Практический анализ данных и машинное обучение: искусственные нейронные сети

Ульянкин Филипп и Соловей Влад

19 марта 2020 г.

Посиделка 8: Object detection

Agenda

- Быстрая история
- YOLO

История и метрики

История

Вначале подход был следующим - мы вручную придумываем фичи из картинки, потом каким-либо алгоритмом придумываем как выделить на картинке объект.
Например мы хотим научиться выделять велосипед - нам нужно сначала научиться выделять части велосипеда, потом скользящим окном выделяем область, где больше всего частей было выделено.
Такой подход хорош, но он медленный - мы много раз используем один и тот же алгоритм на одной картинке, а это больно.
Приход нейронных сетей дал возможность использовать "нейронные знания" об изображении.

Метрики

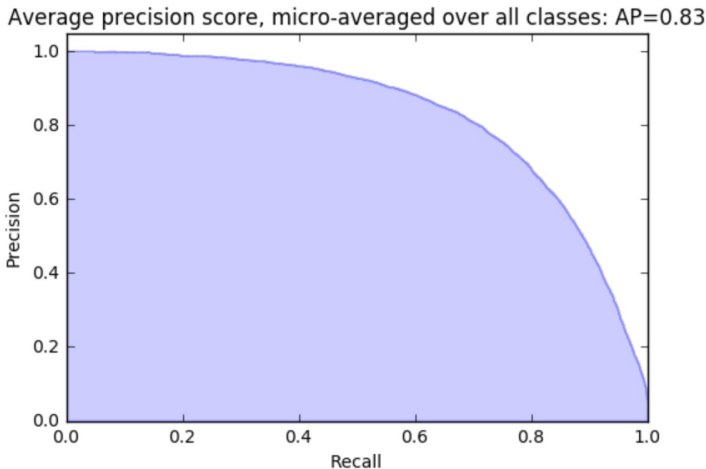
Качество детекции определяется следующими метриками :

1. Bounding box precision - Правильно ли мы подобрали рамочку, не слишком ли она большая/маленькая;
2. Recall - Все ли мы объекты нашли?
3. Class precision - Мы нашли то, что надо или нет?

Еще мы это хотим применять в real time, а для этого нам нужна скорость. Долго скорость была большой проблемой для алгоритмов - для реального времени нам надо более 5ти детекций в секунду.

Метрики

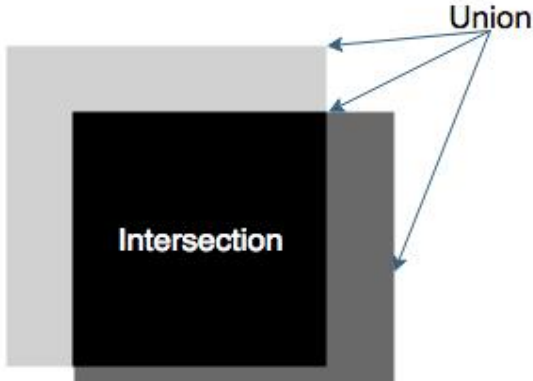
Но и precision и recall зависят от нашего treshhold, и в точке его считать не очень хорошо. Выход? - посчитаем для каждого уровня отсечения и возьмем площадь под кривой.



Метрики

Осталось научиться считать правильность наших рамочек. Для этого используется следующая метрика - Jaccard index (IoU).

$$IoU = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|} \tag{1}$$



YOLO

YOLO - начало

Несколько фактов о YOLO:

1. YOLO-you only look once
2. Самая быстрая архитектура - 170 рамок в секунду на изображении 256 на 256
3. Вышла в 2015 году, сейчас уже третья версия из 2018ого
4. не самая точная, но быстрая за счет небольшой потери качества.

YOLO - ограничения

От чего страдает YOLO - плохо работает с мелкими объектами.
Как пример - Вам будет тяжело выделить отдельную птицу из стаи.
С этим пытаются бороться, но тяжко. И на краях изображения тоже могут возникнуть проблемы.

