Тренды в анализе данных. Статья 3

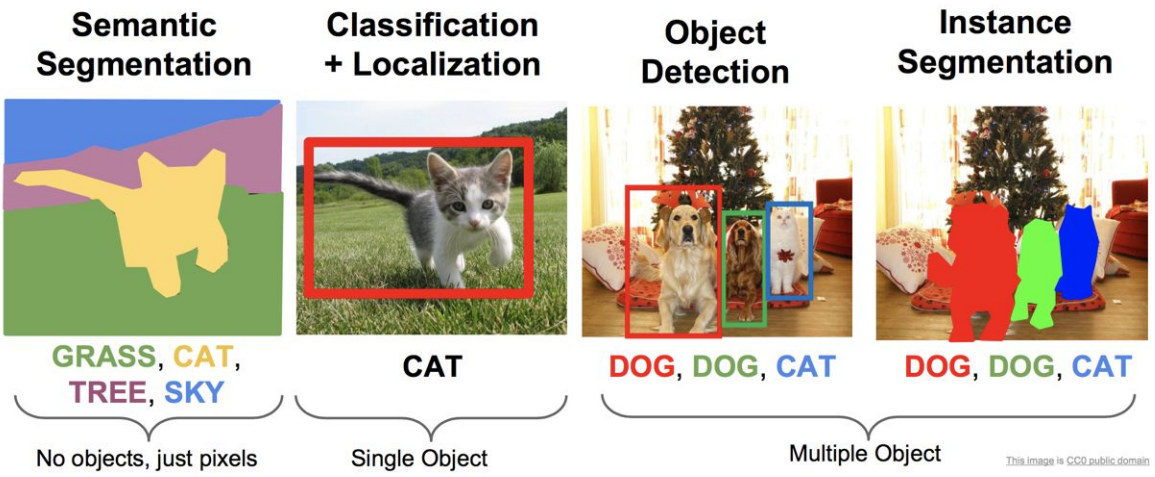
О цикле

Данный цикл статей основан на семинарских занятиях открытого курса Data Mining in Action по направлению “тренды в анализе данных”. На семинарах (и в статьях по ним) мы будем говорить о последних достижениях в области data science.
Общую лекцию для всего потока можно посмотреть здесь, а презентацию к ней скачать здесь.

Сегодня в статье:

1. Semantic Segmentation
2. Classification + Localization
3. Object detection
4. Instance segmentation

В computer vision существуют четыре классические задачи, которые сложнее, чем обычное распознавание:



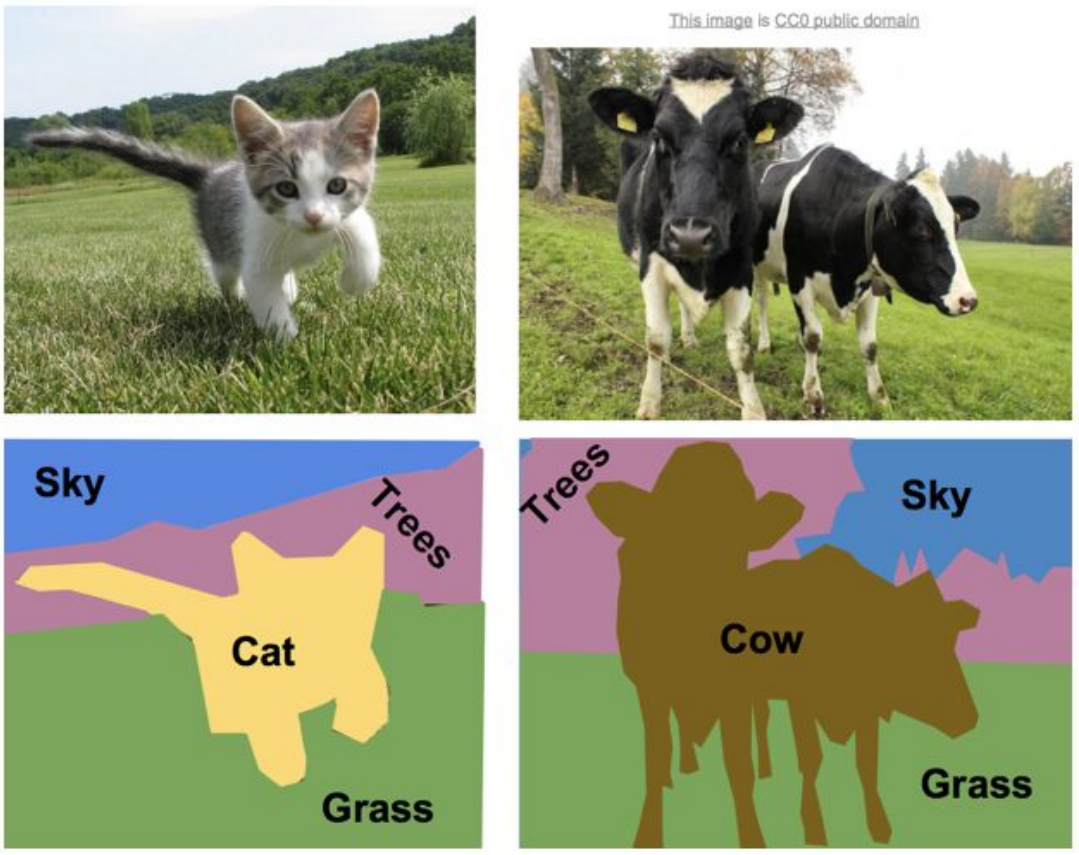
- **Semantic Segmentation**: присвоение каждому пикселю изображения класса: например, пиксель *X* относится к классу “трава”, а пиксель *Y* - к классу “кот”;
- **Classification and Localization**: нам нужно найти объект и обвести его рамкой;

- **Object Detection**: на изображении есть много объектов, и наша задача - найти каждый из них, обвести его рамкой и отнести его к одному из возможных классов;
- **Instance segmentation** - самая сложная из четырех задач. Это та же семантическая сегментация, но в том случае, если объектов на изображении много (сколько точно - неизвестно), и мы хотим не объединять котов в один класс, а прорисовать их отдельно.

Кроме этого, есть и пятая задача - классификация. Ее можно считать решенной в классической постановке (если данных достаточно много, нет шума и так далее).

Semantic Segmentation

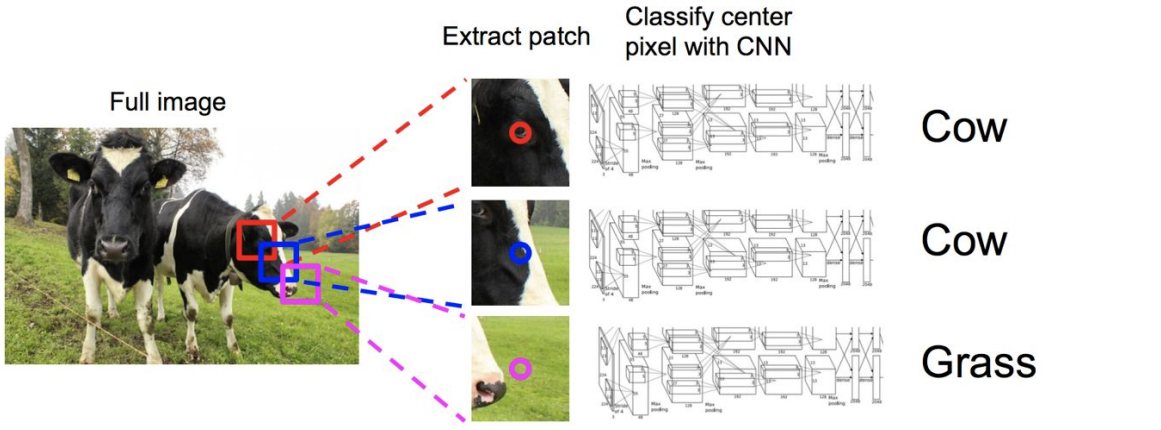
Наша задача - отнести каждый пиксель к какому-то классу. При этом отдельные объекты одного класса мы не различаем. И если нам попадаются два объекта, например, из класса “корова”, мы просто присваиваем всем пикселям объектов этот класс. Нам неважно, что их две и они разные.



