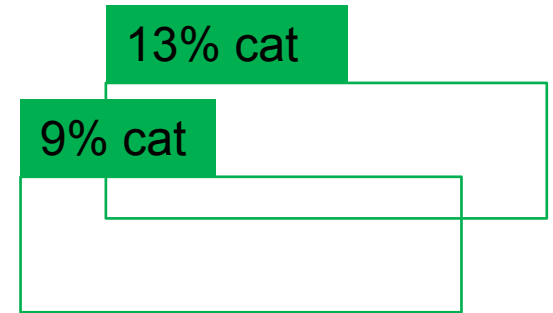


Object Detection 2



Лекция 6

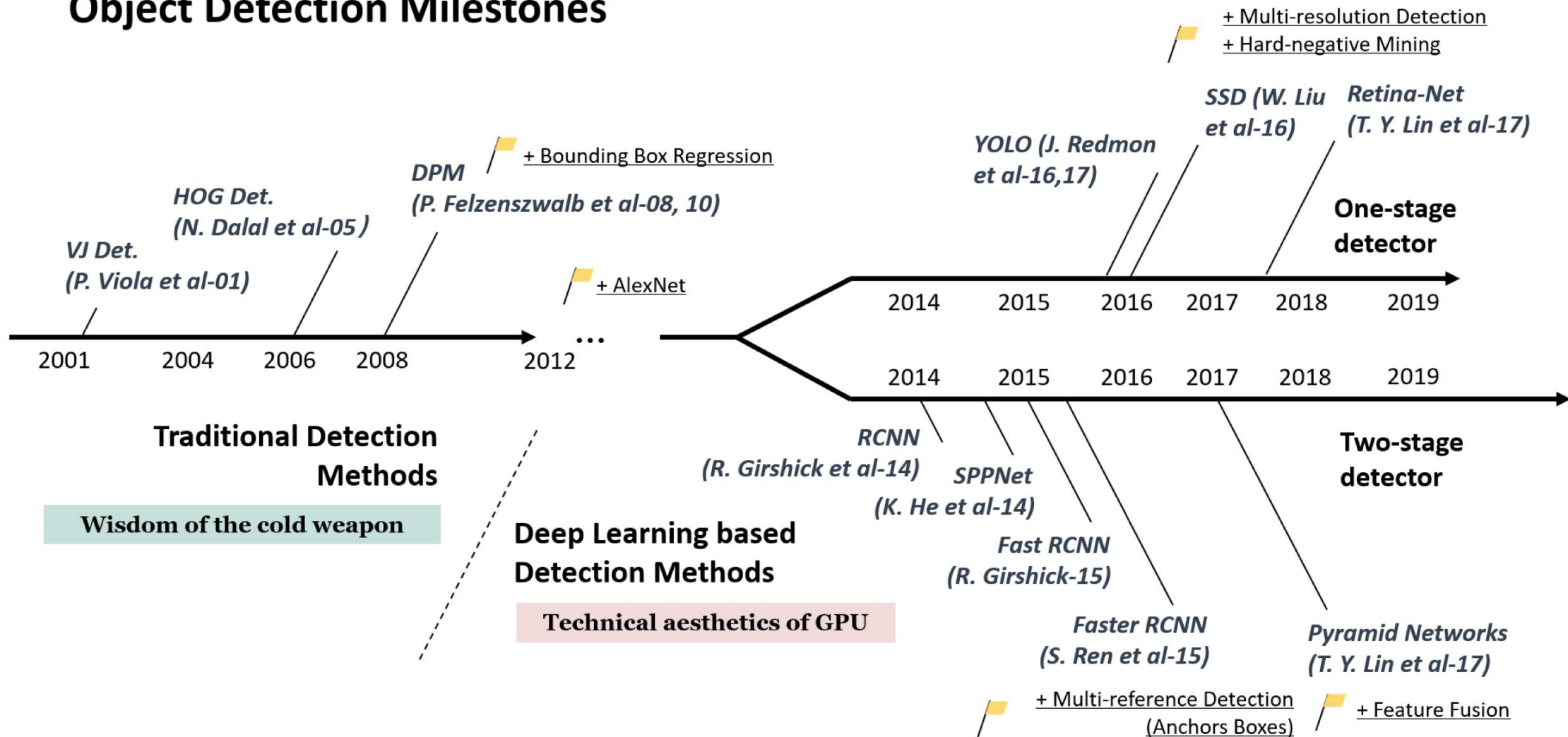
99% ISP RAS

Беляева Оксана Владимировна

2020 год

ИСП РАН

Object Detection Milestones

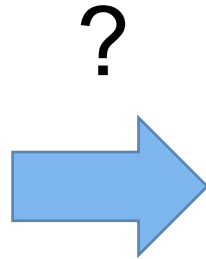


Region-based methods

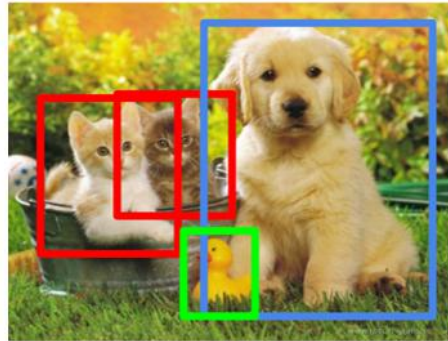
Classification



CAT



Object Detection



CAT, DOG, DUCK

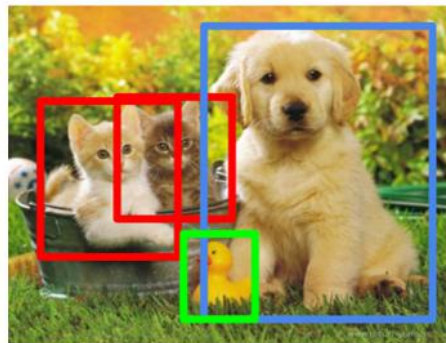
Region-based methods

Classification



CAT

Object Detection



CAT, DOG, DUCK

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

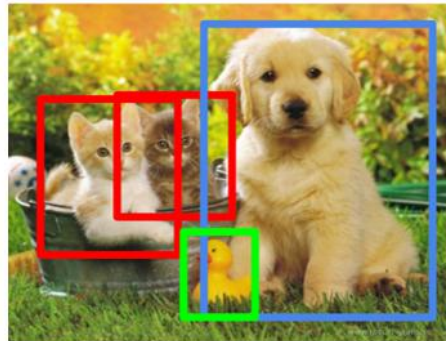
Region-based methods

Classification



CAT

Object Detection

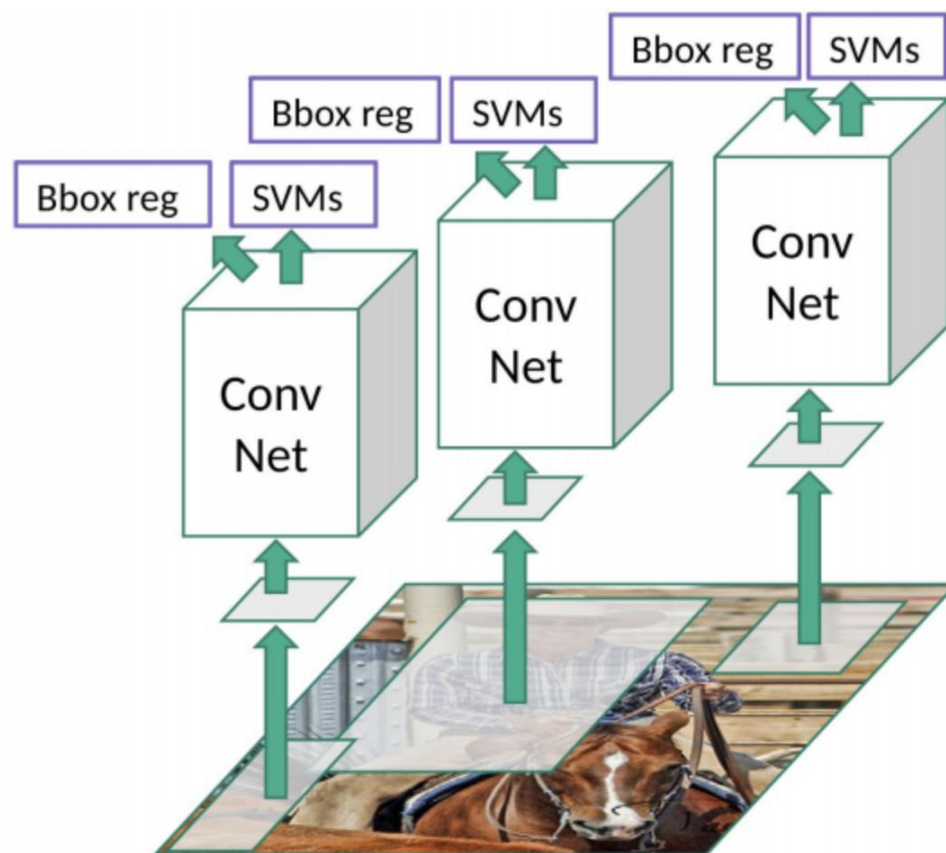


CAT, DOG, DUCK

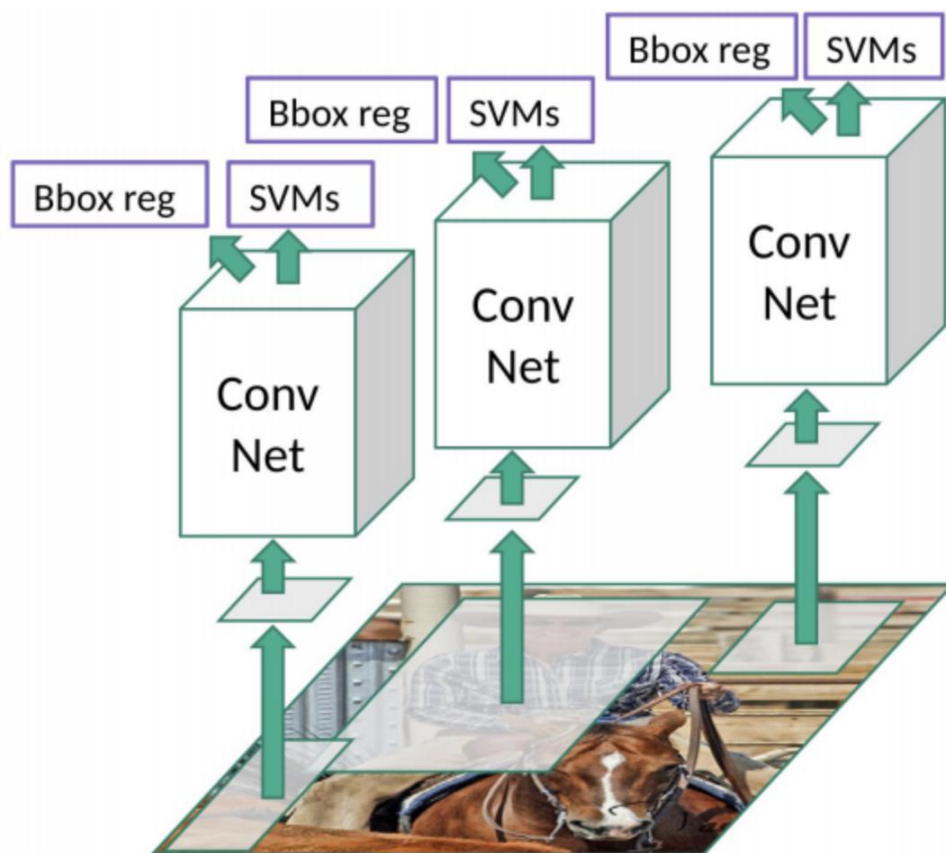
- Используется метод для извлечения регионов (частей изображения) с помощью окон
- Нужно использовать окна разных размеров и соотношение сторон для извлечения разного размера объектов

Two-stage detectors (region-based)

Проблемы ?



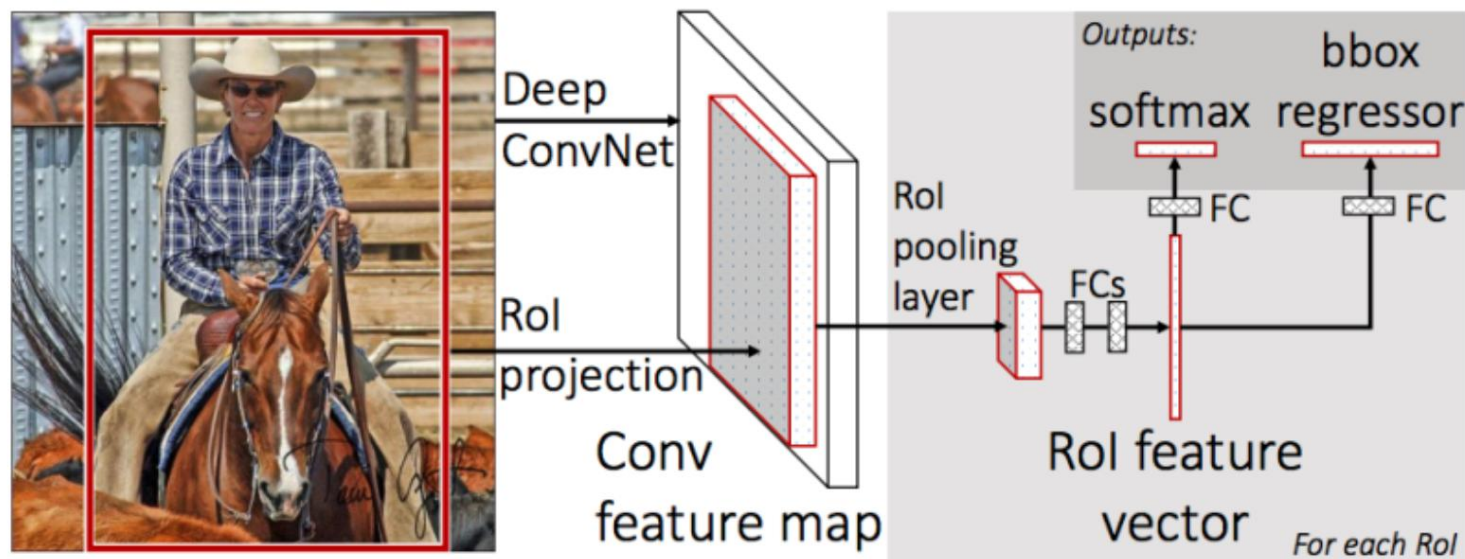
Two-stage detectors (region-based)



Проблемы ?

