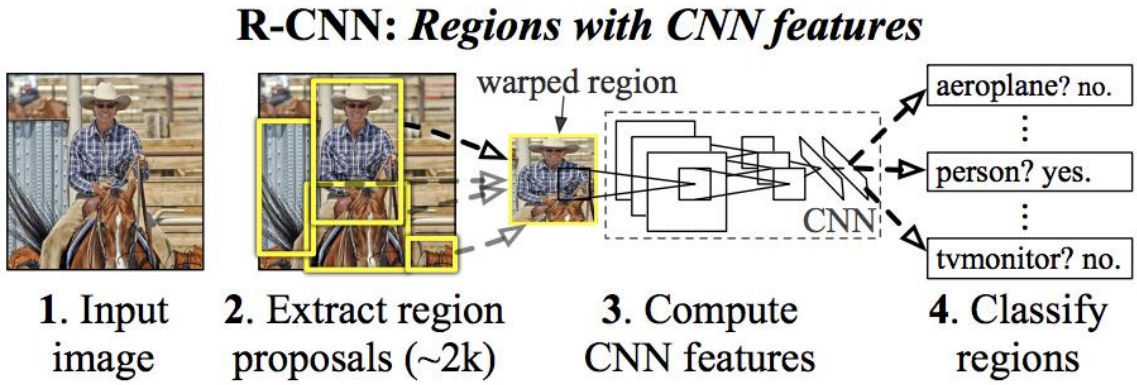
R-CNN

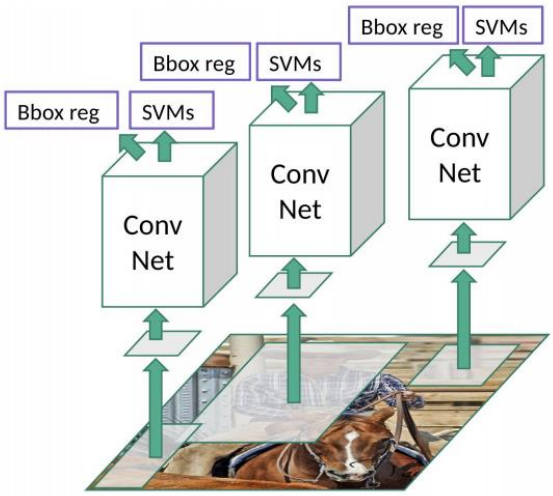
Чтобы обойти проблему выбора огромного количества регионов, Росс Гиршик и др. предложили метод, при котором мы используем выборочный поиск для извлечения из снимка только 2000 регионов, и он назвал их region proposals. Поэтому теперь, вместо того, чтобы пытаться классифицировать огромное количество регионов, можно просто работать с 2000 регионами. Эти 2000 region proposals сгенерированы с использованием алгоритма селективного поиска, который описан ниже.

Selective Search:

1. Generate initial sub-segmentation, we generate many candidate regions
2. Use greedy algorithm to recursively combine similar regions into larger ones
3. Use the generated regions to produce the final candidate region proposals



Как показано на картинке, мы проходим не по всей, а только по конкретным регионам, которых около 2к. Также присутствуют специальные offsets на случай, если не весь класс влезет в границы нашего region proposal. То есть мы немного расширяем bounding box в region proposal если в изначальный влезло только пол-лица (называется bbox reg).

