

# Распознавание объектов II

---

Сергей Николенко

НИУ ВШЭ – Санкт-Петербург

24 октября 2020 г.

---

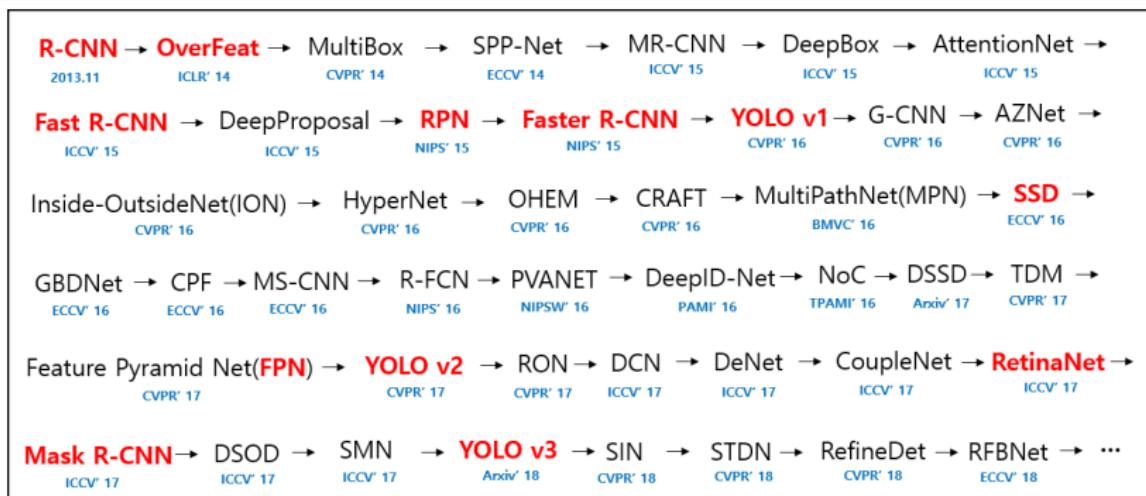
*Random facts:*

- 24 октября в ООН – День ООН; именно 24 октября 1945 г. официально вступил в силу устав организации
- 24 октября 1745 г. Елизавета Петровна повелела завезти в царские дворцы котов для ловли мышей
- 24 октября 1857 г. выпускники Кембриджа основали в Шеффилде первую в мире футбольную команду, Sheffield F.C., а 24 октября 1897 г. в Санкт-Петербурге был проведён первый в истории России официально зафиксированный футбольный матч
- 24 октября 1911 г. с мыса Эванс к Южному полюсу отправился Роберт Скотт и ещё 11 человек; обратно не вернулся никто
- 24 октября 1929 г. индекс Доу-Джонса за один день снизился на 11%, а в целом за неделю – более чем на 40%; «чёрный четверг» стал началом Великой депрессии

# Распознавание объектов

---

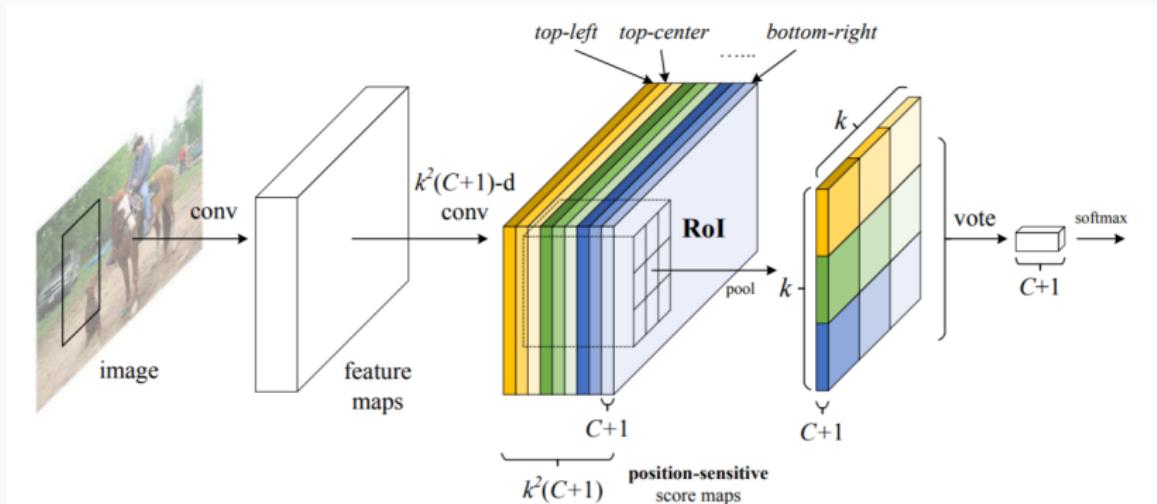
# Plan



- (Dai et al., 2016), R-FCN (Region-based Fully Convolutional Network)
- Можно ещё больше сэкономить в Faster R-CNN, не вычисляя на каждом участке вообще никаких сложных слоёв.
- Но как? Идея такая:
  - лучшие CNN для классификации (ResNet, GoogLeNet) давно fully convolutional, т.е. полносвязный слой только в конце
  - но с object detection почему-то это не работает
  - Faster R-CNN пришлось вставить противовесственный region proposal слой
  - проблема в том, что CNN дают translation invariance, а для object detection этого как раз не надо

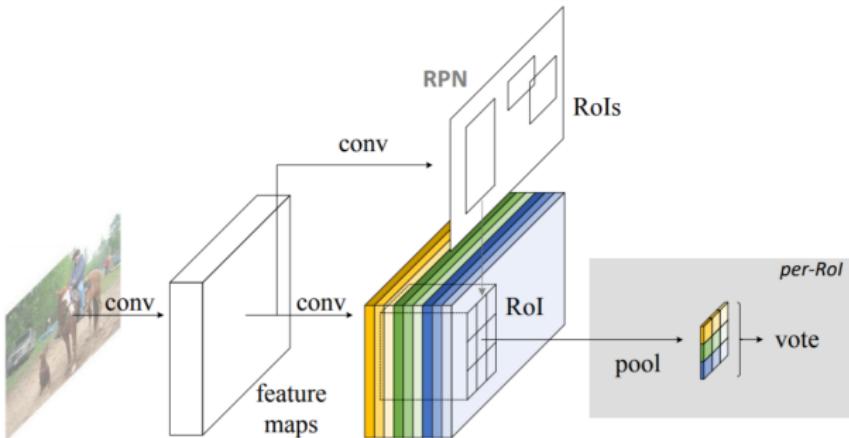
# R-FCN

- Решение R-FCN в том, чтобы сделать специальные карты признаков, которые как раз position-sensitive:

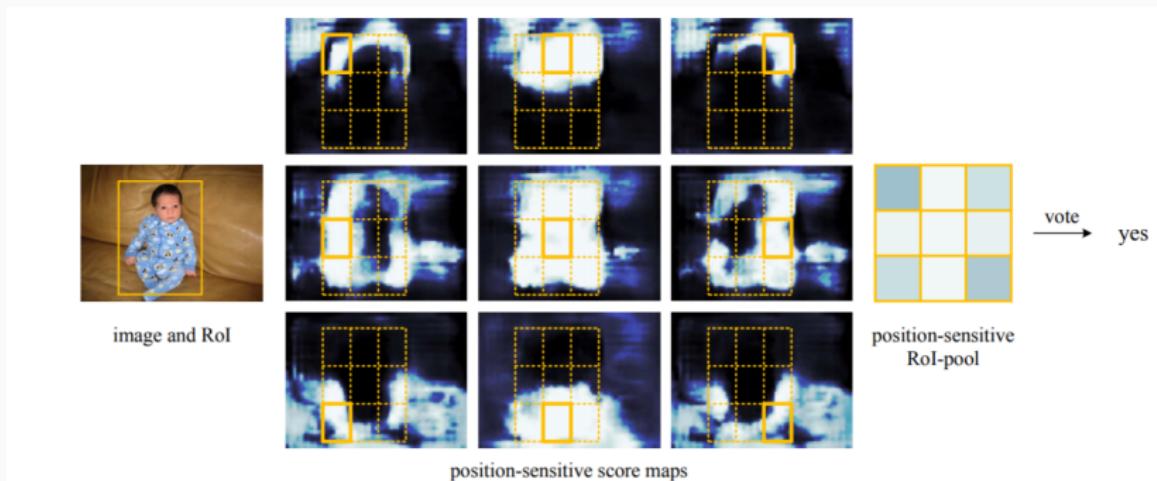


- Пытаемся их обучить так, чтобы они кодировали информацию относительно конкретной позиции («левая часть объекта», «правая часть объекта»)

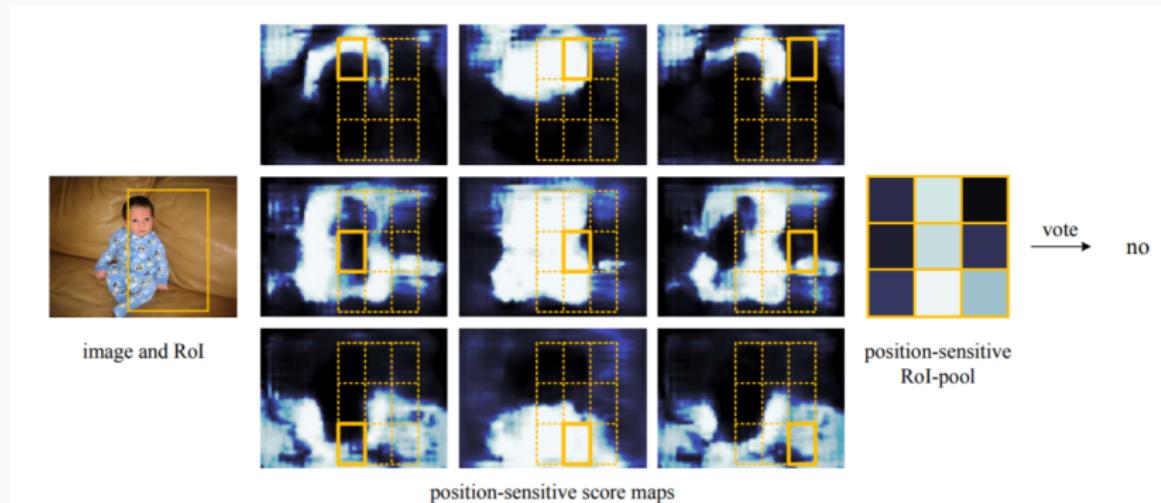
- В остальном как Faster R-CNN – есть отдельная RPN
- А потом RoI pooling, который учитывает наши новые feature maps:



- Визуализация R-FCN активаций для  $3 \times 3$  position-sensitive карт признаков:



- А вот что будет для неправильного окна:



- Это уже топовая архитектура и по сегодняшним меркам:



# R-FCN

- Циферки тоже отличные:

	depth of per-RoI subnetwork	training w/ OHEM?	train time (sec/img)	test time (sec/img)	mAP (%) on VOC07
Faster R-CNN <b>R-FCN</b>	10		1.2	0.42	76.4
	0		0.45	0.17	76.6
Faster R-CNN <b>R-FCN</b>	10	✓ (300 RoIs)	1.5	0.42	79.3
	0	✓ (300 RoIs)	0.45	0.17	<b>79.5</b>
Faster R-CNN <b>R-FCN</b>	10	✓ (2000 RoIs)	2.9	0.42	<i>N/A</i>
	0	✓ (2000 RoIs)	0.46	0.17	79.3

	training data	mAP (%)	test time (sec/img)
Faster R-CNN [9]	07+12	76.4	0.42
Faster R-CNN +++ [9]	07+12+COCO	<b>85.6</b>	3.36
<b>R-FCN</b>	07+12	79.5	0.17
<b>R-FCN</b> multi-sc train	07+12	80.5	0.17
<b>R-FCN</b> multi-sc train	07+12+COCO	<b>83.6</b>	0.17

	training data	mAP (%)	test time (sec/img)
Faster R-CNN [9]	07++12	73.8	0.42
Faster R-CNN +++ [9]	07++12+COCO	<b>83.8</b>	3.36
<b>R-FCN</b> multi-sc train	07++12	77.6 <sup>†</sup>	0.17
<b>R-FCN</b> multi-sc train	07++12+COCO	<b>82.0<sup>‡</sup></b>	0.17

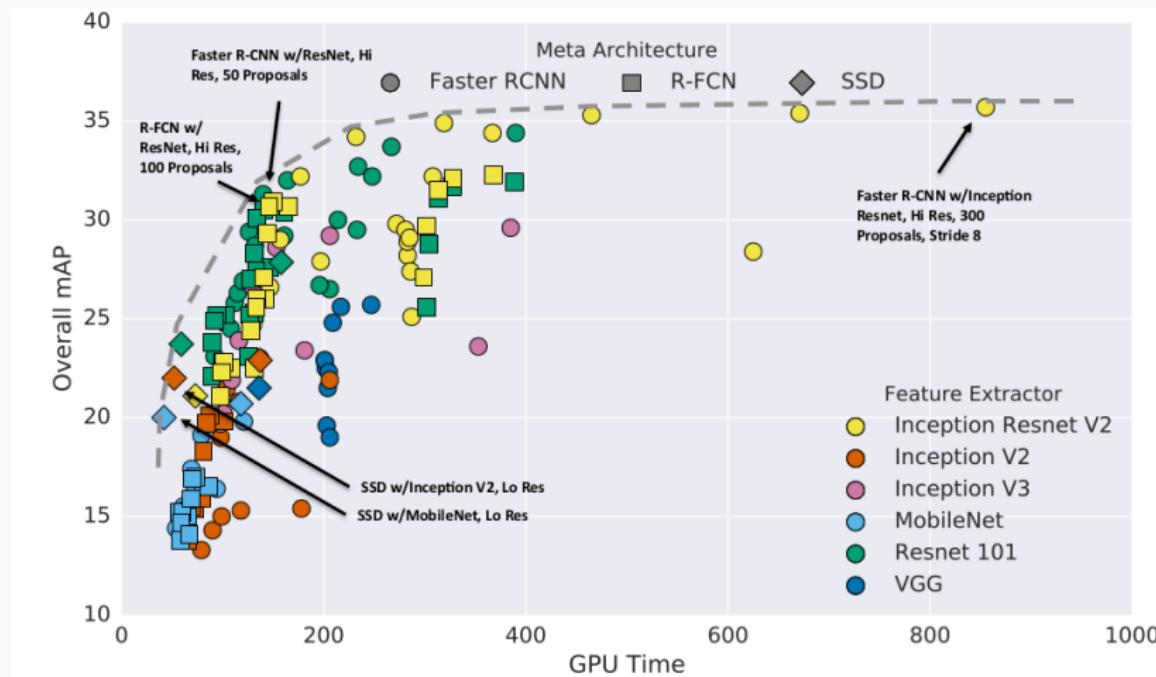
## Что же лучше?

- Обзор (Huang et al., 2017) сравнивает качество и скорость.
- Базовые архитектуры (извлекатели признаков) с качеством на ImageNet:

Model	Top-1 accuracy	Num. Params.
VGG-16	71.0	14,714,688
MobileNet	71.1	3,191,072
Inception V2	73.9	10,173,112
ResNet-101	76.4	42,605,504
Inception V3	78.0	21,802,784
Inception Resnet V2	80.4	54,336,736

# Что же лучше?

- Общее сравнение:



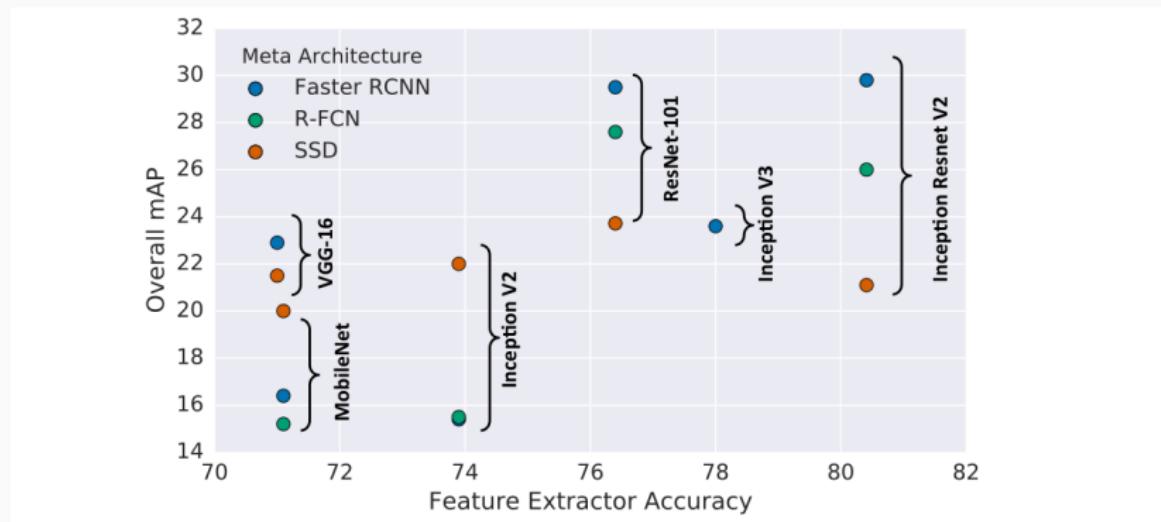
# Что же лучше?

