Зотов Сергей Сергеевич
Zotov Sergei Sergeevich
Аспирант
Postgraduate student
Яковлев Андрей Александрович
Yakovlev Andrey Alexandrovich
Колчинцев Дмитрий Александрович
Kolchintsev Dmitry Alexandrovich
Студенты
Students
Дальневосточный федеральный университет
Far Eastern Federal University

ОБНАРУЖЕНИЕ ОБЪЕКТОВ В РЕАЛЬНОМ ВРЕМЕНИ С ПОМОЩЬЮ АЛГОРИТМОВ РАСПОЗНАВАНИЯ YOLO
OBJECTS DETECTED IN REAL TIME WITH THE YOLO RECOGNITION ALGORITHMS

Аннотация на русском языке: Статья посвящена анализу алгоритма распознавания объектов YOLO. В статье приведены результаты тестирования трёх существующих на данный момент версий YOLO. При анализе алгоритмов использовались методы классификации, синтеза и сопоставительного анализа. На основе проведенного анализа авторами предложены дополнительные возможности алгоритма, позволяющие облегчить работу с ним в том числе неспециалистам. Описывается программная реализация указанных возможностей. В заключении представлено обобщенная сравнительная характеристика всех моделей, отражающая динамику их развития.

The summary in English: The article is devoted to the analysis of the YOLO object recognition algorithm. The article shows the results of testing the three existing versions of YOLO. When analyzing algorithms, methods of classification, synthesis and comparative analysis were used. Based on the analysis, the authors proposed additional features of the algorithm that make it easier to work with him, including non–specialists. The software implementation of these features is described. In conclusion, a generalized comparative characteristic of all models is presented, reflecting the dynamics of their development.

Ключевые слова: компьютерное зрение, нейронные сети, распознавание образов, YOLO, селективный поиск, опорные прямоугольники, обрамляющие прямоугольники, фреймворк Darknet.

Key words: computer vision, neural networks, pattern recognition, YOLO, selective search, support rectangles, framing rectangles, Darknet framework.

С развитием современных технологий, стала возможна реализация нейронных сетей с малой вычислительной погрешностью, на основе которых ученые смогли создать новые системы автоматического распознавания объектов. Однако до сих пор в ряде задач им не удалось достичь удовлетворительного результата. В силу широких возможностей применения

систем распознавания (идентификация и верификация данных, системы слежения и обнаружения объектов, системы автоматического управления транспортом и т.д.) проблемы решения этих задач являются актуальными. С этой целью создается большое количество алгоритмов компьютерного зрения, один из которых рассматривается в данной работе.

YOLO

Предложенный Джозефом Редмоном (Joseph Redmon) алгоритм YOLO представляет собой единую нейронную сеть, которая применяется сразу ко всему изображению. Она одновременно выделяет и предсказывает bounding boxes и вероятности нахождения в них объектов различных классов. Используя эту систему, нужно всего один раз пропустить изображение через нейронную сеть, чтобы предсказать объекты и их расположение. Это обеспечивает YOLO высокую скорость по сравнению с другими методами обнаружения объектов рисунок 1.

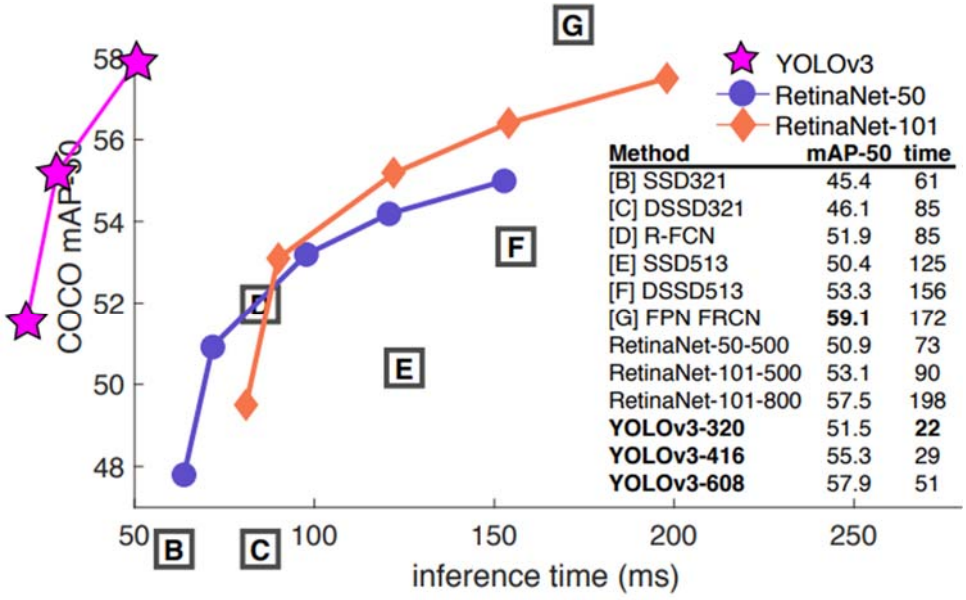

Рисунок 1. Графики зависимости скорости от точности для различных моделей распознавания

Изначально алгоритм YOLO базировался только на фреймворке DARKNET, однако на сегодняшний день он был перенесен на более популярные системы глубокого обучения Tensorflow и Pytorch.

Проанализировав YOLO, компания NVidia разработала собственную нейронною сеть DetectNet с использованием системы глубокого обучения DIGITS и похожей архитектурой [2, с. 2].

