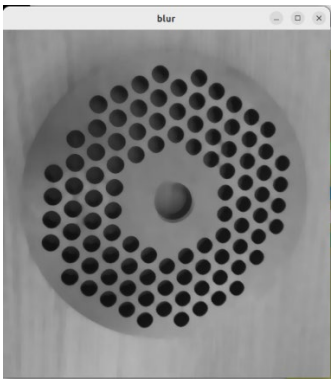

Рис. 7. Результат размытия изображения

4. Для бинаризации необходимо воспользоваться одним из методов, и необходимо достичь того, чтобы отверстия четко выделялись и были белого цвета.

1. Глобальная ручная бинаризация — для изображений необходимо вручную подобрать пороги бинаризации (cv.threshold, cv.THRESH_BINARY).
2. Глобальная бинаризация с автоматическим выбором порога по методу Отцу (cv.threshold, cv.THRESH_BINARY+cv.THRESH_OTSU).
3. Адаптивная бинаризация (cv.adaptiveThreshold):
    1. cv.ADAPTIVE_THRESH_MEAN_C.
    2. cv.ADAPTIVE_THRESH_GAUSSIAN_C.

Для одного изображения работает один метод, для другого другой, экспериментируйте для получения результата. Мы воспользуемся бинаризацией по методу Отцу:

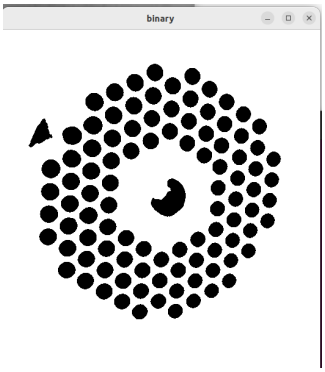

Рис. 8. Результат бинаризации изображения по методу Отцу

Нашей конечной целью является определение отверстий. Метод Отцу определил отверстия как фон (черные цвет), а все остальное как объект. Поэтому нам необходимо инвертировать бинарное изображение. Выполним это с помощью функции cv.bitwise_not. Получим следующий результат:

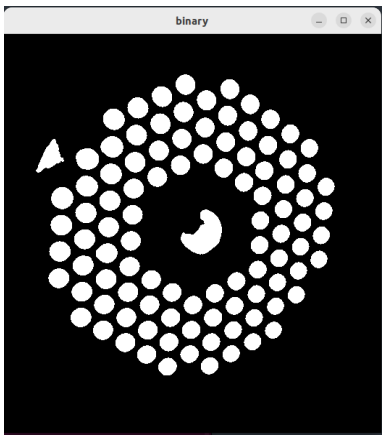

Рис. 9. Инвертированное бинарное изображение

5. После бинаризации, с помощью алгоритма поиска связанных компонент (cv.connectedComponents) необходимо промаркировать каждый объект, который был выделен алгоритмом бинаризации. Функция

cv.connectedComponents — вернет матрицу такого же размера, как и матрица бинарного изображения, но элементы этой матрицы будут содержать метки связанных компонент. Для того, чтобы вывести результат маркировки на экран (с целью убедиться в корректности работы), необходимо каждой метки присвоить случайный RGB цвет и с помощью созданной палитры нарисовать изображение и вывести его:

Листинг 5. Запуск алгоритма поиска связанных компонент и вывод результатов его работы

```
num_labels, labels = cv.connectedComponents(otsu)
label_color = [rgb_random() for i in range(num_labels)]

for i in range(img.shape[0]):
    for j in range(img.shape[1]):
        objects[i, j] = label_color[labels[i, j]]

cv.imshow("labels", objects)
```

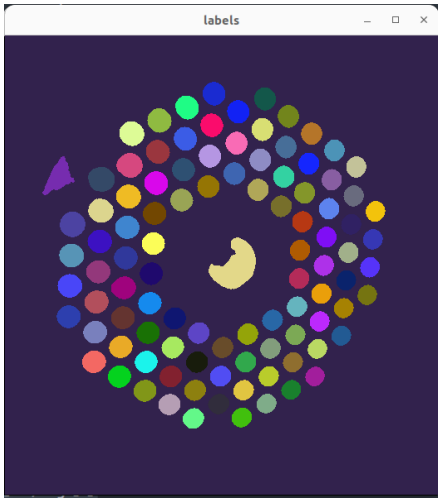

Рис. 10. Результат маркировки бинарного изображения (Листинг 5)

6. Для каждого объекта необходимо построить Bounding Box. OpenCv предоставляет функцию рисования прямоугольников cv.rectangle. Эта функция на вход принимает координаты, где рисовать. Вам необходимо написать функцию, которая выполнит сканирование матрицы меток, и для каждой метки вычислит координаты прямоугольника, который затем необходимо нарисовать на исходном изображении с помощью функции cv.rectangle.
7. Алгоритм поиска координат размеченных объектов:
    1. Заводим массив координат $bounding\_boxes$ для каждого размеченного объекта. Количество объектов нам известно – это количество меток, которые вернул нам алгоритм

cv.connectedComponents. Каждый элемент массива $bounding\_boxes$ – это координаты прямоугольника, в который вписан некоторый объект. Координаты прямоугольника описываются двумя точками. Верхней левой точкой и правой нижней точкой.

1. $bounding\_boxes[k]$ – координаты прямоугольника, в который вписан объект с меткой k.
2. $bounding\_boxes[k].left\_point$ – координаты верхней левой точки прямоугольника.
3. $bounding\_boxes[k].right\_point$ – координаты нижней правой точки прямоугольника.

2. Цикл по матрице меток, строка за строкой:
    1. Пусть l – это текущая метка. i, j – текущие координаты в матрице меток.
    2. Сравниваем координаты текущей позиции для метки l с координатами для этой метки в массиве bounding_boxes.
        - Если $bounding_{boxes[k]}.left_{point}.y < i$;
          То $bounding_{boxes[k]}.left_{point}.y = i$;
        - Если $bounding_{boxes[k]}.left_{point}.x < j$;
          То $bounding_{boxes[k]}.left_{point}.x = j$;
        - Если $bounding_{boxes[k]}.right_{point}.y < i$;
          То $bounding_{boxes[k]}.right_{point}.y = i$;
        - Если $bounding_{boxes[k]}.right_{point}.x < j$;
          То $bounding\_boxes[k].right\_point.x = j$.

Листинг 6. Реализация алгоритма поиска обрамляющих прямоугольников для размеченных объектов

```
import dataclasses
import math

@dataclasses.dataclass
class Point:
    x: int = 0,
    y: int = 0,

class Rectangle:
    def __init__(self, left: Point, right: Point):
        self._up_left_point = left
        self._down_right_point = right

    @property
    def width(self):
```

```
        return self._down_right_point.x - self._up_left_point.x

    @property
    def height(self):
        return self._down_right_point.y - self._up_left_point.y

    @property
    def square(self):
        return self.width*self.height

    @property
    def left_point(self) -> Point:
        return self._up_left_point

    @property
    def right_point(self) -> Point:
        return self._down_right_point

    def __repr__(self):
        return "{[%d, %d]; [%d, %d]}" % (self._up_left_point.x,

self._up_left_point.y,

self._down_right_point.x,

self._down_right_point.y)


def find_bounding_box(labels, number_labels):
    bounding_boxes =
[Rectangle(Point(math.inf, math.inf), Point(-1, -1)) for i in
range(number_labels)]
    for i in range(labels.shape[0]):
        for j in range(labels.shape[1]):
            l = labels[i, j]
            left_point = bounding_boxes[l].left_point
            right_point = bounding_boxes[l].right_point
            if left_point.y > i:
                left_point.y = i
            if left_point.x > j:
                left_point.x = j
```

```
            if right_point.y < i:
                right_point.y = i
            if right_point.x < j:
                right_point.x = j
    return bounding_boxes
```

8. Результат построения Bounding Box для всех объектов, представленных на изображении:

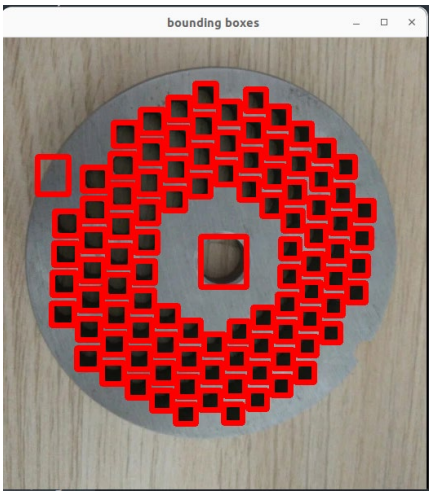

Рис. 11. Результат построения bounding boxes вокруг промаркированных элементов изображения

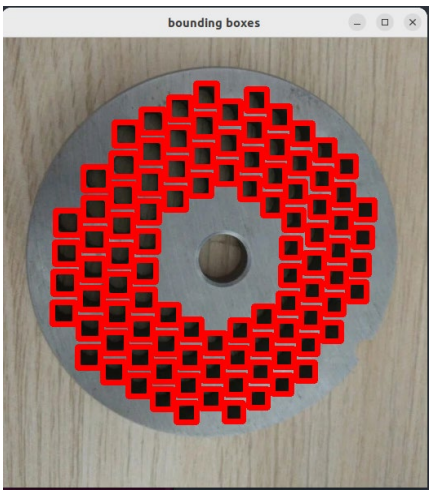

Рис. 12. Результат построения bounding boxes вокруг промаркированных элементов изображения с фильтрацией по площади прямоугольника

9. Перед построением необходимо фильтровать прямоугольники по площади, чтобы не рисовать их для паразитных объектов. В данном случае при фильтрации по площади прямоугольника, мы потерям центральное отверстие.
10. Необходимо разобраться, с алгоритмом работы:

1. Глобальной Бинаризации.
2. Автоматического выбора порога бинаризации по методу Отцу.
3. Адаптивной бинаризации (cv.ADAPTIVE_THRESH_MEAN_C).

