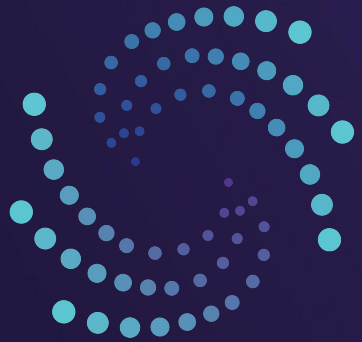
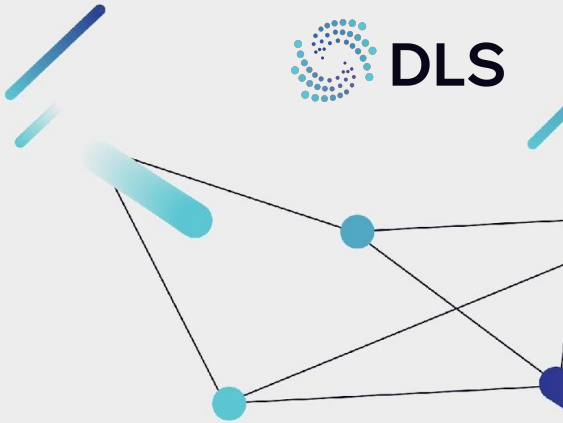


Deep
Learning
School

Задача детекции.
Обзор моделей. Часть1.

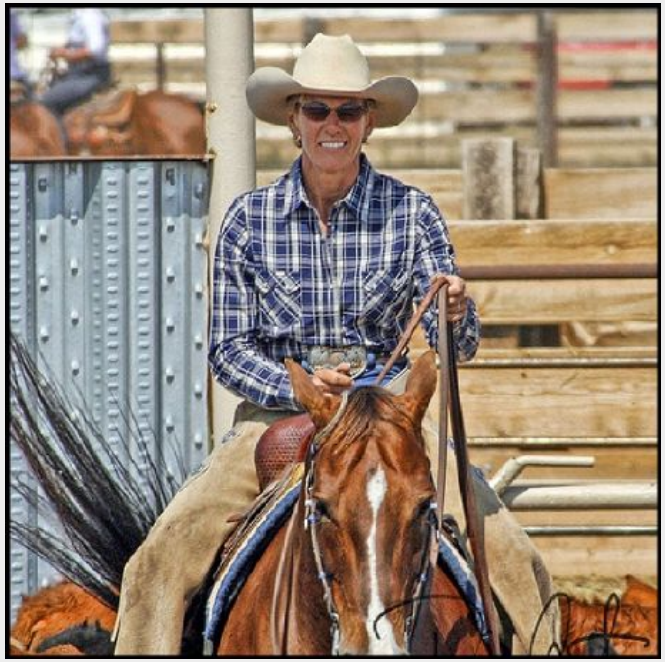
План

1. Модели, с которых началась детекция:
   1.1. R-CNN,
   1.2. Fast R-CNN,
   1.3. Faster R-CNN
   1.4. Mask R-CNN,
   1.5. Cascade R-CNN
2. YOLO
3. Single shot detector (SSD)
4. Feature Pyramid Network (FPN)
5. RetinaNet
6. Fully convolutional One-Stage (FCOS)
7. Path Aggregation Network (PAN)
8. EfficientDet



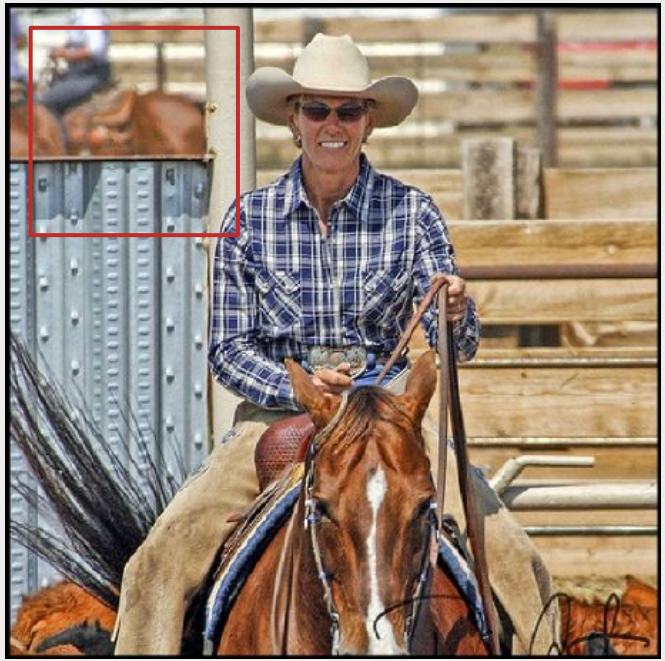
Наивный подход к детекции

Допустим, у нас есть следующее изображение:



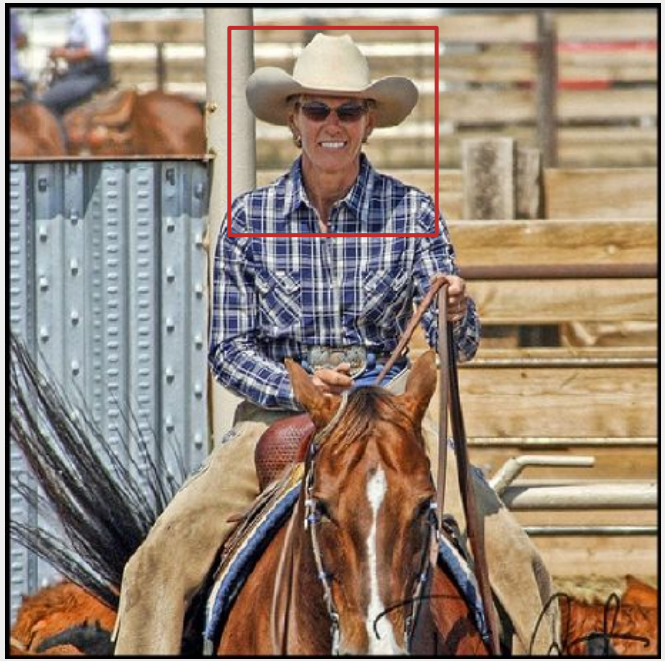
Наивный подход к детекции

Можем пройтись по нему заданным окном NxN и искать объект внутри окна:



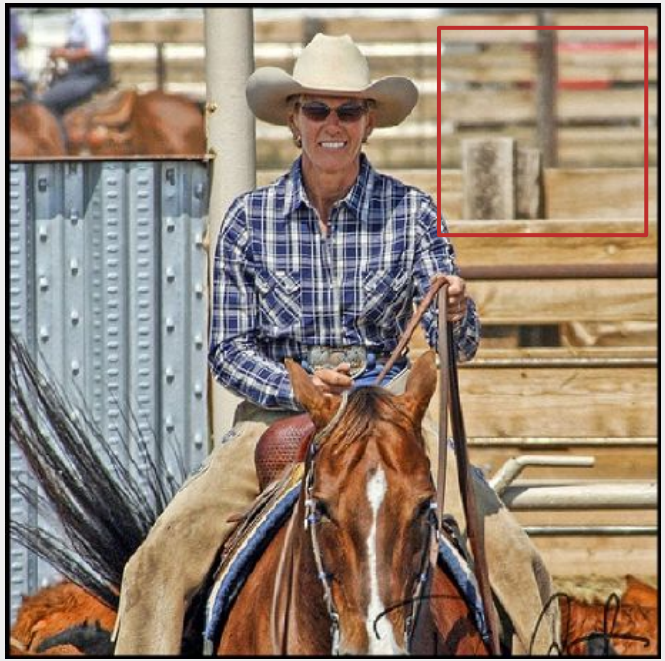
Наивный подход к детекции

Можем пройтись по нему заданным окном NxN и искать объект внутри окна:



Наивный подход к детекции

Можем пройтись по нему заданным окном NxN и искать объект внутри окна:



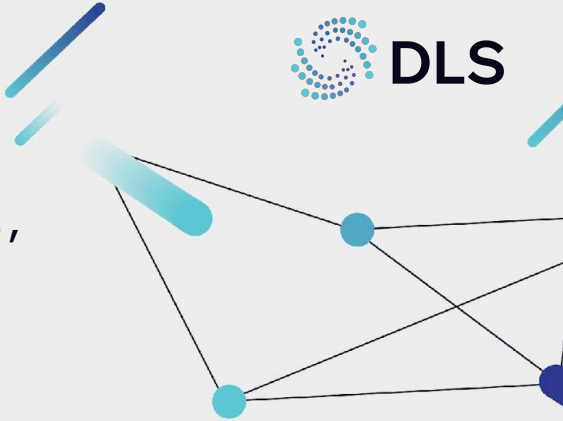
Наивный подход к детекции

Очевидные минусы подхода:

1. Это долго,
2. Окно фиксированного размера не сможет обнаружить все объекты.

Как сделать лучше?

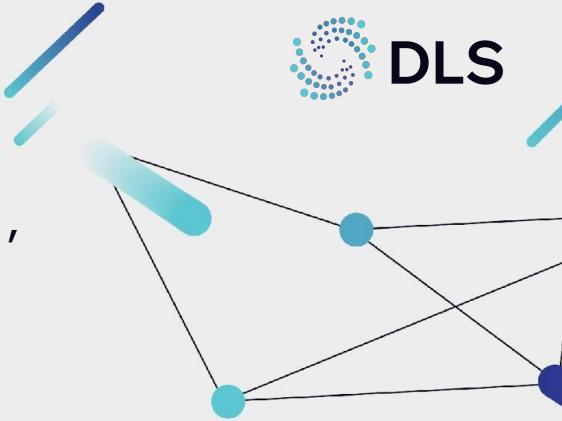
Region proposals



Чтобы уменьшить область поиска перспективных регионов, можно использовать **region proposal** алгоритмы.

Это семейство алгоритмов, которые ищут “кандидатов на объекты” на изображении.

Region proposals

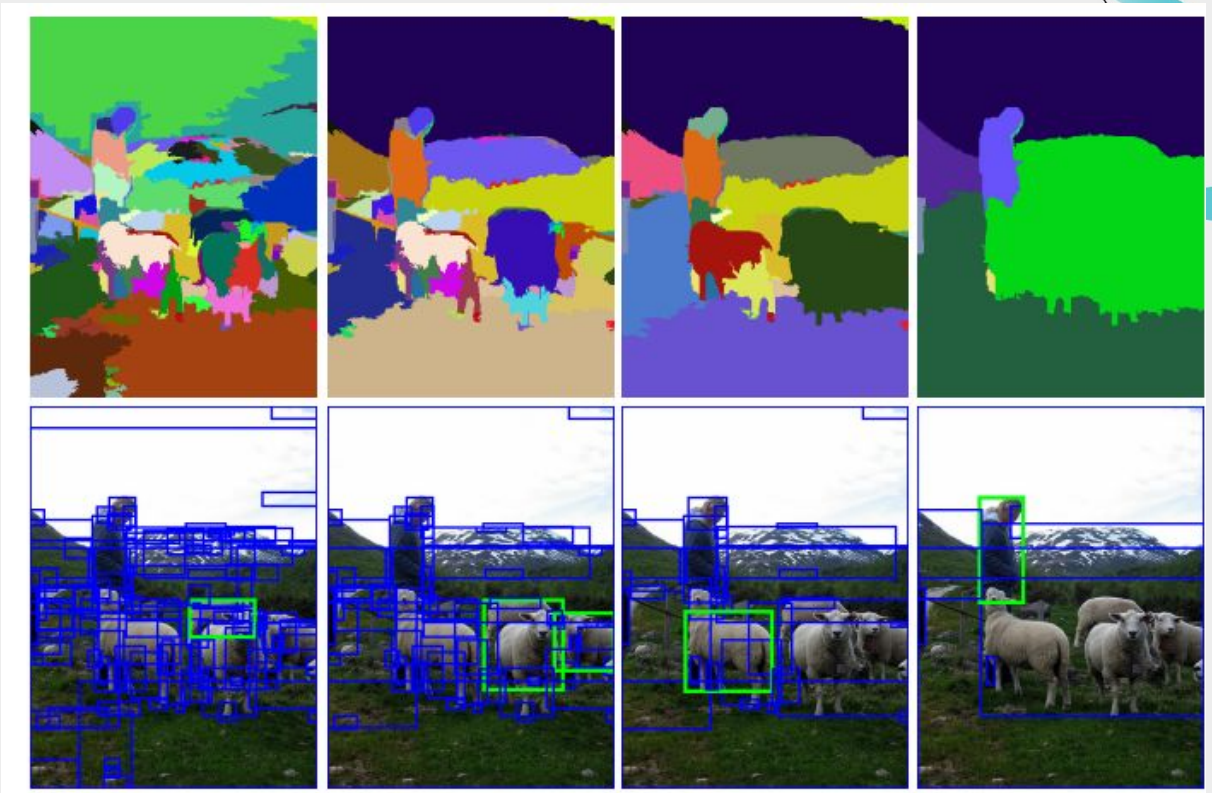


Чтобы уменьшить область поиска перспективных регионов, можно использовать **region proposal** алгоритмы.

Это семейство алгоритмов, которые ищут “кандидатов на объекты” на изображении.

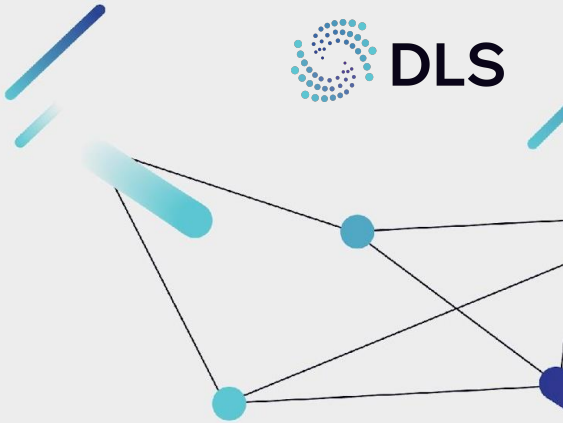
Например, **selective search** — итеративный алгоритм, который разбивает изображение на “суперпиксели” (маленькие области схожего цвета или текстуры) и затем объединяет эти области в более крупные регионы.

Selective search. Пример.



Source: Selective Search for Object Recognition
J. R. R. Uijlings, K. E. A. van de Sande

Selective search.



Плюсы:

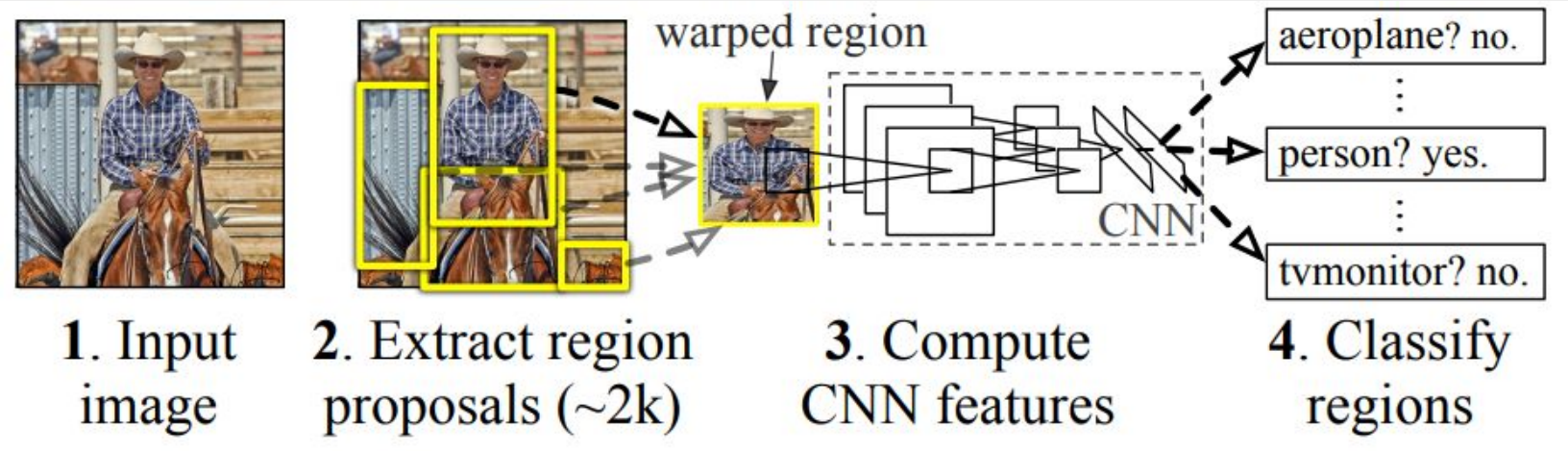
- Меньше потенциальных регионов для поиска объекта, по сравнению с наивным подходом,
- Обладает большим recall,
- Его не нужно обучать,
- Сразу получаем регионы разного размера, что увеличивает разнообразие объектов, которые мы можем найти.

Минусы:

- Алгоритм долгий (>100ms для изображения),
- Выдает достаточно много регионов, где нет объектов.

Region-based CNN (R-CNN)

Несмотря на минусы, *selective search* лег в основу одного из самых первых детекторов на основе нейросетей — **Region-based CNN (2013).**

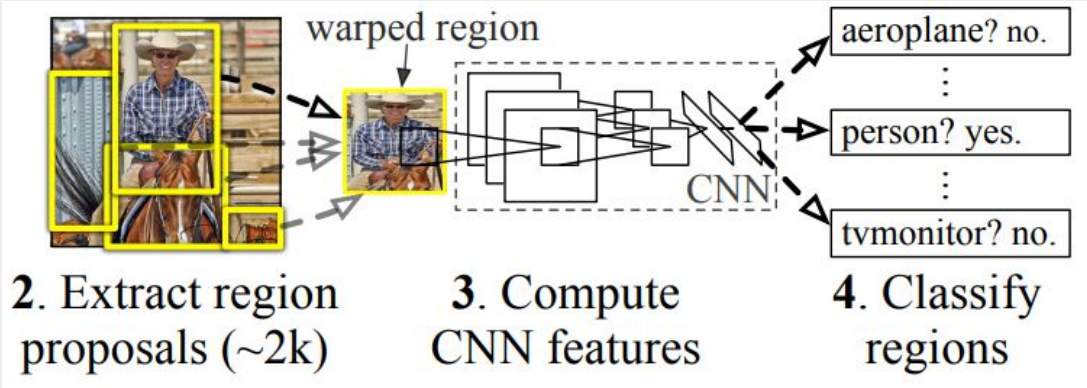


Source: Rich feature hierarchies for accurate object detection and semantic segmentation.
https://doi.org/10.48550/arXiv.1311.2524

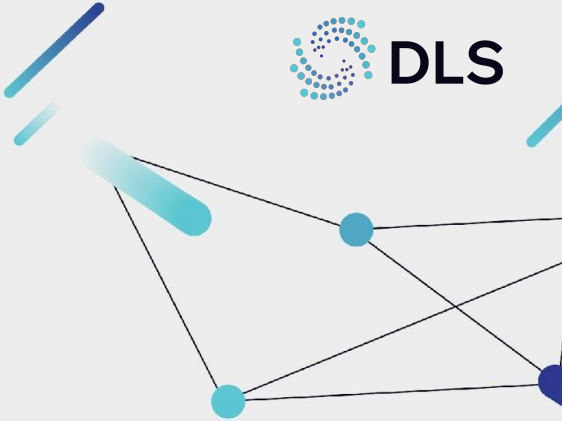
Region-based CNN (R-CNN)

Алгоритм работы R-CNN можно описать примерно так:

1. Генерим примерно 2000 регионов-кандидатов,
2. Приводим все регионы к одному размеру (227x227),
3. Прогоняем каждый регион через AlexNet (предобученный на ImageNet),
4. К полученному вектору фичей размером 4096, применяем (N+1) SVM-ов, чтобы посчитать вероятности принадлежности региона к каждому из классов.



Region-based CNN (R-CNN)



Алгоритм работы R-CNN можно описать примерно так:

1. Генерим примерно 2000 регионов-кандидатов,
   - Уточняем координаты регионов с помощью линейной регрессии.
2. Приводим все регионы к одному размеру (227x227),
3. Прогоняем каждый регион через AlexNet (предобученный на ImageNet),
   - Дообучаем AlexNet для детекции на Pascal VOC.
4. К полученному вектору фичей размером 4096, применяем (N+1) SVM-ов, чтобы посчитать вероятности принадлежности региона к каждому из классов.

Region-based CNN (R-CNN)

Плюсы:

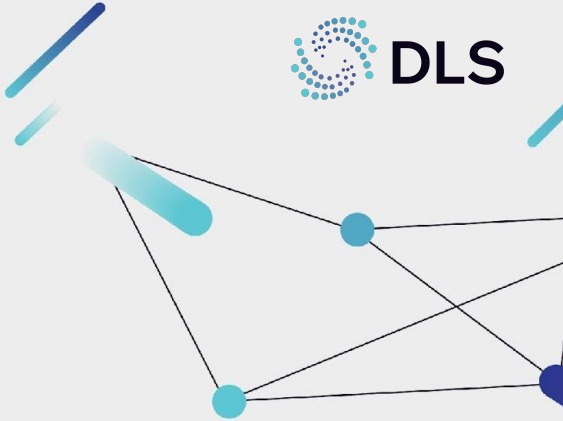
- Одно из первых успешных применений CNN в задаче детекции,
- Достаточно высокая точность при обработке каждого региона,
- **mAP 58.5** на PASCAL VOC 2007.