- Лучшие варианты:

Model summary	minival mAP	test-dev mAP
(Fastest) SSD w/MobileNet (Low Resolution)	19.3	18.8
(Fastest) SSD w/Inception V2 (Low Resolution)	22	21.6
(Sweet Spot) Faster R-CNN w/Resnet 101, 100 Proposals	32	31.9
(Sweet Spot) R-FCN w/Resnet 101, 300 Proposals	30.4	30.3
(Most Accurate) Faster R-CNN w/Inception Resnet V2, 300 Proposals	35.7	35.6

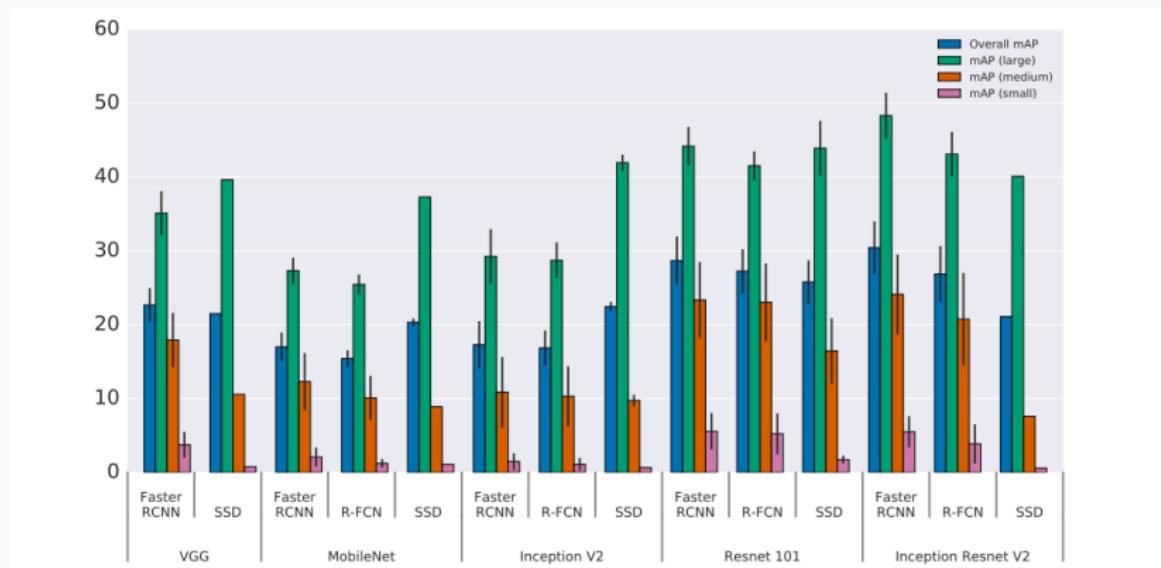
# Что же лучше?

- Архитектура CNN имеет значение, но не абсолютное:



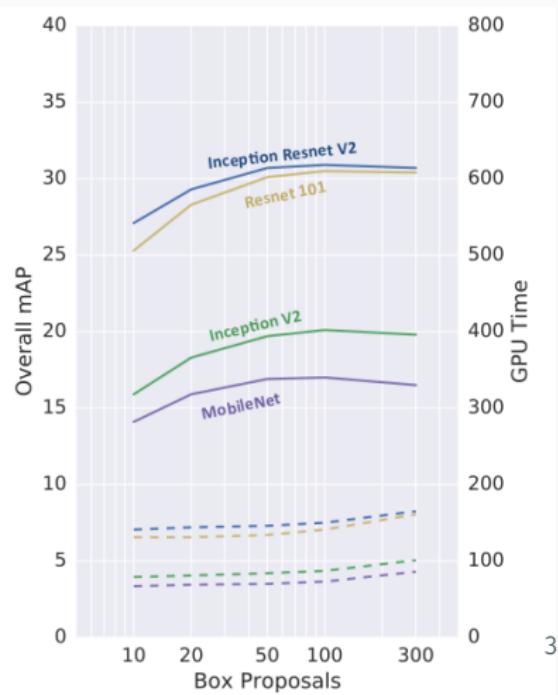
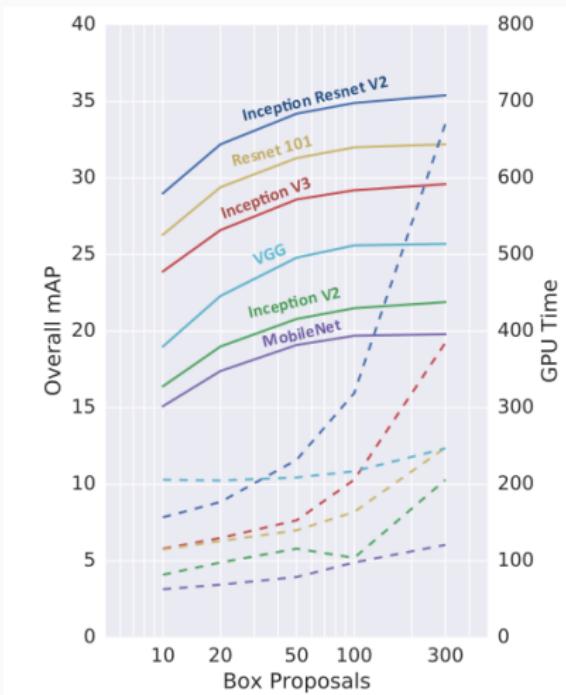
# Что же лучше?

- Размер распознаваемого объекта имеет значение:



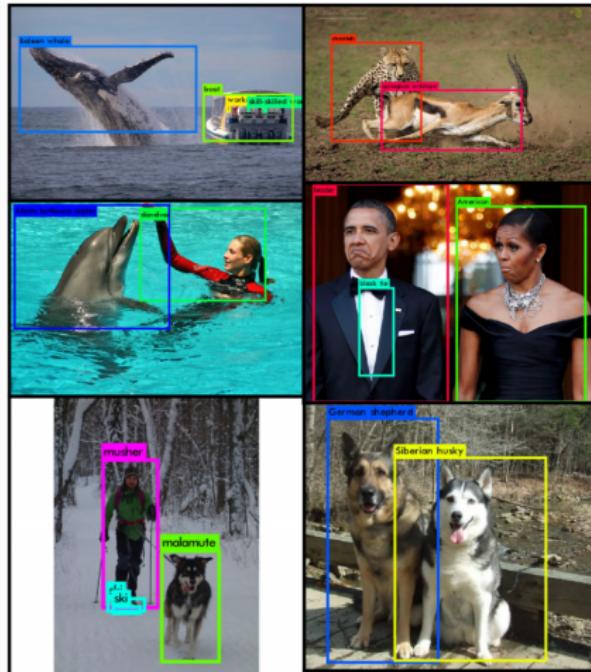
# Что же лучше?

- Можно сильно сократить время обработки, уменьшая число вариантов участков в Faster R-CNN, и это не очень дорого стоит в точности:



# YoLo v2, YOLO9000, YoLo v3

- (Redmon, Farhadi, 2016): следующая итерация YoLo, гораздо больше классов (9000):



## YoLo v2, YOLO9000, YoLo v3

- Что было не так: много ошибок локализации и низкий recall по сравнению с Faster R-CNN
- Что сделали:
  - добавили batchnorm на всех свёрточных уровнях, +2% mAP сразу
  - сделали отдельные классификатор высокого разрешения ( $448 \times 448$ ) вместо предобученного на ImageNet ( $256 \times 256$ ), ещё +4% mAP
  - YoLo предсказывал координаты bbox'ов, а лучше предсказывать offsets в каждой позиции свёрточным образом; класс теперь для каждого anchor box тоже предсказывается
  - размеры anchor boxes выбраны по bbox'ам из тренировочного набора, кластеризацией
  - добавили skip connection признаков с более раннего слоя, чтобы увеличить геометрическое разрешение
  - multi-scale training: обучаемся на разных размерах картинок

## YoLo v2, YOLO9000, YoLo v3

- Получается очень круто:

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	<b>78.6</b>	40

# YoLo v2, YOLO9000, YoLo v3

- И всё это реально помогает:

	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	<b>78.6</b>

- По категориям:

Method	data	mAP	aero	bike	bird	boat	bottle	bus	car	cat	chair	cow	table	dog	horse	mbike	person	plant	sheep	sofa	train	tv
Fast R-CNN [5]	07++12	68.4	82.3	78.4	70.8	52.3	38.7	77.8	71.6	89.3	44.2	73.0	55.0	87.5	80.5	80.8	72.0	35.1	68.3	65.7	80.4	64.2
Faster R-CNN [15]	07++12	70.4	84.9	79.8	74.3	53.9	49.8	77.5	75.9	88.5	45.6	77.1	55.3	86.9	81.7	80.9	79.6	40.1	72.6	60.9	81.2	61.5
YOLO [14]	07++12	57.9	77.0	67.2	57.7	38.3	22.7	68.3	55.9	81.4	36.2	60.8	48.5	77.2	72.3	71.3	63.5	28.9	52.2	54.8	73.9	50.8
SSD300 [11]	07++12	72.4	85.6	80.1	70.5	57.6	46.2	79.4	76.1	89.2	53.0	77.0	60.8	87.0	83.1	82.3	79.4	45.9	75.9	69.5	81.9	67.5
SSD512 [11]	07++12	74.9	87.4	82.3	75.8	59.0	52.6	81.7	81.5	90.0	55.4	79.0	59.8	88.4	84.3	84.7	83.3	50.2	78.0	66.3	86.3	72.0
ResNet [6]	07++12	73.8	86.5	81.6	77.2	58.0	51.0	78.6	76.6	93.2	48.6	80.4	59.0	92.1	85.3	84.8	80.7	48.1	77.3	66.5	84.7	65.6
YOLOv2 544	07++12	73.4	86.3	82.0	74.8	59.2	51.8	79.8	76.5	90.6	52.1	78.2	58.5	89.3	82.5	83.4	81.3	49.1	77.2	62.4	83.8	68.7

## YoLo v2, YOLO9000, YoLo v3

- А как сделать много классов? Новые проблемы: например, классы больше не взаимоисключающие («собака» и сотни пород собак)
- ImageNet берёт классы из WordNet – сети понятий, в которой есть иерархия, в виде направленного графа
- В YOLO сначала сделали из WordNet дерево, WordTree, а потом предсказывают условные вероятности на каждом уровне:

$$Pr(\text{Norfolk terrier}|\text{terrier})$$

$$Pr(\text{Yorkshire terrier}|\text{terrier})$$

$$Pr(\text{Bedlington terrier}|\text{terrier})$$

...

И ПОТОМ

$$Pr(\text{Norfolk terrier}) = Pr(\text{Norfolk terrier}|\text{terrier})$$

$$*Pr(\text{terrier}|\text{hunting dog})$$

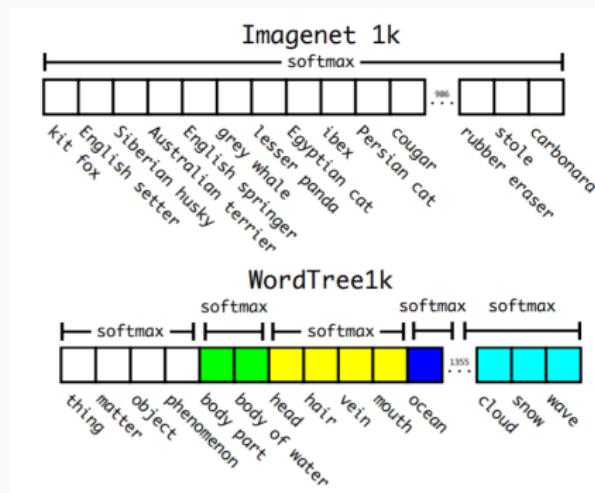
\* . . \*

$$*Pr(\text{mammal}|Pr(\text{animal}))$$

$$*Pr(\text{animal}|\text{physical object})$$

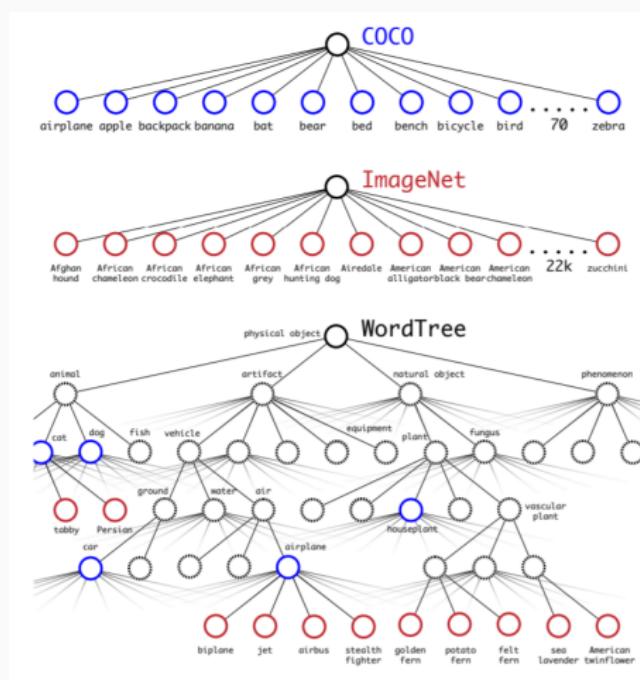
# YoLo v2, YOLO9000, YoLo v3

- Это позволяет строить много маленьких softmax'ов, которые проще обучать:



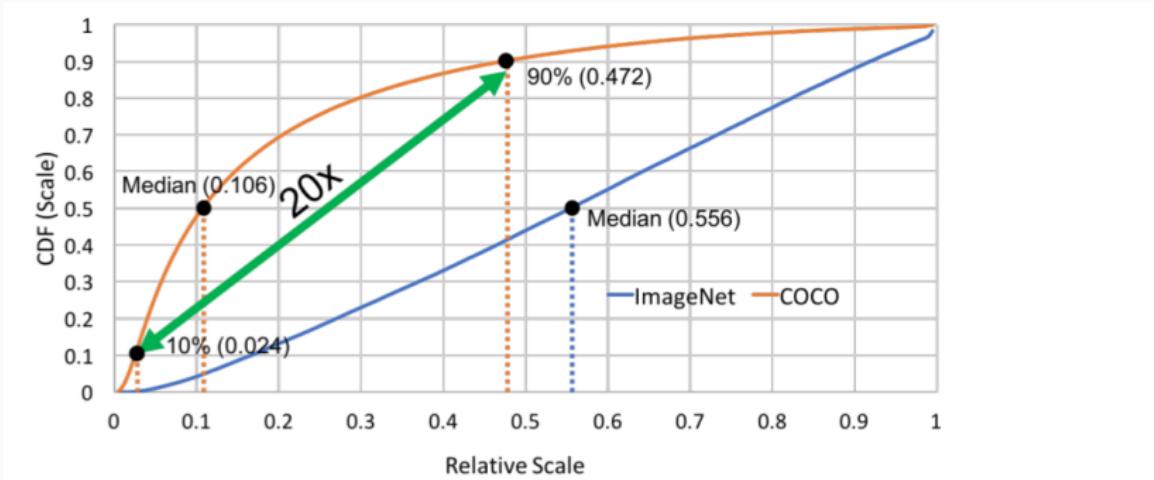
# YoLo v2, YOLO9000, YoLo v3

- И датасеты можно объединить даже



# Проблемы с масштабом

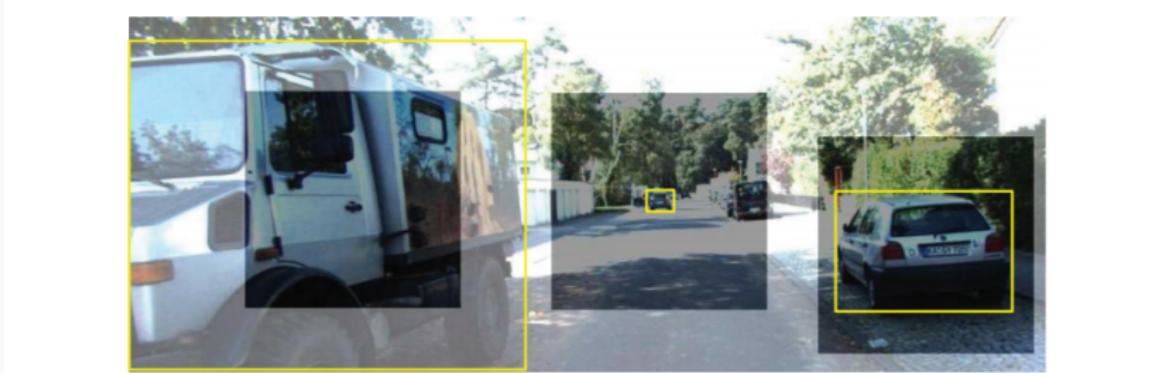
- Детекторы все хорошие, но почему-то распознают максимум 50-60% объектов. Почему на COCO такие плохие результаты?
- Потому что маленькие объекты плохо распознаются, а их в жизни много, и разброс масштабов очень большой.



- Почему так?

