Министерство образования Республики Беларусь

Учреждение образования
«Полоцкий государственный университет им. Евфросинии Полоцкой»

Р. П. Богуш, С. А. Игнатьева

ЦИФРОВАЯ ОБРАБОТКА ИЗОБРАЖЕНИЙ

Методические указания
по выполнению лабораторных работ для студентов специальностей
1-40 01 01 «Программное обеспечения информационных технологий»,
1-40 02 01 «Вычислительные машины, системы и сети»

Текстовое электронное издание

Новополоцк
Полоцкий государственный университет
им. Евфросинии Полоцкой
2022

Об издании – 1, 2

1 – дополнительный титульный экран – сведения об издании

УДК 004.032

Рекомендовано к изданию
методической комиссией факультета информационных технологий
в качестве методических указаний
(протокол № 5 от 30.05.2022 г.)

РЕЦЕНЗЕНТЫ:

канд. техн. наук, доц., заведующий кафедрой технологий программирования
Полоцкого государственного университета им. Евфросинии Полоцкой
В. М. ЧЕРТКОВ

2 – дополнительный титульный экран – производственно-технические сведения

Для создания текстового электронного издания «Цифровая обработка изображений. Методические указания по выполнению лабораторных работ для студентов специальностей 1-40 01 01 «Программное обеспечения информационных технологий», 1-40 02 01 «Вычислительные машины, системы и сети» Р. П. Богуша и С. А. Игнатьевой использованы текстовый процессор Microsoft Office Word и программа Adobe Acrobat XI Pro для создания и просмотра электронных публикаций в формате PDF.

Технические требования:
1 оптический диск.
Системные требования:
PC с процессором не ниже Core 2 Duo;
2 Gb RAM; свободное место на HDD 5 Mb;
Windows XP/7/8/8.1/10/11
привод CD-ROM/DVD-ROM;
мышь.

Редактор С. Е. Рясова

Подписано к использованию 09.09.2022.
Объем издания: 4 Мб. Заказ 527.

Свидетельство о государственной регистрации
издателя, изготовителя, распространителя печатных изданий
№ 1/305 от 22.04.2014.

211440, Ул. Блохина, 29,
г. Новополоцк,
Тел. 8 (0214) 59-95-41, 59-95-44
http://www.psu.by

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

По сравнению с другими органами чувств посредством зрения человек получает наибольший объем информации, поэтому зрительные образы играют самую важную роль в человеческом восприятии. Однако в отличие от людей, которые ограничены визуальным диапазоном электромагнитного спектра, средства формирования изображений охватывают намного больший спектр волн и позволяют получать созданные изображения, например, ультразвуковые изображения, радиолокационные изображения, на основе компьютерной графики и др.

Обработка изображений средствами вычислительной техники все шире используется в разных системах, которые используются на практике. В последнее десятилетие благодаря интенсивному развитию графических процессоров все шире стали применяться искусственные нейронные сети, а для обработки изображений – класс сверточных нейронных сетей, которые позволили существенно упростить решение прикладных задач инженерам.

Разработано и существует множество методов, алгоритмов и инструментов для обработки изображений. Очевидно, что в полном объеме освоить все существующие подходы в рамках одной дисциплины невозможно. Данные методические указания включают расширенный теоретический материал для понимания принципов обработки и базовых алгоритмов, а также современных подходов с использованием сверточных нейронных сетей. Рассматриваются существующие программные библиотеки обработки изображений на персональном компьютере и облачные сервисы для программирования, а также примеры кода программ, что предоставляет возможность студентам самостоятельно подготовиться к лабораторным работам. Кроме полученных теоретических знаний в предметной области при выполнении работ студенты усовершенствуют свои практические навыки программирования, что является важным не только для разработки сложных программных проектов по обработке изображений, но и при обработке других типов информации. Примеры кодов представлены на языке программирования Python, как наиболее широко используемом в настоящее время для обработки изображений средствами стандартных библиотек и фреймворков, однако задания могут быть выполнены на любом другом языке программирования. Также возможно использование пакета MatLab.

В настоящих методческих указаниях представлены четыре лабораторных работы. Первая из них посвящена изучению библиотек компьютерного

зрения OpenCV и dlib, которые широко используются, включают широкий спектр методов и алгоритмов и непрерывно развиваются. Вторая работа направлена на изучение подходов к решению одной из наиболее значимых задач по обнаружению объектов на изображении с использованием сверточных нейронных сетей. Третья работа позволит студентам получить навыки по обучению сверточных нейронных сетей для обнаружения объектов на изображениях. Четвертая работа направлена на решение прикладной задачи сопровождения объектов на видеопоследовательностях, что позволит студентам получить навыки, необходимые при реализации систем компьютерного зрения.

ЛАБОРАТОРНАЯ РАБОТА № 1
Обработка изображений классическими алгоритмами с использованием библиотек компьютерного зрения

Цель работы: получение навыков работы с библиотеками компьютерного зрения OpenCV и dlib для базовых задач обработки изображений.

Теоретические сведения

Обработка изображений, компьютерное зрение, распознавание образов и машинное обучения являются тесно связанными областями и, как правило, при решении сложных задач практически все они реализуются в прикладных системах.

Обработка изображений (в узком смысле) решает задачи преобразования изображений, поэтому исходными и результирующими данными этого этапа являются цифровые изображения. В прикладном плане – это построение некоторой системы (или программного комплекса), которая решает определенный класс задач от получения цифрового изображения до интерпретации его содержания и извлечения некоторой информации.

Компьютерное зрение (Computer (Machine) Vision) занимается анализом образов. Для некоторой сцены (аудитория, комната и др.) компьютер должен дать ее описание (есть ли в ней предметы, какая освещенность и т. д.). Таким образом, компьютерное «зрение» переводит изображение в описание.

Распознавание – это процесс отнесения изображения к некоторому определенному классу. В широком смысле допустимо трактовать распознавание как построение полной модели изображения, которую можно воспроизвести посредством машинной графики.

Изображения представляют собой специфический вид информации и их специфика еще не полностью изучена ни в исследованиях, посвященных изучению зрительного восприятия человека, ни в информатике. Одной из особенностей является избыточность практически любого изображения. Причем изображение одной и той же сцены реального мира несет разную информацию разным пользователям, что не позволяет значительно уменьшить объем данных, содержащихся, например, в космическом снимке, хранящемся в базе данных изображений.

Цифровым изображением называются данные, организованные в виде числового массива, воспроизводящего свойства некоторой реальной сцены или синтезированного человеком объекта (чертежа, карты) и тесно связанного со способом получения этих данных.

С формальной точки зрения цифровое изображение представляется в виде матрицы действительных чисел F с элементами f_{ij} (каждому элементу ставится в соответствие некоторое число – код яркости, обычно от 0 до 255), где *ij* – целочисленные координаты. Причем принято за начало координат принимать левый верхний угол изображения, где $i=0$, $j=0$.

С использованием введенных обозначений можно компактно записать полное цифровое изображение размерами $M \times N$ в форме матрицы F:

$$F = \begin{bmatrix} f_{00} & f_{01} & \cdots & f_{0,N-1} \\ f_{10} & f_{11} & \cdots & f_{1,N-1} \\ \cdots & \cdots & \cdots & \cdots \\ f_{M-1,0} & f_{M-1,1} & \cdots & f_{M-1,N-1} \end{bmatrix} = \left[f_{ij} \right].$$

Каждый элемент этой матрицы называется элементом изображения или пикселем (*pixel* от английского *picture element*). Каждый пиксель цветного изображения в системах компьютерного зрения описывается тремя уровнями яркости – красным (R), зеленым (G) и синим (B) в диапазоне от 0 до 255.

В качестве примеров систем, использующих информацию из изображений и видео, требующих применения методов обработки изображений, компьютерного зрения и распознавания образов можно привести системы управления процессами (промышленные роботы, автономные транспортные средства); системы видеонаблюдения с автоматизированным или автоматическим анализом видеоданных; системы организации информации (например, индексация баз данных изображений); системы моделирования объектов или окружающей среды (анализ медицинских изображений, топографическое моделирование); системы человеко-машинного взаимодействия; системы дополненной реальности и др.

Множество методов обработки изображений делится на методы обработки в частотной и пространственной областях. Термин *пространственная область* относится к плоскости изображения как таковой, и данная категория объединяет подходы, основанные на прямом манипулировании пикселями изображения. Способы обработки изображений в *частотной*

области (после применения дискретного преобразования Фурье) достаточно развиты, но требуют значительно больших вычислительных затрат и для решения практических задач применяются реже.

В общем виде процессы пространственной обработки изображений описываются выражением

$$g_{ij} = T[f_{ij}],$$

где f_{ij} – элементы входного изображения F;

g_{ij} – элементы обработанного изображения G;

T – оператор преобразования входного изображения в окрестности точки с координатами (ij).

Под окрестностью вокруг точки понимают квадратную или прямоугольную область изображения, которая центрирована в точке с координатами (ij). Простейшая форма оператора достигается в случае, когда окрестность имеет размеры 1×1. При этом значение g зависит только от значения f в точке (ij) и оператор T становится функцией градационного преобразования (функцией преобразования интенсивностей или функцией отображения). Примерами являются преобразование цветного изображения в полутоновое, бинаризация, получение негативного изображения, логарифмическое и степенное преобразования и др.).

Преобразование цветного изображения в полутоновое заключается в получении яркости каждой точки по формуле

$$Y = 0{,}3R + 0{,}59G + 0{,}11B,$$

где R, G, B – значение красного, зеленого и синего цветов в обрабатываемой точке, и последующем копировании на все три канала полученной величины.

Бинаризация – это преобразование полутонового изображения к одноцветному (монохромному или бинарному).

Пусть f_{ij} – полутоновое изображение, t – порог и b_0, b_1 – два бинарных значения (для бинарного черно-белого $b_0 = 0$, $b_1 = 255$). Результат порогового разделения – бинарное изображение g_{ij}, полученное следующим образом:

$$g_{ij} = \begin{cases} b_0, & \text{if } f_{ij} \le t; \\ b_1, & \text{if } f_{ij} > t. \end{cases}$$

Основной задачей является выбор значения t с помощью некоторого критерия. Это значение может выбираться как одинаковым для всего изображения, так и различным для различных его частей. Если значения объектов и фона режима достаточно однородны по всему изображению, то может использоваться одно пороговое значение для всего изображения. Использование единственного значения порога для всех пикселей изображения называется *глобальным пороговым разделением*. Методика *локального порогового разделения* предполагает определение порога для области изображения или пикселей. Такие алгоритмы называются адаптивными алгоритмами бинаризации, к ним можно отнести метод Оцу, Брэдли и др.

Логарифмическое преобразование может быть использовано для изменения яркости и определяется как

$$g_{ij} = c\log(1 + f_{ij}),$$

где c – константа, $f_{ij} \in 0, ..., L-1$.

Использование логарифма позволяет узкий диапазон малых значений яркости преобразовать в более широкий диапазон выходных значений. Для больших значений входного сигнала верно противоположное утверждение. Такой тип преобразования используется для растяжения диапазона значений темных пикселей на изображении с одновременным сжатием диапазона значений ярких пикселей.

Слабый контраст – один из наиболее распространенных дефектов фотографических и сканированных изображений, обусловленный ограниченностью диапазона воспроизводимых яркостей. Путем цифровой обработки контраст можно повысить, изменяя яркость каждого элемента изображения и увеличивая диапазон яркостей.

Традиционно для цифрового представления каждого отсчета изображения в компьютере отводится 1 байт запоминающего устройства. Следовательно, входной или выходной сигналы могут принимать одно из 256 значений в диапазоне $0-255$. Предположим, что минимальная и максимальная яркости исходного изображения равны $f_{\min}$ и $f_{\max}$ соответственно. Если эти параметры или один из них существенно отличаются от граничных значений яркостного диапазона, то изображение выглядит как неконтрастное.

В реальных изображениях обычно существует перекос в сторону малых уровней и яркость большинства элементов изображения ниже средней. На темных участках подобных изображений детали часто оказываются

неразличимыми. Одним из методов улучшения таких изображений является видоизменение гистограммы. Этот метод предусматривает преобразование яркостей исходного изображения с тем, чтобы гистограмма распределения яркостей обработанного изображения приняла желаемую форму.

Одним из методов улучшения контраста является линейная растяжка гистограммы, когда уровням исходного изображения, лежащим в интервале $[f_{\min}, f_{\max}]$, присваиваются новые значения с тем, чтобы охватить весь возможный интервал изменения яркости, в данном случае [0, 255]. При линейном контрастировании используется линейное поэлементное преобразование вида

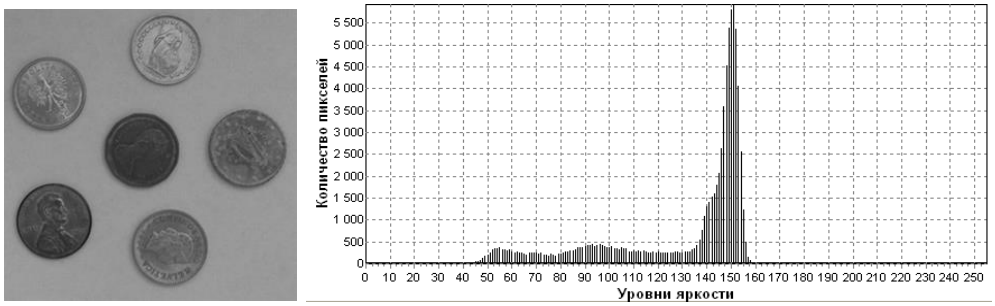
$$g_{ij} = \frac{f_{ij} - f_{\min}}{f_{\max} - f_{\min}}(g_{\max} - g_{\min}) + g_{\min}.$$

Такой алгоритм заложен во многих программных продуктах, реализующих функцию «Autocontrast».

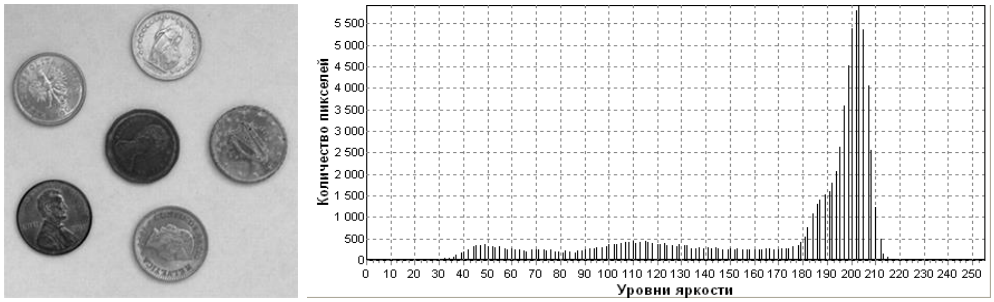
При нормализации гистограммы на весь максимальный интервал уровней яркости [0, 255] растягивается не вся гистограмма, лежащая в пределах $[f_{\min}, f_{\max}]$, а ее наиболее интенсивный участок $[f'_{\min}, f'_{\max}]$, из рассмотрения исключаются малоинформативные начальный и конечный участки. Например, для рассматриваемого изображения на основе анализа гистограммы исходного изображения примем $f'_{\min} = 45$ и $f'_{\max} = 160$.

Целью выравнивания гистограммы (линеаризации, эквализации) является такое преобразование, чтобы в идеале все уровни яркости приобрели бы одинаковую частоту, а гистограмма яркостей отвечала бы равномерному закону распределения. Кроме этого, существует метод видоизменения гистограмм, который обеспечивает экспоненциальную или гиперболическую форму распределения яркостей улучшенного изображения. Применяются также «локальные» методы улучшения на основе обработки части изображения (рисунок 1.1).

Другой подход заключается в применении методов, в результате которых на выходное значение пикселя оказывают влияние текущий пиксель и его соседи, т. е. увеличивается окрестность вокруг пикселя и это приводит к значительно большей гибкости при обработке изображений. При этом используется маска (апертура, окно, ядро, окрестность), которая представляет собой двумерный массив заданного размера, чаще всего размером 3×3. Такие методы называют *обработкой* или *фильтрацией на основе маски*.



а *б*



в *г*

а – исходное изображение; *б* – гистограмма исходного изображения;
в – результат улучшения контраста; *г* – гистограмма после линейной растяжки

Рисунок 1.1. – Результат улучшения контраста

В качестве маски используется множество весовых коэффициентов, заданных во всех точках окрестности, обычно симметрично окружающих рабочую точку кадра. Распространенным видом окрестности, часто применяемым на практике, является квадрат 3×3 с рабочим элементом в центре (рисунок 1.2). На рисунке 1.3 представлены элементы области и изображения под маской. При цифровой обработке изображений используется декартова система координат с началом в левом верхнем углу кадра и с положительными направлениями из этой точки вниз и вправо.

$w_{(-1,-1)}$	$w_{(-1,0)}$	$w_{(-1,1)}$
$w_{(0,-1)}$	$w_{(0,0)}$	$w_{(0,1)}$
$w_{(1,-1)}$	$w_{(1,0)}$	$w_{(1,1)}$

Рисунок 1.2. – Коэффициенты маски
с относительными значениями координат

$f_{(i-1,j-1)}$	$f_{(i-1,j)}$	$f_{(i-1,j+1)}$
$f_{(i,j-1)}$	$f_{(i,j)}$	$f_{(i,j+1)}$
$f_{(i+1,j-1)}$	$f_{(i+1,j)}$	$f_{(i+1,j+1)}$

Рисунок 1.3. – Элементы области
изображения под маской

Сущность процедуры фильтрации заключается в смещении маски фильтра по изображению с шагом один пиксель слева направо, сверху вниз.

При этом отклик задается суммой произведений коэффициентов фильтра на соответствующие значения пикселей в области, покрытой маской фильтра. Для маски размером 3×3 отклик r определяется следующим образом:

$$r = w_{(-1,-1)}f_{(i-1,j-1)} + w_{(-1,0)}f_{(i-1,j)} + w_{(-1,1)}f_{(i-1,j+1)} +$$
$$+w_{(0,-1)}f_{(i,j-1)} + w_{(0,0)}f_{(i,j)} + w_{(0,1)}f_{(i,j+1)} + w_{(1,-1)}f_{(i+1,j-1)} +$$
$$+w_{(1,0)}f_{(i+1,j)} + w_{(1,1)}f_{(i+1,j+1)}.$$

В общем случае фильтрация изображения f_{ij} размером $M \times N$ с использованием маски размером $m \times n$ описывается выражением

$$g_{ij} = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w_{st} f_{i+s,j+t},$$

где $a = (m-1)/2,\ b = (n-1)/2$.

Таким образом, можно использовать маски размером не только 3×3, но и более, например, 5×5, 7×7 и т. п.

Если центр маски находится на границе изображения, то ее края выходят за его пределы. Для обработки краев изображений используются следующие приемы: дополнение средним значением, доопределение повторением крайних отсчетов последовательности, экстраполяция более высокого порядка.

Усредняющая или сглаживающая пространственная фильтрация может служить эффективным средством подавления высокочастотных шумов. Сглаживающие маски, часто называемые шумоподавляющими, имеют вид:

$$H1 = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}, \quad H2 = \frac{1}{10}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 1 \end{bmatrix}, \quad H3 = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}.$$

Повышение резкости изображений может быть достигнуто путем численного дифференцирования. С принципиальной точки зрения величина отклика оператора производной в точке изображения пропорциональна степени разрыва изображения в данной точке. Таким образом, дифференцирование изображения позволяет усилить перепады яркости и другие разрывы (шумы, помехи) и не подчеркивать области с медленными изменениями яркостей.

Для вычисления двумерной второй производной и наложения результата на изображение (высокочастотная фильтрация) используются маски:

$$H4 = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} \quad H5 = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix} \quad H6 = \begin{bmatrix} 1 & -2 & 1 \\ -2 & 5 & -2 \\ 1 & -2 & 1 \end{bmatrix}.$$

Следует отметить, что для сохранения постоянного уровня яркости изображения коэффициент передачи маски, определяемый как сумма всех элементов маски, должен быть равен единице. Это справедливо для представленных масок H1–H6.

Для выделения контуров изображений можно использовать маски H1–H6 с уменьшенным на единицу центральным элементом.

В настоящее время рассмотренные алгоритмы и множество других, применяемых для решения задач обработки изображений и компьютерного зрения, реализованы в различных библиотеках компьютерного зрения, среди которых можно выделить OpenCV [1], dlib [2], PCL, VXL, AForge.Net, LTI и др.

Библиотека компьютерного зрения OpenCV

OpenCV (Open Source Computer Vision, https://opencv.org/) [1] – библиотека компьютерного зрения с открытым исходным кодом и широким спектром возможностей. Функциональность OpenCV доступна на языках C, C++, Python, Java, Ruby. Поддерживаются основные операционные системы: MS Windows, Linux, Mac, Android, iOS.

Библиотека включает большое количество оптимизированных классических и современных алгоритмов обработки изображений, распознавания образов, компьютерного зрения и машинного обучения, которые могут быть использованы для обнаружения и распознавания лиц, идентификации разных классов объектов, анализа действий человека в видео, отслеживания движений камеры, сопровождения движущихся объектов на видео и других.

OpenCV имеет модульную структуру, а это значит, что пакет включает в себя несколько общих или статических библиотек. Библиотека непрерывно обновляется, в версии OpenCV 4.5.5 доступны:

1) модуль *core* определяет основные структуры данных, используемые всеми остальными модулями, такие как:

- операции с массивами;
- сохранение XML/YAML/JSON;

– кластеризация;
– утилиты, системные функции и макросы;
– совместимость с OpenGL;
– алгоритмы оптимизации;
– параллельная обработка;
– аппаратное ускорение;

2) модуль *imgproc* предназначен для обработки изображений и включает:
– фильтрацию изображений;
– геометрические преобразования;
– функции рисования;
– функции преобразования цветного пространства;
– анализ гистограмм;
– обнаружение признаков;
– сегментацию;
– обнаружение объектов;

3) модуль *imgcodecs* используется для чтения и записи изображений;

4) модуль *videoio* предназначен для чтения и записи видео или последовательности изображений;

5) модуль *highgui* – графический интерфейс высокого уровня. Позволяет создавать и управлять окнами, которые могут отображать изображения и «запоминать» их содержимое;

6) модуль *video*, предназначенный для анализа видео, включающий алгоритмы оценки движения, вычитания фона и отслеживания объектов;

7) модуль *calib3d* – калибровка камеры и 3D-реконструкция. Включает основные алгоритмы геометрии с несколькими видами, калибровку одиночной и стереокамеры, оценку позы объекта, алгоритмы стереосоответствия и элементы 3D-реконструкции;

8) модуль *features2d* включает детекторы признаков, дескрипторы и средства сопоставления дескрипторов:
– обнаружение и описание признаков;
– сопоставление дескрипторов;
– рисование ключевых точек и совпадений;
– классификация объектов;

9) модуль *objdetect*, предназначен для обнаружения объектов и экземпляров предопределенных классов (например: лица, глаза, люди, автомобили и т. д.);

10) модуль *dnn*, предназначен для работы с глубокими нейронными сетями:

– инструментарий для загрузки моделей сетей из разных фреймворков;

– реализации готовых слоев нейронных сетей;

– утилиты, для создания новых слоев;

11) модуль *ml* – модуль для машинного обучения. Представляет собой набор классов и функций для статической классификации, регрессии и кластеризации данных;

12) модуль *flann* – модуль, содержащий коллекцию алгоритмов, оптимизированных для быстрого поиска ближайших соседей в больших наборах данных;

13) модуль *photo* – модуль для обработки фотографий:

– шумоподавление;

– HDR-визуализация (High Dynamic Range Rendering);

– обесцвечивание изображений с сохранением контраста;

– бесшовное клонирование, которое позволяет скопировать объект с одного изображения на другое таким образом, чтобы комбинация выглядела бесшовной и естественной;

14) модуль *stitching* используется для сшивания изображений:

– поиск признаков и сопоставление изображений;

– оценка вращения;

– автокалибровка (пытается оценить фокусные расстояния по заданной гомографии исходя из предположения, что камера вращается только вокруг своего центра, и может использоваться для построения панорамных мозаик);

– деформация изображений;

– оценка швов склеенных изображений;

– компенсация экспозиции;

15) модуль *gapi* – модуль, предназначенный для того, чтобы сделать обычную обработку изображений быстрой и переносимой путем внедрения новой графовой модели, суть которой заключается в построении графа на основе используемых операций к объектам данных. Графы представляются неявной структурой, для описания которой используются не «узлы» и «ребра», а операции и объекты данных. Объекты данных при этом

содержат не фактические значения, а описывают зависимости. Модуль выступает в качестве фреймворка.