11. Необходимо описать алгоритм поиска Bounding Box для каждой метки, который вы реализуете.
12. Подготовить отчет, который будет содержать теоретическую справку по алгоритмам бинаризации, построения Bounding Box и описание вашей программы, которая должна иметь следующий интерфейс:
    1. На вход подается путь до изображения.
    2. Программа отображает каждый этап модификации изображения и результирующее изображения с ограничивающими прямоугольниками, которые показывают отверстия в деталях.

Вопросы для самопроверки

1. Что такое бинаризация изображения?
2. Какие виды бинаризации бывают и чем они отличаются?
3. Как метод Отцу вычисляет порог бинаризации?
4. Что такое оператор свертки?
5. Как с помощью оператора свертки выполнить размытие изображения?
6. Как работает оператор Собеля?

Лабораторная работа 3 «Сегментации изображений»

Цель: Знакомство с алгоритмом сегментации водоразделов.

Теоретическая часть

В цифровой обработке изображений и компьютерном зрении сегментация изображения — это процесс разделения цифрового изображения на несколько сегментов изображения, также известных как области изображения или объекты изображения (наборы пикселей). Цель сегментации — упростить и/или изменить представление изображения на нечто более значимое и более простое для анализа.

Точнее, сегментация изображения — это процесс присвоения метки каждому пикселю изображения таким образом, чтобы пиксели с одинаковой меткой имели определенные характеристики.

Алгоритм сегментации изображения на основе подхода «водораздела» выполняет рассмотрение изображения в оттенках серого, как карты высот – рельеф местности. Яркость пикселя описывает высоту земной поверхности в соответствующей точке. Подход «водораздела» подразумевает равномерное затопление рельефа местности водой. На всем изображении поднимается уровень воды (начиная с нуля), таким образом образуются водосборные бассейны. Затопление происходит до тех пор, пока все изображение не будет «залито» водой. При соединении двух и более водосборных бассейнов выполняется выстраивание дамбы. В результате чего, после заливки водой,

изображение будет разбито дамбами (рис 1). Таким образом водосборные бассейны будут отделены друг от друга. Водосборный бассейн – это один сегмент изображения. Дамба – это граница между двумя сегментами изображения. Таким образом метод сегментации watershed, каждый пиксель изображения может отнести к одному из двух классов:

1. Пиксель, принадлежащий к некоторому сегменту (Водосборному бассейну).
2. Пиксель, принадлежащий к границе между сегментами (линия водораздела или дамба).

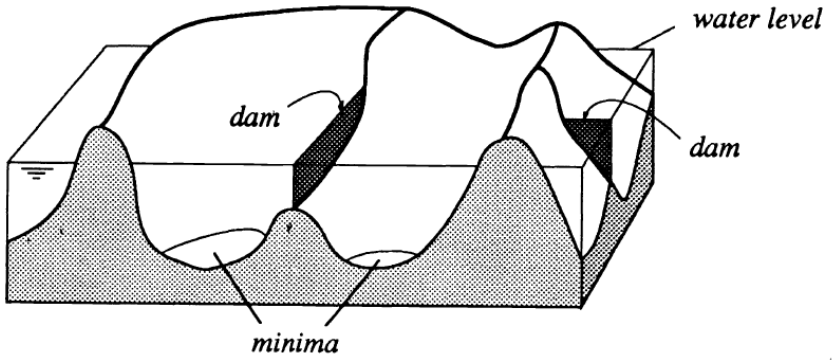

Рис. 1. Представление равномерного затопления топографической местности [4]

Данный подход может быть реализован по-разному. Например, существует непосредственная реализация этой идеи: «Моделирование процесса затопления» [4]. Но данный подход плохо применим, так как любой локальный минимум приводит к формированию сегмента. На зашумленных изображениях алгоритм приводит к чрезмерной сегментации (это когда объекты изображения разбиваются на большое количество мелких сегментов, которые не несут никакой ценности). Для борьбы с этой проблемой были придуманы маркерные алгоритмы сегментации. Маркер — это область изображения (множество пикселей), которая принадлежат одному сегменту изображения. С пикселей, являющимися маркерами, начинается потоп, который заливает все изображение. Маркеры могут быть установлены вручную человеком, или автоматический найдены с помощью дополнительных алгоритмов.

В OpenCV реализован маркерный алгоритм Мейера [5] (cv.watershed). Основная идея которого заключается в следующем:

1. Выбирается набор маркеров.
2. Соседние пиксели каждой отмеченной области вставляются в очередь приоритетов с уровнем приоритета, соответствующим величине градиента пикселя.
3. Пиксель с наивысшим уровнем приоритета извлекается из очереди приоритетов. Если все соседи извлеченного пикселя, которые уже были помечены, имеют одинаковую метку, то пиксель помечается их меткой.

Все не отмеченные соседи, которые еще не находятся в очереди приоритетов, помещаются в эту очередь.

4. Повторяйте шаг 3, пока очередь приоритетов не станет пустой;
5. Не маркированные пиксели представляют собой линии водораздела (границы между сегментами).

Практическая часть

Необходимо реализовать программное обеспечение для ручной сегментации изображения методом watershed. Программа должна предоставлять GUI для ручного рисования множества маркеров, которые затем будут использоваться в алгоритме сегментации.

Необходимо самостоятельно выбрать цветное изображение из интернета, и выполнить его сегментацию на несколько сегментов с помощью разработанного ПО.

Для реализации данного ПО необходимо выполнить следующую последовательность действий:

1. Выполнить загрузку изображения:

```
1: img = cv.imread(filename)
2: img = cv.resize(img, (600, 400))
3: imgCopy = np.copy(img)
4: cv.imshow('img', img)
```

2. Настроить обработку событий мыши окна img, для реализации механизма рисования маркеров:

```
1:    mask = np.zeros(shape = (img.shape[0], img.shape[1]),
2:                     dtype = np.uint8)
3:    cv.setMouseCallback("img", onMouseCallback)
4:    while True:
5:        c = cv.waitKey()
6:            print("c =", c)
7:        if c == 27: # esc
8:            break
9:         if c == 114: # r
10:       mask = np.zeros(shape = (img.shape[0],
11:                                img.shape[1]),
12:                       dtype = np.uint8)
13:       img = np.copy(imgCopy)
14:       cv.imshow('img', img)
15:        if c == 32: # space
```

```
16:             pass # здесь необходимо реализовать логику
17:                  # работы
18:                  #   с алгоритмом сегментации
19:                  # cv.watershed
```

В строчке 3 мы устанавливаем обратный вызов, который будет вызываться каждый раз, когда происходит событие связанное с мышью в окне img. Начиная со строчки 4 мы запускаем цикл событий, который ждет нажатия клавиш клавиатуры и событий мыши. Когда фокус находится на окне img и мы нажимаем «esc», то мы выходим из цикла (строчки 7-8), когда мы нажимаем на букву «r», то выполняется операция очистки нарисованных маркеров (строчки 9-14). Когда нажимаем на «пробел» то запускается механизм сегментации, который будет описан ниже.

