

МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
"ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ"
(ФГБОУ ВО «ВГУ»)

Факультет компьютерных наук

Кафедра информационных систем

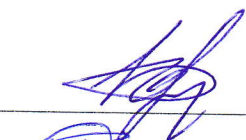
Разработка новых алгоритмов обработки цифровых сигналов в систему на
основе машинного зрения

Отчет по учебной практике, ознакомительной

09.03.02 Информационные системы и технологии

4 семестр 2022/2023 учебного года

Зав. кафедрой



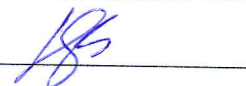
к. т. н., доцент Д.Н. Борисов

Обучающийся



ст. 2 курса оч. отд. Д.С. Котов

Руководитель



ассистент Е. В. Попова

Воронеж 2023

Содержание

Введение	4
1 Постановка задачи.....	5
2 Анализ предметной области.....	6
2.1 Терминология предметной области	6
2.2 Архитектура и принцип работы простейшей нейронной сети	7
2.3 Архитектура и принцип работы свёрточной нейронной сети для обработки изображений.....	9
2.4 Особенности работы нейронной сети YOLO.....	12
3 Средства реализации.....	13
3.1 Используемая аппаратура.....	13
3.1.1 Смартфон OnePlus 9 RT.....	13
3.1.2 Ноутбук Asus VivoBook	13
3.2 Используемые программные средства	14
3.2.1 PyCharm	14
3.2.2 Язык программирования Python.....	14
3.2.3 MakeSense.ai	14
3.2.4 Google Colab	15
4 Реализация задачи	16
4.1 Подготовка тренировочного набора данных	16
4.2 Обучение модели YOLOv5	18
4.3 Разработка интерфейса и написание скрипта	20
4.4 Тестирование приложения.....	22
4.5 Возможности приложения.....	23
4.5.1 Добавление изображения.....	23

4.5.2 Обработка изображения	24
4.5.3 Сохранение изображения	25
Заключение.....	26
Список использованных источников	27

Введение

Машинное зрение является революционной технологией, позволяющей компьютерам "видеть" и интерпретировать изображения и видео таким образом, как это делает человек. Это открывает огромные возможности для автоматизации, оптимизации процессов и создания новых решений в различных областях.

Применение машинного зрения имеет огромную важность в обработке и анализе огромных объемов визуальных данных, с которыми мы сталкиваемся в современном мире. Оно позволяет нам извлекать ценную информацию из изображений и видео, которая ранее оставалась недоступной для автоматизированных систем.

В производстве и промышленности машинное зрение помогает автоматизировать процессы контроля качества, обнаруживать дефекты на производственной линии, оптимизировать логистику и улучшать эффективность операций. Это приводит к повышению производительности, снижению затрат и повышению уровня качества продукции.

1 Постановка задачи

Целью данной работы является разработать (используя средства аппаратно-программного комплекса) и протестировать приложение, основанное на принципах компьютерного зрения, которое должно иметь функцию детекции объектов (детали конструктора LEGO), представленных на изображении, и определения их принадлежности к одному из пяти классов (черный, оранжевый, белый, желтый, серый).

2 Анализ предметной области

Компьютерное или машинное зрение является довольно сложной с технической стороны областью компьютерных наук, в которой применяются алгоритмы машинного обучения и нейронные сети глубокого обучения. И чтобы понимать принцип работы таких компьютерных систем, необходимо погрузиться в теорию.

2.1 Терминология предметной области

Нейрон - узел искусственной нейронной сети, являющийся упрощённой моделью естественного нейрона.

Выход нейронной сети - это итоговое значение функций активации, применённых к взвешенным суммам нейронной сети.

Веса - это действительные числа, отражающие коэффициент связи между конкретными нейронами.

Входные данные - это данные, поступающие к нейрону.

Слой нейронной сети - группа нейронов, находящихся на одном уровне иерархии имеющих общий вход / выход и функцию активации.

Скрытый слой - слой нейронной сети, находящийся между её входным и выходным слоями.

Активационная функция / Функция активации - это функция, определяющая выходной сигнал на основе трансформации входного сигнала: $\text{выход} = \text{функция активации}(\text{вход})$.

Loss / функция ошибки / функция потерь нейронной сети – математическая дифференцируемая функция, характеризующая разницу между «истинным» значением целевой переменной и предсказанным нейронной сетью значением.

Регрессия - задача оценки истинного (числового) значения некоторой независимой переменной (выход нейронной сети) от совокупности зависимых переменных (вход сети).

Свёрточная нейронная сеть - это сеть, работающая по принципу рецептивных полей суть которого заключается в том, что каждый нейрон последующего следующего слоя "смотрит" на небольшой кусочек (например, 3 на 3 пикселя) предыдущего слоя.

Архитектура сети - это совокупность значимых параметров сети, определяющих её назначение и возможности, среди которых: структура и конфигурация слоев нейронной сети а также характер взаимодействия между слоями.

PyTorch — библиотека машинного обучения для языка Python с открытым исходным кодом, созданная на базе Torch. Используется для обработки естественного языка. Разрабатывается преимущественно группой искусственного интеллекта Facebook [1].

2.2 Архитектура и принцип работы простейшей нейронной сети

Перцептрон (англ. Perceptron) — простейший вид нейронных сетей. В основе лежит математическая модель восприятия информации мозгом, состоящая из сенсоров, ассоциативных и реагирующих элементов.

В основе перцептрона лежит математическая модель восприятия информации мозгом. Разные исследователи по-разному его определяют. В самом общем своем виде он представляет систему из элементов трех разных типов: сенсоров, ассоциативных элементов и реагирующих элементов.

