

## Лабораторная работа №6

### Отчет

ФИО, группа: Дериглазов Егор Дмитриевич Б22-513

Язык программирования: C++, Python

Тема: распознавание объектов в игре

Краткое описание: Игровые ИИ активно набирают популярность. Многие из них даже обыгрывают людей в популярных играх. Во многих из них требуется точное распознавание объектов и разных признаков на экране, для определения дальнейших действий ИИ.

Существует 2 эффективных подхода для распознавания объектов:

1. Нейросети
2. Классические алгоритмы(каскады Хаара, метод Виолы-Джонса)

В ходе работы планируется сравнить эти подходы.

Основными метриками сравнения будут точность и скорость распознавания.

Для примера будут использованы игры mario kart и классический mario.

В mario kart основной задачей будет обнаружение различных предметов, которыми “бросаются” персонажи. В mario - обнаружение врагов.

В качестве результата ожидается получить систему, которая будет в реальном времени обрабатывать игровой поток и подсвечивать препятствия на дороге или врагов примерно следующим образом:

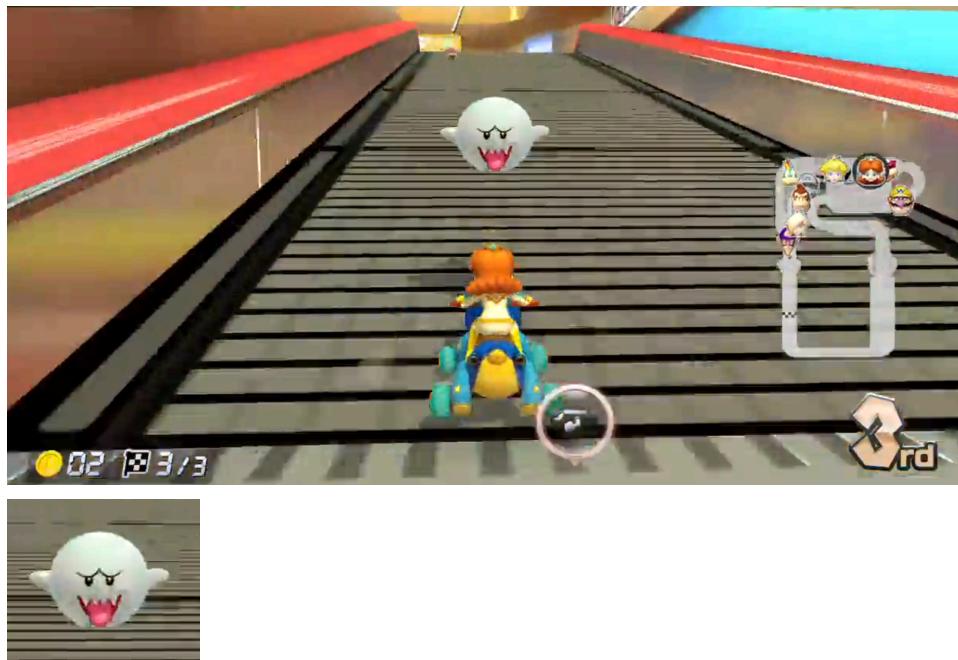


## Полное описание:

В ходе данной работы были обучены классификатор на основе каскадов Хаара, а также дообучена нейросеть YOLOv5. Были написаны прикладные программы для их тестирования на видеопотоке и изображениях.

В качестве упрощённой задачи было выбрано распознавание “приведения Бу” из игры Mario Kart 8.

Рисунок 1 - приведение Бу из игры Mario Kart 8.



## **Данные**

В качестве основного набора данных использовался размеченный датасет [mario-kart-8-deluxe-buu-huu-detection](#).

Поскольку упомянутый набор данных содержал только положительные картинки, т.е. те, в которых приведение есть, а для обучения нужны также негативные, был написан скрипт для автоматического создания скриншотов из видео игры [screen\\_gp.py](#)

Формат разметки данных следующий:

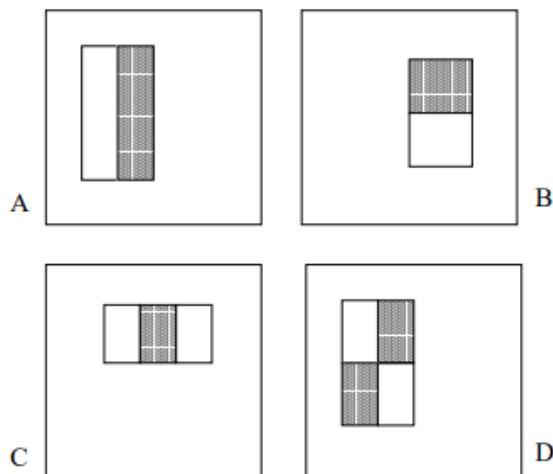
```
0 0.530728813559322 0.681910447761194 0.510593220338983 0.31529850746268656
```

Первое число - номер класса, находящегося на картинке. В данной работе класс только один, поэтому число может быть произвольным. Следующие 2 числа это координаты центра баундинг бокса(прямоугольника, внутри которого находится указанный объект, далее будет упоминаться как BB или bounding box). Последние 2 числа есть высота и ширина картинки соответственно. Важно заметить, что все упомянутые величины нормализованы для обучения нейросети.

## Каскады Хаара

Данный метод основан на фильтрах Хаара. Некоторые из них изображены на рисунке 2.

Рисунок 2 - фильтры Хаара



Данные фильтры основаны на функциях, называемых вейвлетами Хаара. Приведены на рисунке 3.