- Долгое обучение на случайных регионах с проходом через всю CNN для извлечения признаков
- Искажение регионов за счет предварительного resize для CNN
- Эвристический алгоритм Selective Search не обучается и стабильно ошибается
- Обработка одного изображения около 47 секунд :(

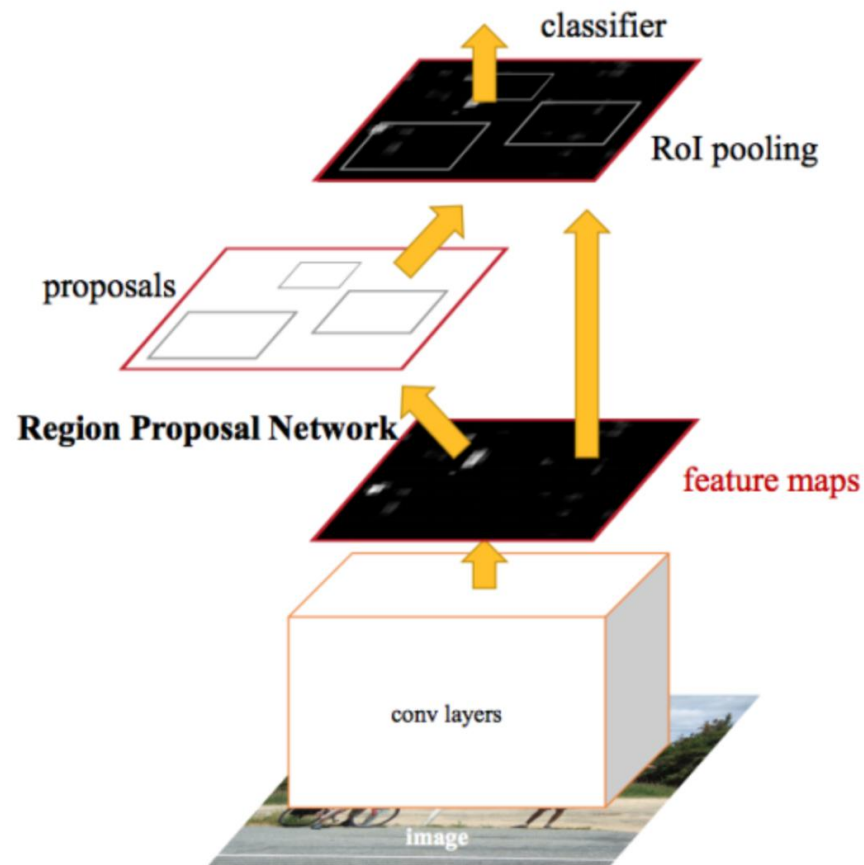
Fast RCNN



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

- 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 $H \times W$ ячеек размером $h/H \times w/W$. (Авторы использовали $W=H=7$).

Faster RCNN



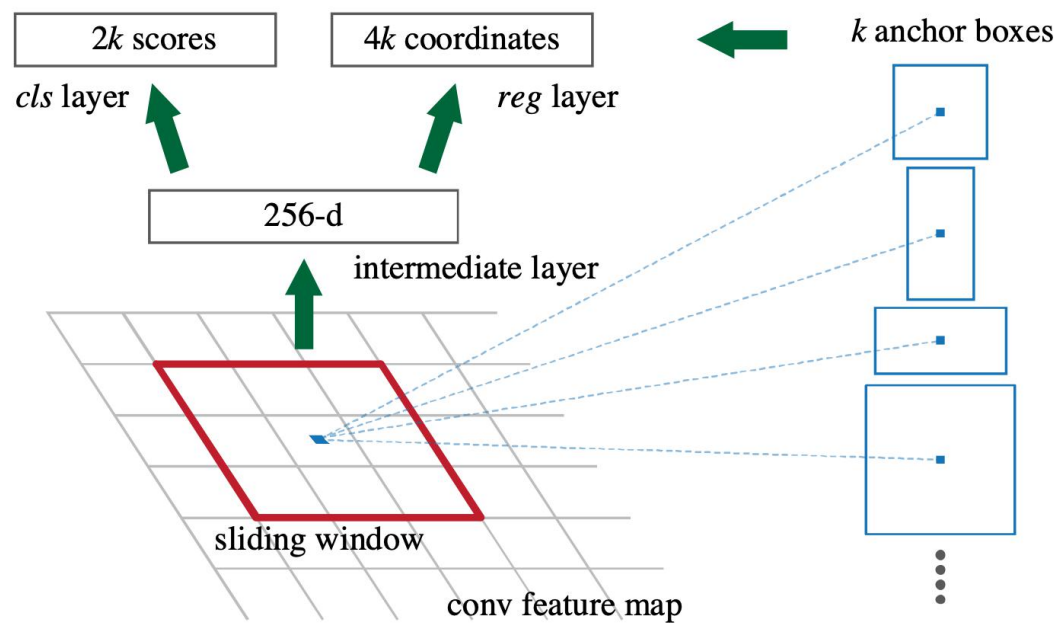
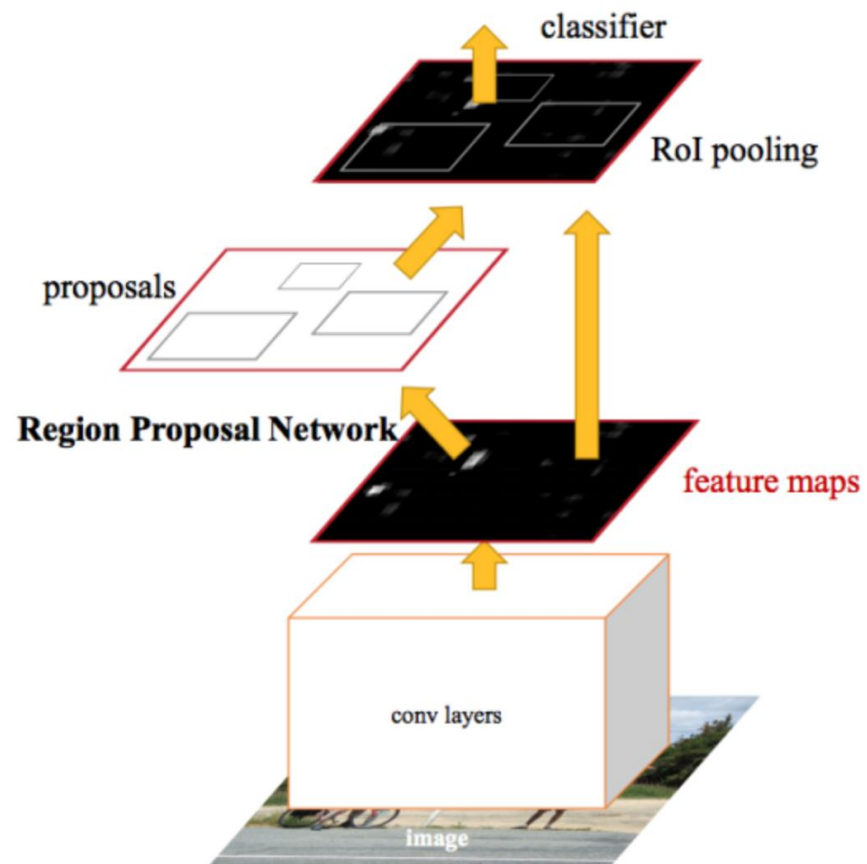
Выделяют 2 части: в two-stage detector:

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

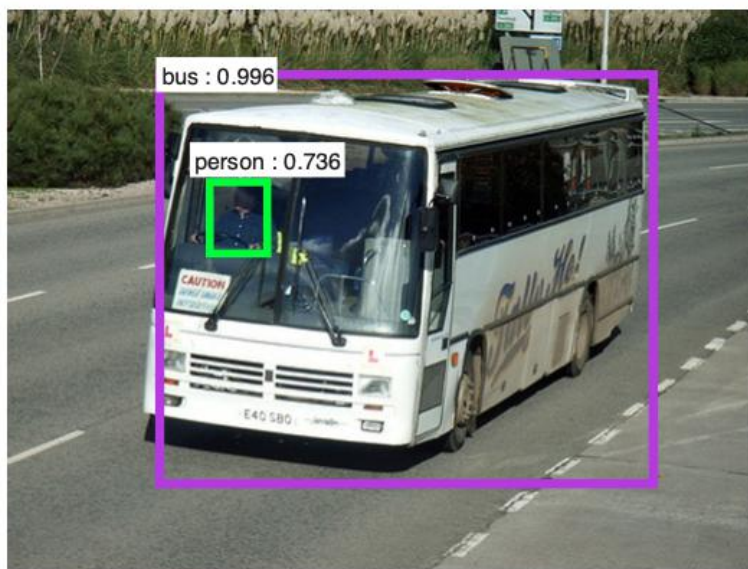
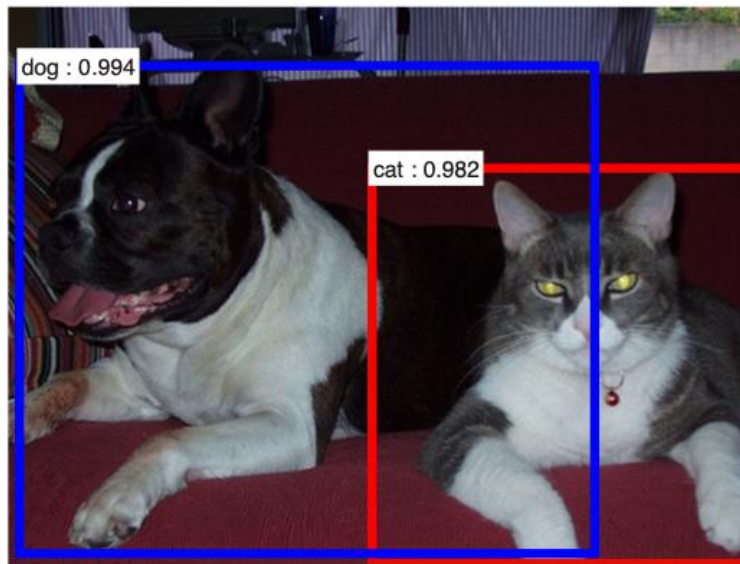
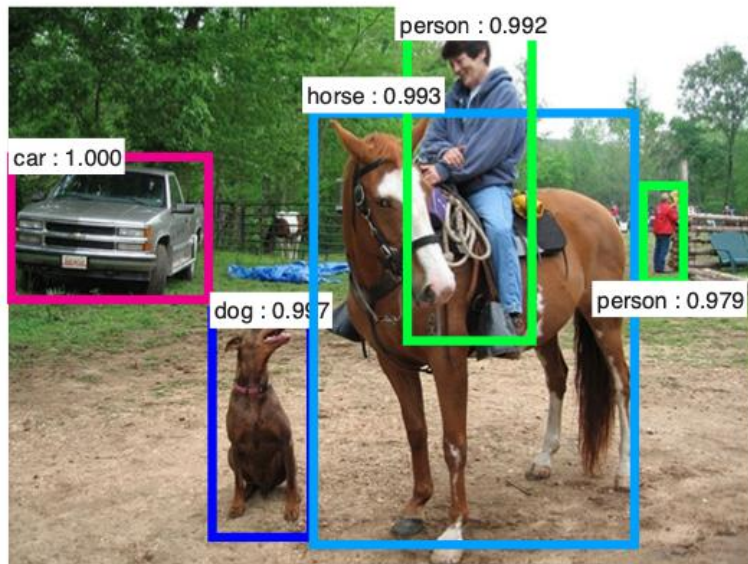
1 - генерация регионов интересов RoI (Region Of Interest)