3. Необходимо реализовать callback на события мыши, с целью создания механизма рисования маркеров:

```
1: mousePressed = False
2: prevPoint = (-1, -1)
3: mask = None
4: img = None
5:
6: def onMouseCallback(event,x,y,flags,param):
7:     global mousePressed
8:     global prevPoint
9:     if event == cv.EVENT_LBUTTONDOWN:
10:             mousePressed = True
11:             print("EVENT_LBUTTONDOWN")
12:         elif event == cv.EVENT_LBUTTONUP:
13:             mousePressed = False
14:             prevPoint = (-1, -1)
15:             print("EVENT_LBUTTONUP")
16:         if mousePressed:
17:             pt = (x, y)
18:             if prevPoint == (-1, -1):
19:                 prevPoint = pt
20:                 return
21:             cv.line(mask,
22:                     prevPoint,
23:                     pt,
24:                     (255, 255, 255),
25:                     5,
```

```
26:                     8,
27:                     0)
28:             cv.line(img,
29:                     prevPoint,
30:                     pt,
31:                     (255, 255, 255),
32:                     5,
33:                     8,
34:                     0)
35:             cv.imshow("img", img)
36:             prevPoint = pt
```

В строчках 1-4 объявляем глобальные переменные. Callback вызывается на различные события мыши, мы будем использовать, такие события как: движение мыши, нажатие и отпускание левой кнопки. На первое событие нажатия левой кнопки мыши, мы будем запоминать позицию курсора (строки 16-20), пока мы держим левую кнопку мыши и двигаем мышь, то между последовательными позициями курсора мыши мы рисуем прямые лини (строки 21-36). Прямые лини рисуются, как на изображении, которое отображается в окне, так и в матрице mask, которая будет использоваться для формирования маркеров.

4. Реализация механизма ручной сегментации, который должен быть вставлен на место строчки 16 в листинге пункта 2.

```
1: print("start watershed")
2: contours, hierarchy = cv.findContours(
3:                             mask,
4:                             cv.RETR_TREE,
5:                             cv.CHAIN_APPROX_SIMPLE)
6: if len(contours) == 0:
7:     continue
8: markers = np.zeros(
9:                 shape = (img.shape[0], img.shape[1]),
10:                     dtype = np.int32)
11:     color = 1
12:     for i in range(len(contours)):
13:         cv.drawContours(markers, contours, i,
14:                             color, 2, cv.LINE_8, hierarchy,
15:                             0)
16:         color = color + 1
17:     imgGray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
```

```
18:    colorTab = [rgb_random() for _ in
  range(len(contours))]
19:    wmarkers = cv.watershed(imgCopy, markers)
20:    wshed = np.zeros_like(img)
21:    for i in range(img.shape[0]):
22:        for j in range(img.shape[1]):
23:            index = wmarkers[i, j]
24:            if index == -1:
25:                wshed[i, j] = [255, 255, 255]
26:            else:
27:                wshed[i, j] = 0.5 * colorTab[index-1]
28:                              + 0.5 * imgGray[i, j]
29:    cv.imshow('img', wshed)
```

Когда мы рисовали кривые в окне img, то они рисовались, как на изображении, которые отображалось в окне, так и матрице mask, которую теперь необходимо преобразовать в маркеры. Каждый маркер должен обладать собственным идентификатором, поэтому нам необходимо разделить маркеры: не пересекающиеся кривые (несвязанные прямым соседством наборы точек) являются разными маркерами. С помощью функции поиска контуров, мы определим все нарисованные кривые (строка 2). После чего мы можем перейти к формированию маркеров:

1. (Строка 8) Сначала мы выделим память для маркеров.
2. (Строка 13) Затем нарисуем все кривые, в качестве цвета для рисования будем использовать идентификатор кривой.
3. (Строка 17) Для каждого будущего сегмента необходимо выбрать цвет, которым мы будем обозначать этот сегмент. Для этого воспользуемся генерацией случайного цвета для каждого идентификатора сегмента.
4. (Строка 18) Запускаем сегментацию.
5. (Строки 19-27) Рисуем сегменты, используем линейную комбинацию цвета сегмента, к которому относится конкретный пиксель, и яркость пикселя.
6. (Строка 28) Отображаем полученный результат.

Для формирования случайного цвета предлагается использовать следующий метод:

```
1: def rgb_random():
2:     return np.random.randint(0,
3:                              255,
4:                              size=3,
5:                              dtype=np.uint8)
```

Пример работы программы:

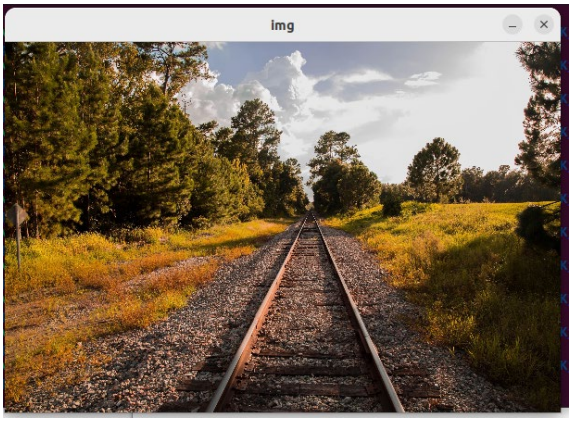

Рис. 2. Исходное состояние окна с изображением

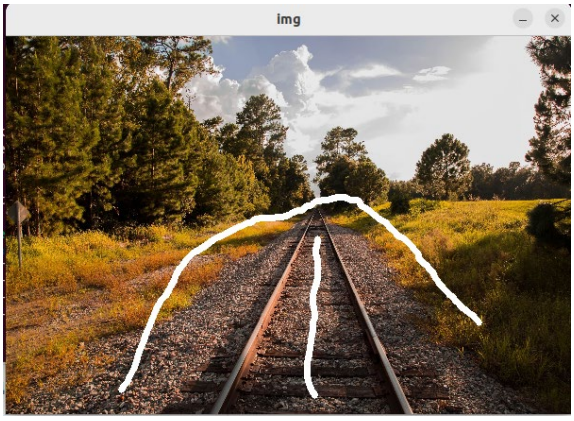

Рис. 3. Окно с изображением, на котором вручную было нанесено два маркера

Рис. 4. Результат сегментации

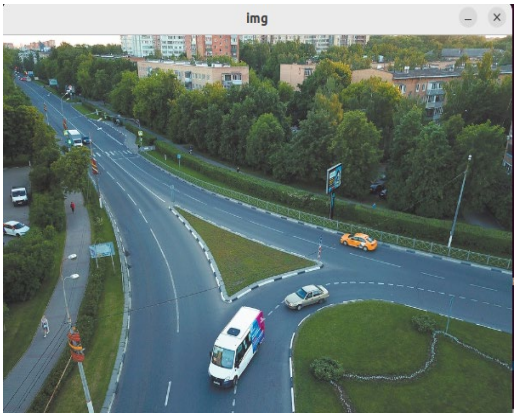

Рис. 5. Исходное состояние окна с изображением

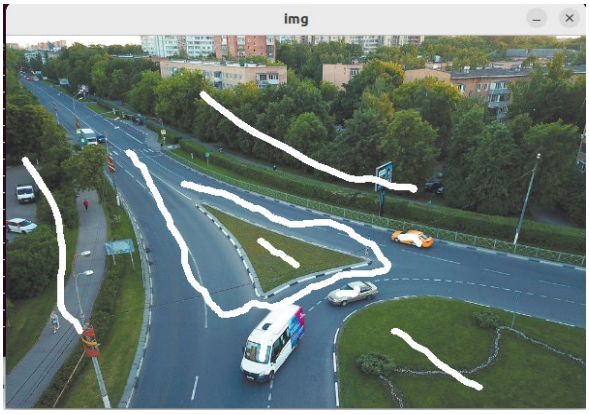

Рис. 6. Окно с изображением, на котором вручную были нанесены маркеры

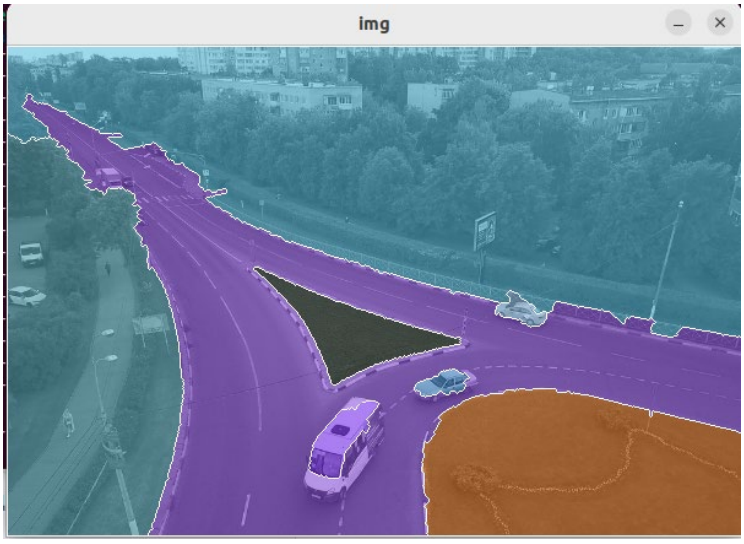

Рис. 7. Результат сегментации

Вопросы для самопроверки

1. Что делает алгоритм сегментация изображения?
2. Что такое чрезмерная сегментация?
3. В чем заключается подход водоразделов для сегментации изображения?
4. Что такое маркер в маркерном алгоритме сегментации изображения?
5. От чего зависит количество сегментов, на которое разбивается изображения, в алгоритме cv.watershed?
6. На какие классы разбиваются пиксели при сегментации методом водоразделов?
7. Объясните логику работы механизма рисования маркеров.
8. Объясните логику работы алгоритма преобразования нарисованных линий от руки в маркеры, которые передаются в алгоритм сегментации.