Рисунок 3 - вейвлеты Хаара

The Haar wavelet's mother wavelet function  $\psi(t)$  can be described as

$$\psi(t) = \begin{cases} 1 & 0 \leq t < \frac{1}{2}, \\ -1 & \frac{1}{2} \leq t < 1, \\ 0 & \text{otherwise.} \end{cases}$$

Its scaling function  $\varphi(t)$  can be described as

$$\varphi(t) = \begin{cases} 1 & 0 \leq t < 1, \\ 0 & \text{otherwise.} \end{cases}$$

По сути, они представляют собой прямоугольники с 2, 3 или 4 тёмными и светлыми областями. Они работают следующим образом, берётся разность между суммами интенсивностей пикселей между светлыми и тёмными областями. В случае 2 прямоугольников, эта разность тривиальна. В случае 3, из среднего вычитаются крайние. В случае 4, разность между парами. Чем больше получившаяся разность по модулю, тем выше отклик фильтра. Фильтры Хаара способны обнаруживать простейшие примитивы на изображениях такие как линии и грани. Существуют вертикальные, горизонтальные и наклонные вариации этих фильтров.

Однако вычисление сумм областей и разность между ними может оказаться вычислительно сложной задачей даже для небольших изображений. Чтобы ускорить процесс обучения, в оригинальной статье Виолы-Джонса предложен метод интегральных изображений. Суть его заключается в предварительном подсчёте для каждой позиции всех пикселей над ней и слева от неё. Демонстрация на рисунке 4.

Рисунок 4 - обычное изображение и его интегральное представление

98	110	121	125	122	129	
99	110	120	116	116	129	
97	109	124	111	123	134	
98	112	132	108	123	133	
97	113	147	108	125	142	
95	111	168	122	130	137	
96	104	172	130	126	130	

Image *I*

98	208	329	454	576	705	
197	417	658	899	1137	1395	
294	623	988	1340	1701	2093	
392	833	1330	1790	2274	2799	
489	1043	1687	2255	2864	3531	
584	1249	2061	2751	3490	4294	
680	1449	2433	3253	4118	5052	

Integral Image *II*

Такое представление позволяет считать сумму внутри любого прямоугольника за счёт 4 операций вычитания и сложения. По сути за константное время  $O(1)$ . Демонстрация на рисунке 5.

Рисунок 5 - подсчёт суммы внутри прямоугольника

98	110	121	125	122	129	
99	110	120	116	116	129	
97	109	124	111	123	134	
98	112	132	108	123	133	
97	113	147	108	125	142	
95	111	168	122	130	137	
96	104	172	130	126	130	

Image *I*

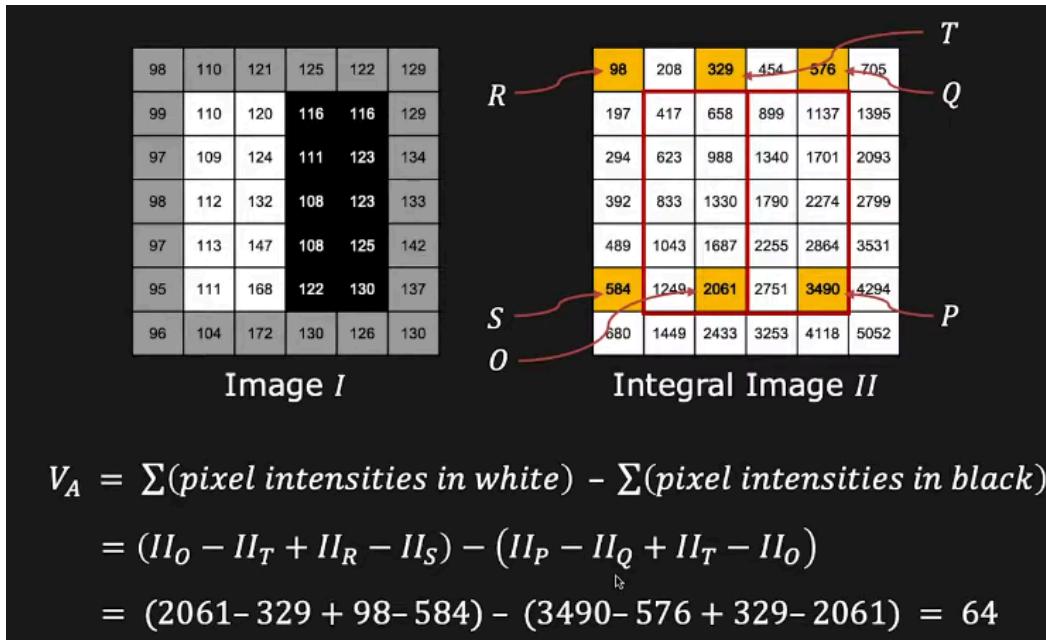
98	208	329	454	576	705	
197	417	658	899	1137	1395	
294	623	988	1340	1701	2093	
392	833	1330	1790	2274	2799	
489	1043	1687	2255	2864	3531	
584	1249	2061	2751	3490	4294	
680	1449	2433	3253	4118	5052	

Integral Image *II*

$$\begin{aligned} \text{Sum} &= II_P - II_Q - II_S + II_R \\ &= 3490 - 1137 - 1249 + 417 = 1521 \end{aligned}$$

Таким образом, чтобы подсчитать разность, например, для фильтра Хаара с 2 прямоугольниками достаточно всего  $4*2 = 8$  операций вычитания и сложения. Демонстрация на рисунке 6.

Рисунок 6 - вычисление отклика фильтра Хаара с помощью интегрального изображения



$$\begin{aligned}
 V_A &= \sum(\text{pixel intensities in white}) - \sum(\text{pixel intensities in black}) \\
 &= (II_O - II_T + II_R - II_S) - (II_P - II_Q + II_T - II_O) \\
 &= (2061 - 329 + 98 - 584) - (3490 - 576 + 329 - 2061) = 64
 \end{aligned}$$

Учитывая то, что сумма для всего изображения может быть посчитана за один проход, то сложность вычисления откликов становится линейной  $O(n)$ . Сейчас почти во всех библиотеках реализующих каскады Хаара используется данная оптимизация.

