

marge_simpson: 96%

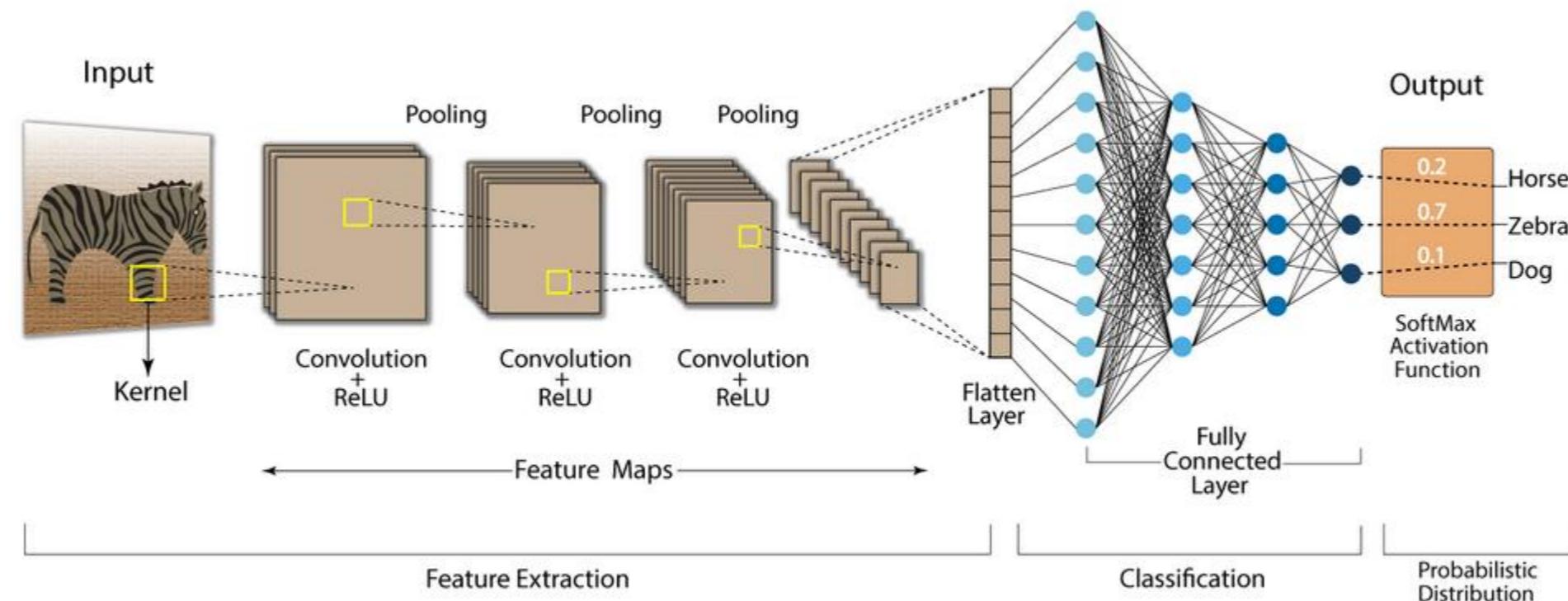
krusty_the_clown: 95%

Детектирование объектов на изображениях

Александр Дьяконов

3 октября 2022 года

Классификация изображений – сейчас уже почти решённая задача



Можно использовать готовую архитектуру / преднастроенную НС

Вход – изображение

Выходы – классы

Ошибка – LogLoss

<https://developersbreach.com/convolution-neural-network-deep-learning/>

Другие задачи типа «что и где изображено»

Классификация + локализация



«cat»

Детектирование объектов (Object Detection)



DOG, DOG, CAT

Семантическая сегментация (Semantic Segmentation)



GRASS, CAT, TREE, SKY

Сегментация объектов (Instance Segmentation)



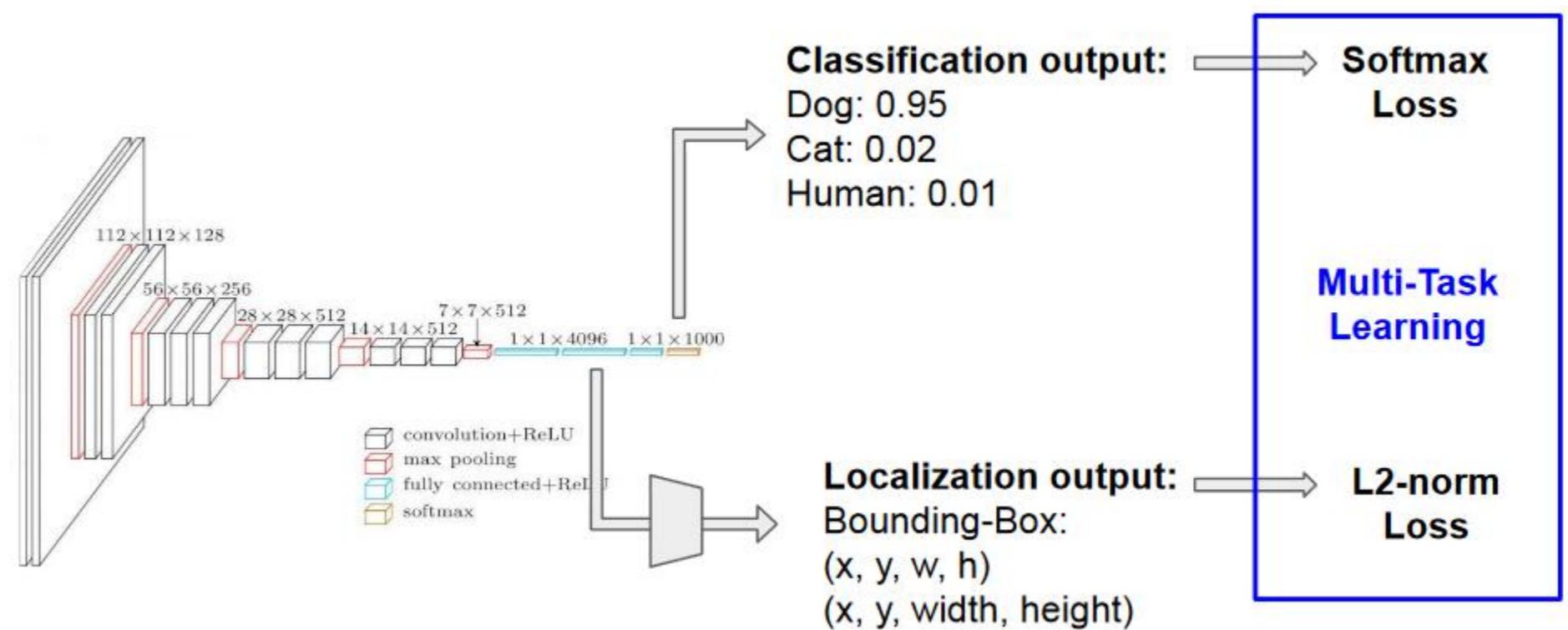
DOG, DOG, CAT

<http://cs231n.stanford.edu/2017/syllabus.html>

Локализация + Классификация

Классификация – что изображено

Локализация (localization) объекта – где изображено



идея решения довольно проста

[CS109B - 2021]

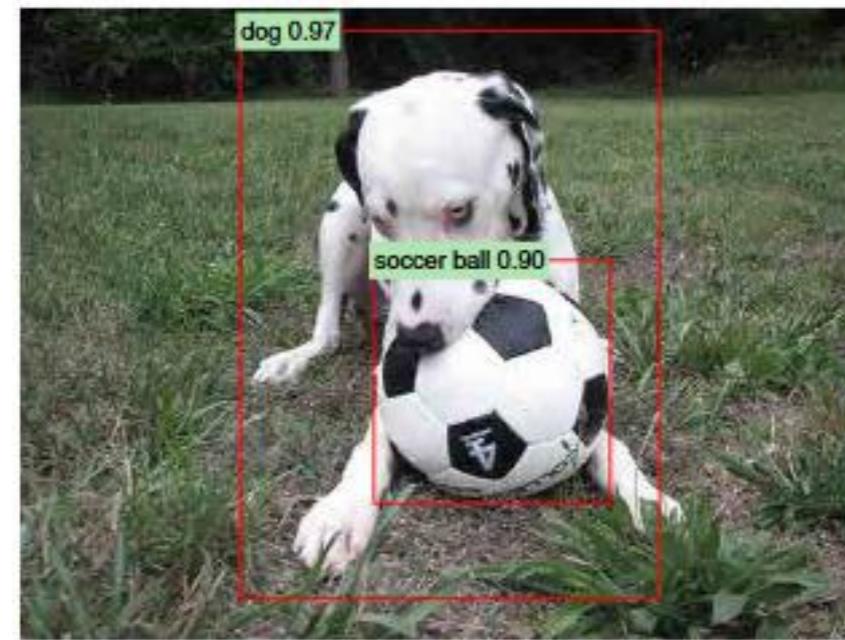
Детектирование объектов = (Локализация + Классификация) × число объектов**Локализация (localization) объекта – где****Классификация – что**
м.б. ещё определяем параметры объекта**Данные для детектирования объектов**

image	X1	X2	Y1	Y2	label
img1.jpg	7	39	3	57	car
img1.jpg	50	157	100	203	dog
Img2.jpg	40	105	207	389	car

выделенное (X1, X2, Y1, Y2, label) – предсказывать<http://cv-tricks.com/object-detection/faster-r-cnn-yolo-ssd/>

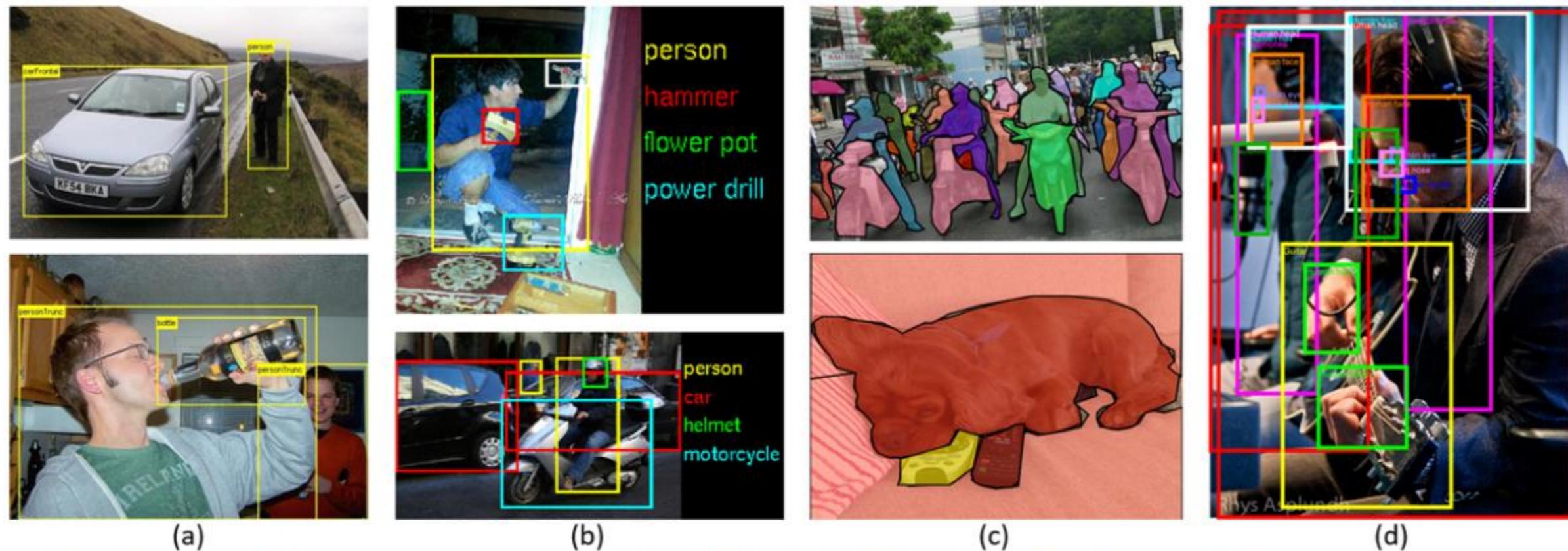
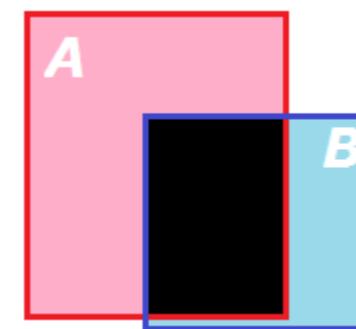


Fig. 4. Some example images and annotations in (a) PASCAL-VOC07, (b) ILSVRC, (c) MS-COCO, and (d) Open Images.

Dataset	train		validation		trainval		test	
	images	objects	images	objects	images	objects	images	objects
VOC-2007	2,501	6,301	2,510	6,307	5,011	12,608	4,952	14,976
VOC-2012	5,717	13,609	5,823	13,841	11,540	27,450	10,991	-
ILSVRC-2014	456,567	478,807	20,121	55,502	476,688	534,309	40,152	-
ILSVRC-2017	456,567	478,807	20,121	55,502	476,688	534,309	65,500	-
MS-COCO-2015	82,783	604,907	40,504	291,875	123,287	896,782	81,434	-
MS-COCO-2018	118,287	860,001	5,000	36,781	123,287	896,782	40,670	-
OID-2018	1,743,042	14,610,229	41,620	204,621	1,784,662	14,814,850	125,436	625,282

Метрики качества

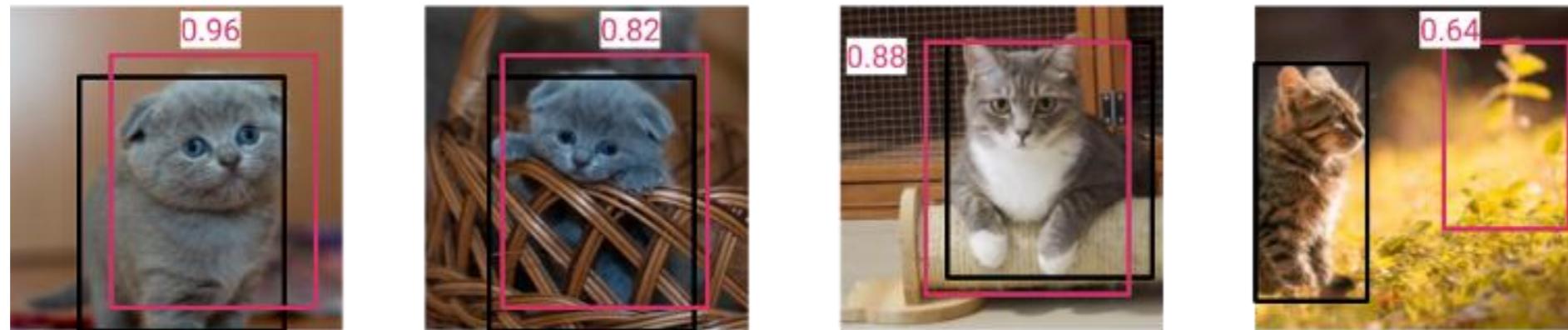
Правильное детектирование, если



$$\text{IoU} = \frac{|A \cap B|}{|A \cup B|} \geq \alpha$$

Intersection Over Union

TP, FN, FP
Precision / Recall

Average Precision (AP)**~ по задаче классификации региона****решение «Positive», если правильный класс и $\text{IoU} > 0.5$, иначе «Negative»**

Confidence	Cat Number	isCorrect?	Precision	Recall
0.96	Cat 1	1	1	0.25
0.88	Cat 3	1	1	0.5
0.82	Cat 2	1	1	0.75
0.64	Cat 4	0	0.75	0.75

**Упорядочиваем по confidence (оценке класса «кот»)****Площадь под кривой.****Mean Average Precision (mAP) – усреднение AP по всем классам**

Детектирование объектов: идея 1 – перебор регионов

Перебрать разные локализации (чаще всего используют прямоугольники) и для каждого – классификация



Генерация регионов: метод Selective Search

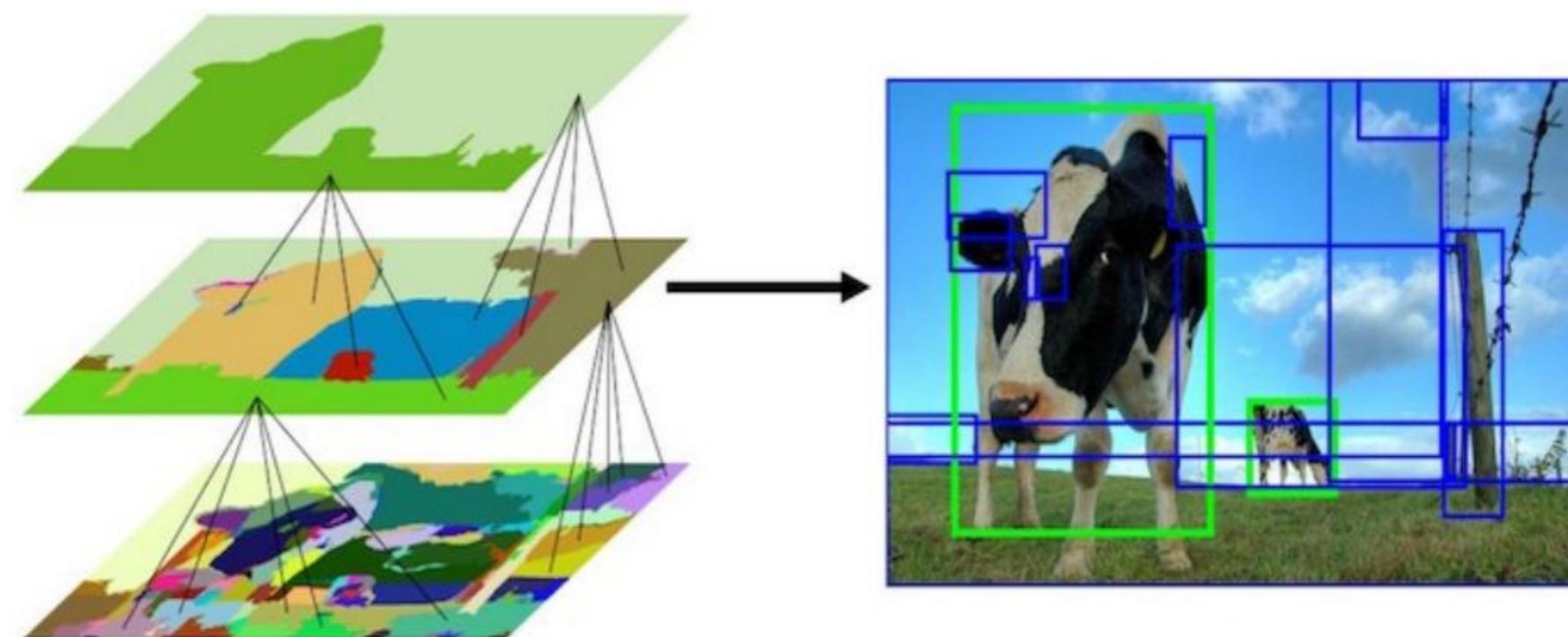


Figure 2: Two examples of our selective search showing the necessity of different scales. On the left we find many objects at different scales. On the right we necessarily find the objects at different scales as the girl is contained by the tv.

<http://www.hupellen.nl/publications/selectiveSearchDraft.pdf>

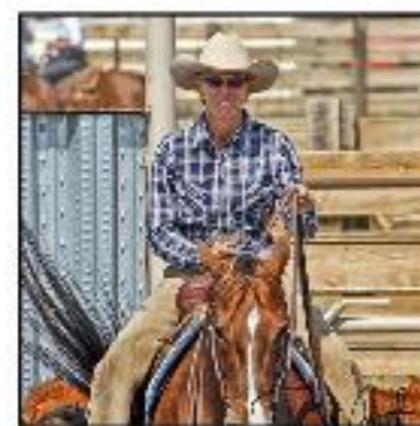
Selective Search

- 0. список = []**
- 1. Разбить изображение на сегменты**
- 2. Для каждого сегмента в список его обрамляющую рамку**
- 3. Объединить соседние похожие сегменты**
- 4. Если число сегментов > 1 перейти к 2**
- 5. Для каждой рамки из списка оценить наличие в ней объекта**



Детектирование объектов: семейство R-CNN

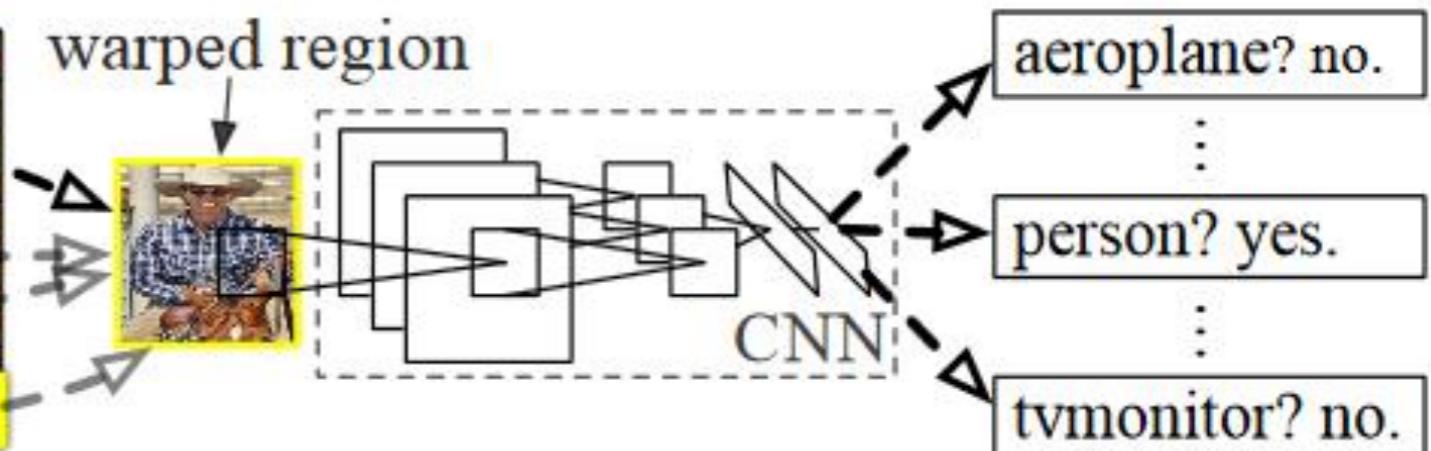
- «**R-CNN – Object detection and semantic segmentation**»
[Girshick et al., 2013 <https://arxiv.org/pdf/1311.2524.pdf>]
- «**Fast R-CNN**» – [Girshick 2015 <https://arxiv.org/pdf/1504.08083.pdf>]
- «**Faster R-CNN**» – [Ren et al., 2015 <https://arxiv.org/abs/1506.01497>]



1. Input
image



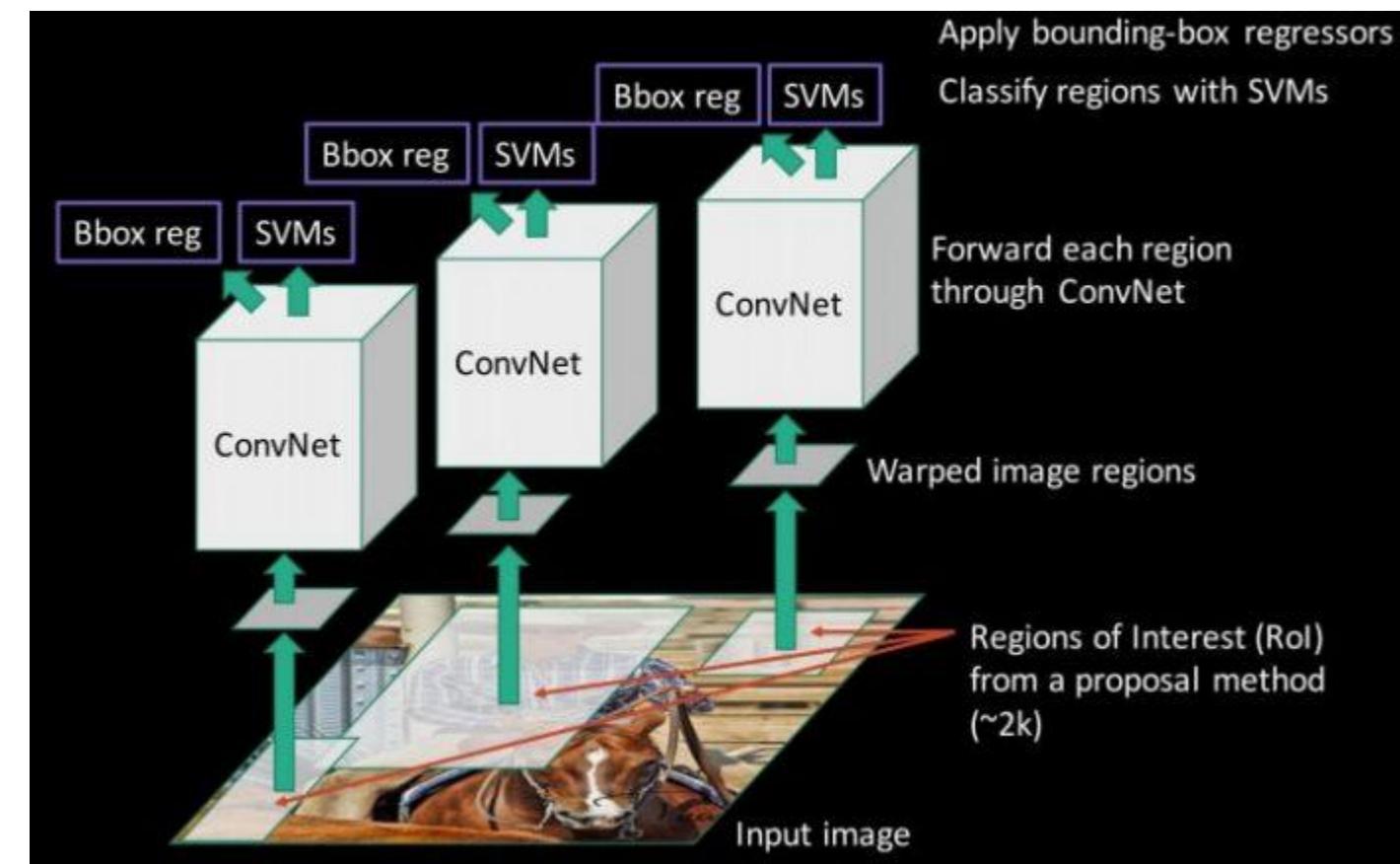
2. Extract region
proposals (~2k)