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

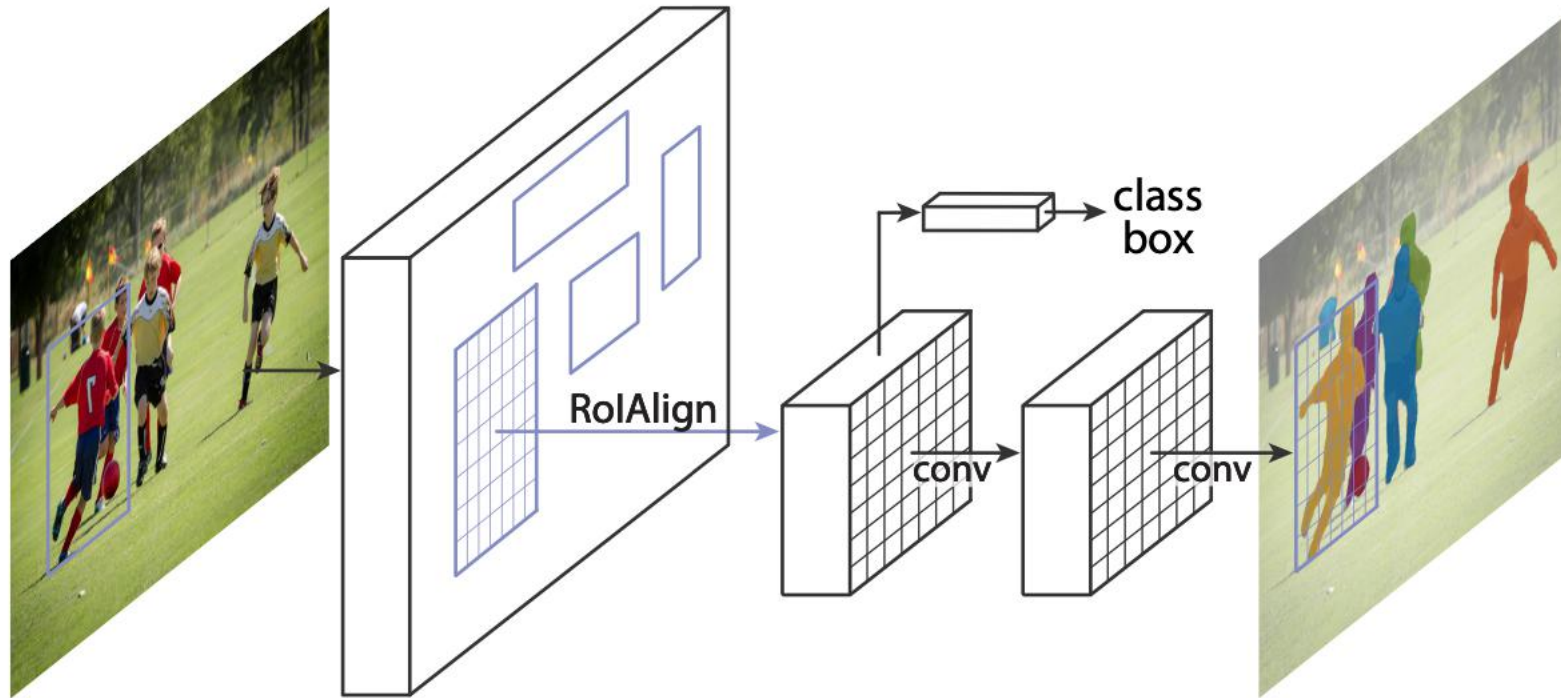
Faster RCNN



Faster RCNN



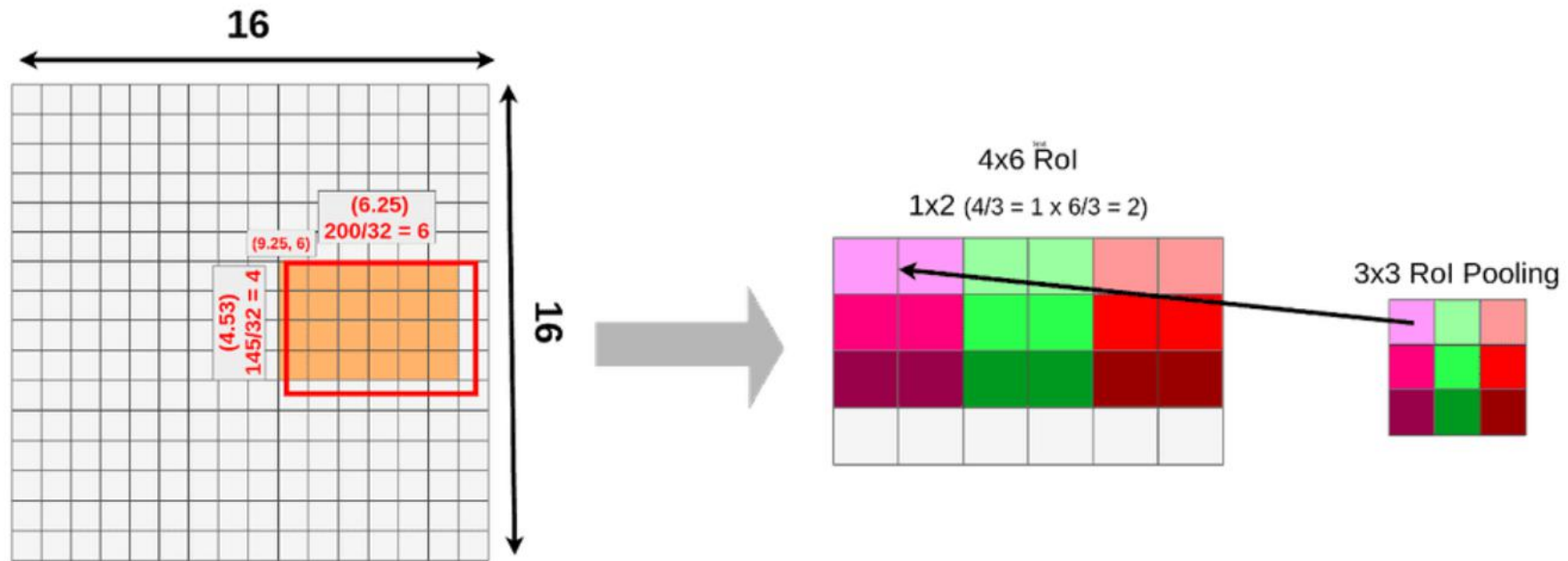
Mask RCNN



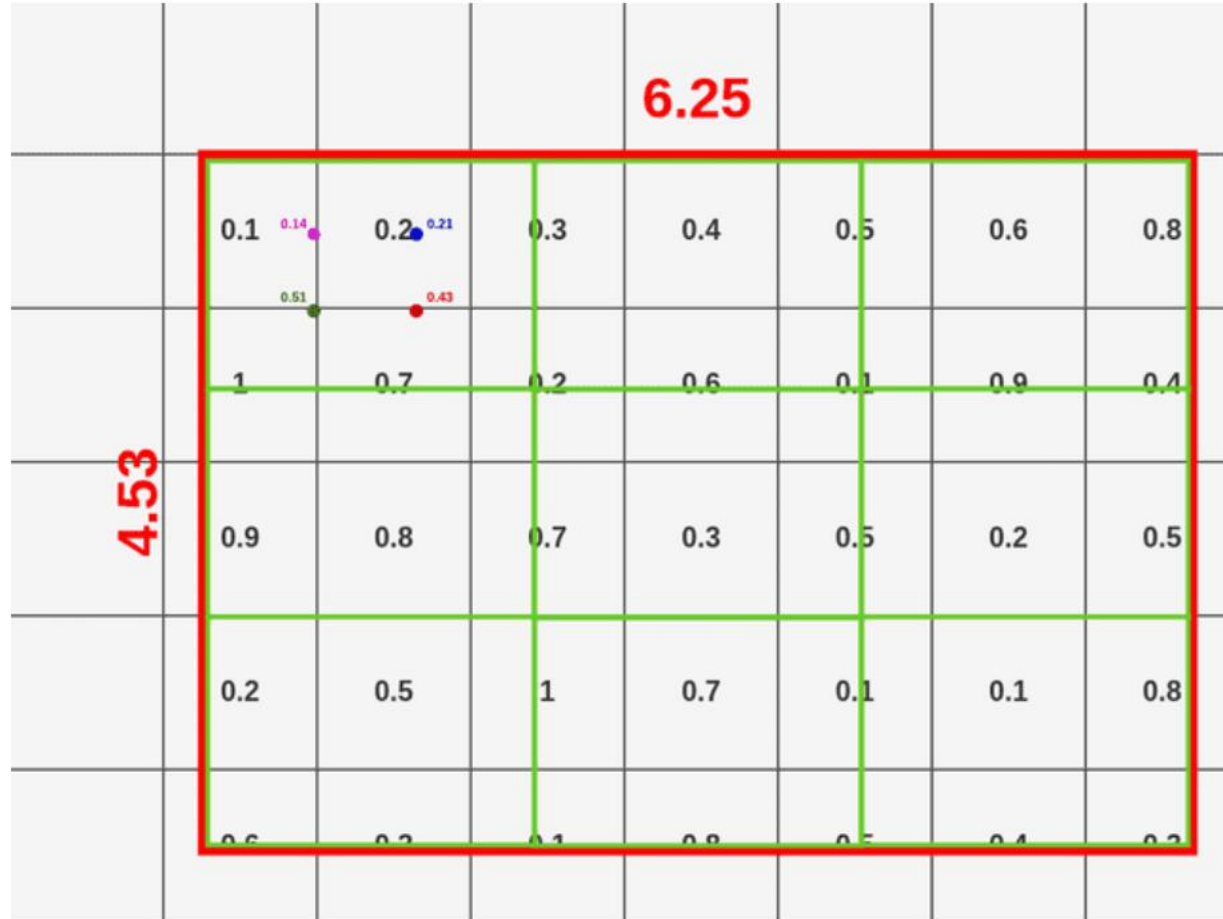
- Использование доп. ветви для решения задачи instance segmentation
- Маска каждого класса определяется независимо от других классов;
- RoIAlign вместо RoIPooling

$$L = \tilde{L}_{cls} + L_{box} + L_{mask}.$$

RoI Pooling



Rol Align



$$1 \times 1 = \text{MAX}(0.14, 0.21, 0.51, 0.43) = 0.51$$

3x3 RolAlign

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

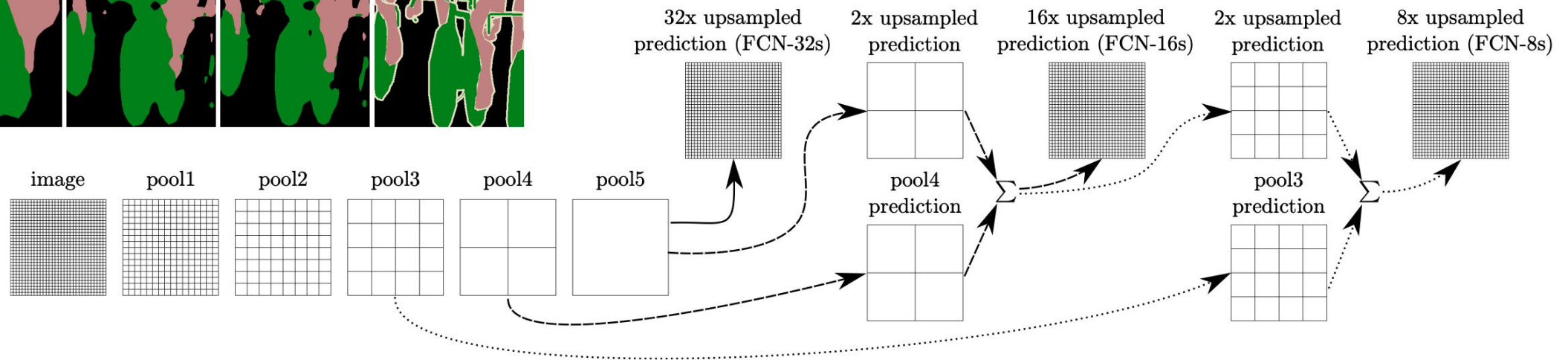
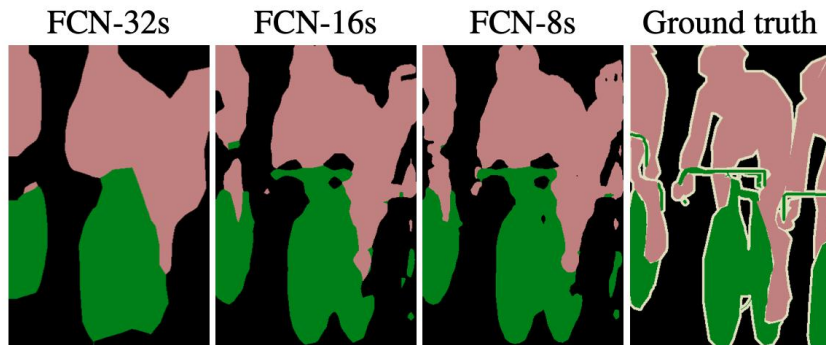
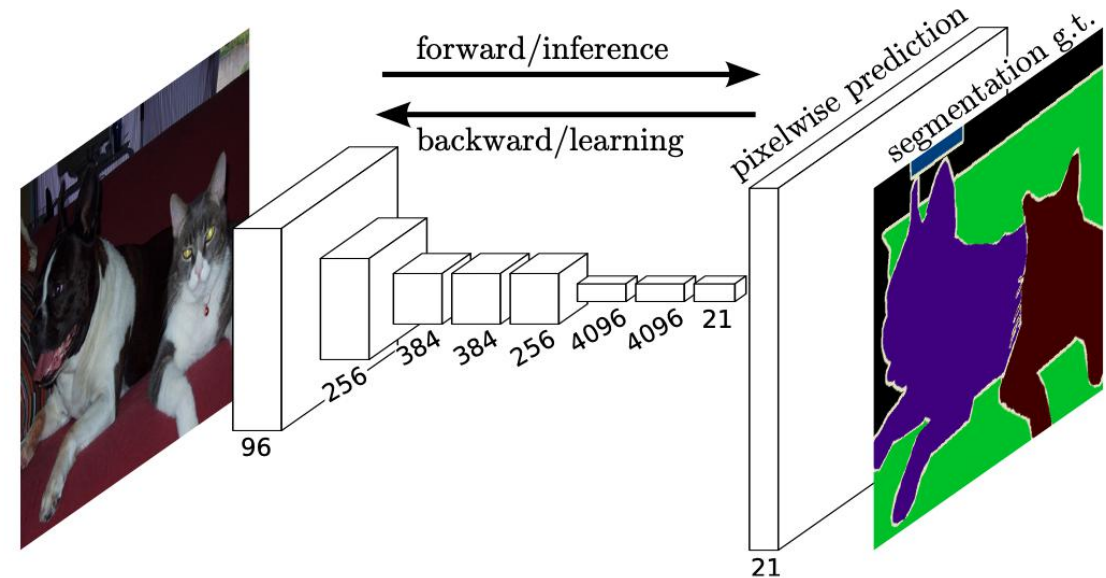
AP^{bb} это average precision для определения bounding boxes.
Тестирование проводилось на ResNet-50-C5 со stride=32.

Mask RCNN

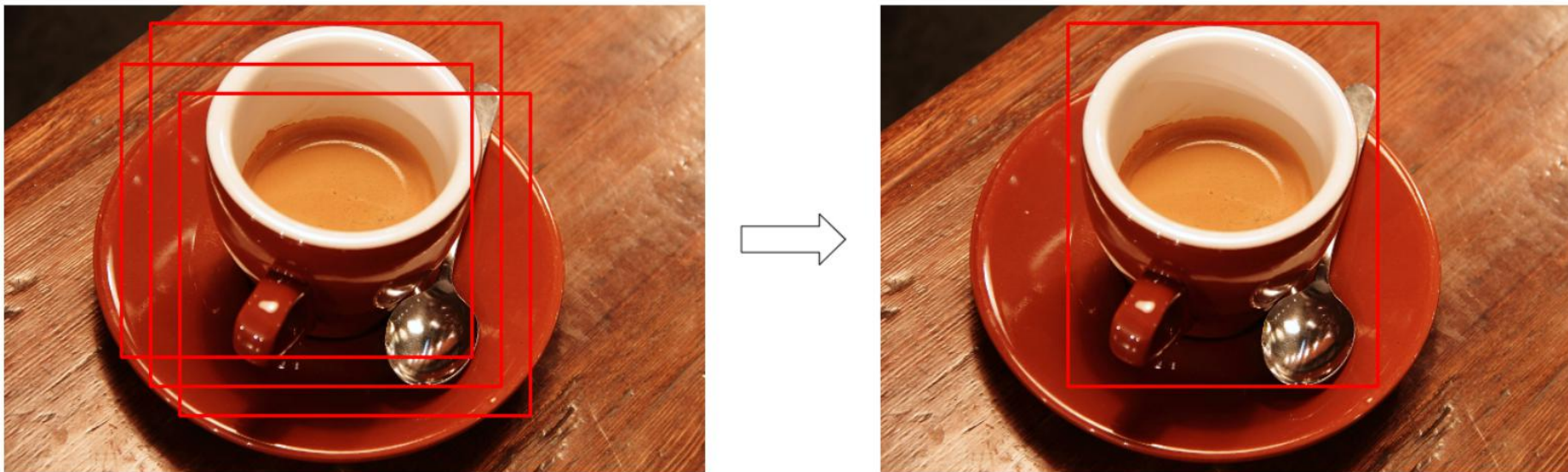


Fully Convolutional Network (FCN)