Для обучения классификатора в оригинальной статье Виолы-Джонса предлагается использовать модификацию Adaboost - алгоритм обучения ансамблем классификаторов. Основные особенности Adaboost по сравнению с классическим RandomForest:

1. Разный вес голосов у weak learner, вычисляемый по формуле приведенной на рисунке 7.

Рисунок 7 - формула вычисления веса голоса weak learner

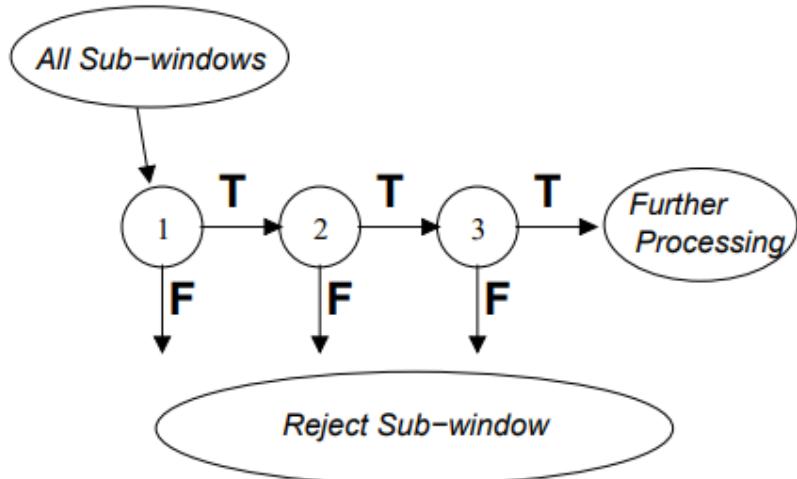
$$\text{Amount of Say} = \frac{1}{2} \log\left(\frac{1 - \text{Total Error}}{\text{Total Error}}\right)$$

2. В качестве weak learner используются “пеньки” или же решающие деревья с корнем и 2 листьями
3. Для каждого окна на изображении(подробнее в следующем абзаце) есть вес и confidence score, которые пересчитываются по ходу обучения.

Процесс обучения напоминает классические алгоритмы свёртки с тем отличием, что вычисление самих фильтров Хаара, вообще говоря не является свёрткой. Алгоритм состоит в применении скользящего окна, заданного размера на разные участки изображения. В каждом окне происходит применение разных фильтров Хаара. На основании этого происходит подсчёт score для данного окна. Если score

больше заданного, то окно переходит на следующую ступень(каскад), если же нет, то больше не рассматривается. Условная графическая схема приведена на рисунке 8.

Рисунок 8 - схема каскадов Хаара



Пример обучения приведён на видео:

[▶ Haar Cascade Visualization](#)

Для обучения классификатора на основе каскадов Хаара мною была использована библиотека OpenCV 3.4, поскольку с версии 4.0 поддержка каскадов Хаара была прекращена.

Так как каскады Хаара работают с окнами из пикселей, был написан скрипт [data\\_prep.py](#), денормализующий разметку и переводящий её в следующий формат:

Оригинальная разметка датасета:

```
0 0.530728813559322 0.681910447761194 0.510593220338983 0.31529850746268656
```

Переведённая:

```
train\images\388.png.rf.cc871b6f210f51227d9b9a3b7f68a82b.jpg 1 199 292 271 206
```

Новая разметка записывалась в файл **Good.dat**. Все негативные примеры без разметки были записаны скриптом в файл **Bad.dat**.

Далее было произведено обучение классификатора с помощью следующих команд:

```
opencv_createsamples -info Good.dat -vec good.vec -w 100 -h 100
```

```
opencv_traincascade -data boo_detect/ -vec good.vec -bg Bad.dat -numPos 50 -numNeg 108 -numStages 10 -w 100 -h 100 -mode ALL
```

Первая команда создаёт векторное представление изображений и переводит их в интегральные изображения. Вторая команда производит само обучение классификатора.

Основные параметры:

- data - там где будет лежать обыченная модель и чекпоинты
- vec - векторный файл с распознанными классами
- вп - файл с негативными изображениями без классов
- numPos - количество позитивных изображений на каждой эпохе. Рекомендуется выбирать в половину меньше, чем негативных.
- numNeg - количество негативных изображений
- numStages - количество эпох. Необязательно соблюдается. Если достигнут нужный HitRate(Precision) или FalsePositiveRate, то обучение прекращается. На практике на данном датасете обучение останавливалось на 5 стадии.
- w, h - размер скользящего окна
- mode - ALL заставляет OpenCV использовать все, в том числе и наклонные, фильтры Хаара.