Лабораторная работа 4
«Распознавание объекта на изображении»

Цель: Знакомство с методикой применения нейронных сетей для решения задач обнаружения объектов на изображениях с использованием открытой библиотеки TensorFlow.

Теоретическая часть

В последнее десятилетие для решения задачи обнаружения и слежения за объектами интереса стали активно применяться нейросетевые решения.

Искусственные нейронные сети используются в случае, когда необходимо решать задачи прогнозирования, классификации или автоматизации, для которых нет аналитических методов решения либо они весьма трудоемки в реализации, но есть большое количество статистических данных о результатах решения тем или иным способом.

Для работы с изображениями применяются сверточные нейронные сети, представляющие из себя набор последовательно и параллельно расположенных связанных между собой слоев, каждый из которых выполняет некоторые преобразования полученного изображения и передает их в следующий слой. В большинстве имеющихся решений задачи распознавания объектов интереса предполагается, что сеть предварительно обучена распознаванию этих объектов.

Основными операциями, которые реализуются в сверточных нейронных сетях для обработки изображений, являются операции свертки и субдискретизации.

Свертка изображения реализуется с помощью отдельного слоя сверточной нейронной сети и заключается в следующем:

1. Входное изображение интерпретируется как двумерный массив пикселов.
2. Входное изображение предварительно обрабатывается с целью нормализации — значения всех пикселов масштабируются для попадания в диапазон [0, 1].
3. Применяется операция свертки изображения с ядром (Формальное определение операции свертки было дано в лабораторной работе 2):
   - Ядро (фильтр) – это матрица коэффициентов (как правило, используется ядро в виде области 3x3 пикселов). Значения ядра изначально инициализируется случайными значениями, а затем регулируется в процессе обучения;
   - Выполняется "сканирование" исходного изображения данным ядром. Сверхточный слой применяется к каждому пикселу входного изображения: пиксел и сверточное ядро центрируются, каждый пиксел изображения умножается на соответствующий пиксел ядра и все значения произведений суммируются. Значение суммы запоминается в качестве результирующего значения пиксела выходного изображения;
4. Граничные пикселы входного изображения, для которых центрированное ядро выходит за границы изображения, обрабатываются особым образом (например, значения за пределами исходного изображения могут считаться нулевыми).

Операция субдискретизации (подвыборки, pooling, max pooling) — это операция уменьшения пространственных размеров изображения путем обработки значений блоков пикселов [6, 7]. Типичная операция субдискретизации выполняется следующим образом.

1. Определяется размер выборки (размер прямоугольной сетки) и величина шага по изображению. Пример: размер выборки 3x3 и шаг 3 пиксела.
2. Находится максимальное значение пиксела, попадающее в выделенную сетку. Это значение "переносится" в генерируемое выходное изображение. Сетка сдвигается на один шаг и процесс выборки максимального значения и его переноса на выходное изображение повторяется.

3. По результатам проделанной операции формируется изображение меньшего размера по сравнению с входным. Размер итогового изображения зависит от выбора размера сетки и величины шага.

Процесс обучения сверточной нейронной сети заключается в оптимизации значений в сверточных слоях.

В качестве примера порядка работы со сверточной нейронной сетью в данной лабораторной работе применяется модель обнаружения объектов (предварительно обученные классификаторы с определенной архитектурой нейронных сетей) из "зоопарка моделей" для библиотеки машинного обучения TensorFlow. Этот репозитарий программных решений на основе библиотеки TensorFlow представлен в открытом доступе [8]. Представленные в нем модели явно или косвенно состоят из двух модулей: модуль определения областей (регионов) расположения объектов на изображении и модуль классификации объектов в найденных областях.

Комбинация методов нахождения областей и классификации обнаруженных объектов позволяет добиваться различных характеристик скорости, точности обнаружения и точности классификации объектов.

При использовании моделей из рассматриваемого репозитария [8] возможен выбор модели для обучения классификатора обнаружения в зависимости от поставленной задачи и устройства, на котором будут выполняться вычисления.

В качестве примера для обнаружения и классификации объектов будет использоваться нейронная сеть Faster-RCNN-Inception-V2, предназначенная для обнаружения объектов по заданной обучающей выборке.

Выбранная нейронная сеть состоит из двух основных модулей: первый модуль представляет собой глубокую сверточную сеть, определяющую области предполагаемых объектов Region Proposal Networks (RPN), второй модуль является детектором, использующим ранее определенные области для классификации объектов в пределах этих областей.

Модуль Faster-RCNN представляет собой модификацию разработанной ранее модели Fast-RCNN, основной особенностью которой является параллельное обучение как сети поиска областей с интересующим объектом, так и сети их классификации. Данная модель характеризуется большей скоростью работы по сравнению с Fast-RCNN.

Для классификации в выбранной нейросетевой модели используется модуль Inception-V2 [9, 10].

В целом, процесс обработки входного изображения с использованием выбранной модели Faster-RCNN-Inception-V2 происходит в следующей последовательности:

1. Построение карты признаков изображения с помощью нейронной сети.
2. Генерация гипотез на основе полученной карты признаков — определение приближенных значений координат и признаков наличия объектов любого поддерживаемого класса.

3. Сопоставление координат в составе гипотез с картой признаков, полученной на первом шаге.
4. Классификация гипотез (для определения конкретного класса) и дополнительное уточнение координат расположения объектов.

Практическая часть

Предполагается, что эти перечисленные в данной работе действия выполняются на компьютере под управлением ОС Windows 10.

Для использования TensorFlow требуется установить дистрибутив Anaconda для поддержки языка программирования Python. Дистрибутив для загрузки доступен на официальном сайте [11].

Кроме дистрибутива Anaconda, для дальнейшей работы необходима среда разработки Microsoft Visual Studio.

Библиотека TensorFlow предполагает использование определенной структуры каталогов, представленной в репозитории GitHub TensorFlow. Для запуска или обучения модели обнаружения объектов также требуется установить ряд дополнительных пакетов Python, модифицировать значения системных переменных окружения PATH и PYTHONPATH и выполнить несколько дополнительных команд настройки установленных программных компонент.

В дальнейшем будет предполагаться, что для обучения и запуска нейросетевой модели используется следующая структура каталогов. Рабочий каталог c:\tensorflow1 будет содержать программные модули TensorFlow, а также обучающие изображения, обучающие данные, обученный классификатор, файлы конфигурации и остальные компоненты, необходимые для работы классификатора обнаружения объектов.

Репозиторий обнаружения объектов TensorFlow следует загрузить по адресу https://github.com/tensorflow/models. Из загруженного архива необходимо извлечь каталог models-master и поместить его в каталог c:\tensorflow1. В данной работе каталог models-master был переименован в models.

Содержимое репозитория моделей TensorFlow обновляется разработчиками. Рассматриваемый пример решения задачи обнаружения объекта был выполнен с TensorFlow версии 1.14.0.

Далее следует загрузить из "зоопарка моделей" TensorFlow [8] нейросетевую модель, которую планируется использовать – в данной работе рассматривается модель faster_rcnn_inception_v2_coco. Загруженный архив требуется перенести в каталог \object_detection, а его содержимое в подкаталог \training. В каталоге c:\tensorflow1\models\research\object_detection необходимо создать три подкаталога inference_graph, training и images. В подкаталоге images необходимы подкаталоги test и train. В процессе работы в них будут загружаться изображения и данные об обучении сетей.

Библиотека TensorFlow непосредственно не работает с изображениями в формате CSV. Модуль LabelImg, который в дальнейшем будет использоваться

для разметки тренировочных и тестовых изображений, генерирует файлы в формате XML. В этих файлах хранятся данные об изображении, например, местоположение, размеры и классы объектов. Для передачи данных о размеченных изображениях в модель TensorFlow требуется выполнить преобразование из формата XML в CSV, а затем в формат TFRecord, который используется TensorFlow при обучении моделей. Данный формат специально оптимизирован для использования с TensorFlow.

Для выполнения преобразований форматов данных предназначены программы generate_tfrecord.py и xml_to_csv.py, тексты которых представлены ниже. Файлы с исходным текстом следует поместить в каталог \object_detection.