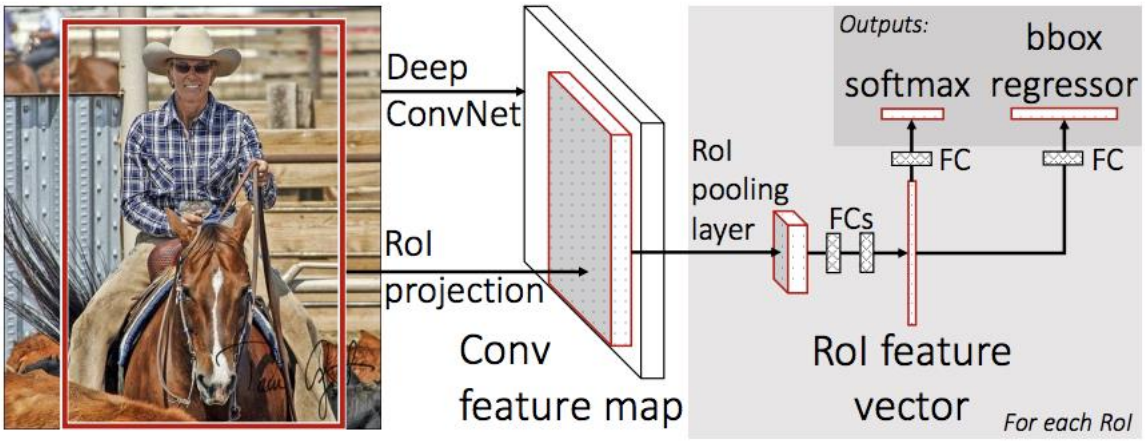


Однако данная сеть очень медленная, обработка 2к регионов это полный кошмар, при том что мы каждый раз должны сначала читать регион (а это тоже картинка, только ее часть), потом делать саму сеть (набор сверток) и повторять этот процесс 2к раз. Поэтому придумали новую сетку.

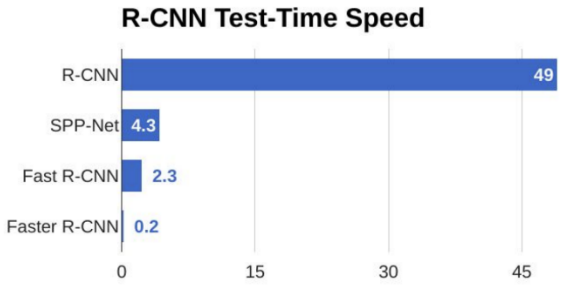
Fast R-CNN

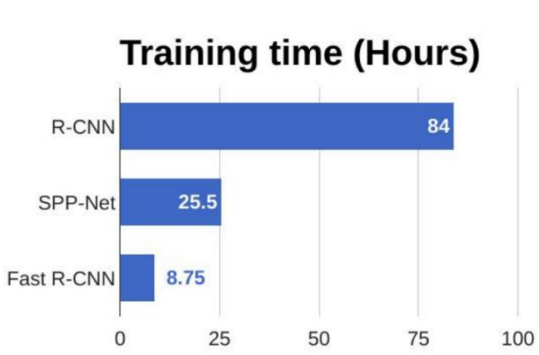
Как не читать картинки 2к раз? Давайте скормим картинку CNN и она выдаст нам convolutional feature map. С этой карты мы выделяем регионы и прогоняем через слой RoI (Region of interest) пулинга, и только дальше уже идет полносвязный слой, выдающий нам вектор региона (у нас может быть много классов на картинке, соответственно один регион – один класс, как для coco датасета), из этого вектора с помощью софтмакса мы определяем к какому классу принадлежит регион + те же самые оффсеты на всякий случай.

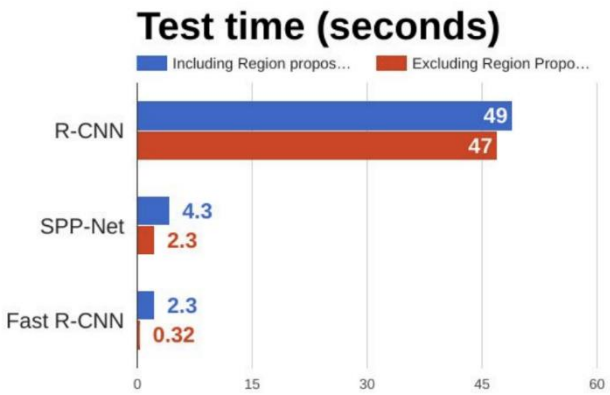
***Заметка**: softmax предполагает эксклюзивность классов, то есть это либо птичка либо корабль, два класса быть не могут в одном регионе. Но Yolo использует сигмоиду, в предположении, что могут быть 2 класса сразу (например «человек» и «женщина»).*