Распознавание происходит с помощью функции OpenCV detectMultiScale, которая распознаёт изображение на разных масштабах. Лучше всего результаты вышли при следующих параметрах:

```
python .\main.py -test -nimg 56 -write -res results -w 100 -he 100 -mnb 50
```

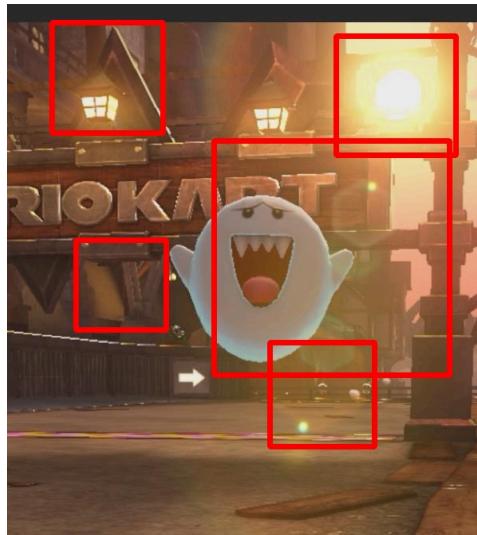
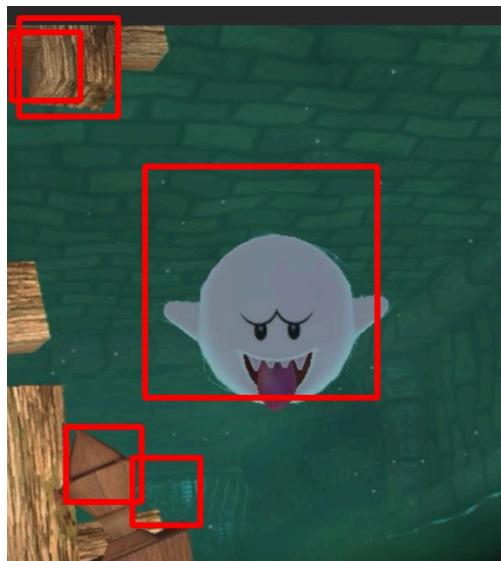
Основные параметры:

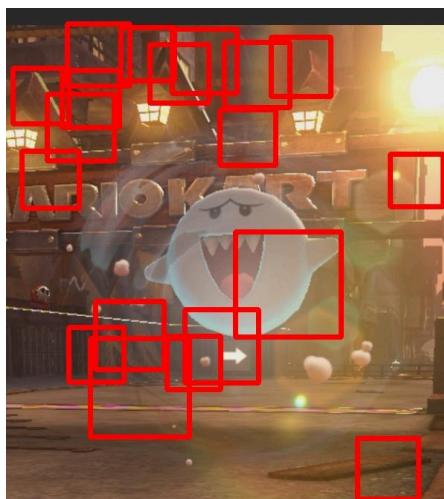
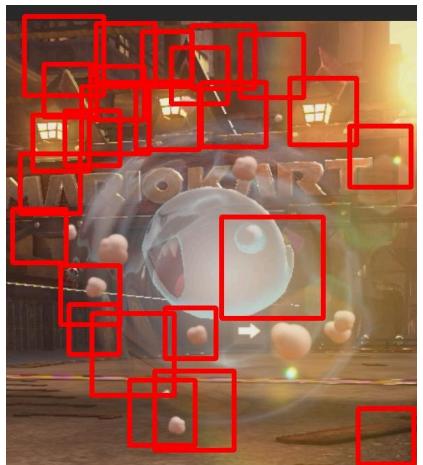
- w, h - минимальный размер окна распознанного объекта. При уменьшении, появляется много шумов, поскольку классификатор начинает цепляться за разные границы и маленькие объекты. При увеличении возможно отсутствие распознавания target объекта. Опытным путём было получено, что размер приведения в среднем появляется в окне 100 на 100.
- mnb - minNeighbours - минимальное количество соседей для bounding box, чтобы оставить его. Чем он ниже, тем больше шумов. Чем выше, возможна потеря target объекта. Из опыта - оптимальное значение для данного датасета 50.

Результаты:

На тестовых изображениях все приведения распознаны корректно.

Рисунки 9-13 - результаты распознавания





Однако на локациях в игре с большим количеством элементов окружения и света(как на последних 2 изображениях) возникает много false positives. Это объясняется тем, что прежде всего фильтры Хаара выявляют примитивные признаки такие как грани, линии, которые присутствуют на множестве объектов. Даже после обучения классификатор на самом деле не понимает что такое приведение “Бу”. В этом и состоит основное отличие обучения классификатора на основе каскадов Хаара от свёрточной нейронной сети. Классификатор не обучает сами фильтры, они известны заранее и остаются постоянными на протяжении всего обучения, поэтому они не могут собраться из примитивов линий и граней в более сложные признаки такие как “зуб”, “глаз” и т.д.

#### **Распознавание в реальном времени:**

Демонстрация распознавания в реальном времени.

[haar\\_cascade\\_mariokart.mkv](#)

На видео видно обильное количество false positives, однако само приведение распознаётся.

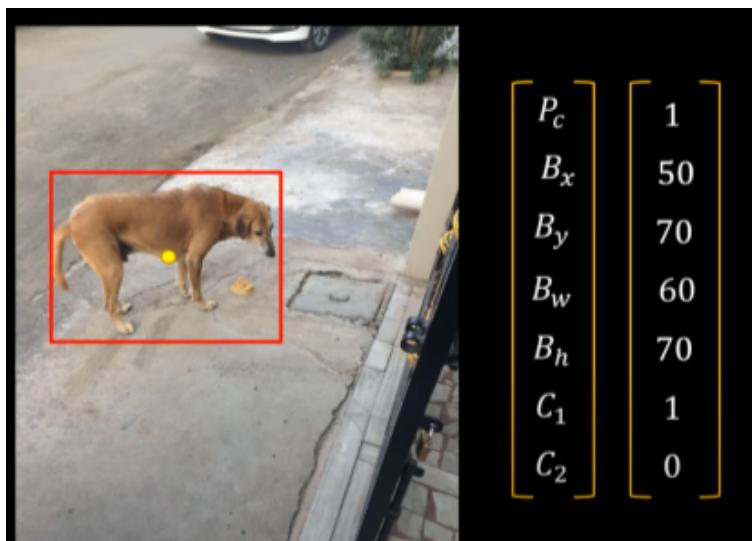
## Свёрточная нейросеть

Данный подход имеет ключевое отличие от каскадов Хаара. Нейросеть учится, а значит в ходе обучения меняются веса свёрток, что позволяет извлекать более сложные признаки из изображения.