Минусы:

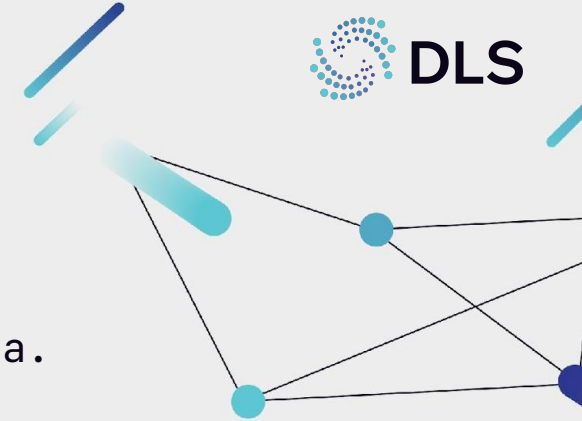
- Очень много вычислений, ведь каждый регион обрабатывается отдельно,
- Длительное время инференса.

Fast R-CNN

Какой самый главный недостаток R-CNN?



Fast R-CNN

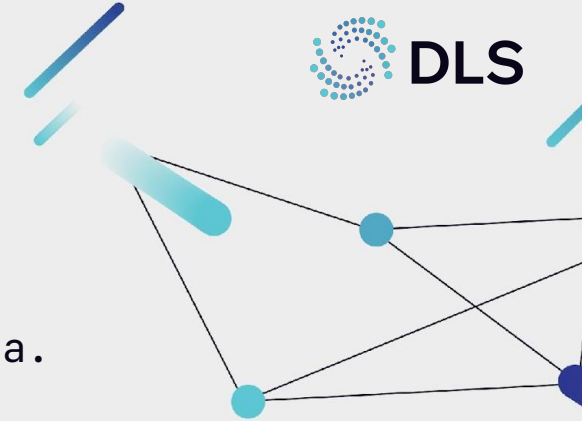


Какой самый главный недостаток R-CNN? ~~200 SVMов на выходе..~~

Использование нейросети отдельно для каждого региона.

Как решить эту проблему?

Fast R-CNN



Какой самый главный недостаток R-CNN?

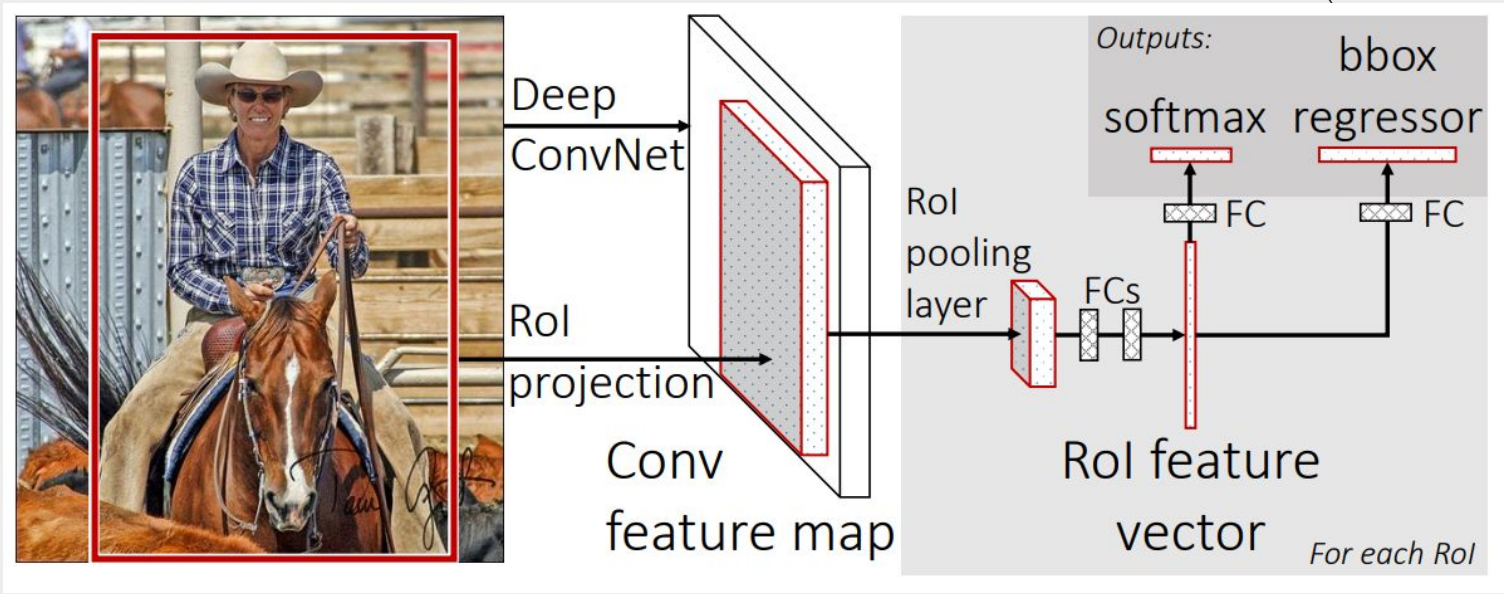
Использование нейросети отдельно для каждого региона.

Как решить эту проблему?

Давайте попробуем уменьшить количество вызовов нейросети:

0. Selective search чтобы вычислить регионы-кандидаты,
1. Считаем feature map по всей картинке,
2. Вырезаем из feature map место, соответствующее позиции региону-кандидату на исходном изображении,
3. Пытаемся предсказать объект внутри feature map’а.

Fast R-CNN. Схема.

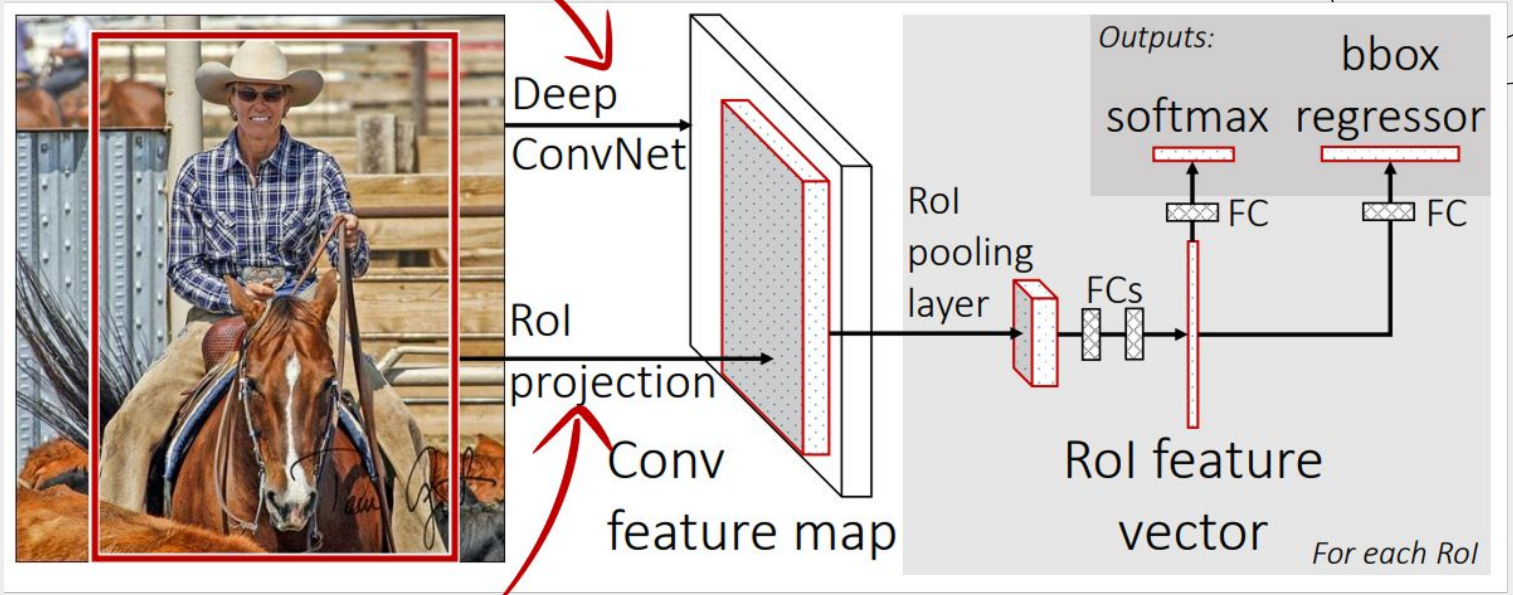


Source: Fast R-CNN
https://doi.org/10.48550/arXiv.1504.08083

* Вышла 30го апреля 2015 года

Fast R-CNN. Схема.

Считаем feature map для изображения (VVG16, AlexNet)

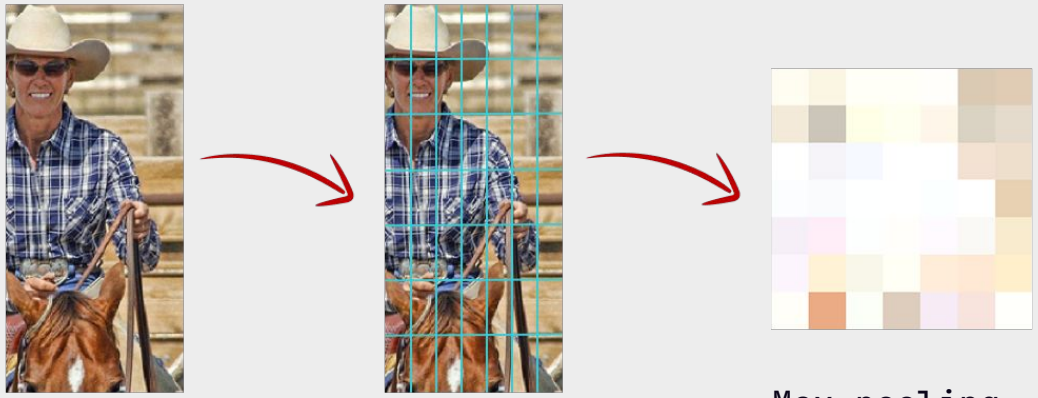


Проецируем регион-кандидат/region proposal (красный квадрат)
на feature map

Fast R-CNN. RoI Pooling.

Для предсказания используется RoI Pooling и несколько полносвязных слоев.

RoI Pooling предназначен для приведения полученных region proposal-ов к одному размеру:

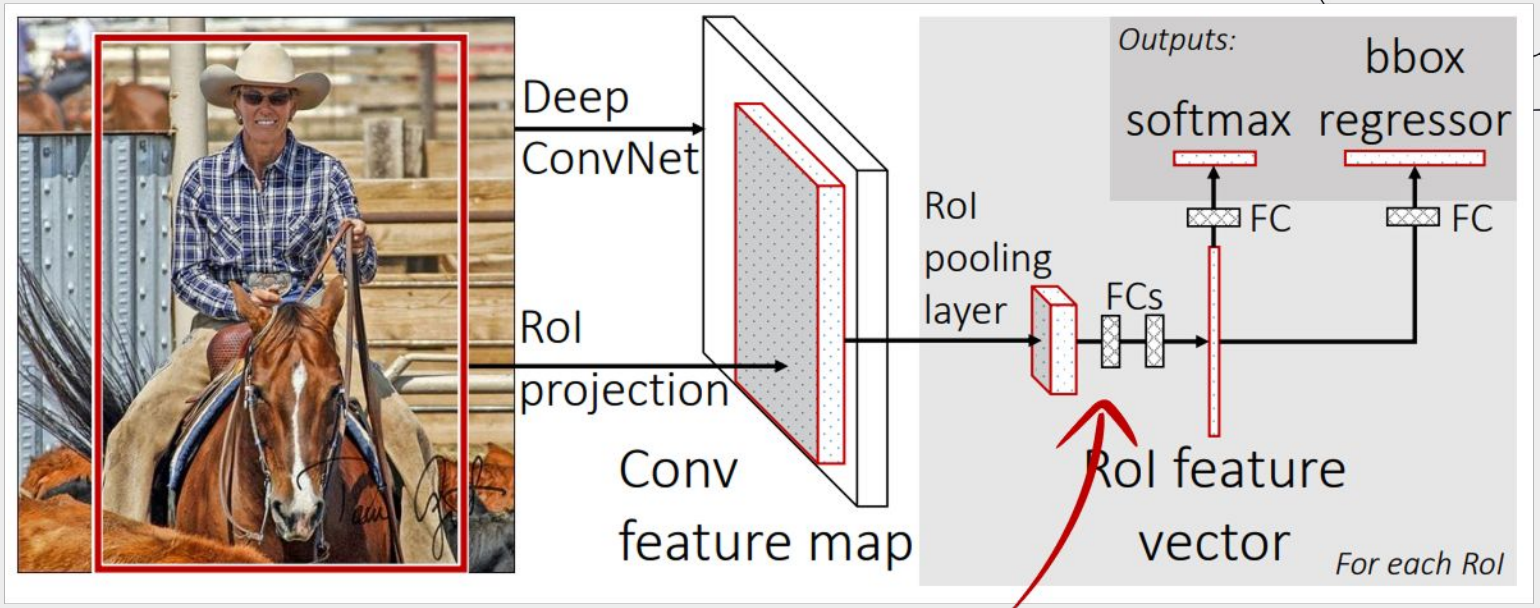


Region Proposal

Накладываем сетку 7x7

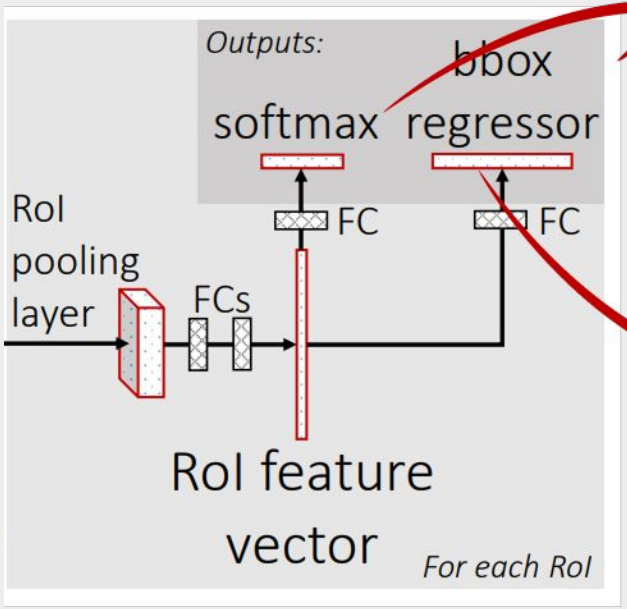
Max pooling внутри каждой ячейки

Fast R-CNN. Схема.



Каждый region proposal независимо прогоняем через несколько полносвязных слоев, получая в итоге 2 вектора: вероятности классов и смещения итогового ббокса.

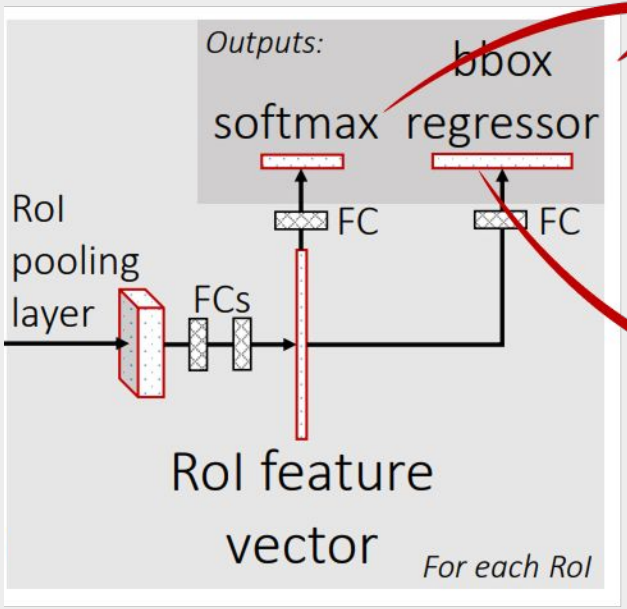
Fast R-CNN. Выход сети.



Вектор размером (N + 1).
Показывает вероятности принадлежности к каждому из классов + задний фон.

4 числа, характеризующие смещение:
t_x - центра по x,
t_y - центра по y,
t_w - ббокса по ширине,
t_h - ббокса по длине.

Fast R-CNN. Выход сети.



Вектор размером (N + 1).
Показывает вероятности принадлежности к каждому из классов + задний фон.

4 числа, характеризующие смещение:
t_x - центра по x,
t_y - центра по y,
t_w - ббокса по ширине,
t_h - ббокса по длине.

Итоговые размеры ббокса:

$$\hat{x} = x + t_x w \qquad \hat{h} = h + exp(t_h)$$

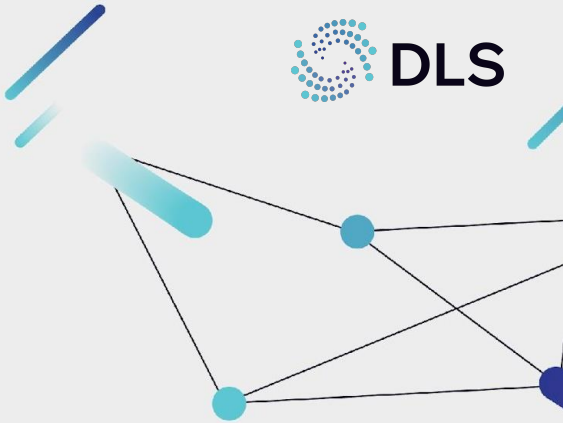
$$\hat{y} = y + t_y w \qquad \hat{w} = w + exp(t_w)$$

Fast R-CNN. Обучение.

$$L = L_{cls}(p, u) + \lambda[u \geq 1]L_{loc}(t, v)$$

p – выходы модели после softmax,
u – истинный класс внутри ббокса (номер класса, где обычно 0 – фон),
t – предсказанные смещения ббокса,
v - истинные смещения ббокса,
[u≥1] – индикаторная функция, 1 везде, кроме фона
λ – коэффициент балансировки между классификационной и локализационной частью. Обычно равен 1.

Fast R-CNN. Обучение.



$$L = L_{cls}(p, u) + \lambda[u \geq 1]L_{loc}(t, v)$$

Классификационная часть:

$$L_{cls}(p, u) = -log(p_u)$$

Регрессионная часть (Huber loss):

$$L_{loc}(t, v) = \sum_{i \in x,y,h,w} smooth L_1(t_i - v_i)$$

$$\text{smooth}\, L_1(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise} \end{cases}$$

Fast R-CNN. Итоги.

Плюсы:

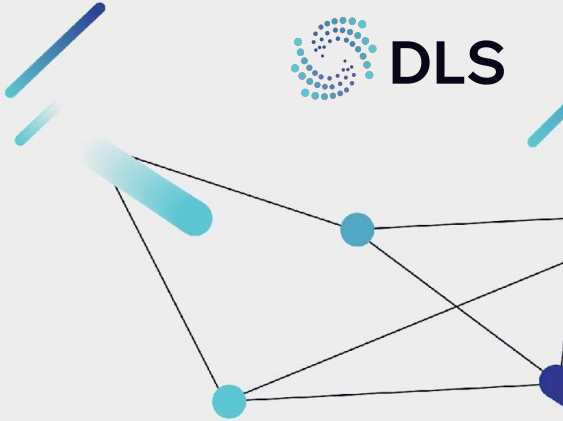
- Избавились от множества костылей и практически пришли к end-to-end модели,
- Получили значительное ускорение по сравнению с RCNN,
- **mAP 70** на PASCAL VOC 2007.

Минусы:

- Регионы-кандидаты (region proposal) все ещё генерируются “руками” (selective search-ом).

Faster R-CNN

Чтобы сделать модель ещё быстрее, нужно перестать использовать selective search, но как?...

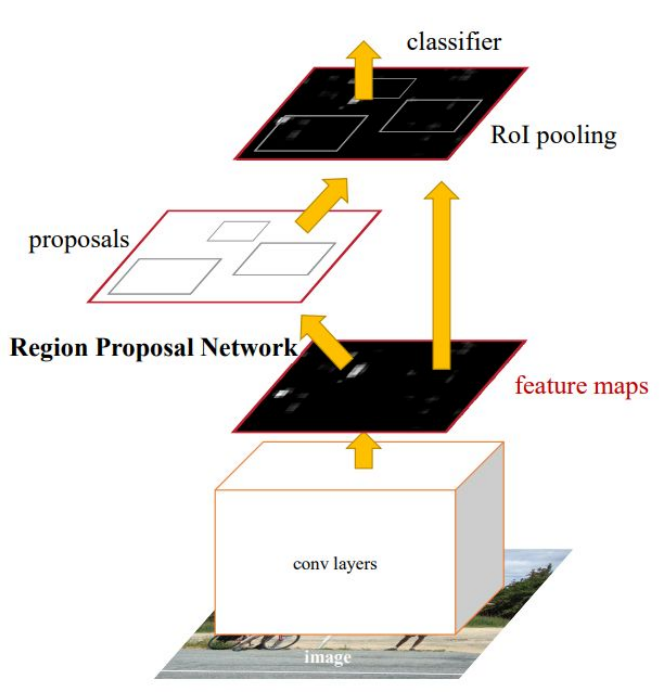


Faster R-CNN

Чтобы сделать модель ещё быстрее, нужно перестать использовать selective search, но как?...

