УНИВЕРСИТЕТ ИТМО

**С.Н. Ярышев, В.А. Рыжова**

# ТЕХНОЛОГИИ ГЛУБОКОГО ОБУЧЕНИЯ И НЕЙРОННЫХ СЕТЕЙ В ЗАДАЧАХ ВИДЕОАНАЛИЗА

**Санкт-Петербург**
**2022**

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

УНИВЕРСИТЕТ ИТМО

**С.Н. Ярышев, В.А. Рыжова**

# ТЕХНОЛОГИИ ГЛУБОКОГО ОБУЧЕНИЯ И НЕЙРОННЫХ СЕТЕЙ В ЗАДАЧАХ ВИДЕОАНАЛИЗА

УЧЕБНОЕ ПОСОБИЕ

РЕКОМЕНДОВАНО К ИСПОЛЬЗОВАНИЮ В УНИВЕРСИТЕТЕ ИТМО
по направлению подготовки 12.04.02 Оптотехника
в качестве учебного пособия для реализации основных профессиональных образовательных программ высшего образования магистратуры


**Санкт-Петербург**
**2022**

Ярышев С.Н., Рыжова В.А., Технологии глубокого обучения и нейронных сетей в задачах видеоанализа – СПб: Университет ИТМО, 2022. – 82 с.

Рецензент(ы):
Арбузов Сергей Николаевич, кандидат физико-математических наук, бренд-менеджер оборудования видеонаблюдения, ООО Легарда;

Учебное пособие предназначено для студентов по направлению подготовки магистров 12.04.02 Оптотехника по основным образовательным программам "Техническое зрение" и "Прикладная оптика" и содержит материалы лабораторных работ по дисциплинам "Видеосистемы и видеоаналитика"/"Video systems and video analytics", "Алгоритмы аналитики видеопотока", "Искусственный интеллект в обработке изображений", "Комплексные системы безопасности". Разработанный авторами лабораторный практикум направлен на формирование у студентов практических навыков программирования глубоких нейронных сетей на языке Python. К лабораторным работам сформулированы контрольные вопросы и предложены варианты выполнения.

УНИВЕРСИТЕТ ИТМО

**Университет ИТМО** – национальный исследовательский университет, ведущий вуз России в области информационных, фотонных и биохимических технологий. Альма-матер победителей международных соревнований по программированию – ICPC (единственный в мире семикратный чемпион), Google Code Jam, Facebook Hacker Cup, Яндекс.Алгоритм, Russian Code Cup, Topcoder Open и др. Приоритетные направления: IT, фотоника, робототехника, квантовые коммуникации, трансляционная медицина, Life Sciences, Art&Science, Science Communication. Входит в ТОП-100 по направлению «Автоматизация и управление» Шанхайского предметного рейтинга (ARWU) и занимает 74 место в мире в британском предметном рейтинге QS по компьютерным наукам (Computer Science and Information Systems). С 2013 по 2020 гг. – лидер Проекта 5–100.

СОДЕРЖАНИЕ

# ВВЕДЕНИЕ

Современное развитие видеоинформационных технологий и систем технического зрения во многом способствует повышению эффективности решения различных производственных и научно-исследовательских задач, связанных с распознаванием объектов на изображениях и в видеопотоке. Одним из наиболее перспективных направлений в решении задач видеоанализа является использование технологии нейронных сетей и глубокого обучения, с помощью которых можно создавать компьютерные системы и приложения, выполняющие задачи, обычно поручаемые людям. Нейронная сеть и глубокое обучение являются одними из самых эффективных технологий выделения и распознавания объектов. В большинстве случаев для этих задач используются сверточные нейронные сети, специально созданные для обработки двумерных матриц, к которым можно отнести и цифровые изображения.

Приемы машинного и глубокого обучения особенно активно развивается в течение последних десяти лет благодаря созданию доступных программных инструментов разработки, обучения и оптимизации нейронных сетей самого широкого применения, в том числе для обработки изображений и распознавания образов.

Предлагаемое пособие предназначено для формирования у магистрантов образовательных программ «Прикладная оптика» и «Техническое зрение» наиболее значимых профессиональных компетенций в области исследования и разработки современных цифровых видеоинформационных систем:

- реализует в своей деятельности принципы оптико-электронного приборостроения и видеотехники;
- учитывает современные тенденции развития научных исследований в области производства и применения видеоинформационных систем;
- разрабатывает методики экспериментальных исследований систем технического зрения в соответствии с поставленными научными и практическими задачами;
- осуществляет запись и обработку видеоинформации в системах технического зрения;
- разрабатывает и реализует эффективные алгоритмы и численные методы на языке программирования высокого уровня;

- проводит экспериментальные исследования, направленные на создание новых элементов, систем и приборов технического зрения.

Настоящее пособие посвящено освоению и исследованию нейронных сетей и глубокого обучения в области видеоанализа. Лабораторный практикум предназначен для закрепления обучающимися теоретических знаний и приобретения ими практических навыков при изучении дисциплин "Видеосистемы и видеоаналитика", "Алгоритмы аналитики видеопотока", "Искусственный интеллект в обработке изображений", "Комплексные системы безопасности", " Методы и средства видеоинформатики".

Пособие содержит 4 (четыре) лабораторные работы, направленные на формирование основных профессиональных компетенций путем приобретения студентами умений и навыков проектирования и эксплуатации видеоинформационных систем безопасности.

Состав лабораторных работ:

1. Изучение методов обнаружения объектов на основе нейронных сетей и глубокого обучения
2. Знакомство с нейронной сетью YOLO. Основные параметры сети.
3. Изучение процесса обучения нейронной сети YOLO. Основные параметры процесса обучения.
4. Подготовка датасета для обучения нейронной сети.

Лабораторные работы содержат описания технологий, реализующихся при цифровой обработке изображений в видеокамерах, и задания для выполнения студентами с использованием библиотек языка Python. Описание каждой лабораторной работы содержит краткие теоретические сведения, порядок выполнения работы с примерами ее выполнения, требования к отчету, контрольные вопросы. В конце пособия приводится список рекомендуемых информационных источников.

Выполнение лабораторной работы производится во время занятий в аудитории, оснащенной необходимыми программно-аппаратными средствами, в присутствии преподавателя. Организация лабораторного практикума предполагает в обязательном порядке проведение инструктажа студентов по соблюдению требований техники безопасности при эксплуатации персональных компьютеров и систем сетевого видеонаблюдения, формирующих лабораторные установки.

**Внимание!**

**Лабораторные установки могут включаться и выключаться только преподавателем или лаборантом, или под их непосредственным наблюдением! На персональном компьютере необходимо выполнять только те действия, которые предусматривает порядок выполнения лабораторной работы. Запрещается запускать приложения, не относящиеся к лабораторной работе, и инсталлировать сторонние ПО и модули Python самостоятельно.**

В процессе выполнения лабораторной работы студент последовательно выполняет задания в соответствии с указанным порядком. Время, отводимое на выполнение, определяется трудоемкостью лабораторного практикума.

По завершении работы обучающийся демонстрирует преподавателю результаты ее выполнения на персональном компьютере и предъявляет файлы со скриншотами и другими материалами для включения их в отчет.

Отчет о лабораторной работе должен включать в себя изображения экрана (скриншоты), полученные в ходе работы, комментарии к изображениям. Для получения скриншота в ходе работы нужно запоминать соответствующие изображения путем нажатия на кнопку Print Screen для помещения скриншота в буфер обмена и перемещения его в заранее открытый документ MS Word. В дальнейшем этот документ с комплектов скриншотов следует использовать для отчета. Отчет должен быть оформлен в электронном виде и распечатан (при необходимости). За основу отчета должен быть взят прилагаемый шаблон**:**

***Отчет по лабораторной работе №*** ____«_____»

*(название лабораторной работы)*

1. Цель и задачи лабораторной работы: ____
2. Методика проведения исследования: ____
3. Результаты: ____
4. Выводы: ____

В имеющемся шаблоне следует заполнить обязательные поля титульного листа, включающие ФИО и группу студента, дату выполнения работы. В качестве результатов используются скриншоты основных окон программы, их описание и комментарии.

Размер отчета должен быть не менее пяти страниц. При необходимости вставленные в шаблон рисунки следует дополнить поясняющими графическими элементами и отмасштабировать.

Лабораторная работа №1. **Изучение методов обнаружения объектов на основе нейронных сетей и глубокого обучения**

**Цели работы:**

1. Изучение теоретических основ методов обнаружения объектов на основе глубокого обучения с использованием нейронных сетей.
2. Получение практических навыков работы в операционной системе Linux и в среде разработки Spyder.
3. Исследование особенностей обнаружения объектов на изображениях с использованием программ на языке программирования Python, модулей и пакетов OpenCV, Keras, Tensorflow.

Работа рассчитана на 2 часа в базовом варианте или на 4 часа в расширенном варианте. В последнем случае преподаватель дает дополнительные задания, связанные с настройкой режимов работы программного обеспечения с использованием нейронных сетей и технологии глубокого обучения.

**Основные теоретические сведения**

Нейро́нная сеть (или искусственная нейронная сеть, ИНС) — математическая модель, а также её программное или аппаратное воплощение, построенная по принципу организации и функционирования биологических нейронных сетей — сетей нервных клеток живого организма [1]. Это понятие возникло при изучении процессов, протекающих в мозге, и при попытке их смоделировать.

Основой ИНС является искусственный нейрон, который является отдаленным подобием биологического нейрона (рис. 1).

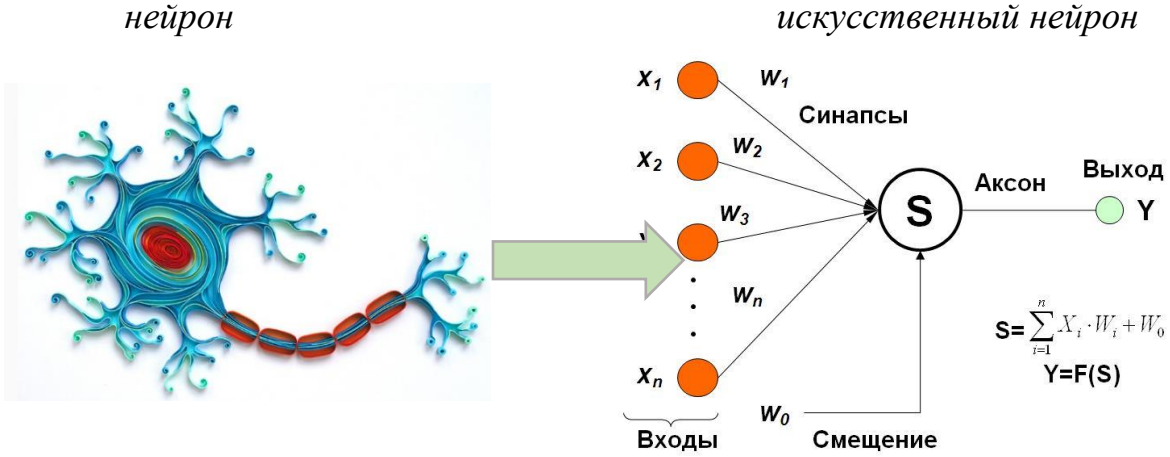



Рисунок 1 – Упрощение от нейрона к схеме искусственного нейрона

Искусственный нейрон имеет несколько входов (аналоги синапсов в биологическом нейроне) и один выход (аналог аксона).

Математически нейрон выполняет функцию суммирования S входных сигналов X с учетом их весов W, и затем результат обрабатывается функцией активации F, например, сигмоидальной функцией или сигмоидой (sigmoid). Это монотонно возрастающая дифференцируемая s-образная нелинейная функция, которая позволяет усиливать слабые сигналы и не насыщаться от сильных сигналов. Результат на выходе Y зависит от входных сигналов X и их весов W, а также от функции активации. Коэффициенты W являются элементами памяти нейрона и основными элементами обучения нейронной сети.

Объединение искусственных нейронов в группу формирует нейронную сеть (рис. 2).

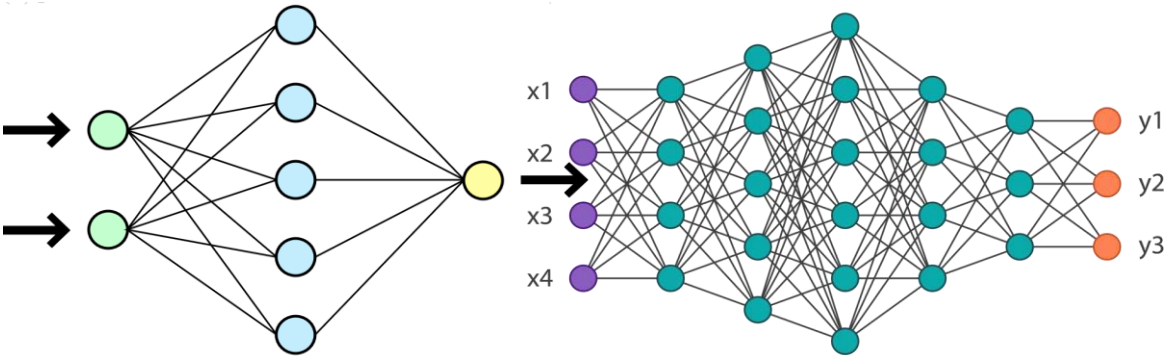



Рисунок 2 – Схема формирования нейронной сети

Простая нейронная сеть состоит из входного слоя, скрытого слоя и выходного слоя. Сети, содержащие много скрытых слоев, часто называют глубинными нейронными сетями.

Среди основных топологий нейронных сетей можно выделить полносвязные, сверточные и рекуррентные нейронные сети [2].

**Полносвязные** нейронные сети имеют несколько слоев, которые связаны между собой таким образом, что каждый нейрон последующего слоя имеет связь со всеми нейронами предыдущего слоя. Сложность сети резко возрастает от увеличения размерности входных данных и от количества скрытых слоев. Так, для анализа изображения форматом 28 x 28 элементов потребуется 784 нейрона в скрытом слое, и каждый из них должен иметь 784 входа для соединения с предыдущим слоем. Другая

проблема заключается в том, что в полносвязной сети изображения представляют собой одномерные последовательности и при этом не учитываются особенности изображений как структуры данных. Тем не менее, для изображений небольших форматов можно использовать и полносвязную сеть.

**Сверточные** нейронные сети предназначены для обработки двумерных структур данных, прежде всего изображений. Сверточная сеть представляет собой комбинацию трех типов слоев:

- слои, которые выполняют функцию свертки над двумерными массивами данных (сверточные слои),

- слои, выполняющие функцию уменьшения формата данных (слой субдискретизации),

- полносвязные слои, завершающие процесс обработки данных.

Структура сверточных нейронных сетей принципиально многослойная. Работа свёрточной нейронной сети обычно интерпретируется как переход от конкретных особенностей изображения к более абстрактным деталям, и далее к ещё более абстрактным деталям вплоть до выделения понятий высокого уровня. При этом сеть самонастраивается и вырабатывает необходимую иерархию абстрактных признаков (последовательности карт признаков), фильтруя маловажные детали и выделяя существенное [1, 3]. Примером классической сверточной нейронной сети является сеть VGG16 (рис. 3).

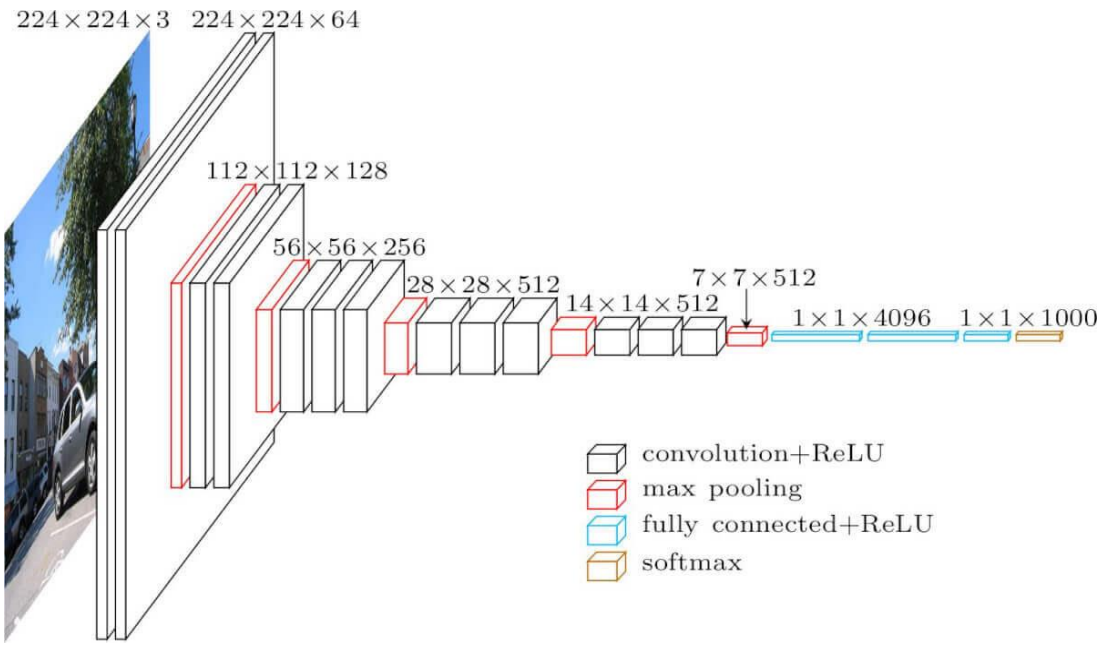



Рисунок 3 – Структура классической сети VGG16 [4]

Сеть VGG-16 имеет 16 слоев и способная работать с изображениями достаточно большого формата 224x224 пикселя [5]. В своей стандартной топологии эта сеть способна работать с датасетом изображений ImageNet, содержащем более 15 млн. изображений, разбитых на 22000 категорий [6].

**Рекуррентные** нейронные сети отличаются от многослойных сетей тем, что могут использовать свою внутреннюю память для обработки последовательностей произвольной длины. Благодаря направленной последовательности связей между элементами рекуррентных сетей они применимы в таких задачах, где нечто целостное разбито на сегменты, например, распознавание рукописного текста или распознавание речи.

Для работы с нейронными сетями требуется их обучение под конкретную задачу. В частности, для решения задачи распознавания объектов на изображении требуется обучение сети по специально подготовленному набору данных, который содержит изображения всех классов распознаваемых объектов, сгруппированных в соответствующие разделы. Такой тип данных носит название **датасет** (набор данных, Data set) [1-3].

Существует большое количество уже собранных и подготовленных датасетов для решения различных задач с использованием нейронных сетей (не только для задач распознавания объектов) [7]. Более того, существуют уже заранее обученные под решение конкретной задачи нейронные сети, которые можно взять в готовом виде [8]. Но перечень таких сетей и датасетов не очень большой, и в общем случае перед разработчиком может стоять задача выбора конфигурации нейронной сети под конкретную задачу и создание соответствующей базы данных (датасета) для ее обучения.

Формирование датасета является наиболее трудоемкой частью процесса разработки, поэтому в первую очередь нужно проверить возможное наличие похожего датасета на доступных ресурсах, например: https://www.kaggle.com/. На этом ресурсе имеется более 50.000 свободно распространяемых датасетов и более 400.000 примеров реализаций нейронных сетей. В ряде случаев имеющиеся датасеты можно объединять, модифицировать и дополнять.

Процесс **обучения нейронных сетей** представляет собой сложный процесс обработки данных, который включает в себя последовательное предъявление данных на вход нейронной сети и сравнение выходных данных с их истинным значением, после чего вносится коррекция весовых

коэффициентов нейронов в сторону уменьшения ошибки выходных данных. Этот процесс производится многократно с использованием данных из датасета. В процессе обучения используется часть датасета, которая носит название **тренировочный набор**. При этом данные из датасета могут предъявляться последовательно несколько раз.

Один цикл обучения с использованием всего датасета носит название **эпоха**. Как правило, для качественного обучения сети требуется много эпох. Процесс обучения нейронных сетей, имеющих много скрытых слоев, часто носит название **глубокого обучения**.

В процессе обучения и контроля за качеством обучения широко используются методы градиентного спуска и обратного распространения ошибки [2]. Метод градиентного спуска (рис. 4) позволяет найти локальный минимум функции (которая является функцией ошибки) от нескольких переменных (весов нейронов) путем последовательного малого изменения этих переменных.

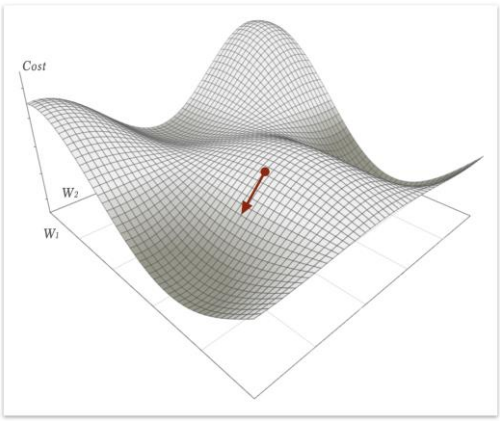



Рисунок 4 – Иллюстрация метода градиентного спуска

Метод обратного распространения ошибки позволяет выполнить процедуру коррекции весов по направлению от выхода ко входам нейронов путем использования частных производных.

Качество обучения можно контролировать путем анализа ошибок сети на ее выходе при сравнении с истинными значениями датасета (рис. 5). В процессе обучения, как правило, эти ошибки уменьшаются по экспоненте, но может наступить момент, когда ошибка сети начинает расти. С этого момента фиксируется так называемый эффект переобучения сети, когда процесс обучения целесообразно остановить.

Для анализа качества обучения используется специальный раздел данных датасета, который носит название **тестовый набор**. Это часть данных, аналогичная тренировочному набору, но не использованная в

процессе обучения сети. То есть эта часть данных сети не знакома, и тестирование качества обучения при предъявлении новых данных является более корректным.

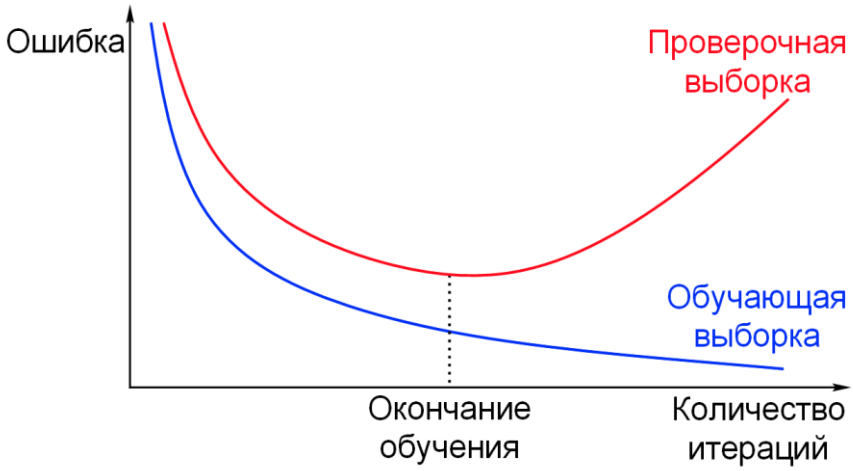



Рисунок 5 – К контролю качества обучения нейросети

Например, известный датасет MNIST [9], представляющий собой набор небольших изображений рукописных цифр форматом 28 x 28 элементов (рис. 6), имеет в своем составе тренировочный набор из 50.000 изображений (разбитых на 10 цифр) и аналогичный тестовый набор из 10.000 изображений. Обученную нейронную сеть можно запомнить в виде файла и использовать в дальнейшем для практических применений.

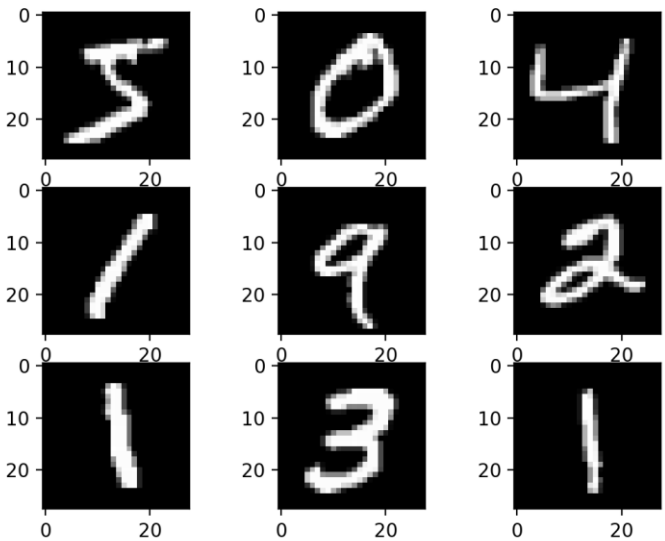


Рисунок 6 – Фрагмент датасета MNIST [9]

Процесс обучения сети может занимать значительное время, и для этого требуется высокопроизводительное оборудование (многоядерные процессоры и мощные видеокарты).

Поэтому процесс обучения можно разбить на этапы и постепенно проводить дообучение сети в целом или отдельных ее частей. Возможно также использование сторонних ресурсов, предоставляющих высокопроизводительное оборудование для ускорения процесса обучения нейронных сетей. Наиболее известная платформа, используемая для этих целей — Google Colabs.

Для оценки качества нейронных сетей часто используются специальные соревнования, которые используют один и тот же датасет (например, MNIST или ImageNet), по результатам обработки которых сравниваются эти сети (рис. 7). Обычно на таких соревнованиях появляются все более новые высокоэффективные топологии нейронных сетей [10].

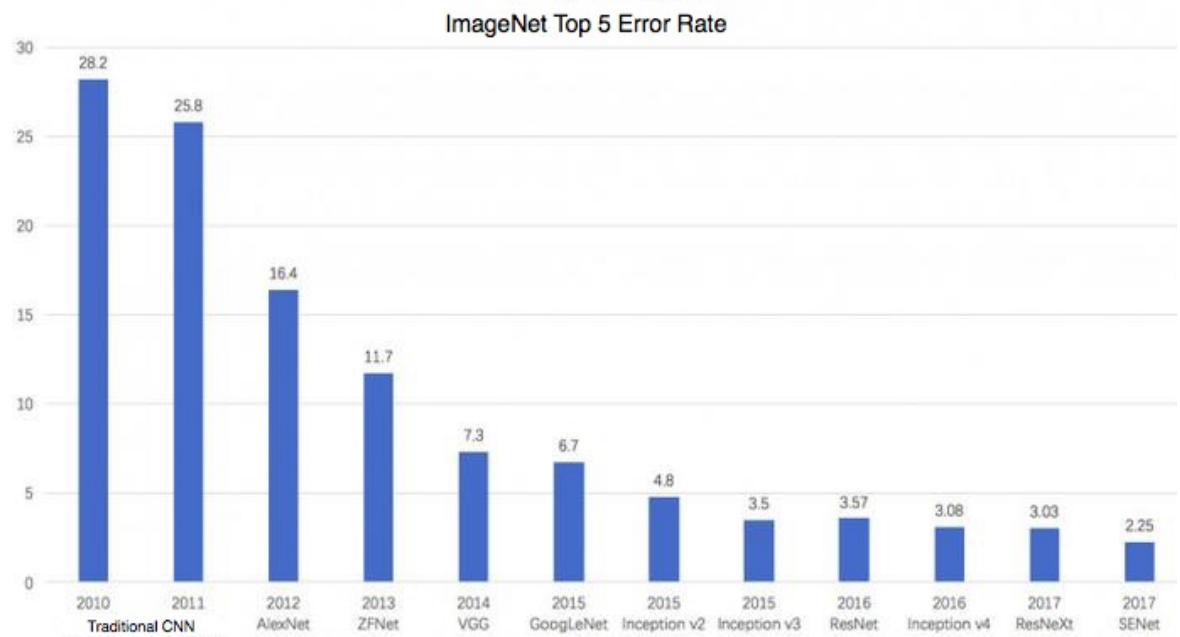



Рисунок 7 – Пример результата оценки качества нейронных сетей

Структурно нейронные сети постоянно усложняются, появляются новые архитектуры сетей со все большим количеством слоев. Построение сети становится все более сложной задачей, требующей введения большого количества параметров, а главное, знания о том, как эти параметры должны быть настроены. Для упрощения такой работы созданы специальные фреймворки, резко упрощающие работу по созданию, настройки и обучению нейронных сетей.

Одним из самых популярных фреймворков по работе с нейронными сетями является Keras. Этот фреймворк включает в себя несколько известных библиотек, в частности, таких, как TensorFlow, которые обеспечивают построение нейронной сети [2, 3].

Keras является надстройкой над этими библиотеками, которая позволяет упростить работу с нейронными сетями путем сокращения количества рутинных операций, таких как создание и настройка отдельных слоев нейронной сети.

В настоящее время технологии нейронных сетей и алгоритмы глубокого обучения совершенствуются очень быстро, поэтому информация устаревает и ее требуется регулярно обновлять.

**Порядок выполнения лабораторной работы**

Работа выполняется на персональном компьютере с установленной операционной системой Linux (Ubuntu 20.04 или старше) с подключенной к нему usb-камерой.

