Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ _____ **Информатика и системы управления**_____

КАФЕДРА _____ **Автоматизированные системы обработки информации и управления**_____

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ ПО ОБРАБОТКЕ И АНАЛИЗУ ДАННЫХ

НА ТЕМУ:

Обнаружение объектов с использованием алгоритма YOLO

Студент ____ИУ5И-32М____ _____ __ **Ахмад Жабер Али Диб** __
(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель курсового проекта _____ __ **Юрий Евгеньевич Гапанюк** __
(Подпись, дата) (И.О.Фамилия)

Консультант _____ _____
(Подпись, дата) (И.О.Фамилия)

2020 г.

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

УТВЕРЖДАЮ
Заведующий кафедрой ___ ИУ5___
(Индекс)
_____ __ В. М. Черненький __
(И.О.Фамилия)
« _____ » _____ 20 ____ г.

З А Д А Н И Е
на выполнение НИР по обработке и анализу данных

Студент группы _____ИУ5И-32м_____

_____Али Диб Ахмад Жабер_____
(Фамилия, имя, отчество)

Тема научно-исследовательской работы __ ***Обнаружение объектов с использованием*** _____
_____ ***алгоритма YOLO*** _____
_____

Направленность КП (учебный, исследовательский, практический, производственный, др.)
_____ **исследовательский** _____
Источник тематики (кафедра, предприятие, НИР) _____

График выполнения проекта: 25% к ___ нед., 50% к ___ нед., 75% к __ нед., 100% к ___ нед.

Задание__ Использование алгоритма YOLO для построения нейронной сети, которая может классифицировать объекты и определять их местоположение на изображении. Тестирование построенной модели на тестовых изображениях и отображение результатов _____
_____

Оформление научно-исследовательской работы:

Расчетно-пояснительная записка на _____ листах формата А4.
Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.)
_____
_____
_____
_____

Дата выдачи задания « ___ » _____ 20__ г.

Руководитель курсового проекта _____ __ **Юрий Евгеньевич Гапанюк** __
(Подпись, дата) (И.О.Фамилия)

Студент _____ __ **Ахмад Жабер Али Диб** __
(Подпись, дата) (И.О.Фамилия)

Примечание: Задание оформляется в двух экземплярах: один выдается студенту, второй хранится на кафедре.

ОГЛАВЛЕНИЕ

Список рисунков

1- Введение

Обнаружение объекта является одной из классических проблем в компьютерном зрении и определяется как:

Определите, какие объекты находятся внутри данного изображения, а также где они находятся на изображении.

Обнаружение является более сложной проблемой, чем классификация, которая также может распознавать объекты, но не говорит нам точно, где находится объект на изображении, и не будет работать для изображений, которые содержат более одного объекта.

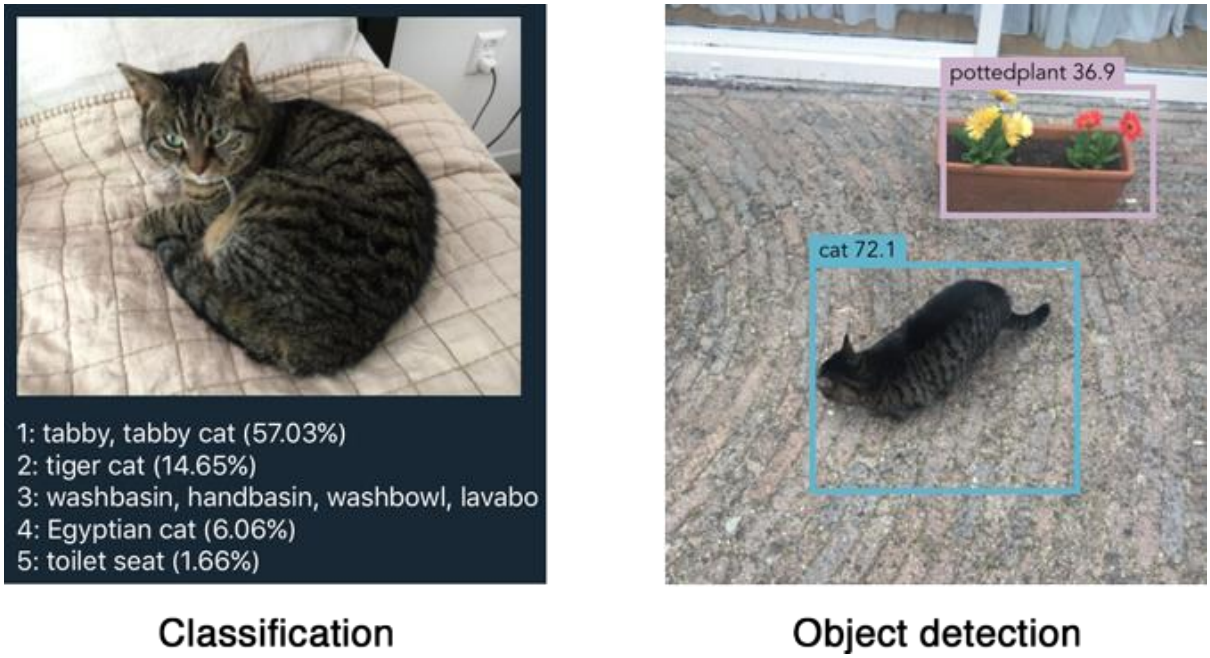


Рис 1. Comparison between classification and object detection.

Мы можем взять классификатор, такой как VGGNet или Inception, и превратить его в детектор объектов, сдвинув небольшое окно по изображению. На каждом шаге мы запускаем классификатор, чтобы получить прогноз того, какой объект находится внутри текущего окна, именно так работает DPM (модели деформируемых деталей). Использование скользящего окна дает несколько сотен или тысяч предсказаний для этого изображения, но мы оставляем только те, о которых классификатор наиболее уверен.

Этот подход работает, но он, очевидно, будет очень медленным, поскольку нам нужно многократно запускать классификатор.

Несколько более эффективный подход заключается в том, чтобы сначала предсказать, какие части изображения содержат интересную информацию - так называемые предложения по регионам, - а затем запустить классификатор только по этим регионам (семейство R-CNN). Классификатор должен выполнять меньше работы, чем со скользящими окнами, но он все равно многократно запускается.

YOLO использует совершенно другой подход. Это не традиционный классификатор, который используется как детектор объектов. YOLO на самом деле смотрит на изображение только один раз (отсюда и его название: « You Only Look Once »), но умным способом.

Традиционные детекторы объектов - это методы, основанные на классификаторе, где классификатор либо запускается на частях изображения в режиме скользящего окна, так работает DPM (модели деформируемых деталей), либо работает на предложениях

регионов, которые рассматриваются как потенциальные ограничивающие рамки, это относится к семейству R-CNN (R-CNN, Fast R-CNN и Faster R-CNN). Эти детекторы хорошо преформуются, особенно Faster R-CNN с непревзойденной точностью 73,2% mAP, однако из-за их сложного конвейера они работают плохо с точки зрения скорости - 7 кадров в секунду, что делает их неэффективными для обнаружения в реальном времени.

С точки зрения скорости, YOLO является одной из лучших моделей в распознавании объектов, способной распознавать объекты и обрабатывать кадры со скоростью до 67 кадров в секунду, и дает mAP 76,8%.

I. ***Детали модели YOLOv1:***

YOLO использует инновационную стратегию для решения проблемы обнаружения объектов как проблему регрессии, она определяет координаты ограничивающего прямоугольника и вероятности класса непосредственно из изображения.

YOLO делит входное изображение на сетку S × S. Если центр объекта попадает в ячейку сетки, эта ячейка сетки отвечает за обнаружение этого объекта (**каждая ячейка сетки предсказывает только один объект**). Каждая ячейка сетки предсказывает B ограничивающих рамок и оценки достоверности для этих блоков, как показано на рисунке 1. Эти оценки достоверности отражают, насколько модель уверена в том, что блок содержит объект Pr (Object), а также насколько точна прогнозируемая рамка путем оценки его перекрытие с ограничивающим прямоугольником, измеренным пересечением над объединением $\mathrm{IoU}_{\mathrm{pred}}^{\mathrm{truth}}$.

