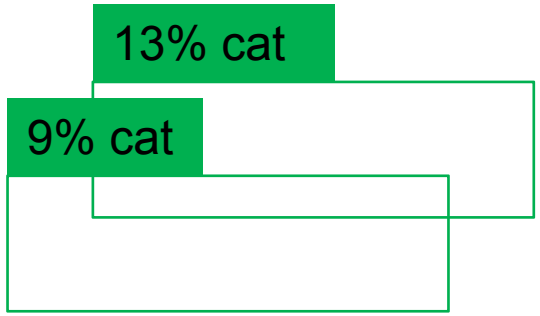
Object Detection



99% ISP RAS

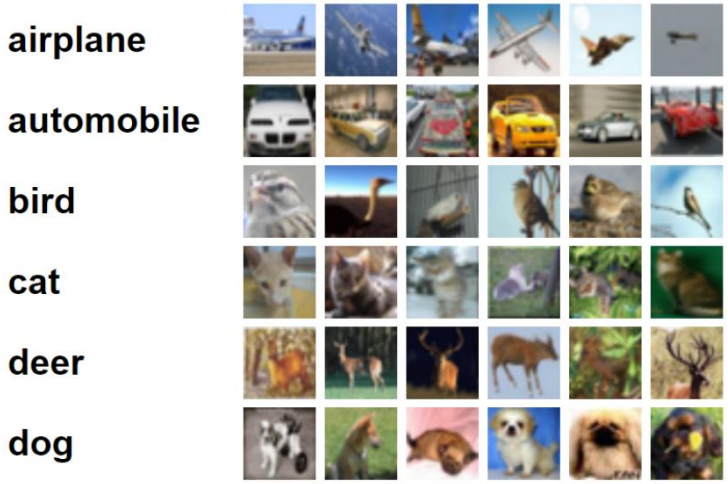
Беляева Оксана Владимировна

2021 год

ИСП РАН

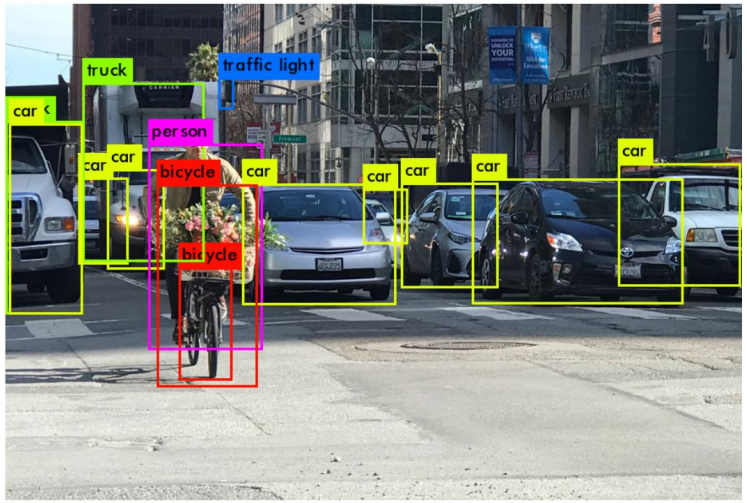
Задачи компьютерного зрения

Классификация



+

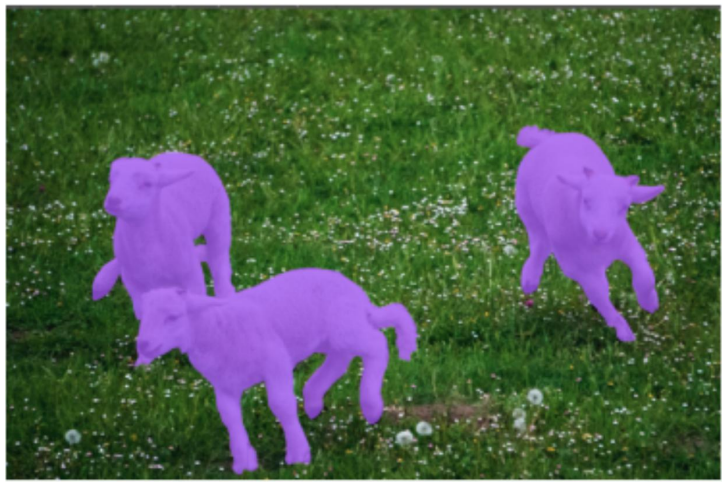
Детекция (Object Detection)



И другие задачи:

- pose estimation
- face recognition
- super resolution
- генерация картинок (GAN)
- и др.

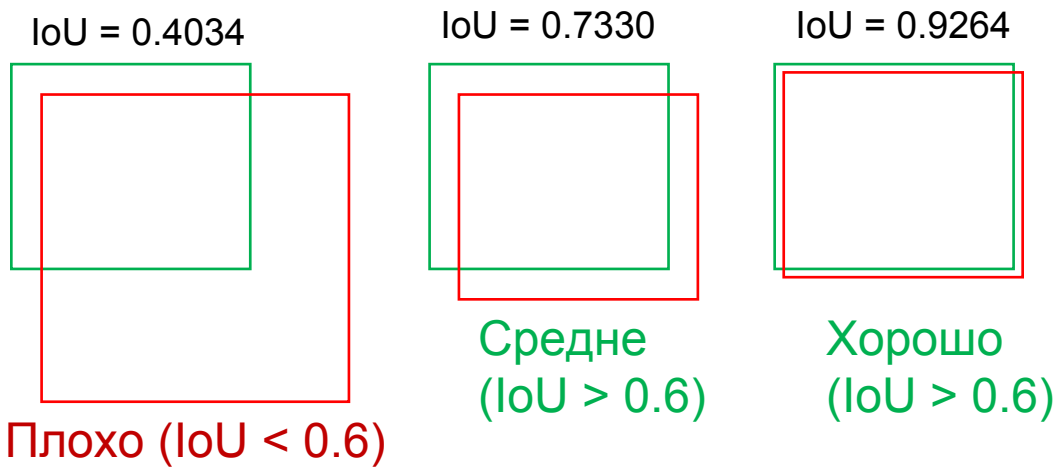
Semantic Segmentation

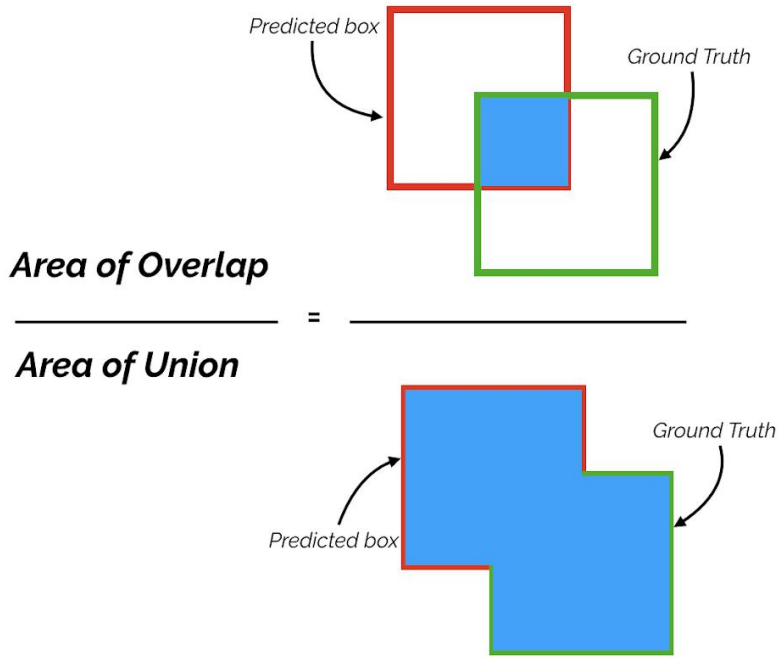


Instance Segmentation

Метрики оценки качества (IoU)

- вычисляем Intersection Over Union (IoU)
- задаем порог IoU threshold (например 0.6)



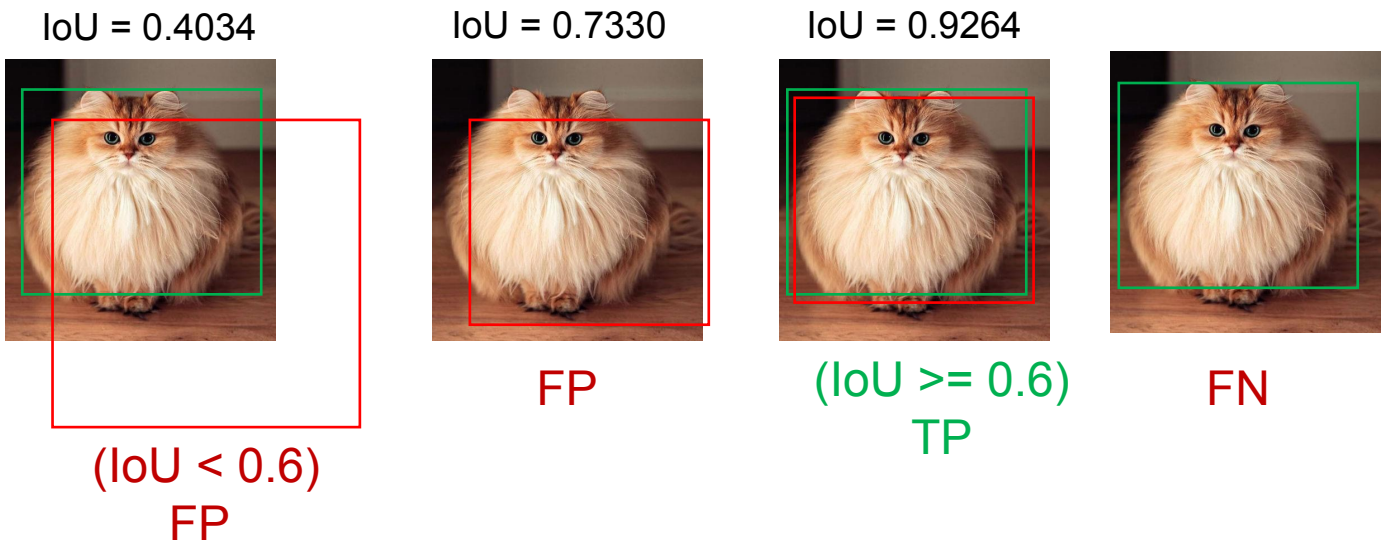


Метрики оценки качества (Prec-n, Recall)

- True Positive (TP) : IoU >= порога, нашли объект
- False Positive (FP) : IoU < порога, нашли объект там, где его нет
- False Negative (FN) : не смогли найти объект, там где он есть
- True Negative (TN) : не вычисляется

$$Precision = \frac{TP}{TP\ +\ FP} = \frac{TP}{all\_detections}$$

$$Recall = \frac{TP}{TP\ +\ FN} = \frac{TP}{all\_ground\_truths}$$



COCO (Common Objects in Context) dataset метрика:

mean Average Precision (AP @ [. 5: .95]) - (AP в COCO данных) это среднее значение для Precision для нескольких порогов IoU (от 0,5 до 0,95 с размером шага 0,05), расчитанное для каждой категории (классу).

Avarage Recall вычисляется аналогично

Pascal VOC метрика набора данных Pascal, основана только на пороге 0.5

Average Precision (AP):

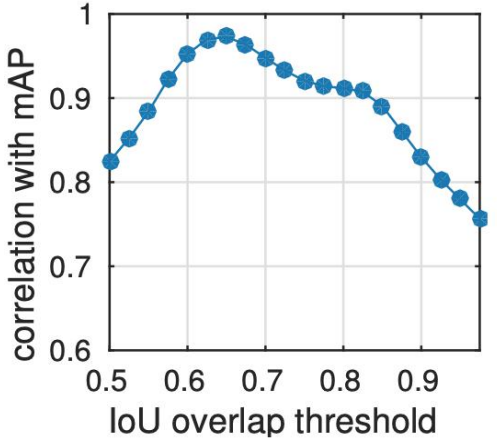
AP	% AP at IoU=.50:.05:.95 **(primary challenge metric)**
$AP^{IoU=.50}$	% AP at IoU=.50 (PASCAL VOC metric)
$AP^{IoU=.75}$	% AP at IoU=.75 (strict metric)

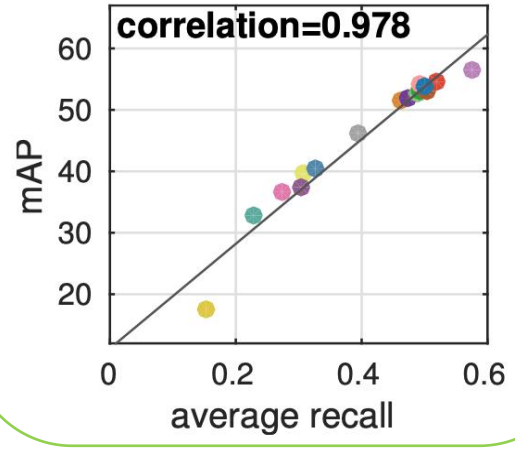
AP Across Scales:

AP^{small}	% AP for small objects: area < 32^2
AP^{medium}	% AP for medium objects: 32^2 < area < 96^2
AP^{large}	% AP for large objects: area > 96^2

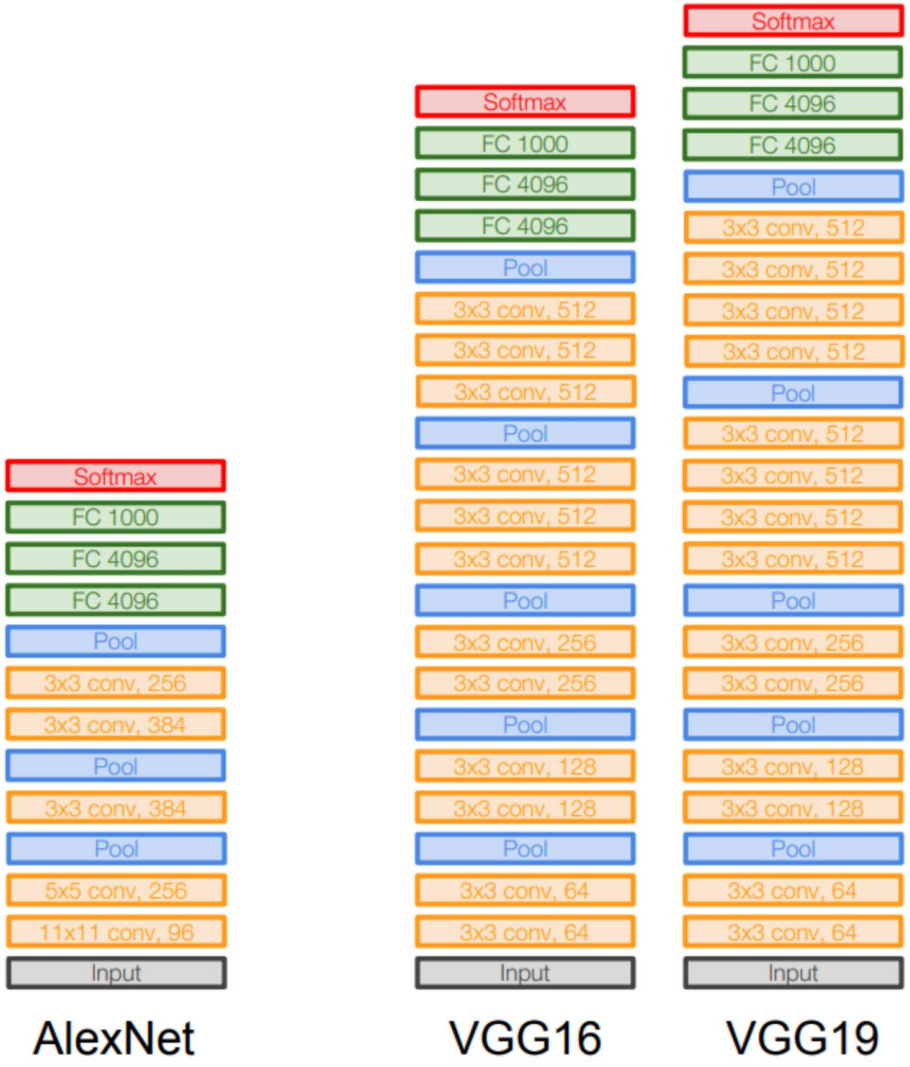
https://cocodataset.org/#detection-eval
https://arxiv.org/pdf/1502.05082.pdf

Замеры
на модели R-CNN:





VGG Net



- сеть глубже AlexNet
- использование только малых фильтров (3x3) свертки -> меньше обучаемых параметров в сети
- 2 фильтра (3x3) ≈ фильтр (5x5)
  3 фильтра (3x3) ≈ фильтр (7x7)
- есть проблема затухающих градиентов

Проблема глубины сети

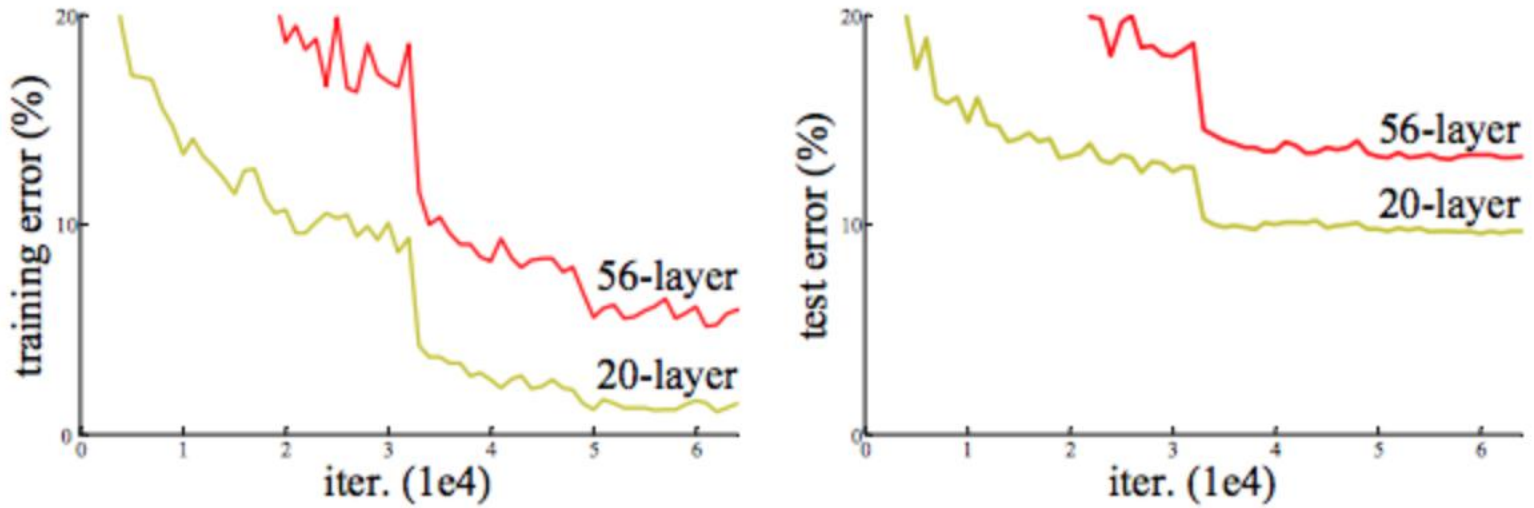
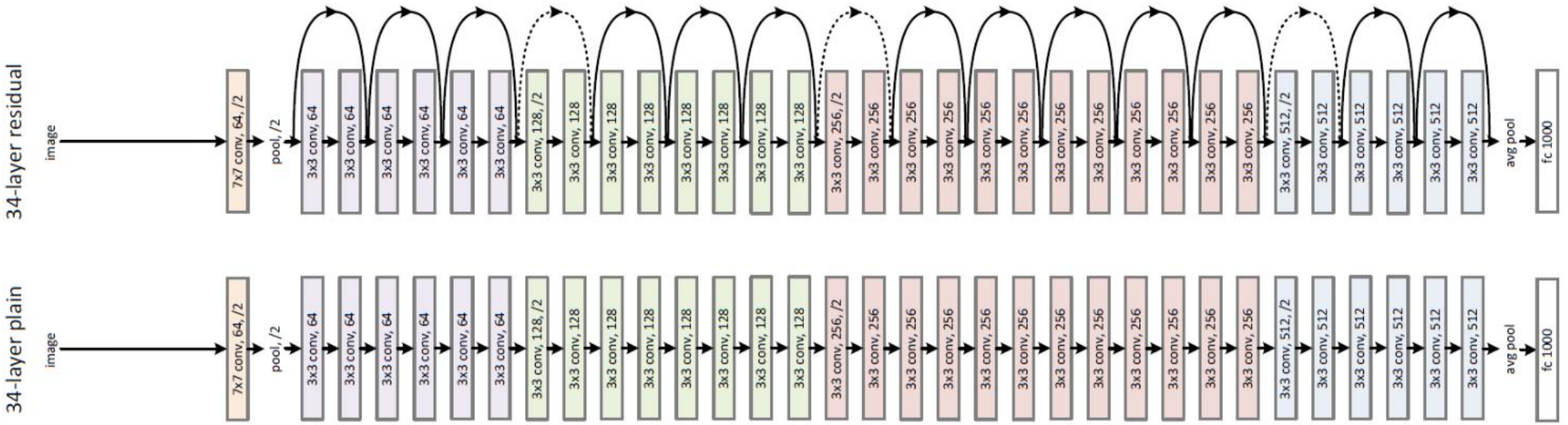