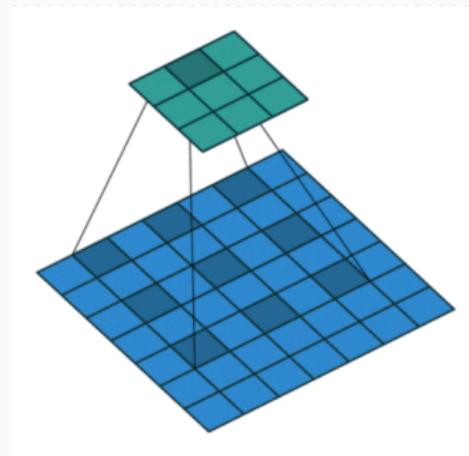
# Проблемы с масштабом

- Effective stride: к моменту region proposal network уже геометрия исходной картинки сжимается.
- В базовой конструкции Faster R-CNN получается effective stride 16, т.е. от объекта  $16 \times 16$  пикселей остаётся 1 пиксель.
- А если пытааться уменьшить stride, то сети выходят совсем гигантские и неподъёмные. Что делать?..



# Проблемы с масштабом

- Во-первых, свёртки в таких сетях часто немного другие.  
Dilated/atrous convolutions:



- Кстати:  
[https://github.com/vdumoulin/conv\\_arithmetic](https://github.com/vdumoulin/conv_arithmetic)
- Зачем нужны dilated convolutions?

# Проблемы с масштабом

- А чтобы не повторяясь покрыть гораздо большее receptive field за меньшее число слоёв:

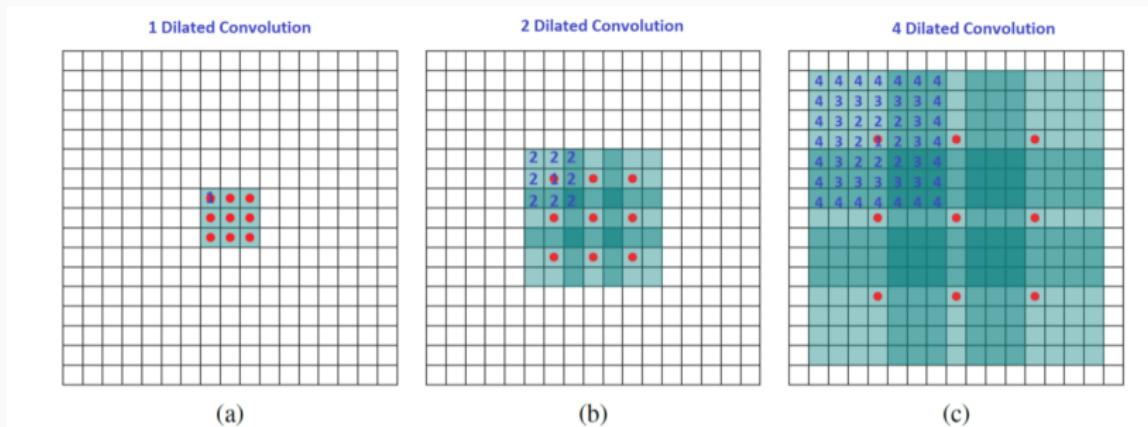
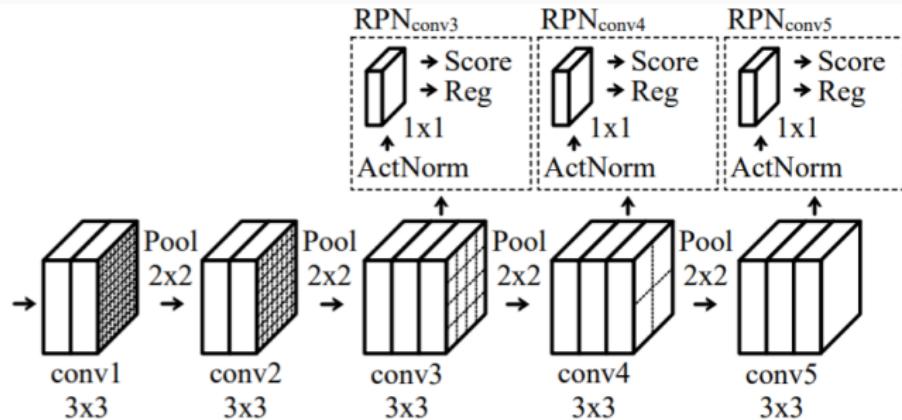


Figure 1: Systematic dilation supports exponential expansion of the receptive field without loss of resolution or coverage. (a)  $F_1$  is produced from  $F_0$  by a 1-dilated convolution; each element in  $F_1$  has a receptive field of  $3 \times 3$ . (b)  $F_2$  is produced from  $F_1$  by a 2-dilated convolution; each element in  $F_2$  has a receptive field of  $7 \times 7$ . (c)  $F_3$  is produced from  $F_2$  by a 4-dilated convolution; each element in  $F_3$  has a receptive field of  $15 \times 15$ . The number of parameters associated with each layer is identical. The receptive field grows exponentially while the number of parameters grows linearly.

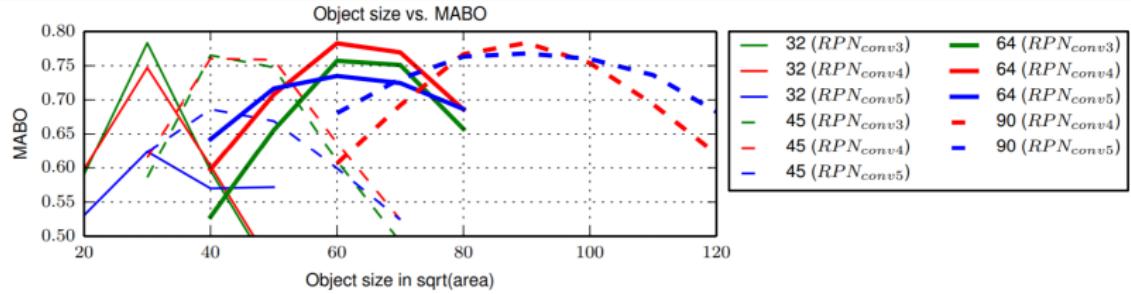
# Проблемы с масштабом

- Dilated convolutions используются в Faster R-CNN, R-FCN и других.
- (Yu, Koltun, 2016): Multi-scale context aggregation by dilated convolutions; собирают весь контекст с разных dilated convolutions вместе, применяют к сегментации.
- Можно ещё пытаться сделать несколько разных RPN; (Eggert et al., 2017) – возьмём базовый Faster R-CNN и встроим в него три RPN с разным масштабом.

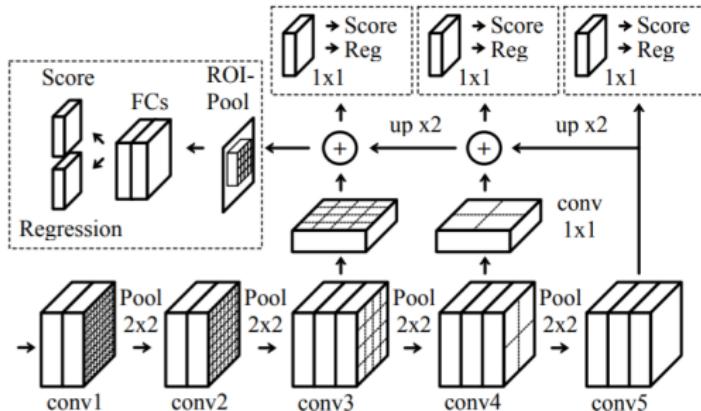


# Проблемы с масштабом

- Разные RPN нужны для объектов разного размера:

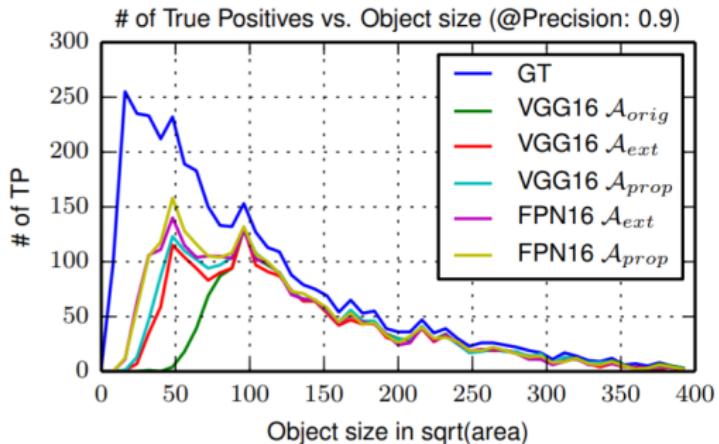


- И делают нечто похожее на FPN, чтобы объединить признаки:



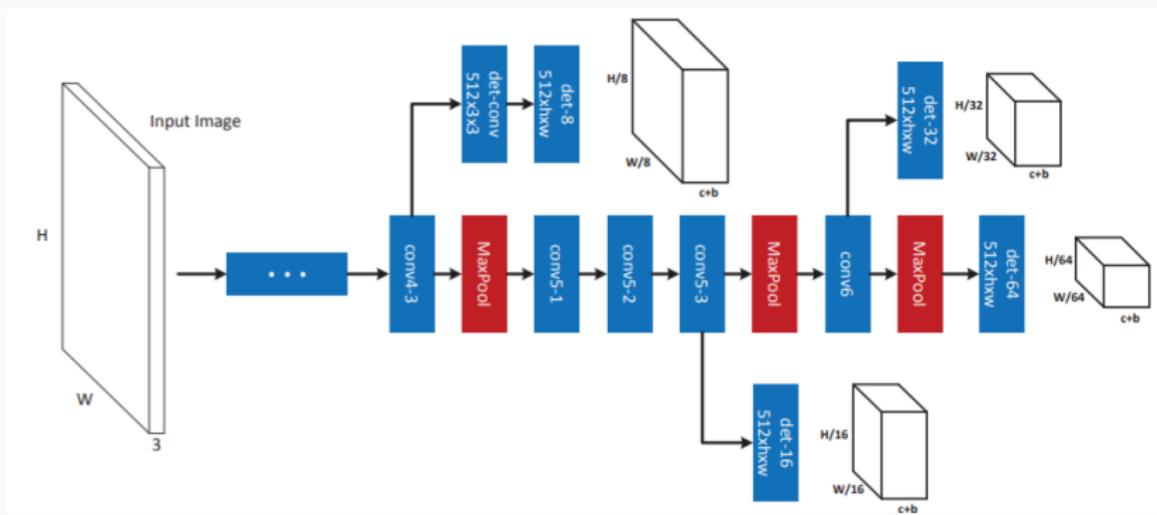
# Проблемы с масштабом

- Получается хорошо именно за счёт маленьких объектов:



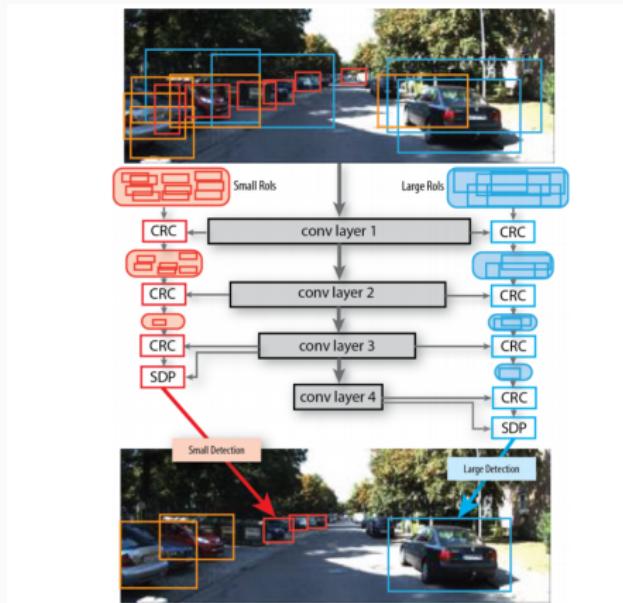
# Проблемы с масштабом

- (Cai et al., 2016): A unified multiscale deep convolutional neural network for fast object detection, аналогичная идея
- У object proposal network несколько веток с выходами разных масштабов:



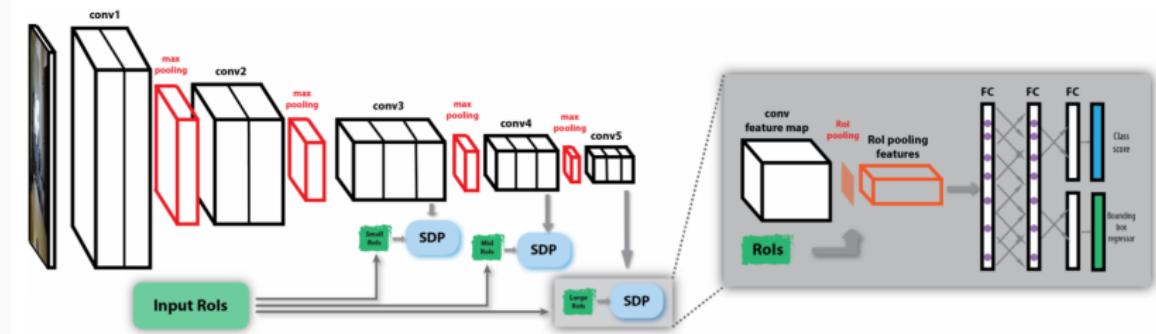
# Exploit all the layers

- (Yang et al., 2016): Exploit All the Layers; другой подход к тому, чтобы объединить все слои – cascaded rejection classifiers и scale-dependent pooling



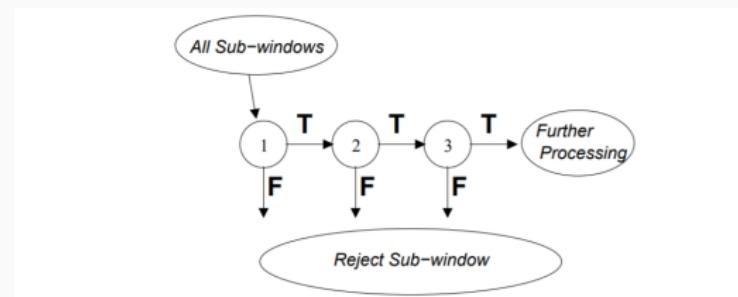
# Exploit all the layers

- Scale-dependent pooling: давайте будем брать в RoI pooling layer не один и тот же слой, а в зависимости от размера RoI proposal:



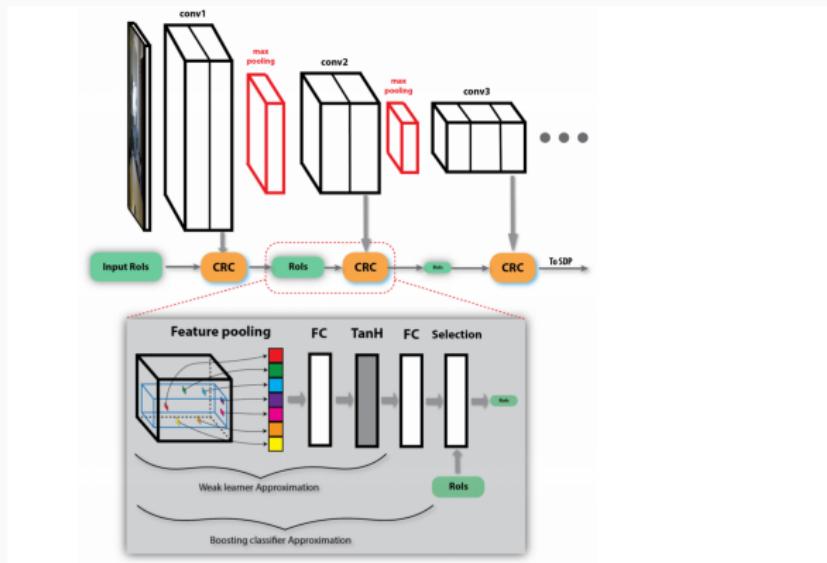
# Exploit all the layers

- Главный bottleneck в Faster R-CNN – это оценка proposals; их десятки тысяч.
- Идея: давайте сделаем каскадный классификатор, который будет быстро отмечать большинство.
- Это, мягко скажем, не новая идея. Viola-Jones object detection (2001) – это каскадный классификатор, в котором кандидат в bbox (лицо в исходной статье) должен последовательно пройти несколько классификаторов (бустингом обученных):



# Exploit all the layers

- И тут так же: AdaBoost для отбрасывания proposals на каждом уровне



# Exploit all the layers

- Тоже с маленькими объектами лучше получается:

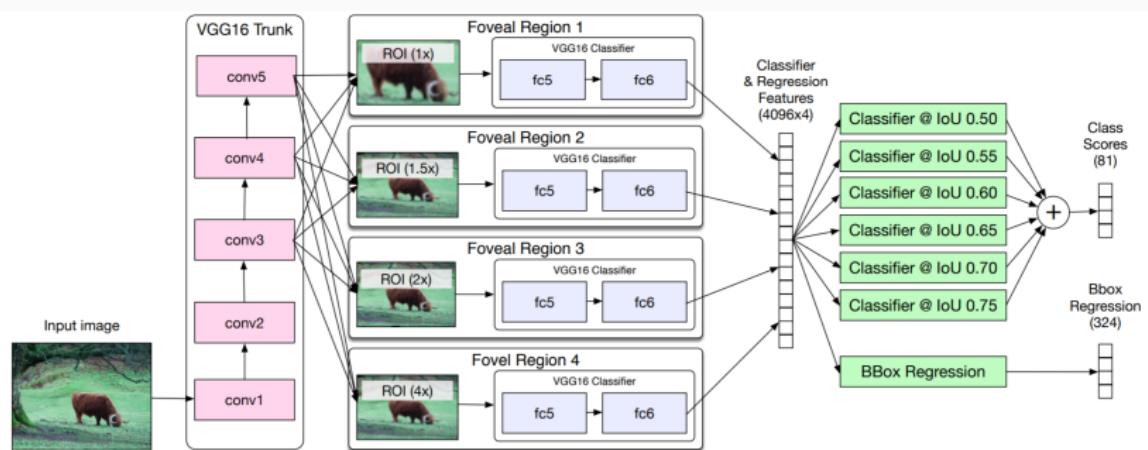


# MultiPath Object Detection

- (Zagoruyko et al., 2016): A MultiPath Network for Object Detection
- Решают всё ту же проблему с масштабом, особенно на COCO; по сравнению с другими датасетами, там
  - гораздо шире спектр разных масштабов объектов, больше маленьких;
  - объекты менее выделяются, часто частично скрыты, среди других объектов и т.п.;
  - метрика качества больше связана на точную локализацию объектов.
- Идея MultiPath в том, чтобы выделять proposals так же, как в Fast(er) R-CNN, но потом оценивать их более сложным образом...