Формально мы определяем доверительную оценку как Pr (Объект) $*$ $\mathrm{IoU}_{\mathrm{pred}}^{\mathrm{truth}}$. Если в этой ячейке нет объекта, доверительные оценки должны быть равны нулю. В противном случае мы хотим, чтобы показатель достоверности равнялся пересечению по объединению (IOU) между предсказанной рамкой и основополагающей правдой.

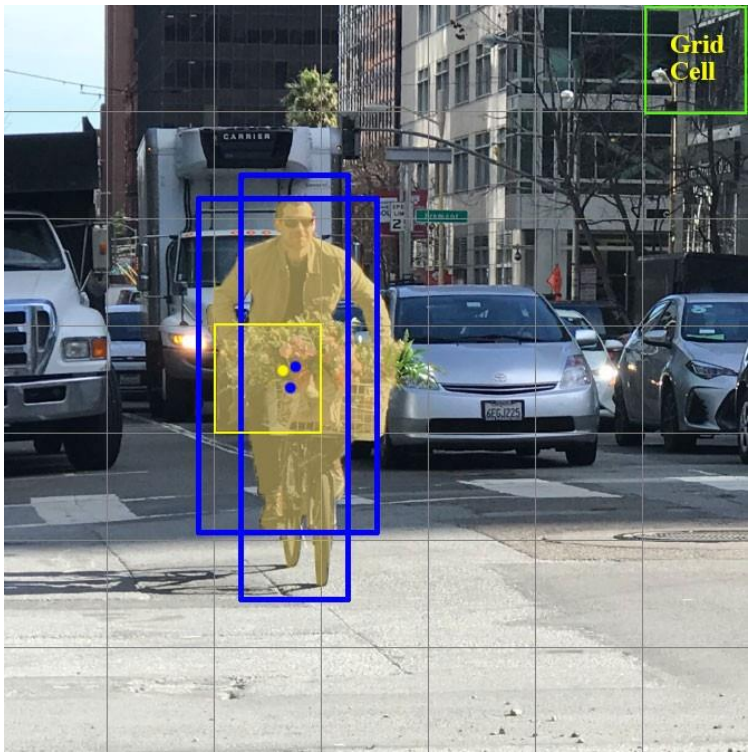


Рис 2. Желтая ячейка сетки делает два предсказания (синие прямоугольники), чтобы определить местонахождение человека.

Каждый ограничивающий прямоугольник состоит из 5 предсказаний: x, y, w, h и достоверность. Координаты (x, y) представляют центр прямоугольника относительно границ ячейки сетки. Ширина и высота прогнозируются относительно всего изображения. Наконец, доверительный прогноз представляет собой IoU между прогнозируемым блоком и любым базовым блоком истинности.
Каждая ячейка сетки также предсказывает C вероятности условного класса, Pr (Classi | Object). Эти вероятности обусловлены ячейкой сетки, содержащей объект. Мы прогнозируем только один набор вероятностей классов для каждой ячейки сетки, независимо от количества блоков B.
Итак, для каждой ячейки сетки:

- он предсказывает **B** граничных блоков, и каждый блок имеет один показатель достоверности.
- обнаруживает только **один** объект независимо от количества ящиков B.
- он предсказывает вероятности условного класса **C** (по одному на класс для вероятности класса объекта).

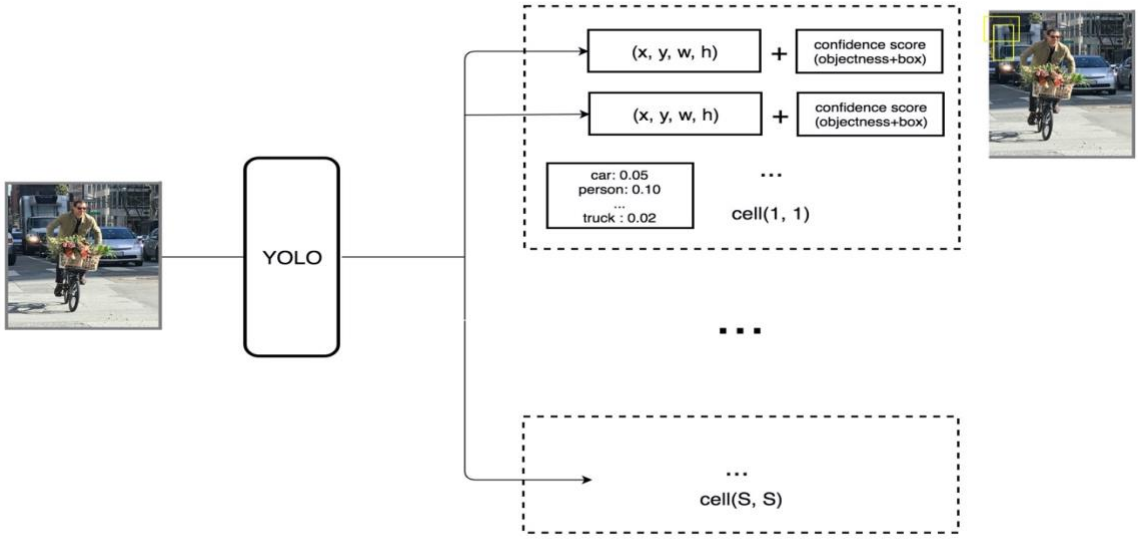


Рис 3. YOLO делает прогнозы SxS с помощью B граничных блоков.

Во время тестирования модель назначает каждому блоку свой показатель доверия для конкретного класса, умножая показатель доверия каждого блока на соответствующие условные вероятности класса, эти оценки оценивают вероятность появления класса в блоке и то, насколько точен блок. координаты (измеряет достоверность как по классификации, так и по локализации).

$$\text{Class confidence score} = \text{box confidence score X class-conditional probabilities}$$
$$= P_r(\text{Class}_i|\text{Object}) * P_r(\text{Object}) * \text{IoU}_{\text{pred}}^{\text{truth}} = P_r(\text{Class}_i) * \text{IoU}_{\text{pred}}^{\text{truth}}$$

Где:
Pr (Object) - это вероятность того, что поле содержит объект.
IoU - это IoU (пересечение делиться на объединение) между предсказанной рамкой и основополагающей истиной.
Pr (Class_i | Object) - вероятность того, что объект принадлежит к классу, если объект присутствует.
Pr (Class_i) - вероятность того, что объект принадлежит Classi.

Чтобы сделать окончательный прогноз, мы сохраняем людей с высокими показателями достоверности.

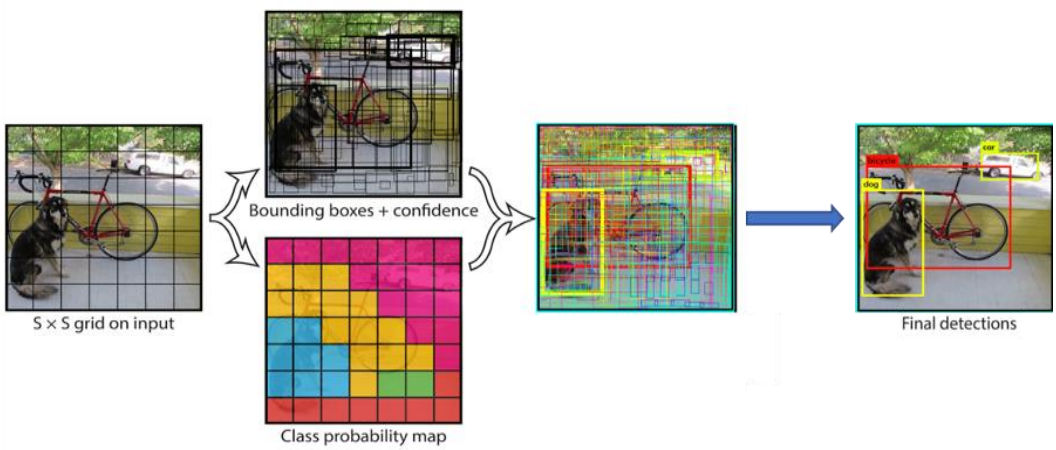


Рис 4. Прогнозы кодируются как тензор $S \times S \times (B * 5 + C)$.