Как это сделать?

Первая идея

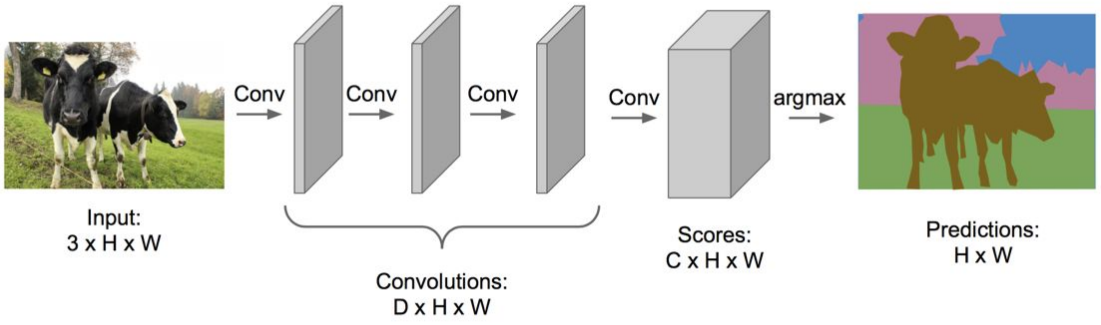
Можем пройтись по всей картинке каким-то окном (например, 10x10). И поставить в соответствие центральному пикселю окна какой-то класс. В чем проблема такого подхода? Во первых, в одном окне может быть недостаточно информации (например, в зеленом поле лежит зеленый пакет - его мы не распознаем). Во вторых, прогонять каждое окно через нейросеть - это очень долго.



Farabet et al, "Learning Hierarchical Features for Scene Labeling," TPAMI 2013
Pinheiro and Collobert, "Recurrent Convolutional Neural Networks for Scene Labeling", ICML 2014

Вторая идея

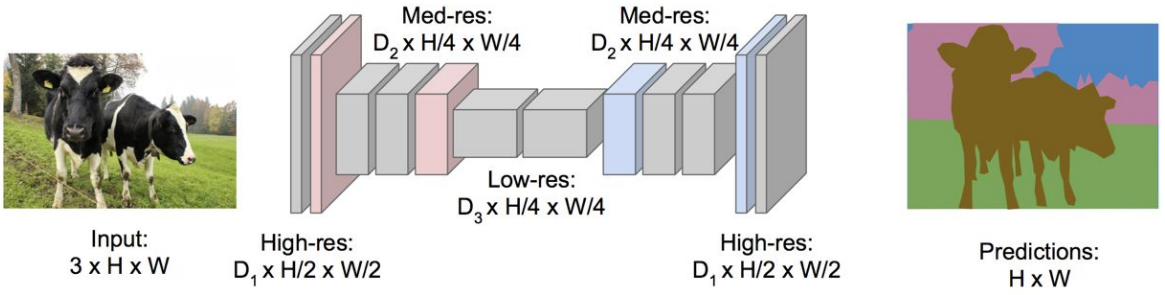
Можно прогнать изображение через свертки так, чтобы они не понижали его размерность, а оставляли ее такой же. На выходе мы получим стек карт признаков (feature maps), в котором ширина и высота будет такая же, как у оригинального изображения, а глубина - количество классов. Это значит, что значение каждого пикселя (по глубине) мы пропустим через softmax и получим для него вероятности классов. После этого можем сегментировать картинку.



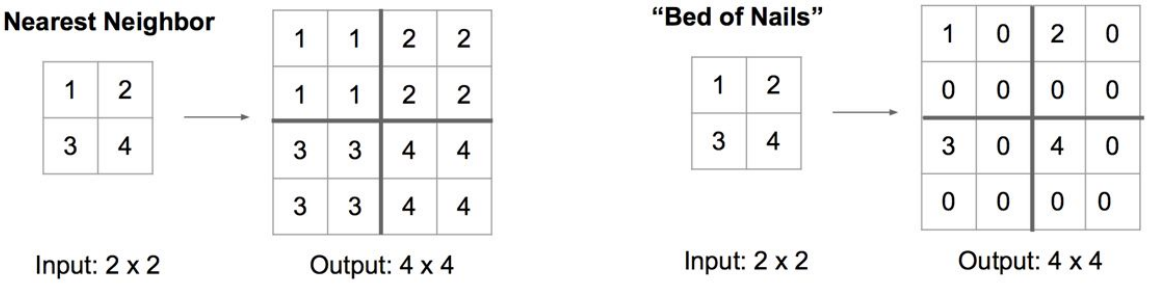
В чем здесь может быть проблема? Это требует слишком много памяти и времени, так как карты признаков будут сохранять ширину и высоту оригинального изображения, а значит, будут довольно большими.

Третья идея

Чтобы избавиться от проблемы с памятью, можно модифицировать предыдущий подход следующим образом. Будем использовать downsampling, чтобы сжать картинку и upsampling, чтобы вернуть ее первоначальный размер. То есть, сначала снижать высоту и ширину картинки, а потом увеличивать обратно. Нам важно сохранить разрешение картинки, чтобы у последней карты признаков высота и ширина не отличались от исходных. Кроме этого, у нас в любом случае останется глубина - количество классов. А что будет внутри сети - нам неважно, поэтому картинку можно ужать. Главное, чтобы сеть могла обучиться: тогда мы сэкономим вычисления, а результат получим тот же.
Заметим, что эти методы можно применять как к первоначальному изображению, так и к картам признаков.



Downsampling делают обычным способом - с помощью pooling и convolution слоев. А как делать upsampling? Можем делать unpooling или transpose convolution. Идея unpooling очень простая. Чтобы увеличить изображение, мы интерполируем его и заполняем пропуски соседними значениями. Этот метод unpooling называется Nearest Neighbor. Можно заполнить пропуски чем-то другим, например, нулями или любой другой константой, как в Bed of Nails, но этим обычно не пользуются.