YOLO предполагает обнаружение объектов как единую задачу регрессии, которая начинается с пикселей входного изображения, а завершается вычислением координат обрамляющих прямоугольников и вероятностей принадлежности классам. При прогнозировании YOLO рассматривает изображение глобально. В отличие от методов скользящего окна и методов областных предложений, YOLO обрабатывает все изображение целиком во время обучения и тестирования, поэтому он неявно кодирует контекстную информацию о классах, а также их внешний вид, благодаря чему в этой модели ошибочное срабатывание детектора на фоне изображения менее вероятно, по сравнению с Fast R–CNN. При обучении на естественных изображениях и тестировании на художественных работах YOLO также превосходит DPM и R–CNN, с большим отрывом. Поскольку YOLO использует сильное обобщение, вероятность его ошибки при применении к новым входным данным также мала. [3, с. 5]

YOLOv1

Архитектура YOLO была основана на модели распознавания GoogleNet. YOLOv1 имеет 24 сверточных слоя, за которыми следуют 2 полносвязанных слоя FC рисунок 2. Вместо входных слоев, используемых в GoogleNet YOLOv1 использует редукционные слои 1 × 1, уменьшая глубину карт признаков, за которыми следуют слои 3×3. В качестве активационной функции в YOLOv1 применяется Leaky rectified linear unit (LReLu) (1):

$$\begin{cases} f(x) = \alpha x \text{ при } x < 0, \\ f(x) = x \text{ при } x \geq 0; \end{cases} \quad (1)$$

где α – малая константа (~ 0,01).

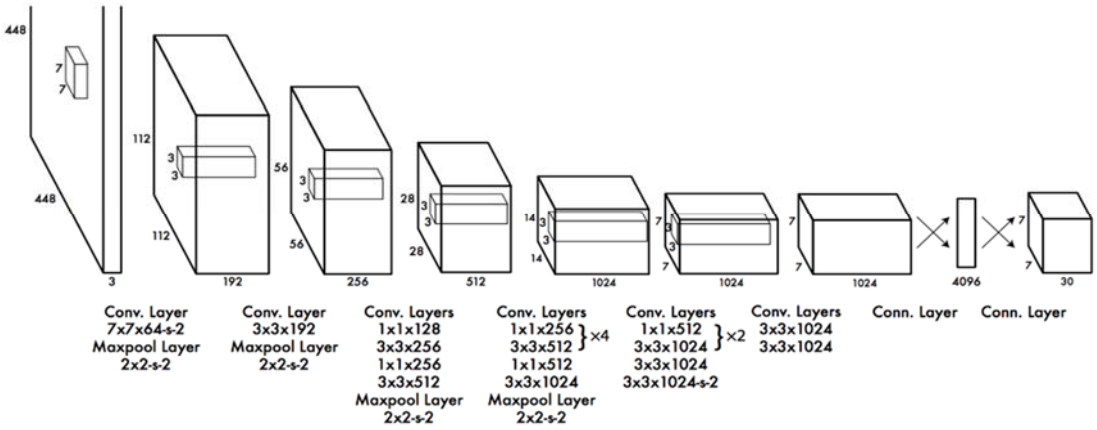

Рисунок 2. Архитектура YOLOv1

На вход сверточной нейронной сети подаётся трёхканальное изображение размером 448×448. После прохождения 20 сверточных слоёв исходный тензор, содержащий полученные карты признаков, имеет размерность 14×14×1024. К нему применяется ряд слоев с LRelu, после чего размерность тензора становится 7×7×30, и к нему применяется процедура детектирования.

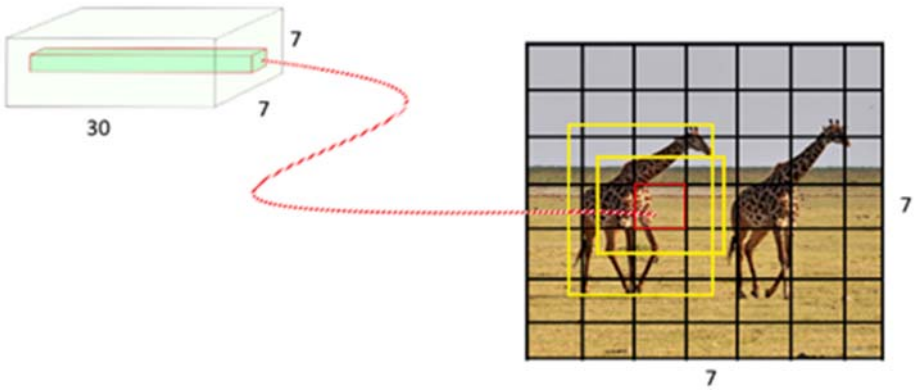

Рисунок 3. Вектор из тензора 7×7×30, соответствующий выделенной ячейке

Фактически, на исходное изображение, наносится сетка размером 7×7 (рисунок 3). Каждой ячейке сети сопоставляются по два bounding box, в которых могут находиться объекты. Однако YOLOv1 способен регистрировать лишь один из объектов, центры которых лежат в одной ячейке. Далее каждой ячейке ставится в соответствие вектор 1×30. В 5

первых компонентах вектора передаются следующие параметры первого bounding box в данной ячейке:

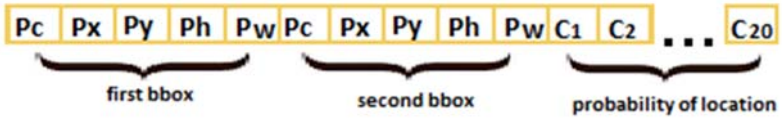

Рисунок 4. Вектор bounding box

где P_C – вероятность правильного определения границ bounding box. P_x, P_y– координаты центра bounding box. P_h, P_w – высота и ширина bounding box (от 0 до 1 по отношению ко всему изображению).