1. С помощью преподавателя или лаборанта включить компьютер и дождаться загрузки операционной системы.

2. На рабочем столе найти папку LR, зайти в нее и скопировать папку LR_neural_network на рабочий стол. В дальнейшем все действия по ходу работы выполнять в этой папке.

3. Пользуясь боковой панелью, запустить среду разработки Spyder и дождаться ее загрузки.

**Изучение датасета MNIST**

4. Загрузить программу test_mnist.py ру в основное окно среды разработки. Для этого следует в меню **Файл** выбрать команду **Открыть**, в появившемся окне выбрать **Рабочий стол** и в открывшемся списке выбрать папку LR_neural_network. Далее в открывшемся списке выбрать файл test_mnist.py. Посмотрите внимательно на основные элементы программы. Что она выполняет?

5. Запустите программу, нажав на зеленый треугольник. Вы увидите в окне консоли девять изображений из тренировочного набора MNIST. Рассмотрите эти изображения. Какие основные особенности этих изображений? Сделайте скриншоты.

6. Найдите в тексте программы строчки:

```
for i in range(9):
```

```
    pyplot.subplot(330 + 1 + i)
```

Замените в них значения на:

```
for i in range(5):
    pyplot.subplot(230 + 1 + i)
```

Что изменилось в работе программы? Сделайте скриншоты.

**Изучение полносвязной нейронной сети**

7. Загрузить программу Dence_NN.py в основное окно среды разработки. Для этого следует в меню **Файл** выбрать команду **Открыть**, в появившемся окне выбрать **Рабочий стол** и в открывшемся списке выбрать папку LR_neural_network. Далее в открывшемся списке выбрать файл Dence_NN.py. Посмотрите внимательно на основные элементы программы. Что она выполняет?

8. Запустите программу, нажав на зеленый треугольник. Дождитесь окончания работы программы. Рассмотрите полученные изображения в консоли, прокрутив линейку просмотра вверх. Какие результаты работы в табличном виде вы видите? Какие графики построены? Определите, достигла ли сеть стадии переобучения и на каком этапе? Сделайте скриншоты таблиц и графиков.

9. Найдите строчку в тексте программы

```
history=network.fit(train_images, train_labels,
validation_data=(test_images, test_labels), epochs=10, batch_size=64)
```

Поменяйте количество эпох на число, при котором переобучение не достигается. Запустите программу и дождитесь окончания ее выполнения. Сделайте скриншоты полученных графиков.

**Изучение сверточной нейронной сети**

10. Загрузить программу CNN.py в основное окно среды разработки. Для этого следует в меню **Файл** выбрать команду **Открыть**, в появившемся окне выбрать **Рабочий стол** и в открывшемся списке выбрать папку LR_neural_network. Далее в открывшемся списке выбрать файл CNN.py. Посмотрите внимательно на основные элементы программы. Что она выполняет?

11. Запустите программу, нажав на зеленый треугольник. Дождитесь окончания работы программы. Рассмотрите полученные изображения в

консоли, прокрутив линейку просмотра вверх. Какие результаты работы в табличном виде вы видите? Какие графики построены? Определите, достигла ли сеть стадии переобучения и на каком этапе? Сделайте скриншоты таблиц и графиков.

12. Найдите строчку в тексте программы

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

Поменяйте количество эпох на число, при котором переобучение не достигается. Запустите программу и дождитесь окончания ее выполнения. Сделайте скриншоты полученных графиков.

**Изучение готовых нейронных сетей, представленных в Keras**

13. Загрузить программу loading_models.py в основное окно среды разработки. Для этого следует в меню **Файл** выбрать команду **Открыть**, в появившемся окне выбрать **Рабочий стол** и в открывшемся списке выбрать папку LR_neural_network. Далее в открывшемся списке выбрать файл loading_models.py. Посмотрите внимательно на основные элементы программы. Что она выполняет?

14. Запустите программу, нажав на зеленый треугольник. Дождитесь окончания работы программы (первый запуск программы может занимать довольно много времени). Рассмотрите полученную информацию в консоли, прокрутив линейку просмотра вверх. Какие результаты работы в табличном виде вы видите? Какие сети представлены, какие параметры сетей вы видите. Чем они отличаются? Сделайте скриншоты таблиц.

**Изучение нейронной сети VGG16**

15. Загрузить программу VGG16.py в основное окно среды разработки. Для этого следует в меню **Файл** выбрать команду **Открыть**, в появившемся окне выбрать **Рабочий стол** и в открывшемся списке выбрать папку LR_neural_network. Далее в открывшемся списке выбрать файл VGG16.py. Посмотрите внимательно на основные элементы программы. Что она выполняет?

16. Проверьте, что в строчке с названием файла изображения выбран файл nemo0.jpg. Запустите программу, нажав на зеленый треугольник. Дождитесь окончания работы программы (первый запуск программы может занимать довольно много времени).

17. Найдите в консоли результаты работы программы. Для файла nemo0.jpg вы увидите:


Рисунок 8 – Пример результата работы программы VGG16.py

Вы видите, что на рисунке, помимо анализируемого изображения, содержится следующая информация:

Img dimension (224, 224, 3) — размерность анализируемого изображения;

ImageNet class 393 — класс изображения в датасете ImageNet (один из 1000);

[('n02607072', 'anemone_fish', 0.968795), ('n01914609', 'sea_anemone', 0.02827463), ('n09256479', 'coral_reef', 0.0016436366), ('n02606052', 'rock_beauty', 0.0008524554), ('n01950731', 'sea_slug', 0.000101410966)] — информация, содержащая пять наиболее вероятных объектов из базы данных ImageNet с указанием вероятности распознавания объектов по убыванию.

Посмотрите, удачно ли распозналось исходное изображение нейронной сетью?

Откройте страницу с классификацией изображений ImageNet [6] и найдите в ней полученный класс.

18. Замените в тексте программы файл изображения на hippo1.jpg. Запустите программу на выполнение. Найдите в консоли результаты работы программы.

19. Выполните п.п.16-18 для всех файлов изображений, указанных в таблице 1. Внесите результаты работы нейронной сети в таблицу:

Таблица 1– Результаты работы нейронной сети VGG16

| Название файла | Класс изображения ImageNet | Наиболее вероятный объект | Наибольшая вероятность объекта | Второй по вероятности объект | Вероятность второго объекта |
|---|---|---|---|---|---|
| nemo0.jpg | 393 | anemone_fish | 0.968795 | sea_anemone | 0.02827463 |
| hippo1.jpg | | | | | |
| hippo2.jpg | | | | | |
| lo1.jpg | | | | | |
| lo2.jpg | | | | | |
| lo3.jpg | | | | | |
| wb1.jpg | | | | | |
| lama1 | | | | | |
| lama2 | | | | | |
| camel1 | | | | | |
| camel2 | | | | | |
| cat1 | | | | | |
| cat2 | | | | | |
| cat3 | | | | | |
| hb1 | | | | | |
| street1 | | | | | |
| street2 | | | | | |
| face1 | | | | | |

20. Проанализируйте полученные результаты. Почему ряд объектов не распознается? Откройте страницу с классификацией изображений ImageNet [6] и найдите в ней полученный класс для таких изображений.

**Изучение работы нейронной сети с камерой в реальном времени**

21. Загрузить программу ImageNet_camera.py в основное окно среды разработки. Для этого следует в меню **Файл** выбрать команду **Открыть**, в появившемся окне выбрать **Рабочий стол** и в открывшемся списке выбрать папку LR_neural_network. Далее в открывшемся списке выбрать файл ImageNet_camera.py. Посмотрите внимательно на основные элементы программы. Что она выполняет?

22. Проверьте, что в строчке с названием файла изображения выбран файл nemo0.jpg. Запустите программу, нажав на зеленый треугольник. Дождитесь запуска программы (первый запуск программы может занимать довольно много времени). Вы получите окно с изображением, формируемым камерой, а в консоли будет выводиться информация о

времени работы распознавания изображения нейронной сетью и двух наиболее вероятных объектах распознавания и соответствующих им вероятностях.

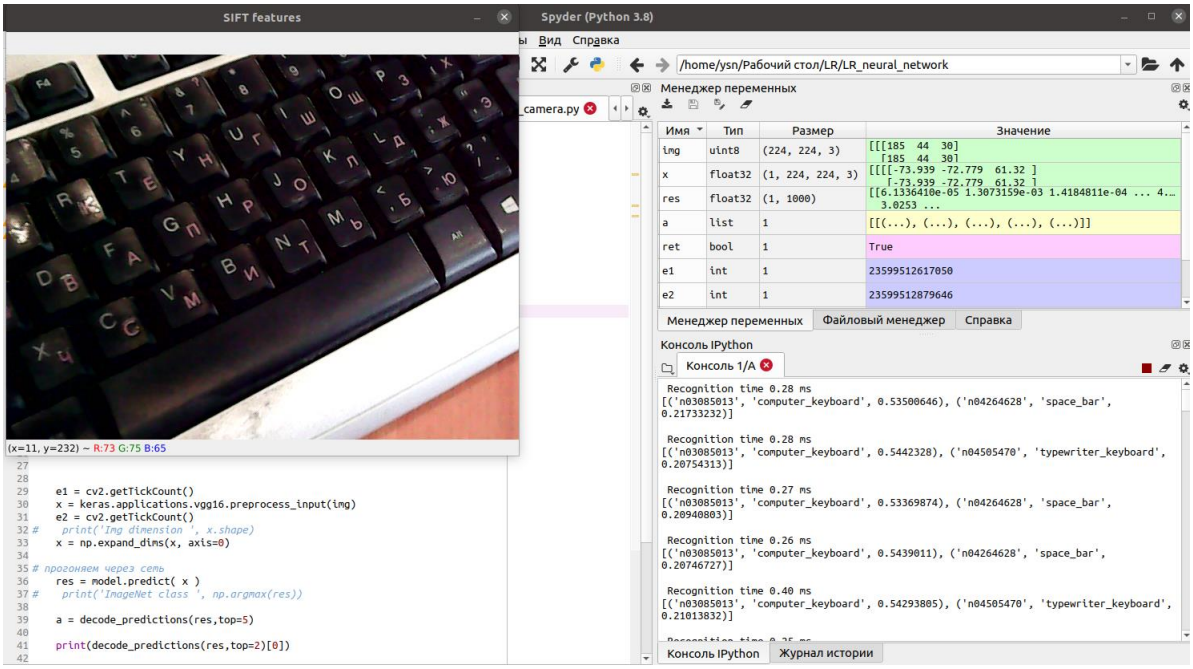



Рисунок 9 – Пример результата работы программы VGG16.py с видеокамерой

23. Наведите камеру на один из знакомых объектов (например, компьютерная мышь, очки или часы). Дождитесь, когда камера настроится на новое изображение. Получите очередное окно с изображением от видеокамеры. Посмотрите информацию о времени выполнения, распознанных предметах и вероятностях. Занесите параметры в таблицу 2.

Таблица 2 – Результаты работы нейронной сети VGG16 с камерой

| Предмет | Время выполнения | Наиболее вероятный объект | Вероятность для наиболее вероятного объекта | Второй по вероятности объект | Вероятность для второго по вероятности объекта |
|---|---|---|---|---|---|
| Клавиатура компьютера | | | | | |
| …. | | | | | |
| …. | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

24. Последовательно выберите десять знакомых объектов в вашем окружении. Направляйте на них камеру и фиксируйте информацию о времени выполнения, распознанных предметах и вероятностях в таблице 2. Для выхода из программы нажмите кнопку Q.

25. Скопируйте все полученные скриншоты на собственный носитель для составления отчета. Обратитесь к преподавателю или лаборанту для выключения компьютера.

**Содержание отчета**

1. Краткие сведения о нейронных сетях.
2. Сведения об используемом в работе датасете.
3. Таблицы с топологией использованных нейронных сетей.
4. Скриншоты и параметры исследуемых нейроных сетей.
5. Скриншоты и параметры исследуемых изображений из файлов, составленную таблицу.
2. Скриншоты и параметры исследуемых изображений, сформированных видеокамерой, составленную таблицу.
3. Краткие выводы по работе.

**Контрольные вопросы**

1. Что подразумевается под искусственной нейронной сетью? Какова структура простой нейронной сети?
2. Укажите признаки, по которым классифицируют нейронные сети. Перечислите известные Вам классификации ИНС.
3. В чем состоит задача обучения нейронной сети для решения задачи распознавания объектов на изображении?
4. Какие методы могут использоваться при обучении нейронной сети, и какие трудности при этом возникают?
5. Что такое тестовый набор данных и с какой целью он используется?
6. В чем особенность структуры нейронной сети VGG16, каковы ее свойства и параметры?

Лабораторная работа №2. **Знакомство с нейронной сетью YOLO. Основные параметры сети**

**Цели работы:**

1. Изучение теоретических основ построения и функционирования систем распознавания объектов с использованием нейронных сетей.
2. Знакомство с нейронной сетью YOLO и изучение основных параметров сети.
3. Получение практических навыков работы с основными элементами программного обеспечения, которое используется в работе.
4. Исследование основных параметров программы распознавания объектов.

**Краткие теоретические сведения**

Создание системы распознавания объектов на основе нейронной сети

Задача создания системы распознавания объектов может быть разделена на несколько этапов [1 - 3].

1. **Постановка задачи распознавания объектов.** На этом этапе следует определиться с номенклатурой распознаваемых объектов, а также с условиями работы системы распознавания, основными параметрами и характеристиками ее основных элементов, возможными ограничениями в процессе работы.

2. **Выбор архитектуры нейронной сети.** Это один из ответственных этапов работы, на котором следует выбрать тип и основные параметры нейронной сети. Учитывая большое разнообразие возможных архитектур сетей и, в особенности, скорость появления новых эффективных архитектур, следует выбрать оптимальное на момент формирования сети сочетание ее сложности, быстродействия и эффективности работы непосредственно с заданными классами объектов.

3. **Выбор или создание датасета для обучения и тестирования нейронной сети.** На этом этапе производится выбор набора обучающих данных. Прежде всего производится поиск среди уже созданных датасетов, которые представлены в открытых или платных базах данных. Ввиду все большего распространения нейронных сетей довольно высока вероятность, что именно такая или похожая задача уже решалась и уже существует соответствующий датасет, который можно использовать целиком или

адаптировать под свою задачу. Иногда можно найти даже обученную по выбранному датасету нейронную сеть. Но в общем случае следует быть готовым к тому, что датасет придется формировать под свою задачу с нуля. В этом случае потребуется выполнить поиск исходных данных в виде изображений (фотографий) или сделать необходимые фотографии самостоятельно, затем выполнить разметку полученных изображений и разместить эти данные по каталогам в соответствии со структурой датасета.

4. **Выполнить процесс обучения нейронной сети** с выбранной архитектурой с использованием созданного или выбранного датасета, точнее, его части — обучающего набора. С учетом сложности архитектуры нейронной сети и объема датасета этот процесс может занимать много времени. Требования к компьютеру для выполнения данной задачи могут оказаться достаточно высокими. Результатом обучения будет массив весов нейронной сети, который загружается в выбранную архитектуру сети.

5. **Проверка качества обучения нейронной сети.** На этом этапе обученную нейронную сеть подвергают тестированию для проверки качества обучения. Для этого используется вторая часть датасета — тестовый или проверочный набор. Часто процесс тестирования проводится совместно с этапом обучения путем их чередования в рамках каждой эпохи обучения. В этом случае также производится проверка на обнаружение факта переобучения.

6. **Перенос модели обученной нейронной сети на целевую архитектуру аппаратных средств.** Уже обученная нейронная сеть может быть перенесена на другую архитектуру, например, встроенный процессор сетевой камеры или видеорегистратора.

## Постановка задачи распознавания объектов

Предположим, что планируется использовать систему технического зрения для распознавания нескольких типов объектов в поле зрения. В составе такой системы, как правило, имеется одна или несколько камер, видеосигнал с которых оцифровывается и передается в цифровое устройство обработки на базе персонального компьютера или другой подходящей архитектуры. В готовом изделии довольно часто используется процессор для встроенных применений на основе архитектуры ARM (Advanced RISC Machines), который работает под управлением операционной системы Linux.

Однако в процессе разработки ПО для такого процессора обычно используется персональный компьютер. Для успешного переноса программного обеспечения требуется максимальная совместимость программного обеспечения для архитектур ARM и ПК. Поэтому средства разработки на ПК чаще всего также используют ОС Linux.

Любая система технического зрения независимо от ее назначения имеет несколько основных параметров, таких как разрешающая способность, минимальная освещенность на объекте или чувствительность камеры, отношение сигнал/шум, скорость смены кадров и некоторые другие. Эти параметры выбираются исходя из особенностей работы системы, количества и типа обнаруживаемых объектов.

Соответственно, выбирается тип камеры, камерного модуля или микросхемы КМОП-фотоприемника изображения. Далее исходя из предполагаемой сложности архитектуры нейронной сети, требований к быстродействию и к энергопотреблению, выбирают вычислительное устройство на базе ПК, микропроцессора ARM или других аппаратных средств.

## Выбор архитектуры нейронной сети

Для большинства задач, связанных с распознаванием объектов на изображении, основным типом архитектуры нейронной сети является **сверточная** нейронная сеть. Одним из наиболее перспективных типов сверточных нейронных сетей, разработанных под различные задачи распознавания объектов, является семейство сетей YOLO [11, 12].

Среди достоинств этого типа сетей выделяются высокое быстродействие, возможность одновременного поиска большого количества объектов на изображении, фиксация координат зоны обнаружения объектов, их классификация с указанием вероятности правильного распознавания.

В настоящее время актуальной является версия сети YOLOv5. Ее выбор представляется наиболее целесообразным для задачи распознавания объектов в поле зрения камеры в реальном времени [11].

YOLOv5 имеет несколько моделей, которые отличаются высокой вероятностью правильного распознавания объектов на изображениях различного разрешения, а, следовательно, сложностью реализации и существенным количеством весов. При этом выбор модели, в основном, связан с выбранным разрешением изображений.

Вся информация по проекту YOLOv5 имеется на сайте [13]. В том числе, имеется несколько моделей сетей YOLOv5 различной сложности. В таблице 3 приведены основные параметры нескольких, отличающихся сложностью моделей YOLO version5, обученных с использованием датасета COCO [14].

Таблица 3 Основные параметры моделей YOLO version5 [15]

| Model | size (pixels) | $mAP^{val}$ 0.5:0.95 | $mAP^{val}$ 0.5 | Speed CPU b1 (ms) | Speed V100 b1 (ms) | Speed V100 b32 (ms) | params (M) | FLOPs @640 (B) |
|---|---|---|---|---|---|---|---|---|
| YOLOv5n | 640 | 28.4 | 46.0 | **45** | **6.3** | **0.6** | **1.9** | **4.5** |
| YOLOv5s | 640 | 37.2 | 56.0 | 98 | 6.4 | 0.9 | 7.2 | 16.5 |
| YOLOv5m | 640 | 45.2 | 63.9 | 224 | 8.2 | 1.7 | 21.2 | 49.0 |
| YOLOv5l | 640 | 48.8 | 67.2 | 430 | 10.1 | 2.7 | 46.5 | 109.1 |
| YOLOv5x | 640 | 50.7 | 68.9 | 766 | 12.1 | 4.8 | 86.7 | 205.7 |
| | | | | | | | | |
| YOLOv5n6 | 1280 | 34.0 | 50.7 | 153 | 8.1 | 2.1 | 3.2 | 4.6 |
| YOLOv5s6 | 1280 | 44.5 | 63.0 | 385 | 8.2 | 3.6 | 16.8 | 12.6 |
| YOLOv5m6 | 1280 | 51.0 | 69.0 | 887 | 11.1 | 6.8 | 35.7 | 50.0 |
| YOLOv5l6 | 1280 | 53.6 | 71.6 | 1784 | 15.8 | 10.5 | 76.8 | 111.4 |
| YOLOv5x6 + TTA | 1280<br>1536 | 54.7<br>**55.4** | **72.4**<br>72.3 | 3136<br>- | 26.2<br>- | 19.4<br>- | 140.7<br>- | 209.8<br>- |

Как видно из таблицы 3, младшая из предложенных моделей YOLOv5n работает с изображениями размером до 640 пикселей по горизонтали и вертикали. Это уже является существенным достижением, если учесть, что большинство ранее разработанных сверточных нейронных сетей работают с изображениями не более 226 пикселей.

Сеть YOLOv5n имеет 1,9 млн. параметров (весов) и наибольшее быстродействие. Однако вероятность правильного распознавания mAP — наименьшая. Старшая модель YOLOv5x6+TTA работает с изображениями значительно большего формата со стороной до 1536 пикселов, что фактически дает возможность использовать видеопоток с разрешением FullHD [16]. Такая модель имеет более 200 млн. параметров (весов), имеет лучшие значения вероятности правильного распознавания mAP, но отличается значительной сложностью, что, естественно, влияет на быстродействие. Зависимости параметров распознавания и быстродействия от сложности модели показаны на рисунке 10.

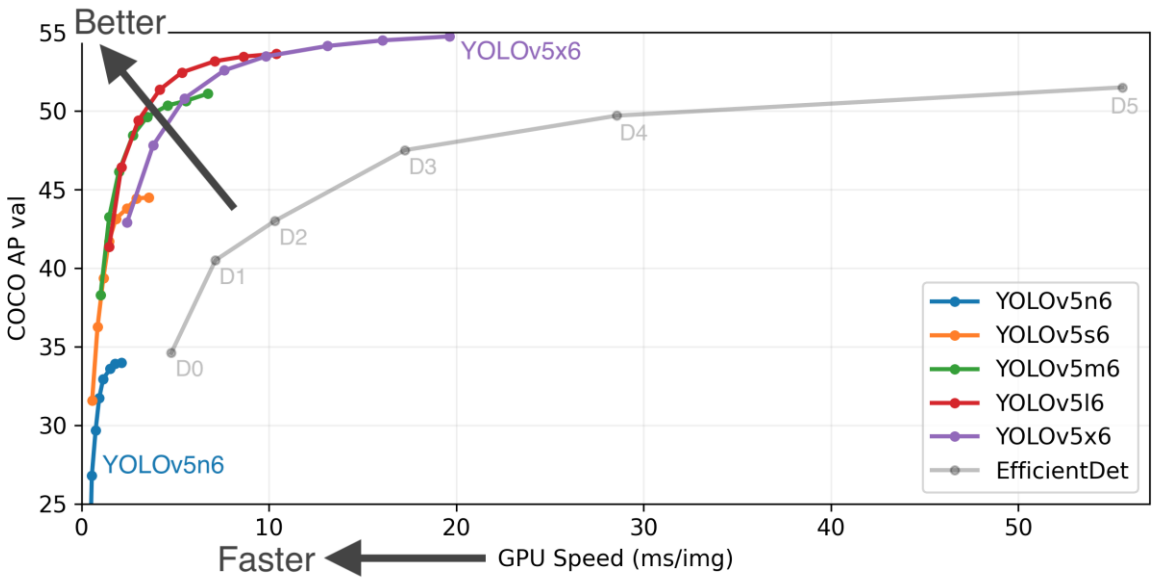



Рисунок 10. - Зависимости параметров распознавания и быстродействия от сложности модели [17]

Сложность моделей также можно оценить по размеру файлов, которые предлагаются для скачивания в формате фреймворка PyTorch и являются уже готовой к использованию обученной нейронной сетью или ее заготовкой для последующей модернизации (рис. 11).

| | |
|---|---|
| yolov5l.pt | 89.2 MB |
| yolov5l6.pt | 147 MB |
| yolov5m.pt | 40.7 MB |
| yolov5m6.pt | 68.7 MB |
| yolov5m_Objects365.pt | 43 MB |
| yolov5n.pt | 3.77 MB |
| yolov5n6.pt | 6.56 MB |
| yolov5s.pt | 14 MB |
| yolov5s6.pt | 24.5 MB |
| yolov5x.pt | 166 MB |
| yolov5x6.pt | 269 MB |

Рисунок 11 - Файлы для скачивания, содержащие веса обученных нейронных сетей YOLOv5 различной сложности и размеров [16]

В качестве демонстрации предлагаются уже обученные модели по датасету COCO [18], содержащему более 200 тысяч размеченных изображений объектов 80 категорий. Соответственно, эти примеры сетей YOLO способны классифицировать объекты 80 классов. Можно посмотреть список этих классов, и если это именно те объекты, которые вас интересуют, то для реализации можно выбрать уже готовую обученную сеть. В противном случае за основу берется необученная модель сети соответствующей сложности, у которой меняется выходной нейронный слой (классификатор) в соответствии с количеством классов распознаваемых объектов.

**Выбор или создание датасета**

Наилучшим вариантом является выбор уже имеющегося и опробованного датасета, который можно получить с одного из специализированных ресурсов, например, с сайта kaggle.com. Если такой датасет имеется, то можно приступать к обучению выбранной нейронной сети. Например, там присутствует тот самый датасет COCO, содержащий более 200 тысяч изображений объектов 80 классов. Общее количество размеченных объектов составляет 500 тысяч (рис. 12).


Рисунок 12 - Примеры размеченных изображений датасета COCO [18]

Однако изображения этого датасета носят слишком общий характер. Их сложно использовать для специализированных задач поиска и распознавания конкретных объектов. Чаще всего этот датасет используют в

качестве теста для проведения сравнения эффективности работы разрабатываемых нейронных сетей (аналогично более простому датасету MNIST [9]).

**В качестве примера поиска датасета** рассмотрим задачу создания сверточной нейронной сети для обнаружения средств индивидуальной защиты (СИЗ, медицинских масок) на лице. Используем в качестве доступной базы данных датасетов сайт kaggle.com.

По ключевым словам ***face mask*** отыскалось более 307 датасетов, из которых следует выбрать наиболее подходящий (рис.13).

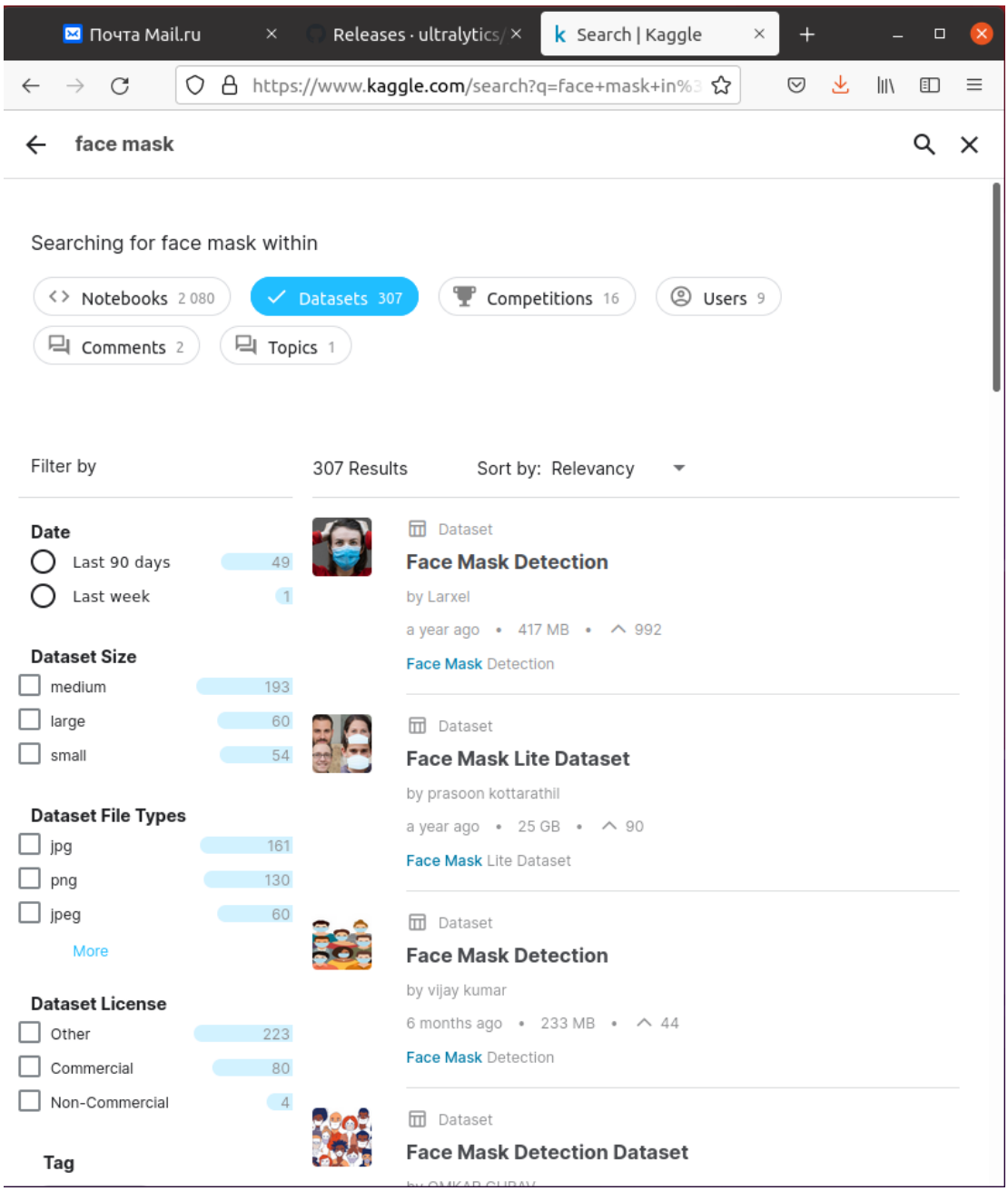



Рисунок 13 – Пример выбора датасета на сайте kaggle.com

При выборе подходящих датасетов следует принять во внимание несколько замечаний:

- датасет должен содержать достаточное количество изображений — желательно несколько тысяч или больше;
- изображения должны быть достаточно разнообразны (лучше всего, если это будут фотографии естественных объектов, снятых в условиях, близких к предполагаемым условиям работы);
- следует избегать искусственных изображений (например, имеются датасеты, предназначенные для обнаружения масок, которые включают в себя фотографии лиц без масок и такие же фотографии, на которые наложены искусственно созданные изображения масок).