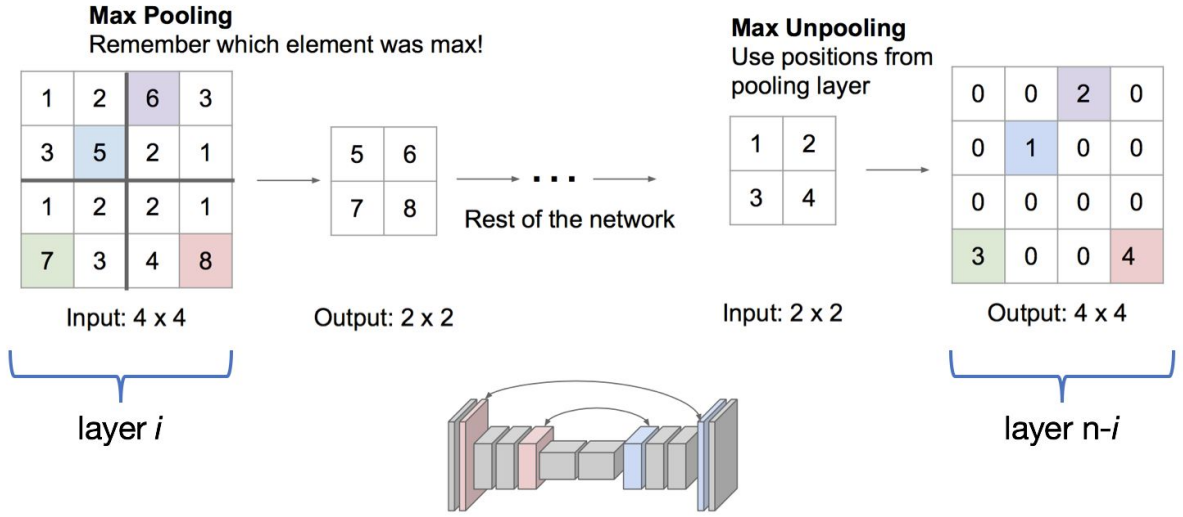


Кроме того, как мы поняли из прошлой статьи, можно делать любую дифференцируемую интерполяцию. Например, в случае с билинейной интерполяцией - заполняем средним значением между соседними пикселями.

Вывод: сжимая картинку, мы сможем обучаться быстрее. Однако какая-то информация все равно теряется.

Четвертая идея

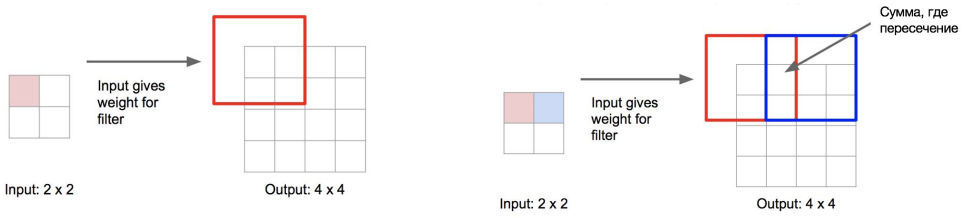
Есть более элегантное решение - pooling и unpooling должны соответствовать друг другу.



Каждому pooling слою на другом конце сети должен соответствовать на такой же позиции слой unpooling. В таком случае мы можем запомнить позицию, на которой находился максимум во время max pooling, и поставить его обратно во время соответствующего unpooling. Остальное можно заполнить нулями (Bed of Nails). Таким образом мы сохраняем больше информации.

Другая хорошая идея: Transpose convolution

Итак, было бы здорово заставить сеть выучить, как делать upsampling. Когда мы делаем свертки, у нас есть forward и backward pass этого слоя. Давайте поменяем их местами. Тогда мы получим свертку наоборот.



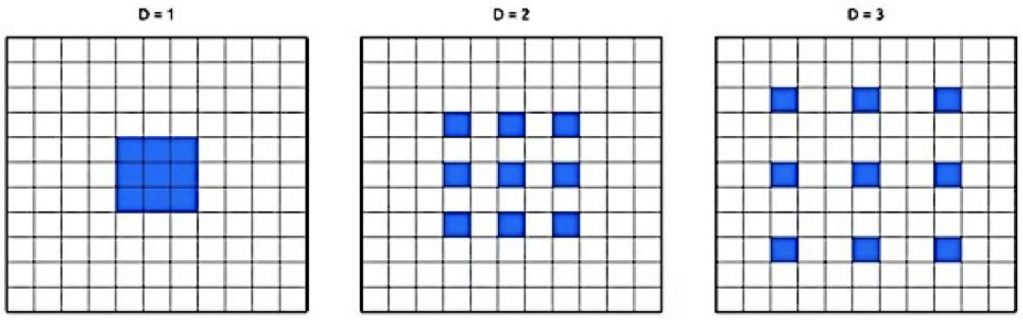
Transpose convolution, filter = 3x3, stride = 2, pad = 1

Как это работает? Допустим, у нас есть сжатое изображение 2x2, и мы хотим сделать его размером 4x4. Обычно у нас есть большое исходное изображение, по которому мы проходим фильтром (например, 3x3) и получаем уменьшенный

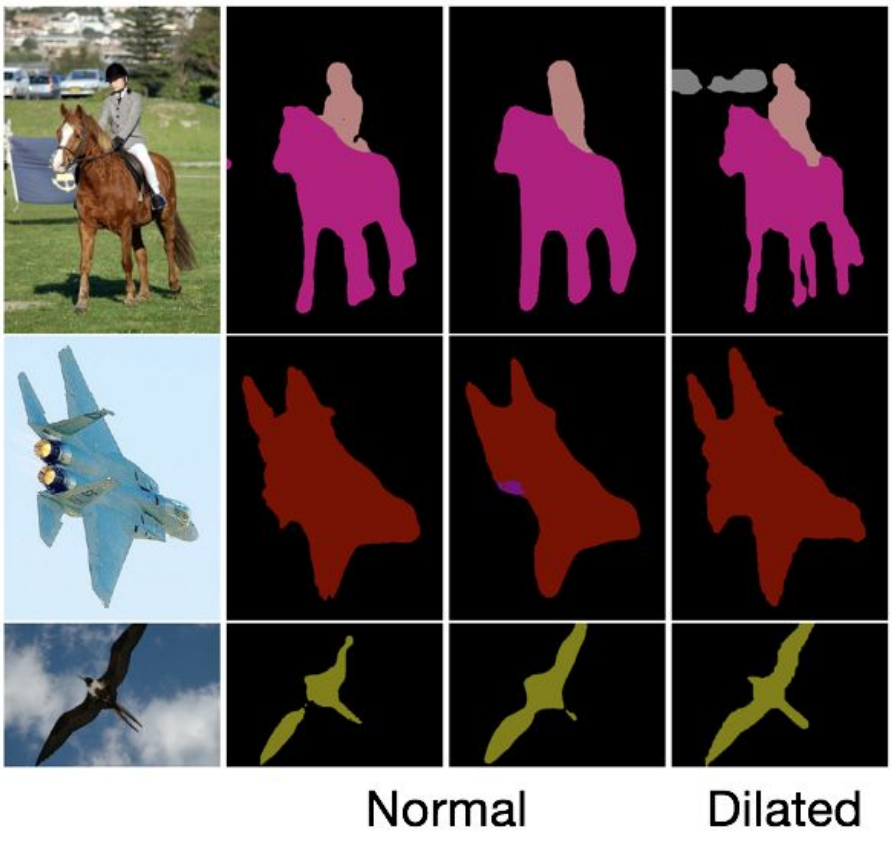
output. А почему бы не делать наоборот? Можем сделать матрицу 4x4, заполнить ее нулями и пройти по ней фильтром. То есть, мы по порядку умножаем один пиксель из сжатого изображения на каждое число из фильтра и вставляем на соответствующее место в выходном изображении. Таким образом из одного квадратика (пикселя) слева мы получаем девять квадратиков (пикселей) справа.