Следующие 5 позиций занимают параметры, соответствующих второму bounding box этой же ячейки. Оставшиеся 20 компонент вектора C_1, ..., C_{20} соответствуют классам обнаружения. В каждой из них находится вероятность прогноза того, что центр объекта данного класса расположен в данной ячейке. Однако данное значение не соотнесено с конкретными bounding box. Для установления этого соответствия вероятности P_C для обоих прямоугольников перемножаются поочередно на каждую из 20 С–вероятностей, в результате чего получаются два вектора размерностью 1×20, характеризующие вероятности наличия в данных прямоугольниках объекта каждого из классов рисунок 5.

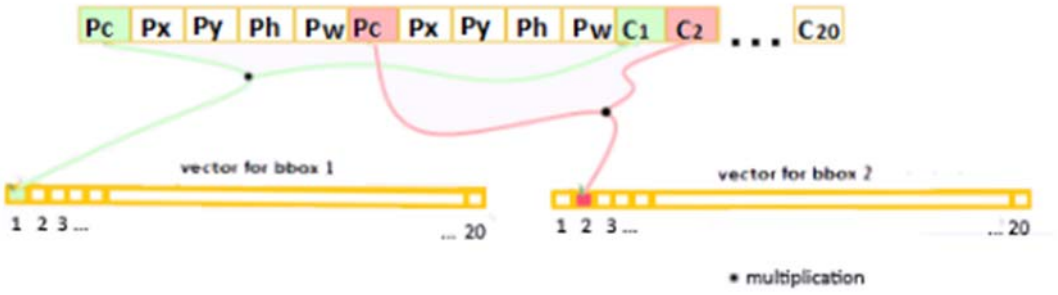

Рисунок 5. Процесс перемножения P_C на С–вероятности

Такая последовательность действий повторяется для каждой ячейки, в итоге образуется 98 векторов вероятностей как показано на рисунке 6.

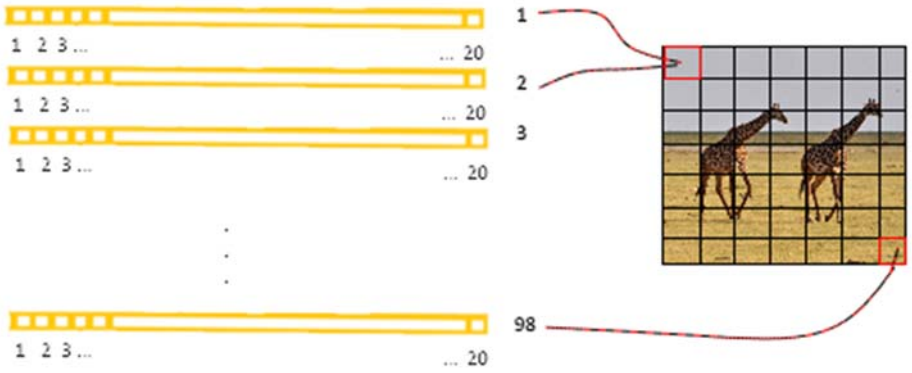

Рисунок 6. Вектора вероятностей

Далее в каждом из векторов последовательно рассматривается один из классов. Компонентам, у которых значение меньше порогового числа, ставятся в соответствие нули, после чего вектора сортируются по убыванию. Однако на данном этапе возможна ситуация, когда один с тот же объект с высокой вероятностью обнаруживается несколькими bounding box. Для устранения данной проблемы в YOLOv1 применяется алгоритм Non–Maximum Suppression (NMS). Он заключается в простом сравнении прямоугольников с наибольшим значением вероятности нахождения в нём объекта класса со всеми остальными прямоугольниками. Сравнение производится с помощью функции Interception over Union (IoU) (рисунок 7), которая отбрасывает прямоугольники, пересекающиеся со сравниваемым прямоугольником больше, чем на 50%. Этот алгоритм проводится для всех последующих ненулевых значений, последовательно для каждого класса.