2- Дизайн сети

GoogleLeNet сильно влияет на архитектуру YOLO, YOLO сеть состоит из 24 сверточных слоев для извлечения объектов, за которыми следуют 2 полностью соединенных слоя для прогнозирования координат ограничивающего прямоугольника и соответствующих им вероятностей объектов, заменяя начальные модули GoogleLeNet на сверточные слои 1x1, чтобы уменьшить измерение глубины карты объектов.
Конечный результат сети - тензор предсказаний $7 \times 7 \times 30$.

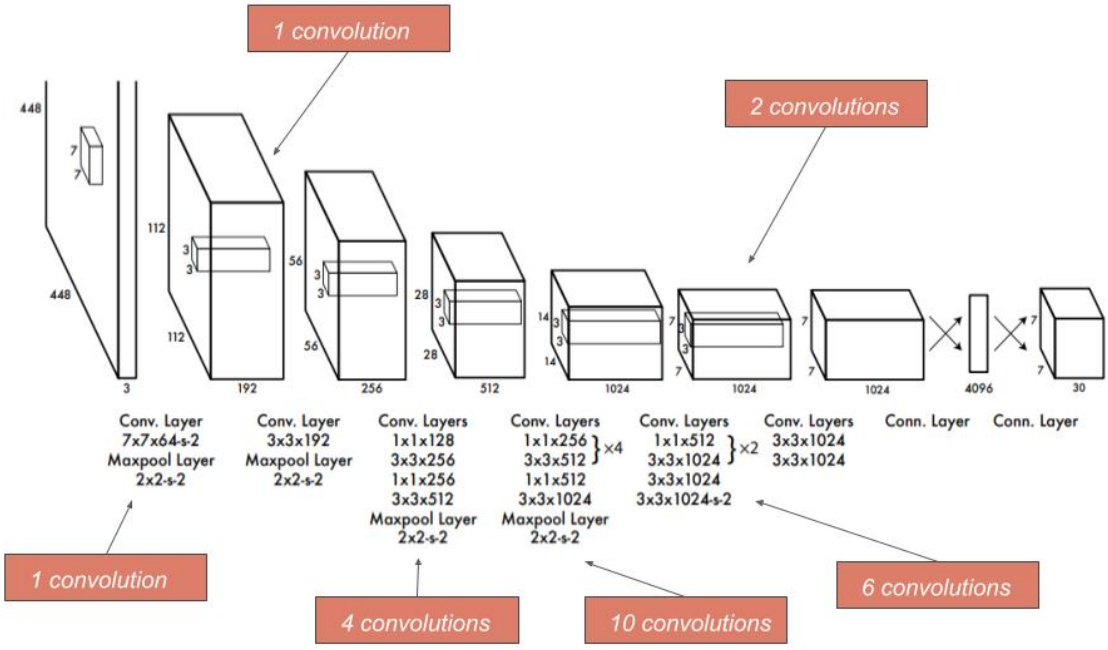


Рис 5. YOLO architecture.

Тренинг YOLO состоит из 2 этапов:

- Сначала сеть выполняет классификацию с входным разрешением 224x224 в ImageNet, используя только первые 20 слоев свертки, за которыми следует среднее pooling и полностью подключенный слой.

- Во-вторых, сеть обучается обнаружению, добавляя четыре сверточных слоя и два полностью связанных слоя. Для обнаружения мы обучаемся на входах с разрешением 448x448, поскольку постепенное обучение на изображениях с более высоким разрешением значительно повышает точность, учитывая, что тонкая визуальная информация более очевидна для обнаружения.

3- Функция потери

YOLO предсказывает несколько ограничивающих рамок на ячейку сетки. Чтобы вычислить потери для истинного позитива, мы хотим, чтобы только один из них отвечал за объект. Для этой цели мы выбираем ту, которая имеет наивысшее значение IoU (пересечение по объединению) с основной истиной. Эта стратегия приводит к специализации среди ограничивающих предсказаний. Каждый прогноз улучшается при прогнозировании определенных размеров и соотношений сторон.
YOLO использует сумму квадратов ошибок между прогнозами и истинной величиной для расчета потерь. Функция потерь состоит из:

- классификация потерь.
- потеря локализации (ошибки между предсказанным граничным квадратом и наземной истинностью).
- потеря доверия (объектность коробки).

Классификация потерь.

Для потери классификации YOLO использует ошибку суммы квадратов для сравнения условных вероятностей класса для всех классов, и для дальнейшего упрощения обучения мы хотим, чтобы функция потерь штрафовала ошибку классификации, только если объект присутствует в ячейке сетки, поэтому мы устанавливаем следующую маску $\mathbb{I}_{\mathrm{i}}^{\mathrm{obj}}$.

$$\mathbb{I}_{\mathrm{i}}^{\mathrm{obj}} = \begin{cases} 1 & if\ the\ object\ exists\ in\ the\ i^{th}\ cell \\ 0 & Otherwise \end{cases}$$

$$\sum_{i=0}^{S^2} \mathbb{I}_{\mathrm{i}}^{\mathrm{obj}} \sum_{c \in classes} (P_i(c) - \hat{P}_i(c))^2$$

Где:
$\mathbb{I}_{\mathrm{i}}^{\mathrm{obj}} = 1$ если объект появляется в ячейке i, иначе 0.
$\hat{P}_i(c)$ обозначает условную вероятность класса для класса c в ячейке i.

Потеря локализации.

Потеря локализации измеряет ошибки в прогнозируемых положениях и размерах границ.
функция потерь только штрафует ошибку локализации для блока предиктора, который имеет самый высокий IoU с блоком истинности земли в каждой ячейке сетки (блок, ответственный за обнаружение объекта).
Суммарная ошибка также одинаково взвешивает ошибки в больших коробках и маленьких коробках. Метрика ошибки должна отражать, что небольшие отклонения в больших ячейках имеют меньшее значение, чем в маленьких. Чтобы частично решить эту проблему, потеря связана с корнем квадратным из ширины и высоты ограничительной рамки вместо ширины и высоты напрямую.

Чтобы увеличить потери для координат ограничивающего прямоугольника и больше сосредоточиться на обнаружении, мы установили $\lambda_{\text{coord}} = 5$.

$$\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{I}_{\text{i}}^{\text{obj}} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2]$$

$$+\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{I}_{\text{i}}^{\text{obj}} [(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2]$$

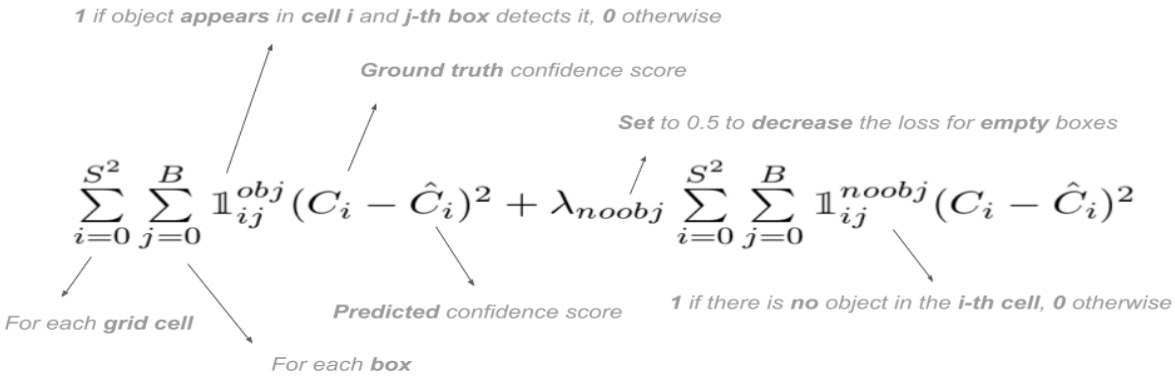
Где:

$\mathbb{I}_{\text{i}}^{\text{obj}} = 1$ если j^{th} граничный блок в ячейке i отвечает за обнаружение объекта, иначе 0.