Модно: Dilated Convolutions

А что, если мы будем проходить не сплошным фильтром, а смотреть через один/два пикселя?

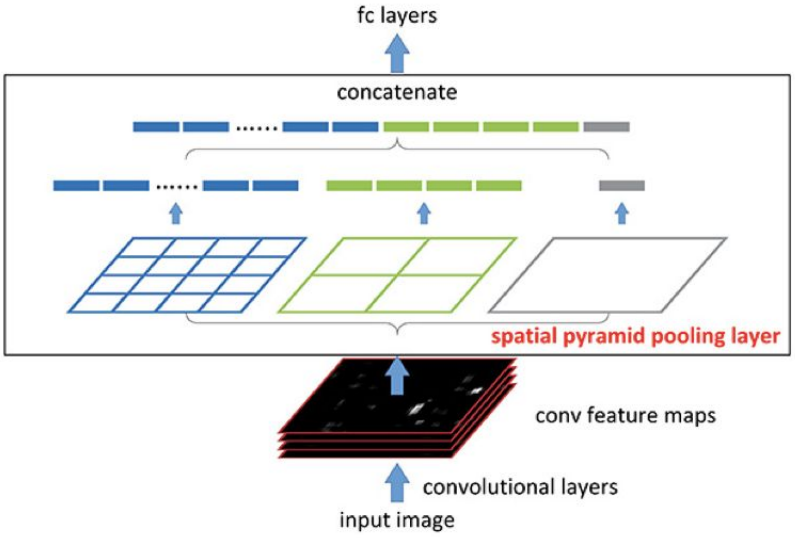


Когда мы делаем unpooling, мы теряем много деталей. Так как изначальная цель - это разметить входное изображение, это невыгодно. Кроме этого, мы теряем четкие границы из-за природы операции свертки. Этот подход немного спасает ситуацию:



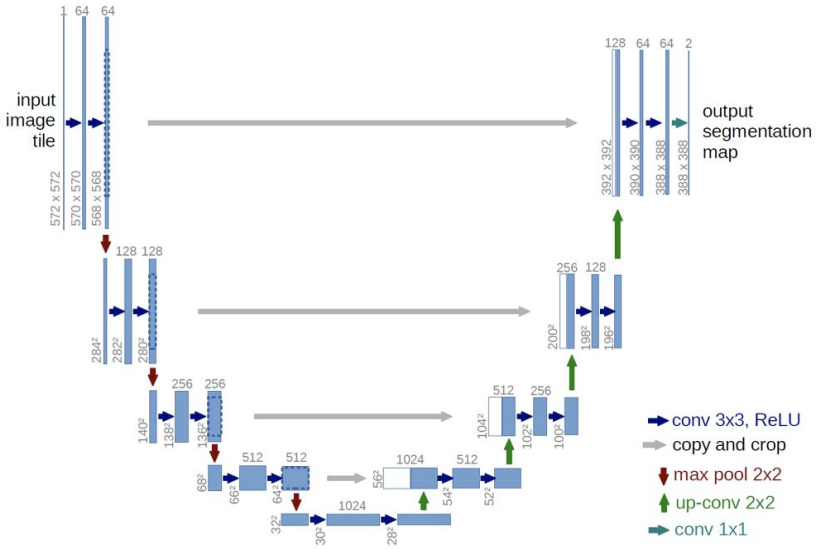
Еще одна интересная идея: Pyramid pooling

Представим, что у нас есть некоторая карта признаков 20x20. Мы делаем pooling с размером окна 5x5 (получаем выход 4x4), pooling 10x10 (получаем 2x2), а еще 20x20 (как global pooling) и получаем выход 1x1. После чего все это конкатенируем. Это интересная идея, хоть она и не увеличивает качество в точности классификации, но повышает качество в задачах сегментации.



Суть в том, что pooling дает некоторую инвариантность к сдвигам. И чем больше слоев pooling, тем больше инвариантность. Если мы все выражаем в итоге одним числом - это абсолютная инвариантность (то есть, как ни двигай объект, выходное число останется тем же). Соответственно, когда мы сложим все вместе, наша нейросеть научится обращать внимание на тот размер инвариантности, который ей нужен.

Классная идея: Skip-connections (U-net)



Итак, есть еще одна хорошая идея, как не потерять детали при downsampling и upsampling. Как мы уже говорили, мы можем сделать так, чтобы при сжатии и разжатии размеры карты признаков совпадали (то есть,. каждому pooling слою соответствует unpooling слой). Тогда почему бы не конкатенировать карты признаков, которые у нас получались в начале сети при сжатии изображения, к тому, что у нас получается при восстановлении? Таким образом нам лучше удастся сохранить детали. Такой подход использовали для обучения сети U-net, которая должна была сегментировать клетки в biomedical imaging задаче.

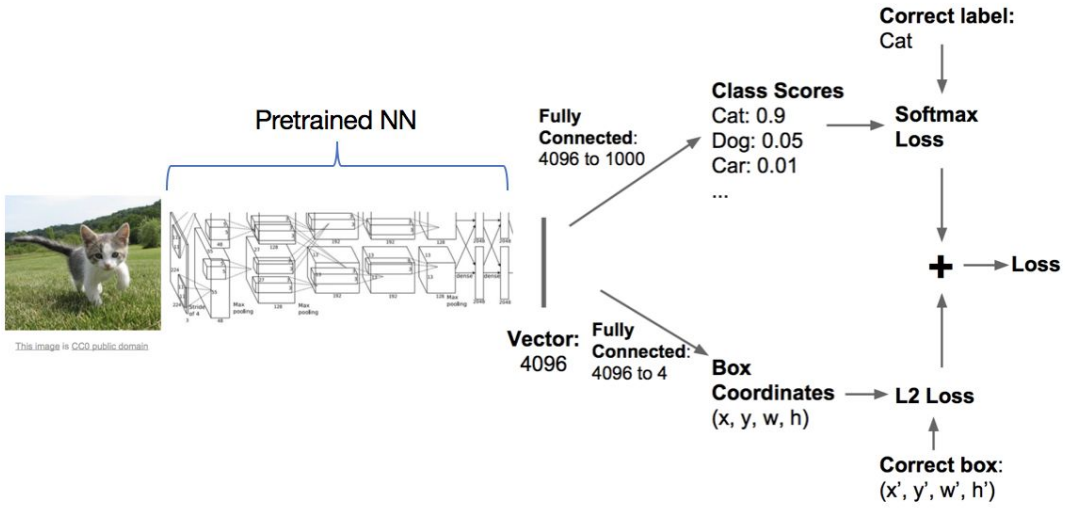
Подведем итоги

Чтобы считать быстрее, мы понижаем размерность обычным способом - convolutions и pooling, а повышаем обратно с помощью unpooling и transpose convolutions. Чтобы потерять как можно меньше информации, делаем Dilated Convolutions или Skip-connections. Также можно попробовать Pyramid pooling.