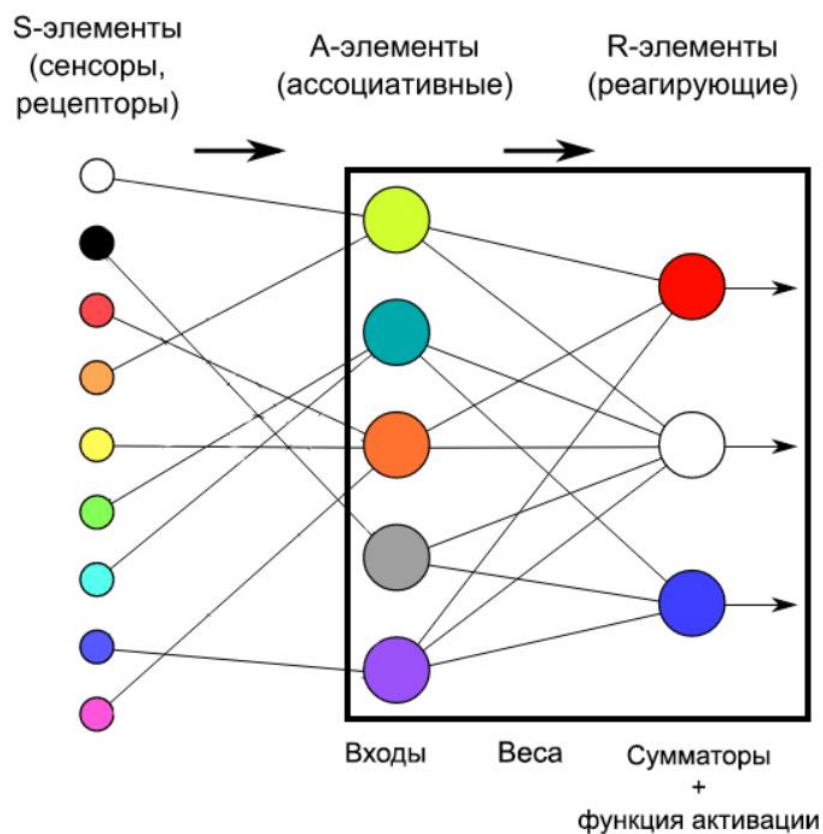


Рисунок 1 - Схема однослойного перцептрона

Принцип работы перцептрона следующий:

Первыми в работу включаются S-элементы. Они могут находиться либо в состоянии покоя (сигнал равен 0), либо в состоянии возбуждения (сигнал равен 1);

Далее сигналы от S-элементов передаются A-элементам по так называемым S-A связям. Эти связи могут иметь веса, равные только -1, 0 или 1;

Затем сигналы от сенсорных элементов, прошедших по S-A связям, попадают в A-элементы, которые еще называют ассоциативными элементами;

— Одному A-элементу может соответствовать несколько S-элементов;

- Если сигналы, поступившие на А-элемент, в совокупности превышают некоторый его порог θ , то этот А-элемент возбуждается и выдает сигнал, равный 1;
- В противном случае (сигнал от S-элементов не превысил порога А-элемента), генерируется нулевой сигнал;

Далее сигналы, которые произвели возбужденные А-элементы, направляются к сумматору (R-элемент), действие которого нам уже известно. Однако, чтобы добраться до R-элемента, они проходят по А-R связям, у которых тоже есть веса (которые уже могут принимать любые значения, в отличие от S-A связей);

R-элемент складывает друг с другом взвешенные сигналы от А-элементов, а затем

- если превышен определенный порог, генерирует выходной сигнал, равный 1;
- если порог не превышен, то выход перцептрона равен -1.

Для элементов перцептрона используют следующие названия:

- S-элементы называют сенсорами;
- А-элементы называют ассоциативными;
- R-элементы называют реагирующими [2].

2.3 Архитектура и принцип работы свёрточной нейронной сети для обработки изображений

Свёрточная нейронная сеть (англ. convolutional neural network, CNN) — специальная архитектура искусственных нейронных сетей, предложенная Яном Лекуном в 1988 году и нацеленная на эффективное распознавание образов, входит в состав технологий глубокого обучения (англ. deep learning).

В обычном перцептроне, который представляет собой полносвязную нейронную сеть, каждый нейрон связан со всеми нейронами предыдущего слоя, причём каждая связь имеет свой персональный весовой коэффициент. В свёрточной нейронной сети в операции свёртки используется лишь ограниченная матрица весов небольшого размера, которую «двигают» по всему обрабатываемому слою (в самом начале — непосредственно по входному изображению), формируя после каждого сдвига сигнал активации для нейрона следующего слоя с аналогичной позицией. То есть для различных нейронов выходного слоя используется одна и та же матрица весов, которую также называют ядром свёртки. Её интерпретируют как графическое кодирование какого-либо признака, например, наличие наклонной линии под определённым углом. Тогда следующий слой, получившийся в результате операции свёртки такой матрицей весов, показывает наличие данного признака в обрабатываемом слое и её координаты, формируя так называемую карту признаков (англ. feature map). Естественно, в свёрточной нейронной сети набор весов не один, а целая гамма, кодирующая элементы изображения (например линии и дуги под разными углами). При этом такие ядра свёртки не закладываются исследователем заранее, а формируются самостоятельно путём обучения сети классическим методом обратного распространения ошибки. Проход каждым набором весов формирует свой собственный экземпляр карты признаков, делая нейронную сеть многоканальной (много независимых карт признаков на одном слое). При переборе слоя матрицей весов её передвигают обычно не на полный шаг (размер этой матрицы), а на небольшое расстояние. Так, например, при размерности матрицы весов 5×5 её сдвигают на один или два нейрона (пикселя) вместо пяти, чтобы не «перешагнуть» искомый признак.