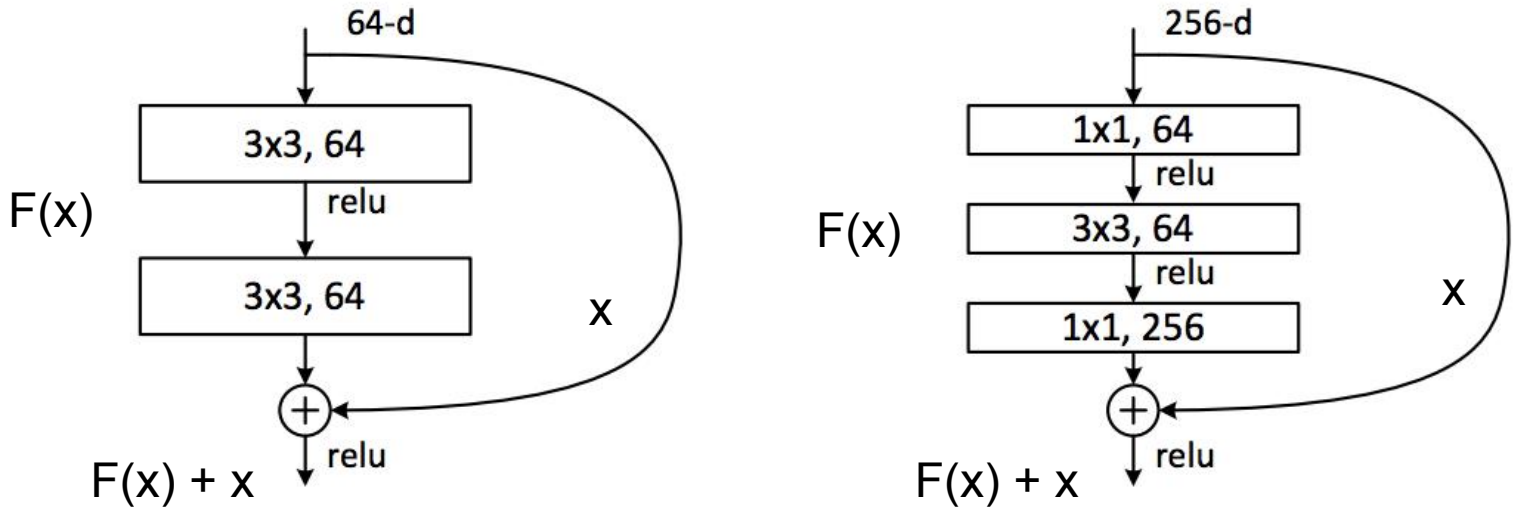
Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

ResNet (2015)

- ResNet-34 и простая 34-слойная сеть
- добалены Skip Connections передачи
- Пунктиром — это свертки 1 × 1, изменяющие количество каналов



Residual Block (Остаточный блок)



- F(x) и x должны иметь одинаковую размерность при операции сложении
- свертка 1x1 позволяет изменять кол-во фильтров

ResNeXt (2016)

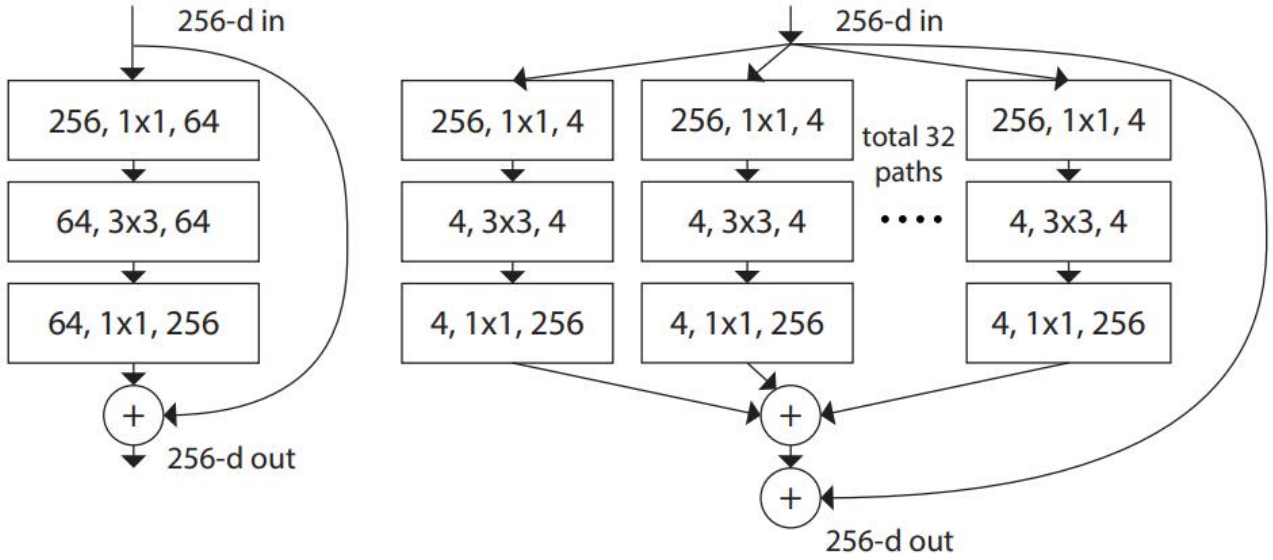


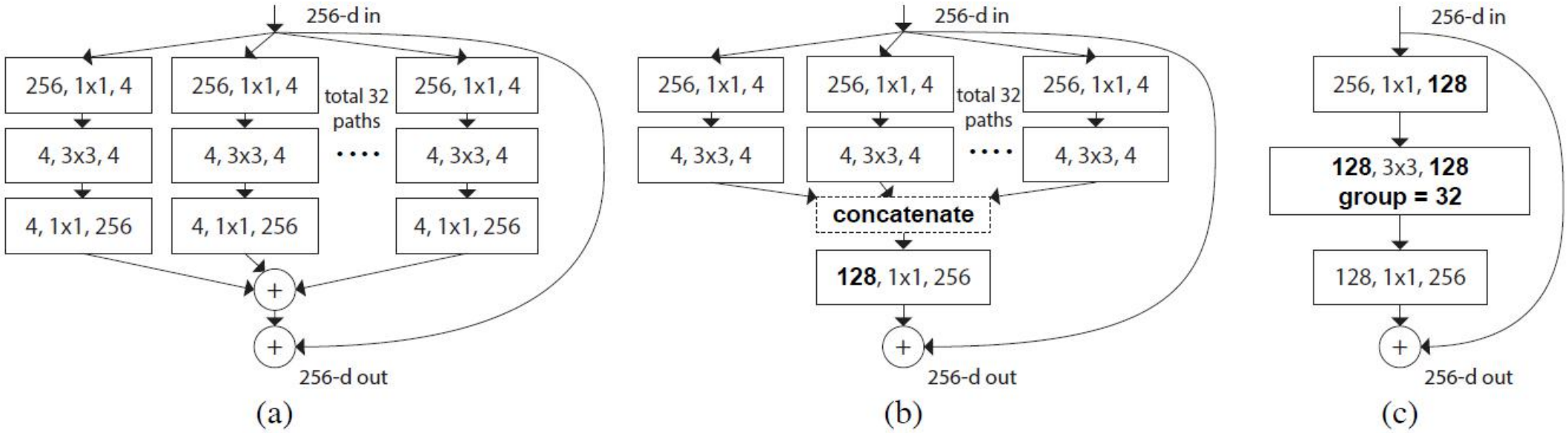
Figure 1. **Left**: A block of ResNet [14]. **Right**: A block of ResNeXt with cardinality = 32, with roughly the same complexity. A layer is shown as (# in channels, filter size, # out channels).

Идея: разделение на группы сверток

+Разделение на группы выступает в качестве регуляризации
+ Эффективность обучения (параллельное вычисление групп)

Количество параметров сверток:
256·64+3·3·64·64+64·256 VS 32·(256·4+3·3·4·4+4·256)
= 69632 = 70144

Эквивалентные представления ResneXt блока



C (cardinality) мощность или кол-во групп = 32
d (dimension) кол-во фильтров = 4
Сумма размерностей всех групп d * C = 4 * 32 = 128

*В блоке (c) используется идея “Grouped Convolution”

Grouped Convolution

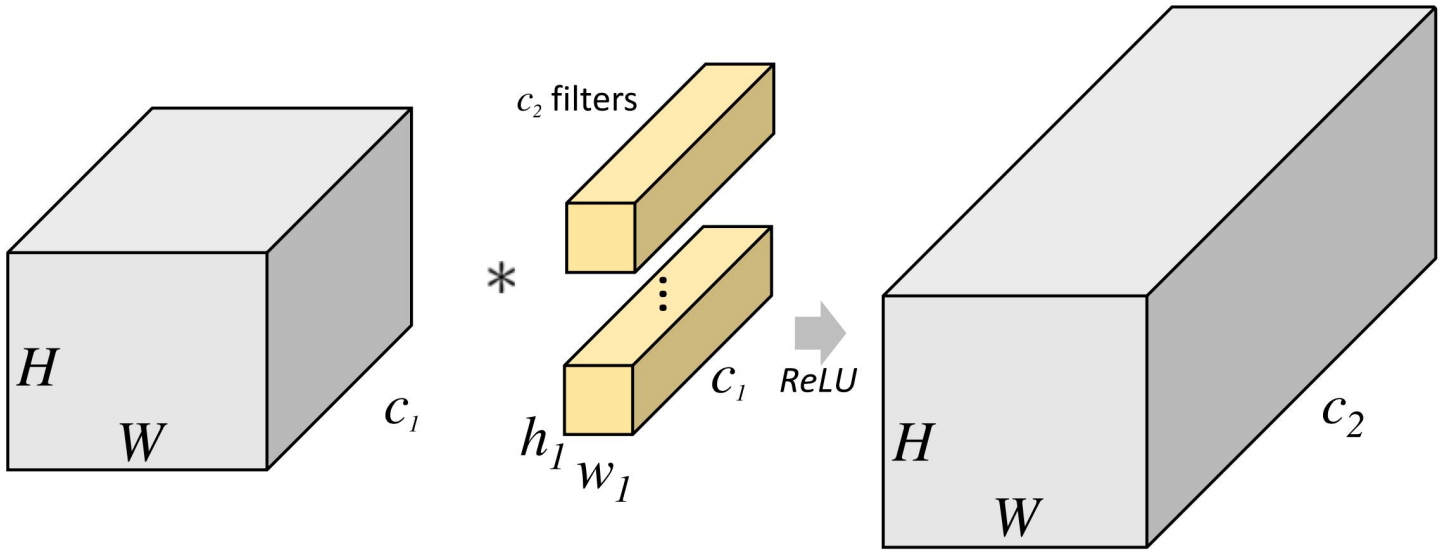


Рисунок 1 - Обычная операция свертки

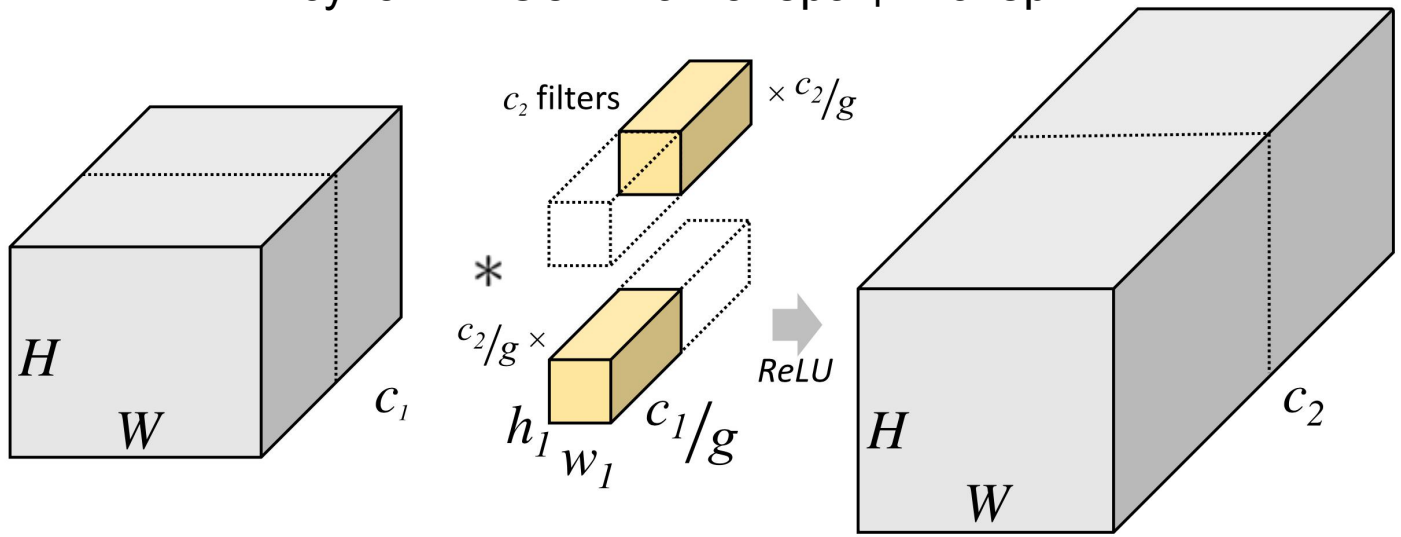


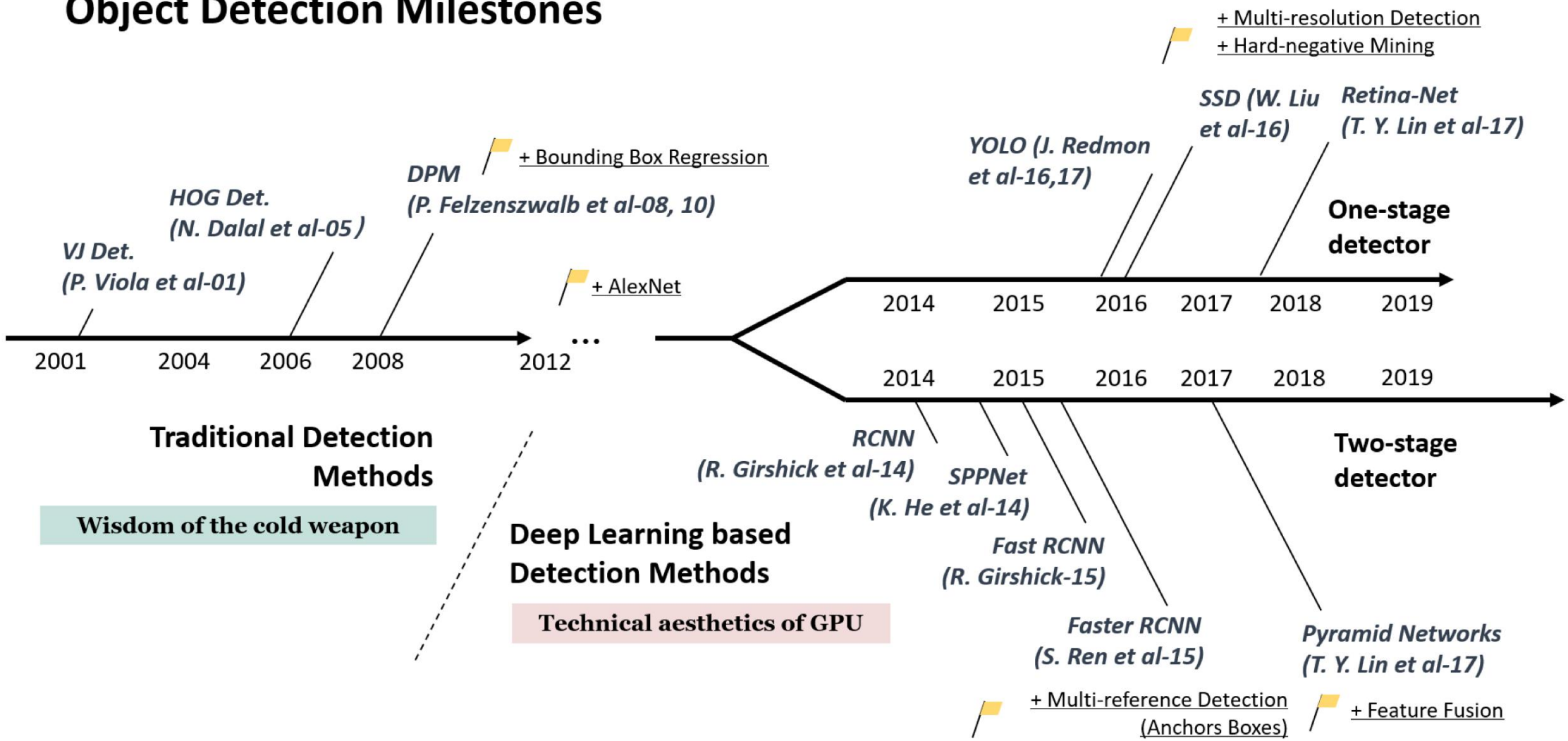
Рисунок 2 - Свертка двумя группами

Backbone и task-specific архитектуры

- SOTA (State of the art) архитектуры для задач компьютерного зрения часто можно разбить на **backbone**-часть (извлечение признаков) и **task-specific** части (например предсказание bounding box-ов, сегментирование объектов). Также выделяют **head**-архитектуры для улучшения распознавания признаков, полученных от **backbone**-части (например Feature Pyramid Network)
- Transfer Learning - можно обучать на больших доступных наборах данных (например ImageNet, COCO) с последующим дообучением на требуемые малые наборы данных

- Мы рассмотрели часть популярных backbone-архитектур
- Далее рассмотрим task-specific-архитектуры (детекторы, сегментаторы) и head-архитектуры

Object Detection Milestones



https://arxiv.org/pdf/1905.05055.pdf

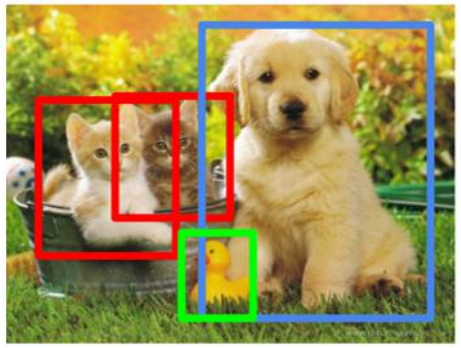
Region-based methods

Classification

CAT

?