Classification + Localization

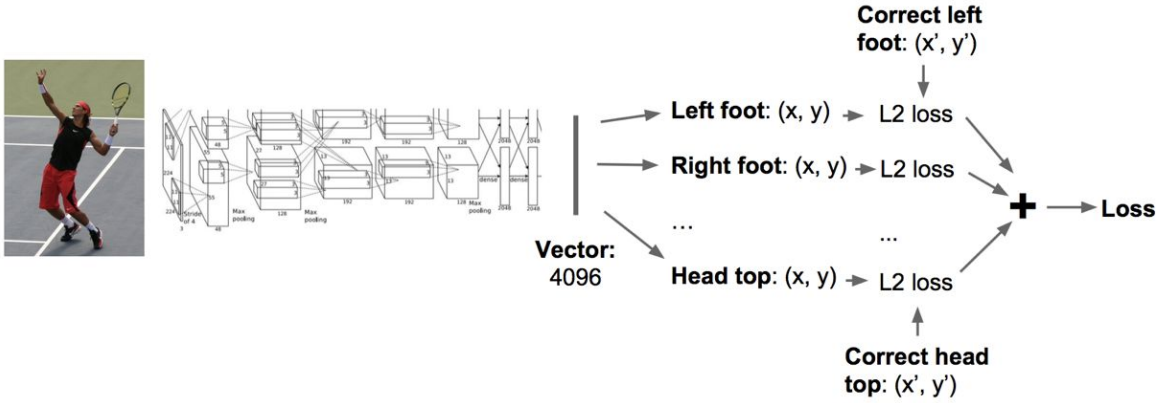
Напомним: задача состоит в том, чтобы локализовать объект (обвести его рамкой) и классифицировать его. И мы всегда имеем дело только с одним объектом.

Идея: давайте пропустим наше изображение через сеть, которая выдаст нам класс (как обычно) и четыре числа, которые будут характеризовать рамку, в которой находится объект. Обозначим эти числа как *(x, y, w, h)*. Первые два числа - координаты середины рамки, а вторые два - ее размер.



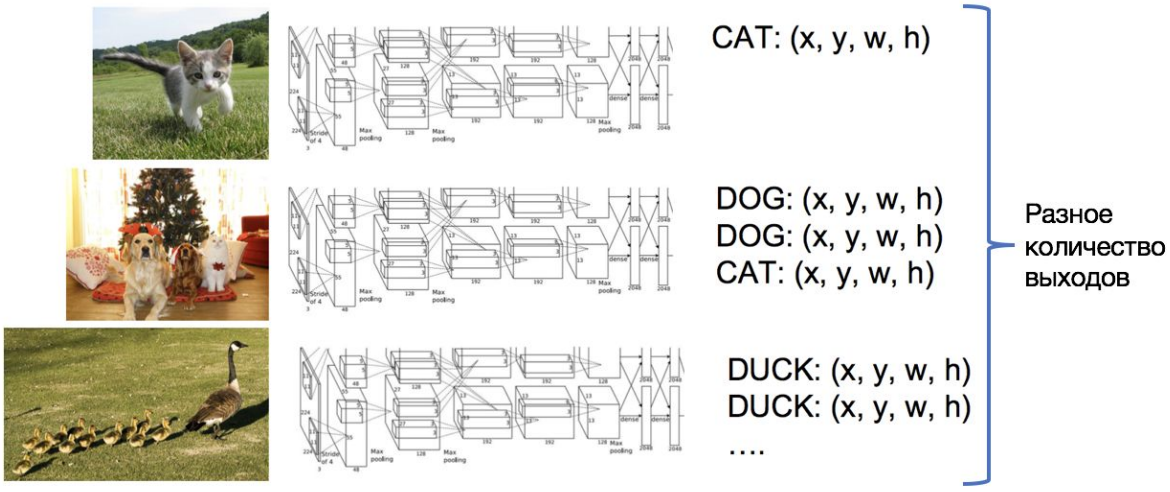
Далее мы тренируем эту сеть с двумя функциями потерь, одна из которых отвечает за качество классификации, а другая - за качество нахождения рамки. Складываем их и получаем финальную функцию потерь. Задача решена.

Эта идея нашла отклик и в других задачах. Например, мы можем похожим образом распознавать позу человека.



Здесь мы хотим найти координаты для каждой части тела человека, чтобы впоследствии как-то классифицировать позу. Для этого мы считаем функцию потерь для каждой части тела, а потом складываем их в одну.
С помощью этой сети мы собираем некоторый вектор - все полученные координаты - например, из десяти чисел, который описывает позу человека (где находится голова, где левое колено, правый локоть и так далее). А после, если мы, к примеру, решаем задачу распознавания агрессивной позы, и наши изображения были размечены, мы можем обучить логистическую регрессию для этой задачи на этих векторах.

Object detection

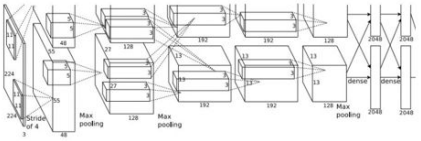


Эта задача уже более сложная. Первая проблема: мы не знаем заранее, сколько объектов на картинке. То есть, мы не можем пользоваться решением из предыдущего раздела и пытаться предсказывать какое-то фиксированное количество цифр (например, десять - как с позой человека). Вторая проблема: в этой задаче часто бывает дисбаланс классов. Например, мы пытаемся

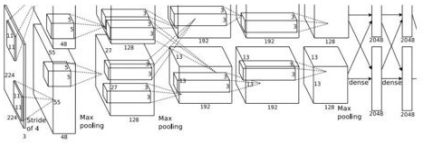
детектировать кошек и собак, а остальное относим в класс background. Очевидно, что представителей класса background будет много, а интересующих нас рамок будет мало.
Таким образом, нам сначала нужно понять, сколько у нас объектов, а потом для каждого объекта обвести его и предсказать класс.

Первая идея

Давайте просто передвигать некоторое окно и пытаться угадать, что внутри.

Dog? NO
Cat? NO
Background? YES

Dog? YES
Cat? NO
Background? NO

Этот подход нам не подходит - получится слишком долго. Скорее всего объекты будут разного масштаба, и нам придется проходить по изображению несколько раз окнами разного размера. И тогда скорее всего мы часто будем получать один и тот же объект в разных рамках. Как нам взять именно те рамки, которые подходят лучше всего?

Intersection over union

intersection

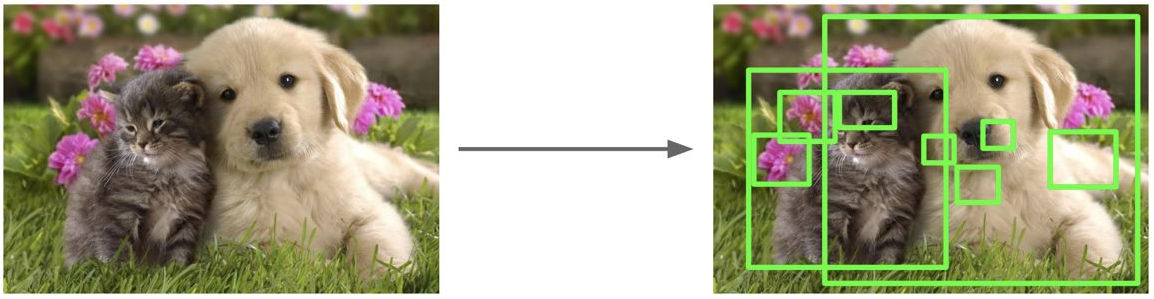
union

IoU = intersection / Union

Давайте рассмотрим концепт, который поможет нам справится с проблемой. Допустим, мы ищем на картинке машину; она находится в красной рамке, а предсказали мы зеленую. Обозначим их пересечение синим цветом, а объединение площадей - желтым. Тогда IoU - это intersection, разделенный на union. Нам пригодится этот концепт, когда мы будем учить модель.
Итак, у нас может быть предсказано, например, 500 рамок, а на самом деле нужных нам всего 20. Однако мы не хотим считать функцию потерь для каждой из 500 предсказанных рамок с каждой из двадцати реальных. Тогда мы можем сначала посчитать их IoU, а потом отсечь ненужные рамки с каким-то порогом, и по оставшимся считать функцию потерь.

