

# Компьютерное зрение

Лекция 6.

Сегментация и детекция с помощью нейросетей

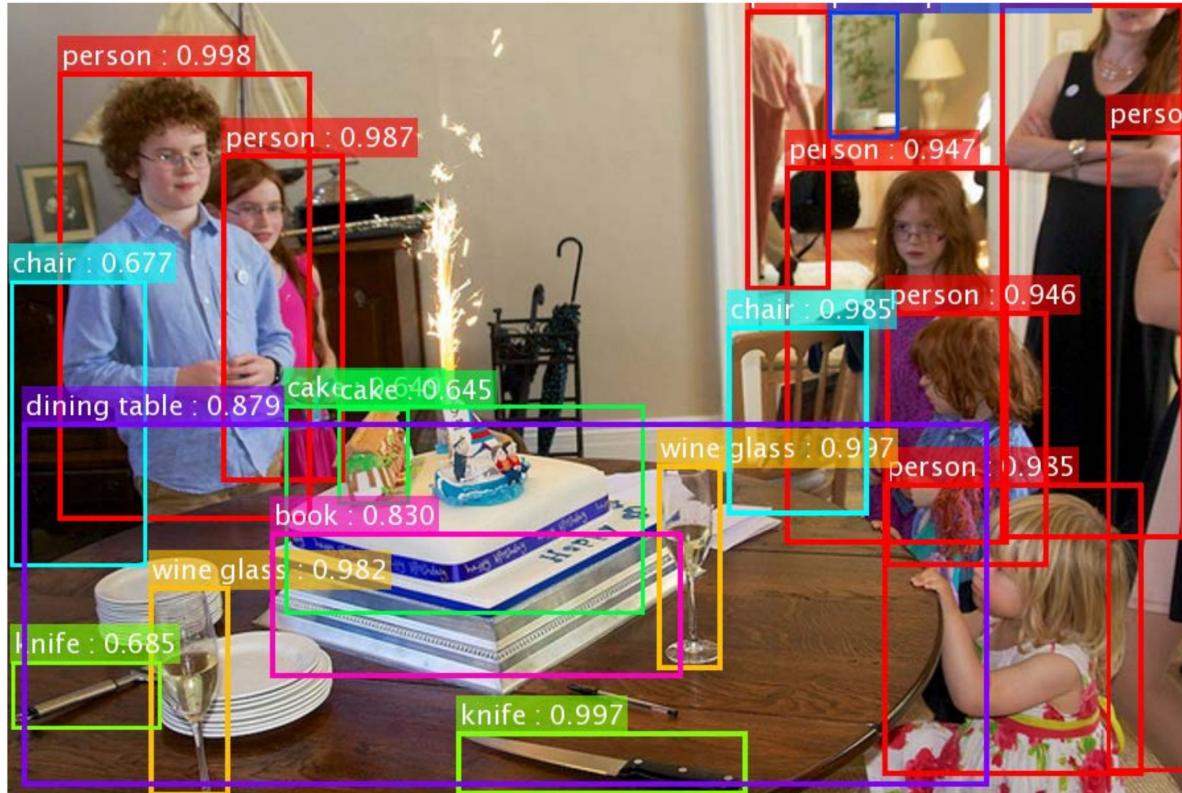
25.01.2020

Руслан Рахимов

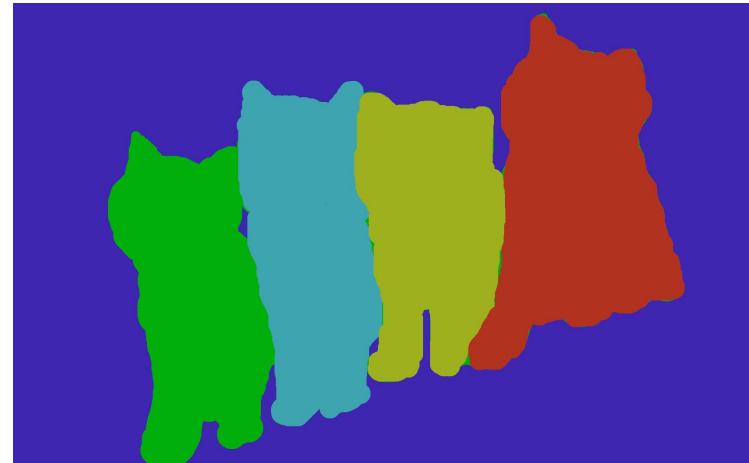
# Semantic Segmentation



# Object Detection



# Instance Segmentation



# Отвечаю на вопрос “где”?

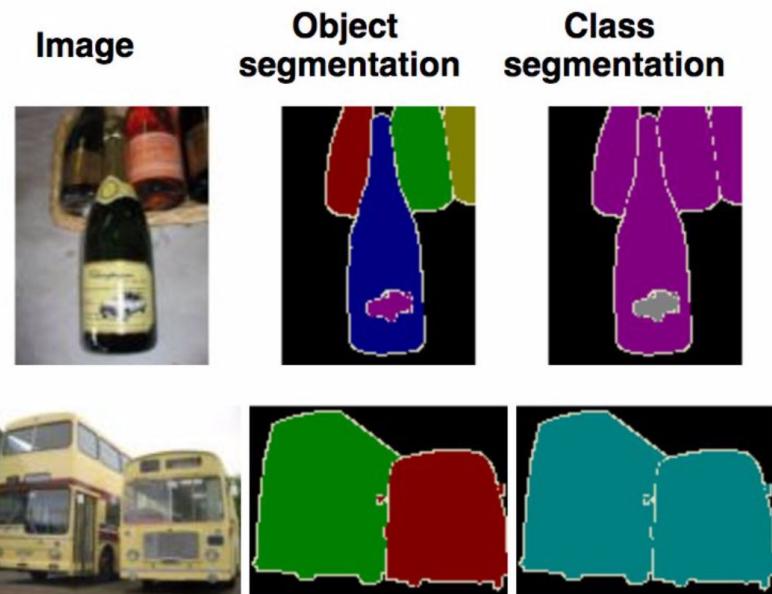
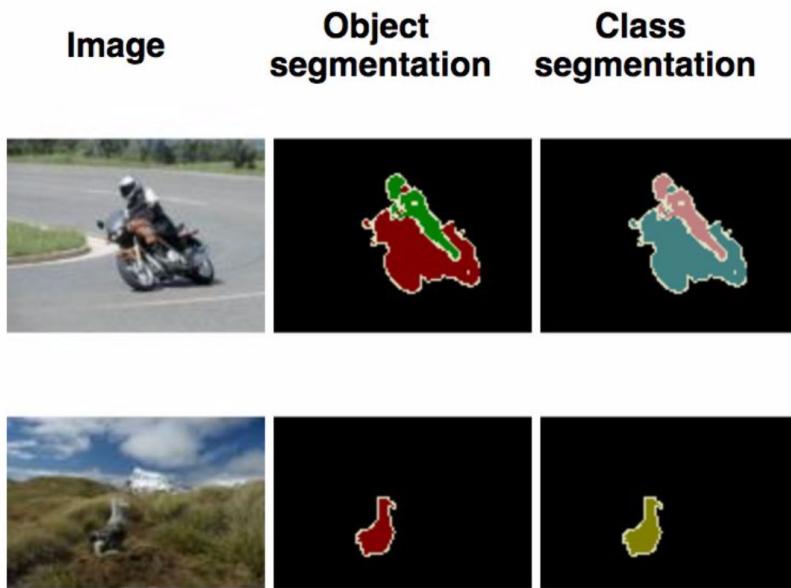
- Semantic segmentation:
  - Relatively fast/easy
  - Allows “complete” explanation
  - Merges instances
- Object detection
  - Relatively fast/easy
  - Distinguishes instances
  - Inaccurate for some classes
  - Incomplete
- Instance segmentation
  - Complete
  - Distinguish instances
  - Accurate
  - Slow/hard

# Semantic Segmentation

# Постановка задачи

- Для каждого пикселя на изображении предсказываем семантический класс (собака, кошка, человек, самолет и т.д.)

# Pascal VOC' 12 Dataset

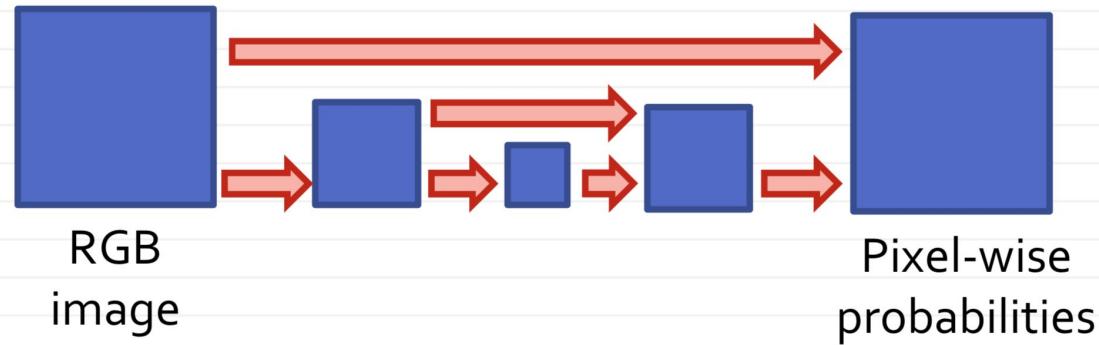


# Подходы

- за основу решений с помощью нейронных сетей лежит архитектура классифицирующей сверточной сети
- новая архитектура состоит из двух частей
- **encoder**: последовательность сверток генерирующая сжатое представление изображения
- **decoder**: классификатор пикселей на основе сжатого представления

# Подходы

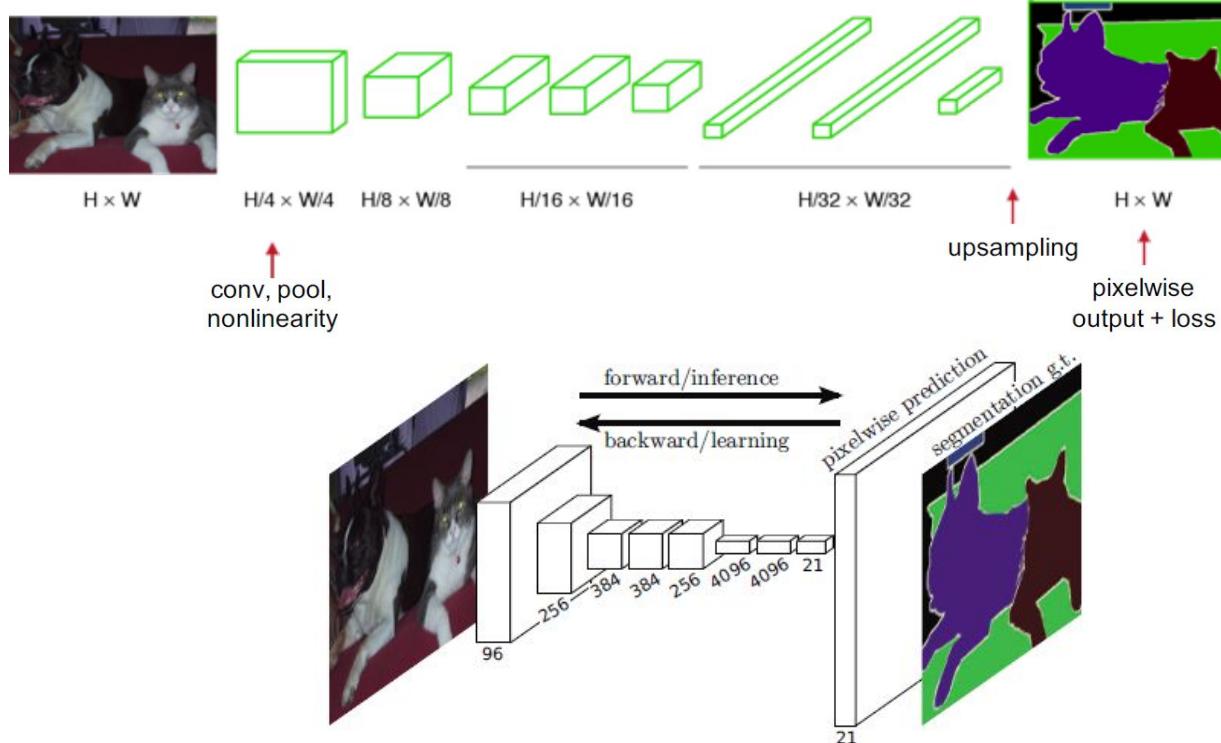
- Архитектуры обычно выглядят следующим образом



- Нижний поток обеспечивает большой receptive field
- Skip connections обеспечивают четкие детали