Давайте попробуем заставить модель саму генерить ббоксы используя **Region Proposal Network.**

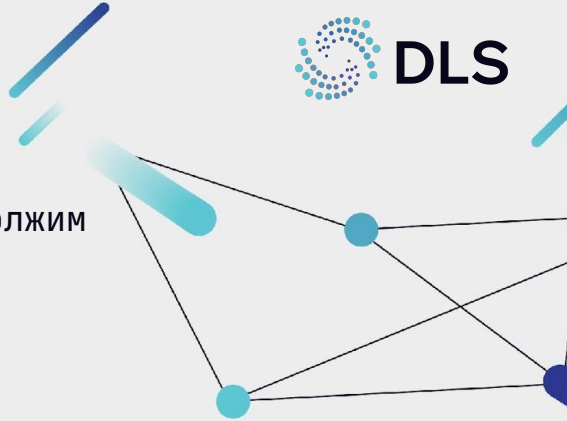
*вышла через пару месяцев после Fast R-CNN (4 июня 2015)



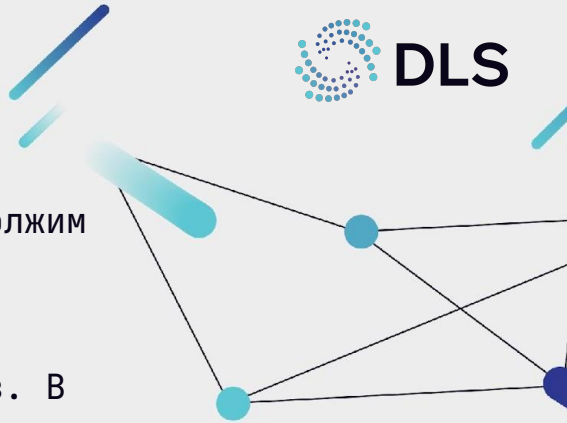
Source: Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks
https://doi.org/10.48550/arXiv.1506.01497

Faster R-CNN. PRN.

Генерация регионов с нуля задача очень сложная, поэтому продолжим работать со смещениями.



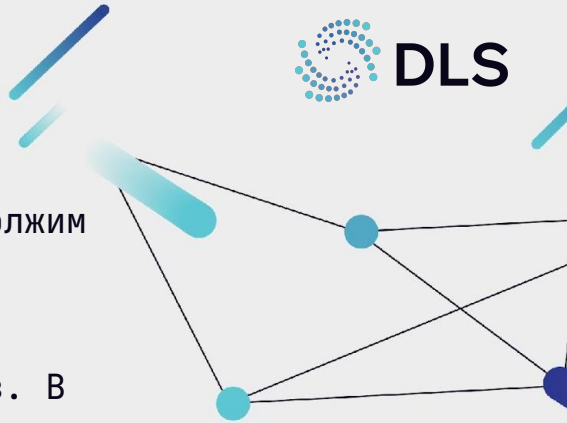
Faster R-CNN. PRN.



Генерация регионов с нуля задача очень сложная, поэтому продолжим работать со смещениями.

Идея: Давайте для каждого пикселя (координаты feature map, в нашем случае) придумаем несколько базовых регионов-кандидатов. В литературе они называются
anchors/анкеры или якоря.

Faster R-CNN. PRN.



Генерация регионов с нуля задача очень сложная, поэтому продолжим работать со смещениями.

Идея: Давайте для каждого пикселя (координаты feature map, в нашем случае) придумаем несколько базовых регионов-кандидатов. В литературе они называются
anchors/анкеры или якоря.

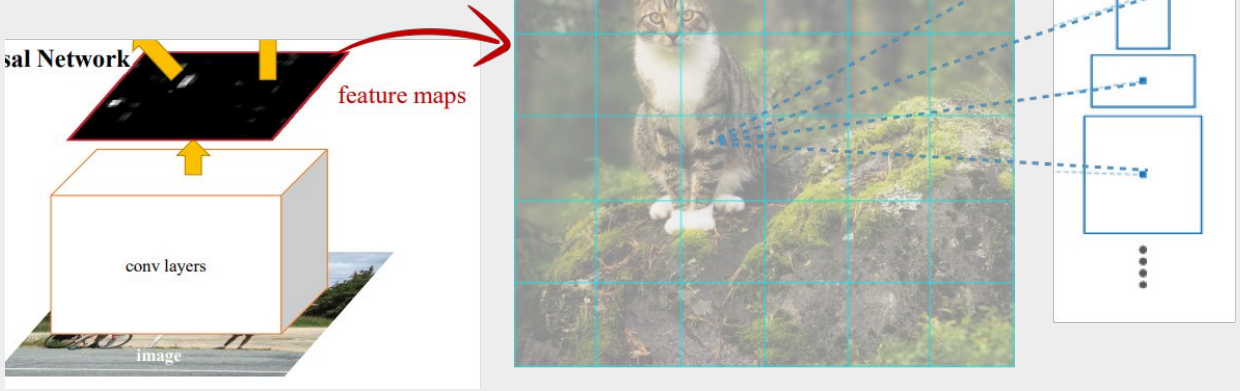
Размеры якорей можно считать по-разному:

- Задать базовый размер (например 256x256), и брать якоря с разными соотношениями сторон (1:1, 2:1, 1:2) + скейлить базовый размер (x0.5, x2, x4).
- Посчитать на основе истинных ббоксов вашего train set и взять несколько усредненных размеров.

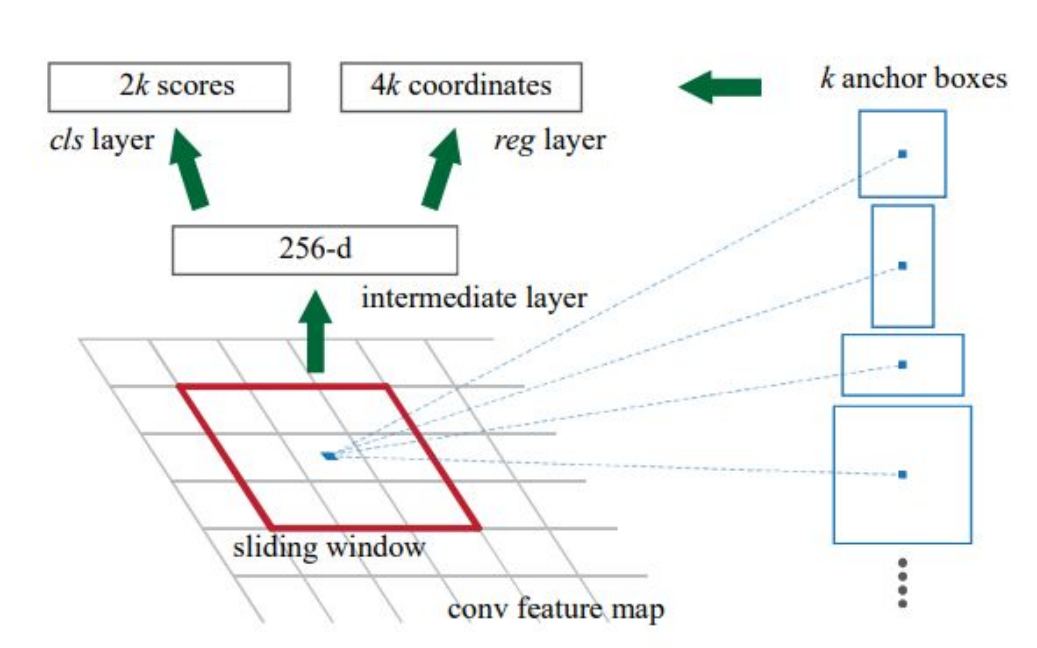
Faster R-CNN. PRN.

Генерация регионов с нуля задача очень сложная, поэтому продолжим работать со смещениями.

Идея: Давайте для каждого пикселя (координаты feature map, в нашем случае) придумаем несколько базовых регионов-кандидатов. В литературе они называются **anchors/анкеры или якоря.**



Faster R-CNN. PRN.

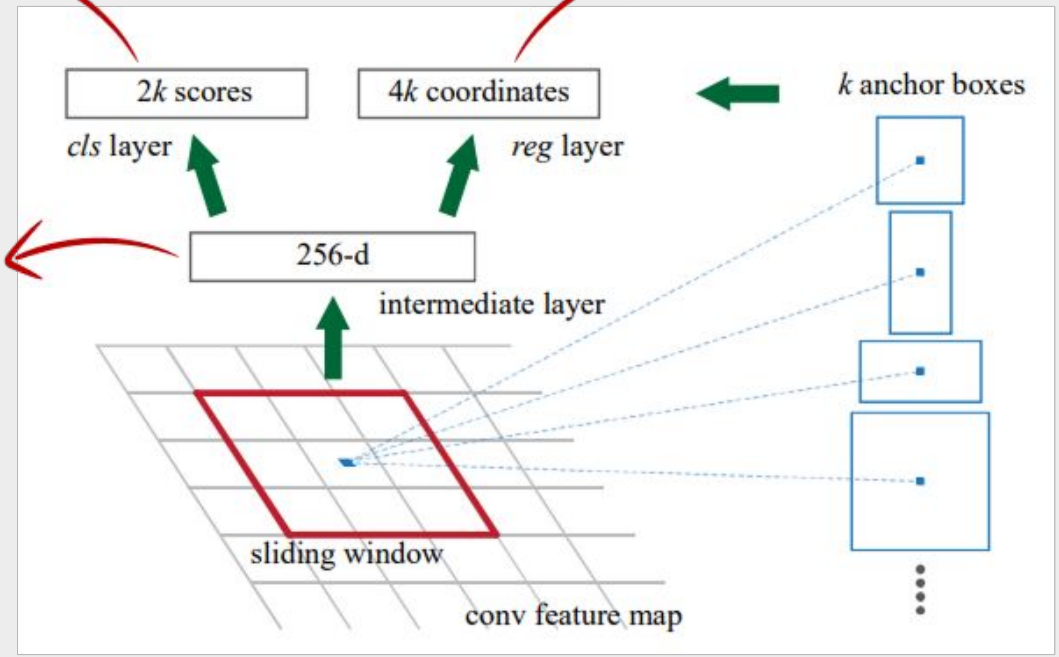


Faster R-CNN. PRN.

Conv layer 1x1

Conv layer 1x1

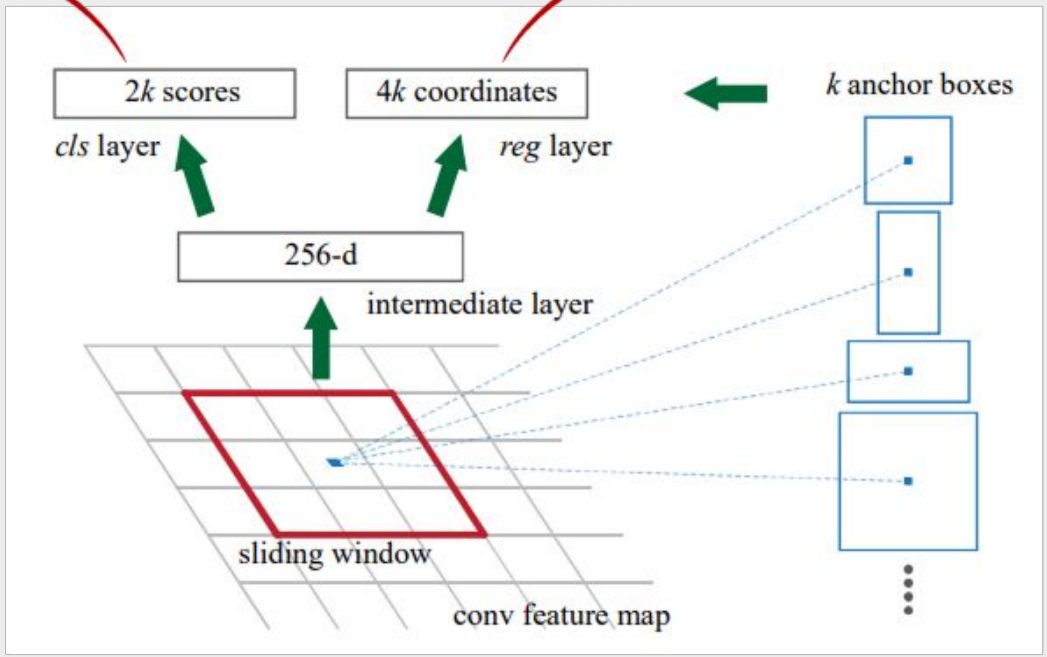
Conv layer 3x3



Faster R-CNN. PRN.

Objectness score для каждого якоря

Координаты смещений для каждого якоря



Objectness score – вероятность что в данном якоре есть объект.

Faster R-CNN

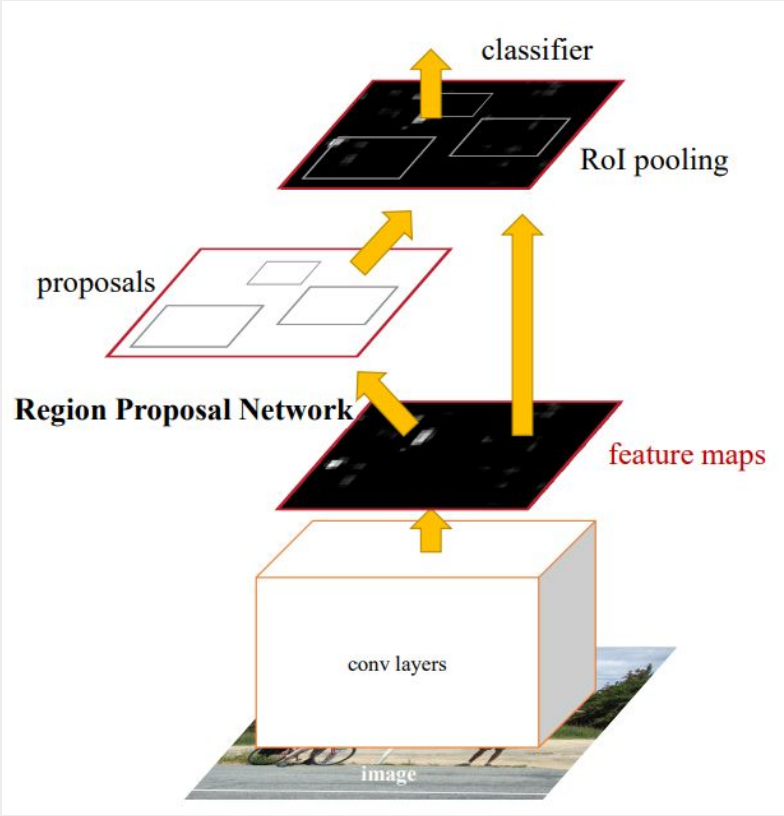
Итоговый выход RPN – **proposals** (якоря + смещения)

А итоговое предсказание делается с помощью Fast R-CNN head.

Выход модели следующий:

- Вероятности принадлежности к N + 1 классу,
- 4 координаты смещений для каждого якоря.

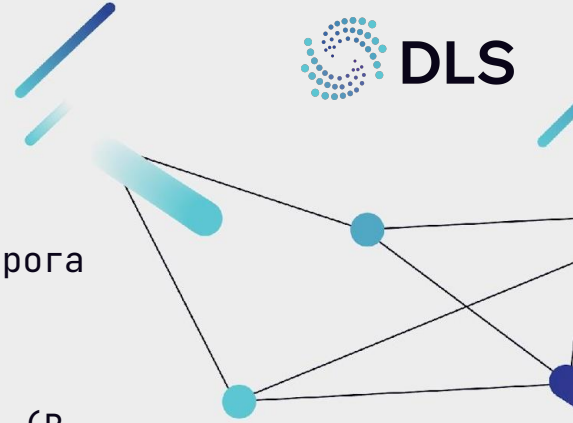
Важно: До Fast R-CNN head доходят не все пропозалы! Они фильтруются сначала по confidence score, а потом проходят через NMS.



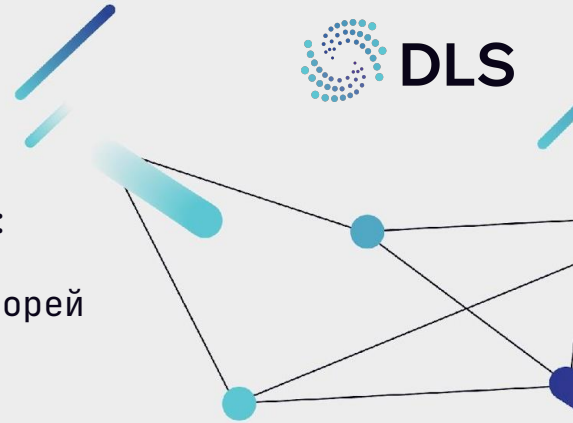
PRN. Label assignment.

В случае PRN, каждому якорю присваивается метка:

- Положительный (Object): Если IoU с каким-либо GT выше порога (допустим 0.7),
- Отрицательный (Background): Если IoU с GT ниже порога (например 0.3),
- Игнорируемый: Если IoU с GT располагается между порогами (В нашем случае от 0.3 до 0.7).



PRN. Training.

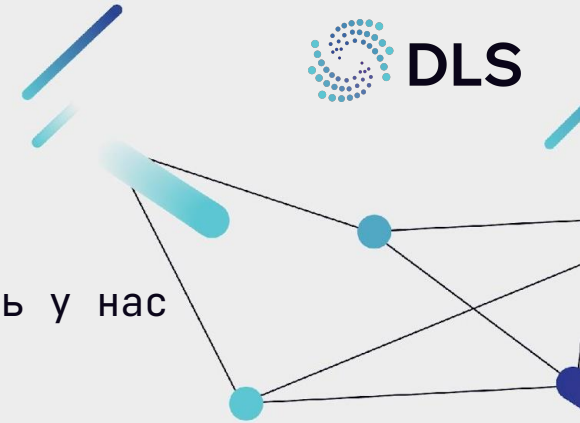


Для обучения RPN также используется составная функция потерь:

- Классификационная: Log loss, по всем якорям.
- Регрессионная: SmoothL1, которая считается только для якорей с положительной меткой.

PRN может обучаться как отдельно, так и вместе со всей остальной частью архитектуры.

Faster R-CNN. Итоги.



Плюсы:

- Избавились от дополнительных алгоритмов и теперь у нас есть честная end-to-end модель,
- Значительно повысилась скорость инференса,
- Получили самый точной детектор,
- **mAP 73.2** на PASCAL VOC 2007, Average **mAP 16.4** COCO test dev.

Минусы:

- Архитектура все ещё остается двухэтапной: есть RPN и Fast R-CNN head, это усложняет процесс обучения и замедляет инференс,
- Модель чувствительна к начальным размерам якорей.

Mask и Cascade R-CNN

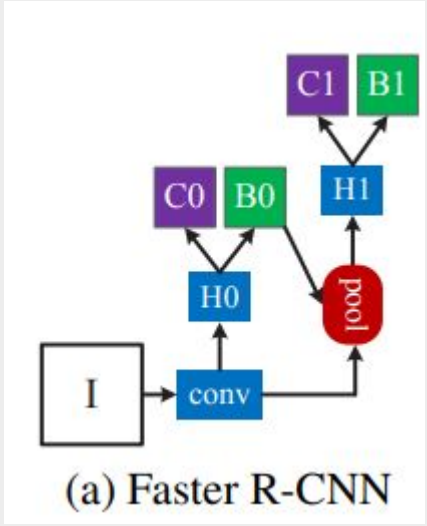
Mask RCNN (2017):

- Расширение Faster R-CNN на задачу instance segmentation, то есть выдает не только ббокс объекта, но и маску для него.

Cascade RCNN (2017):

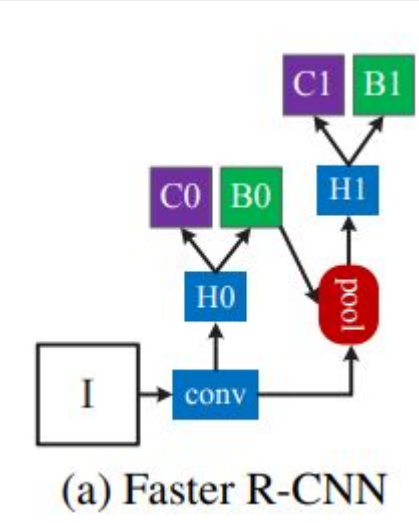
- Улучшенная версия Faster R-CNN за счет усложнения Fast R-CNN head,

Cascade R-CNN

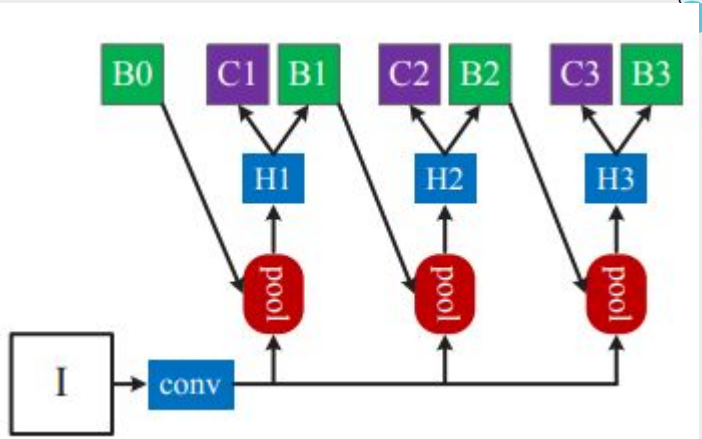


Source: Cascade R-CNN: Delving into High Quality Object Detection
https://doi.org/10.48550/arXiv.1712.00726

Cascade R-CNN



(a) Faster R-CNN

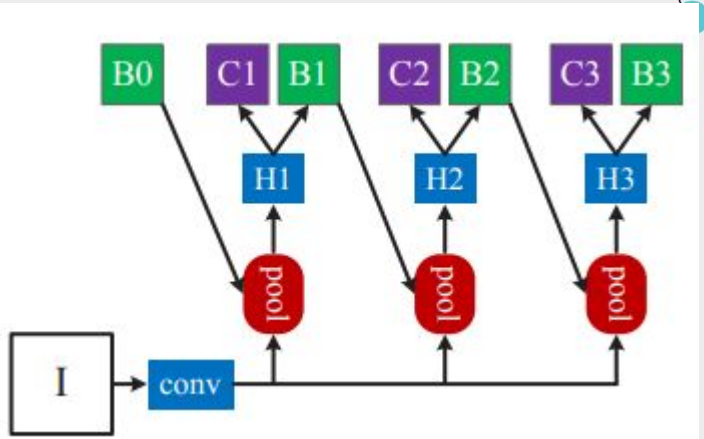


На каждом шаге последовательно уточняются положения якорей.