С чего все начиналось: алгоритмические методы без ML

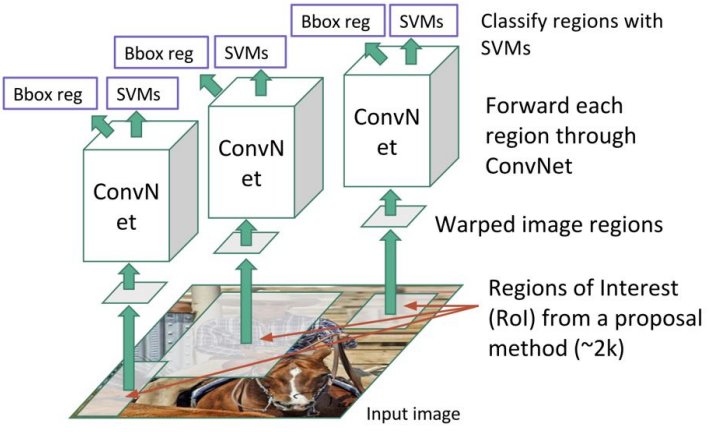
Один из таких алгоритмов - Selective Search.
Работает неплохо, может вычислить порядка 2000 регионов за несколько секунд. Регион - это место, в котором *может быть* какой-то объект. При этом мы сами задаем, сколько регионов мы хотим найти (это гиперпараметр).



У алгоритма очень высокий False Positive, и нам придется как-то избавляться от мусора, который он принял за объект. Но зато, если на картинке есть хотя бы какой-то объект, он точно найдет его.

Первые успехи: R-CNN

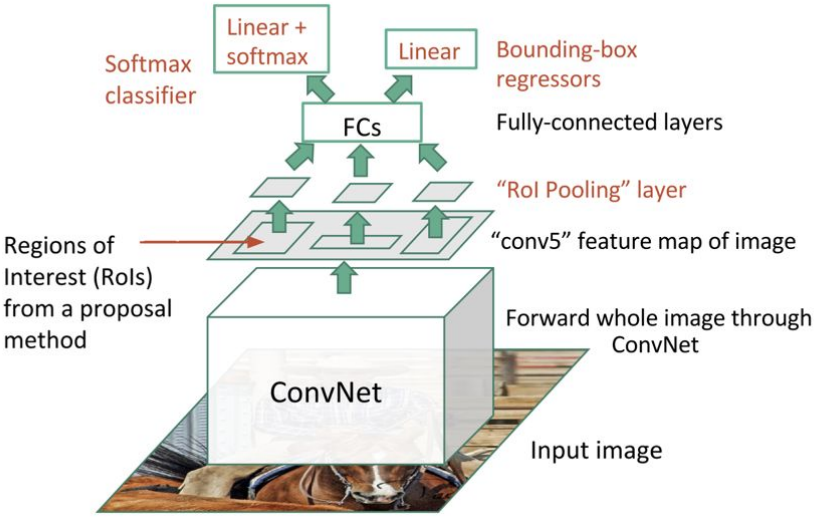
Мы берем все регионы, которые находит нам Selective Search, и пропускаем через сетку.



Здесь у нас есть обычная CNN, Bbox reg и SVM. Bbox reg уточняет координаты рамки (может сжать, удлинить, немного сдвинуть), а SVM предсказывает класс объекта из рамки. Для каждого изображения нам нужно прогнать все регионы через сеть, а значит, обучаться эта сеть будет очень долго. Более того, на каждый из этих модулей у нас будет своя функция потерь, поэтому все будет учиться отдельно, а это тоже очень медленно. На тесте *предсказание* такой сетки делается за *47 секунд*.

Давайте быстрее: Fast R-CNN

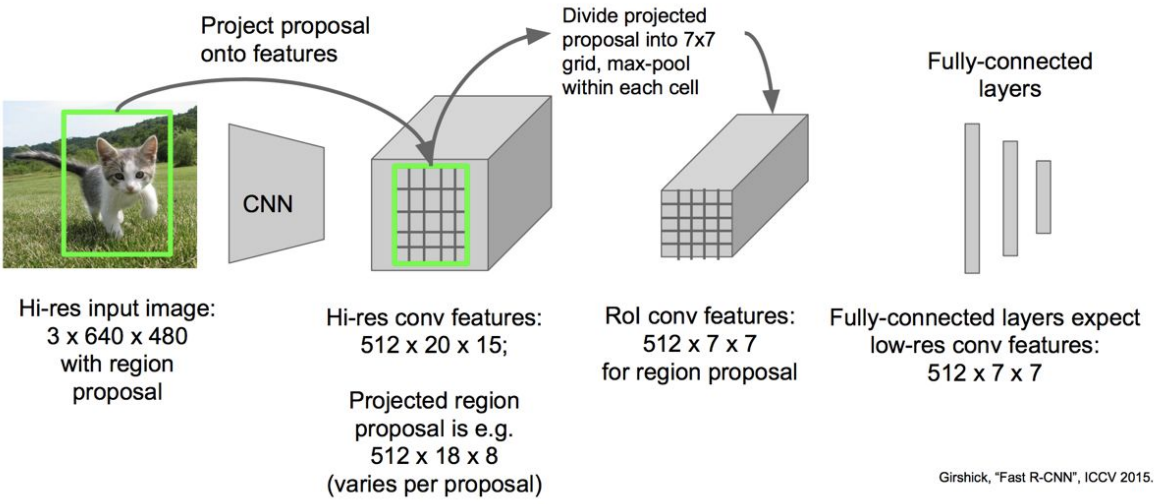
Итак, было бы здорово как-то сократить количество проходов по нейросети. Например, можно искать регионы как-то иначе, чтобы не приходилось прогонять все регионы через всю нейросеть.



Тогда мы можем сразу подать картинку в CNN и сделать Selective Search на стеке карт признаков после какого-то convolution слоя. Дальше идет такая же архитектура, как в R-CNN, но чтобы еще убыстрить сеть, решили считать общую функцию потерь как сумму функций потерь из каждого модуля. Таким образом, модули учатся вместе.

Чтобы отправить все регионы на слои FC здесь добавляется слой RoI pooling. Он помогает нам сделать все регионы одного размера.