Датасет должен быть размечен и соответствовать структуре, которую поддерживает фреймворк нейронной сети. В частности, формат YOLO использует текстовый формат разметки, при котором каждому изображению image_name_N.jpg соответствует текстовый файл разметки image_name N.txt (рис.14).


**000000000113.txt** (692 B)

```
56 0.409675 0.645094 0.376851 0.098875
56 0.112825 0.950797 0.17262 0.096781
0 0.19274 0.404625 0.364663 0.711062
41 0.230937 0.763773 0.124038 0.095516
41 0.860721 0.264437 0.052115 0.029313
43 0.666815 0.635563 0.113486 0.136
55 0.655144 0.779773 0.625769 0.233703
0 0.910325 0.413461 0.179351 0.512891
60 0.523293 0.806984 0.953413 0.386031
0 0.591178 0.371906 0.470192 0.6045
41 0.709892 0.169898 0.061322 0.041828
41 0.861106 0.123367 0.056346 0.034453
41 0.854519 0.172797 0.057933 0.039062
41 0.934447 0.169328 0.042404 0.036406
41 0.494615 0.236281 0.048221 0.033813
41 0.451791 0.233664 0.056659 0.038766
41 0.443882 0.187086 0.047139 0.040047
41 0.360457 0.233461 0.045096 0.034797
```

Рисунок 14 – Пример размеченного изображения в формате YOLO из датасета COCO, содержащего размеченные по 80 классам изображения. В данном примере: класс 0 — person, 41 — cup, 55 — cake и другие [19]

В таком текстовом файле содержится информация об одном или нескольких размеченных объектах, содержащихся в этом изображении, в частности, класс объекта, его координаты и размеры.

В формате YOLO разметка осуществляется в виде квадрата, в который вписан искомый объект. Формат разметки может быть другим, в этом случае с помощью специальных конвертеров можно перевести разметку из одного формата в другой. В некоторых случаях датасет вообще не размечен или размечен не так, как требуется (размечены не те объекты). В этом случае приходится выполнить разметку заново.

Если изображения не размечены, то можно воспользоваться, например, программой разметки LabelImg (рис. 15). Для этого размечаемые изображения размещаются в отдельном каталоге (например, images), а для файлов разметки создать другой каталог (например, labels).

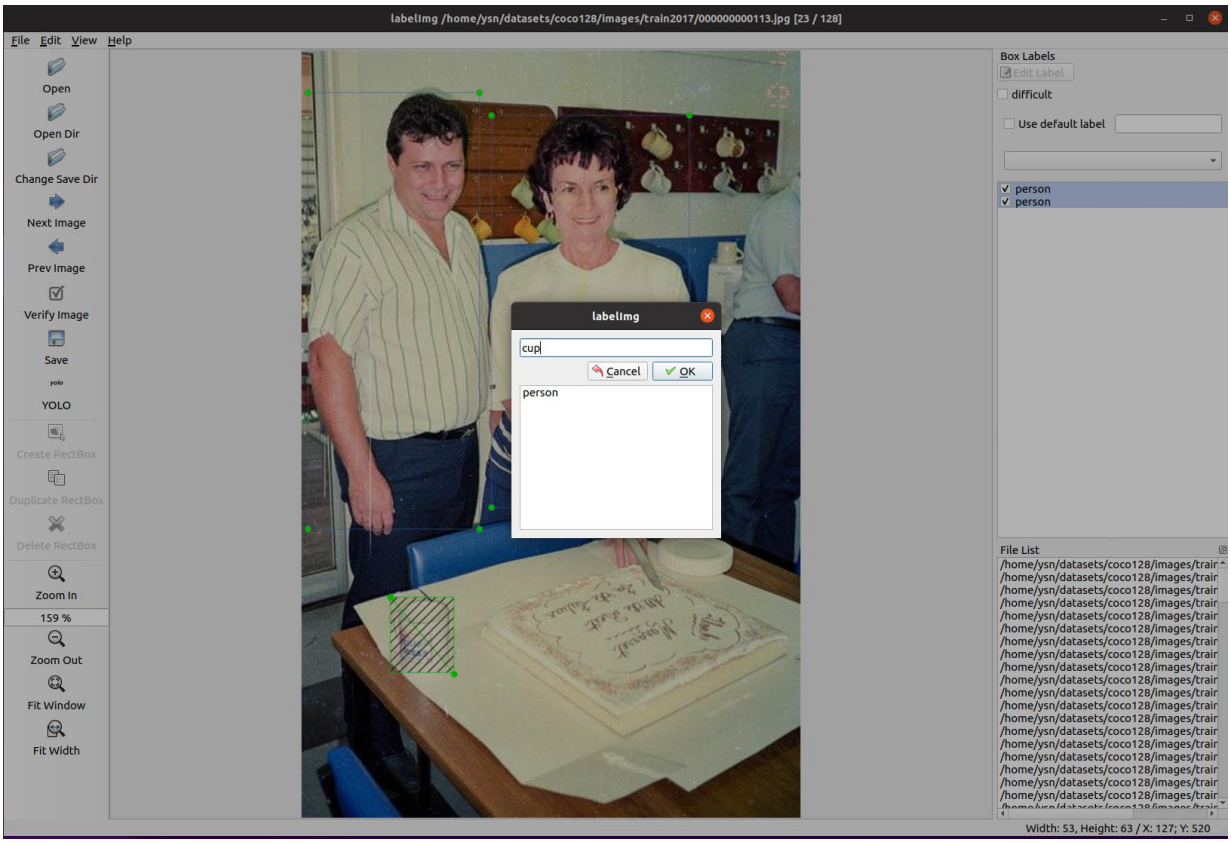



Рисунок 15 - Программа разметки labelImg

В процессе настройки программы необходимо указать эти каталоги. Работа с программой интуитивно понятна и не представляет трудностей.

По окончании разметки создается датасет, содержащий каталог с изображениями, каталог с файлами разметки, а также файл classes.txt, в котором содержится перечень классов объектов.

## Процесс обучения нейронной сети по выбранному датасету

Для нейронной сети YOLO существует свободно распространяемый проект, который можно взять, обратившись к крупнейшему веб-сервису для хостинга IT-проектов github.com, и перейти для этого по ссылке [13].

Проект включает в себя практически все необходимые элементы для создания сети YOLO, включая выбор архитектуры сети, выбор датасета для обучения, сам процесс обучения с последующей проверкой качества обучения. Проект построен на основе фреймворка глубокого обучения Torch.

Проект предусматривает работу в ОС Linux и написан на языке Python. Поэтому для успешной работы с проектом необходимо иметь ПК с ОС Linux (например, Ubuntu), а также установленную среду разработки программ на языке Python, например, Spyder, теперь входящий в состав пакета Anaconda [20].

Далее следует скопировать проект yolov5 на свой компьютер с сайта GitHub for enterprises [13]. Следуя инструкции по установке, это можно сделать командой: $ git clone https://github.com/ultralytics/yolov5. После этого весь проект будет установлен в каталог **yolov5**.

Для успешного запуска программ на языке Python, входящих в проект, потребуется установка нескольких модулей библиотек, например, OpenCV (cv2) и Torch (pytorch). Их можно установить самостоятельно, но более удобно будет воспользоваться файлом пакетной установки компонентов requirements.txt, входящего в проект **yolov5**. В этом файле содержатся все необходимые сведения для установки этих дополнительных модулей (рис.16). Для установки всех модулей следует перейти в каталог **yolov5**:

$ cd yolov5

а затем выполнить установку с использованием файла requirements.txt:

$ pip install -r requirements.txt

Чтобы убедиться в работоспособности только что установленного проекта, следует запустить программу detect.py, которая загружает одну из младших моделей нейронной сети YOLOv5, обученной по датасету COCO. Запускаем файл командой:

python3 detect.py --source 0

```
# pip install -r requirements.txt

# Base ----------------------------------------
matplotlib>=3.2.2
numpy>=1.18.5
opencv-python>=4.1.2
Pillow>=7.1.2
PyYAML>=5.3.1
requests>=2.23.0
scipy>=1.4.1
torch>=1.7.0
torchvision>=0.8.1
tqdm>=4.41.0

# Logging -------------------------------------
tensorboard>=2.4.1
# wandb

# Plotting ------------------------------------
pandas>=1.1.4
seaborn>=0.11.0

# Export --------------------------------------
# coremltools>=4.1  # CoreML export
# onnx>=1.9.0  # ONNX export
# onnx-simplifier>=0.3.6  # ONNX simplifier
# scikit-learn==0.19.2  # CoreML quantization
# tensorflow>=2.4.1  # TFLite export
# tensorflowjs>=3.9.0  # TF.js export

# Extras --------------------------------------
# albumentations>=1.0.3
# Cython  # for pycocotools https://github.com/cocodataset/cocoapi/issues/172
# pycocotools>=2.0  # COCO mAP
# roboflow
thop  # FLOPs computation
```

Рисунок 16 - Файл requirements.txt для проекта yolov5 [21]

Здесь в качестве параметра --source 0 указана веб-камера. Если в конфигурации компьютера она присутствует, то после запуска программы откроется окно просмотра видео в реальном времени с камеры (рис. 17). Если в поле зрения камеры присутствуют объекты, входящие в список классов датасета COCO, то обнаруженные в кадре объекты будут выделены рамкой, а класс объекта и вероятность правильного распознавания будут указаны сверху.

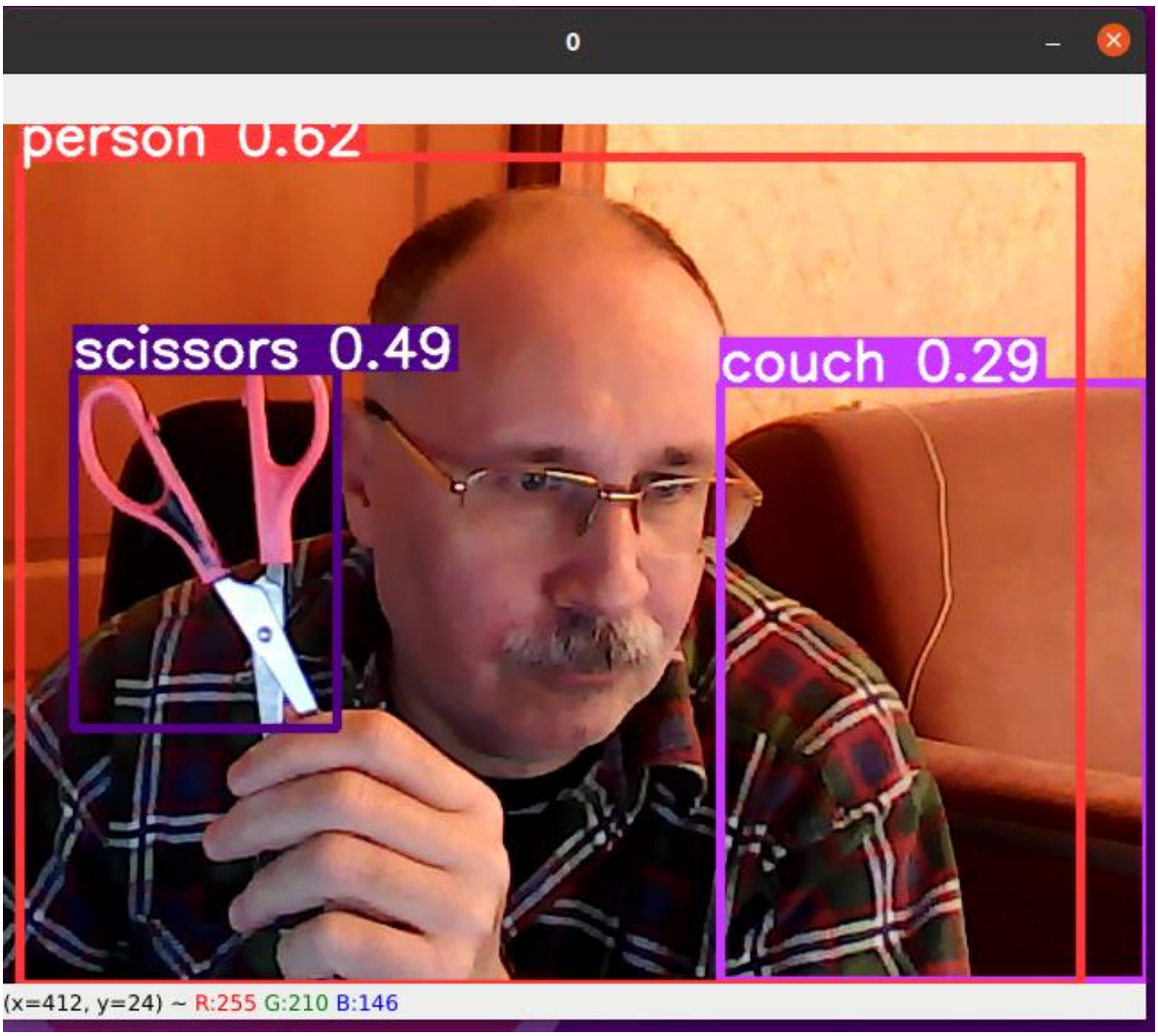



Рисунок 17 - Пример работы программы detect.py

В данном примере использована уже готовая архитектура нейронной сети YOLO в ее самом простом варианте yolov5s.pt. При этом данная сеть уже обучена по датасету COCO с использованием 80 классов объектов.

В реальной ситуации требуется провести обучение сети с использованием имеющегося датасета. С этой целью можно воспользоваться другим элементом проекта yolov5, а именно, программой обучения train.py.

Ее также можно запускать командной строкой, но гораздо удобнее воспользоваться средой разработки Spyder. Если она уже установлена на компьютере, то ее достаточно запустить, например, с панели задач Ubuntu. Затем следует выбрать файл train.py, используя путь *домашняя папка*/yolov5/train.py (рис. 18). При нажатии на зеленый треугольник программа будет запущена с параметрами по умолчанию.

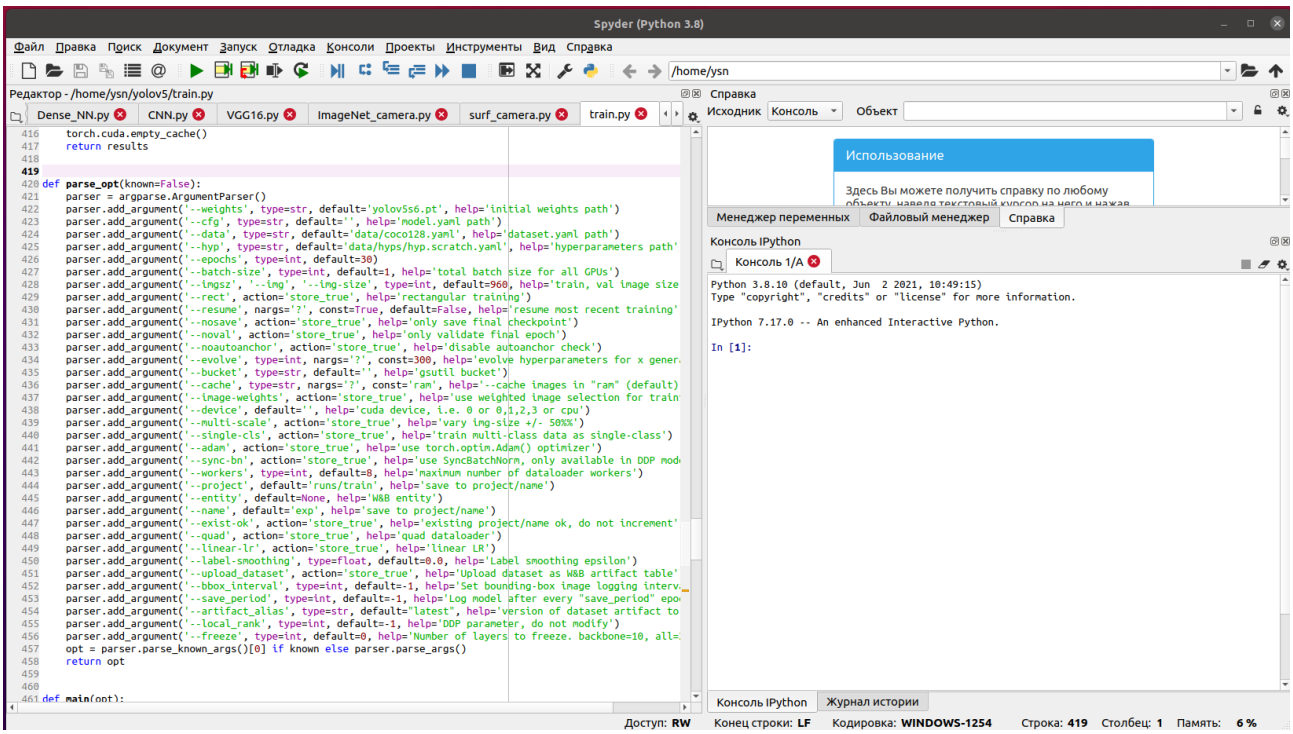



Рисунок 18 - Внешний вид среды разработки Spyder с открытой программой train.py

При запуске программы будет запущена конфигурация по умолчанию, параметры которой приведены в функции parser_opt (рис. 19). Наиболее существенными параметрами здесь являются следующие:

weights — файл, содержащий модель сети и веса, полученные в результате обучения по датасету COCO. При запуске обучения из этого файла берется архитектура сети, а веса инициализируются. По умолчанию — файл yolov5s.pt.

data — содержит путь к используемому датасету. В данном случае в качестве примера используется сильно урезанный датасет COCO128, содержащий всего 128 файлов изображений и столько же файлов разметки. Используются те же 80 классов объектов.

epochs — количество эпох обучения. Каждая эпоха содержит обучение по всем изображениям, входящим в датасет. В данном примере количество эпох составит 300.

batch-size — количество изображений, входящих в мини-пакет, на которые разделяется датасет при обучении. Мини-пакеты приходится использовать в случаях, когда датасет не может полностью разместиться в оперативной памяти компьютера. В данном примере значение batch-size выбрано 16. Соответственно, весь датасет разбивается на 8 мини-пакетов.

Imgsz — размер изображений, с которыми будет работать нейронная сеть после обучения. В данном примере размер изображений 640 пикселей.

device — задает аппаратные средства, которые используются при обучении. Если в компьютере используется мощный графический процессор Nvidia, то с использованием технологии CUDA можно достичь существенного ускорения процесса обучения (в десятки раз). Если такого ускорителя нет, то используется центральный процессор компьютера.

```
441  def parse_opt(known=False):
442      parser = argparse.ArgumentParser()
443      parser.add_argument('--weights', type=str, default=ROOT / 'yolov5s.pt', help='initial weights path')
444      parser.add_argument('--cfg', type=str, default='', help='model.yaml path')
445      parser.add_argument('--data', type=str, default=ROOT / 'data/coco128.yaml', help='dataset.yaml path')
446      parser.add_argument('--hyp', type=str, default=ROOT / 'data/hyps/hyp.scratch.yaml', help='hyperparameters path')
447      parser.add_argument('--epochs', type=int, default=300)
448      parser.add_argument('--batch-size', type=int, default=16, help='total batch size for all GPUs, -1 for autobatch')
449      parser.add_argument('--imgsz', '--img', '--img-size', type=int, default=640, help='train, val image size (pixels)')
450      parser.add_argument('--rect', action='store_true', help='rectangular training')
451      parser.add_argument('--resume', nargs='?', const=True, default=False, help='resume most recent training')
452      parser.add_argument('--nosave', action='store_true', help='only save final checkpoint')
453      parser.add_argument('--noval', action='store_true', help='only validate final epoch')
454      parser.add_argument('--noautoanchor', action='store_true', help='disable autoanchor check')
455      parser.add_argument('--evolve', type=int, nargs='?', const=300, help='evolve hyperparameters for x generations')
456      parser.add_argument('--bucket', type=str, default='', help='gsutil bucket')
457      parser.add_argument('--cache', type=str, nargs='?', const='ram', help='--cache images in "ram" (default) or "disk"')
458      parser.add_argument('--image-weights', action='store_true', help='use weighted image selection for training')
459      parser.add_argument('--device', default='', help='cuda device, i.e. 0 or 0,1,2,3 or cpu')
460      parser.add_argument('--multi-scale', action='store_true', help='vary img-size +/- 50%%')
461      parser.add_argument('--single-cls', action='store_true', help='train multi-class data as single-class')
462      parser.add_argument('--adam', action='store_true', help='use torch.optim.Adam() optimizer')
463      parser.add_argument('--sync-bn', action='store_true', help='use SyncBatchNorm, only available in DDP mode')
464      parser.add_argument('--workers', type=int, default=8, help='max dataloader workers (per RANK in DDP mode)')
465      parser.add_argument('--project', default=ROOT / 'runs/train', help='save to project/name')
466      parser.add_argument('--name', default='exp', help='save to project/name')
467      parser.add_argument('--exist-ok', action='store_true', help='existing project/name ok, do not increment')
468      parser.add_argument('--quad', action='store_true', help='quad dataloader')
469      parser.add_argument('--linear-lr', action='store_true', help='linear LR')
470      parser.add_argument('--label-smoothing', type=float, default=0.0, help='Label smoothing epsilon')
471      parser.add_argument('--patience', type=int, default=100, help='EarlyStopping patience (epochs without improvement)')
472      parser.add_argument('--freeze', type=int, default=0, help='Number of layers to freeze. backbone=10, all=24')
473      parser.add_argument('--save-period', type=int, default=-1, help='Save checkpoint every x epochs (disabled if < 1)')
474      parser.add_argument('--local_rank', type=int, default=-1, help='DDP parameter, do not modify')
475
476      # Weights & Biases arguments
477      parser.add_argument('--entity', default=None, help='W&B: Entity')
478      parser.add_argument('--upload_dataset', nargs='?', const=True, default=False, help='W&B: Upload data, "val" option')
479      parser.add_argument('--bbox_interval', type=int, default=-1, help='W&B: Set bounding-box image logging interval')
480      parser.add_argument('--artifact_alias', type=str, default='latest', help='W&B: Version of dataset artifact to use')
481
482      opt = parser.parse_known_args()[0] if known else parser.parse_args()
483      return opt
```

Рисунок 19 - Параметры по умолчанию, заданные в программе train.py

Запустим программу train.py с параметрами по умолчанию, только количество эпох уменьшим до 10.

После загрузки основных параметров программы будет построена модель нейронной сети (рис. 20). Процесс работы программы можно видеть в окне консоли программы Spyder.

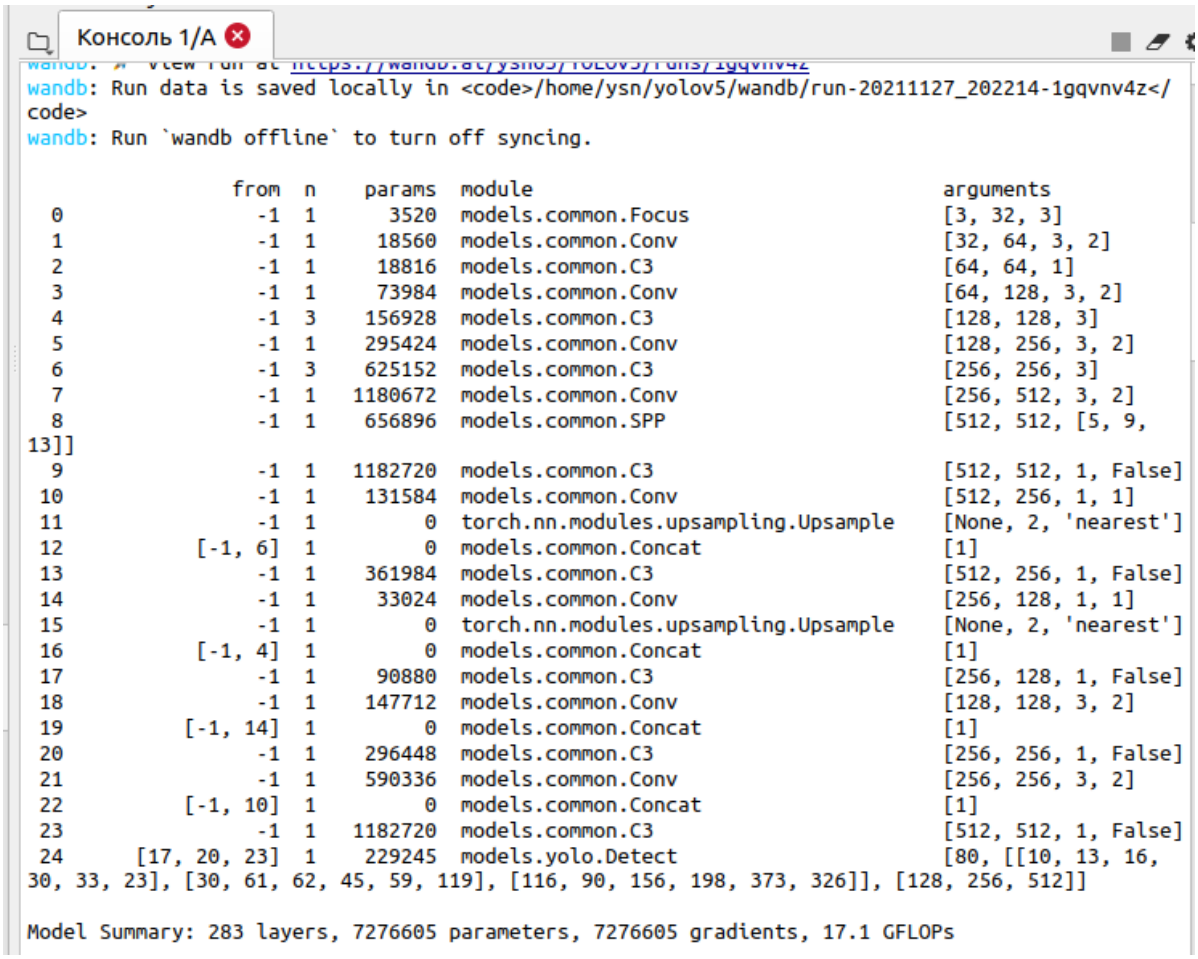



Рисунок 20 – Модель построенной нейронной сети

Затем будет произведена загрузка и анализ датасета (рис. 21). В данном случае в качестве демонстрации выбран датасет COCO128. Анализ показывает, что загружены все 128 изображений с аннотациями.

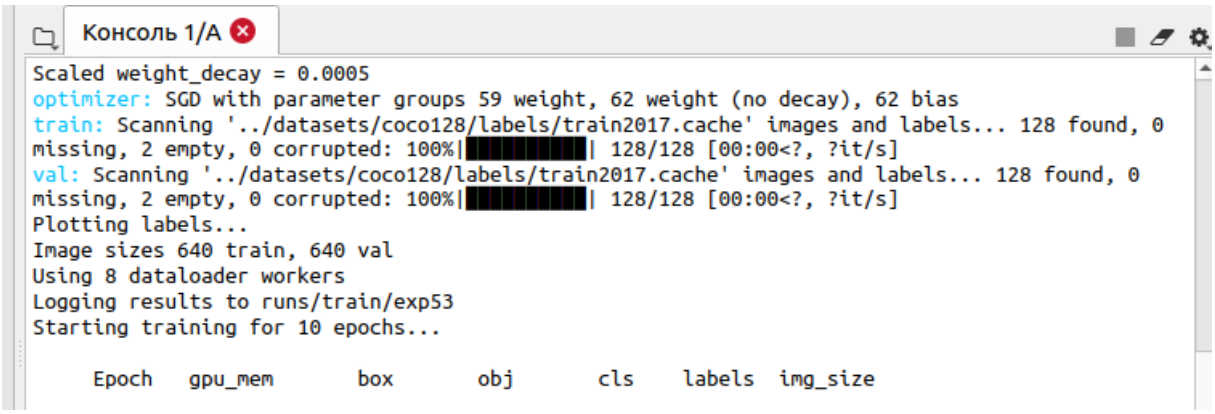



Рисунок 21 – Подтверждение загрузки изображений из датасета для обучения сети

Далее будет виден процесс обучения, который производится путем предъявления сети всех 128 изображений. На этих изображениях присутствуют 929 объектов 80 классов из датасета COCO128. После каждого цикла обучения по всему датасету, который принято называть эпохой обучения, выводятся промежуточные итоги обучения в виде значения **вероятности**.