λ_{coord} увеличьте вес потери в граничной координате.

Потеря уверенности.

Из-за парадигмы сетки YOLO сеть генерирует большой набор блоков, которые не содержат никаких объектов, поэтому эти блоки прогнозирования получают нулевой показатель достоверности. Обратите внимание, что эти пустые блоки перекрывают объекты, содержащие объекты, тем самым подавляя их потерю и градиент. Чтобы решить эту проблему, мы уменьшаем потерю прогнозов с низкой достоверностью, устанавливая $\lambda_{\text{noobj}} = 0{,}5$.



Следовательно, функция потерь YOLO складывает потери локализации, достоверности и классификации вместе и выражается следующим образом:

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$

$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2$$

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2$$

4- Non-maximum подавление

Учитывая большое количество предсказанных блоков, в которых несколько блоков обнаруживают один и тот же объект, YOLO выполняет не-максимальное подавление (NMS), чтобы сохранить только наилучшую ограничивающую рамку для объекта и эффективно отбросить остальные.

Вот как работает NMS:

- Во-первых, откажитесь от ящиков с низким показателем достоверности (то есть пустых ящиков) с порогом 0,6.

Для остальных, для каждого класса:

- Выберите коробку с наибольшим доверительным баллом (то есть коробку, которая больше всего соответствует объекту)
- Затем откажитесь от блоков с IoU выше назначенного порога, эти блоки в основном перекрываются с нашим лучшим блоком.

5- Основные особенности YOLO

- Предсказывая координаты ограничивающего прямоугольника непосредственно из входных изображений, YOLO рассматривает обнаружение объекта как проблему регрессии, в отличие от методов на основе классификатора.
- Простая и освежающая сеть YOLO обучается совместно, позволяя прогнозировать скорость в режиме реального времени со скоростью 45 кадров в секунду.
- Более того, поскольку он одновременно предсказывает ограничивающие рамки из всего изображения для всех классов, сеть глобально рассуждает обо всех объектах в изображении.

6- Ограничения YOLOv1

- YOLO плохо работает с небольшими объектами, особенно с теми, которые появляются в группах, поскольку каждая ячейка сетки предсказывает только B=2 ограничивающие рамки одного и того же класса, создавая пространственное ограничение, ограничивающее число объектов, прогнозируемых на ячейку сетки.
- Несмотря на то, что функция потерь предсказывает квадратный корень из высоты и веса ограничивающего прямоугольника вместо роста и веса непосредственно в попытке решить уравнение ошибок для больших и маленьких прямоугольников, это решение частично устраняет проблему, приводя к значительному количеству ошибок локализации.
- Кроме того, YOLO не может должным образом обнаруживать объекты новых или необычных форм, поскольку он не обобщается далеко за пределы ограничительных рамок в обучающем наборе.

II. YOLOv2

Поскольку YOLOv1 страдает от ошибок локализации и низкого прогноза отзыва, YOLOv2 демонстрирует значительные улучшения, чтобы увеличить компромисс между скоростью и точностью.

Корректировки для лучшего YOLO

- **Batch нормализация**
  При добавлении нормализации партии на всех сверточных слоях в YOLO производительность улучшается на 2% MAP, норма партии имеет регуляризующий эффект, поэтому мы можем удалить выпадение без переобучения.

- **Классификация на входах высокого разрешения**
  YOLO предварительно обучает классификатор в ImageNet, используя входы разрешения 224x224, затем увеличивает разрешение до 448x448 для обнаружения, требует, чтобы сетевое время одновременно адаптировалось к входу нового разрешения и выполняло задачу обнаружения. Однако YOLOv2 разрешает это следующим образом:
  - Сначала обучите классификатор разрешению 224x224 в ImageNet.
  - Во-вторых, настройте классификатор на разрешение 448x448 для 10 эпох, это заставит сетевые фильтры настраиваться на входы с более высоким разрешением.
  - Затем выполните точную настройку сети при обнаружении.

  Точная настройка классификатора на входах с более высоким разрешением повышает точность на 4% MAP.

- **Сверточный с якорными ящиками**
  Якорные блоки (также называемые блоками по умолчанию) представляют собой набор предварительно определенных форм блоков, выбранных для соответствия ограничивающим прямоугольникам наземной истинности, поскольку большинство объектов в наборе обучающих данных или вообще в мире (например, человек, велосипед и т. Д.) Имеют типичную высоту и соотношение ширины Поэтому при прогнозировании ограничивающих рамок мы просто корректируем и уточняем размер этих якорных рамок, чтобы соответствовать объектам, отсюда и слово «смещение».
  Как указано в статье YOLO, ранняя тренировка подвержена нестабильным градиентам. Первоначально YOLO делает произвольные предположения о граничных прямоугольниках. Эти предположения могут хорошо работать для некоторых объектов, но плохо для других, что приводит к резким изменениям градиента. На ранних этапах обучения, предсказания сражаются друг с другом в зависимости от того, на каких формах специализироваться.
  В реальной области границы не являются произвольными. Автомобили имеют очень похожие формы, а у пешеходов приблизительное соотношение сторон составляет 0,41.
  Поскольку нам нужно только одно предположение, чтобы быть правым, начальное обучение будет более стабильным, если мы начнем с разных догадок, которые являются общими для реальных объектов.

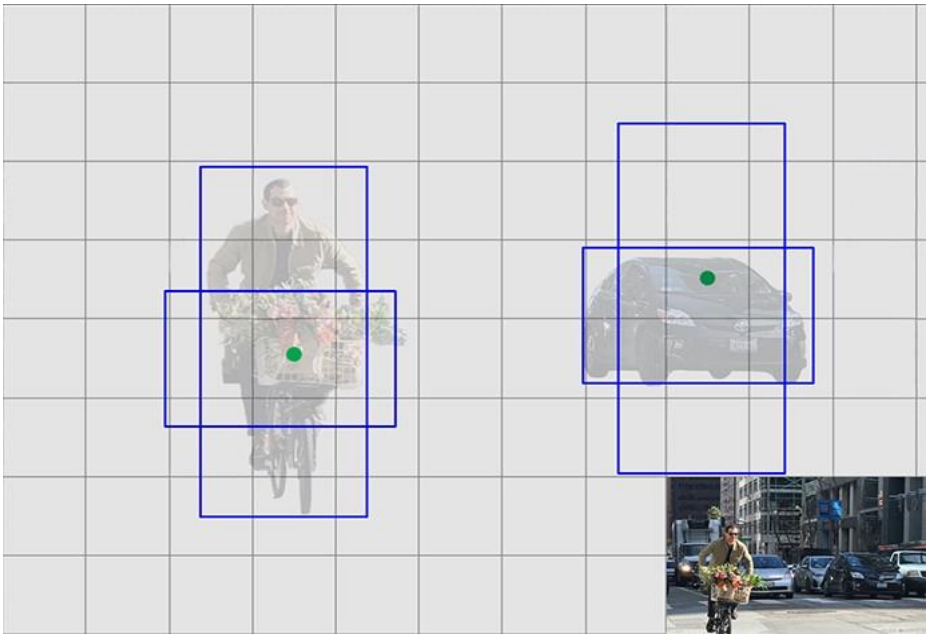


Рис 6. Illustration of anchor boxes.

изменения, которые мы вносим в сеть:

- Удалите полностью связанные слои, отвечающие за прогнозирование граничной рамки.