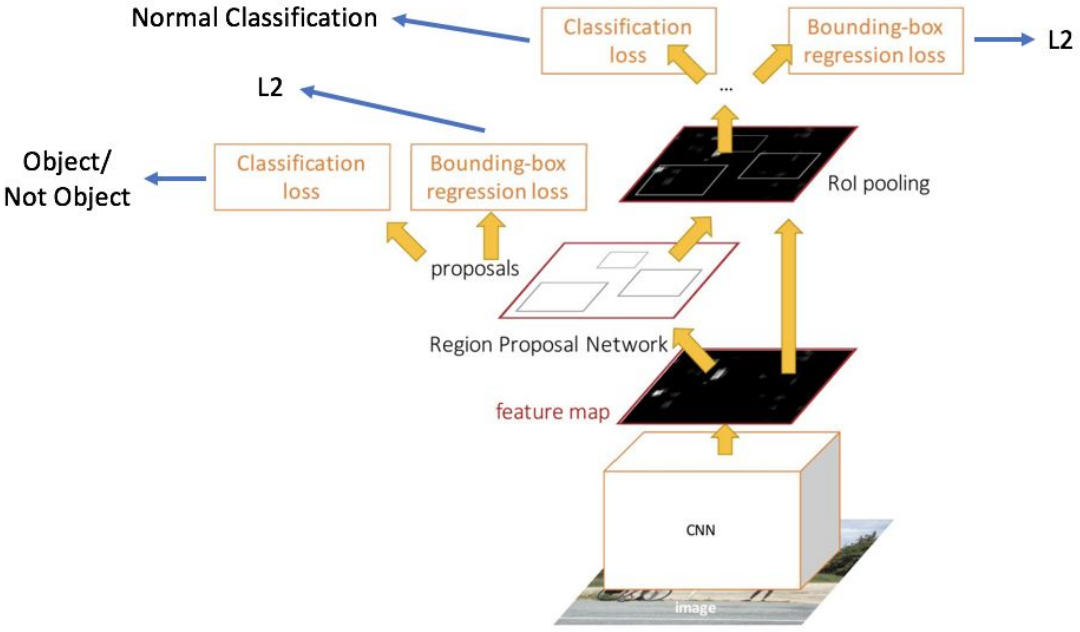
Если мы хотим получить все регионы одного размера, то можем взять некоторую сетку, например, 5x5 и разбить регион на 25 одинаковых частей, а после этого сделать max pooling по каждой части. Тогда на выходе мы получим регион 5x5. После этого мы уже можем отправить все регионы в Fully-connected слои (они принимает только объекты одинакового размера), и сделать предсказание, которое будет занимать значительно меньше времени, чем в R-CNN.



Таким образом, на тесте получается 2,5 секунды. Причем из них две секунды занимает Selective Search.

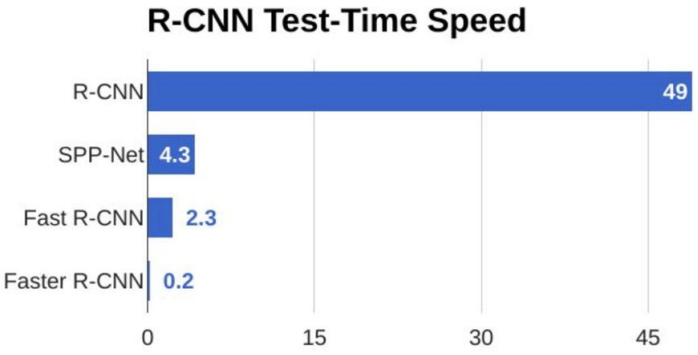
Еще быстрее: Faster R-CNN

Авторы этого метода предложили вместо Selective Search использовать еще одну нейросеть - Region proposal network, которая должна находить регионы.



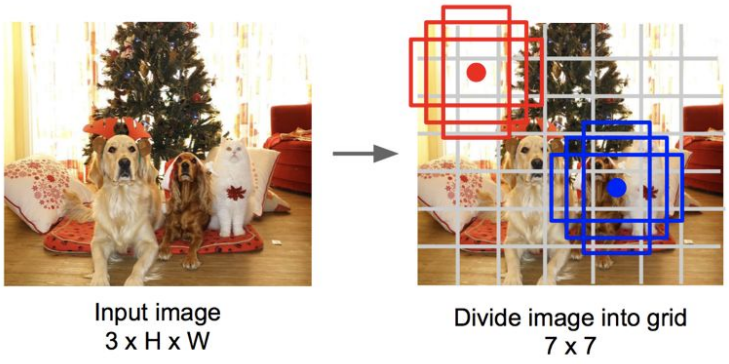
Здесь у нас уже не две, а четыре функции потерь, которые мы складываем и учим вместе. Classification loss для Region proposal network считает только то, угадали ли мы, что в предложенном регионе есть хотя бы какой-то объект, или нет. А classification loss выше отвечает за то, как хорошо мы угадали, что за объект мы нашли. Скорость здесь достигается за счет того, что Region proposal network выдает нам меньше мусора, нежели Selective Search, и мы сразу смотрим на те регионы, где действительно что-то есть. Этот метод вряд ли

удастся использовать в режиме реального времени, но у них получилось достигнуть 0,2 секунды на тесте и находить очень хорошие рамки.



Наконец, Real-Time

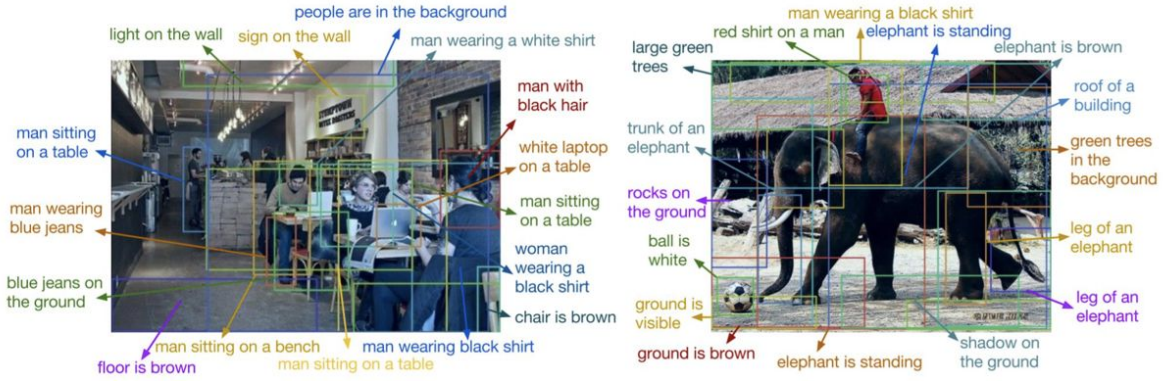
Рассмотрим метод YOLO, который можно использовать в режиме реального времени. В частности, его используют для self-driving cars.



Идея этого метода в том, что мы можем сильно ускориться, если не будем пропускать через сеть каждый регион отдельно, а пройдем по всему изображению. Нужно сразу разбить картинку на некоторую сетку и внутри каждой клетки предсказывать какое-то количество рамок (это будет гиперпараметр). Далее для каждой рамки предсказываем координаты центра относительно клетки, а высоту и ширину относительно всего изображения. Это позволяет двигать рамку относительно центра клетки, а также сжимать и разжимать ее. Кроме того, нейросеть выдает какую-то уверенность в том, что в рамке есть объект. Наконец, предсказываем класс для каждой рамки. После этого мы можем оставить только те рамки, у которых уверенность в том, что в них есть какой-то объект, будет выше некоторого порога. Это работает очень быстро именно за счет того, что нам не нужно прогонять через сеть каждый регион отдельно. На тесте примерно 50 картинок в секунду.