# MultiPath Object Detection

- Три идеи: skip connections, foveal regions и integral loss.



# MultiPath Object Detection

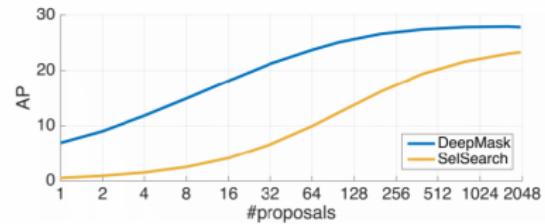
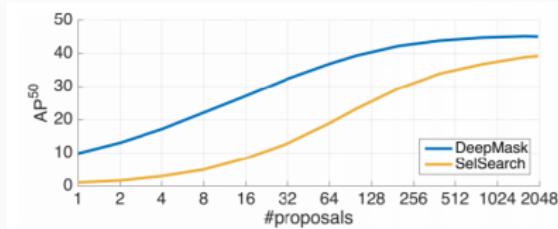
- Skip connections: добавляем RoI-pooled признаки с более ранних слоёв, чтобы лучше работать с маленькими объектами
- Foveal structure: одного proposal'a может быть мало для оценки; добавляем в разных пропорциях контекст, т.е. уменьшаем масштаб картинки вокруг proposal'a
- Integral loss: обучаем несколько классификаторов на разных порогах для IoU и усредняем по ним

integral loss	foveal	skip	AP <sup>50</sup>	AP
✓			43.4	25.2
	✓		42.2	25.6
✓	✓		45.2	25.8
✓	✓		44.4	26.9
	✓	✓	<b>46.4</b>	27.0
✓	✓	✓	44.8	<b>27.9</b>

integral loss	context	#regions	AP <sup>50</sup>	AP
	none	1	43.4	25.2
	multiregion	10	44.0	25.5
	foveal	4	<b>45.2</b>	<b>25.8</b>
✓	none	1	42.2	25.6
✓	multiregion	10	43.1	26.3
✓	foveal	4	<b>44.4</b>	<b>26.9</b>

# MultiPath Object Detection

- Они использовали модель DeepMask вместо selective search для порождения proposal'ов, что само по себе лучше:

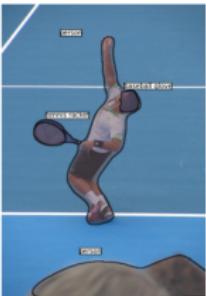
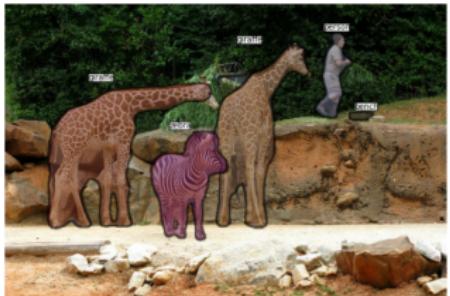


- А потом ещё скармливали полученные bbox'ы обратно в DeepMask и получали отличную сегментацию:



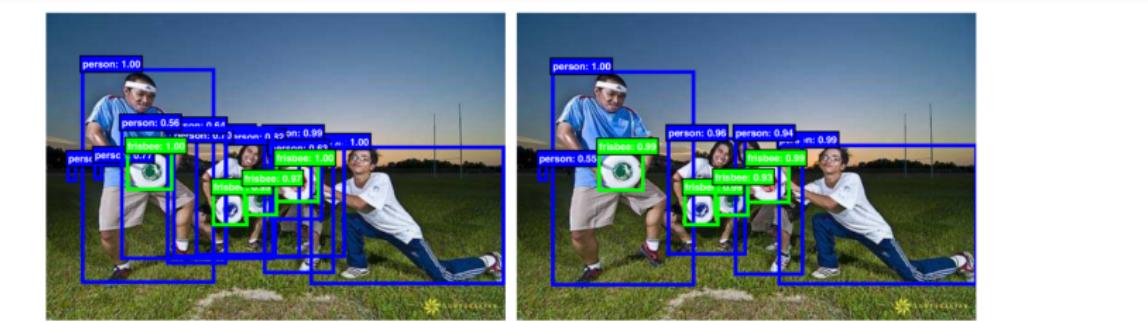
# MultiPath Object Detection

- Прямо правда хорошую (но о сегментации потом):



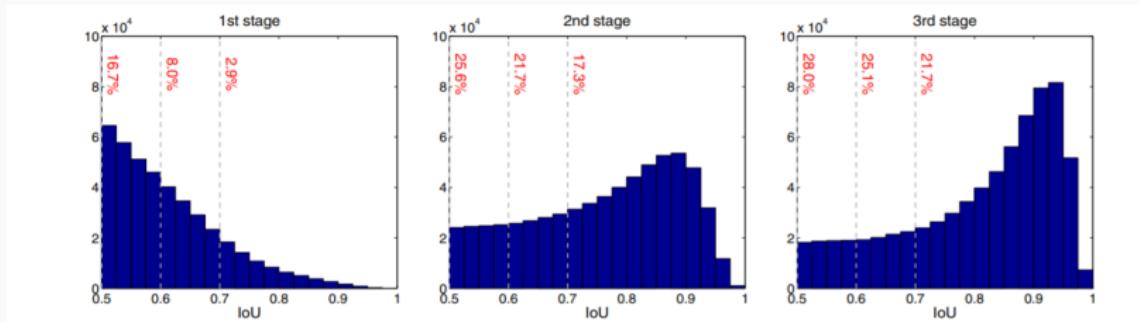
# Cascade R-CNN

- (Cai, Vasconcelos, 2018): Cascade R-CNN – продолжение идеи каскадов
- Как избежать большого числа false positives от region proposals?
- В двухуровневых сетях нужно определить порог по IoU для обучения, и обычный порог вроде 0.5 пропускает много лишнего, и в результате в детекторе тоже будет много false positives



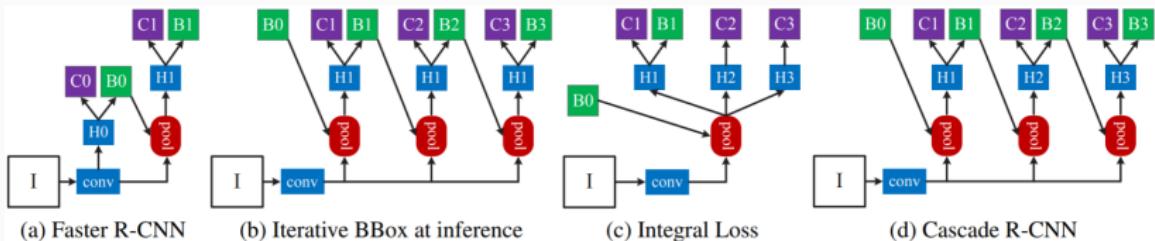
# Cascade R-CNN

- Первая идея: обучим ансамбль детекторов с разными значениями порога; но трудно обучать на высоких значениях, там мало примеров
- Iterative BBox (Gidaris, Komodakis, 2015): давайте итеративно применять bbox-регрессию несколько раз
- Но если это один и тот же регрессор, то он будет сильно не оптимален на итерациях после первой, распределение IoU сильно изменится:



# Cascade R-CNN

- Integral loss (Zagoruyko et al., 2016): давайте совместим функции потерь классификаторов
- Поэтому Cascade R-CNN использует каскад из нескольких разных регрессоров для уточнения bbox



# Cascade R-CNN

- И действительно получается лучше, чем integral loss

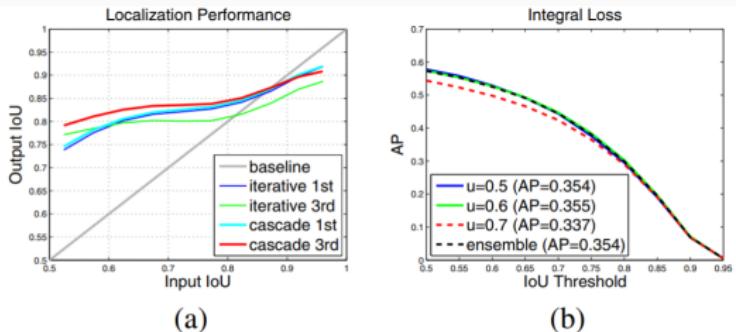


Figure 7. (a) is the localization comparison, and (b) is the detection performance of individual classifiers in the *integral loss* detector.

	AP	AP <sub>50</sub>	AP <sub>60</sub>	AP <sub>70</sub>	AP <sub>80</sub>	AP <sub>90</sub>
FPN+ baseline	34.9	57.0	51.9	43.6	29.7	7.1
Iterative BBox	35.4	57.2	52.1	44.2	30.4	8.1
Integral Loss	35.4	57.3	52.5	44.4	29.9	6.9
Cascade R-CNN	<b>38.9</b>	<b>57.8</b>	<b>53.4</b>	<b>46.9</b>	<b>35.8</b>	<b>15.8</b>

# Cascade R-CNN

- И этот метод так или иначе улучшает все state of the art детекторы:

	backbone	AP	AP <sub>50</sub>	AP <sub>75</sub>	AP <sub>S</sub>	AP <sub>M</sub>	AP <sub>L</sub>
YOLOv2 [29]	DarkNet-19	21.6	44.0	19.2	5.0	22.4	35.5
SSD513 [25]	ResNet-101	31.2	50.4	33.3	10.2	34.5	49.8
RetinaNet [24]	ResNet-101	39.1	59.1	42.3	21.8	42.7	50.2
Faster R-CNN++ [18]*	ResNet-101	34.9	55.7	37.4	15.6	38.7	50.9
Faster R-CNN w FPN [23]	ResNet-101	36.2	59.1	39.0	18.2	39.0	48.2
Faster R-CNN w FPN+ (ours)	ResNet-101	38.8	61.1	41.9	21.3	41.8	49.8
Faster R-CNN by G-RMI [19]	Inception-ResNet-v2	34.7	55.5	36.7	13.5	38.1	52.0
Deformable R-FCN [5]*	Aligned-Inception-ResNet	37.5	58.0	40.8	19.4	40.1	52.5
Mask R-CNN [16]	ResNet-101	38.2	60.3	41.7	20.1	41.1	50.2
AttracitioNet [11]*	VGG16+Wide ResNet	35.7	53.4	39.3	15.6	38.0	52.7
<b>Cascade R-CNN</b>	ResNet-101	<b>42.8</b>	<b>62.1</b>	<b>46.3</b>	<b>23.7</b>	<b>45.5</b>	<b>55.2</b>

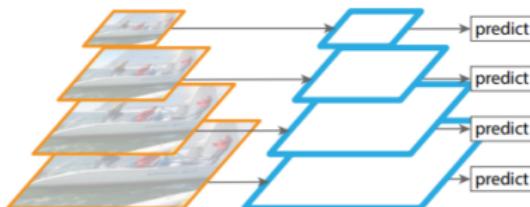
Table 5. The state-of-the-art *single-model* detectors on COCO `test-dev`. The entries denoted by “\*” used bells and whistles at inference.

	backbone	cascade	train speed	test speed	model size	val (5k)						test-dev (20k)					
						AP	AP <sub>50</sub>	AP <sub>75</sub>	AP <sub>S</sub>	AP <sub>M</sub>	AP <sub>L</sub>	AP	AP <sub>50</sub>	AP <sub>75</sub>	AP <sub>S</sub>	AP <sub>M</sub>	AP <sub>L</sub>
Faster R-CNN	VGG	✗	0.12s	0.075s	278M	23.6	43.9	23.0	8.0	26.2	35.5	23.5	43.9	22.6	8.1	25.1	34.7
		✓	0.14s	0.115s	704M	27.0	44.2	27.7	8.6	29.1	42.2	26.9	44.3	27.8	8.3	28.2	41.1
R-FCN	ResNet-50	✗	0.19s	0.07s	133M	27.0	48.7	26.9	9.8	30.9	40.3	27.1	49.0	26.9	10.4	29.7	39.2
		✓	0.24s	0.075s	184M	31.1	49.8	32.8	10.4	34.4	48.5	30.9	49.9	32.6	10.5	33.1	46.9
R-FCN	ResNet-101	✗	0.23s	0.075s	206M	30.3	52.2	30.8	12.0	34.7	44.3	30.5	52.9	31.2	12.0	33.9	43.8
		✓	0.29s	0.083s	256M	33.3	52.0	35.2	11.8	37.2	51.1	33.3	52.6	35.2	12.1	36.2	49.3
FPN+	ResNet-50	✗	0.30s	0.095s	165M	36.5	58.6	39.2	20.8	40.0	47.8	36.5	59.0	39.2	20.3	38.8	46.4
		✓	0.33s	0.115s	272M	40.3	59.4	43.7	22.9	43.7	54.1	40.6	59.9	44.0	22.6	42.7	52.1
FPN+	ResNet-101	✗	0.38s	0.115s	238M	38.5	60.6	41.7	22.1	41.9	51.1	38.8	61.1	41.9	21.3	41.8	49.8
		✓	0.41s	0.14s	345M	42.7	61.6	46.6	23.8	46.2	57.4	42.8	62.1	46.3	23.7	45.5	55.2

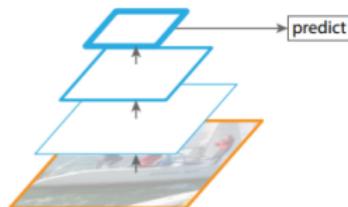
Table 6. Detailed comparison on multiple popular baseline object detectors. All speeds are reported per image on a single Titan Xp GPU.