3. Compute
CNN features

4. Classify
regions

R-CNN (=Regions with CNN features)



- **Selective Search** для генерации регионов (гипотез, где объекты)
Есть много методов просто выбран этот 2000 раз
- **Подгонка региона (resize) для подачи в CNN: 277×277**
Сеть имеет вход фиксированного размера – натренирована (ILSVRC2012 classification)
- **Классификация регионов (CNN + SVM)**
Krizhevsky Caffe выдаёт 4096 признаков + линейный SVM + дотренировка на данных
- **Оптимизация регионов с помощью регрессии**
Bounding-box regression

R-CNN: дотренировка



**Если 30% (важный коэффициент) пересечения
с истинным объектом «машина»,
то это класс машина**

**Классов = число объектов детектирования + 1 (фон)
– теперь столько нейронов в последнем слое**

batch = 32 окна с объектом + 96 с фоном
выбирается из 2000 регионов, сгенерированных по изображению
это решение проблемы дисбаланса (фона больше), ещё используют (не здесь) focal loss

R-CNN: регрессия

Для обучения только регионы $(\bar{x}, \bar{y}, \bar{h}, \bar{w})$

с 30% пересечением с истинным (x, y, h, w)

целевые значения – $\left(\frac{x - \bar{x}}{w}, \frac{y - \bar{y}}{h}, \ln(\bar{w} / w), \ln(\bar{h} / h) \right)$

линейная регрессия на CNN-признаках, MSE

ясно, как пересчитать в координаты регионов ответы

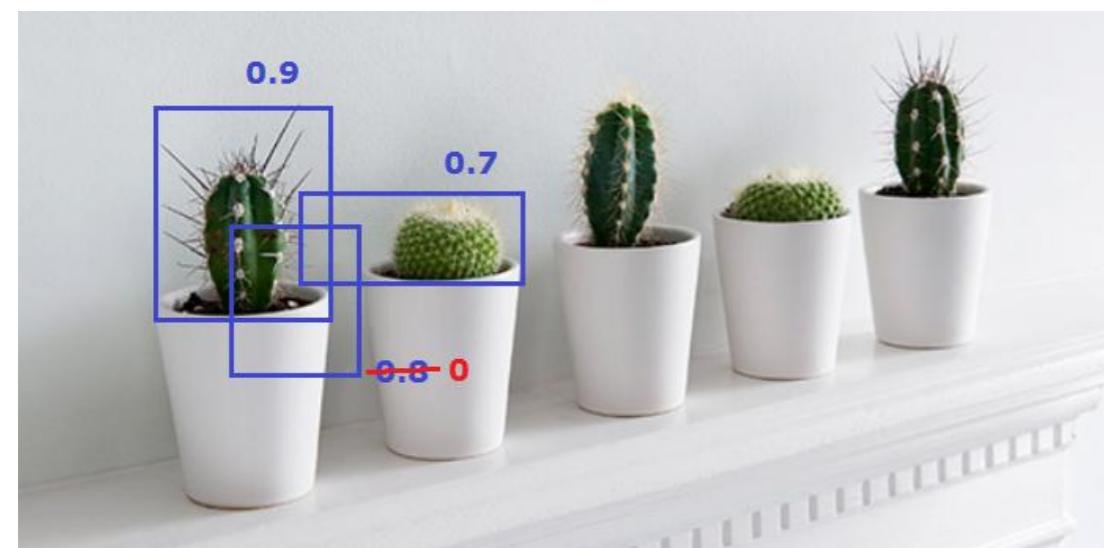
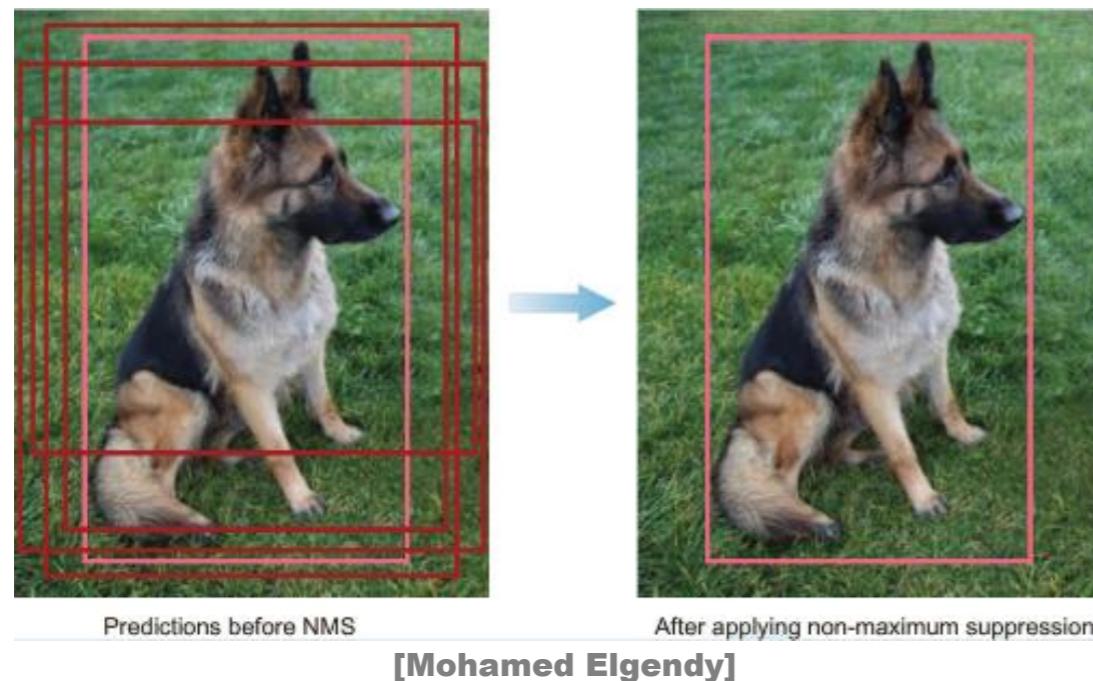
R-CNN: работа сети

- Выбор 2000 регионов (**Selective Search**)
- Регионы → **224×224**
- регионы → **CNN** → **SVM/Regression**
- non maximum suppression (**NMS**) **далше**

R-CNN: Недостатки

- Дообучиваем CNN (**log_loss**), потом ещё SVM, потом bounding-box-регрессию...
очень долгое обучение
 - Для каждого региона запускаем CNN

NMS = Non Maximum Suppression



Из семейства похожих рамок надо оставить одну для каждого объекта

Класс «Кактус»

**Упорядочим регионы: 0.9, 0.8, 0.7, ...
(ниже порога сразу занулим)**

Идём от MAX (i):

**Идём от текущего (j):
регион j имеет большое пересечение с i ⇒
зануляем**

Минутка кода: NMS

```
torchvision.ops.nms(boxes: torch.Tensor, scores: torch.Tensor, iou_threshold: float) →  
    torch.Tensor
```

[\[SOURCE\]](#)

Performs non-maximum suppression (NMS) on the boxes according to their intersection-over-union (IoU).

NMS iteratively removes lower scoring boxes which have an IoU greater than `iou_threshold` with another (higher scoring) box.

If multiple boxes have the exact same score and satisfy the IoU criterion with respect to a reference box, the selected box is not guaranteed to be the same between CPU and GPU. This is similar to the behavior of `argsort` in PyTorch when repeated values are present.

Parameters

- **boxes** (`Tensor[N, 4]`) – boxes to perform NMS on. They are expected to be in `(x1, y1, x2, y2)` format with `0 <= x1 < x2` and `0 <= y1 < y2`.
- **scores** (`Tensor[N]`) – scores for each one of the boxes
- **iou_threshold** (`float`) – discards all overlapping boxes with $\text{IoU} > \text{iou_threshold}$

Returns

`int64` tensor with the indices of the elements that have been kept by NMS, sorted in decreasing order of scores

Return type

`Tensor`

<https://pytorch.org/vision/stable/ops.html>

SPP-net: Spatial Pyramid Pooling

Разделение свёрточных слоёв для всех регионов одного изображения

Полносвязной НС нужен вход определённых размеров.

Чисто свёрточная НС может работать со входом любых размеров!

Идея – добавим к свёрточной сети слой: может генерировать фиксированный размер

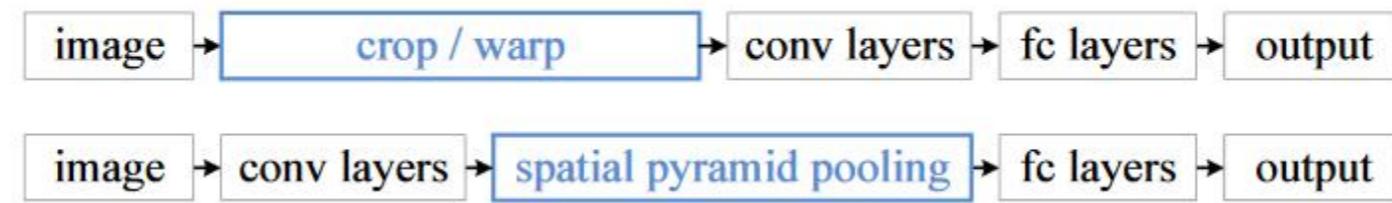


Figure 1: Top: cropping or warping to fit a fixed size. Middle: a conventional CNN. Bottom: our spatial pyramid pooling network structure.

SPP-net: The Spatial Pyramid Pooling Layer

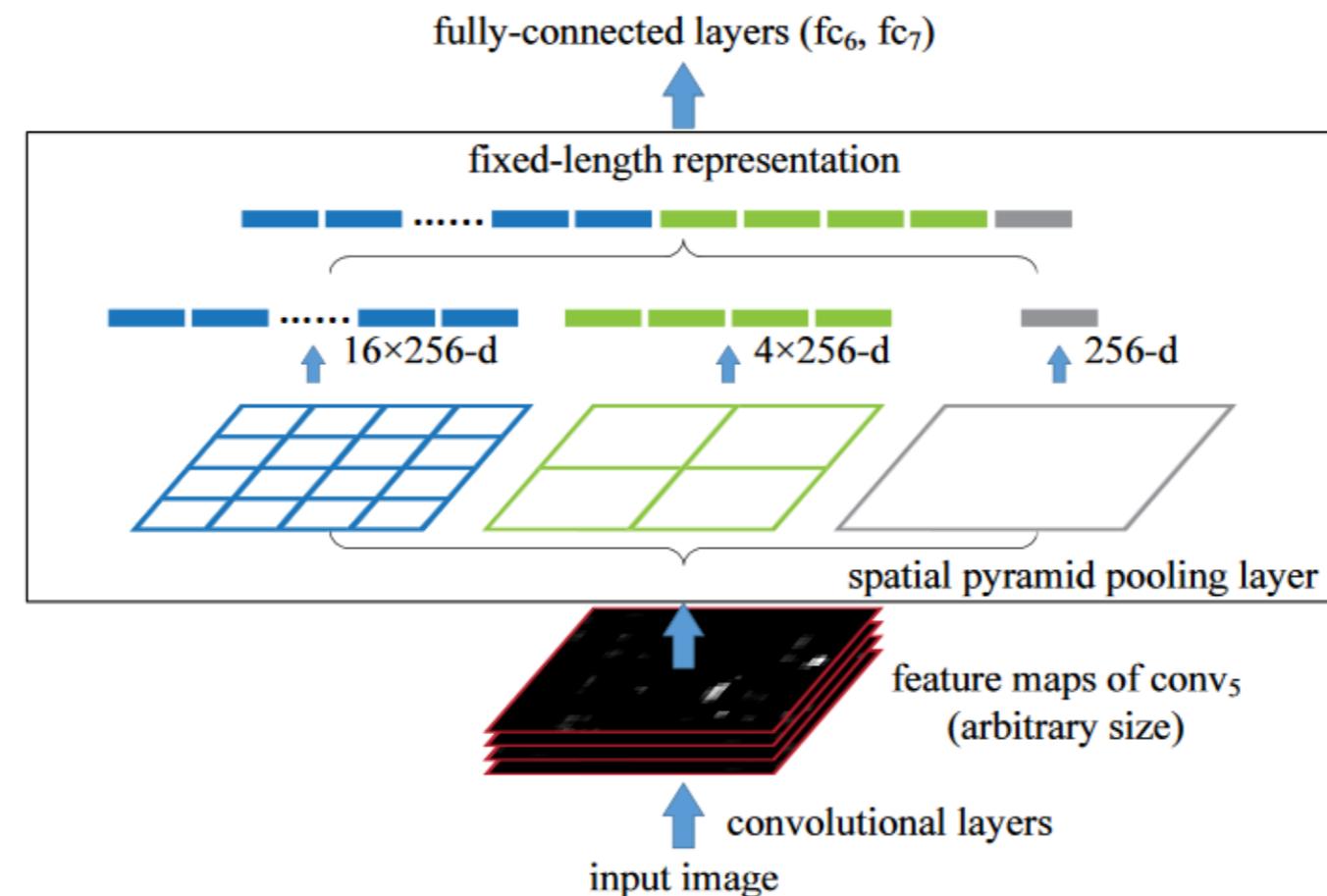
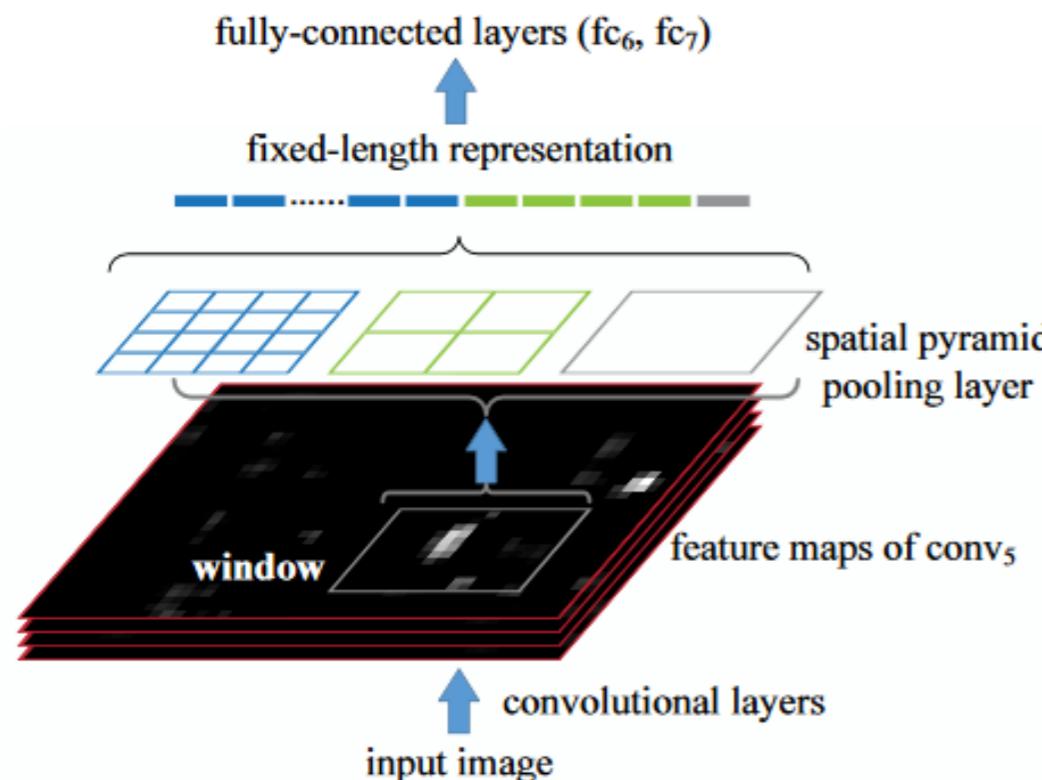


Figure 3: A network structure with a **spatial pyramid pooling layer**. Here 256 is the filter number of the conv₅ layer, and conv₅ is the last convolutional layer.

The Spatial Pyramid Pooling Layer



**Считаем CNN-представление для изображений
один раз**

**С помощью SPP-net для каждого региона
получаем признаковое представление
(фиксированной длины)**

Выигрыш по скорости 10x – 100x!

**Быстрое решение проблемы того, что регионы разных размеров!
+ сначала-то мы считаем CNN-представление!
а SPP-net просто эффективный способ перевода его в признаки
Сеть не работает на каждом регионе! Сразу на изображении!**

Минутка кода

```
import math
def spatial_pyramid_pool(self, previous_conv, num_sample, previous_conv_size, out_pool_size):
    """
    previous_conv: a tensor vector of previous convolution layer
    num_sample: an int number of image in the batch
    previous_conv_size: an int vector [height, width] of the matrix features size of previous convolution layer
    out_pool_size: a int vector of expected output size of max pooling layer
    returns: a tensor vector with shape [1 x n] is the concentration of multi-level pooling
    """
    for i in range(len(out_pool_size)):
        h_wid = int(math.ceil(previous_conv_size[0] / out_pool_size[i]))
        w_wid = int(math.ceil(previous_conv_size[1] / out_pool_size[i]))
        h_pad = (h_wid*out_pool_size[i] - previous_conv_size[0] + 1)/2
        w_pad = (w_wid*out_pool_size[i] - previous_conv_size[1] + 1)/2
        maxpool = nn.MaxPool2d((h_wid, w_wid), stride=(h_wid, w_wid), padding=(h_pad, w_pad))
        x = maxpool(previous_conv)
        if(i == 0):
            spp = x.view(num_sample, -1)
        else:
            spp = torch.cat((spp, x.view(num_sample, -1)), 1)
    return spp
```

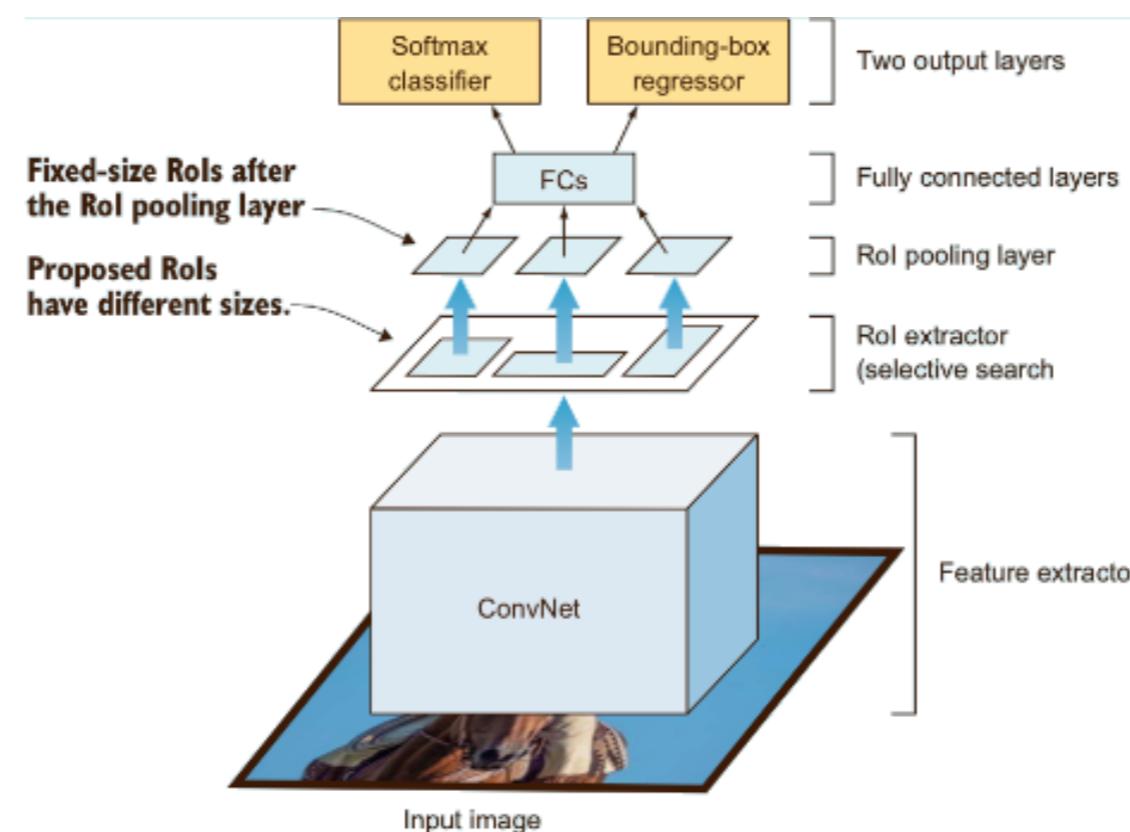
https://github.com/yueruchen/sppnet-pytorch/blob/master/spp_layer.py

есть адаптивный пулинг – выход конкретно размера!

Замечание: есть ещё Temporal Pyramid Pooling (т.е. 1D)

Fast R-CNN

Fast R-CNN = R-CNN + SPP (только по-другому: RoI-pool) + регрессию встроили в НС
Обучение в одну стадию (раньше CNN → SVM → regression)!