В данном примере процесс обучения состоит из 10 эпох. По окончании десятой эпохи программа выдает сообщение об окончании обучения и сохраняет результат обучения в виде файлов, содержащих веса нейронной сети YOLO выбранной ранее архитектуры. (рис. 22).

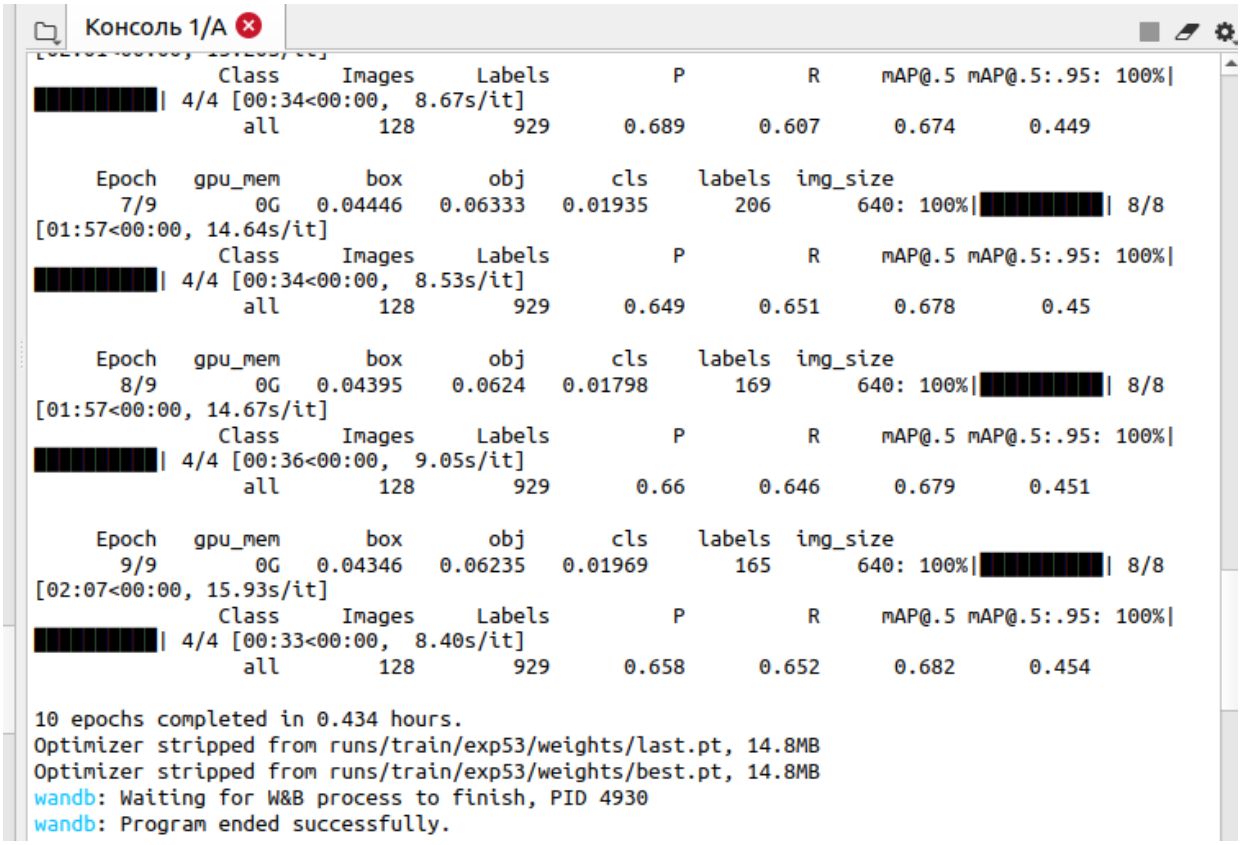



Рисунок 22 – Вид окна консоли после окончания обучения сети

Файл last.pt содержит веса после прохождения последней эпохи обучения. Файл best.pt содержит веса нейронной сети после эпохи, на которой достигнута **наименьшая ошибка**.

Кроме двух файлов, содержащих веса нейронной сети, программа формирует каталог с результатами обучения и его анализом. Этот каталог находится по адресу домашняя папка/yolov5/runs/train/exp_, при этом каждый новый запуск программы train.py будет формировать новую папку

exp__ с очередным номером. В каждой папке exp__ содержатся результаты работы программы, в том числе два вышеупомянутых файла с весами (в папке weights), а также несколько файлов в виде графиков и рисунков, иллюстрирующих процесс обучения (рис 23).

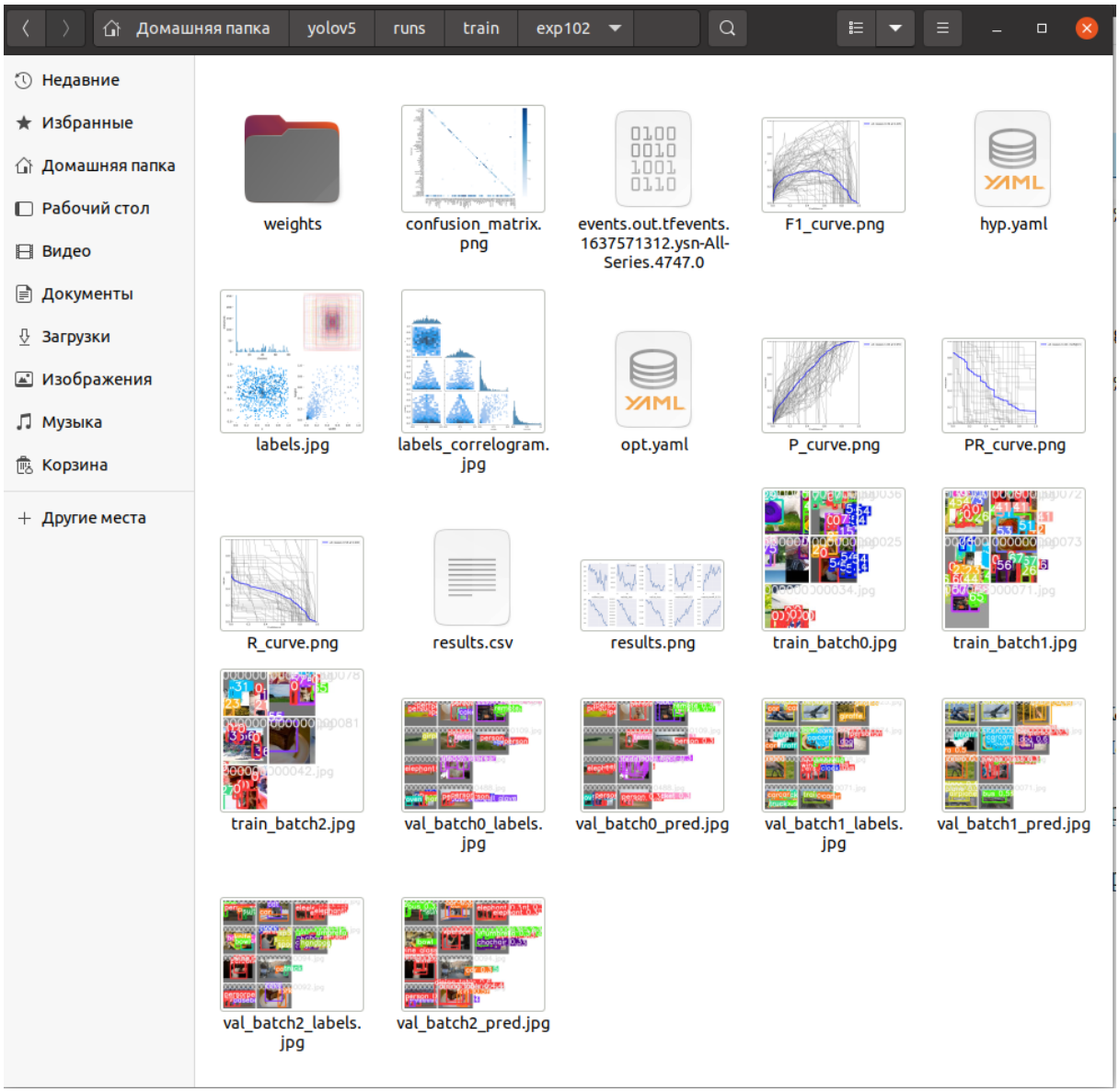



Рисунок 23 - Папка, содержащая результаты обучения модели нейронной сети, а также некоторые результаты анализа обучения

### Анализ результатов обучения

Анализ результатов обучения может быть произведен несколькими способами с разной степенью подробности.

Текущий анализ процесса обучения отображается после прохождения каждой очередной эпохи и сопровождается краткими сведениями о достигнутом уровне качества обучения (рис. 22).

Однако здесь трудно судить о результатах, за исключением того факта, что после каждой эпохи вероятность становится все выше. Если это не так, то процесс обучения нужно останавливать и разбираться, что не так с моделью сети, датасетом или гиперпараметрами. Если текущие значения вероятностей (точнее, процесс их изменения в нужную сторону) вас устраивают, то можно ждать окончания процесса обучения.

По окончании последней эпохи обучения в окне программы будет выведен итоговый блок результатов, содержащих вероятности и графические иллюстрации их изменения в процессе обучения (рис. 24).


Рисунок 24 - Итоговый блок результатов работы программы глубокого обучения train.py

Однако для подробного анализа процесса обучения данных, содержащихся в этом блоке, недостаточно. Поэтому итоговый блок следует рассматривать как вспомогательный. Подробную информацию о процессе обучения можно получить из файлов, содержащихся в папке exp N, где N — номер последнего запуска программы train.py.

Открыв эту папку, мы обнаружим вложенную папку weights (рис. 23), содержащую два файла с весами, а также несколько файлов в виде графиков и рисунков, иллюстрирующих процесс обучения.

Результаты анализа процесса обучения, которые здесь представлены, можно разделить на две группы.

В первой группе представлены результаты анализа датасета. Как уже указывалось ранее, от выбора датасета и его качества во многом зависит качество результата обучения любой нейронной сети.

В данном случае мы используем пример датасета COCO, который содержит в себе файлы изображений на основе реальных фотографий, сжатых алгоритмом JPEG. Фотографии содержат в себе изображения объектов 80 классов. Фотографии размечены в соответствии с требованиями YOLO, то есть ограничены квадратными рамками, внутри которой содержится объект.

В первую очередь следует убедиться в том, что в датасете представлены в достаточном количестве изображения всех классов, которые входят в состав датасета COCO.

В файле labels.jpg (рис. 23) представлены все 80 классов, входящих в датасет и количество объектов, входящих в каждый класс (рис. 25, левый верхний рисунок). Из этого рисунка видно, что некоторые классы изображений представлены в достаточно большом количестве, например, объект класса 0 «человек — person». В то же время видно, что некоторые классы объектов не представлены совсем — нет ни одного изображения.

Это получилось вследствие того, что в данном случае используется демонстрационный датасет COCO128, который специально уменьшен до 128 фотографий, чтобы сократить время демонстрационного процесса обучения (исходный датасет COCO имеет в своем составе 100000 фотографий). Понятно, что после обучения с использованием такого урезанного датасета наша сеть не сможет определить некоторые объекты, изображения которых отсутствуют в этом датасете.

При выборе или создании датасета нужно стремиться к равномерному распределению объектов по классам и к возможно

большему количеству и разнообразию изображений объектов по каждому классу. Как правило, это достигается в датасетах, содержащих большое количество изображений, как минимум, несколько тысяч.

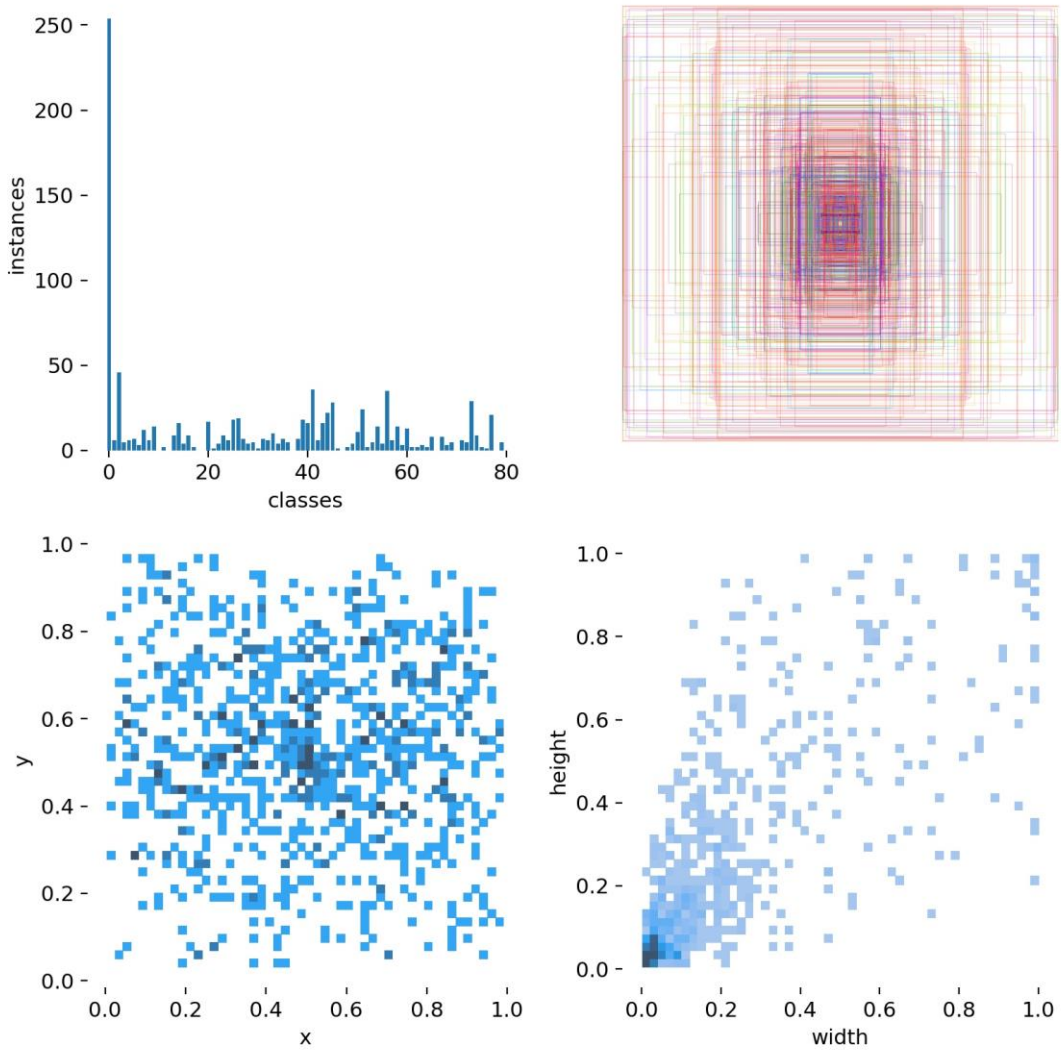



Рисунок 25 – Содержимое файла labels.jpg, отображающее результат обучения по 80 классам объектов

В ряде случаев допускается искусственное увеличение количества исходных изображений за счет изменения свойств исходных изображений, например с использованием зеркального отображения, разворота или масштабирования исходного изображения.

Кроме того, возможно создание дополнительных составных изображений, в которые входят несколько исходных изображений, подвергнутых вышеупомянутым преобразованиям. Такой процесс создания искусственных дополнительных изображений с целью увеличения разнообразия датасета применительно к глубокому обучению получил название **аугментации** (Augmentation) [1-3]. Однако процесс аугментации

имеет свои ограничения и не может полноценно заменить процесс увеличения датасета за счет увеличения количества исходных натуральных изображений. Процесс аугментации иллюстрируется примерами, которые содержатся в файлах train_batch.jpg (рис. 23, 26).

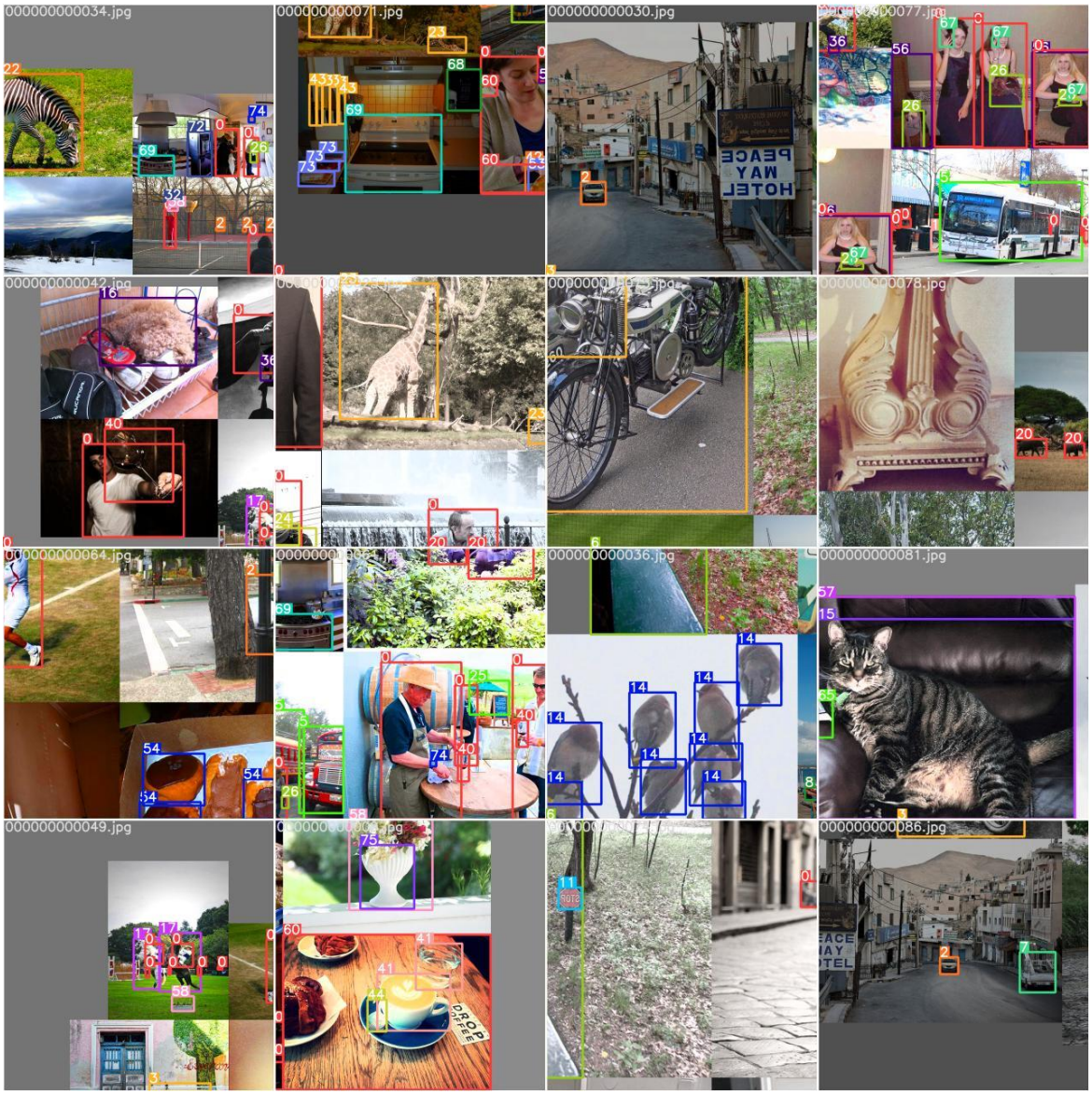


Рисунок 26 – Пример увеличения разнообразия датасета

Еще более подробные результаты процесса обучения можно получить, если настроить специальный сервис с использованием доступа к серверу wandb.ai (рис. 27).

Этот онлайн сервис предназначен для получения, хранения и визуализации данных глубокого обучения, в том числе, и для работы с рассмотренной программой train.py для обучения нейронных сетей YOLO.

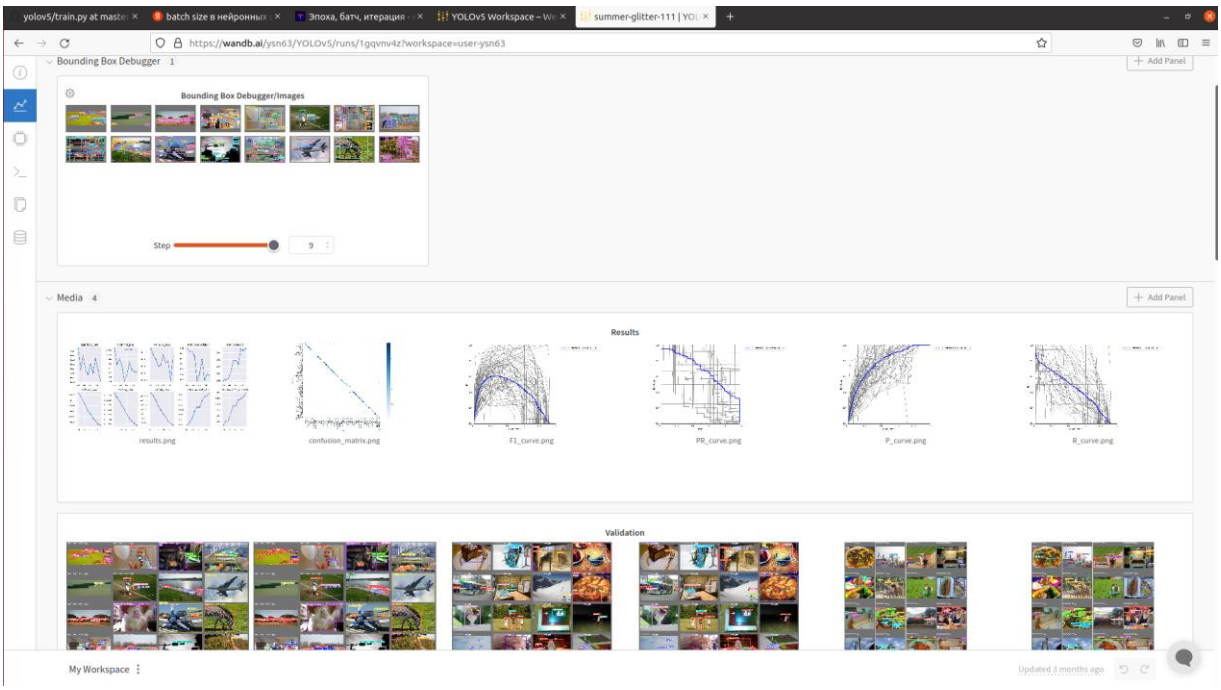



Рисунок 27 - Окно онлайн-сервиса wandb.ai, предназначенного для анализа процесса глубокого обучения нейронных сетей

Среди параметров, которые определяют качество обучения, обычно преобладают вероятностные параметры. Традиционная матрица ошибок может быть представлена как совокупность четырех событий:

Правильное обнаружение / True Positive (TP);

Правильное необнаружение / True Negative (TN);

Ложная тревога / False Positive (FP) — Ошибка I-го рода;

Пропуск события / False Negative (FN) — Ошибка II -го рода.

Эти события для одного класса объектов можно графически проиллюстрировать на рисунке 28 слева.

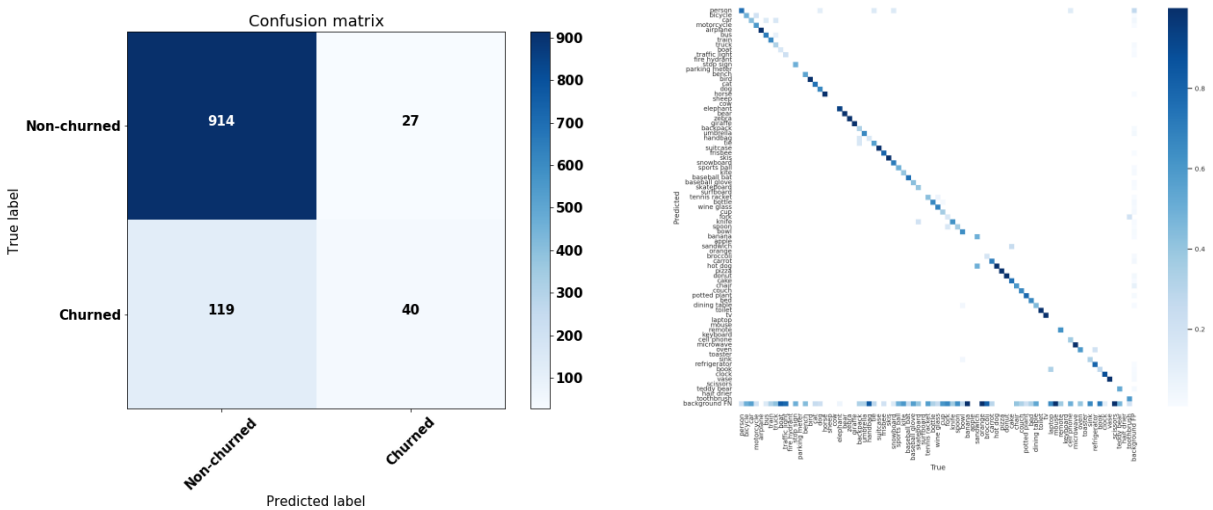



Рисунок 28 – Пример формирования матрицы ошибок

В случае задачи распознавания многих классов объектом матрица ошибок показана на рисунке 28 справа.

Обычно выделяют три основные метрики для оценки качества обучения [2, 22, 23]:

1. *Accuracy* — доля правильных результатов работы алгоритма распознавания:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}.$$

2. *Precision* — доля объектов, которые названы алгоритмом распознавания положительными и при этом являющиеся таковыми (правильное распознавание):

$$precision = \frac{TP}{TP + FP}.$$

3. *Recall* — доля объектов, которые названы алгоритмом положительными из всех объектов положительного класса, которые этот алгоритм нашел:

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

Метрику accuracy можно использовать в задачах обнаружения объектов одного класса, однако ее редко используют в задачах распознавания объектов разных классов, особенно в тех случаях, когда количество одних объектов существенно больше, чем других (несбалансированная выборка). В этом случае используются метрики precision и recall, которые не зависят от соотношения классов. При этом требуется найти оптимальный баланс между этими двумя метриками. Кроме того, на практике удобнее пользоваться одной метрикой, которая включает в себя две предыдущих.

Существует несколько способов объединения метрик *precision* и *recall* в единый критерий качества, например, в виде критерия, названного *F*-мера, которая представляет собой среднее гармоническое от значений двух критериев:

$$F_\beta = (1 + \beta^2) \cdot \frac{precision \cdot recall}{(\beta^2 \cdot precision) + recall}.$$

где β определяет вес параметра *precision* в метрике (при равных значениях веса β = 1). *F*-мера соответствует максимуму при значениях *precision* = 1 и *recall* = 1 и близка к 0, если один из параметров также близок к 0.

В процессе обучения нас может интересовать не только конечное значение этих параметров, но и их изменение в процессе обучения от эпохи к эпохе. Поэтому довольно часто можно видеть графики, которые иллюстрируют эти изменения (рис. 29).

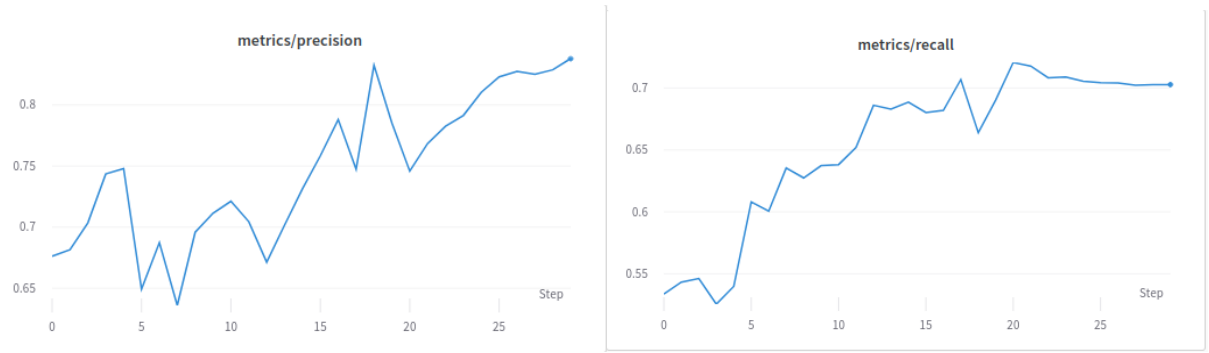



Рисунок 29 – Пример графиков изменения метрик [22]

Кроме вышеперечисленных параметров качества, для задач обнаружения объектов используются более специфичные метрики, которые получили название **mAP** (mean Average Precision) **— средняя точность.**

Здесь в качестве дополнительного параметра используется параметр Intersection over Union (IoU). Суть параметра заключается в том, что он описывает процесс обнаружения объекта, который был размечен прямоугольниками (например, в формате YOLO), и обнаруженный объект выделяется прямоугольником.

При этом параметр IoU рассматривается как отношение площади пересечения прямоугольной разметки объекта и площади объекта при его обнаружении к площади прямоугольной разметки объекта (рис. 30).