Код программы преобразования файлов формата csv в record «generate_tfrecord.py»:

```
from __future__ import division
from __future__ import print_function
from __future__ import absolute_import

import os
import io
import pandas as pd

from tensorflow.python.framework.versions import VERSION
if VERSION >= "2.0.0a0":
    import tensorflow.compat.v1 as tf
else:
    import tensorflow as tf

from PIL import Image
from object_detection.utils import dataset_util
from collections import namedtuple, OrderedDict

flags = tf.app.flags
flags.DEFINE_string('csv_input', '', 'Path to the CSV input')
flags.DEFINE_string('image_dir', '',
                    'Path to the image directory')
flags.DEFINE_string('output_path', '',
                    'Path to output TFRecord')
FLAGS = flags.FLAGS

def class_text_to_int(row_label):
    if row_label == 'target':
        return 1
    else:
```

```
        None

def split(df, group):
    data = namedtuple('data', ['filename', 'object'])
    gb = df.groupby(group)
    return [data(filename, gb.get_group(x)) for filename, x in
            zip(gb.groups.keys(), gb.groups)]


def create_tf_example(group, path):
    with
        tf.gfile.GFile(os.path.join(path,
                                    '{}'.format(group.filename)),
                       'rb') as fid:
        encoded_jpg = fid.read()
       encoded_jpg_io = io.BytesIO(encoded_jpg)
    image = Image.open(encoded_jpg_io)
    width, height = image.size

    filename = group.filename.encode('utf8')
    image_format = b'jpg'
    xmins = []
    xmaxs = []
    ymins = []
    ymaxs = []
    classes_text = []
    classes = []

    for index, row in group.object.iterrows():
        xmins.append(row['xmin'] / width)
        xmaxs.append(row['xmax'] / width)
        ymins.append(row['ymin'] / height)
        ymaxs.append(row['ymax'] / height)
        classes_text.append(row['class'].encode('utf8'))
        classes.append(class_text_to_int(row['class']))

    tf_example = tf.train.Example(
                    features=tf.train.Features(
                                feature={
     'image/height': dataset_util.int64_feature(height),
     'image/width': dataset_util.int64_feature(width),
     'image/filename': dataset_util.bytes_feature(filename),
```

```
    'image/source_id': dataset_util.bytes_feature(filename),
    'image/encoded': dataset_util.bytes_feature(encoded_jpg),
    'image/format': dataset_util.bytes_feature(image_format),
    'image/object/bbox/xmin':
                        dataset_util.float_list_feature(xmins),
    'image/object/bbox/xmax':
                        dataset_util.float_list_feature(xmaxs),
    'image/object/bbox/ymin':
                        dataset_util.float_list_feature(ymins),
    'image/object/bbox/ymax':
                        dataset_util.float_list_feature(ymaxs),
    'image/object/class/text':
                dataset_util.bytes_list_feature(classes_text),
    'image/object/class/label':
                    dataset_util.int64_list_feature(classes),
                                        }
                                )
                )
    return tf_example

def main(_):
    writer = tf.python_io.TFRecordWriter(FLAGS.output_path)
    path = os.path.join(os.getcwd(), FLAGS.image_dir)
    examples = pd.read_csv(FLAGS.csv_input)
    grouped = split(examples, 'filename')
    for group in grouped:
        tf_example = create_tf_example(group, path)
        writer.write(tf_example.SerializeToString())

    writer.close()
    output_path = os.path.join(os.getcwd(), FLAGS.output_path)
    print('Successfully created the TFRecords:
           {}'.format(output_path))

if __name__ == '__main__':
    tf.app.run()
```

Код программы преобразования файлов формата xml в csv «xml_to_csv.py»:

```
import os
import glob
import pandas as pd
import xml.etree.ElementTree as ET
```

```
def xml_to_csv(path):
    xml_list = []
    for xml_file in glob.glob(path + '/*.xml'):
        tree = ET.parse(xml_file)
        root = tree.getroot()
        for member in root.findall('object'):
            value = (root.find('filename').text,
                     int(root.find('size')[0].text),
                     int(root.find('size')[1].text),
                     member[0].text,
                     int(member[4][0].text),
                     int(member[4][1].text),
                     int(member[4][2].text),
                     int(member[4][3].text)
                     )
            xml_list.append(value)
    column_name = ['filename',
                   'width',
                   'height',
                   'class',
                   'xmin',
                   'ymin',
                   'xmax',
                   'ymax']
    xml_df = pd.DataFrame(xml_list, columns=column_name)
    return xml_df

def main():
    for folder in ['train','test']:
        image_path = os.path.join(os.getcwd(),
                                  ('images/' + folder))
        xml_df = xml_to_csv(image_path)
        xml_df.to_csv(('images/' + folder + '_labels.csv'),
                      index=None)
        print('Successfully converted xml to csv.')

main()
```

В среде MS Windows требуется открыть окно с командной строкой Anaconda Prompt (посредством запуска от имени администратора). В появившемся командном терминале создайте новую виртуальную среду под названием tensorflow1 с помощью следующей команды:

```
conda create -n tensorflow1 pip python
```

В данной работе при настройке TensorFlow использовалась среда Anaconda с версией Python 3.6.13, при дальнейшей работе (для обработки изображений и видеопоследовательностей) использовалась среда Visual Studio 2019 с версией Python 3.8.3. Затем следует активировать среду и обновить систему управления программными пакетами Python pip (в данной работе использовался pip версии 21.0.1):

```
C:\> activate tensorflow1
(tensorflow1) C:\>python -m pip install --upgrade pip
```

Далее следует произвести установку библиотеки TensorFlow. Для установки доступны пакеты tensorflow и tensorflow-gpu. Пакет tensorflow предназначен для обработки с использованием центрального процессора, а пакет tensorflow-gpu – с использованием графического процессора. В процессе настройки и тестирования в данной работе использовалась версия TensorFlow для расчетов с использованием только центрального процессора версии 1.14.0:

```
pip install tensorflow==1.14.0
```

При наличии графического процессора возможно выполнить установку пакета tensorflow-gpu:

```
(tensorflow1) C:\> pip install --ignore-installed
                    --upgrade tensorflow-gpu
```

При работе с пакетом tensorflow-gpu с использованием графического процессора архитектуры NVidia следует убедиться, что используются версии библиотек NVidia CUDA и NVidia cuDNN, совместимые с используемой версией TensorFlow [12]. Как правило, Anaconda автоматически устанавливает версии CUDA и cuDNN, необходимые для использования TensorFlow на графическом процессоре.

Установите необходимые для дальнейшей работы пакеты Python, выполнив следующие команды:

```
(tensorflow1) C:\> conda install -c anaconda protobuf
(tensorflow1) C:\> pip install pillow
(tensorflow1) C:\> pip install lxml
(tensorflow1) C:\> pip install Cython
(tensorflow1) C:\> pip install contextlib2
(tensorflow1) C:\> pip install jupyter
(tensorflow1) C:\> pip install matplotlib
(tensorflow1) C:\> pip install pandas
(tensorflow1) C:\> pip install opencv-python
```

В процессе дальнейшей работы при запуске обучения нейронной сети или последующей обработки ею поступающих на вход изображений возможно возникновение ошибок, связанных с несовместимостью версий вспомогательных библиотек между собой из-за постоянного выпуска новых версий, изменения стиля обращений к функциям и т.д. При возникновении ошибок такого рода следует определить установленную версию библиотеки и

проверить её совместимость с другими библиотеками. При необходимости, понизить версию той или иной библиотеки в окружении.

Указанные выше пакеты pandas и opencv-python не требуются для работы TensorFlow, но будут использоваться для формирования вспомогательных файлов и работы с файлами изображений и видеопоследовательностей.

Для среды выполнения TensorFlow требуется создать переменную PYTHONPATH, которая будет хранить пути к каталогам \models, \models\research и \models\research\slim:

```
(tensorflow1) C:\> set PYTHONPATH=c:\tensorflow1\models;
                   c:\tensorflow1\models\research;
                   c:\tensorflow1\models\research\slim
```

Переменную PYTHONPATH следует настраивать при каждом запуске виртуальной среды tensorflow1. Для хранения сведений о сетевой модели и параметрах ее обучения в TensorFlow используются файлы формата Protocol Buffers [13]. Эти файлы .proto необходимо обработать с помощью компилятора protoc для получения исходных файлов с реализацией доступа данных на языке Python, например:

```
cd c:\tensorflow1\models\research
protoc                                          --python_out=.
.\object_detection\protos\anchor_generator.proto
```

В каталоге \object_detection\protos необходимо обработать несколько десятков файлов данных формата .proto, в т.ч.: argmax_matcher.proto, bipartite_matcher.proto, …, string_int_label_map.proto, train.proto. В результате обработки для всех файлов name.proto в каталоге \object_detection\protos будут созданы файлы name_pb2.py. В случае, если были обработаны не все файлы данных, впоследствии TensorFlow может сгенерировать сообщение об ошибке вида “ImportError: cannot import name ‘XXX’”.

Далее следует выполнить команды для настройки сетевой модели обнаружения объектов:

```
(tensorflow1) c:\tensorflow1\models\research> python
              setup.py build
(tensorflow1) c:\tensorflow1\models\research> python
              setup.py install
```

После выполненной установки и настройки сетевой модели требуется подготовить изображения, которые будут использоваться для обучения нового классификатора обнаружения. Предпочтительно, чтобы обучающие изображения представляли характерные обрабатываемые сцены – содержащие объекты интереса, возможные посторонние объекты, демонстрирующие различные условия фона и освещения.