Операция субдискретизации (англ. subsampling, англ. pooling, также переводимая как «операция подвыборки» или операция объединения), выполняет уменьшение размерности сформированных карт признаков. В данной архитектуре сети считается, что информация о факте наличия

искомого признака важнее точного знания его координат, поэтому из нескольких соседних нейронов карты признаков выбирается максимальный и принимается за один нейрон уплотнённой карты признаков меньшей размерности. За счёт такой операции, помимо ускорения дальнейших вычислений, сеть становится более инвариантной к масштабу входного изображения.

Типовая сеть состоит из большого количества слоёв. После начального слоя (входного изображения) сигнал проходит серию свёрточных слоёв, в которых чередуется свёртка и субдискретизация (пулинг). Чередование слоёв позволяет составлять «карты признаков», на каждом следующем слое карта уменьшается в размере, но увеличивается количество каналов. На практике это означает способность распознавания сложных иерархий признаков. Обычно после прохождения нескольких слоёв карта признаков вырождается в вектор или даже скаляр, но таких карт признаков возникают сотни. На выходе свёрточных слоёв сети дополнительно устанавливают несколько слоёв полносвязной нейронной сети (перцептрон), на вход которых подаются окончательные карты признаков [3].

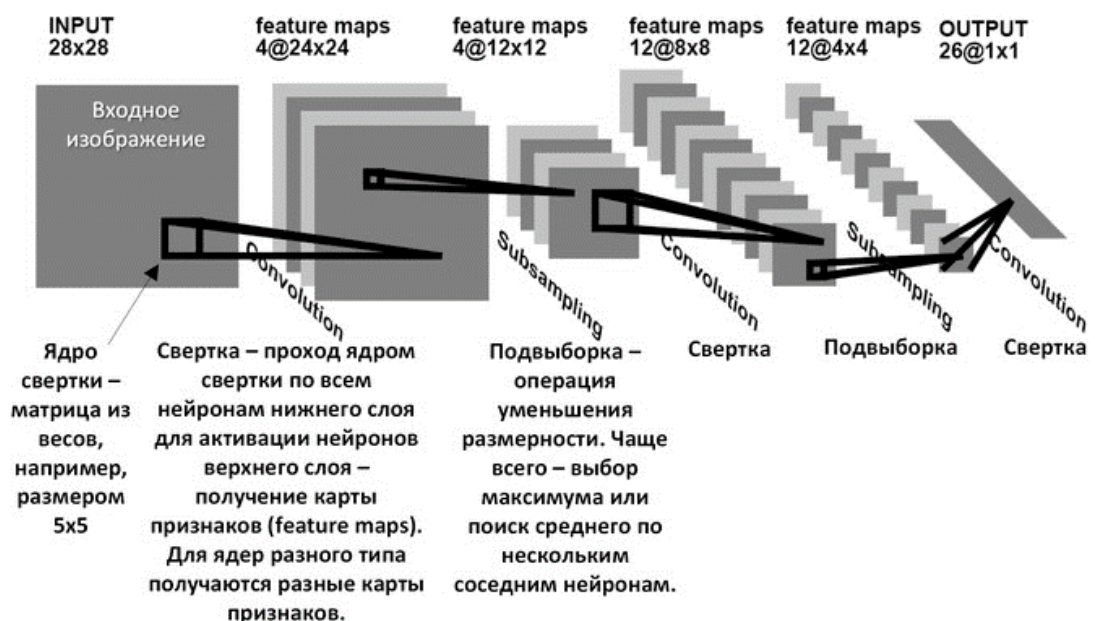


Рисунок 2 - Архитектура свёрточной нейронной сети

2.4 Особенности работы нейронной сети YOLO

YOLO (You Only Look Once) — архитектура нейронных сетей, предназначенная для детекции объектов на изображении. Отличительной особенностью YOLO является подход к решению задачи детекции.

Один из способов решения задачи детекции заключается в разбиении изображения на квадратные области, затем классификация этих областей на наличие объекта и классификация самого объекта. Таким образом, изображение просматривается дважды (один раз для определения областей, где есть объект, второй — для классификации этого объекта.) Этот способ работает долго и требует больших затрат вычислительных мощностей.

YOLO же использует другой принцип. Исходное изображение сжимается таким образом, чтобы получить квадратную матрицу размером 13 на 13, в каждой клетке которой записана информация о наличии объекта и классе этого объекта на соответствующей части картинке. Таким образом, YOLO просматривает картинку один раз, что существенно увеличивает скорость обработки.

YOLOv5 – усовершенствованная пятая версия YOLO, реализованная на фреймворке PyTorch. YOLOv5 встроена в одноименный модуль для Python3, который можно установить с `pip`. Этот модуль предоставляет очень хорошую инфраструктуру как для обучения модели, так и для тестирования с построением графиков всех ключевых показателей [4].

3 Средства реализации

В процессе реализации поставленной цели одну из главных ролей играли средства аппаратно-программного комплекса, благодаря которым удалось разработать приложение.

3.1 Используемая аппаратура

3.1.1 Смартфон OnePlus 9 RT

Технические характеристики камеры устройства:

- Сенсор: Sony IMX766
- Мегапикселей: 50
- Количество линз: 6
- Оптическая стабилизация: есть
- Электронная стабилизация: есть
- Фокусное расстояние: 24мм
- Диафрагма: $f/1,8$

3.1.2 Ноутбук Asus VivoBook

Технические характеристики устройства, отвечающие за производительность:

- Процессор Intel(R) Core(TM) i5-9300H CPU 2.40 GHz
- 4 ядра процессора
- Видео карта NVIDIA GeForce GTX 1650
- 4.0 ГБ видеопамяти
- 16,0 ГБ оперативной памяти

3.2 Используемые программные средства

3.2.1 PyCharm

PyCharm — это кроссплатформенная интегрированная среда разработки для языка программирования Python, разработанная компанией JetBrains на основе IntelliJ IDEA. Предоставляет пользователю комплекс средств для написания кода и визуальный отладчик [5].