В качестве нейросети была выбрана предобученная YOLO. YOLO - you only look once - названа так, поскольку позволяет за один feed forward получить и bounding box, и вероятность, с которой объект в нём находится. В отличии от каскадов Хаара здесь нет скользящего окна. Изображение обрабатывается за один пропуск через нейронную сеть.

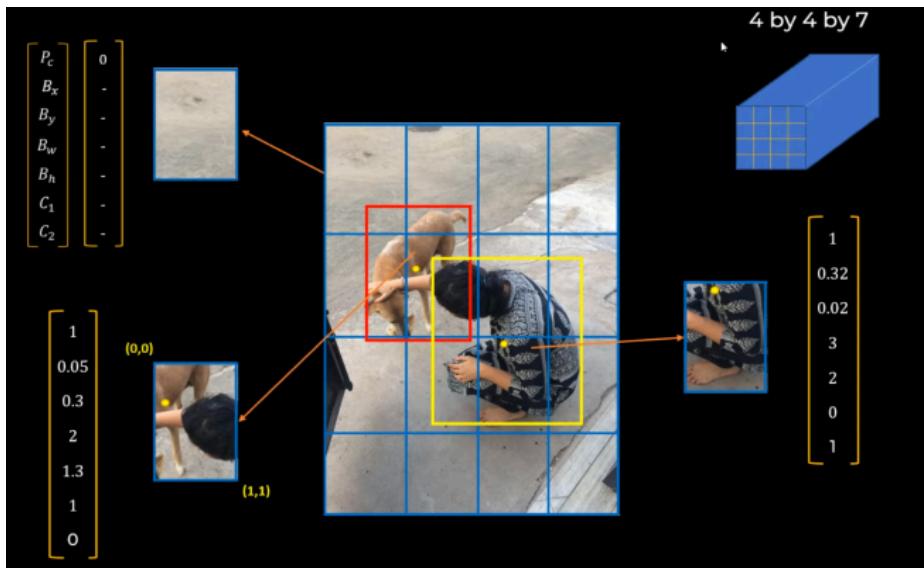
Данные представлены векторами с вероятностью, координатами bounding box и классом. Визуальное представление приведено на рисунке 14.

Рисунок 14 - пример разметки данных для YOLO



Ключевая идея YOLO состоит в том, что изображение разбивается на клетки - grid cells. Каждой клетке сопоставляется точно такой же вектор, как упомянутый ранее, где теперь координаты представляют собой центр bounding box, который попал в эту клетку. Координаты должны быть нормализованы. Это позволило не менять разметку на данном датасете, поскольку он уже соответствовал формату. Таким образом каждое изображение, разбитое на  $C \times C$  клеток можно представить как тензор  $C \times C \times \text{vec\_dim}$ . В нашем случае  $\text{vec\_dim}=6$ (вероятность, центр\_x, центр\_y, ширина, высота, класс 0/1). Этот тензор и подаётся на последние слои нейронной сети, которые вырабатывают предсказание. Демонстрация на рисунке 15.

Рисунок 15 - представление изображения как сетки клеток



Основные проблемы, которые могут возникнуть:

1. Несколько bounding box для одного и того же объекта. Часто встречаются ситуации, в которых один и тот же объект распознан несколько раз и имеет несколько bounding boxes.

Для предотвращения этого существует техника NMS - non max suppression. Она основана на метрике IoU - intersection over union, также известной как Jaccard Index. Формула приведена на рисунке 16.

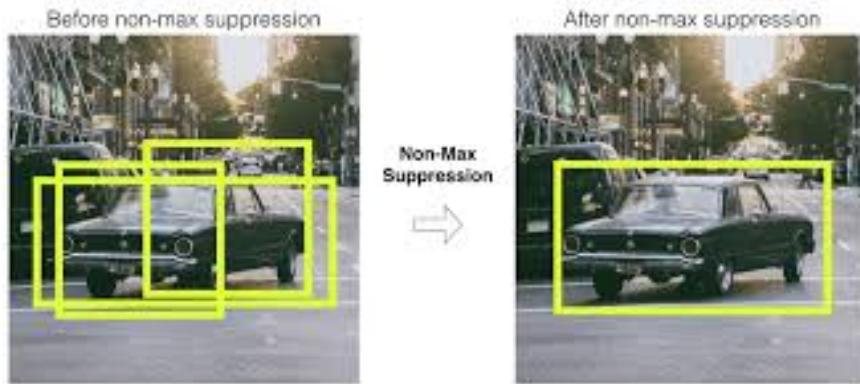
Рисунок 16 - индекс Жаккарда

$$JaccardIndex(A, B) = \frac{|A \cap B|}{|A \cup B|} :$$

Он показывает относительную меру перекрытия множеств.

Применительно к bounding box показывает насколько один закрывает другой относительно к их общей площади. Возникает резонный вопрос: почему бы просто не взять bounding box с наибольшей вероятностью? Дело в том, что могут быть изображения, на которых 2 объекта действительно находятся очень близко друг к другу и распознаны правильно, но с разной вероятностью. Удаление одного из них приведёт к неправильному результату. Поэтому проверяется перекрытие и если оно меньше заданной отсечки, то bounding box с меньшей вероятностью удаляется. Демонстрация NMS на рисунке 17.