Opencv-contrib – отдельный репозиторий, содержащий новые расширяющие функциональность модули, которые еще не включены в основную часть. Это связано с тем, что новые модули часто не имеют стабильного интерфейса и плохо протестированы, поэтому их и не включают в официальный дистрибутив OpenCV. *Opencv-contrib 4.5.5* содержит ряд модулей, среди них можно выделить основные:

– *alphamat* (альфа-матирование) предназначен для отделения объекта на переднем плане от фона обеспечивая нерезкие границы. Его можно использовать для дальнейших операций;

– *aruco* предоставляет возможности обнаружения двоичных квадратных реперных маркеров и инструменты для их использования при оценке позы человека и калибровке камеры;

– *barcode* служит для обнаружения и декодирования штрих-кодов;

– *bgsegm* применяется для вычитания фона при обработке видео;

– *bioinspired* позволяет обрабатывать как изображения, вызывающие оптические иллюзии, так и обычные;

– *ccalib* предназначен для калибровки одной камеры, стереопары и многокамерных систем. Позволяет удалять большие искажения на изображениях, а также реконструировать 3D-сцены на основе двух стереоизображений;

– *cvv* может быть использован для отладки приложений с использованием визуального пользовательского интерфейса;

– *dnn_objdetect* служит для обнаружения объектов с использованием глубоких сверточных нейронных сетей;

– *dnn_superres* позволяет обрабатывать изображения сверхвысокого разрешения, также включает функции масштабирования фото и видео;

– *face* предназначен для обнаружения и распознавания лиц на изображениях;

– *fuzzy*. Модуль реализует алгоритмы обработки изображений, основанные на нечеткой математике;

– *mcc* применяется для цветокоррекции с использованием Color correction Model;

– *tracking* используется для сопровождения различных объектов на видео;

– *ximgproc* предназначен для фильтрации несоответствий на стереоизображениях, может быть использован для быстрого обнаружения границ с использованием алгоритма Structured Forest;

– *xphoto* применяется для восстановления поврежденных или отсутствующих частей изображения [3].

Библиотека компьютерного зрения dlib

Библиотека dlib включает реализацию множества традиционных математических функций, операций линейной алгебры, а также алгоритмов сжатия данных, классической обработки изображений и алгоритмов на основе машинного обучения, реализованных в основном на языке C++, однако ряд инструментов dlib возможно использовать и на Python.

Лицензирование dlib с открытым исходным кодом позволяет использовать ее бесплатно [2]. Следует выделить следующие основные инструменты в dlib версии 19.23:

1) *для обработки изображений* – функции для работы с различными типами пикселей; чтение и запись различных форматов изображений, таких как BMP, PNG, JPEG, GIF и формат dlib файла DNG; автоматическое преобразование цветового пространства; алгоритмы поиска объектов на изображениях; традиционные операции с изображениями, такие как поиск краев и морфологические операции и др.; алгоритмы извлечения признаков SURF (Speeded Up Robust Features, Ускоренные надежные признаки), HOG (Histogram of Oriented Gradients, Гистограмма направленных градиентов) и FHOG (Felzenszwalb Histogram of Oriented Gradients, Гистограмма направленных градиентов Фельзеншвальба);

2) *для машинного обучения* – алгоритмы бинарной и многоклассовой классификации, регрессионного и структурного анализа, кластеризации, глубокого обучения, обучения без учителя, обучения с подкреплением;

3) *для метапрограммирования* – предоставляются функциональные возможности для отладки, выполнения различных операций с шаблонами. Включает большое число макрофункций;

4) *для вычислений:*

– быстрые алгоритмы обработки матричных объектов;

– многочисленные функции линейной алгебры и математические операции;

5) *для оптимизации* dlib включает нелинейные оптимизаторы общего назначения;

6) *для работы с сетью* в dlib имеется набор объектов для создания многопоточных программ, для обеспечения соединения между сетевыми приложениями, для создания сервера и т. д.

7) *для работы с текстом* имеются инструменты для анализа и различных манипуляций: преобразования кодировок, работы со строками, анализа параметров аргументов командной строки и др.

Задания к лабораторной работе

При необходимости установить OpenCV и dlib. В приложении 1.1 показана последовательность установки для OpenCV4.5.5 и dlib 19.23.

1. *Загрузить полноцветное изображение с использованием OpenCV и выполнить преобразования согласно своему варианту. Исследовать влияние параметров функций, необходимых для выполнения преобразования. Описание функций OpenCV4.5.5, необходимых для выполнения данной работы представлено в приложении 1.2. Подписать на изображении выполненное преобразование и свою фамилию.*

Реализовать следующие операции обработки изображений:

Вариант 1: Кадрирование, перевод изображения в полутоновое (в градациях серого).

Вариант 2: Поворот, перевод изображения в бинарное.

Вариант 3: Масштабирование, размытие изображение.

Вариант 4: Кадрирование, увеличение четкости.

Вариант 5: Поворот, линейное улучшение контраста.

Вариант 6: Масштабирование, улучшение контраста на основе эквализации гистограммы (clahe).

Вариант 7: Кадрирование, выделение контуров.

Вариант 8: Поворот, эффект рельефа.

Вариант 9: Масштабирование, увеличение четкости.

Вариант 10: Кадрирование, линейное улучшение контраста.

Вариант 11: Поворот, улучшение контраста на основе эквализации гистограммы (clahe).

Вариант 12: Масштабирование, размытие изображения.

Вариант 13: Кадрирование, эффект рельефа.

Вариант 14: Поворот, выделение контуров.

Вариант 15: Масштабирование, адаптивная бинаризация.

2. *Загрузить полноцветное изображение лица человека с использованием OpenCV. Применить функцию выделения особых точек лица библиотеки dlib для изображений одного и тоже человека, полученных в различных условиях сьемки. Сделать выводы о корректности нахождения точек лица на изображениях.*

Содержание отчета

1. Титульный лист.
2. Цель лабораторной работы.
3. Ход работы с пояснениями алгоритмов из первого задания, примерами обработки по заданиям и их анализом.
4. Результаты выполнения второго пункта задания.
5. Листинг кода.
6. Выводы о проделанной работе.

Контрольные вопросы

1. Что понимают под цифровым изображением?
2. Сформулируйте различия между обработкой изображений, компьютерной графикой и распознаванием образов.
3. Какова основная особенность градационных преобразований?
4. Сформулируйте процедуру преобразования цветного изображения в полутоновое (полутонового в бинарное).
5. В чем заключается сущность масочной обработки изображений и ее отличие от градационных преобразований?
6. После применения какого фильтра на изображении будем наблюдать наибольшее размывание контуров и почему?

$$M_1 = \frac{1}{14}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 6 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad M_2 = \frac{1}{12}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 4 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad M_3 = \frac{1}{10}\begin{bmatrix} 0 & 1 & 0 \\ 1 & 6 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

7. Приведите примеры масок для ВЧ-фильтрации.
8. В каких областях используется цифровая обработка изображений? Приведите примеры.
9. Какие утилиты используются для установки OpenCV и dlib?
10. Перечислите основные модули, входящие в состав OpenCV.
11. Какими основными функциональными возможностями обладает библиотека dlib?

Приложение 1.1
Установка библиотек компьютерного зрения в ОС Windows

Для установки OpenCV в Windows необходимо с официального сайта https://opencv.org/releases/ скачать версию OpenCV для Windows (рисунок 1.4).

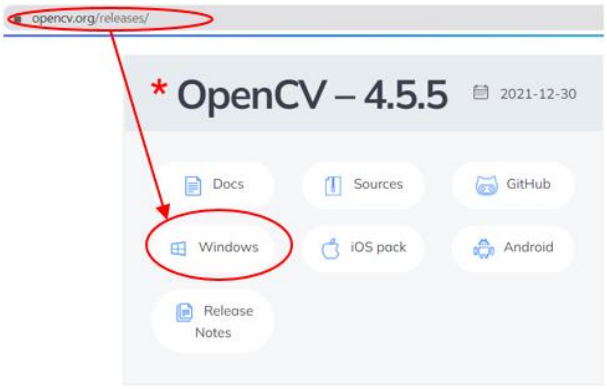


Рисунок 1.4. – Расположение OpenCV 4.5.5 на официальном ресурсе

Распаковывать архив в C:\opencv_4.5.5\build и установить значение системной переменной, как показано на рисунке 1.5.

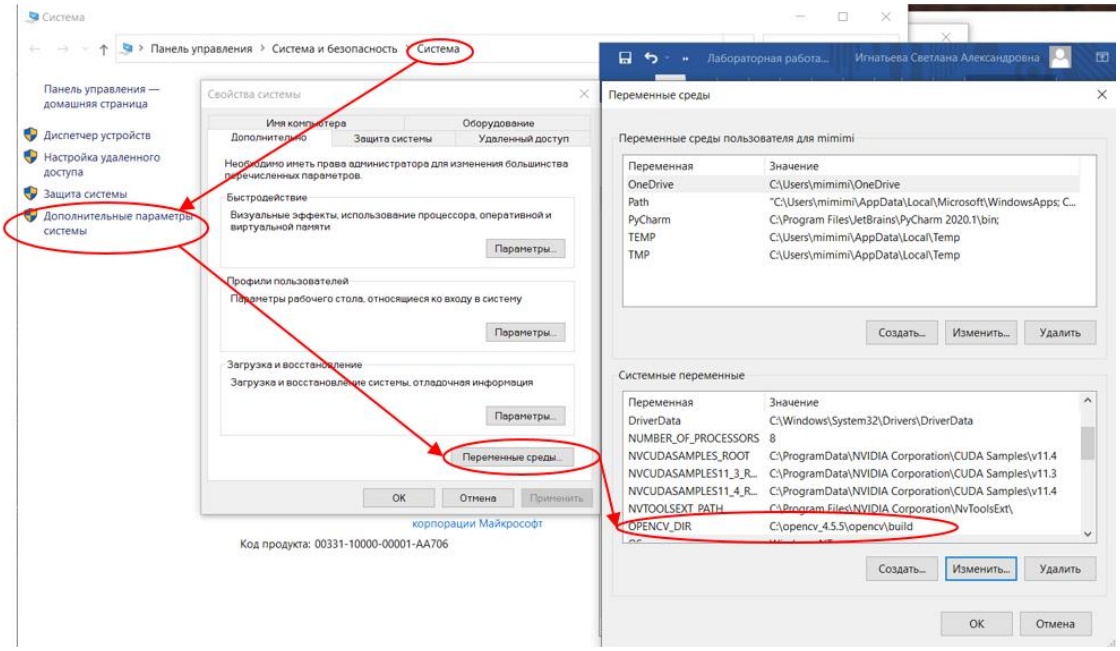


Рисунок 1.5. – Установка OpenCV 4.5.5 на персональный компьютер

Установка OpenCV в среду Python:

```
pip install opencv-python
```

Установка OpenCV-contrib в среду Python:

```
pip install opencv-contrib-python
```

Для установки библиотеки dlib необходимо использовать кроссплатформенную утилиту CMake [4], которая обладает возможностями автоматизации сборки программного обеспечения из исходного кода. Для этого необходимо загрузить установочный файл CMake, доступный по ссылке https://cmake.org/download/ (рисунок 1.6).

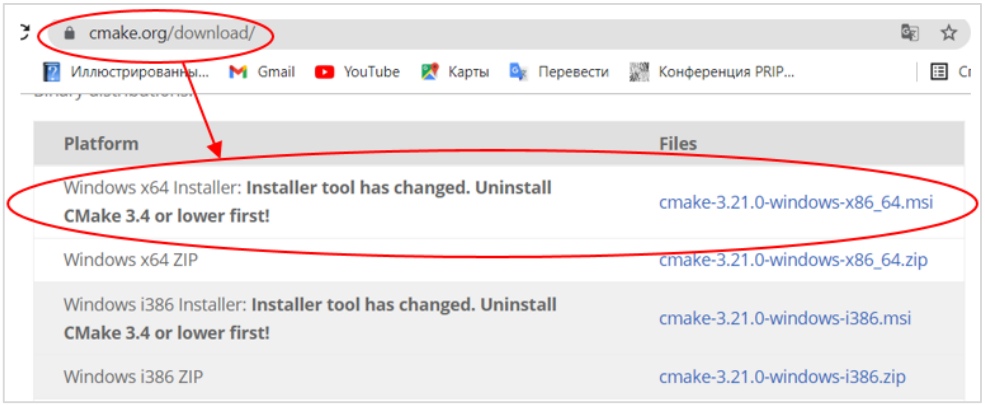


Рисунок 1.6. – Доступ к CMake на веб-ресурсе

При установке рекомендуется выбрать опцию «Add CMake to the system PATH for all users» («Добавить пути в переменные среды для всех пользователей»).

Dlib разработана на основе языка программирования C, следовательно, необходим компилятор. Автономный компилятор MS Visual Studio можно скачать по ссылке: https://visualstudio.microsoft.com/ru/visual-cpp-build-tools/. После завершения установки необходимо установить дополнительные пакеты для программирования C/C++, т. е. Packages CMake tools для Windows.

Чтобы установить dlib в среду Python требуется инсталлировать cmake: `pip install cmake`, а затем dlib: `pip install dlib`.

Приложение 1.2
Функции библиотеки OpenCV

`cv2.imread ()` – чтение изображений в Python с помощью OpenCV. Возвращает 2D- или 3D-матрицу в зависимости от количества цветных каналов, присутствующих на изображении.

Синтаксис: `cv2.imread('Path/filename', flag)`.

`flag` является необязательным и может принимать одно из следующих значений: `cv2.imread_color` (считывает изображение с цветами RGB, но без канала прозрачности. Это значение по умолчанию для флага, когда значение не указано в качестве второго аргумента для `cv2.imread()`);

cv2.imread_grayscale (читает изображение как серое. Если исходное изображение является цветным, значение серого для каждого пикселя вычисляется путем получения среднего значения цветовых каналов и считывается в массив); cv2.imread_unchanged (читает изображение как есть из источника).

cv2.imwrite () – запись изображения в Python с помощью OpenCV.

Синтаксис: cv2.imwrite ('Path/filename', image)

cv.imshow() – вывод изображения на экран.

Синтаксис: cv.imshow("filename", image)

cv.waitKey() – задержка отображения окна в течении заданных миллисекунд, или до нажатия какой-либо клавиши на клавиатуре. Если в качестве параметра принято число, то время ожидания до закрытия окна ограничено указанным числом в миллисекундах. Если указан 0 или параметр не указан, окно закроется по нажатию любой клавиши клавиатуры.

cv2.resize – изменение размера изображения.

Синтаксис:

cv2.resize(image, new_size, interpolation=cv2. INTER_XXX)

interpolation=cv2.INTER_XXX – алгоритм интерполяции, может принимать значения: cv2.INTER_NEAREST (интерполяция ближайшего соседа); cv2.INTER_LINEAR (билинейная интерполяция); cv2.INTER_CUBIC (бикубическая интерполяция); cv2.INTER_AREA (пересчет с использованием отношения площадей пикселей); cv2.INTER_LANCZOS4 (интерполяция Lanczoc по окресности 8×8).

cv2.flip() – переворачивает 2D-массив вокруг вертикальной, горизонтальной или обеих осей. Может принимать значения: 1 – переворот вокруг оси *y*, 0 – переворот вокруг оси *z*, –1 – переворот по обеим осям.

cv2.split() – разделить изображение по каналам.

Синтаксис:

```
b, g, r = cv2.split(image)
```

cv2.merge() – объединить каналы изображения.

Синтаксис:

```
cv2.split([b, g, r])
```

cv2.blur() – размытие изображения на основе свертки с ядром $n \times n$. Пиксель в центре матрицы устанавливается равным среднему значению всех остальных пикселей. Чем больше ядро свертки (т. е. *n*), тем больше размытие.

Синтаксис:

```
cv2.blur(image, (n, n))
```

cv2.GaussianBlur() – размытие по Гауссу. Похоже на предыдущее размытие, однако вместо простого среднего используется взвешенное, т. е.

чем ближе пиксели к центральному, тем большее влияние они оказывают на среднее значение.

Синтаксис:

`cv2.GaussianBlur(image, (n, n), s)`, где *n* – ядро свертки, *s* – стандартное отклонение ядра Гаусса. Установив значение 0, оно будет вычисляться автоматически.

`cv2.putText()` – наложение текста на изображение.

Синтаксис:

`cv2.putText(image, text, org, font, fontScale, color[, thickness[, lineType]])`, где `image` – изображение; `text` – строка текста; `org` – координаты нижнего левого угла текстовой строки *x* и *y*; `font` – шрифт, например `FONT_HERSHEY_SIMPLEX`, `FONT_HERSEY_PLAIN` и т. д.; `fontScale` – масштабный коэффициент шрифта, который умножается на базовый размер; `color` – цвет строки, `thickness` – толщина; `lineType` – необязательный параметр, который определяет тип линии.

`cv2.line()` – начертить линию на изображении.

Синтаксис:

`cv2.line(image, org, color[, thickness[, lineType[, shift]]])`, где `image` – изображение; `text` – строка текста; `org` – координаты; `color` – цвет; `thickness` – толщина; `lineType` – необязательный параметр, который определяет тип линии; `shift` – цвет.

`cv2.rectangle()` начертить прямоугольник на изображении.

Синтаксис: `cv2.line(image, org1, org2 color[, thickness[, lineType[, shift]]])`, где `image` – изображение; `text` – строка текста; `org1` – координаты верхнего левого угла; `org2` – координаты нижнего правого угла; `color` – цвет; `thickness` – толщина; `lineType` – необязательный параметр, который определяет тип линии; `shift` – цвет.

`cv2.cvtColor()` – преобразование из одного цветового пространства в другое.

Синтаксис: `cv2.cvtColor(image, code[, outImg[, CoutImg]])`, где `image` – изображение, `code` – код цветового пространства; `outImg` – необязательный параметр, выходное изображение того же размера и глубины, что и исходное; `CoutImg` – количество каналов в целевом изображении. Если равен 0, то количество каналов определяется автоматически, на основе исходного изображения. Является необязательным параметром.

`cv2.threshold()` – функция бинаризации. Для каждого пикселя применяется одно и то же пороговое значение. Если значение пикселя меньше

порогового значения, оно устанавливается равным 0, в противном случае оно устанавливается на максимальное значение.

Синтаксис: cv2. threshold (image, thr, max, TypeThr), где image – исходное изображение, которое должно быть изображением в градациях серого; thr – это пороговое значение, которое используется для классификации значений пикселей; max – это максимальное значение, которое присваивается значениям пикселей, превышающим пороговое значение. OpenCV предоставляет различные типы пороговых значений, которые задаются четвертым параметром функции: cv2.THRESH_BINARY, cv2.THRESH_BINARY_INV, cv2.THRESH_TRUNC, cv2.THRESH_TOZERO, cv2.THRESH_TOZERO_INV.

cv2.adaptiveThreshold() – функция адаптивной бинаризации, устанавливает порог для пикселя на основе небольшой области вокруг него. Таким образом, для разных областей изображения используется разный порог.

Синтаксис: cv2.adaptiveThreshold(image, maxVal, adaptiveMethod, TypeThr, blockSize, const), где image – исходное одноканальное изображение; maxVal – максимальное значение, которое может быть присвоено; adaptiveMethod – адаптивный метод, который определяет как рассчитывается пороговое значение. Может принимать значение cv2.ADAPTIVE_THRESH_MEAN_C(), cv2.ADAPTIVE_THRESH_GAUSSIAN_C().

cv2.findContours() – функция выделения контуров на изображении.

Синтаксис: cv2.findContours(image, mode, method), где image – исходное изображение; mode – один из четырех режимов группировки найденных контуров: CV_RETR_LIST – выдаёт все контуры без группировки; CV_RETR_EXTERNAL – выдаёт только крайние внешние контуры; CV_RETR_CCOMP – группирует контуры в двухуровневую иерархию. На верхнем уровне – внешние контуры объекта. На втором уровне – контуры отверстий, если таковые имеются. Все остальные контуры попадают на верхний уровень; CV_RETR_TREE – группирует контуры в многоуровневую иерархию; method – один из трёх методов упаковки контуров: CV_CHAIN_APPROX_NONE – упаковка отсутствует и все контуры хранятся в виде отрезков, состоящих из двух пикселей; CV_CHAIN_APPROX_SIMPLE – склеивает все горизонтальные, вертикальные и диагональные контуры; CV_CHAIN_APPROX_TC89_L1, CV_CHAIN_APPROX_TC89_KCOS – применяет к контурам метод упаковки (аппроксимации) Teh-Chin.

`cv2.filter2D()` – может применяться для сглаживания или увеличения резкости изображений в зависимости от ядра свертки и выполнять другие преобразования изображений.

Синтаксис: `cv2.Filter2D(src, ddepth, kernel)`, где `src` – исходное изображение; `ddepth` – глубина, значение -1 означает, что результирующее изображение будет иметь ту же глубину, что и исходное; `kernel` – матрица фильтра, применяемая к изображению.

Для повышения резкости может использоваться одна из масок Н3–Н6, для получения эффекта рельефа – ядро

$$kernel = \begin{bmatrix} -2 & -1 & 0 \\ -1 & 1 & 1 \\ 0 & 1 & 2 \end{bmatrix}.$$

`cv2.convertTo()` – выполняет преобразование изображения для изменения контраста и яркости $new\_img = \alpha \cdot img + \beta$, где new_img – преобразованное изображение, α и β – параметры, для управления яркостью и контрастом.

Синтаксис: `img.convertTo(new_img, ddepth, alfa, beta)`, где `img` – исходное изображение; `new_img` – изображение с преобразованием; `ddepth` – глубина; `alfa` и `beta` – параметры для управления яркостью и контрастом.

`cv2.createCLAHE()` – алгоритм адаптивной коррекции гистограмм с ограниченным контрастом. Изображение разделяется на небольшие блоки, и гистограмма выравнивается для каждого из них, после чего для объединения каждого блока применяется билинейная интерполяция для устранения искусственных границ. Применяется для улучшения контрастности изображений. Имеет два параметра: `clipLimit` (устанавливает порог для ограничения контраста, по умолчанию 40) и `tileGridSize` (устанавливает количество плиток в строке и в столбце, по умолчанию 8×8).

Функции библиотеки dlib

Для выполнения лабораторной работы необходимы следующие инструменты из библиотеки dlib:

– `get_frontal_face_detector()` – функция для детектирования лиц. Для обнаружения лиц dlib использует методы HOG (Histogram of Oriented Gradients) и SVM (Support Vector Mashines) [5].

– `shape_predictor()` – представляет собой инструмент, который выбирает область изображения, содержащую некоторый объект, и выводит набор местоположений точек, определяющих положение объекта. В качестве аргумента необходимо указать используемую обученную модель shape_predictor_68_face_landmarks.dat.

Пример 68-точечной модели определения особых точек лица в dlib показан на рисунке 1.7.

Рисунок 1.7. – Расположение ключевых точек лица в модели dlib

Примеры реализации базовых процедур обработки изображений на языке программирования Python с использованием OpenCV

Преобразование цветного изображения в градации серого

```
import os, cv2 as cv
image = cv.imread("./img2.jpg")
cv.imshow("original", image)
gray_image = cv.cvtColor(image, cv.COLOR_BGR2GRAY)
cv.imshow("gray_image", gray_image)
cv.waitKey()
```

Добавление тестовой надписи на изображение

```
import os, cv2 as cv
image = cv.imread("./img2.jpg")
output = image.copy()
cv.putText(output, "Image with text", (50, 50),
  cv.FONT_HERSHEY_SIMPLEX, 2, (250, 80, 50), 3)
cv.imshow("Изображение с текстом", output)
cv.waitKey()
```

Пример фрагмента кода python для детектирования особых точек лица с использованием библиотеки dlib

Выбор детекторов:

```
# детектор лица на изображении
detector = dlib.get_frontal_face_detector()
# детектор контура лица (68 точек)
predictor = dlib.shape_predictor('shape_predic'
       'tor_68_face_landmarks.dat')
for file in face_images:
    image = cv2.imread(file)
```

Поиск и отображение на изображение особых точек лица:

```
rects = detector(gray, 1)
for rect in rects:
# нахождение 68 точек для каждого найденного лица
    shape = predictor(gray, rect)
for i in range(0, 68):
    # отображение точек
    cv2.circle(image, (shape.part(i).x, shape.part(i).y),
         1, (0, 0, 255), -1)
cv2.imwrite(f'./output/{os.path.split(file)[-1]}', image)
```

ЛАБОРАТОРНАЯ РАБОТА № 2
Обнаружение объектов заданных классов на изображениях с использованием сверточных нейронных сетей

Цель работы: изучить принципы и инструменты программной реализации обнаружения объектов заданных классов на изображениях с использованием существующих фреймворков, реализующих сверточные нейронные сети.

Теоретические сведения

Обнаружение объекта – это определение местоположения заданного объекта на изображении, при этом его размеры меньше размеров изображения, а количество объектов на изображении заведомо неизвестно.

Объекты на изображении задаются значениями некоторых признаков, которые представляются в виде вектора конечной размерности. Наборы признаков, т. е. длина векторов, являются одинаковыми для всех объектов. Совокупность признаков объекта определяется правилами его описания (например, описание полутонового изображения уровнями яркостей всех пикселей изображения или описание этого изображения его яркостной гистограммой). Соответственно, признаки могут выражаться в различными числовыми значениями.

На всём множестве объектов, которые могут присутствовать на изображениях, существует разбиение на подмножества, т. е. классы объектов. Объекты в пределах одного класса описываются наборами признаков, по которым они более схожи внутри этого класса и значительно отличаются от объектов других классов. Кроме этого, они имеют одно имя класса, по которому неразличимы. Например, класс «люди» и класс «автомобили».