Object Detection



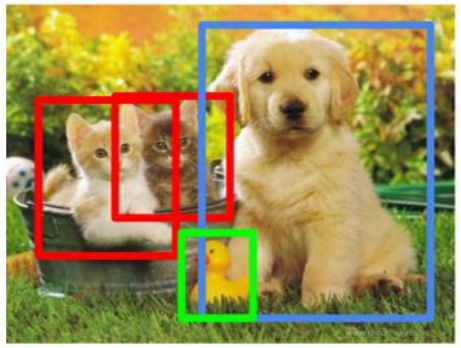
CAT, DOG, DUCK

Region-based methods

Classification

CAT

Object Detection



CAT, DOG, DUCK

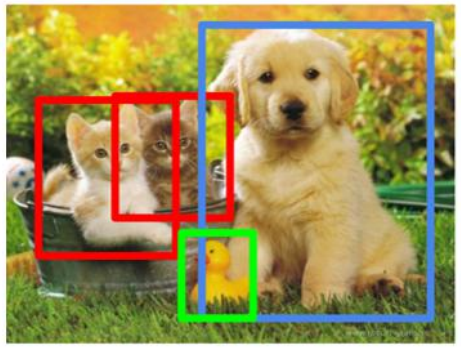
- Используется метод для извлечения регионов (частей изображения) с помощью окон

Region-based methods

Classification

CAT

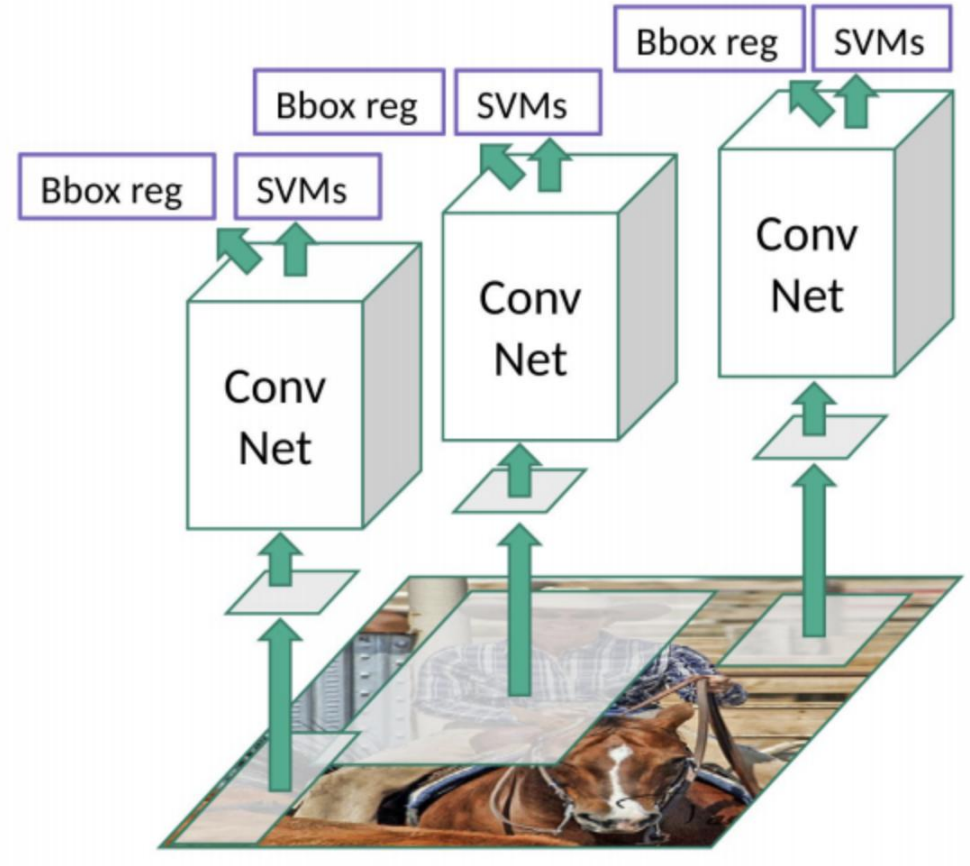
Object Detection



CAT, DOG, DUCK

- Используется метод для извлечения регионов (частей изображения) с помощью окон
- Нужно использовать окна разных размеров и соотношение сторон для извлечения разного размера объектов

RCNN

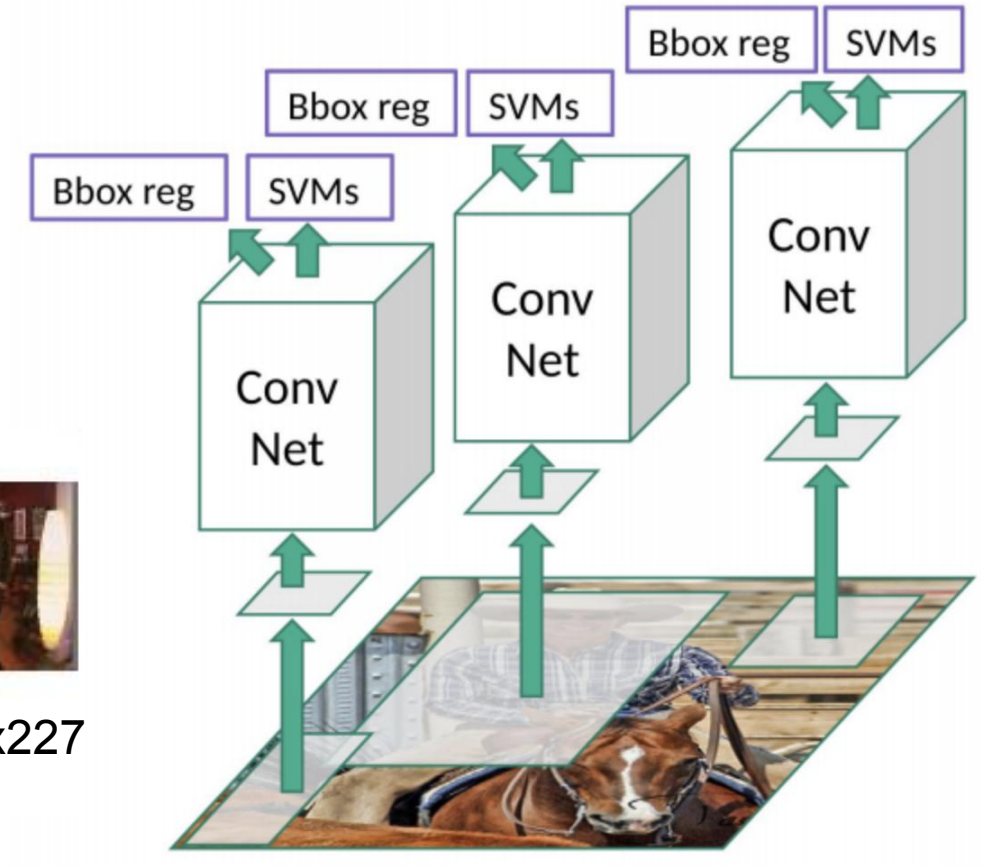


Проблемы ?

RCNN

227x227

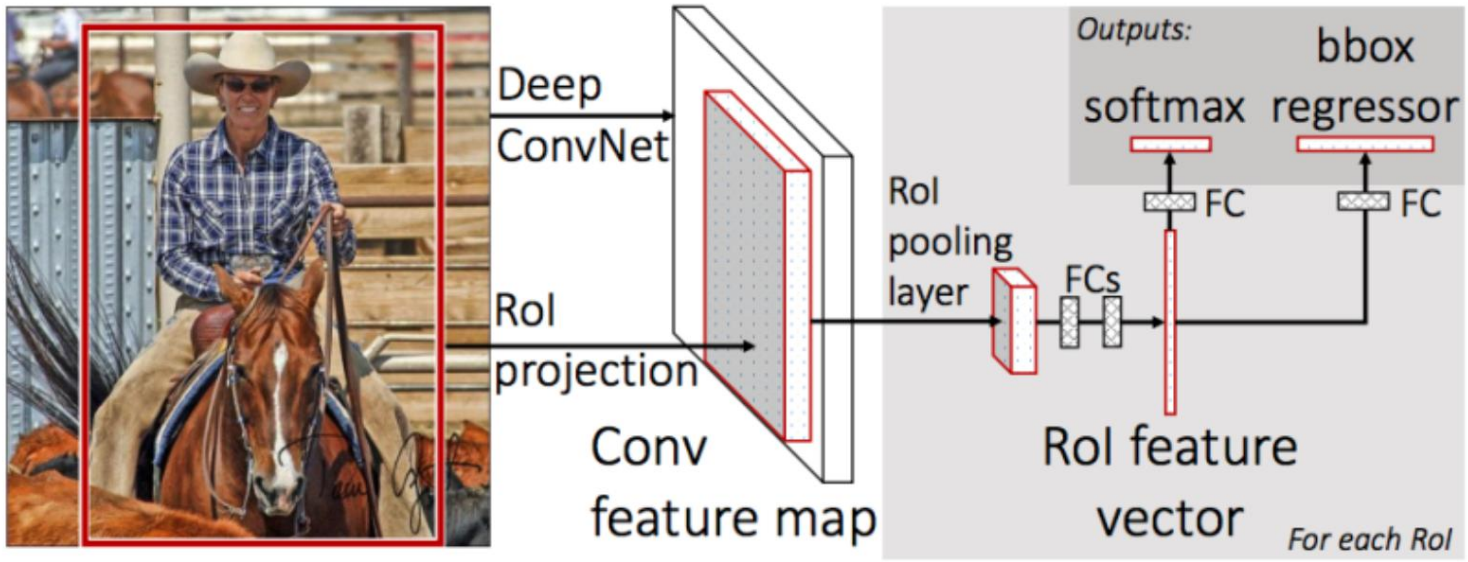
Исходный регион



Проблемы ?

- Долгое обучение на случайных регионах с прогоном через всю CNN для извлечения признаков
- CNN использовала на вход фиксированный размер 227x227 регионов. --> Требовалось брать наименьший охватывающий квадрат региона, дополнять нулями
- Эвристический алгоритм Selective Search не обучается и стабильно ошибается
- Обработка одного изображения около 47 секунд :(

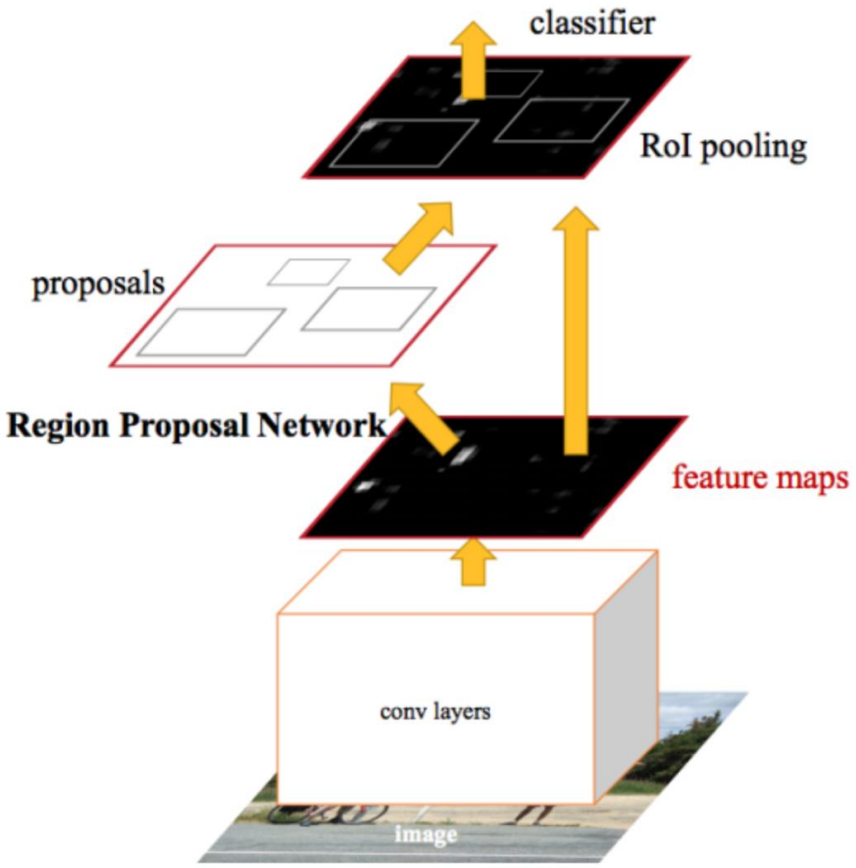
Fast RCNN



$$L(p, u, t^u, v) = L_{cls}(p, u) + \lambda[u \geq 1]L_{loc}(t^u, v)$$

- Прогон всего изображения через CNN;
- Вместо независимого обучения трёх моделей (CNN, SVM, bbox regressor) совместить все процедуры тренировки в одну end-to-end;
- Использование RoIPooling слоя. Окно региона шириной w и высотой h делилось на сетку, имеющую H×W ячеек размером h/H × w/W. (Авторы использовали W=H=7).

Faster RCNN



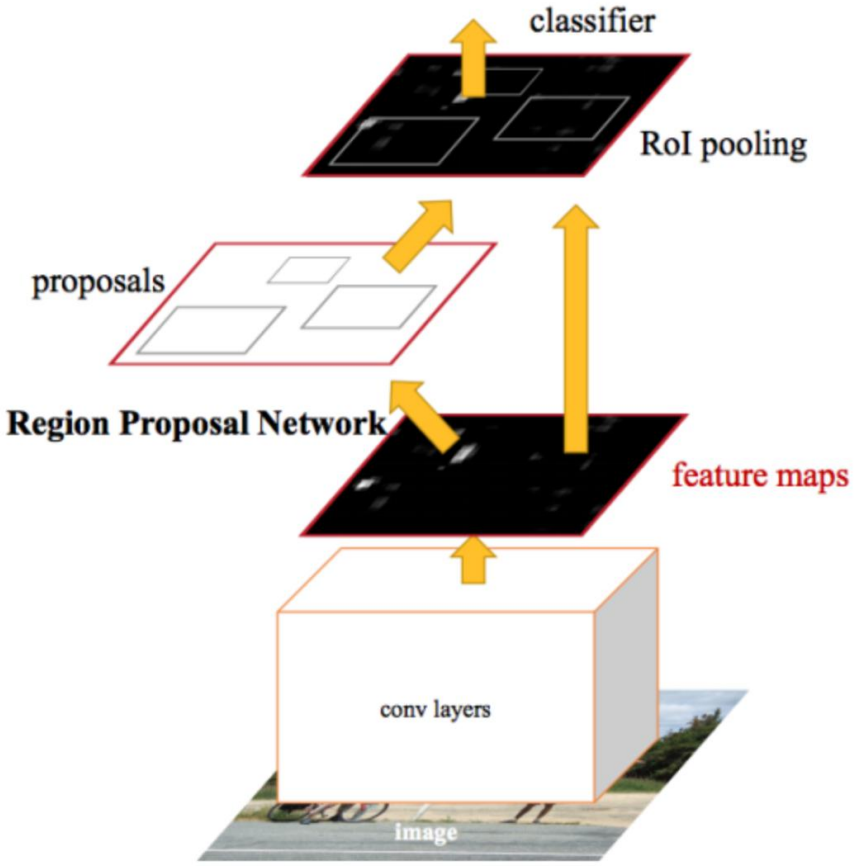
Выделяют 2 части: в two-stage detector:

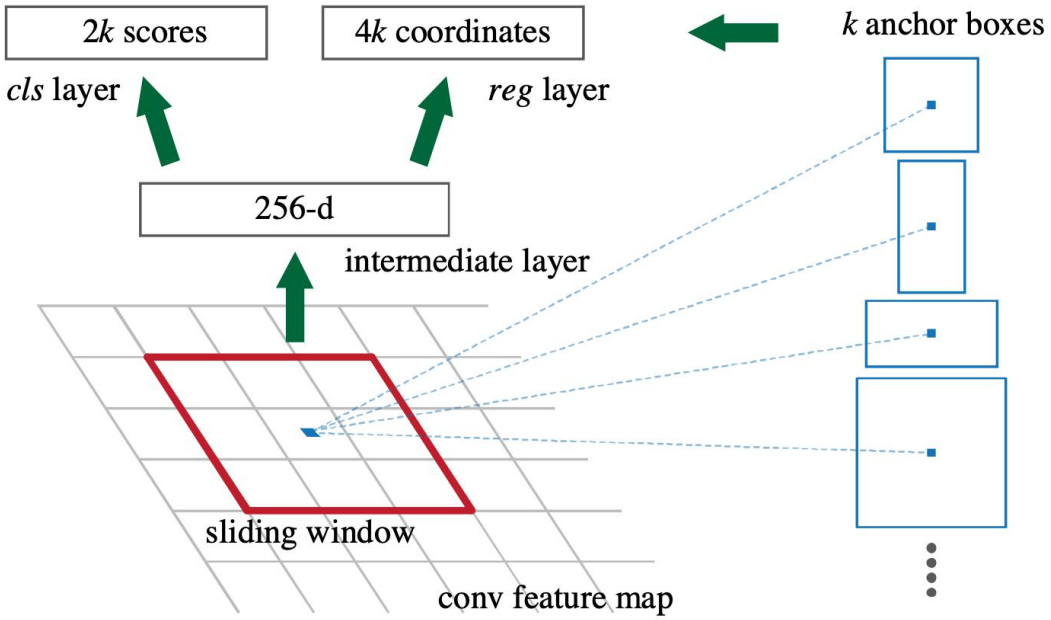
* - backbone - генерирует высокоуровневые признаки на изображении (ResNet, VGG, MobileNet и т.д.)