Ну почему быстрее? Потому что мы уже не читаем регионы картинок 2к раз, а читаем какую-то матрицу, содержащую инфу о регионах, это намного-намного быстрее. В итоге сама картинка в архитектуре читается только 1 раз.

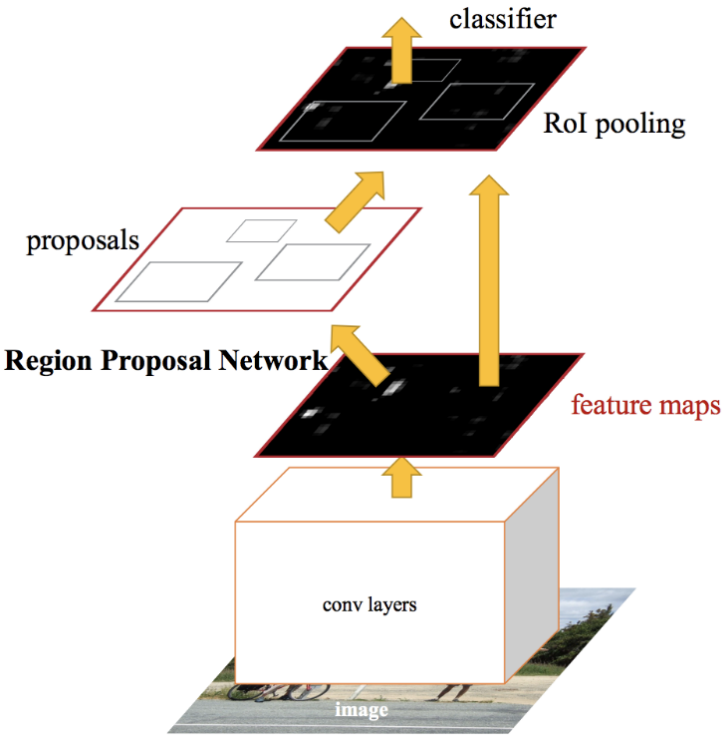






Faster R-CNN (вообще дичь че придумали)

Предыдущие архитектуры использовали selective search чтобы находить регионы, а он тоже медленный. Поэтому и его решили заменить на сетку 🙂



Как видите здесь сетка в сетке. Сначала мы скармливаем картинку CNN, которая выдает нам feature map. Далее вместо селективного поиска для регионов используется отдельная сеть, которая предсказывает правильные регионы (region proposal network

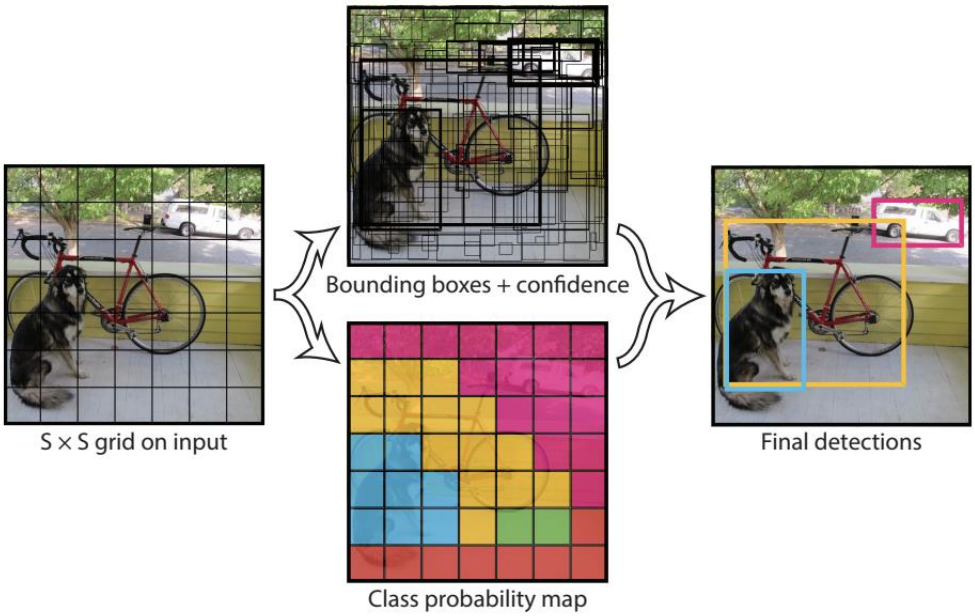
(RPN)). Далее снова пулинг RoI, вектор конкретного региона и предсказание класса + оффсеты.