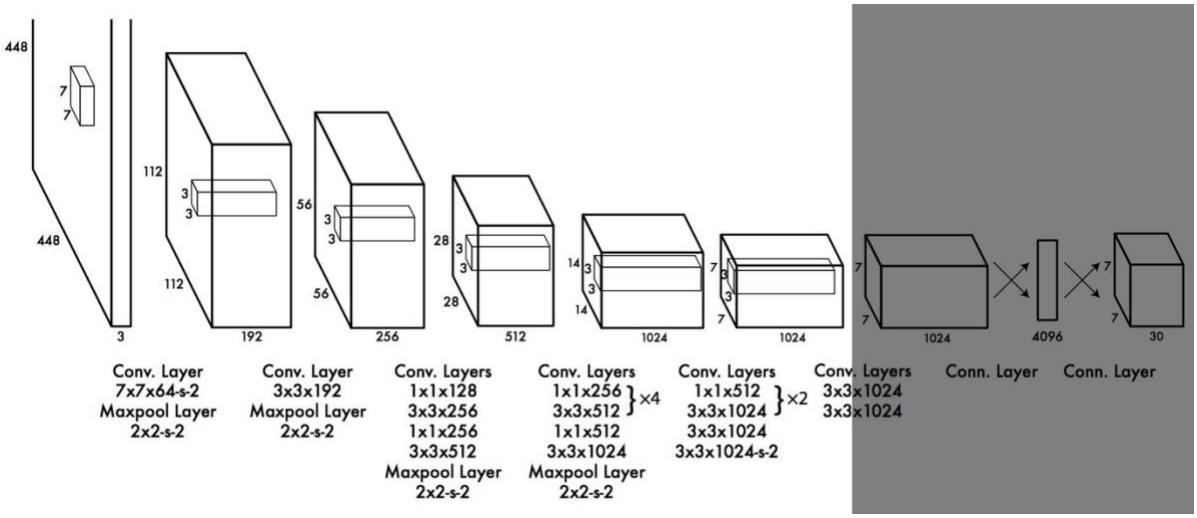


Рис 7. YOLOv2 architecture.

- Мы перемещаем прогнозирование класса с уровня ячейки на уровень граничного блока. Теперь каждое предсказание включает в себя 4 параметра для граничного блока, 1 показатель достоверности (объектность) и 20 вероятностей классов. то есть 5 граничных блоков с 25 параметрами: 125 параметров на ячейку сетки. Так же, как и YOLO, предсказание объектности все еще предсказывает IOU основной истины и предложенную рамку.

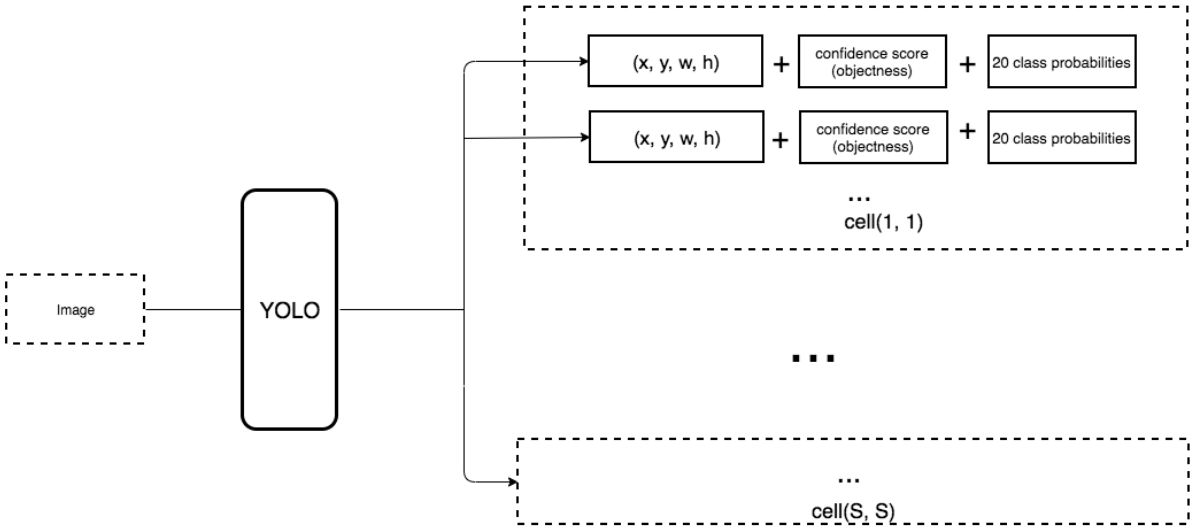
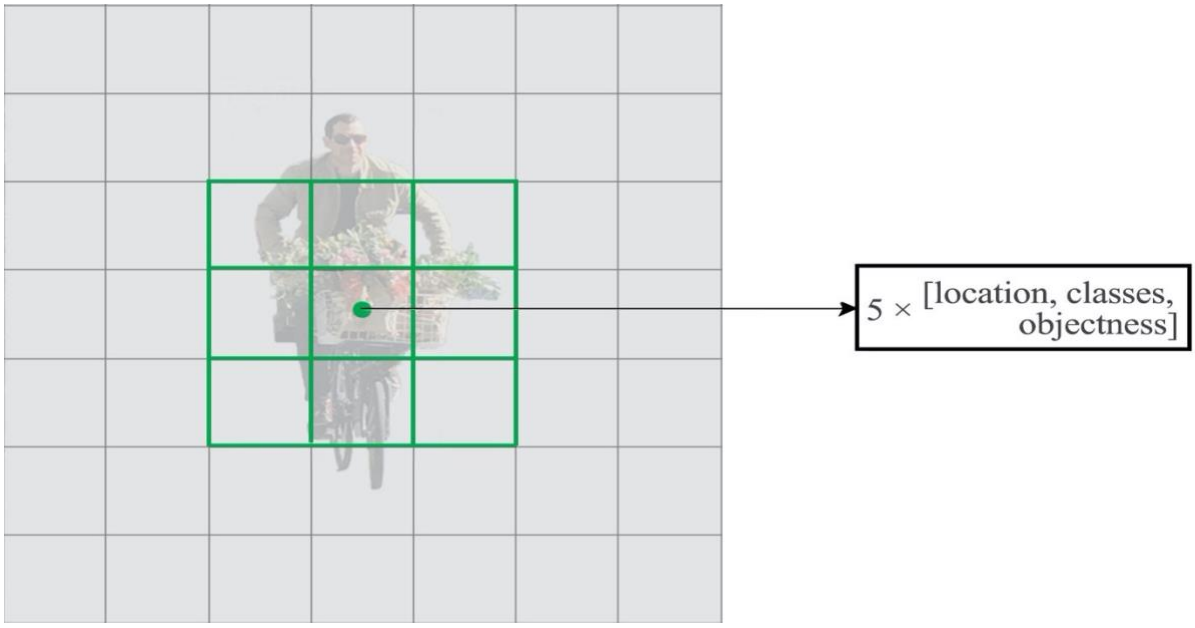
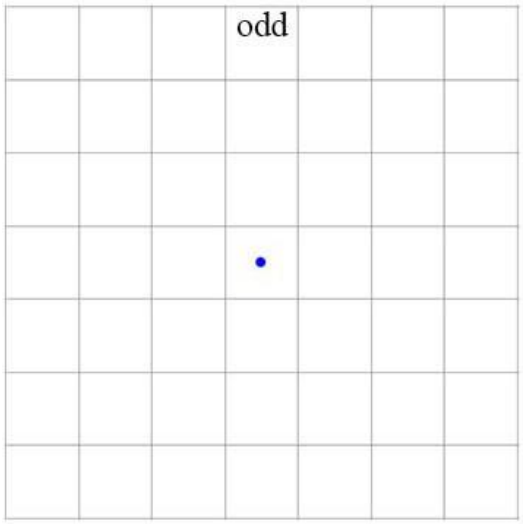


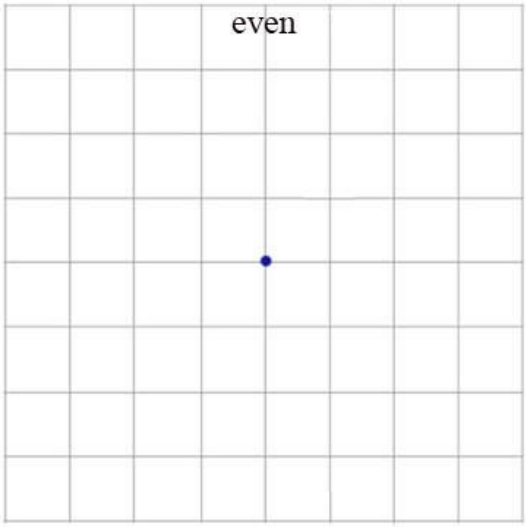
Рис 8. YOLO делает прогнозы SxS с помощью B граничных блоков.