Класс задается указанием некоторых признаков, присущих всем его членам. Соответственно, основными задачами являются выбор наиболее эффективного набора признаков для объектов и метода сравнения этих признаков для принятия решения об отношении объекта к определенному классу.

При обнаружении объектов на изображении, как правило, неизвестно количество искомых объектов, их расположение и размеры, что значительно усложняет задачу. Одним из первых методов обнаружения был подход на основе сопоставления с шаблоном, т. е. искомым объектов. Несмотря на то, что такой подход может обеспечить максимальную точность, он неустойчив даже к незначительным трансформациям искомого объекта на изображении

относительно исходного. Поэтому для описания объектов в дальнейшем использовали такие наборы признаков, как гистограммные признаки, ключевые точки объекта, признаки на основе спектральных преобразований (например, Фурье, Хаара, Адамара, вейвлет и др.). Такие подходы позволяют повысить возможность обнаружения отличных по масштабу и повороту объектов, однако все же не обеспечивают максимально возможную точность. Кроме этого, неясно как получить наиболее эффективный набор признаков.

Благодаря стремительному развитию вычислительной мощности компьютерной техники обнаружение объектов на изображениях все чаще выполняется с применением предварительно обученных сверточных нейронных сетей (СНС) на больших базах данных изображений, которые позволяют формировать эффективные наборы признаков объектов и их классифицировать. СНС в настоящее время обладают наилучшей обобщающей способностью.

СНС является одним из видов искусственных нейронных сетей (ИНС). ИНС – это существенно параллельно распределенный процессор, который обладает способностью к сохранению и репрезентации опытного знания. Она схожа с мозгом в двух аспектах: знание приобретается сетью в процессе обучения; для сохранения знания используются межнейронные соединения (синаптические соединения).

Работа ИНС состоит в преобразовании входных данных во времени, в результате чего меняется внутреннее состояние сети и формируются выходные воздействия. Обычно ИНС оперирует цифровыми, а не символьными величинами. Большинство моделей ИНС требуют обучения. В общем случае обучение – это такой выбор параметров сети, при котором сеть лучше всего справляется с поставленной проблемой.

Основным элементом нейронной сети является нейрон, который осуществляет операцию нелинейного преобразования суммы произведений входных сигналов на весовые коэффициенты:

$$y = \varphi(b_0 + \sum_{k,j} w_k x_j),$$

где $X = (x_1, x_2, \ldots, x_n)$ – вектор входного сигнала;

$W = (w_1, w_2, \ldots, w_n)$ – весовой вектор;

φ – функция активации;

b_0 – смещение.

Модель нейронного элемента изображена на рисунке 2.1. Каждому входу нейрона соответствует весовой коэффициент (синапс), который характеризует силу синаптической связи по аналогии с биологическим нейроном.

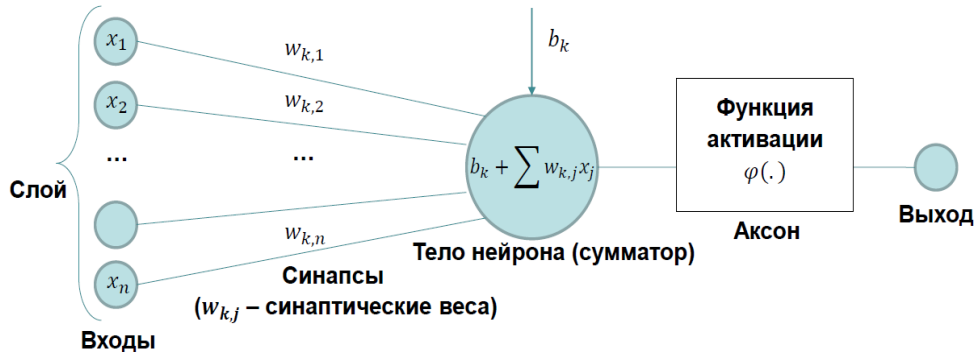


Рисунок 2.1. – Модель искусственного нейрона

Весовой вектор и функция активации определяют поведение любого нейрона, т. е. его реакцию на входные данные. Величина веса w_i определяет степень влияния входа *i* на выход нейрона, а знак – характер влияния. Положительные веса характерны для возбуждающих связей, способствующих повышению активности нейрона. Отрицательные, как правило, соответствуют тормозящим связям. Каждый из входов связан с некоторым источником информации (рецептор, формирующий признаки, распределительная ячейка, выход другого нейрона). Коэффициент b_0 является характеристикой, задающей начальный уровень активности (при нулевом входе). Изменение его эквивалентно смещению пороговой функции по оси абсцисс.

Функция активации используется для ограничения выхода нейрона в заданном диапазоне и для нелинейного преобразования взвешенной суммы. Последнее позволяет нейронному классификатору аппроксимировать любую нелинейную границу между классами в пространстве образов. Функция активации выбирается для конкретной задачи и является неизменной.

Веса конкретного нейрона являются настраиваемыми параметрами. Они содержат знания нейрона, определяющие его поведение. Процесс настройки этих знаний с целью получения нужного поведения называется обучением.

Слоем нейронной сети называется множество нейронных элементов, на которые в каждый такт времени параллельно поступает информация от других нейронных элементов сети. Помимо слоев нейронов часто используется понятие входного распределительного слоя. Распределительный слой передает входные сигналы на первый обрабатывающий слой нейронных элементов.

Сверточные нейронные сети (СНС) – это вид нейронных сетей, которые хотя бы в одном из своих слоев в качестве преобразования используют операцию свертки. Типичная архитектура СНС представляет собой многоступенчатый каскад сетей прямого распространения, т. е. является однонаправленной, и включает сверточные слои, слои подвыборки (субдискретизации, пуллинга) и полносвязные слои. При этом используется преимущество двумерной структуры изображений и с помощью метода локальной связности, ограничивая количество связей между нейронами скрытого сверточного слоя и входными данными, т. е. каждый нейрон скрытого слоя связан только с локальным участком изображения. Сверточные слои основаны на математической операции свертки входного изображения с набором фильтров, которые описываются весами нейронов скрытого слоя. Данная операция в общем виде представляется как

$$y_{r,c,k} = \sum_{i=-H/2}^{H/2} \sum_{j=-W/2}^{W/2} \sum_{d=1}^{D} x_{i+r,j+c,d} \cdot w_{i,j,d,k} + b_k,$$

где (r,c) – положение фильтра на предыдущем слое;

x – выход нейрона предыдущего слоя;

H, W, D – высота, ширина и глубина k-го фильтра;

$w_{i,j,k,d}$ – веса фильтра;

b_k – вес связи с постоянным значением (смещение).

Таким образом, можно сказать, что *сверточный слой* – это набор карт (карты признаков – матрицы), сформированный на основе синаптического ядра (сканирующее ядро, фильтр), которое скользит по всей области предыдущей карты и находит определенные признаки объектов. *Шаг свертки* – это количество пикселей, на которые перемещается матрица фильтра по входному изображению. Если шаг равен единице, то ядро свертки смещается каждый раз на один пиксель, если шаг равен двум, то происходит смещение на два пикселя и т. п. Чем больше шаг, тем меньшего размера карты признаков получаются на выходе. Количество карт определяется требованиями к задаче: большое количество карт повышает качество распознавания, но увеличивает вычислительную сложность. Традиционно используется соотношение «один к двум», т. е. каждая карта предыдущего слоя связана с двумя картами сверточного слоя.

Размер у всех карт сверточного слоя одинаковый и вычисляется как

$$(w,h)=(mW-kW+1,mH-kH+1),$$

где mW, mH – ширина и высота предыдущей карты соответственно;

kW, kH – ширина и высота ядра соответственно.

Каждое ядро находит определенные признаки объектов, например, это могут быть вертикальные или горизонтальные линии, или же линии, расположенные под определёнными углами. Ядро представляет собой систему разделяемых весов или синапсов и является одной из главных особенностей СНС. В обычной многослойной ИНС много связей между нейронами, что замедляет процесс обнаружения. В СНС общие веса позволяют сократить число связей и находить один и тот же признак по всей области изображения.

Размер ядра нечетный от 3×3 до 7×7, так как обработка ведется относительно центрального элемента ядра. Необходимо чтобы размер карт сверточного слоя был четным для исключения потери информации при уменьшении размерности в подвыборочном слое. Малый размер ядра не сможет выделить какие-либо признаки большего размера, а при большом размере ядра увеличивается количество связей между нейронами и, соответственно, время обучения и работы СНС.

Подвыборочный слой также имеет карты, их количество совпадает с количеством карт предыдущего сверточного слоя. Подвыборочный слой служит для уменьшения размерности карт предыдущего слоя. Такой подход позволяет снизить вариативность данных, повысить устойчивость СНС к небольшим изменениям изображений в пределах локальной области.

В процессе сканирования карты предыдущего сверточного слоя ядром подвыборочного слоя сканирующее ядро на нем не пересекается, что является отличием от организации свертки. Обычно каждая карта имеет ядро размером 2×2, и это позволяет уменьшить предыдущие карты сверточного слоя в два раза.

Для этого вся карта признаков разделяется на ячейки 2×2 элемента, из которых выбираются определенные значения. Наиболее используемые два типа объединения: максимальное (max pooling) и среднее (average pooling) (рисунок 2.2). Первое возвращает максимальное значение из покрытой ядром части изображения, второе возвращает среднее значение из всех значений покрытой ядром части.

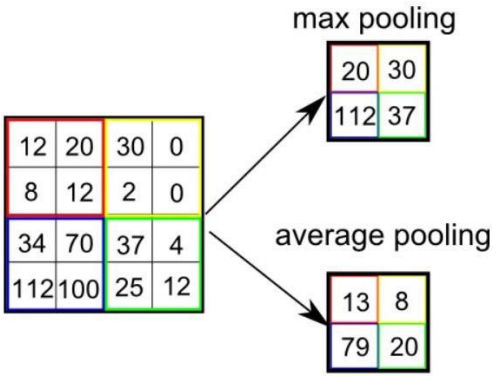


Рисунок 2.2. – Примеры типов объединений слоя подвыборки

Результат свертки после слоя подвыборки поступает на функцию активации, которая определяет выходной сигнал и является нелинейной, т. е. позволяет оставлять для дальнейшего анализа только значимые нейроны, в качестве которых желательно использовать нелинейные активационные функции, например, сигмоидального вида. Следует отметить, что широко используются активационные функции, состоящие из линейных участков, например, RELU (rectified linear unit) и PRELU, которая использует наклон отрицательной части в качестве параметра в процессе обучения с использованием алгоритма обратного распространения ошибки, что позволяет увеличить точность работы СНС.

Формально данный этап может быть описан выражением

$$y_p^l = \varphi\left(a^l \cdot \beta\left(x^{l-1}\right) + b^l\right),$$

где y_p^l – выход текущего подвыборочного слоя;

φ – функция активации;

a^l, b^l – коэффициенты сдвига;

β – функция выборки значений из локальной области.

Слои свёртки и подвыборки вместе образуют *i*-й слой свёрточной нейронной сети. За счет использования нескольких попеременных слоев свертки и подвыборки СНС позволяет получать представления, независимые от конкретного расположения локального признака в изображении, и одинаковым образом реагировать на интересующие объекты, т. е. обнаруживать объекты на любом участке изображения. Количество таких слоёв может варьировать в зависимости от сложности изображений, чтобы точнее выделять признаки, однако увеличение слоев требует увеличения вычислительной мощности и большего объема памяти.

СНС обеспечивает инвариантность по отношению к сдвигу, однако не предусматривает устойчивости к другим аффинным преобразованиям, например, вращению или зеркальному отражению. Для решения этой проблемы, как правило, используются эвристические методы, такие как выравнивание изображения по линии горизонта, использование многомасштабного представления и различных отражённых копий. Последний блок СНС является многослойным персептроном, машиной опорных векторов или другим классификатором. Он служит для задачи классификации, моделирует сложную нелинейную функцию, оптимизацией которой улучшается качество распознавания. Нейроны каждой карты предыдущего подвыборочного слоя связаны с одним нейроном скрытого слоя. Связи могут быть разными, например: только часть нейронов какой-либо из карт подвыборочного слоя быть связана с первым нейроном скрытого слоя, а оставшаяся часть со вторым, либо все нейроны первой карты связаны с нейронами первого и второго скрытого слоя. Пример топологии СНС для многоклассовой классификации показан на рисунке 2.3.

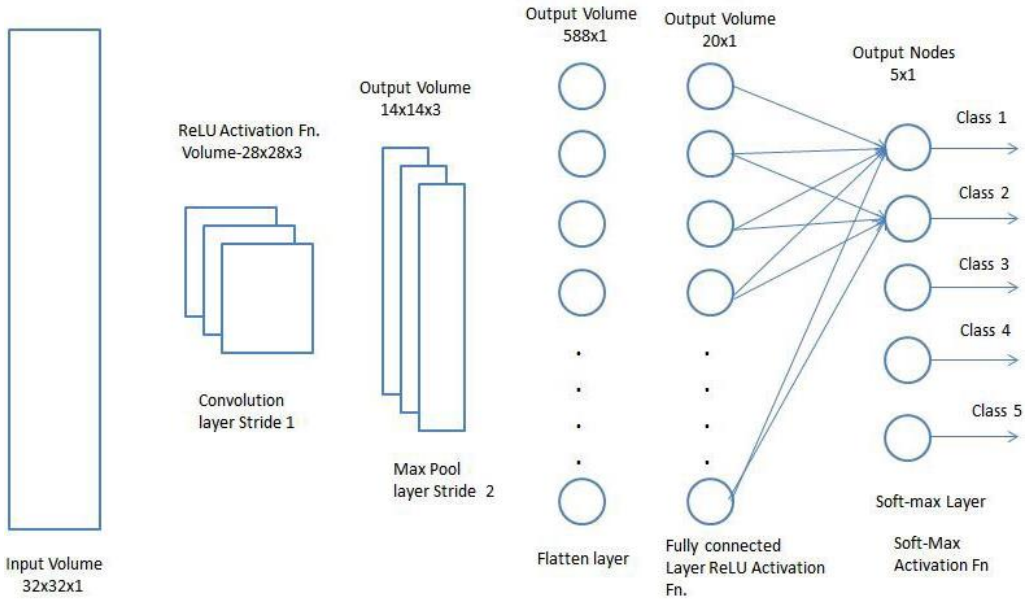


Рисунок 2.3. – Топология СНС для многоклассовой классификации

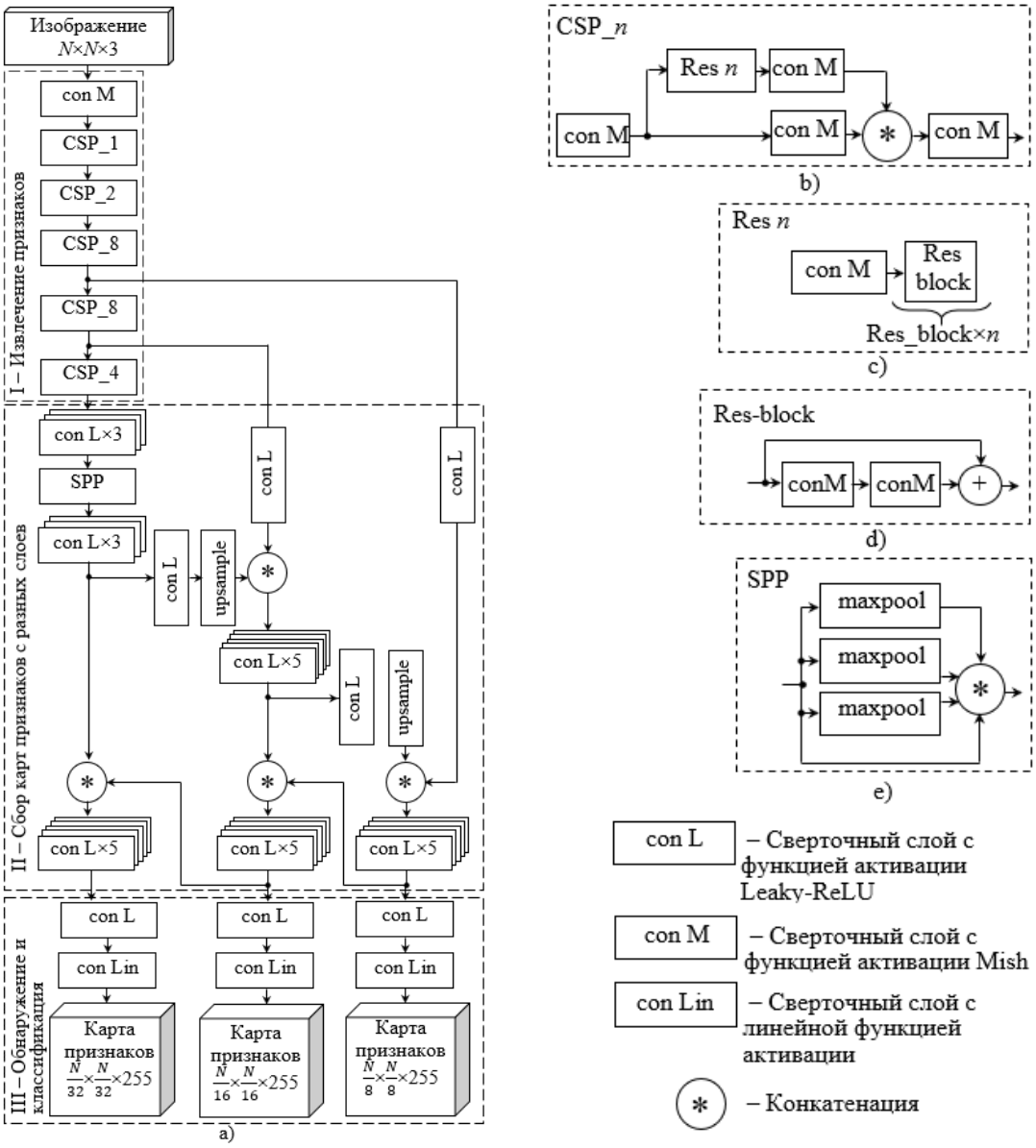
Среди известных моделей СНС, отличающихся архитектурой, точностью и быстродействием, для обнаружения объектов на изображениях и видеопоследовательностях следует выделить двухэтапные СНС, которые предполагают на первом этапе локализацию регионов-претендентов, а на втором этапе выполняется их классификация. Однопроходные СНС позволяют осуществлять локализацию и классификацию в составе одной модели, и не требуют обработки одного и того же изображения повторно.

Среди двухпроходных следует выделить модификации R-CNN (Region-based Convolutional Neural Networks) и ResNet (Residual Networks). Первая версия модели R-CNN основана на предварительном выделении регионов на изображении, вычислении признаков с использованием СНС (например, AlexNet) и применении классификатора для идентификации объектов. Данная модель получила развитие в версиях Fast R-CNN и Faster R-CNN. Архитектура Fast R-CNN направлена на уменьшение использования числа регионов на изображении за счет применения слоя ROI, который действует по принципу слоев субдискретизации, позволяя получать области уменьшенного размера. В Faster R-CNN предполагается выполнение двух этапов. На первом из них определяются области на изображении, в которых предположительно могут быть расположены объекты, с применением глубокой полносвязной сети (Region Proposal Network). На втором этапе используется детектор Fast R-CNN, который выполняет поиск объектов в предложенных регионах. Модель Faster R-CNN неустойчива к зашумленным изображениям и требует значительных вычислительных затрат.

Архитектура СНС ResNet использует пропускающие (residual) соединения, которые позволяют минимизировать ухудшение качества работы при увеличении количества слоев СНС, если на некотором слое сети достигнут предел точности. Коэффициент ошибок составляет 3,57% в метрике top 5. В работе представлена модель Inseption-ResNet как развитие модели Inseption путем ввода замыкающих соединений (shortcut connection), которые при необходимости позволяют пропускать слой и, соответственно, обнулять его влияние на результат работы детектора. Это даёт возможность изменять архитектуру сети так, чтобы конечное количество слоёв определялось для конкретной задачи в процессе обучения, что позволяет уменьшить коэффициент ошибки до 3,1% в метрике top 5. Вместе с тем представленные модели характеризуются высокими временными затратами. Поэтому для решения многих задач используются усечённые варианты архитектур с уменьшенным количеством слоев (например, ResNet-34, ResNet-50 и др.), которые, однако, не позволяют достичь точности базовой модели.

Семейство СНС YOLO относится к однопроходным и даёт возможность осуществлять локализацию и идентификацию объектов одной моделью. Алгоритм обнаружения объектов YOLOv3 обладает достаточно высокой точностью, однако скорость работы меньше по сравнению с предыдущими версиями YOLO, и разработка YOLOv4 [6] была направлена на увеличение

скорости работы и оптимизацию параллельных вычислений. Структура[1] YOLOv4 представлена на рисунке 2.4.



a) – общая схема; b) – структура блока с CSP-соединением;
c) – объединение *n* Res-блоков; d) – структура Res-блока; e) – блок SPP

Рисунок 2.4. – Архитектура YOLOv4

Структуру YOLOv4 можно разделить на три основных блока: блок извлечения признаков, который служит для выявления характерных особенностей объектов на входном изображении; блок сбора карт признаков с разных

[1] Для построения схемы использовался файл конфигурации YOLOv4.cfg: https://github.com/-AlexeyAB/darknet/blob/master/cfg/yolov4.cfg и веб приложение: netron.app/?url=-https://raw.githubusercontent.com/AlexeyAB/darknet/master/cfg/yolov4.cfg.

слоев, который собирает и передает карты признаков с различных уровней нейронной сети на блок обнаружения и классификации, который, в свою очередь, формирует выходные карты признаков для разных масштабов, что позволяет предсказывать координаты ограничительных рамок и классифицировать содержимое каждой ячейки на входном изображении.

На рисунке 2.4, a) блок извлечения признаков представляет собой СНС CSPDarknet-53, в основе которой находится Darknet-53, состоящая из 53 сверточных слоев. Отличие заключается в использовании межэтапных соединений (CSP – Cross-Stage-Partial connection) для снижения вычислительной сложности. На рисунке 2.4, b) представлена структура блока с CSP-соединением. В данном блоке карты признаков разделяются на две части: одна из них проходит через группу Res-блоков (Residual blocks), другая поступает на сверточный слой, после чего выходные данные объединяются. Res-блоки представлены в группах по 1, 2, 4 или 8 блоков (см. рисунок 2.4, c), каждый из которых состоит из сверточных слоев 1×1 и 3×3 с замыкающим соединением (shortcut connection) (см. рисунок 2.4, d). Использование такого соединения обеспечивает альтернативный путь для градиента, что приводит к лучшей сходимости модели. Другим отличием CSPDarknet-53 является использование функции активации Mish, которая представляет собой комбинацию из функции идентичности, гиперболического тангенса, softplus и характеризуется достаточно низкой вычислительной сложностью и определяется как

$$f(x) = x\tanh\left(softplus(x)\right) = x\tanh\left(\ln\left(1+e^{x}\right)\right).$$

Кроме этого, Mish является немонотонной, гладкой и непрерывной функцией, неограниченной сверху, но ограниченной снизу. Отсутствие верхней границы позволяет избегать насыщения, которое может приводить к замедлению обучения, а значит позволяет ускорить процесс обучения. Наличие нижней границы обеспечивает эффект регуляризации. Немонотонность сохраняет небольшие отрицательные значения, что стабилизирует градиентный поток, а гладкость и непрерывность эффективны при обобщении и оптимизации результатов.

Блок сбора карт признаков с разных слоев включает модифицированный модуль SPP (SPP – Spatial Pyramid Pooling) (см. рисунок 2.4, e) для увеличения рецептивного поля и модифицированный модуль PAN для создания пирамиды признаков. Первый из них выполняет операцию maxpool по картам

признаков $\frac{N}{32}\times\frac{N}{32}\times 255$ с разным размером ядра $k=\{1,5,9,13\}$, но идентичным заполнением, которое используется для сохранения пространственного размера. Затем четыре комплекта соответствующих карт признаков объединяются в один, размером $\frac{N}{32}\times\frac{N}{32}\times 2048$. Это увеличивает область анализа, улучшая таким образом точность модели, при этом без существенных вычислительных затрат.

Для агрегирования характеристик объектов в YOLOv4 используется модифицированная версия PAN (Path Aggregation Networks), использующая пирамиды признаков, которые служат для объединения карт признаков нижних и верхних уровней сети, то есть сочетают семантическую информацию с верхних уровней и более точную с нижних. В этой модификации PAN вместо операции суммирования результатов соседних слоев используется конкатенация, что позволяет увеличить точность прогнозов.