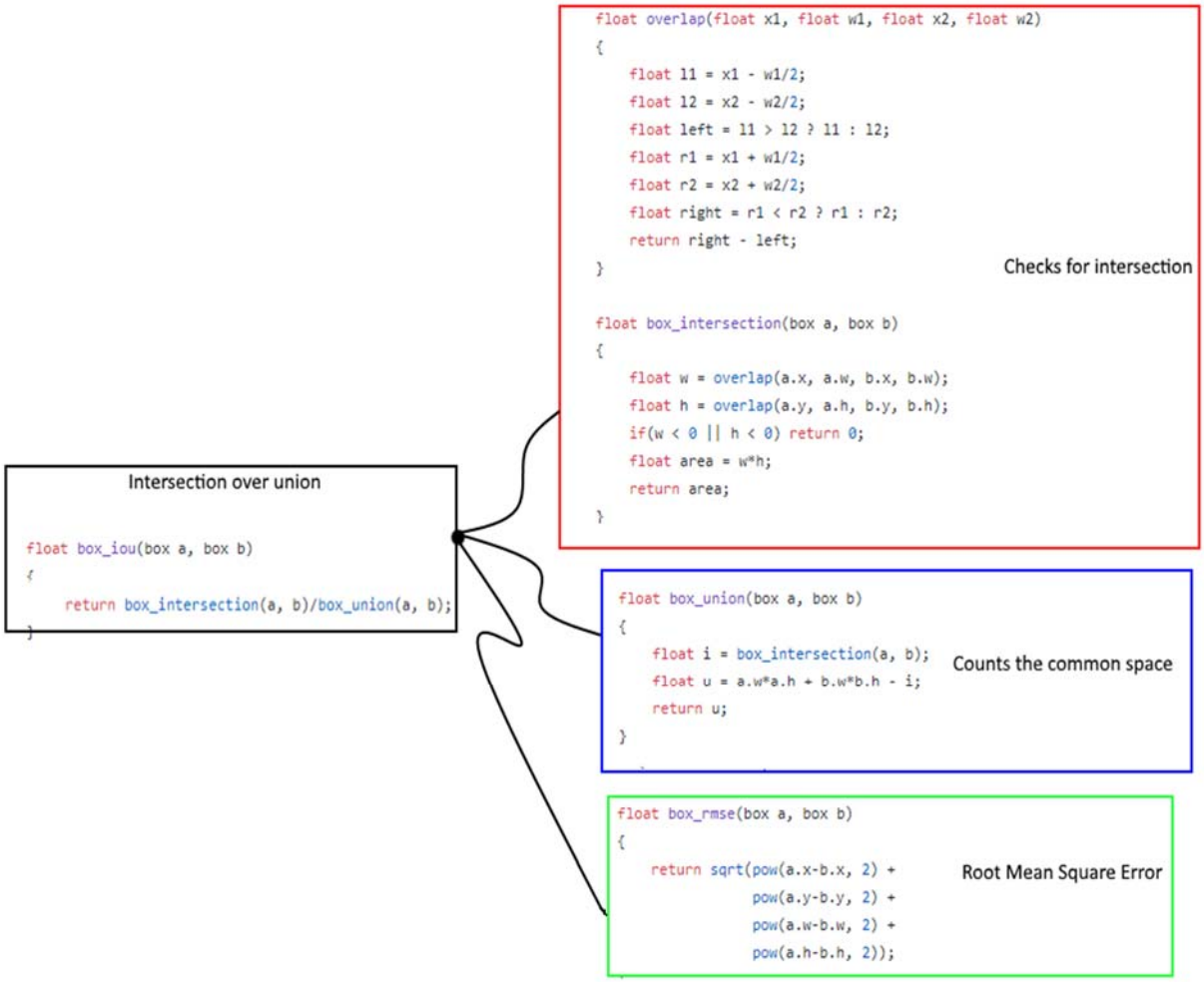

Рисунок 7. Реализация IoU в YOLO

Таким образом из имеющихся 98 выделенных bounding boxes отбрасывается часть, несоответствующая критериям отбора, а остальные печатаются на изображении.

YOLOv2

Тестирование выявило ряд значительных недостатков YOLOv1. Анализ показал, что по сравнению с Fast R–CNN YOLOv1 делает значительное количество ошибок в локализации объектов и при этом имеет относительно низкую точность. Происходит это по причине того, что в каждой ячейке YOLOv1 может определить только один объект, но довольно часто центры двух и более объектов попадают в одну ячейку. Таким образом, главной задачей YOLOv2 стало повышение точности и качества локализации при

сохранении своей скорости. Вместо растущей тенденции к укрупнению сети, для повышения производительности в YOLOv2 упрощается ее работа и вывод.

Если первая версия алгоритма предсказывает координаты bounding boxes напрямую, используя полносвязные слои поверх сверточного экстрактора признаков, то в YOLOv2 этот процесс происходит иначе, с использованием опорных прямоугольников (англ. – anchor boxes) – стандартного набора прямоугольников с определенными значениями длины и ширины, которые получаются на этапе обучения с помощью метода кластеризации k–means. Каждой ячейке сетки сопоставляются по 5 таких anchor boxes разного размера (рисунок 8), и вместо прямого предсказания bounding box сеть определяет отклонения от самого подходящего к объекту anchor box. Такие отклонения ограничены, что позволяет каждому anchor box фокусироваться на объекте определенной формы. Тоже происходит в случае, если центры разных объектов находятся в одной ячейке: каждому из них ставится соответствующий anchor box.

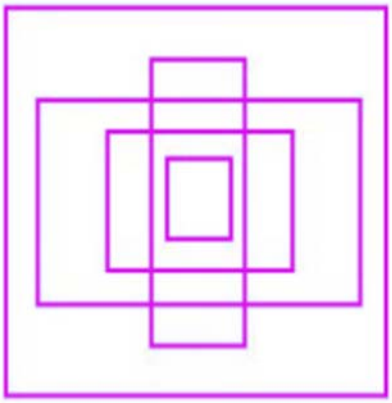

Рисунок 8. Anchor boxes

Данное нововведение позволило YOLOv2 избавиться от полносвязных слоев, ранее отвечавших за предсказание bounding boxes, выводя вместо них результат, полученный от сверточных слоев, т.е YOLOv2 перешла на

полностью сверточную нейронную сеть (FCN). Ниже представлено сравнение YOLO и YOLOv2 обученных на выборке Pascal VOC 2007+2012 [4, с. 3].

Таблица 1. Сравнение YOLOv1 и YOLOv2

Detection Frameworks	**mAP**	**FPS**
YOLOv1	63.4	45
YOLOv2 288 × 288	69.0	91
YOLOv2 352 × 352	73.7	81
YOLOv2 416 × 416	76.8	67
YOLOv2 480 × 480	77.8	59
YOLOv2 544 × 544	78.6	40

Также в YOLOv2 предсказания классов прикреплены к каждому bounding box, а не к ячейке, как это было в YOLOv1. Из–за введения anchor box и добавления новых классов детекции размеры векторов вероятности изменились на 1×425. Их структура представлена на рисунке 9.