- Чтобы сгенерировать предсказания в форме 7 × 7 × 125, мы заменяем последний сверточный слой тремя сверточными слоями 3 × 3, каждый из которых выводит 1024 выходных канала. Затем мы применяем конечный сверточный слой 1 × 1, чтобы преобразовать выходные данные 7 × 7 × 1024 в 7 × 7 × 125.



- Измените размер входного изображения с 448 × 448 на 416 × 416. Это создаст пространственное измерение нечетного числа (7 × 7 против 8 × 8 ячейки сетки). Центр картины часто занят крупным объектом. С ячейкой сетки нечетных чисел, это более определенно, где принадлежит объект.





- Удалите один слой пула, чтобы сделать пространственный вывод сети 13 × 13 (вместо 7 × 7).

Якорные ящики немного уменьшают mAP с 69,5 до 69,2, но отзыв улучшается с 81% до 88%. даже точность немного снижается, но это увеличивает шансы обнаружения всех наземных объектов правды.

- **Прямой прогноз местоположения**

Мы делаем прогнозы относительно смещений к якорям. Тем не менее, если он не ограничен, наши догадки снова будут рандомизированы. YOLO прогнозирует 5 параметров (tx, ty, tw, th, и to) и применяет сигма-функцию для ограничения ее возможного диапазона смещения.

$$b_x = \sigma(t_x) + c_x$$
$$b_y = \sigma(t_y) + c_y$$
$$b_w = p_w e^{t_w}$$
$$b_h = p_h e^{t_h}$$
$$P_r(object) * IoU(b, object) = \sigma(t_o)$$

Где:
tx, ty, tw, th - это прогнозы, сделанные YOLO.
cx, cy - верхний левый угол ячейки сетки якоря.
pw, ph - ширина и высота якоря.
cx, cy, pw, ph нормализуются по ширине и высоте изображения.
bx, by, bw, bh - предсказанные граничные рамки.
σ(to) - показатель достоверности коробки.

На приведенном ниже рисунке показаны ограничивающие прямоугольники с априорными размерами и местоположением, где синее поле ниже является предсказанным ограничивающим прямоугольником, а пунктирный прямоугольник - якорем.

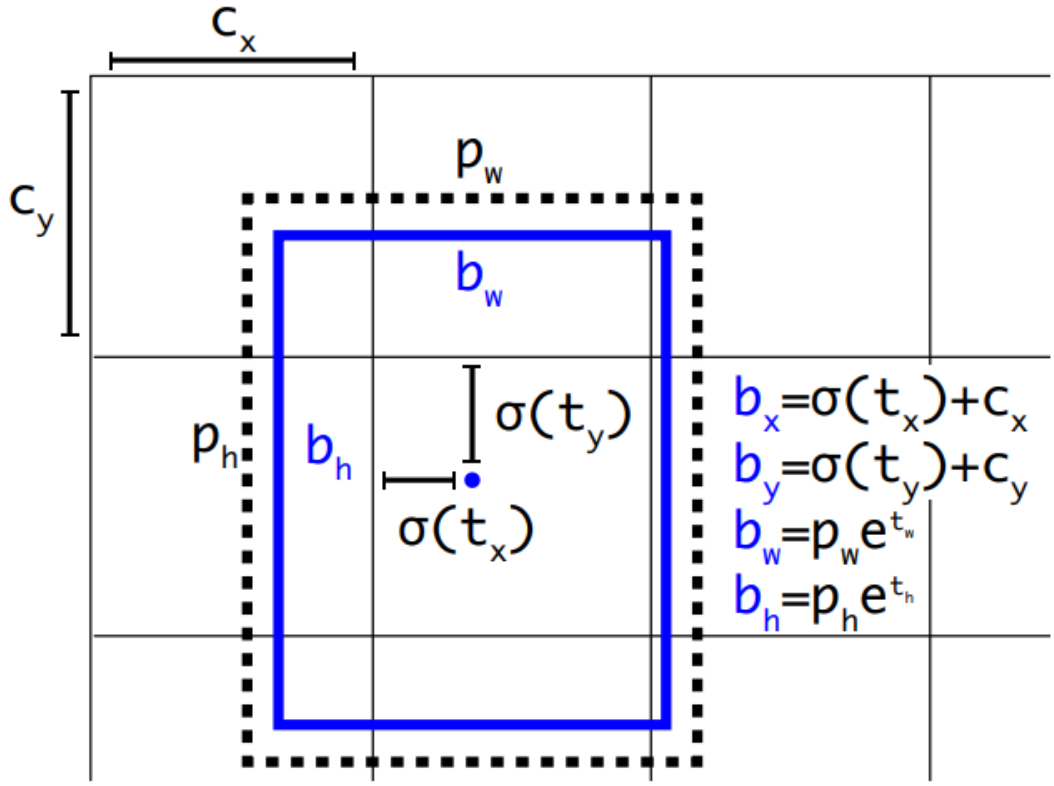


Рис 9. Ограничительные рамки с размерными приорами и расположением.

	YOLO							YOLOv2
batch norm?		✓	✓	✓	✓	✓	✓	✓	✓
hi-res classifier?			✓	✓	✓	✓	✓	✓	✓
convolutional?				✓	✓	✓	✓	✓	✓
anchor boxes?				✓	✓				
new network?					✓	✓	✓	✓	✓
dimension priors?						✓	✓	✓	✓
location prediction?						✓	✓	✓	✓
passthrough?							✓	✓	✓
multi-scale?								✓	✓
hi-res detector?									✓
VOC2007 mAP	63.4	65.8	69.5	69.2	69.6	74.4	75.4	76.8	**78.6**

Рис 10. Точность после применения методов улучшения.

7- Модельное обучение

Для задачи классификации модель обучается на задаче классификации ImageNet-1000 для 160 эпох с начальной скоростью обучения 0,1, снижением веса 0,0005 и импульсом 0,9 с использованием архитектуры Darknet-19. Для этого тренинга применяются некоторые стандартные методы дополнения данных.

Для обнаружения есть некоторые модификации, сделанные в архитектуре Darknet-19, которые мы обсуждали выше. Модель обучается в течение 160 эпох с начальной скоростью обучения 10-3, снижением веса 0,0005 и импульсом 0,9. Та же самая стратегия использовалась для обучения модели на COCO и VOC.

Мы будем тренировать модель с архитектурой, показанной на рисунке 11, а затем отобразим оценку функции потерь в процессе обучения, как показано на рисунке 12.

```
[ ]
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 608, 608, 3)       0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 608, 608, 32)      864
_________________________________________________________________
batch_normalization_1 (BatchNor (None, 608, 608, 32)      128
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 608, 608, 32)      0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2D)  (None, 304, 304, 32)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 304, 304, 64)      18432
_________________________________________________________________
batch_normalization_2 (BatchNor (None, 304, 304, 64)      256
_________________________________________________________________
leaky_re_lu_2 (LeakyReLU)    (None, 304, 304, 64)      0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2D)  (None, 152, 152, 64)      0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 152, 152, 128)     73728
_________________________________________________________________
batch_normalization_3 (BatchNor (None, 152, 152, 128)     512
_________________________________________________________________
leaky_re_lu_3 (LeakyReLU)    (None, 152, 152, 128)     0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 152, 152, 64)      8192
_________________________________________________________________
batch_normalization_4 (BatchNor (None, 152, 152, 64)      256
_________________________________________________________________
leaky_re_lu_4 (LeakyReLU)    (None, 152, 152, 64)      0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 152, 152, 128)     73728
_________________________________________________________________
batch_normalization_5 (BatchNor (None, 152, 152, 128)     512
_________________________________________________________________
leaky_re_lu_5 (LeakyReLU)    (None, 152, 152, 128)     0
_________________________________________________________________
max_pooling2d_3 (MaxPooling2D)  (None, 76, 76, 128)       0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 76, 76, 256)       294912
_________________________________________________________________
batch_normalization_6 (BatchNor (None, 76, 76, 256)       1024
_________________________________________________________________
leaky_re_lu_6 (LeakyReLU)    (None, 76, 76, 256)       0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 76, 76, 128)       32768
```