Вход: изображение + параметры регионов (регионы тоже с помощью SS)
end2end: ошибка = сумма ошибок классификатора и регрессора

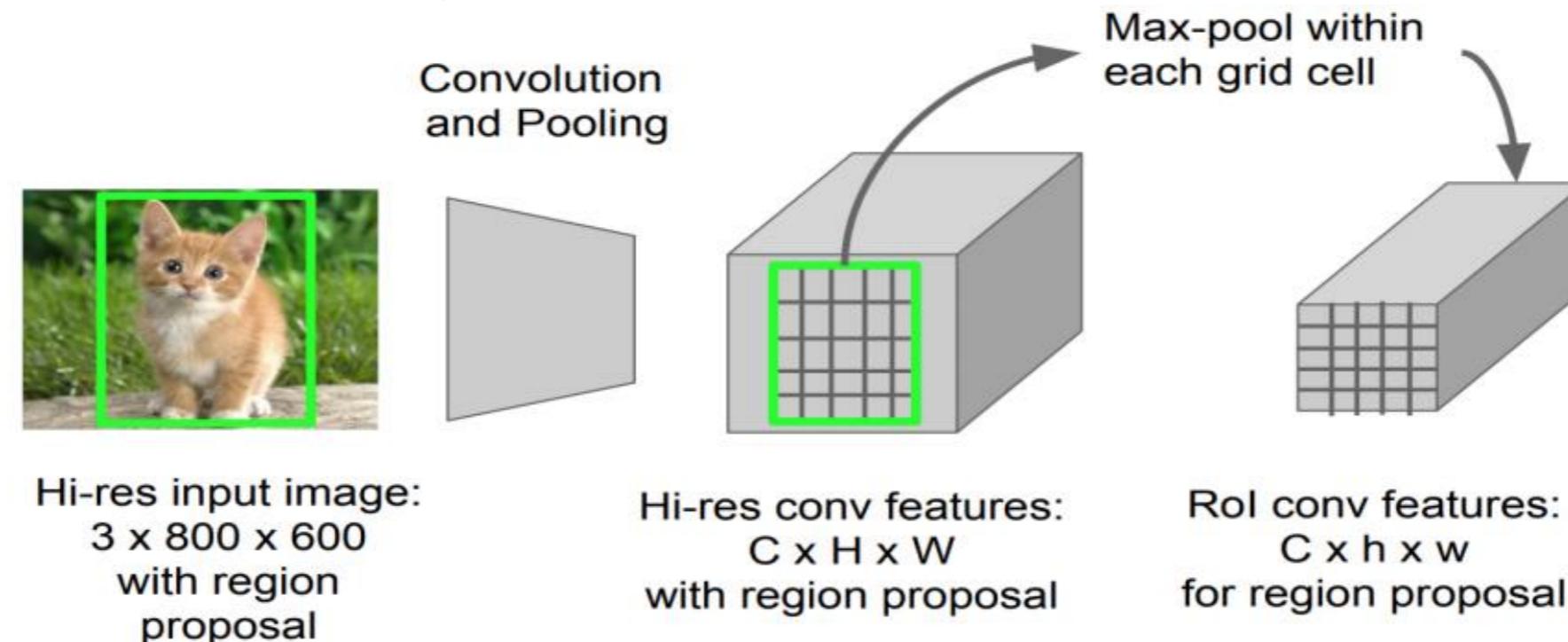
Fast R-CNN

**пропускаем через CNN-сеть изображение целиком
(а не каждый регион в отдельности, как раньше)**

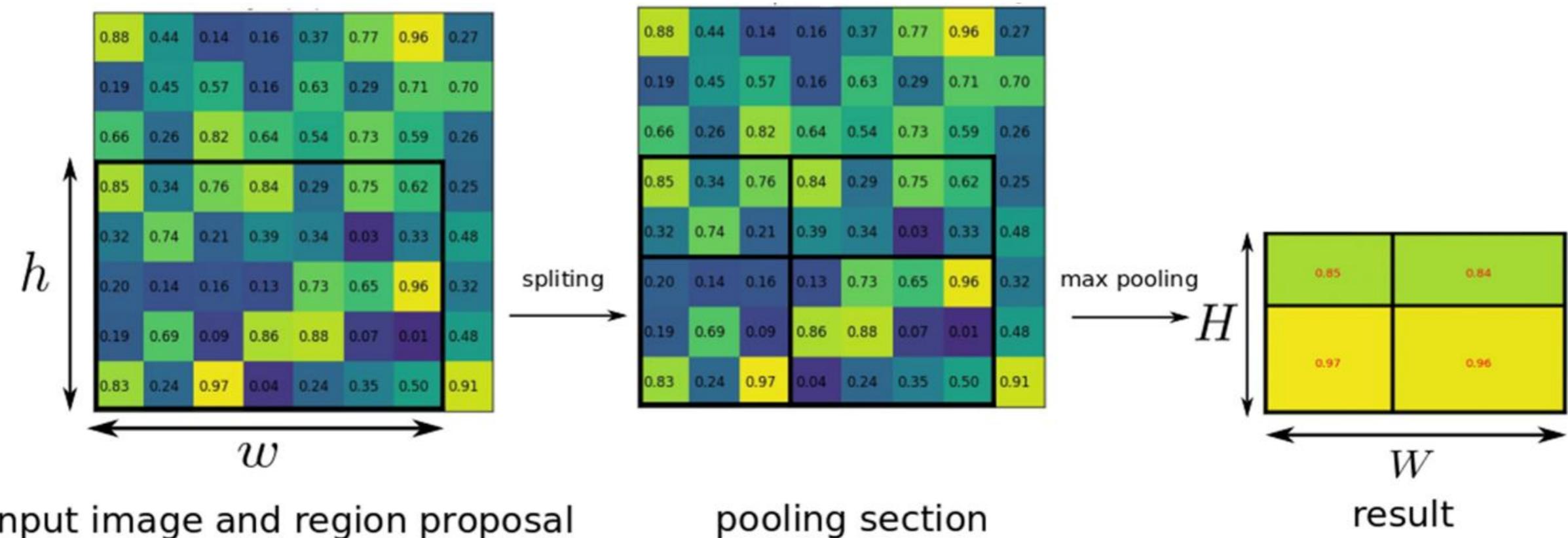
а регионы накладываются на полученную карту признаков

признаки из разных регионов приводятся в одну размерность с помощью ROI Pooling

$H \times W \rightarrow$ пулинг по сетке $H/7 \times W/7 \rightarrow$ сетка 7×7



Fast R-CNN: ROI Pooling Layer

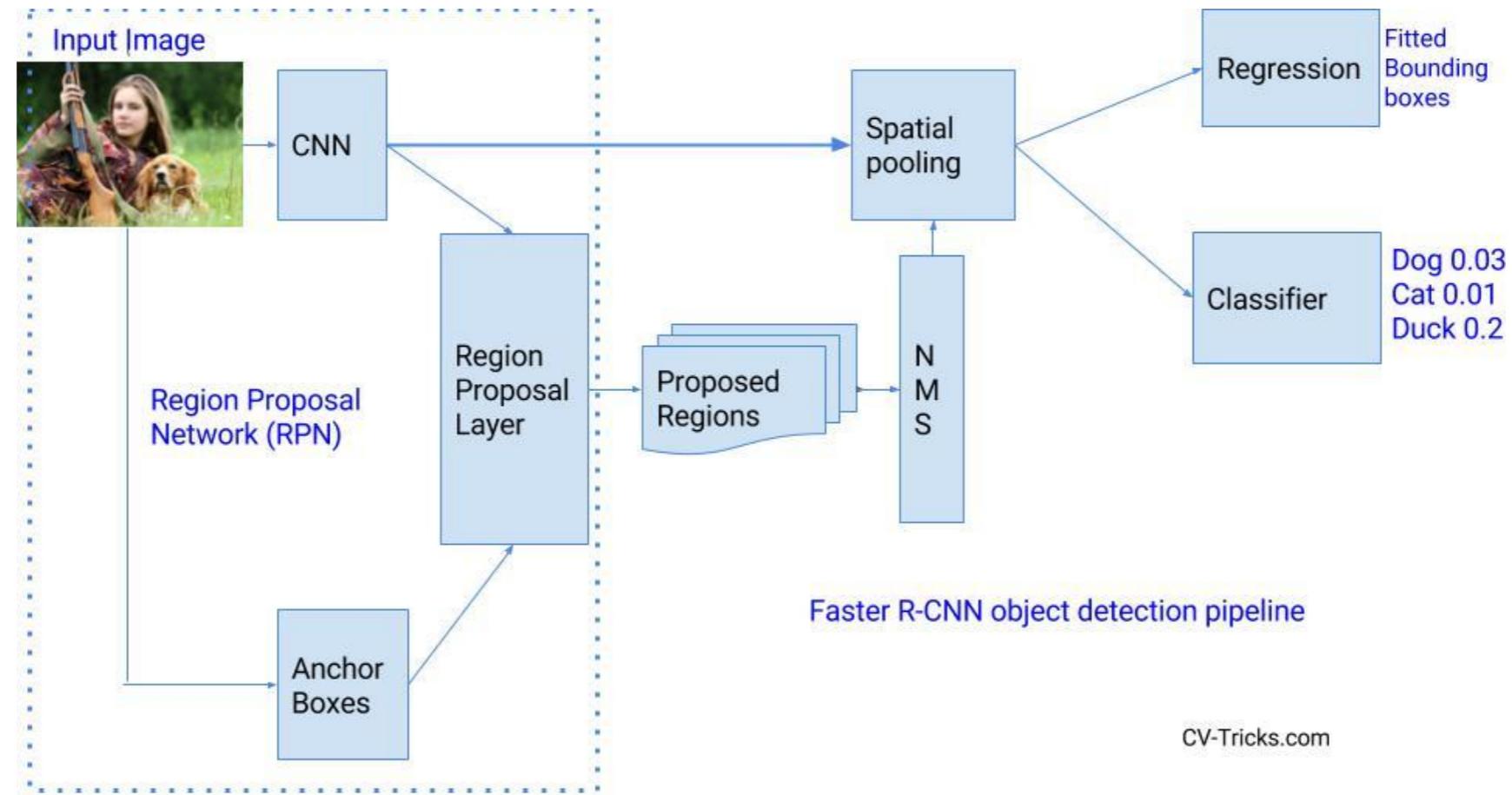


~ один уровень SPP-слоя в SPPNet

+ небольшая тонкость с обучением

(ошибка = сумма ошибок классификации и регрессии)

Faster R-CNN: SS заменяем на Region Proposal Network (RPN)



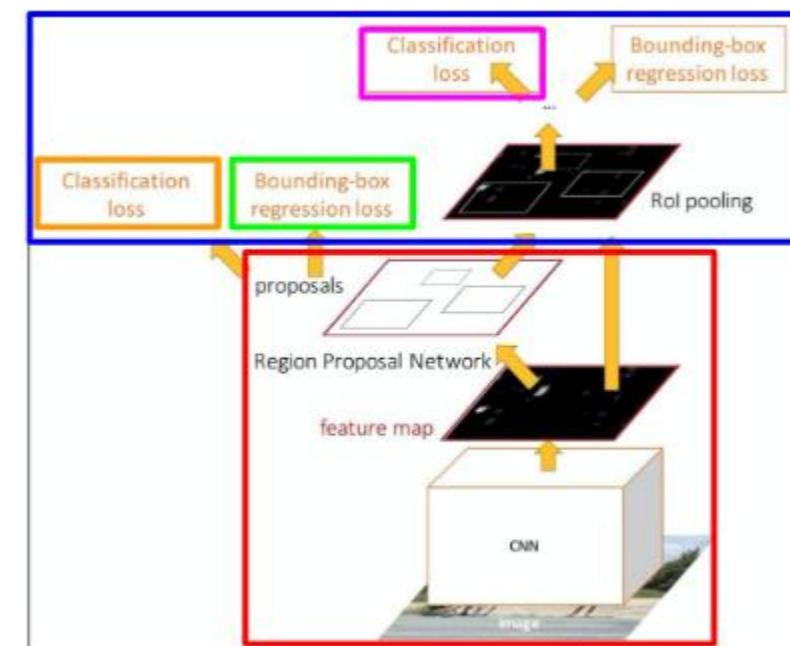
**Полностью end2end: «где смотреть» решает НС
DL-генерация регионов**

картинка → свёрточные слои, потом Region Proposal Network (RPN)

Faster R-CNN: Region Proposal Network (RPN)

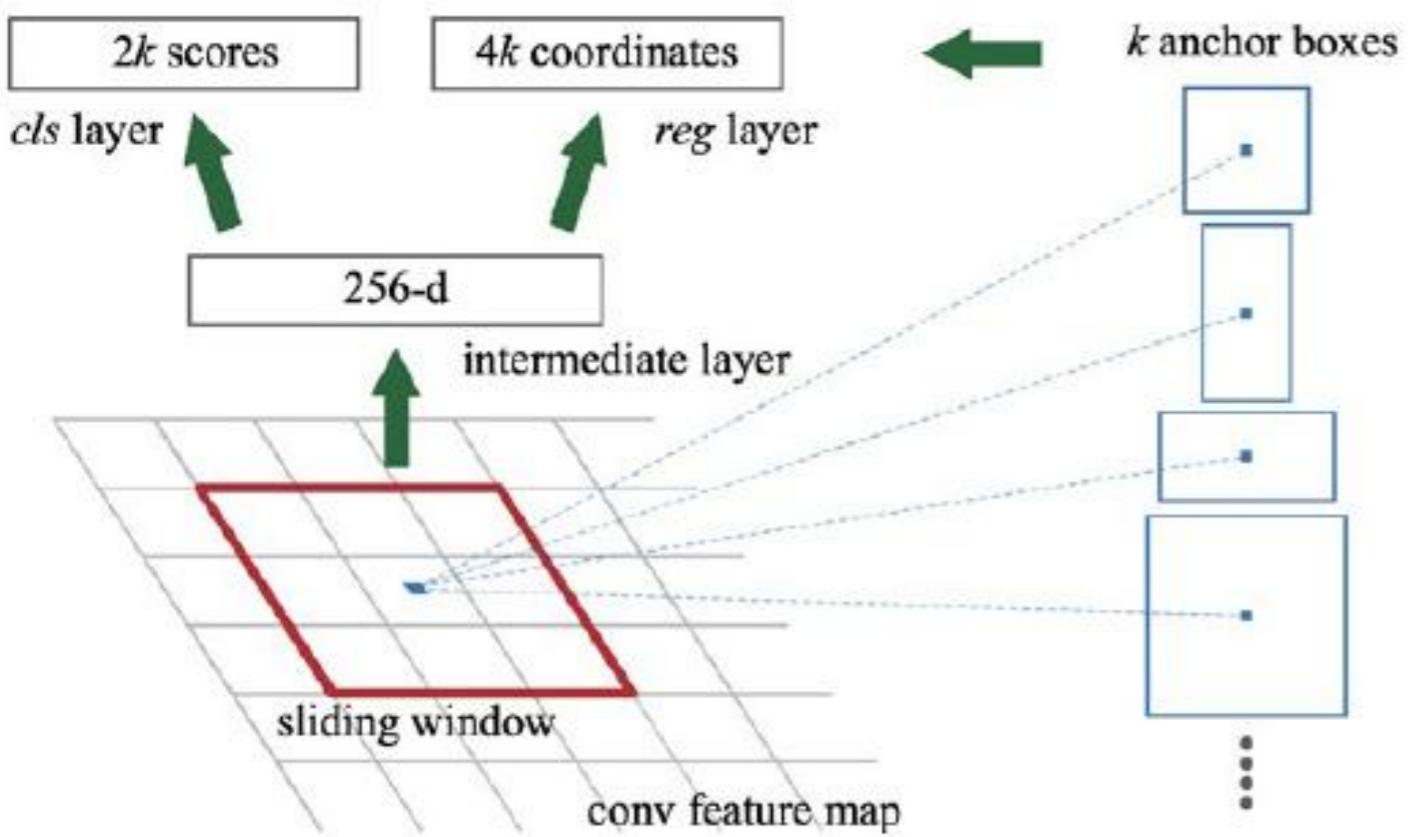
Идея: для большого набора якорных регионов (anchor bbox)

- определить, где содержится объект (ответ: да / нет)
 - для топ300 регионов уточнить границы



[Stanford CS231n]

Faster R-CNN: Region Proposal Network (RPN)



хотим, чтобы генерация регионов –
обучаемая процедура

проходимся окном 3×3 и генерируем к
кандидатов на роль регионов
к **anchor boxes** – какие-то заданные
шаблоны (размеры 128, 256, 512 и
соотношения 1:1, 1:2, 2:1)
каждая точка – регион

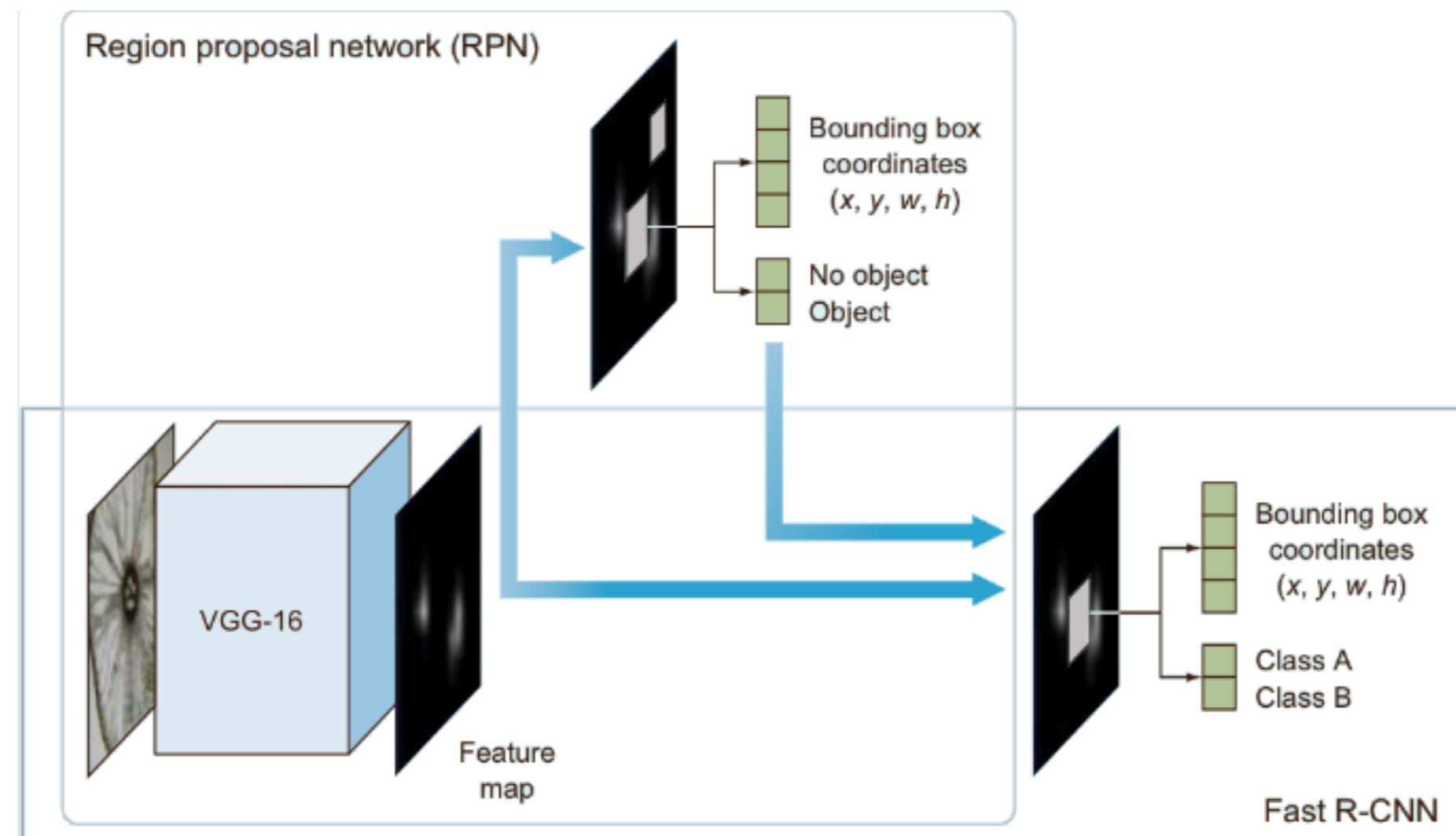
регионы с большой вероятностью
содержания объекта передаются
далее для уточнения региона и
детектирования

2k scores (2 = объект или нет)

Для $W \times H$ -карты признаков получается $W \cdot H \cdot k$ регионов (anchors)

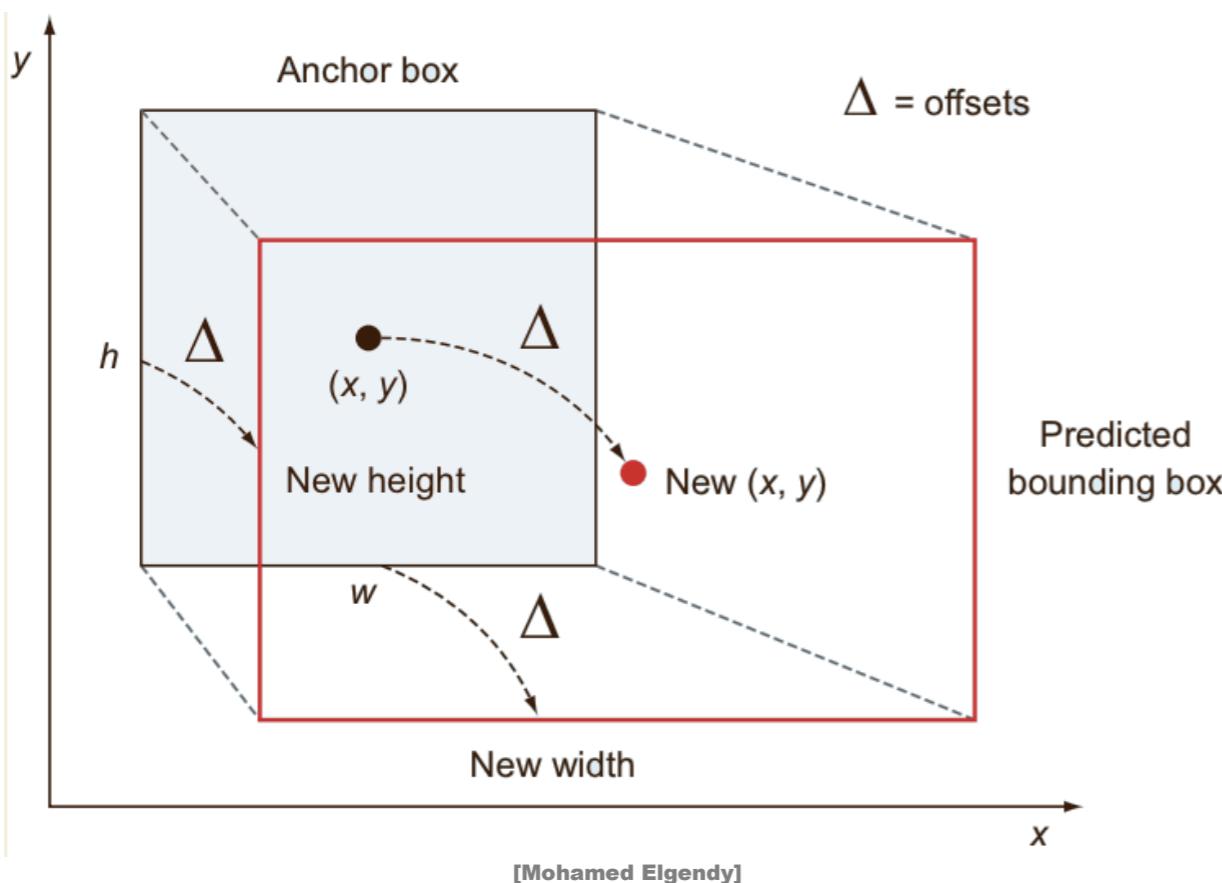
Faster R-CNN: Region Proposal Network (RPN)

Ошибка = л/к ошибок классификации и регрессии



Faster R-CNN: Bounding Box Regression

Предсказываем смещение относительно якоря



Как происходит «приписывание» –
какой Anchor box к какому региону приводить:

- для каждого региона выбрать с $\max \text{IoU}$
- если $\text{IoU} > 0.7$

$$\sum_i (x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 + (\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2$$

Минутка кода: RPN (не самая лучшая реализация)

```
class RPN(nn.Module):  
  
    def __init__(self, in_channels=512, mid_channels=512, n_anchor=9):  
        super(RPN, self).__init__()  
        self.mid_channels = mid_channels  
        self.in_channels = in_channels # depends on the output feature map. in vgg 16 it is equal to 512  
        self.n_anchor = n_anchor # Number of anchors at each location  
        self.conv1 = nn.Conv2d(self.in_channels, self.mid_channels, 3, 1, 1)  
        self.reg_layer = nn.Conv2d(mid_channels, n_anchor *4, 1, 1, 0)  
        self.cls_layer = nn.Conv2d(mid_channels, n_anchor *2, 1, 1, 0)  
  
        self.conv1.weight.data.normal_(0, 0.01) # conv sliding layer  
        self.conv1.bias.data.zero_()  
        self.reg_layer.weight.data.normal_(0, 0.01) # Regression layer  
        self.reg_layer.bias.data.zero_()  
        self.cls_layer.weight.data.normal_(0, 0.01) # classification layer  
        self.cls_layer.bias.data.zero_()  
  
    def forward(self, k):  
        bat_num = k.shape[0]  
        x = self.conv1(k)  
        pred_anchor_locs = self.reg_layer(x)  
        pred_cls_scores = self.cls_layer(x)  
  
        pred_anchor_locs = pred_anchor_locs.permute(0, 2, 3, 1).contiguous().view(bat_num, -1, 4)  
        pred_cls_scores = pred_cls_scores.permute(0, 2, 3, 1).contiguous()  
        objectness_score = pred_cls_scores.view(bat_num, 50, 50, 9, 2)[:, :, :, :, 1].contiguous().view(bat_num, -1)  
        pred_cls_scores = pred_cls_scores.view(bat_num, -1, 2)  
  
        return pred_anchor_locs, pred_cls_scores, objectness_score
```