3.2.2 Язык программирования Python

Python (в русском языке встречаются названия питон или пайтон) — высокоуровневый язык программирования общего назначения с динамической строгой типизацией и автоматическим управлением памятью, ориентированный на повышение производительности разработчика, читаемости кода и его качества, а также на обеспечение переносимости написанных на нём программ. Язык является полностью объектно-ориентированным в том плане, что всё является объектами [6].

3.2.3 MakeSense.ai

MakeSense.ai — это бесплатный онлайн-инструмент для маркировки фотографий. Благодаря использованию браузера он не требует сложной установки. Он идеально подходит для небольших проектов глубокого обучения компьютерного зрения, делая процесс подготовки набора данных намного проще и быстрее. Готовые этикетки можно загрузить в одном из нескольких поддерживаемых форматов. Приложение было написано на TypeScript и основано на дуэте React/Redux [7].

3.2.4 Google Colab

Google Colab — сервис, созданный Google, который предоставляет возможность работать с кодом на языке Python через Jupyter Notebook, не устанавливая на свой компьютер дополнительных программ. В Google Colab можно применять различные библиотеки на Python, загружать и запускать файлы, анализировать данные и получать результаты в браузере. Этот сервис особенно полезен для разработчиков и студентов, изучающих программирование на Python [8].

4 Реализация задачи

4.1 Подготовка тренировочного набора данных

Подготовка тренировочного набора данных является одним из наиболее важных аспектов успешного обучения нейронной сети. Качество и разнообразие тренировочных данных напрямую влияют на способность нейросети обобщать и делать точные прогнозы на новых данных.

Сначала было сделано около сотни фотографий деталей LEGO на белом листе бумаги под разными углами и с разными комбинациями фигурок.



Рисунок 3 - Пример изображения из набора данных

Затем, с помощью сайта makesense.ai, нужно было на каждой фотографии обвести все детали LEGO в прямоугольную рамку и отнести их к определенному классу.

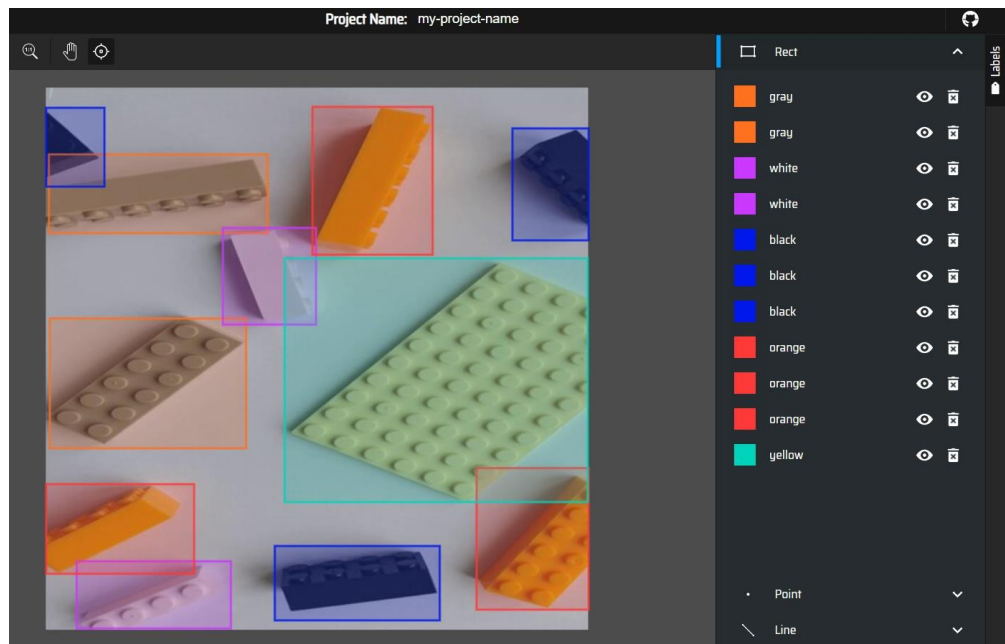


Рисунок 4 - Разметка объектов на сайте makesense.ai

Сайт удобен тем, что он предоставляет возможность сохранить данные о нахождении объектов на изображении в таком виде, в котором это будет удобно модели нейронной сети YOLOv5.