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

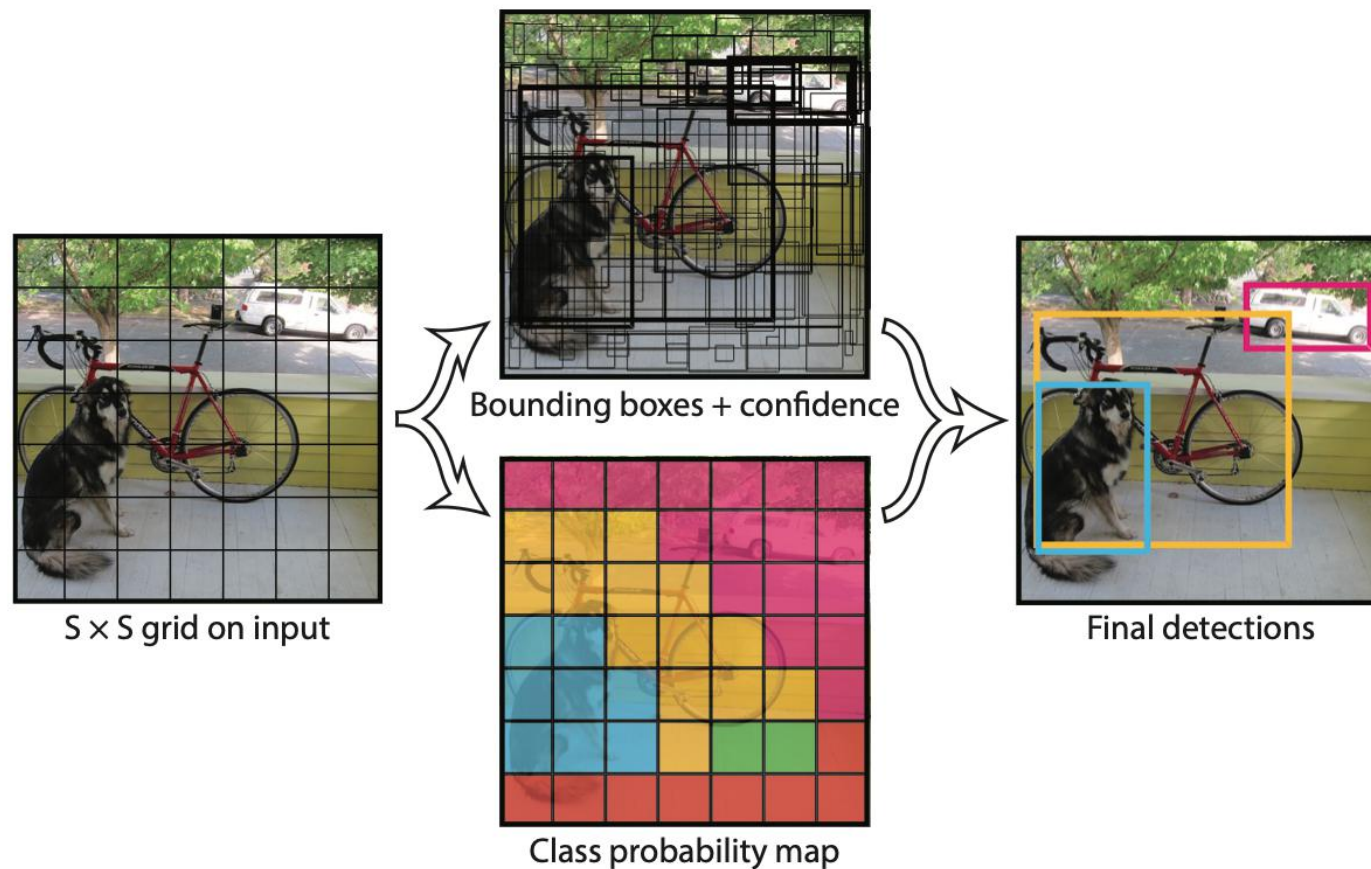


Non-Maximum-Suppression (NMS)

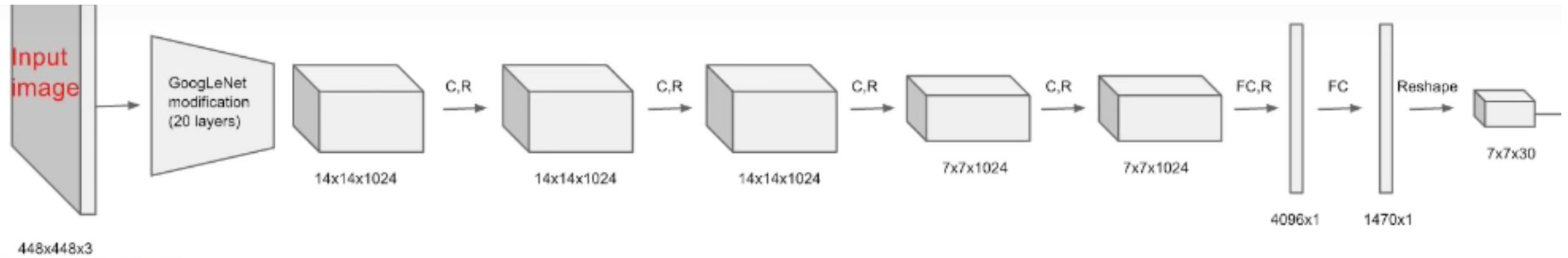


1. Гипотезы сортируются по их «уверенности».
 2. В цикле выбирается первая гипотеза A (имеет наибольшую величину уверенности) и добавляется в результирующий набор.
 3. В цикле выбирается следующая B.
 4. Если между выбранными гипотезами $\text{IoU}(A, B) > \text{threshold}$, то вторая гипотеза B отбрасывается и далее не присутствует в результирующем наборе.
- Все повторяется, начиная с шага 2 до момента полного перебора гипотез.

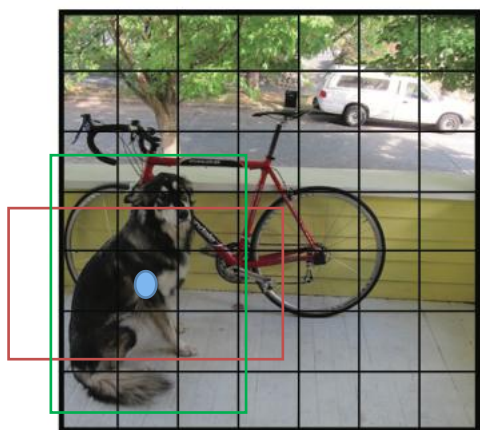
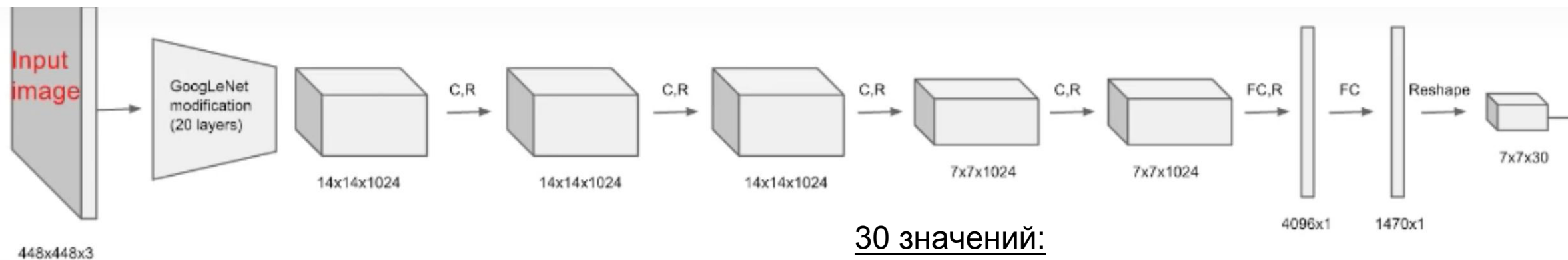
YOLO (You Only Look Once)



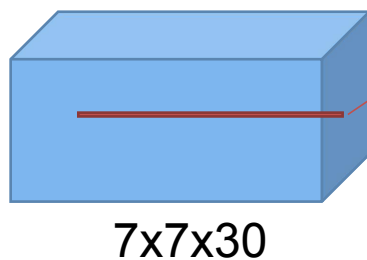
YOLO



YOLO



$S \times S$ grid on input
 $S = 7$



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

$\text{Conf_box_class} = \text{conf_box} * \text{conf_class}$

Total: $7 \times 7 \times 2 = 98$ boxes

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

YOLO

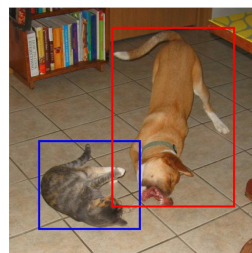
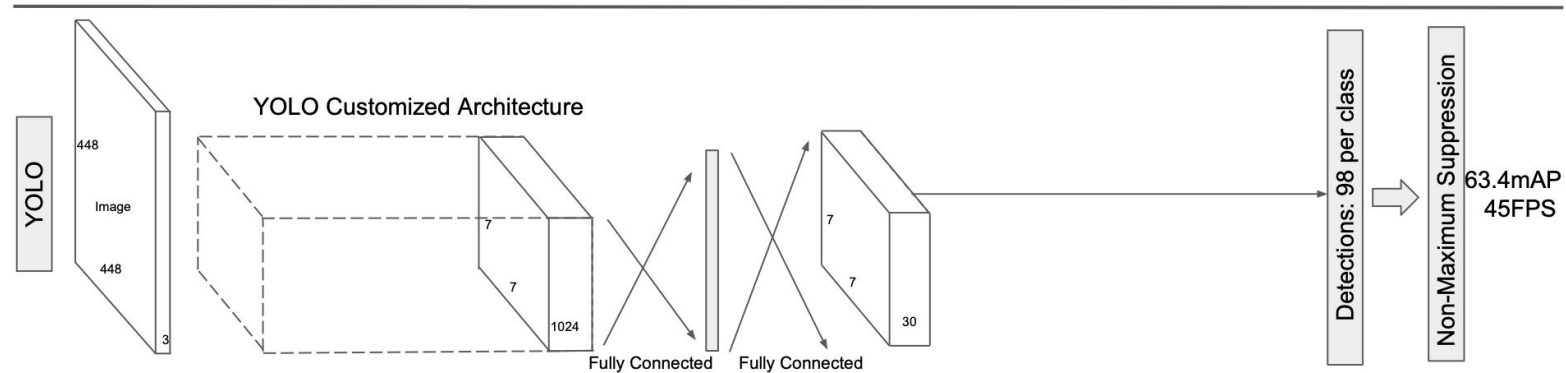
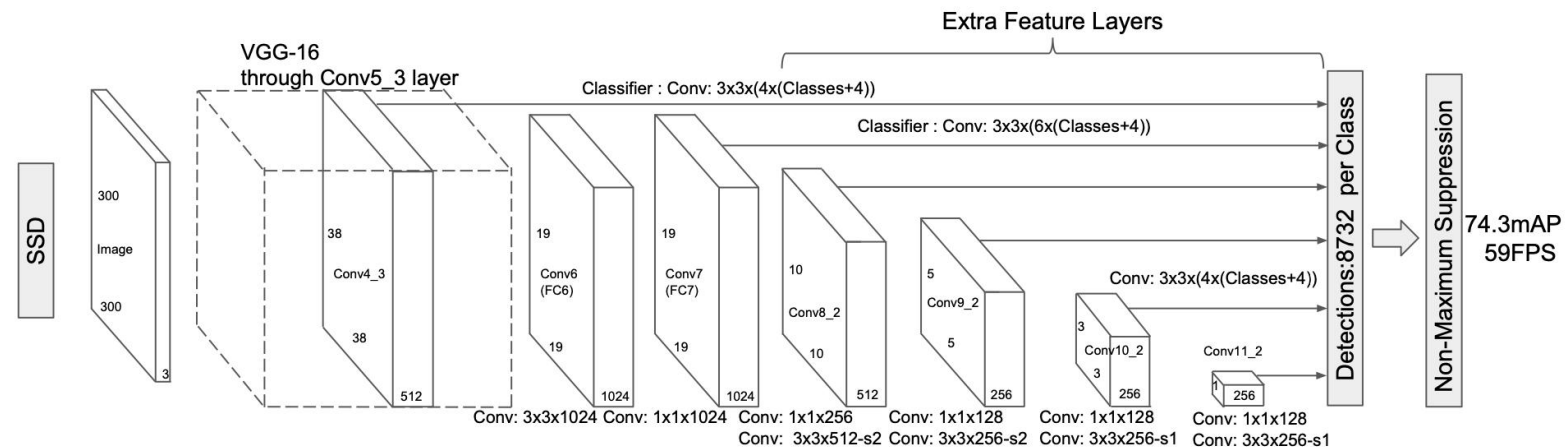
Особенности:

- применение NMS (Non Maximum Suppression)
- 45 fps, YoloTiny - 155 fps
- end-to-end training
- меньше false positive срабатываний (детектировали объект, но его там нет)
- высокоточен на изображениях различных областей (например, картины), чем Faster RCNN

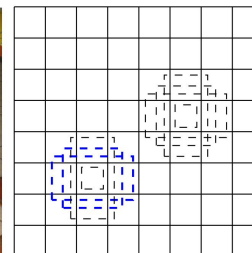
SSD - Single Shot Multibox Detector

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

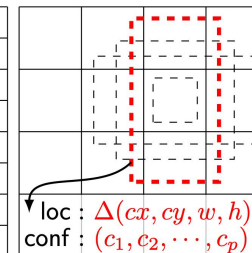
- Для каждой feature map (их несколько) кол-во_ячеек_в_карте * колчество default_anchor_boxes * (4 + количество_классов)



(a) Image with GT boxes



(b) 8×8 feature map



(c) 4×4 feature map

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:

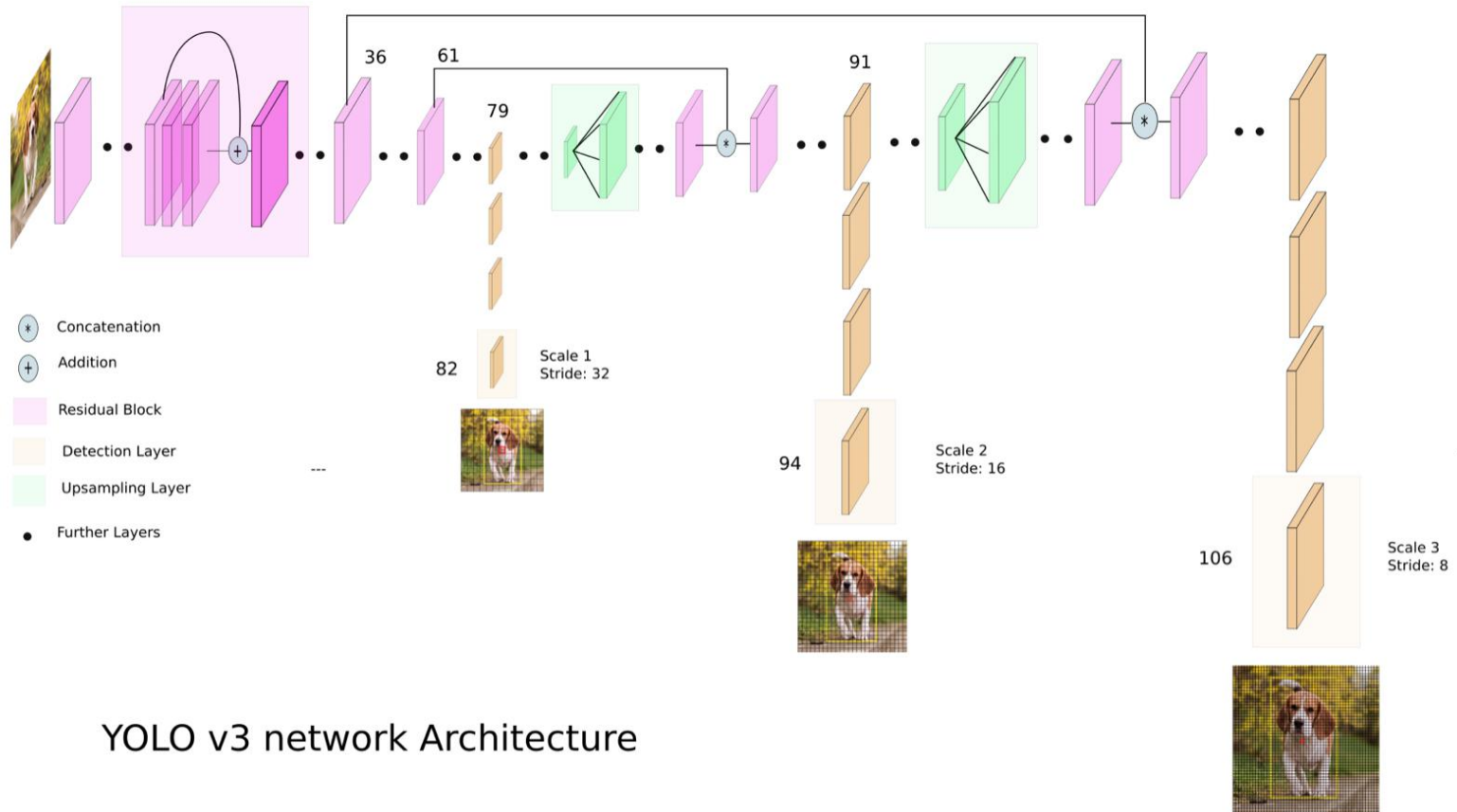
- добавлена batch normalization
- тренировались на увеличенном разрешении 2 раза изображений ImageNet 448x448
- задание anchor boxes -> предсказание больше 2-х объектов в ячейке сетки (default == 5)
- изменение лоса вычисления бокса (относительно начало сетки)
- определяет более мелкие объекты с помощью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