Для обучения сетевой модели необходимо разделить файлы видеопоследовательностей на отдельные кадры с использованием открытой программы VirtualDub.

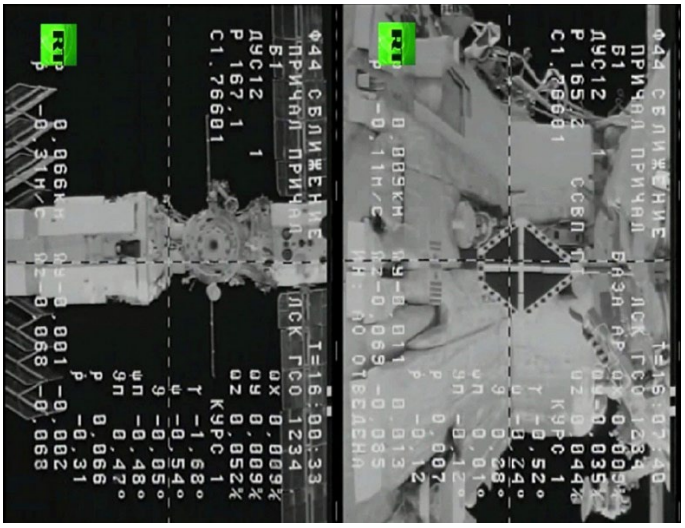

Рис. 1. Примеры обучающих изображений из видеопоследовательности [14]

Точность результатов распознавания обученной сетевой модели зависит от состава обучающих изображений. К общим рекомендациям относится увеличение количества изображений с отмеченными целевыми объектами, а также включение в обучающую выборку изображений с возможными вариантами фона (частей изображения, не относящихся к целевым объектам). Большие по размерам и общему объему изображения увеличивают время обучения и работы классификатора. В зависимости от выбранной сетевой модели [8] рекомендуется подавать на вход изображения различных размеров. После формирования обучающих и тестовых изображений, 20% из них были перемещены в каталог \object_detection\images\test, а 80% – в каталог \object_detection\images\train (данные значения взяты из рекомендаций [8] и могут варьироваться).

Разметка набора обучающих изображений производилась с использованием открытого приложения LabelImg [15, 16]. Перед началом работы следует определиться с наименованиями объектов, которые требуется в дальнейшем классифицировать на изображениях и придерживаться их как на этапе разметки изображений, так и при настройке модели TensorFlow. В данной работе сеть обучалась на поиск единственного класса объектов — "target", представляющего собой прямоугольную область, в которой располагается стыковочная мишень (рис. 2).

На части изображений искомый объект интереса может отсутствовать, при его наличии в кадре с течением времени его размер и положение изменяются. Эти факторы учитываются для повышения разнообразия выборки тестовых изображений.

После загрузки и установки LabelImg [16], при запуске программы был задан рабочий каталог \images\train. Затем на каждом обучающем изображении была отмечена прямоугольная область объекта интереса и указан его класс. Эти данные в виде карты меток были сохранены в каталоге обучающих изображений

изображения. Эта процедура также была выполнен для всех изображений в каталоге изображений тестовой выборки \images\test.

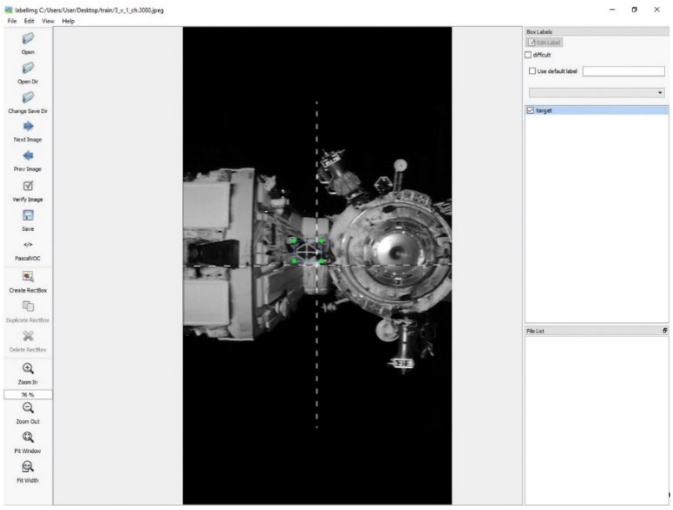

Рис. 2. Окно приложения LabelImg [15] для разметки изображений

Приложение LabelImg формирует файлы .xml, содержащие данные разметки для отдельных изображений. Эти файлы .xml будут использоваться для создания структур формата TFRecord, которые могут использоваться в качестве входных данных для библиотеки TensorFlow. После разметки всех изображений, в каталогах \test и \train для каждого изображения будет по одному файлу формата XML.

Для размеченных изображений следует сгенерировать соответствующие файлы формата TFRecord, которые служат входными данными для модели обучения TensorFlow. Сценарии xml_to_csv.py и generate_tfrecord.py.

В каталоге \object_detection после выполнения следующей команды в командной строке Anaconda:

```
(tensorflow1)
 c:\tensorflow1\models\research\object_detection>
      python xml_to_csv.py
```

будут созданы файлы train_labels.csv и test_labels.csv в каталоге \object_detection\images. С помощью текстового редактора в сценарий generate_tfrecord.py необходимо внести изменения. Требуется заменить карту меток на собственную карту, в которой каждому объекту будет присвоен идентификационный номер. Такое же присвоение номеров будет использовано при настройке файла labelmap.pbtxt на следующем шаге.

```
def class_text_to_int(row_label):
                if row_label == 'target':
                     return 1
                else:
                     None
```

Для генерации файлов формата TFRecord в каталоге \object_detection выполняются следующие команды:

```
python generate_tfrecord.py
  --csv_input=images\train_labels.csv
  --image_dir=images\train --output_path=train.record

python generate_tfrecord.py
  --csv_input=images\test_labels.csv
  --image_dir=images\test --output_path=test.record
```

Эти команды генерируют файлы train.record и test.record в \object_detection. Они будут использоваться для обучения нового классификатора обнаружения объектов.

Последнее, что нужно выполнить перед обучением, — это создать карту меток и отредактировать файл конфигурации обучения.

Карта меток сообщает, что представляет собой каждый объект, определяя сопоставление имен классов с номерами идентификаторов классов. С помощью текстового редактора следует создать новый файл и сохранить его под именем labelmap.pbtxt в каталоге c:\tensorflow1\models\research\object_detection\training. Идентификационные номера карты меток должны совпадать с изменениями, выполненными ранее в файле generate_tfrecord.py. В нашей задаче файл labelmap.pbtxt должен выглядеть так:

```
item {
    id: 1
    name: 'target'
}
```

У рассматриваемой нейросетевой модели есть набор настраиваемых параметров, которые влияют на ее обучение. Они хранятся в файле конфигурации faster_rcnn_inception_v2_pets.config. Этот файл необходимо скопировать из каталога

c:\tensorflow1\models\research\object_detection\samples\configs

в каталог \object_detection\training. Вносимые изменения заключаются в указании количества классов объектов интереса, количества и расположения файлов с обучающими данными. Эти изменения файла fast_rcnn_inception_v2_pets.config перечислены в табл. 1. Пути задаются с помощью одинарных косых черт (не обратных косых черт), в противном случае TensorFlow выдаст ошибку пути к файлу при попытке обучения модели. Кроме того, пути должны быть заключены в двойные кавычки ("), а не в одинарные кавычки ('). При выборе для дальнейшей работы иной нейронной сети следует выбрать иной файл конфигурации XXX.config, где XXX – наименование выбранной сети, в таком случае в дальнейшем по тексту работы fast_rcnn_inception_v2_pets или fast_rcnn_inception_v2_coco будет заменено на

соответствующий тип XXX, например, faster_rcnn_resnet50_coco.config или ssd_mobilenet_v1_coco.config.

Таблица 1

Перечень изменений в файле конфигурации нейросетевой модели

Строка	Изменения
9	Параметр num_classes задает количество классов объектов интереса, которые должен обнаруживать классификатор. Для рассматриваемой задачи num_classes: 1.
106	Указание контрольной точки обучения (для дообучения сети): fine_tune_checkpoint: "c:/tensorflow1/models/research/object_detection/faster_rcnn_inception_v2_coco_2018_01_28/model.ckpt"
123	Местоположение обучающих данных в разделе train_input_reader: input_path: "c:/tensorflow1/models/research/object_detection/train.record"
125	Перечисление классов объектов интереса в разделе train_input_reader: label_map_path: "c:/tensorflow1/models/research/object_detection/training/labelmap.pbtxt"
130	Количество тестовых изображение: параметр num_examples равен количеству изображений в каталоге \images\test.
135	Местоположение тестовых данных в разделе eval_input_reader: input_path: "c:/tensorflow1/models/research/object_detection/test.record"
137	Перечисление классов объектов интереса в разделе eval_input_reader: label_map_path: "c:/tensorflow1/models/research/object_detection/training/labelmap.pbtxt"

Начиная с версии TensorFlow 1.9, для обучения предназначен файл model_main.py. Использовавшийся ранее файл train.py доступен в каталоге /object_detection/legacy. Будем полагать, что файл train.py перемещен из /object_detection/legacy в каталог /object_detection. Для запуска процесса обучения в каталоге \object_detection следует выполнить команду:

```
python train.py --logtostderr --train_dir=training/
   --pipeline_config_path=
            training/faster_rcnn_inception_v2_pets.config
```

При корректном запуске будет запущен процесс обучения TensorFlow. В начале обучения производится инициализация, длительность которой зависит от производительности процессора (может составлять несколько десятков секунд). После запуска обучения в консольном окне будут выводиться информационные сообщения (рис. 3).