<https://github.com/sorg20/RPN/blob/master/rpn.ipynb>

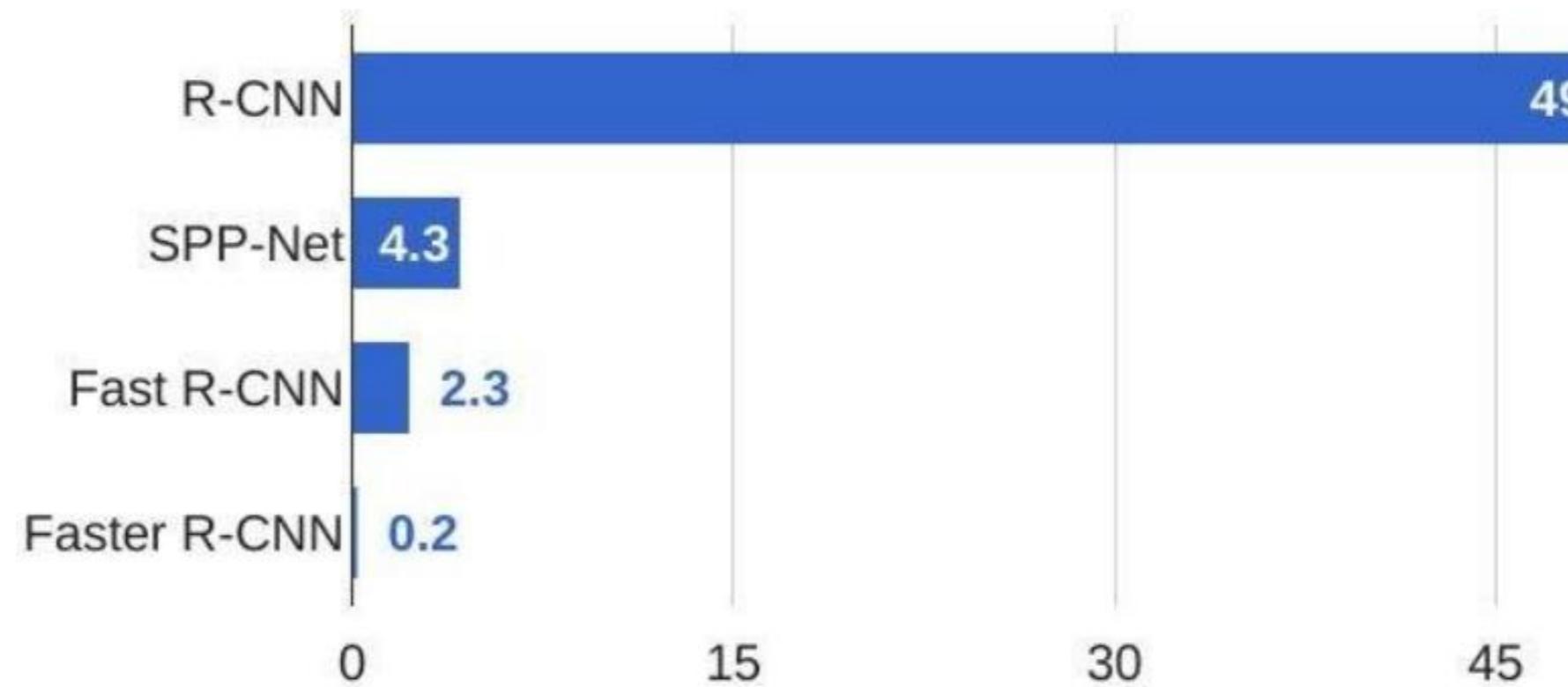
Faster R-CNN: обучение

Обучение довольно нетривиальное
(сначала RPN-часть, потом Fast-RCNN-часть – можно итерационно)
ошибка = ошибка классификации + ошибка регрессии

Faster R-CNN: Работа (inference)

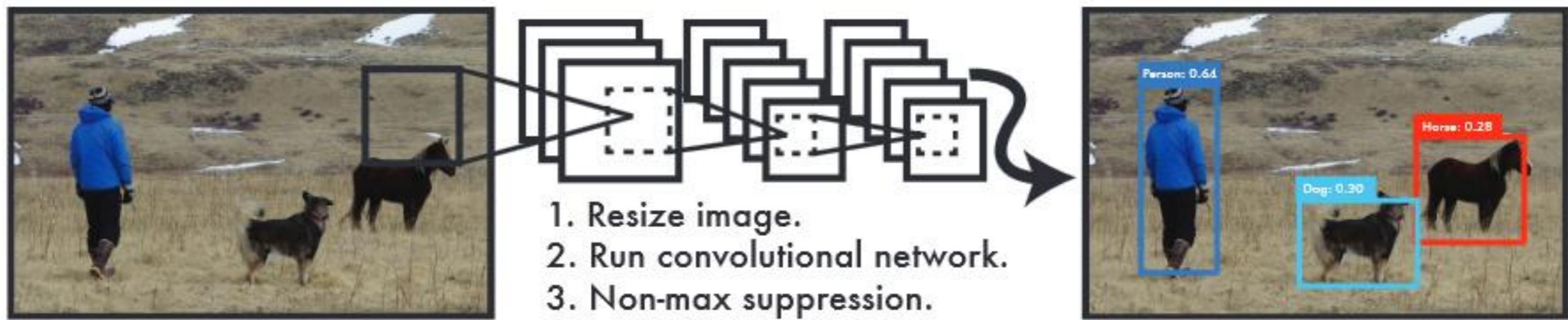
- **изображение → CNN + RPN**
- **оставляем 6000 перспективных регионов**
- **удаляем очень узкие и выходящие за изображения**
- **NMS с большим порогом IoU=0.9**
- **регионы → RoI pooling + регрессия + классификация**
- **NMS по классам**

Скорость *-R-CNN сетей



указано сек / на 1 изображение

YOLO: You only Live Look Once (2016)



- **Изменение масштаба → 448×448**
- **CNN (одна!)**
- **Детектирование – задача регрессии;)**
- **Пороговое принятие решения**
- **Запуск сразу на всём изображении – очень быстро**

Сеть видит изображение целиком, а не регионами

Очень быстрая, но точность хуже (особенно для мелких объектов)

https://pjreddie.com/media/files/papers/yolo_1.pdf

YOLO: изображение → тензор (с вероятностями классов и координатами рамок)

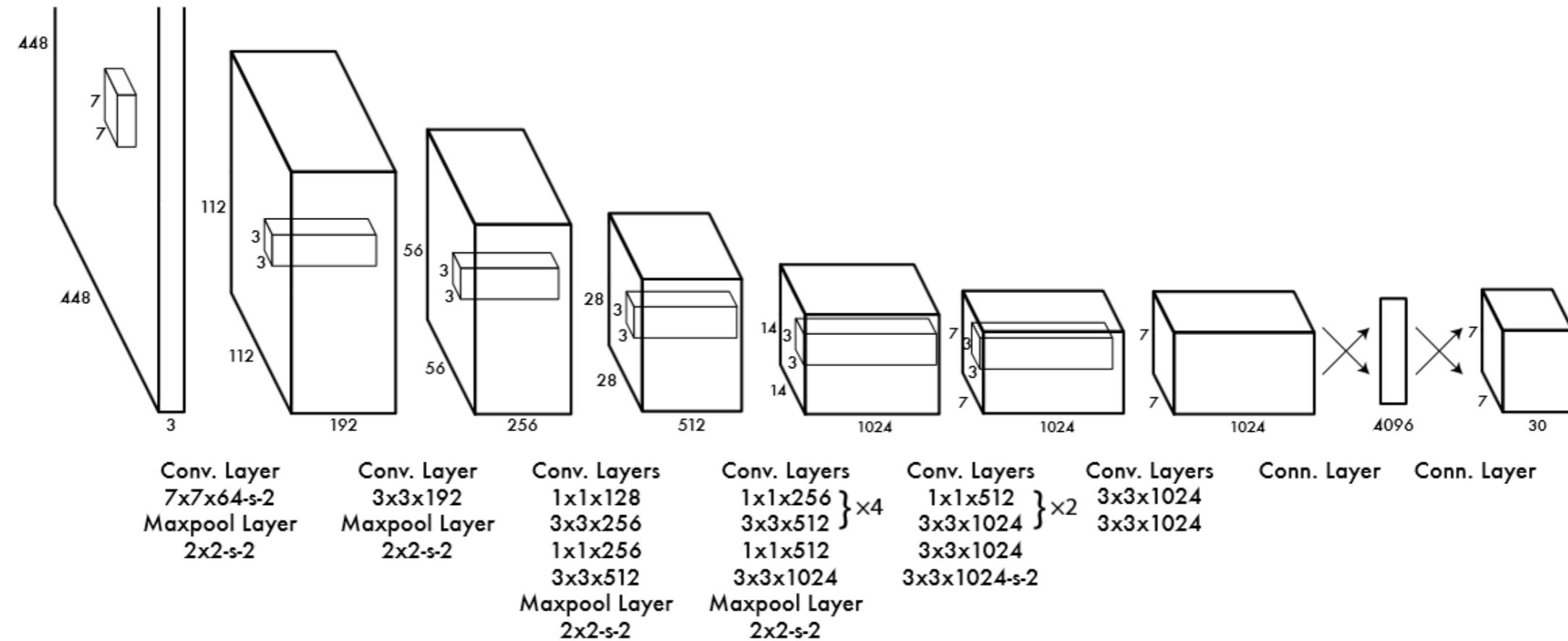
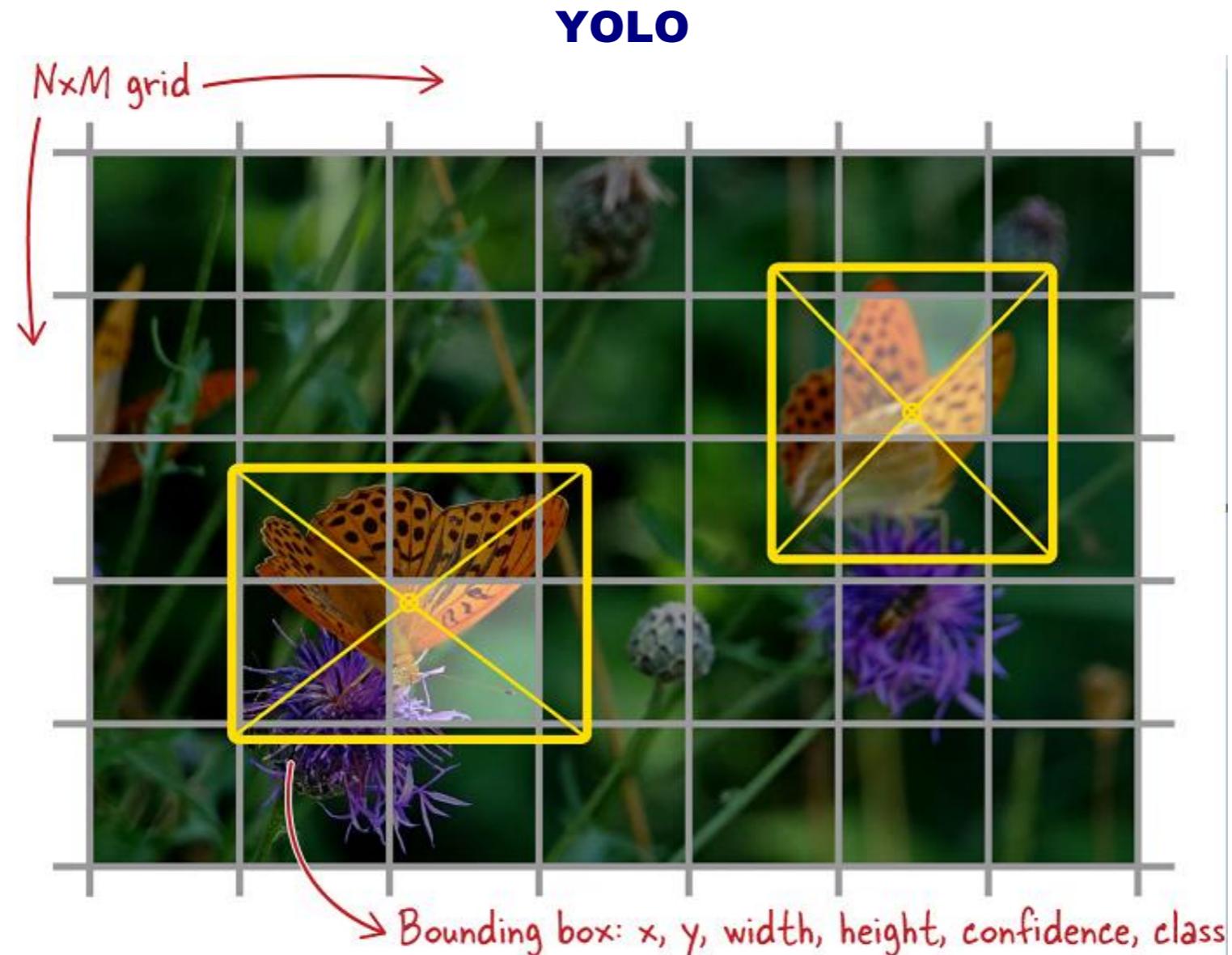


Figure 3: The Architecture. Our detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating 1×1 convolutional layers reduce the features space from preceding layers. We pretrain the convolutional layers on the ImageNet classification task at half the resolution (224×224 input image) and then double the resolution for detection.

Классика: свёртки + полносвязные слои



предсказание региона (соотв. ячейке) – предсказание 6 чисел
[Practical Machine Learning for Computer Vision]

YOLO

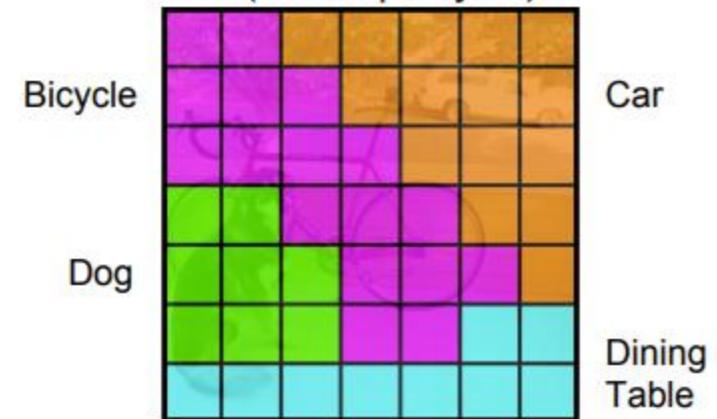
Split the image into a grid



P(Object)



P(Class | Object)



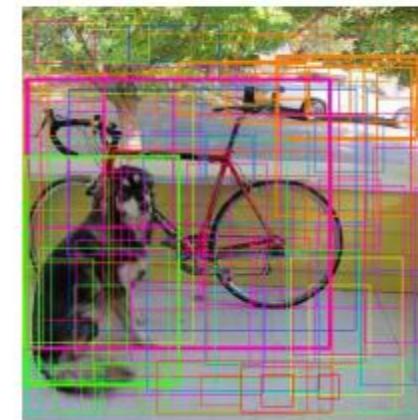
Bicycle

Car

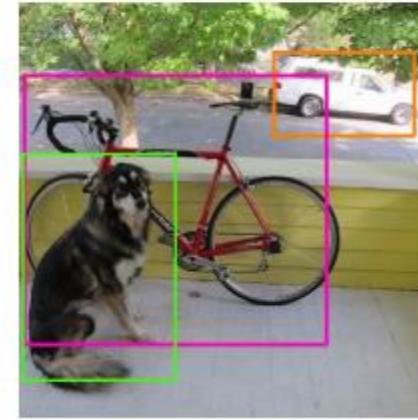
Dog

Dining
Table

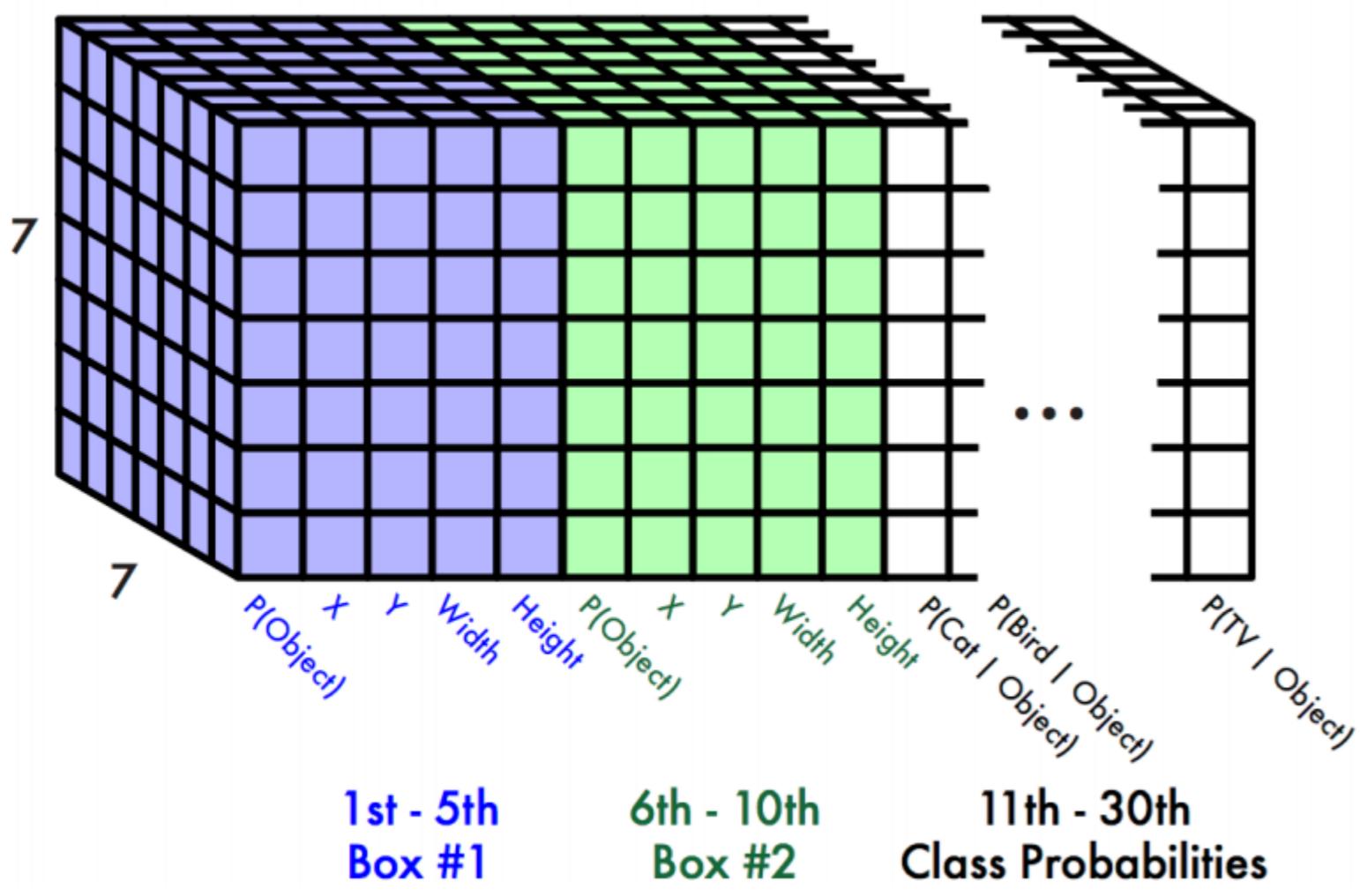
Combine the box and class predictions



Finally do NMS and threshold detections

**каждая клетка – свой класс, для каждого бокса своя вероятность, что там есть объект**

YOLO: выход модели



В тензоре $7 \times 7 \times 30$ закодированы
все регионы оценки за классы

7×7 – это сетка;)

$30 = 5 + 5 +$
(почему-то 2 региона для \forall ячейки)
+ 20 (# классов ~ Pascal VOC)

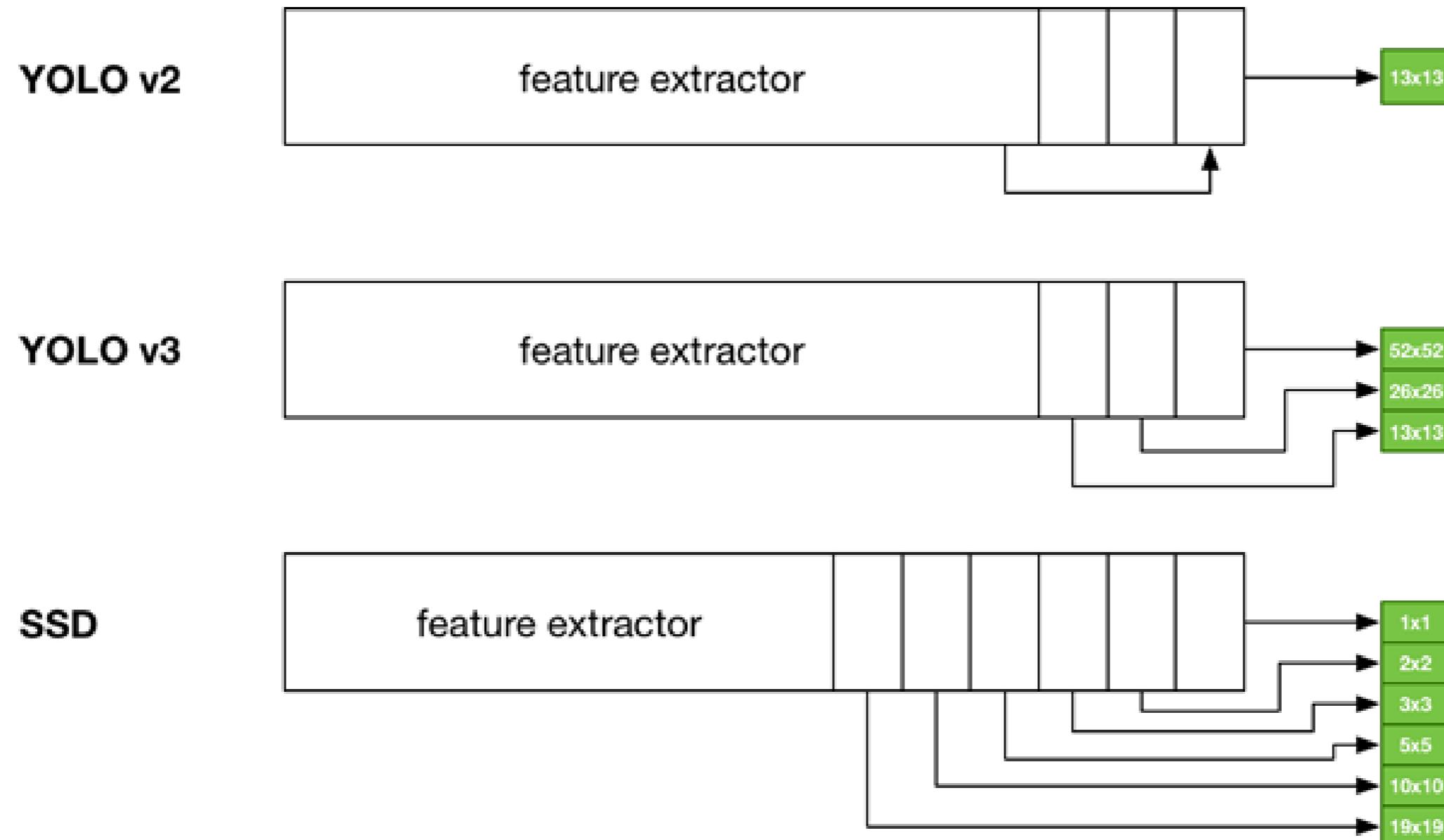
$5 = |(x, y, w, h, c)|$
 **x, y – координаты в центре соотв.
ячейки**
 **c – уверенность, что регион
правильный**

YOLO

- **Предположение: в каждой ячейке центры максимум 2x классов**
- **Первые 20 свёрточных слоёв предтренированы на ImageNet**
- **448 × 448 вместо 224 × 224,**
- **Leaky ReLU (везде)**
- **1 × 1 bottleneck filters**
- **dropout после 1го полносвязного слоя**
- **квадратичная ошибка (для координат, уверенности и оценок!)**
- **борьба с очень большими регионами и пустыми регионами (...)**
- **momentum 0.9, decay $5 \cdot 10^{-4}$**
- **аугментация (scaling, translation, HSV transformation)**
- **быстрая 45 fps, YOLO-tiny – 155 fps, хотя качество и хуже SOTA**
- **проблема с детекцией маленьких объектов и толпы (ограничение 2 объекта в ячейке)**

J. Redmon, S. Divvala, R. Girshick, A. Farhadi «You Only Look Once: Unified, Real-Time Object Detection» <https://arxiv.org/abs/1506.02640>

Эволюция YOLO + SSD



<https://machinethink.net/blog/object-detection/>

YOLOv2 + YOLO9000 (2016)

YOLO9000 модификация на базе архитектуры YOLOv2 (были SoTA)

усложнили, предобучили на ImageNet-е на разрешении 448×448 + COCO

убрали полносвязные слои, теперь свёртки + якоря (идея из RPN)

якоря (их формы и размеры) с помощью k-means
(для определения форм специфичных для данного датасета)

детектирование ~9000 объектов (классов)
(качество не очень, но быстрая)

определение более 1000 регионов у YOLO Было 98

иерархия классов

BN-слои

Redmon and Farhadi «YOLO9000: Better, Faster, Stronger» // <https://arxiv.org/pdf/1612.08242.pdf>

YOLOv2: кластеризация якорей

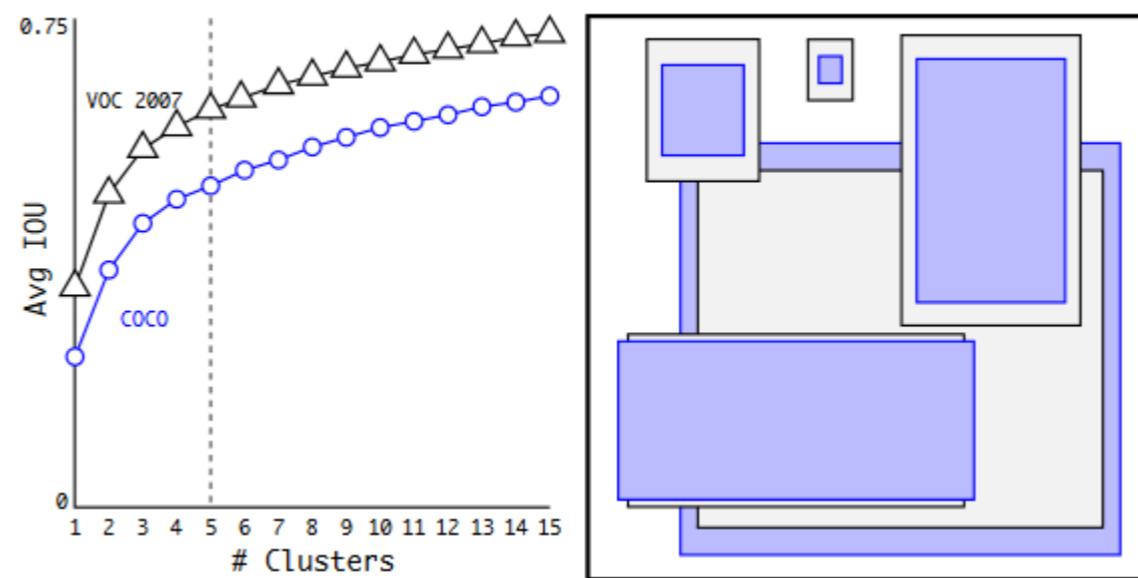


Figure 2: Clustering box dimensions on VOC and COCO. We run k-means clustering on the dimensions of bounding boxes to get good priors for our model. The left image shows the average IOU we get with various choices for k . We find that $k = 5$ gives a good tradeoff for recall vs. complexity of the model. The right image shows the relative centroids for VOC and COCO. Both sets of priors favor thinner, taller boxes while COCO has greater variation in size than VOC.