Рисунок 17 - работа NMS

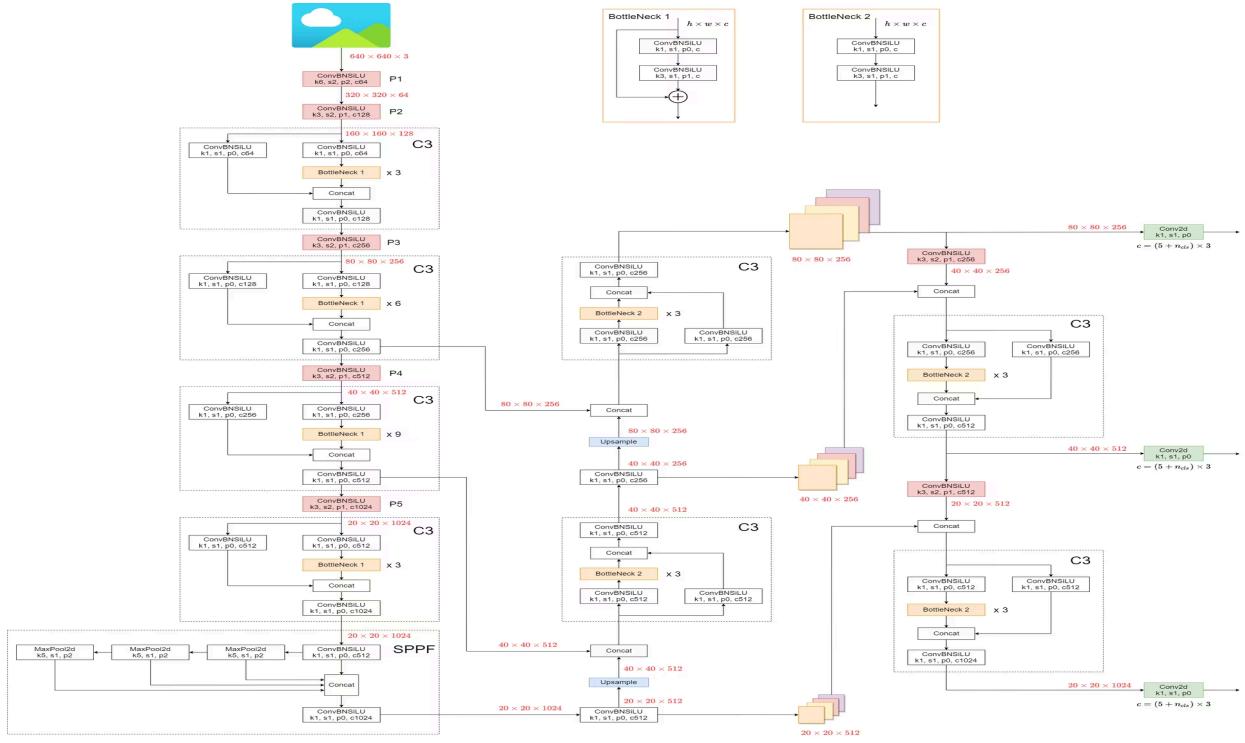


2. Проблема подсчёта bounding boxes. На изображении может быть множество объектов и пересчёт bounding box снова и снова может стать bottleneck для нейросети. Особенно это критично, если она работает в реальном времени. Для этого придумали технику Anchor boxes. Anchor boxes - это bounding boxes, которые высчитываются на этапе обучения из статистических значений bounding boxes из датасета. Как бы “отпечаток” данных. При подсчёте bounding box какого-либо объекта будут пересчитываться лишь отступы от ближайшего anchor box, центр которого лежит в той же клетке, где и предположительно находится объект. Подсчёт отступов от уже существующего Anchor box значительно менее расходная операция, чем создание нового. Впервые, этот подход был представлен в YOLOv5, которая и используется в данной работе.

Коротко о самой архитектуре YOLO:

На рисунке 18 приведена полная архитектура YOLO5

Рисунок 18 - архитектура YOLO5 с официального сайта ultralytics



Состоит из 3 частей: backbone, neck и head.

Backbone - последовательные свёртки и их конкатенация. Извлечение признаков. Вплоть до YOLO5 на этом этапе использовалась другая нейронная сеть Darknet-53 с 53 свёрточными слоями(информация из оригинальной статьи YOLO3). В ней последовательно накладываются Conv layer + Batchnorm + LeakyRELU. Однако можно было обнаружено, что в YOLO5 используется другая функция активации - SiLU - sigmoid linear unit. Есть также слои Bottleneck, которые нужны для снижения размерности для подачи на следующие слои.

Neck - комбинирование и конкатенация признаков

Head - свёртки для выдачи предсказания.

Основным отличием также является использование слоёв SPPF - spartial pyramid pooling fast на этапе backbone. Этот слой формирует признаки сразу из нескольких вариантов pooling, что позволяет не потерять мелкие объекты, а также иметь возможность распознавать объекты на разном масштабе. В отличии от YOLO, для классических CNN это большая проблема, т.к. они ожидают на входе изображение с объектом такого же масштаба, что и при обучении. Демонстрация работы этого слоя приведена на рисунке 19.