# FCN (Fully Convolutional Network)

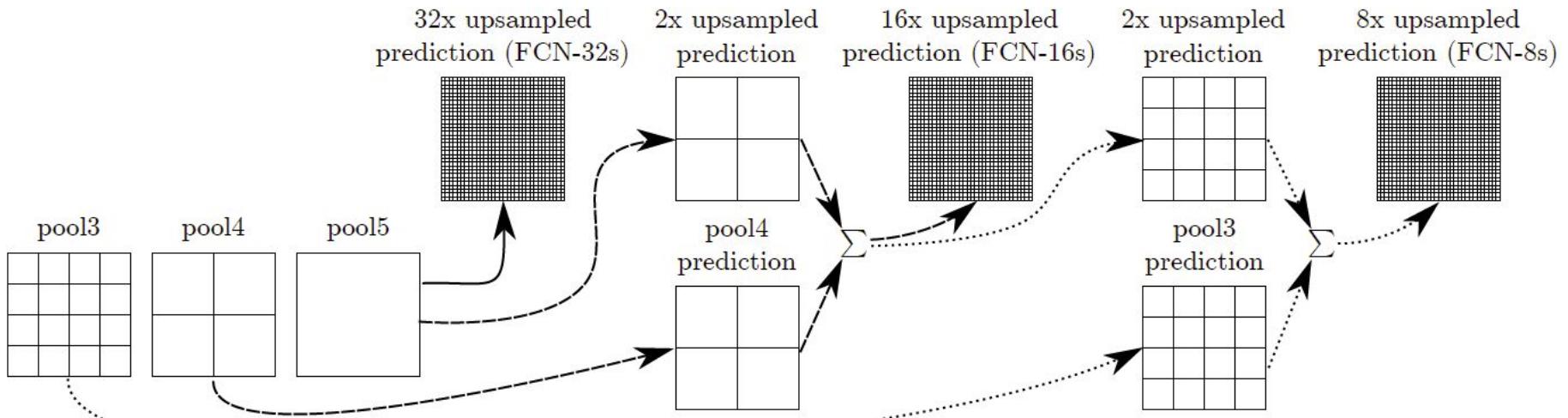
convolution



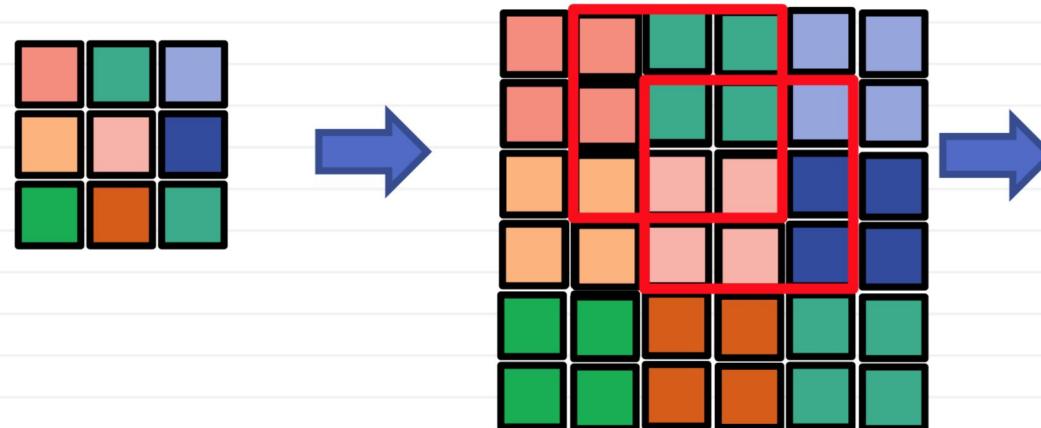
# FCN (Fully Convolutional Network)

- Основная идея, что все операции в сети представляют собой свертки
- Тензоры на выходе из последних слоев CNN меньше гораздо по ширине и высоте, чем на входе
  - > **сложнее восстанавливать четкие детали**
- **Идея как улучшить:**  
комбинировать предсказания, полученные на признаках с различных уровней из CNN

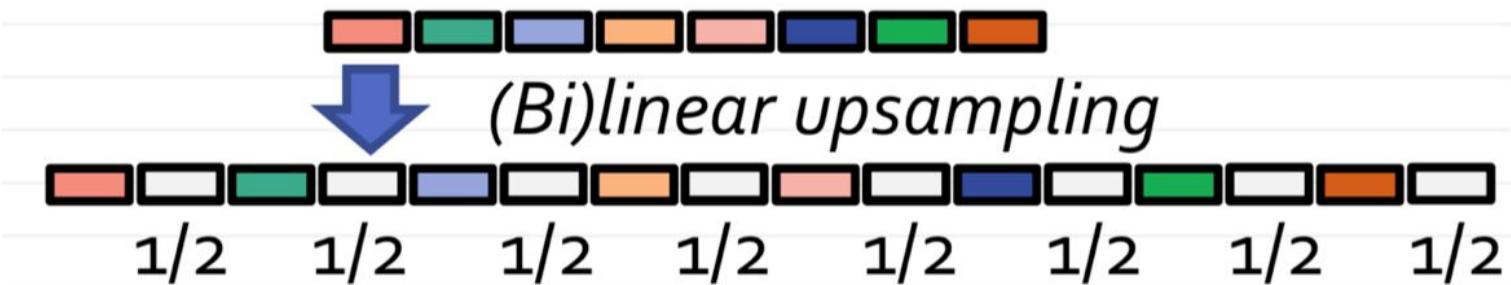
# FCN



# Nearest upsampling

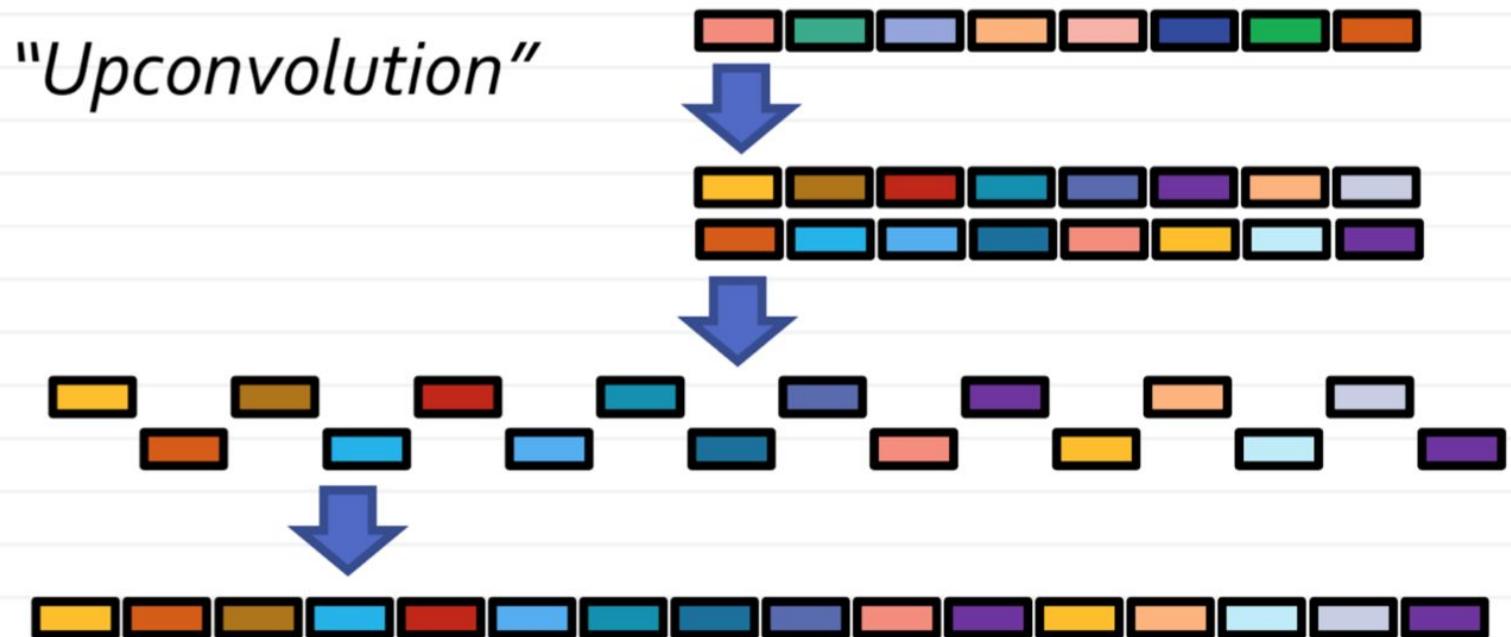


# Bilinear upsampling

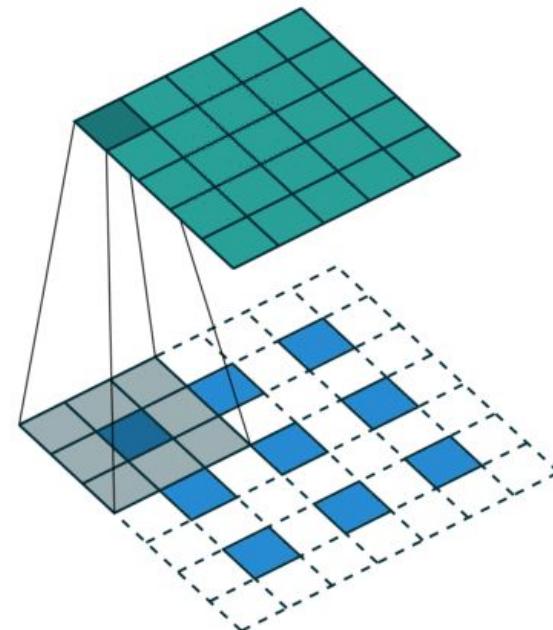
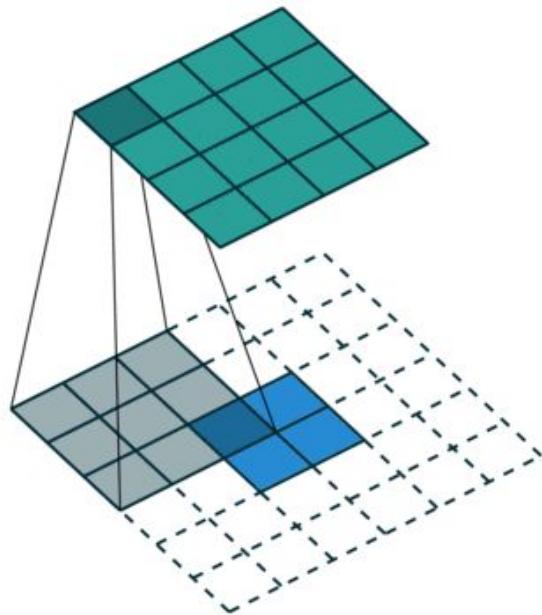


# PixelShuffle (Upconvolution)

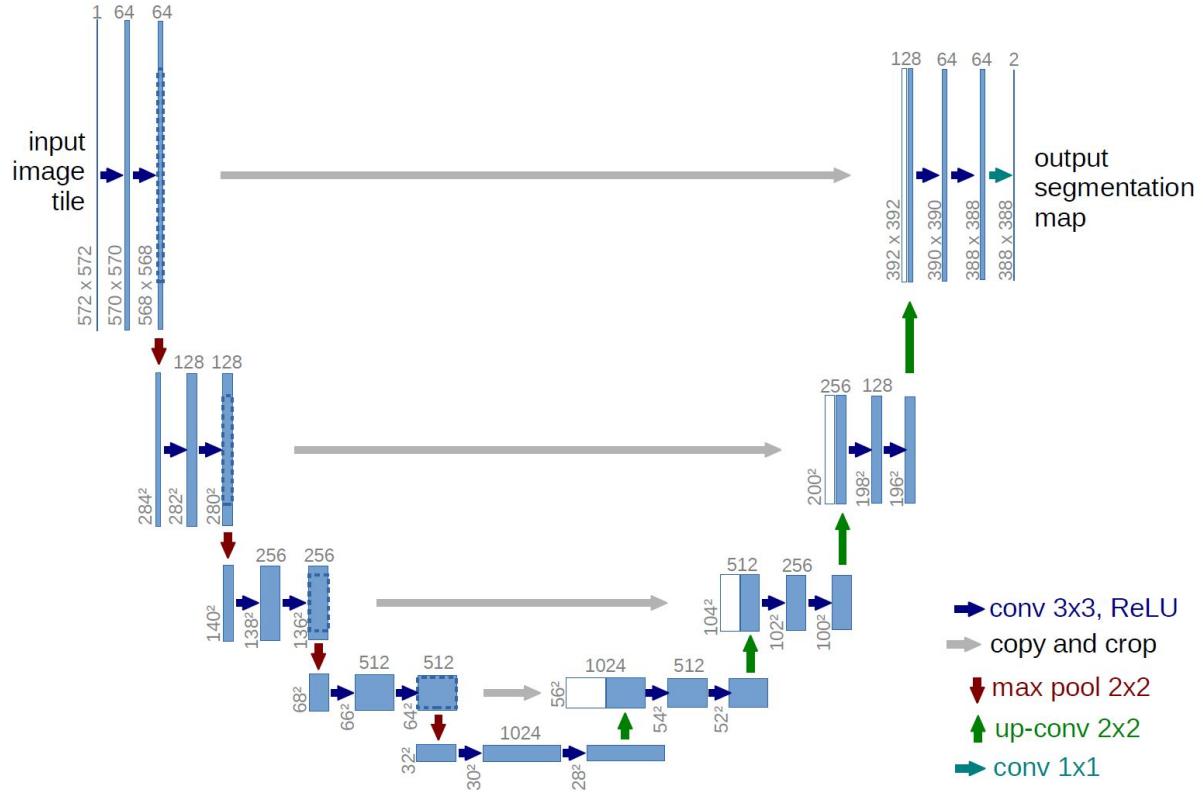
*"Upconvolution"*



# Transposed convolution (Deconvolution)



# UNET (Ronnerberger et al., 2015)



# UNET

- конкатенирует активации энкодера с увеличенными (после upsampling-а) признаками декодера
- такой подход позволяет энкодеру быстрее выучивать необходимые признаки (вспоминаем resnet, densenet и т.д.)
- Архитектура получила популярность из-за простоты реализации