1 - генерация регионов интересов RoI (Region Of Interest)

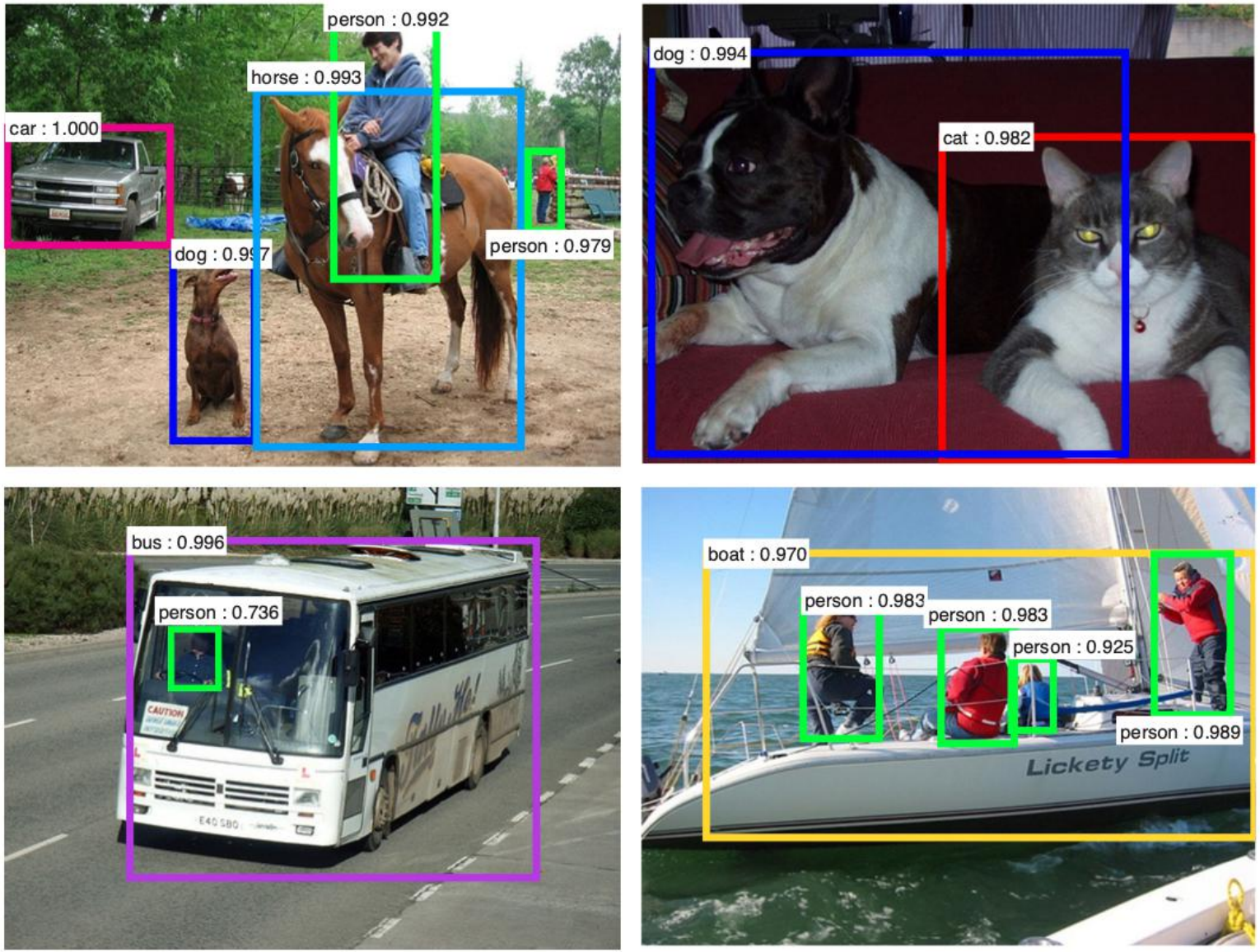
2 - прогон через слой RoI Pooling, классификация региона и регрессия bounding box

Faster RCNN

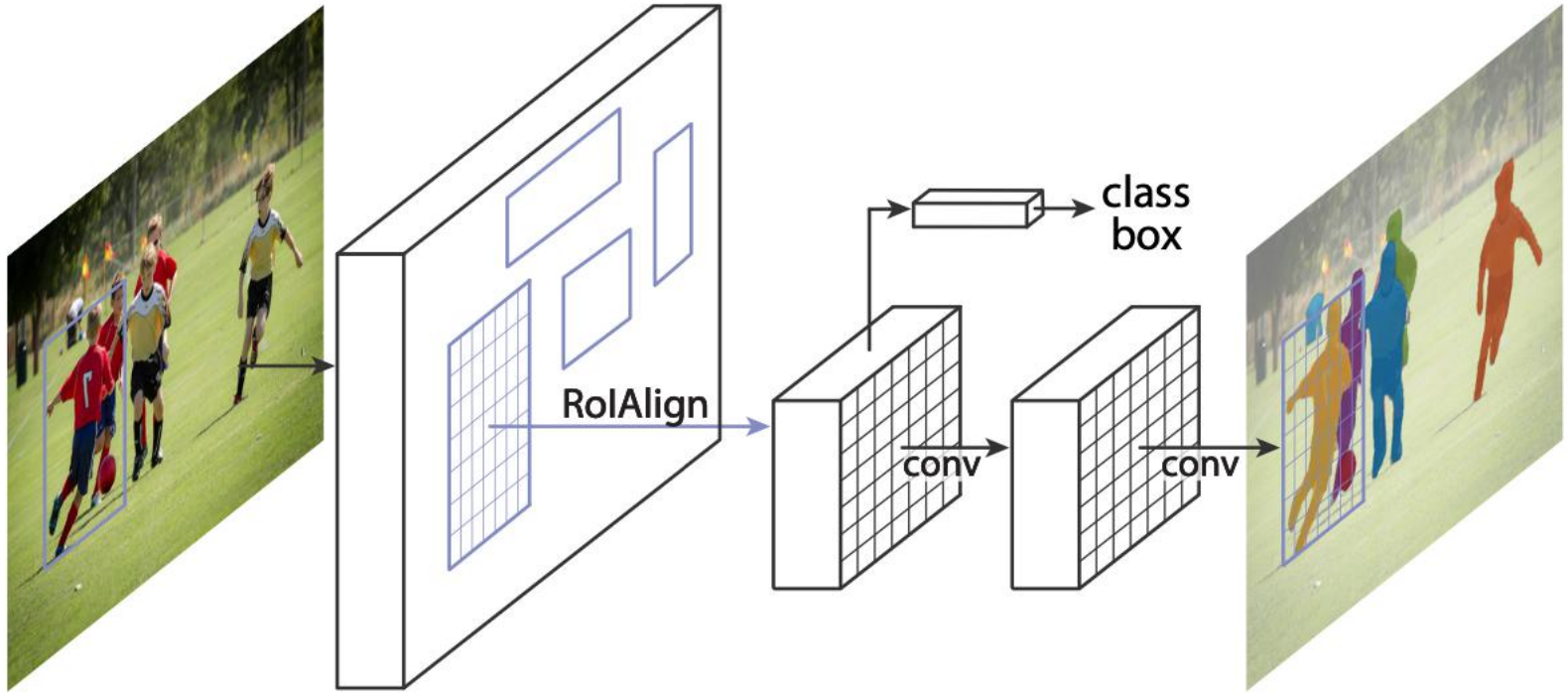

classifier
RoI pooling
proposals
Region Proposal Network
feature maps
conv layers
image


2k scores
4k coordinates
k anchor boxes
cls layer
reg layer
256-d
intermediate layer
sliding window
conv feature map

Faster RCNN


person : 0.992
horse : 0.993
car : 1.000
dog : 0.9
person : 0.979
dog : 0.994
cat : 0.982
bus : 0.996
person : 0.736
boat : 0.970
person : 0.983
person : 0.983
person : 0.925
person : 0.989
Lickety Split

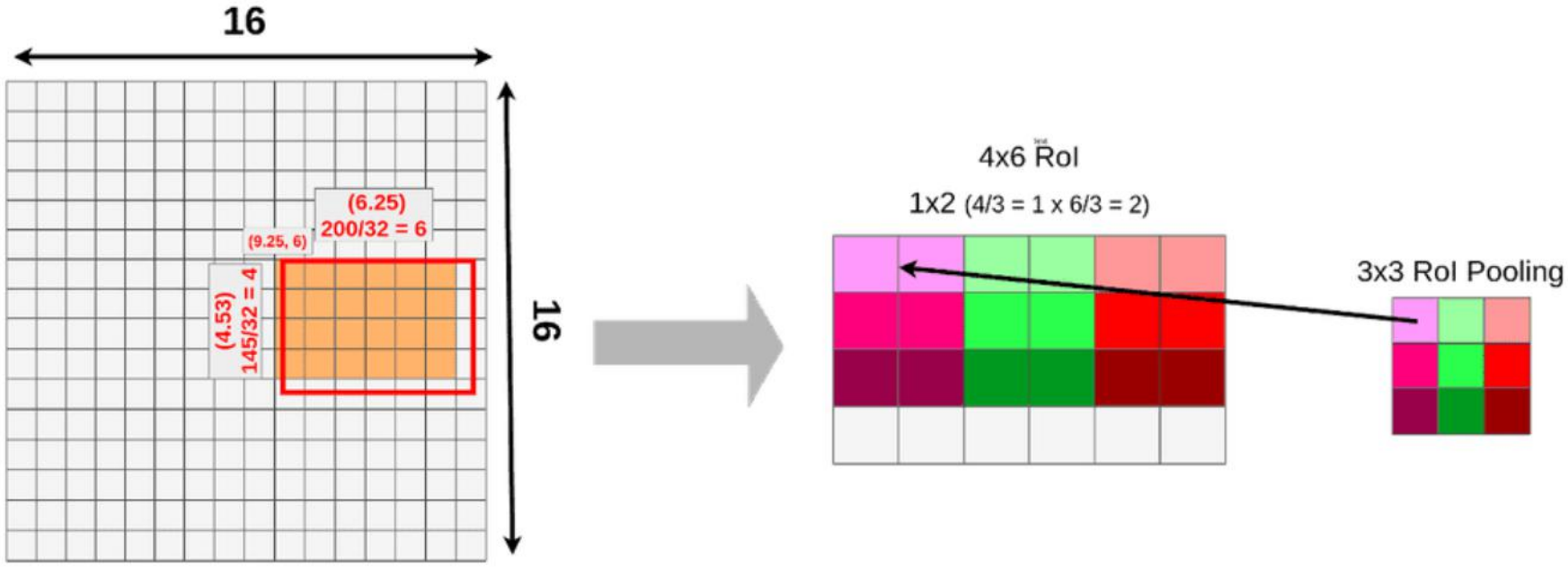
Mask RCNN



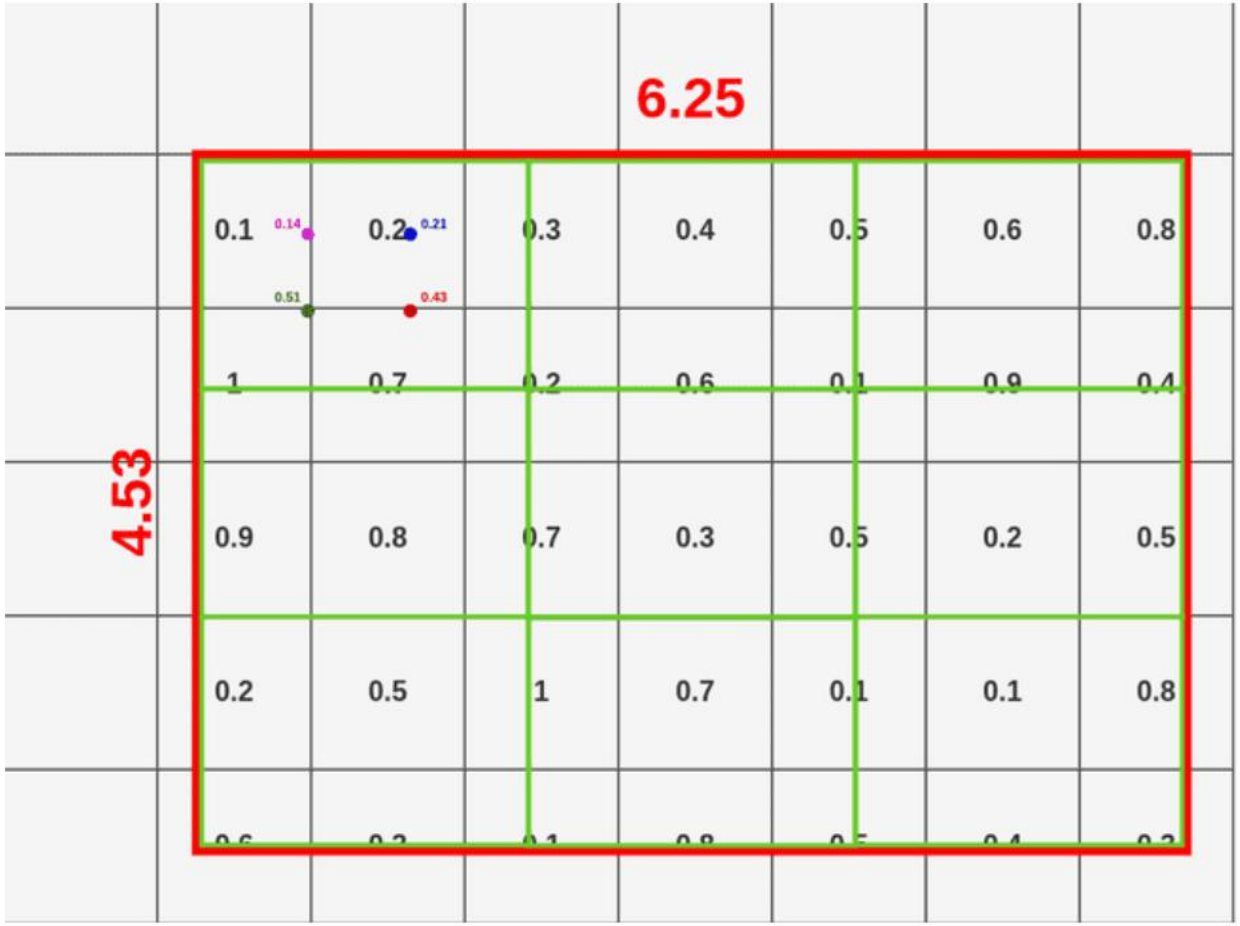
$$L = L_{cls} + L_{box} + L_{mask}.$$

- Использование доп. ветви для решения задачи instance segmentation
- Маска каждого класса определяется независимо от других классов;
- RoIAlign вместо RoIPooling

RoI Pooling


16
16
(6.25)
200/32 = 6
(9.25, 6)
(4.53)
145/32 = 4
4x6 RoI
1x2 (4/3 = 1 x 6/3 = 2)
3x3 RoI Pooling

RoI Align



1x1 = MAX(0.14, 0.21, 0.51, 0.43) = 0.51

3x3 RoIAlign

0.51		

$$P \approx \frac{y_2 - y}{y_2 - y_1}\left(\frac{x_2 - x}{x_2 - x_1}Q_{11} + \frac{x - x_1}{x_2 - x_1}Q_{21}\right) + \frac{y - y_1}{y_2 - y_1}\left(\frac{x_2 - x}{x_2 - x_1}Q_{12} + \frac{x - x_1}{x_2 - x_1}Q_{22}\right)$$

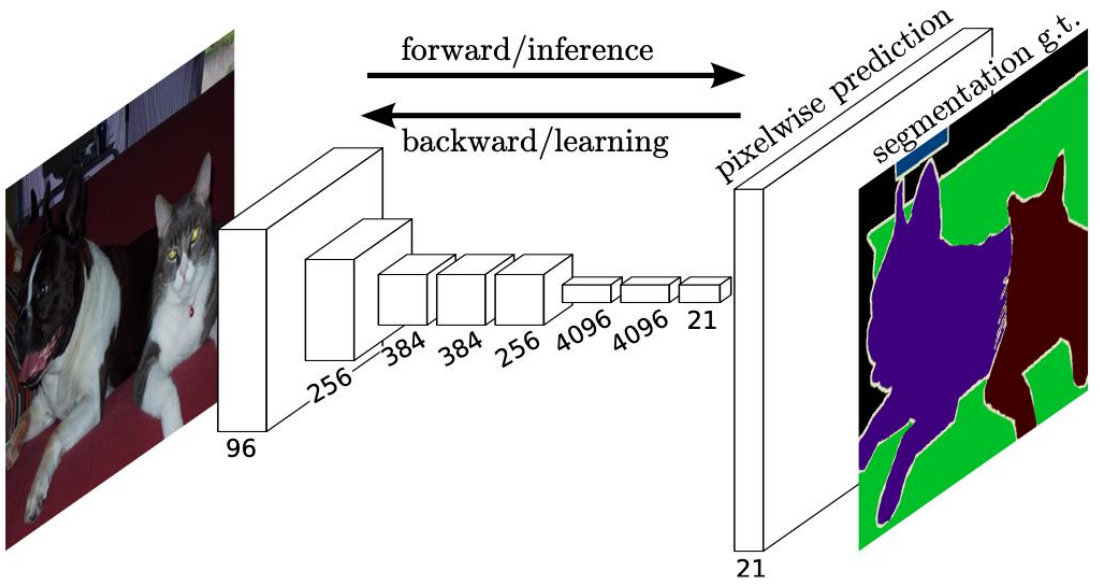
	AP	AP_{50}	AP_{75}	AP^{bb}	AP^{bb}_{50}	AP^{bb}_{75}
RoIPool	23.6	46.5	21.6	28.2	52.7	26.9
RoIAlign	30.9	51.8	32.1	34.0	55.3	36.4
	+7.3	+5.3	+10.5	+5.8	+2.6	+9.5

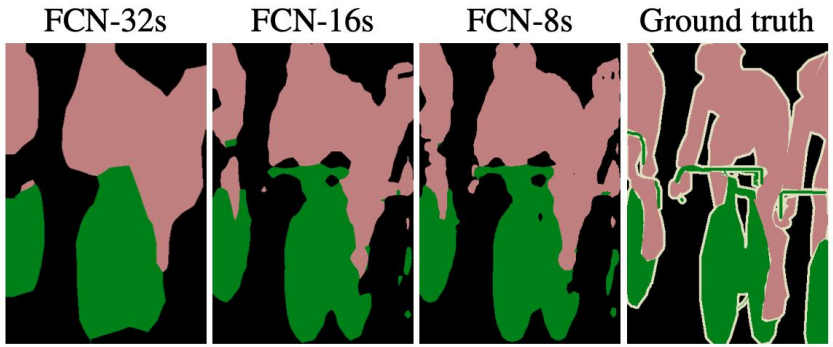
APbb это average precision для определения bounding boxes.
Тестирование проводилось на ResNet-50-C5 со stride=32.