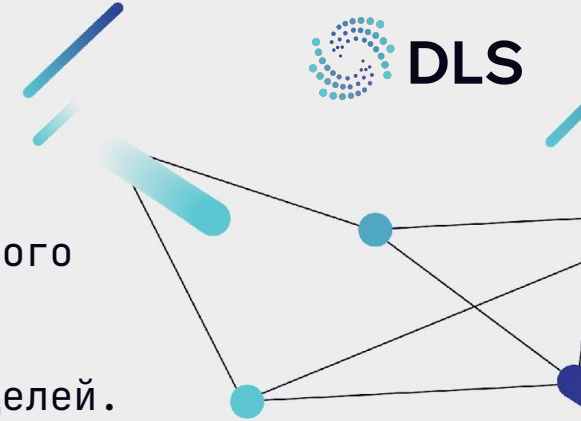
Cascade R-CNN

В процессе обучения, каждая стадия обучается с разными порогами IoU для положительных якорей:
B0 - Использует порог 0.5 для положительных примеров,
B1 — 0.6,
B2 — 0.7 и тд.

Таким образом, входящие боксы на новой стадии будут ближе к GT.



Cascade R-CNN. Итоги.



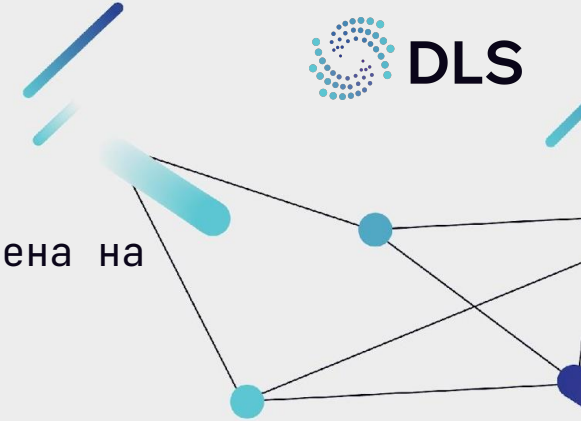
1. Введение многоступенчатой обработки и постепенного уточнения предсказания позволило значительно повысить итоговое качество,
2. Архитектура имела важное влияние в развитии моделей. Например, на ней основана другая популярная модель Hybrid Task Cascade (HTC),
3. Average **mAP 42.8** на COCO test dev.

Минусы:

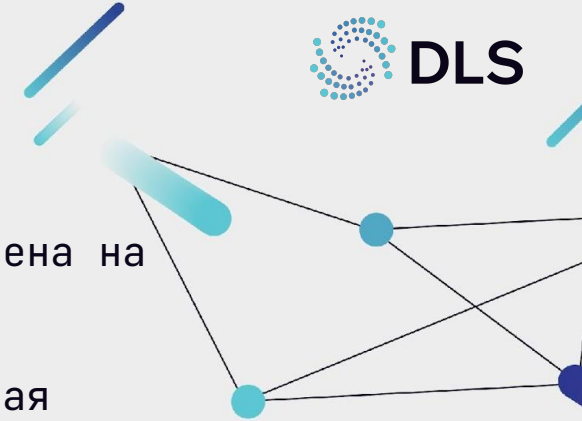
- Остаются все минусы Faster R-CNN.

You Look Only Once

Первая модель из YOLO семейства, которая была нацелена на решение задачи детекции в real-time.



You Look Only Once

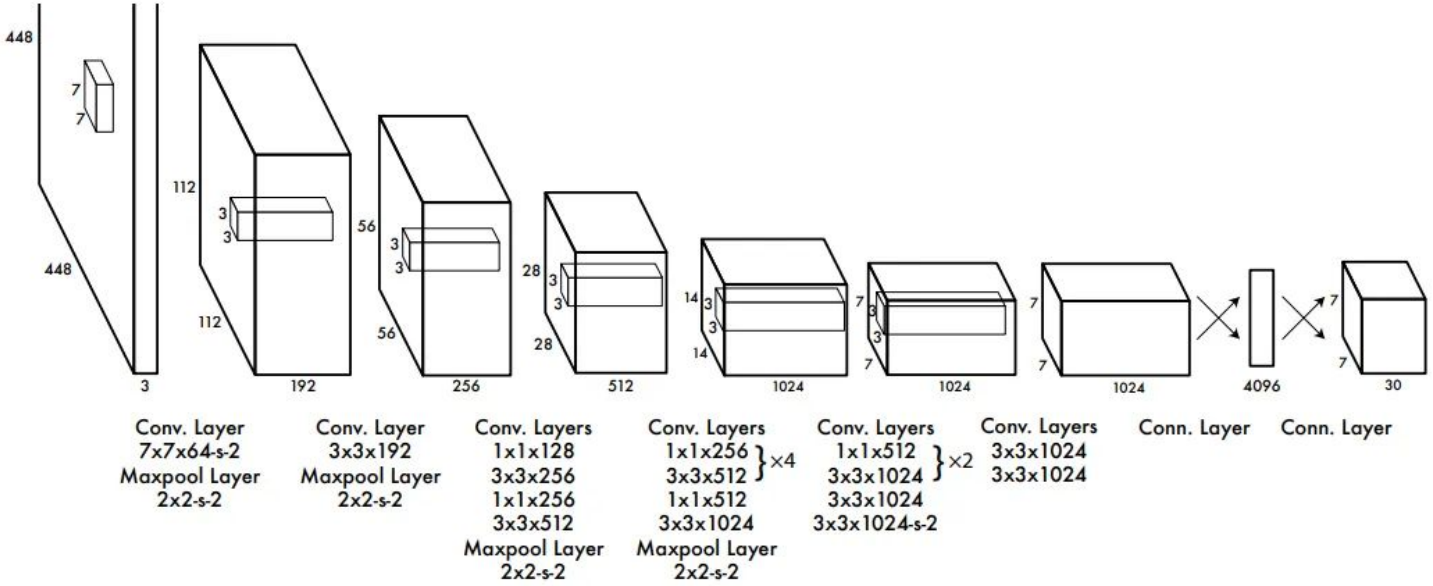


Первая модель из YOLO семейства, которая была нацелена на решение задачи детекции в real-time.

На момент выхода модели, SOTA — Faster R-CNN, которая работает со скоростью 17 FPS.

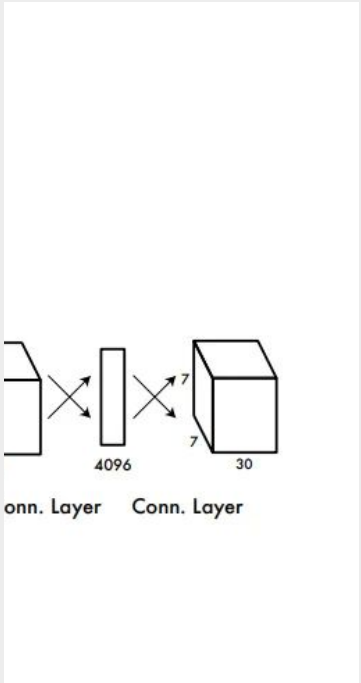
Цель YOLO — необязательно SOTA качество, главное быстрее.

YOLO. Архитектура.



Модель состоит из 24 сверточных слоев и 2х полносвязных:
Backbone — 20 первых слоев GoogLeNet, предобученных на Imagenet,
Head — 4 сверточных слоя + 2 полносвязных.

YOLO. Выход модели.



В YOLO не используются якори, фактически это один из первых anchor-free детекторов.

Размер выходной feature map’ы вычисляется как 10+N, где:

- 10 — это координаты **двух** ббоксов + 2 confidence score,
  - x, y, w, h в нотации YOLO,
  - Confidence score — уверенность что центр объекта находится в данной ячейке.
- N — вероятности для каждого из классов в датасете.

YOLO. Loss.

Детекционная часть

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

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

λ_{coord} – Нормализационная константа, чтобы увеличить вес для ячеек в которых находится центр объекта.
1_{ij}^{obj} – индикаторная функция, 1 если в i-й ячейке (пикселе) есть центр объекта.

Для каждого пикселя на выходной feature map’е, где расположен центр объекта, в лоссе учитывается тот ббокс, у которого больше пересечение с GT по IoU.

* Модель предсказывает нормализованные координаты (x, y), те в интервале [0, 1) и квадратные корни ширины и высоты ббокса.

YOLO. Loss.

Детекционная часть

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

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

Но почему мы предсказываем именно корень от размеров ббокса?

YOLO. Loss.

Детекционная часть

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

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

Но почему мы предсказываем именно корень от размеров ббокса?

Чтобы учитывать относительные ошибки, а не абсолютные.
Например, если у нас есть 2 ббокса размерами (100, 100) и (20, 20). Допустим модель ошиблась на 10 для обоих ббоксов, тогда абсолютная ошибка будет равна $10^2 + 10^2$, а относительная:

$$(\sqrt{100} - \sqrt{90})^2 + (\sqrt{100} - \sqrt{90})^2 = 0.98$$

$$(\sqrt{20} - \sqrt{10})^2 + (\sqrt{20} - \sqrt{10})^2 = 2.87$$

YOLO. Loss.

Confidence score

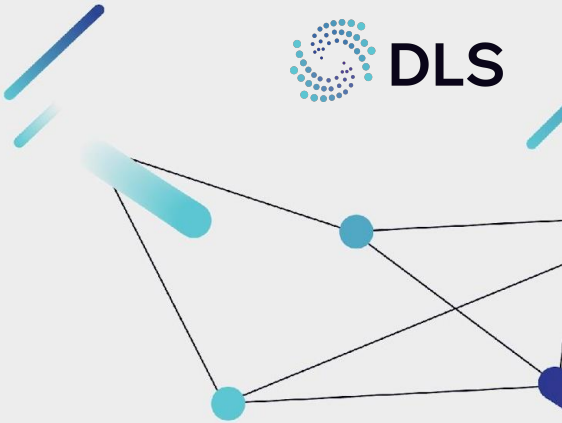
$$+\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{\text{obj}}\left(C_i-\hat{C}_i\right)^2$$

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

Confidence score учитывается по-разному для положительных и отрицательных ббоксов.

λ_{noobj}– вес для ббоксов без объектов (обычно 0.5).

YOLO. Loss.

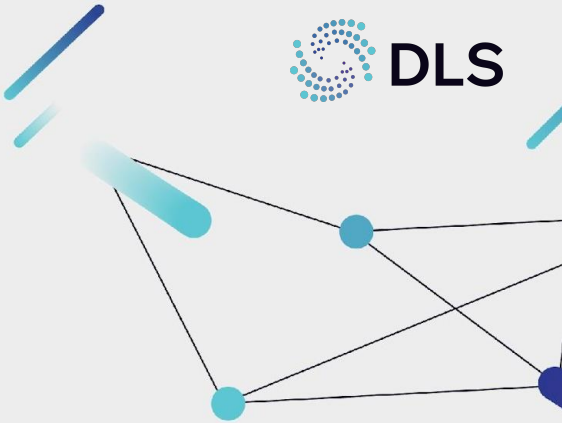


Классификация

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

Классификационная часть лосса также считается только для пикселей, в которых расположен центр объекта.

YOLO. Loss.



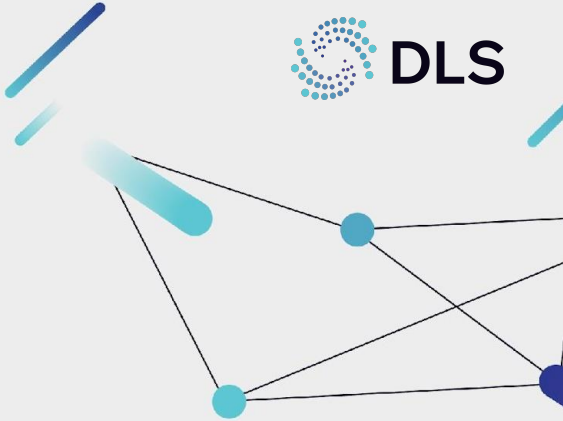
Классификация

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

Классификационная часть лосса также считается только для пикселей, в которых расположен центр объекта.

Да, MSE используется для задачи классификации:)

YOLO. Итоги.



Самый быстрый детектор на 2015 год, работающий с хорошей точностью. 45 FPS при **mAP 63.4** на PASCAL VOC2007. (Faster R-CNN имела **mAP 73.2**)

Плюсы:

- Очень простой в реализации детектор,
- Имеет супер мало параметров, поэтому работает быстро.

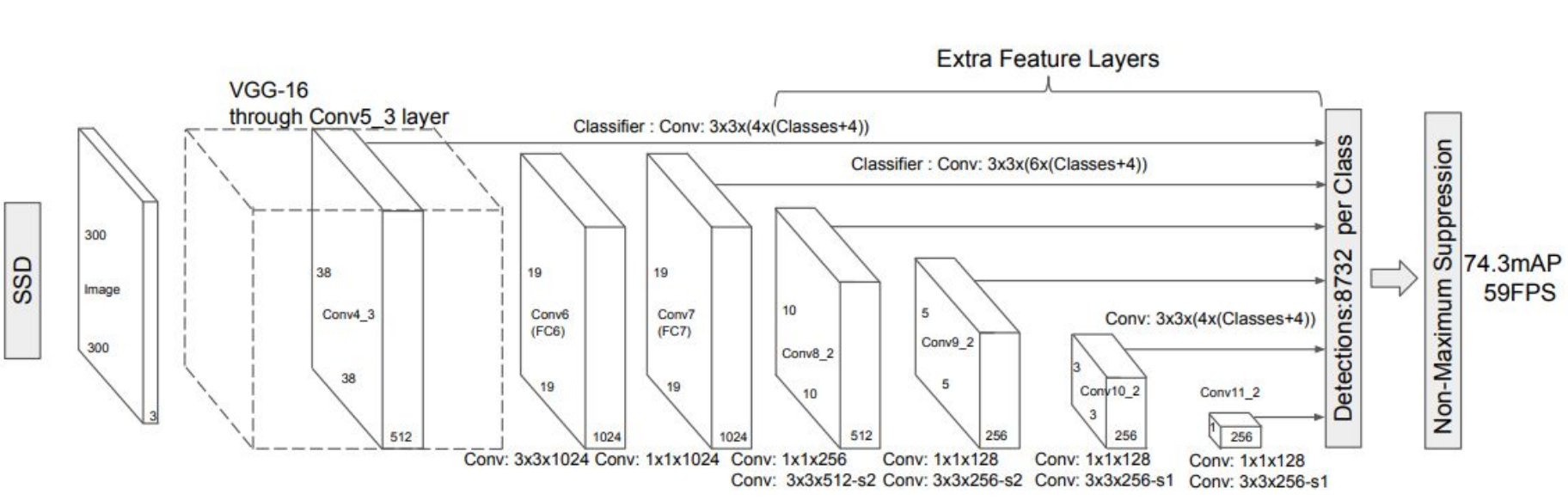
Минусы:

- Плохо работает с маленькими объектами и группами объектов, расположенных рядом,
- Качество все таки достаточно плохое.

YOLO. Фактичек.

О качестве: Во время live-demo, yolo распознала дверь за выступающим как туалет:)

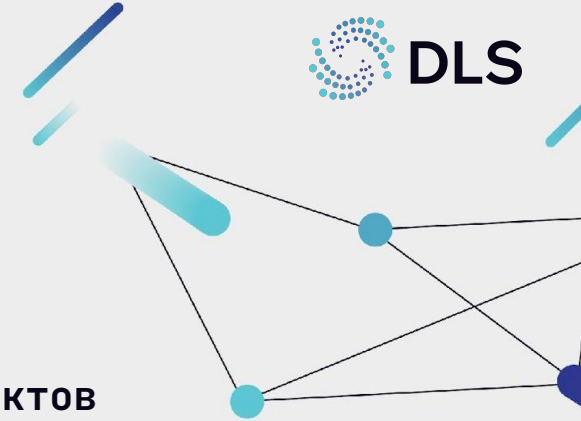
Single Shot Detector (SSD)



Source: SSD: Single Shot MultiBox Detector
https://doi.org/10.48550/arXiv.1512.02325

* Вышла 8 декабря 2015 года

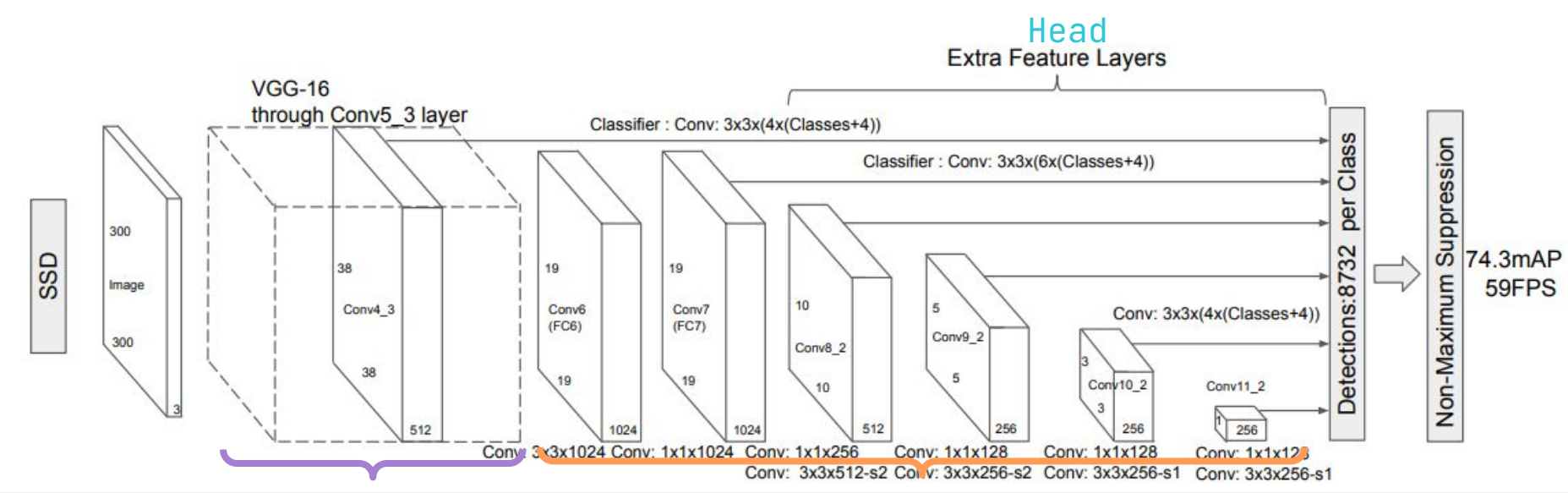
Single Shot Detector (SSD)



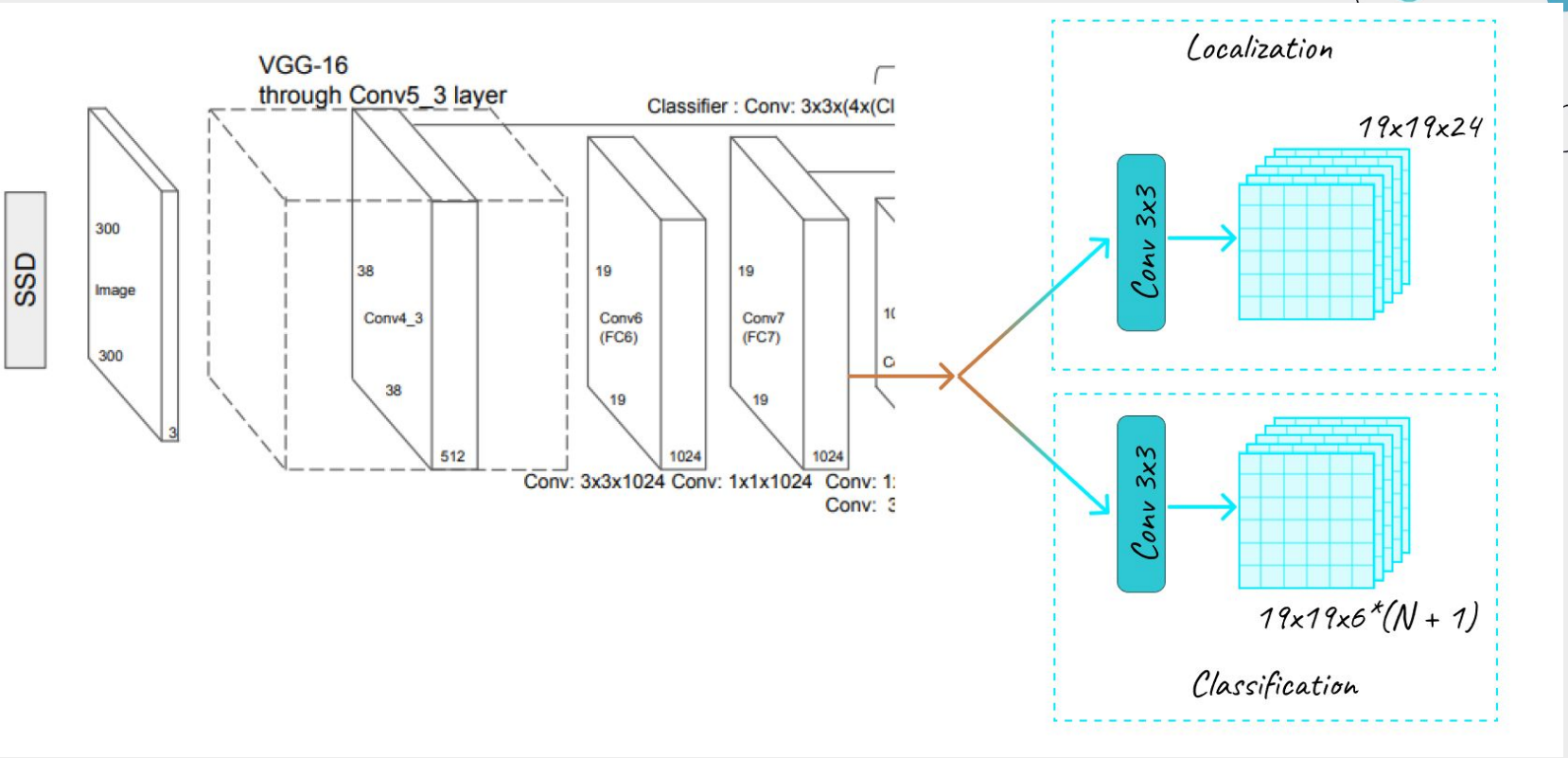
Модель SSD вышла 8го декабря 2015 года.