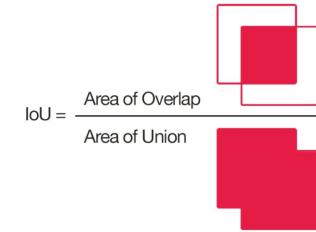
# Лоссы и метрики

Метрики:

- Dice coefficient
- Jaccard Index (Intersection over Union, IoU)

$$DC = \frac{2TP}{2TP + FP + FN} = \frac{2|X \cap Y|}{|X| + |Y|}$$

$$IoU = \frac{TP}{TP + FP + FN} = \frac{|X \cap Y|}{|X| + |Y| - |X \cap Y|}$$



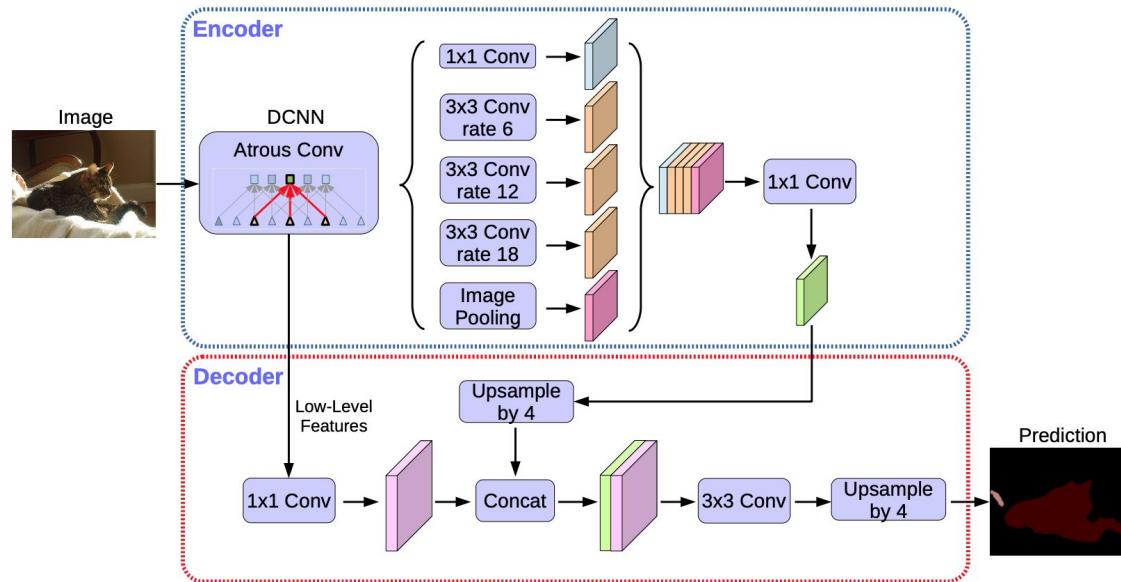
Лоссы:

- Cross entropy  $CE(p, \hat{p}) = -(p \log(\hat{p}) + (1-p) \log(1-\hat{p}))$
- Weighted cross entropy  $WCE(p, \hat{p}) = -(\beta p \log(\hat{p}) + (1-p) \log(1-\hat{p}))$
- Balanced cross entropy  $BCE(p, \hat{p}) = -(\beta p \log(\hat{p}) + (1-\beta)(1-p) \log(1-\hat{p}))$
- Focal Loss  $FL(p, \hat{p}) = -(\alpha(1-\hat{p})^\gamma p \log(\hat{p}) + (1-\alpha)\hat{p}^\gamma(1-p) \log(1-\hat{p}))$
- Dice loss  $DL(p, \hat{p}) = 1 - \frac{2 \sum p_{h,w} \hat{p}_{h,w}}{\sum p_{h,w} + \sum \hat{p}_{h,w}}$

# Основные приемы в semantic segmentation

- Convolutions
- Dilated (Atrous) convolutions
- Upsampling layers
- Skip-connections для сохранения четких мелких деталей

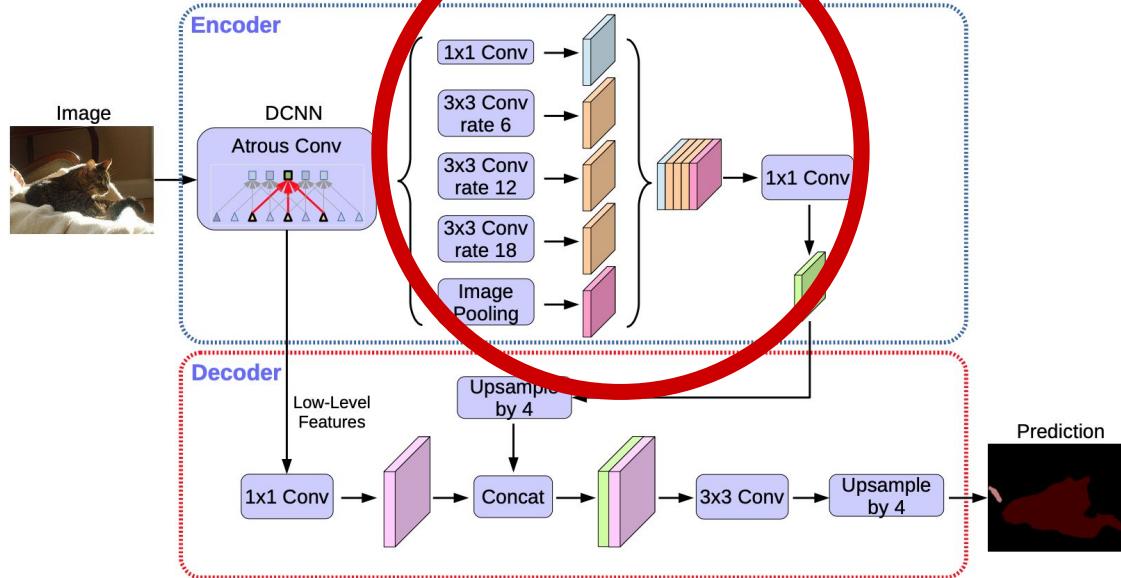
# DeepLab v3+ (2018)



**Fig. 2.** Our proposed DeepLabv3+ extends DeepLabv3 by employing a encoder-decoder structure. The encoder module encodes multi-scale contextual information by applying atrous convolution at multiple scales, while the simple yet effective decoder module refines the segmentation results along object boundaries.

# DeepLab v3+ (2018)

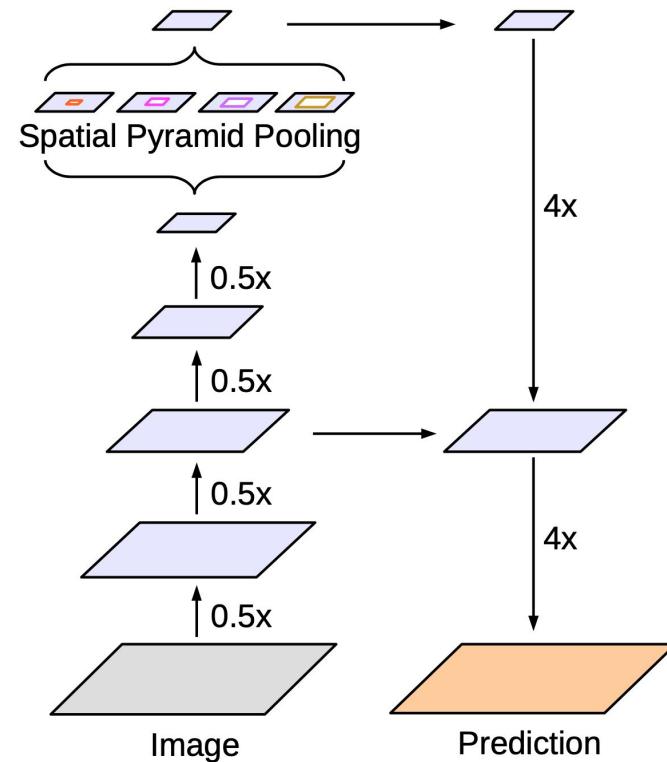
ASPP Block



**Fig. 2.** Our proposed DeepLabv3+ extends DeepLabv3 by employing a encoder-decoder structure. The encoder module encodes multi-scale contextual information by applying atrous convolution at multiple scales, while the simple yet effective decoder module refines the segmentation results along object boundaries.

# DeepLab v3+ (2018) - ASPP Block

- новый блок под названием:  
Atrous Spatial Pyramid Pooling
- приобрел очень большую  
популярность



# Object detection

# Постановка задачи

- определить область на изображении, содержащую объект
- классифицировать найденный объект
- необходимо находить и классифицировать несколько объектов на изображении

# Особенности задачи

- объекты сняты под различными ракурсами
- объекты имеют различные масштабы
- на изображении объекты могут перекрывать друг-друга

# PASCAL VOC 2012

- 20 классов
- 11к изображений



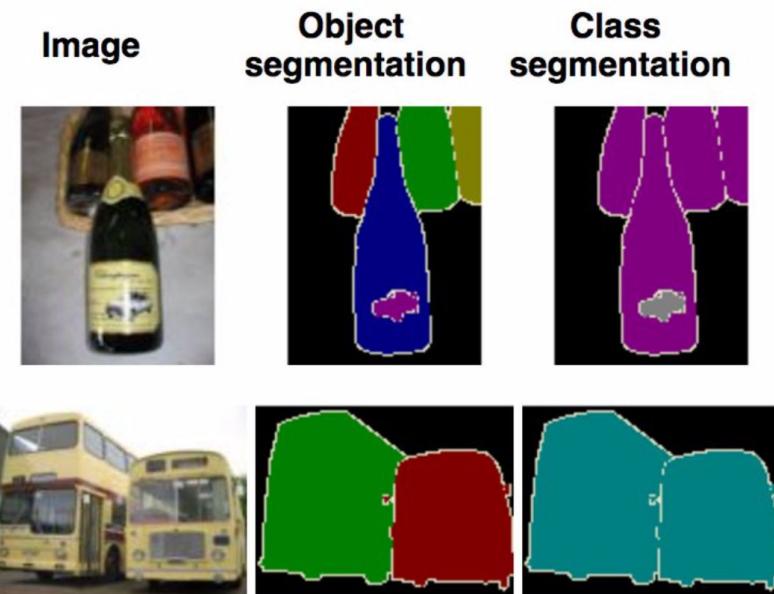
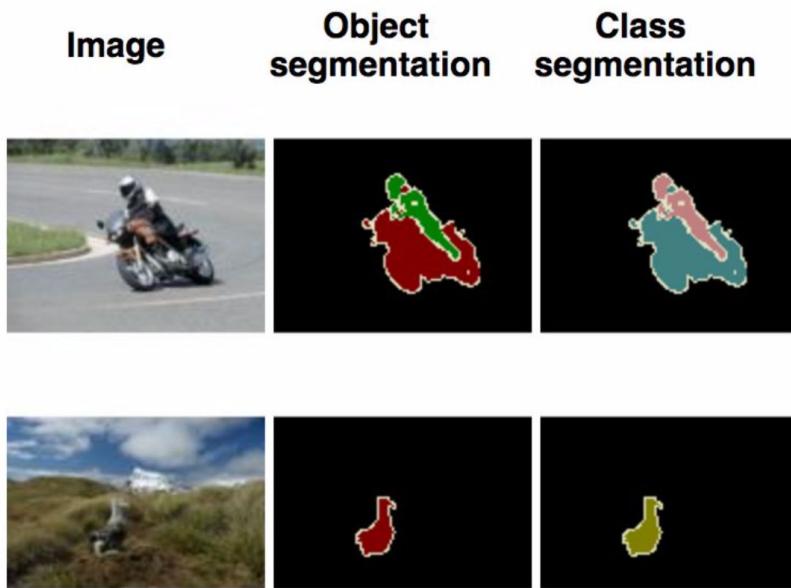
<http://host.robots.ox.ac.uk/pascal/VOC/voc2012/>