Далее выставляется пороговое значение параметра IoU, например 0,5. Превышение установленного порога соответствует положительному прогнозу. Этот процесс производится для всего датасета, и доля правильных ответов (превышающих выбранный порог 0,5) представляет собой метрику **mAP** _0.5.

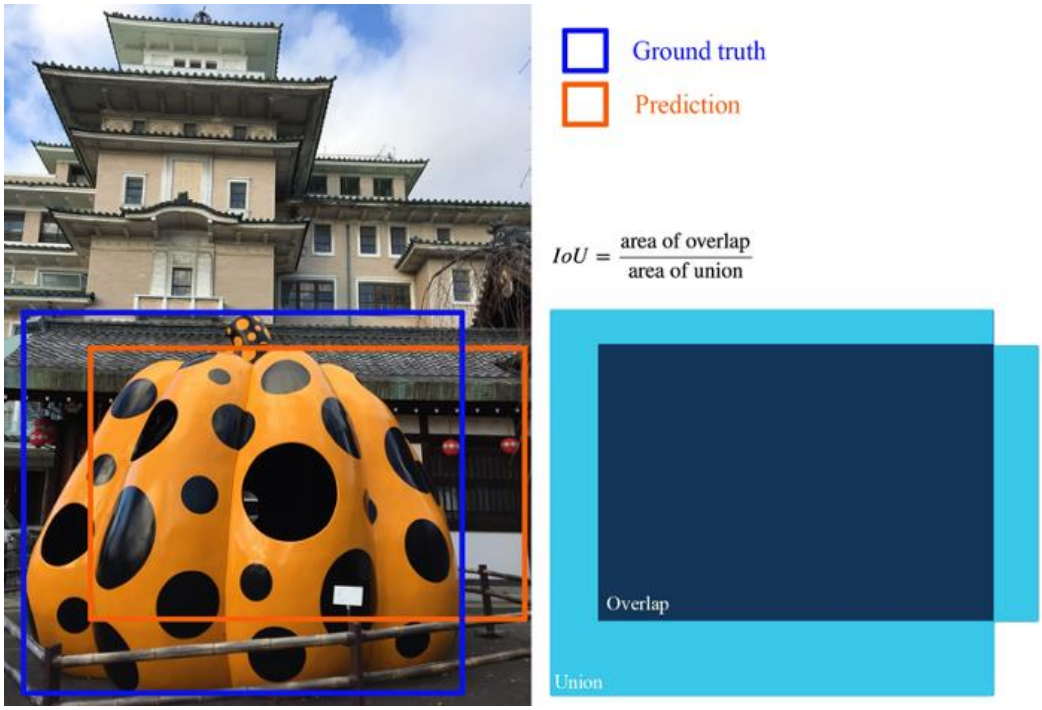



Рисунок 30 – К определению параметра IoU [22]

Более сложный вариант предполагает, что порог метрики IoU изменяется в некотором диапазоне. Чаще всего используется диапазон от 0,5 до 0,95 (рис. 31).

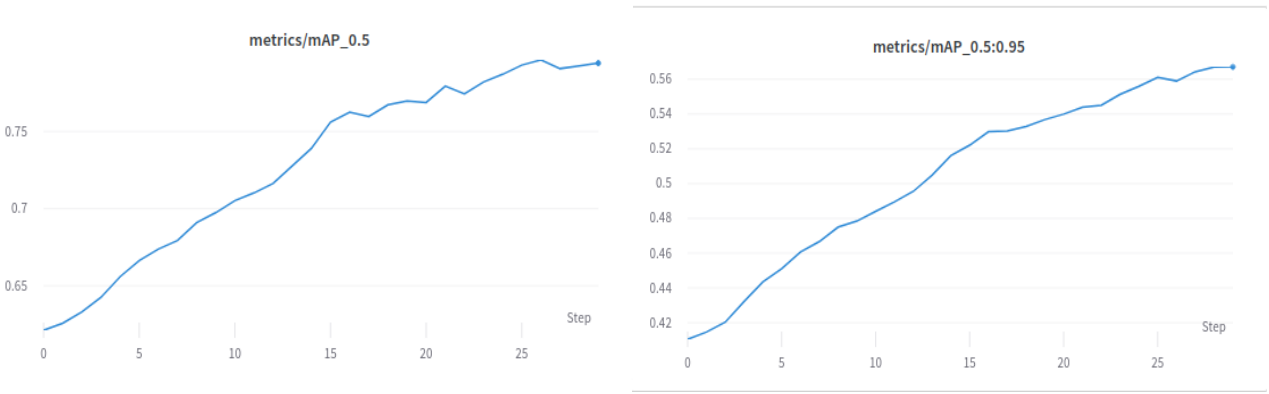



Рисунок 31 – Графики изменения средней точности при различных порогах параметра IoU [22]

**Порядок выполнения работы**

1. С помощью преподавателя или лаборанта включите компьютер и загрузите операционную систему Linux Ubuntu. Если вы ранее не работали в ОС Ubuntu, рекомендуется заранее посмотреть информацию об основах работы в этой ОС.

После загрузки ОС следует с помощью программы просмотра файлов зайти в домашнюю папку и найти в ней папку yolov5. Откройте эту папку и ознакомьтесь с ее содержимым (рис. 32)

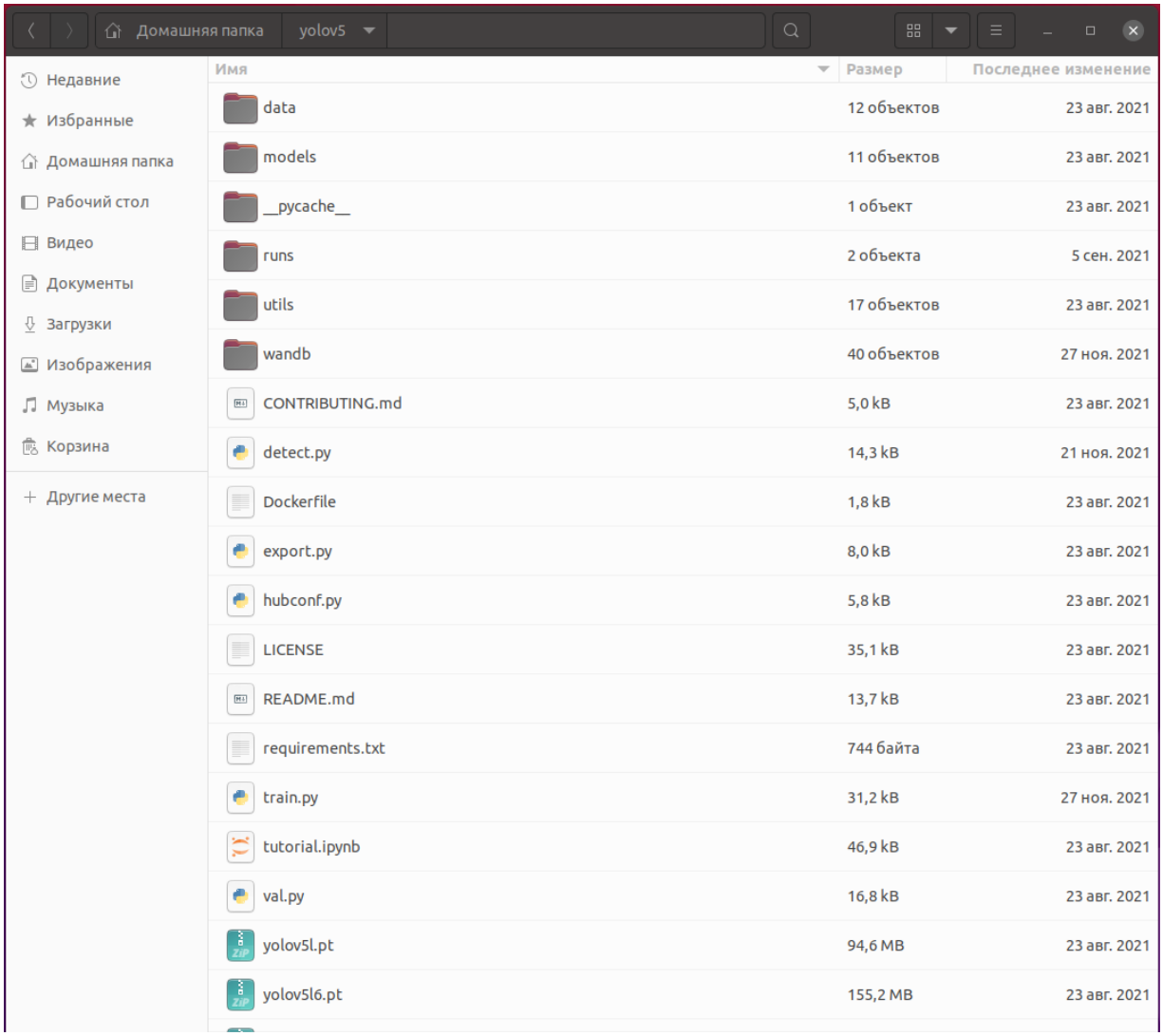



Рисунок 32 - Содержимое папки yolov5

В данной папке содержится проект, включающий в себя практически все ПО, необходимое для работы с нейронными сетями YOLO, начиная от знакомства с сетью, ее тестирования на уже заранее обученных моделях разной сложности, обучения на собственном датасете, экспорта полученных весов в другие модели, анализа основных результатов обучения сети и многое другое. Этот проект находится на сайте https://github.com/ultralytics/yolov5 и постоянно обновляется.

Для выполнения данной лабораторной работы этот проект уже установлен на локальном компьютере, и повторная его установка не требуется.

2. Откройте файл requirements.txt и ознакомьтесь с его содержимым (рис. 33).

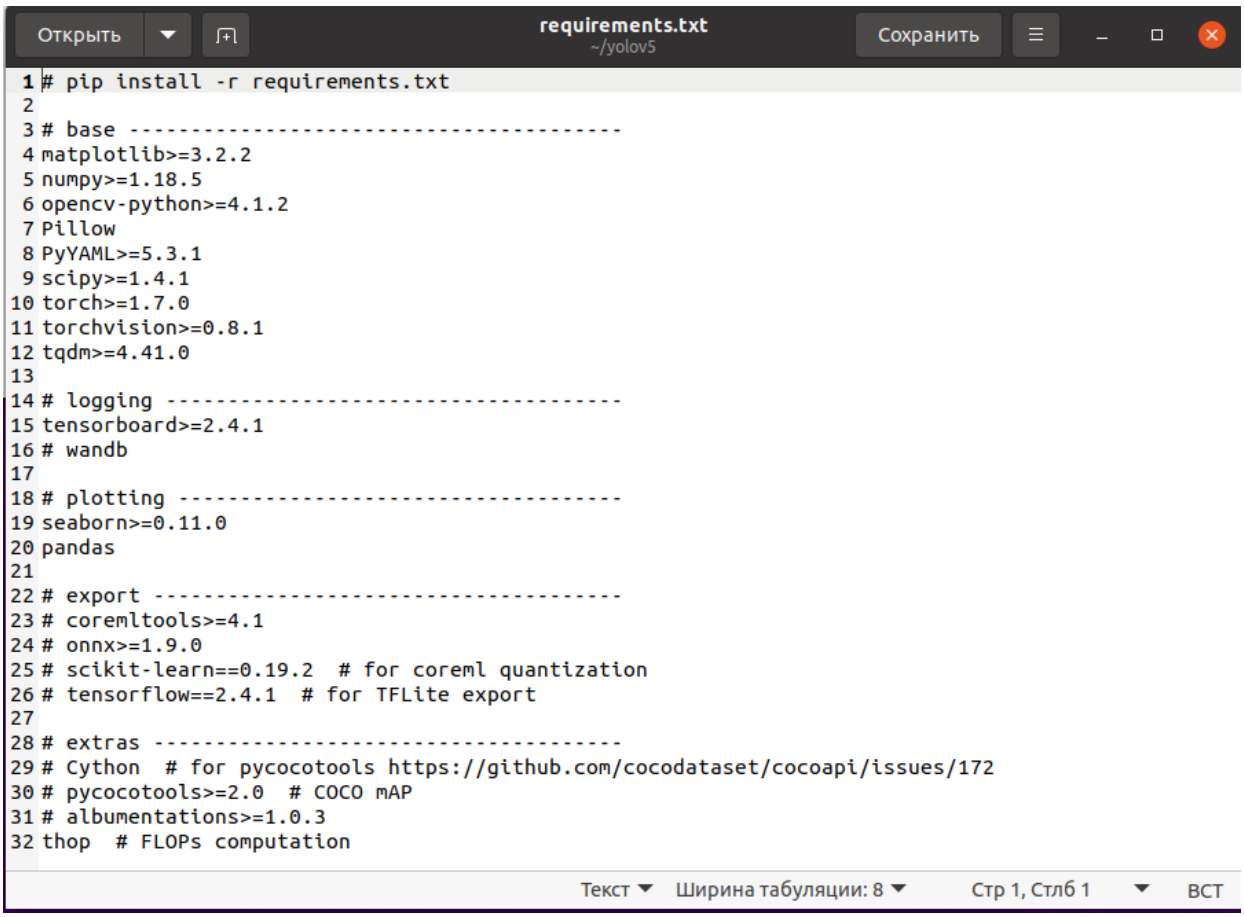



Рисунок 33 – Содержимое файла requirements.txt в окне редактора

Этот файл является пакетным файлом установки необходимых для данного проекта пакетов и библиотек. В данном случае они уже установлены на локальном компьютере. Посмотрите внимательно состав этих пакетов и библиотек. Перечислите пакеты, которые вам знакомы.

3. Убедитесь, что в папке yolov5 имеется файл detect.py. С помощью этого файла можно запустить нейронную сеть yolov5 с использованием файла, содержащего веса уже обученной сети. Согласно описанию используемого пакета, можно выбрать несколько источников изображений или видеосигналов для сети, а также выбрать файл с весовыми коэффициентами. Самый простой вариант запуска сети не требует работы в среде разработки. Можно воспользоваться командами непосредственно в терминале. Для этого откройте терминал, воспользовавшись панелью задач.

После открытия терминала по умолчанию вы находитесь в домашней папке. Наберите команду dir и посмотрите содержимое этой папки. Затем зайдите в папку yolov5, для чего наберите команду cd yolov5 и затем посмотрите содержимое этой папки командой dir (рис. 34).

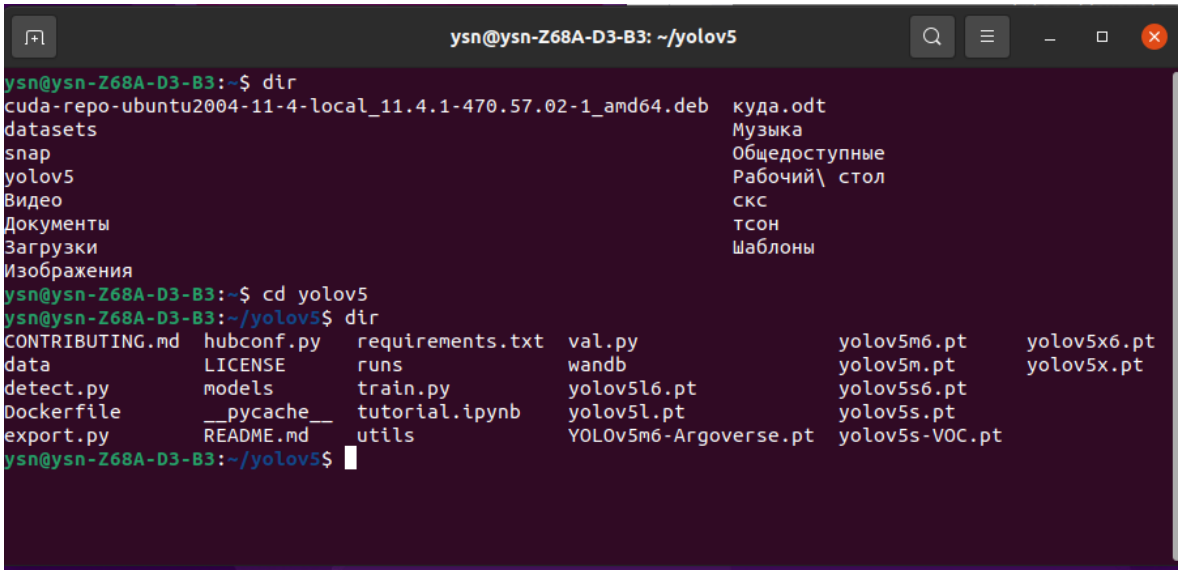



Рисунок 34 - Работа в терминале

Поскольку файл detect.py является программой, написанной на языке Python, запустить ее можно командой python3 detect.py. Программа запустится с параметрами по умолчанию (рис.35).

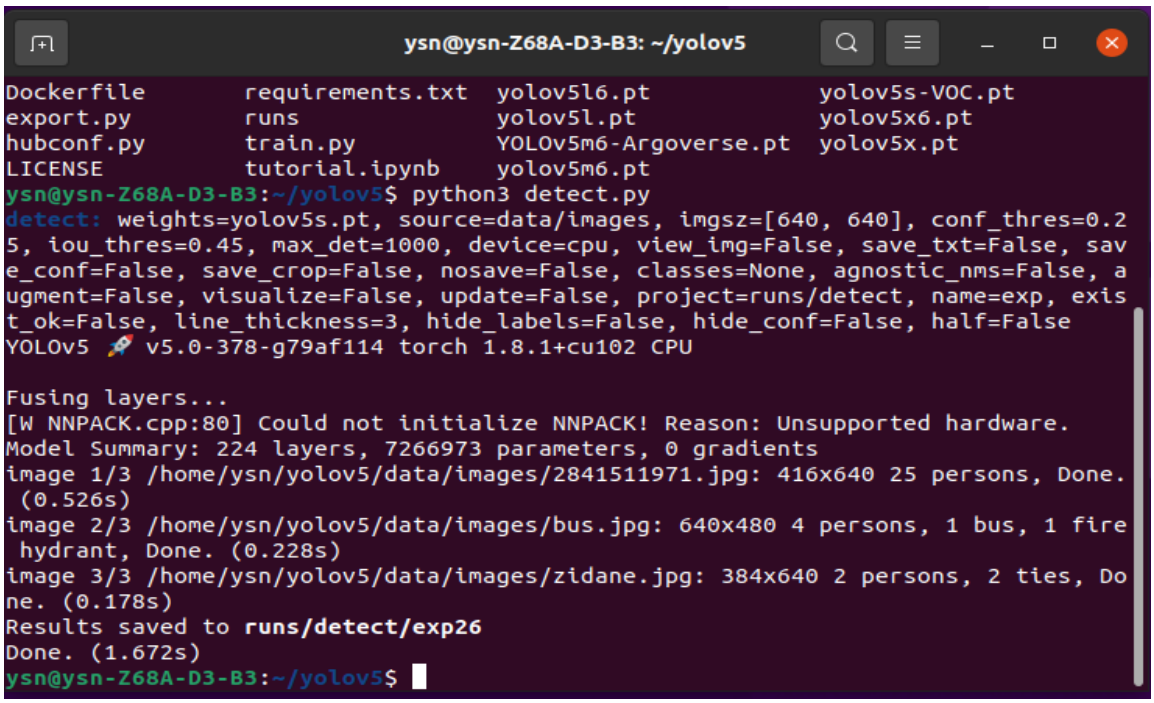



Рисунок 35 - Результат запуска программы python3 detect.py

В данном случае нейронная сеть использует файл с весами из файла yolov5s.pt, в котором содержится вся архитектура нейронной сети с предобученными весами.

Вариант по умолчанию предполагает также работу сети с файлами изображений, находящихся в папке /yolov5/data/images. В данной работе в качестве тестовых могут быть использованы различные изображения.

4. Используя программу просмотра файлов, откройте папку /yolov5/data/images и посмотрите изображения в ней (рис. 36).

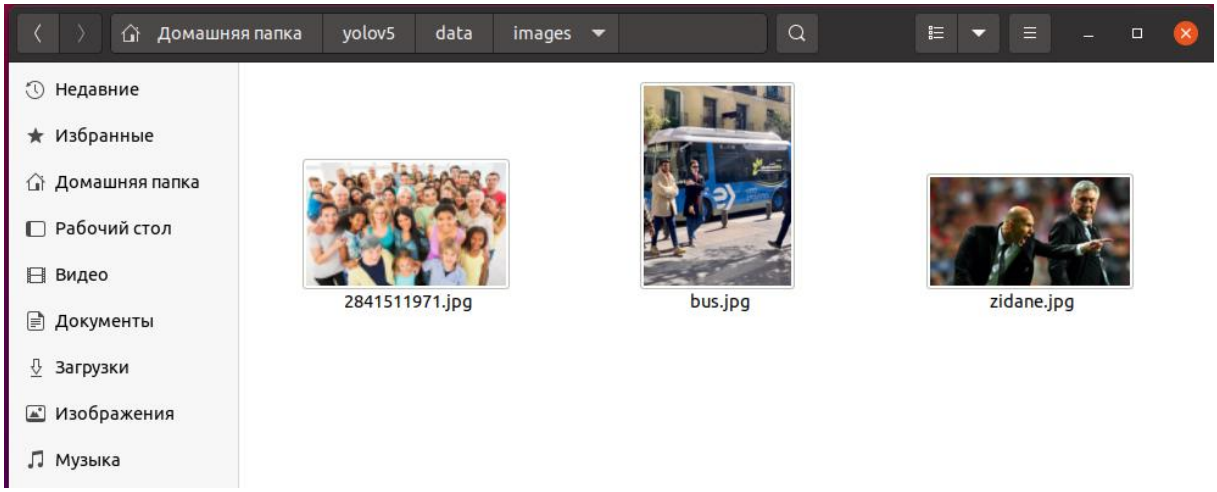



Рисунок 36 – Содержимое папка с исходными изображениями для задачи распознавания объектов

Результаты работы программы сохраняются в папках runs/detect/exp.

Откройте последнюю папку exp и посмотрите результаты работы вашей программы. Вы увидите те же изображения, на которых нейронная сеть нашла объекты, соответствующие классам объектов, используемых при обучении нейронной сети (рис. 37).

Внимательно рассмотрите полученные изображения с выделенными объектами. Отметьте характерные недостатки работы сети на этих изображениях. Обратите внимание, что для каждого обнаруженного объекта на изображении отмечается его класс и вероятность распознавания этого класса. В терминале видно также время выполнения обработки каждого изображения.


Рисунок 37 - Результат работы нейронной сети

5. Как ранее отмечалось в общем описании сети, можно использовать нейронную сеть yolo различной сложности, которые отличаются количеством используемых в ней весов (рис. 38).

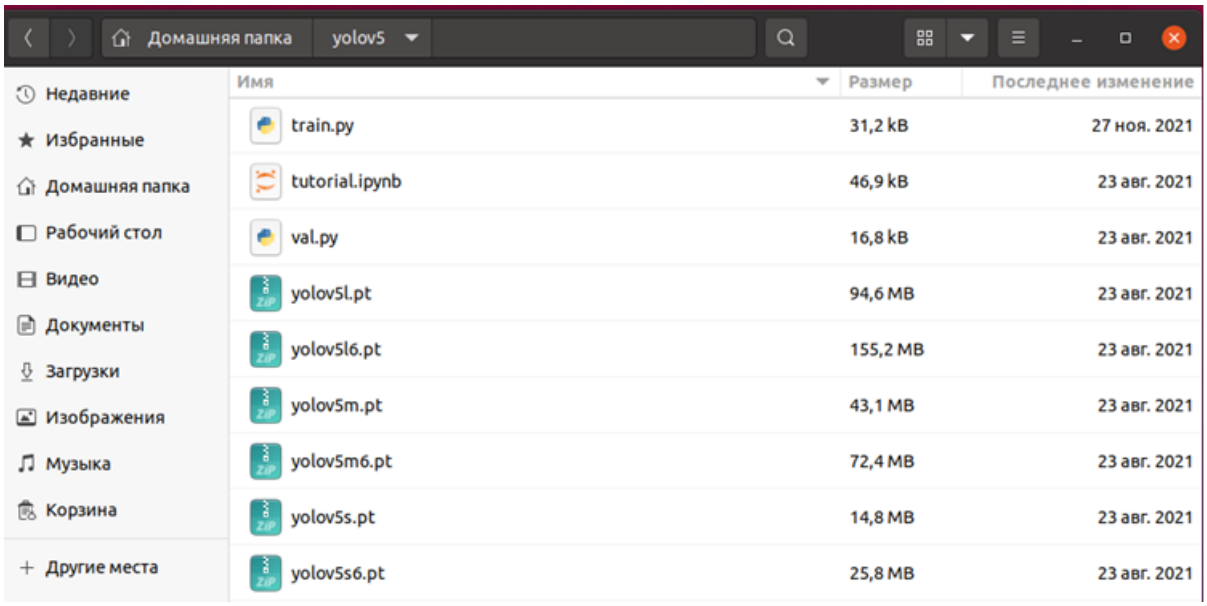



Рисунок 38 - Домашняя папка с семейством сетей YOLO

В папке yolov5 имеются несколько таких файлов с расширением *.pt. По умолчанию используется файл с архитектурой сети yolov5s.pt.

Для запуска нейронной сети с архитектурой, отличной от используемой по умолчанию, следует воспользоваться командой с параметром: python3 detect.py --weights yolov5l.pt

В данном случае запустится нейронная сеть с более сложной архитектурой yolov5l.

В проекте предусмотрены варианты архитектур, параметры которых сведены в Таблицу 3 (https://github.com/ultralytics/yolov5). Проект регулярно пополняется новыми архитектурами сетей. Соответствующие файлы с весами могут быть загружены с сайта проекта и быть запущены в программе на локальном компьютере. Используя команду с параметрами —weights, следует последовательно запустить работу нейронной сети с тестовыми изображениями и при этом использовать все доступные файлы с весами. Результаты следует свести в таблицу 4.

Таблица 4 - Результаты работы сетей YOLO различных архитектур

| Model | size (pix) | Время выполнения для изображения 1 | Время выполнения для изображения 2 | Время выполнения для изображения 3 | Время выполнения для изображения 4 | Вероятность распознавания объекта 1 | Вероятность распознавания объекта 2 | Вероятность распознавания объекта 3 |
|---|---|---|---|---|---|---|---|---|
| YOLOv5n | 640 | | | | | | | |
| YOLOv5s | 640 | | | | | | | |
| YOLOv5m | 640 | | | | | | | |
| YOLOv5l | 640 | | | | | | | |
| YOLOv5x | 640 | | | | | | | |
| YOLOv5n6 | 1280 | | | | | | | |
| YOLOv5s6 | 1280 | | | | | | | |
| YOLOv5m6 | 1280 | | | | | | | |
| YOLOv5l6 | 1280 | | | | | | | |
| YOLOv5x6 + TTA | 1280 1536 | | | | | | | |

При этом из всех распознаваемых объектов следует выбрать четыре характерных объекта разных классов, например: человек, автомобиль,

велосипед и пр. И для каждого варианта нейронной сети разной архитектуры внести в таблицу вероятность распознавания этих объектов.

Постарайтесь объяснить полученную разницу результатов для различных архитектур с учетом данных, полученных вами и с учетом параметров, приведенных в Таблице 3 теоретического описания к работе.

6. Классы объектов, которые способна распознать нейронная сеть, должны быть заложены в обучающий набор датасета, по которому производится тренировка сети. В данном случае для демонстрации работы сети yolo используется датасет COCO. Посмотрите информацию о нем, например, на сайте https://www.kaggle.com/awsaf49/coco-2017-dataset

В нашем примере информация об этом датасете берется из файла coco.yaml, который находится в папке yolov5/data (рис. 39). Там же можно найти информационные файлы для использования нескольких других датасетов.