# Feature Pyramid Networks

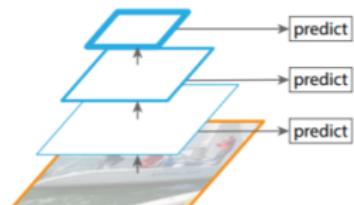
- (Lin et al., 2017): Feature Pyramid Networks for Object Detection
- Базовая идея – строим пирамиду признаков и используем их все:



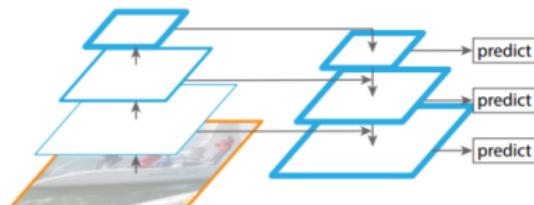
(a) Featurized image pyramid



(b) Single feature map



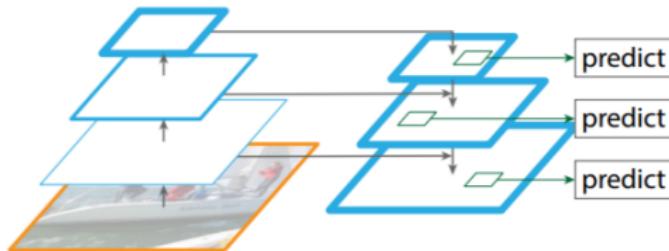
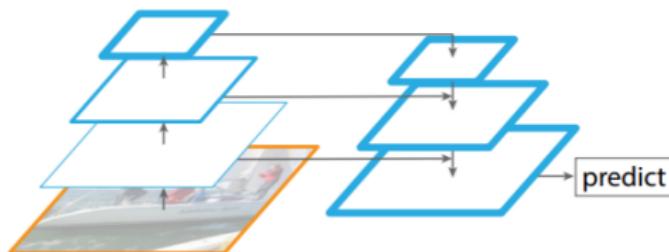
(c) Pyramidal feature hierarchy



(d) Feature Pyramid Network

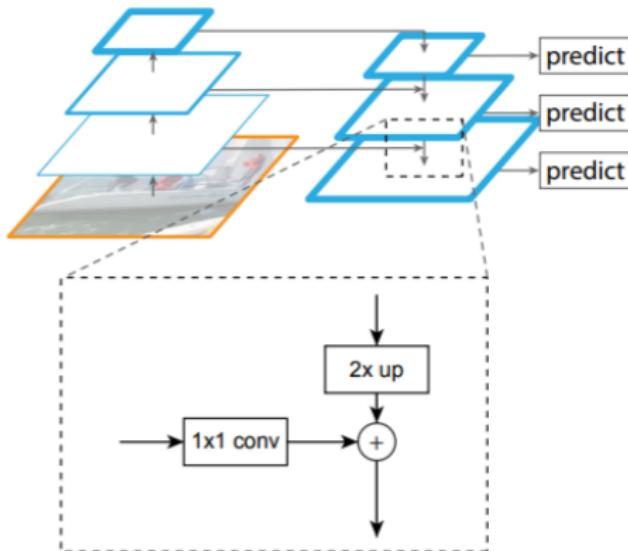
# Feature Pyramid Networks

- Раньше так не делали, потому что очень долго и сложно
- FPN пытается сделать пирамиду признаков, у которой было бы много семантики на каждом уровне; сверху обычная архитектура, снизу FPN:



# Feature Pyramid Networks

- Когда идём сверху вниз, добавляем признаки сверху, в них сила, то бишь семантика:



- Эту идею можно куда угодно применить, и в RPN, и в Faster R-CNN

# Feature Pyramid Networks

- И, опять же, очень круто получается:

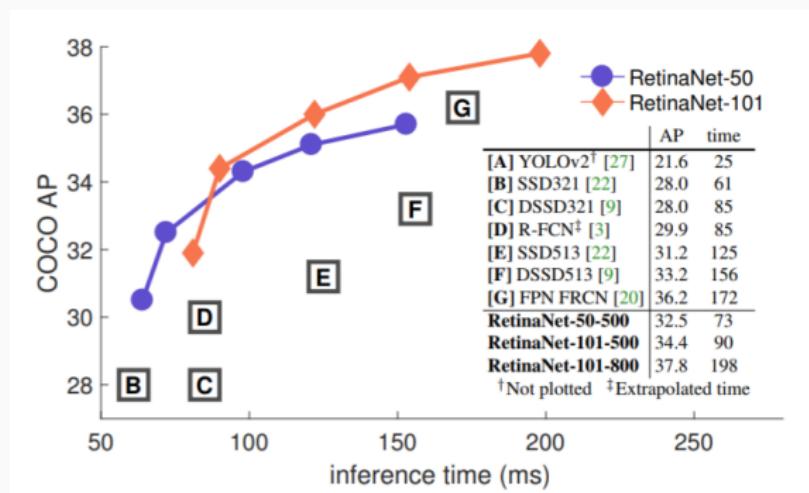
method	backbone	competition	image pyramid	test-dev					test-std				
				AP@.5	AP	AP <sub>s</sub>	AP <sub>m</sub>	AP <sub>l</sub>	AP@.5	AP	AP <sub>s</sub>	AP <sub>m</sub>	AP <sub>l</sub>
ours, Faster R-CNN on FPN	ResNet-101	-		<b>59.1</b>	<b>36.2</b>	<b>18.2</b>	<b>39.0</b>	48.2	<b>58.5</b>	<b>35.8</b>	<b>17.5</b>	<b>38.7</b>	47.8
<i>Competition-winning single-model results follow:</i>													
G-RMI <sup>†</sup>	Inception-ResNet	2016	-	34.7	-	-	-	-	-	-	-	-	-
AttractioNet <sup>‡</sup> [10]	VGG16 + Wide ResNet <sup>§</sup>	2016	✓	53.4	35.7	15.6	38.0	<b>52.7</b>	52.9	35.3	14.7	37.6	<b>51.9</b>
Faster R-CNN +++ [16]	ResNet-101	2015	✓	55.7	34.9	15.6	38.7	50.9	-	-	-	-	-
Multipath [40] (on minival)	VGG-16	2015		49.6	31.5	-	-	-	-	-	-	-	-
ION <sup>‡</sup> [2]	VGG-16	2015		53.4	31.2	12.8	32.9	45.2	52.9	30.7	11.8	32.8	44.8

- И можно убедиться, что действительно все эти пирамиды нужны, без них хуже:

Fast R-CNN	proposals	feature	head	lateral?	top-down?	AP@0.5	AP	AP <sub>s</sub>	AP <sub>m</sub>	AP <sub>l</sub>
(a) baseline on conv4	RPN, $\{P_k\}$	$C_4$	conv5			54.7	31.9	15.7	36.5	45.5
(b) baseline on conv5	RPN, $\{P_k\}$	$C_5$	2fc			52.9	28.8	11.9	32.4	43.4
(c) FPN	RPN, $\{P_k\}$	$\{P_k\}$	2fc	✓	✓	<b>56.9</b>	<b>33.9</b>	<b>17.8</b>	<b>37.7</b>	<b>45.8</b>
<i>Ablation experiments follow:</i>										
(d) bottom-up pyramid	RPN, $\{P_k\}$	$\{P_k\}$	2fc	✓		44.9	24.9	10.9	24.4	38.5
(e) top-down pyramid, w/o lateral	RPN, $\{P_k\}$	$\{P_k\}$	2fc		✓	54.0	31.3	13.3	35.2	45.3
(f) only finest level	RPN, $\{P_k\}$	$P_2$	2fc	✓	✓	56.3	33.4	17.3	37.3	45.6

# Detectron

- (Lin et al., 2018): Focal Loss for Dense Object Detection
- Изменили функцию ошибки, стали работать гораздо быстрее и лучше:



- Как это?

- Что мешает one-shot detectors? Почему они хуже, чем two-stage (типа Faster R-CNN)?
- Lin et al. утверждают, что мешает class imbalance: надо очень много выходов сразу предсказать, и в основном это очень простые примеры из чистого background
- В two-stage детекторах есть фильтрация, region proposal networks, т.е. у них imbalance совсем не такой сильный

- Чтобы с этим справиться, предлагается focal loss. Начинаем с перекрёстной энтропии:

$$\text{CE}(p, y) = \begin{cases} -\log(p), & \text{если } y = 1, \\ -\log(1 - p), & \text{в противном случае;} \end{cases}$$

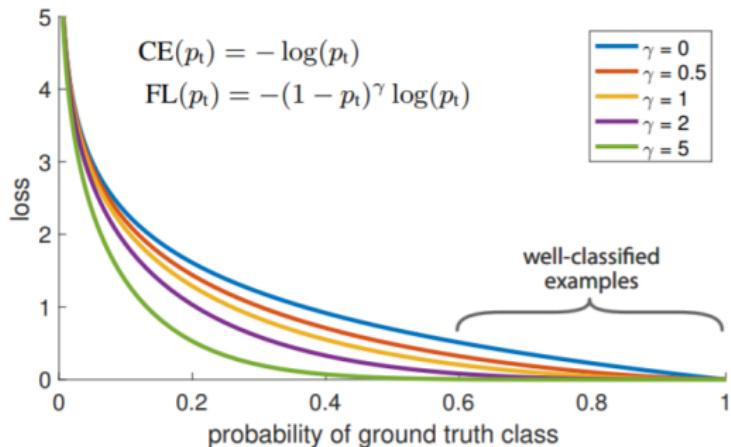
переобозначим  $p_t = [y = 1]p + [y = 0](1 - p)$ , и тогда  
 $\text{CE}(p, y) = -\log(p_t)$ .

- Проблема тут в том, что даже простые примеры с  $p_t > 0.8$  дают нетривиальную потерю, и если всё просуммировать, то забивают «сигнал».

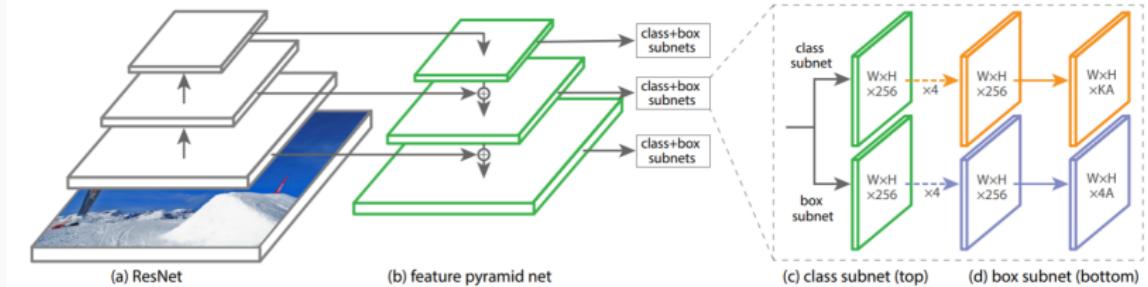
- Focal loss:

$$\text{FL}(p_t, y) = -(1 - p_t)^\gamma \log(p_t).$$

- Просто перевзвешиваем так, чтобы потеря было меньше на хорошо классифицируемых примерах.



- RetinaNet: FPN c focal loss

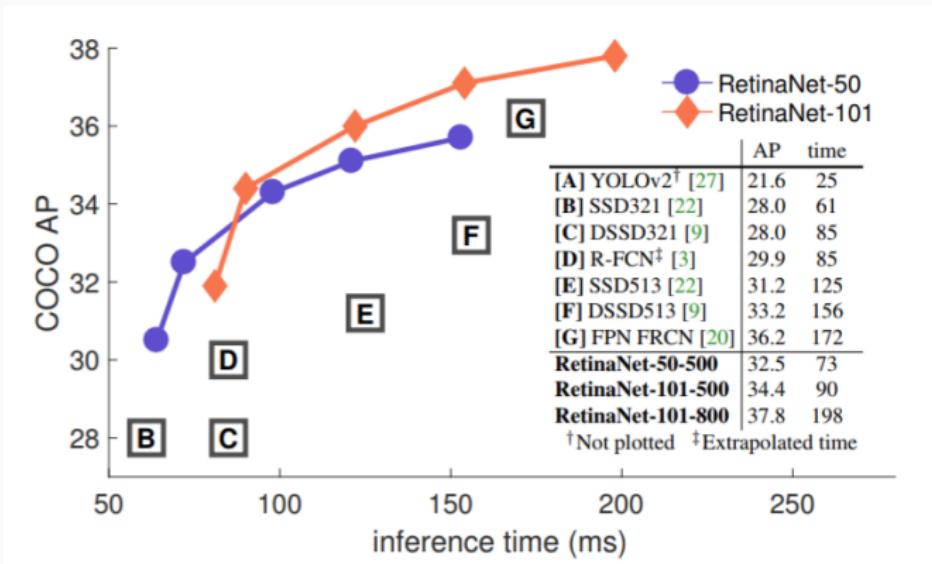


- Получается очень хорошо:

	backbone	AP	AP <sub>50</sub>	AP <sub>75</sub>	AP <sub>S</sub>	AP <sub>M</sub>	AP <sub>L</sub>
<i>Two-stage methods</i>							
Faster R-CNN+++ [16]	ResNet-101-C4	34.9	55.7	37.4	15.6	38.7	50.9
Faster R-CNN w FPN [20]	ResNet-101-FPN	36.2	59.1	39.0	18.2	39.0	48.2
Faster R-CNN by G-RMI [17]	Inception-ResNet-v2 [34]	34.7	55.5	36.7	13.5	38.1	52.0
Faster R-CNN w TDM [32]	Inception-ResNet-v2-TDM	36.8	57.7	39.2	16.2	39.8	<b>52.1</b>
<i>One-stage methods</i>							
YOLOv2 [27]	DarkNet-19 [27]	21.6	44.0	19.2	5.0	22.4	35.5
SSD513 [22, 9]	ResNet-101-SSD	31.2	50.4	33.3	10.2	34.5	49.8
DSSD513 [9]	ResNet-101-DSSD	33.2	53.3	35.2	13.0	35.4	51.1
<b>RetinaNet</b> (ours)	ResNet-101-FPN	39.1	59.1	42.3	21.8	42.7	50.2
<b>RetinaNet</b> (ours)	ResNeXt-101-FPN	<b>40.8</b>	<b>61.1</b>	<b>44.1</b>	<b>24.1</b>	<b>44.2</b>	51.2

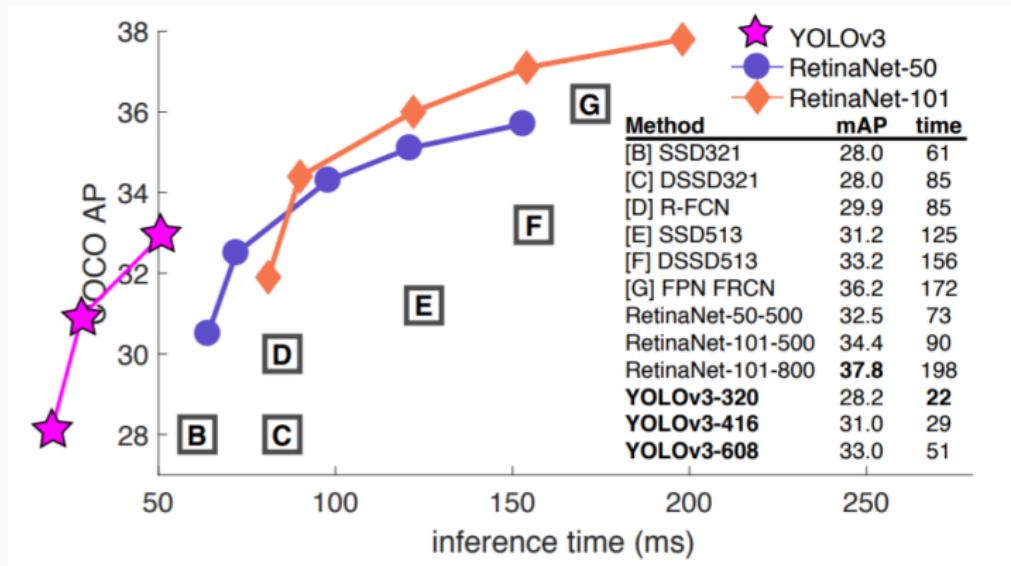
# Detectron

- Очень хорошо...



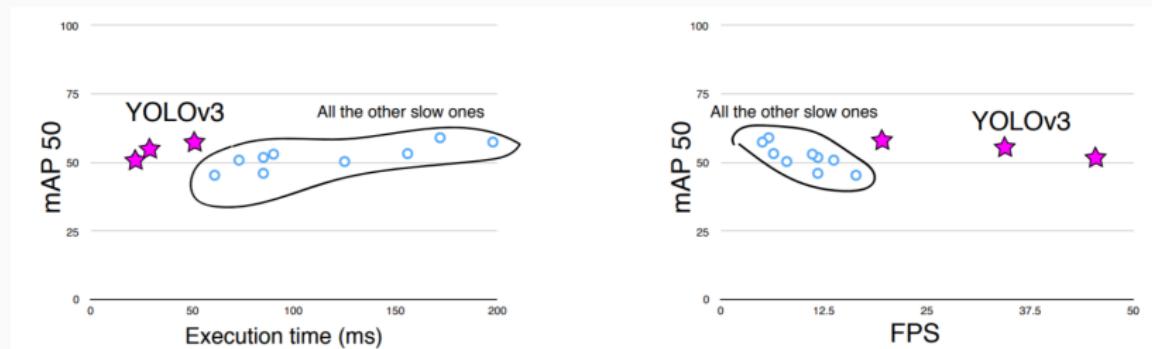
# YOLOv3

- ...но YOLO всё равно лучше



# YOLOv3

- YOLOv3 – это новый вариант YOLO, в котором использовали focal loss.
- Быстро:



- И хорошо: [https://www.youtube.com/watch?time\\_continue=2&v=MPU2HistivI](https://www.youtube.com/watch?time_continue=2&v=MPU2HistivI)

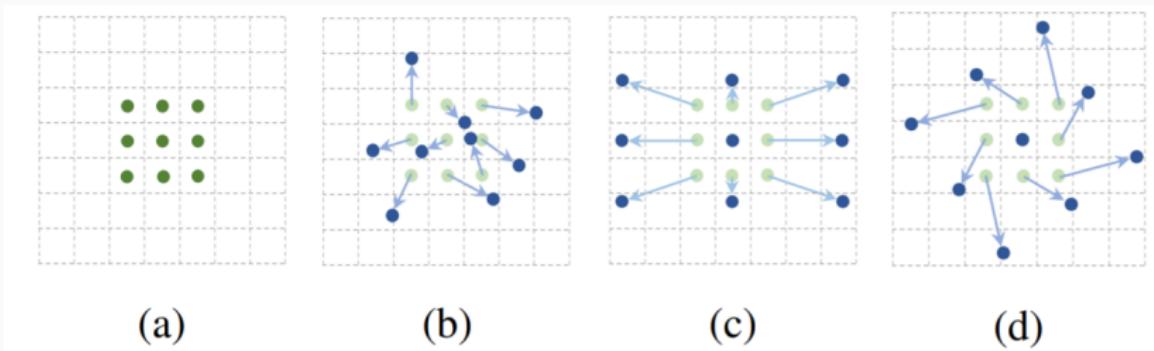
## Такие разные свёртки

---

- И ёщё о разных свёртках; на самом деле свёртки довольно ограничены геометрически
- То, что они так много всего могут – это скорее от большой аугментации, общей выразительности моделей и каких-то ad hoc решений вроде max-pooling для инвариантности к сдвигам
- (Dai et al., 2017), Deformable Convolutional Networks: давайте попробуем сделать свёртки более интересными...