Рис 11. Модельное обучение

8- Реализация

```
[4] def yolo_filter_boxes(box_confidence, boxes, box_class_probs, threshold = .6):
        """Filters YOLO boxes by thresholding on object and class confidence.

        Arguments:
        box_confidence -- tensor of shape (19, 19, 5, 1)
        boxes -- tensor of shape (19, 19, 5, 4)
        box_class_probs -- tensor of shape (19, 19, 5, 80)
        threshold -- real value, if [ highest class probability score < threshold], then get rid of the corresponding box

        Returns:
        scores -- tensor of shape (None,), containing the class probability score for selected boxes
        boxes -- tensor of shape (None, 4), containing (b_x, b_y, b_h, b_w) coordinates of selected boxes
        classes -- tensor of shape (None,), containing the index of the class detected by the selected boxes

        Note: "None" is here because you don't know the exact number of selected boxes, as it depends on the threshold.
        For example, the actual output size of scores would be (10,) if there are 10 boxes.
        """

        # Step 1: Compute box scores
        box_scores = box_confidence*box_class_probs   # tensor of shape (19, 19, 5, 80)

        # Step 2: Find the box_classes using the max box_scores, keep track of the corresponding score
        box_classes = K.argmax(box_scores,axis=-1)   # tensor of shape (19, 19, 5, 1)
        box_class_scores = K.max(box_scores,axis=-1)    # tensor of shape (19, 19, 5, 1)

        # Step 3: Create a filtering mask based on "box_class_scores" by using "threshold". The mask should have the
        # same dimension as box_class_scores, and be True for the boxes you want to keep (with probability >= threshold)
        filtering_mask = (box_class_scores>threshold)    # tensor of shape (19, 19, 5, 1)

        # Step 4: Apply the mask to box_class_scores, boxes and box_classes
        scores = tf.boolean_mask(box_class_scores,filtering_mask)
        boxes = tf.boolean_mask(boxes,filtering_mask)
        classes = tf.boolean_mask(box_classes,filtering_mask)

        return scores, boxes, classes
```

```
[5] def yolo_non_max_suppression(scores, boxes, classes, max_boxes = 10, iou_threshold = 0.5):
        """
        Applies Non-max suppression (NMS) to set of boxes

        Arguments:
        scores -- tensor of shape (None,), output of yolo_filter_boxes()
        boxes -- tensor of shape (None, 4), output of yolo_filter_boxes() that have been scaled to the image size (see later)
        classes -- tensor of shape (None,), output of yolo_filter_boxes()
        max_boxes -- integer, maximum number of predicted boxes you'd like
        iou_threshold -- real value, "intersection over union" threshold used for NMS filtering

        Returns:
        scores -- tensor of shape (, None), predicted score for each box
        boxes -- tensor of shape (4, None), predicted box coordinates
        classes -- tensor of shape (, None), predicted class for each box

        Note: The "None" dimension of the output tensors has obviously to be less than max_boxes. Note also that this
        function will transpose the shapes of scores, boxes, classes. This is made for convenience.
        """

        max_boxes_tensor = K.variable(max_boxes, dtype='int32')     # tensor to be used in tf.image.non_max_suppression()
        K.get_session().run(tf.variables_initializer([max_boxes_tensor])) # initialize variable max_boxes_tensor

        # Use tf.image.non_max_suppression() to get the list of indices corresponding to boxes you keep
        nms_indices = tf.image.non_max_suppression(boxes,scores,max_boxes_tensor,iou_threshold)

        # Use K.gather() to select only nms_indices from scores, boxes and classes
        scores = K.gather(scores,nms_indices)
        boxes = K.gather(boxes,nms_indices)
        classes = K.gather(classes,nms_indices)

        return scores, boxes, classes
```

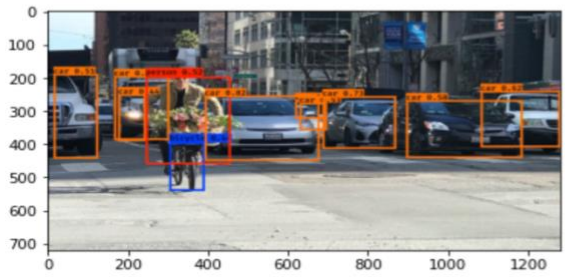
```
[6] def yolo_eval(yolo_outputs, image_shape = (720., 1280.), max_boxes=25, score_threshold=.3, iou_threshold=.5):
        """
        Converts the output of YOLO encoding (a lot of boxes) to your predicted boxes along with their scores, box coordinates and classes.

        Arguments:
        yolo_outputs -- output of the encoding model (for image_shape of (608, 608, 3)), contains 4 tensors:
                        box_confidence: tensor of shape (None, 19, 19, 5, 1)
                        box_xy: tensor of shape (None, 19, 19, 5, 2)
                        box_wh: tensor of shape (None, 19, 19, 5, 2)
                        box_class_probs: tensor of shape (None, 19, 19, 5, 80)
        image_shape -- tensor of shape (2,) containing the input shape, in this notebook we use (608., 608.) (has to be float32 dtype)
        max_boxes -- integer, maximum number of predicted boxes you'd like
        score_threshold -- real value, if [ highest class probability score < threshold], then get rid of the corresponding box
        iou_threshold -- real value, "intersection over union" threshold used for NMS filtering

        Returns:
        scores -- tensor of shape (None, ), predicted score for each box
        boxes -- tensor of shape (None, 4), predicted box coordinates
        classes -- tensor of shape (None,), predicted class for each box
        """


        # Retrieve outputs of the YOLO model (≈1 line)
        box_confidence, box_xy, box_wh, box_class_probs = yolo_outputs

        # Convert boxes to be ready for filtering functions (convert boxes box_xy and box_wh to corner coordinates)
        boxes = yolo_boxes_to_corners(box_xy, box_wh)

        # Use one of the functions you've implemented to perform Score-filtering with a threshold of score_threshold (≈1 line)
        scores, boxes, classes = yolo_filter_boxes(box_confidence, boxes, box_class_probs, threshold = score_threshold)

        # Scale boxes back to original image shape.
        boxes = scale_boxes(boxes, image_shape)

        # Use one of the functions you've implemented to perform Non-max suppression with
        # maximum number of boxes set to max_boxes and a threshold of iou_threshold (≈1 line)
        scores, boxes, classes = yolo_non_max_suppression(scores, boxes, classes, max_boxes = max_boxes,iou_threshold=iou_threshold)


        return scores, boxes, classes
```

```
[13] import imageio
```

```
[14] def predict(sess, image_file):
         """
         Runs the graph stored in "sess" to predict boxes for "image_file". Prints and plots the predictions.

         Arguments:
         sess -- your tensorflow/Keras session containing the YOLO graph
         image_file -- name of an image stored in the "images" folder.

         Returns:
         out_scores -- tensor of shape (None, ), scores of the predicted boxes
         out_boxes -- tensor of shape (None, 4), coordinates of the predicted boxes
         out_classes -- tensor of shape (None, ), class index of the predicted boxes

         Note: "None" actually represents the number of predicted boxes, it varies between 0 and max_boxes.
         """

         # Preprocess your image
         image, image_data = preprocess_image("/content/gdrive/My Drive/yolo/images/" + image_file, model_image_size = (608, 608))

         # Run the session with the correct tensors and choose the correct placeholders in the feed_dict.
         # You'll need to use feed_dict={yolo_model.input: ... , K.learning_phase(): 0})
         out_scores, out_boxes, out_classes = sess.run(fetches=[scores,boxes,classes],
            feed_dict={yolo_model.input: image_data,
                       K.learning_phase():0
            })

         # Print predictions info
         print('Found {0} boxes for {1}'.format(len(out_boxes), image_file))
         # Generate colors for drawing bounding boxes.
         colors = generate_colors(class_names)
         # Draw bounding boxes on the image file
         draw_boxes(image, out_scores, out_boxes, out_classes, class_names, colors)
         # Save the predicted bounding box on the image
         image.save(os.path.join("/content/gdrive/My Drive/yolo/out", image_file), quality=90)
         # Display the results in the notebook
         output_image = imageio.imread(os.path.join("/content/gdrive/My Drive/yolo/out", image_file))
         #output_image = scipy.misc.imread(os.path.join("/content/gdrive/My Drive/yolo/out", image_file))
         imshow(output_image)

         return out_scores, out_boxes, out_classes
```

```
[15] out_scores, out_boxes, out_classes = predict(sess, "test1.jpeg")
```

```
Found 11 boxes for test1.jpeg
car 0.33 (618, 267) (695, 324)
car 0.33 (166, 195) (304, 388)
car 0.44 (176, 250) (245, 391)
bicycle 0.44 (304, 395) (392, 541)
car 0.51 (15, 189) (126, 444)
person 0.52 (245, 197) (457, 461)
car 0.53 (629, 276) (697, 359)
car 0.58 (895, 268) (1186, 442)
car 0.62 (1081, 240) (1280, 410)
car 0.73 (686, 256) (870, 415)
car 0.82 (392, 254) (679, 451)
```




```
[16] out_scores, out_boxes, out_classes = predict(sess, "cartest3.jpg")
```

```
Found 1 boxes for cartest3.jpg
car 0.90 (393, 278) (904, 477)
```


```
[17] out_scores, out_boxes, out_classes = predict(sess, "testcar1.jpeg")
```

```
Found 25 boxes for testcar1.jpeg
car 0.58 (947, 137) (1031, 186)
car 0.59 (102, 132) (183, 207)
car 0.59 (369, 126) (461, 204)
car 0.61 (940, 329) (1065, 406)
car 0.61 (961, 241) (1070, 320)
car 0.61 (1041, 323) (1164, 415)
car 0.62 (333, 276) (433, 348)
car 0.63 (719, 342) (844, 430)
car 0.64 (98, 107) (167, 163)
car 0.64 (21, 10) (897, 403)
car 0.64 (845, 251) (945, 326)
car 0.65 (456, 138) (549, 202)
car 0.67 (175, 127) (270, 210)
car 0.69 (1093, 133) (1186, 184)
car 0.69 (779, 139) (871, 194)
car 0.71 (817, 323) (948, 410)
car 0.72 (757, 247) (853, 328)
car 0.73 (101, 349) (209, 460)
car 0.74 (556, 255) (644, 333)
car 0.76 (482, 341) (608, 448)
car 0.76 (649, 253) (752, 337)
car 0.76 (610, 344) (723, 439)
car 0.77 (228, 350) (329, 446)
car 0.79 (434, 276) (525, 338)
car 0.80 (356, 353) (469, 451)
```