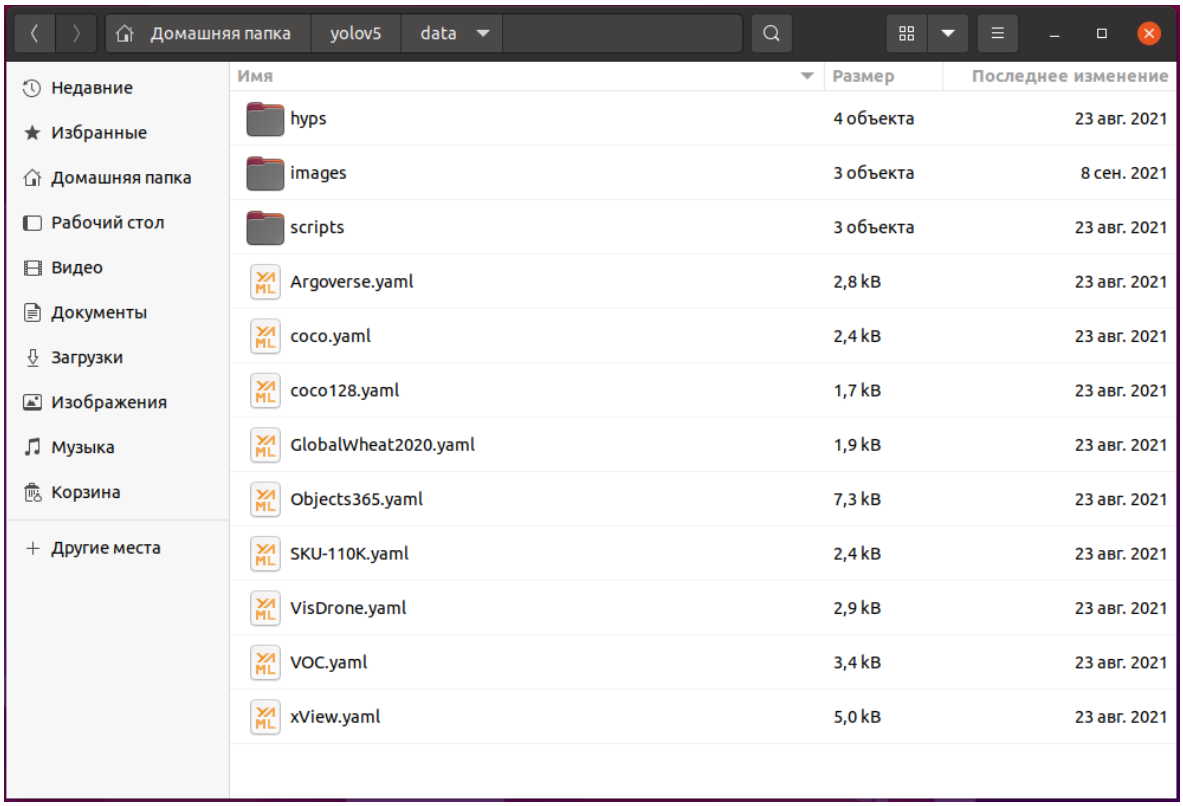



Рисунок 39 - Папка yolov5/data с информационными файлами датасетов

Откройте файл coco.yaml и посмотрите его содержимое. Вы найдете информацию о расположении файлов датасета при его скачивании, структуру каталогов, количество и состав классов распознаваемых

объектов. Посмотрите и отметьте, какие классы объектов используются в этом датасете для обучения (рис. 40).


```yaml
# YOLOv5 🚀 by Ultralytics, GPL-3.0 license
# COCO 2017 dataset http://cocodataset.org
# Example usage: python train.py --data coco.yaml
# parent
# ├── yolov5
# └── datasets
#     └── coco  ← downloads here


# Train/val/test sets as 1) dir: path/to/imgs, 2) file: path/to/imgs.txt, or 3) list: [path/to/imgs1, path/to/imgs2, ..]
path: ../datasets/coco  # dataset root dir
train: train2017.txt  # train images (relative to 'path') 118287 images
val: val2017.txt  # train images (relative to 'path') 5000 images
test: test-dev2017.txt  # 20288 of 40670 images, submit to https://competitions.codalab.org/competitions/20794

# Classes
nc: 80  # number of classes
names: ['person', 'bicycle', 'car', 'motorcycle', 'airplane', 'bus', 'train', 'truck', 'boat', 'traffic light',
        'fire hydrant', 'stop sign', 'parking meter', 'bench', 'bird', 'cat', 'dog', 'horse', 'sheep', 'cow',
        'elephant', 'bear', 'zebra', 'giraffe', 'backpack', 'umbrella', 'handbag', 'tie', 'suitcase', 'frisbee',
        'skis', 'snowboard', 'sports ball', 'kite', 'baseball bat', 'baseball glove', 'skateboard', 'surfboard',
        'tennis racket', 'bottle', 'wine glass', 'cup', 'fork', 'knife', 'spoon', 'bowl', 'banana', 'apple',
        'sandwich', 'orange', 'broccoli', 'carrot', 'hot dog', 'pizza', 'donut', 'cake', 'chair', 'couch',
        'potted plant', 'bed', 'dining table', 'toilet', 'tv', 'laptop', 'mouse', 'remote', 'keyboard', 'cell phone',
        'microwave', 'oven', 'toaster', 'sink', 'refrigerator', 'book', 'clock', 'vase', 'scissors', 'teddy bear',
        'hair drier', 'toothbrush']  # class names


# Download script/URL (optional)
download: |
  from utils.general import download, Path

  # Download labels
  segments = False  # segment or box labels
  dir = Path(yaml['path'])  # dataset root dir
  url = 'https://github.com/ultralytics/yolov5/releases/download/v1.0/'
  urls = [url + ('coco2017labels-segments.zip' if segments else 'coco2017labels.zip')]  # labels
  download(urls, dir=dir.parent)

  # Download data
  urls = ['http://images.cocodataset.org/zips/train2017.zip',  # 19G, 118k images
          'http://images.cocodataset.org/zips/val2017.zip',  # 1G, 5k images
          'http://images.cocodataset.org/zips/test2017.zip']  # 7G, 41k images (optional)
  download(urls, dir=dir / 'images', threads=3)
```

Рисунок 40 - Информационный файл для датасета COCO

7. Проверьте наличие в составе лабораторной установки видеокамеры, подключенной к порту USB.

Обратите внимание, что в параметрах программы detect.py есть возможность выбора источника изображений или видеосигнала (рис. 41):


Рисунок 41 - Выбор источников видеосигнала для файла detect.py

По умолчанию источником изображений является папка yolov5/data/images. Однако в качестве источника видеосигнала можно выбрать и видеофайл. Это может быть видео в формате mpeg4, или видео, полученное непосредственно с Youtube, или любой видеопоток протокола rtsp, полученный, например, с локальной IP-камеры или камеры, вещающей через интернет. Кроме того, в качестве источника видеосигнала может использоваться камера, подключенная к локальному компьютеру посредством стандартного видеодрайвера ОС Linux.

Запустите программу detect.py с параметром —source 0, выбрав в качестве источника видеоинформации камеру с номером 0:

python3 detect.py --source 0

Дождитесь запуска нейронной сети и откройте окно с видео.

8. Найдите вокруг себя несколько предметов, соответствующим некоторым классам из обучающего набора COCO. Используя команду с параметрами, запускайте работу нейронной сети с камерой, меняя при этом архитектуру сети. Для этого потребуется запускать программу detect.py со всеми доступными архитектурами сети, например, для сети yolo5l запускается команда с параметрами:

python3 detect.py --weights yolov5l.pt --source 0

Обратите внимание (рис. 42), что в окне с изображением отображаются обнаруженные и распознанные объекты с указанием вероятности распознавания, а в окне терминала присутствует информация о количестве и классе распознанных объектов, а также о скорости обработки одного изображения в видеопотоке.

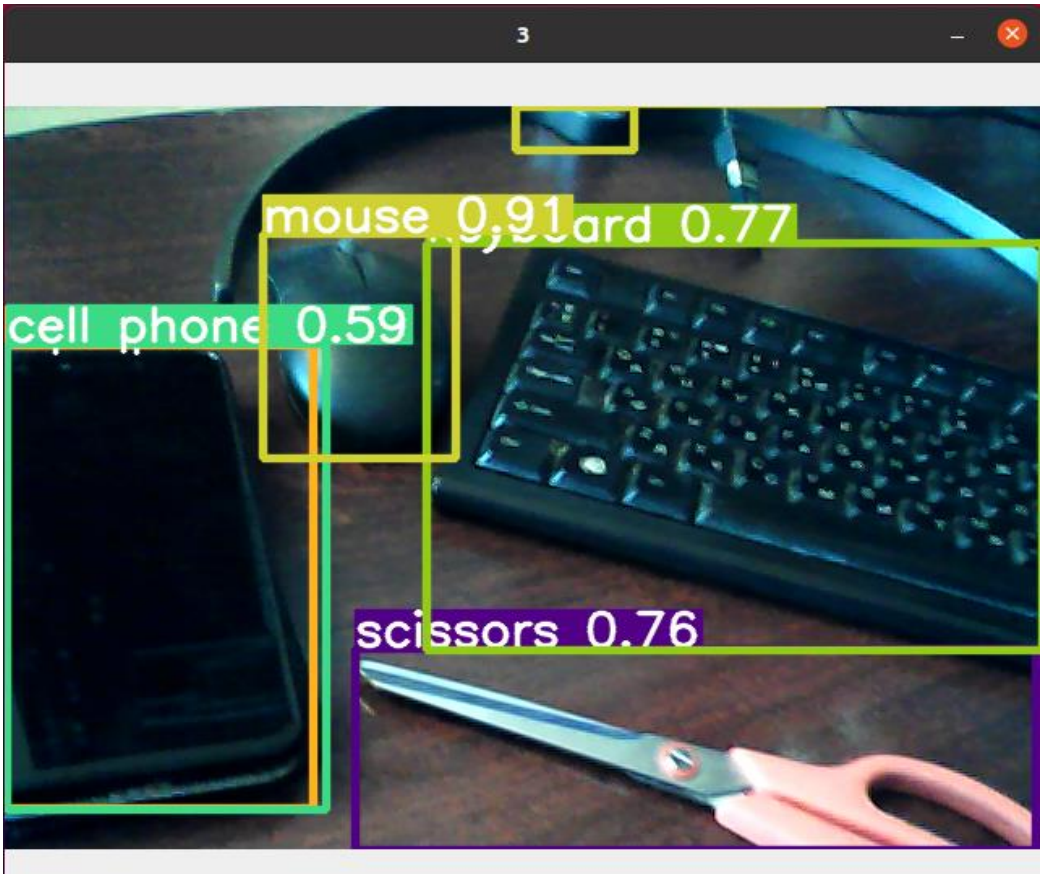



Рисунок 42 - Изображение, полученное с камеры
при запуске программы detect.py

9. Используя команду запуска с параметрами, запускайте программу с различными архитектурами сети и предъявляйте ей через камеру ваши предметы. Заполните таблицы с вероятностями распознавания различных объектов (таблица 5) и с оценкой быстродействия сетей различных архитектур (таблица 6).

В последнем случае время распознавания будет зависеть от количества обнаруженных объектов.

Таблица 5 - Вероятности распознавания различных объектов

| Model | size (pixels) | Вероятность распознавания объекта 1 | Вероятность распознавания объекта 2 | Вероятность распознавания объекта 3 | Вероятность распознавания объекта 4 | Вероятность распознавания объекта 5 | | |
|---|---|---|---|---|---|---|---|---|
| YOLOv5n | 640 | | | | | | | |
| YOLOv5s | 640 | | | | | | | |
| YOLOv5m | 640 | | | | | | | |
| YOLOv5l | 640 | | | | | | | |
| YOLOv5x | 640 | | | | | | | |
| YOLOv5n6 | 1280 | | | | | | | |
| YOLOv5s6 | 1280 | | | | | | | |
| YOLOv5m6 | 1280 | | | | | | | |
| YOLOv5l6 | 1280 | | | | | | | |
| YOLOv5x6 + TTA | 1280 1536 | | | | | | | |

Таблица 6 - Быстродействия сетей различных архитектур

| Model | size (pixels) | Время распознавания 1 объекта | Время распознавания 2 объектов | Время распознавания 3 объектов | Время распознавания 4 объектов | Время распознавания 5 объектов | | |
|---|---|---|---|---|---|---|---|---|
| YOLOv5n | 640 | | | | | | | |
| YOLOv5s | 640 | | | | | | | |
| YOLOv5m | 640 | | | | | | | |
| YOLOv5l | 640 | | | | | | | |
| YOLOv5x | 640 | | | | | | | |
| YOLOv5n6 | 1280 | | | | | | | |
| YOLOv5s6 | 1280 | | | | | | | |
| YOLOv5m6 | 1280 | | | | | | | |
| YOLOv5l6 | 1280 | | | | | | | |
| YOLOv5x6 + TTA | 1280 1536 | | | | | | | |

10. С помощью программы просмотра файлов найдите в папке yolov5 файл detect.py и откройте его. Найдите фрагмент текста программы, который позволяет запускать программу с параметрами (рис. 43)

```
def parse_opt():
    parser = argparse.ArgumentParser()
    parser.add_argument('--weights', nargs='+', type=str, default=ROOT / 'yolov5s.pt', help='model path(s)')
    parser.add_argument('--source', type=str, default=ROOT / 'data/images', help='file/dir/URL/glob, 0 for webcam')
    parser.add_argument('--data', type=str, default=ROOT / 'data/coco128.yaml', help='(optional) dataset.yaml path')
    parser.add_argument('--imgsz', '--img', '--img-size', nargs='+', type=int, default=[640], help='inference size h,w')
    parser.add_argument('--conf-thres', type=float, default=0.25, help='confidence threshold')
    parser.add_argument('--iou-thres', type=float, default=0.45, help='NMS IoU threshold')
    parser.add_argument('--max-det', type=int, default=1000, help='maximum detections per image')
    parser.add_argument('--device', default='', help='cuda device, i.e. 0 or 0,1,2,3 or cpu')
    parser.add_argument('--view-img', action='store_true', help='show results')
    parser.add_argument('--save-txt', action='store_true', help='save results to *.txt')
    parser.add_argument('--save-conf', action='store_true', help='save confidences in --save-txt labels')
    parser.add_argument('--save-crop', action='store_true', help='save cropped prediction boxes')
    parser.add_argument('--nosave', action='store_true', help='do not save images/videos')
    parser.add_argument('--classes', nargs='+', type=int, help='filter by class: --classes 0, or --classes 0 2 3')
    parser.add_argument('--agnostic-nms', action='store_true', help='class-agnostic NMS')
    parser.add_argument('--augment', action='store_true', help='augmented inference')
    parser.add_argument('--visualize', action='store_true', help='visualize features')
    parser.add_argument('--update', action='store_true', help='update all models')
    parser.add_argument('--project', default=ROOT / 'runs/detect', help='save results to project/name')
    parser.add_argument('--name', default='exp', help='save results to project/name')
    parser.add_argument('--exist-ok', action='store_true', help='existing project/name ok, do not increment')
    parser.add_argument('--line-thickness', default=3, type=int, help='bounding box thickness (pixels)')
    parser.add_argument('--hide-labels', default=False, action='store_true', help='hide labels')
    parser.add_argument('--hide-conf', default=False, action='store_true', help='hide confidences')
    parser.add_argument('--half', action='store_true', help='use FP16 half-precision inference')
    parser.add_argument('--dnn', action='store_true', help='use OpenCV DNN for ONNX inference')
    opt = parser.parse_args()
    opt.imgsz *= 2 if len(opt.imgsz) == 1 else 1  # expand
    print_args(FILE.stem, opt)
    return opt
```

Рисунок 43 - Параметры программы detect.py

Просмотрите внимательно набор параметров и постарайтесь определить, что это за параметры и на какие функции программы они влияют. Запустите программу с параметрами:

python3 detect.py --source 0 --classes 0

Посмотрите на результат работы программы. Отметьте, что изменилось в работе программы. Как это сказалось на быстродействии и вероятности распознавания объектов?
Запустите программу с параметрами:

python3 detect.py --source 0 —conf-thres 0.1

Посмотрите на результат работы программы. Отметьте, что изменилось в работе программы. Как это сказалось на быстродействии и вероятности распознавания объектов?

11. Сохраните свои данные для составления отчета.

**Содержание отчета**

1. Краткие теоретически сведения о работе сверточных нейронных сетей yolo.
2. Скриншоты основных результатов работы нейронной сети в режиме работы с одиночными кадрами.
3. Скриншоты основных результатов работы нейронной сети в режиме с потоковым видео с камеры.
4. Таблицы с результатами измерений.
5. Краткие выводы по работе.

**Контрольные вопросы**

1. Перечислите этапы решения задачи по созданию системы распознавания объектов. Какие факторы необходимо принимать во внимание при создании системы на основе нейронной сети?
2. Перечислите достоинства сетей семейства yolo. Какие параметры и особенности работы указанных сетей обеспечивают эффективность их работы при решении задач по распознаванию объектов?
3. Какие алгоритмы обучения нейронных сетей вы знаете?
4. Какие параметры используются для оценки качества работы сети?
5. Каковы особенности использования количественных и точностных метрик?

Лабораторная работа №3. **Изучение процесса обучения нейронной сети YOLO. Основные параметры процесса обучения**

**Цели работы:**

1. Изучение процесса обучения нейронной сети YOLO.
2. Исследование основных параметров процессов обучения сети.
3. Получение практических навыков работы с основными элементами программного обеспечения, которое используется в работе.
4. Исследование факторов, влияющих на качество обучения сети.

Перед началом работы следует ознакомиться с краткими теоретическими сведениями и описанием основных элементов программного обеспечения, которое используется в работе (общее описание представлено перед порядком выполнения лабораторной работы №2).

**Порядок выполнения работы**

1. С помощью преподавателя или лаборанта включите компьютер и загрузите операционную систему Linux Ubuntu. Если вы ранее не работали в ОС Ubuntu, рекомендуется заранее посмотреть информацию об основах работы в этой ОС.

После загрузки ОС следует с помощью программы просмотра файлов зайти в домашнюю папку и найти в ней папку yolov5. Откройте эту папку и ознакомьтесь с ее содержимым.

В данной папке содержится проект, включающий в себя практически все ПО, необходимое для работы с нейронными сетями YOLO, начиная от знакомства с сетью, ее тестирования на уже заранее обученных моделях разной сложности, обучения на собственном датасете, экспорт полученных весов в другие модели, анализ основных результатов обучения сети и многое другое. Этот проект находится на сайте https://github.com/ultralytics/yolov5 и постоянно обновляется.

Для выполнения данной лабораторной работы этот проект уже установлен на локальном компьютере, и повторная его установка не требуется.

2. Убедитесь, что в папке yolov5 имеется файл train.py. С помощью этого файла можно запустить процесс обучения нейронной сети yolov5.

Самый простой вариант запуска сети не требует работы в среде разработки. Можно воспользоваться командами непосредственно в терминале.

Для этого откройте терминал, воспользовавшись панелью задач (рис. 44). После открытия терминала по умолчанию вы находитесь в домашней папке. Наберите команду dir и посмотрите содержимое этой папки. Затем зайдите в папку yolov5, для чего наберите команду cd yolov5 и затем посмотрите содержимое этой папки командой dir.

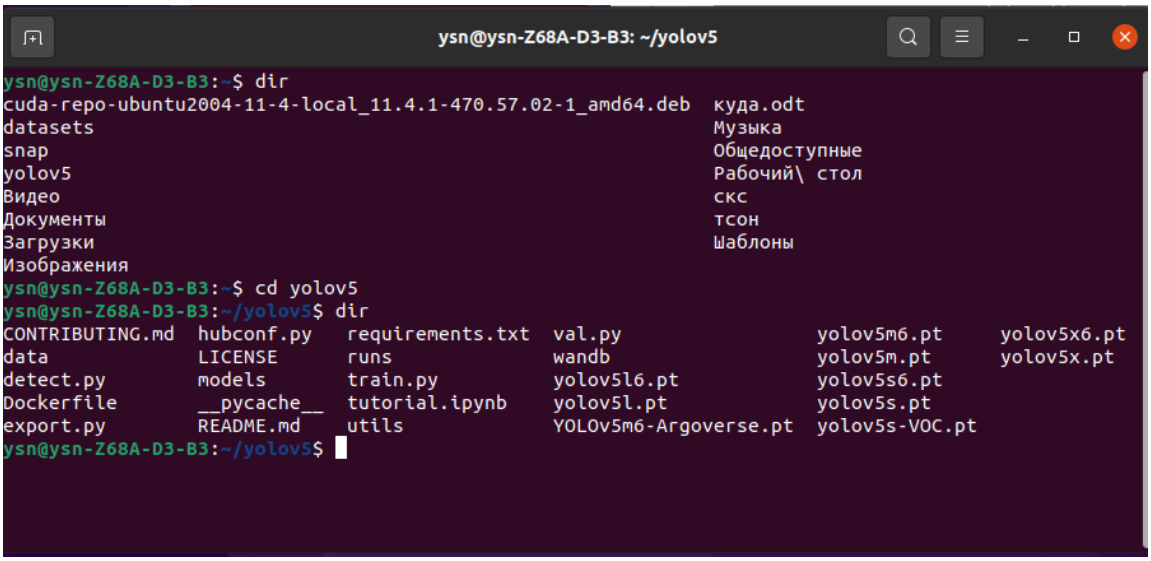



Рисунок 44 - Работа в терминале

Программа train.py позволяет провести обучение нейронной сети YOLO с использованием выбранной архитектуры сети, выбранного датасета для обучения, количества эпох обучения, размера минипакета и многих других параметров.

По умолчанию для демонстрации процесса обучения используется усеченная версия датасета COCO, содержащая всего 128 изображений (COCO128). При этом вряд ли получится достигнуть хороших результатов обучения, но можно хорошо проиллюстрировать этот процесс в реальном времени.

В реальной ситуации на практике используется для обучения датасет значительно большего размера (например, тот же датасет COCO, содержащий 100.000 изображений). Но в этом случае процесс обучения будет очень длительный.

4. Ознакомьтесь с используемым в работе датасетом COCO128. Для этого с помощью программы просмотра файлов откройте папку

datasets\coco128. Внутри будут находиться две папки: images, содержащие изображения датасета, и labels, содержащие файлы разметки. Посмотрите содержимое этих файлов и отметьте, что вы видите в файлах разметки.

Найдите в папке yolov5 папку data и откройте ее. Найдите и откройте файл описания coco128.yaml. Откройте этот файл и рассмотрите его содержимое (рис. 45).

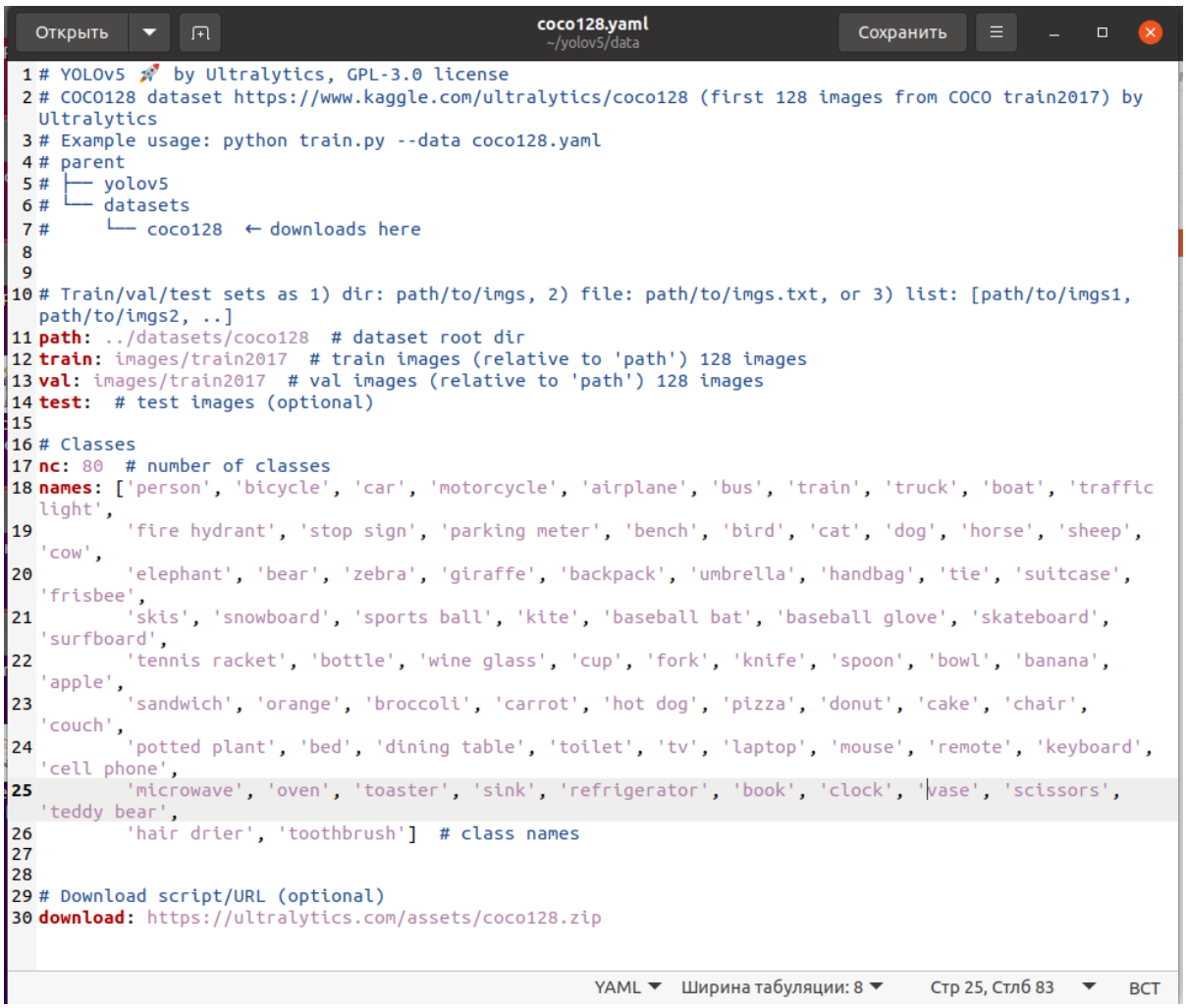



Рисунок 45 - Содержимое файла coco128.yaml

Объясните, какую информацию содержит это файл (более подробную информацию о структуре датасета можно найти в описании к лабораторной работе №4).

5. Запустите в окне терминала файл train.py. Поскольку файл detect.py является программой, написанной на языке Python, запустить ее можно командой:

python3 detect.py

Программа запустится с параметрами по умолчанию.

Поскольку изначально программа train.py рассчитана на использование сравнительно мощного компьютера с большой производительностью и большой оперативной памятью, может получиться так, что программа полностью не запустится и выдаст сообщение об ошибке (рис. 46).

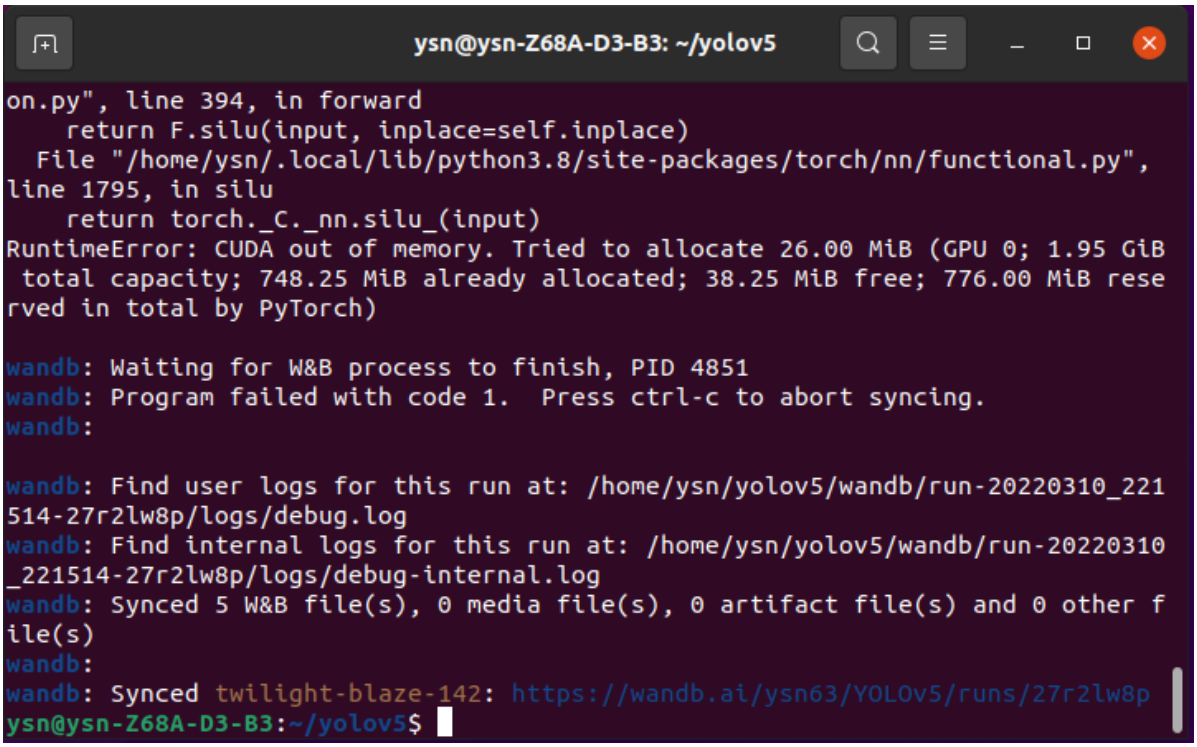



Рисунок 46 - Возможное сообщение об ошибке после первого старта программы train.py с параметрами по умолчанию

6. С помощью программы просмотра файлов найдите в папке yolov5 файл train.py и откройте его. Ближе к концу программы найдите фрагмент, в котором описаны параметры запуска (рис. 47).