# COCO 2014

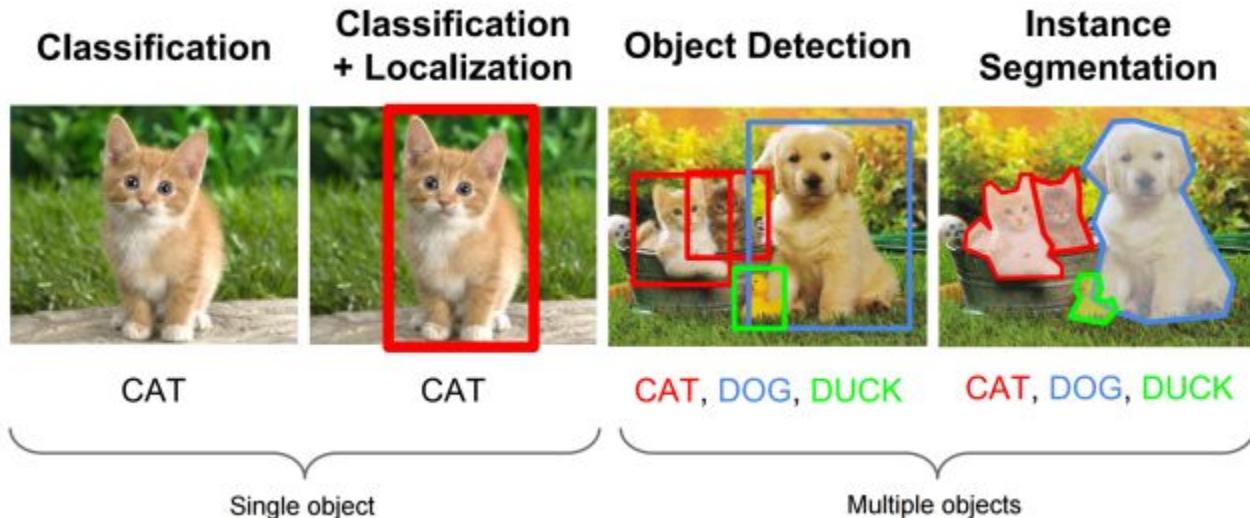
- 80 классов
- 200к размеченных изображений



# Pascal VOC' 12 Dataset

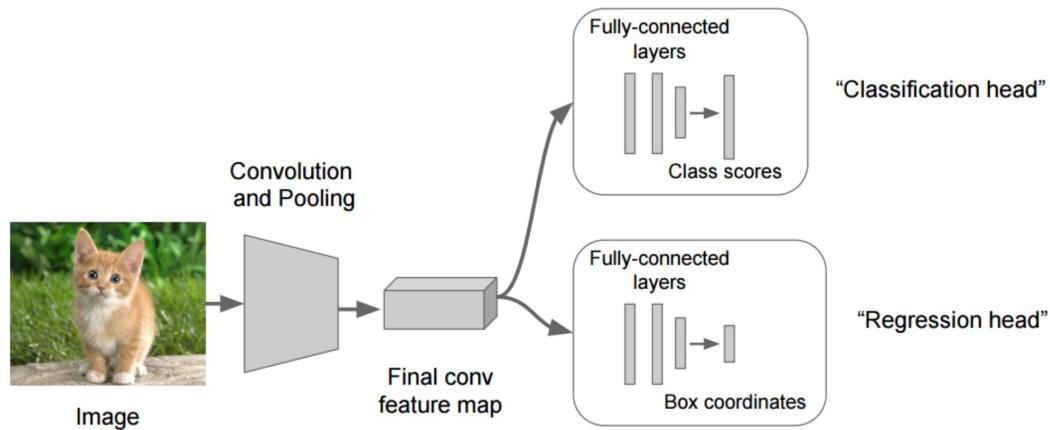


# Object detection



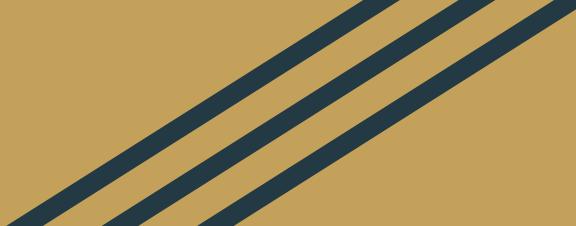
# Локализация 1 объекта с помощью регрессии

- Научимся детектировать один объект только
- Будем предсказывать  $(x_0, y_0, w, h)$  относящиеся к bounding box
- Имея ground truth bbox, используем L2 distance в качестве лосса



# Метрика качества - AP (Average Precision)

[https://medium.com/@jonathan\\_hui/map-mean-average-precision-for-object-detection-45c121a31173](https://medium.com/@jonathan_hui/map-mean-average-precision-for-object-detection-45c121a31173)



# Two-stage detectors R-CNN family

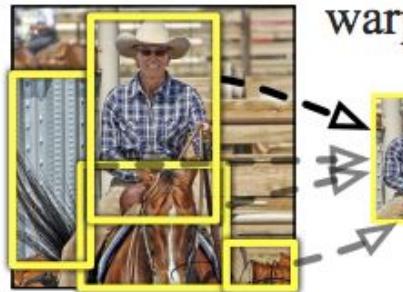


# R-CNN (Girshick, 2014) = regions + CNN

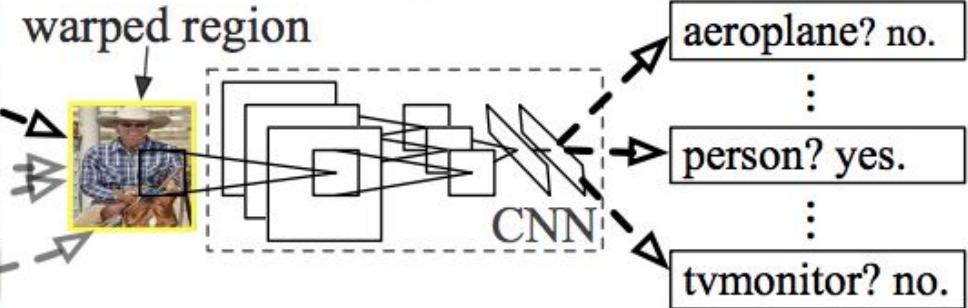
## R-CNN: *Regions with CNN features*



1. Input image



2. Extract region proposals (~2k)



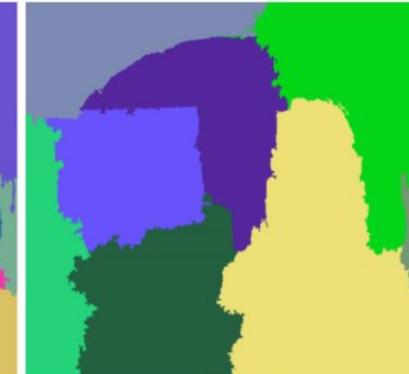
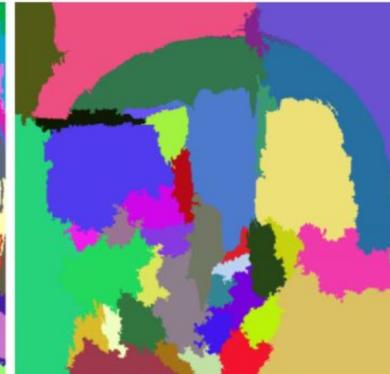
3. Compute CNN features

4. Classify regions

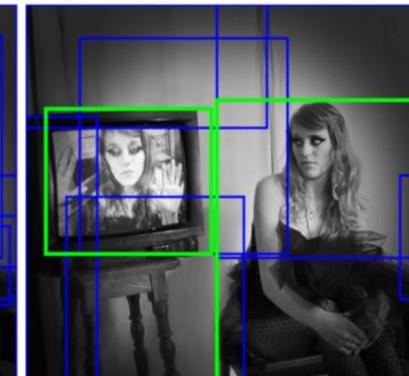
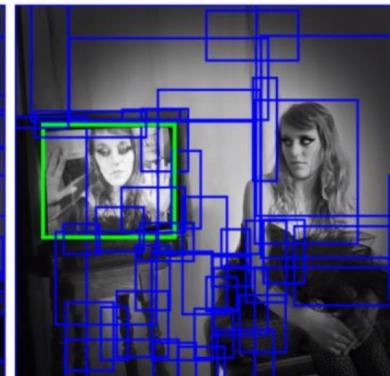
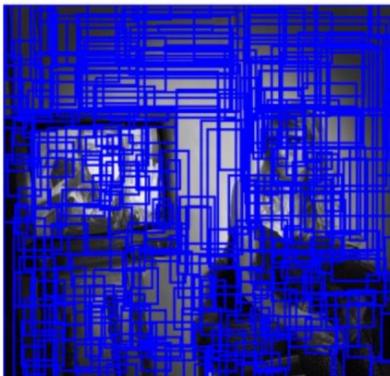
# Selective Search

- предлагается сегментировать объекты на небольшие части (SuperPixel)
- затем, итеративно, двигаясь снизу вверх, объединять эти части
- части объединяются с учетом вероятности принадлежности одному объекту
- в результате объединения частей получаем область, которая наиболее вероятно содержит объект

# Selective Search



Input Image



# R-CNN

- Предобучаем CNN на задаче классификации, например VGG или ResNet на ImageNet датасете.
- Используем Selective Search, чтобы предложить кандидатов - регионы изображения различных размеров
- Приводим всех кандидатов к одинаковому размеру, чтобы подавать на вход CNN
- Файнтюним CNN на K+1 класс, где +1 отвечает за фон
- Получив кандидата, находим его признаки, пропустив через CNN, и подаем на вход бинарным SVM, которые обучены предсказывать каждый класс в отдельности
- Класс определен как положительный, если кандидат имеет  $\text{IoU} \geq 0.3$
- Чтобы уменьшить ошибки локализации, регрессионная модель корректирует положение предсказанного окна, используя CNN признаки (bbox regression)

# Bounding Box Regression

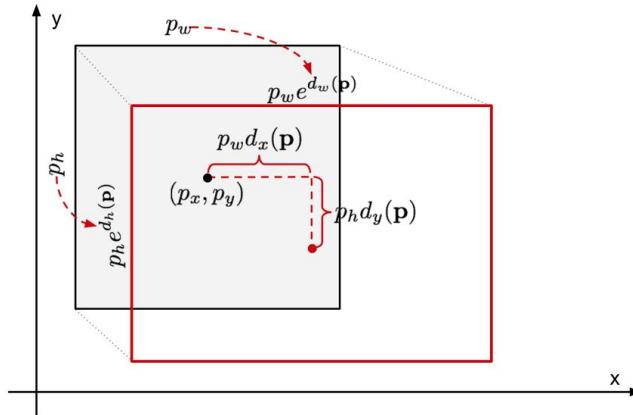
- тренируем регрессор предсказывать scale-invariant transformation между предсказанными координатами  $\hat{r}$  и ground truth координатами  $g$

$$\hat{g}_x = p_w d_x(\mathbf{p}) + p_x$$

$$\hat{g}_y = p_h d_y(\mathbf{p}) + p_y$$

$$\hat{g}_w = p_w \exp(d_w(\mathbf{p}))$$

$$\hat{g}_h = p_h \exp(d_h(\mathbf{p}))$$



# Loss для Bounding Box Regression

$$t_x = (g_x - p_x)/p_w$$

$$t_y = (g_y - p_y)/p_h$$

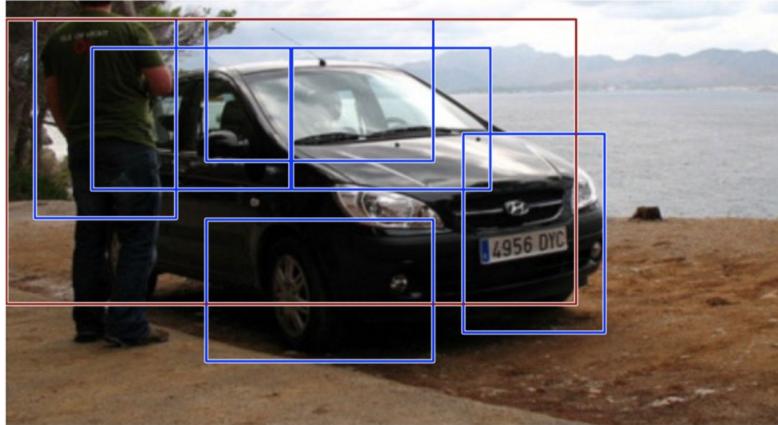
$$t_w = \log(g_w/p_w)$$

$$t_h = \log(g_h/p_h)$$

$$\mathcal{L}_{\text{reg}} = \sum_{i \in \{x, y, w, h\}} (t_i - d_i(\mathbf{p}))^2 + \lambda \|\mathbf{w}\|^2$$

- Необходима регуляризация
- Лосс считается только для тех bbox-ов которые имеют пересечение с ground truth на уровне 0.6 и выше

# Non-Maximum Suppression



Before non-max suppression



After non-max suppression

1. предсказания для одного объекта как правило дублируются
2. определяем  $\text{confidence score} = \text{probability} * \text{IoU}$
3. сортируем bbox по confidence score
4. выбираем bbox с максимальным confidence score
5. все bbox-ы, которые имеют большой IoU (н.  $\geq 0.5$ ) с выбранным на шаге 2, удаляем
6. повторяем шаг 2, 3