Есть только один вопрос. Как обучать RPN и основную сеть? Можно по отдельности, можно одновременно, здесь пути расходятся. В основной статье предлагается поочередной подход:

1. train the RPN, then freeze RPN layers,
2. train RCNN, then freeze RCNN layers,
3. train RPN, then freeze RPN layers
4. train RCNN.

Yolo

Все предыдущие алгоритмы обнаружения объектов используют регионы для локализации объекта на изображении. Сеть не смотрит на полное изображение. Вместо этого, используются части изображения, которые имеют высокую вероятность содержания объекта. YOLO - это алгоритм обнаружения объектов, значительно отличающийся от вышеописанных. В YOLO одна сеть предсказывает bbox-ы и классовые вероятности для них.

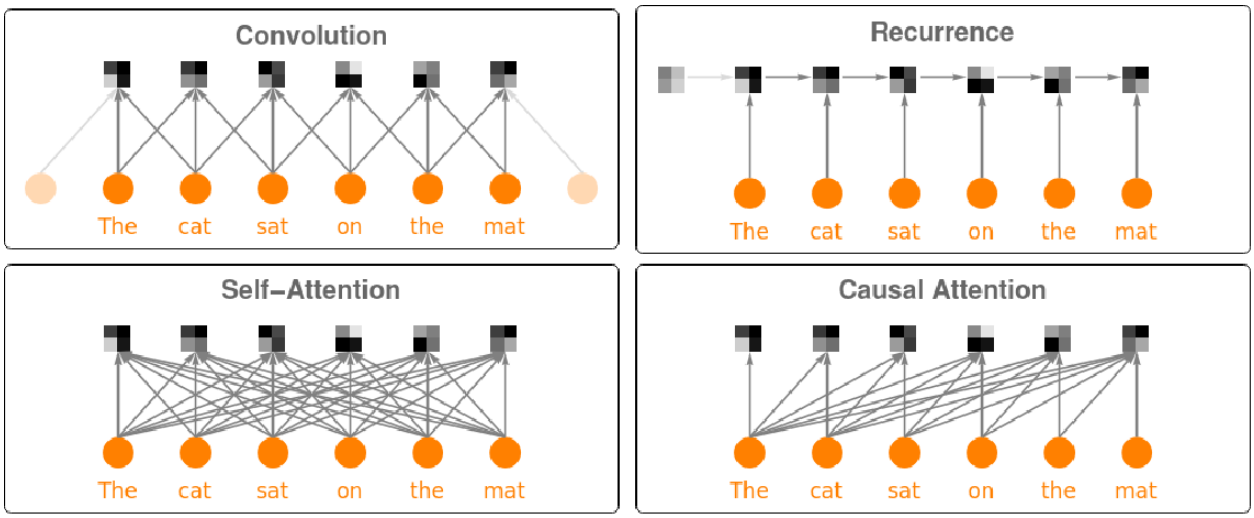


Как работает YOLO, мы берем изображение и разделяем его на сетку (grid) SxS, внутри каждой части мы берем m bbox-ов. Для каждого box-а сеть выводит вероятность класса и оффсет. Bbox-ы с вероятностью класса выше порогового значения выбираются и используются для определения местоположения объекта на изображении.