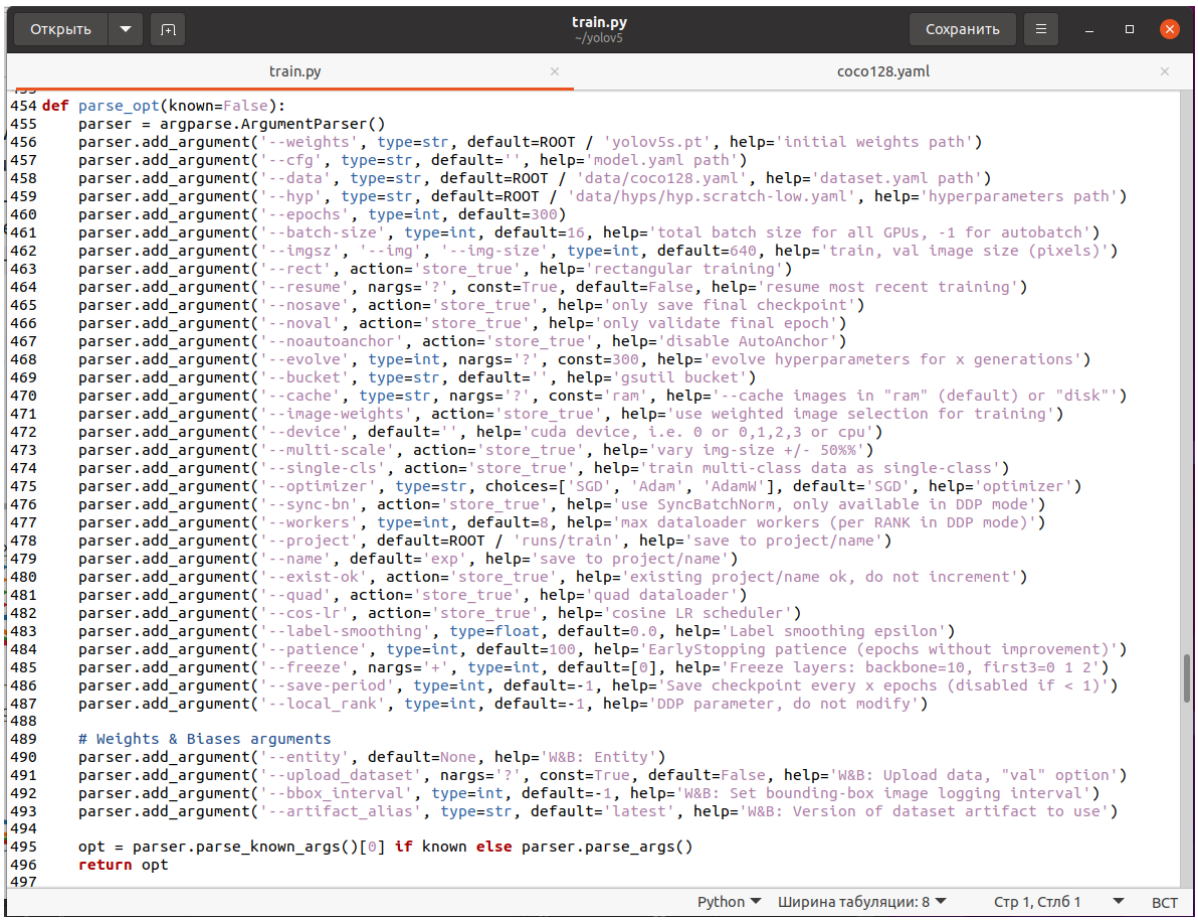
Обратите внимание, что среди параметров по умолчанию в программе установлены:

- файл с предустановленными весами, которые соответствует выбранной архитектуре сети —weights;
- выбранный для обучения датасет и путь к нему —data;
- число эпох обучения —epochs;
- размер изображений для обучения и тестирования —img;
- размер минипакета —batch-size;
- возможность использования графического ускорителя (GPU) — device.

Объясните выбранные основные параметры программы train.py.

7. Для уменьшения требований к памяти системы следует уменьшить размер мини-пакета, который по умолчанию равен 16. Следует запустить программу train.py с уменьшенным значением этого параметра:

python3 train.py --batch-size 2




```python
def parse_opt(known=False):
    parser = argparse.ArgumentParser()
    parser.add_argument('--weights', type=str, default=ROOT / 'yolov5s.pt', help='initial weights path')
    parser.add_argument('--cfg', type=str, default='', help='model.yaml path')
    parser.add_argument('--data', type=str, default=ROOT / 'data/coco128.yaml', help='dataset.yaml path')
    parser.add_argument('--hyp', type=str, default=ROOT / 'data/hyps/hyp.scratch-low.yaml', help='hyperparameters path')
    parser.add_argument('--epochs', type=int, default=300)
    parser.add_argument('--batch-size', type=int, default=16, help='total batch size for all GPUs, -1 for autobatch')
    parser.add_argument('--imgsz', '--img', '--img-size', type=int, default=640, help='train, val image size (pixels)')
    parser.add_argument('--rect', action='store_true', help='rectangular training')
    parser.add_argument('--resume', nargs='?', const=True, default=False, help='resume most recent training')
    parser.add_argument('--nosave', action='store_true', help='only save final checkpoint')
    parser.add_argument('--noval', action='store_true', help='only validate final epoch')
    parser.add_argument('--noautoanchor', action='store_true', help='disable AutoAnchor')
    parser.add_argument('--evolve', type=int, nargs='?', const=300, help='evolve hyperparameters for x generations')
    parser.add_argument('--bucket', type=str, default='', help='gsutil bucket')
    parser.add_argument('--cache', type=str, nargs='?', const='ram', help='--cache images in "ram" (default) or "disk"')
    parser.add_argument('--image-weights', action='store_true', help='use weighted image selection for training')
    parser.add_argument('--device', default='', help='cuda device, i.e. 0 or 0,1,2,3 or cpu')
    parser.add_argument('--multi-scale', action='store_true', help='vary img-size +/- 50%%')
    parser.add_argument('--single-cls', action='store_true', help='train multi-class data as single-class')
    parser.add_argument('--optimizer', type=str, choices=['SGD', 'Adam', 'AdamW'], default='SGD', help='optimizer')
    parser.add_argument('--sync-bn', action='store_true', help='use SyncBatchNorm, only available in DDP mode')
    parser.add_argument('--workers', type=int, default=8, help='max dataloader workers (per RANK in DDP mode)')
    parser.add_argument('--project', default=ROOT / 'runs/train', help='save to project/name')
    parser.add_argument('--name', default='exp', help='save to project/name')
    parser.add_argument('--exist-ok', action='store_true', help='existing project/name ok, do not increment')
    parser.add_argument('--quad', action='store_true', help='quad dataloader')
    parser.add_argument('--cos-lr', action='store_true', help='cosine LR scheduler')
    parser.add_argument('--label-smoothing', type=float, default=0.0, help='Label smoothing epsilon')
    parser.add_argument('--patience', type=int, default=100, help='EarlyStopping patience (epochs without improvement)')
    parser.add_argument('--freeze', nargs='+', type=int, default=[0], help='Freeze layers: backbone=10, first3=0 1 2')
    parser.add_argument('--save-period', type=int, default=-1, help='Save checkpoint every x epochs (disabled if < 1)')
    parser.add_argument('--local_rank', type=int, default=-1, help='DDP parameter, do not modify')

    # Weights & Biases arguments
    parser.add_argument('--entity', default=None, help='W&B: Entity')
    parser.add_argument('--upload_dataset', nargs='?', const=True, default=False, help='W&B: Upload data, "val" option')
    parser.add_argument('--bbox_interval', type=int, default=-1, help='W&B: Set bounding-box image logging interval')
    parser.add_argument('--artifact_alias', type=str, default='latest', help='W&B: Version of dataset artifact to use')

    opt = parser.parse_known_args()[0] if known else parser.parse_args()
    return opt
```

Рисунок 47 - Фрагмент программы train.py с параметрами обучения

После этого программа корректно запустится, загрузит все необходимые данные и приступит к процессу тренировки сети. Этот процесс иллюстрируется проведением эпох обучения, которых в реальной ситуации может быть несколько сотен, а в данном случае по умолчанию их количество равно 300.

Замерьте время выполнения одной эпохи и оцените полное время выполнения обучения нейронной сети. Обратите внимание, что каждая эпоха включает в себя два процесса. Первый из них — собственно процесс обучения с использованием всех 128 изображений датасета. Второй процесс — тестирование пока не полностью обученной сети по тестовому набору.

Рассчитайте полное время обучения сети для всех 300 эпох обучения.

Поскольку это время слишком большое для данной работы, прервите процесс обучения нажав на клавиатуре Ctrl+c. Программа остановится.

8. Запустите процесс обучения с уменьшенным количеством эпох обучения. Для этого запустите программу с параметрами:

python3 train.py --batch-size 2 —epochs 30

Дождитесь конца выполнения программы. По окончании ее работы будут выданы краткие сведения о результатах процесса обучения (рис. 48, 49).

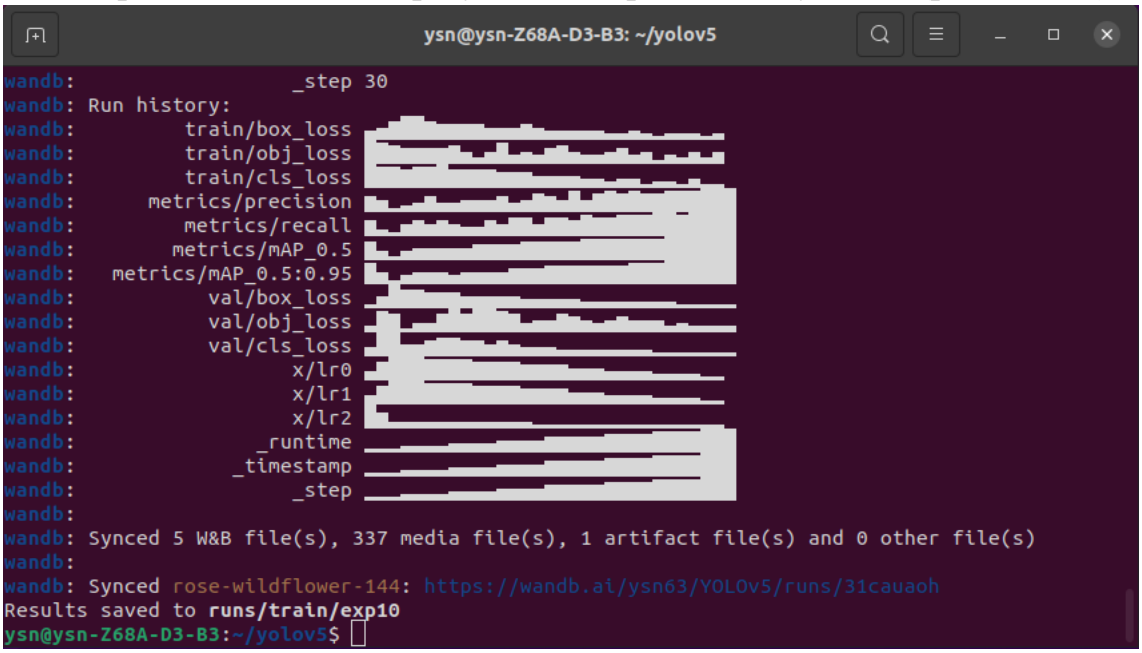



Рисунок 48 - Процесс выполнения обучения с помощью программы train.py

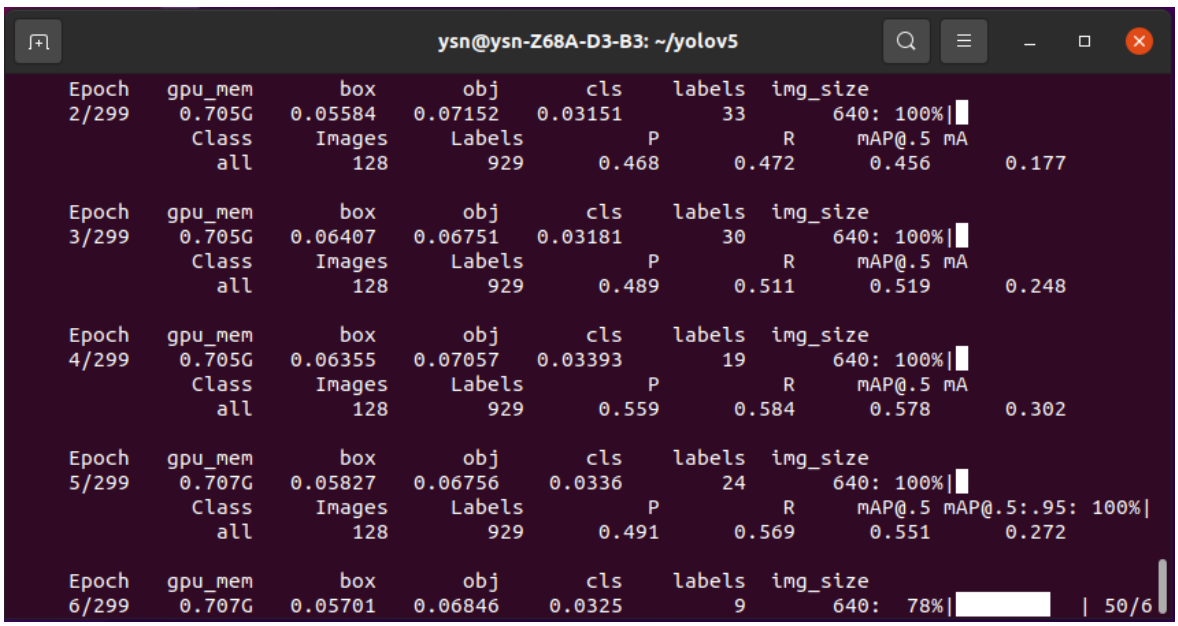



Рисунок 49 - Окончание работы программы train.py

Обратите внимание, что в самом конце (рис. 48) есть сведения о расположении более подробных результатов работы программы (информация о папке exp10).

С помощью программы просмотра файлов найдите указанную папку с результатами. Откройте эту папку (рис. 50). Вы увидите целый ряд графических изображений, иллюстрирующих как сам процесс проведенного обучения, так и его результаты. Кроме того, вы увидите папку weights, в которой два файла с весами только что обученной сети. При этом файл last.pt содержит веса, полученные по окончании всех эпох обучения (в данном случае 30). В файле best.pl содержатся веса обученной нейронной сети, которые соответствуют эпохе с наименьшей ошибкой распознавания, если в процессе работы отмечено переобучение.

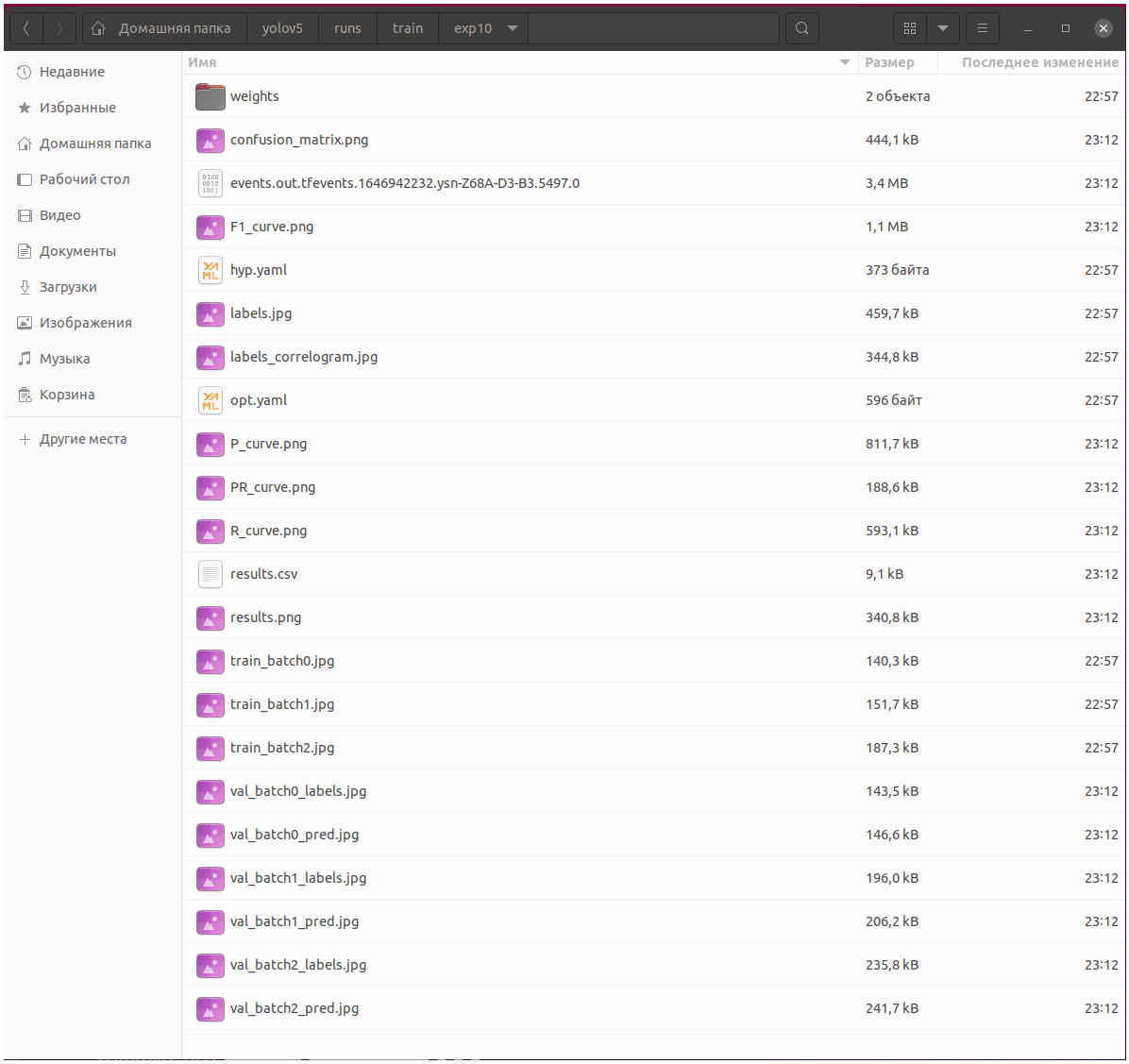



Рисунок 50 - Результаты обучения нейронной сети, которые присутствуют в папке exp

9. Проверьте работу только что обученной нейронной сети. Для этого скопируйте файл с весами last.pl в папку yolov5. Запустите команду работы нейронной сети с выбранными весами:

python3 detect.py --weights last.pt

В этом случае программа detect.py выполнит процесс выделения и распознавания объектов на изображениях, входящих в папку yolov5/data/images. Результат распознавания в виде изображений будет лежать в папке, которая отмечена в конце выполнения программы, например: Results saved to runs/detect/exp4 (рис. 51)

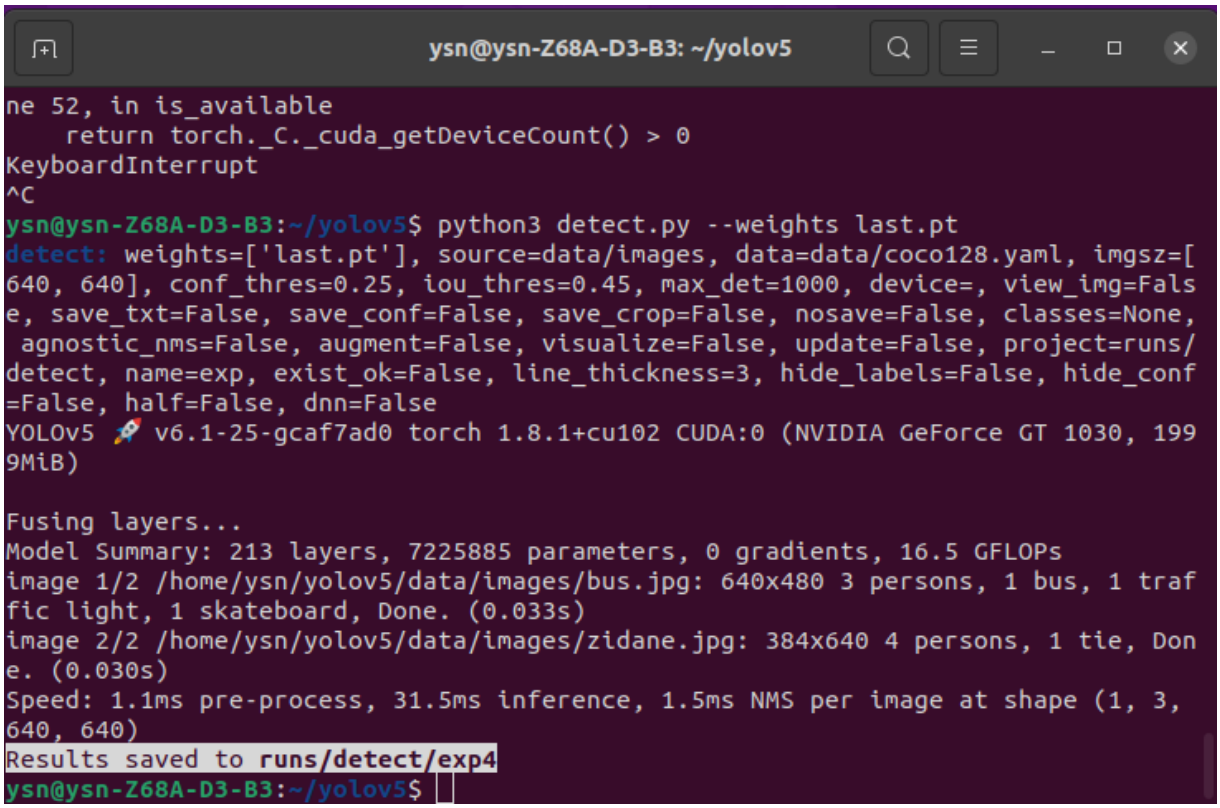



Рисунок 51 - Результат работы программы detect.py

Для сравнения с работой программы, использующей предобученную архитектуру с уже готовыми весами, выполните еще раз эту программу с параметрами по умолчанию:

python3 detect.py

Результаты сохранятся в следующей по номеру папке exp.

Откройте папки с результатами и сравните попарно изображения и результатами распознавания (рис. 52).

Рисунок 52 - Визуальное сравнение результатов распознавания

Опишите замеченную разницу в результатах распознавания. Постарайтесь ее объяснить.

11. Запустите только что обученную нейронную сеть с использованием видеосигнала с камеры. Для этого наберите команду с параметрами:

python3 detect.py --weights last.pt --source 0

Появится изображение с камеры с результатами распознавания. Предъявите насколько объектов, входящих в список классов датасета COCO. Опишите полученные результаты. Сделайте несколько скриншотов.

12. Если полученные результаты обучения неудовлетворительны, то среди прочих причин выделяется недостаточное количество эпох обучения. При этом, если остальные параметры обучения не меняются, то можно запустить процесс дообучения той же самой модели сети.

Для этого достаточно запустить программу обучения, выбрав в качестве параметра модели файл last.pt, и продолжить обучение, указав конечное количество эпох:

python3 train.py --weights last.pt --batch-size 2 --epochs 100

Запустите процесс продолжения обучения и дождитесь его окончания. Найдите результаты обучения в соответствующей папке exp.

13. Повторите пункты 9–11 применительно к дообученной сети. Сравните результаты и опишите, что изменилось в результате дообучения.

14. Существенную роль в качестве обучения имеет выбор архитектуры нейронной сети. Для семейства YOLO предусмотрено несколько отличающихся по сложности архитектур, приведенных в Таблице 3 теоретического описания к работе №2. Эти модели отличаются по количеству параметров (весов, которые следует обучать), максимальному разрешению изображений обучающего набора данных, что, естественно будет влиять как на скорость обучения, так и на скорость работы уже обученной нейронной сети для распознавания объектов.

Запустите процесс обучения для каждой из перечисленных архитектур сети только для одной эпохи. Оцените время выполнения обучения для одной эпохи и занесите в таблицу. Для этого следует выполнить команду, например для архитектуры yolov5l6:

python3 train.py --weights yolov5l6.pt --batch-size 1 --epochs 1 --device cpu

Затем запустите процесс работы обученной нейронной сети с теми же архитектурами. Для этого воспользуйтесь командой с параметрами, например для архитектуры yolov5l6:

python3 detect.py --weights yolov5l6.pt --source 0

Предъявляйте сети одни и те же изображения. Занесите время обработки одного кадра для разных архитектур в таблицу 7.

Таблица 7 – Скорость работы обученных сетей различных архитектур

| Model | size (pixels) | Время тренировки одной эпохи | Скорость работы уже обученной модели | params (M) |
|---|---|---|---|---|
| YOLOv5n | 640 | | | **1.9** |
| YOLOv5s | 640 | | | 7.2 |
| YOLOv5m | 640 | | | 21.2 |
| YOLOv5l | 640 | | | 46.5 |
| YOLOv5x | 640 | | | 86.7 |
| | | | | |
| YOLOv5n6 | 1280 | | | 3.2 |
| YOLOv5s6 | 1280 | | | 12.6 |
| YOLOv5m6 | 1280 | | | 35.7 |
| YOLOv5l6 | 1280 | | | 76.8 |
| YOLOv5x6 | 1280 | | | 140.7 |

15. На качество и скорость процесса обучения будет влиять размер изображения датасета. В данной программе есть возможность изменять размер входного изображения перед предъявлением их нейронной сети. Для этого предусмотрен соответствующий параметр --img.

По умолчанию размер изображения в программе train.py установлен 640. Это примерно соответствует работе нейронной сети с входными изображениями с телевизионной камеры стандартного разрешения.

Для смены размера воспользуйтесь командой для запуска программы с параметрами:

python3 train.py --weights yolov5l6.pt --batch-size 1 --epochs 1 --device cpu --img 320

В этой команде задана модель сети yolov5l6.pt и размер изображений 320. Определите время выполнения одной эпохи обучения и занесите его в таблицу 8.

Таблица 8 – Скорость работы сетей для различных размеров изображения

| Model | size (pixels) | Время тренировки одной эпохи для размера 320 | Время тренировки одной эпохи для размера 640 | Время тренировки одной эпохи для размера 1280 |
|---|---|---|---|---|
| YOLOv5n6 | 1280 | | | |
| YOLOv5s6 | 1280 | | | |
| YOLOv5m6 | 1280 | | | |
| YOLOv5l6 | 1280 | | | |
| YOLOv5x6 | 1280 | | | |

То же самое сделайте для других имеющихся архитектур сети при различных размерах изображений и занесите результаты в таблицу 8.

В некоторых случаях (при выборе сложных моделей и высокого разрешения) процесс обучения даже в рамках одной эпохи может оказаться слишком длительным. В этом случае можно воспользоваться оценкой времени обучения для одной эпохи.

Процесс обучения иллюстрируется последней строкой программы и может выглядеть примерно так:

```
  Epoch   gpu_mem       box       obj       cls    labels  img_size
    0/0        0G   0.01734   0.05642   0.01147        14      1280:  10%|▉         |
13/128 [06:58<55:55, 29.18s/it]
```

Среди прочих параметров здесь вы видите номер эпохи, процент выполнения одной эпохи, и далее в квадратных скобках время уже проведенного процесса обучения и оценка оставшегося времени обучения.

В данном примере за 6 мин. 58 сек выполнено 10% обучения, оставшееся время оценивается как 55 мин 55 сек. Общее время обучения является суммой двух этих отрезков времени.

На основе анализа полученной таблицы объясните, чем вызвано различное время выполнения обучения для одной эпохи.

16. Сохраните свои данные для составления отчета.

**Содержание отчета**

1. Краткие теоретически сведения по работе сверточных нейронных сетей yolo.
2. Скриншоты и основные пояснения основных результатов работы программы обучения нейронной сети с различными параметрами процесса обучения.
3. Таблицы с результатами измерений и краткие выводы по работе.

**Контрольные вопросы**

1. С какой целью проводится анализ процесса обучения сетей и какие действия могут быть предприняты по результатам такого анализа?
2. Что такое аугментация? Когда она может быть применена и какие она имеет ограничения?
3. Перечислите параметры запуска программы train.py. Объясните, на что влияют выбранные значения основных параметров.
4. В каком случае проводится процесс дообучения модели сети? В чем он состоит?
5. Как влияет архитектура нейронных сетей YOLO на качество их обучения? По каким параметрам можно оценить эффективность обучения сетей разной архитектуры?
6. Как влияет размер датасета на качество обучения нейронных сетей YOLO? По каким параметрам можно оценить эффективность работы сетей, обученных по разным датасетам?

Лабораторная работа №4. **Подготовка датасета для обучения нейронной сети**

**Цели работы:**

1. Изучение структуры датасета для обучения нейронной сети YOLO.
2. Изучение этапов подготовки датасета для процедуры обучения.
3. Получение практических навыков работы по созданию специализированного датасета для обучения нейронной сети YOLO.
4. Исследование факторов, влияющих на эффективность подготовки датасета.

Перед началом выполнения работы следует ознакомиться с краткими теоретическими сведениями и описанием основных элементов программного обеспечения, которое используется в работе (общее описание представлено перед порядком выполнения лабораторной работы №2).

**Порядок выполнения работы**

1. С помощью преподавателя или лаборанта включите компьютер и загрузите операционную систему Linux Ubuntu. Если вы ранее не работали в ОС Ubuntu, рекомендуется заранее посмотреть информацию об основах работы в этой ОС.