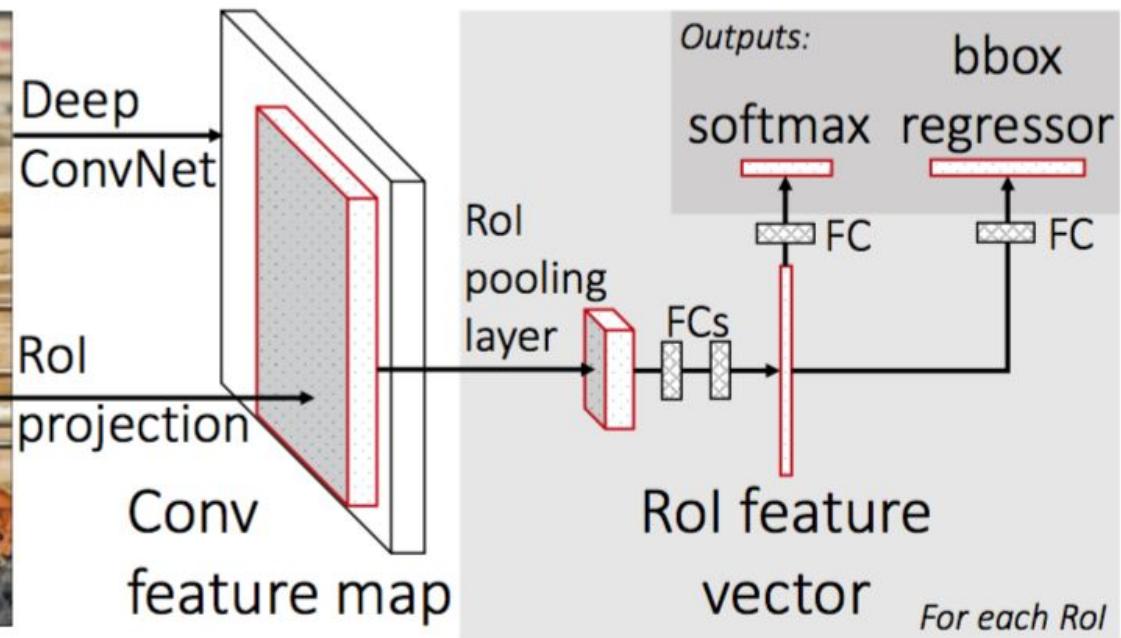
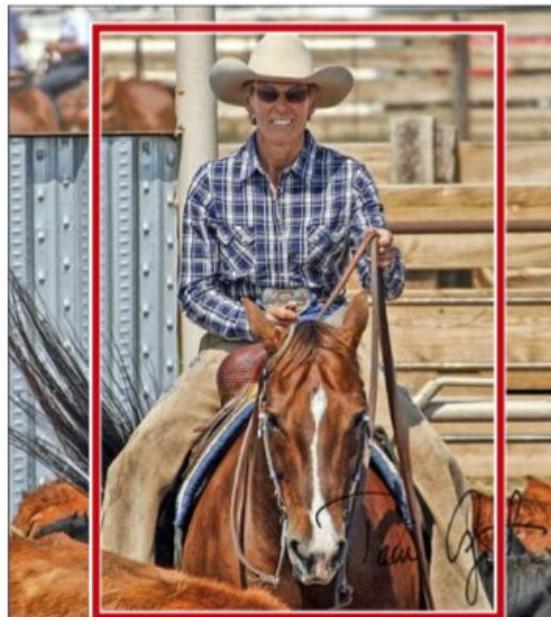
# Hard Negative Mining

- рассматриваем bbox без объектов как негативный пример
- негативные примеры не эквивалентны по уровню сложности определения (простой фон - easy negative, кусок содержит часть объекта - hard negative)
- будем определять hard negative как false positive примеры
- включаем false positive примеры в тренировочную выборку

# Недостатки R-CNN

- Модель медленная и сложная
- Время работы ~15 сек / изображение на Tesla K40
- Selective search ищет 2000 кандидатов
- CNN считает признаки для каждого изображения и кандидата ( $N$  images \* 2000)
- Весь процесс включает 3 модели, которые работают по-отдельности:
  - CNN для вычисления признаков
  - SVM для классификации
  - Регрессия для точной локализации

# Fast R-CNN (Girshick, 2015)



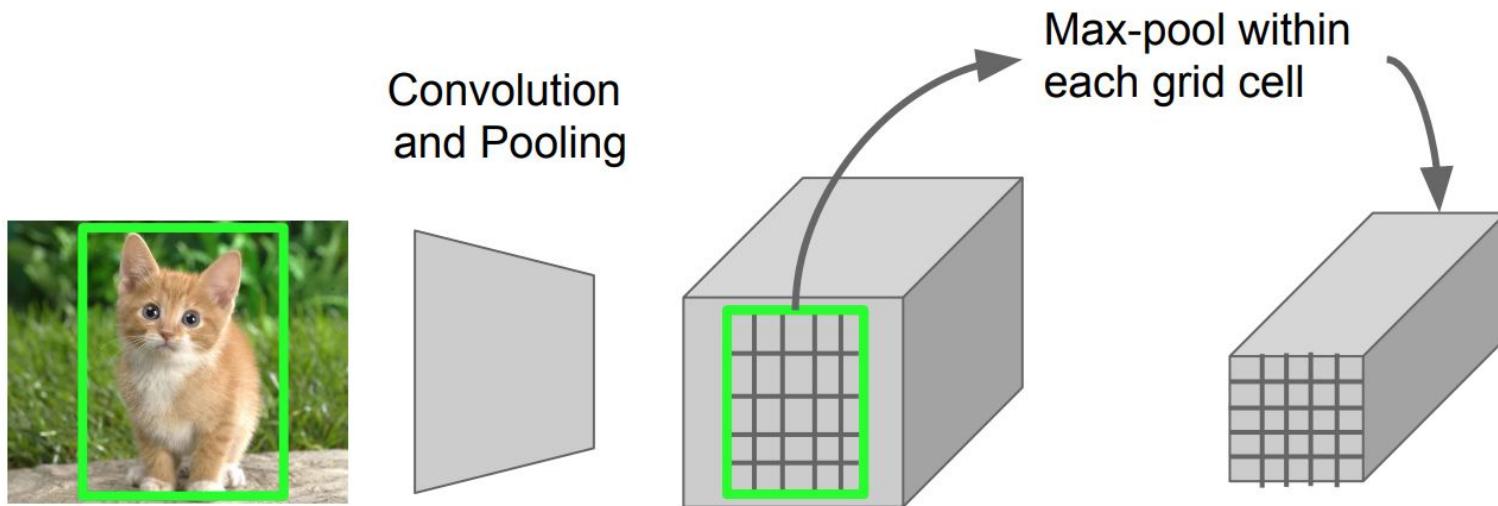
# Fast R-CNN

- для ускорения R-CNN предложено объединить три модели в одну
- время обучения в 9 раз быстрее R-CNN
- время применения в 213 раз быстрее R-CNN
- CNN считает признаки не для каждого кандидата в отдельности, а для всего изображения целиком
- интересующие куски признаков вырезаются с помощью RoI Pooling и подаются на вход классификатору и bbox регрессору

# RoI Pooling

- для построения модели классификации необходимо, чтобы число признаков у объектов совпадало
- т.к. объекты на изображении могут иметь различные размеры, то размеры соответствующих областей в Pooling слое также будут различаться
- слой RoI Pooling решает эту проблему с помощью операции Max Pooling
- причем размер окна и шаг Max Pooling зависит от размера интересующей нас области

# Roi Pooling



Hi-res input image:  
 $3 \times 800 \times 600$   
with region  
proposal

Hi-res conv features:  
 $C \times H \times W$   
with region proposal

Roi conv features:  
 $C \times h \times w$   
for region proposal

# Fast R-CNN

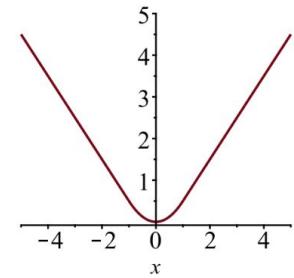
- предобучаем CNN на задаче классификации, например VGG или ResNet на ImageNet датасете.
- используем Selective Search, чтобы предложить кандидатов - регионы изображения различных размеров
- меняем архитектуру предобученной CNN
  - Заменяем Max Pooling на RoI Pooling, который выдает признаки фиксированного размера для кандидатов
  - Заменяем последний fully-connected на новый fully-connected
- создаем на конце две ветки
  - Одна ветка это softmax на  $K+1$  класс
  - Вторая ветка предсказывает сдвиги относительно кандидата-региона (bounding box regression)

# Loss для Fast R-CNN

## Symbol Explanation

- $u$  True class label,  $u \in 0, 1, \dots, K$  ; by convention, the catch-all background class has  $u = 0$  .
- $p$  Discrete probability distribution (per ROI) over  $K + 1$  classes:  
 $p = (p_0, \dots, p_K)$  , computed by a softmax over the  $K + 1$  outputs of a fully connected layer.
- $v$  True bounding box  $v = (v_x, v_y, v_w, v_h)$  .
- $t^u$  Predicted bounding box correction,  $t^u = (t_x^u, t_y^u, t_w^u, t_h^u)$

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



$$\mathcal{L}(p, u, t^u, v) = \mathcal{L}_{\text{cls}}(p, u) + \mathbb{1}[u \geq 1] \mathcal{L}_{\text{box}}(t^u, v)$$

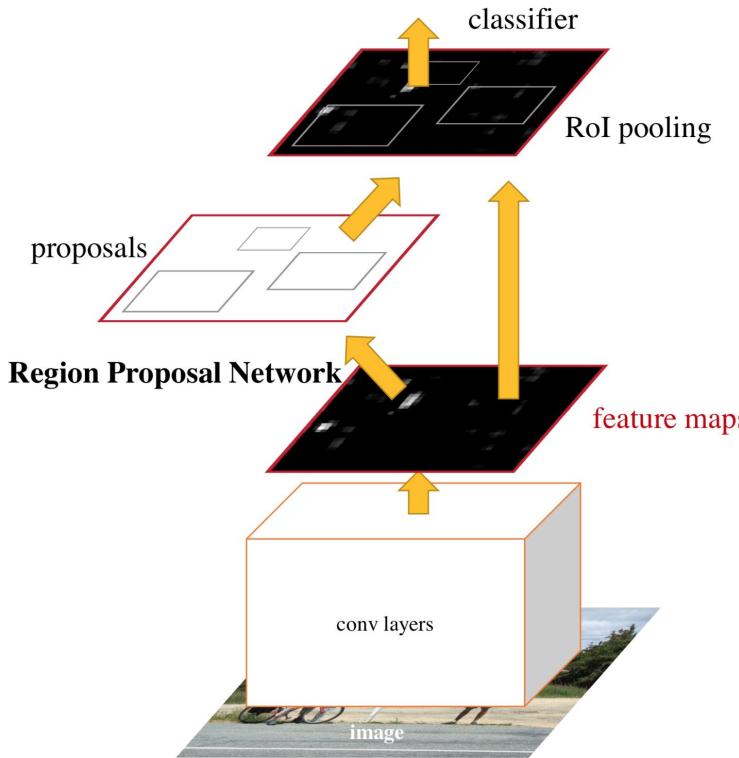
$$\mathcal{L}_{\text{cls}}(p, u) = -\log p_u$$

$$\mathcal{L}_{\text{box}}(t^u, v) = \sum_{i \in \{x, y, w, h\}} L_1^{\text{smooth}}(t_i^u - v_i)$$

# Недостатки Fast R-CNN

- Fast R-CNN стал быстрее на трейне и на teste, но регионы-кандидаты до сих пор считались на отдельной модели и это замедляет процесс