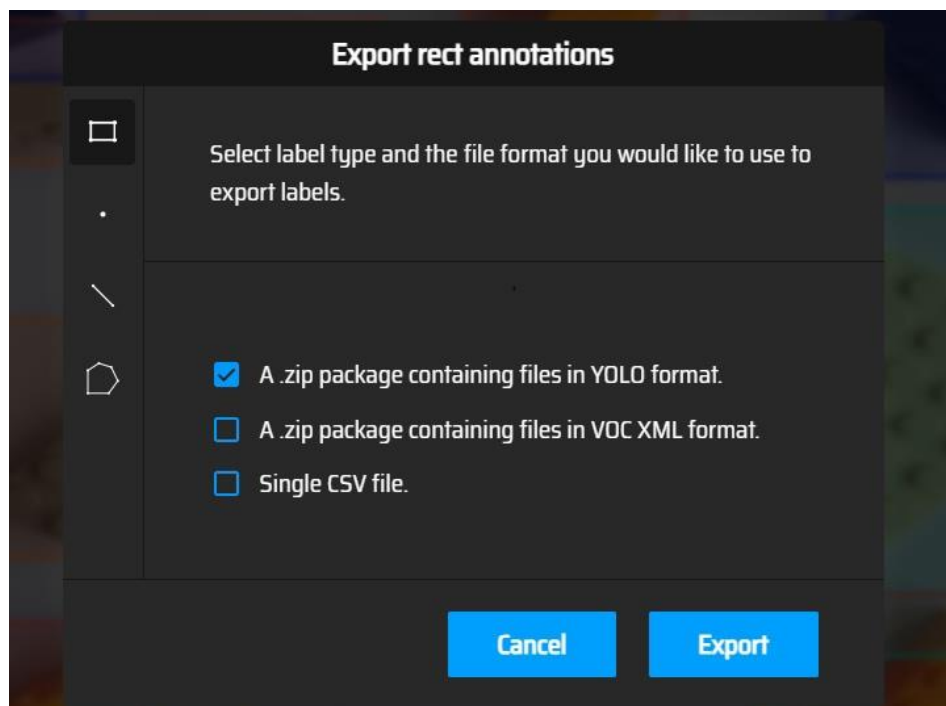


Рисунок 5 - Формат YOLO для модели YOLOv5

Сохранив данные и разбив их на тренировочные и валидационные выборки, можно было приступить к обучению модели нейронной сети.

4.2 Обучение модели YOLOv5

Обучение модели проводилось в Google Colab, так как он предоставляет широкий выбор библиотек машинного обучения и мощный графический процессор, с помощью которого модель могла быстро обучаться.

Сначала необходимо было загрузить тренировочный и валидационный наборы данных и прописать пути к ним. Также необходимо было указать список классов объектов, которые будут на изображениях.

```
1 train: /content/data/images/train # train images (relative to 'path') 128 images
2 val: /content/data/images/val # val images (relative to 'path') 128 images
3
4
5 # Classes
6 names:
7 0: black
8 1: orange
9 2: white
10 3: yellow
11 4: gray
```

Рисунок 6 - Данные конфигурационного файла модели YOLOv5

После этого оставалось запустить небольшой скрипт, указав некоторые параметры, и ждать, пока модель не закончит обучаться.

```
!python train.py --img 640 --batch 16 --epochs 300 --data coco128.yaml --weights yolov5s.pt --cache

train: weights=yolov5s.pt, cfg=, data=coco128.yaml, hyp=data/hyps/hyp.scratch-low.yaml, epochs=300, batch_size=16, imgs
github: up to date with https://github.com/ultralytics/yolov5 ✓
YOLOv5 🚀 v7.0-183-g878d9c8 Python-3.10.12 torch-2.0.1+cu118 CUDA:0 (Tesla T4, 15102MiB)

hyperparameters: lr0=0.01, lrf=0.01, momentum=0.937, weight_decay=0.0005, warmup_epochs=3.0, warmup_momentum=0.8, warmu
TensorBoard: Start with 'tensorboard --logdir runs/train', view at http://localhost:6006/
COMET INFO: Experiment is live on comet.com https://www.comet.com/sslazengerr/yolov5/a136358fbaec44cf814aa5d8fc552c95
```

Рисунок 7 - Начало обучение модели YOLOv5

Процесс обучения занимает много времени, так как он включает в себя несколько процессов таких как:

- Инициализация весов: Начальные значения весов в сети обычно выбираются случайным образом. Веса определяют силу связей между нейронами и будут обновляться в процессе обучения для достижения оптимальных результатов.
- Прямое распространение (forward propagation): Данные передаются через нейронную сеть от входных слоев к выходным. Каждый нейрон выполняет операцию на основе входных данных и своих весов, применяет активационную функцию и передает результат следующему слою.
- Вычисление функции потерь: Сравнивается выход нейронной сети с ожидаемыми значениями и вычисляется функция потерь, которая измеряет разницу между предсказанными и фактическими значениями. Цель состоит в минимизации функции потерь.
- Обратное распространение (backpropagation): Ошибка, определенная функцией потерь, распространяется назад через сеть. Веса нейронов обновляются в соответствии с градиентом функции потерь по отношению к весам. Это делается с использованием оптимизационного алгоритма, такого как стохастический градиентный спуск (SGD) или его модификаций.
- Обучение на нескольких эпохах: Обучение процесса нейронной сети обычно происходит на нескольких эпохах. Одна эпоха означает один проход по всем обучающим данным. В каждой эпохе веса обновляются с целью улучшения производительности сети.

Epoch 0/299	GPU_mem 3.91G	box_loss 0.04618	obj_loss 0.07209	cls_loss 0.01703	Instances 232	Size 640: 100%		8/8 [00:08<00:00, 1.07s/it]
Class		Images	Instances	P	R	mAP50	mAP50-95: 100%	4/4 [00:01<00:00, 2.31it/s]
all		128	929	0.667	0.602	0.68	0.45	
Epoch 1/299	GPU_mem 4.76G	box_loss 0.04617	obj_loss 0.06875	cls_loss 0.01806	Instances 201	Size 640: 100%		8/8 [00:02<00:00, 2.89it/s]
Class		Images	Instances	P	R	mAP50	mAP50-95: 100%	4/4 [00:02<00:00, 1.96it/s]
all		128	929	0.711	0.633	0.715	0.478	
Epoch 2/299	GPU_mem 4.76G	box_loss 0.04358	obj_loss 0.0641	cls_loss 0.01644	Instances 227	Size 640: 100%		8/8 [00:01<00:00, 4.62it/s]
Class		Images	Instances	P	R	mAP50	mAP50-95: 100%	4/4 [00:01<00:00, 2.50it/s]
all		128	929	0.749	0.659	0.75	0.491	
Epoch 3/299	GPU_mem 4.76G	box_loss 0.04416	obj_loss 0.06189	cls_loss 0.01576	Instances 215	Size 640: 100%		8/8 [00:01<00:00, 4.30it/s]
Class		Images	Instances	P	R	mAP50	mAP50-95: 100%	4/4 [00:01<00:00, 2.60it/s]
all		128	929	0.793	0.644	0.768	0.504	
Epoch 4/299	GPU_mem 4.76G	box_loss 0.04424	obj_loss 0.06351	cls_loss 0.01485	Instances 248	Size 640: 100%		8/8 [00:02<00:00, 3.78it/s]
Class		Images	Instances	P	R	mAP50	mAP50-95: 100%	4/4 [00:02<00:00, 1.55it/s]
all		128	929	0.716	0.703	0.781	0.482	