статья:

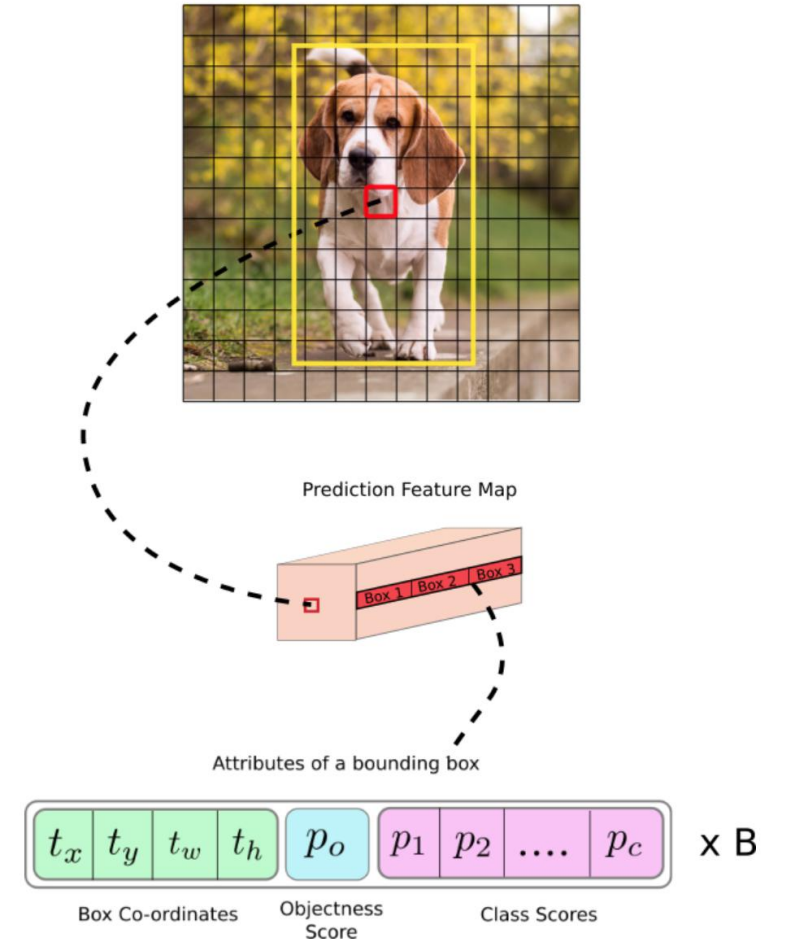
<https://arxiv.org/pdf/1612.08242.pdf>

	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

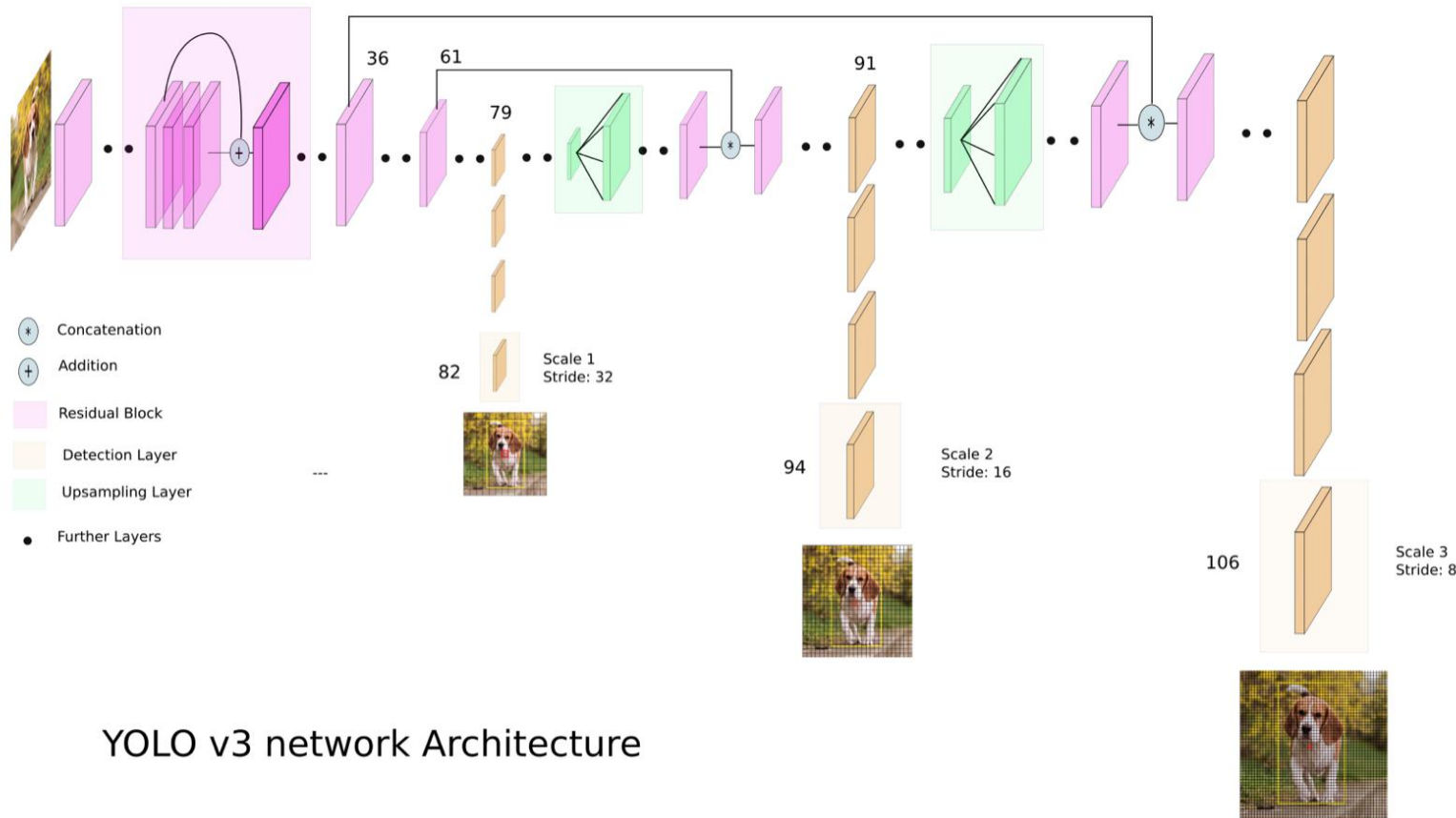
YOLO v3



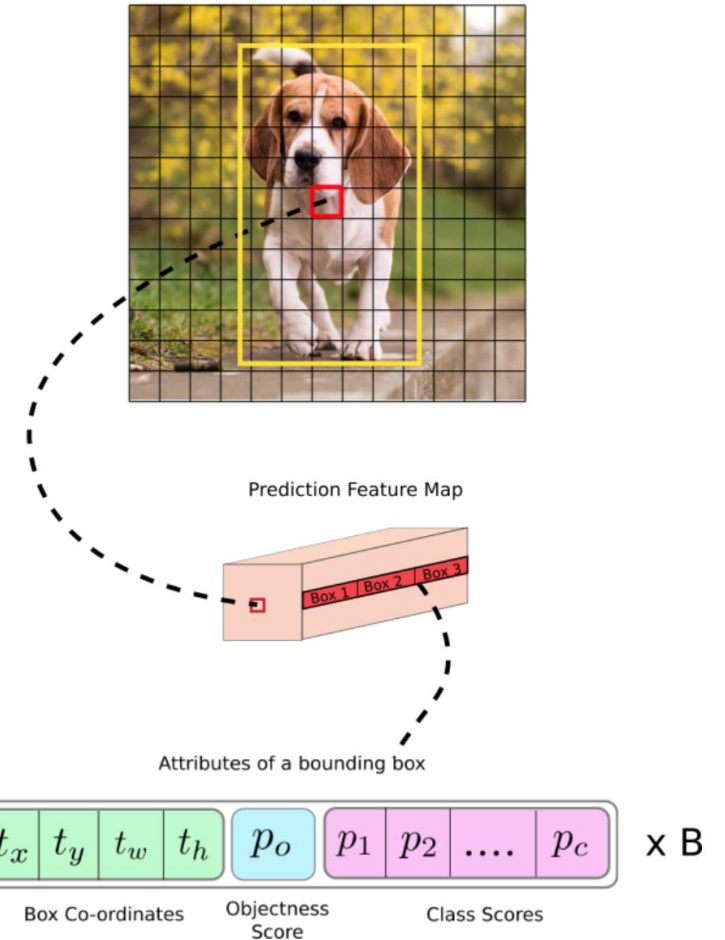
YOLO v3 network Architecture



YOLO v3



YOLO v3 network Architecture



На выходе тензор: $13 \times 13 \times (B \times (5 + C))$, где
 13×13 - это сетка на изображении,
 B - количество боксов в одной ячейке сетки;
 5 - это значения (t_x, t_y, t_w, t_h, p_0) бокса
 C - количество предсказанных классов

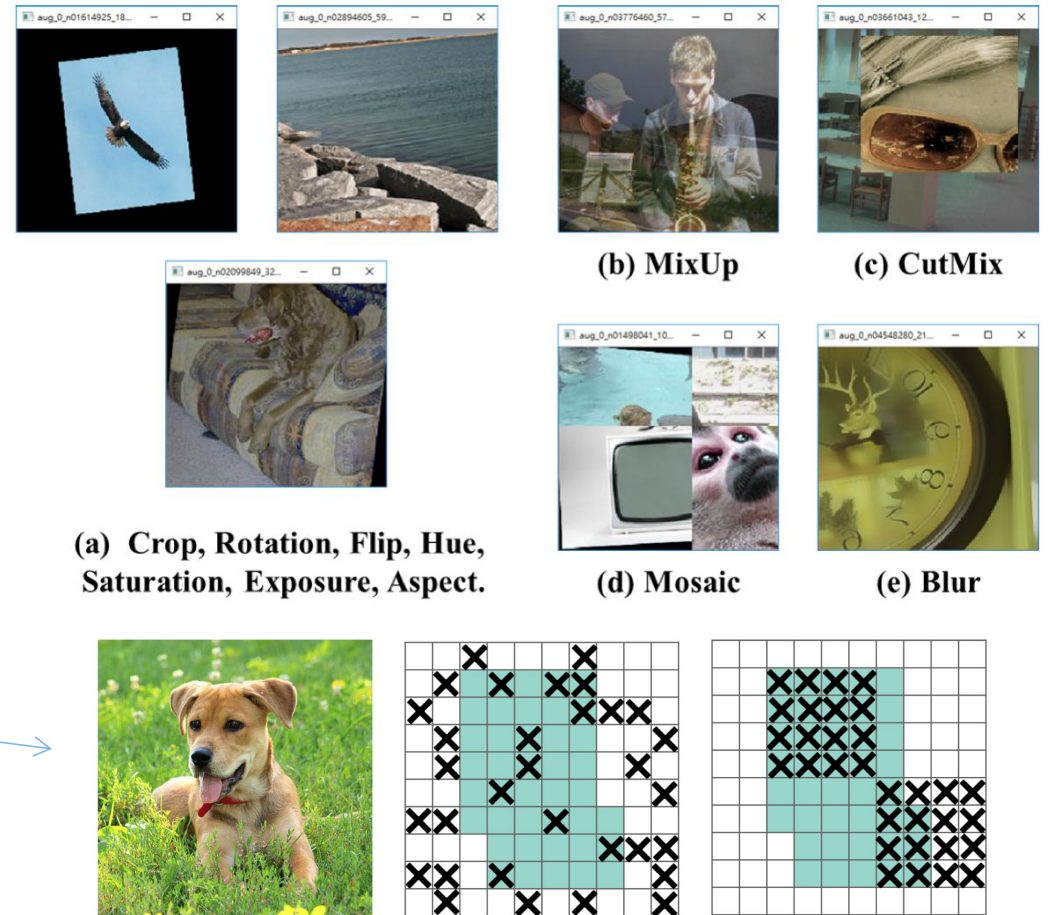
Total:
 $((52 \times 52) + (26 \times 26) + (13 \times 13)) \times 3 = 10647$ bboxes
 Потом фильтруем боксы:
 по порогу и NMS

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.