Блок обнаружения и классификации используется для нахождения ограничительной рамки и классификации содержимого в каждой ячейке. Изображение размером $N\times N\times 3$, подаваемое на вход нейронной сети, разделяется на $S\times S$ ячеек – которые представляют собой область интереса (RoI – Region of Interest). Количество ячеек прямо пропорционально размеру входного изображения и обратно пропорционально шагу дискретизации на каждом из масштабов. Карты признаков на выходе сети YOLOv4 используют шаги дискретизации 32, 16 и 8, которые показывают во сколько раз уменьшился размер выходного слоя относительно входного изображения и позволяют осуществлять обнаружение в трех масштабах. Карты признаков на каждом из трех масштабов будут иметь размер $S\times S\times\left(N_B\times\left(5+C\right)\right)$, где N_B – количество ограничительных рамок, которые могут быть предсказаны для каждой ячейки, C – количество классов объектов. В YOLOv4 для каждой ячейки используется 3 anchor-рамки, представляющие собой прямоугольники различных размеров с разным соотношением сторон. Вектор признаков каждой ячейки предсказывает 85 параметров для трех anchor-рамок: предсказывается смещение координат t_x и t_y и отклонения размеров t_w и t_h, на основании которых определяется положение предсказанных рамок (рисунок 2.5), вероятность обнаружения объекта и вероятность принадлежности объекта к классу.

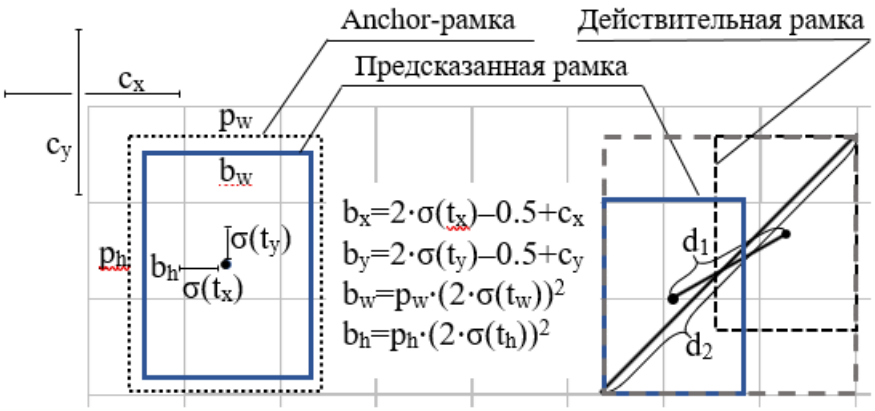


Рисунок 2.5. – Построение ограничительных рамок при обнаружении объектов: c_x, c_y – координаты верхней левой ячейки; p_h и p_w – размеры anchor-рамки; b_x, b_y, b_w, b_h – координаты центра и размеры предсказанной ограничительной рамки; $\sigma(t)$ – функция сигмоиды, t_x, t_y, t_w, t_h – отклонение координат и размеров предсказанной рамки; d_1 – расстояние между центральными точками действительной и предсказанной рамки, d_2 – диагональ минимальной рамки, охватывающей две рамки

Тогда вектор признаков для каждой ячейки изображения с учетом трех anchor-рамок будет равен 1×1×255. На основании вектора признаков определяются параметры предсказанной рамки. Предсказанная рамка также имеет 85 параметров: P_c – вероятность правильного нахождения ограничительной рамки, b_x и b_y – координаты центра, b_h и b_w – высота и ширина предсказанной ограничительной рамки, $c_1 \ldots c_{80}$ – вероятность нахождения объекта одного из 80 классов. Класс объекта, ограниченного предсказанной рамкой, определяется с использованием $c_1 \ldots c_{80}$ и сигмоидальной функции. Для определения положения ограничительных рамок применяется функция потерь L_{CIoU} :

$$L_{CIoU} = 1 - IoU + \frac{d_1^2}{d_2^2} + \alpha v,$$

где b и b_{gt} – центральные точки B и B_{gt}. $B = \left(b_x, b_y, b_w, b_h\right)$ – предсказанная рамка объекта, $B_{gt} = \left(x_{gt}, y_{gt}, w_{gt}, h_{gt}\right)$ – действительная рамка;

d_1 – евклидово расстояние между центральными точками предполагаемой и действительной ограничительной рамки;

d_2 – диагональ минимальной рамки, которая формируется для предсказанной и действительной ограничительных рамок;

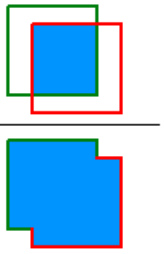
α – параметр, зависящий от взаимного расположения предсказанной и действительной рамки:

$$\alpha = \begin{cases} 0, \ IoU < 0{,}5; \\ \dfrac{v}{(1 - IoU) + v}, \ IoU \geq 0{,}5, \end{cases}$$

где $v = \dfrac{4}{\pi^2}\left(\arctan\dfrac{w_{gt}}{h_{gt}} - \arctan\dfrac{b_w}{b_h}\right)^2$ – мера согласованности сторон ограничительных рамок;

IoU – отношение между пересечением и объединением предсказанной и действительной рамок, т. е. для прямоугольников, описывающих найденные (R) и аннотированные (O) объекты на изображениях базы данных:

$$IoU(O,R) = \frac{O \cap R}{O \cup R} =$$



Объект считается правильно обнаруженным, если $\left(IoU(O,R)\right) \geq 0{,}5$. Примеры перекрытия объектов для $\left(IoU(O,R)\right)$ показаны на рисунке 2.6.

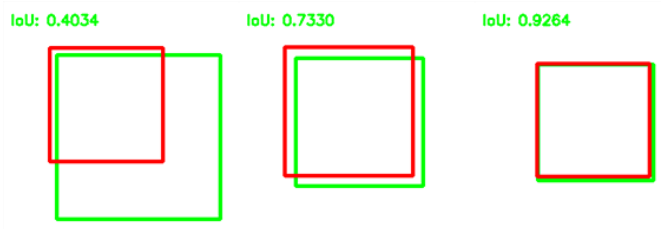


Рисунок 2.6. – Примеры перекрытия объектов и значения метрики $\left(IoU\right)$

При расчете потерь учитывается площадь перекрытия, расстояние между центральными точками и соотношение сторон. L_{CIoU} позволяет увеличить площадь перекрытия действительной и предсказанной ограничительной рамки, минимизировать расстояние между центральными точками и сохранить постоянное соотношение сторон. L_{CIoU} обеспечивает лучшую скорость и точность в задаче приближения предсказанной ограничительной рамки к истинному положению по сравнению с L_{DIoU} (Distance-IoU loss) и L_{GIoU} (Generalized-IoU loss) потерями.

Подавление немаксимумов (NMS) применяется для обработки ситуации, когда для одного и того же объекта предсказано несколько рамок с высокими вероятностями соответствия. В YOLOv4 используется подход на основе greedy NMS, которая следует гипотезе о том, что обнаруженные предполагаемые рамки с большим количеством перекрытий соответствуют одному и тому же объекту. При этом рамки-кандидаты сортируются по степени достоверности классификации и удаляются дублирующие.

При обнаружении объектов могут быть следующие ситуации:

– искомые объекты обнаружены правильно, т. е. истинно-положительные случаи или правильные обнаружения (True Positive, TP);

– ложные объекты обнаружены как искомые, т. е. ложно-положительные случаи или ложные обнаружения(False Positive, FP);

– ложные объекты обнаружены как ложные, т. е. истинно-отрицательные случаи (True Negative, TN);

– искомые объекты обнаружены как ложные, т. е. ложно-отрицательные случаи или пропуски (False Negative, FN).

Для оценки эффективности обнаружения объектов используют следующие показатели (метрики):

– точность (precision) определяет долю правильно обнаруженных объектов к общему числу срабатываний детектора:

$$p = \frac{TP}{TP + FP};$$

– полнота (recall) характеризует долю объектов, реально относящихся к искомому классу, обнаруженных верно:

$$r = \frac{TP}{TP + FN}.$$

На основе полученных значений рассчитывается средняя точность (AP) обнаружения объектов для каждого класса для 11 пороговых уровней в диапазоне от 0 до 1 с шагом 0,1 по формуле:

$$AP = \frac{1}{11} \cdot \sum_{i=0}^{10} p_{\text{int}}(r_i),$$

где $p_{\text{int}}(r_i) = \max p(\tilde{r})$, $\tilde{r} \geq r_i$.

При вычислении метрики mAP (meanAveragePrecision), которая предполагает усреднение значений метрики AP (AveragePrecision) по классам, значения полученной средней точности усредняются для заданных классов объектов.

Наиболее широко в качестве баз данных для тестирования алгоритмов обнаружения объектов используются:

– PASCAL VOC 2007, 2012 – изображения 20 классов объектов [7];

– MS COCO – набор изображений, содержащих разметку для 80 категорий объектов [8];

– Open Images Dataset – изображения, содержащие разметку 600 классов объектов [9];

– Street View House Numbers (SVHN) – изображения с разметкой окаймляющих прямоугольников для цифр, содержащихся на естественных надписях (в частности, на номерах домов) [10];

– Stanford Dogs Dataset – изображения собак 120 пород [11].

Задания к лабораторной работе

1. *Реализовать обнаружение объектов на изображениях с использованием СНС YOLOv4.* Для этого использовать файлы:

– *yolov4.cfg* – файл конфигурации, который описывает архитектуру используемой сети;

– *yolov4.weights* – файл весовых коэффициентов для обученной модели;

– *coco.names* – файл, содержащий имена классов объектов для изображений из базы данных MS COCO [8], на которых была обучена модель. Имена классов представлены в приложении 2.1.

Указанные файлы расположены в репозитории Darknet [12] на GitHub: https://github.com/AlexeyAB/darknet.

Тестовый набор MS COCO доступен по ссылке http://images.cocodataset.org/zips/test2017.zip.

В приложении 2.2 представлены фрагменты кода на языке Python, для реализации алгоритма обнаружения объектов заданных классов.

Размер входного слоя должен быть кратен 32, т. к. в YOLOv4 по мере прохождения по сети входное изображение со сторонами $N \times N$ уменьшается с шагом дискретизации d, равным 8, 16 и 32, и на каждом из трех масштабов предсказания входной ячейке размером N / d будет соответствовать вектор признаков с размерностью $1 \times 1 \times K$, где K – количество карт признаков размером $N / d \times N / d$.

Степень уверенности сети (`confidence`) отражает вероятность того, что ограничительная рамка объекта предсказана верно, и пороговое значение степени уверенности (`confidenceThreshold`) определяет будет ли эта ограничительная рамка представлена как результат обнаружения.

2. *Вычислить точность и полноту детектора при обнаружении объектов на 10 изображениях из базы данных MS COCO для различных степеней уверенности детектора в заданном диапазоне и с определенным шагом, например, от 0,3 до 0,9 с шагом 0,2. Каждое изображение должно содержать не менее 3 объектов разного класса. Данные представить в виде таблицы. Сделать выводы.*

Таблица. – Представление результатов обнаружения объектов

Уверенность СНС	1		2		…	10	
	p	*r*	*p*	*r*		*p*	*r*

Пример обнаружения объектов на изображении показан на рисунке 2.7. Согласно визуальной оценке на изображении, представленном на этом рисунке, верно детектировано девять объектов, ложного обнаружения не было, пропущен один объект, т. к. не обнаружен один из четырех автомобилей в мозаике.

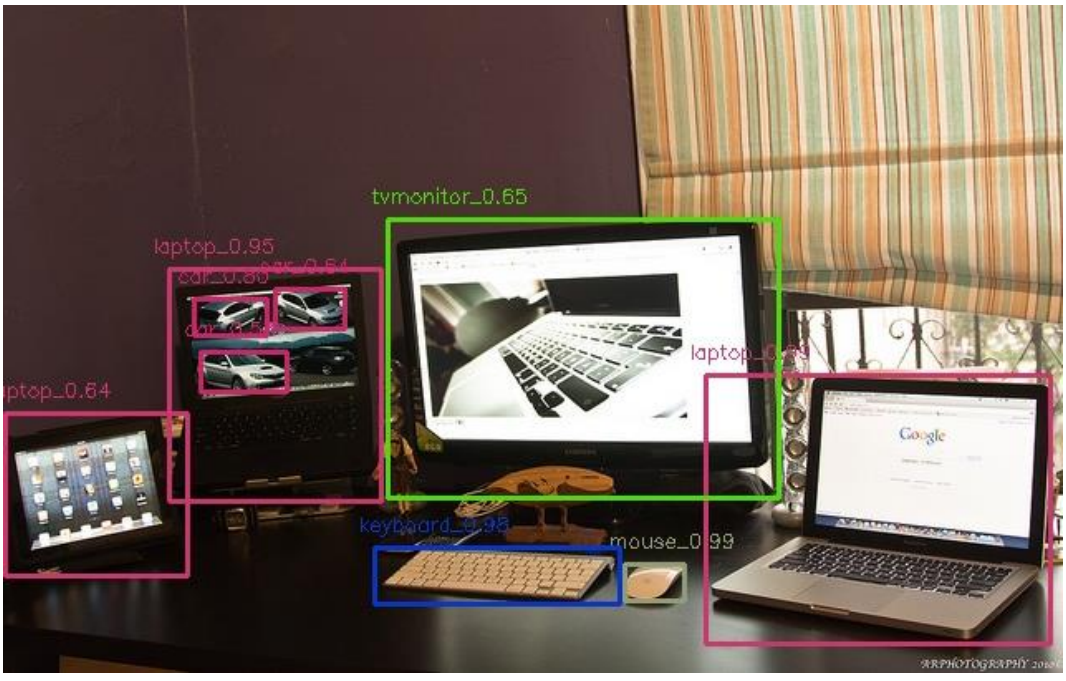


Рисунок 2.7. – Пример обнаружения объектов различных классов на изображении

3. *Внести изменения в код для обнаружения объектов только заданных классов по вариантам.*

Для обнаружения объектов только заданных классов необходимо добавить условие проверки соответствия требуемому номеру класса. Например, если требуется обнаружение объектов, которые принадлежат только классам «автомобиль» и «человек», то следует добавить строку

```
if class_id == 0 or class_id == 2:
```

после определения класса обнаруженного объекта в строке 46 с соблюдением табуляции. Следует учитывать, что нумерация классов начинается с 0.

Для детектирования использовать параметры СНС, определенные как наилучшие в пункте 2. Представить примеры обнаружения заданных объектов на изображениях, на которых присутствуют объекты пяти разных классов или более.

Вариант 1: Велосипед и мотоцикл.

Вариант 2: Автомобиль и грузовик.

Вариант 3: Поезд и автобус.

Вариант 4: Кот и собака.

Вариант 5: Лошадь и корова.

Вариант 6: Рюкзак и зонтик.

Вариант 7: Бейсбольная бита и бейсбольная перчатка.

Вариант 8: Птица и воздушный змей.

Вариант 9: Яблоко и банан.

Вариант 10: Зубная щетка и фен для волос.

Вариант 11: Микроволновая печь и духовка.

Вариант 12: Светофор и пожарный гидрант.

Вариант 13: Человек и сумка.

Вариант 14: Собака и овцы.

Вариант 15: Скамейка и знак остановки.

Вариант 16: Лыжи и сноуборд.

Вариант 17: Бутылка и бокал.

Вариант 18: Вилка и ложка.

Вариант 19: Чашка и миска.

Вариант 20: Мышь и клавиатура.

Вариант 21: Пульт и мобильный телефон.

Вариант 22: Фрисби и спортивный мяч.

Вариант 23: Зебра и жираф.

Вариант 24: Самолет и лодка.

Вариант 25: Брокколи и морковь.

Вариант 26: Медведь и слон.

Вариант 27: Стул и обеденный стол.

Вариант 28: Ваза и растения в горшке.

Вариант 29: Торт и пицца.

Вариант 30: Книга и плюшевый мишка.

Содержание отчета

1. Титульный лист.
2. Цель работы.
3. Ход работы с пояснениями выполнения всех заданий.
4. Результаты оценки эффективности обнаружения в виде таблицы по заданию 2.
5. Представить примеры не менее трех худших результатов обнаружения объектов на изображениях из базы данных MS COCO по заданию 2. Пояснить результаты обнаружения.
6. Результаты выполнения пункта 3.
7. Листинг кода.
8. Выводы о проделанной работе.

Контрольные вопросы

1. Состав модели искусственного нейрона?
2. Назначение и примеры функций активации?
3. Топология СНС для многоклассовой классификации? Сформулируйте назначение каждого типа слоя СНС.
4. Приведите пример вычисления свертки для фрагмента изображения и произвольного ядра слоя.
5. Что понимается под задачей обнаружения объектов на изображении?
6. Какие критерии могут быть использованы для оценки эффективности обнаружения объектов на изображениях с использованием СНС?
7. Какие современные модели СНС применяются для обнаружения объектов на изображении? Дайте определение однопроходным и двухпроходным моделям СНС.
8. В чем премущества моделей семейства Yolo?
9. Какой фреймворк поддерживает использование Yolo для детктирования объектов и каковы особенности его применения?
10. Из каких основных блоков состоит СНС YOLOv4?

Приложение 2.1
Классы объектов изображений в базе данных MS COCO

1	person	человек	41	wine glass	бокал
2	bicycle	велосипед	42	cup	чашка
3	car	автомобиль	43	fork	вилка
4	motorbike	мотоцикл	44	knife	нож
5	aeroplane	самолет	45	spoon	ложка
6	bus	автобус	46	bowl	миска
7	train	поезд	47	banana	банан
8	truck	грузовик	48	apple	яблоко
9	boat	лодка	49	sandwich	бутерброд
10	traffic light	светофор	50	orange	апельсин
11	fire hydrant	пожарный гидрант	51	broccoli	брокколи
12	stop sign	знак остановки	52	carrot	морковь
13	parking meter	парковочный счетчик	53	hot dog	хот-дог
14	bench	скамейка	54	pizza	пицца
15	bird	птица	55	donut	пончик
16	cat	кот	56	cake	торт, пирог
17	dog	собака	57	chair	стул
18	horse	лошадь	58	sofa	диван
19	sheep	овца	59	potted plant	растение в горшке
20	cow	корова	60	bed	кровать
21	elephant	слон	61	dining table	обеденный стол
22	bear	медведь	62	toilet	туалет
23	zebra	зебра	63	tvmonitor	монитор
24	giraffe	жираф	64	laptop	ноутбук
25	backpack	рюкзак	65	mouse	мышь
26	umbrella	зонт	66	remote	пульт
27	handbag	сумка	67	keyboard	клавиатура
28	tie	галстук	68	cell phone	мобильный телефон
29	suitcase	чемодан	69	microwave	микроволновая печь
30	frisbee	фрисби	70	oven	духовка
31	skis	лыжи	71	toaster	тостер
32	snowboard	сноуборд	72	sink	раковина
33	sports ball	спортивный мяч	73	refrigerator	холодильник
34	kite	воздушный змей	74	book	книга
35	baseball bat	бейсбольная бита	75	clock	часы
36	baseball glove	бейсбольная перчатка	76	vase	ваза
37	skateboard	скейтборд	77	scissors	ножницы
38	surfboard	доска для серфинга	78	teddy bear	плюшевый мишка
39	tennis racket	теннисная ракетка	79	hair drier	фен для волос
40	bottle	бутылка	80	toothbrush	зубная щетка

Приложение 2.2
Особенности и примеры программной реализации поиска объектов

Подключение модуля OpenCV dnn для работы с фреймворком Darknet:

```
net = cv2.dnn.readNetFromDarknet(config_file, weights_file)
```

Чтение классов объектов из файла:

```
with open(classes_file, 'r') as f:
classes = [line.strip() for line in f.readlines()]
```

Для каждого значения степени уверенности сети, для каждого из выбранных изображений создается blob-объект, который затем передается в сеть для обработки:

```
confidenceThreshold = [0.3, 0.5, 0.7, 0.9]
  for th in confidenceThreshold:
    for image_file in images:
      image = cv2.imread(image_file)
      width = image.shape[1]
      height = image.shape[0]
      blob = cv2.dnn.blobFromImage(image, 0.00319, (s[0],
s[1]), (0, 0, 0), True, crop=False)
# передача изображения в сеть
      net.setInput(blob)
      ln = net.getLayerNames()
```

YOLOv4 имеет 3 несвязанных выхода, и для получения предсказаний на этих выходах, обнаружения регионов-кандидатов для объектов и получения координат ограничительных рамок для лучших предсказаний используется алгоритм подавления немаксимумов NMS:

```
ln = [ln[i[0] - 1] for i in net.getUnconnectedOutLayers()]
      outs = net.forward(ln)
      class_ids = []
      confidences = []
      boxes = []
      nms_threshold = 0.4
# Для каждого из 3-х выходных слоев YOLOv4:
      for out in outs:
        for detection in out:
          scores = detection[5:]
```

```
            class_id = np.argmax(scores)
            confidence = scores[class_id]
            if confidence > th:
              center_x = int(detection[0] * width)
              center_y = int(detection[1] * height)
              w = int(detection[2] * width)
              h = int(detection[3] * height)
              x = center_x - w / 2
              y = center_y - h / 2
              class_ids.append(class_id)
              confidences.append(float(confidence))
              boxes.append([x, y, w, h])
# Определение координат предсказанных объектов:
      indices = cv2.dnn.NMSBoxes(boxes, confidences, th,
nms_threshold)
      for i in indices:
      i = i[0]
      box = boxes[i]
      x = box[0]
      y = box[1]
      w = box[2]
  h = box[3]
```

Нанесение на изображение ограничительной рамки обнаруженных объектов с указанием предсказанного класса объекта и степени уверенности сети в своем предсказании:

```
draw_bounding_box(image, class_ids[i], confidences[i],
round(x), round(y), round(x + w), round(y + h))
```

ЛАБОРАТОРНАЯ РАБОТА № 3
Обучение сверточной нейронной сети для обнаружения объектов на изображениях

Цель работы: изучение и применение обучения сверточной нейронной сети методом обратного распространения ошибки для программной реализации обнаружения заданных классов объектов.

Теоретические сведения

Обучение ИНС предполагает определение весов соединений между нейронами таким образом, чтобы ИНС приближала необходимую функцию с заданной точностью. Применяются три подхода к обучению ИНС: обучение с учителем (supervised learning), обучение без учителя (unsupervised learning) и обучение с подкреплением (reinforcement learning). При выполнении данной работы необходимо применить первый подход, т. е. обучение с учителем, который предполагает, что имеется обучающее множество с заранее известными правильными ответами. Для заданных весовых коэффициентов ИНС сравнивается полученный результат с правильным ответом и в зависимости от полученной ошибки принимается решение для корректировки весов [13].

Формально общая постановка задачи обучения с учителем представляется следующим образом. Пусть имеется обучающее множество, состоящее из N примеров. Каждый обучающий пример задаётся в следующем виде: $\{(x_1, y_1), \ldots, ([x_N, y_N])\}$, где x_i – вектор входных признаков i-го примера, y_i – вектор, задающий выходное значение i-го примера.

Тогда алгоритм такого обучения должен определить функцию g: $X \to Y$, где X – пространство входов модели, Y – пространство выходов. Функция g является элементом пространства функций G, которое называют также пространством гипотез.

Существует два основных подхода к выбору функции g: минимизация эмпирического риска и минимизация структурного риска. Минимизация эмпирического риска ищет функцию, которая наилучшим образом соответствует обучающим данным. Минимизация структурного риска включает в себя функцию штрафа, которая ищет компромисс между смещением и дисперсией (дилемма смещение-дисперсия – чем меньше смещение оценки параметра

модели, тем выше её дисперсия, и наоборот). В обоих случаях предполагается, что обучающее множество состоит из независимых и одинаково распределенных пар (x_i, y_i). С целью проверить, насколько хорошо функция соответствует обучающим данным, определяется функция потерь (целевая функция, функция ошибки, функция затрат или функция стоимости) $L(y_i, \hat{y})$, где $\hat{y}$ – значение, предсказанное моделью для примера (x_i, y_i), которая может быть определена как разница между желаемым выходным значением y и прогнозируемым выходным значением $\hat{y}$.

Риск $R(g)$ определяется как потери g которые на обучаемых данных могут быть оценены следующим образом:

$$R(g) = \frac{1}{N}\sum_{i=1}^{N} L(y_i, g(x_i)).$$

Для обучения сверточных нейронных сетей часто используется метод обратного распространения ошибки. Общий принцип метода состоит из трех этапов: прямой проход (вычисление градиентов), вычисление значения функции ошибки (функции потерь) и ее градиента, обратный проход (коррекция весовых коэффициентов) [14]. Коррекция весовых коэффициентов направлена на минимизацию значения функции потерь, и наиболее популярным методом, определяющим минимальное значение функции потерь, является градиентный спуск [15].