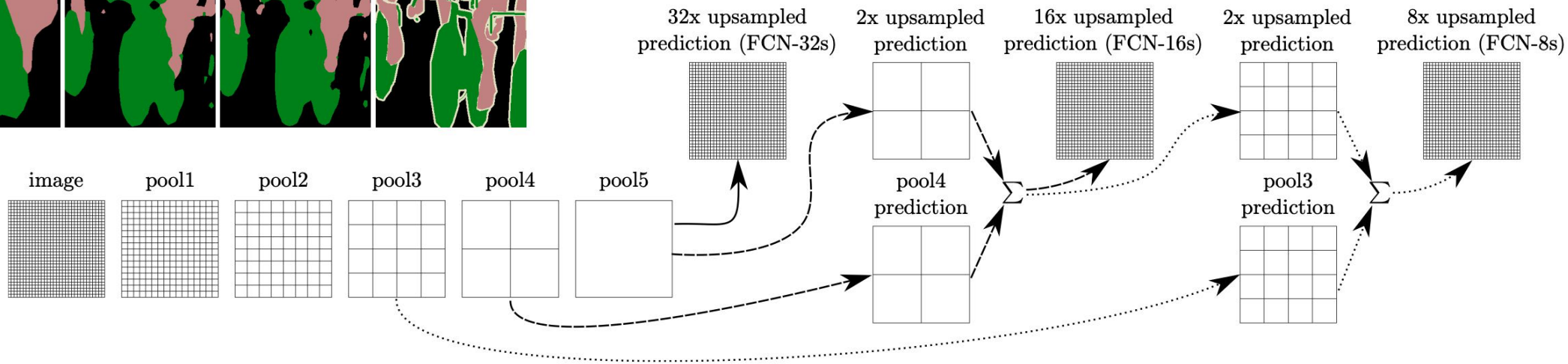
Mask RCNN

Fully Convolutional Network (FCN)

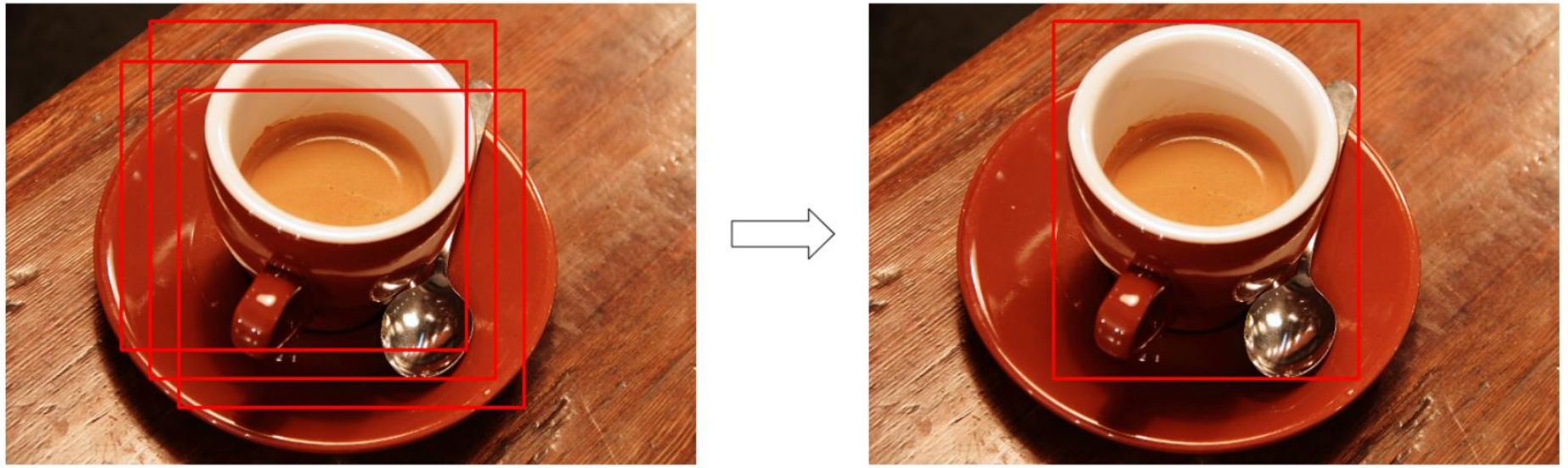
- решение задачи Semantic Segmentation
- отстутствие FullyConnected слоев
- использование Deconvolution слоев (backwards convolution) - upsample layers
- использование image различных размеров







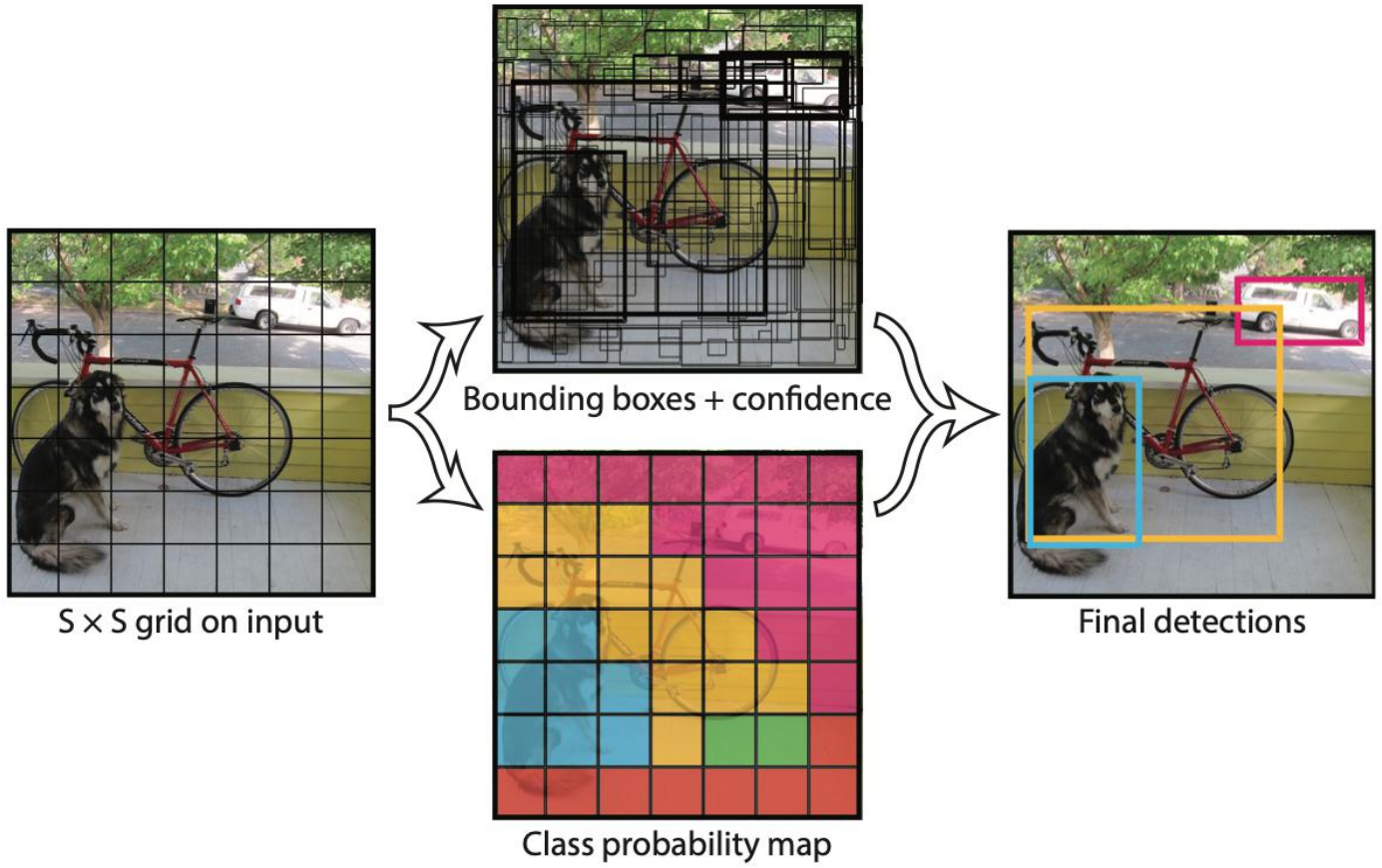
Non-Maximum-Suppression (NMS)



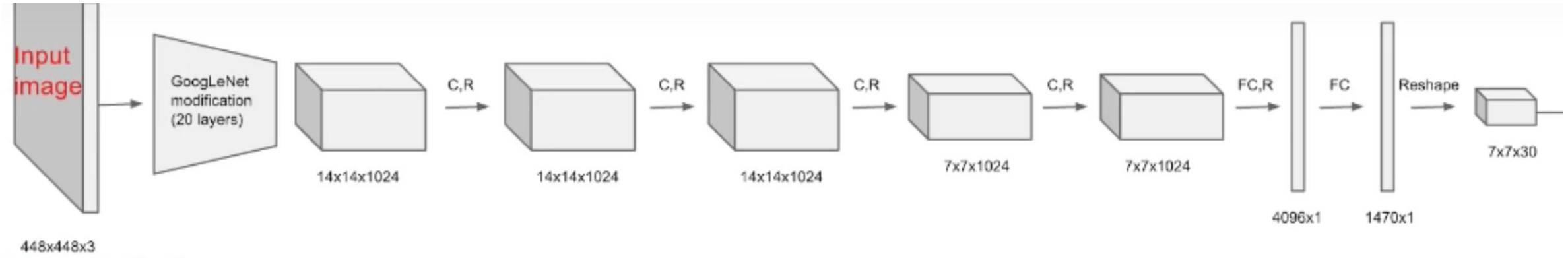
1. Гипотезы сортируются по их «уверенности».
2. В цикле выбирается первая гипотеза A (имеет наибольшую величину уверенности) и добавляется в результирующий набор.
3. В цикле выбирается следующая B.
4. Если между выбранными гипотезами IoU(A, B) > threshold, то вторая гипотеза B отбрасывается и далее не присутствует в результирующем наборе.

Все повторяется, начиная с шага 2 до момента полного перебора гипотез.

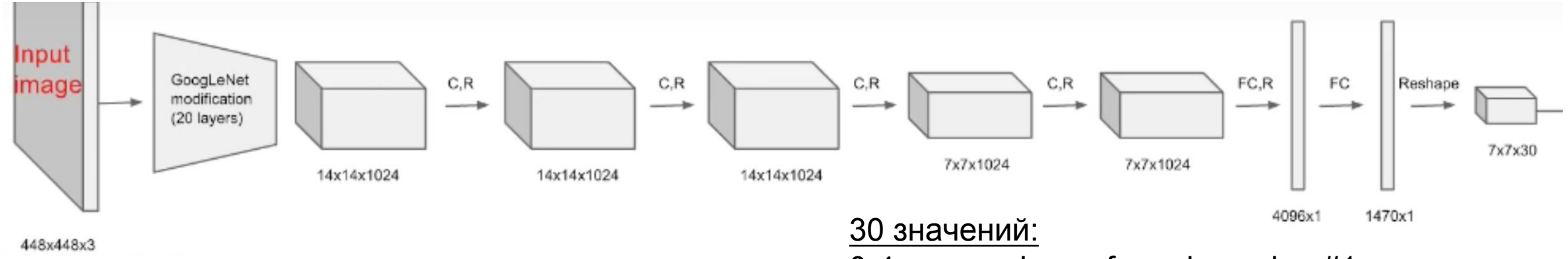
YOLO (You Only Look Once)

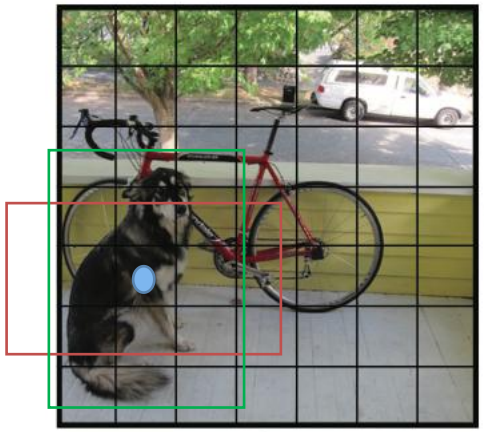


YOLO


Input image
448x448x3
GoogLeNet modification (20 layers)
14x14x1024
C,R
14x14x1024
C,R
14x14x1024
C,R
7x7x1024
C,R
7x7x1024
FC,R
4096x1
FC
1470x1
Reshape
7x7x30

YOLO





S × S grid on input

S = 7

7x7x30

30 значений:

0-4: x, y, w, h, conf для box с box#1

5-8: x, y, w, h, conf для box с box#2

9-29: вероятность каждого из 20 классов

Conf_box_class = conf_box * conf_class

Total: 7x7x2 = 98 boxes

$$\Pr(\text{Class}_i|\text{Object}) * \Pr(\text{Object}) * \text{IOU}^{\text{truth}}_{\text{pred}} = \Pr(\text{Class}_i) * \text{IOU}^{\text{truth}}_{\text{pred}}$$

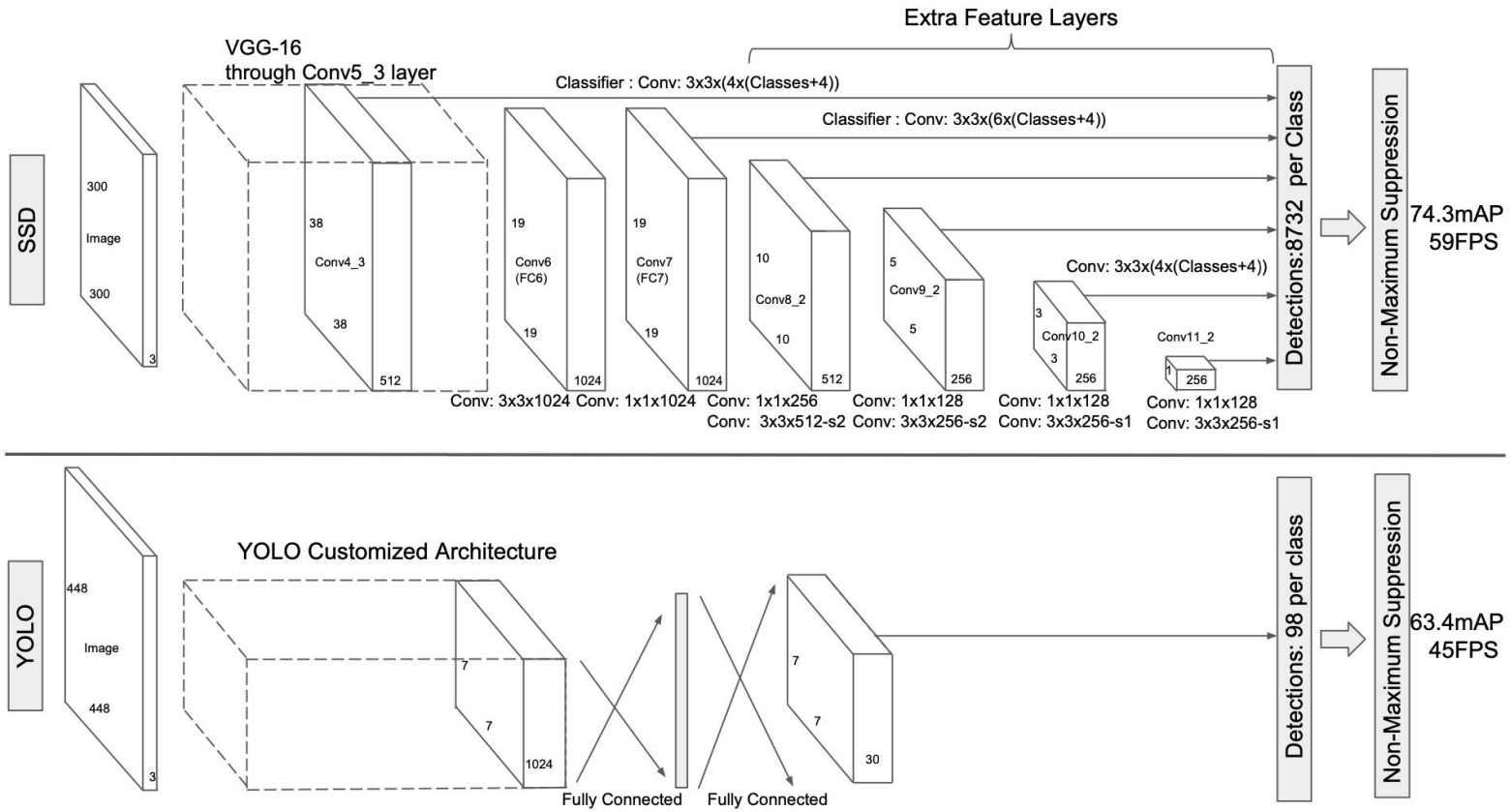
YOLO

Особенности:

- применение NMS (Non Maximum Suppression)
- 45 fps, YoloTiny - 155 fps
- end-to-end training
- меньше false positive срабатываний (детектировали объект, но его там нет)
- ограничение количества объектов на ячейку (и целое на изображение)

SSD - Single Shot Multibox Detector

- 59 FPS на 300x300 входе
- Использование NMS
- MultiScale feature map



Для каждой выходной карты (их несколько):
кол-во_ячеек_в_карте * колчество default_anchor_boxes * (4 + количество_классов)

Сравнение архитектур

Method	mAP	FPS	batch size	# Boxes	Input resolution
Faster R-CNN (VGG16)	73.2	7	1	~ 6000	~ 1000 × 600
Fast YOLO	52.7	155	1	98	448 × 448
YOLO (VGG16)	66.4	21	1	98	448 × 448
SSD300	74.3	46	1	8732	300 × 300
SSD512	76.8	19	1	24564	512 × 512
SSD300	74.3	59	8	8732	300 × 300
SSD512	76.8	22	8	24564	512 × 512

Источник SSD. PASCAL VOC 2007, 2012 и MS COCO на 300 × 300 и 512 × 512 input images.

Detection Frameworks	Train	mAP	FPS
Fast R-CNN [5]	2007+2012	70.0	0.5
Faster R-CNN VGG-16[15]	2007+2012	73.2	7
Faster R-CNN ResNet[6]	2007+2012	76.4	5
YOLO [14]	2007+2012	63.4	45
SSD300 [11]	2007+2012	74.3	46
SSD500 [11]	2007+2012	76.8	19
YOLOv2 288 × 288	2007+2012	69.0	91
YOLOv2 352 × 352	2007+2012	73.7	81
YOLOv2 416 × 416	2007+2012	76.8	67
YOLOv2 480 × 480	2007+2012	77.8	59
YOLOv2 544 × 544	2007+2012	**78.6**	40

Источник Yolov2. PASCAL VOC 2007 test set.