YOLOv2: иерархия классов

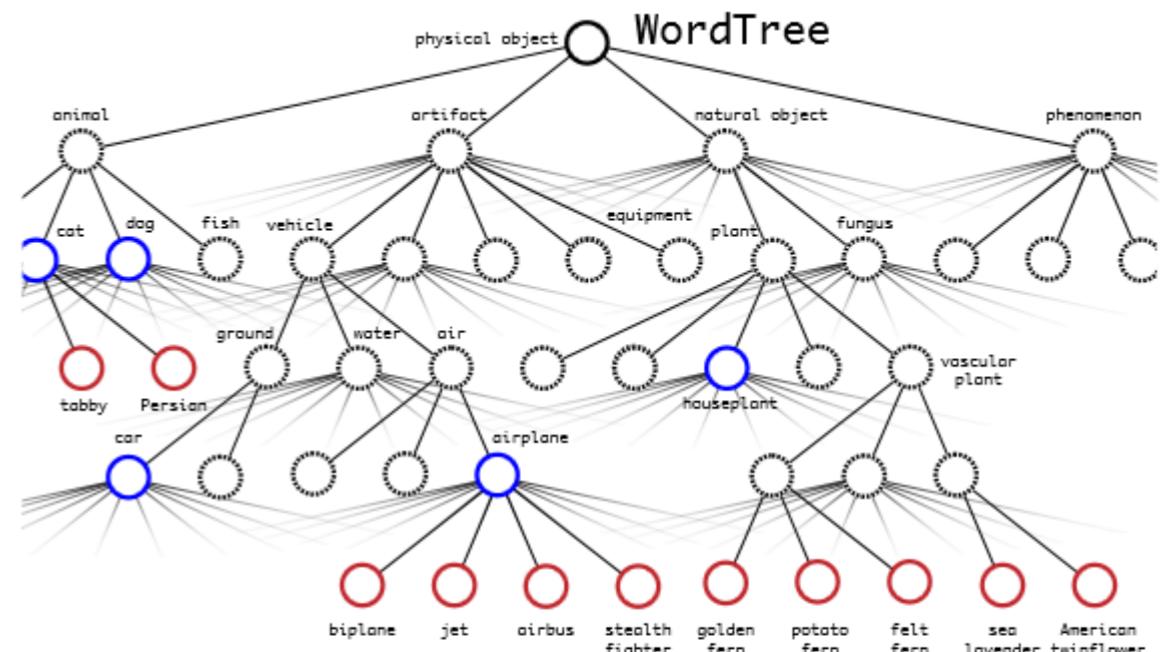
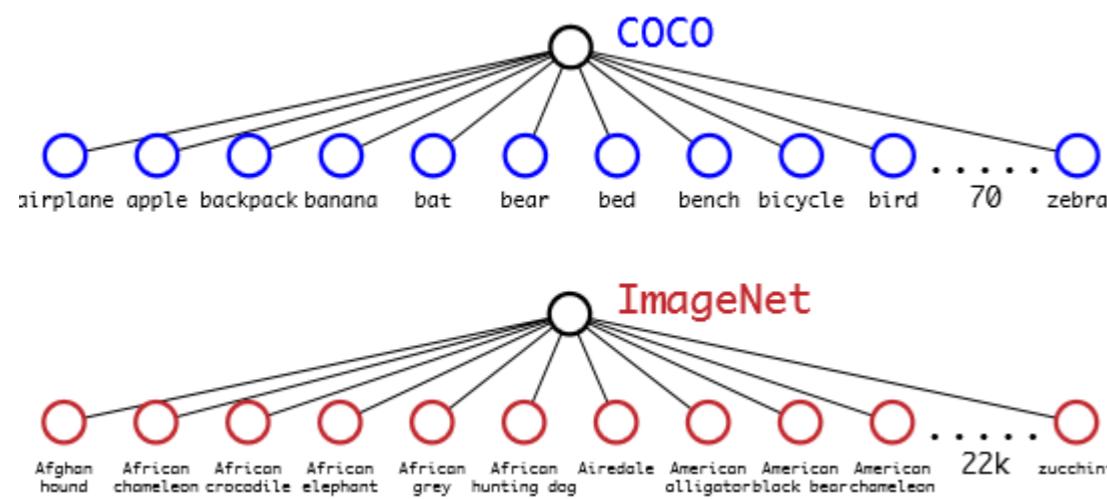
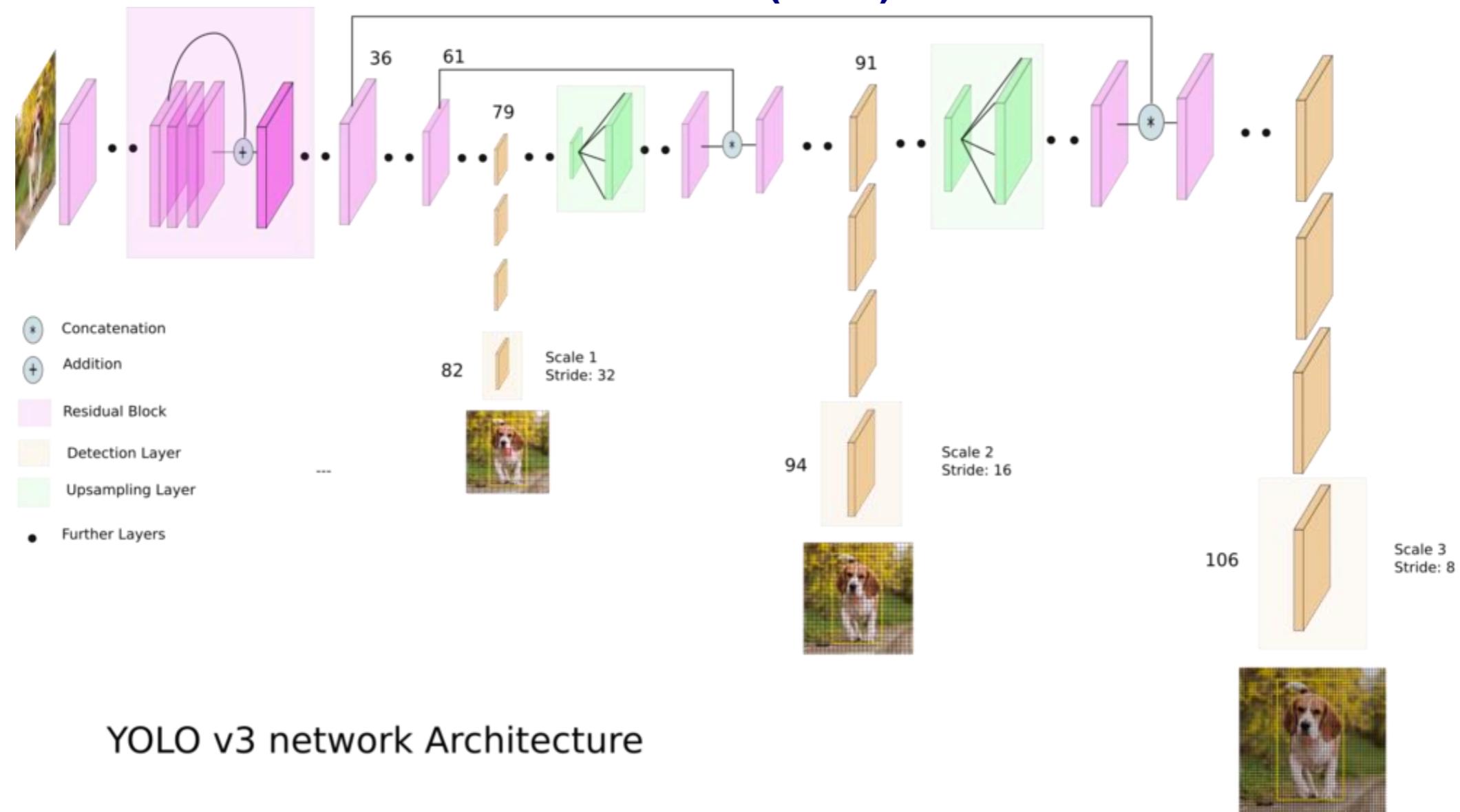


Figure 6: Combining datasets using WordTree hierarchy. Using the WordNet concept graph we build a hierarchical tree of visual concepts. Then we can merge datasets together by mapping the classes in the dataset to synsets in the tree. This is a simplified view of WordTree for illustration purposes.

WordTree для обучения на разных датасетах (с разными классами)

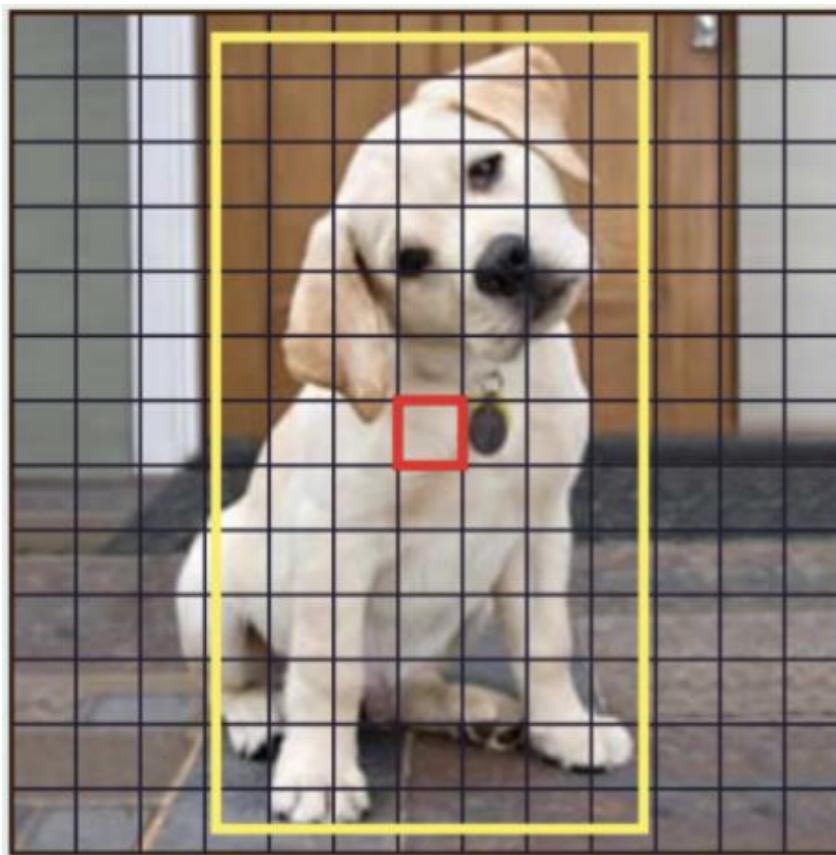
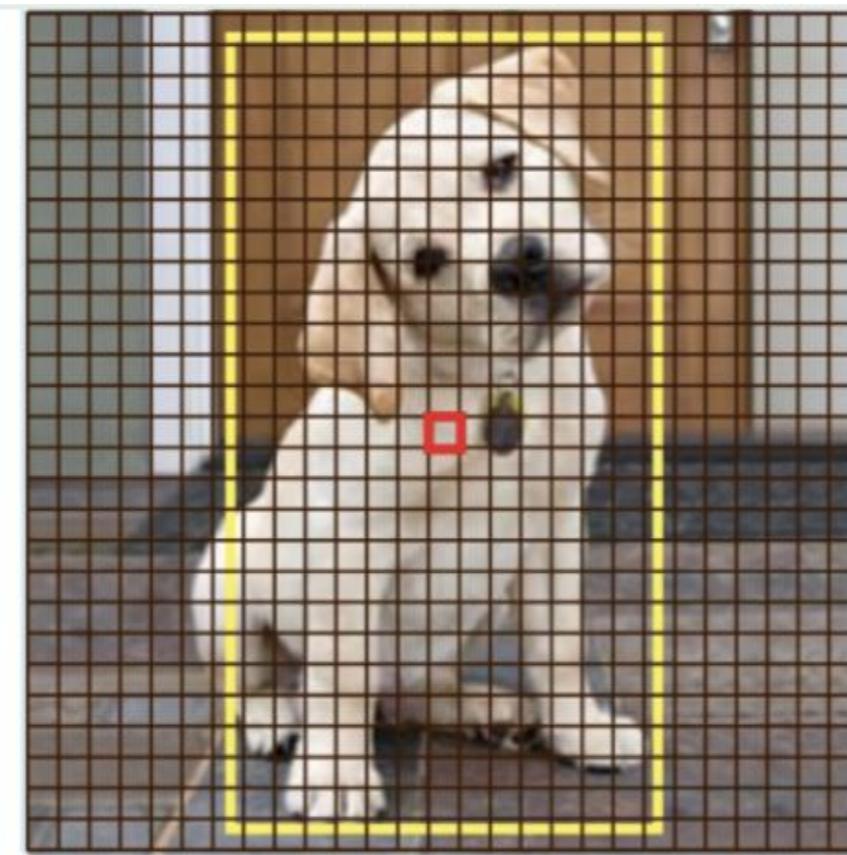
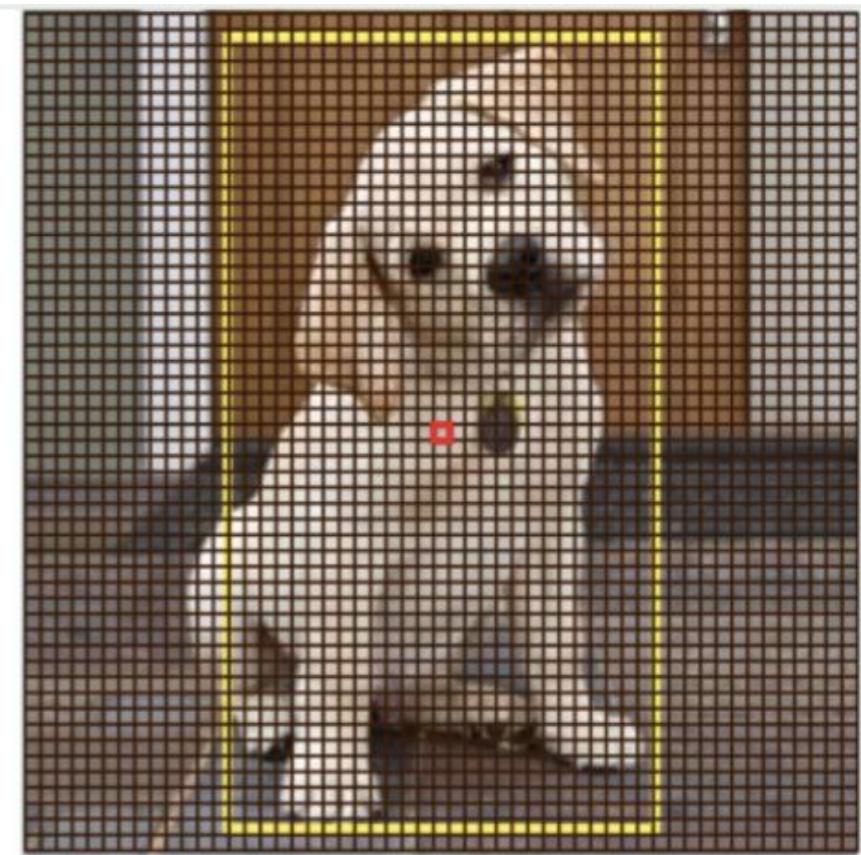
YOLOv3 (2018)



YOLO v3 network Architecture

**на выходе есть три слоя – для обнаружения объектов разного размера,
нет pool (есть stride)**

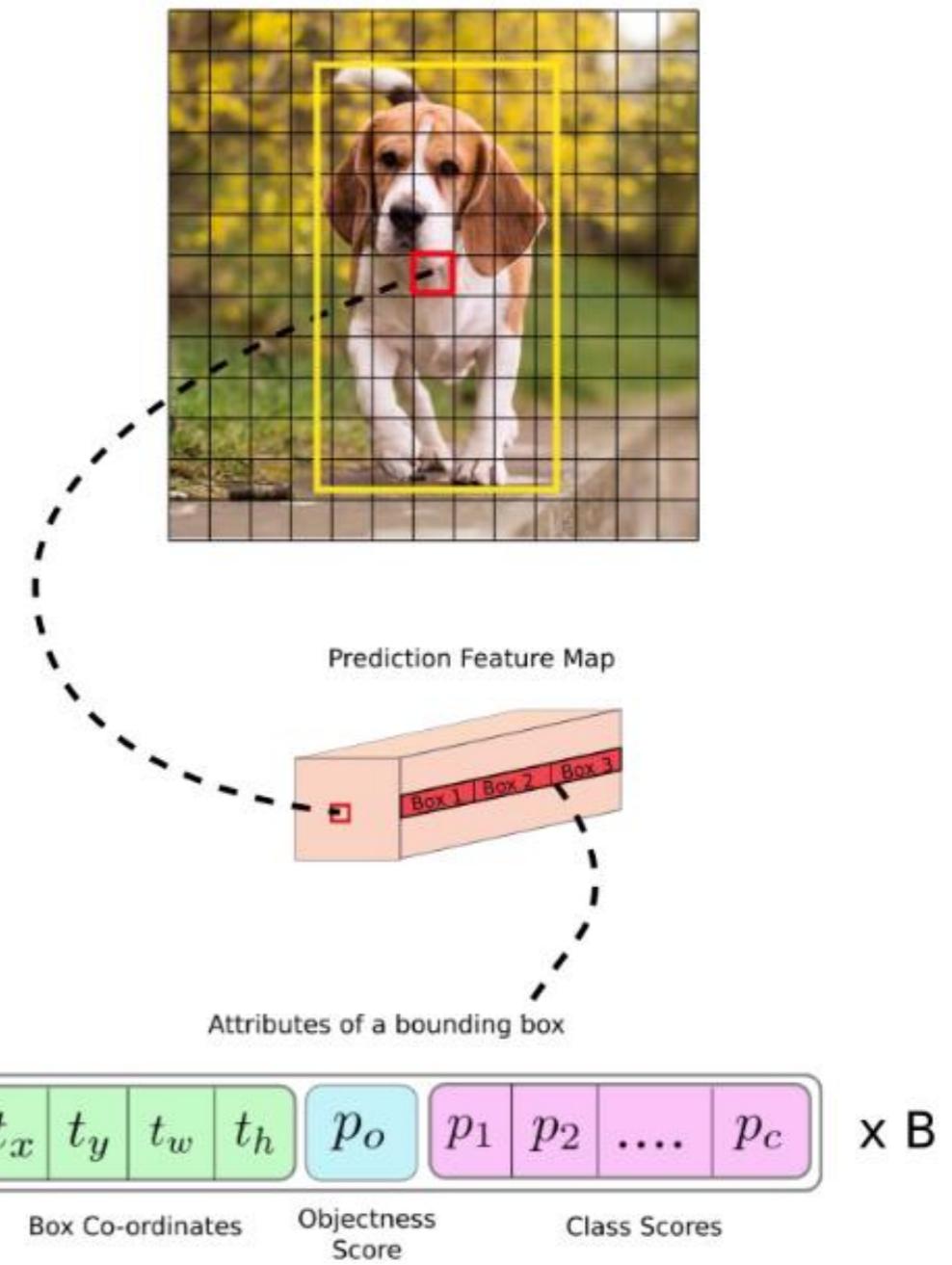
YOLOv3

 13×13  26×26  52×52

работа на разных разрешениях
[Mohamed Elgendi]

YOLOv3

- на вход сети изображение**
- проходит через предобученную
(на классификации)
свёрточную часть Darknet-53**
- на выходе тензор размера $13 \times 13 \times B(5+C)$**
- выучаются смещение якорей**
- отсечение по порогу objectness score
и NMS (на всех размерах)**



Single Shot MultiBox Detector (SSD)

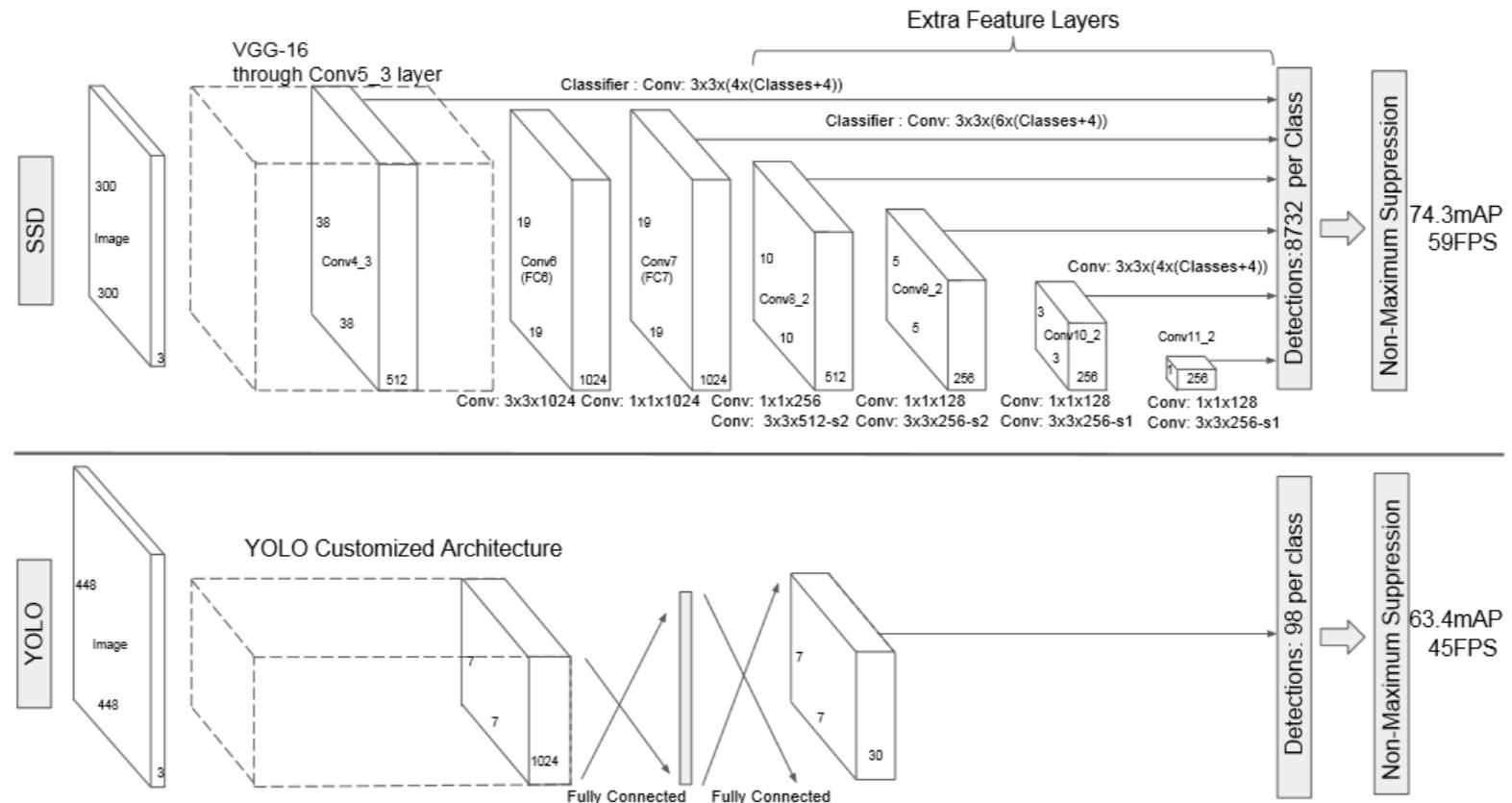


Fig. 2: A comparison between two single shot detection models: SSD and YOLO [5]. Our SSD model adds several feature layers to the end of a base network, which predict the offsets to default boxes of different scales and aspect ratios and their associated confidences. SSD with a 300×300 input size significantly outperforms its 448×448 YOLO counterpart in accuracy on VOC2007 test while also improving the speed.