На первом этапе обучения необходимо инициализировать веса ИНС и определить функцию потерь, т. к. на результаты обучения большое влияние оказывает подбор начальных значений весов ИНС. Идеальными считаются значения, достаточно близкие к оптимальным [16], однако определить их практически невозможно для задачи обнаружения объектов на изображении. Корректный выбор весов позволяет предотвратить резкое насыщение (взрыв) или исчезновения выходов активационного слоя во время прямого прохода через глубокую, т. е. с большим количеством слоев, ИНС. Это приведет к тому, что либо потребуется большее количество этапов обучения (эпох) для достижения необходимой точности сети, либо в принципе невозможно будет обучить ИНС. Пока не существует универсального метода подбора весов, который может гарантировать нахождение наилучшей начальной точки для любой решаемой задачи, однако получены и применяются следующие рекомендации:

– не использовать нулевые начальные веса. В противном случае все нейроны будут вести себя одинаково и сеть не сможет обучиться для разных признаков;

– исходные значения весов должны быть ассиметричны, т. е. два скрытых блока с одинаковой функцией активации, подключенные к одним и тем же входам, должны иметь разные начальные параметры;

– случайная инициализация считается универсальным способом установления начальных значений весам сети. Она должна обеспечивать такую стартовую точку активации нейронов, которая лежала бы достаточно далеко от зоны насыщения. Для этого диапазон допустимых значений ограничивается. Может быть использован алгоритм, который позволяет формировать массив, элементами которого являются случайные величины, распределённые по нормальному закону с математическим ожиданием, равным нулю, и среднеквадратичным отклонением, равным единице. Однако особенность этого подхода в том, что он удовлетворительно работает при обучении небольших архитектур ИНС, но результативность для глубоких ИНС ухудшается.

Современные библиотеки машинного обучения используют вариации инициализации весов маленькими случайными числами с разными способами распределения случайных чисел.

В качестве функции потерь наиболее часто используют следующие:

– средняя квадратичная ошибка (MSE). Является наиболее распространенной и широко используется в линейной регрессии, в которой зависимость переменной от одной или нескольких других переменных определяется линейной функцией, в качестве показателя эффективности. Для расчёта используется формула

$$MSE = \frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2,$$

где y_i – фактический ожидаемый результат, $\hat{y}_i$ – прогноз модели;

– среднеквадратическая ошибка (RMSE):

$$RMSE = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2} = \sqrt{MSE};$$

– средняя абсолютная ошибка (MAE). Представляет собой усредненную сумму модулей разницы между реальными и предсказанными значениями:

$$MAE = \frac{1}{m}\sum_{i=1}^{m}|y_i - \hat{y}_i|;$$

– среднеквадратичная логарифмическая ошибка (RMSLE). Представляет собой RMSE, рассчитанную в логарифмическом масштабе:

$$RMSLE = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(\log(y_i+1)-\log(\hat{y}_i+1))^2} = RMSE(\log(y_i+1),\log(\hat{y}_i+1)) =$$
$$= \sqrt{MSE(\log(y_i+1),\log(\hat{y}_i+1))};$$

– кросс-энтропия (или логарифмическая функция потерь) измеряет расхождение между двумя вероятностными распределениями. Чем меньше значение кросс-энтропии, тем больше распределения похожи друг на друга. Кросс энтропия определяется как

$$H(P,Q) = -\sum_{x} P(x)\log Q(x);$$

где P – распределение истинных ответов;

Q – распределение вероятностей прогнозов модели.

Существуют разные подходы для нахождения минимального значения функции потерь, однако наиболее распространенным является метод градиентного спуска. Градиентный спуск – метод нахождения локального экстремума (минимума или максимума) функции с помощью движения вдоль градиента. Суть заключается в определении наклона графика функции с помощью частной производной. Наклон графика функции указывает на ближайшую впадину. На рисунке 3.1 отображен принцип нахождения минимума функции:

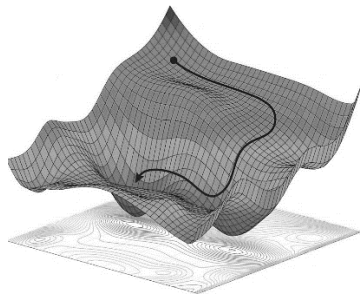
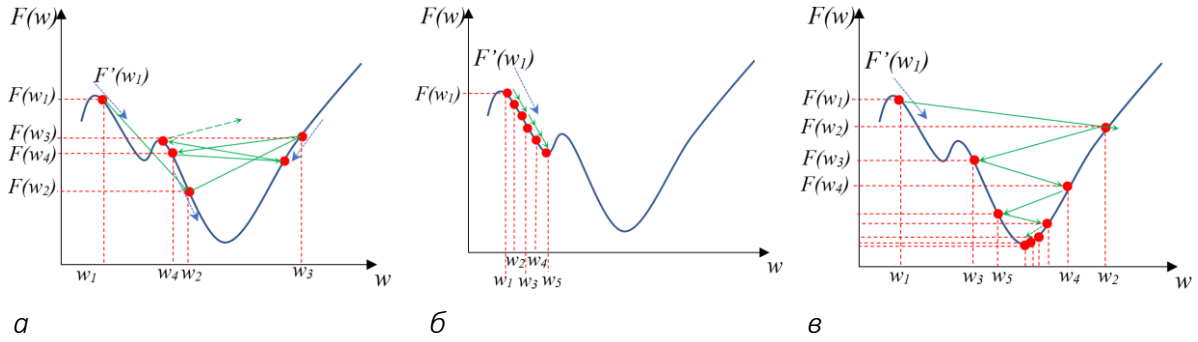


Рисунок 3.1. – Принцип нахождения минимума функции методом градиентного спуска

Метод градиентного спуска предполагает, что выбирается случайная точка, находится направление самого быстрого убывания функции и движется до ближайшего минимума вдоль этого направления (см. рисунок 3.1). Если функция в выбранной точке убывает, то вес следует увеличить, если функция возрастает – то уменьшить. Шаг этого изменения носит название *скорость обучения* и определяет, насколько будет сдвигаться точка на функции потерь.

Настройка скорости обучения так же очень важный процесс: если установить значение скорости обучения слишком большим, может быть вероятность того, что минимум вообще не будет достигнут, т. к. алгоритм всегда будет перешагивать его, а градиент разворачивать обратно (рисунок 3.2, *а*), если скорость будет очень малой величиной, то модель будет обучаться очень долго, и обучение может не завершиться, т. к. есть вероятность остановиться в локальном минимуме, где значение функции потерь будет неудовлетворительным (рисунок 3.2, *б*). Наиболее эффективным решением является подход, когда скорость обучения меняется по мере обучения (рисунок 3.2, *в*):



а *б* *в*

а – большая скорость обучения; *б* – низкая скорость обучения;
в – скорость обучения изменяется в процессе обучения

Рисунок 3.2. – Метод градиентного спуска с разной скоростью обучения:
$F(w)$ – значение функции потерь (величина ошибки),
$F'(w)$ – частная производная в точке, w – вес

Пример обучения ИНС с использованием метода обратного распространения ошибки

Рассмотрим пример обучения ИНС, состоящей из скрытого и выходного слоя, каждый из которых включает три нейрона (рисунок 3.3) [17]. Начальные веса инициализируются случайным образом.

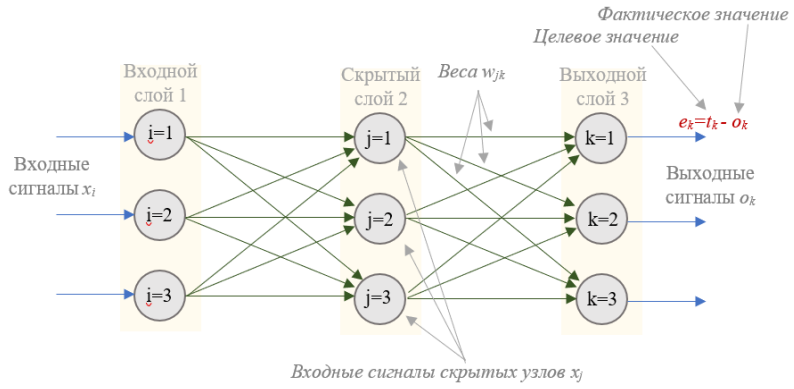


Рисунок 3.3. – Трехслойная полносвязная ИНС

Введем следующие обозначения: $w_{i,j}$ – веса между входным и скрытым слоями, $w_{j,k}$ – веса между скрытым и выходным слоями. Матрица весов представляется в виде

$$W = \begin{pmatrix} w_{1,1} & w_{2,1} & w_{3,1} \\ w_{1,2} & w_{2,2} & w_{3,2} \\ w_{1,3} & w_{2,3} & w_{3,3} \end{pmatrix}.$$

Допустим, на входе сети присутствуют данные $S = (0{,}9;\ 0{,}1;\ 0{,}8)^{\mathrm{T}}$. Веса зададим случайным образом:

$$W_{вх\_скр} = \begin{pmatrix} 0{,}9 & 0{,}3 & 0{,}4 \\ 0{,}2 & 0{,}8 & 0{,}2 \\ 0{,}1 & 0{,}5 & 0{,}6 \end{pmatrix}, \qquad W_{скр\_вых} = \begin{pmatrix} 0{,}3 & 0{,}7 & 0{,}5 \\ 0{,}6 & 0{,}5 & 0{,}2 \\ 0{,}8 & 0{,}1 & 0{,}9 \end{pmatrix}.$$

Метод обратного распространения ошибки предполагает, что сначала сигнал распространяется в прямом направлении. Входной сигнал для каждого нейрона на каждом слое

$$X = W \cdot S. \tag{3.1}$$

$$X_{\text{скр}} = \begin{pmatrix} 0{,}9 & 0{,}3 & 0{,}4 \\ 0{,}2 & 0{,}8 & 0{,}2 \\ 0{,}1 & 0{,}5 & 0{,}6 \end{pmatrix} \cdot \begin{pmatrix} 0{,}9 \\ 0{,}1 \\ 0{,}8 \end{pmatrix} = \begin{pmatrix} 1{,}16 \\ 0{,}42 \\ 0{,}62 \end{pmatrix}.$$

Таким образом, на вход скрытого слоя будут поступать значения 1,16; 0,42; 0,62.

К данным, поступающим на вход скрытого слоя, необходимо применить функцию активации, например, функцию сигмоиды $y = \dfrac{1}{1+e^{-x}}$. В результате на выходе скрытого слоя будут получены значения

$$Y_{\text{скр}} = sigmoid \begin{pmatrix} 1{,}16 \\ 0{,}42 \\ 0{,}62 \end{pmatrix} = \begin{pmatrix} \dfrac{1}{1+e^{-1{,}16}} \\ \dfrac{1}{1+e^{-0{,}42}} \\ \dfrac{1}{1+e^{-0{,}62}} \end{pmatrix} = \begin{pmatrix} 0{,}761 \\ 0{,}603 \\ 0{,}650 \end{pmatrix}.$$

По формуле (3.1) рассчитаем значения для выходного слоя:

$$X_{\text{вых}} = \begin{pmatrix} 0,3 & 0,7 & 0,5 \\ 0,6 & 0,5 & 0,2 \\ 0,8 & 0,1 & 0,9 \end{pmatrix} \cdot \begin{pmatrix} 0,761 \\ 0,603 \\ 0,650 \end{pmatrix} = \begin{pmatrix} 0,975 \\ 0,888 \\ 1,254 \end{pmatrix}.$$

После применения функции активации получаем значения на выходе нейронной сети (рисунок 3.4)

$$Y_{\text{вых}} = sigmoid \begin{pmatrix} 0,975 \\ 0,888 \\ 1,254 \end{pmatrix} = \begin{pmatrix} \frac{1}{1+e^{-0,975}} \\ \frac{1}{1+e^{-0,888}} \\ \frac{1}{1+e^{-1,254}} \end{pmatrix} = \begin{pmatrix} 0,726 \\ 0,708 \\ 0,778 \end{pmatrix}.$$

После применения функции активации получаем значения на выходе нейронной сети (см. рисунок 3.4)

$$Y_{\text{вых}} = sigmoid \begin{pmatrix} 0,975 \\ 0,888 \\ 1,254 \end{pmatrix} = \begin{pmatrix} \frac{1}{1+e^{-0,975}} \\ \frac{1}{1+e^{-0,888}} \\ \frac{1}{1+e^{-1,254}} \end{pmatrix} = \begin{pmatrix} 0,726 \\ 0,708 \\ 0,778 \end{pmatrix}.$$

Таким образом входные данные распространяются по сети в прямом направлении. Следующий шаг заключается в сравнении выходного сигнала с входными сигналами и вычислении ошибки с помощью функции потерь. Ошибка будет оказывать влияние на вес каждого входящего в нейрон синапса пропорционально его вкладу.

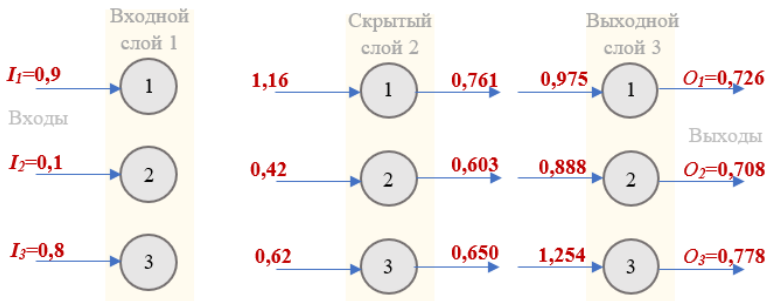


Рисунок 3.4. – Результат работы ИНС при случайной инициализации весов

В качестве функции потерь *E* будем использовать квадрат разности полученного результата и желаемого. Рассчитаем функцию потерь, которая представляет собой сумму возведенных в квадрат разностей между целевым и фактическим (выходным) значениями, где суммирование осуществлялось по всем *n* узлам:

$$\frac{\partial E}{\partial w_{jk}} = \frac{\partial}{\partial w_{jk}} \sum_n (t_n - y_n)^2 .$$

Данное выражение можно упростить, исходя из того, что выходной сигнал y_n на узле *n* зависит лишь от весов w_{jk}, которые с ним соединены. Тогда

$$\frac{\partial E}{\partial w_{jk}} = \frac{\partial}{\partial w_{jk}} (t_k - y_k)^2 .$$

В связи с тем, что целевое значение t_k – константа, то от w_{jk} зависит только фактическое значение y_k, и тогда

$$\frac{\partial E}{\partial w_{jk}} = \frac{\partial E}{\partial y_k} \cdot \frac{\partial y_k}{\partial w_{jk}} = -2(t_k - y_k) \cdot \frac{\partial y_k}{\partial w_{jk}},$$

где y_k – выходной сигнал узла предыдущего слоя, который, как рассматривалось выше, был получен в результате применения функции активации сигмоиды. Соответственно,

$$\frac{\partial E}{\partial w_{jk}} = -2(t_k - y_k) \cdot \frac{\partial}{\partial w_{jk}} sigmoid(\sum_j w_{jk} \cdot y_j).$$

Так как $$\frac{\partial}{\partial x} simoid(x) = sigmoid(x)(1 - sigmoid(x)),$$

то $$\frac{\partial E}{\partial w_{jk}} = -2(t_k - y_k) \cdot sigmoid(\sum_j w_{jk} \cdot y_j)(1 - sigmoid(\sum_j w_{jk} \cdot y_j)) \cdot \frac{\partial}{\partial w_{jk}} (\sum_j w_{jk} \cdot y_j) =$$

$$= -2(t_k - y_k) \cdot sigmoid(\sum_j w_{jk} \cdot y_j)(1 - sigmoid(\sum_j w_{jk} \cdot y_j)) \cdot y_j .$$

В связи с тем, что интерес представляет только направление градиента, то скалярный множитель два можно игнорировать. Тогда окончательное выражение, которое будет использоваться для изменения веса w_{jk},

$$\frac{\partial E}{\partial w_{jk}} = -(t_k - y_k) \cdot sigmoid(\sum\nolimits_j w_{jk} \cdot y_j)(1 - sigmoid(\sum\nolimits_j w_{jk} \cdot y_j)) \cdot y_j . \qquad (3.2)$$

Полученное выражение, по сути, является градиентом функции потерь. На его основе могут быть получены новые значения весовых коэффициентов w_{jk}^{v}:

$$w_{jk}^{v} = w_{jk}^{v-1} - \alpha \cdot \frac{\partial E}{\partial w_{jk}},$$

где w_{jk}^{v-1} – весовые коэффициенты предыдущего шага прямого прохода ИНС;

α – скорость обучения, знак «минус» перед поправкой позволяет увеличивать вес при отрицательной производной и уменьшать при положительной.

Аналогичным образом изменение весов выполняется на каждом слое. Одним из способов вычисления ошибки на скрытом слое ($e_{скр}$) является распределение ошибок выходного слоя ($e_{вых}$) между соответствующими связями пропорционально весу каждой связи с последующим объединением соответствующих разрозненных частей ошибки на каждом внутреннем узле:

$$e_{скр} = w^{T}{}_{скр\_вых} \cdot e_{вых}, \quad (3.3)$$

где $W^{T}{}_{скр\_вых}$ – транспонированная матрица весов между скрытым и выходным слоем.

Рассчитаем новые весовые коэффициенты между скрытым и выходным слоями. В выражении (3.2) величина ошибки $(t_k - y_k)$ для выходного слоя будет равна

$$e_{вых} = \begin{pmatrix} 0{,}9 - 0{,}726 \\ 0{,}1 - 0{,}708 \\ 0{,}8 - 0{,}778 \end{pmatrix} = \begin{pmatrix} 0{,}174 \\ -0{,}608 \\ 0{,}022 \end{pmatrix}.$$

Второй множитель выражения (3.2) $sigmoid(\sum_j w_{jk} \cdot y_j)$ будет равен выходу сети:

$$sigmoid\begin{pmatrix} 0,3 & 0,7 & 0,5 \\ 0,6 & 0,5 & 0,2 \\ 0,8 & 0,1 & 0,9 \end{pmatrix} \cdot \begin{pmatrix} 0,761 \\ 0,603 \\ 0,650 \end{pmatrix} = \begin{pmatrix} \frac{1}{1+e^{-0,975}} \\ \frac{1}{1+e^{0,888}} \\ \frac{1}{1+e^{-1,254}} \end{pmatrix} = \begin{pmatrix} 0,726 \\ 0,708 \\ 0,778 \end{pmatrix}.$$

Третий множитель выражения (3.2) равен

$$1 - sigmoid(\sum_j w_{jk} \cdot y_j) = \begin{pmatrix} 0,274 \\ 0,292 \\ 0,222 \end{pmatrix}.$$

Перемножив соответствующие элементы матриц согласно выражению (3.2), получим

$$\frac{\partial E}{\partial w_{jk}} = \begin{pmatrix} -0,026 \\ 0,076 \\ -0,002 \end{pmatrix}.$$

Тогда новые весовые коэффициенты между скрытым и выходным слоями по выражению (3.3), если скорость обучения $\alpha = 0,5$

$$w_{jk}^{v} = \begin{pmatrix} 0,3+0,5 \cdot 0,026 & 0,7+0,5 \cdot 0,026 & 0,5+0,5 \cdot 0,026 \\ 0,6-0,5 \cdot 0,076 & 0,5-0,5 \cdot 0,076 & 0,2-0,5 \cdot 0,076 \\ 0,8+0,5 \cdot 0,002 & 0,1+0,5 \cdot 0,002 & 0,9+0,5 \cdot 0,002 \end{pmatrix} =$$

$$= \begin{pmatrix} 0,313 & 0,713 & 0,513 \\ 0,562 & 0,462 & 0,162 \\ 0,801 & 0,101 & 0,901 \end{pmatrix}.$$

Рассчитаем новые весовые коэффициенты между входным и скрытым слоями. Ошибка на скрытом слое

$$e_{\text{скр}} = \begin{pmatrix} 0,3 & 0,7 & 0,5 \\ 0,6 & 0,5 & 0,2 \\ 0,8 & 0,1 & 0,9 \end{pmatrix}^{T} \cdot \begin{pmatrix} 0,174 \\ -0,608 \\ 0,022 \end{pmatrix} = \begin{pmatrix} 0,3 & 0,6 & 0,8 \\ 0,7 & 0,5 & 0,1 \\ 0,5 & 0,2 & 0,9 \end{pmatrix} \cdot \begin{pmatrix} 0,174 \\ -0,608 \\ 0,022 \end{pmatrix} = \begin{pmatrix} -0,295 \\ -0,18 \\ -0,0148 \end{pmatrix}.$$

Второй множитель выражения (3.2) $sigmoid(\sum_j w_{jk} \cdot y_j)$ будет равен выходу скрытого слоя:

$$sigmoid\begin{pmatrix} 0,9 & 0,3 & 0,4 \\ 0,2 & 0,8 & 0,2 \\ 0,1 & 0,5 & 0,6 \end{pmatrix} \cdot \begin{pmatrix} 0,9 \\ 0,1 \\ 0,8 \end{pmatrix} = \begin{pmatrix} \frac{1}{1+e^{-1,16}} \\ \frac{1}{1+e^{-0,42}} \\ \frac{1}{1+e^{-0,62}} \end{pmatrix} = \begin{pmatrix} 0,761 \\ 0,603 \\ 0,650 \end{pmatrix}.$$

Третий множитель выражения (3.2) равен

$$1 - sigmoid(\sum_j w_{jk} \cdot y_j) = \begin{pmatrix} 0,239 \\ 0,397 \\ 0,35 \end{pmatrix},$$

тогда

$$\frac{\partial E}{\partial w_{jk}} = \begin{pmatrix} 0,048 \\ 0,004 \\ 0,003 \end{pmatrix}.$$

И новые весовые коэффициенты между входным и скрытым слоями по выражению (3), если скорость обучения $\alpha = 0,5$

$$w_{jk}^{v} = \begin{pmatrix} 0,9-0,5\cdot 0,048 & 0,3-0,5\cdot 0,048 & 0,4-0,5\cdot 0,048 \\ 0,2-0,5\cdot 0,004 & 0,8-0,5\cdot 0,004 & 0,2-0,5\cdot 0,004 \\ 0,1-0,5\cdot 0,003 & 0,5-0,5\cdot 0,003 & 0,6-0,5\cdot 0,003 \end{pmatrix} =$$

$$= \begin{pmatrix} 0,876 & 0,276 & 0,376 \\ 0,198 & 0,798 & 0,198 \\ 0,0985 & 0,4985 & 0,5985 \end{pmatrix}.$$

Как можно видеть, после одного прохода по сети веса изменяются довольно незначительно, однако после выполнения тысяч итераций весовые коэффициенты образуют устойчивую конфигурацию, которая позволит получать на выходе сети значения, соответствующие входным.

Выполним еще один прямой проход по сети. Входные значения скрытого слоя

$$X_{\text{скр}} = \begin{pmatrix} 0{,}876 & 0{,}276 & 0{,}376 \\ 0{,}198 & 0{,}798 & 0{,}198 \\ 0{,}0985 & 0{,}4985 & 0{,}5985 \end{pmatrix} \cdot \begin{pmatrix} 0{,}9 \\ 0{,}1 \\ 0{,}8 \end{pmatrix} = \begin{pmatrix} 0{,}1168 \\ 0{,}4164 \\ 0{,}6173 \end{pmatrix}.$$

После функции активации

$$Y_{\text{скр}} = sigmoid \begin{pmatrix} 1{,}1168 \\ 0{,}4164 \\ 0{,}6173 \end{pmatrix} = \begin{pmatrix} \dfrac{1}{1+e^{-1{,}1168}} \\ \dfrac{1}{1+e^{-0{,}4164}} \\ \dfrac{1}{1+e^{-0{,}6173}} \end{pmatrix} = \begin{pmatrix} 0{,}7534 \\ 0{,}6026 \\ 0{,}6496 \end{pmatrix}.$$

Для выходного слоя

$$X_{\text{вых}} = \begin{pmatrix} 0{,}313 & 0{,}713 & 0{,}513 \\ 0{,}562 & 0{,}462 & 0{,}162 \\ 0{,}801 & 0{,}101 & 0{,}901 \end{pmatrix} \cdot \begin{pmatrix} 0{,}7534 \\ 0{,}6026 \\ 0{,}6496 \end{pmatrix} = \begin{pmatrix} 0{,}9987 \\ 0{,}807 \\ 1{,}2496 \end{pmatrix}.$$

После применения функции активации получаем значения на выходе нейронной сети

$$Y_{\text{вых}} = sigmoid \begin{pmatrix} 0{,}9987 \\ 0{,}807 \\ 1{,}2496 \end{pmatrix} = \begin{pmatrix} \dfrac{1}{1+e^{-0{,}9987}} \\ \dfrac{1}{1+e^{-0{,}807}} \\ \dfrac{1}{1+e^{-1{,}2496}} \end{pmatrix} = \begin{pmatrix} 0{,}7308 \\ 0{,}6914 \\ 0{,}7772 \end{pmatrix}.$$

Если сравнить значения на выходе сети после первой и второй итерации, то можно заметить, что выходное значение первого нейрона немного увеличилось, а для второго нейрона немного уменьшилось, что говорит о верном направлении изменения весов и медленно приближает к целевому значению. Выходное значение третьего нейрона изначально было достаточно близким к целевому, и тут возможны колебания как в сторону увеличения

точности, так и в сторону небольшого снижения в процессе обучения, что произошло на текущей итерации, однако при дальнейшей корректировке весов на следующих итерациях это будет исправлено.

Рассмотренная в примере функция активации *sigmoid* нормализует значения нейронов в диапазоне от 0 до 1. В общем случае нормализация позволяет привести все выходные значения к одному масштабу и распределить их в диапазоне от [–1, 1] или [0, 1], что позволяет ускорить вычисления за счет того, что числа будут одного порядка. Кроме этого, ненормализованные данные приводят к неравномерному распределению весов, и часть из них будет иметь очень маленькие значения, а другая часть – очень большие, что будет приводить к тому, что прежде, чем алгоритм градиентного спуска сможет приблизиться к минимуму функции потерь, будут наблюдаться большие колебания в области плато.