Главная особенность:
Использование multi-scale фичей для нахождения объектов разного размера.

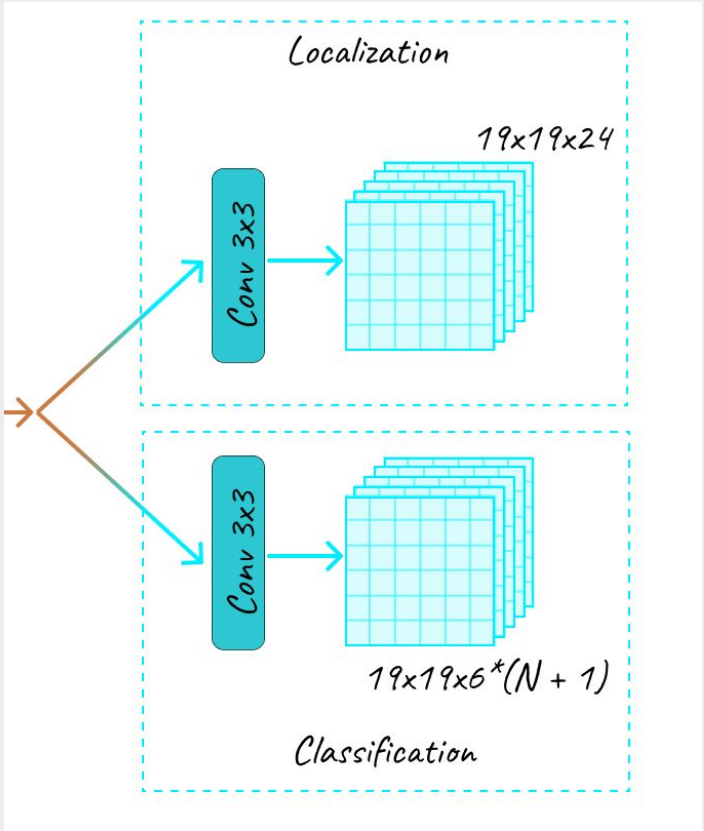
Single Shot Detector (SSD)



Single Shot Detector (SSD)



Single Shot Detector (SSD)



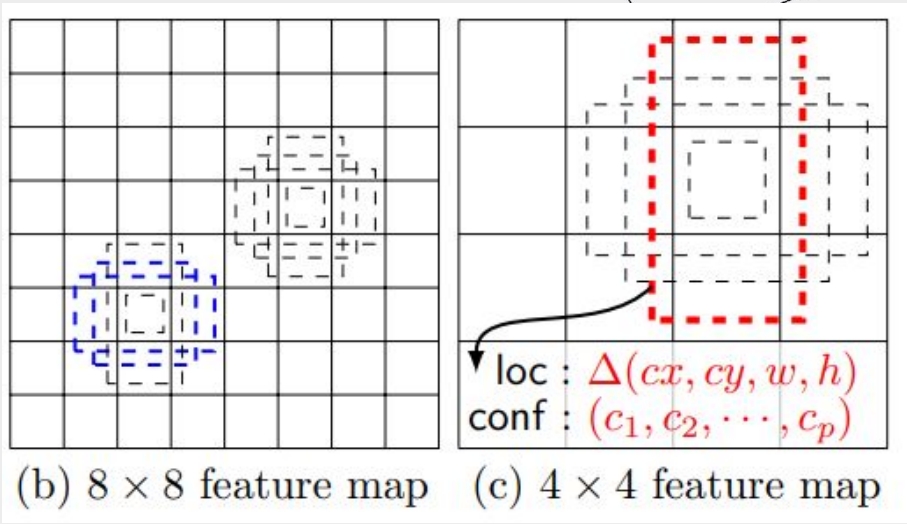
Localization output:
24 = 6 якорей по 4 координаты смещений.

Classification output:
6 якорей по N+1 confidence score (или вероятность принадлежности якоря к каждому из классов).

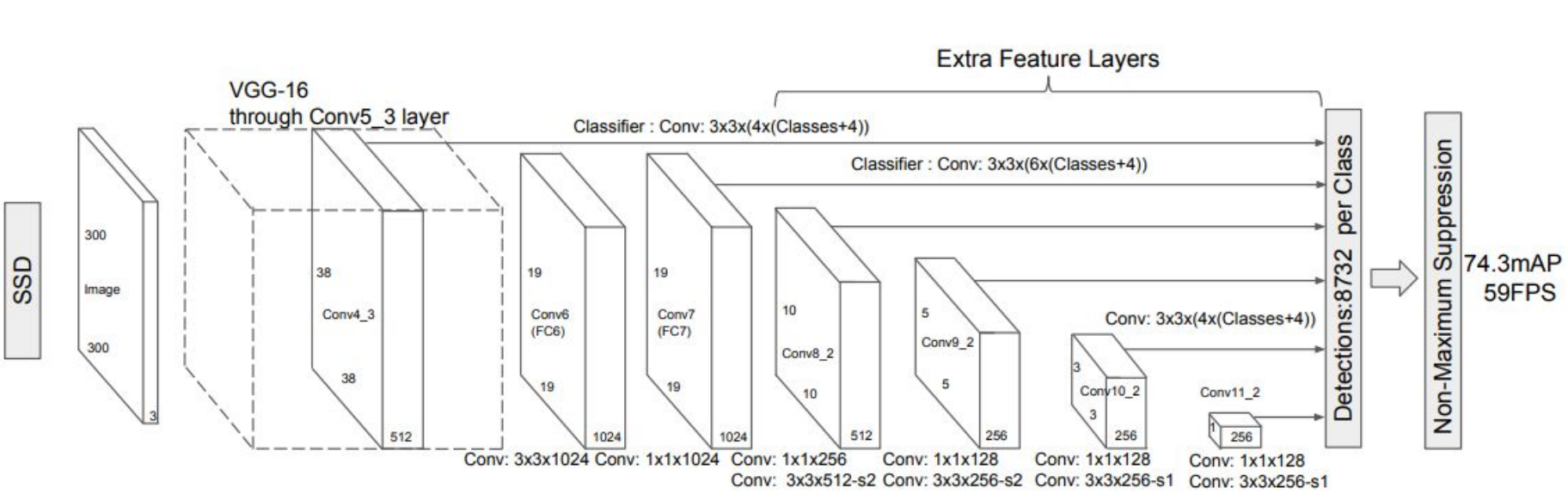
Single Shot Detector (SSD)

Для каждого из feature map’ов, участвующих в предсказании ббоксов, задавались якоря разного размера.

Изначальный размер якорей увеличивается, с уменьшением размера feature map.



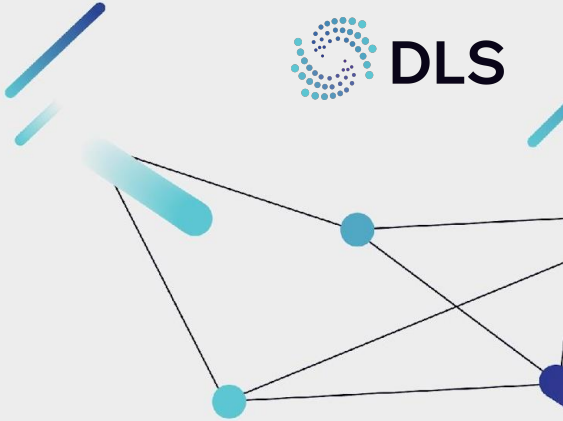
Single Shot Detector (SSD)



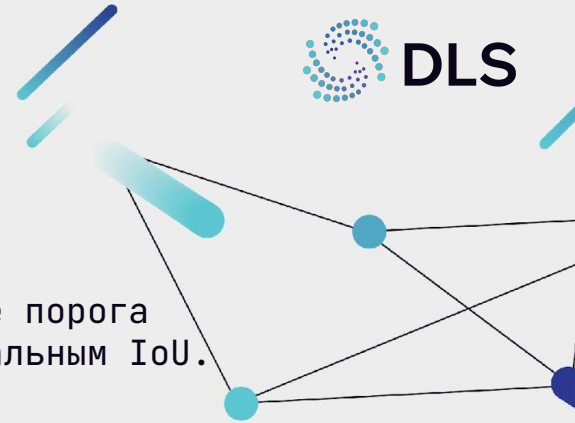
SSD. Label assignment.

Алгоритм Label assignment’а в SSD достаточно простой:

1. Вычисляем IoU между каждым GT и всеми якорями.



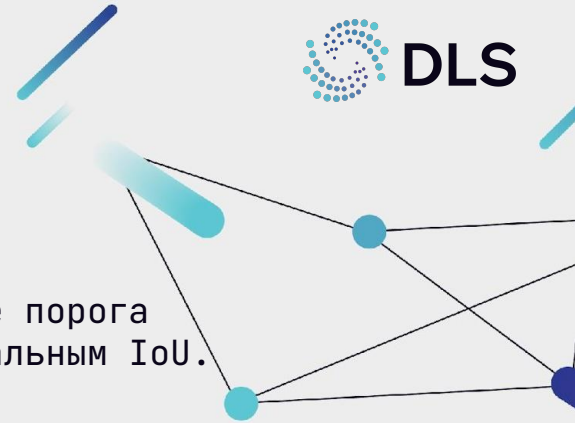
SSD. Label assignment.



Алгоритм Label assignment’а в SSD достаточно простой:

1. Вычисляем IoU между каждым GT и всеми якорями.
2. Для каждого GT выбираем все якоря, с которыми IoU больше порога (например 0.5). Если таких нет, выбираем якорь с максимальным IoU.

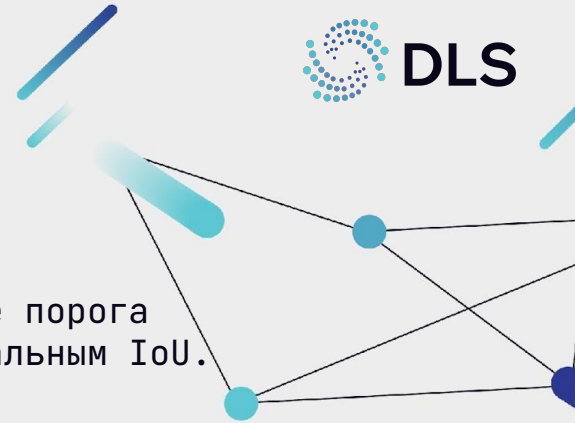
SSD. Label assignment.



Алгоритм Label assignment’а в SSD достаточно простой:

1. Вычисляем IoU между каждым GT и всеми якорями.
2. Для каждого GT выбираем все якоря, с которыми IoU больше порога (например 0.5). Если таких нет, выбираем якорь с максимальным IoU.
3. Отмечаем выбранные якоря как “положительные”.
   На основе предсказаний для “положительных” якорей считается локализационный лосс.

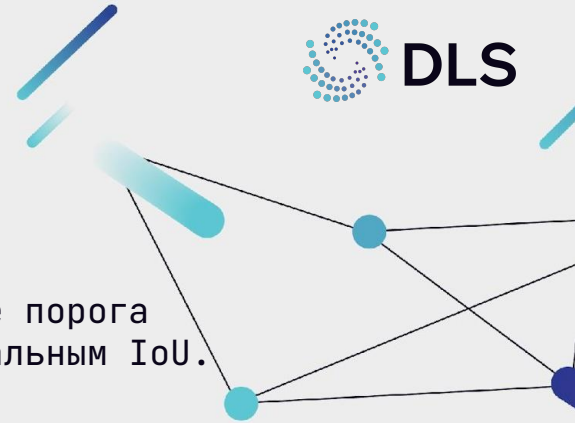
SSD. Label assignment.



Алгоритм Label assignment’а в SSD достаточно простой:

1. Вычисляем IoU между каждым GT и всеми якорями.
2. Для каждого GT выбираем все якоря, с которыми IoU больше порога (например 0.5). Если таких нет, выбираем якорь с максимальным IoU.
3. Отмечаем выбранные якоря как “положительные”.
   На основе предсказаний для “положительных” якорей считается локализационный лосс.
4. У “положительных” якорей проставляются истинные метки классов (такие же как у GT к которым они относятся), остальные якоря помечаются как “фон”.

SSD. Label assignment.



Алгоритм Label assignment’а в SSD достаточно простой:

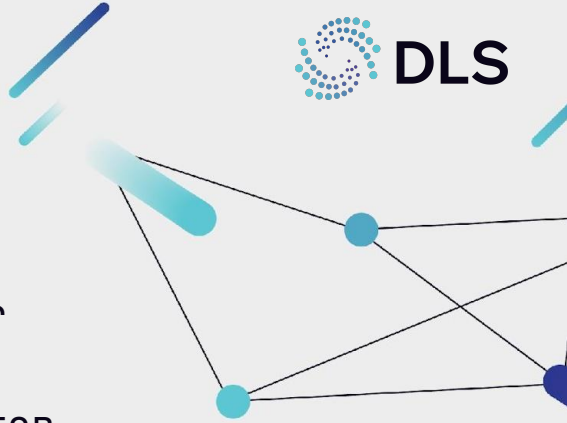
1. Вычисляем IoU между каждым GT и всеми якорями.
2. Для каждого GT выбираем все якоря, с которыми IoU больше порога (например 0.5). Если таких нет, выбираем якорь с максимальным IoU.
3. Отмечаем выбранные якоря как “положительные”.
   На основе предсказаний для “положительных” якорей считается локализационный лосс.
4. У “положительных” якорей проставляются истинные метки классов (такие же как у GT к которым они относятся), остальные якоря помечаются как “фон”.
5. Чтобы уравновесить и уменьшить дисбаланса между положительными (обекты) и отрицательными (фон) якорями, применяется процедура **hard negative mining**.

   В классификационном лоссе учитываются не все фоновые якори, а только самые плохие (с максимальной ошибкой классификации). Соотношение “отрицательных” (фон) и “положительных” якорей обычно 3:1.

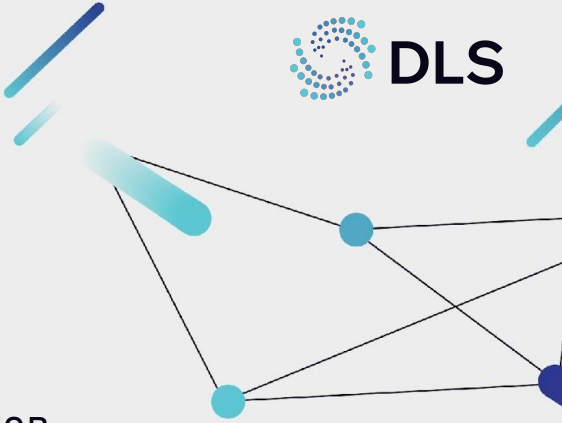
Лосс совпадает с Faster RCNN (LogLoss + SmoothL1).

SSD. Итоги.

- Одна из первых моделей, решающих задачу детекции одной моделью (single shot),
- Очень быстрый детектор (~60FPS Nvidia Titan X for 512x512),
- Использует multi-scale фичи для нахождения объектов разного размера,
- Также как и Faster R-CNN использует якоря,
- Сделали модель в двух вариациях:
  a. SSD 300 — модель, которую мы подробно рассмотрели,
  b. SSD 500 — вариация обученная на картинках 512x512.



SSD. Итоги.



- Одна из первых моделей, решающих задачу детекции одной моделью (single shot),
- Очень быстрый детектор (~60FPS Nvidia Titan X for 512x512),
- Использует multi-scale фичи для нахождения объектов разного размера,
- Также как и Faster R-CNN использует якоря,
- Сделали модель в двух вариациях:
  a. SSD 300 — модель, которую мы подробно рассмотрели,
  b. SSD 500 — вариация обученная на картинках 512x512.

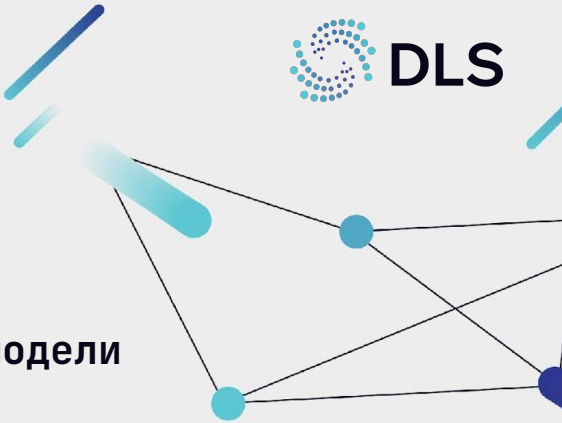
Метрики:

- PASCAL VOC2007: SSD512 **81.6 mAP**, SSD300 **79.6 mAP,**
- PASCAL VOC2012: SSD512 **80 mAP**, SSD300 **77.5 mAP,**
- COCO test-dev2015: SSD512 **26.8 mAP**, SSD300 **23.2 mAP**

Feature Pyramid Network

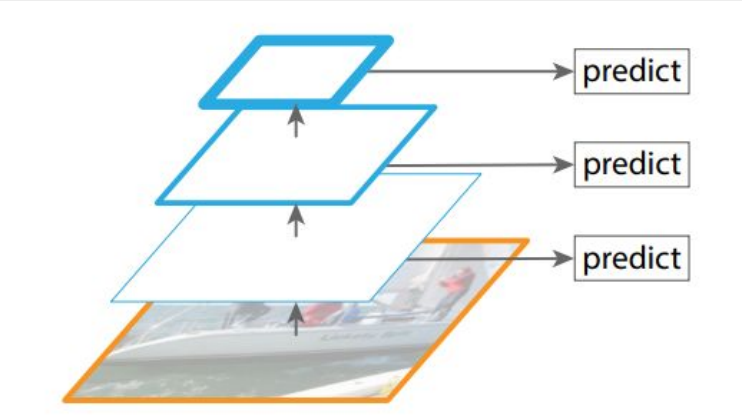
Модель FPN вышла 9го декабря 2016 года.