После загрузки ОС следует с помощью программы просмотра файлов зайти в домашнюю папку и найти в ней папку yolov5. Откройте эту папку и ознакомьтесь с ее содержимым.

В данной папке содержится проект, включающий в себя практически все ПО, необходимое для работы с нейронными сетями YOLO, начиная от знакомства с сетью, ее тестирования на уже заранее обученных моделях разной сложности и обучения на собственном датасете, экспорт полученных весов в другие модели, анализ основных результатов обучения сети и многое другое. Этот проект находится на сайте https://github.com/ultralytics/yolov5 и постоянно обновляется.

**Внимание! Для выполнения данной лабораторной работы этот проект уже установлен на локальном компьютере, и повторная его установка не требуется.**

2. С помощью программы просмотра файлов найдите в папке yolov5/data файл coco128.yaml. Откройте файл и посмотрите его содержимое (рис. 45).

Вы увидите основные параметры датасета COCO128, который использовался в примере обучения нейронной сети в программе train.py. В начале файла приводится адрес в сети, по которому находится более подробная информация по этому датасету.

Далее следует описание структуры каталогов датасета, которую нужно соблюдать, чтобы обучающая программа train.py смогла найти все элементы датасета. Далее указываются количество используемых классов объектов и их названия, что также является необходимым условием процесса обучения сети. В конце файла приводится информация об адресе датасета в сети при необходимости его скачивания на локальный компьютер.

Как уже отмечалось ранее, датасет COCO128 не является полноценным. Он служит для демонстрации процесса обучения сети за сравнительно небольшой промежуток времени.

3. Закройте окно с файлом coco128.yaml и с помощью программы просмотра файлов зайдите в папку, в которой находится датасет. Вы увидите две папки images и labels. Откройте первое изображение в папке images/train2017 и первый файл разметки в папке labels/train2017 (рис. 53).

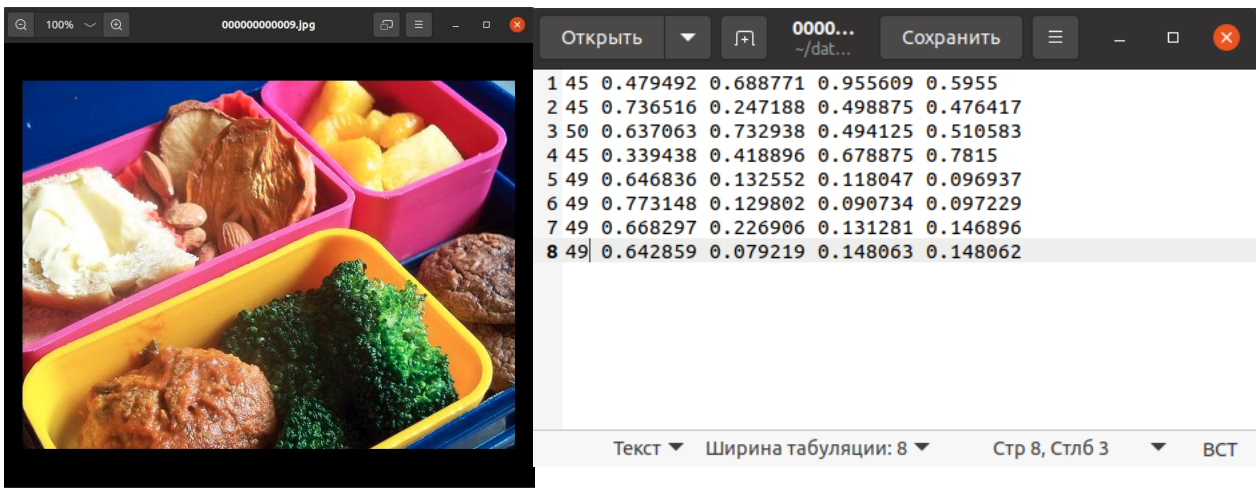



Рисунок 53 - Содержимое файла изображения и соответствующего ему файла разметки

Объясните, какая информация содержится в файле разметки. Сохраните скриншоты для отчета.

Откройте любое другое изображение из датасета, а также соответствующий ему файл разметки. Объясните, что изменилось в файле разметки. Сохраните скриншоты для отчета.

4. С помощью программы просмотра файлов найдите файл описания датасета COCO coco.yaml (файл находится в той же папке, что и файл описания coco128.yaml. Откройте и посмотрите этот файл.
Объясните, какие элементы описания датасета COCO вы видите? Какие отличия вы заметили по сравнению с датасетом COCO128? Какие основные параметры этого датасета?

5. С помощью помощью программы просмотра файлов зайдите в папку, в которой находится датасет COCO. Зайдите в эту папку и посмотрите, какие элементы содержит этот датасет. Какие отличия вы видите по сравнению со структурой датасета COCO128? Отметьте изменения в отчете.

6. Не запуская процесс обучения сети с помощью программы train.py, оцените время обучения сети. Для этого можно воспользоваться данными из предыдущей лабораторной работы, которые вы свели в таблицу 8.

При оценке времени следует учесть, что время обучения сети пропорционально сумме объектов различных классов на всех изображениях датасета. По сравнению с COCO128, в датасете COCO содержится примерно в тысячу раз больше объектов. С учетом этого фактора заполните таблицу 9:

Таблица 9 – Скорость работы сетей для различных размеров изображения, обученных с использованием датасета COCO

| Model | size (pixels) | Время тренировки одной эпохи для размера 320 | Время тренировки одной эпохи для размера 640 | Время тренировки одной эпохи для размера 1280 |
|---|---|---|---|---|
| YOLOv5n6 | 1280 | | | |
| YOLOv5s6 | 1280 | | | |
| YOLOv5m6 | 1280 | | | |
| YOLOv5l6 | 1280 | | | |
| YOLOv5x6 | 1280 | | | |

Объясните, чем вызвано различное время выполнения обучения для одной эпохи. Оцените полное время обучения с учетом того, что для качественного обучения может понадобиться 300 эпох и более.

7. Теперь предположим, что у вас нет подходящего датасета для обучения вашей нейронной сети. В этом случае предстоит довольно долгая работа по ее созданию. В качестве примера будем решать задачу распознавания предметов двух классов, например болтов и гаек для индустриальной системы технического зрения.

Сначала нужно найти подходящие изображения для создания датасета. Например, в поиске Yandex набрать последовательно «Гайки», найти и сохранить несколько изображений (рис. 54).

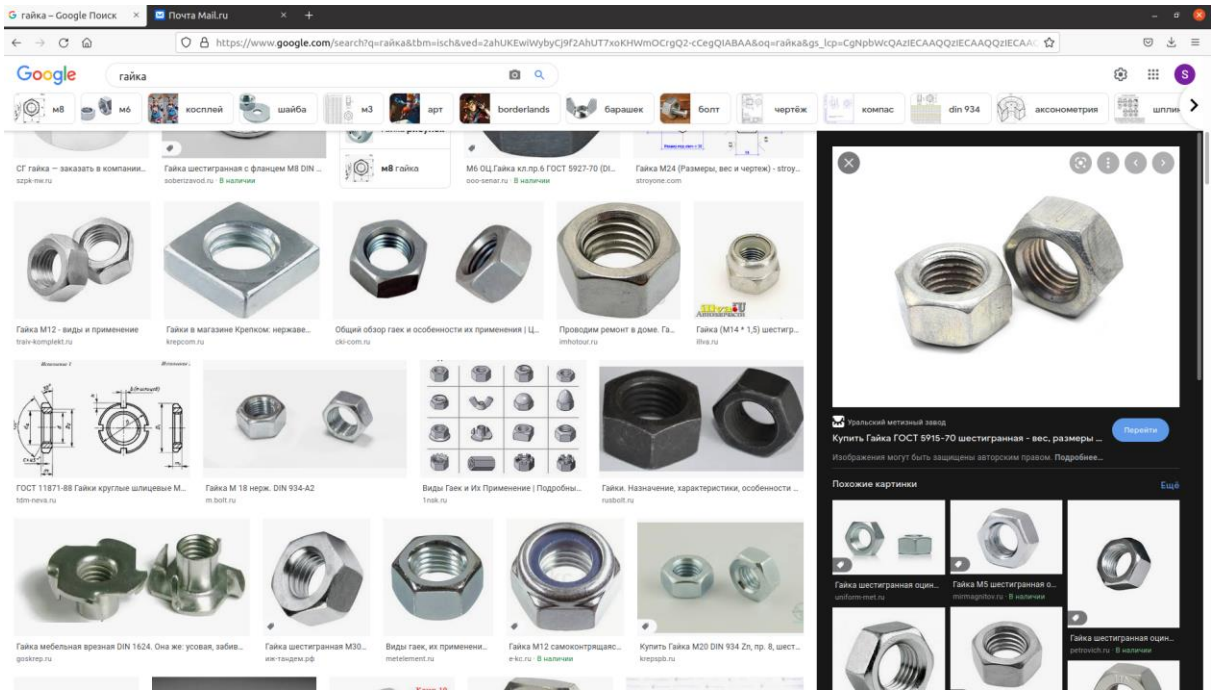



Рисунок 54 - Поиск подходящих изображений для датасета на сайте Yandex

Подберите по 10 изображений объектов двух классов (названия объектов укажет преподаватель или лаборант) и сохраните их в отдельной папке, например в папке datasets/test/images. Внутри папки test следует создать еще одну папку test/labels. В этой папке будут храниться файлы разметки.

8. Теперь следует выполнить разметку датасета. Для этого воспользуемся одной из известных программ разметки labelImg. Для этого в терминале следует ввести команду: labelImg. После загрузки программы вы увидите ее окно, как это показано на рисунке 55.

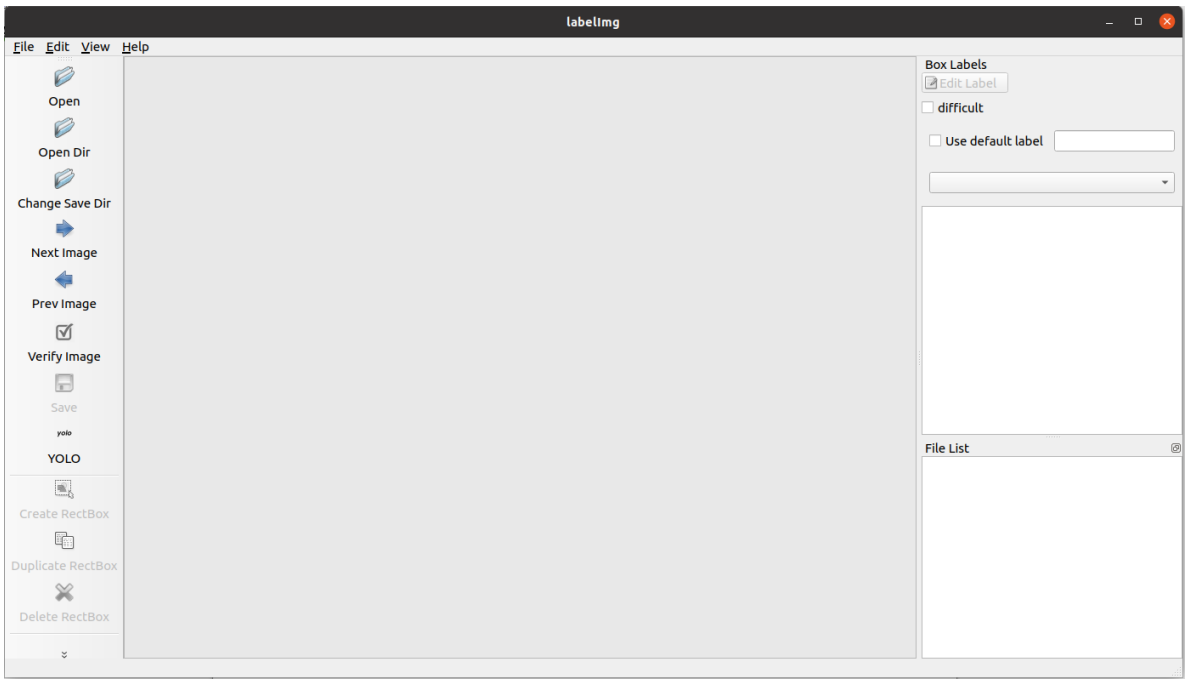



Рисунок 55 - Окно программы labelImg

Большинство команд программы открываются кнопками слева (рис. 56). Сначала нужно установить с помощью команды Open Dir путь к папке с файлами изображений datasets/test/images. После этого вы увидите первое изображение будущего датасета, а в окне File List увидите список всех файлов изображений.

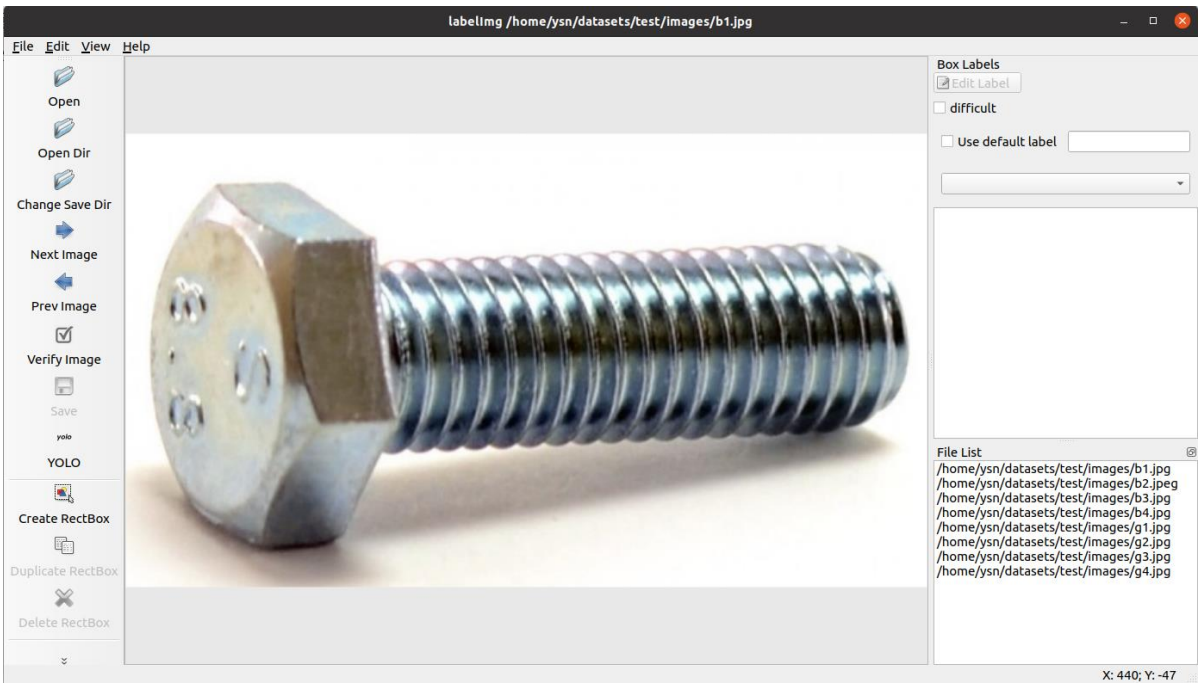



Рисунок 56 - Установка пути к файлам изображений

Теперь нужно установить путь к файлам разметки. Для этого аналогично следует установить путь с помощью команды Change Save Dir путь к папке datasets/test/labels.

9. Теперь можно приступать к разметке. Первый файл изображения вы уже видите на экране. Нажмите кнопку Create RectBox и с помьщью мыши выделите прямоугольную облать, в которую вписывается объект нужного класса. При этом после выделения объекта вы увидите окно, в которое можно внести название нужного класса, например «bolt» или выбрать из уже имеющихся названий (рис. 57).

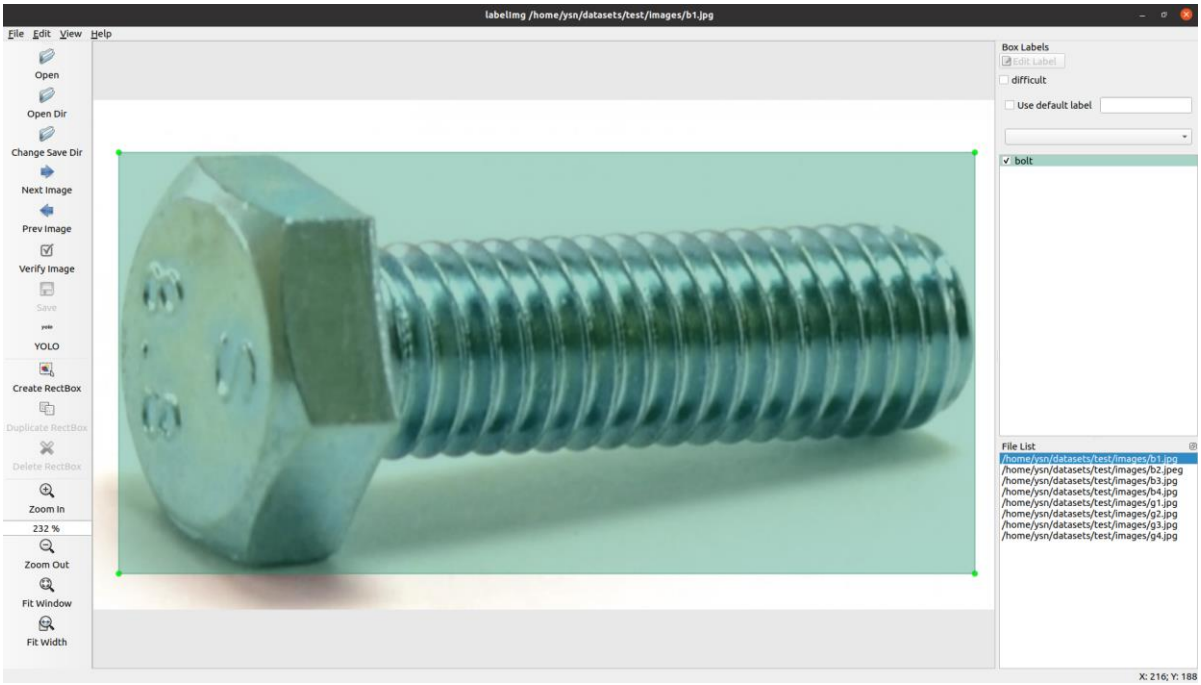



Рисунок 57 - Разметка изображения

Если в окне несколько изображений, то п. 9. следует повторить. После выделения всех изображений следует сохранить файл разметки, нажав на кнопку Save.

Для выбора следующего изображения нужно нажать кнопку Next Image. После этого процесс разметки следует повторить для следующего изображения. Разметку следует выполнить для всех изображений, входящих в папку datasets/test/images.

По окончании разметки зайдите в папку datasets/test/labels и убедитесь, что там присутствуют файлы разметки для всех изображений. Найдите файл classes.txt и откройте его. Объясните, какая информация содержится в этом файле.

10. Для запуска процесса обучения с вашим собственным датасетом понадобится файл описания датасета.

Для ускорения процесса написания этого файла следует найти уже известный вам файл coco128.yaml и скопировать его в ту же папку под называнием bg.yaml. Откройте файл bg.yaml и отредактируйте в нем информацию следующим образом:

a) в строках, указывающих на расположение датасета, сменить пути к папкам вашего датасета, то есть в строках:

```
# Train/val/test sets as 1) dir: path/to/imgs, 2) file: path/to/imgs.txt, or 3) list: [path/to/imgs1, path/to/imgs2, ..]
path: ../datasets/coco128  # dataset root dir
train: images/train2017  # train images (relative to 'path') 128 images
val: images/train2017  # val images (relative to 'path') 128 images
test:  # test images (optional)
```

поменять пути на:

```
# Train/val/test sets as 1) dir: path/to/imgs, 2) file: path/to/imgs.txt, or 3) list: [path/to/imgs1, path/to/imgs2, ..]
path: ../datasets/test  # dataset root dir
train: images  # train images (relative to 'path') 128 images
val: images  # val images (relative to 'path') 128 images
test:  # test images (optional)
```

b) в строках, следующих ниже строк расположения датасета, поменять число классов на 2 и поменять перечень классов:

```
# Classes
nc: 2  # number of classes
names: ['bolt', 'gaika']  # class names
```

c) не забудьте сохранить изменения.

11. Теперь запустите процесс обучения с помощью программы train.py, выбрав в качестве параметров свой датасет bg.yaml и количество эпох 300, для чего введите команду в терминале:

```
python3 train.py --data bg.yaml --epochs 300
```

Процесс обучения будет выполняться для 300 эпох, и после завершения обученная нейронная сеть будет находится в каталоге yolov5/runs/train/exp с последним номером.

В этой папке вы найдете папку yolov5/runs/train/exp *weights*. Скопируйте файл last.pt в папку yolov5.

12. Проверьте работу вашей нейронной сети путем запуска в терминале программы detect.py с параметром --weights last.pt:

python3 detect.py --source 0 --weights last.pt

После загрузки программы вы увидите открытое окно с изображением, полученным с веб-камеры (рис. 58). Наведите камеру на тестовые предметы или их изображения на экране компьютера. Объясните полученный результат.

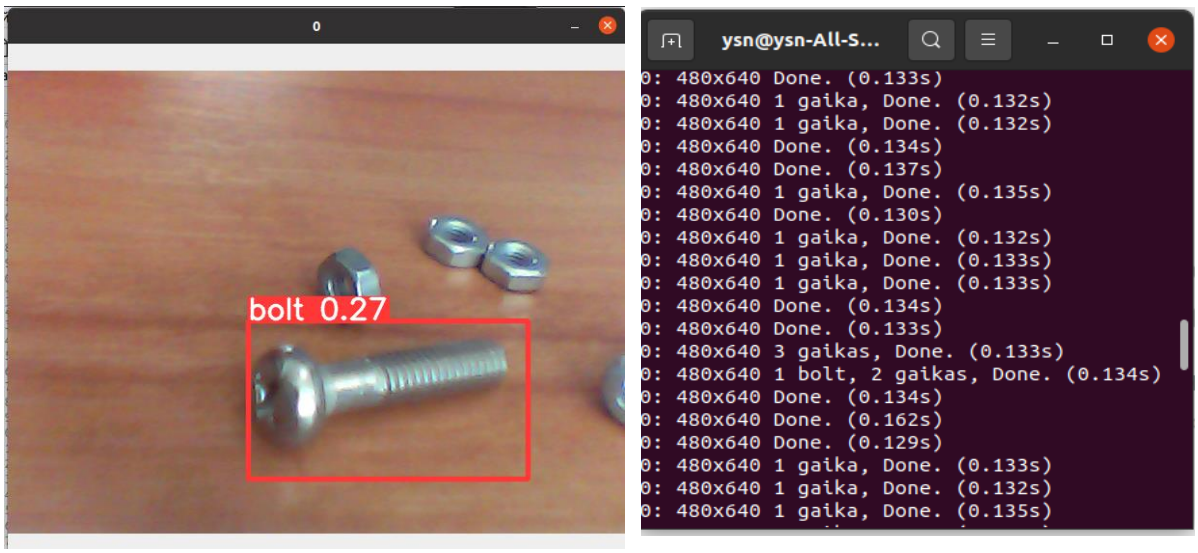



Рисунок 58 - Результат работы программы detect.py

В терминале вы также увидите результат работы программы. Запишите эти результаты. Объясните, почему результат обучения получился именно таким и что нужно сделать, чтобы улучшить результат обучения?

13. Сохраните свои данные для составления отчета.

## Содержание отчета

1. Краткие теоретически сведения по работе сверточных нейронных сетей yolo.
2. Скриншоты и основные пояснения основных результатов работы программы обучения нейронной сети с различными параметрами процесса обучения.
3. Таблицы с результатами измерений и краткие выводы по работе.

## Контрольные вопросы

1. Что такое структура датасета? Каковы особенности структуры датасета для обучения сети YOLO?
2. Перечислите этапы подготовки датасета для проведения обучения сети. Для чего нужен файл разметки? Какую информацию он содержит?
3. От чего зависит время обучения сети?
4. Перечислите факторы, влияющие на эффективность подготовки датасета.

# СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ

1. Ян Лекун. Как учится машина. Революция в области нейронных сетей и глубокого обучения. (Библиотека Сбера: Искусственный интеллект). — М.: Альпина нон-фикшн, 2021. — ISBN 978-5-907394-29-2.
2. Шарден, Б. Крупномасштабное машинное обучение вместе с Python: учебное пособие / Б. Шарден, Л. Массарон, А. Боскетти; перевод с английского А. В. Логунова. — Москва : ДМК Пресс, 2018. — 358 с. — ISBN 978-5-97060-506-6. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/105836 (дата обращения: 28.02.2022). — Режим доступа: для авториз. пользователей.
3. Франсуа Шолле, Эрик Нильсон, Стэн Бэйлесчи, Шэкуинг Цэй JavaScript для глубокого обучения: TensorFlow.js. – СПб.: Питер, 2021. – 576 с.: ил. – Серия «Библиотека программиста»). ISBN 978-5-4461-1697-3
4. VGG16 — сверточная сеть для выделения признаков изображений- URL: https://neurohive.io/ru/vidy-nejrosetej/vgg16-model/, (дата обращения 01.03.2022)
5. Примеры архитектур сверточных сетей VGG-16 и VGG-19 - URL: https://proproprogs.ru/neural_network/primery-arhitektur-svertochnyh-setey-vgg16-i-vgg19, (дата обращения 01.03.2022)
6. Классификация изображений ImageNet - URL: https://gist.github.com/yrevar/942d3a0ac09ec9e5eb3a (дата обращения 20.03.2022)
7. Перечень датасетов, встроенных в Keras - URL: https://keras.io/api/datasets/ (дата обращения 25.03.2022)
8. Перечень предобученных сетей в Keras - URL: https://keras.io/api/applications/ , (дата обращения 25.03.2022)
9. Набор данных MNIST в Python - базовый импорт и построение - URL: https://dev-gang.ru/article/nabor-dannyh-mnist-v-pythonbazovyi-import-i-postroenie-aiypw9sw1l/ , (дата обращения 28.03.2022)
10. Image classification from scratch (Классификация изображений с нуля) - URL: https://keras.io/examples/vision/image_classification_from_scratch/, (дата обращения 10.04.2022)
11. YOLOv5: state-of-the-art модель для распознавания объектов – URL: https://neurohive.io/ru/papers/yolov5-state-of-the-art-model-dlya-raspoznavaniya-obektov/ , (дата обращения 11.06.2022)
12. Астапова М.А., Уздяев М.Ю. Детектирование дефектов неисправных элементов линий электропередач при помощи нейронных сетей семейства

YOLO. Моделирование, оптимизация и информационные технологии. 2021;9(4). Доступно по: https://moitvivt.ru/ru/journal/pdf?id=1115 DOI: 10.26102/2310-6018/2021.35.4.035
13. GitHub for enterprises. Ultralytics. Yolov5 - URL: https://github.com/ultralytics/yolov5 , (дата обращения 12.06.2022)
14. COCO. Common Objects in Context – URL: https://cocodataset.org/#home , (дата обращения 14.06.2022)
15. Pretrained Checkpoints – URL: https://github.com/ultralytics/yolov5#pretrained-checkpoints , (дата обращения 14.06.2022)
16. v6.1 - TensorRT, TensorFlow Edge TPU and OpenVINO Export and Inference - URL: https://github.com/ultralytics/yolov5/releases , (дата обращения 16.06.2022)
17. New Results - URL: https://user-images.githubusercontent.com/26833433/136901921-abcfcd9d-f978-4942-9b97-0e3f202907df.png , (дата обращения 16.06.2022)
18. COCO 2017 Dataset - URL: https://www.kaggle.com/awsaf49/coco-2017-dataset , (дата обращения 16.06.2022)
19. YOLOv5. Python - coco128. Data - URL: https://www.kaggle.com/ultralytics/yolov5/data , (дата обращения 18.06.2022)
20. Anaconda - Среда Разработки Python. Интегрированная среда разработки для научного программирования на языке Python и R – URL: https://xn--90abhbolvbbfgb9aje4m.xn--p1ai/anaconda-sreda-razrabotki-python/ , (дата обращения 18.06.2022)
21. YOLOv5 requirements – URL: https://github.com/ultralytics/yolov5/blob/master/requirements.txt, (дата обращения 18.06.2022)
22. Jonathan Hui. mAP (mean Average Precision) for Object Detection - URL: https://jonathan-hui.medium.com/map-mean-average-precision-for-object-detection-45c121a31173 , (дата обращения 20.06.2022)
23. Метрики в задачах машинного обучения. Блог компании Open Data Science. - URL: https://habr.com/ru/company/ods/blog/328372/, (дата обращения 20.03.2022)

Ярышев Сергей Николаевич
Рыжова Виктория Александровна

# Технологии глубокого обучения и нейронных сетей в задачах видеоанализа

**Учебное пособие**

В авторской редакции
Редакционно-издательский отдел Университета ИТМО
Зав. РИО Н.Ф. Гусарова
Подписано к печати
Заказ №
Тираж
Отпечатано на ризографе

**Редакционно-издательский отдел**
**Университета ИТМО**
197101, Санкт-Петербург, Кронверкский пр., 49, литер А