[W. Liu и др. <https://arxiv.org/pdf/1512.02325.pdf>]

Single Shot MultiBox Detector (SSD)

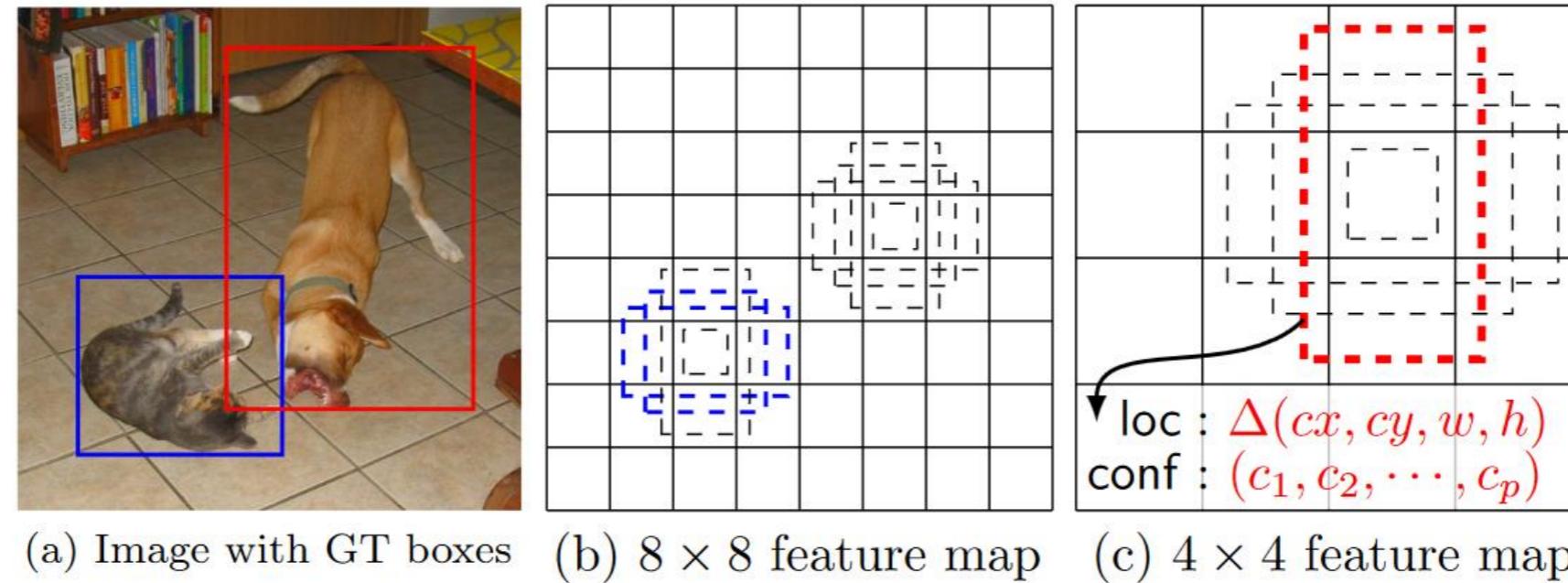


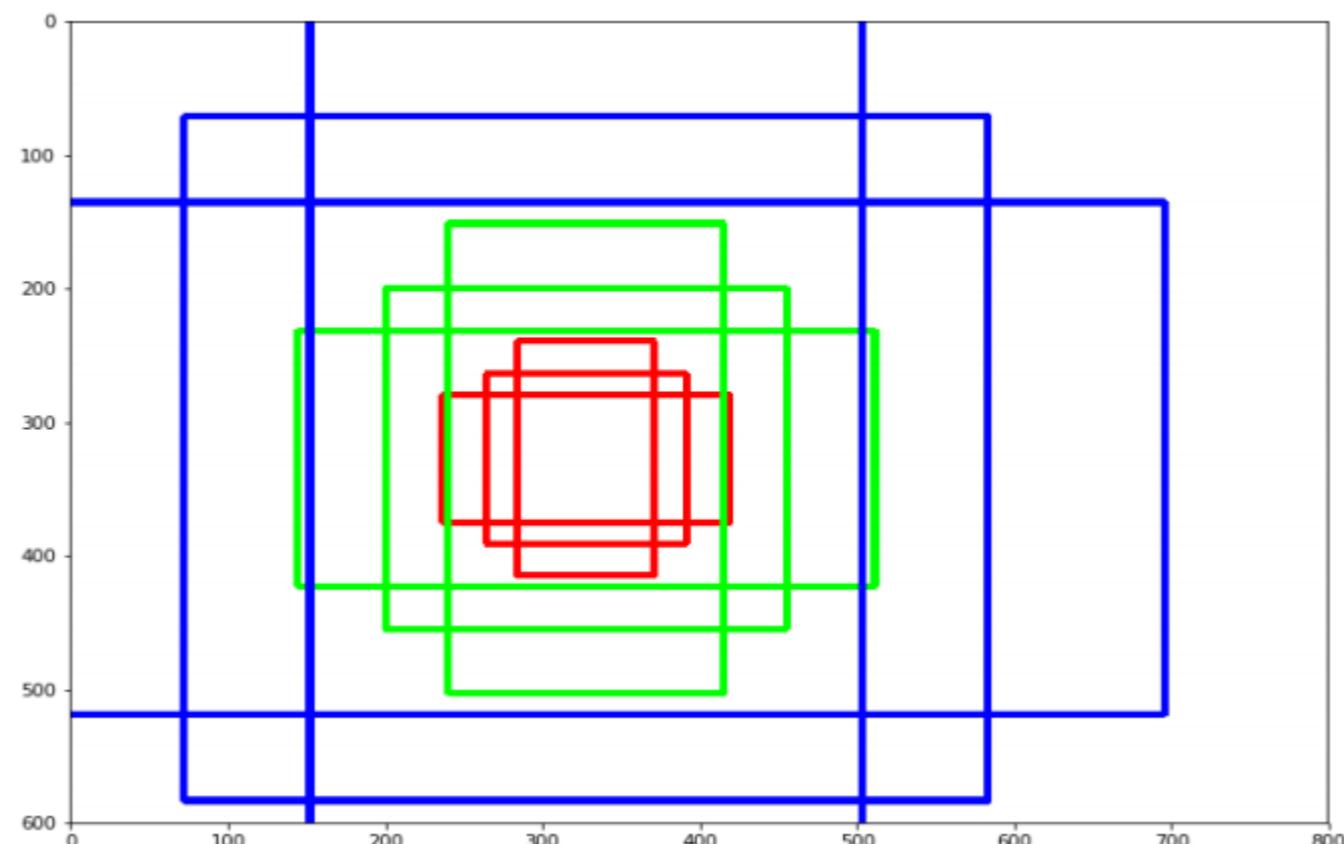
Fig. 1: **SSD framework.** (a) SSD only needs an input image and ground truth boxes for each object during training. In a convolutional fashion, we evaluate a small set (e.g. 4) of default boxes of different aspect ratios at each location in several feature maps with different scales (e.g. 8×8 and 4×4 in (b) and (c)). For each default box, we predict both the shape offsets and the confidences for all object categories ((c_1, c_2, \dots, c_p)). At training time, we first match these default boxes to the ground truth boxes. For example, we have matched two default boxes with the cat and one with the dog, which are treated as positives and the rest as negatives. The model loss is a weighted sum between localization loss (e.g. Smooth L1 [6]) and confidence loss (e.g. Softmax).

SSD: якоря (anchors)

base anchor (пусть 128×128)

anchor scales (пусть $\{1, 2, 3\}$)

anchors aspect ratios (пусть $1:1, 1:2, 2:1$)



Идея как в Region Proposal Network (RPN), но там для каждого региона
«есть объект / нет», а тут сразу класс определяем

Single Shot MultiBox Detector (SSD)

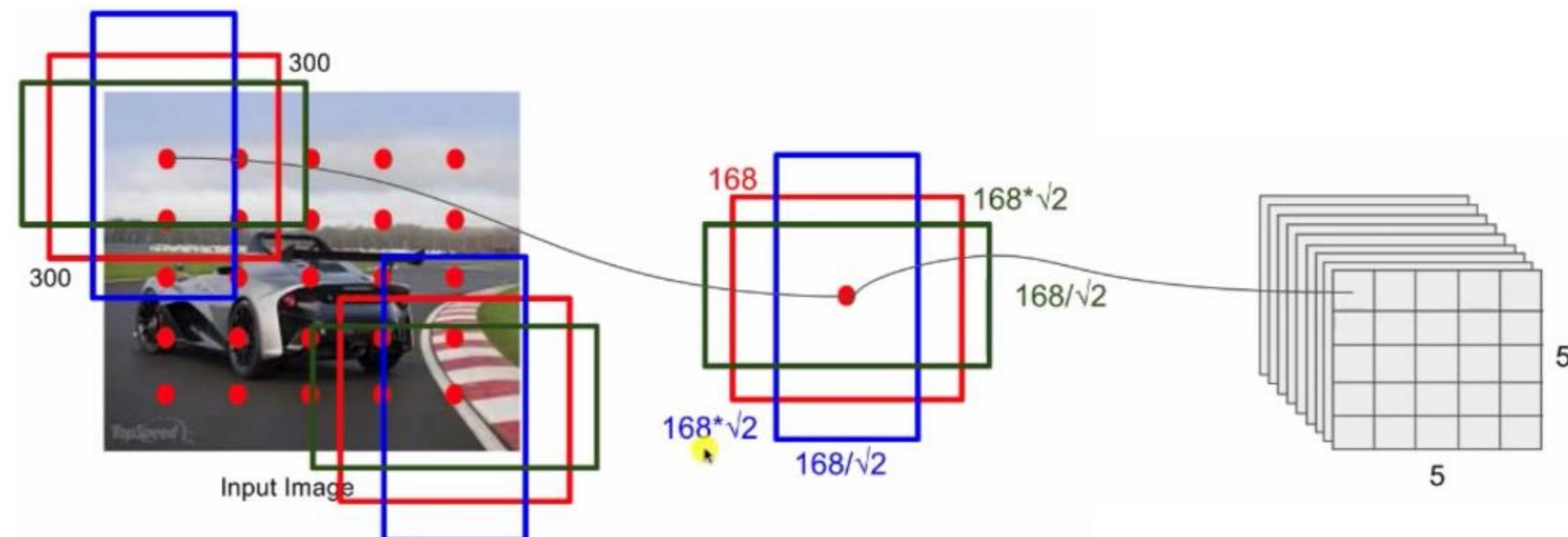
Вход 300×300 изображение

Нужны истинные рамки (на обучении)

Задаём несколько (на картинке 3) «default boxes» – для каждого

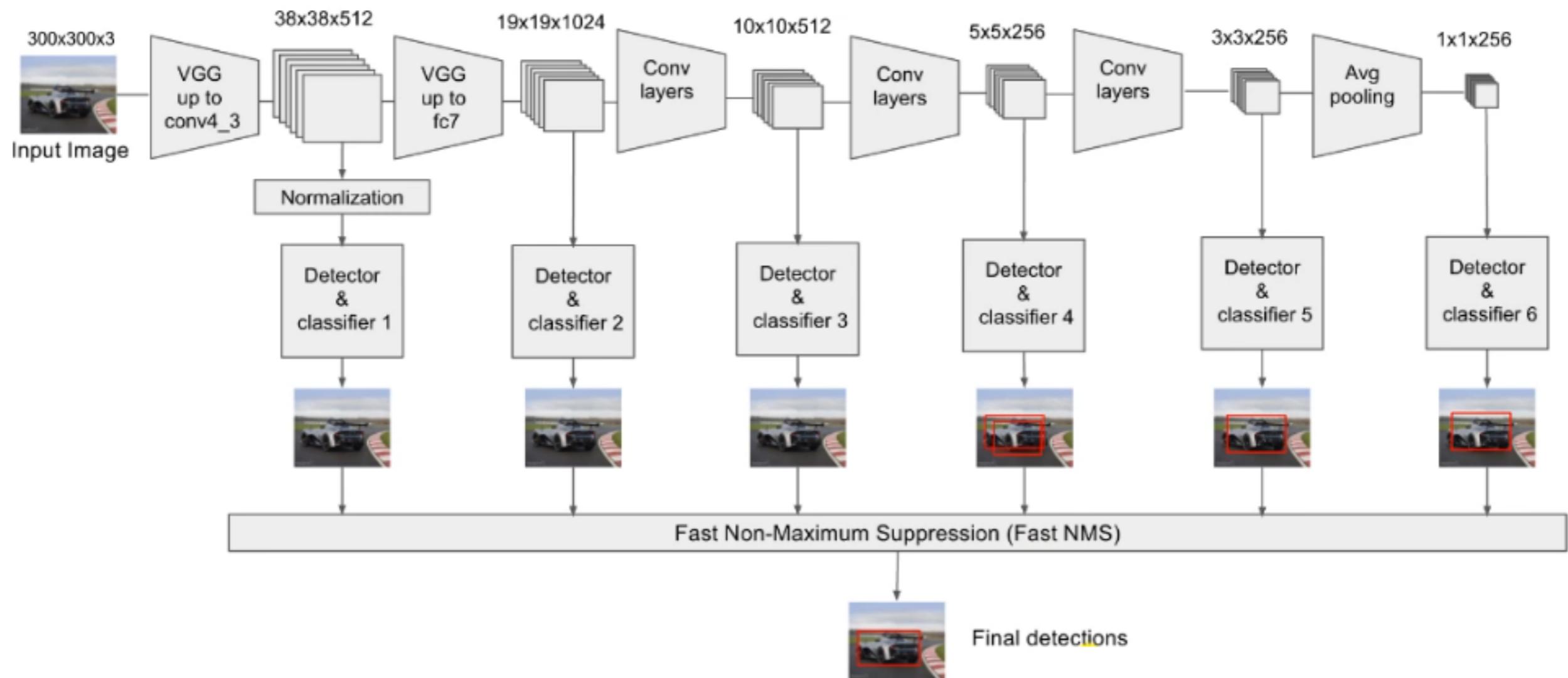
- коррекция положения

- оценки принадлежности к классам



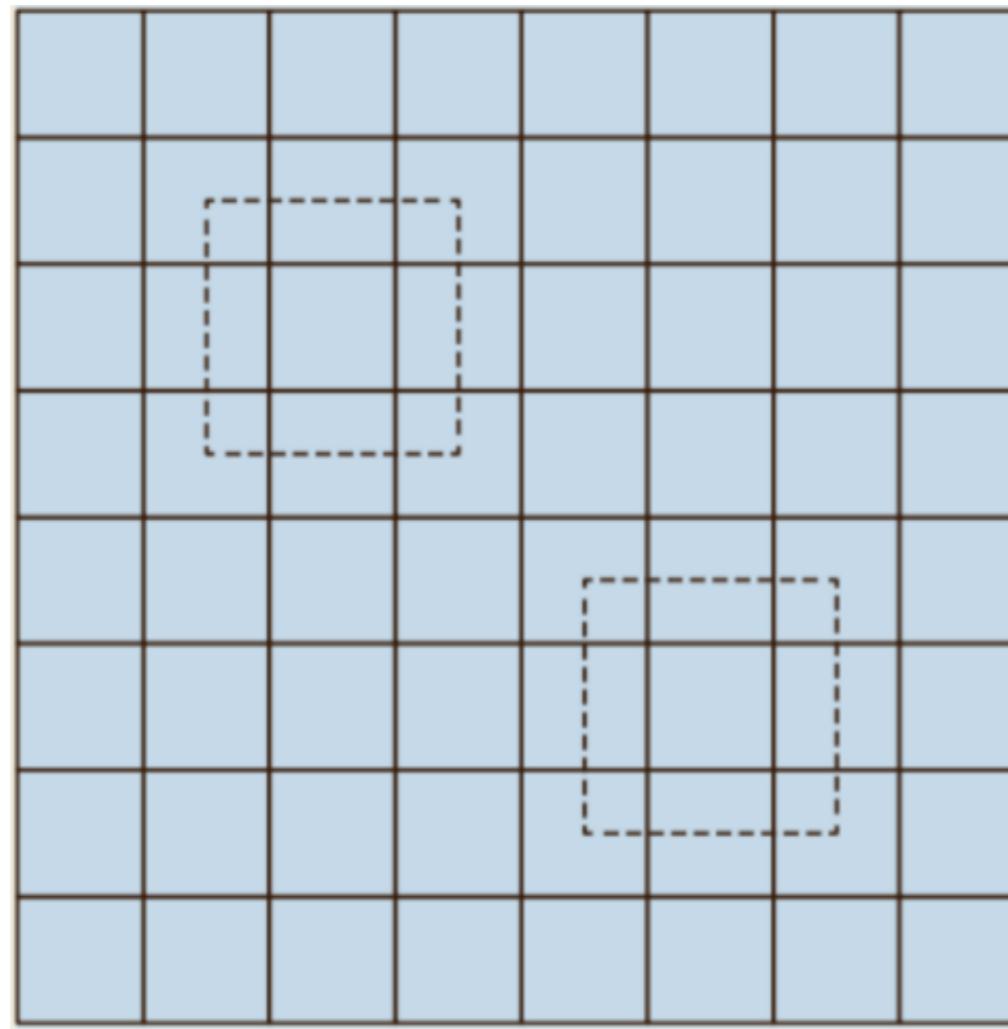
<https://deepsystems.ai>

Single Shot MultiBox Detector (SSD)

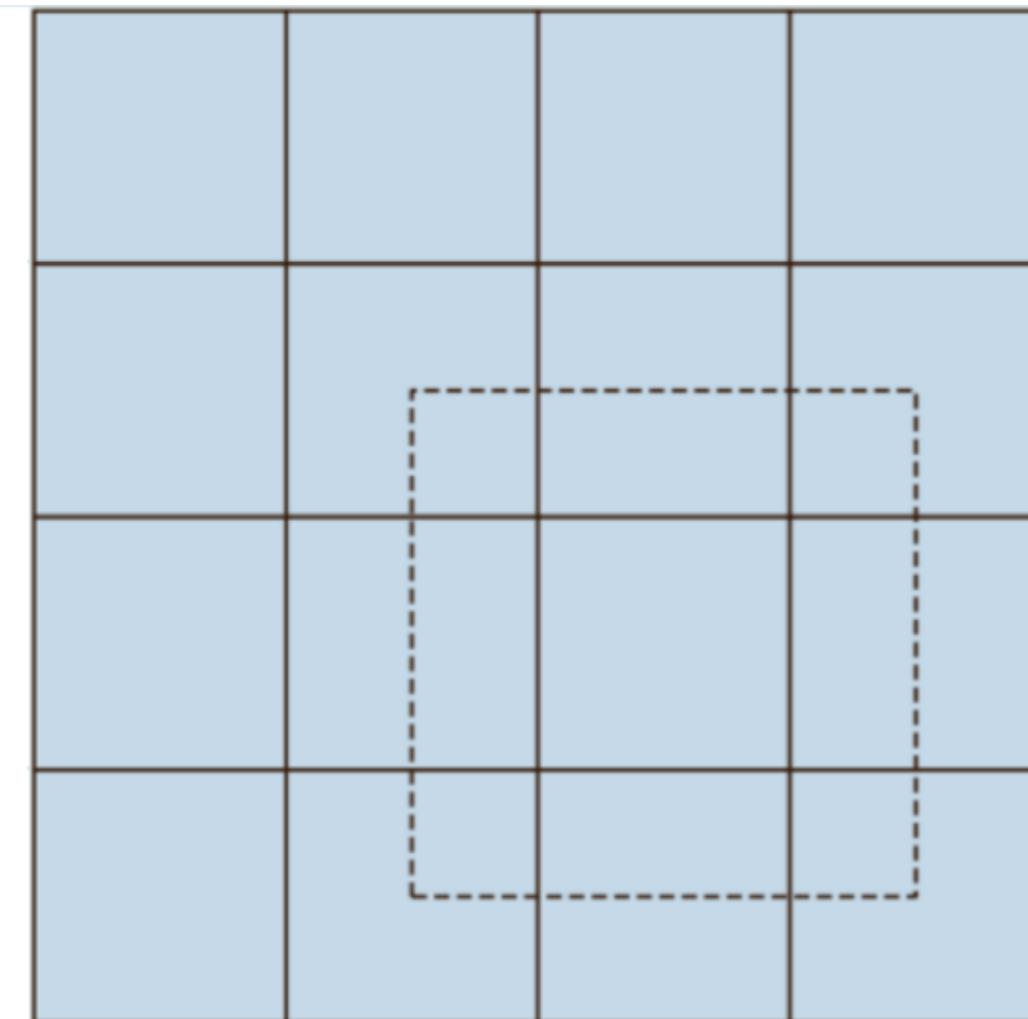


Много регионов «default boxes» на разных масштабах
<https://deepsystems.ai>

Single Shot MultiBox Detector (SSD)



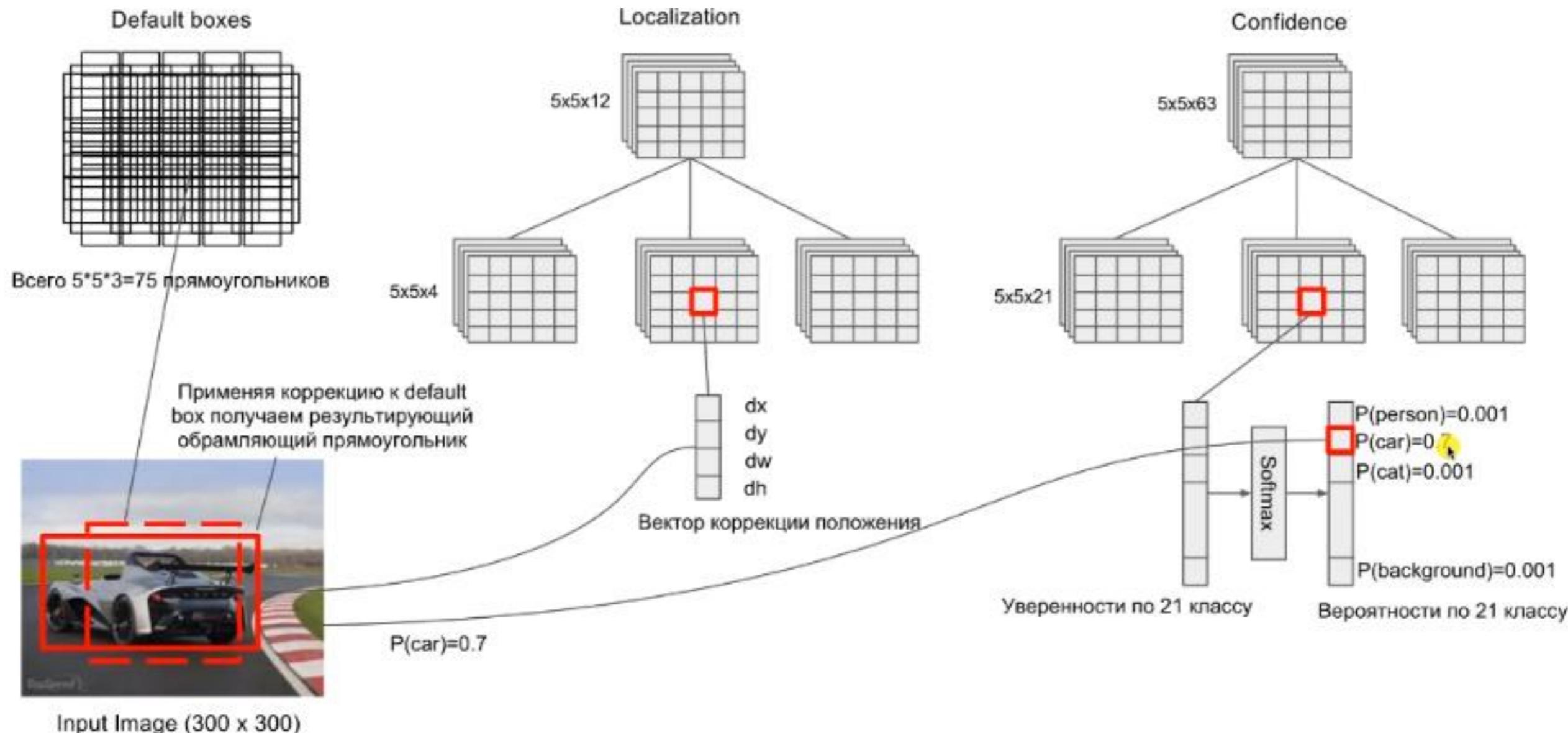
8×8 feature map



4×4 feature map

детектирование на разных масштабах
[Mohamed Elgendi]