Рисунок 8 - Эпохи и их результаты

Когда модель закончила обучение, нужно было скачать лучшие веса и приступить к следующему этапу.

4.3 Разработка интерфейса и написание скрипта

Интерфейс является одним из важнейших элементов программы, так как с ним напрямую взаимодействует пользователь. Задача была сделать простой и в то же время понятный интерфейс. В этом помогла библиотека PyQt5.

Было решено в верхней части окна расположить виды объектов их отношения к классам, которые модель должна обнаруживать. Посчиталось, что это даст понимание пользователю, каких результатов стоит ожидать.

Чуть ниже расположились две области с кнопками “Load image” и “Save image” под ними. Левая область предназначено для загрузки фотографии в нее, а правая, для сохранения фотографии, которая получится в результате обработки модели YOLOv5.

Сам процесс обработки или детекции можно начать, нажав на кнопку “Start detecting”, расположившуюся в самом низу посередине.

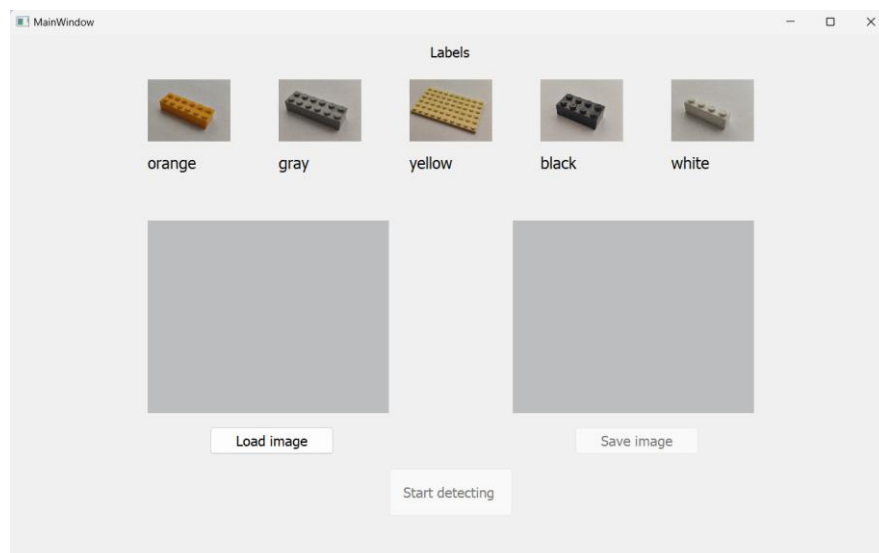


Рисунок 9 - Интерфейс приложения

Задача скрипта была в том, чтобы после нажатия на кнопку, выбранная фотография передавалась в нейронную сеть, и началась ее обработка.

Сложность заключалась в том, что модель нейронной сети запускалась, через консоль. Но благодаря огромному числу библиотек в языке Python, которые предоставляют широкие возможности, удалось этот процесс автоматизировать. Данное решение сильно облегчило использование приложения, и теперь пользователю придется не вручную прописывать код в консоли для запуска детекции, а делать это одним нажатием клавиши.

```
import subprocess

script = 'python yolov5/detect.py --weights yolov5/best.pt --source ' + 'image_path'

subprocess.call(script, shell=True)
```

Рисунок 10 - Автоматизация работы с консолью

4.4 Тестирование приложения

Когда приложение было готово и казалось, что поставленная задача выполнена, пришло время к такому важному этапу, как тестирование.

Задача тестирования заключалась в том, чтобы выявить проблемы и ошибки в ходе использования приложения, зафиксировать точность распознавания объектов и измерить его производительность.

В среднем, точность обнаружения и классификации объекта на тестовой выборке достигла 70%, что довольно неплохой результат с учетом того, что обучающий набор данных составлял около ста картинок.

Скорость обработки одной фотографии занимала около пяти секунд, что является не самым лучшим результатом, но данные показатели могут разниться, в зависимости от вычислительных мощностей машины.

А проблема, хоть и единственная, но нашлась, и пришлось потратить немало времени, чтобы ее выявить и исправить.

После завершения детекции каждая обработанная фотография загружалась в отдельную папку, которая расположена в одной из директорий модели YOLOv5, затем она копировалась и помещалась в область интерфейса. Этот процесс производился путем доступа к директории, обращении к последней папке и получения ее нулевого элемента. Таким образом получался полный путь до картинка, который передавался в объект интерфейса.

Проблема была в том, что, после создании девятой папки с обработанным изображением, логика ее извлечения по каким-то причинам переставала работать. Проявлялось это в том, что после детекции фотографии в область с полученным результатом добавлялась картинка из девятой папки, а не из последней. Причину проблемы выявить не удалось, но зато получилось ее решить.