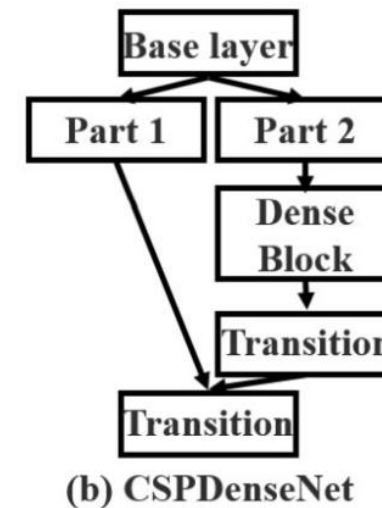
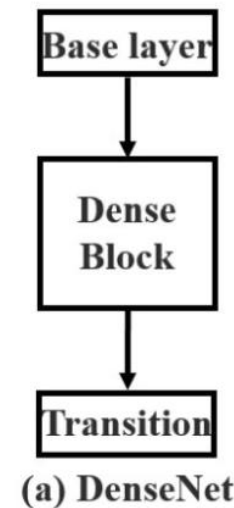
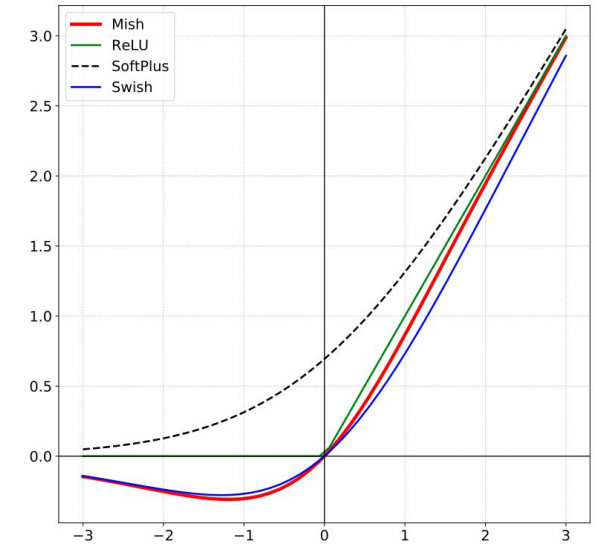
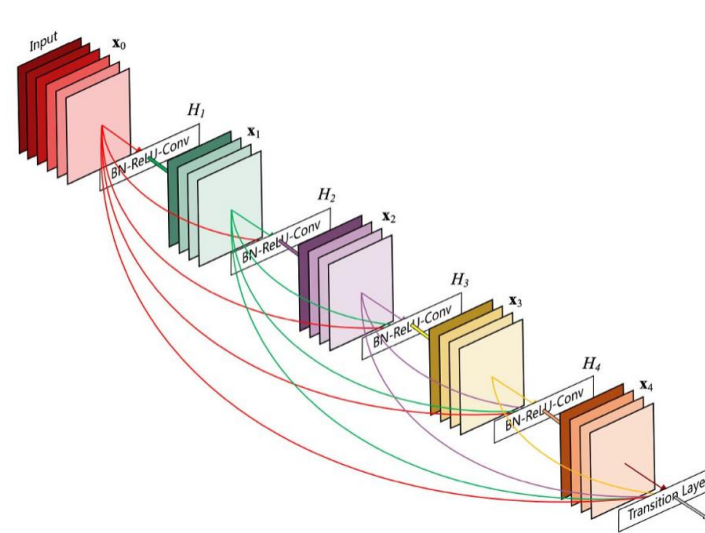


YOLO v4 (sota Apr 2020)

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



EfficientNet (2019 May)

Принцип - масштабирование сверточных сетей (с помощью AutoML):

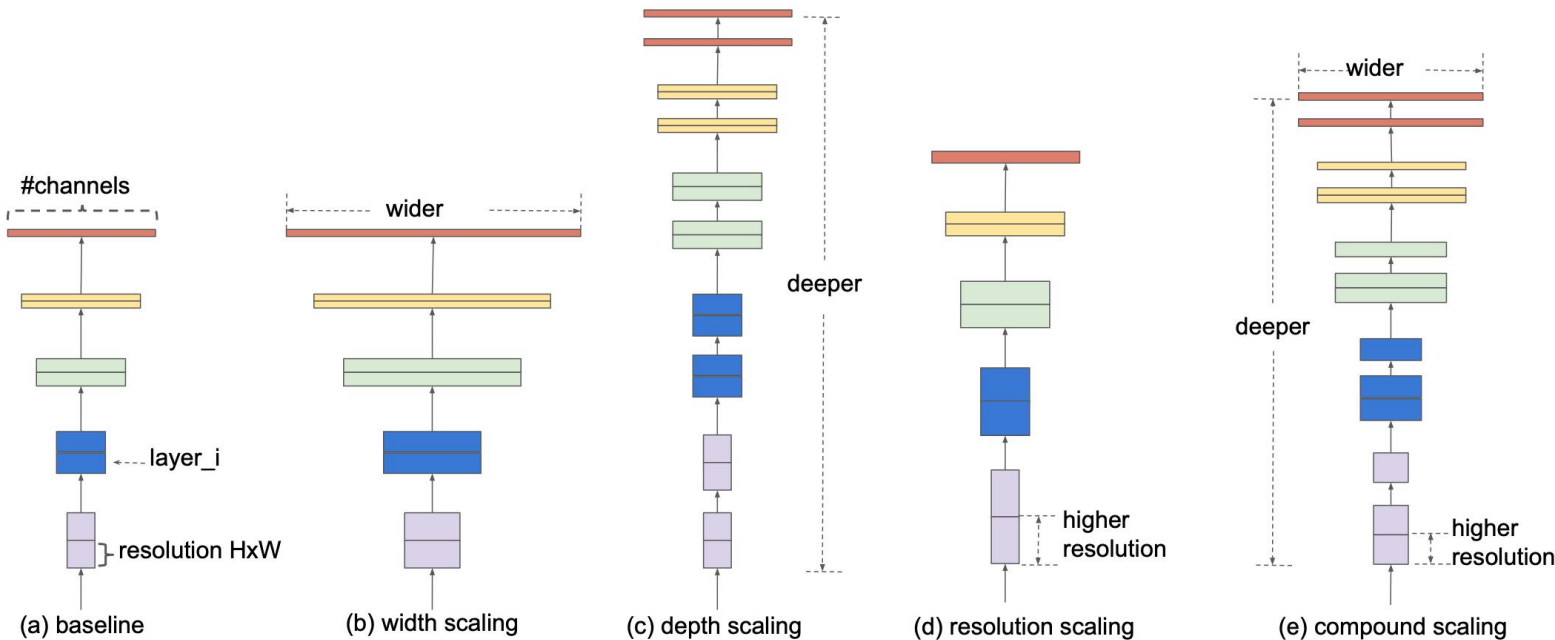
	Top1 Acc.	#Params
ResNet-152 (He et al., 2016)	77.8%	60M
EfficientNet-B1	79.1%	7.8M
ResNeXt-101 (Xie et al., 2017)	80.9%	84M
EfficientNet-B3	81.6%	12M
SENet (Hu et al., 2018)	82.7%	146M
NASNet-A (Zoph et al., 2018)	82.7%	89M
EfficientNet-B4	82.9%	19M
GPipe (Huang et al., 2018) [†]	84.3%	556M
EfficientNet-B7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



- масштабирование по ширине слоя(рис. а)/по глубине сети(увеличение слоев) (рис. с)/по глубине слоя (рис. d)
- комбинированное масштабирование (рис. е)



EfficientNet (2019 May)

Принцип - масштабирование сверточных сеток (с помощью AutoML):

	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*



- масштабирование по ширине слоя(рис. а)/по глубине сети(увеличение слоев) (рис. с)/по глубине слоя (рис. d)
- комбинированное масштабирование (рис. е)

Алгоритм:

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

$$\text{depth: } d = \alpha^\phi$$

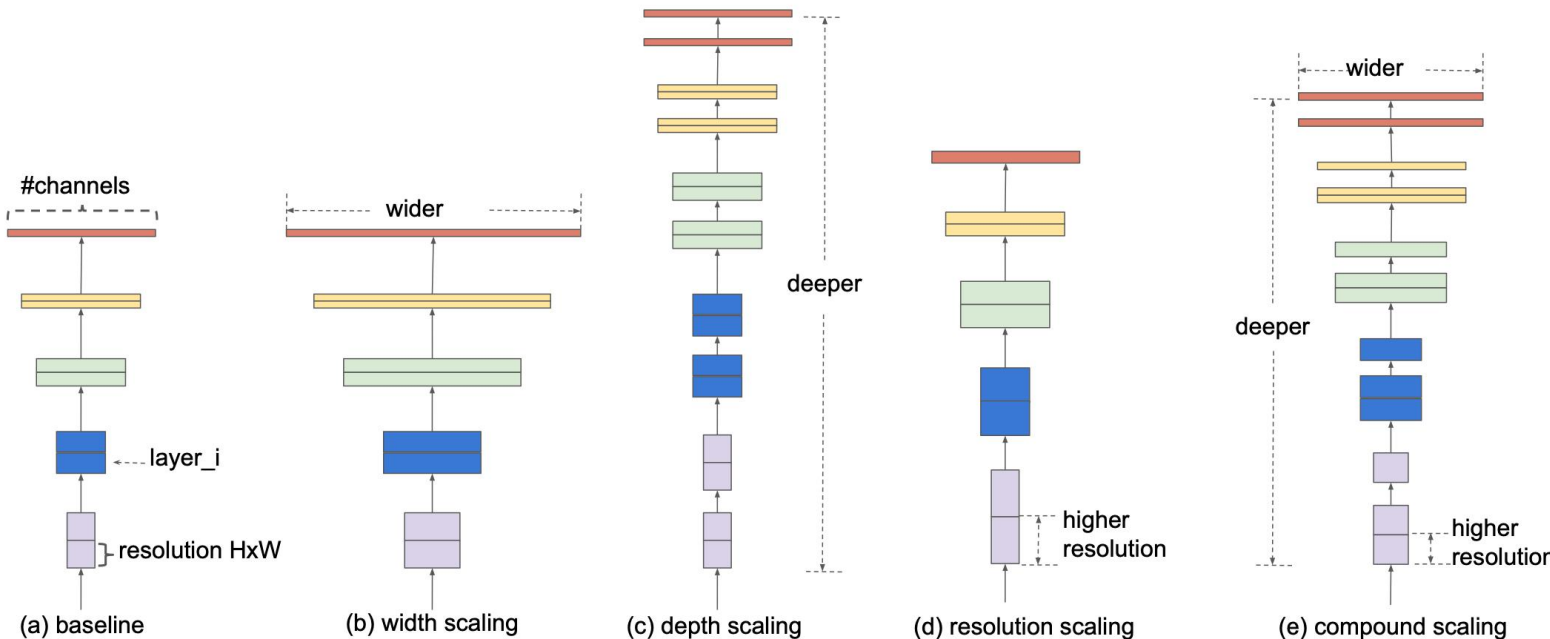
$$\text{width: } w = \beta^\phi$$

$$\text{resolution: } r = \gamma^\phi$$

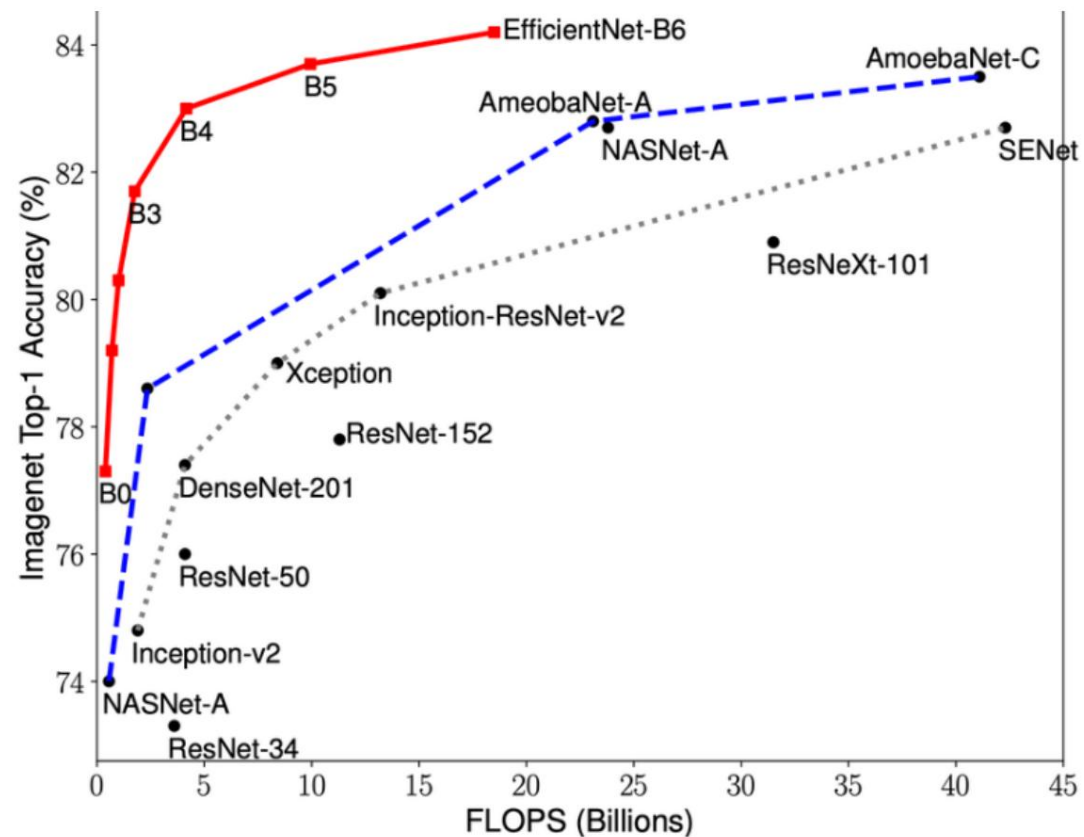
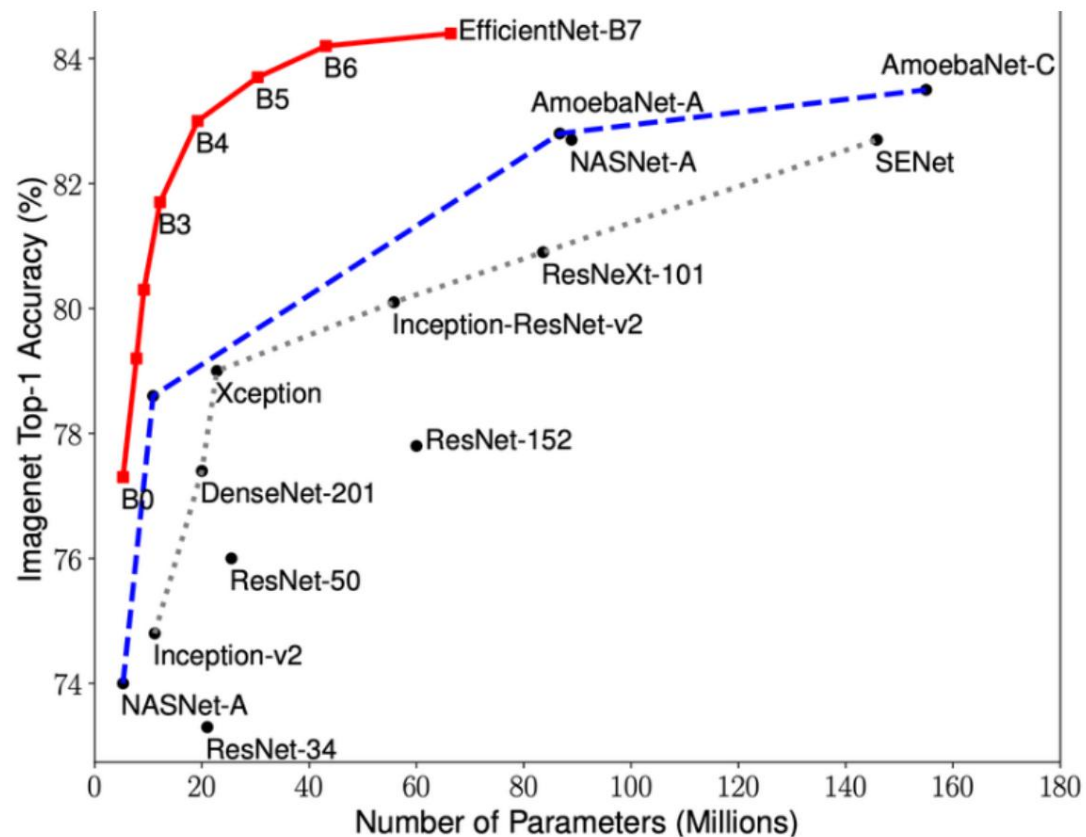
$$\text{s.t. } \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$$