YOLO v2 (YOLO9000)

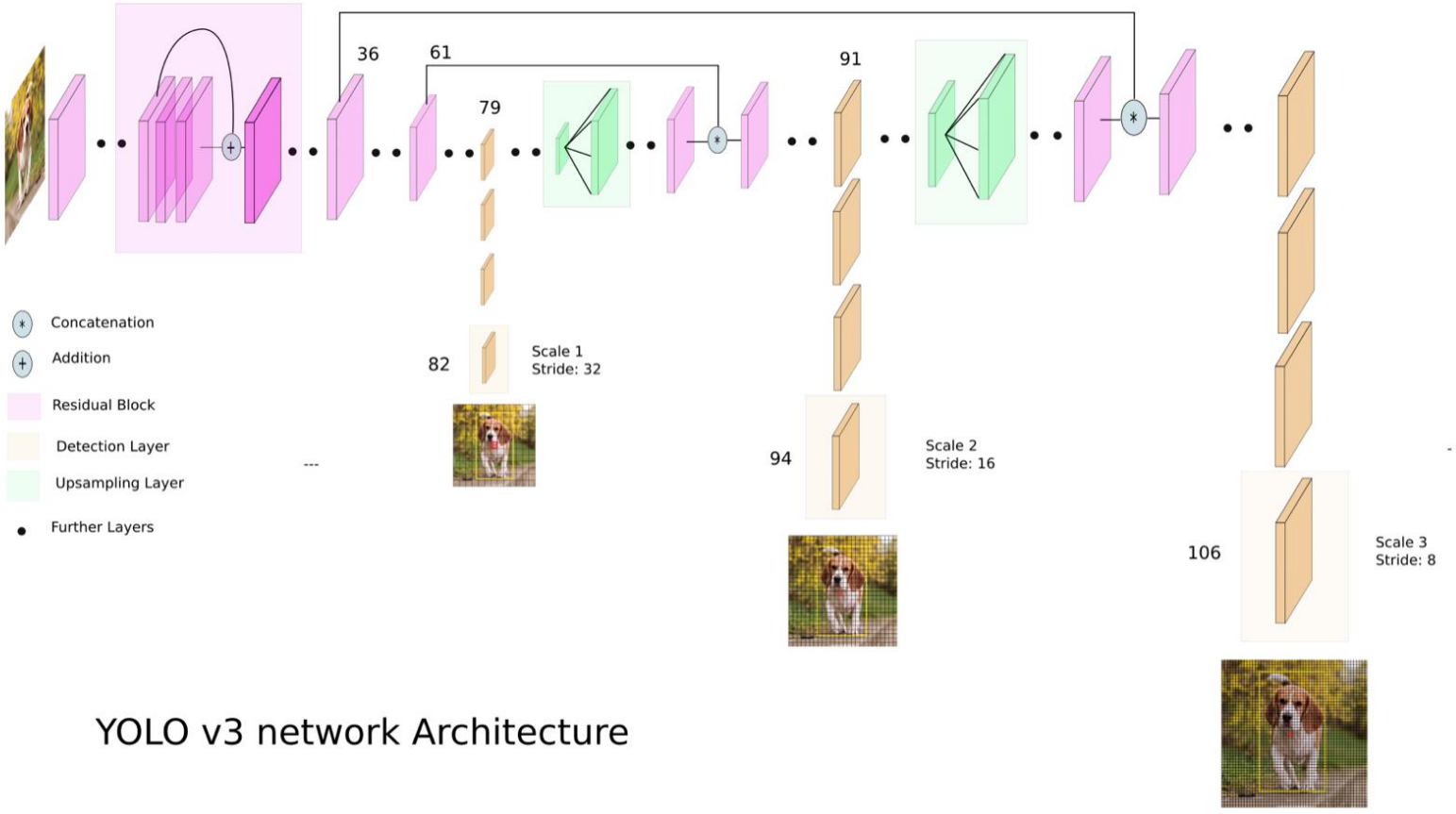
Много маленьких улучшений в сети:

- добавлена batch normalization
- тренировались на увеличенном разрешении 2 раза изобр-ях ImageNet 448x448
- задание anchor boxes -> предсказние больше 2-х объектов в ячейке сетки (default == 5)
- изменение лоса вычисления бокса (относительно начало сетки)
- определяет более мелкие объекты с помощью дополнительной сетки 26x26. Итоговая конкатинация сетки 26x26 с 13x13
- добавили обработку изображений разных размеров (MultuScale)
- 9000 категорий объектов
- от 40 до 67 FPS

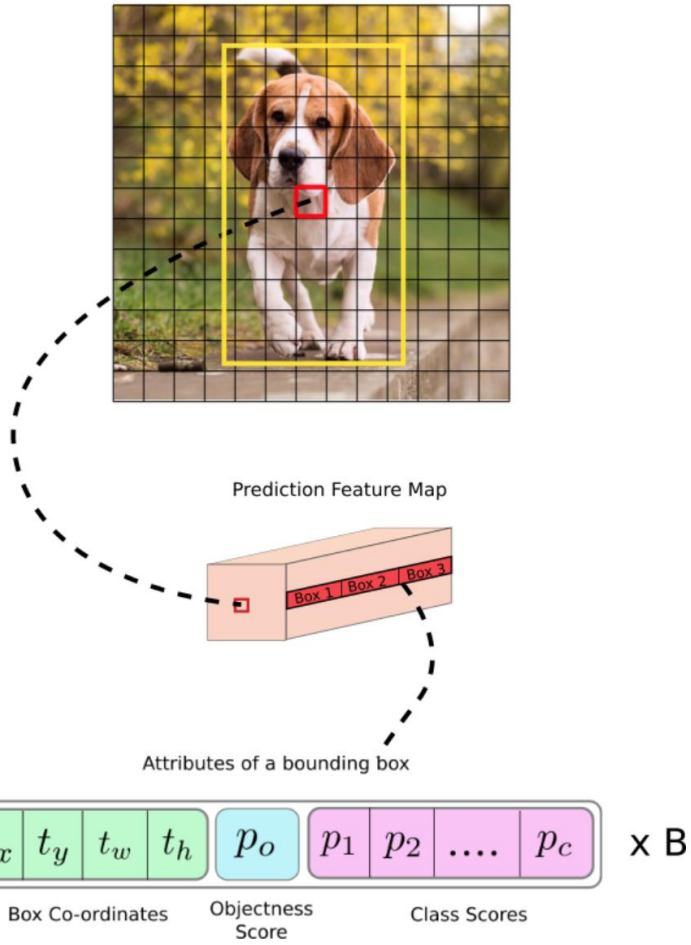
	YOLO								YOLOv2
batch norm?		✓	✓	✓	✓	✓	✓	✓	✓
hi-res classifier?			✓	✓	✓	✓	✓	✓	✓
convolutional?				✓	✓	✓	✓	✓	✓
anchor boxes?				✓	✓				
new network?					✓	✓	✓	✓	✓
dimension priors?						✓	✓	✓	✓
location prediction?						✓	✓	✓	✓
passthrough?							✓	✓	✓
multi-scale?								✓	✓
hi-res detector?									✓
VOC2007 mAP	63.4	65.8	69.5	69.2	69.6	74.4	75.4	76.8	**78.6**

https://arxiv.org/pdf/1612.08242.pdf

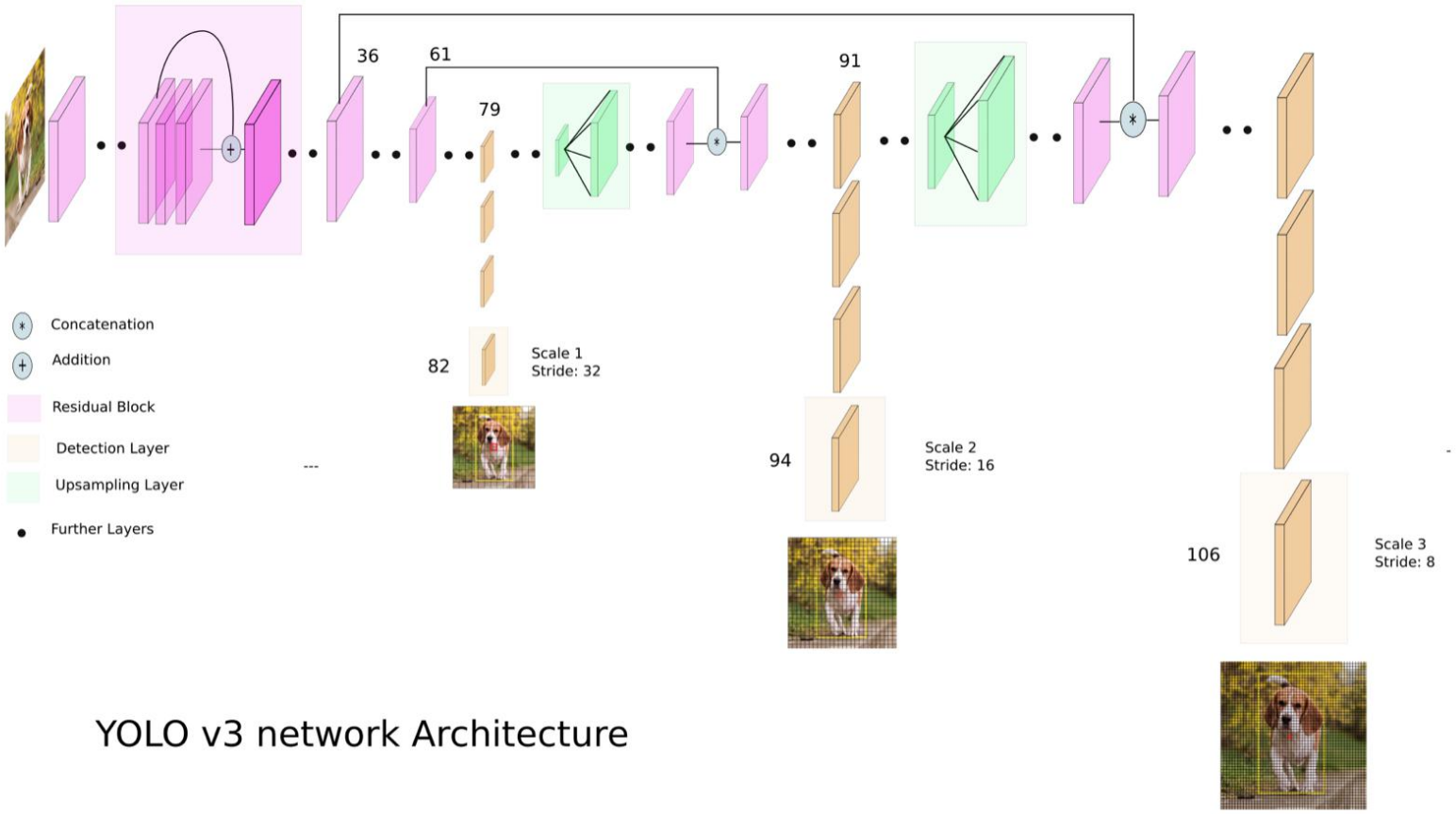
YOLO v3



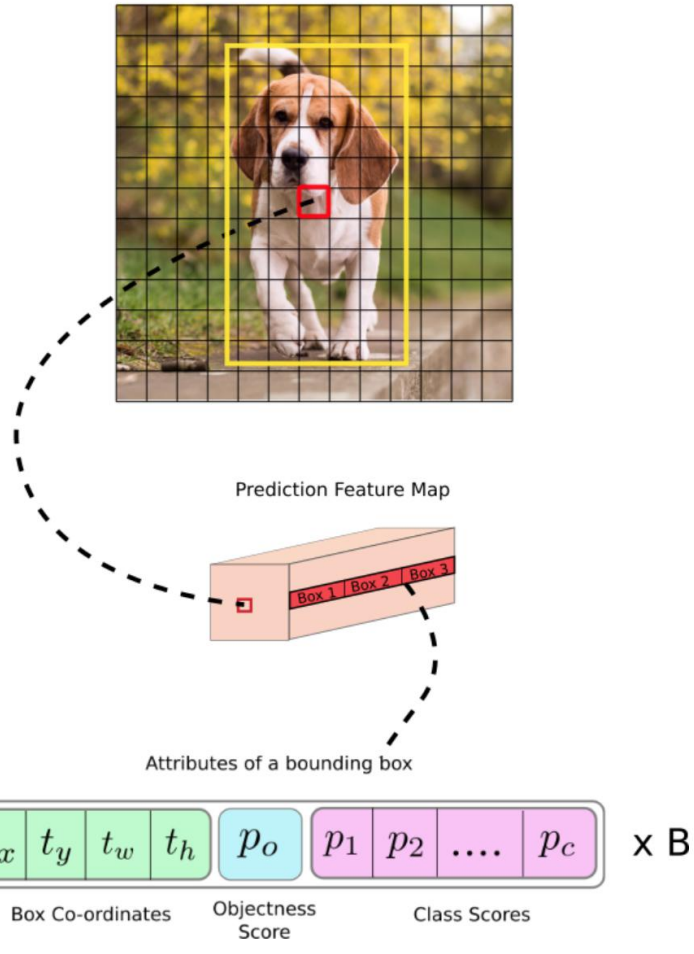
YOLO v3 network Architecture



YOLO v3



YOLO v3 network Architecture



На выходе тензор: 13x13 x (B x (5+C)), где
13x13 - это сетка на изображении,
B - количество боксов в одной ячейке сетки;
5 - это значения (tx, ty, tw, th, p0) бокса
C - количество предсказанных классов

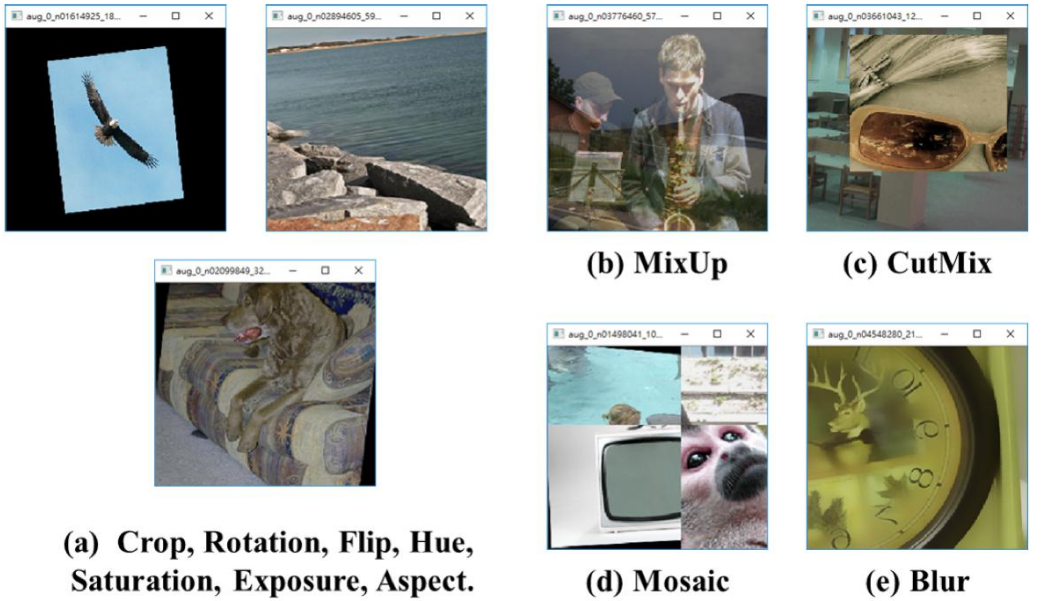
Total:
((52 x 52) + (26 x 26) + (13 x 13)) x 3 = 10647 bboxes
Потом фильтруем боксы:
по порогу и NMS

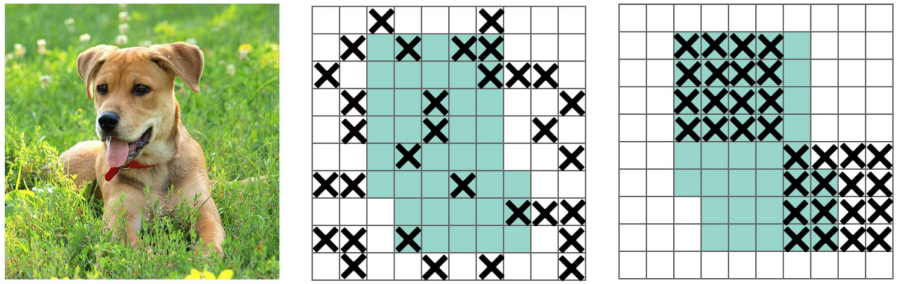
YOLO v4 (sota Apr 2020)

- **Bug of freebies:**
  улучшения на этапе тренировки. Влияет на точность модели (data augmentation, class imbalance, cost function, soft labeling)
- **Bug of specials:**
  улучшения на уровне архитектуры модели
  Влияет немного на точность и сильно на производительность (skip-connections, FPN и post-processing как non-maximum suppression (NMS), разные backbones, использовать разные претренировочные веса)

Что улучшили то?

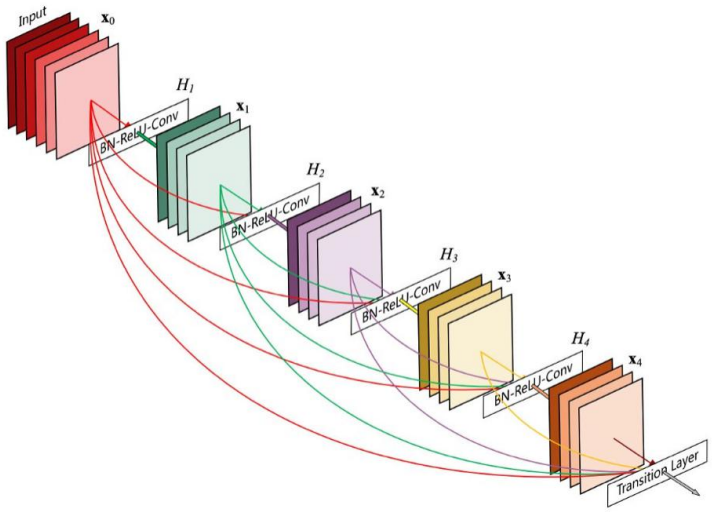
- Использовали BoF: CutMix и Mosaic data augmentation, DropBlock regularization, CmBN, DIoU Loss и т.д.

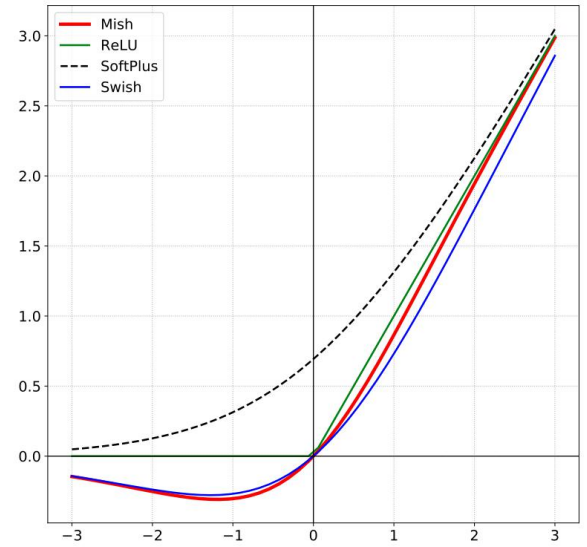




YOLO v4 (sota Apr 2020)

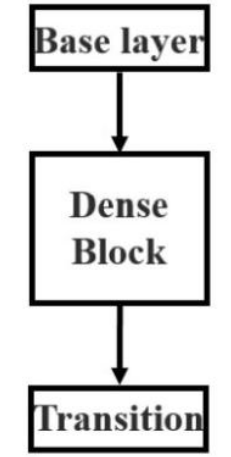
- **Bug of freebies:**
  улучшения на этапе тренировки. Влияет на точность модели (data augmentation, class imbalance, cost function, soft labeling)
- **Bug of specials:**
  улучшения на уровне архитектуры модели
  Влияет немного на точность и сильно на производительность (skip-connections, FPN и post-processing как non-maximum suppression (NMS), разные backbones, использовать разные претренировочные веса)



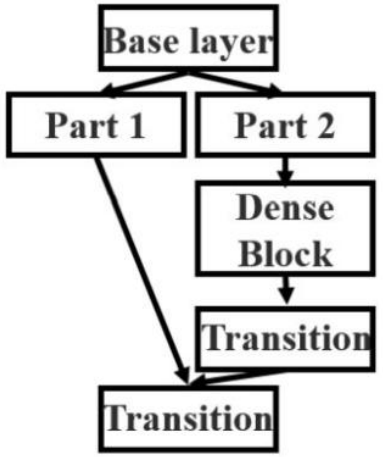


Что улучшили то?

- Использовали BoF: CutMix и Mosaic data augmentation, DropBlock regularization, CmBN, DIoU Loss и т.д.
- Использовали BoS:
  DenseBlock, PANet,
  Cross-Stage-Partial-connections (CSP),
  Mish-активация и т.д.