Рисунок 19 - работа слоя SPPF

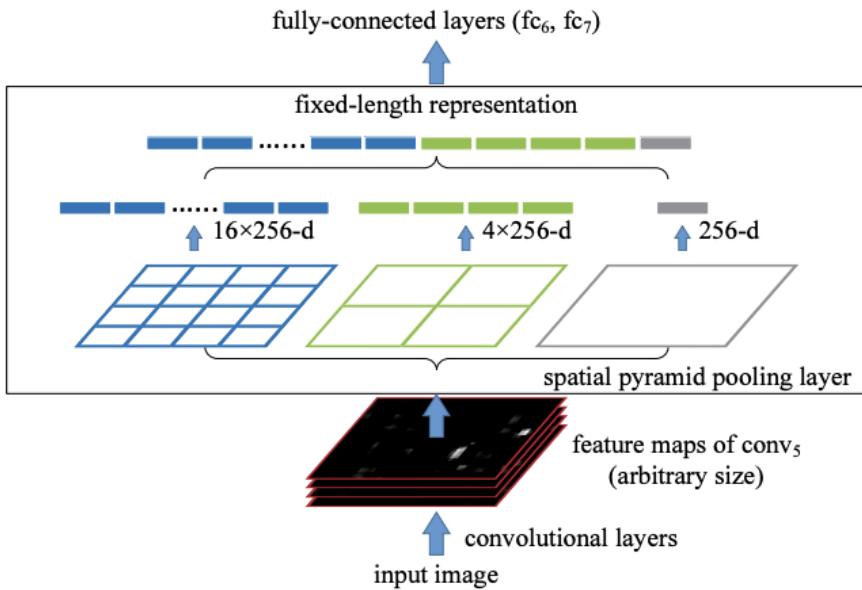
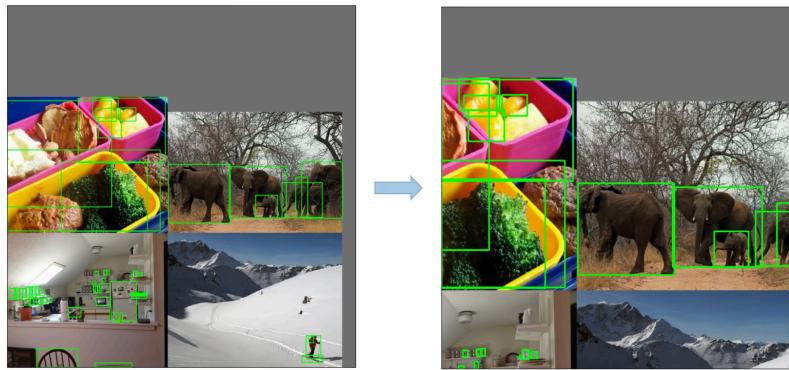


Figure 3: A network structure with a **spatial pyramid pooling layer**. Here 256 is the filter number of the  $\text{conv}_5$  layer, and  $\text{conv}_5$  is the last convolutional layer.

Last, but not least, аугментация данных. В YOLO5 были представлены разнообразные техники, которые позволяют улучшить multi scale распознавание и обучение, когда нейросеть обучается на изображениях увеличенных и уменьшенных в размере(обычно от 0.5 до 1.5 от размера картинки). Примеры применяемых аугментаций данных приведены на рисунках 20, 21.

Рисунки 20, 21 - аугментации данных в YOLO5





Для дообучения предобученной YOLO5 использовался тот же датасет, что и в случае с каскадами хаара. Однако для yolo5 требовалось описание датасета в yaml формате, которое было сделано в этом файле [dataset.yaml](#)

В ходе множества экспериментов было выявлено, что обучение с следующими параметрами даёт наилучший по точности результат:

```
python ..\yolov5\train.py --imgsz 590 --single-cls --multi-scale  
--epochs 10 --data .\dataset.yaml --weights .\yolov5s.pt
```

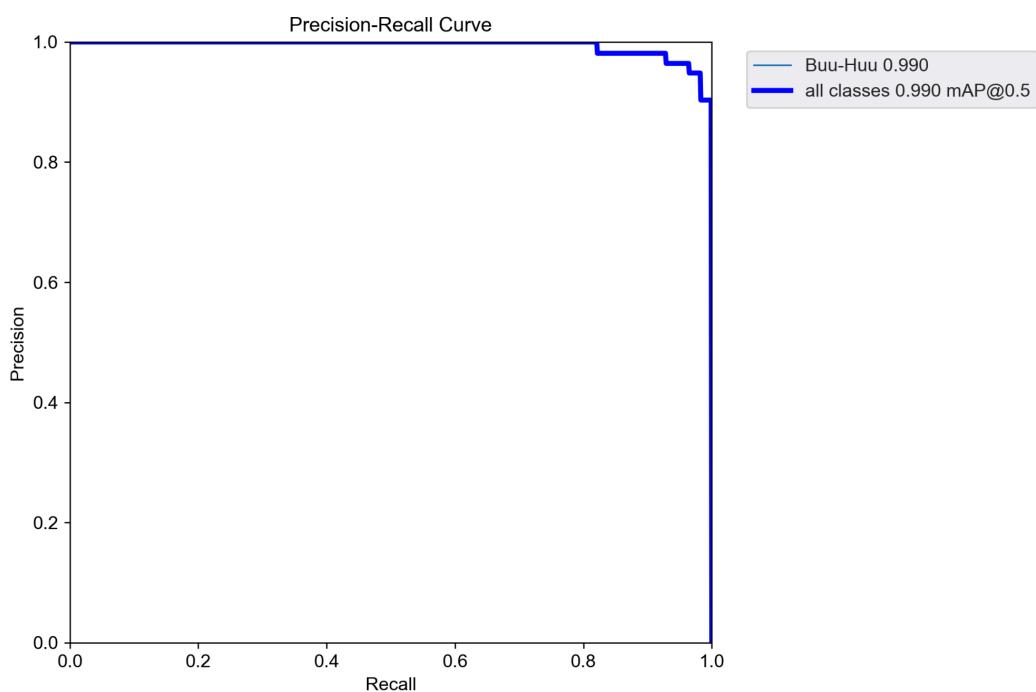
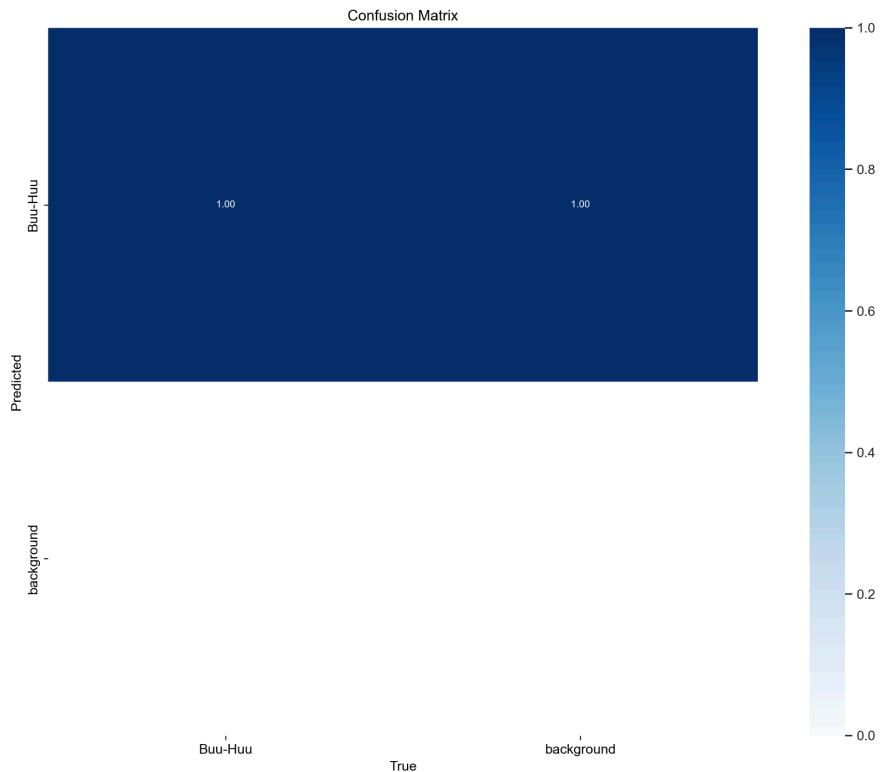
Основные параметры:

- imgsz - размер, к которому будут приводится изображения. Оригинальные изображения размера 590 на 670 будут приведены к 590 на 590
- single-cls - обучение на одном классе.
- multi-scale - обучение на разных масштабах изображения. Стоит сделать важное замечание, что без этой опции YOLO почти ни чем не лучше, чем обычные CNN, поскольку принимает только изображения того же масштаба. Распознавание объектов в реальной игре было бы невозможно.
- epochs - количество эпох
- data - путь к yaml описанию датасета
- weights - изначальные веса

### Результаты обучения:

Результаты с confusion matrix, precision-recall кривыми и результатами распознавания приведены на рисунках 22-26.

Рисунки 22-26 - анализ полученных результатов обучения



Результаты валидации:





## Аугментация данных:



## Результаты на тестовой выборке:

На тестовой выборке результаты распознавания показали высокую точность.

Результаты на рисунке 27-29.

Рисунки 27-29 - результаты на тестовой выборке.





Важно подчеркнуть, что на рисунке 27 такое распознавание стало возможным только благодаря аугментации и multiscale обучению. Без этого, вообще говоря, нейросеть не могла бы распознать приведение, поскольку оно отличается по размеру от тренировочной выборки более чем в 1.5 раза, что критично для свёрток.

**Распознавание в реальном времени:**

Демонстрация распознавания в реальном времени:

[yolo\\_mariokart.mkv](#)

## **Сравнение и выводы:**

В данной статье были 2 подхода к распознаванию изображений - свёрточная нейронная сеть YOLO и классификатор на основе каскадов Хаара. В отличии от каскадов Хаара, нейросеть не выдаёт такого количества false positives ни при распознавании в реальном времени, ни при распознавании изображения. Более того, bounding box всегда присутствует на правильно распознанных изображениях и соответствует размеру из разметки с небольшой погрешностью. Это объясняется тем, что нейросеть обучена извлекать более сложные признаки и с разных масштабов, что делает её более гибкой к разным размерам изображений, а также пассивной к граням и линиям, потому что они уже преобразованы в более сложные и редкие признаки такие как "зуб", "лицо", "глаз" и т.д. Каскады Хаара не показали такой же гибкости в распознавании. Поэтому, с точки зрения точности распознавания, лидерство за нейросетью.

Однако из видео с распознаванием в реальном времени наглядно видно, что нейросеть распознаёт примерно в 10 раз медленнее, чем классификатор на основе каскадов Хаара. Это можно объяснить тем, что YOLO, как и любая свёрточная нейронная сеть, хорошо параллелизуется и лучше работает на GPU, чем на CPU. Каскады Хаара же имеют последовательную природу, описанную на рисунке 8, поэтому лучше работают на CPU. Эксперимент с распознаванием в реальном времени мною был проведён на компьютере без мощного графического ускорителя. Таких компьютеров большинство, поэтому с точки зрения скорости и доступности лидерство за каскадами Хаара.

## Источники:

1. <https://pjreddie.com/media/files/papers/YOLOv3.pdf>
2. <https://www.cs.cmu.edu/~efros/courses/LBMV07/Papers/viola-cvpr-01.pdf>
3. [https://docs.ultralytics.com/yolov5/tutorials/architecture\\_description/#1-model-structure](https://docs.ultralytics.com/yolov5/tutorials/architecture_description/#1-model-structure)
4. <https://youtu.be/svn9-xV7wjk?si=2ZdmkfLOIBZ15iiO>
5. <https://youtu.be/ag3DLKsl2vk?si=QPOGj-awOMiFGvVD>
6. <https://youtu.be/ZSgg-fZJ9tQ?si=L8mQypOs1mCwfmNp>
- 7.

## Исходный код:

Ссылка на репозиторий с исходным кодом:

[https://github.com/kargamant/mariokart\\_recogniser](https://github.com/kargamant/mariokart_recogniser)