Рисунок 9. Структура вектора вероятностей

В YOLOv2 вектор состоит из 5 разделов, по одному для каждого из anchor box. Параметры границ обнаруженных объектов вписываются в компоненты соответствующих anchor boxex. Для тех anchor boxes, для которых не нашлось подходящего объекта с центром в данной ячейке изображения компоненты Pc = 0.

YOLOv2 также может работать с различными разрешениями для достижения компромисса между скоростью и точностью. Несмотря на разные входные разрешения обученная модель остается той же самой, без изменения весов.

В основной концепции YOLOv2 использует входное разрешение 416 × 416 и убран один pooling слой, чтобы увеличить пространственный выход сети до 13×13 (вместо 7×7 в YOLOv1). Это позволило улучшить обнаружение небольших объектов, так как повысилось количество предсказанных bounding box.

Еще одним новшеством YOLOv2 стало введение Batch Normalization. На выходе сверточного слоя значения в полученной карте признаков имеют хаотичное распределение. При поступлении этих данных на следующий слой большая дивергенция значений может приводить к возникновению исключений сети. Добавление batch нормализации на всех сверточных слоях в YOLOv2 привело к упорядочиванию модели, а также позволило повысить ее точность более чем на 2% [6, с. 5].

YOLOv2 использует пользовательскую глубокую архитектуру darknet–19, изначально 19–слойную сеть, дополненную еще 11 слоями для обнаружения объектов. С 30–слойной архитектурой YOLOv2 часто не справляется с обнаружением небольших объектов [5, с. 5]. Это было связано с потерей мелкозернистых признаков по мере того, как слои уменьшали входные данные. Чтобы исправить это, YOLOv2 объединял карты признаков из предыдущего слоя для их захвата.

Однако архитектуре YOLOv2 по–прежнему не хватало некоторых из наиболее важных элементов, которые теперь являются основными в большинстве современных алгоритмов. Отсутствие остаточных и блоков пропускания и отсутствие повышения дискретизации. YOLOv3 включает в себя все эти недостающие функции.

YOLOv3

YOLO v3 использует вариант Darknet, который изначально имеет 53–слойную сеть, обученную на базе данных Imagenet [1, с 3]. Для задачи обнаружения на нее накладывается еще 53 слоя, что дает нам полностью

сверточную архитектуру для YOLO v3 рисунок 10. Это является причиной медлительности YOLOv3 по сравнению с YOLOv2.

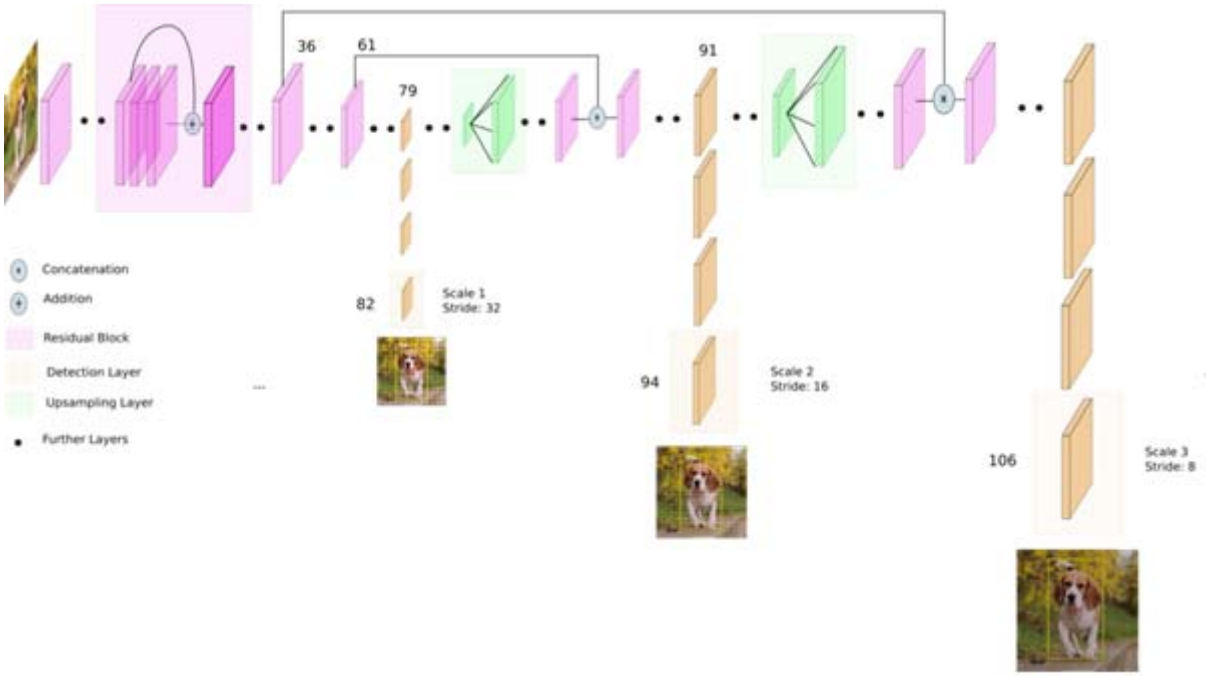

Рисунок 10. Архитектура YOLOv3

Новая архитектура может похвастаться недостающими в Darknet–19 функциональными слоями. Самой характерной особенностью YOLO v3 является то, что он делает обнаружение в трех разных масштабах, которые точно задаются путем понижающей дискретизации размеров входного изображения на 32, 16 и 8 соответственно [6, с. 7]. В YOLO v3 обнаружение выполняется путем применения 1 x 1 ядер обнаружения на картах признаков трех разных размеров в трех разных уровнях сети [1, с. 3,].