9- Результаты

Мы говорили о ключевых концепциях, которые делают алгоритм локализации объекта YOLO быстрым и точным.
Наконец, я хочу поднять 2 ограничения алгоритма YOLO:

- Поскольку для каждой ячейки сетки существуют ограниченные якорные ячейки, YOLO испытывает трудности при обнаружении групп небольших объектов, таких как стая птиц или группа людей.

- Выходные данные модели были обучены распознавать различные объекты в форме якорного ящика. Объекты со странным соотношением сторон будет трудно обнаружить.

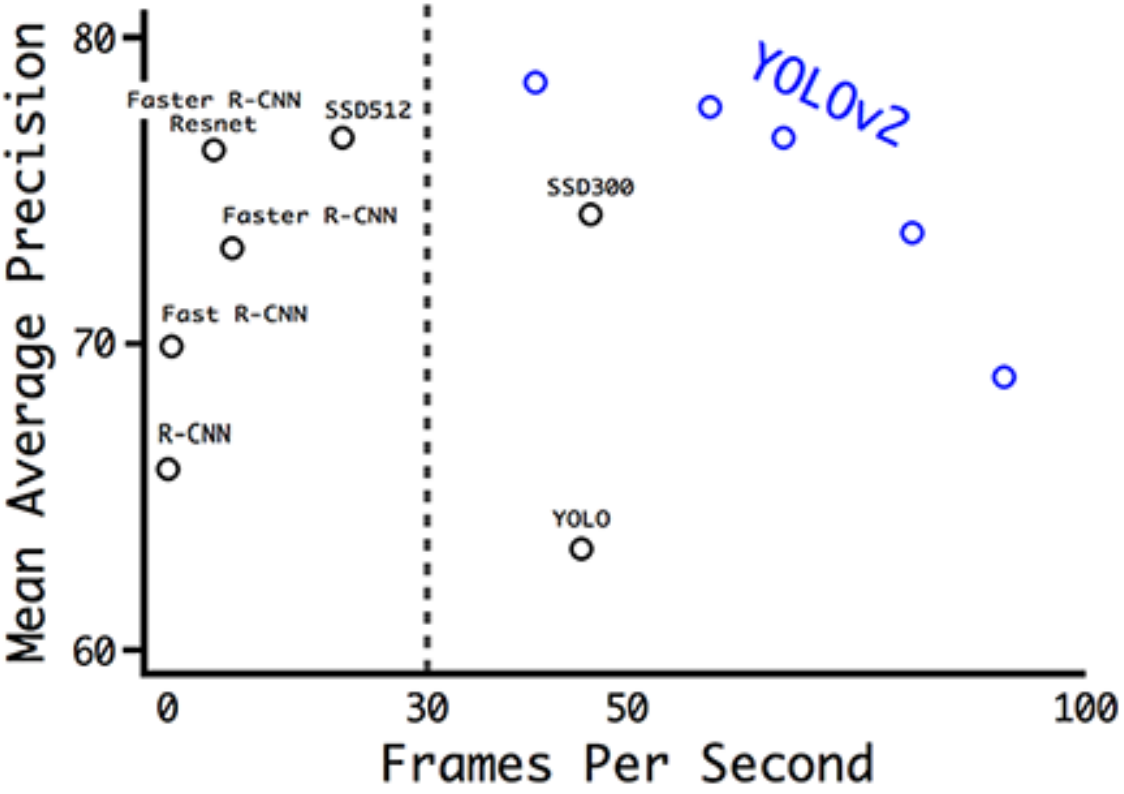


Рис 12. Accuracy comparison for different detectors.

10- Выводы

- YOLOv2 работает быстрее, чем другие алгоритмы обнаружения в разных наборах данных обнаружения. Он способен работать с изображениями различного размера и может обеспечить плавный компромисс между точностью и скоростью обработки в реальном времени.
- YOLOv2 - это среда в реальном времени для обнаружения более 9000 категорий объектов путем совместной оптимизации обнаружения и классификации.

11- Ссылка на Github

https://github.com/ahmadalideeb/YOLOv2-Object-Detection/blob/main/YOLO2.ipynb

12- Список литераторы

1- Joseph Redmon, Santosh Divvala, Ross Girshick, & Ali Farhadi. (2015). You Only Look Once: Unified, Real-Time Object Detection. https://arxiv.org/pdf/1506.02640.pdf

2- Joseph Redmon, & Ali Farhadi. (2016). YOLO9000: Better, Faster, Stronger. https://arxiv.org/pdf/1612.08242.pdf

3- Coursera — Convolutional Neural Network

4- https://medium.com/@jonathan_hui/real-time-object-detection-with-yolo-yolov2-28b1b93e2088

5- https://manalelaidouni.github.io/manalelaidouni.github.io/Understanding%20YOLO%20and%20YOLOv2.html