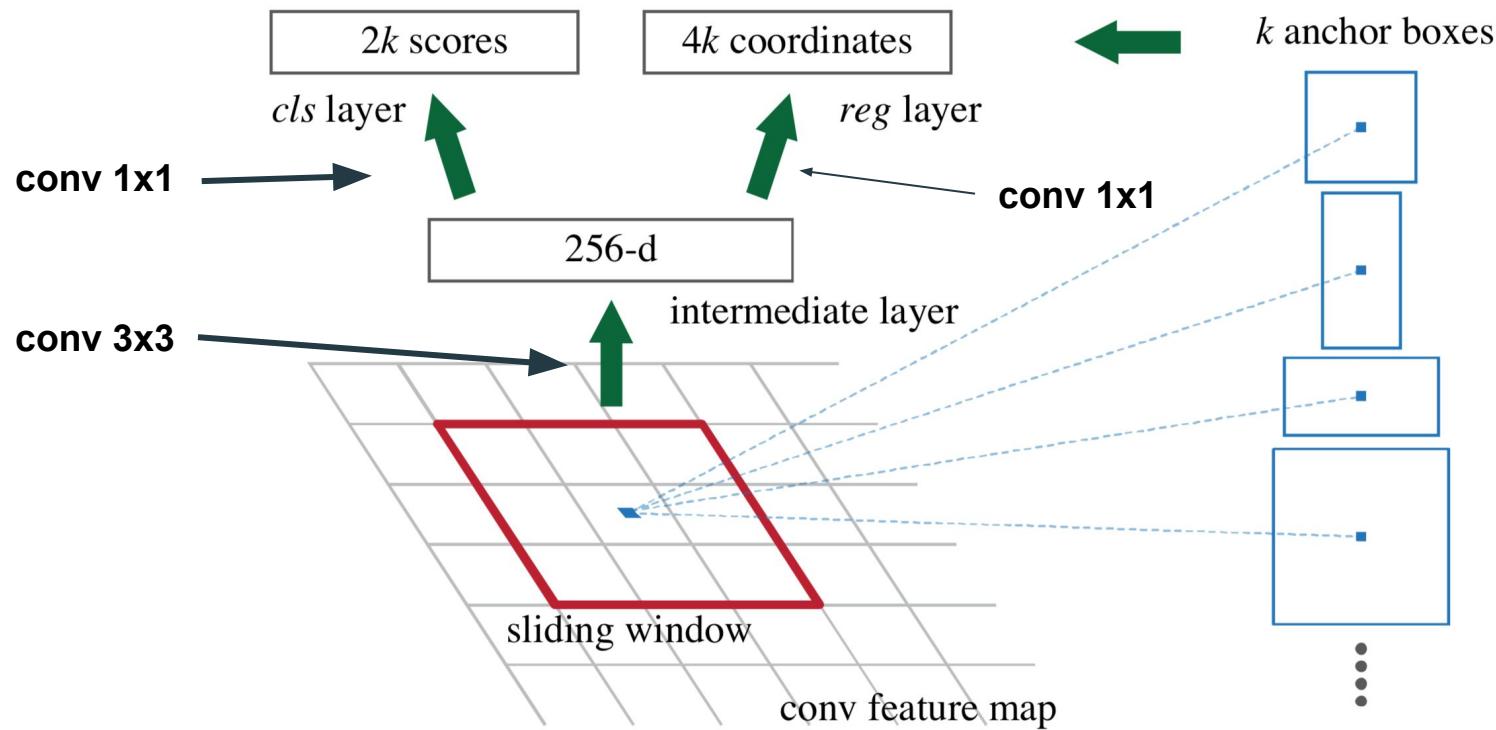
# Faster RCNN (Ren, 2016)



# Faster RCNN

- Интуитивный способ ускорить сеть - это включить поиск кандидатов в CNN
- в отличие от R-CNN и Fast R-CNN эта модель сама генерирует RoI
- для генерации RoI используется Region Proposal Network (RPN)
- скорость работы ~5fps

# Region Proposal Network (RPN)



# Обучаем RPN - классификация

- RPN предсказывает два класса - объект (1) и не объект (0)
- Как зададим таргет для anchors?
  - Для каждого ground truth bbox ищем anchor с максимальным IoU и даем ему таргет класс 1
  - Если anchor имеет IoU с каким либо ground truth bbox  $> 0.7$  даем ему таргет класс 1
  - Если anchor имеет IoU со всеми ground truth bbox  $< 0.3$  даем ему таргет класс 0
  - Все остальные anchors не учитываем

# Обучаем RPN - bbox regression

For bounding box regression, we adopt the parameterizations of the 4 coordinates following [5]:

$$\begin{aligned} t_x &= (x - x_a)/w_a, & t_y &= (y - y_a)/h_a, \\ t_w &= \log(w/w_a), & t_h &= \log(h/h_a), \\ t_x^* &= (x^* - x_a)/w_a, & t_y^* &= (y^* - y_a)/h_a, \\ t_w^* &= \log(w^*/w_a), & t_h^* &= \log(h^*/h_a), \end{aligned} \tag{2}$$

where  $x$ ,  $y$ ,  $w$ , and  $h$  denote the box's center coordinates and its width and height. Variables  $x$ ,  $x_a$ , and  $x^*$  are for the predicted box, anchor box, and ground-truth box respectively (likewise for  $y, w, h$ ). This can

# Faster R-CNN

1. предобучаем CNN на задаче классификации (на выходе выдает признаки размером  $W \times C \times H$ ), назовем это CNN-backbone
2. файнтюним RPN на задаче объект-не объект
  - Общее количество anchors равно  $W \times H \times k$  ( $k=9$  число шаблонов для anchors)
3. обучаем Fast R-CNN используя кандидатов от RPN, которые были получены отдельно
4. присоединяем RPN к бэкбону Fast R-CNN
5. файнтюним только слои относящиеся непосредственно к RPN
6. файнтюним уникальные слои для Fast-R-CNN
7. повторяет шаги 5, 6

# Loss для Faster R-CNN

---

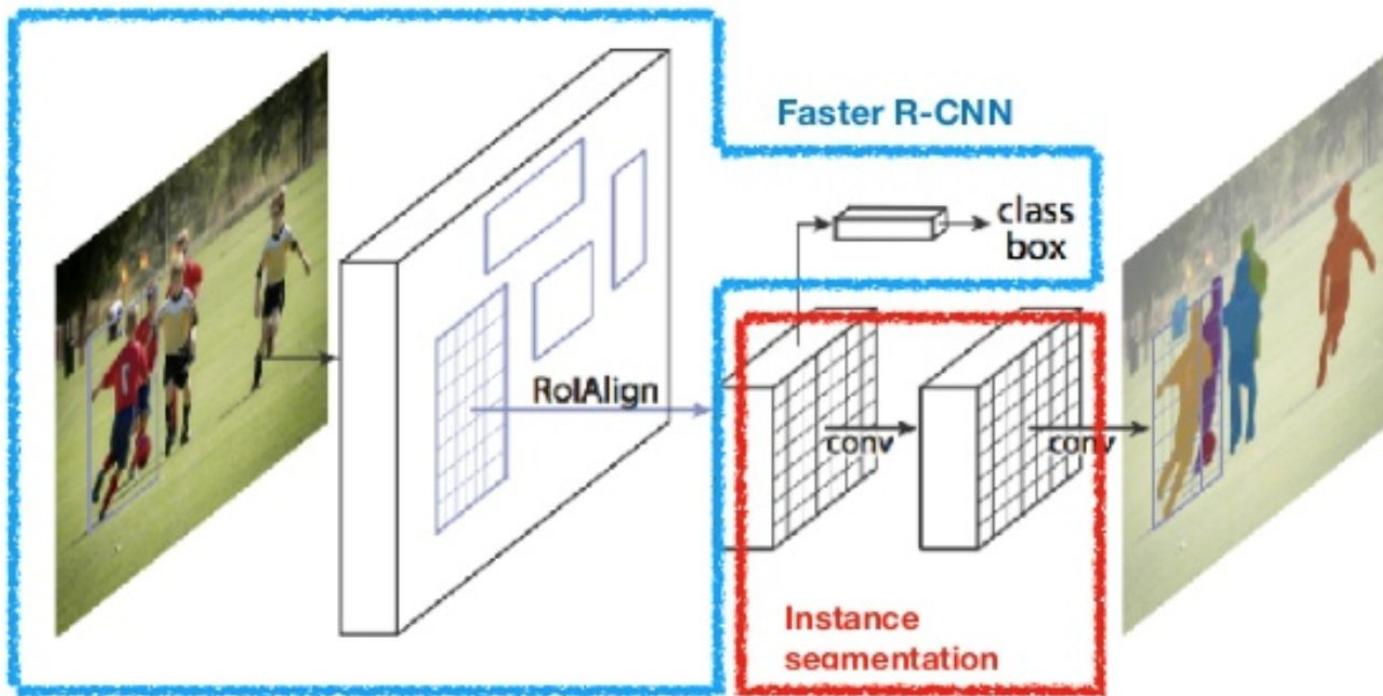
## Symbol Explanation

$p_i$	Predicted probability of anchor $i$ being an object.
$p_i^*$	Ground truth label (binary) of whether anchor $i$ is an object.
$t_i$	Predicted four parameterized coordinates.
$t_i^*$	Ground truth coordinates.
$N_{\text{cls}}$	Normalization term, set to be mini-batch size (~256) in the paper.
$N_{\text{box}}$	Normalization term, set to the number of anchor locations (~2400) in the paper.
$\lambda$	A balancing parameter, set to be ~10 in the paper (so that both $\mathcal{L}_{\text{cls}}$ and $\mathcal{L}_{\text{box}}$ terms are roughly equally weighted).

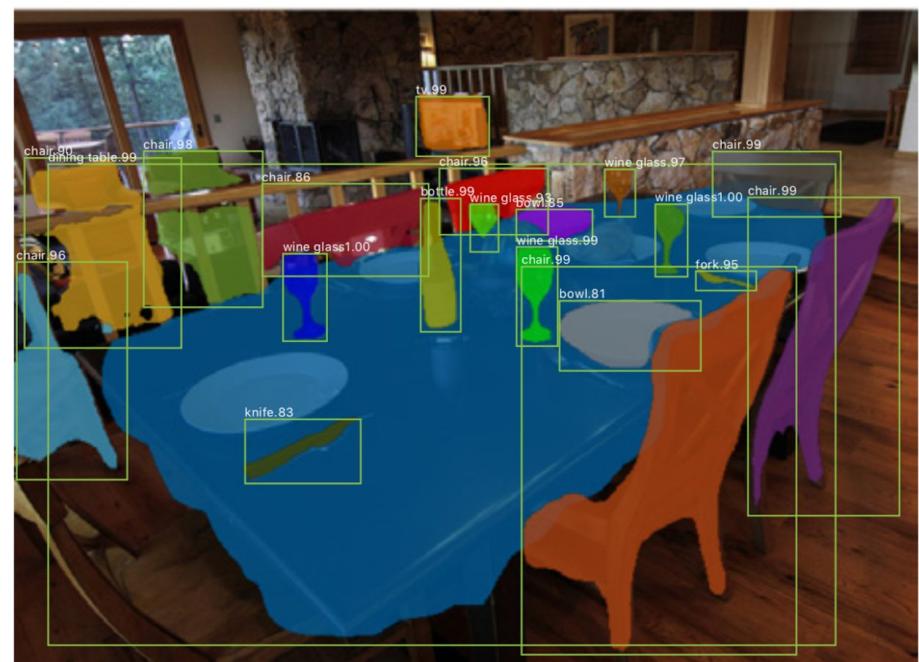
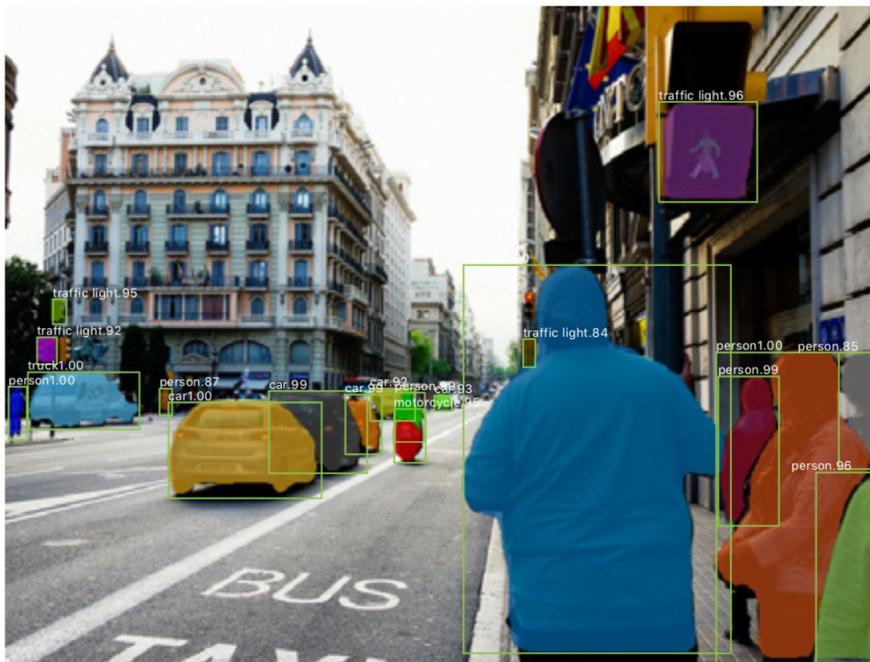
---

$$\begin{aligned}\mathcal{L} &= \mathcal{L}_{\text{cls}} + \mathcal{L}_{\text{box}} \\ \mathcal{L}(\{p_i\}, \{t_i\}) &= \frac{1}{N_{\text{cls}}} \sum_i \mathcal{L}_{\text{cls}}(p_i, p_i^*) + \frac{\lambda}{N_{\text{box}}} \sum_i p_i^* \cdot L_1^{\text{smooth}}(t_i - t_i^*)\end{aligned}$$

# Mask R-CNN (He, et al. 2017)



# Mask R-CNN (task: Instance Segmentation)



# Mask R-CNN

- добавляет к Faster-RCNN голову для предсказания маски объекта
- решает задачу Object Detection + Instance Segmentation
- ключевая особенность - разделить предсказания семантического класса объекта (одно число) и по-пиксельной маски
- для этого добавляется **новая голова (mask head)**, которая параллельно предсказывает маску объекта
- так как предсказание маски требует аккуратного соответствия / выравнивания фичей, добавлен **RoIAlign** слой

# RoIAlign

- Т.к. в RoIPooling слое у нас координаты на “фичемапах” на выходе CNN, соответствующие позициям на изображении, округлялись - это создает ситуацию когда “фичи” для восстановления попиксельной итоговой маски плохо выровнены
- Для этого с помощью bilinear interpolation мы можем явно посчитать “фичи” соответствующие например явно  $x/16$  вместо округления  $[x/16]$  как было в RoIPooling