(a) DenseNet



(b) CSPDenseNet

- https://arxiv.org/pdf/2004.10934.pdf - сама Yolo4
- https://medium.com/@jonathan_hui/yolov4 - c9901eaa8e61 - полезная статья про Yolo4

EfficientNet (2019)

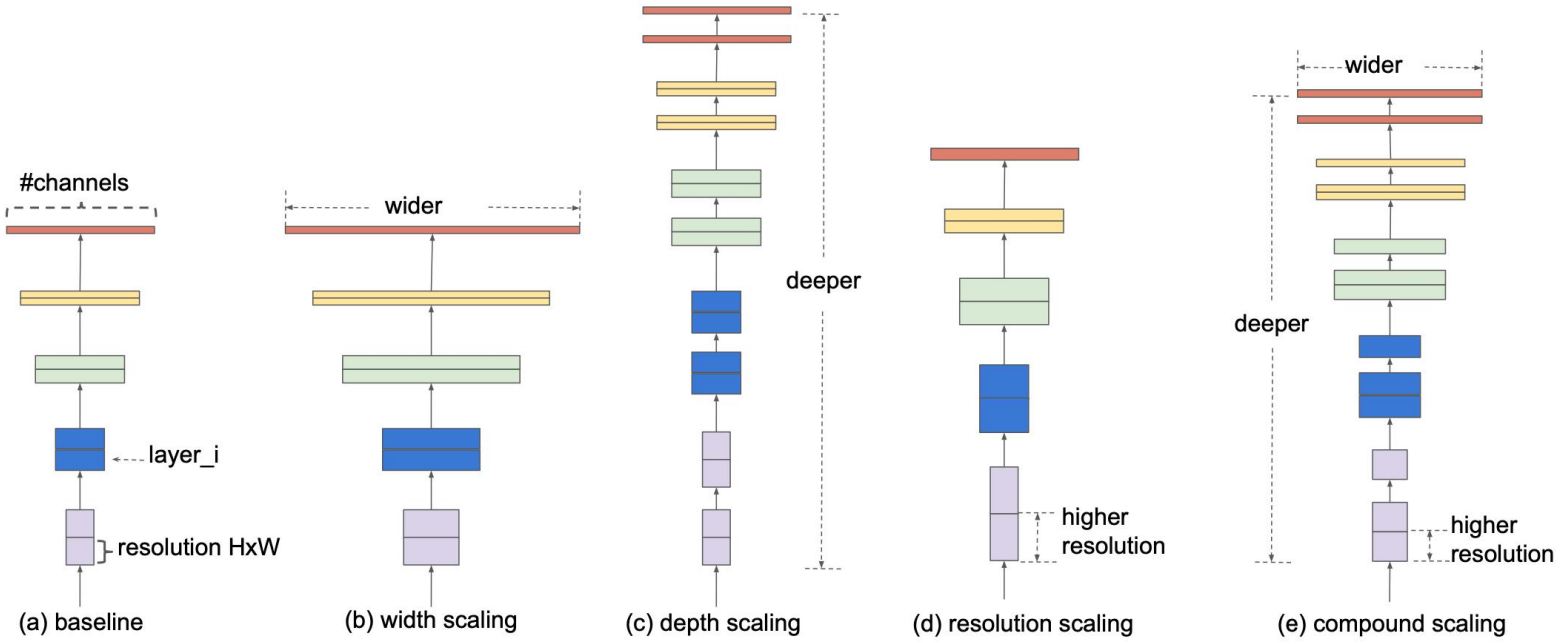
	Top1 Acc.	#Params
ResNet-152 (He et al., 2016)	77.8%	60M
EfficientNet-B1	**79.1%**	**7.8M**
ResNeXt-101 (Xie et al., 2017)	80.9%	84M
EfficientNet-B3	**81.6%**	**12M**
SENet (Hu et al., 2018)	82.7%	146M
NASNet-A (Zoph et al., 2018)	82.7%	89M
EfficientNet-B4	**82.9%**	**19M**
GPipe (Huang et al., 2018) †	84.3%	556M
EfficientNet-B7	**84.3%**	**66M**

†Not plotted

- state-of-the-art на ImageNet 84,3% accuracy*
- в 5.7 раз быстрее ResNet-152*
- в 7.6 раз меньше Resnet-152*
- на 2% точнее Resnet-152*

Принцип - маштабирование сверточных сеток (с помощью AutoML):

- маштабирование по ширине слоя(рис. a)/по глубине сети(увеличение слоев) (рис. c)/по глубине слоя (рис. d)
- комбинированное маштаброване (рис. e)



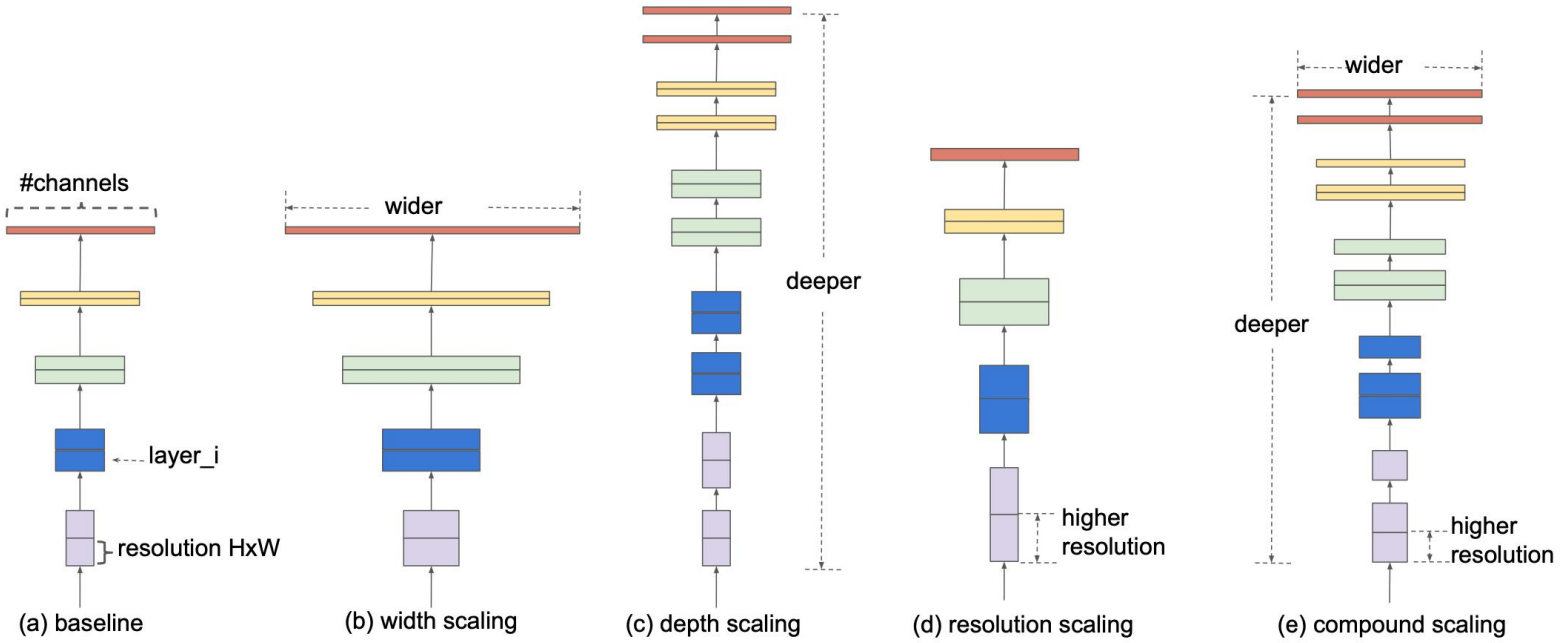
*https://arxiv.org/pdf/1905.11946.pdf - статья

EfficientNet (2019)

	Top1 Acc.	#Params
ResNet-152 (He et al., 2016)	77.8%	60M
EfficientNet-B1	**79.1%**	**7.8M**
ResNeXt-101 (Xie et al., 2017)	80.9%	84M
EfficientNet-B3	**81.6%**	**12M**
SENet (Hu et al., 2018)	82.7%	146M
NASNet-A (Zoph et al., 2018)	82.7%	89M
EfficientNet-B4	**82.9%**	**19M**
GPipe (Huang et al., 2018) †	84.3%	556M
EfficientNet-B7	**84.3%**	**66M**

†Not plotted

- state-of-the-art на ImageNet 84,3% accuracy*
- в 5.7 раз быстрее ResNet-152*
- в 7.6 раз меньше Resnet-152*
- на 2% точнее Resnet-152*



*https://arxiv.org/pdf/1905.11946.pdf - статья

Принцип - маштабирование сверточных сеток (с помощью AutoML):

- маштабирование по ширине слоя(рис. a)/по глубине сети(увеличение слоев) (рис. c)/по глубине слоя (рис. d)
- комбинированное маштабрование (рис. e)

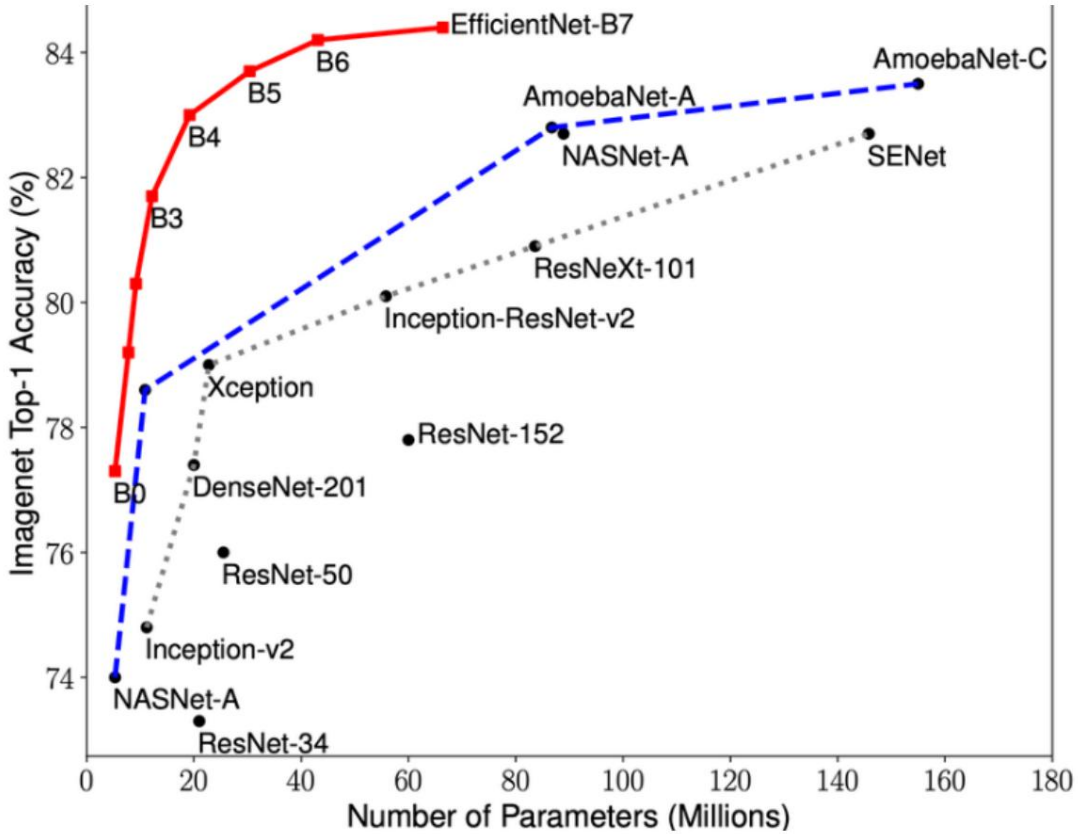
Алгоритм:

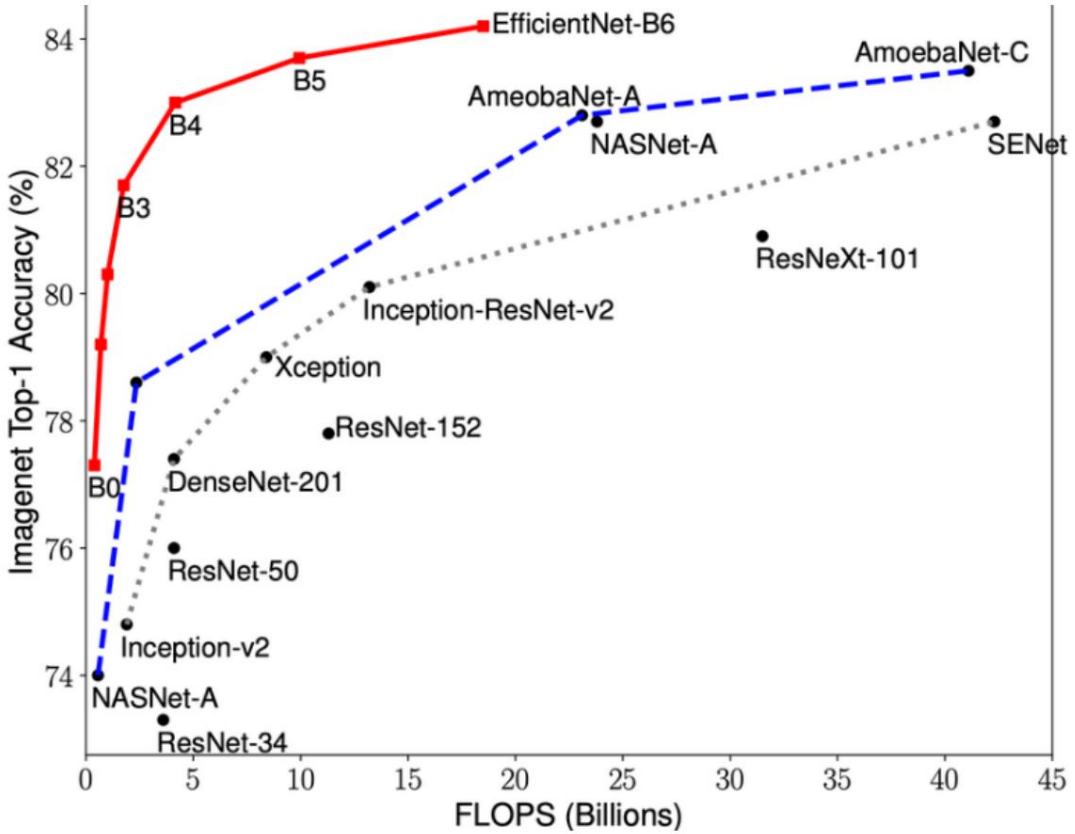
1. Важно найти коэффициенты маштабирования alpha, beta, gamma из выражения:

$$\text{depth: } d = \alpha^\phi$$
$$\text{width: } w = \beta^\phi$$
$$\text{resolution: } r = \gamma^\phi$$
$$\text{s.t. } \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$$
$$\alpha \geq 1, \beta \geq 1, \gamma \geq 1$$

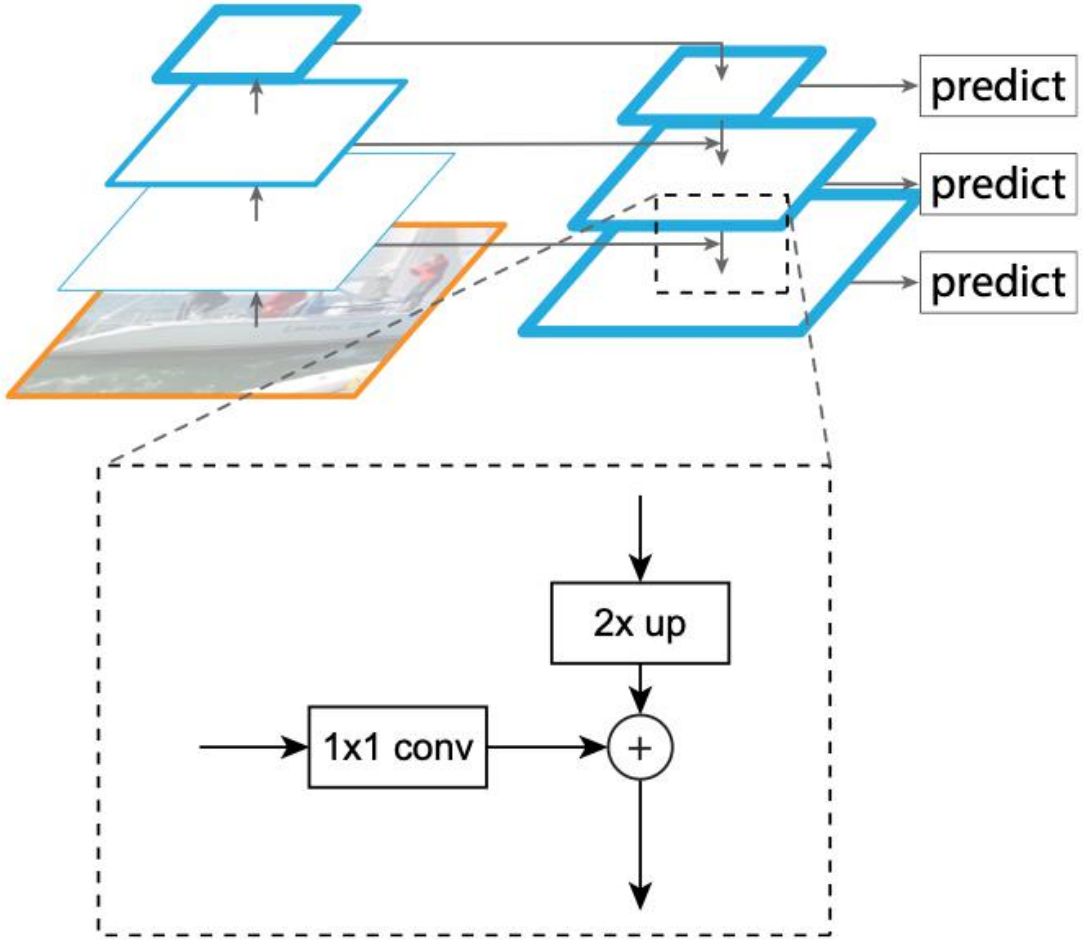
2. при зафиксированных alpha, beta, gamma маштабируется сеть увеличиваем fi на 1 (fi ==2,3,4..8)на каждой итерации и получили сети EfficientNet-B1 до EfficientNet-B7

EfficientNet





Feature Pyramid Network (FPN)



https://arxiv.org/pdf/1612.03144.pdf

PANet - Path Aggregation Network (2018 Jun)