# Такие разные свёртки

- Деформируемая свёртка – давайте добавим возможность делать свёртки по разным множествам, с разными сдвигами:



- (c) и (d) показывают, что как частные случаи и dilated получатся, и вращения, что угодно

## Такие разные свёртки

- Формально у нас обычная свёртка – это

$$y = \sum_{p_n \in \mathcal{R}} w(p_n)^\top x(p_0 + p_n),$$

где  $\mathcal{R}$  – какое-то заранее заданное множество.

- А деформируемая – это

$$y = \sum_{p_n \in \mathcal{R}} w(p_n)^\top x(p_0 + p_n + \Delta p_n),$$

где  $\Delta p_n$  – сдвиги.

## Такие разные свёртки

- Поскольку сдвиги обычно дробные, делаем билинейную интерполяцию

$$x(p) = \sum_q G(q, p)^\top x(q),$$

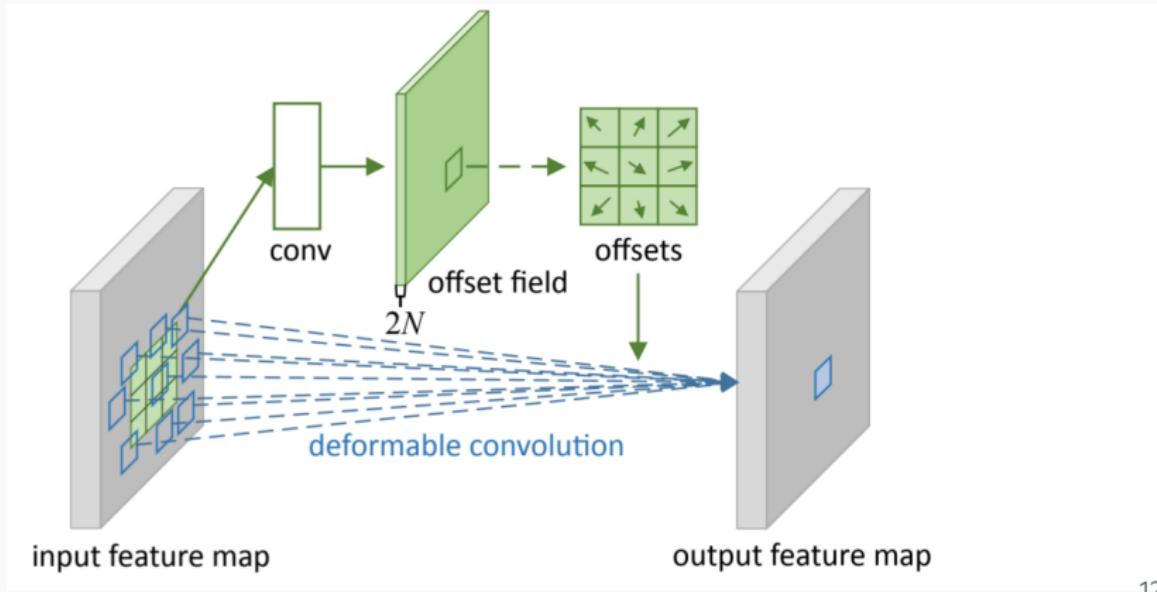
где  $p = p_0 + p_n + \Delta p_n$  – дробная точка,  $q$  пробегает близлежащие целочисленные,  $G(q, p)$  – ядро.

- В статье используют

$$G(q, p) = g(q_x, p_x)g(q_y, p_y), \text{ где } g(a, b) = \max(0, 1 - |a - b|).$$

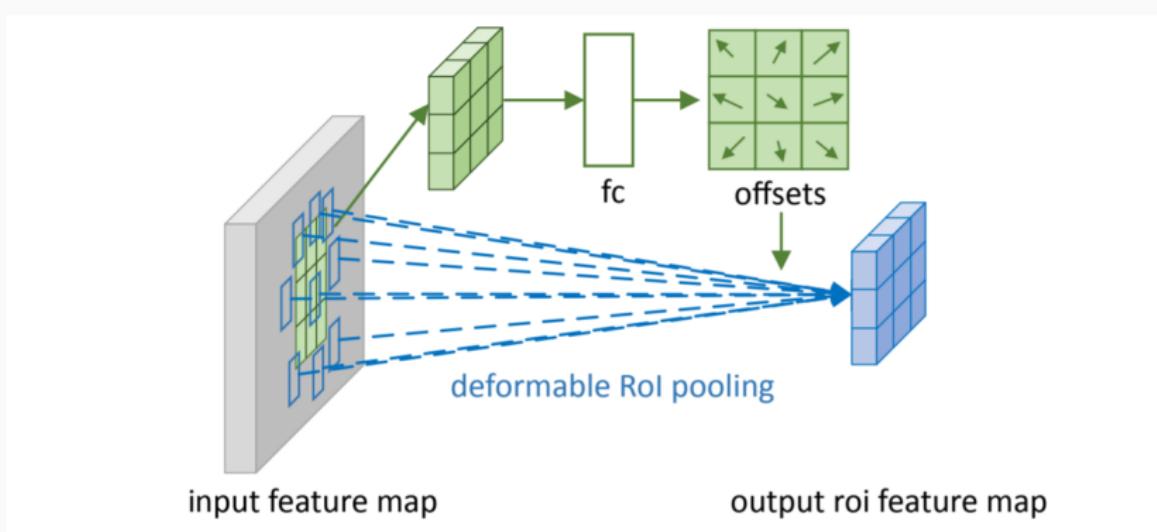
# Такие разные свёртки

- А откуда возьмутся сдвиги  $\Delta p_n$ ?
- Мы их тоже будем обучать как результат другого свёрточного слоя на той же карте признаков;  $2N$  – это  $N$  сдвигов размерности 2



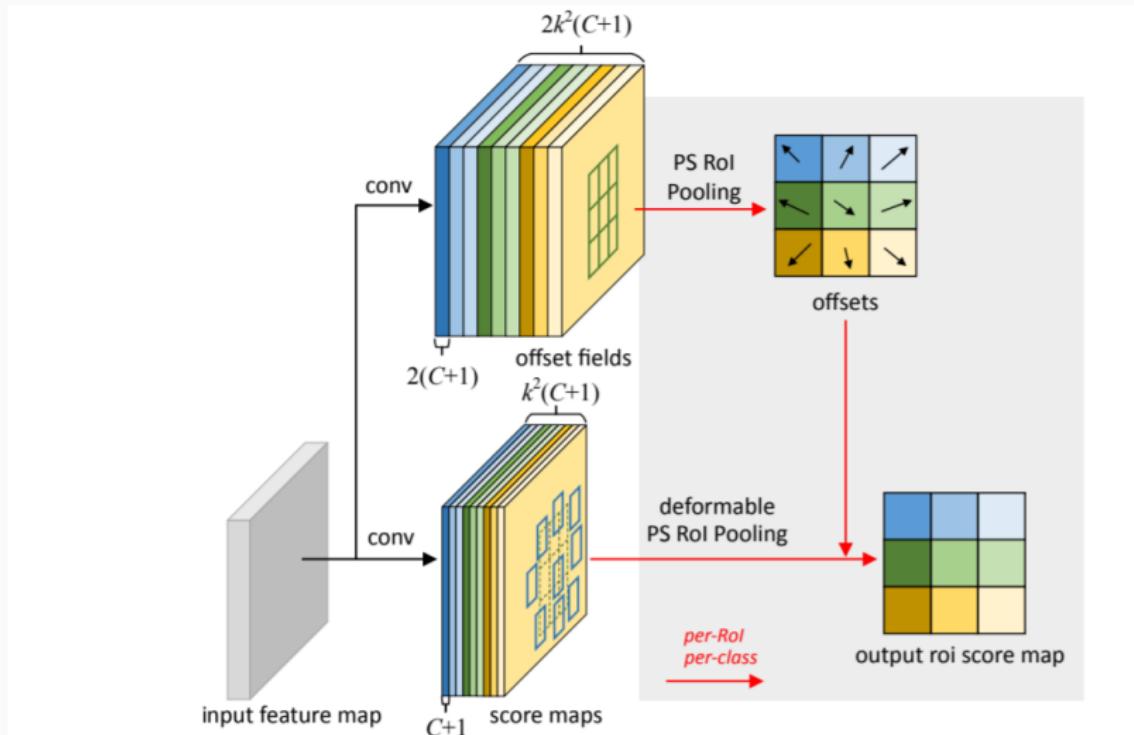
## Такие разные свёртки

- Аналогично можно сделать деформируемый RoI pooling
- Обычный RoI pooling делит окно на подокна  $k \times k$  и усредняет по подокнам; а тут просто добавим сдвиги к этим подокнам, которые тоже обучаются вместе со всей сетью:



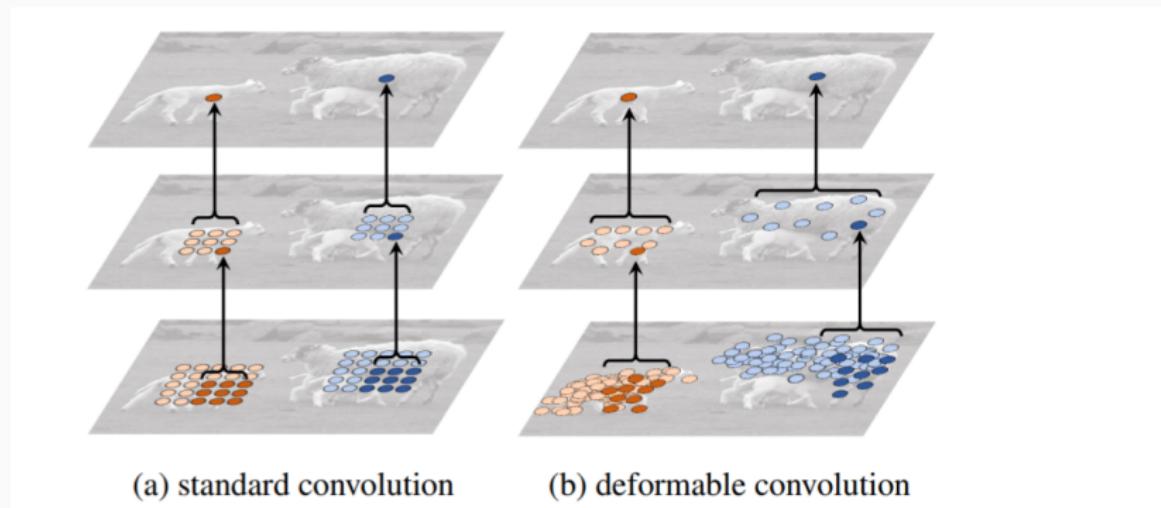
# Такие разные свёртки

- И ещё fully convolutional position-sensitive RoI pooling из R-FCN тоже можно сделать деформируемым:



# Такие разные свёртки

- В итоге получается, что receptive fields отдельных признаков становятся адаптивными и могут следовать семантике:



(a) standard convolution

(b) deformable convolution

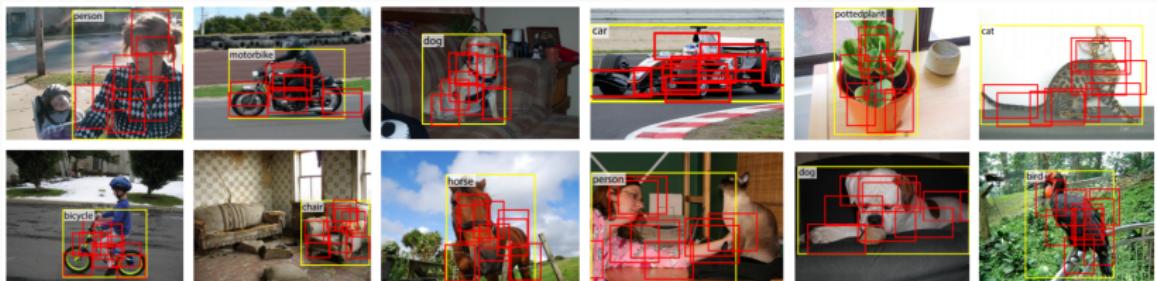
# Такие разные свёртки

- На практике тоже – базовые точки (всего их  $9^3 = 729$ ) для одного центра окна сильно зависят от того, какому объекту он принадлежит:



# Такие разные свёртки

- A deformable RoI pooling своими подокнами довольно точно покрывает сам объект, а фон не покрывает:



# Такие разные свёртки

- И тоже все детекторы становятся лучше:

method	backbone architecture	M	B	mAP@[0.5:0.95]	mAP <sup>r</sup> @0.5	mAP@[0.5:0.95] (small)	mAP@[0.5:0.95] (mid)	mAP@[0.5:0.95] (large)
class-aware RPN	ResNet-101			23.2	42.6	6.9	27.1	35.1
<b>Ours</b>				<b>25.8</b>	<b>45.9</b>	<b>7.2</b>	<b>28.3</b>	<b>40.7</b>
Faster RCNN	ResNet-101			29.4	48.0	9.0	30.5	47.1
<b>Ours</b>				<b>33.1</b>	<b>50.3</b>	<b>11.6</b>	<b>34.9</b>	<b>51.2</b>
R-FCN	ResNet-101			30.8	52.6	11.8	33.9	44.8
<b>Ours</b>				<b>34.5</b>	<b>55.0</b>	<b>14.0</b>	<b>37.7</b>	<b>50.3</b>
Faster RCNN	Aligned-Inception-ResNet			30.8	49.6	9.6	32.5	49.0
<b>Ours</b>				<b>34.1</b>	<b>51.1</b>	<b>12.2</b>	<b>36.5</b>	<b>52.4</b>
R-FCN	Aligned-Inception-ResNet			32.9	54.5	12.5	36.3	48.3
<b>Ours</b>				<b>36.1</b>	<b>56.7</b>	<b>14.8</b>	<b>39.8</b>	<b>52.2</b>
R-FCN	Aligned-Inception-ResNet	✓		34.5	55.0	16.8	37.3	48.3
<b>Ours</b>		✓		37.1	57.3	18.8	39.7	52.3
R-FCN		✓	✓	35.5	55.6	17.8	38.4	49.3
<b>Ours</b>		✓	✓	<b>37.5</b>	<b>58.0</b>	<b>19.4</b>	<b>40.1</b>	<b>52.5</b>

# Spatial Transformer Networks

- Вообще идея обучать какие-то геометрические преобразования данных тоже часто появляется.
- (Jaderberg et al., 2016) – Spatial Transformer Networks, статья от DeepMind
- Предлагают конструкцию слоя, который может делать нужные преобразования над данными без дополнительной разметки:

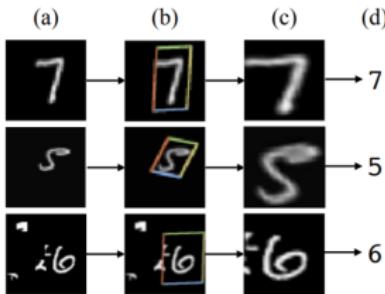


Figure 1: The result of using a spatial transformer as the first layer of a fully-connected network trained for distorted MNIST digit classification. (a) The input to the spatial transformer network is an image of an MNIST digit that is distorted with random translation, scale, rotation, and clutter. (b) The localisation network of the spatial transformer predicts a transformation to apply to the input image. (c) The output of the spatial transformer, after applying the transformation. (d) The classification prediction produced by the subsequent fully-connected network on the output of the spatial transformer. The spatial transformer network (a CNN including a spatial transformer module) is trained end-to-end with only class labels – no knowledge of the groundtruth transformations is given to the system.

# Spatial Transformer Networks

- Делается довольно просто: отдельная сеть обучает генератор новой сетки для карты признаков, а она применяется к карте и производит деформацию:

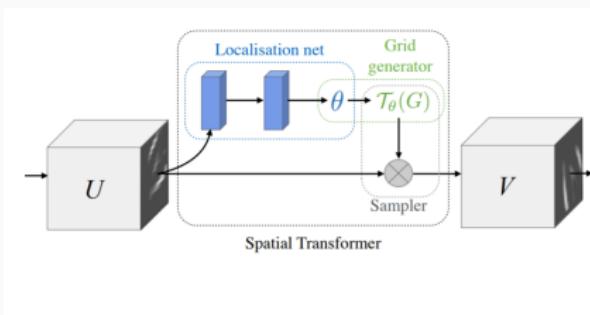
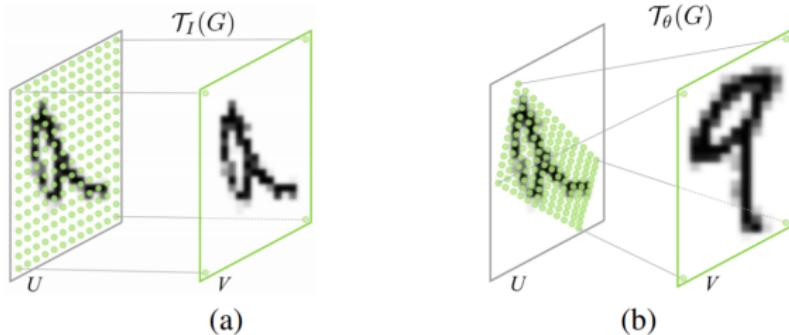


Figure 2: The architecture of a spatial transformer module. The input feature map  $U$  is passed to a localisation network which regresses the transformation parameters  $\theta$ . The regular spatial grid  $G$  over  $V$  is transformed to the sampling grid  $T_\theta(G)$ , which is applied to  $U$  as described in Sect. 3.3, producing the warped output feature map  $V$ . The combination of the localisation network and sampling mechanism defines a spatial transformer.