$$\alpha \geq 1, \beta \geq 1, \gamma \geq 1$$

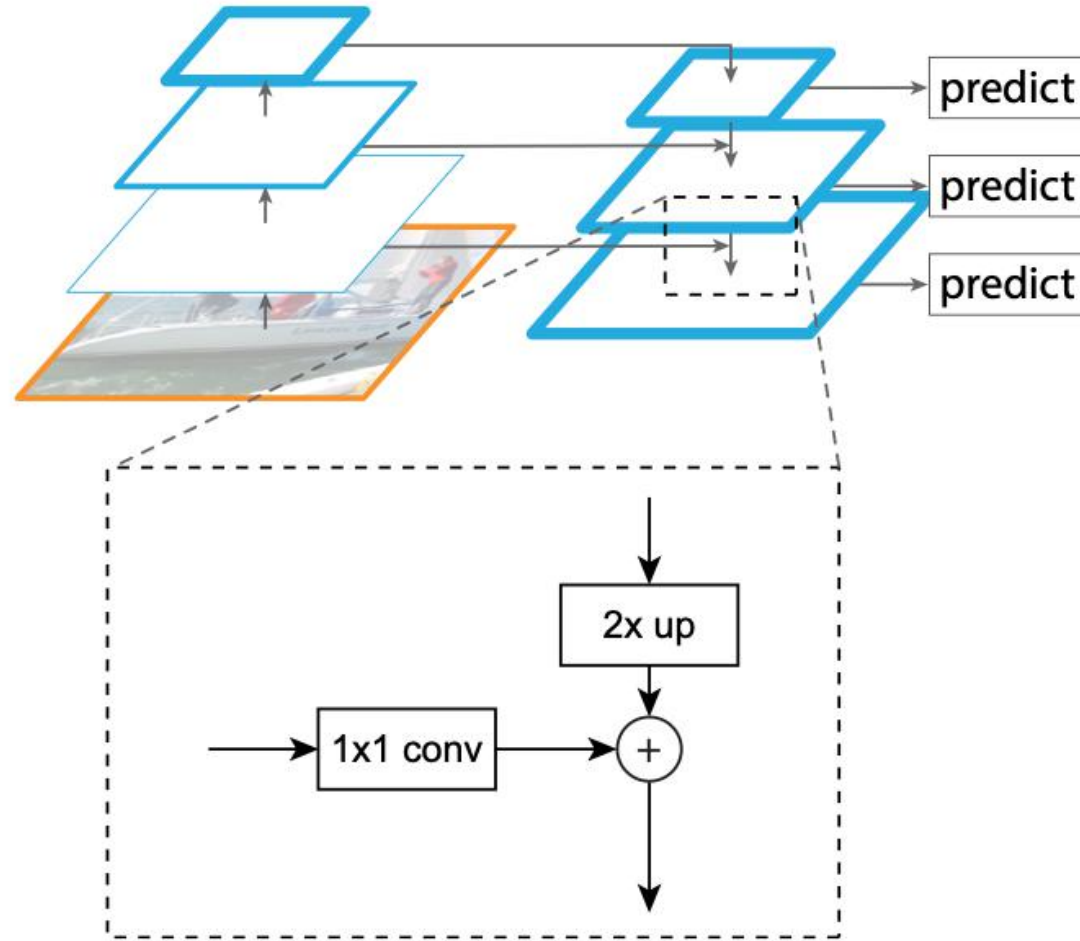
2. при зафиксированных alpha, beta, gamma масштабируется сеть увеличиваем ϕ на 1 ($\phi = 2, 3, 4 \dots 8$) на каждой итерации и получили сети EfficientNet-B1 до EfficientNet-B7



EfficientNet



Feature Pyramid Network (FPN)



PANet - Path Aggregation Network (2018 Jun)

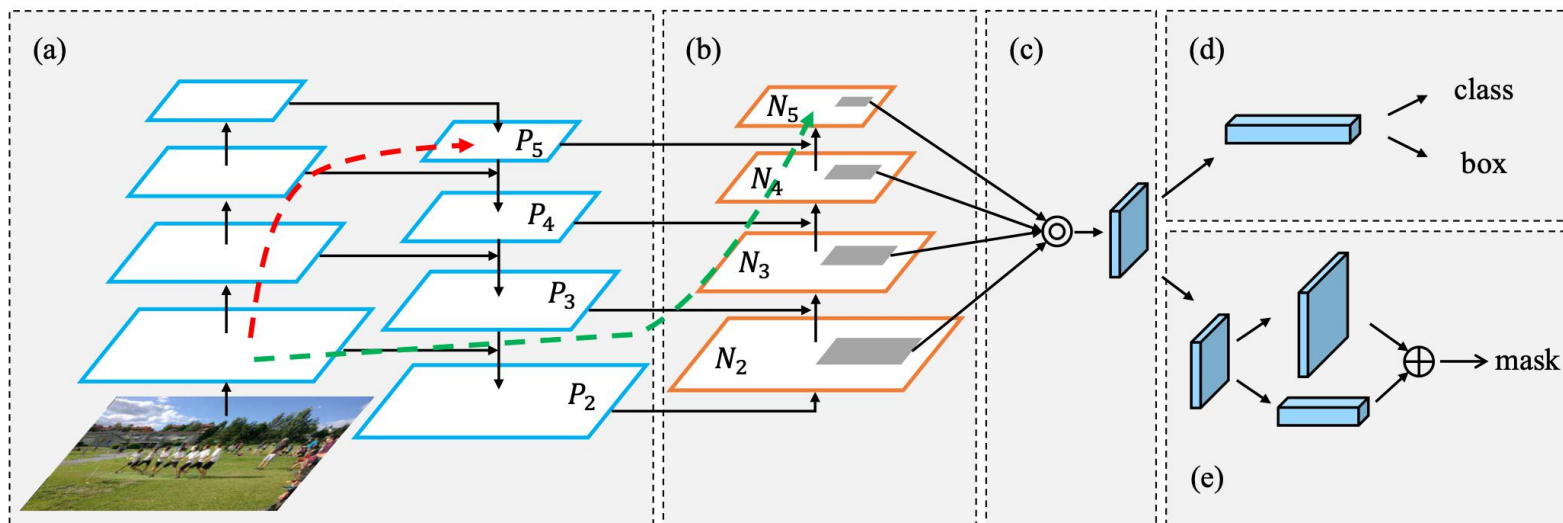
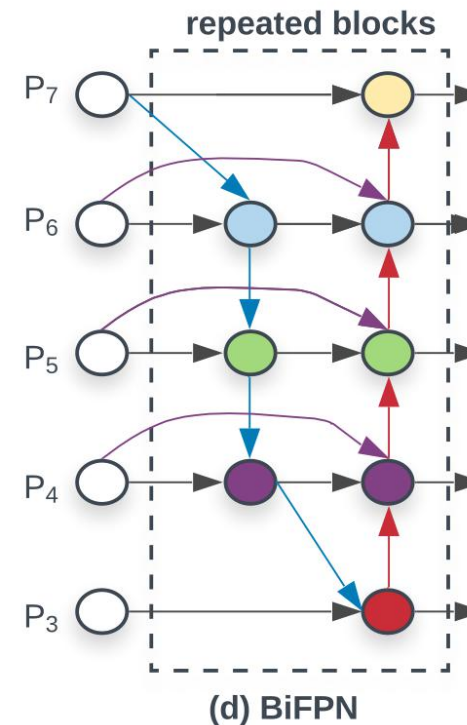
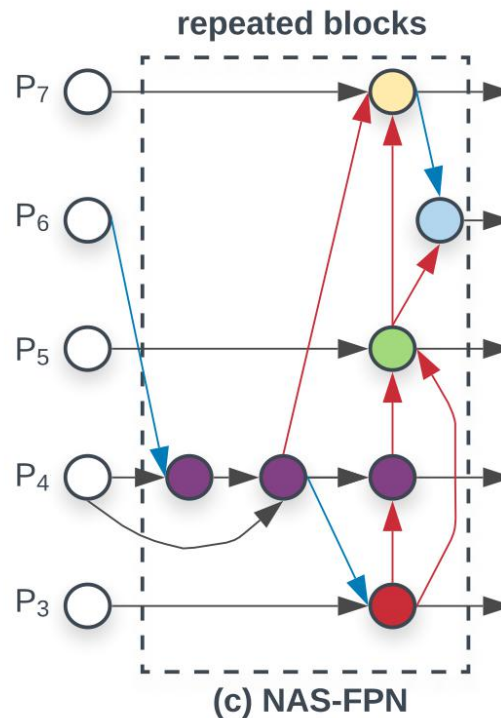
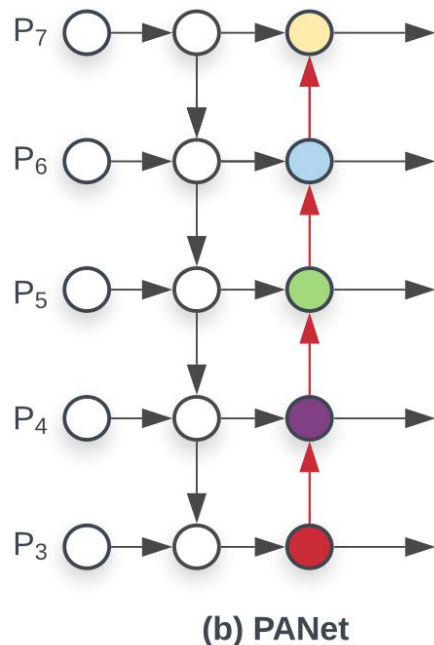
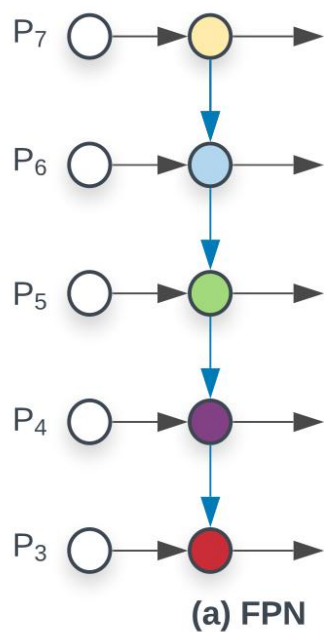


Figure 1. Illustration of our framework. (a) FPN backbone. (b) Bottom-up path augmentation. (c) Adaptive feature pooling. (d) Box branch. (e) Fully-connected fusion. Note that we omit channel dimension of feature maps in (a) and (b) for brevity.

- backbone - ResNetXt101
- Task - Instance Segmentation
- FPN + bottom-top path
- ROIAlign with (element-wise max or sum)
- достигает 47.4 AP

Method	AP^{bb}	AP_{50}^{bb}	AP_{75}^{bb}
Champion 2016 [27]	41.6	62.3	45.6
RentinaNet [36]	39.1	59.1	42.3
Mask R-CNN [21] + FPN [35]	38.2	60.3	41.7
Mask R-CNN [21] + FPN [35]	39.8	62.3	43.4
PANet / PANet [ms-train]	41.2 / 42.5	60.4 / 62.3	44.4 / 46.4
PANet / PANet [ms-train]	45.0 / 47.4	65.0 / 67.2	48.6 / 51.8

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

$$P_7^{out} = Conv(P_7^{in})$$

$$P_6^{out} = Conv(P_6^{in} + Resize(P_7^{out}))$$

...

$$P_3^{out} = Conv(P_3^{in} + Resize(P_4^{out}))$$

EfficientDet (2020)