Single Shot MultiBox Detector (SSD)



<https://deepsystems.ai>

Single Shot MultiBox Detector (SSD)

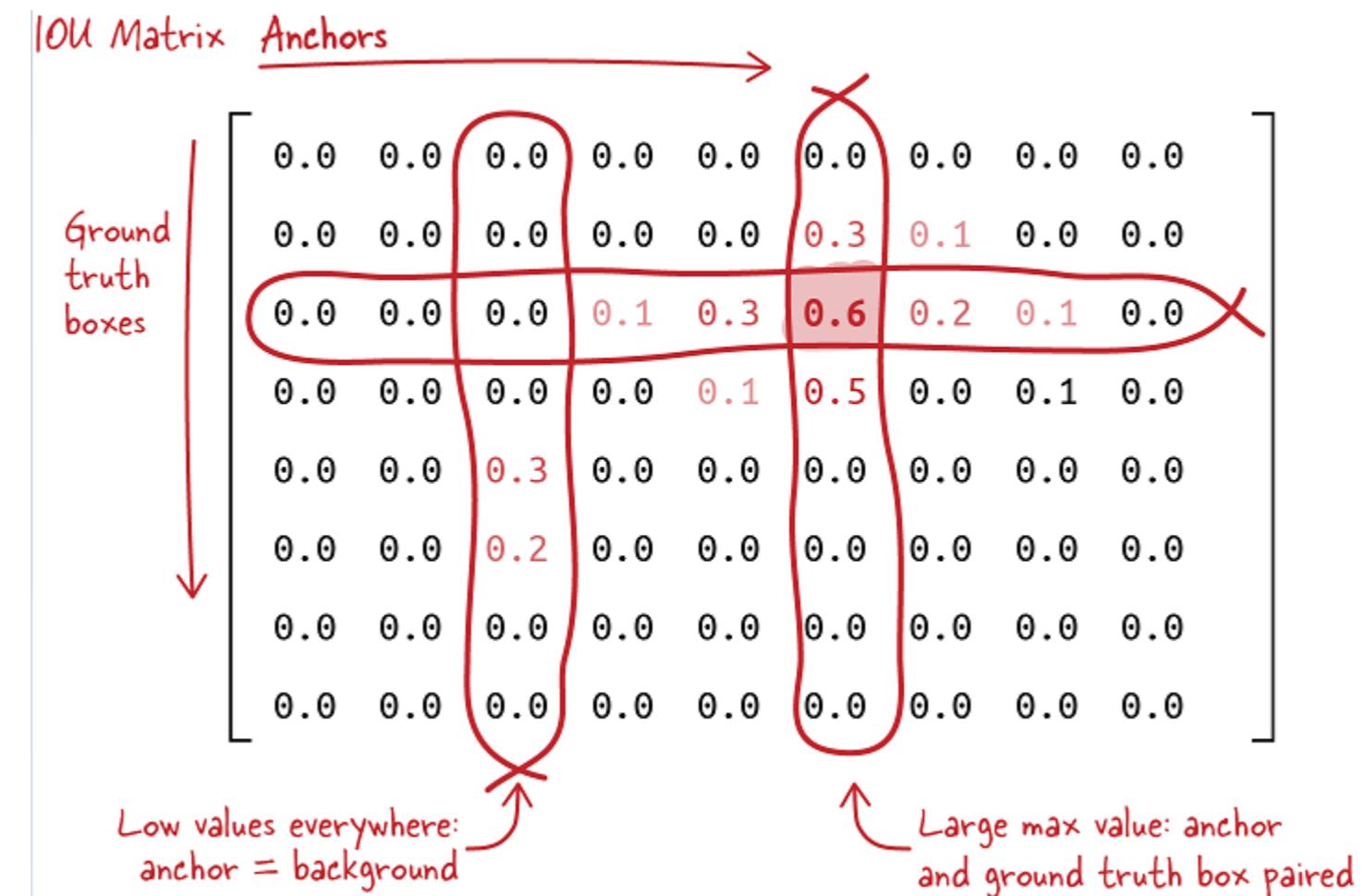
**размеры выхода: 5×5 (сетка) \times #якорей (default boxes) \times (# классов + 1)
 5×5 (сетка) \times #якорей (default boxes) \times 4 (регрессия для регионов)**

- очень быстро с хорошим качеством
- детектирование на разных масштабах
- большое число default boxes на разных масштабах

быстрая, почти как YOLO
но снимается ограничение на 2 объекта в ячейке

- в отличии от YOLO
- нет `r_obj` – а только вероятности
 - VGG в основе, а не Darknet
 - hard-negative mining (отношение $+/- = 1/3$)
 - другая ошибка (без подробностей)

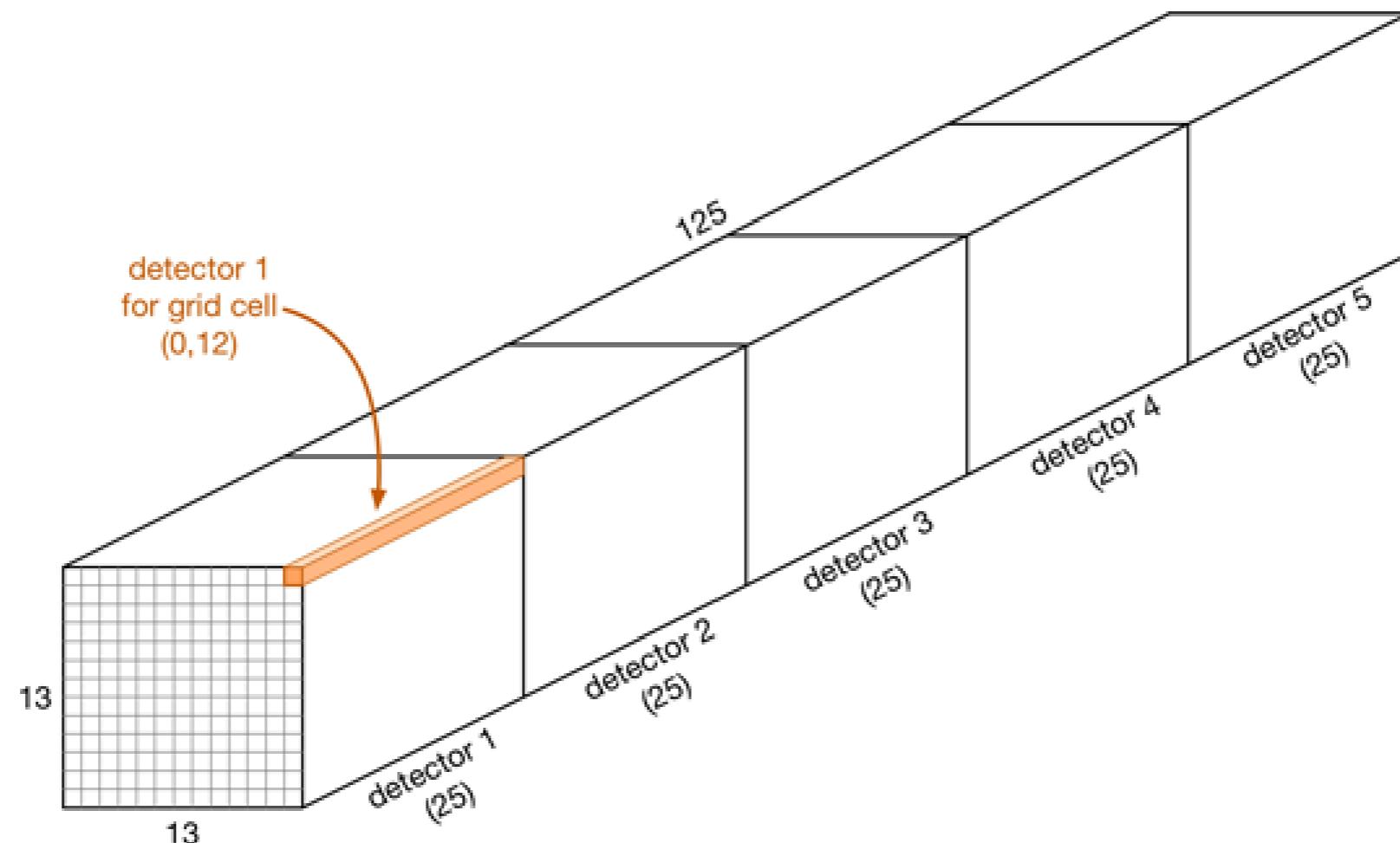
Как сопоставить якоря ↔ истинные регионы



надо найти наиболее подходящие
(и пытаться «модифицировать» форму и размеры к ним)
так точно в RetinaNet делается

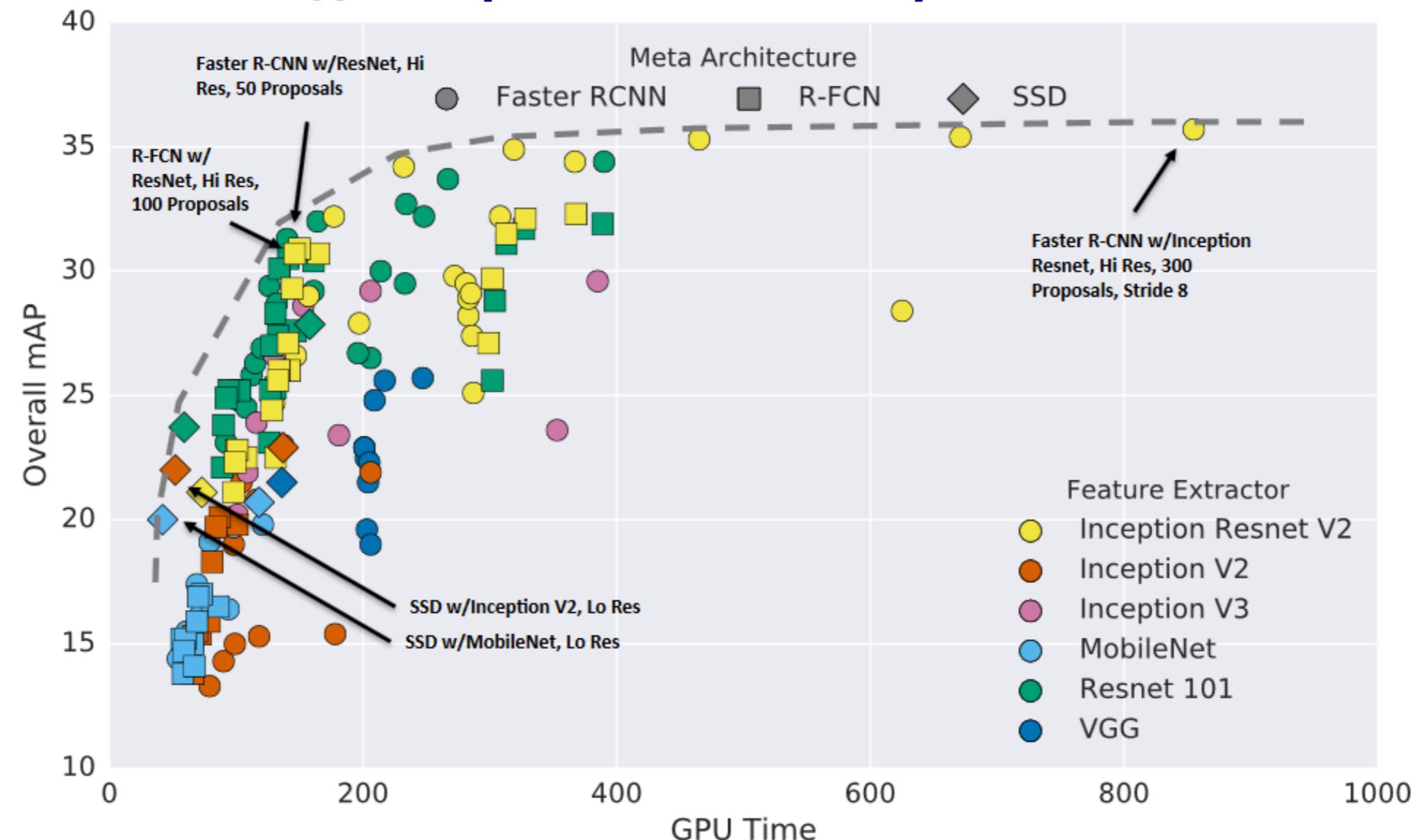
Признак One-stage-детектора

сетка ответов по классам и разбивка на разные детекторы



<https://machinethink.net/blog/object-detection/>

Детектирование объектов: сравнение



Speed/accuracy trade-offs for modern convolutional object detectors

[Jonathan Huang и др. 2017 <https://arxiv.org/pdf/1611.10012.pdf>]

- Детектирование объектов: Feature Pyramid Networks (FPN)

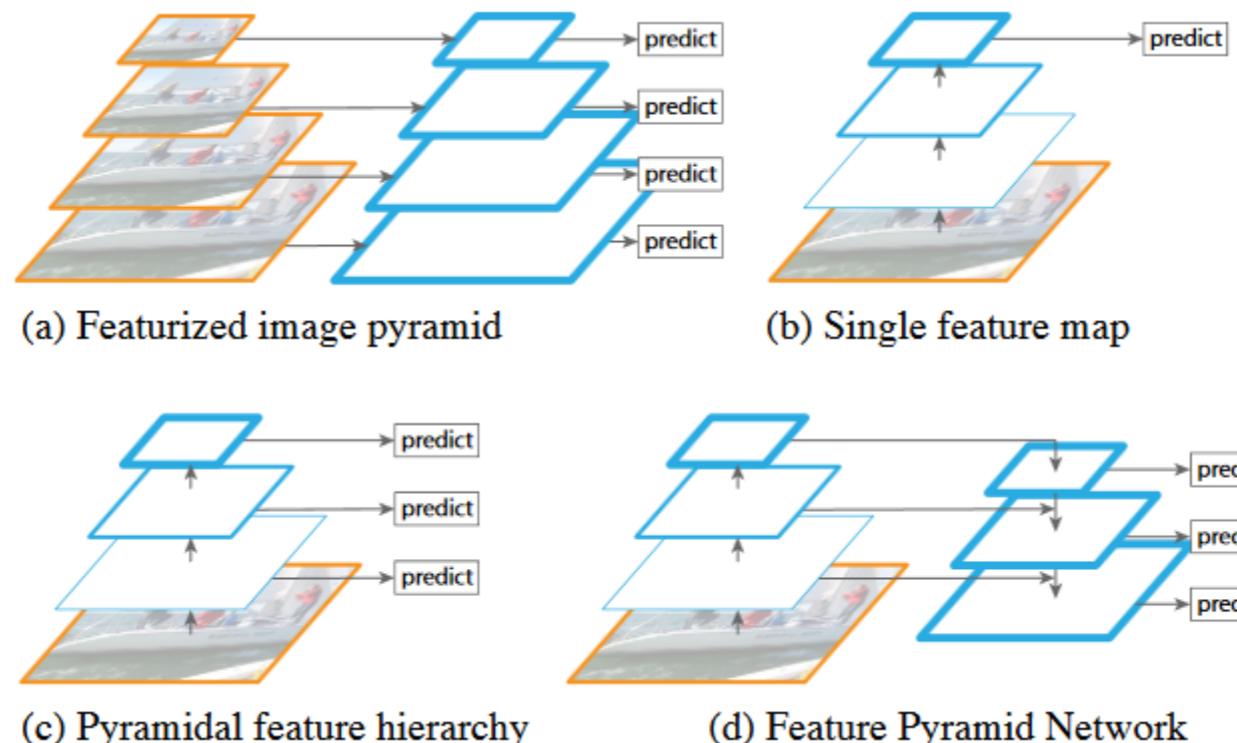
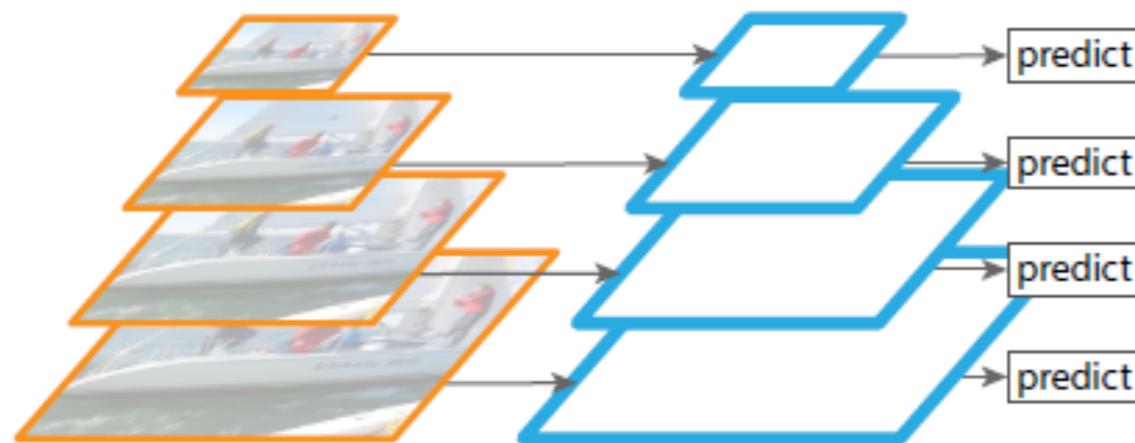


Figure 1. (a) Using an image pyramid to build a feature pyramid. Features are computed on each of the image scales independently, which is slow. (b) Recent detection systems have opted to use only single scale features for faster detection. (c) An alternative is to reuse the pyramidal feature hierarchy computed by a ConvNet as if it were a featurized image pyramid. (d) Our proposed Feature Pyramid Network (FPN) is fast like (b) and (c), but more accurate. In this figure, feature maps are indicated by blue outlines and thicker outlines denote semantically stronger features.

**для инвариантности к масштабу делают «пирамиды изображений»
обычно используются только во время тестирования (т.к. долгое end2end-обучение)
но свёрточные сети получают «пирамиды признаков»
это всё – способы получения карт признаков. Можно использовать с любыми
архитектурами НС! <https://arxiv.org/abs/1612.03144>**

Разные архитектуры

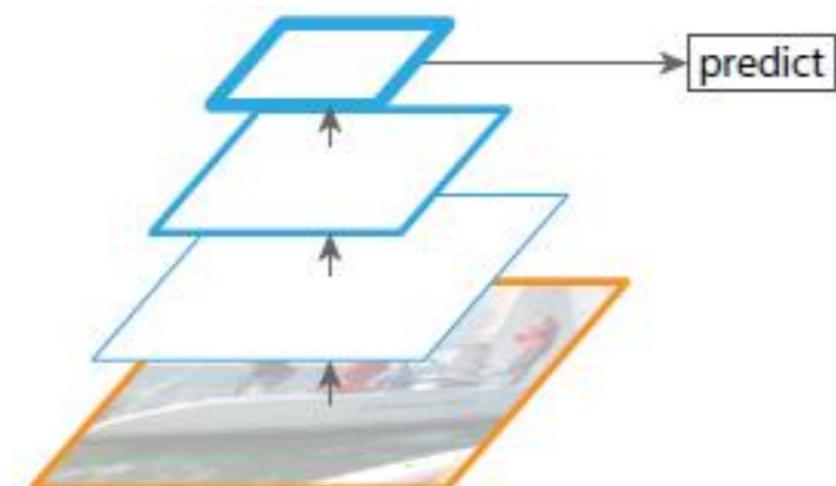


(a) Featurized image pyramid

Использовались ещё до эры DL

Точные

Медленные



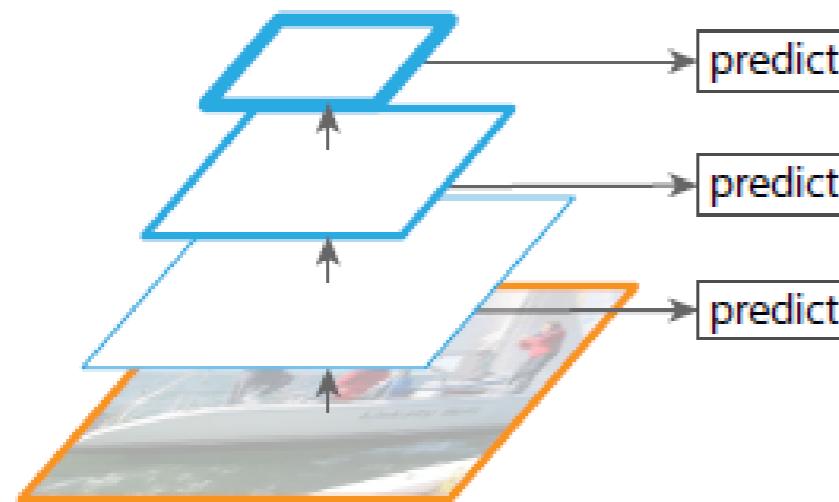
(b) Single feature map

Стандартное CNN-решение

Реализовано в YOLO

**Не учитывают низкоуровневую
информацию**

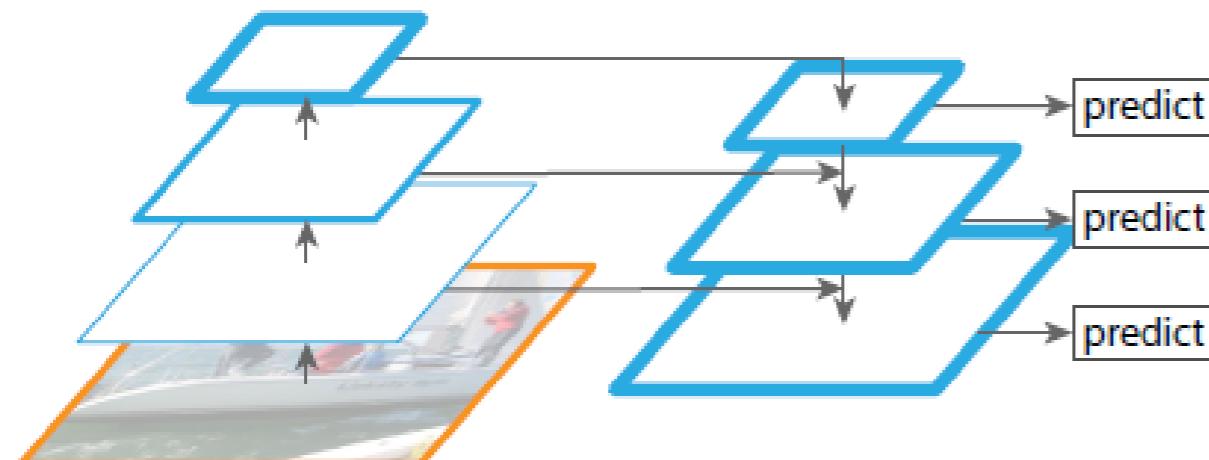
Разные архитектуры



(c) Pyramidal feature hierarchy

Реализовано в SSD

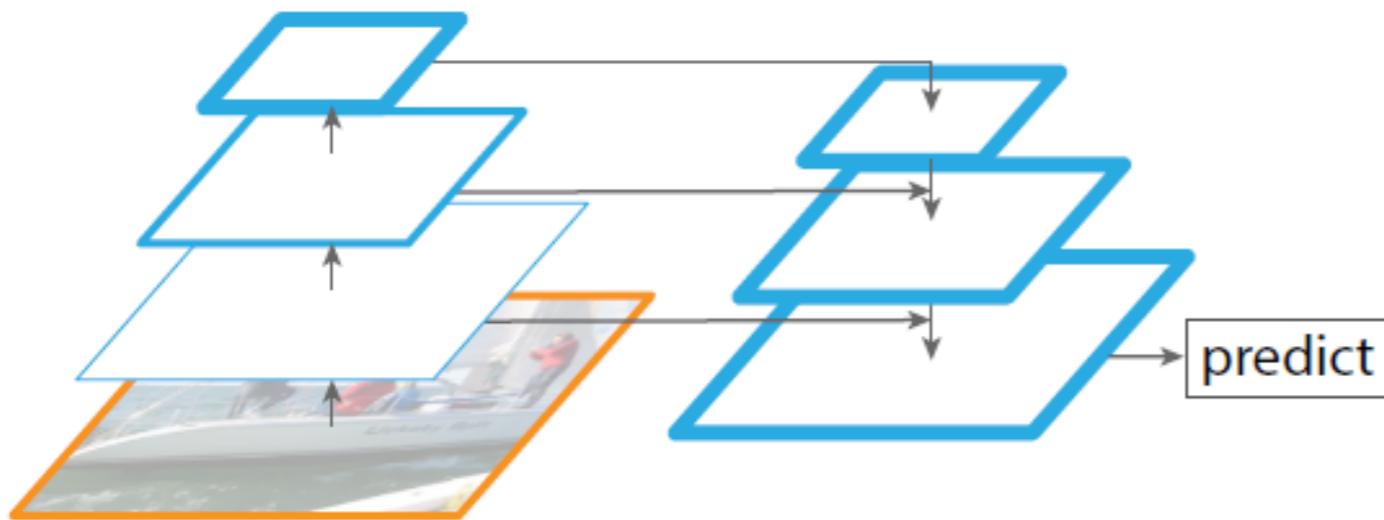
Не учитывают контекст на низком уровне



(d) Feature Pyramid Network

точные и быстрые

Разные архитектуры



(e) Similar Structure with (d)

Иногда применяется схожая архитектура

В сегментации это U-Net

Anchor-Free (Proposal Free) Object Detection

Следующее поколение детекторов...
«Безъякорное детектирование»

новая прорывная идея!

anchors – очень затратны
есть гиперпараметры, от которых сильно зависит качество
все регионы-кандидаты при обучении надо размечать
(опять же – долго + гиперпараметры)

Anchor-Free Object Detection

CornerNet – предсказываем «углы»

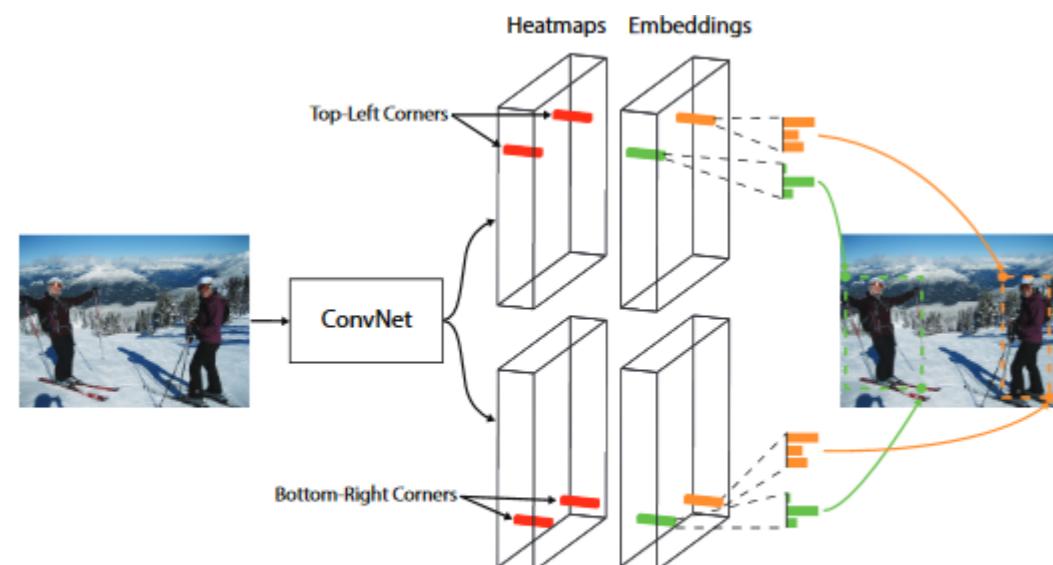


Fig. 1 We detect an object as a pair of bounding box corners grouped together. A convolutional network outputs a heatmap for all top-left corners, a heatmap for all bottom-right corners, and an embedding vector for each detected corner. The network is trained to predict similar embeddings for corners that belong to the same object.