Размер ядра или фильтра обнаружения равен $1 \times 1 \times (B \times (5 + C))$. Здесь B – количество bounding box, которое может предсказать ячейка, число 5 включает в себя 4 – атрибута bounding box и вероятность правильного определения объекта, а C – количество классов. В YOLO v3, обученной на базе данных COCO, B = 3 и C = 80, поэтому размер ядра равен $1 \times 1 \times 255$.

Для первых 81 слоев разрешение изображения уменьшается сетью, так что 81–й уровень имеет шаг 32. Под шагом сети понимается коэффициент, который показывает, во сколько раз выходное изображение слоя меньше

входного изображения в сеть. Поэтому для исходного изображения с разрешением 416 × 416, результирующая карта признаков будет иметь размер 13 × 13. Первое обнаружение производится на слое 82 с использованием фильтра 1 × 1, что дает тензор 13 × 13 × 255.

На следующем этапе сеть вновь обращается к карте признаков из слоя 79, пропуская ее через несколько сверточных слоев и повышая дискретизацию в два раза, так что размер карты становится 26×26. Для того, чтобы учесть признаки более раннего слоя, полученная карта объединяется с картой признаков из слоя 61 и результат объединения вновь подвергается нескольким сверточным слоям 1×1, после чего на 94–м слое выполняется второе обнаружение, которое выдает тензор 26 × 26 × 255.

Аналогичная процедура повторяется для слоя 91, объединение которого уже совершается с картой признаков из слоя 36. На 106–м слое происходит финальное детектирование с результирующим тензором размером 52 × 52 × 255.

Обнаружение в разных слоях помогает решить проблему распознавания небольших объектов, которая была актуальна для YOLO v2. Слой 13 × 13 отвечает за обнаружение больших объектов, слой 52×52 за обнаружение маленьких объектов, а слой 26×26 отвечает за средние объекты. Так же слои полученные путем понижающей дискретизации объединяются с предыдущими слоями, что помогает сохранить признаки, которые способствуют обнаружению небольших объектов.

Для входного изображения того же размера YOLOv3 предсказывает больше bounding box, чем YOLOv2. Например, для изображения с разрешением 416 × 416 YOLOv2 предсказывает 13 × 13 × 5 = 845 bounding box. В YOLOv3 для того же изображения количество прогнозируемых прямоугольников составляет 10 647. Каждая ячейка может прогнозировать 3

bounding box, используя 3 anchor box. Поскольку имеется три масштаба, количество anchor box составляет 10647 (2):

$$13*13*3+26*26*3+52*52*3 = 10647 \quad (2)$$

Этим и обуславливается медлительности YOLO v3 по сравнению с YOLOv2.

Программная реализация

После рассмотрения существующих алгоритмов YOLO была исследована возможность их использования для дальнейшего внедрения в системы анализа. При исследовании были использованы версии YOLO написанные с использованием фреймворка Darknet–53.

Так как система распознавания обрабатывает динамически изменяюшуюся обстановку, возможны ложные обнаружения. В оригинальном YOLO реализован покадровый вывод результатов работы программы с видеопотоком, поэтому происходящие в кадрах изменения сильно отражаются на выходных данных. Зачастую из–за низкого разрешения фрагмента видеофайла или из–за присутсвия в кадре объекта, напоминающего с определенного ракурса представителя другого класса, происходят ложные срабатывания детектора. Для их устранения было принято решение изменить исходный код системы распознавания и понизить частоту выводных значений до 1 Hz, с последующим вычислением усредненного количество объектов для каждого из классов, обнаруженных за секунду. Функция усреднения высчитывается, исходя из текущего значения FPS устройства.

Для тестирования был выбран видео фрагмент, в каждом кадре которого находились две собаки и один человек. Из–за того, что объекты постоянно перемещались, оригинальная YOLOv3 ошибочно обнаружила в кадре представителя класса “cat”, а также на другом из кадров зафиксировала лишь один из двух объектов класса “dog”. В измененной

версии же за счет усреднения вывода данных ошибок не наблюдалось, рисунок 11.

Формат выводных значений был так же переработан: при обработке кадра вместо последовательного вывода информации о каждом обнаруженном объекте на выход поступает суммарное количество обнаруженных объектов по каждому из классов. Вся полученная информация сохраняется в журналах детектирования. К тому же был разработан графический интерфейс, который позволяет использовать модель YOLO.