YOLO - ключевая идея

Перевод задачи в простую задачу регрессии. Для каждой рамки мы должны спрогнозировать следующие параметры :

1. Центр рамки
2. Ширину и высоту рамки
3. Вероятность того, что в рамке лежит объект
4. Ну и класс объекта

Это все чиселки, мы можем считать по ним регрессию! (но пока не будем)

YOLO - ключевая идея 2

Всю картинку мы покрываем сеткой, из выбираем объекты внутри этой сетки.

Размер сетки(grid) - наш гиперпараметр. Количество всех возможных рамок - w*h*B(пока оставим за скобками B, тоже наш гиперпараметр)

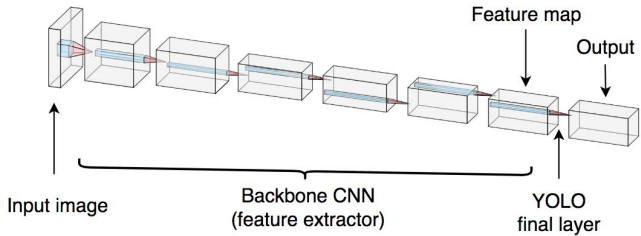
YOLO - вход

Начнем разбирать модель с применения.

Признаки YOLO обычно берет из других моделей (backbone model). Выбор этой модели влияет на итоговый результат. YOLO берет куб выхода из других моделей.

Итоговая сетка(которая на картинке) зависит от следующих факторов :

1. Насколько сильно снижает размерность модель feature extractor. VGG-16 сжимает в 16 раз (но оставляет 512 каналов)
2. Входной размер картинки



YOLO - выход

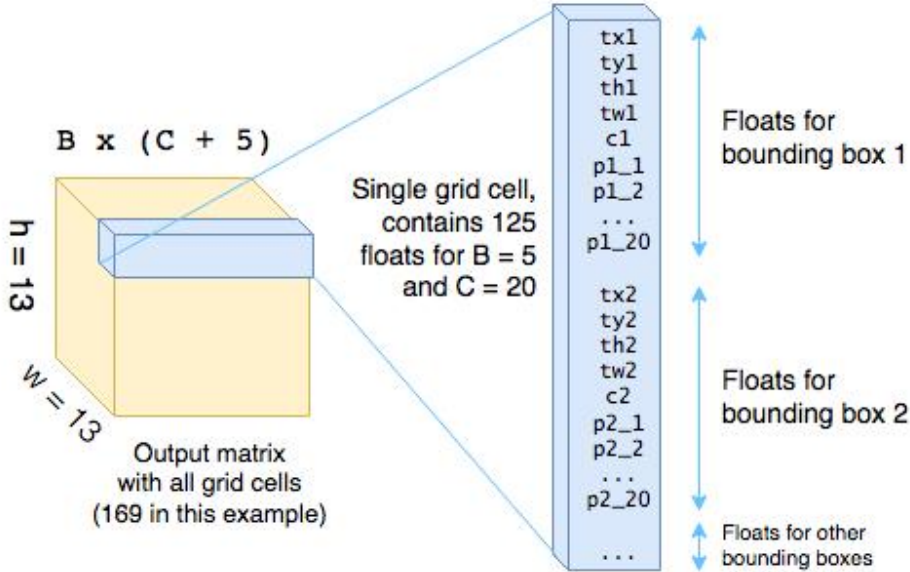
Итоговый выход YOLO - w*h*M, где M = B*(C+5).