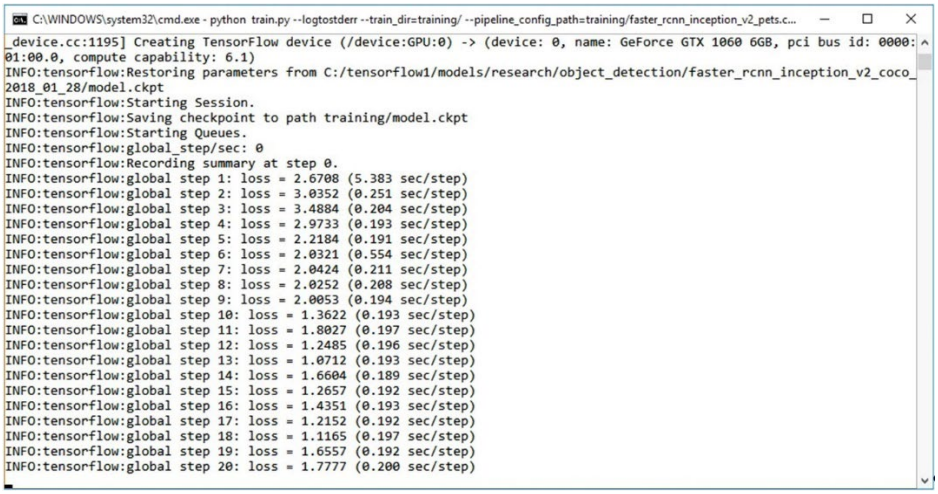

Рис. 3. Консольное окно с информацией TensorFlow в процессе обучения

На каждом шаге обучения отображается значение функции ошибки. Значения этой функции должны уменьшаться по мере обучения модели (рис 3). Поведение функции ошибок и её значения зависят от используемой нейросетевой модели.

При запуске обучения нейросетевой модели возможно возникновение ошибок, наиболее частые из них и решения по их устранению представлены ниже.

Ошибка	`ModuleNotFoundError: No module named 'tf_slim'`
Решение	(tensorflow1) C:\tensorflow1\models\research\object_detection>pip install tf_slim
Ошибка	`ModuleNotFoundError: No module named 'scipy'`
Решение	C:\tensorflow1\models\research\object_detection>pip install scipy
Ошибка	`ImportError: cannot import name 'fpn_pb2'`
Решение	C:\tensorflow1\models\research> protoc --python_out=. .\object_detection\protos\fpn.proto
Ошибка	`ImportError: cannot import name 'resnet'`
Решение	Откройте с помощью Блокнота файл resnet_v1.py, расположенный в папке C:\tensorflow1\models\research\object_detection\models\keras_models и замените строку from tensorflow.python.keras.applications import resnet строкой

	from tensorflow.python.keras.applications import resnet50
Ошибка	`ImportError: cannot import name 'center_net_pb2'`
Решение	(tensorflow1) C:\tensorflow1\models\research> protoc --python_out=. .\object_detection\protos\center_net.proto

Контроль процесса обучения можно дополнить данными, отображаемыми с помощью пакета TensorBoard. Для этого необходимо открыть новый экземпляр окна командной строки Anaconda Prompt, выбрать виртуальную среду tensorflow1, перейти в каталог c:\tensorflow1\models\research\object_detection и выполнить следующую команду:

```
(tensorflow1)
   c:\tensorflow1\models\research\object_detection>
     tensorboard --logdir=training
```

В результате с помощью веб-браузера можно будет открыть страницу, доступную по адресу localhost:6006. На этой странице пакет TensorBoard будет выводить информацию и графики, характеризующие ход процесса обучения (рис. 4).

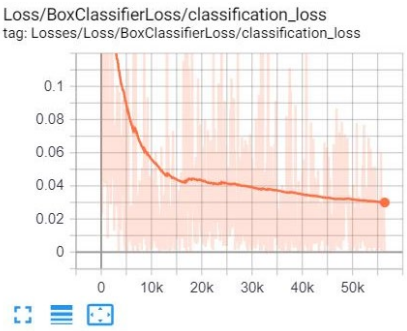

Рис. 4. Пример функции ошибки классификации объект в пакете TensorBoard

Программа обучения периодически сохраняет контрольные точки, в которых содержится информация, позволяющая продолжить обучение сети с соответствующего состояния или выполнить поиск объектов интереса на обрабатываемых изображениях. Процесс обучения можно прервать нажатием клавиш Ctrl+C в консольном окне.

После обучения нейронной сети, последним шагом является создание замороженного графа вывода (файл .pb). В каталоге \object_detection следует выполнить следующую команду, где "XXXX" в "model.ckpt-XXXX" следует заменить на имя файла .ckpt с наибольшим номером в каталоге обучения:

```
python export_inference_graph.py
    --input_type image_tensor
    --pipeline_config_path
       training/faster_rcnn_inception_v2_pets.config
```

```
        --trained_checkpoint_prefix training/model.ckpt-XXXX
        --output_directory inference_graph
```

Например:

```
    (tensorflow1)
c:\tensorflow1\models\research\object_detection>
    python export_inference_graph.py
        --input_type image_tensor
        --pipeline_config_path
            training/faster_rcnn_inception_v2_pets.config
        --trained_checkpoint_prefix training/model.ckpt-1040
        --output_directory inference_graph
```

В каталоге \object_detection\inference_graph будет сформирован файл frozen_inference_graph.pb. Этот файл .pb содержит обученный классификатор обнаружения объектов интереса.

Перед запуском сценариев Python необходимо изменить переменную NUM_CLASSES в сценарии, чтобы она была равна количеству классов, которые требуется обнаружить.

Далее рассматривается использование сценариев Python для применения обученной нейросетевой модели. Предполагается использование среды разработки Microsoft Visual Studio 2019. После ее установки требуется выполнить перечисленные далее действия 1-8.

1. В файл label_map_util.py в каталоге c:\tensorflow1\models\research\object_detection\utils необходимо ввести изменения. Они заключаются в комментировании или удалении части исходного текста:

```
if item.HasField('frequency'):
  if item.frequency ==
                    string_int_label_map_pb2
                                          .LVISFrequency
                                          .Value('FREQUENT'):
    category['frequency'] = 'f'
  elif item.frequency ==
                     string_int_label_map_pb2
                                            .LVISFrequency
                                            .Value('COMMON'):
    category['frequency'] = 'c'
  elif item.frequency ==
                        string_int_label_map_pb2
                                              .LVISFrequency
                                              .Value('RARE'):
    category['frequency'] = 'r'
```

```
if item.HasField('instance_count'):
   category['instance_count'] = item.instance_count
if item.keypoints:
  keypoints = {}
  list_of_keypoint_ids = []
  for kv in item.keypoints:
    if kv.id in list_of_keypoint_ids:
      raise
       ValueError('Duplicate keypoint ids are not allowed. '
                  'Found {} more than once'.format(kv.id))
    keypoints[kv.label] = kv.id
    list_of_keypoint_ids.append(kv.id)
  category['keypoints'] = keypoints
categories.append(category)
```