Логично вытекающая проблема – невозможность детектировать маленькие объекты.

DETR

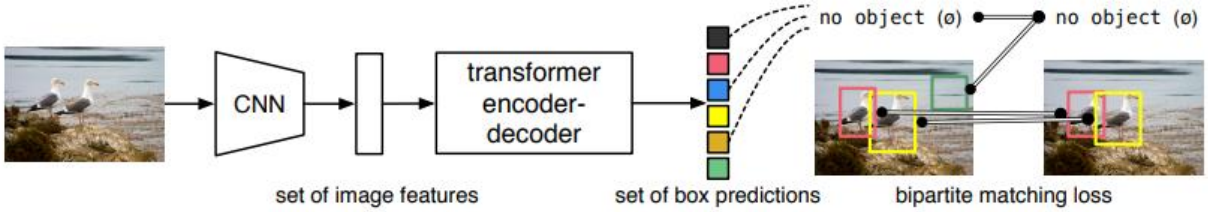
Сначала про трансформеры. В основном они используются для задач машинного перевода и основаны на том, что мы переводим следующее слово не только смотря на предыдущее, но смотря на все слова в предложении. Такой слой называется Attention. Архитектура сети Трансформер как раз основана на Attention. С картинками происходит то же самое, но вместо слов у нас пиксели, которые связаны друг с другом, или связаны не пиксели, а части одной и той же картинки.



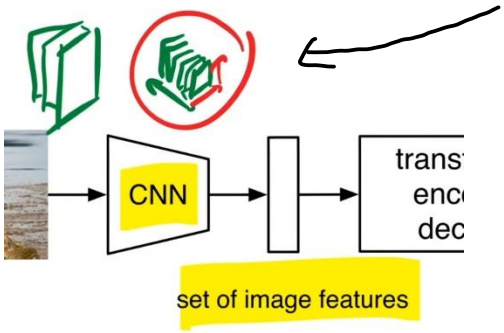
DETR совершенно новая сеть, основанная на трансформерах, не на свертках. Она использует в разы меньше параметров и намного проще, быстрее обучается и детектит. Вот сравнения самих авторов DETR на coco-датасете.

Table 1: Comparison with Faster R-CNN with a ResNet-50 and ResNet-101 backbones on the COCO validation set. The top section shows results for Faster R-CNN models in Detectron2 [50], the middle section shows results for Faster R-CNN models with GIoU [38], random crops train-time augmentation, and the long **9x** training schedule. DETR models achieve comparable results to heavily tuned Faster R-CNN baselines, having lower AP_S but greatly improved AP_L. We use torchscript Faster R-CNN and DETR models to measure FLOPS and FPS. Results without R101 in the name correspond to ResNet-50.