1. C-количество классов, которое у нас всего может быть
2. B-количество рамок (как их задавать поговорим позже)

Осталось еще 5 штук:

1. t_x и t_y - координаты центра рамки
2. t_w и t_h - ширина и высота рамки
3. c - вероятность того, что у нас в рамке вообще что-то есть
4. ну и p_1,....., P_C - вероятности того, что в рамке лежит объекта класса 1,.... C

YOLO - выход


B x (C + 5)
h = 13
w = 13
Output matrix
with all grid cells
(169 in this example)
Single grid cell,
contains 125
floats for B = 5
and C = 20
tx1
ty1
th1
tw1
c1
p1_1
p1_2
...
p1_20
tx2
ty2
th2
tw2
c2
p2_1
p2_2
...
p2_20
...
Floats for
bounding box 1
Floats for
bounding box 2
Floats for other
bounding boxes

YOLO - якорные рамки

Наконец-то мы поговорим об этом **B**. Учить ширину, высоту и координаты центра рамки прям в лоб тяжело, объекты слишком разные по размеру. Поэтому мы вводим понятие - якорная рамка и будем немного корректировать эти рамки под конкретные объекты.
На практике таких рамок обычно от 3 до 25. Обычно используют 9 рамок:

1. квадратные (Большая, средняя , маленькая)
2. вертикально-прямоугольные (Большая, средняя , маленькая)
3. горизонтально-прямоугольные (Большая, средняя , маленькая)

Размеры рамок задаются под наш датасет руками.

YOLO -корректировка рамок

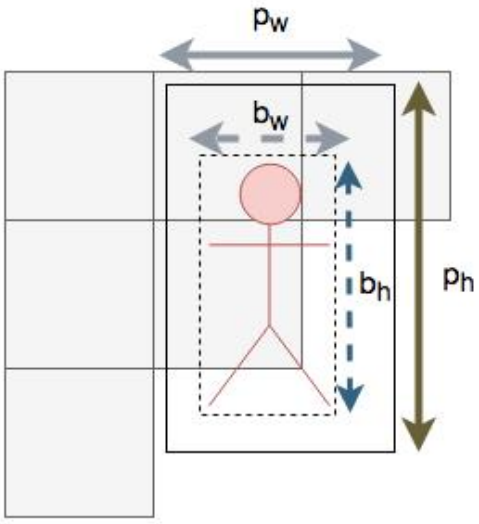
Рамки корректируются под конкретный объект по следующему правилу:

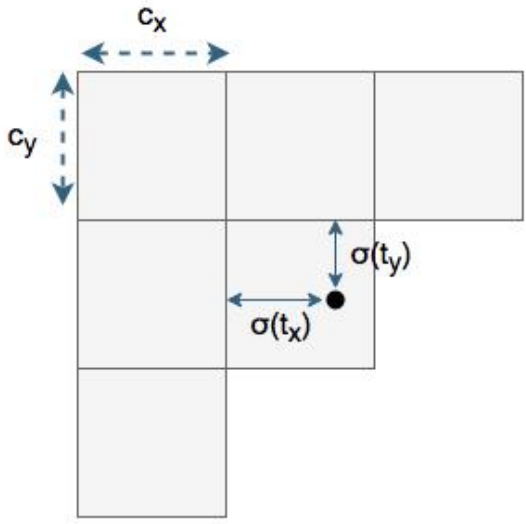
1. $b_x = sigmoid(t_x) + c_x$
2. $b_y = sigmoid(t_y) + c_y$
3. $b_w = p_w * exp(t_w)$
4. $b_h = p_h * exp(t_h)$

Где:

1. t_x,t_y,t_w,t_h - выход последнего слоя
2. b_x,b_y,b_w,b_h - итоговая оценка рамок
3. p_w,p_h - размеры якорной рамки
4. c_x,c_y - координаты конкретного места в сетке, где мы нашли рамку (включаются на границе)

YOLO -корректировка рамок





YOLO -пост-процессинг рамок

У нас получилось огромное количество рамок - надо их как-то отобрать.
Для начала выкидываем все, вероятность нахождение объекта в которых ниже определенной грани (опять же сами выбираем уровень).
Потом оставляем рамки с объектом, который принадлежит классу с наибольшей вероятностью. Получается следующая картинка:

YOLO -NMS

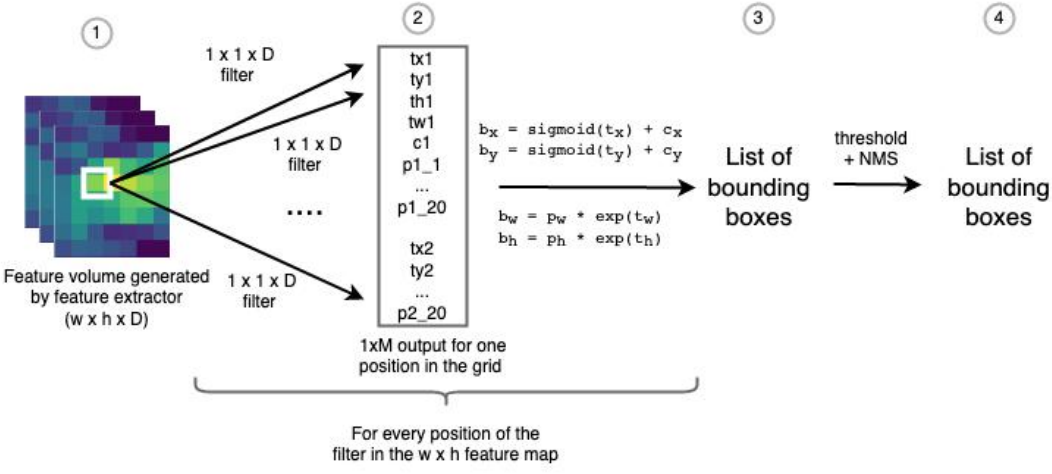
NMS - Non-max suppression.

Выкидываем все рамки, у которых IoU ниже какого-то порога (мы сами его выбираем). И следующим шагом выбираем рамку, у которой IoU имеет максимальное значение.

YOLO - Подводим итоги применения

1. Считаем карту признаков через backbone CNN
2. Считаем через сверточный слой корректировку для якорных рамок, вероятности нахождения объекта в рамке и вероятность класса внутри рамки.
3. Корректируем размеры нашей рамки
4. Отбираем наши рамки.

YOLO - Подводим итоги применения



YOLO - обучение

Модель backbone к нам приходит откуда угодно. Обычно учат на большом дата сете (типа imagenet) и потом используют.
Можно учить все сразу, но это больно (вот прям совсем больно). Можно предобучать как автоэнкодер.
Специально для YOLO была разработана архитектура, которая наиболее эффективно выделяет фичи из картинки (по словам авторов статей).
Данная структура называется Darknet. И да - основная модельная сложность в экстракторе фичей, если хотим использовать на мобилке, уменьшаем сложность этой модели.

Bounding box loss

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

Где

1. λ - насколько сильно мы хотим сконцентрироваться на рамках в итоговых потерях
2. 1_{ij}^{obj} - индикаторная функция, которая 1 если найденная рамка отвечает за объект. Из всех рамок выбирается рамка с наибольшим IoU.
3. Корень нам дает штраф на маленьких рамках больше, чем на больших

Object confidence loss

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

Где

1. λ - все также вес в итоговых потерях
2. C - содержит ли рамка вообще объект
3. 1_{ij}^{noobj} - индикаторная функция, которая 1 если найденная рамка никак не относится к объекту.

Classification loss

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

Тут все просто - угадали или нет с объектом.
Итоговый наш лосс - это сумма всех трех потерь сразу.

Несколько хинтов при обучении

1. Аугументация данных и dropout - обязательные куски, без них сетка сразу переобучается.
2. Каждый n эпох меняем входную размерность изображения, чтобы научить сеточку быть независимой от размерности
3. Предобучаем сеточку, все вместе не учим - ну очень больно
4. Градиент может взорваться - следим за этим.

Итого

Всей этой историей хотелось показать следующее - есть отдельные кусочки из которых состоят нейронки (свертки, dropout, batchnorm...), а есть отдельные архитектуры в которых очень много неочевидных хаков.
И прям учить архитектуры - плохая идея, слишком они быстро меняются и зависят от данных.