Главная особенность:
Объединение признаков (feature maps) разных с слоев модели



Feature Pyramid Network

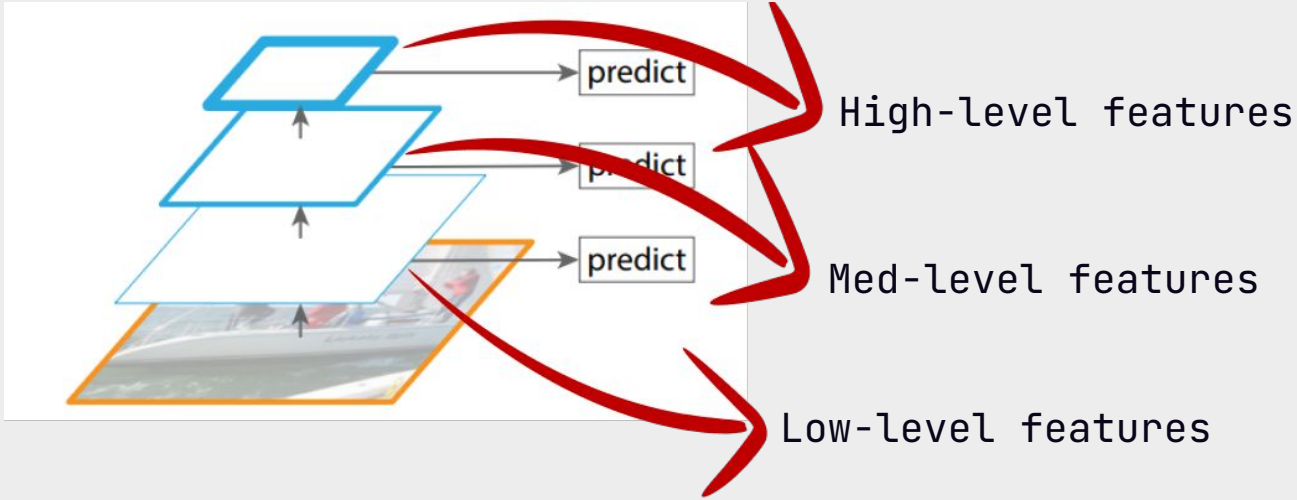
Схематичное представление SSD



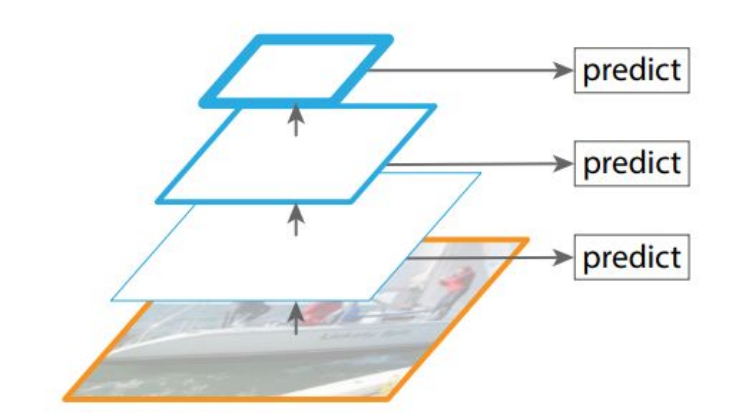
Source: Feature Pyramid Networks for Object Detection
https://doi.org/10.48550/arXiv.1612.03144

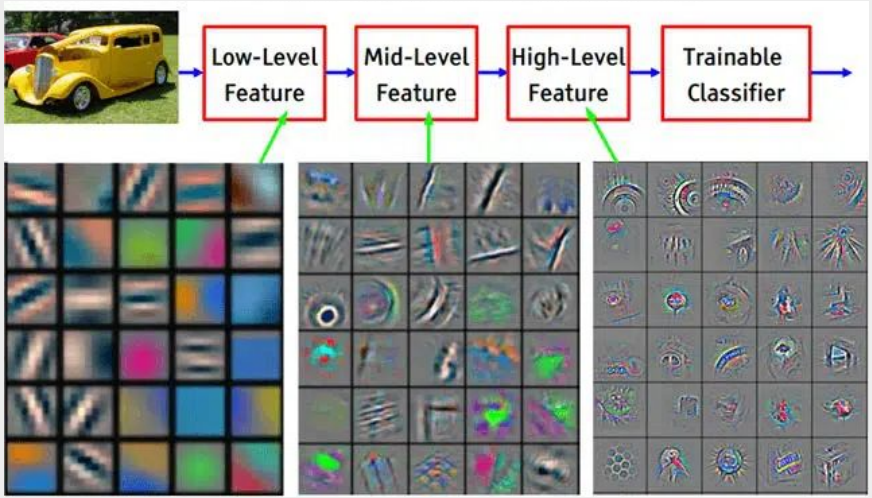
Feature Pyramid Network

Схематичное представление SSD

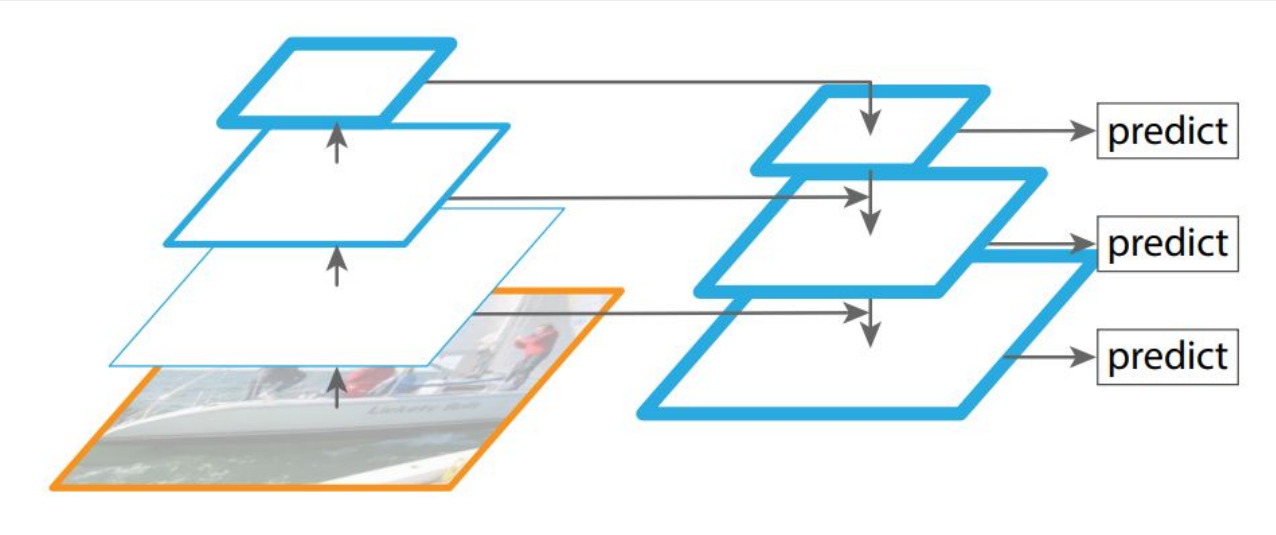


Feature Pyramid Network





Feature Pyramid Network



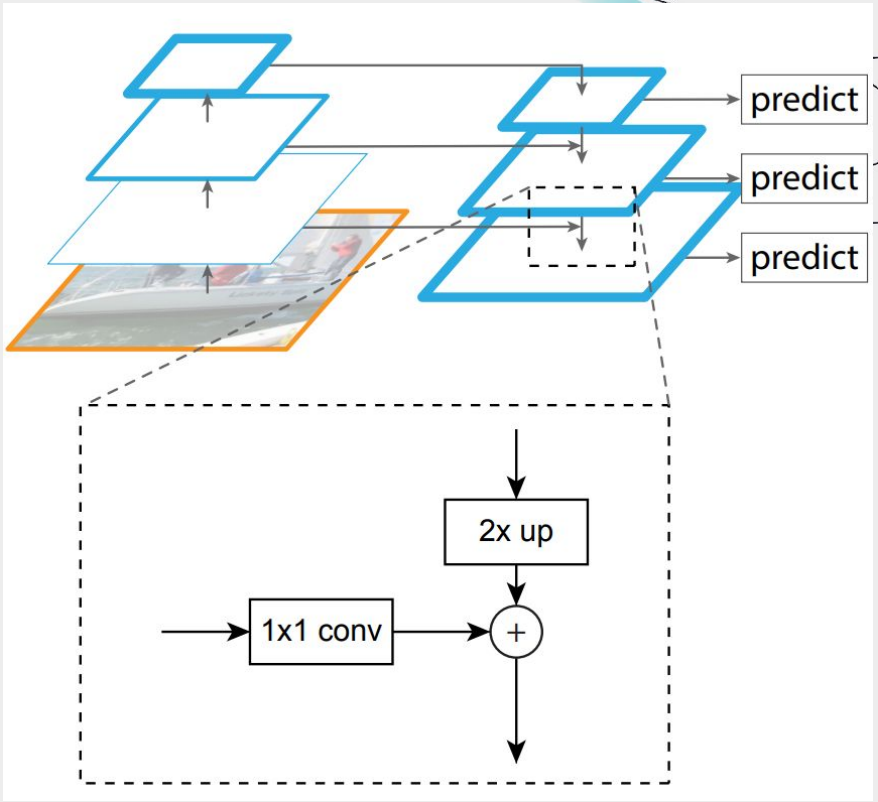
Продолжим идею SSD и добавим несколько слоев (heck / top-down pathway) в которых будем увеличивать размер feature map, а также добавлять фичи из исходной модели.

* Очень похоже на структуру Unet’а или любой Encoder-Decoder архитектуры.

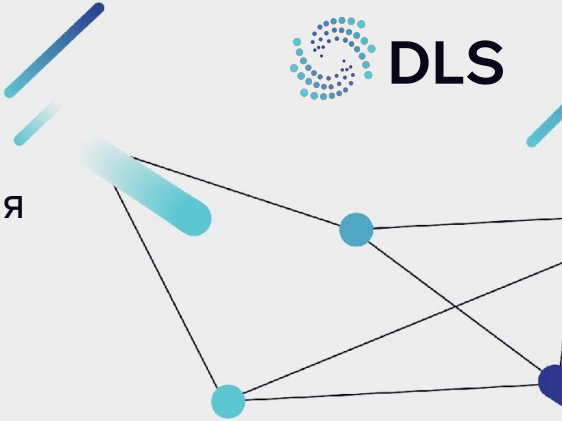
Feature Pyramid Network

Для увеличения размеров feature map используется простая интерполяция по ближайшим соседям
(Nearest neighbor upsampling).

Так как feature maps с разных уровней складываются поэлементно, во всей сети использовалось одинаковое количество каналов=256.



FPN. Применение.

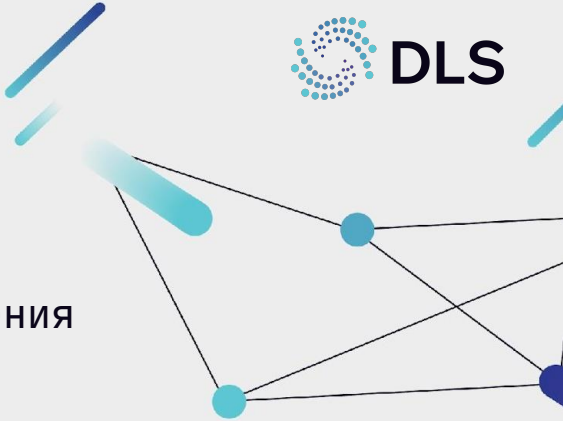


Главная цель статьи — показать преимущество добавления top-down pathway к архитектуре.

Поэтому авторы не придумывали новых подходов к предсказанию или применению якорей. Вместо этого они адаптировали Faster RCNN, заменив RPN на FPN.

Единственное что изменилось — размеры якорей на каждом выходе FPN:
В стандартном FPN 5 выходных feature maps разного размера, для каждого из которых был выбран базовый размер якоря: {32, 64, 128, 256, 512} пикселей + разные соотношения сторон. В итоге 15 различных якорей.

FPN. Итоги.

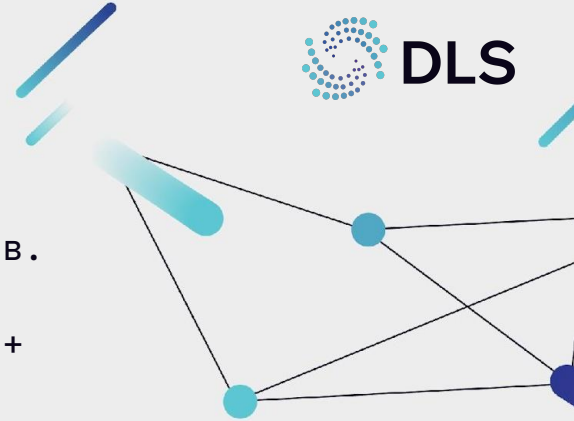


- Авторы показали широкий спектр возможностей для применения top-down pathway,
- Ещё лучше использует multi-scale фичи для нахождения объектов разного размера,
- Стал базовым подходом для построения архитектуры моделей для детекции на долгое время.

Метрики:

- COCO test-dev2015: **36.2 mAP**

FPN. Где использовался?



RetinaNet (2017):

- Использование **Focal Loss** для борьбы с дисбалансов классов.

DSSD (2017):

- Улучшение SSD посредствам использования PAN архитектуры + небольшое усложнение головы (head).

YOLOv3 (2018):

- Представитель самого большого семейство архитектур для детекции, которое насчитывает 11 моделей.

FCOS (2019):

- Детекционная модель, не использующая якоря.

VariFocalNet (2021):

- Использование **VariFocal Loss**, который учитывает предсказание ббокса при расчете классификационного лосса.

И многие другие, у статьи 31k цитированний.

FPN. Где использовался?

RetinaNet (2017):

- Использование **Focal Loss** для борьбы с дисбалансов классов.

DSSD (2017):

- Улучшение SSD посредствам использования PAN архитектуры + небольшое усложнение головы (head).

YOLOv3 (2018):

- Представитель самого большого семейство архитектур для детекции, которое насчитывает 11 моделей.

FCOS (2019):

- Детекционная модель, не использующая якоря.

VariFocalNet (2021):

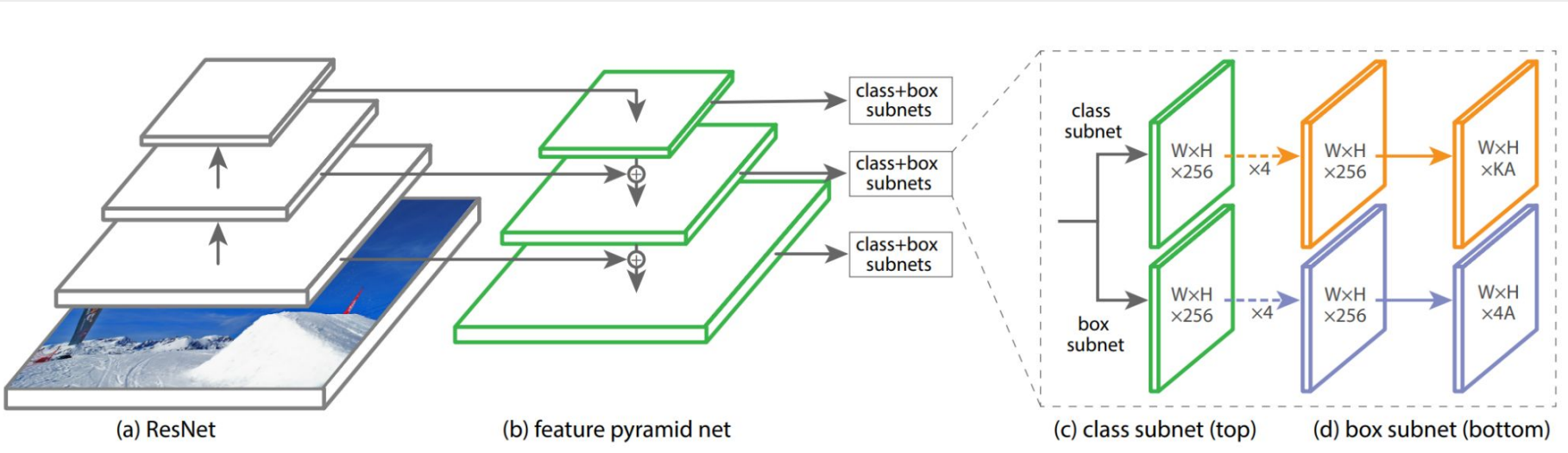
- Использование **VariFocal Loss**, который учитывает предсказание ббокса при расчете классификационного лосса.

Вдохновил:

- PANet, NAS-FPN, BiFPN, ... — улучшения подхода FPN, которые используются в современных моделях.

RetinaNet

Архитектура модели достаточна проста:
Backbode: ResNet,
Neck: FPN,
Head: Classification Subnet + Regression Subnet.



Source: Focal Loss for Dense Object Detection
https://doi.org/10.48550/arXiv.1708.02002

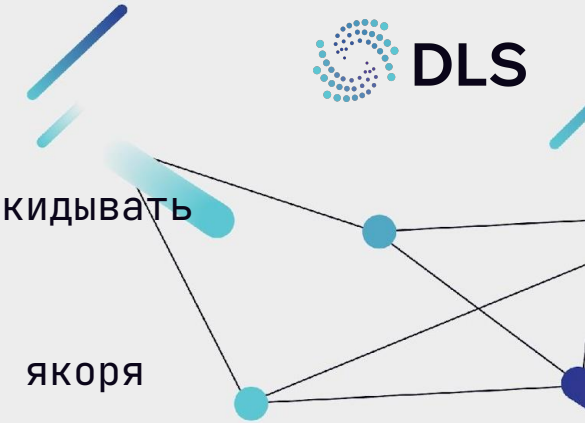
RetinaNet. Loss.

Давайте посмотрим как выглядит loss для RetinaNet:

$$L = \frac{1}{N_{\text{cls}}} \sum_i \text{FL}\left(\hat{p}_i, p_i\right) + \lambda \frac{1}{N_{\text{reg}}} \sum_i 1\left\{p_i = 1\right\} \text{SmoothL}\,1\left(\hat{t}_i - t_i\right)$$

$\hat{p}_i$ – Предсказанные вероятности для i-го якоря,
p_i – Истинные вероятности,
$\hat{t}_i$ – Предсказанные смещения для i-го якоря,
t_i – Истинные смещения,
N_{reg} и N_{cls} – нормализационные коэффициенты (обычно это количество положительных примеров).

RetinaNet. Итоги.

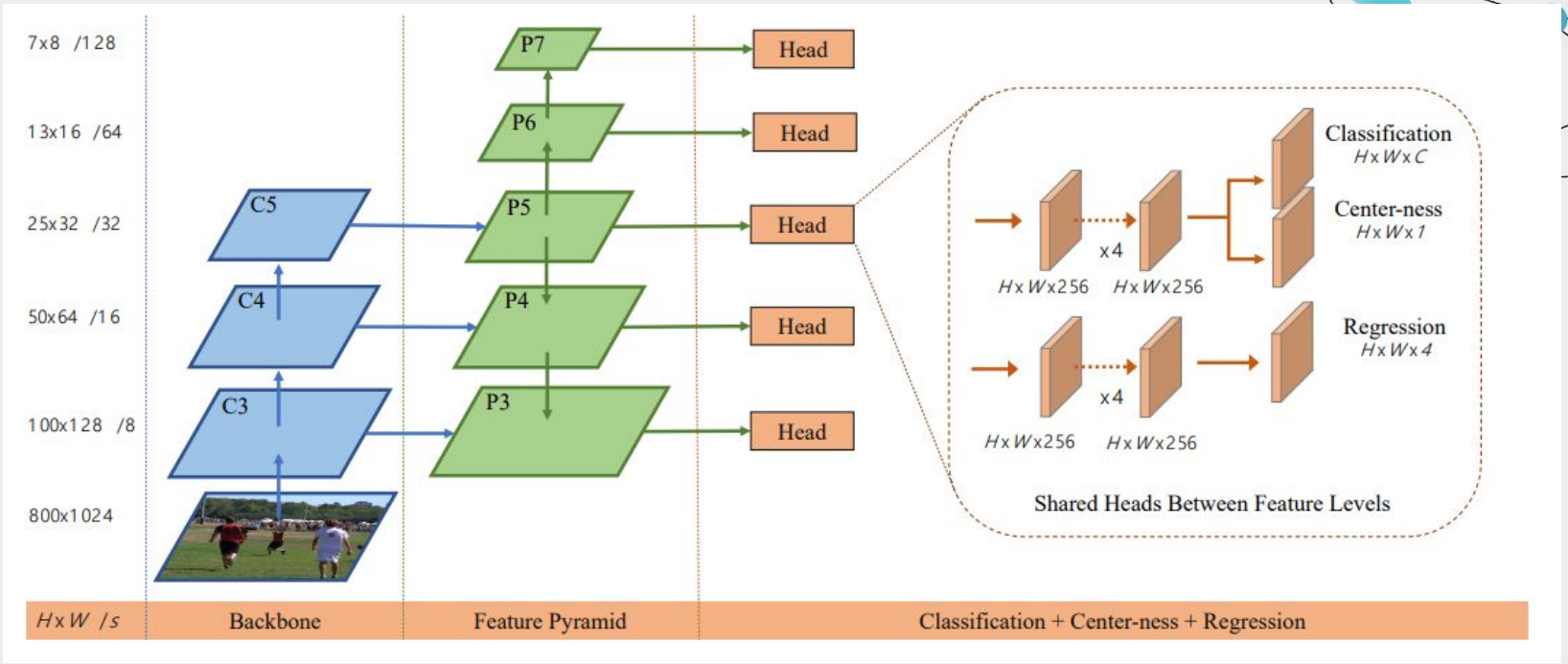


1. Использование Focal loss позволило авторам не выкидывать негативные предсказания, в отличии от SSD,
2. Пороги в Label assignment:
   - IoU(anchor, GT) ∈ [0, 0.4) - “отрицательные” якоря или “фон”
   - IoU(anchor, GT) ∈ [0.4, 0.5) - не участвуют в обучении,
   - IoU(anchor, GT) > 0.5 - участвуют в локализационном лоссе и имеют метку класса, совпадающую с GT.
3. Оптимальные константы для Focal loss это (α=0.25, γ=2).