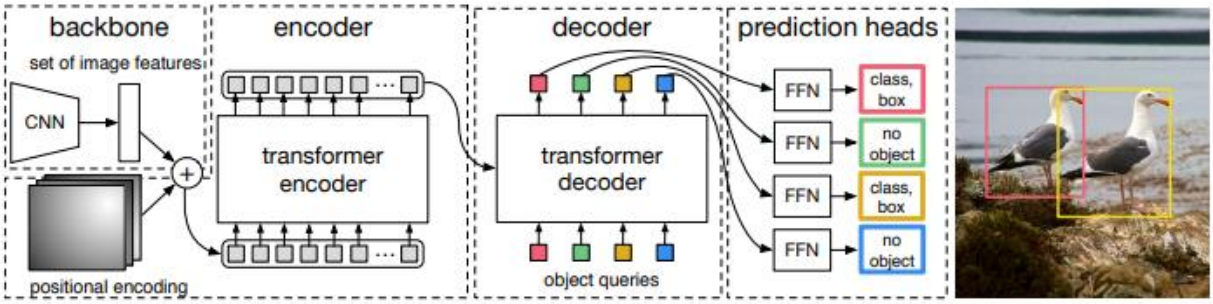
Model	GFLOPS/FPS	#params	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Faster RCNN-DC5	320/16	166M	39.0	60.5	42.3	21.4	43.5	52.5
Faster RCNN-FPN	180/26	42M	40.2	61.0	43.8	24.2	43.5	52.0
Faster RCNN-R101-FPN	246/20	60M	42.0	62.5	45.9	25.2	45.6	54.6
Faster RCNN-DC5+	320/16	166M	41.1	61.4	44.3	22.9	45.9	55.0
Faster RCNN-FPN+	180/26	42M	42.0	62.1	45.5	26.6	45.4	53.4
Faster RCNN-R101-FPN+	246/20	60M	44.0	63.9	**47.8**	**27.2**	48.1	56.0
DETR	86/28	41M	42.0	62.4	44.2	20.5	45.8	61.1
DETR-DC5	187/12	41M	43.3	63.1	45.9	22.5	47.3	61.1
DETR-R101	152/20	60M	43.5	63.8	46.4	21.9	48.0	61.8
DETR-DC5-R101	253/10	60M	**44.9**	**64.7**	47.7	23.7	**49.5**	**62.3**



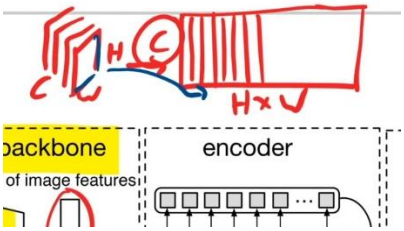
Сначала картинка проходит через сверточные слои, чтобы получить image features. CNN выдает нам все еще картинку, но с большим количеством каналов.



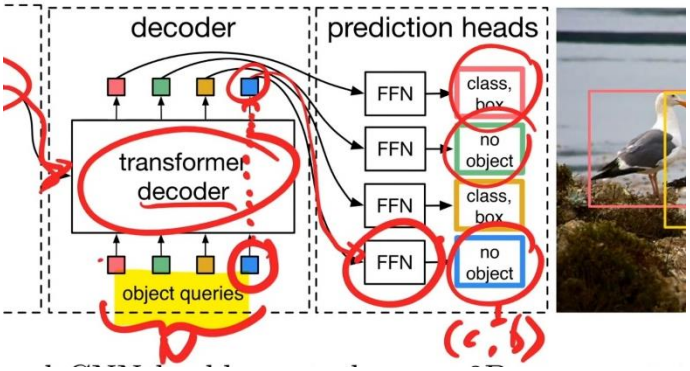
Далее set of features идет в трансформер, где происходит вся магия и на выходе мы получаем набор предсказаний в форме кортежей типа (class, bounding box). Например (bird, x=2 y=5). Особенность данной архитектуры в том, что у нас присутствует в списке классов нулевой/пустой. То есть один из аутпутов может быть (Ø, x=7 y=8). Bipartite matching loss борется с повторяющимися классами на картинке. В случае на изображении с чайками, мы избегаем того, что bbox будет обводить обе чайки и называть их чайкой.



В backbone с помощью CNN извлекаются image features (они все еще имеют природу изображений). Далее flatten, так как трансформеры работают с последовательностями (как слова в предложении). Мы раскручиваем feature images в плоскую последовательность и подаем в энкодер.



Декодер берет последовательность object queries (наши классы). Каждый из queries потом станет bbox.



Сначала эти queries являются рандомными (так как мы не знаем координаты bbox), потом используя информацию из энкодера они обучаются. Object query с помощью attention обучается на выводе энкодера и выдает bbox в результате.