Однако большинство функций активации не обладает свойством нормализации, и для того, чтобы привести все числовые значения к одному масштабу, между выходными значениями слоя и функцией активации вводится пакетная нормализация (batch normalization).

Суть пакетной нормализации заключается в следующем: пусть в функцию активации поступает вектор размерности d: $x=\left(x(1),...,x(d)\right)$. Его нормализация по каждой из размерностей будет

$$\hat{x}^{(k)}=\frac{x^{(k)}-E\left(x^{(k)}\right)}{\sqrt{D\left(x^{(k)}\right)}},$$

где $E(x)$ – математическое ожидание;

$D(x)$ – дисперсия, которые вычисляются по всей обучающей выборке.

Применение пакетной нормализации позволяет получить нормальное распределение на каждом слое и ускорить обучение модели.

Для обучения предлагается использовать сверточную нейронную сеть YOLOv4-tiny, которая является упрощенной моделью YOLOv4, что позволит сократить временные затраты на обучение. Схема YOLOv4-tiny представлена на рисунке 3.5. YOLOv4-tiny также использует межэтапные соединения и остаточные блоки, однако в меньшем количестве. Другим отличием является обнаружение объектов только в двух масштабах (N/32 и N/16, где N – размер входного слоя) на 30-м и 37-м слое. В качестве функции активации после

сверточных слоев используется функция Leake-ReLU, функция потерь – CIOU (см. лабораторную работу № 2).

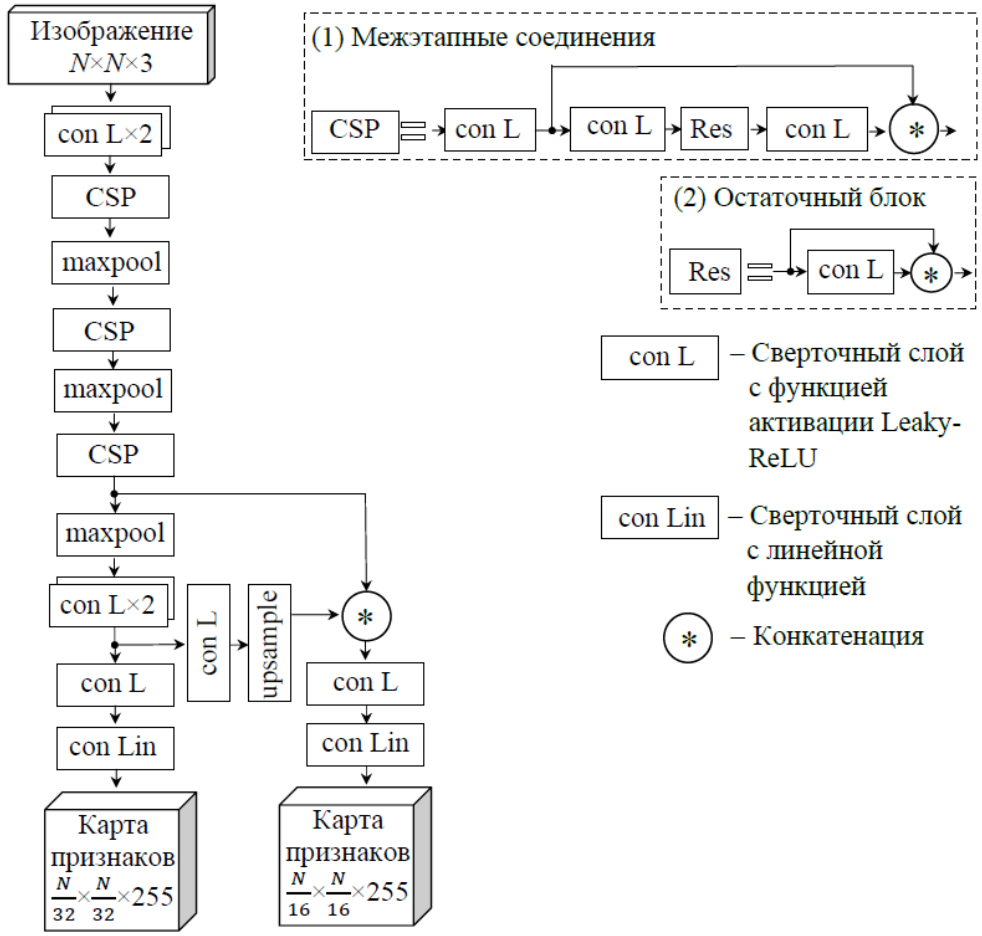


Рисунок 3.5. – Схема сверточной нейронной сети YOLOv4-tiny[2]

Задания к лабораторной работе

1. *Сформировать набор данных для обучения, включающий объекты согласно варианту заданий.*

В качестве примера рассмотрим создание базы данных изображений для обучения сверточной нейронной сети YOLOv4 для детектирования объектов трех классов: «подарок», «мышь» и «чайник». При формировании набора данных следует ориентироваться на следующие правила:

– при разметке изображений убедитесь, что каждый объект правильно размечен и ни один не пропущен. Большинство проблем с обучением связано с некорректно размеченными данными для обучения;

[2] https://netron.app/?url=https://raw.githubusercontent.com/AlexeyAB/darknet/master/cfg/yolov4-tiny.cfg

– для каждого объекта, который следует обнаруживать в обучающей выборке должен быть хотя бы один пример, содержащий похожий объект (по размеру, форме, цвету, углу поворота и т. д.);

– набор данных должен включать изображения объектов с разных сторон, в разных масштабах и на разном фоне;

– следует учитывать размер объектов на тренировочной и тестовой выборке: если в большинстве примеров объект занимает 80–90% изображения, то такой объект не будет обнаруживаться, если будет занимать 1–10% изображения.

Пример изображений для обучения показан на рисунке 3.6.

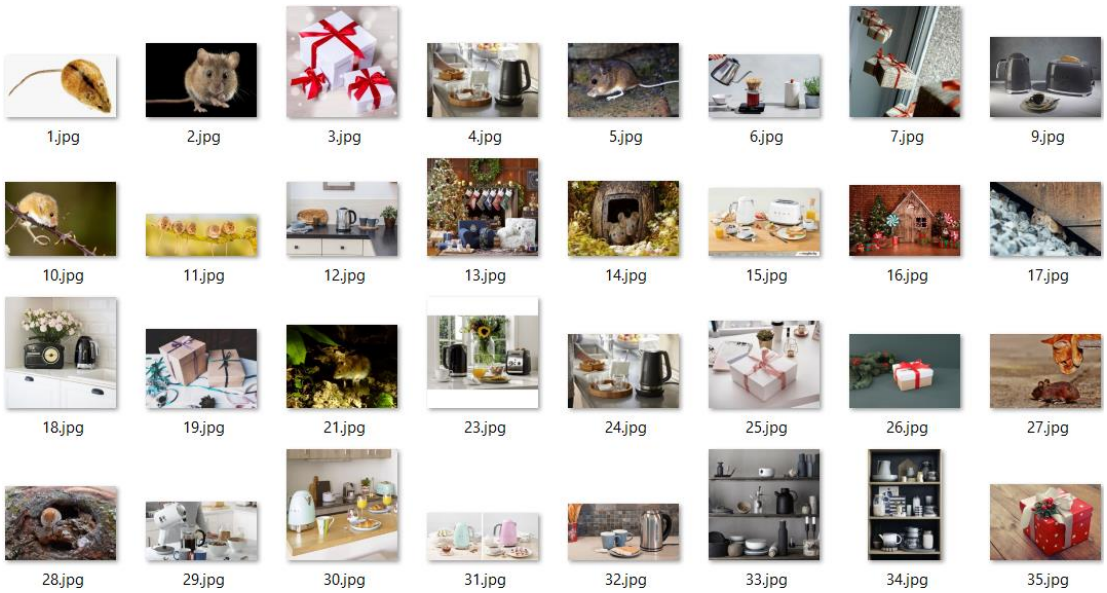


Рисунок 3.6. – Пример изображений для обучения

Варианты заданий:

Вариант 1: Бант, кастрюля, яблоко.

Вариант 2: Гербера, собака, кружка.

Вариант 3: Ноутбук, птица, клубника.

Вариант 4: Клубника, вилка, кот.

Вариант 5: Лошадь, велосипед, ромашка.

Вариант 6: Гриб, ананас, кувшин.

Вариант 7: Огурец, тарелка, гербера.

Вариант 8: Кувшин, шляпа, лошадь.

Вариант 9: Лилия, птица, земляника.

Вариант 10: Роза, мотоцикл, огурец.

Вариант 11: Улитка, яблоко, ведро.

Вариант 12: Роза, карандаш, банан.
Вариант 13: Бабочка, ежевика, велосипед.
Вариант 14: Гриб, улитка, метла.
Вариант 15: Лилия, бант, еж.

Разметка ограничительных рамок в ОС Windows

Репозиторий с графическим интерфейсом для разметки ограничительных рамок объектов и создания файлов аннотаций для YOLOv4: https://github.com/AlexeyAB/Yolo_mark.

Клонирование репозитория на компьютер (рисунок 3.7):

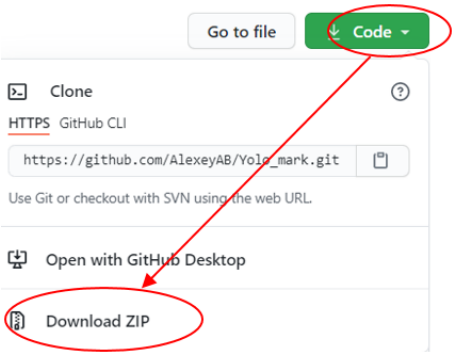


Рисунок 3.7. – Перенос репозитория на компьютер

Открыть `yolo_mark.sln` в MS Visual Studio. Далее перейти во вкладку «Проект» → «Свойства: yolo_mark.sln» → C/C++ → «Общие» → «Дополнительные каталоги включаемых файлов» и «Дополнительные каталоги» и указать путь к OpenCV, установленному компьютере пользователя (рисунок 3.8). Затем выполнить компиляцию в x64&Release (рисунок 3.9).

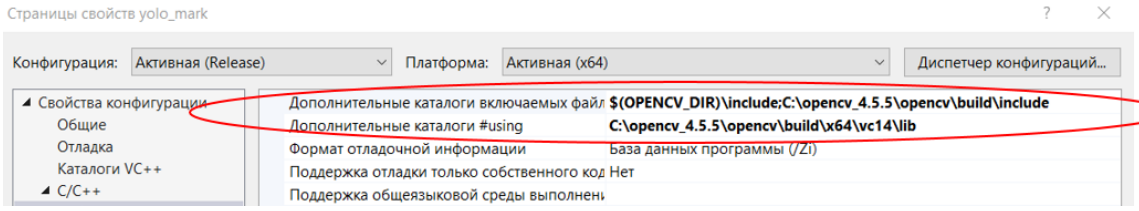


Рисунок 3.8. – Подключение OpenCV

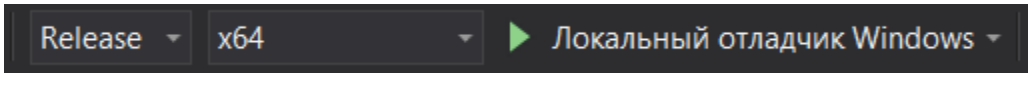


Рисунок 3.9. – Выбор опций для компиляции в x64&Release

Для маркировки изображений необходимо выполнить действия:
– удалить все файлы из каталога x64/Release/data/img;

– поместить свои изображения .jpg в каталог x64/Release/data/img;

– изменить количество классов объектов в файле x64/Release/data/obj.data, указав число 3;

```
classes= 3
train  = data/train.txt
valid  = data/train.txt
names = data/obj.names
backup = backup/
```

– редактировать файл x64/Release/data/obj.names в соответствии с именами классов изображений (рисунок 3.10);

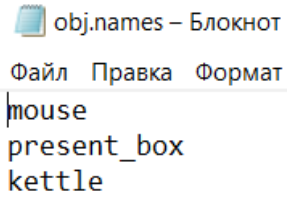


Рисунок 3.10. – Отображение содержимого файла obj.names

– запустить файл x64/Release/yolo_mark.cmd;

– для каждого изображения необходимо выбрать class id и поместить в прямоугольник соответствующий объект, после чего перейти к следующему изображению. Пример разметки объекта показан на рисунок 3.11.

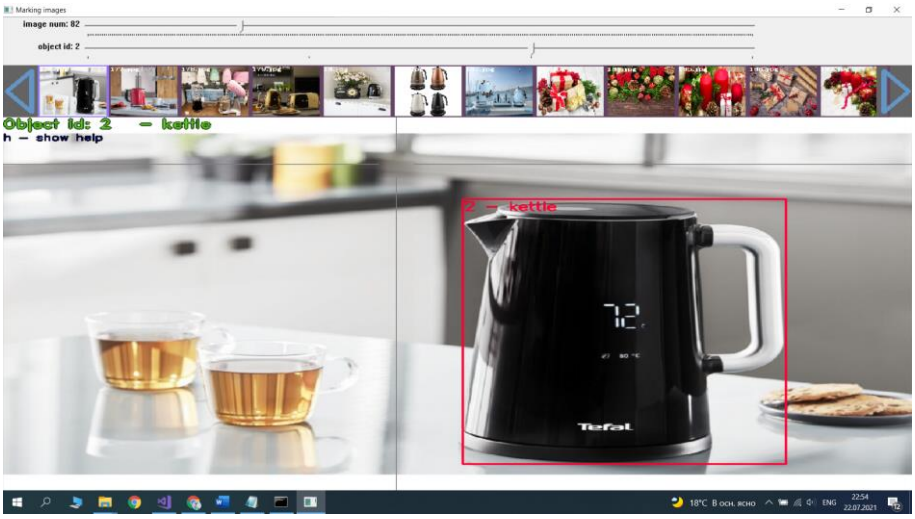


Рисунок 3.11. – Разметка объекта класса «чайник»

В результате работы программы для каждого размеченного изображения будет создан отдельный .txt файл с координатами выбранных объектов, и один файл train.txt, содержащий адреса расположения всех рассмотренных изображений.

Таблица 3.3. – Назначение клавиш клавиатуры для разметки изображений

Клавиша	Действие
	Следующее изображение
	Предыдущее изображение
	Удалить выделенное поле (при наведении курсора)
	Удалить все отметки на текущем изображении
	Копировать предыдущую отметку
	Следить за объектами
	Закрыть приложение
	Один объект на изображении
	Идентификатор объекта
	Показать координаты
	Ширина линии
	Скрыть название объекта
	Помощь

2. *Обучение СНС YOLOv4-tiny*

Для возможности запуска СНС YOLO необходимо установить фреймворк Darknet одним из рассмотренных ниже способов. Для этого нужно загрузить архив Darknet с официального репозитория (https://github.com/AlexeyAB/darknet) и распаковать его.

Одним из способов компиляции Darknet является использование утилиты CMake. В CMake заполняется поле адреса расположения фреймворка Darknet и параметры для сборки проекта. На рисунке 3.12 цифрами обозначена последовательность действий.

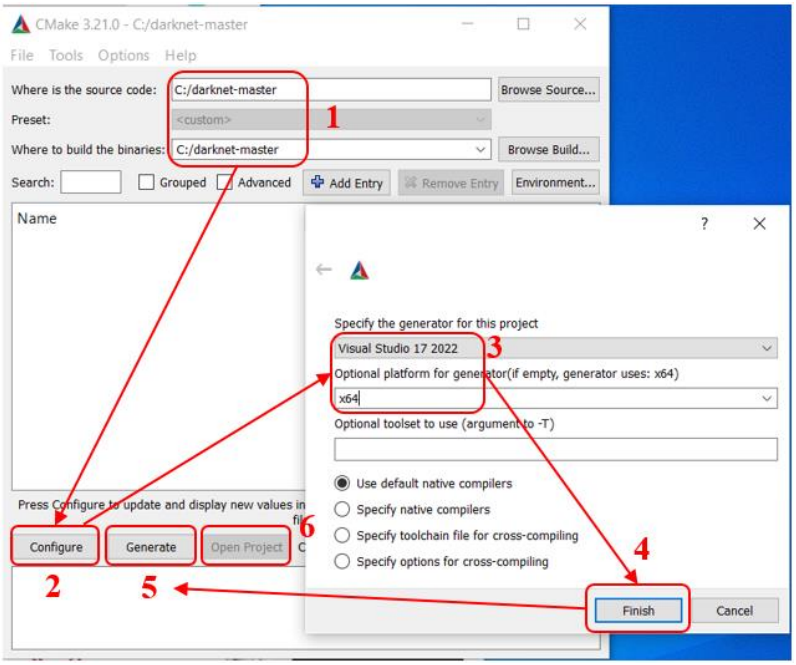


Рисунок 3.12. – Настройки для в CMake для компиляции фреймворка darknet

После нажатия Open Project откроется окно MS Visual Studio, где необходимо выбрать Release и x64, после чего выполнить сборку (рисунок 3.13).

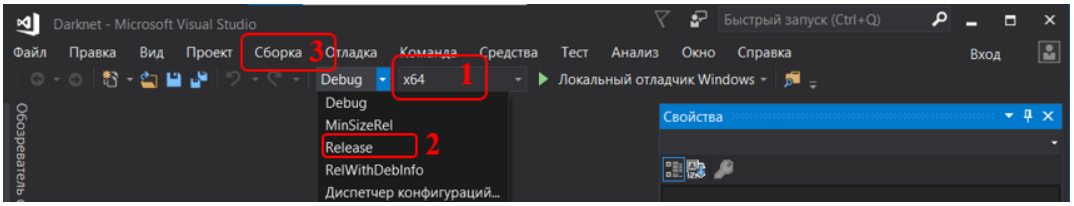


Рисунок 3.13. – Порядок действий в MS Visual Studio

При наличии видеокарты Nvidia с поддержкой CUDA можно использовать следующий способ компиляции Darknet:

1) установить MS Visual Studio;

2) установить CUDA, включив интеграцию с MSVS во время установки. Пояснения по установке CUDA и cuDNN приведены в приложении 3.1;

3) скачать и распаковать Darknet;

4) открыть с помощью приложения «Блокнот» файл `build/darknet/darknet.vcxproj`;

5) заменить следующие строки на значения, соответствующие пользовательским установкам и параметрам компьютера пользователя:

– `C:/opencv_2.4.9/opencv/...` на `C:/opencv_4.5.5/opencv/...` (или другую версию OpenCV, установленную на компьютере пользователя);

– `CUDA 11.1.props` … на `CUDA 11.4.props` (или другую версию CUDA, установленную на компьютере пользователя);

– `CUDA 11.1.targets` … на `CUDA 11.4.targets` (или другую версию CUDA, установленную на компьютере пользователя);

– `compute_30, sm30` на `compute_86, sm86` (проверить подходящее для пользователя значение можно по ссылке http://arnon.dk/matching-sm-architectures-arch-and-gencode-for-various-nvidia-cards/).

6) сохранить изменения;
7) запустить Build/darknet/darknet.sln с помощью MS VS;
8) в настройки проекта добавить:
   – *Include path:*
     C:\opencv_4.5.5\opencv\build\include;
     C:\opencv_4.5.5\opencv\sources\3rdparty\include;
     $(CUDA_PATH)\include;
   – *Library path:*
     C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v11.4\lib\x64;
     $(CUDA_PATH)\lib\$(PlatformName);
     C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v8.0\lib\x64;
   – *Linker input:*
     ..\..\3rdparty\lib\x64\pthreadVC2.lib;cublas.lib;curand.lib;cudart.lib;
9) выполнить сборку (см. рисунок 3.13).

Возможно возникновение ошибки, как показано на рисунке 3.14.

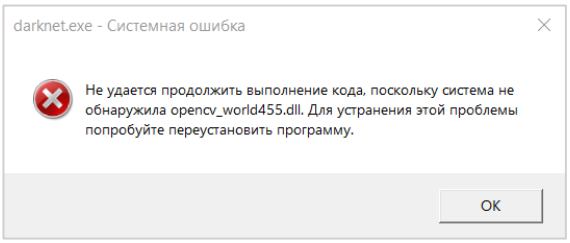


Рисунок 3.14. – Отображение системной ошибки при сборке фреймворка darknet

При такой ошибке следует скопировать файлы «opencv_world455.dll» и «opencv_world455d.dll» из «C:\opencv_4.5.5\opencv\build\x64\vc14\bin» в «C:\Windows\System32». Другие способы установки рассмотрены на веб-ресурсе https://github.com/AlexeyAB/darknet.

Для обучения yolov4-tiny.cfg необходимо выполнить следующую последовательность действий:

1) создать в каталоге /darknet-master/build/darknet/x64 файл yolov4-tiny-obj.cfg с тем же содержимым, что и файл yolov4-tiny.cfg;

2) для обучения СНС обнаружению заданного количества классов файл .cfg необходимо отредактировать:

– изменить размер пакета: `batch = 64` и `subdivision = 16`;

– изменить строку `max_batches` на `classes*2000`, но не менее количества обучающих образцов и не менее 6000. Например, если количество классов = 3, то `max_batches = 6000`;

– изменить строку step на значение, составляющее 80% и 90% от max_batches, т. е. `step = 4800, 5400`;

– размер входного слоя: `weight = 416`, `height = 416` или любое значение, кратное 32.

– измените строку `classes = 80` на ваше значение числа классов для каждого из двух слоев обнаружения `yolo`;

– для каждого из двух слоев, предшествующих слою `yolo`, необходимо изменить количество фильтров с `filters = 255` на значение, равное `(classes + 5) × 3`. Для трех классов значение `filters = 24`;

3) скачать файл предобученных весов для сверточных слоев и поместить их в каталог darknet-master/build/darknet/x64:

– для СНС yolov4-tiny.cfg: https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v4_pre/yolov4-tiny.conv.29;

4) созданные ранее файлы `obj.data` и `obj.names` скопировать в каталог /darknet-master/build/darknet/x64/data;

5) созданный ранее набор данных переместить в каталог /darknet-master/build/darknet/x64/data/img вместе с .txt-файлами, созданными при разметке;

6) созданный при разметке файл `train.txt` переместить в каталог /darknet-master/build/darknet/x64/data;

7) начать обучение командной строки (рисунок 3.15):

```
darknet.exe detector train data/obj.data yolov4-
tiny-obj.cfg yolov4-tiny.conv.29 -gpus 0.
```

`-gpus 0` обеспечивает обучение на GPU. Если используется более одной видеокарты, то первую 1000 итераций обучения необходимо выполнить на первой GPU, затем остановить обучение и продолжить его на уже частично обученной модели с использованием нескольких видеокарт: `darknet.exe detector train data/obj.data yolov4-tiny-obj.cfg yolov4-tiny.conv.29 -gpus 0,1,2,3`.

```
C:\darknet-master\build\darknet\x64>darknet.exe detector train data/obj.data yolov4-tiny-obj.cfg yolov4-tiny.conv.29 -gpus 0
```

Рисунок 3.15. – Командная строка для начала обучения

При обучении на CPU «`-gpus 0`» не указывать;

8) если возникает ошибка нехватки памяти, то следует уменьшить размер пакета или увеличить Subdivision до 32 или 64, но не более, чем размер пакета. Пример окна командной строки при обучении показан на рисунке 3.16.

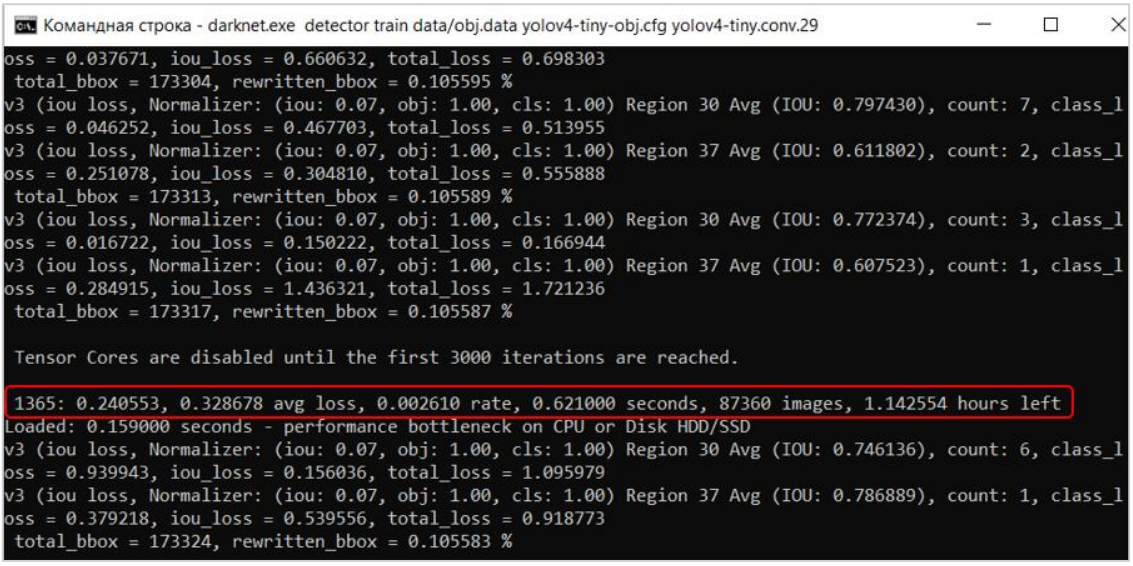


Рисунок 3.16. – Командная строка при обучении: 1365 – текущая итерация / пакет обучения; 0.240553 – общая потеря; 0.328678 – среднее значение потерь (avg loss); 0.002610 – текущая скорость обучения; 0.621000 – общее время, затраченное на обработку пакета; 87360 – общее количество изображений, обработанное к текущему моменту

Среднее значение потерь, должно быть как можно меньше. Хорошим результатом считается, если это значение меньше чем 0,05. Однако при сложном наборе данных и большой модели это значение может достигать значения 3.