# Spatial Transformer Networks

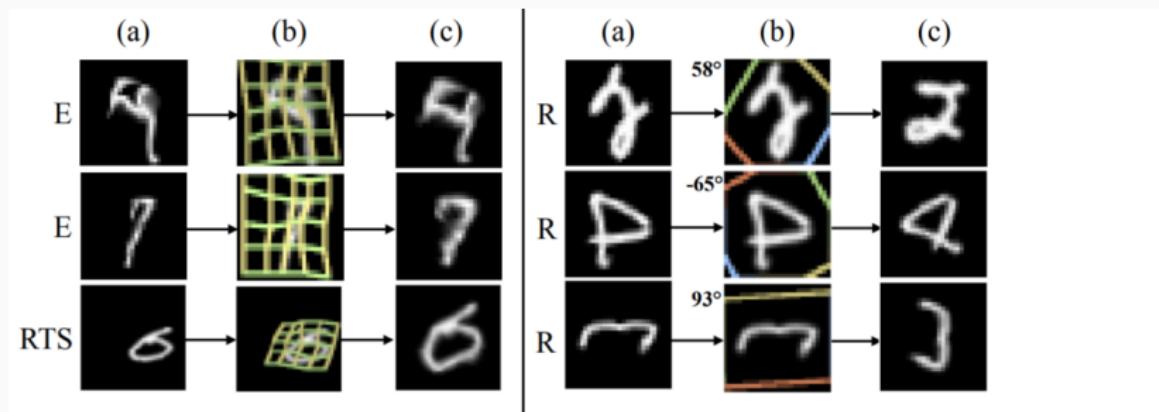
- По сути просто обучаем преобразование к сетке новых пикселей:



- Формально это опять сэмплирование из исходной сетки вокруг новых центров с неким ядром, которое делает это всё дифференцируемым.

# Spatial Transformer Networks

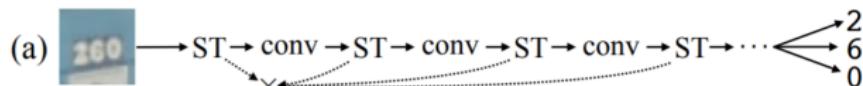
- Преобразования хорошо обучаются:



- <https://goo.gl/qdEhUu>
- <https://towardsdatascience.com/convnets-series-spatial-transformer-networks-cff4756>

# Spatial Transformer Networks

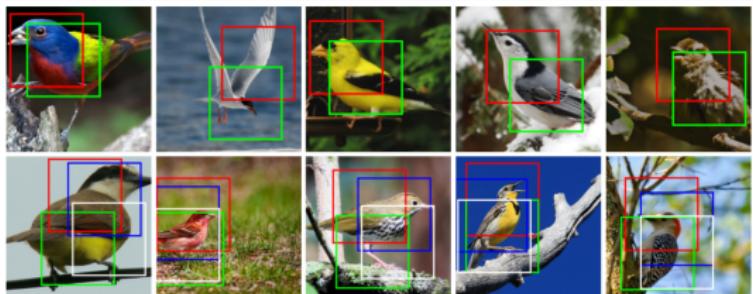
- И можно запускать последовательно, чтобы прочитать несколько цифр:



# Spatial Transformer Networks

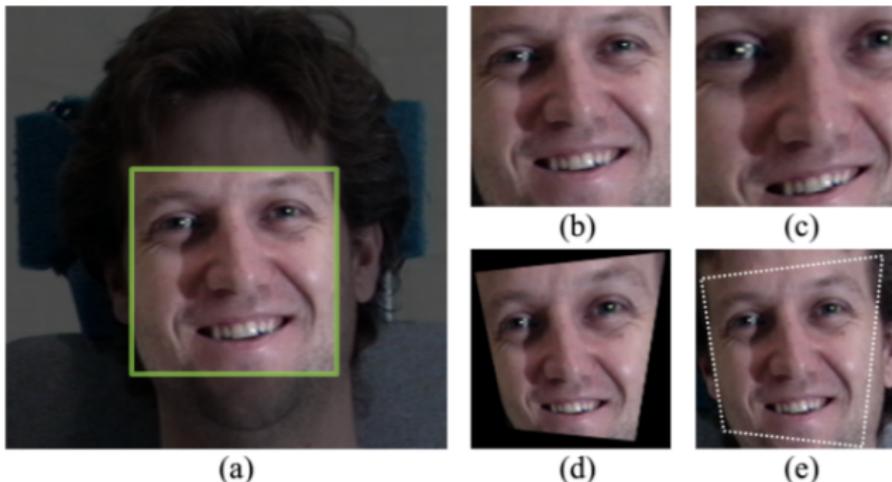
- И запускать параллельно, чтобы улучшать классификацию таким своеобразным механизмом внимания:

Model	
Cimpoi '15 [4]	66.7
Zhang '14 [30]	74.9
Branson '14 [2]	75.7
Lin '15 [20]	80.9
Simon '15 [24]	81.0
CNN (ours) 224px	82.3
2×ST-CNN 224px	83.1
2×ST-CNN 448px	83.9
4×ST-CNN 448px	<b>84.1</b>



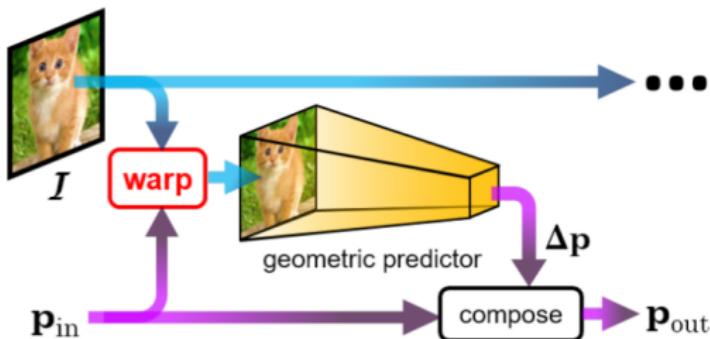
# Spatial Transformer Networks

- Но у STN есть недостатки – признаки надо менять, кое-что теряется
- Вырезаем (b), и если есть zoom-out, то в STN получаются краевые эффекты (d), а хочется вернуть информацию как на (e):



# Spatial Transformer Networks

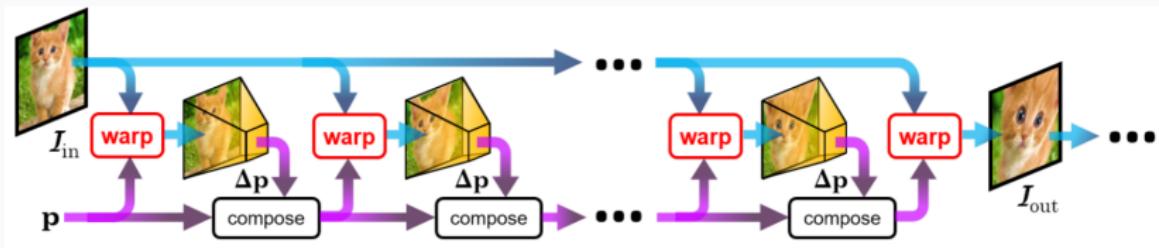
- (Lin, Lucey, 2017): Inverse Compositional Spatial Transformer Networks
- Давайте передавать не сами изменённые изображения (карты признаков), а исходные карты плюс обученные параметры преобразования:



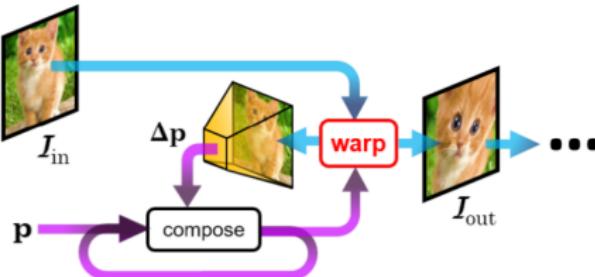
- Т.е. тут мы сохраняем геометрию в  $p$ , а в  $x$  сохраняем исходные пиксели, и их можно подставить обратно при необходимости.

# Spatial Transformer Networks

- Теперь такие преобразования можно спокойно конкатенировать в композицию:

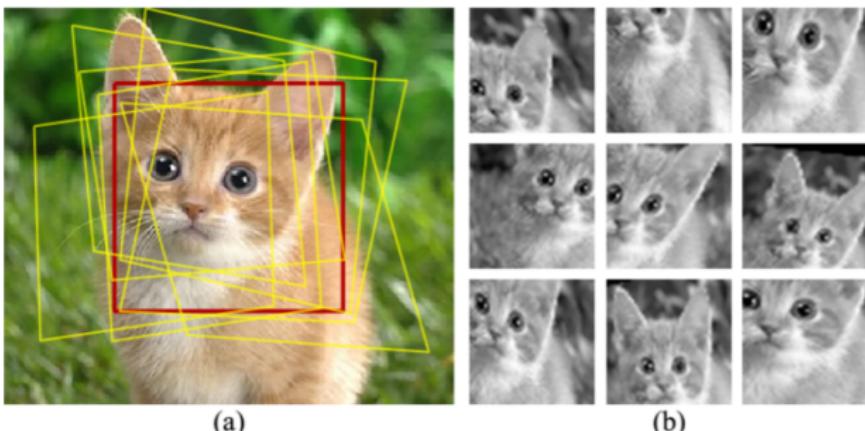


- И даже делать рекуррентные обновления преобразований:



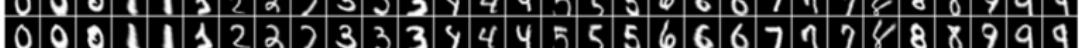
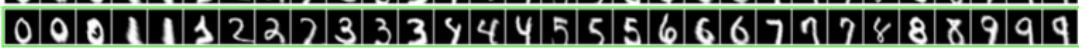
# Spatial Transformer Networks

- Получаются вполне семантически разумный набор преобразований:



# Spatial Transformer Networks

- Вот итеративный результат на MNIST со специальным шумом; внизу STN, у которого сплошные краевые эффекты:

init.	
(1)	
(2)	
(3)	
final	
STN	

# Spatial Transformer Networks

- А вот на более сложной задаче классификации:



- И вот как становится чётче средний преобразованный элемент класса по мере увеличения числа итераций:



# Spatial Transformer Networks

- (Detlefsen et al., 2018): Deep Diffeomorphic Transformer Networks, новое важное развитие идеи
- Давайте попробуем расширить класс допустимых преобразований, прямо диффеоморфизмы всякие будем обучать:

Original  
Accuracy: 0.78



Affine  
Accuracy: 0.84



Diffeomorphic  
Accuracy: 0.87

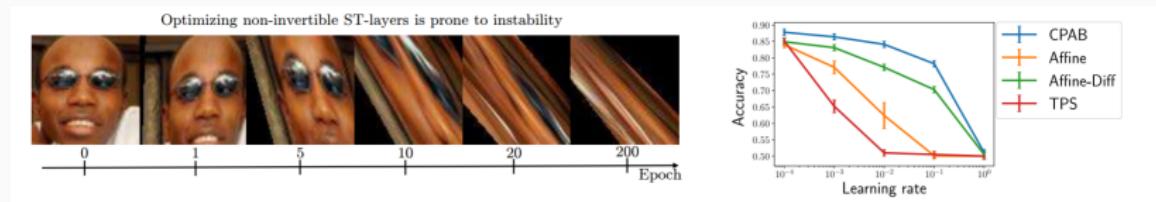


Affine+Diffeomorphic  
Accuracy: 0.89



# Spatial Transformer Networks

- Т.е. мы хотим, чтобы преобразования были дифференцируемы и имели дифференцируемое обратное
- Кстати, просто аффинные в обычном STN могут и разойтись:



- Как выделить семейство удачных диффеоморфизмов?  
Например, из аффинных преобразований (т.е. обратимые аффинные)?

# Spatial Transformer Networks

- Если  $\tilde{\mathbf{x}} = \begin{pmatrix} \mathbf{x} \\ 1 \end{pmatrix} \in \mathbb{R}^3$  – представление  $\mathbf{x} \in \mathbb{R}^2$  в проективных координатах, то аффинное преобразование – это

$$\begin{pmatrix} {}^T\theta(\mathbf{x}) \\ 1 \end{pmatrix} = \begin{pmatrix} \theta_1 & \theta_2 & \theta_3 \\ \theta_4 & \theta_5 & \theta_6 \\ 0 & 0 & 1 \end{pmatrix} \tilde{\mathbf{x}}.$$

- Чтобы было обратимое преобразование, можно, например, начать с поля скоростей  $v^\theta$ :

$$v^\theta : \mathbf{x} \mapsto A\tilde{\mathbf{x}}, \text{ где } A = \begin{pmatrix} \theta_1 & \theta_2 & \theta_3 \\ \theta_4 & \theta_5 & \theta_6 \end{pmatrix},$$

а затем взять матричную экспоненту:

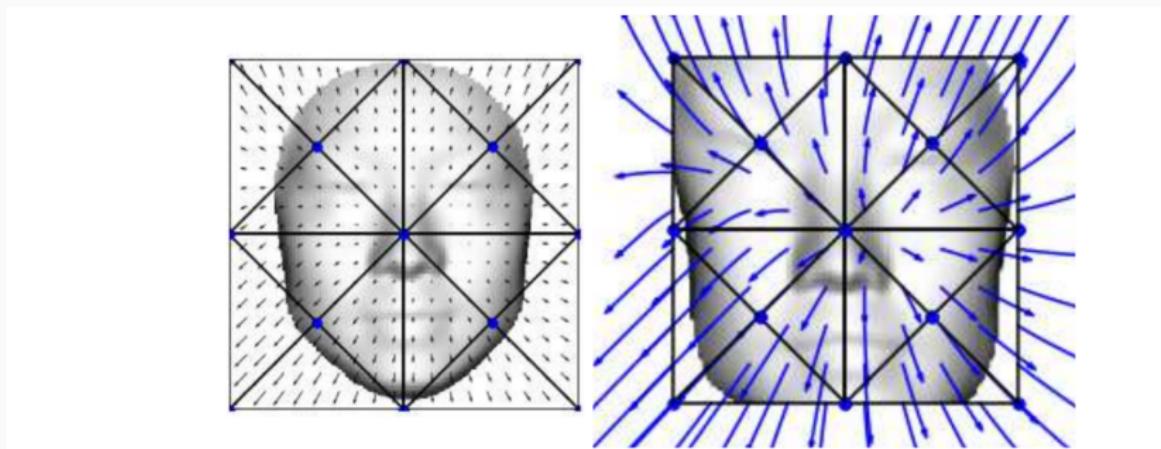
$$\begin{pmatrix} {}^T\theta(\mathbf{x}) \\ 1 \end{pmatrix} = \exp(\tilde{A})\tilde{\mathbf{x}}, \text{ где } \tilde{A} = \begin{pmatrix} \theta_1 & \theta_2 & \theta_3 \\ \theta_4 & \theta_5 & \theta_6 \\ 0 & 0 & 0 \end{pmatrix}.$$

- Тогда получится именно аффинный диффеоморфизм, решение интегрального уравнения

$$\phi^\theta(\mathbf{x}; 1) = \mathbf{x} + \int_0^1 v^\theta(\phi^\theta(\mathbf{x}; \tau)) d\tau$$

# Spatial Transformer Networks

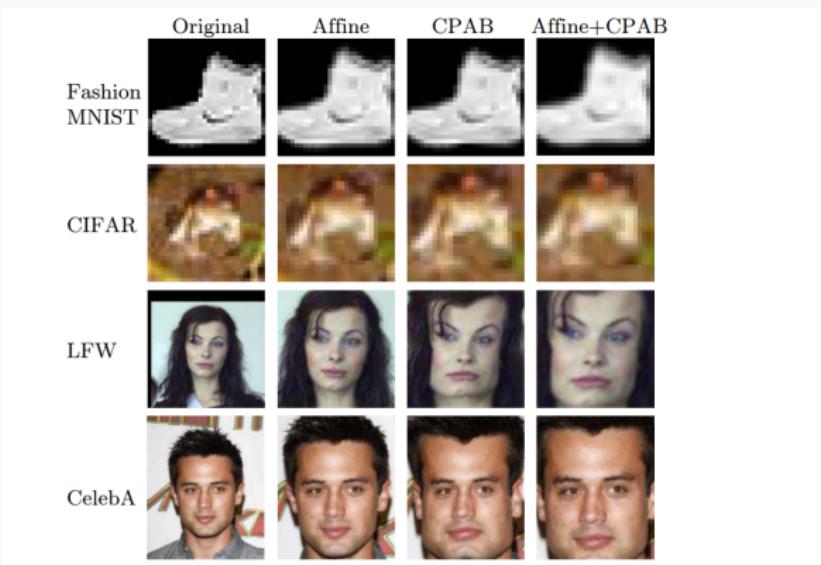
- Для этого подхода всё равно, какое поле скоростей, можно взять и нелинейное. Так и будем делать: выбирать выразительное поле скоростей  $v^\theta$ .
- На самом деле – будем делать кусочно-аффинное  $v^\theta$ , разбивая картинку на ячейки; CPAB transformations (continuous piecewise-affine based):



- Слева поле скоростей, справа результат интегрирования.