Пришлось изменить подход к получению последней папки директории. Было замечено, что каждая папка директории имела название “exp”, к которому приписывалось целое число. После очередной детекции картинка

помещалась в новую папку с названием “ехр” но с числом на одно больше, чем у предыдущей. Тогда был создан счетчик, который при инициализации программы был равен количеству элементов в директории, а затем, после каждой детекции, увеличивался на один. И для доступа к последней папке оставалось только добавить значение счетчика к “ехр”. Это решение помогло избавиться от этой проблемы, и программа стала работать без ошибок.

4.5 Возможности приложения

4.5.1 Добавление изображения

В пользовательский интерфейс добавлена возможность добавления изображения, которое необходимо обработать. Это было достигнуто, путем добавления кнопки “Load image”, которая после нажатия открывала проводник и предоставляла пользователю выбрать изображение в форматах: jpg, png, jpeg, bmp.

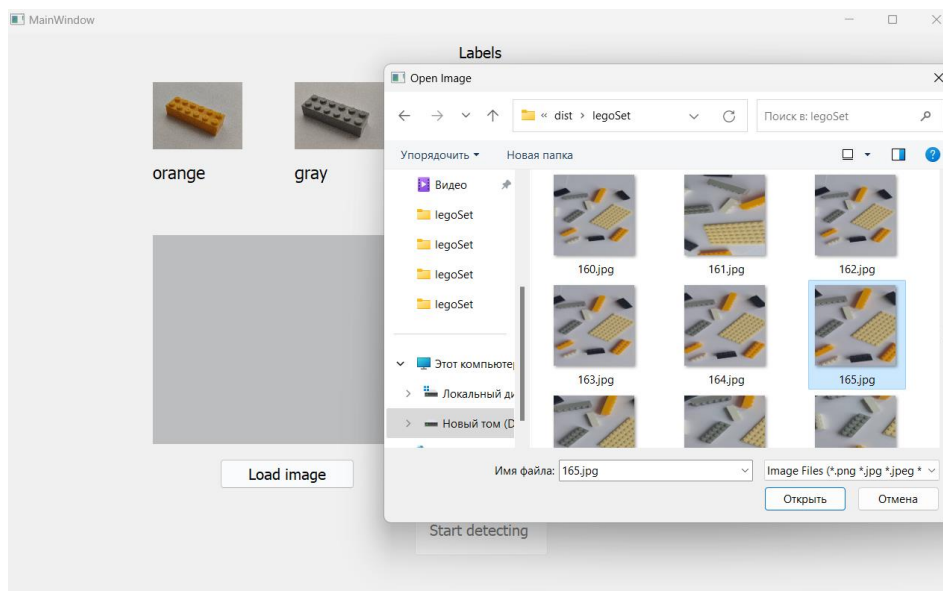


Рисунок 11 - Выбор изображения в проводнике

После выбора изображения оно загружалось в интерфейс, и кнопка, запускающая детекцию, становилась доступна для нажатия.

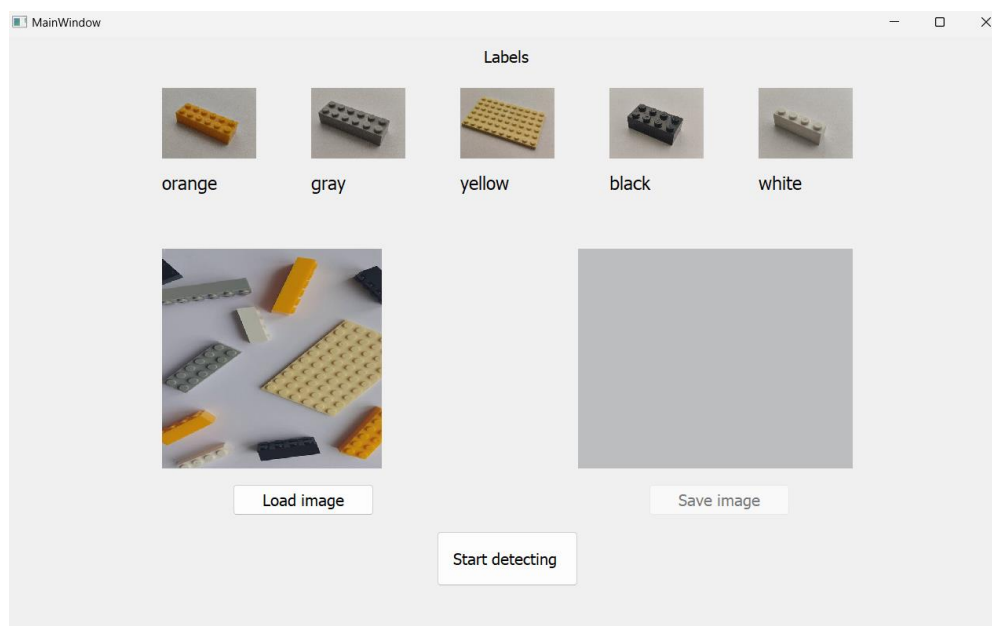


Рисунок 12 - Добавление изображения в интерфейс

4.5.2 Обработка изображения

Чтобы начать процесс обработки фотографии, нужно нажать на кнопку “Start detecting”, после чего запускается скрипт, который передает загруженное изображение в модель. Подождав около пяти секунд во второй области появляется результат или выход модели YOLOv5.