Кроме этого, отображается .png-файл, отражающий график потерь при обучении. Пример графика по окончании обучения показан на рисунке 3.17. Для итерации с номером 5864 получено среднее значение потерь avg loss = 0,0661, что является удовлетворительным результатом.

Другие параметры, с которыми было завершено обучение для данного примера, отражены в командной строке (рисунок 3.18). Из рисунка видно, что среднее значение потерь уменьшилось до avg loss = 0,063014.

Для обнаружения объектов заданных классов с помощью обученной СНС можно использовать программный код из лабораторной работы № 2 или применить командную строку (рисунок 3.19).

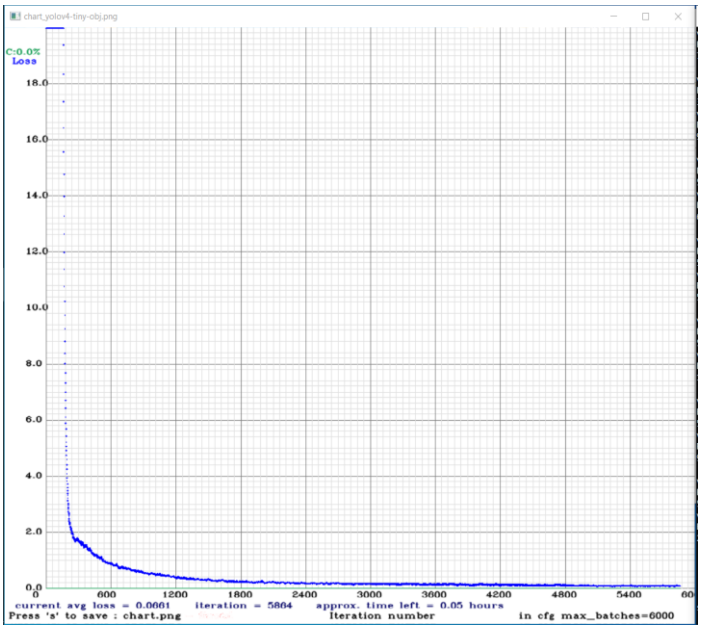


Рисунок 3.17. – График потерь при обучении СНС

```
 6000: 0.055966, 0.063014 avg loss, 0.000026 rate, 0.607000 seconds, 384000 images, 0.022319 hours left
Saving weights to backup//yolov4-tiny-obj_6000.weights
Saving weights to backup//yolov4-tiny-obj_last.weights
Saving weights to backup//yolov4-tiny-obj_final.weights
If you want to train from the beginning, then use flag in the end of training command: -clear
```

Рисунок 3.18. – Отображение достигнутых параметров обучения СНС в командной строке

```
C:\darknet-master\build\darknet\x64>darknet.exe detector test data/obj.data yolov4-tiny-obj.cfg backup-tiny/yolov4-tiny-obj_final.weights data/3.jpg
```

Рисунок 3.19. – Командная строка для обнаружения объектов на изображении с использованием обученной СНС

Для тестирования работы обученной СНС следует выбирать изображения, которые не входили в обучающую выборку.

Обучение сверточной нейронной сети YOLOv4-tiny с помощью облачного сервиса Google Colab

Google Colab – бесплатная интерактивная облачная среда для работы с кодом, которая предоставляет мощные графические процессоры GPU, а также тензорные процессоры TPU, позволяющие проводить сложные вычисления, в том числе и в области машинного обучения. Для работы в виртуальной среде Google Colab необходимо иметь учетную запись Google Cloud Disk: https://drive.google.com. При работе с Google Colab следует помнить, что после перезапуска среды загруженные файлы могут быть удалены.

Обучение СНС YOLOv4-tiny с использованием облачного сервиса Google Colab требует выполнения следующих шагов:

1) открыть Google Cloud Disk, затем с помощью правой кнопки мыши открыть всплывающее окно и выбрать функцию «Подключить другие приложения». В появившемся окне выбрать опцию «Colaboratory» (рисунок 3.20), выполнить установку и переход на данный продукт;

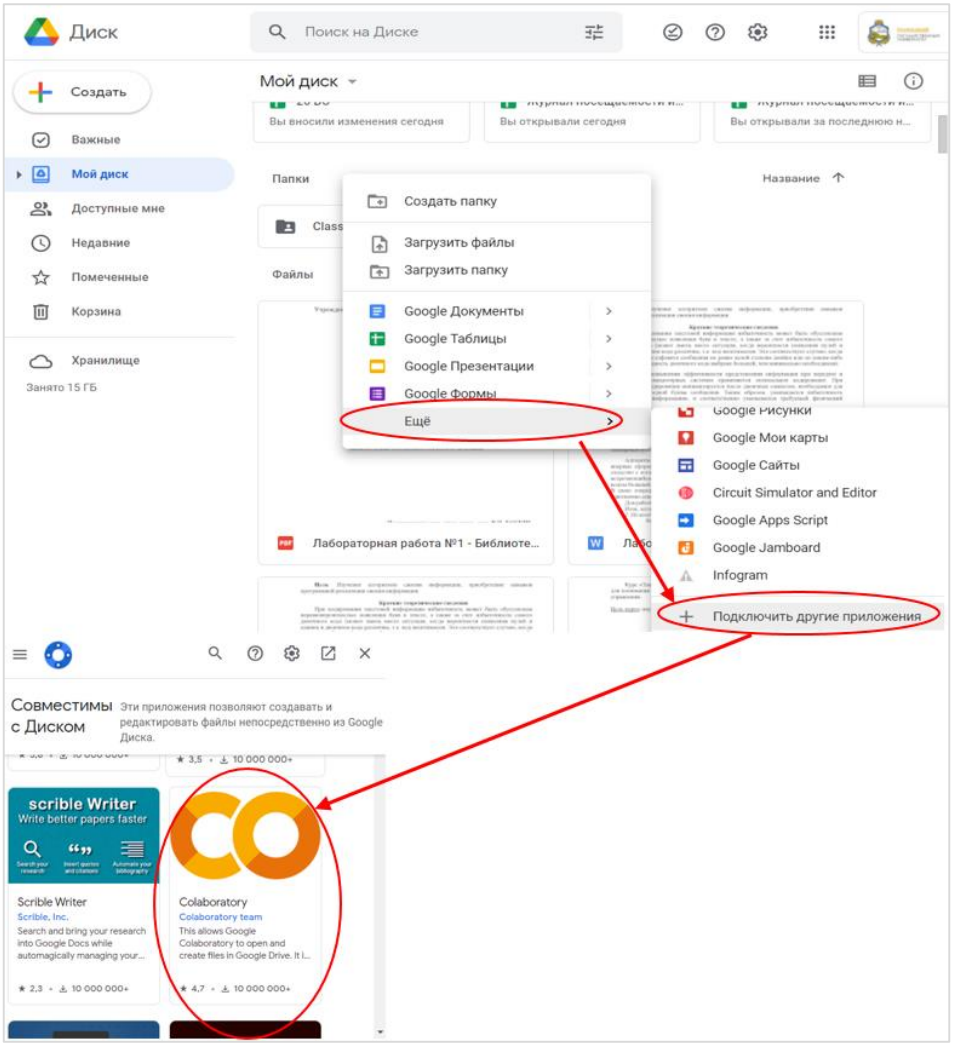


Рисунок 3.20. – Подключение бесплатной интерактивной облачной среды Colab

2) после перехода в Colaboratory на предыдущем шаге автоматически будет создан файл Untitled0.ipybn. Чтобы упростить передачу файлов с облачным графическим процессором, выделенным Colab, необходимо установить связь с Google Cloud Disk. Для этого в окне командной строки Untitled0.ipybn применить команды, как показано на рисунок 3.21. После успешного выполнения импорта данных станет доступен «My Drive» – файлы на Google-диске в Colab (рисунок 3.22);

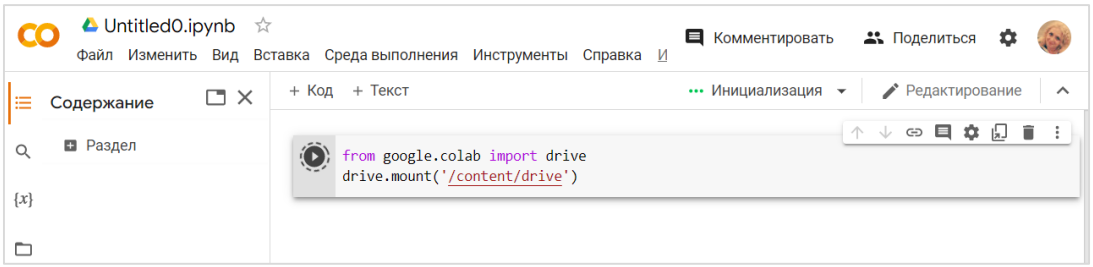


Рисунок 3.21. – Команды для импорта данных с Colab на Google Cloud Disk

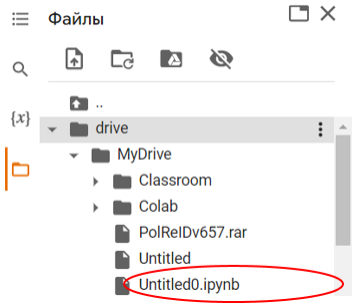


Рисунок 3.22. – Отображение файла Untitled0.ipybn на Google Cloud Disk

3) для удобства можно переименовать файл Untitled0.ipybn в YOLOv4.ipybn и переместить его в папку Colab;

4) выполнить действия по подключению графического процессора и проекта darknet с помощью git для возможности обучения в Colab:

– подключение графического процессора показано на рисунке 3.23;

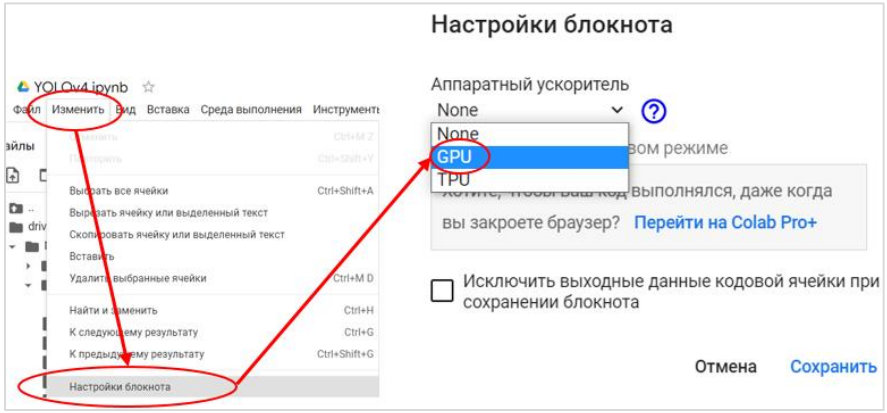


Рисунок 3.23. – Настройки для подключения графического процессора

– клонирование проекта для создания каталога darknet (рисунок 3.24). После успешного выполнения в окне «Файлы» будет создан каталог Darknet (рисунок 3.25). https://colab.research.google.com/drive/;

Рисунок 3.24. – Клонирование проекта для создания каталога darknet

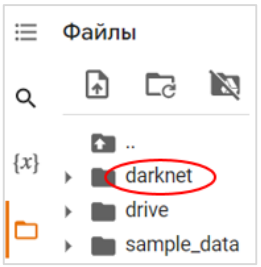


Рисунок 3.25. – Отображение каталога darknet на Google-диске

– изменить make-файл проекта с целью предоставления возможности скомпилированной модели СНС использования GPU и OpenCV:

```
%cd darknet
!sed -i 's/OPENCV=0/OPENCV=1/' Makefile
!sed -i 's/GPU=0/GPU=1/' Makefile
!sed -i 's/CUDNN=0/CUDNN=1/' Makefile
```

В случае успешного применения изменений содержимое Darknet/Makefile изменится для выбранных позиций (рисунок 3.26);

Makefile

```
1 GPU=0
2 CUDNN=0
3 CUDNN_HALF=0
4 OPENCV=0
5 AVX=0
6 OPENMP=0
7 LIBSO=0
8 ZED_CAMERA=0
9 ZED_CAMERA_v2_8=0
10
```

а

Makefile

```
1 GPU=1
2 CUDNN=1
3 CUDNN_HALF=0
4 OPENCV=1
5 AVX=0
6 OPENMP=0
7 LIBSO=0
8 ZED_CAMERA=0
9 ZED_CAMERA_v2_8=0
10
```

б

а – до изменений; *б* – после внесения изменений

Рисунок 3.26. – Содержимое Makefile

– скомпилировать проект Darknet командой `!make`;

– в каталоге /darknet/cfg создать файл yolov4-tiny-obj.cfg путем переноса содержимого из файла yolov4-tiny.cfg и отредактировать его для обнаружения заданного количества классов:

а) изменить размер пакета: `batch = 64` и `subdivision = 16`;

б) изменить строку `max_batches` на `classes*2000`, но не менее количества обучающих образцов и не менее 6000. Например, если количество классов = 3, то `max_batches = 6000`;

в) изменить строку `step` на значение, составляющее 80% и 90% от `max_batches`, т. е. `step = 4800, 5400`;

г) размер входного слоя: `weight` = 416, `height` = 416 или любое значение, кратное 32;

д) изменить строку `classes` = 80 на ваше значение числа классов для каждого из двух слоев обнаружения `yolo`;

е) для каждого из двух слоев, предшествующих слою yolo, необходимо изменить количество фильтров с `filters` = 255 на значение, равное `(classes` + 5) × 3. Для трех классов значение `filters` = 24;

– поместить в каталог /darknet файл предобученных весов для yolov4-tiny.cfg: https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v4_pre/yolov4-tiny.conv.29;

– созданные ранее файлы obj.data, obj.names и train.txt скопировать в каталог /darknet/data;

– набор данных с файлами разметки .txt поместить в /darknet/data/img;

– для старта обучения необходимо выполнить

```
!./darknet detector train data/obj.data cfg/yolov4-tiny-obj.cfg yolov4-tiny.conv.29 -dont_show
```

При обучении отображается информация, показанная на рисунке 3.27.

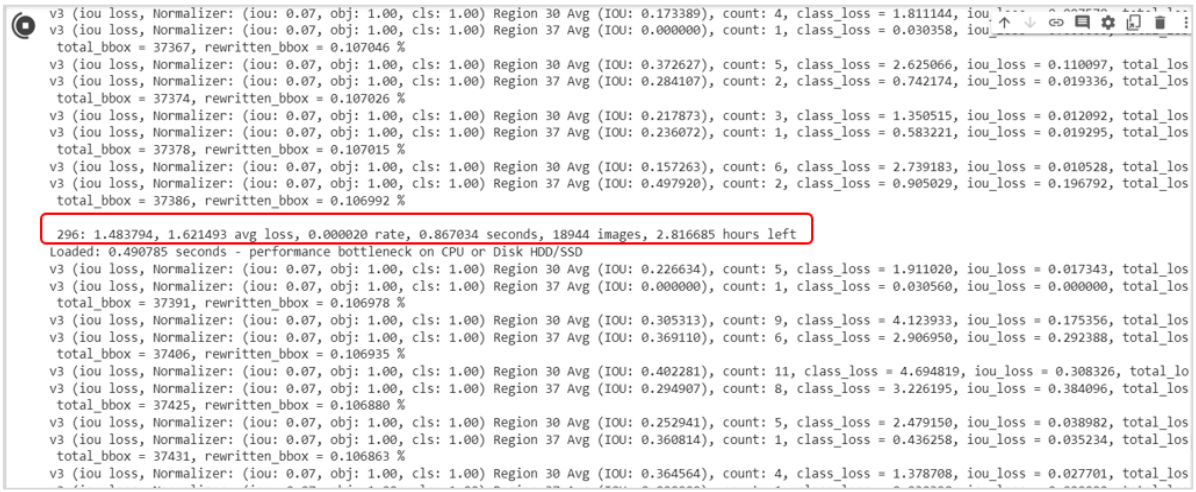


Рисунок 3.27. – Отображение данных при обучении СНС на облачном сервисе Google Colab: 296 – текущая итерация / пакет обучения; 1.483794 – общие потери; 1.621493 – среднее значение потерь; 0.000020 – текущая скорость обучения; 0.867034 – общее время, затраченное на обработку пакета; 18974 – общее количество изображений, обработанное к текущему моменту; 2,816685 – приблизительное значение времени, которое необходимо для завершения обучения СНС

Отличием обучения на облачном сервисе Google Colab от обучения на GPU компьютера является отсутствие возможности просмотра графика потерь в режиме онлайн, однако его можно загрузить двойным щелчком мыши или скачать. Также он будет доступен по окончании тренировки СНС (рисунок 3.28).

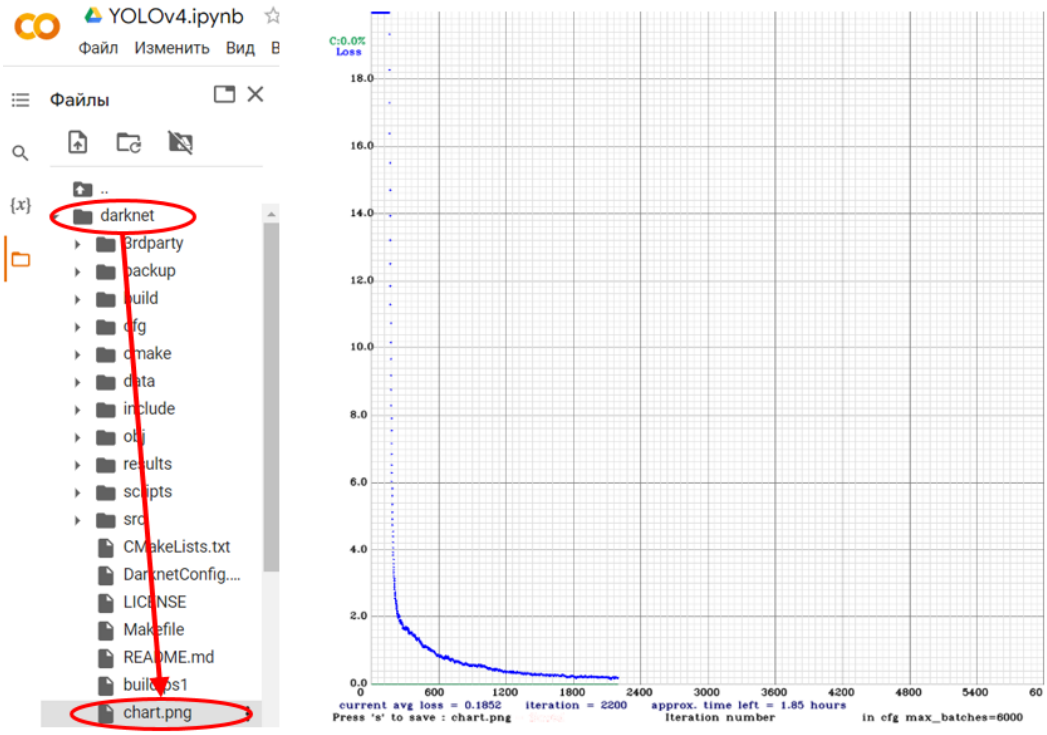


Рисунок 3.28. – График потерь при обучении СНС на облачном сервисе Google Colab

По окончании обучения отображаются все данные и сообщение о сохранении весов СНС (рисунок 3.29).

```
 6000: 0.057582, 0.062380 avg loss, 0.000026 rate, 0.902357 seconds, 384000 images, 0.047547 hours left
Saving weights to backup//yolov4-tiny-obj_6000.weights
Saving weights to backup//yolov4-tiny-obj_last.weights
Saving weights to backup//yolov4-tiny-obj_final.weights
If you want to train from the beginning, then use flag in the end of training command: -clear
```

Рисунок 3.29. – Результаты обучения СНС на облачном сервисе Google Colab

Для применения обученной СНС к задаче обнаружения объектов заданных классов необходимо использовать командную строку

```
!./darknet detector test data/obj.data cfg/yolov4-
tiny-obj.cfg yolov4-tiny-obj_final.weights data/3.jpg
```

В результате выполнения обнаружения объектов заданных классов создается файл Prediction.jpg (рисунок 3.30)

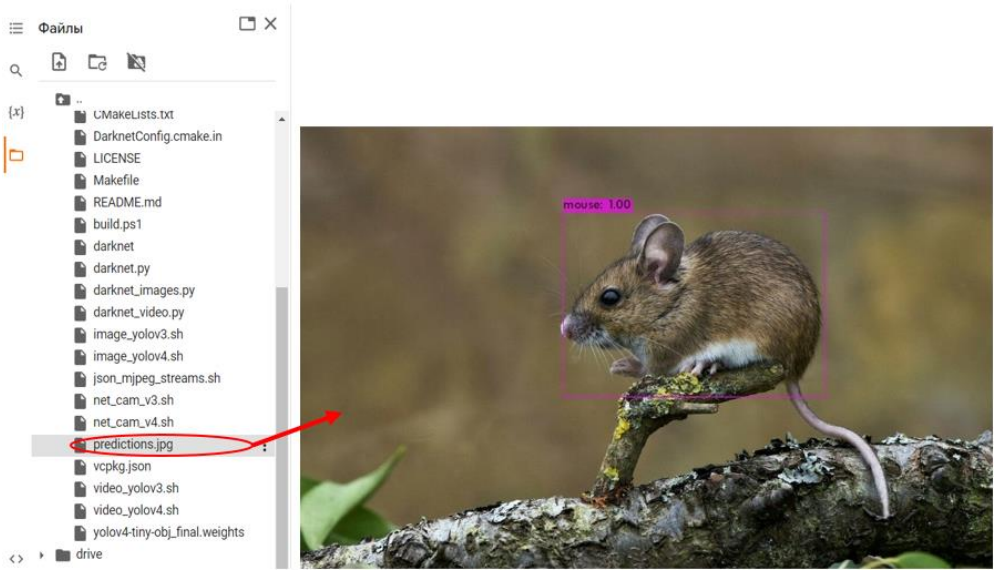


Рисунок 3.30. – Пример обнаружения объекта с использованием обученной СНС на облачном сервисе Google Colab

Содержание отчета

1. Титульный лист.
2. Цель работы и краткие теоретические сведения.
3. Ход работы, включая скриншоты с результатами выполнения всех заданий с пояснением и анализом результатов обучения.
4. Архив с используемой для обучения базой изображений.
5. Листинг кода.
6. Выводы о проделанной работе.

Контрольные вопросы

1. Что понимается под задачей обучения сверточной нейронной сети?
2. Поясните принцип обучения ИНС с учителем.
3. В чем заключается суть метода обратного распространения ошибок?
4. Какие виды функций потерь используются при обучении? Какая функция используется при обучении СНС YOLOv4?
5. Сформулируйте назначение, расскажите о принципе и проблемах метода градиентного спуска.
6. Каковы требования к заданию весов ИНС?
7. Поясните назначение и суть пакетной нормализации.
8. Как определить по графику потерь при обучении СНС успешность данной процедуры?

9. Поясните организацию обучения СНС YOLOv4-tiny на персональном компьютере.

10. Поясните организацию обучения СНС YOLOv4-tiny на облачном ресурсе.

Приложение
Установка CUDA Toolkit и cuDNN в Windows

CUDA Toolkit:

– загрузить версию CUDA с официального сайта: https://developer.nvidia.com/cuda-toolkit и распаковать;

– в параметрах установки выбрать «Выборочно» и убедиться, что интеграция с MS VS включена (рисунок 3.31);

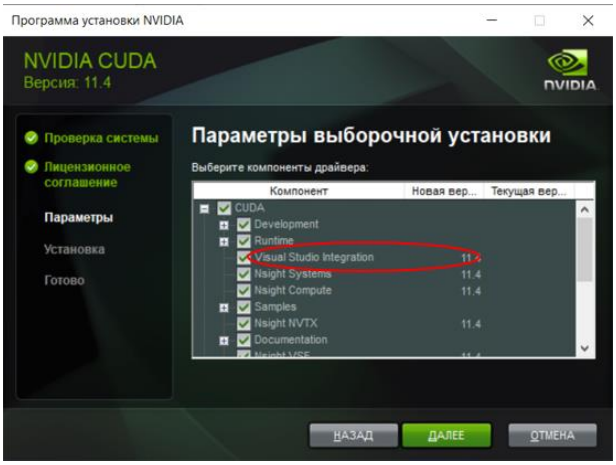


Рисунок 3.31. – Параметры выборочной установки CUDA

– следовать инструкции до завершения установки;

– убедиться, что инструментарий CUDA может правильно находить оборудование, поддерживающее CUDA, и взаимодействовать с ним. Для этого необходимо скомпилировать некоторые из включенных примеров программ или проверить версию CUDA;

– версию CUDA Toolkit можно проверить, запустив `nvcc -V` в командной строке (рисунок 3.32).