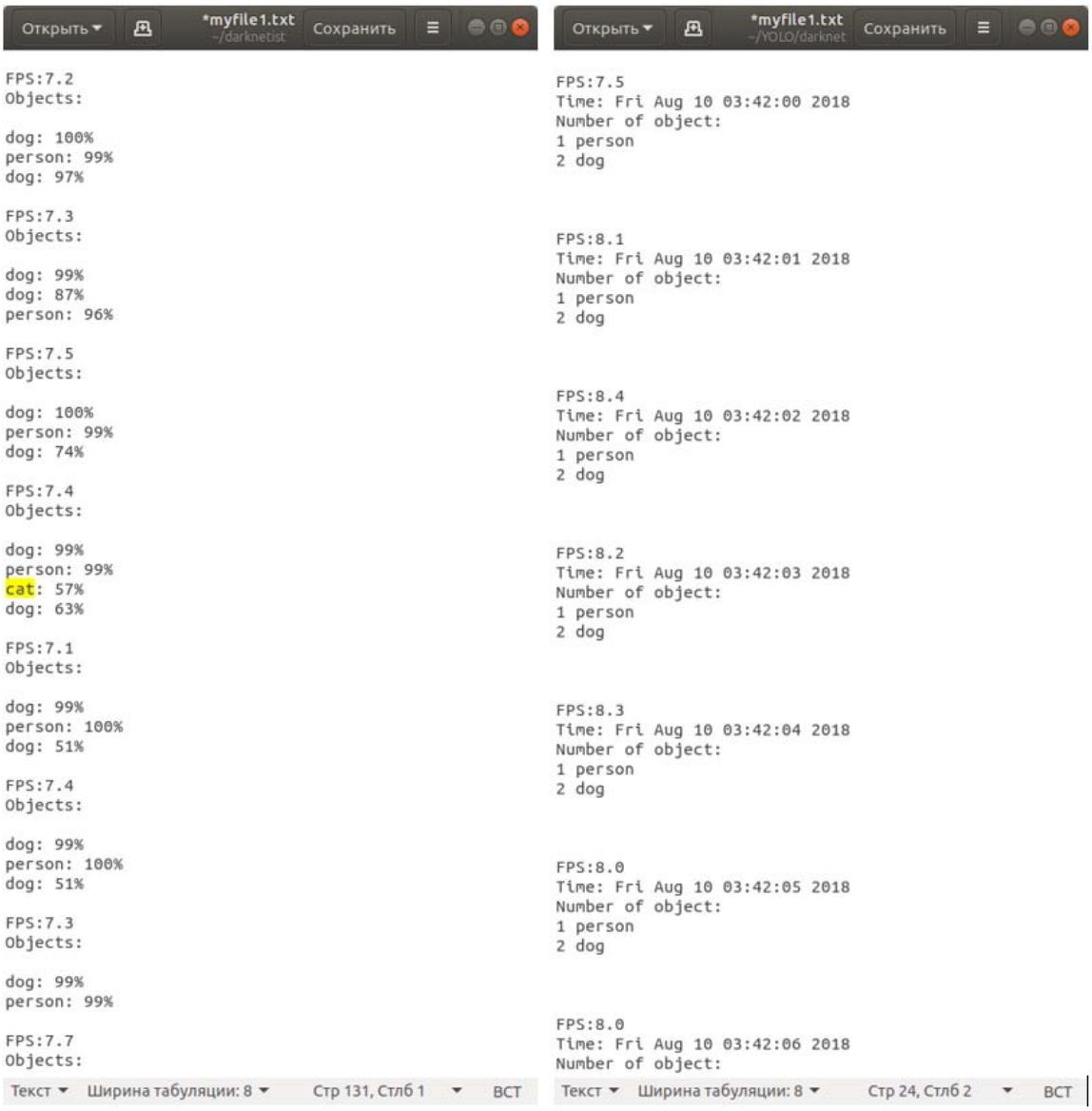

Рисунок 11. Результаты анализа видеофайла, записанные в журнале детектирования

При работе с алгоритмом авторами была обнаружена проблема в отсутствии графического интерфейса и формализации результатов работы программы для неспециалистов. С этой целью было разработано соответствующее программное обеспечение на базе операционной системы LINUX представленное на рисунках 12–13. Оно позволило использовать все возможности YOLO в более удобном формате.

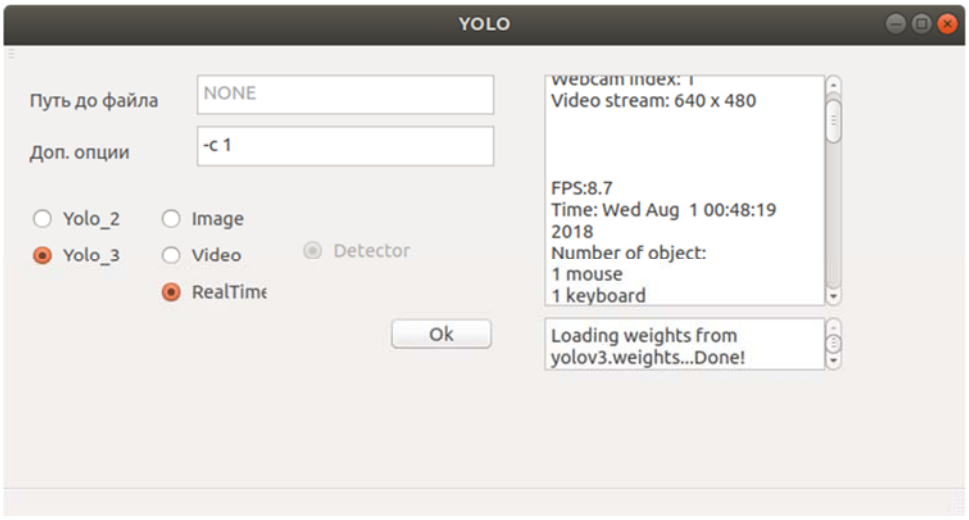

Рисунок 12. Графический интерфейс программы

Так же была произведена формализация выходных данных. При обработке изображения выводится суммарное количество обнаруженных объектов по каждому из классов. В режиме реального обнаружения, а также при обработке видеофайла вместо вывода результатов обнаружения для каждого кадра, записывается среднее значение найденных объектов за одну секунду, а также соответствующее этой секунде время. Это также позволило снизить процент неверно найденных объектов. Все результаты работы можно найти в правом верхнем окне программы, а также в файлах журнала детектирования, рисунок 12–13.