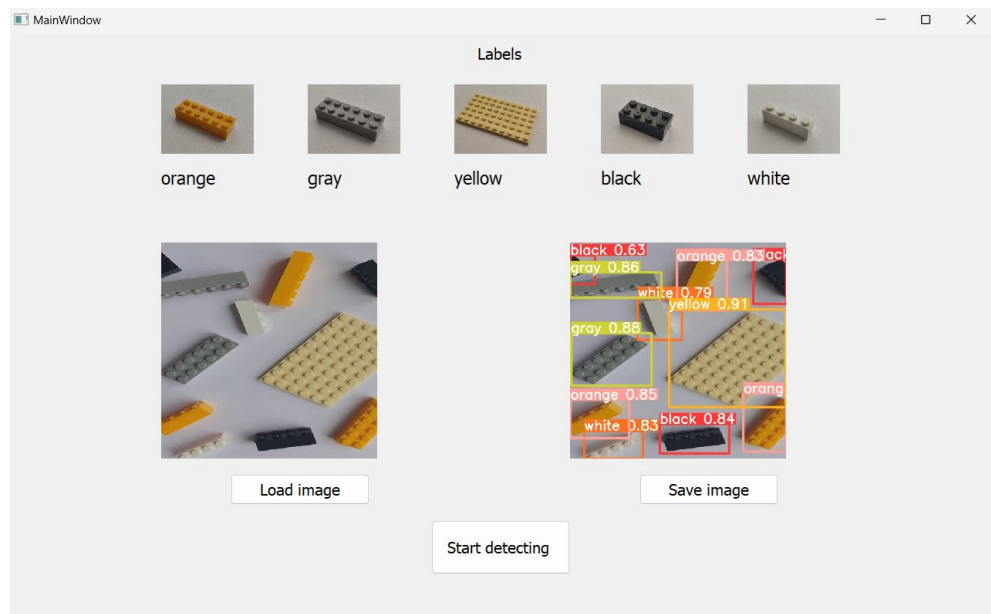


Рисунок 13 - Результат обработки изображения

4.5.3 Сохранение изображения

Получившееся изображение возможно сохранить, нажав на кнопку “Save image”, которая открывает проводник и предоставляет пользователю выбрать директорию.

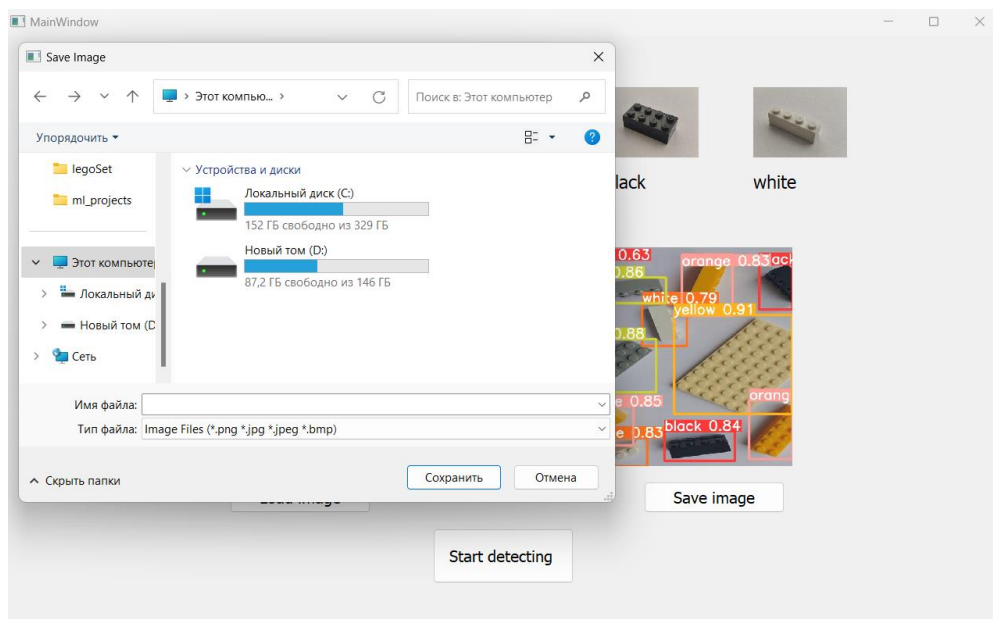


Рисунок 14 - Сохранение изображения

Заключение

В рамках данной работы было разработано и протестировано приложение, основанное на модели YOLOv5, способное детектировать детали LEGO на изображениях и определять их принадлежность к одному из пяти классов: черный, оранжевый, белый, желтый и серый.

Результаты тестирования показали, что разработанное приложение способно обнаруживать объекты и определять их класс с высокой вероятностью. Точность и производительность приложения были оптимизированы с помощью средств аппаратно-программного комплекса, что позволило достичь быстрой и эффективной обработки изображений в реальном времени.

Список использованных источников

1. Словарь терминов по нейронным сетям. <https://neural-university.ru/vocabulary-neural-networks>. Текст: электронный.
2. Вики-конспект Института точной механики и оптики. Нейронные сети, перцептрон. https://neerc.ifmo.ru/wiki/index.php?title=Нейронные_сети,_перцептрон. Текст: электронный.
3. Википедия. Свёрточная нейронная сеть. https://ru.wikipedia.org/wiki/Свёрточная_нейронная_сеть. Текст: электронный
4. Использование YOLOv5 для задач детекции. <https://vc.ru/newtechaudit/326571-ispolzovanie-yolov5-dlya-zadachi-detekcii>.
Текст: электронный.
5. Википедия. PyCharm. <https://ru.wikipedia.org/wiki/PyCharm>.
Текст: электронный.
6. Википедия. Язык программирования Python. <https://ru.wikipedia.org/wiki/Python>. Текст: электронный.
7. Github MakeSense.ai. <https://github.com/SkalskiP/make-sense/>.
Текст: электронный.
8. Что такое Google Colab и кому он нужен. <https://blog.skillfactory.ru/chto-takoe-google-colaboratory-i-komu-on-nuzhen/>.
Текст: электронный.