Метрики:

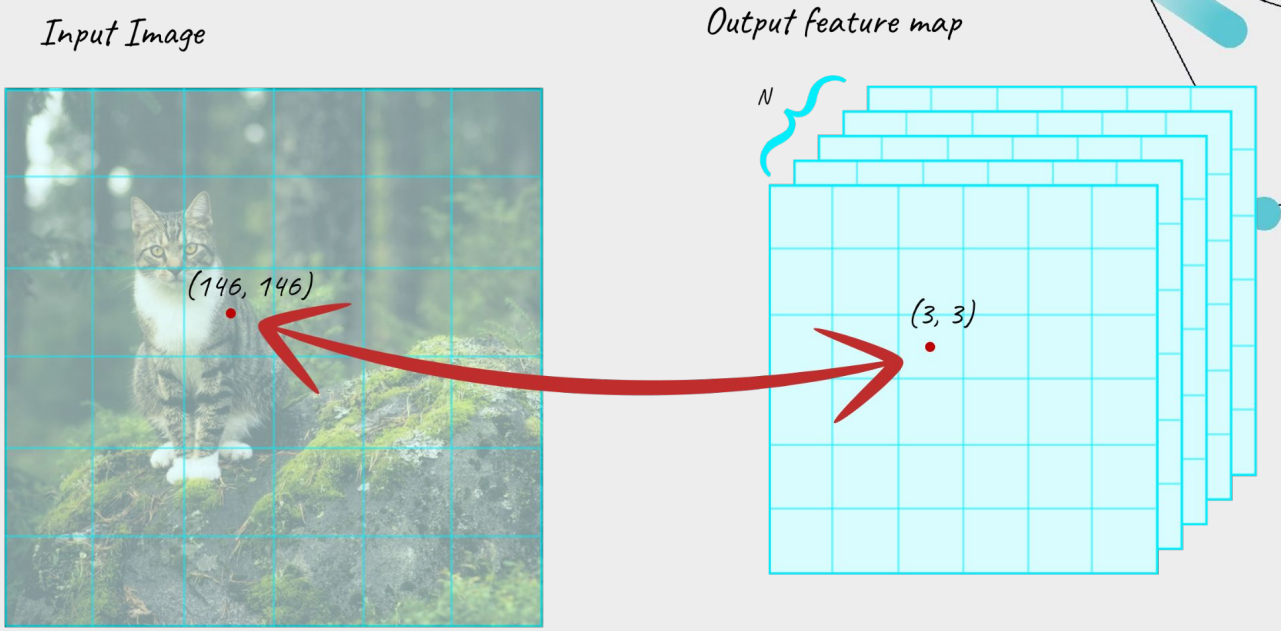
- COCO test-dev **40.8 mAP**

FCOS



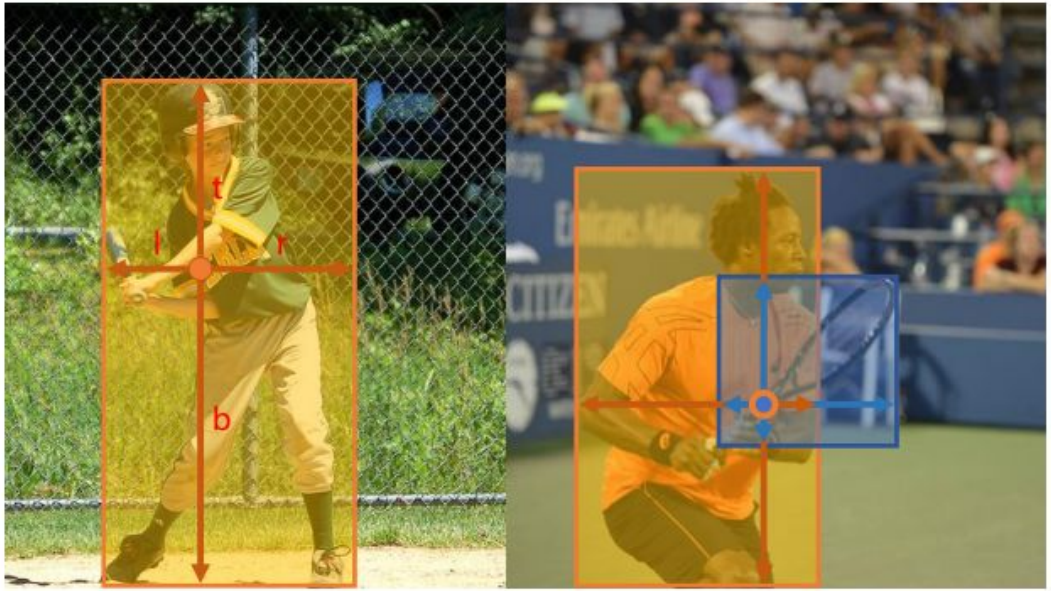
Source: FCOS: Fully Convolutional One-Stage Object Detection
https://arxiv.org/abs/1904.01355

FCOS



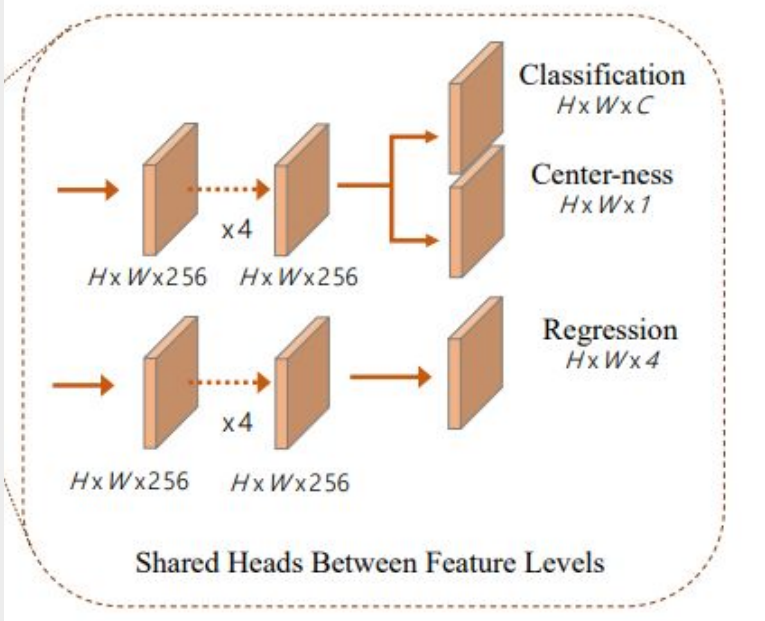
Каждая точка (x, y) в выходном векторе фичей имеет соответствие на входном изображении.

FCOS. Output.



Пример предсказаний из статьи

FCOS. Output.



Для точки (x, y) на feature map’е предсказывается:

Classification:
N+1 вероятность принадлежности к каждому классу в датасете + фон.

Center-ness:
Уверенность что центр объекта расположен внутри этого “пикселя” feature map’а.

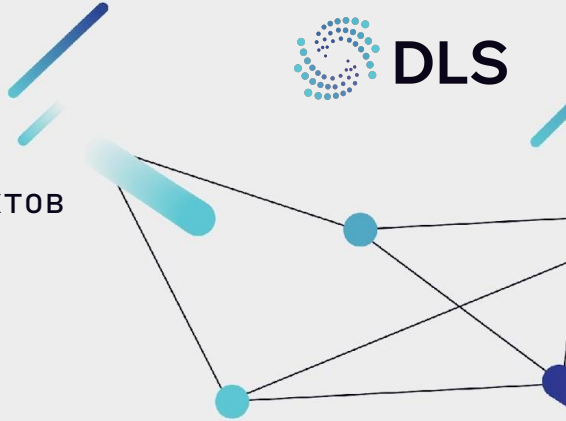
Regression:
4 отступа от точки (x, y) до:

- l - левой,
- t - верхней,
- r - правой,
- b - нижней границ объекта.

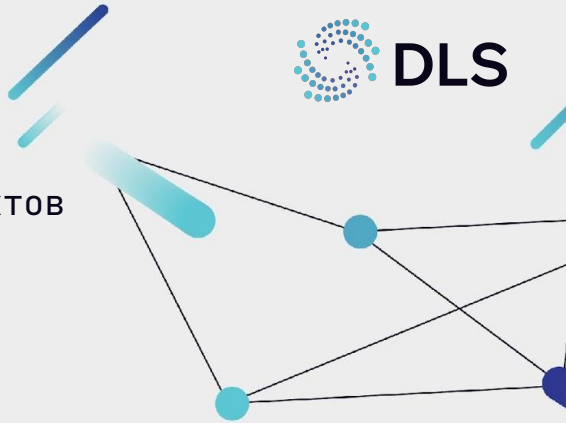
FCOS. Label assignment.

Каждый уровень FPN (P3-P7) предназначен для обнаружения объектов разного размера:

- P3 – нацелен на маленькие объекты,
- ...
- P7 – на самые больше.



FCOS. Label assignment.



Каждый уровень FPN (P3-P7) предназначен для обнаружения объектов разного размера:

- P3 – нацелен на маленькие объекты,
- ...
- P7 – на самые больше.

Поэтому, для каждого уровня, авторы задают диапазон размера объектов, который может предсказать каждый слой:

- P3 – [0, 64),
- P4 – [64, 128),
- И тд.

Под размером объекта подразумевается **максимальная сторона объекта.**

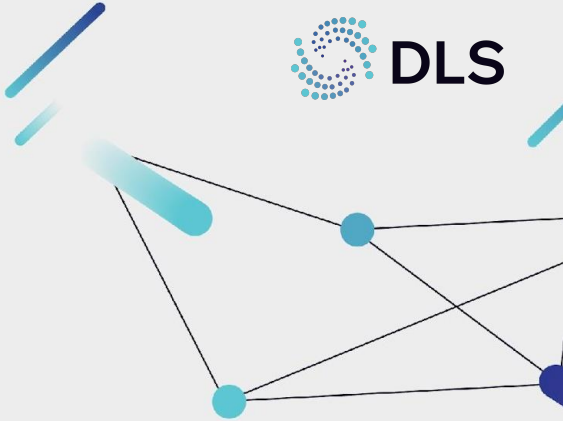
Это сделано для того, чтобы каждый уровень модели учился определять объекты определенного размера.

* В каком то смысле это похоже на anchor-based модели, где для разных уровней задаются разные размеры якорей.

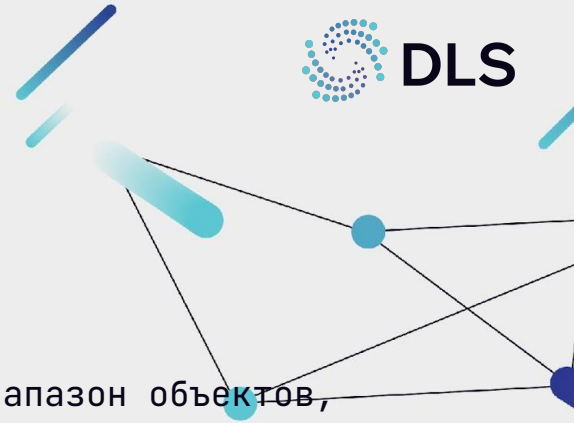
FCOS. Label assignment.

В итоге процесс label assignment выглядит следующим образом:

1. Для выбранного GT вычисляем расстояния (l, t, r, b) относительно точек на feature map’e, которые находятся внутри GT,



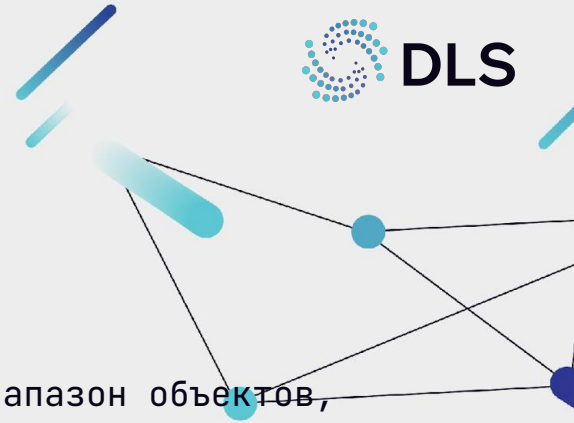
FCOS. Label assignment.



В итоге процесс label assignment выглядит следующим образом:

1. Для выбранного GT вычисляем расстояния (l, t, r, b) относительно точек на feature map’e, которые находятся внутри GT,
2. Назначаем GT уровень, если max(l, t, r, b) попадает в диапазон объектов, которые может регрессировать данный уровень. Если условие выполняется для нескольких уровней, тогда выбираем тот:
    a. У которого меньше расстояние до центра GT,
    b. Или где меньше значение max(l, t, r, b).

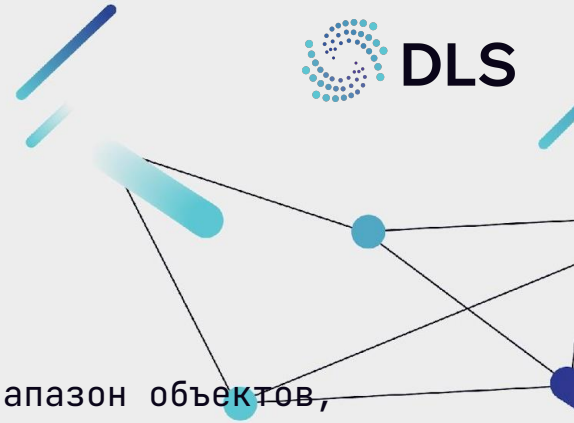
FCOS. Label assignment.



В итоге процесс label assignment выглядит следующим образом:

1. Для выбранного GT вычисляем расстояния (l, t, r, b) относительно точек на feature map’e, которые находятся внутри GT,
2. Назначаем GT уровень, если max(l, t, r, b) попадает в диапазон объектов, которые может регрессировать данный уровень. Если условие выполняется для нескольких уровней, тогда выбираем тот:
   a. У которого меньше расстояние до центра GT,
   b. Или где меньше значение max(l, t, r, b).
3. Все пиксели этого слоя, внутри GT считаются **положительными**. Для них будет считаться регрессионный и классификационный лосс,

FCOS. Label assignment.



В итоге процесс label assignment выглядит следующим образом:

1. Для выбранного GT вычисляем расстояния (l, t, r, b) относительно точек на feature map'e, которые находятся внутри GT,
2. Назначаем GT уровень, если max(l, t, r, b) попадает в диапазон объектов, которые может регрессировать данный уровень. Если условие выполняется для нескольких уровней, тогда выбираем тот:
   a. У которого меньше расстояние до центра GT,
   b. Или где меньше значение max(l, t, r, b).
3. Все пиксели этого слоя, внутри GT считаются **положительными**. Для них будет считаться регрессионный и классификационный лосс,
4. Пиксели, которые не относятся к какому-либо GT, считаются **фоном**. Они участвуют только в классификационном лоссе, но не в регрессионном,

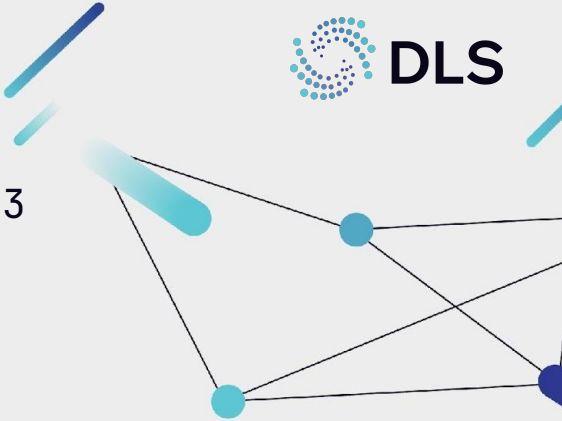
FCOS. Label assignment.

В итоге процесс label assignment выглядит следующим образом:

1. Для выбранного GT вычисляем расстояния (l, t, r, b) относительно точек на feature map'е, которые находятся внутри GT,
2. Назначаем GT уровень, если max(l, t, r, b) попадает в диапазон объектов, которые может регрессировать данный уровень. Если условие выполняется для нескольких уровней, тогда выбираем тот:
   a. У которого меньше расстояние до центра GT,
   b. Или где меньше значение max(l, t, r, b).
3. Все пиксели этого слоя, внутри GT считаются **положительными**. Для них будет считаться регрессионный и классификационный лосс,
4. Пиксели, которые не относятся к какому-либо GT, считаются **фоном**. Они участвуют только в классификационном лоссе,
5. Centerness score для GT считается как соотношение расстояний:

$$c = \sqrt{\frac{\min(l, r) \cdot \min(t, b)}{\max(l, r) \cdot \max(t, b)}}$$

FCOS. Loss.

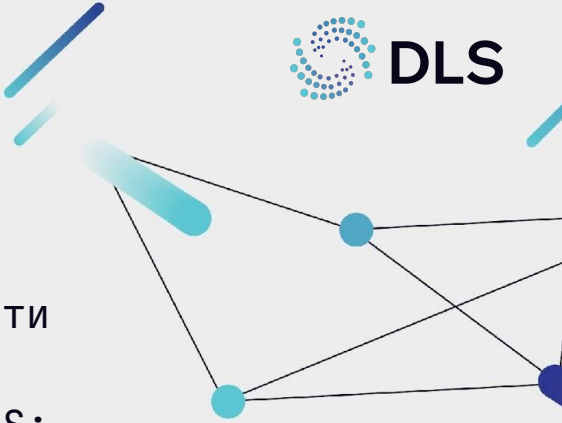


В отличии от ранее рассмотренных моделей, FCOS имеет 3 выхода, поэтому составляющих лосса тоже 3:

$$L = L_{cls} + L_{reg} + L_{centerness}$$

L_{cls} – Focal Loss,
L_{reg} – IoU Loss, $L_{reg} = 1 - IoU(\hat{t}, t)$
$L_{centerness}$ – Binary Cross-Entropy.

FCOS. Итоги.

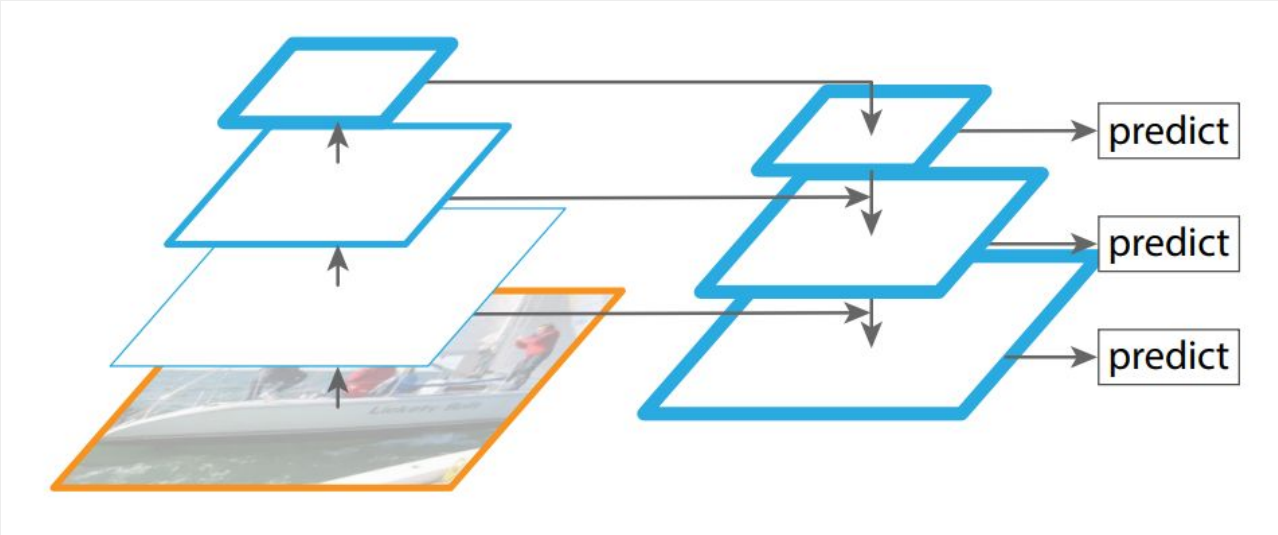


1. Авторы доказали что использование якорей не обязательно в задаче детекции,
2. Гибкая стратегия Label Assignment без необходимости подбора anchor-ов,
3. Благодаря centerness score, улучшается процесс NMS:
   - Перемножаем Centerness score с вероятностями классов, уменьшая ложные предсказания на границах объектов
4. Разные уровни модели позволяют корректно обнаруживать объекты разных размеров, даже если их центры совпадают.

Метрики:

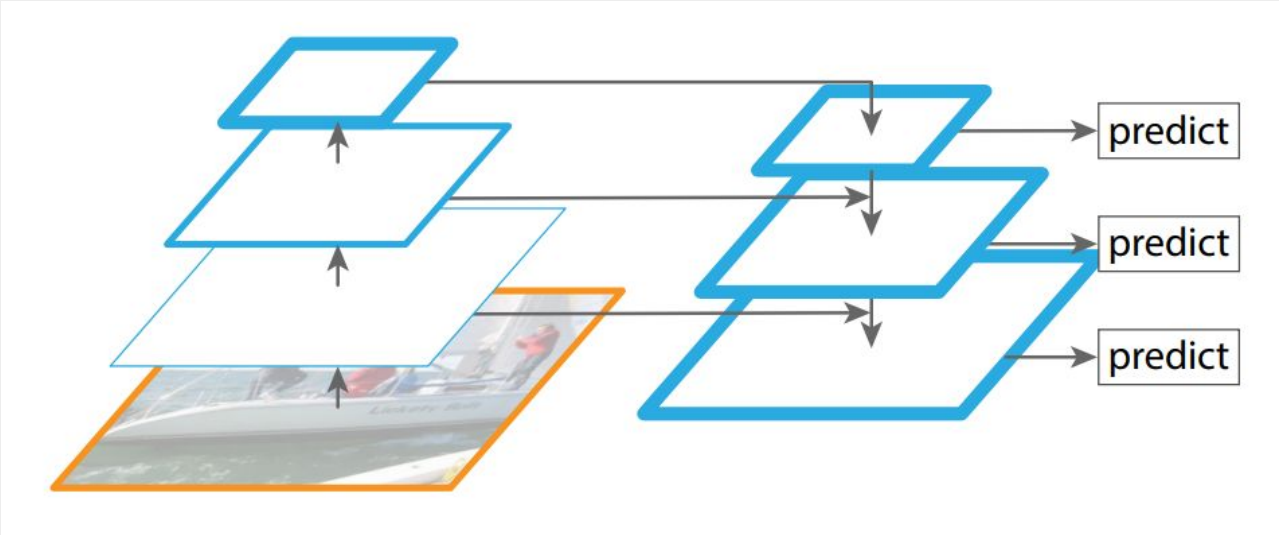
- COCO test-dev **44.7 mAP**

Path Aggregation Network



Посмотрим на архитектуру FPN ещё раз и попробуем понять какие у неё есть недостатки?