<https://arxiv.org/pdf/1808.01244.pdf>

**CornerNet-Lite – ускоренная версия,
Начинает с уменьшенной картинки, карта
внимания (attention map)**

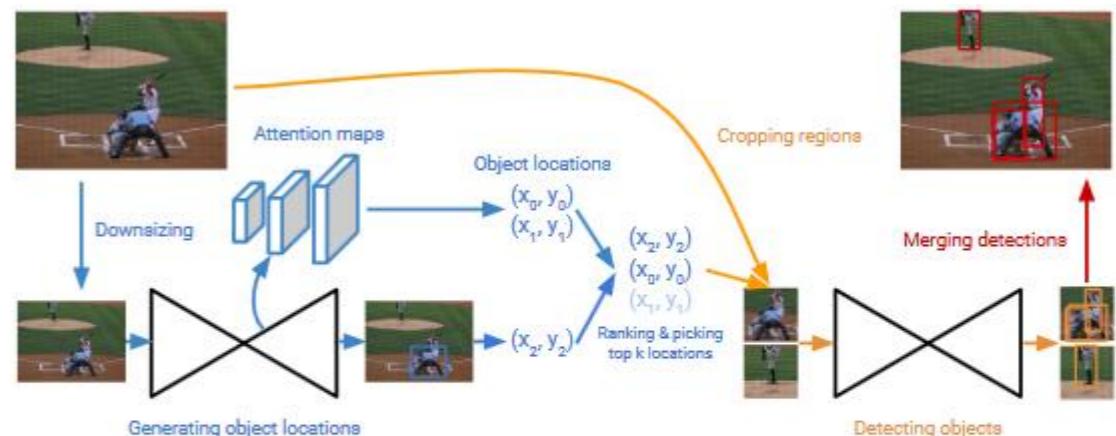
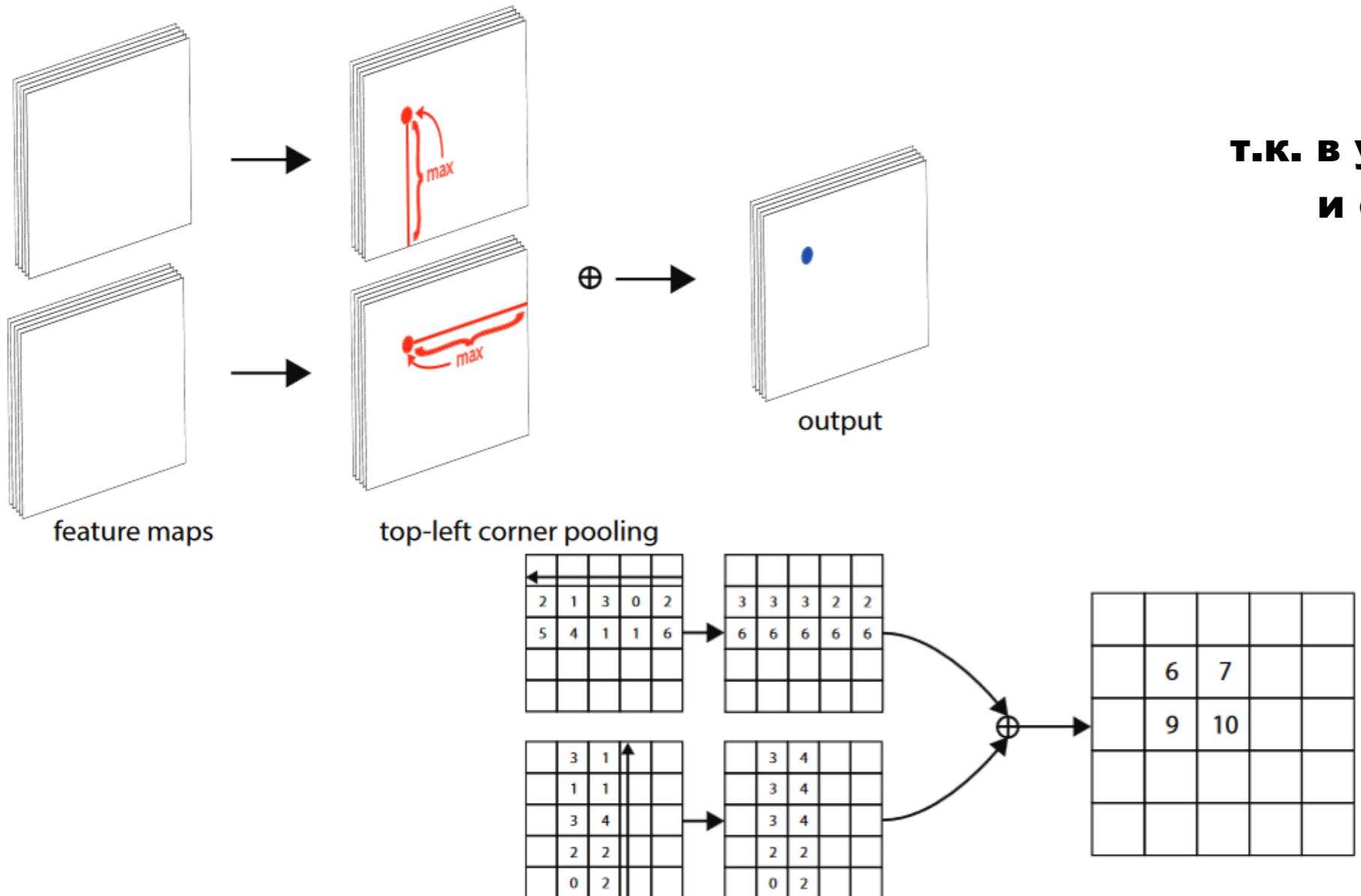


Figure 2: Overview of CornerNet-Saccade. We predict a set of possible object locations from the attention maps and bounding boxes generated on a downsized full image. We zoom into each location and crop a small region around that location. Then we detect objects in top k regions and merge the detections by NMS.

<https://arxiv.org/pdf/1904.08900.pdf>

немного истории этого более свежего направления

CornerNet: особый вид пулинга



т.к. в углу рамки обычно и объекта-то нет!

Fig. 6 The top-left corner pooling layer can be implemented very efficiently. We scan from right to left for the horizontal max-pooling and from bottom to top for the vertical max-pooling. We then add two max-pooled feature maps.

CornerNet: особый вид пулинга



Fig. 8 Qualitative examples showing corner pooling helps better localize the corners.

CornerNet: выход сети

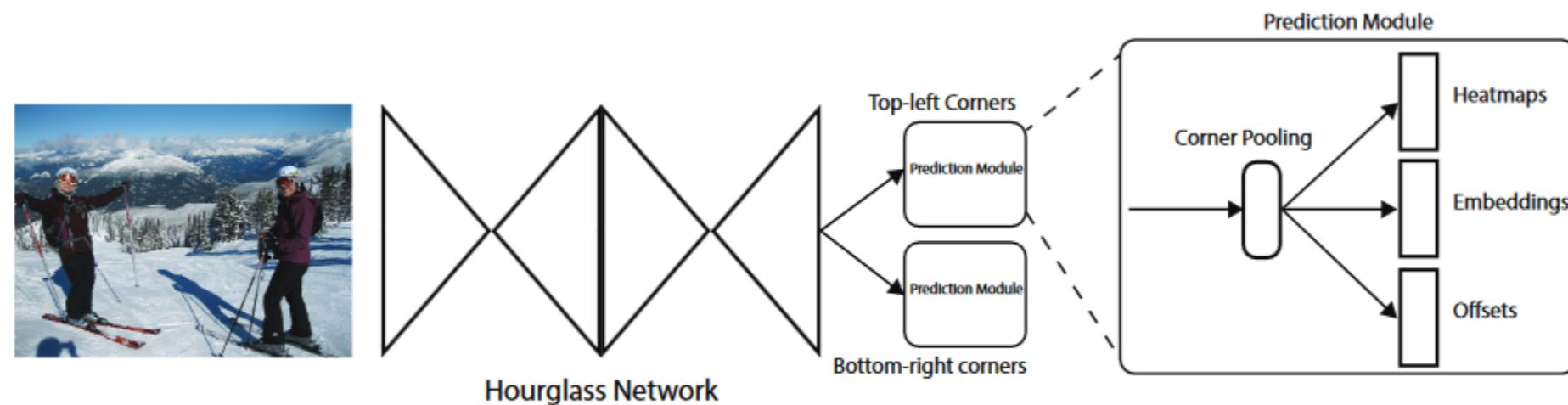


Fig. 4 Overview of CornerNet. The backbone network is followed by two prediction modules, one for the top-left corners and the other for the bottom-right corners. Using the predictions from both modules, we locate and group the corners.

**для «top-left» и «bottom-right» предсказываем:
heatmaps (вероятности быть углом для каждого из классов) тут нет класса «фон»
embedding: расстояния между представлениями углов одной рамки минимально
offsets – для редактирования координат углов**

**в каждом модуле (tl и br) свой corner pooling module
тут нет признаков с разных масштабов**

CornerNet: обучение



Fig. 5 “Ground-truth” heatmaps for training. Boxes (*green dotted rectangles*) whose corners are within the radii of the positive locations (*orange circles*) still have large overlaps with the ground-truth annotations (*red solid rectangles*).

Позитивные примеры – истинные углы, но также считаем позитивным всё, что недалеко от углов

focal loss

FCOS: Fully Convolutional One-Stage Object Detection

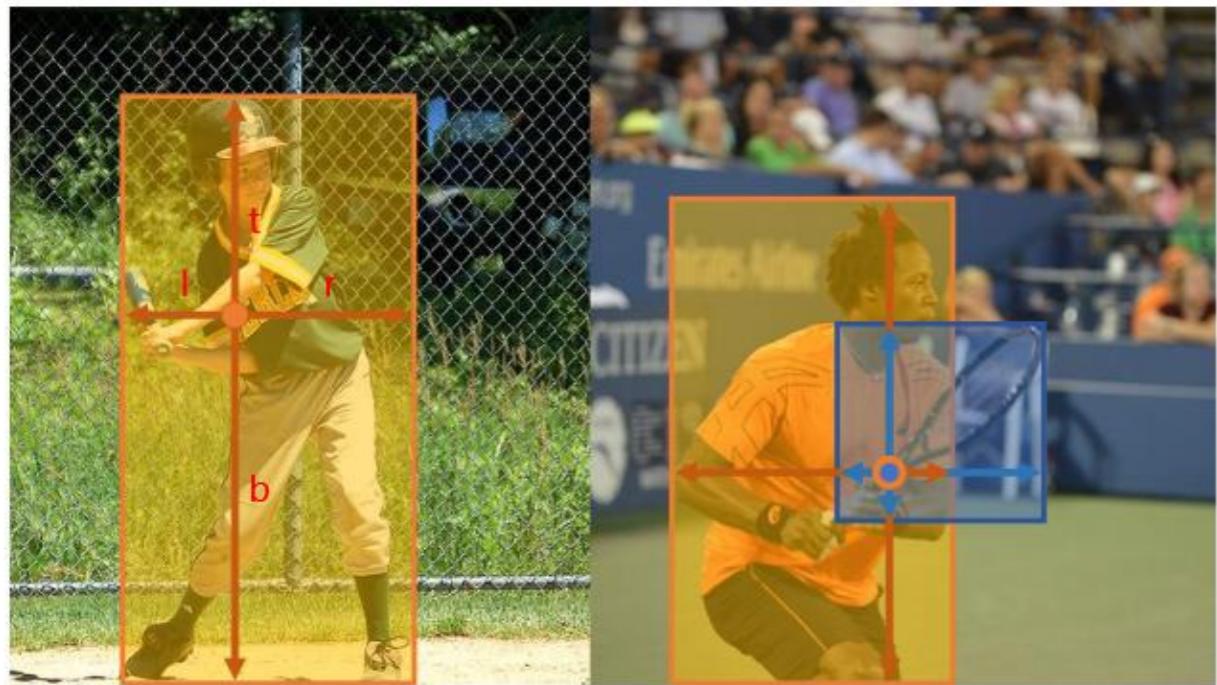


Figure 1 – As shown in the left image, FCOS works by predicting a 4D vector (l, t, r, b) encoding the location of a bounding box at each foreground pixel (supervised by ground-truth bounding box information during training). The right plot shows that when a location residing in multiple bounding boxes, it can be ambiguous in terms of which bounding box this location should regress.

Zhi Tian, Chunhua Shen, Hao Chen, Tong He «FCOS: Fully Convolutional One-Stage Object Detection» //
<https://arxiv.org/pdf/1904.01355.pdf>

есть целевые регионы – им сразу и обучаемся

«One-Stage» – нет как раньше отдельных подзадач

**для любой точки на изображении:
если она попадает в целевой регион –
«позитивная»**

**если в несколько – «двузначная»
(относим к минимальному)**

основа: ResNet-50

FCOS: Fully Convolutional One-Stage Object Detection

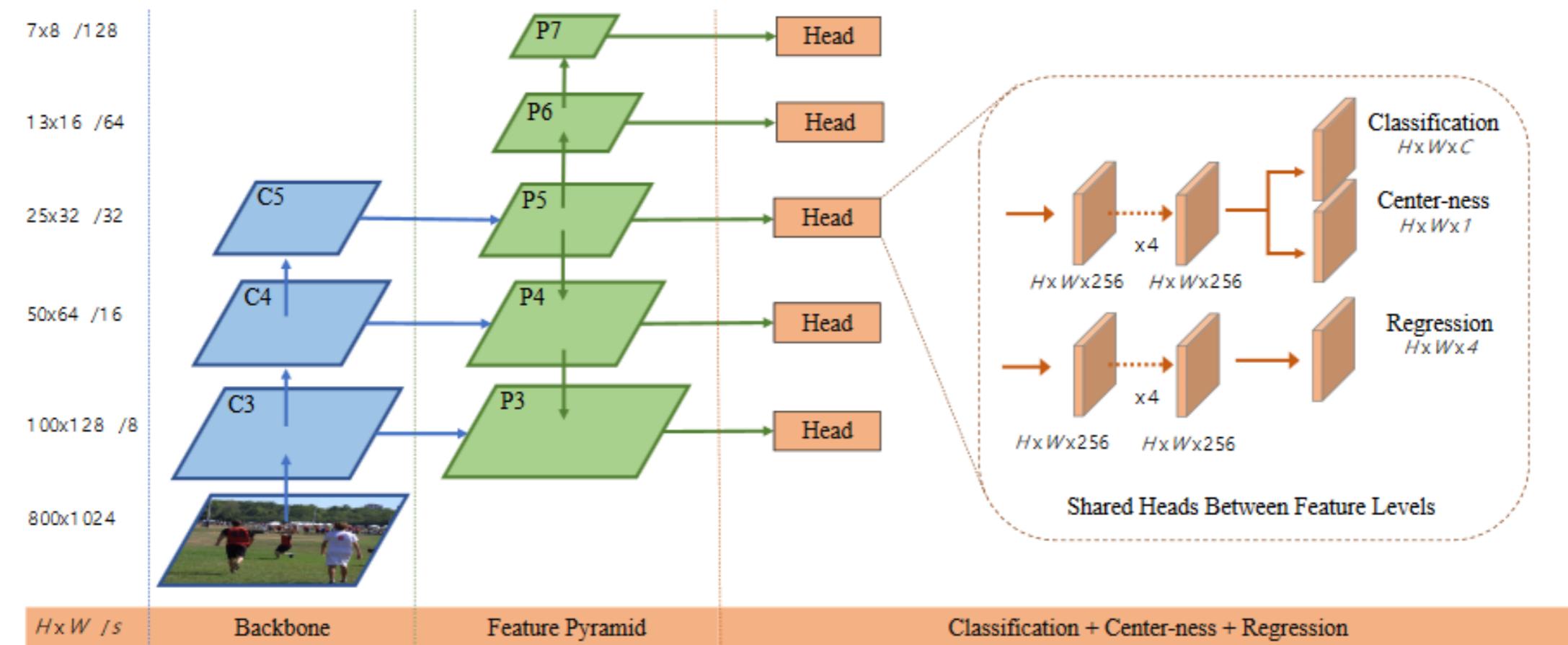


Figure 2 – The network architecture of FCOS, where C3, C4, and C5 denote the feature maps of the backbone network and P3 to P7 are the feature levels used for the final prediction. $H \times W$ is the height and width of feature maps. ‘/s’ ($s = 8, 16, \dots, 128$) is the down-sampling ratio of the feature maps at the level to the input image. As an example, all the numbers are computed with an 800×1024 input.

Feature Pyramid Network(FPN) + выходы для центра, регрессии и класса

FCOS: Center-ness

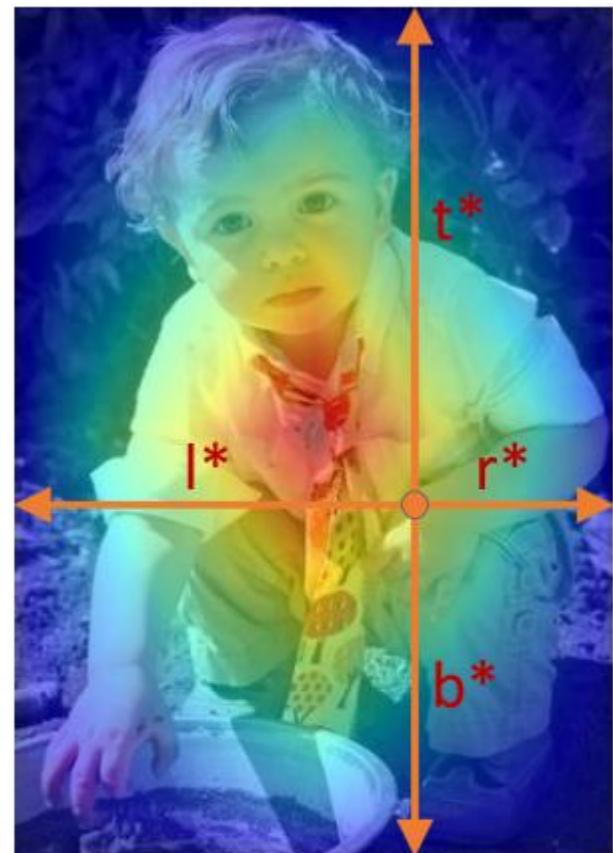


Figure 3 – Center-ness. Red, blue, and other colors denote 1, 0 and the values between them, respectively. Center-ness is computed by Eq. (3) and decays from 1 to 0 as the location deviates from the center of the object. When testing, the center-ness predicted by the network is multiplied with the classification score thus can down-weight the low-quality bounding boxes predicted by a location far from the center of an object.

**В каждой точке предсказывается (l , t , r , b),
а не (x , y , w , h)
+ класс (метка из C) + центральность (число)**

$$\text{centerness}^* = \sqrt{\frac{\min(l^*, r^*)}{\max(l^*, r^*)} \times \frac{\min(t^*, b^*)}{\max(t^*, b^*)}}.$$

**center-ness на [0, 1] обучается с помощью
binary cross entropy (BCE) loss**

**при работе к точкам с большим значением
можно применить NMS**

FCOS: Fully Convolutional One-Stage Object Detection

Method	Backbone	AP	AP ₅₀	AP ₇₅	AP _S	AP _M	AP _L
Two-stage methods:							
Faster R-CNN w/ FPN [14]	ResNet-101-FPN	36.2	59.1	39.0	18.2	39.0	48.2
Faster R-CNN by G-RMI [11]	Inception-ResNet-v2 [27]	34.7	55.5	36.7	13.5	38.1	52.0
Faster R-CNN w/ TDM [25]	Inception-ResNet-v2-TDM	36.8	57.7	39.2	16.2	39.8	52.1
One-stage methods:							
YOLOv2 [22]	DarkNet-19 [22]	21.6	44.0	19.2	5.0	22.4	35.5
SSD513 [18]	ResNet-101-SSD	31.2	50.4	33.3	10.2	34.5	49.8
DSSD513 [5]	ResNet-101-DSSD	33.2	53.3	35.2	13.0	35.4	51.1
RetinaNet [15]	ResNet-101-FPN	39.1	59.1	42.3	21.8	42.7	50.2
CornerNet [13]	Hourglass-104	40.5	56.5	43.1	19.4	42.7	53.9
FSAF [34]	ResNeXt-64x4d-101-FPN	42.9	63.8	46.3	26.6	46.2	52.7
FCOS	ResNet-101-FPN	41.5	60.7	45.0	24.4	44.8	51.6
FCOS	HRNet-W32-51 [26]	42.0	60.4	45.3	25.4	45.0	51.0
FCOS	ResNeXt-32x8d-101-FPN	42.7	62.2	46.1	26.0	45.6	52.6
FCOS	ResNeXt-64x4d-101-FPN	43.2	62.8	46.6	26.5	46.2	53.3
FCOS w/ improvements	ResNeXt-64x4d-101-FPN	44.7	64.1	48.4	27.6	47.5	55.6

Table 5 – FCOS vs. other state-of-the-art two-stage or one-stage detectors (*single-model and single-scale results*). FCOS outperforms the anchor-based counterpart RetinaNet by 2.4% in AP with the same backbone. FCOS also outperforms the recent anchor-free one-stage detector CornerNet with much less design complexity. Refer to Table 3 for details of “improvements”.



Итог

история перехода к полностью НС-решениям

**локализация объектов не обязательно должна быть классоориентированной
(class-specific)!**

в одном регионе можно одновременно детектировать несколько объектов!

**есть способ генерирования «якорей» – большого набора потенциальных регионов
есть способ обойтись без генерации регионов!**

«Пирамидные сети» – способ формирования признакового пространства

**Хорошая идея: каждая точка потенциальный представитель рамки
(оцениваем её параметры)**

Ссылки

Подборка материалов по детекции объектов

<https://github.com/XiongweiWu/Awesome-Object-Detection>

ещё одна...

<https://github.com/XinZhangNLPR/awesome-anchor-free-object-detection>

неплохой обзор от практика

<https://machinethink.net/blog/object-detection/>

самый последний (на 21.10.2021) обзор по детектированию объектов

«A Survey of Modern Deep Learning based Object Detection Models»

<https://arxiv.org/pdf/2104.11892v2.pdf>