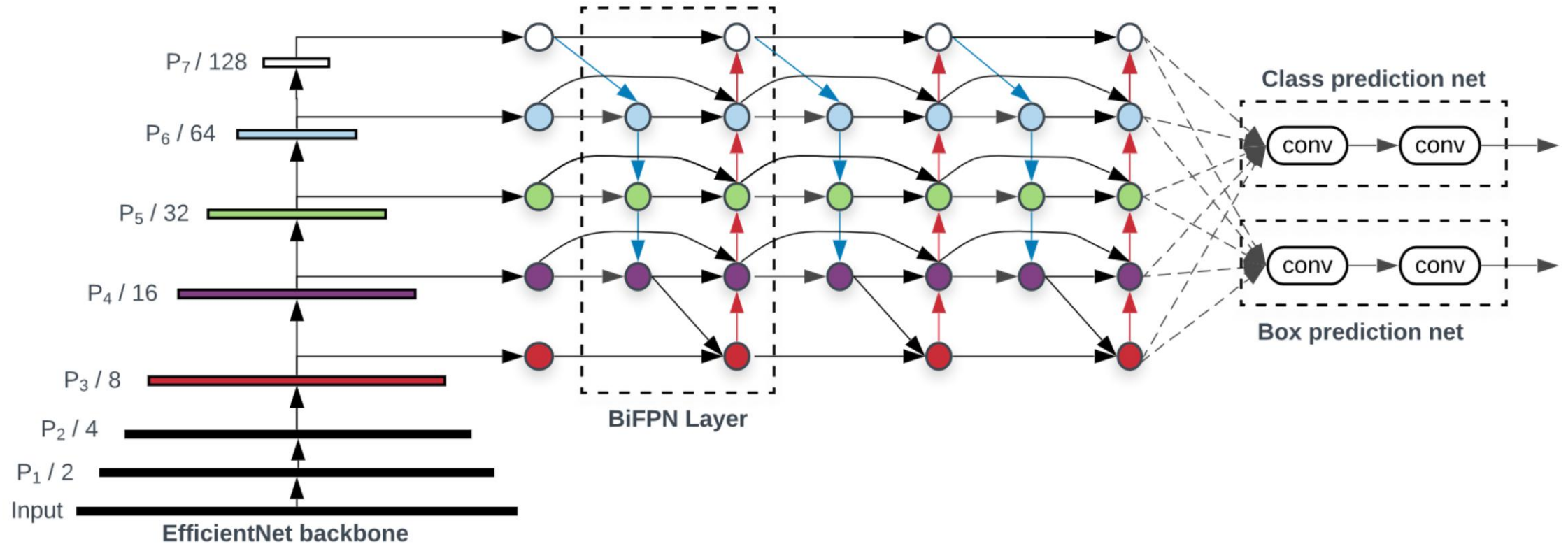
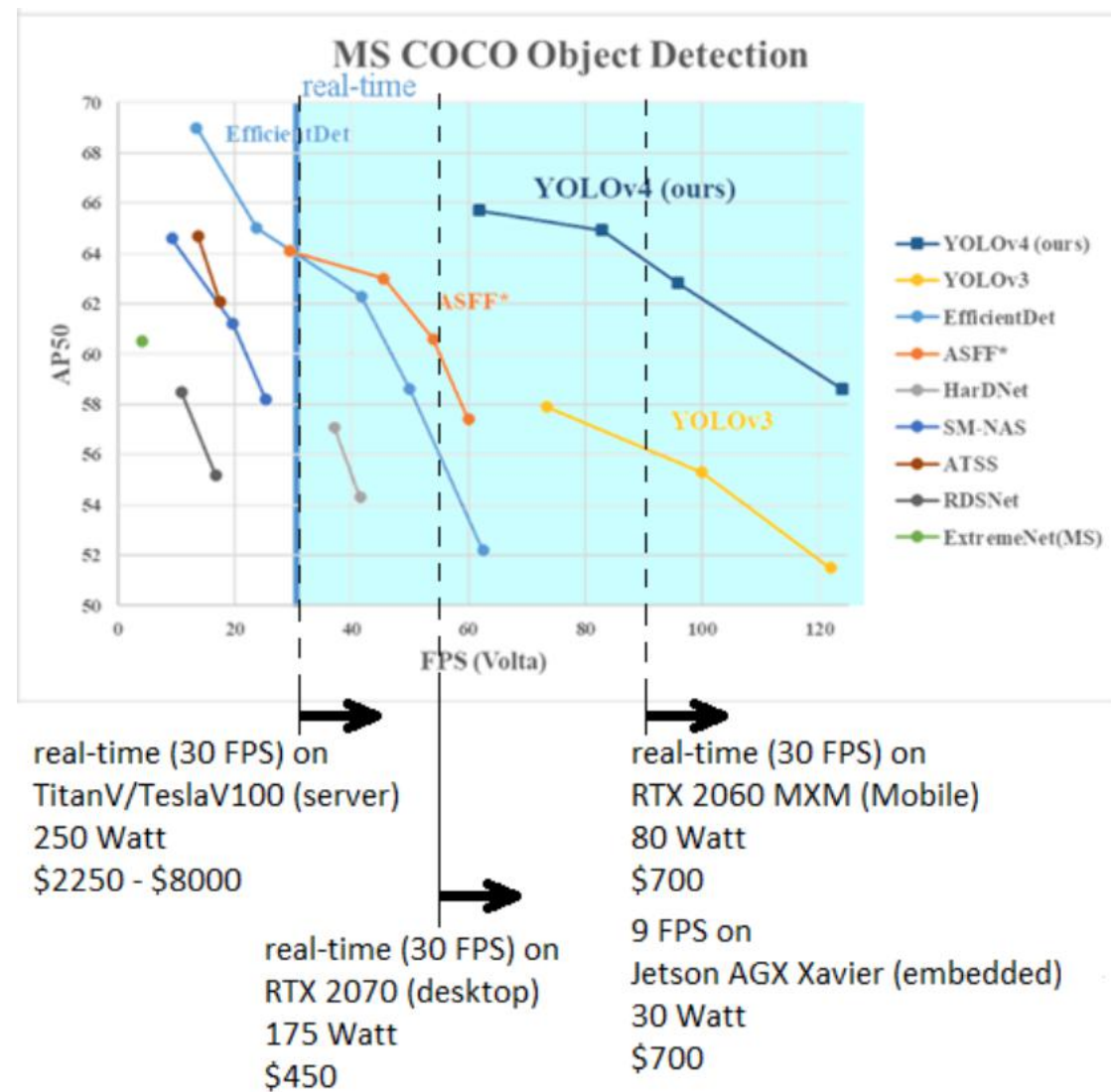
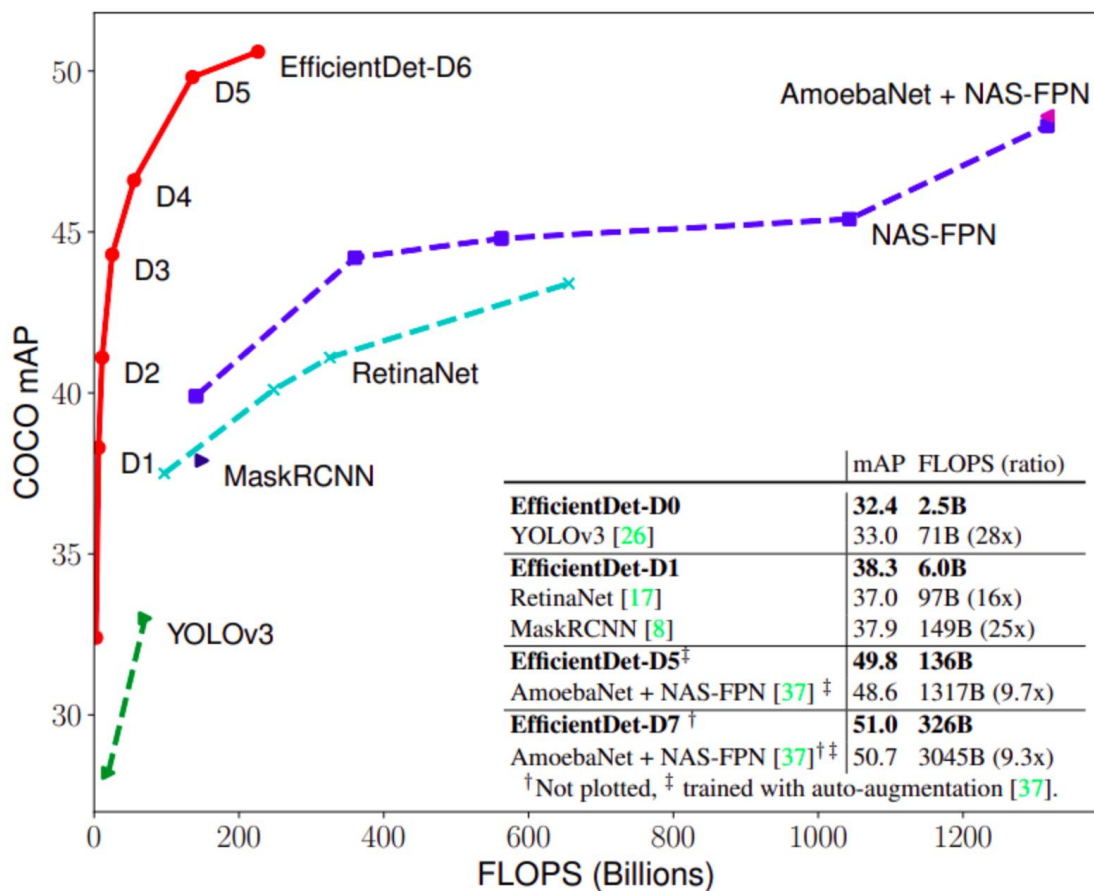


Рисунок — Архитектура EfficientDet == EfficientNet + BiFPN + сеть вычисления класс/рамка

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

Сравнение



Сравнение

Эксперименты EfficientDet на Titan RTX
(Watt 250, GFLOPS half precision = 24 883.20)

1) с постобработкой (top-k, nms, conversion to coco fmt)

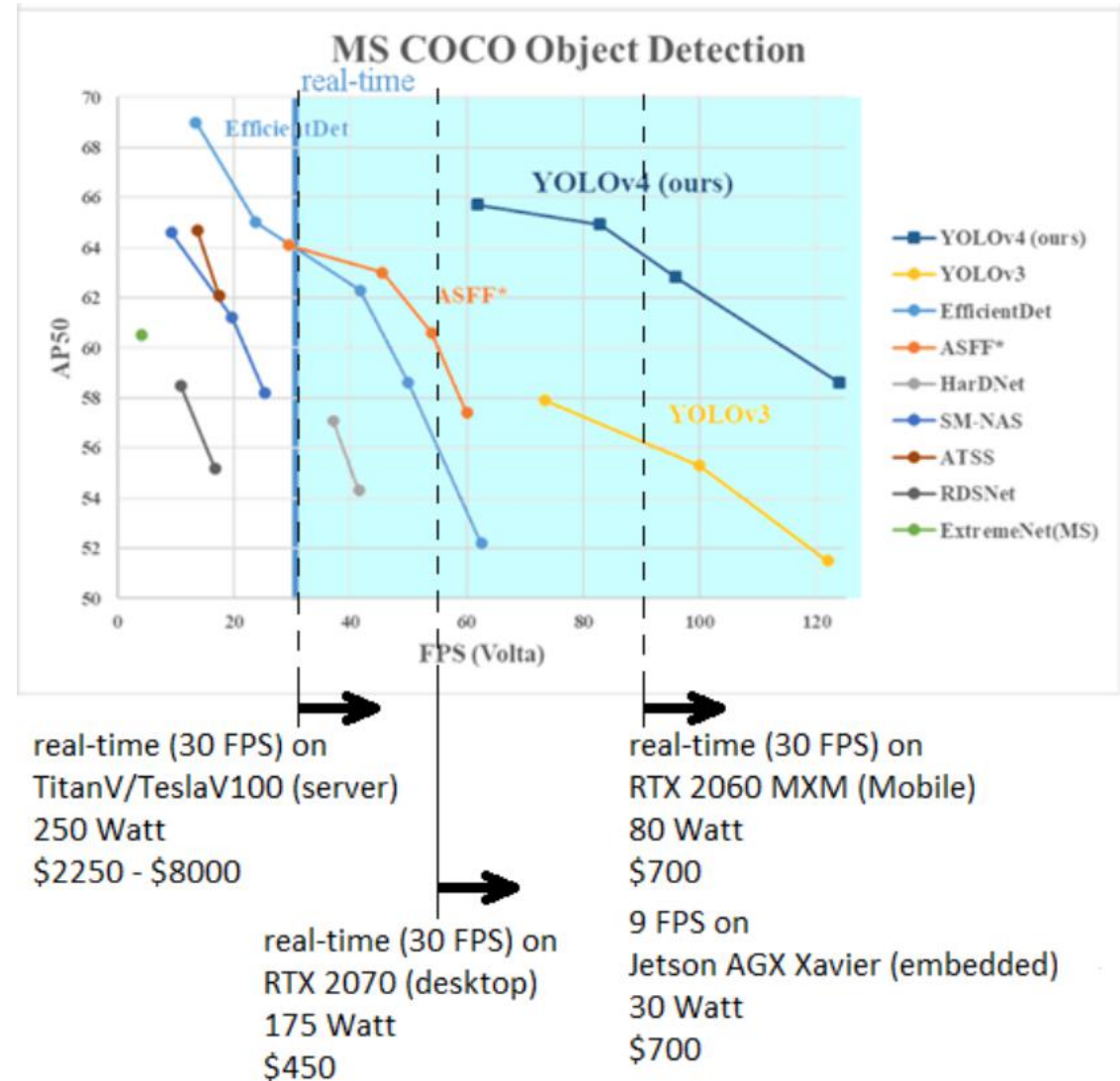
batch_size=128, FPS=200

batch_size=1, FPS=16

2) без постобработки

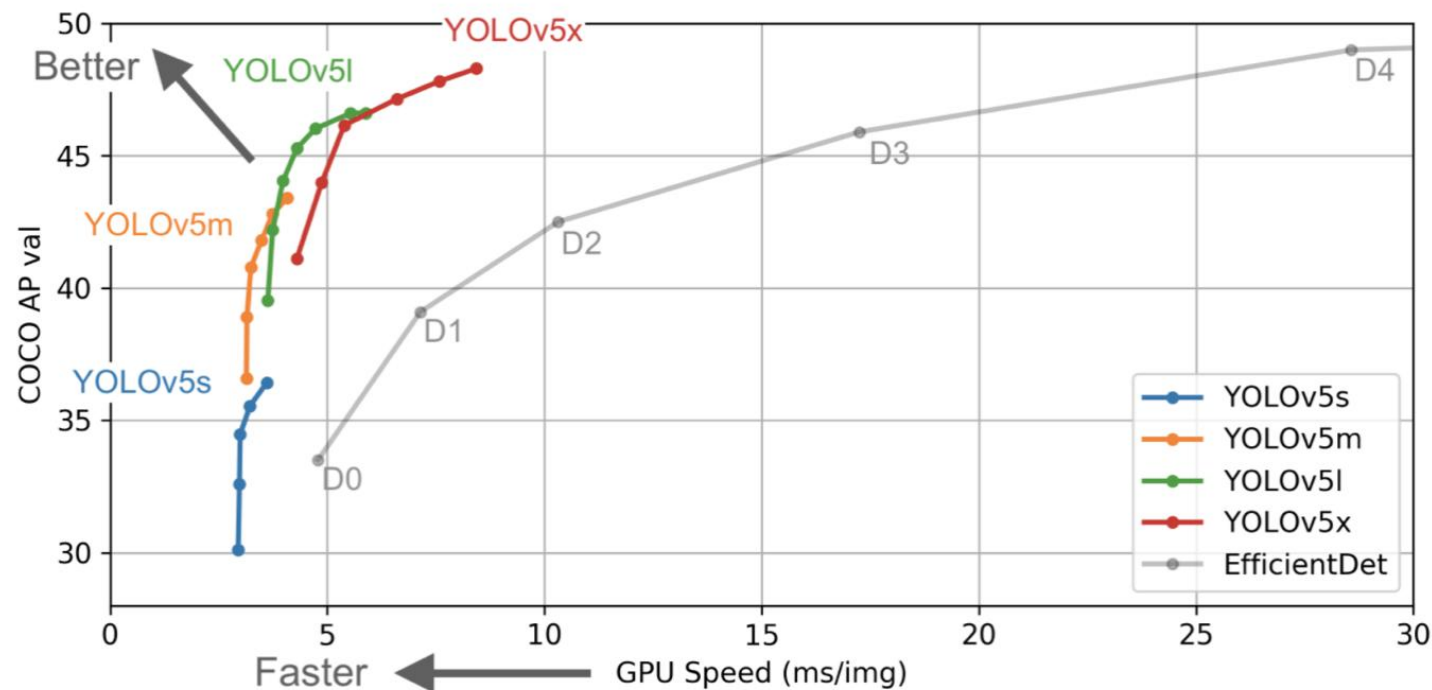
batch_size=128, FPS=480

batch_size=1, FPS=27



YOLOv5 (2020 June)

- теперь реализация на PyTorch
- квантизация 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