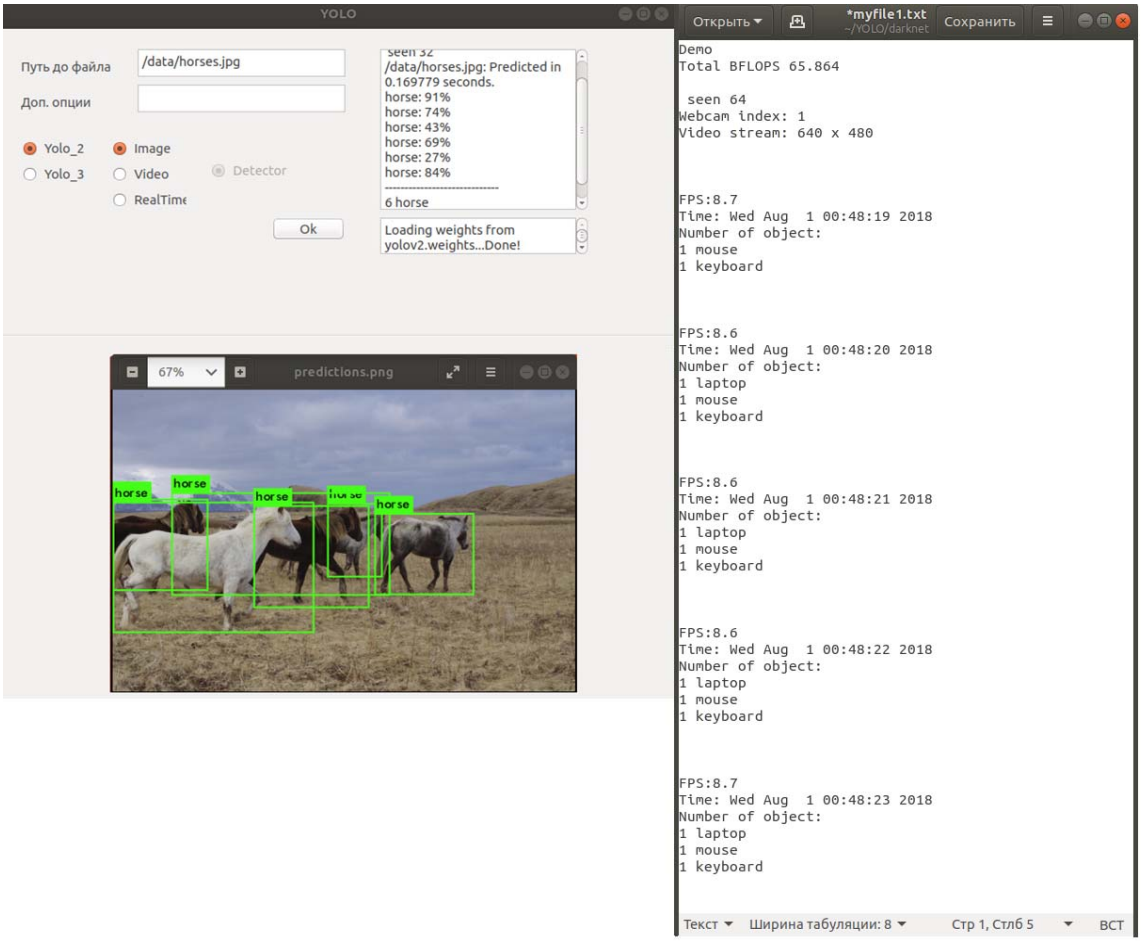

Рисунок 13. Вид графического интерфейса и журнал детектирования

Заключение.

В данной работе был произведен сравнительных анализ трех версий алгоритма распознавания YOLO и реализовано ПО на основе данных алгоритмов. Четко прослеживается явная модернизация технологий ведущее к последовательному увеличению точности распознавания без значительной потери скорости.

Таблица 2. Поддерживаемые технологии в YOLO

Технология	**YOLO**	**YOLOv2**	**YOLOv3**
Anchor boxes	No	5	3
Convolution layers	29	29	106
Cell	7x7	13x13	13x13, 26x26, 52x52

Несмотря на то, что даже самая совершенная версия YOLO все еще уступает некоторым алгоритмам, таким как Faster R–CNN и SSD, в точности распознавания, YOLO остается актуальным из–за своей скорости (рисунок 1).

Перспективы развития алгоритма добавляет тот факт, что YOLO активно переносится на популярные фреймворки. Так же, в отличии от конкурентов, YOLO не требует больших аппаратных мощностей, благодаря чему данную модель можно реализовать на различных платформах, вплоть до мобильных устройств.

Литература:

1. Kathuria A. What's new in YOLO v3?. [Электронный ресурс] URL: https://towardsdatascience.com/yolo–v3–object–detection–53fb7d3bfe6b (дата обращения 12.07.2018).

2. Redmon J. Darknet: Open Source Neural Networks in C. [Электронный ресурс] URL: http://pjreddie.com/darknet/ .(дата общения 05.07.2018).

3. Redmon J. YOLO: Real–Time Object Detection. [Электронный ресурс] URL: http://pjreddie.com/darknet/yolo/ (дата общения 31.06.2018).

4. Huang J., Rathod V., Sun C., Zhu M., Korattikara A., Speed/accuracy trade–offs for modern convolutional object detectors / arXiv:1611.10012. (2017). [Электронный ресурс] URL: https://arxiv.org/pdf/1611.10012.pdf (дата обращения 7.07.2018).

5. R. Girshick. Fast R–Cnn / arXiv: 1504.08083 (2015). [Электронный ресурс] URL: https://arxiv.org/pdf/1504.08083.pdf (дата обращения 07.07.2018).

6. Nagaraj S., Muthiyan B., Ravi S., Menezes V., Kapoor K., Edge–Based Street Object Detection. [Электронный ресурс] URL: http://smart–city–sjsu.net/AICityChallenge/papers/NVIDIA_AI_City_Challenge_2017_paper_14.pdf (дата обращения 12. 07.2018).