2. В каталоге c:\tensorflow1\models\research\object_detection следует создать файл Object_detection_image.py и открыть его в среде Visual Studio, ввести следующий код в файл:

```
import os
import cv2
import numpy as np
import tensorflow as tf
import sys

import tensorflow.compat.v1 as tf

sys.path.append("..")

from utils import label_map_util
from utils import visualization_utils as vis_util

MODEL_NAME = 'inference_graph'
IMAGE_NAME = 'test1.jpeg'
CWD_PATH = os.getcwd()
PATH_TO_CKPT =
 os.path.join(CWD_PATH,MODEL_NAME,'frozen_inference_graph.pb')
PATH_TO_LABELS =
 os.path.join(CWD_PATH,'training','labelmap.pbtxt')
PATH_TO_IMAGE = os.path.join(CWD_PATH,IMAGE_NAME)
NUM_CLASSES = 1

label_map = label_map_util.load_labelmap(PATH_TO_LABELS)
categories =
```

```
 label_map_util.convert_label_map_to_categories(label_map,
max_num_classes=NUM_CLASSES, use_display_name=True)
category_index =
 label_map_util.create_category_index(categories)

detection_graph = tf.Graph()
with detection_graph.as_default():
    od_graph_def = tf.compat.v1.GraphDef()
    with tf.io.gfile.GFile(PATH_TO_CKPT, 'rb') as fid:
        serialized_graph = fid.read()
        od_graph_def.ParseFromString(serialized_graph)
        tf.import_graph_def(od_graph_def, name='')

    sess = tf.Session(graph=detection_graph)

image_tensor =
 detection_graph.get_tensor_by_name('image_tensor:0')
detection_boxes =
 detection_graph.get_tensor_by_name('detection_boxes:0')
detection_scores =
 detection_graph.get_tensor_by_name('detection_scores:0')
detection_classes =
 detection_graph.get_tensor_by_name('detection_classes:0')
num_detections =
 detection_graph.get_tensor_by_name('num_detections:0')

image = cv2.imread(PATH_TO_IMAGE)
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
image_expanded = np.expand_dims(image_rgb, axis=0)

(boxes, scores, classes, num) = sess.run(
    [detection_boxes,
     detection_scores,
     detection_classes,
     num_detections],
      feed_dict={image_tensor: image_expanded})

print(boxes)
print(classes)
print(scores)
vis_util.visualize_boxes_and_labels_on_image_array(
    image,
```

```
    np.squeeze(boxes),
    np.squeeze(classes).astype(np.int32),
    np.squeeze(scores),
    category_index,
    use_normalized_coordinates=True,
    line_thickness=8,
    min_score_thresh=0.07
)

cv2.imshow('Object detector', image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

3. В качестве рабочей среды в Visual Studio должна быть выбрана среда Python (рис. 6).

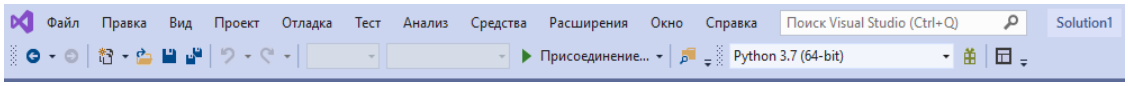

Рис. 6. Используемая рабочая среда Python в MS Visual Studio 2019

4. С помощью кнопки панели инструментов надо вызвать окно «Окружения Python».
5. В окне «Окружения Python» следует установить или обновить пакеты, необходимые для дальнейшей работы 5.1-5.3.

   5.1. Обновление менеджера пакетов pip.

   В окне «Окружения Python» требуется выбрать pip и нажать кнопку со стрелкой вверх, обозначающей команду обновления пакета. При ее отсутствии данный шаг можно пропустить (в таком случае в окружении уже установлена последняя версия pip);

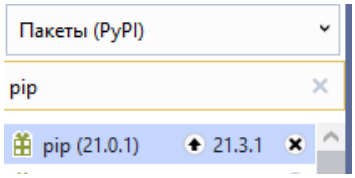

Рис. 7. Окно «Окружения Python» с возможностью обновления менеджера пакетов pip

   5.2. Установка пакета opencv-python.

   В окне «Окружения Python» наберите opencv-python и выберите пункт «Выполните команду: pip install opencv-python» и дождитесь окончания установки пакета. При необходимости установить конкретную версию библиотеки можно использовать команду pip

install opencv-python == XXX, где XXX — версия библиотеки (пример: pip install opencv-python == 4.5.5.62);

5.3. Установка пакетов библиотек numpy, tensorflow, matplotlib (производится аналогично п. 5.2).

6. Для проверки детектора объектов требуется переместить изображение объекта или объектов в каталог \object_detection и изменить переменную IMAGE_NAME в Object_detection_image.py, чтобы она соответствовала имени файла изображения.
7. Возможно задать толщину линии для выделения найденного объекта интереса и значение минимальной вероятности, при которой он будет выделен (параметры line_thickness и min_score_thresh).
8. Для запуска нейросетевой модели следует выбрать команду «Запуск с отладкой» (верхнее меню «Проект» среды Visual Studio). При успешном запуске будет запущен поиск объекта интереса на изображении и затем поверх этого изображения будут выведены результаты поиска (рис. 8).

Кроме обработки отдельных изображений, возможно применение нейросетевой модели для обработки большого набора файлов изображений и видеопоследовательностей. Пример программного кода для реализации, а также более подробный разбор работы с библиотекой TensorFlow можно взять в [17].

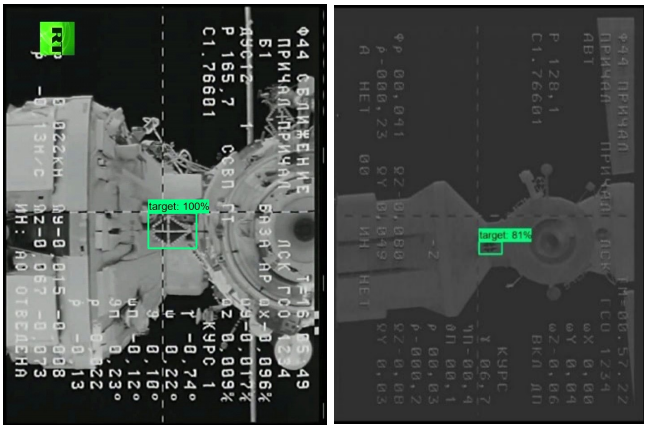

Рис. 8. Примеры обнаружения стыковочной мишени

Задание

1. Найти массив изображений или видеофрагмент с некоторым объектом интереса. При работе с видеофрагментом выполнить раскадровку в VirtualDub.
2. Подготовить базу размеченных изображений выбранного объекта интереса в программе LabelImg.
3. Выбрать искусственную нейронную сеть из «зоопарка моделей» TensorFlow, настроить её и провести обучение.
4. Проверить полученный результат на новых данных, имеющих выбранный объект интереса и не размещенных в созданных ранее выборках.

Список источников и литературы

1. Понс Ж., Форсайт Д. Компьютерное зрение. Современный подход // М.: Изд. д. Вильямс. – 2004. – Т. 465.
2. Стокман Д. Линда Шапиро. Компьютерное зрение // М.: Бином. Лаборатория знаний. – 2006. – С. 752.
3. Stockman A. Cone fundamentals and CIE standards // Current Opinion in Behavioral Sciences. – 2019. – Т. 30. – С. 87-93.
4. Soille P., Vincent L. M. Determining watersheds in digital pictures via flooding simulations // Visual Communications and Image Processing'90: Fifth in a Series. – SPIE, 1990. – Т. 1360. – С. 240-250.
5. Meyer F. Topographic distance and watershed lines // Signal processing. – 1994. – Т. 38. – №. 1. – С. 113-125.
6. Кэлер А., Брэдски Г. Изучаем OpenCV 3 / пер. с англ. А.А. Слинкина. — М.: ДМК Пресс, 2017. — 826 с.
7. Шолле Ф. Глубокое обучение на Python — СПб.: Питер, 2018. — 400 с.
8. TensorFlow 1 Detection Model Zoo // Google. — https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf1_detection_zoo.md
9. Culurciello E. Neural Network Architectures // Towards Data Science. — 2017. — https://towardsdatascience.com/neural-network-architectures-156e5bad51ba
10. Николенко С. Архитектуры сверточных сетей (2019) // СПбГУ, 16 ноября 2019 г. — https://logic.pdmi.ras.ru/~sergey/teaching/mlspsu18/28-cnn2.pdf
11. Anaconda (Python distribution). — 2020. — https://www.anaconda.com/
12. TensorFlow — Tested build configuratins. — 2023. — https://www.tensorflow.org/install/source#tested_build_configurations.
13. Google LLC. Protocol Buffers — a language-neutral, platform-neutral extensible mechanism for serializing structured data. — 2023. — https://protobuf.dev/
14. LabelImg — image annotation tool — 2023. — https://github.com/heartexlabs/labelImg
15. Трансляция стыковки ракеты-носителя «Союз-2.1а» с Международной космической станцией // RT на русском. — 29.10.2014. — https://www.youtube.com/watch?v=y-K-I1ijhf0
16. Huynh S. How to install LabelImg in Windows with Anaconda? — 2023. — https://medium.com/@sanghuynh_73086/how-to-install-labelimg-in-windows-with-anaconda-c659b27f0f

17. Решение задачи обнаружения объекта с помощью нейросетевых технологий / С.О. Власов [и др.] // Препринты ИПМ им. М.В.Келдыша. 2023. № 16. 27 с. https://doi.org/10.20948/prepr-2023-16

Подписано в печать 27.09.2024. Формат 60x84/16. Усл. печ. л. 3,8. Тираж 100 экз. Заказ М-7
ИПМ им. М.В. Келдыша РАН. 125047, Москва, Миусская пл., 4