Path Aggregation Network



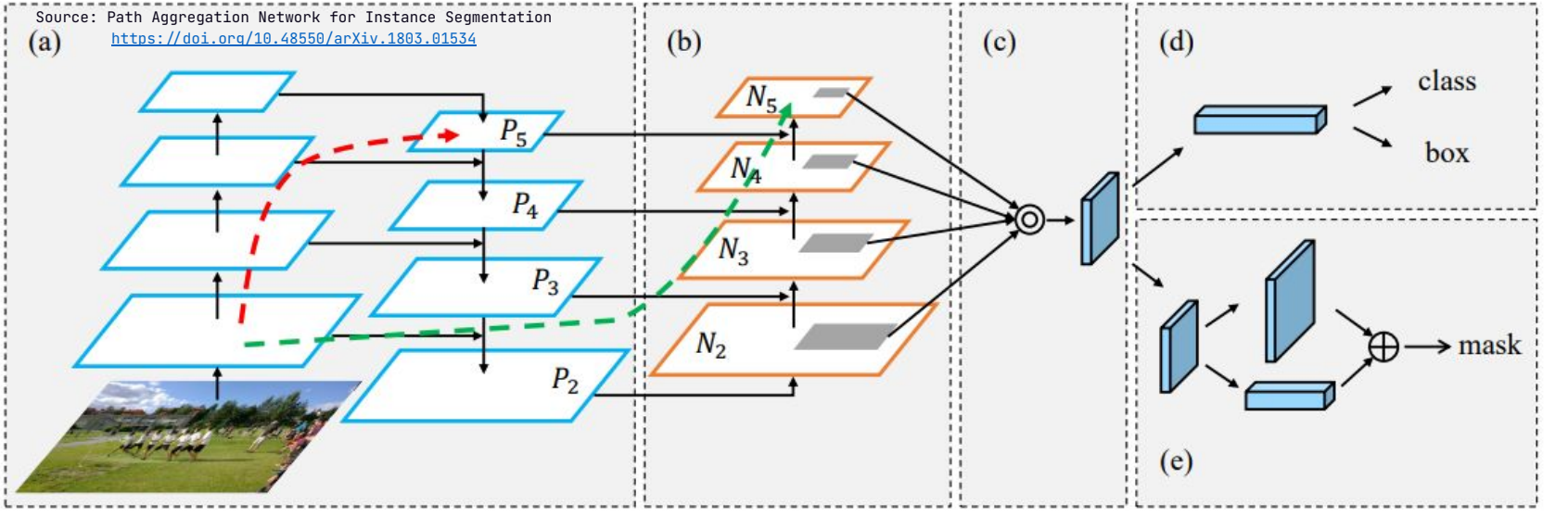
Последний выход получает получает разнообразный набор фичей:

- “Простые” фичи с начальных слоев,
- “Сложные” фичи, которые получились после прохождения всех слоев.

Однако самый первый выход не будет обогащен информацией с ранних слоев.

* В литературе это называется “one-way information flow”

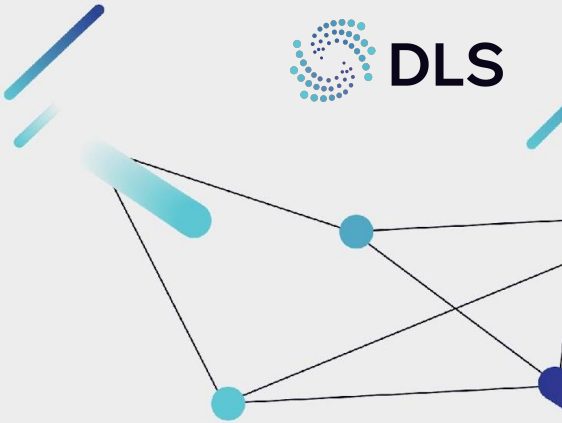
Path Aggregation Network



Авторы решают проблему добавлением ещё одного потока обогащения признаков.

Так, путь до выходного слоя с самой маленькой пространственной размерностью (в данном случае N5) значительно сокращается.

PAN. Итоги.



1. Улучшив популярную архитектуру FPN, позволили ещё больше повысить качество детекторов.

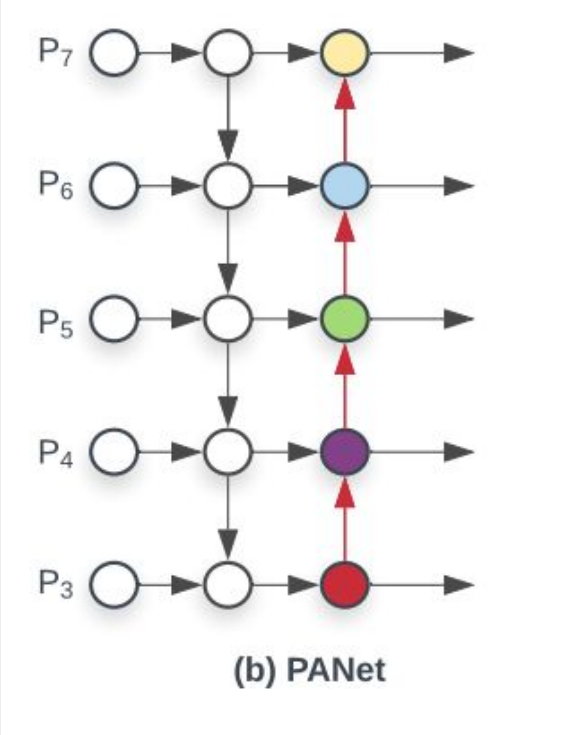
Метрики:

- COCO test-dev **42.0 mAP**

Модели, где используется PAN:

- YOLOv4 и дальше,
- Легла в основу EfficientDet.

BiFPN

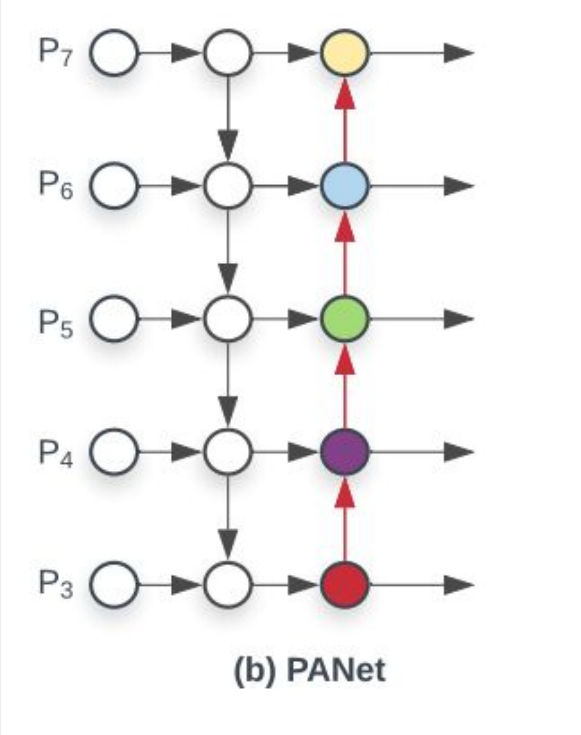


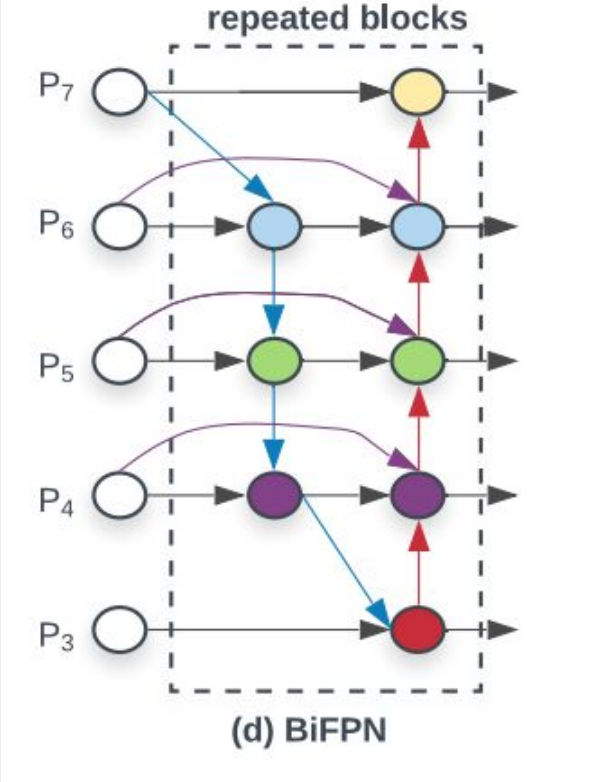
Source: EfficientDet: Scalable and Efficient Object Detection
https://doi.org/10.48550/arXiv.1911.09070

Давайте ещё раз посмотрим на архитектуру PAN.

Что мы можем добавить в неё, чтобы сделать модель ещё лучше?

BiFPN

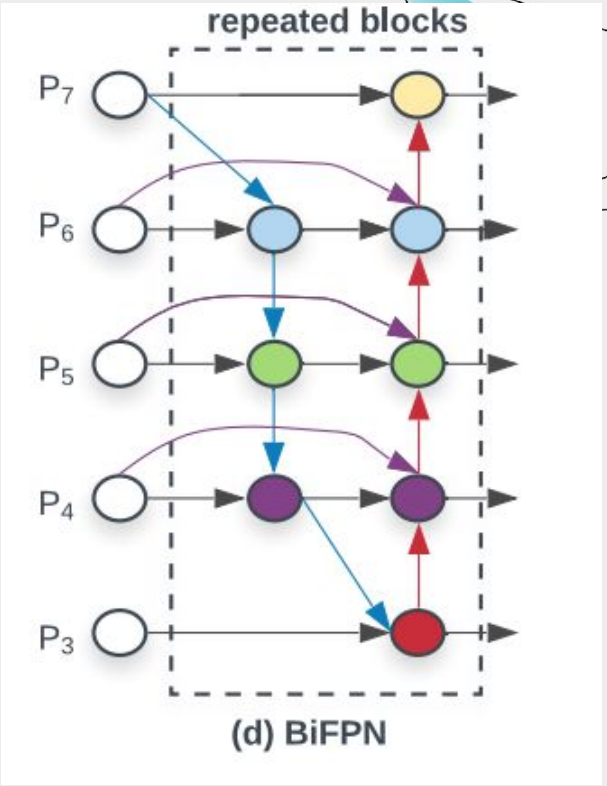




Добавим больше скипконекшеннов!

BiFPN. Детали.

1. Для увеличения пространственной размерности используется билинейная интерполяция,
2. Для уменьшения обычно применяется свертка со страйдом 2.



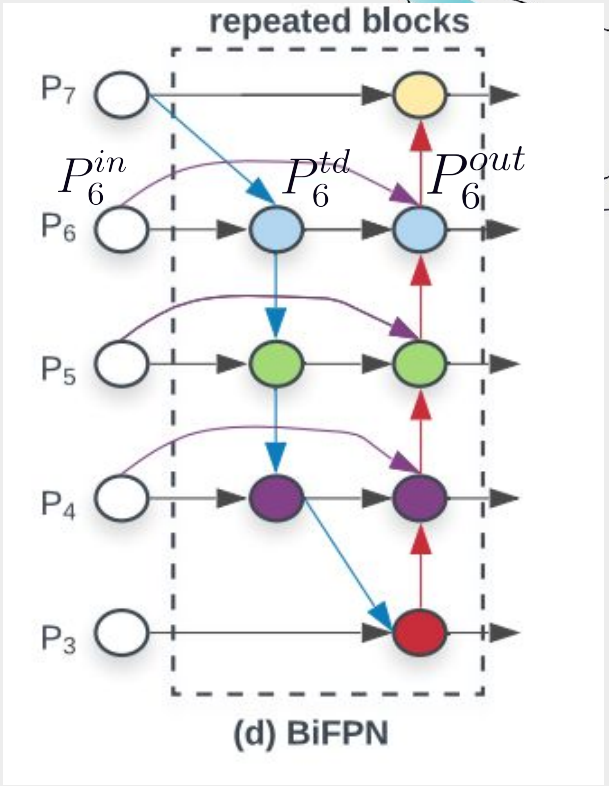
BiFPN. Детали.

1. Для увеличения пространственной размерности используется билинейная интерполяция,
2. Для уменьшения обычно применяется свертка со страйдом 2.
3. Вместо суммирования фичей с разных слоев, используется более сложная структура с обучаемыми параметрами:

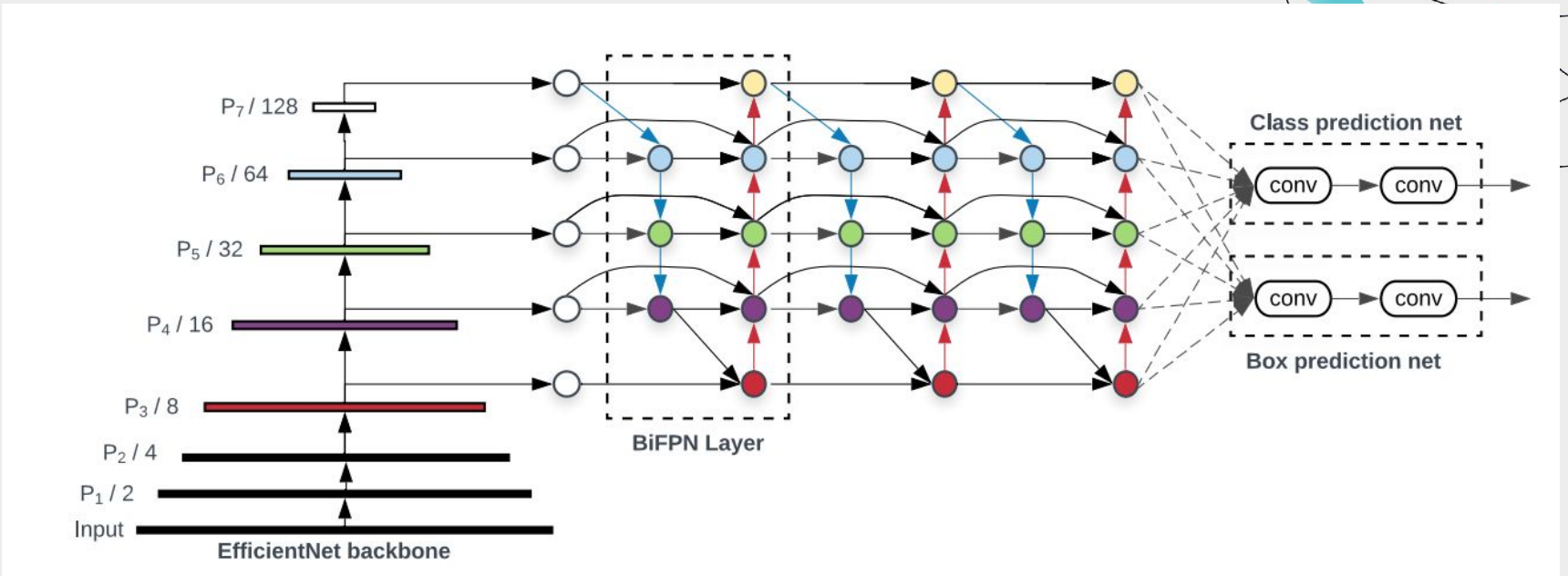
$$P_6^{td} = Conv\left(\frac{w_1 \cdot P_6^{in} + w_2 \cdot Resize(P_7^{in})}{w_1 + w_2 + \epsilon}\right)$$

$$P_6^{out} = Conv\left(\frac{w_1' \cdot P_6^{in} + w_2' \cdot P_6^{td} + w_3' \cdot Resize(P_5^{out})}{w_1' + w_2' + w_3' + \epsilon}\right)$$

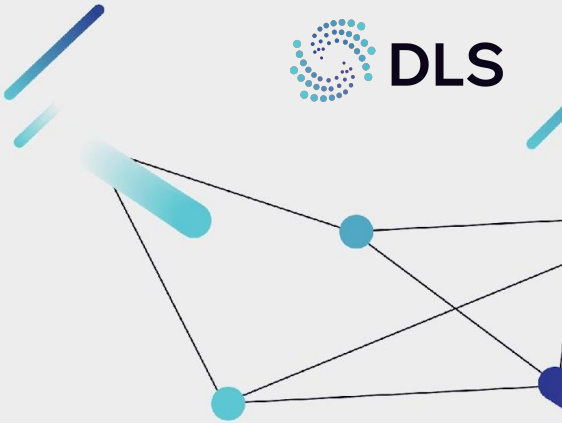
При этом, к каждому весу w применяется ReLU, поэтому гарантируется что $w \geqslant 0$



EfficientDet



EfficientDet. Compound Scaling.



Так же как EfficientNet, детекционная модель имеет свой подход к масштабированию, позволяющий эффективно применять EfficientDet на разных задачах.

Метод масштабирования позволяет одновременно увеличивать размер входного изображения, ширину и глубину сети (число каналов и количество BiFPN блоков). Для этого вводится коэффициент масштабирования ψ (phi).

EfficientDet. Compound Scaling.

Метод масштабирования позволяет одновременно увеличивать размер входного изображения, ширину и глубину сети (число каналов и количество BiFPN блоков). Для этого вводится коэффициент масштабирования ψ (phi).

	Input size R_{input}	Backbone Network	BiFPN #channels W_{bifpn}	BiFPN #layers D_{bifpn}	Box/class #layers D_{class}
D0 ($\phi = 0$)	512	B0	64	3	3
D1 ($\phi = 1$)	640	B1	88	4	3
D2 ($\phi = 2$)	768	B2	112	5	3
D3 ($\phi = 3$)	896	B3	160	6	4
D4 ($\phi = 4$)	1024	B4	224	7	4
D5 ($\phi = 5$)	1280	B5	288	7	4
D6 ($\phi = 6$)	1280	B6	384	8	5
D7 ($\phi = 7$)	1536	B6	384	8	5
D7x	1536	B7	384	8	5

EfficientDet. Итоги.

Model	test-dev AP	AP_{50}	AP_{75}
EfficientDet-D0 (512)	**34.6**	**53.0**	**37.1**
YOLOv3 [34]	33.0	57.9	34.4
EfficientDet-D1 (640)	**40.5**	**59.1**	**43.7**
RetinaNet-R50 (640) [24]	39.2	58.0	42.3
RetinaNet-R101 (640)[24]	39.9	58.5	43.0
EfficientDet-D2 (768)	**43.9**	**62.7**	**47.6**
Detectron2 Mask R-CNN R101-FPN [1]	-	-	-
Detectron2 Mask R-CNN X101-FPN [1]	-	-	-
EfficientDet-D3 (896)	**47.2**	**65.9**	**51.2**
ResNet-50 + NAS-FPN (1024) [10]	44.2	-	-
ResNet-50 + NAS-FPN (1280) [10]	44.8	-	-
ResNet-50 + NAS-FPN (1280@384)[10]	45.4	-	-
EfficientDet-D4 (1024)	**49.7**	**68.4**	**53.9**
AmoebaNet+ NAS-FPN +AA(1280)[45]	-	-	-
EfficientDet-D5 (1280)	**51.5**	**70.5**	**56.1**
Detectron2 Mask R-CNN X152 [1]	-	-	-
EfficientDet-D6 (1280)	**52.6**	**71.5**	**57.2**
AmoebaNet+ NAS-FPN +AA(1536)[45]	-	-	-
EfficientDet-D7 (1536)	**53.7**	**72.4**	**58.4**
EfficientDet-D7x (1536)	**55.1**	**74.3**	**59.9**

1. BiFPN обеспечивает более эффективное и адаптивное объединение признаков разных масштабов,
2. За счет метода масштабирования, можно создавать модели с оптимальным соотношением скорости и точности.
3. Модель показала хорошую точность на COCO **55.1 mAP** у самой большой модели.

Summary

Давайте повторим какие архитектуры мы сегодня разобрали:

R-CNN	Один из самых первых детекторов, который состоял из нескольких этапов: selective search + CNN + SVM.
Fast R-CNN	Упрощение и ускорение R-CNN за счет введения RoI pooling.
Faster R-CNN	End-to-end архитектура, одна из первых использовала “якоря” для детекции
Cascade R-CNN	Улучшения Faster R-CNN и введение многоступенчатого подхода.
YOLO	Очень простой и быстрый детектор.
SSD	Находим объекты разного размера используя много выходов с разных слоев.
FPN	Добавление top-down пути для смешения признаков разного размера.
RetinaNet	Использование Focal Loss для борьбы с дисбалансов классов.
FCOS	Детекционная модель, не использующая якоря.
PAN	Улучшение FPN за счет добавление bottom-up пути, который равномерно распределил обогащение информации внутри сети
EfficientDet	Улучшение PAN за счет оптимизации, а также использование моделей разного размера.

В следующей серии.

1. Семейство YOLO:
    a. Какое влияние оказало семейство на развитие детекторов,
    b. Новые подходы для детекции.
2. Необычные подходы к задаче детекции:
    a. CenterNet,
    b. CornerNet,
    c. Использование поворотных bounding boxes.