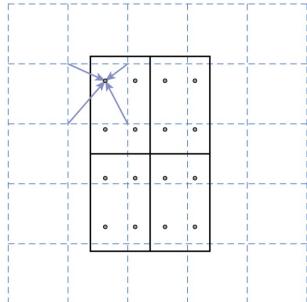
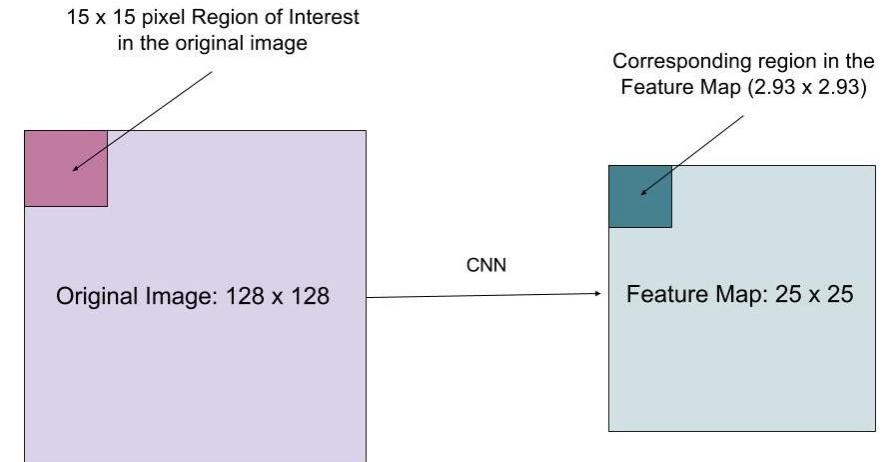


Figure 3. **RoIAlign:** The dashed grid represents a feature map, the solid lines an ROI (with  $2 \times 2$  bins in this example), and the dots the 4 sampling points in each bin. RoIAlign computes the value of each sampling point by bilinear interpolation from the nearby grid points on the feature map. No quantization is performed on any coordinates involved in the ROI, its bins, or the sampling points.



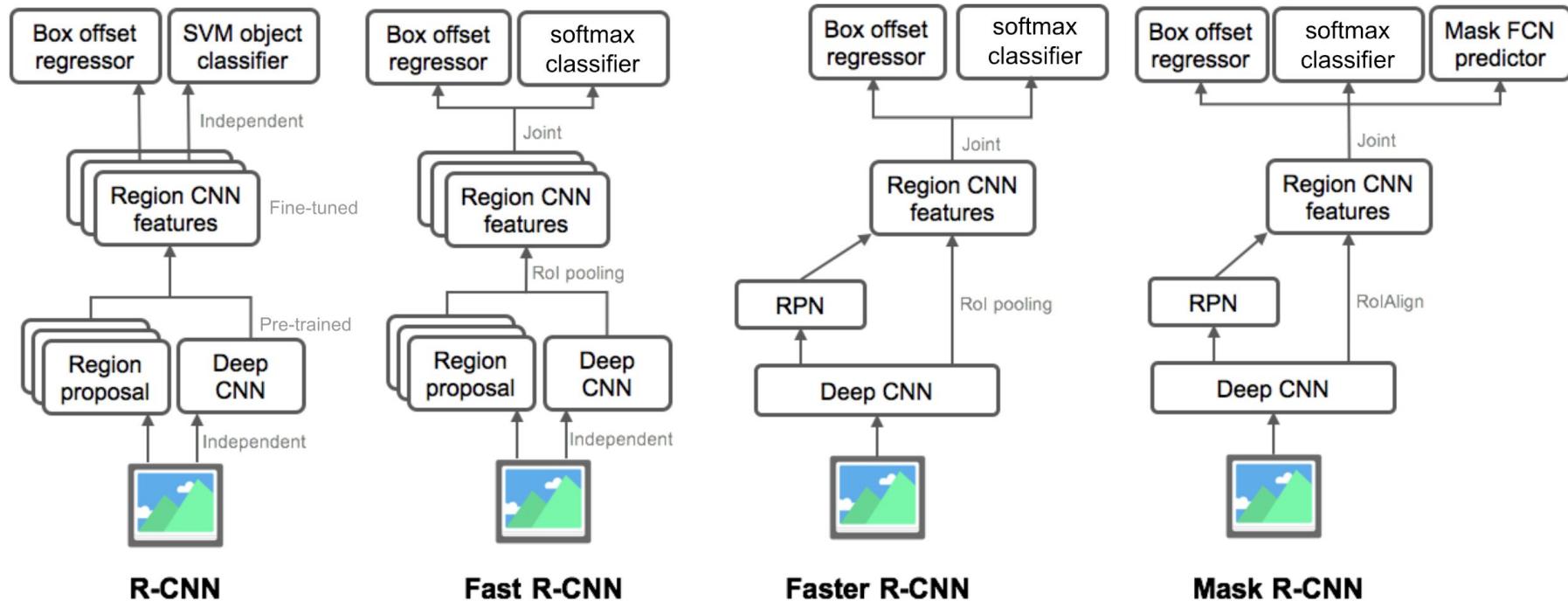
# Mask R-CNN loss function

- лосс для Mask R-CNN аналогичен Faster R-CNN, только добавляется еще лосс для маски
- предсказывается маска для каждого ROI (региона-кандидата) размером  $m \times m$  (обычно  $m=14$ )
- всего  $K$  классов, поэтому на маску имеет размер  $K \times m \times m$
- маска предсказывается для каждого класса в отдельности, а итоговая выбирается в зависимости от того какой класс выбрала “классификационная голова”, соответственно на ней и будет считаться лосс

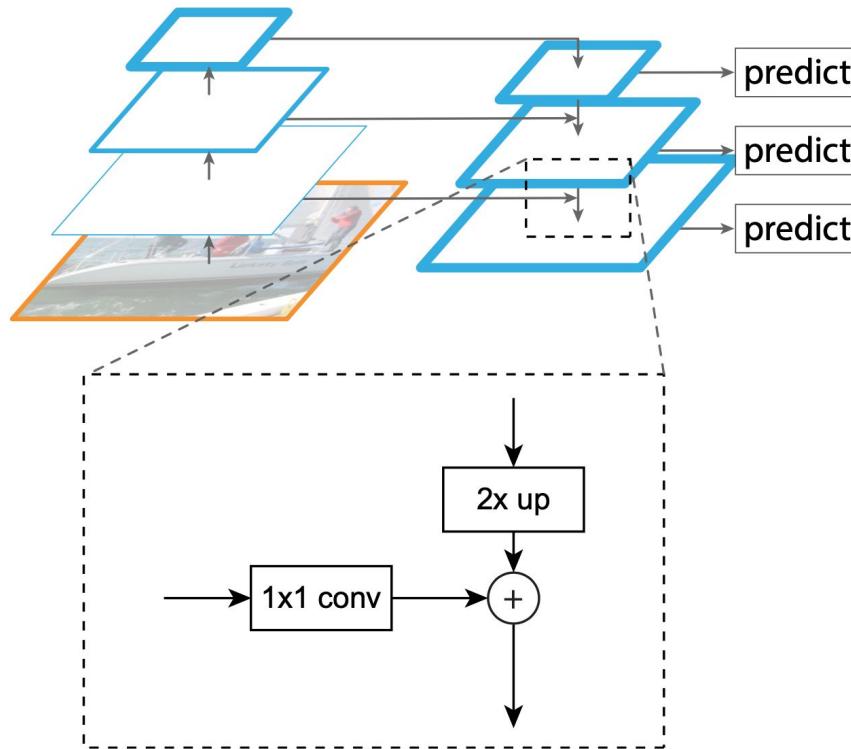
$$\mathcal{L} = \mathcal{L}_{\text{cls}} + \mathcal{L}_{\text{box}} + \mathcal{L}_{\text{mask}}$$

$$\mathcal{L}_{\text{mask}} = -\frac{1}{m^2} \sum_{1 \leq i, j \leq m} \left[ y_{ij} \log y_{ij}^k + (1 - y_{ij}) \log(1 - y_{ij}^k) \right]$$

# Семейство R-CNN



# FPN (Feature Pyramid Network, 2016)



# FPN

- маленькие объекты сложно детектировать на признаках, полученных с глубоких слоев сетки
- поэтому используется feature pyramid, и предсказания делаются на каждом уровне пирамиды
- один из самых популярных и основных компонентов для сеток по object detection, instance segmentation
- FPN по сути это UNET :D

# One-stage detectors

# YOLO: You Only Look Once (2015)

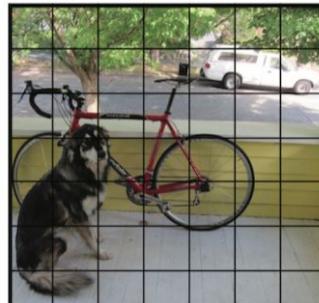
Алгоритм:

1. Предобучаем CNN на классификационной задачи
2. Делим изображение на  $S \times S$  клеток. Клетка, в которую попал центр объекта, становится ответственной за предсказание этого объекта. Для каждой клетки будем предсказывать параметры для bounding боксов
  - a. Координаты  $(x, y, w, h)$  для каждого бокса в клетке
  - b. Confidence score для каждого бокса, который оценивает вероятность того что клетка содержит объект в этом боксе
  - c. Предсказание одного из  $K$  классов для клетки. Исходим из того что каждый бокс в клетке имеет один и тот же класс
3. Итого сетка на выходе имеет  $S \times S \times (5B + K)$  предсказаний

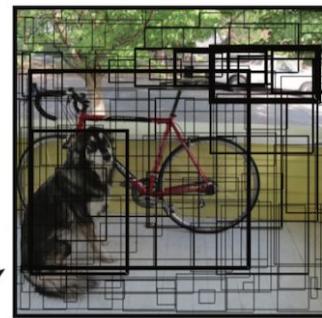
# YOLO: You Only Look Once (2015)

$S \times S \times B$  bounding boxes

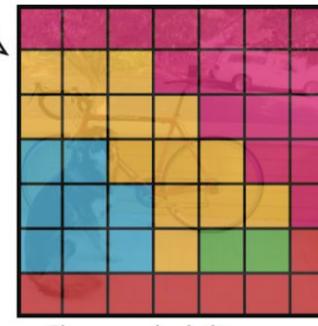
**confidence** =  $Pr(\text{object}) \times \text{IoU}(\text{pred, truth})$



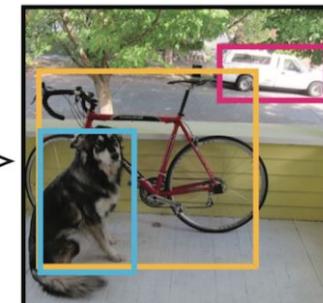
$S \times S$  grid on input



Bounding boxes + confidence



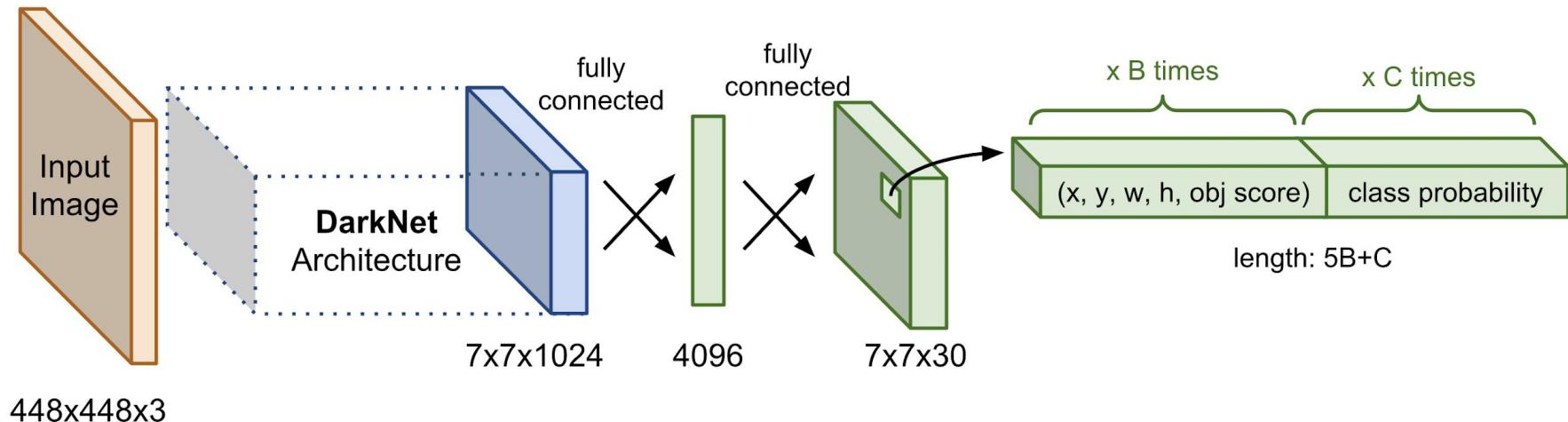
Class probability map



Final detections

$Pr(\text{Class}_i | \text{object})$

# YOLO: You Only Look Once (2015)



# YOLO: You Only Look Once (2015)

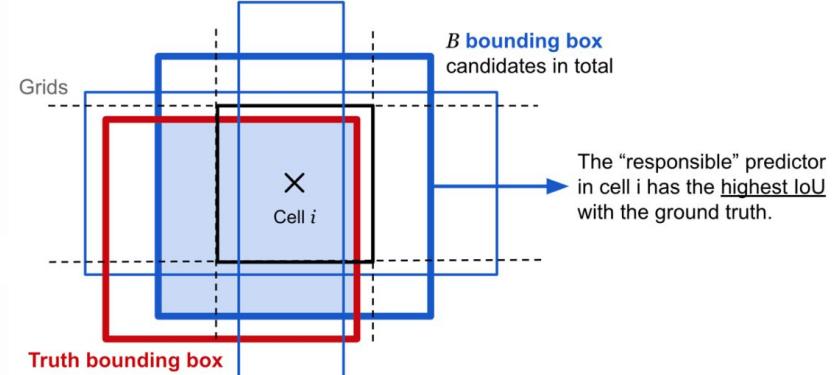
$$\mathcal{L}_{\text{loc}} = \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^B \mathbb{1}_{ij}^{\text{obj}} [(x_i - x_i^*)^2 + (y_i - y_i^*)^2 + (\sqrt{w_i} - \sqrt{w_i^*})^2 + (\sqrt{h_i} - \sqrt{h_i^*})^2]$$