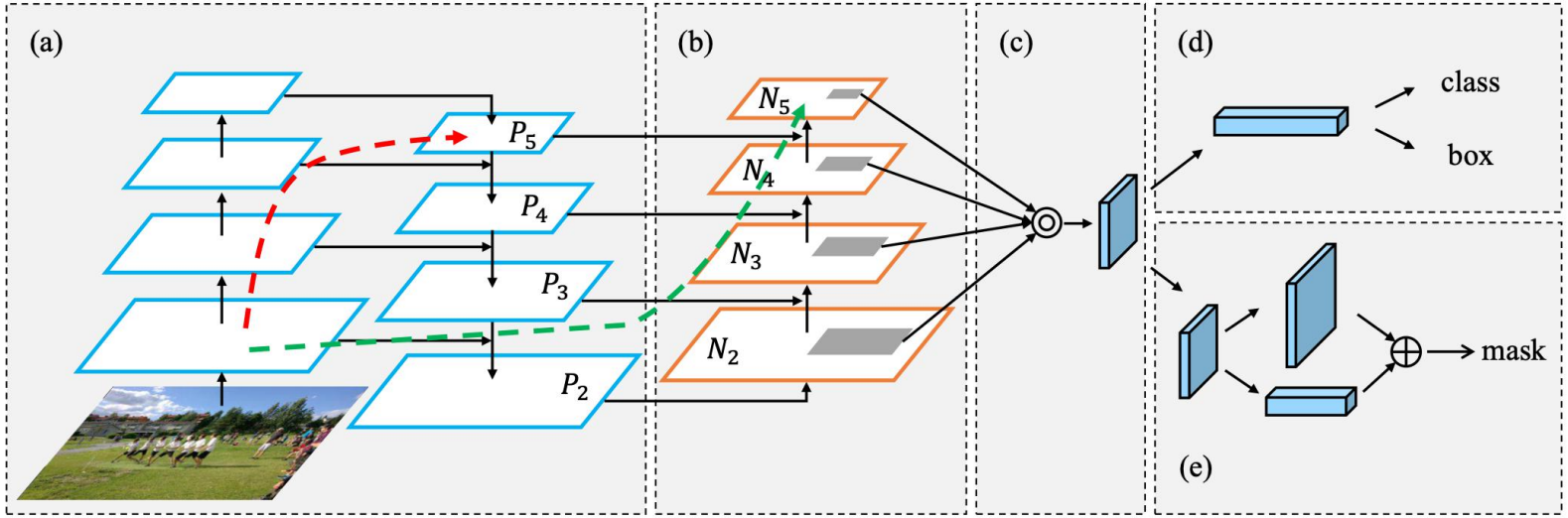
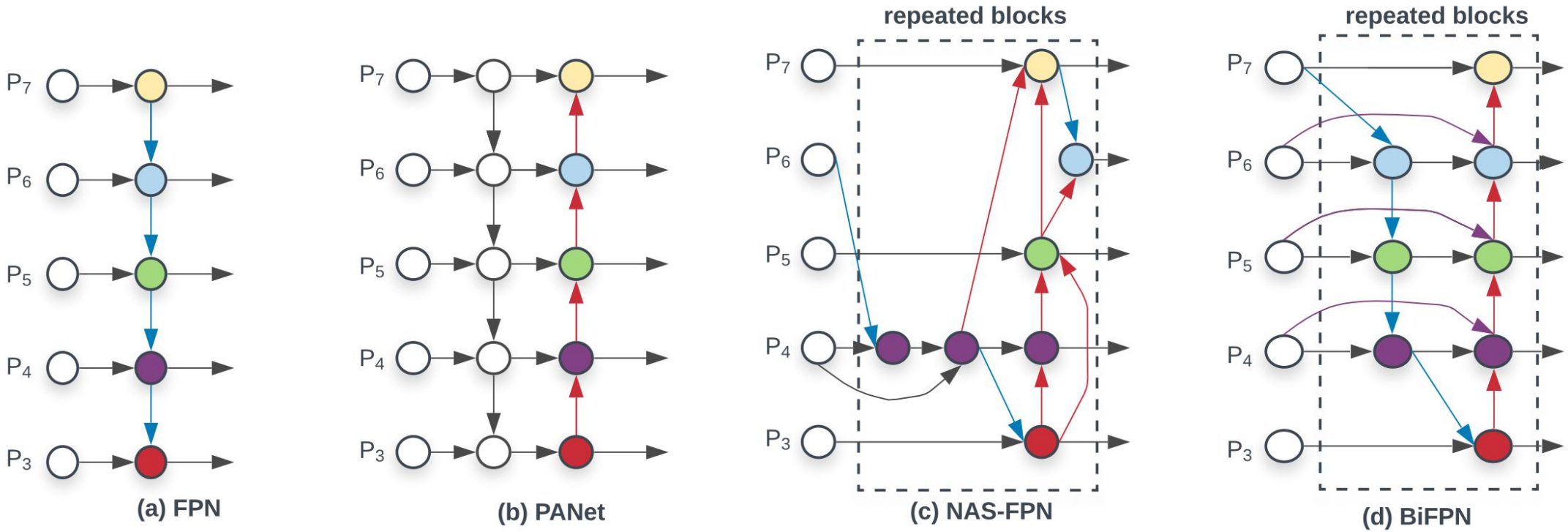


Figure 1. Illustration of our framework. (a) FPN backbone. (b) Bottom-up path augmentation. (c) Adaptive feature pooling. (d) Box branch. (e) Fully-connected fusion. Note that we omit channel dimension of feature maps in (a) and (b) for brevity.

- FPN + bottom-top path
- (c) ROIAlign with (element-wise max or sum)

https://openaccess.thecvf.com/content_cvpr_2018/papers/Liu_Path_Aggregation_Network_CVPR_2018_paper.pdf

BiFPN and others



	AP	#Params ratio	#FLOPs ratio
Repeated top-down FPN	42.29	1.0x	1.0x
Repeated FPN+PANet	44.08	1.0x	1.0x
NAS-FPN	43.16	0.71x	0.72x
Fully-Connected FPN	43.06	1.24x	1.21x
BiFPN (w/o weighted)	**43.94**	**0.88x**	**0.67x**
BiFPN (w/ weighted)	**44.39**	**0.88x**	**0.68x**

$$P_7^{out} = Conv(P_7^{in})$$
$$P_6^{out} = Conv(P_6^{in} + Resize(P_7^{out}))$$
$$\dots$$
$$P_3^{out} = Conv(P_3^{in} + Resize(P_4^{out}))$$

EfficientDet (2020)

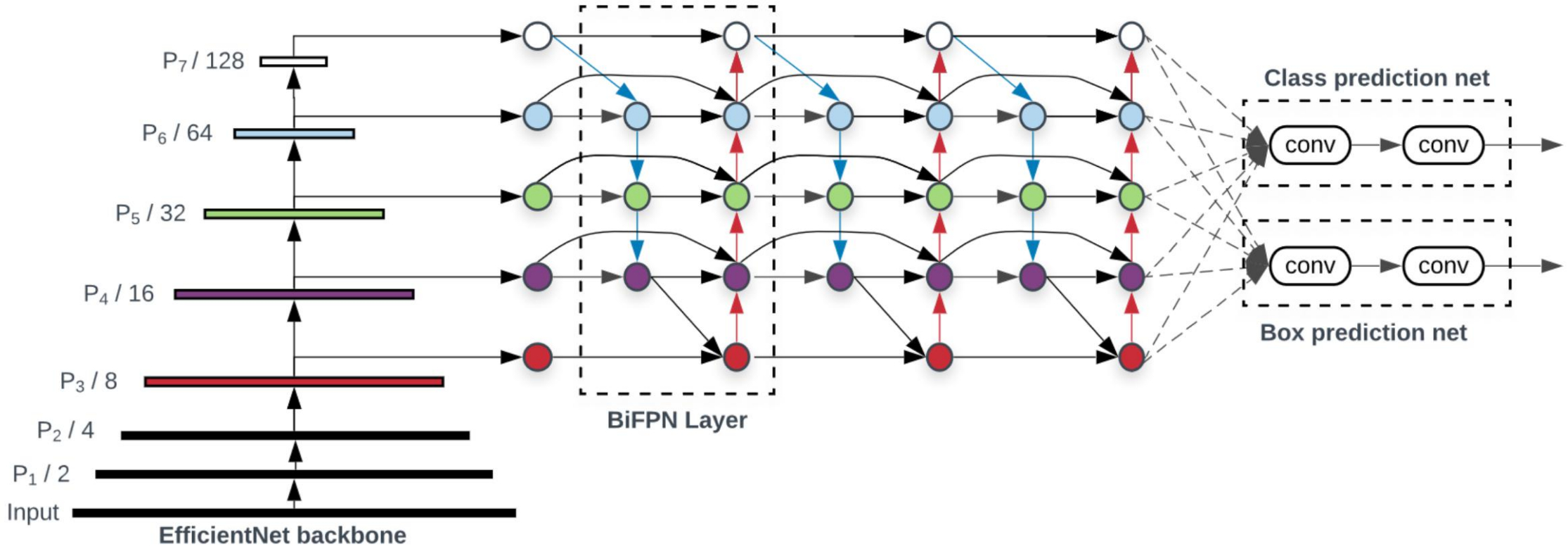
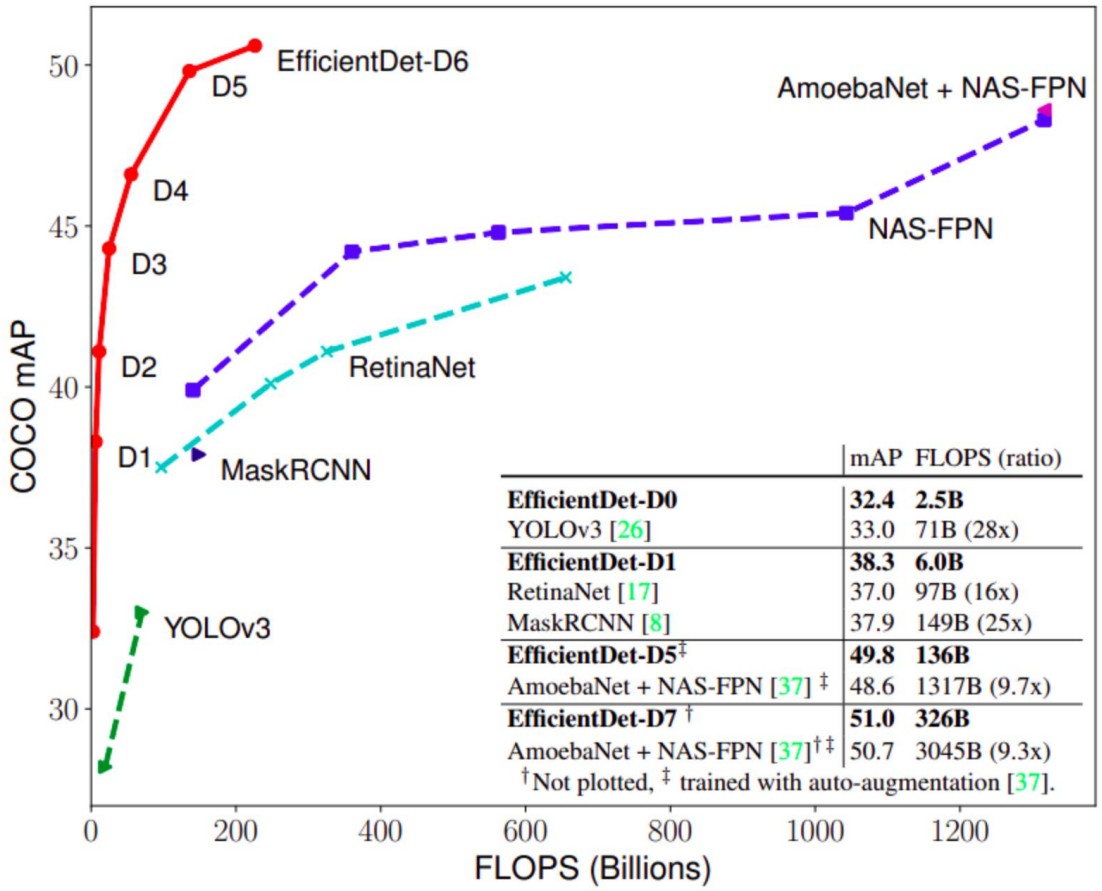


Рисунок — Архитектура EfficientDet == EfficientNet + BiFPN + сеть вычисления класс/рамка

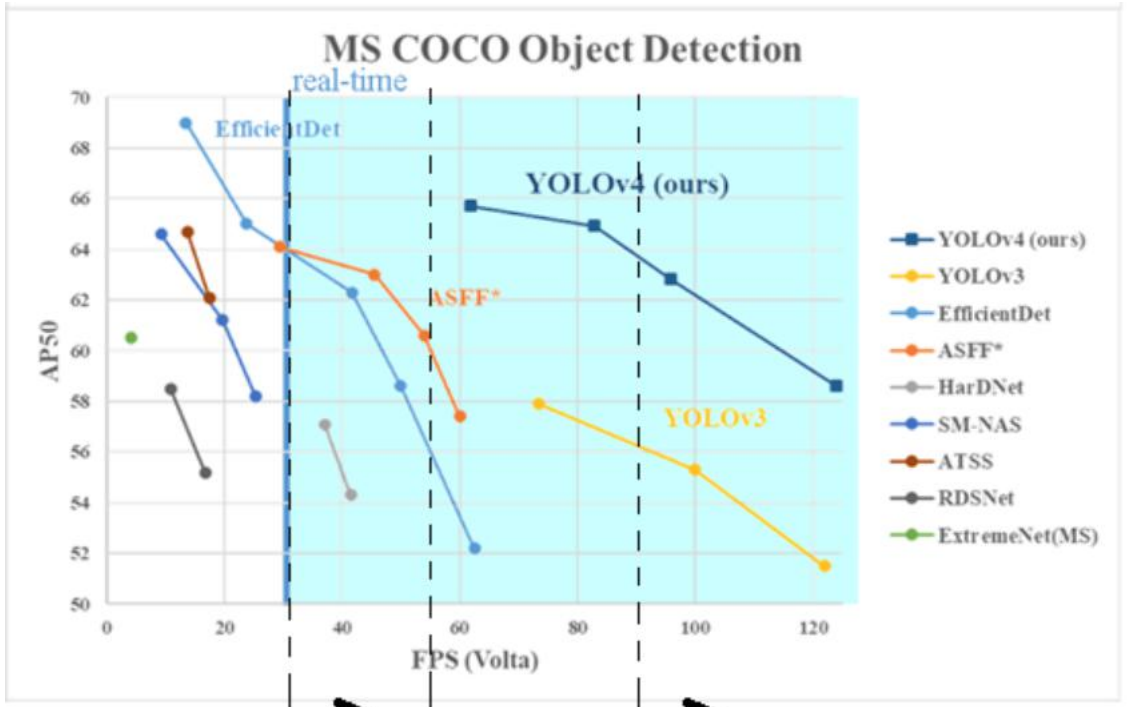
https://arxiv.org/pdf/1911.09070.pdf

Сравнение



	mAP	FLOPS (ratio)
EfficientDet-D0	**32.4**	**2.5B**
YOLOv3 [26]	33.0	71B (28x)
EfficientDet-D1	**38.3**	**6.0B**
RetinaNet [17]	37.0	97B (16x)
MaskRCNN [8]	37.9	149B (25x)
EfficientDet-D5‡	**49.8**	**136B**
AmoebaNet + NAS-FPN [37] ‡	48.6	1317B (9.7x)
EfficientDet-D7 †	**51.0**	**326B**
AmoebaNet + NAS-FPN [37]†‡	50.7	3045B (9.3x)

†Not plotted, ‡ trained with auto-augmentation [37].



real-time (30 FPS) on
TitanV/TeslaV100 (server)
250 Watt
$2250 - $8000

real-time (30 FPS) on
RTX 2070 (desktop)
175 Watt
$450

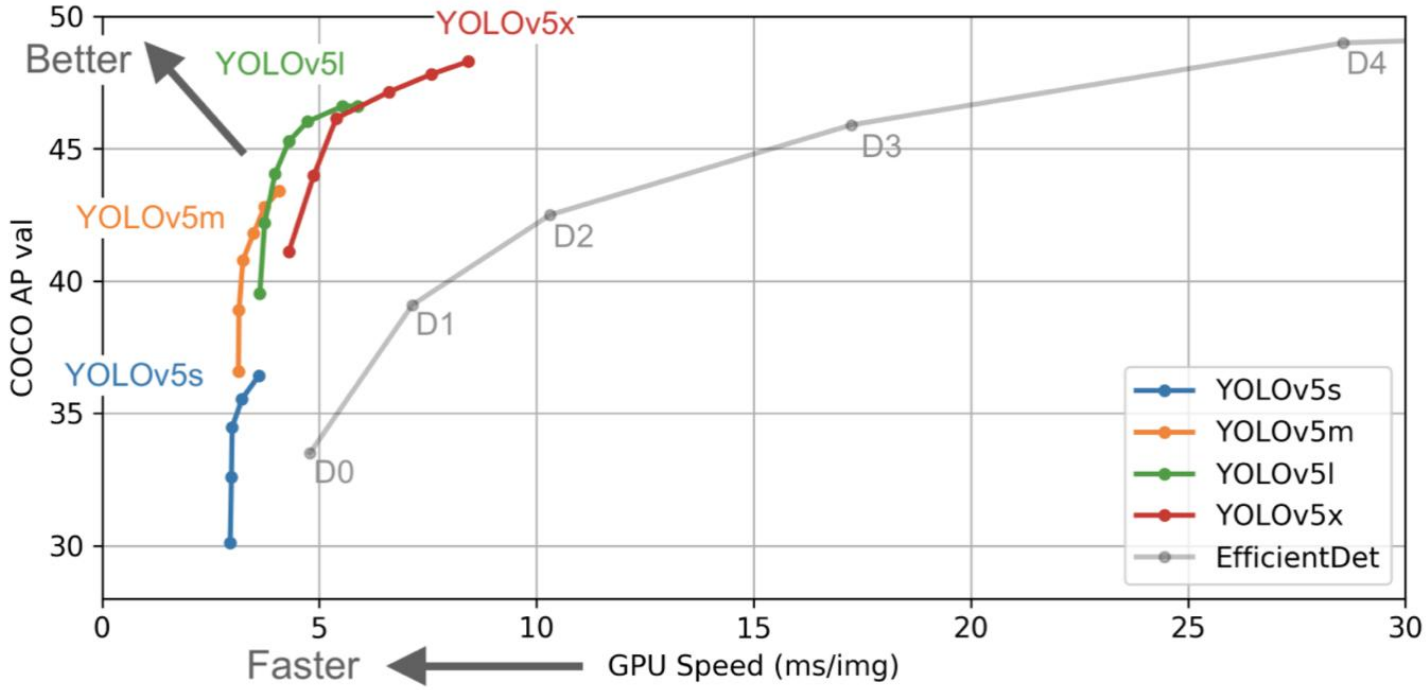
real-time (30 FPS) on
RTX 2060 MXM (Mobile)
80 Watt
$700

9 FPS on
Jetson AGX Xavier (embedded)
30 Watt
$700

*Измерение FLOPS не чистое, из-за использования GPU разных мощностей в разных моделей

YOLOv5 (2020 June)

- теперь реализация на PyTorch
- кванитизация модели с 32bit до 16bit
- V100 GPU with postprocessing and NMS
- обучение с аугментацией от Yolo4
- создание нескольких моделей с разным количеством параметров small, medium, large, very large



https://github.com/ultralytics/yolov5 - репозиторий
https://blog.roboflow.com/yolov5-improvements-and-evaluation/ - описание

Полезные ссылки

- https://towardsdatascience.com/fasterrcnn-explained-part-1-with-code-599c16568cff - пример FasterRCNN на pytorch
- https://towardsdatascience.com/understanding-region-of-interest-part-2-roi-align-and-roi-warp-f795196fc193 - ROIAlign
- https://arxiv.org/pdf/1512.02325.pdf - статья SSD
- https://arxiv.org/pdf/1411.4038.pdf - статья FCN
- https://pjreddie.com/publications/ - сайт автора YOLO
- https://arxiv.org/pdf/1905.11946.pdf - статья EfficientNet
- https://paperswithcode.com/methods/area/computer-vision - архитектуры с кодом
- https://openaccess.thecvf.com/content_CVPR_2020/papers/Tan_EfficientDet_Scalable_and_Efficient_Object_Detection_CVPR_2020_paper.pdf - статья EfficientDet
- https://github.com/ultralytics/yolov5 - репозиторий YOLOv5