Интересная идея: Dense Captioning

Можно соединять идеи и расширять задачи. Например, авторы статьи Dense Captioning не только предсказывают, что за объект в рамке, но и дают к нему какое-то описание.



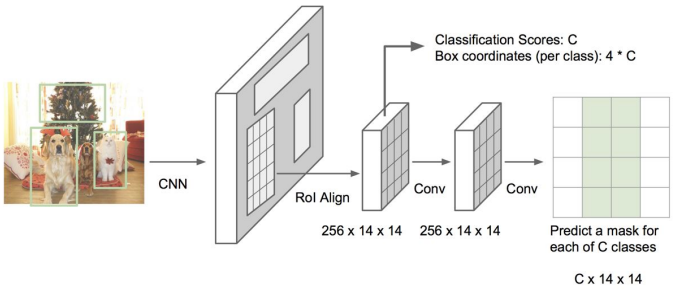
Например, на левой картинке они не только нашли объект “человек”, но и как-то описали его (сидящий человек, в джинсах и так далее). Разумеется, для того, чтобы обучать что-то подобное, нужны очень хорошо размеченные картинки и много лейблов. И все же качество модели получилось очень хорошим.

Instance segmentation

Напомним, что в этой задаче мы хотим сегментировать какое-то неопределенное количество объектов и не объединять представителей одного класса в один объект, а различать их на картинке. Основная сложность именно в последнем.

Mask R-CNN

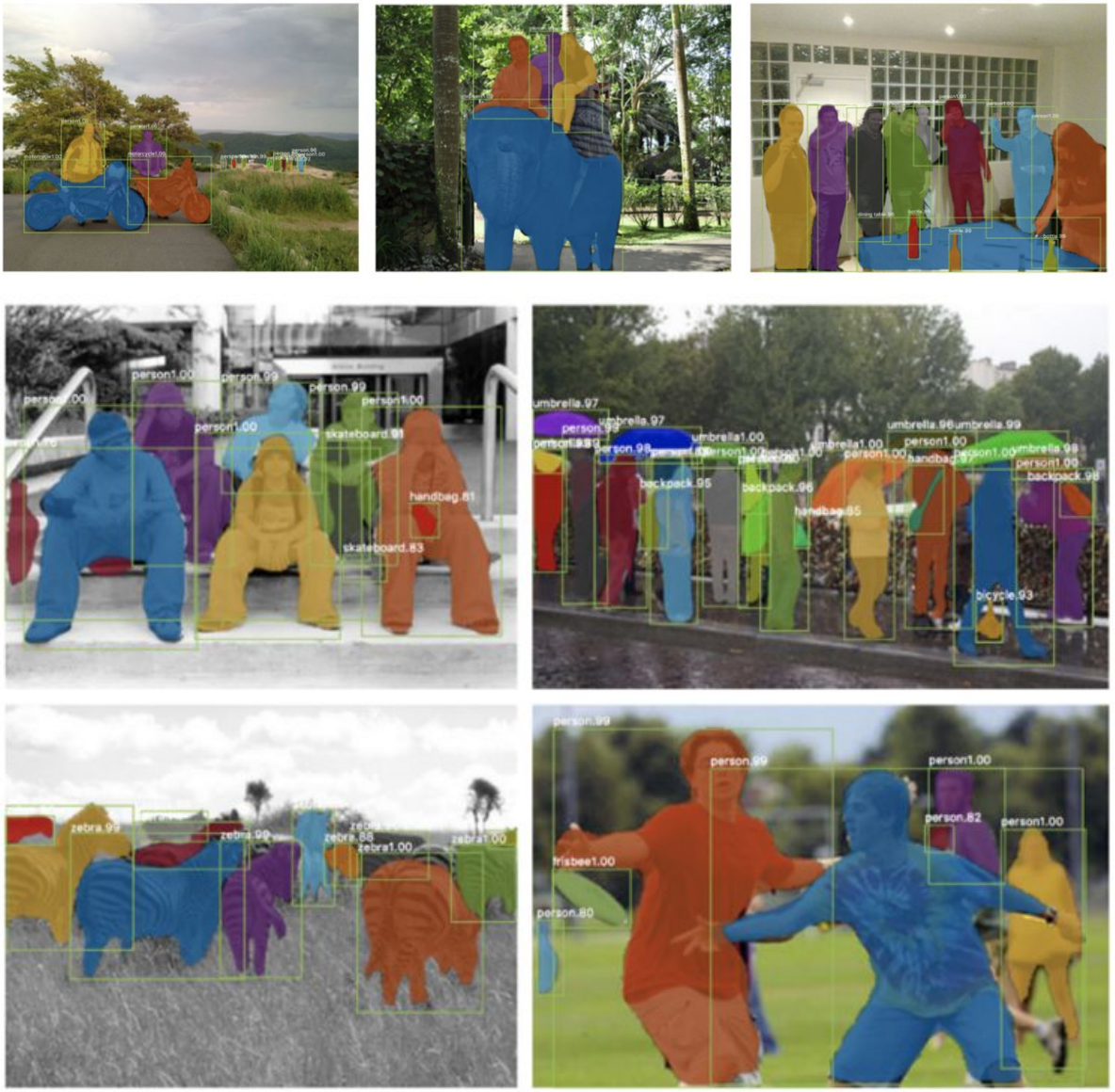
Давайте возьмем Faster R-CNN и добавим еще одну функцию потерь, которая будет отвечать за маску, которую мы ищем. Этого будет достаточно, чтобы перейти от задачи Object segmentation к Instance segmentation.



Кроме этого, авторы статьи предложили свой метод разбиения на сетку вместо RoI pooling. Они сделали свой слой, который назвали RoIAlign, чтобы

избавиться от проблем с округлением. В Faster R-CNN, если мы хотим разбить картинку 20x20 сеткой 7x7, то нам придется интерполировать, так как разбить на целые пиксели не получится. А значит мы теряем какую-то информацию. Мы не будем рассматривать как именно работает RoI Align, вы можете прочитать об этом здесь.

Результаты работы метода очень хорошие:



Эта статья вышла в начале 2017 года, и до сих пор сделать лучше никто не смог.

Оставить отзыв по прочитанной статье вы можете здесь. По всем вопросам пишите на почту курса: dmia@applieddatascience.ru

Список литературы

Видео

Лекция базируется на видео лекции Stanford University School of Engineering по классификации изображений (Fei-Fei Li, Justin Johnson, Serena Yeung). Часть слайдов взята из презентации. Оригинальную лекцию можно посмотреть здесь, а здесь скачать к ней слайды.

Статьи

1. Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition
2. Multi-Scale Context Aggregation by Dilated Convolutions
3. Encoder-Decoder with Atrous Separable Convolution for Semantic Image Segmentation
4. U-Net: Convolutional Networks for Biomedical Image Segmentation
5. The One Hundred Layers Tiramisu: Fully Convolutional DenseNets for Semantic Segmentation
6. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks
7. DenseCap: Fully Convolutional Localization Networks for Dense Captioning
8. YOLO9000: Better, Faster, Stronger
9. Mask R-CNN