$$\mathcal{L}_{\text{cls}} = \sum_{i=0}^{S^2} \sum_{j=0}^B (\mathbb{1}_{ij}^{\text{obj}} + \lambda_{\text{noobj}}(1 - \mathbb{1}_{ij}^{\text{obj}})) (C_{ij} - \hat{C}_{ij})^2 + \sum_{i=0}^{S^2} \sum_{c \in C} \mathbb{1}_i^{\text{obj}} (p_i(c) - \hat{p}_i(c))^2$$

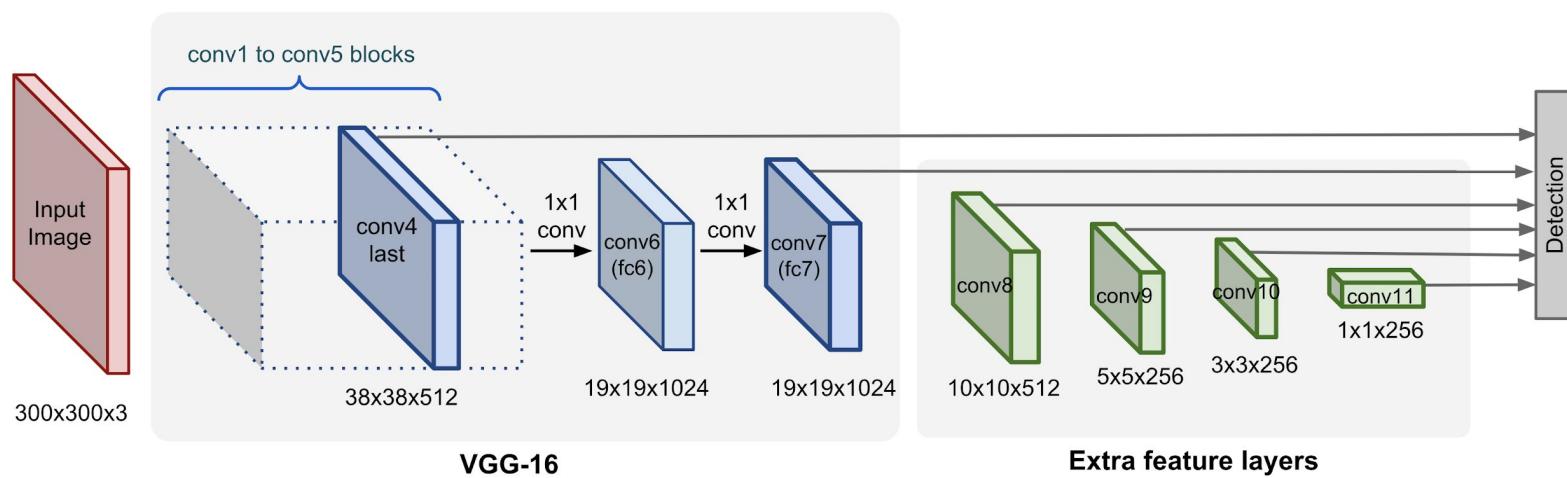
$$\mathcal{L} = \mathcal{L}_{\text{loc}} + \mathcal{L}_{\text{cls}}$$

where,

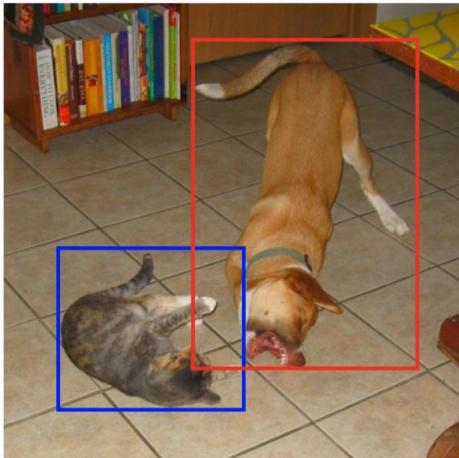
- $\mathbb{1}_i^{\text{obj}}$ : An indicator function of whether the cell  $i$  contains an object.
- $\mathbb{1}_{ij}^{\text{obj}}$ : It indicates whether the  $j$ -th bounding box of the cell  $i$  is “responsible” for the object prediction (see Fig. 3).
- $C_{ij}$ : The confidence score of cell  $i$ ,  $\Pr(\text{containing an object}) * \text{IoU}(\text{pred}, \text{truth})$ .
- $\hat{C}_{ij}$ : The predicted confidence score.
- $C$ : The set of all classes.
- $p_i(c)$ : The conditional probability of whether cell  $i$  contains an object of class  $c \in C$
- $\hat{p}_i(c)$ : The predicted conditional class probability.



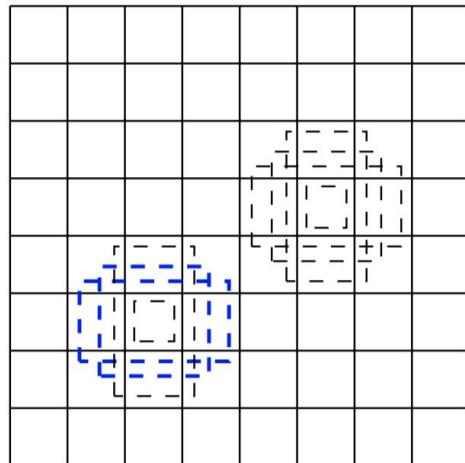
# SSD: Single Shot MultiBox Detector



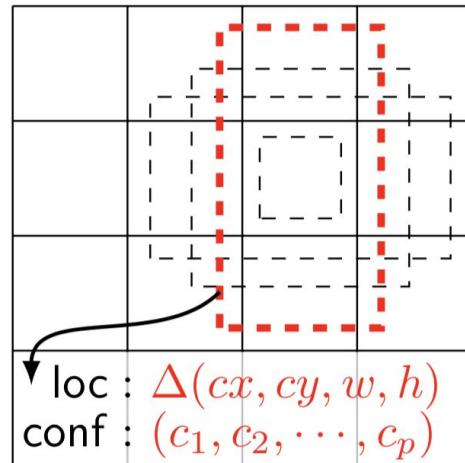
# SSD: Single Shot MultiBox Detector



(a) Image with GT boxes



(b)  $8 \times 8$  feature map



loc :  $\Delta(cx, cy, w, h)$   
conf :  $(c_1, c_2, \dots, c_p)$

(c)  $4 \times 4$  feature map

# SSD: Single Shot MultiBox Detector

level index:  $\ell = 1, \dots, L$

$$\text{scale of boxes: } s_\ell = s_{\min} + \frac{s_{\max} - s_{\min}}{L - 1}(\ell - 1)$$

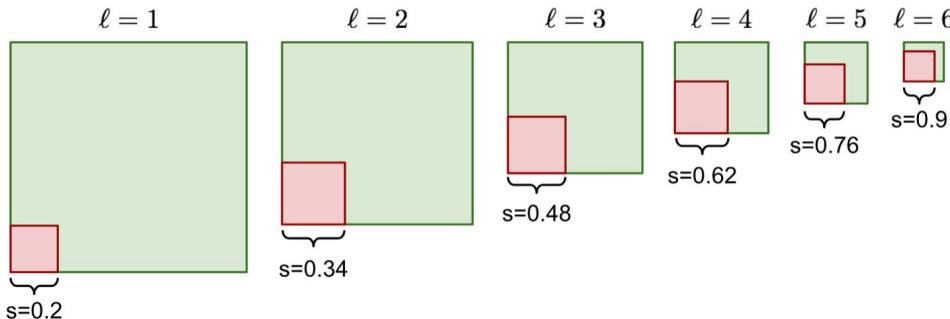
aspect ratio:  $r \in \{1, 2, 3, 1/2, 1/3\}$

additional scale:  $s'_\ell = \sqrt{s_\ell s_{\ell+1}}$  when  $r = 1$  thus, 6 boxes in total.

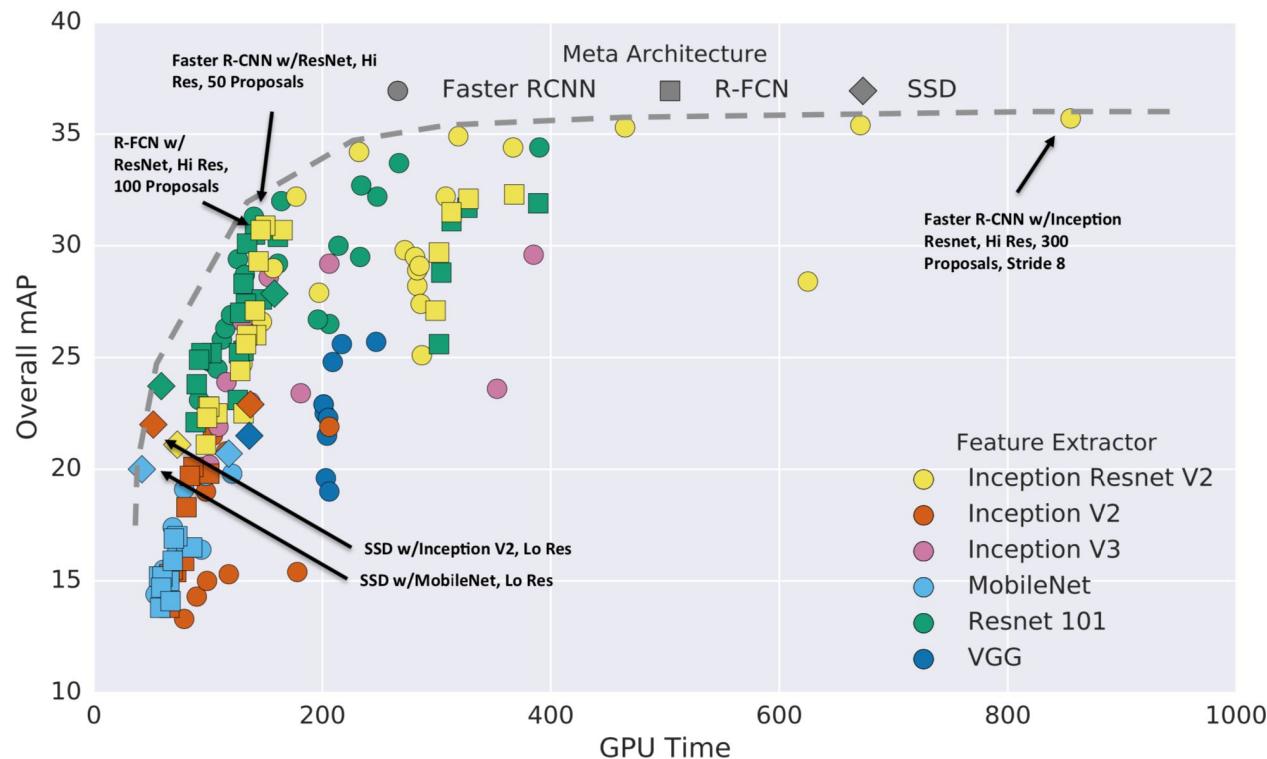
$$\text{width: } w_\ell^r = s_\ell \sqrt{r}$$

$$\text{height: } h_\ell^r = s_\ell / \sqrt{r}$$

$$\text{center location: } (x_\ell^i, y_\ell^j) = \left( \frac{i + 0.5}{m}, \frac{j + 0.5}{n} \right)$$



# Сравнение качества моделей детекции



# awesome github

- [Detectron2](#)
- [MMDetection](#)
- [Segmentation Models](#) со своим каналом в ODS: #tool\_segmentation
- в каждом из репозитории сверху есть набор туториалов