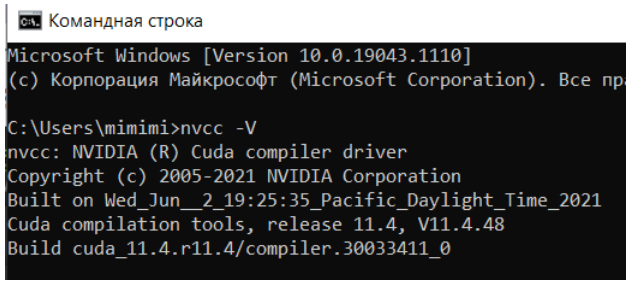


Рисунок 3.32. – Отображение версии CUDA

cuDNN:

– установить наиболее поздние драйверы на NVIDIA;

– скачать cuDNN с официального сайта: https://developer.nvidia.com/-cudnn (рисунок 3.33), но перед этим убедитесь, что зарегистрированы в программе разработчиков NVIDIA: https://developer.nvidia.com/developer-program.

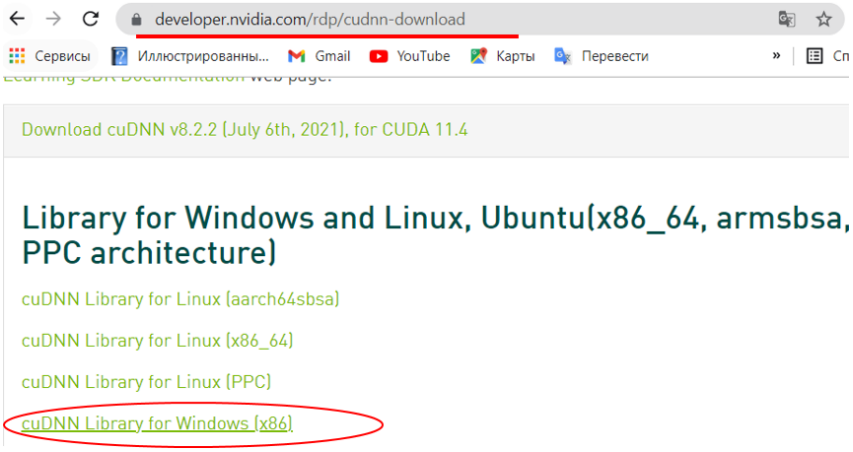


Рисунок 3.33. – Выбор cuDNN

– распаковать архив;

– перейти в каталог, содержащий фалы cuDNN. Скопировать указанные ниже файлы в каталог CUDA Toolkit. Следует обратить внимание, что пути на компьютерах пользователей могут отличаться.

а) копировать C:/cuda/bin/cudnn*.dll в C:/Program Files/NVIDIA GPU Computing Toolkit/CUDA/v11.4/bin;

б) копировать C:/cuda/include/cudnn*.h в C:/Program Files/NVIDIA GPU Computing Toolkit/CUDA/v11.4/include;

в) копировать C:/cuda/lib/x64/cudnn*.lib в C:/Program Files/NVIDIA GPU Computing Toolkit/CUDA/v11.4/lib/x64;

– проверить, чтобы CUDA_PATH был добавлен в переменные среды [19].

ЛАБОРАТОРНАЯ РАБОТА № 4
Сопровождение объекта на видеопоследовательности

Цель работы: изучение принципов обработки видео с использованием библиотек OpenCV и dlib в приложении к задаче сопровождения одного объекта на видео.

Теоретические сведения

Видеопоследовательность (видео, видеопоток, видеоряд) – это непрерывная последовательность цифровых изображений F (кадров) $V=\{F_k\}$, где k – номер изображения в последовательности.

Движущийся объект на последовательности изображений характеризуется изменением одного или нескольких основных параметров: формы, размеров, координат в течение некоторого интервала времени t. Трансформация формы и (или) размеров объекта приводит к изменению его признаков на кадрах ($ft_{Ob_q}^{F_k}$). Такой объект может быть представлен формальной моделью

$$Ob_q^D=\left\{ft_{Ob_q}^{F_k},x_{Ob_q}^{F_k},y_{Ob_q}^{F_k},Ns_{Ob_q}^{F_k}\right\},$$

где $x_{Ob_q}^{F_k}$, $y_{Ob_q}^{F_k}$ – координаты объекта;

$Ft_{Ob_q}^D$ – множество признаков движущегося объекта, $Ft_{Ob_q}^D \supseteq ft_{Ob_q}^{F_k}$, $\forall k \in t$, причём $\left(ft_{Ob_q}^{F_k} \cap ft_{Ob_q}^{F_{k+i}}\right) \in Ft_{Ob_q}^D$, т. е. для одного и того же движущегося объекта на последовательности кадров характерно изменение его признаков.

Сопровождение движущегося объекта – определение местоположения одного и того же объекта на каждом кадре видеопоследовательности в течении интервала времени t. Благодаря этому возможно построение траектории движения объекта, определение скорости и ускорения, что важно для различных прикладных задач.

Траектория движения объекта представляет собой последовательное отображение данного объекта на видеопоследовательности:

$$Tr\left(Ob^D\right)=\left(Ob_{F_k}^D\right),\ \forall k \in t.$$

Например, резкое изменение траектории движения человека в помещении может свидетельствовать о его падении, что важно для системы типа «умный дом». Как правило, движение рассматривается в системе координат изображения.

Для осуществления сопровождения необходимо выполнить процедуры обнаружения и локализации. Используются различные формы описания объектов сопровождения:

– одной точкой, характеризующей центр масс объекта или центр минимально возможного прямоугольника, описанного вокруг объекта;

– набором ключевых точек, по которым объект может быть однозначно опознан на последующих кадрах;

– геометрическим примитивом, описанным вокруг объекта (обычно эллипс или прямоугольник);

– внешним контуром объекта;

– набором областей, максимально устойчивым при движении, или всей областью объекта;

– инвариантными характеристиками объекта (такими как текстура, цветовая гамма и др.);

– признаками, формируемыми ИНС.

Разделяют два типа задачи: сопровождение одного объекта (Visual object tracking, VOT) и сопровождение множества объектов (Multiple object tracking, MOT). Фрагменты кадров с сопровождением объекта показаны на рисунке 4.2.

Особенности VOT:

– объект обнаружен и локализован на первом кадре;

– другие движущиеся объекты не детектируются ни на первом кадре, ни на последующих и, соответственно, не анализируются при таком типе сопровождения;

– сопровождение одного объекта традиционно выполняется на небольших временных интервалах.

Особенности MOT:

– множество объектов заданных классов или типов обнаруживаются и локализуются на первом кадре и на последующих при появлении новых объектов;

– обнаруженные объекты классифицируются;

– для правильного сопровождения и построения траектории движения выполняется сопоставление объектов на кадрах при их множественном

пересечении и скрытии за другими движущимися объектами или объектами фона. Примеры вариантов построения траекторий движения для объектов при множественном сопровождении и пересечении их траекторий показаны на рисунке 4.1.

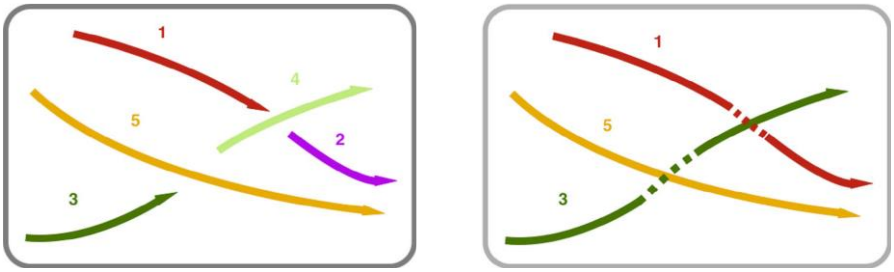


Рисунок 4.1. – Траектории движения при пересечении объектов

Рисунок 4.2. – Фрагменты кадров с сопровождением одного объекта

В кадре могут присутствовать несколько одновременно движущихся объектов. Причём объекты могут иметь практически идентичные визуальные признаки, выходить за пределы сцены на непродолжительное время или совсем ее покидать, могут появляться новые объекты, причем практически в местах выхода предыдущих, например, в дверном проеме при входе в помещение. Таким образом, возможна потеря объекта, которая возникает из-за его пересечения со схожим, или перекрытие элементом заднего плана. Сопровождение множества объектов выполняется на длительных временных интервалах, допускается прогнозирование местоположения на последующих кадрах. Примеры кадров с сопровождением множества объектов показаны на рисунок 4.3.

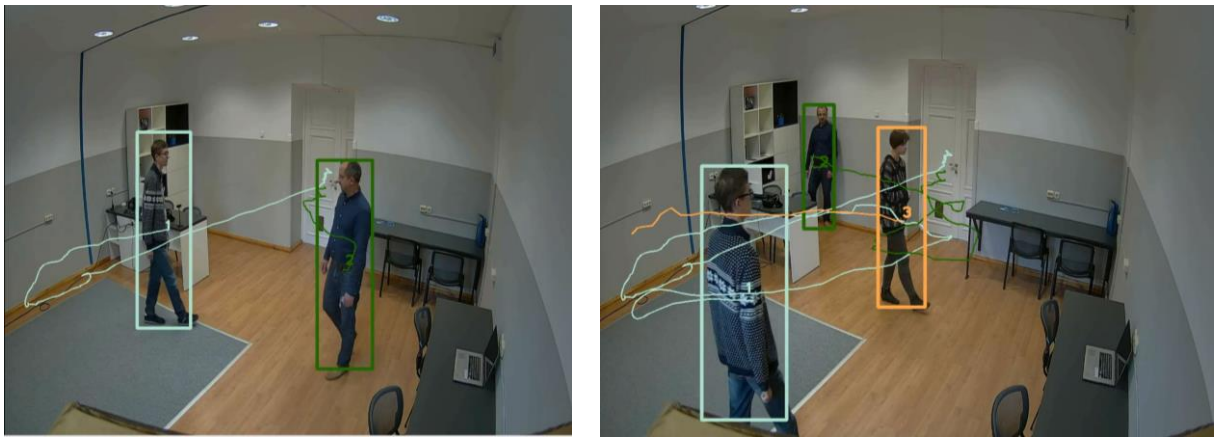


Рисунок 4.3. – Кадры с сопровождением нескольких объектов
и отображением их траектории движения

Существует множество алгоритмов сопровождения, но из-за возможности значительного визуального изменения объекта, изменения освещённости, помехи на заднем плане, окклюзий полностью задача сопровождения не решена, особенно сопровождения множества объектов. Для сопровождения одного объекта получены лучшие результаты.

Оба метода сопровождения, VOT и MOT, требуют на первом шаге обнаружения объектов. В практических приложениях эта задача может быть выполнена и оператором путем выделения заданного объекта на первом кадре, но такой подход используется, как правило, только при необходимости сопровождения одного объекта, VOT. Далее формируется вектор признаков, описывающий объект, например: изображение объекта, его координаты и размеры. На следующем кадре выполняется поиск обнаруженного объекта, как правило, с учетом ограниченной области поиска, которая зависит от скорости движения объекта. Существующие алгоритмы VOT отличаются используемым набором признаков, метриками или функциями схожести для сопоставления объектов на кадрах и принципом поиска.

Для установления соответствия между объектами $O = \{o_{ij}\}$ и $B = \{b_{ij}\}$ размером $N \times N$ может использоваться множество функций, наиболее применяемые:

– нормированное значение корреляции изменяется от 0 до 1, максимальное значение при полном сходстве объектов, вычисляется как

$$S^{NCC} = \frac{\sum_{i=1}^{N}\sum_{j=1}^{N} o_{ij} b_{ij}}{\sqrt{\sum_{i=1}^{N}\sum_{j=1}^{N} (o_{ij})^2}\sqrt{\sum_{i=1}^{N}\sum_{j=1}^{N} (b_{ij})^2}};$$

– метрика евклидова, при полном сходстве объектов значение равно 0:

$$S^{ED} = \frac{1}{LN}\sqrt{\sum_{i=1}^{N}\sum_{j=1}^{N}(o_{ij} - b_{ij})^2},$$

где L – количество уровней яркости;

– метрика на основе среднего значения суммы квадратов разностей, при полном сходстве объектов значение равно 0:

$$S^{SSD} = 1 - \frac{1}{LN^2}\sum_{i=1}^{N}\sum_{j=1}^{N}(o_{ij} - b_{ij})^2.$$

Установление соответствия объектов выполняется с учётом ограниченной величины возможного смещения объекта, т. е. обнаруженные области движения на соседних кадрах принадлежат одному и тому же объекту, если они значительно перекрываются между собой. Для проверки данного критерия используется условие, согласно которому центр одного объекта должен находиться в области другого, либо метрика *IoU*. Если для объектов на кадрах $IoU \geq 0{,}5$, то данный объект является сопровождаемым.

Для сравнения эффективности VOT-алгоритмов сопровождения применяется такой критерий, как точность

$$PRC = \frac{TP}{TP + FP},$$

где TP – количество правильных обнаружений сопровождаемого объекта на всех кадрах видеопоследовательности;

FP – количество ложных обнаружений сопровождаемого объекта на всех кадрах видеопоследовательности.

При оценке качества сопровождения используется метрика *IoU*, которая рассчитывается для обнаруженной и известной прямоугольной области объекта в кадре. Значение *IoU* должно быть не менее 0,5 для принятия решения о правильном сопровождении объекта.

Надежность показывает, сколько раз при сопровождении теряется объект, и он должен быть повторно инициализирован. Сопровождаемый объект считается потерянным, когда значение $IoU < 0{,}5$.

Задание к лабораторной работе

Необходимо программно реализовать VOT-сопровождение объекта на видео с использованием библиотек OpenCV и dlib. Для этого:

1) инициализировать файлы конфигурации, весов и имен классов СНС YOLOv4 для обнаружения объектов на первом кадре;

2) захватить первый кадр видео для обработки и применить СНС для обнаружения объектов;

3) сформировать массивы для хранения признаков объекта: идентификатор, степень уверенности сети и координаты объекта;

4) реализовать сопровождение объекта, который должен быть определен алгоритмически согласно варианту задания с использованием реализованного алгоритма сопровождения (трекера) в библиотеке dlib correlation_tracker. Описание необходимых инструментов из OpenCV и dlib представлено в приложении 4.1, в нем также показаны примеры фрагментов кода для работы с видео.

Варианты заданий:

Вариант 1: обнаруженный на первом кадре легковой автомобиль с наибольшей степенью уверенности.

Вариант 2: обнаруженный на первом кадре легковой автомобиль с наименьшей степенью уверенности.

Вариант 3: обнаруженный на первом кадре грузовой автомобиль с наибольшей степенью уверенности.

Вариант 4: обнаруженный на первом кадре грузовой автомобиль с наименьшей степенью уверенности.

Вариант 5: обнаруженный на первом кадре автобус с наибольшей степенью уверенности.

Вариант 6: обнаруженный на первом кадре автобус с наименьшей степенью уверенности.

Вариант 7: обнаруженный на первом кадре человек с наибольшей степенью уверенности.

Вариант 8: обнаруженный на первом кадре человек с наименьшей степенью уверенности.

Вариант 9: обнаруженный на первом кадре велосипедист с наибольшей степенью уверенности.

Вариант 10: обнаруженный на первом кадре велосипедист с наименьшей степенью уверенности.

Вариант 11: обнаруженная на первом кадре лошадь с наибольшей степенью уверенности.

Вариант 12: обнаруженная на первом кадре лошадь с наименьшей степенью уверенности.

Вариант 13: обнаруженная на первом кадре птица с наибольшей степенью уверенности.

Вариант 14: обнаруженная на первом кадре птица с наименьшей степенью уверенности.

Вариант 15: обнаруженный на первом кадре мотоциклист с наибольшей степенью уверенности.

Содержание отчета

1. Титульный лист.
2. Цель работы.
3. Ход работы, включая программную реализацию с описанием алгоритма выбора объекта для п. 4.
4. Примеры кадров с сопровождением, обязательно примеры срыва сопровождения объекта с пояснением причин.
5. Листинг кода с комментариями.
6. Выводы о проделанной работе.

Контрольные вопросы

1. Что понимают под видеопоследовательностью?
2. Какие параметры объекта на видео изменяются, если он является движущимся?
3. Что такое «сопровождение объектов на видеопоследовательностях» и в каких практических областях используется (приведите конкретные примеры)?
4. Приведите пример объекта, который движется, но его движение на видеопоследовательности обнаружено быть не может? Какие для этого должны быть условия?
5. Какие основные типы сопровождения объектов на видео различают? Назовите их отличия.
6. Сформулируйте общий подход к реализации алгоритма сопровождения одного объекта на видео.
7. Какие функции и метрики могут использоваться для установления схожести между объектами на кадрах?

8. Какие критерии применяются для оценки эффективности сопровождения объекта на видео и что они характеризуют?

9. Какие функции из OpenCV использованы для реализации сопровождения объекта и их назначение?

10. Какие функции из dlib использованы для реализации сопровождения объекта и каково их назначение?

Приложение
Функции библиотек OpenCV и dlib для сопровождения объектов и примеры фрагментов кода

`cv2.namedWindow()` – используется для создания окна с подходящим именем и размером для отображения изображений и видео на экране. Изображение по умолчанию отображается в своем исходном размере, поэтому может потребоваться изменить размер изображения, чтобы оно соответствовало экрану. Функция ничего не делает, если окно с таким именем существует.

`cv2.imshow()` – используется для отображения изображения в окне. Окно автоматически подстраивается под размер изображения.

`cv2.rectangle()` – инструмент, позволяющий отобразить прямоугольник на изображении по заданным координатам верхнего левого и правого нижнего углов.

`cv2.destroyAllWindows()` – функция, которая позволяет закрыть сразу все окна.

`cv2.waitKey()` – задержка отображения окна в течение заданных миллисекунд или до нажатия какой-либо клавиши на клавиатуре. Если в качестве параметра принято число, то время ожидания до закрытия окна ограничено указанным числом миллисекунд, если 0 или параметр не указан, окно закроется по нажатию любой клавиши клавиатуры.

`cv2.VideoCapture()` – инструмент для захвата видео. В качестве аргумента выступают индекс устройства или имя видеофайла.

`cv2.VideoWriter()` – инструмент для сохранения обработанных кадров в видеофайл.

`dlib.correlation_tracker()` – инструмент для отслеживания движущихся объектов в видеопотоке.

`start_track()` – функция из `correlation_tracker`, принимает на вход изображение и координаты ограничительного прямоугольника `dlib.rectangle()`.

Листинг 4.1. – Инициализация файлов конфигурации, весов и имен классов для СНС YOLOv4

```
weights_file = 'models/yolov4.weights'
config_file = 'models/yolov4.cfg'
classes_file = 'models/coco.names'
```

Листинг 4.2. – Извлечение имен классов объектов обученной модели YOLOv4 и подключение фреймворка Darknet с помощью модуля OpenCV dnn

```
with open(classes_file, 'r') as f:
  classes = [line.strip() for line in f.readlines()]
net = cv2.dnn.readNetFromDarknet(config_file, weights_file)
```

Листинг 4.3. – Функция для отрисовки ограничительных рамок объектов и добавления к ним идентификаторов обнаруженных объектов в формате ind_class-id_confidence, т. е. номер идентификатора, название класса объекта и степень уверенности сети

```
COLORS = np.random.uniform(0, 255, size=(len(classes), 3))
def draw_bounding_box(img,ind, class_id, confidence, x, y,
          x_plus_w, y_plus_h):
  label = str(ind) + '_'+ str(classes[class_id]) + '_' +
          str(confidence)[:4]
  color = COLORS[ind]
  cv2.rectangle(img, (x, y), (x_plus_w, y_plus_h), color, 2)
  cv2.putText(img, label, (x - 10, y - 10),
  cv2.FONT_HERSHEY_SIMPLEX, 1, color, 2)
```

Листинг 4.4. – Формирование массивов для хранения данных для обнаруженных объектов: initBB для хранения координат ограничительных рамок всех обнаруженных объектов; classid для хранения идентификаторов классов всех обнаруженных объектов; object – где будут храниться 6 значений для каждого из объектов (идентификатор, степень уверенности сети и четыре координаты)

```
  initBB = []
  classid = []
  object = []
  for i in indices:
    i = i
    if class_ids[i] in show_classes:
      classid = class_ids[i]
```

```
      counter_per_class[classid] += 1
    x, y, w, h = boxes[i]
    if x >= 0 and y >= 0 and w >= 0 and h >= 0:
      initBBB = [x, y, x+w, y+h]
      initBB.append(initBBB)
      nn=[class_ids[i], confidences[i], initBBB[0], init
          BBB[1], initBBB[2], initBBB[3]]
      object.append(nn)
  for i in range(len(initBB)):
      box=initBB[i][:]
      draw_bounding_box(image, (i+1), object[i][0],
          object[i][1], object[i][2], object[i][3],
          object[i][4], object[i][5])
else:
  break
```

Листинг 4.5. – Инициализация трекера dlib

```
tracker = [dlib.correlation_tracker()
```

Листинг 4.6. – Обновление координат, передаваемых в трекер, в соответствии с изменившемся положением объекта

```
tracker.update(img)
rect = tracker.get_position()
pt1 = (int(rect.left()), int(rect.top()))
pt2 = (int(rect.right()), int(rect.bottom()))
print ("Object {} tracked at [{}, {}] \r".format(i+1, pt1, pt2),)
obj=detect_point.object
```

ЛИТЕРАТУРА

1. OpenCV [Electronic resource]. – Mode of access: https://opencv.org/.
2. Dlib C++ Library [Electronic resource]. – Mode of access: http://dlib.net/.
3. OpenCV. Tutorials for contrib modules [Электронный ресурс]. – Mode of access: https://docs.opencv.org/3.2.0/d3/d81/tutorial_contrib_root.html.
4. CMake [Electronic resource]. – Mode of access: https://cmake.org/install/.
5. Study tonight. Dlib 68 point Face landmark Detection with OpenCV and Python [Electronic resource]. – Mode of access: https://www.studytonight.com/post/dlib-68-points-face-landmark-detection-with-opencv-and-python.
6. Bochkovskiy, A. YOLOv4: Optimal Speed and Accuracy of Object Detection. ArXiv, abs/2004.10934, 2020 [Electronic resource] / A. Bochkovskiy, C. Wang, H. Liao. – Mode of access: https://arxiv.org/pdf/2004.10934.pdf.
7. The PASCAL Visual Object Clfsses Homepage [Electronic resource]. – Mode of access: http://host.robots.ox.ac.uk/pascal/VOC.
8. COCO. Common Objects in Context [Electronic resource]. – Mode of access: http://images.cocodataset.org/zips/test2017.zip.
9. Open Images Dataset V6 + Extensions [Electronic resource]. – Mode of access: https://storage.googleapis.com/openimages/web/index.html.
10. The Street View House Numbers (SVHN) Dataset [Electronic resource]. – Режим доступа: http://ufldl.stanford.edu/housenumbers.
11. Stanford Dogs Dataset [Electronic resource]. – Mode of access: http://vision.stanford.edu/aditya86/ImageNetDogs.
12. Darknet. GitHab [Electronic resource]. – Mode of access: https://github.com/AlexeyAB/darknet.
13. Созыкин, А. В. Обзор методов обучения глубоких нейронных сетей / А. В. Созыкин // Вестник ЮУрГУ. Сер. Вычислительная математика и информатика. – 2017. – Т. 6, № 3. – С. 28–59. – DOI: 10.14259/cmse170303.
14. Кустикова, В. Д. Образовательный курс «Методы глубокого обучения для решения задач компьютерного зрения». Лекция 4. «Сверточные нейронные сети» [Электронный ресурс] / В. Д. Кустикова. – Режим доступа: http://hpc-education.unn.ru/files/courses/intel-neon-course/Rus/Lectures/Presentations/4_CNN.pdf.
15. Neurohive. Градиентный спуск: все, что нужно знать [Электронный ресурс]. – Режим доступа: https://neurohive.io/ru/osnovy-data-science/gradient-descent/.

16. Осовский, С. Нейронные сети для обработки информаци / С. Осовский ; пер. с польского И. Д. Рудинского. – М. : Финансы и статистика, 2002. – 344 с.

17. Рашид, Тарик. Создаем нейронную сеть / Тарик Рашид ; пер. с англ. – СПб. : ООО «Альфа-книга», 2017. – 272 с.

18. Nvidia. Developer zone. Cuda Toolkit Dokumentation [Electronic resource]. – Mode of access: https://docs.nvidia.com/cuda/cuda-installation-guide-microsoft-windows/index.html.