# Spatial Transformer Networks

- Обучаются интересные преобразования, результаты получше, но пока это выглядит как очень интересный work in progress:

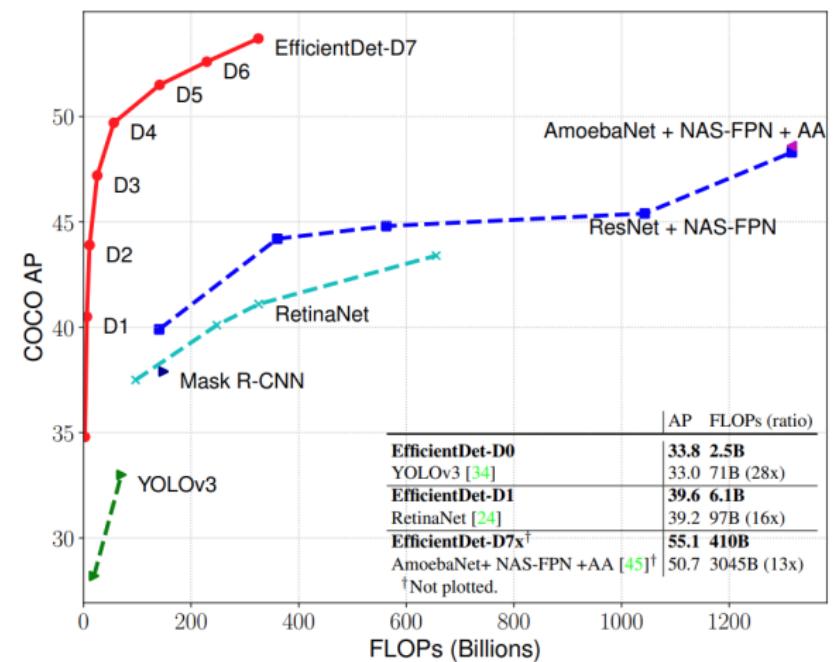


## Последние новости

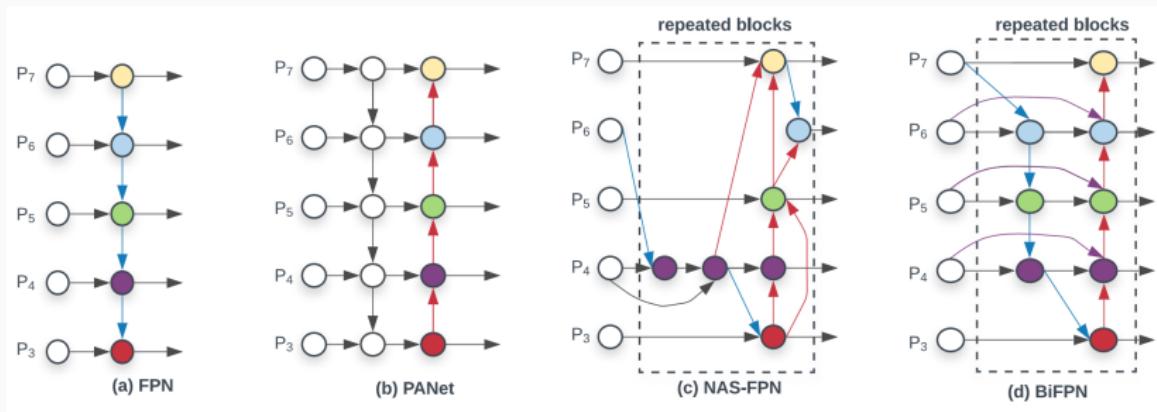
---

# EfficientDet

- (Tan et al., 2019): EfficientDet – neural architecture search  
добрался и до object detection, конечно

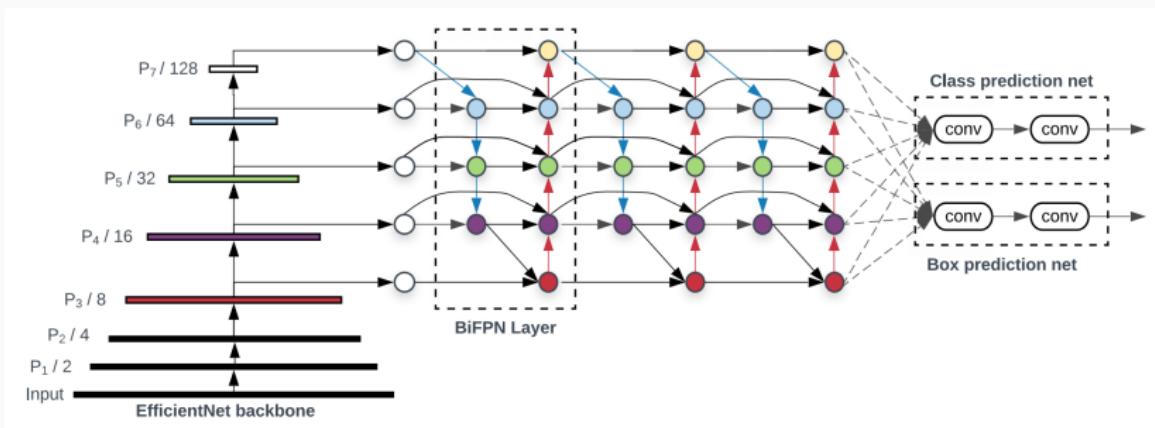


- Новые типы блоков для пирамид признаков:



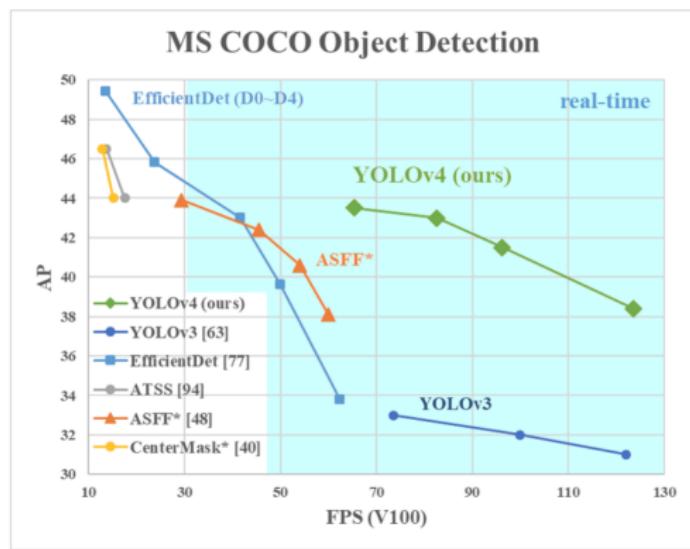
# EfficientDet

- А дальше просто подставим EfficientNet в качестве backbone:



# YOLOv4 и YOLOv5

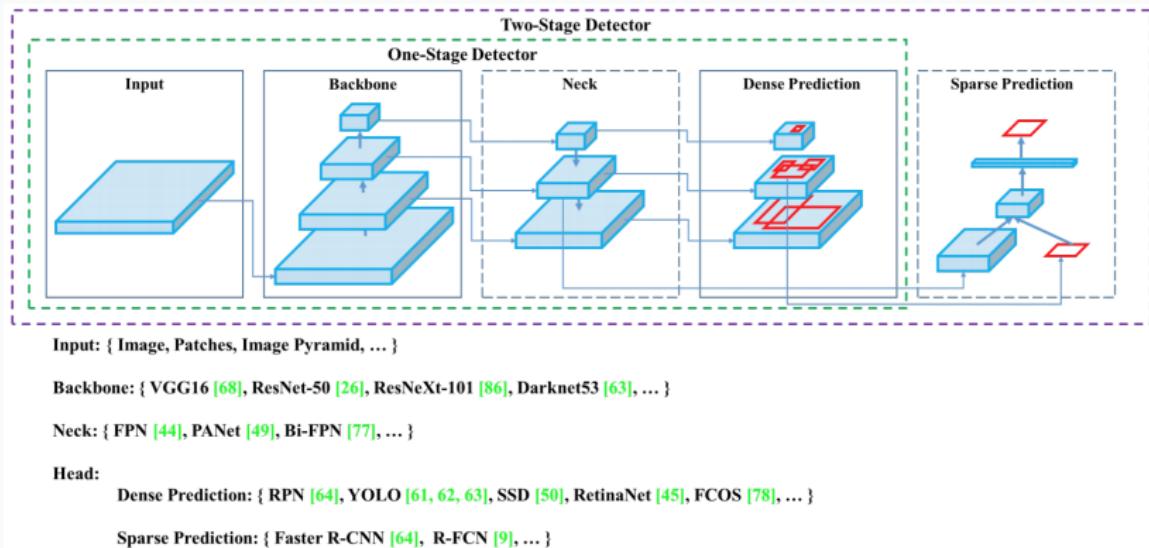
- А теперь любопытная история из 2020 года
- (Bochkovskiy et al., 23 Apr 2020): YOLOv4



- Что они сделали?

# YOLOv4 и YOLOv5

- Основательно подошли к перебору разных вариантов:



# YOLOv4 и YOLOv5

- Инкрементальная, но мощная идея bag of freebies / bag of specials:
  - сети тренируются оффлайн, поэтому можно сильно усложнять процесс тренировки, на скорость inference это не повлияет, это «бесплатно»;
  - в основном разные методы аугментации (self-adversarial training!) плюс новые функции ошибки для регрессии (IoU loss и варианты);
  - а ещё есть bag of specials, которые чего-то стоят на inference, но по возможности дёшевы;
  - это функции активации (Mish), постпроцессинг bbox'ов, механизм внимания (довольно простой) и т.д.

# YOLOv4 и YOLOv5

- Итого всё понемножку добавляет и становится лучше.

Table 4: Ablation Studies of Bag-of-Freebies. (CSPResNeXt50-PANet-SPP, 512x512).

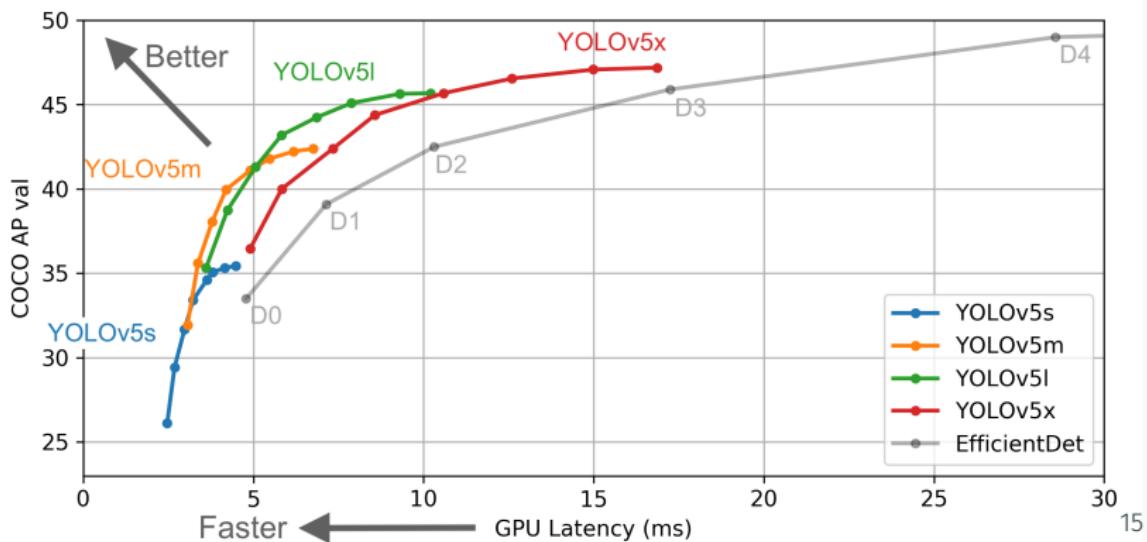
S	M	IT	GA	LS	CBN	CA	DM	OA	loss	AP	AP <sub>50</sub>	AP <sub>75</sub>
✓									MSE	38.0%	60.0%	40.8%
	✓								MSE	37.7%	59.9%	40.5%
		✓							MSE	<b>39.1%</b>	<b>61.8%</b>	<b>42.0%</b>
			✓						MSE	36.9%	59.7%	39.4%
				✓					MSE	<b>38.9%</b>	<b>61.7%</b>	<b>41.9%</b>
					✓				MSE	33.0%	55.4%	35.4%
						✓			MSE	<b>38.4%</b>	<b>60.7%</b>	<b>41.3%</b>
							✓		MSE	<b>38.7%</b>	<b>60.7%</b>	<b>41.9%</b>
								✓	MSE	35.3%	57.2%	38.0%
✓									GIoU	<b>39.4%</b>	59.4%	<b>42.5%</b>
✓									DIoU	<b>39.1%</b>	58.8%	<b>42.1%</b>
✓									CloU	<b>39.6%</b>	59.2%	<b>42.6%</b>
✓	✓	✓	✓	✓					CloU	<b>41.5%</b>	<b>64.0%</b>	<b>44.8%</b>
✓	✓	✓	✓	✓				✓	CloU	36.1%	56.5%	38.4%
✓	✓	✓	✓	✓				✓	MSE	<b>40.3%</b>	<b>64.0%</b>	<b>43.1%</b>
✓	✓	✓	✓	✓				✓	GIoU	<b>42.4%</b>	<b>64.4%</b>	<b>45.9%</b>
✓	✓	✓	✓	✓				✓	CloU	<b>42.4%</b>	<b>64.4%</b>	<b>45.9%</b>

Table 5: Ablation Studies of Bag-of-Specials. (Size 512x512).

Model	AP	AP <sub>50</sub>	AP <sub>75</sub>
CSPResNeXt50-PANet-SPP	42.4%	64.4%	45.9%
CSPResNeXt50-PANet-SPP-RFB	41.8%	62.7%	45.1%
CSPResNeXt50-PANet-SPP-SAM	<b>42.7%</b>	<b>64.6%</b>	<b>46.3%</b>
CSPResNeXt50-PANet-SPP-SAM-G	41.6%	62.7%	45.0%
CSPResNeXt50-PANet-SPP-ASFF-RFB	41.1%	62.6%	44.4%

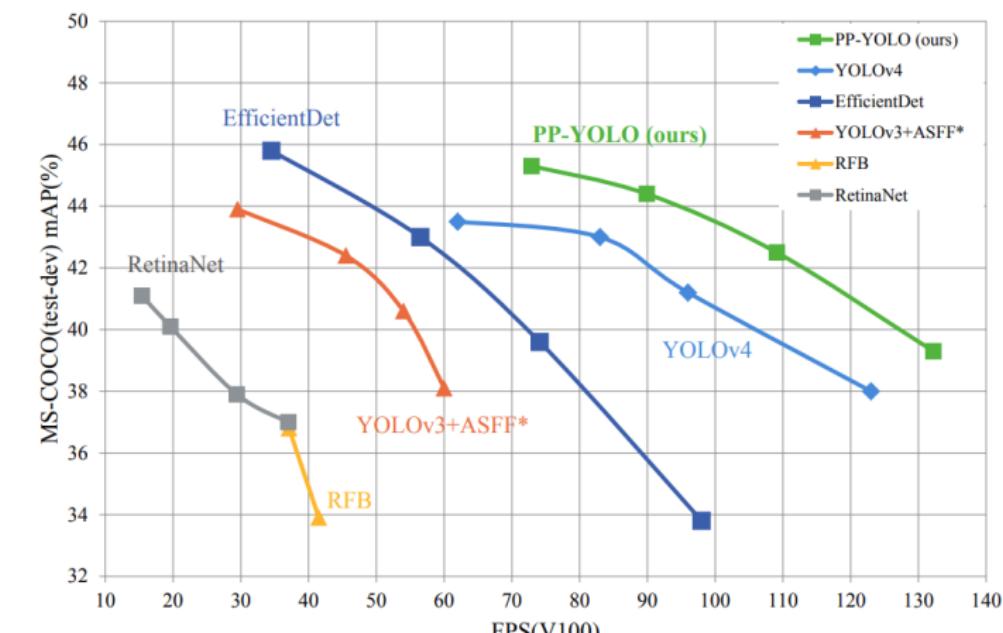
## YOLOv4 и YOLOv5

- Но история не в этом, а в том, что меньше чем через два месяца появился YOLOv5!
- От совсем других исследователей (Glenn Jocher, компания Ultralytics), статьи нет до сих пор, но есть код и результаты красивые: всё гораздо быстрее с тем же качеством.



# YOLOv4 и YOLOv5

- Чем это закончится, пока непонятно, а ведь есть ещё и PP-YOLO от Baidu (Long et al., 3 Aug 2020), который основан на YOLOv3, тоже соединяет разные новые трюки, и работает, говорят, лучше YOLOv4.



Спасибо!

Спасибо за внимание!

