Neural Object Detection

Ilya Zakharkin | Deep Learning School

About

МФТИ

Skoltech

Skolkovo Institute of Science and Technology

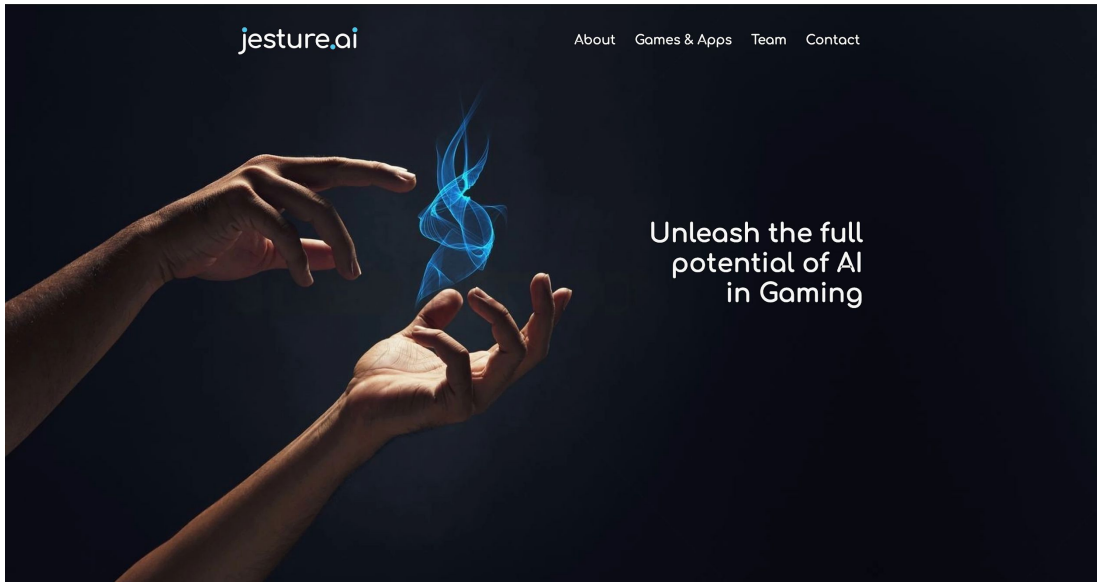
Deep Learning School

Физтех-школа
Прикладной
Математики
И Информатики

ШКОЛА АНАЛИЗА ДАННЫХ

About

SAMSUNG



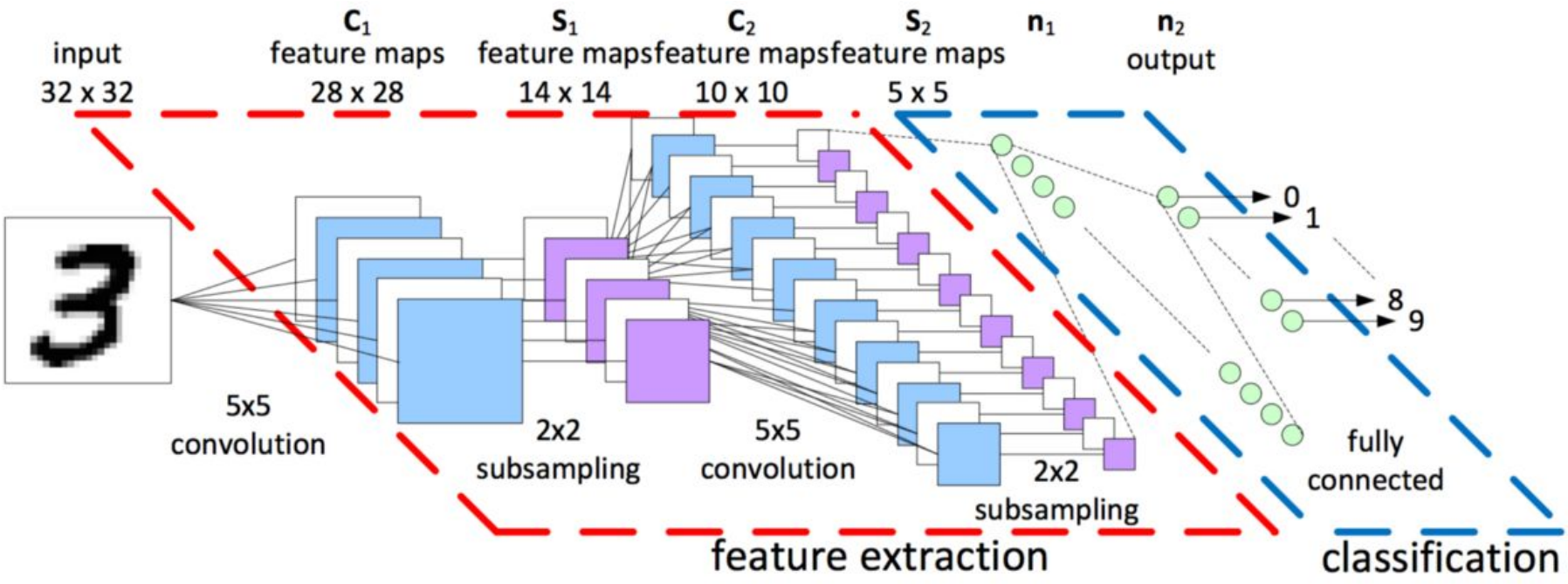
https://www.jesture.ai/

Lecture plan

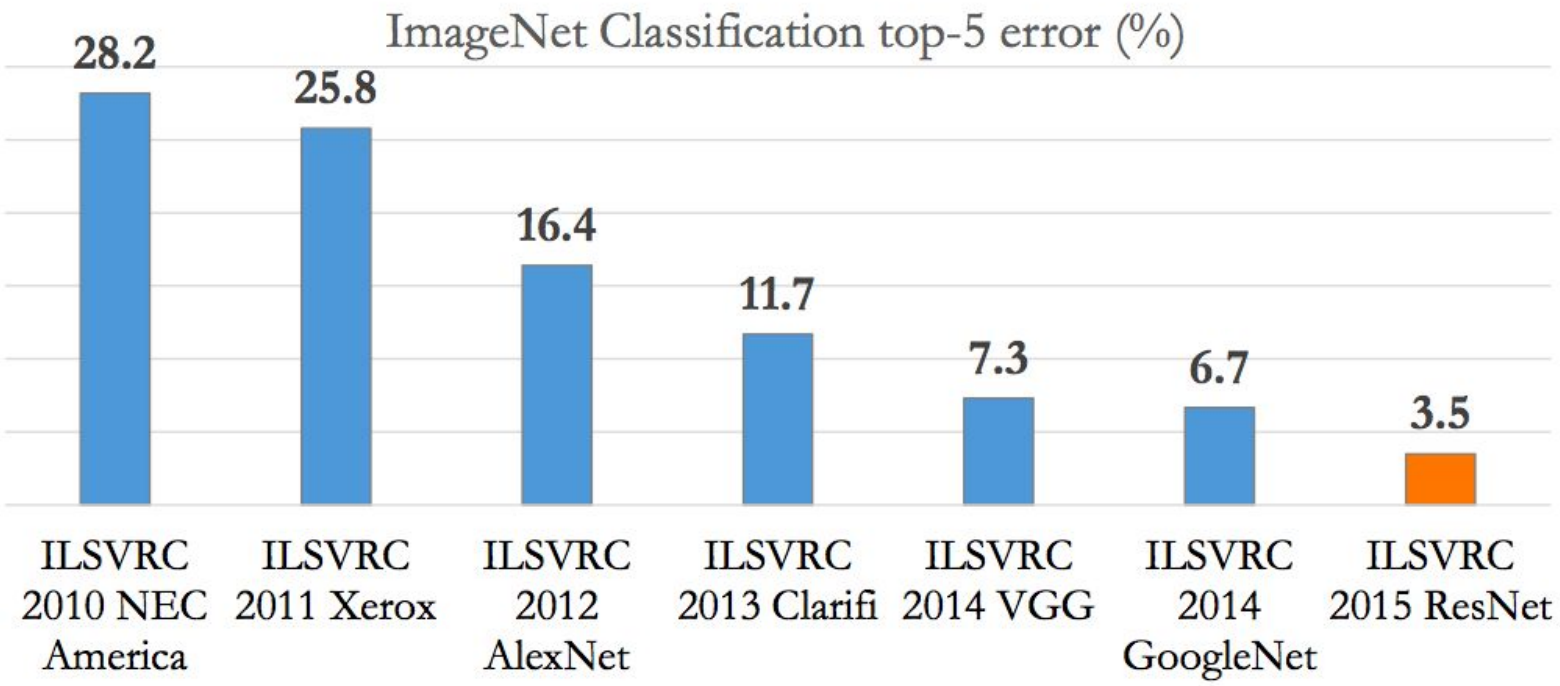
1. Classification recap
2. Object Detection basics
3. Two-stage (proposal-based) methods
4. One-stage (single-shot) methods
5. Modern approaches & tasks

Image Classification

Recap


input
32 x 32
C1
feature maps
28 x 28
S1
feature maps
14 x 14
C2
feature maps
10 x 10
S2
feature maps
5 x 5
n1
n2
output
0
1
8
9
5x5
convolution
2x2
subsampling
5x5
convolution
2x2
subsampling
fully
connected
feature extraction
classification

Classification: ILSVRC



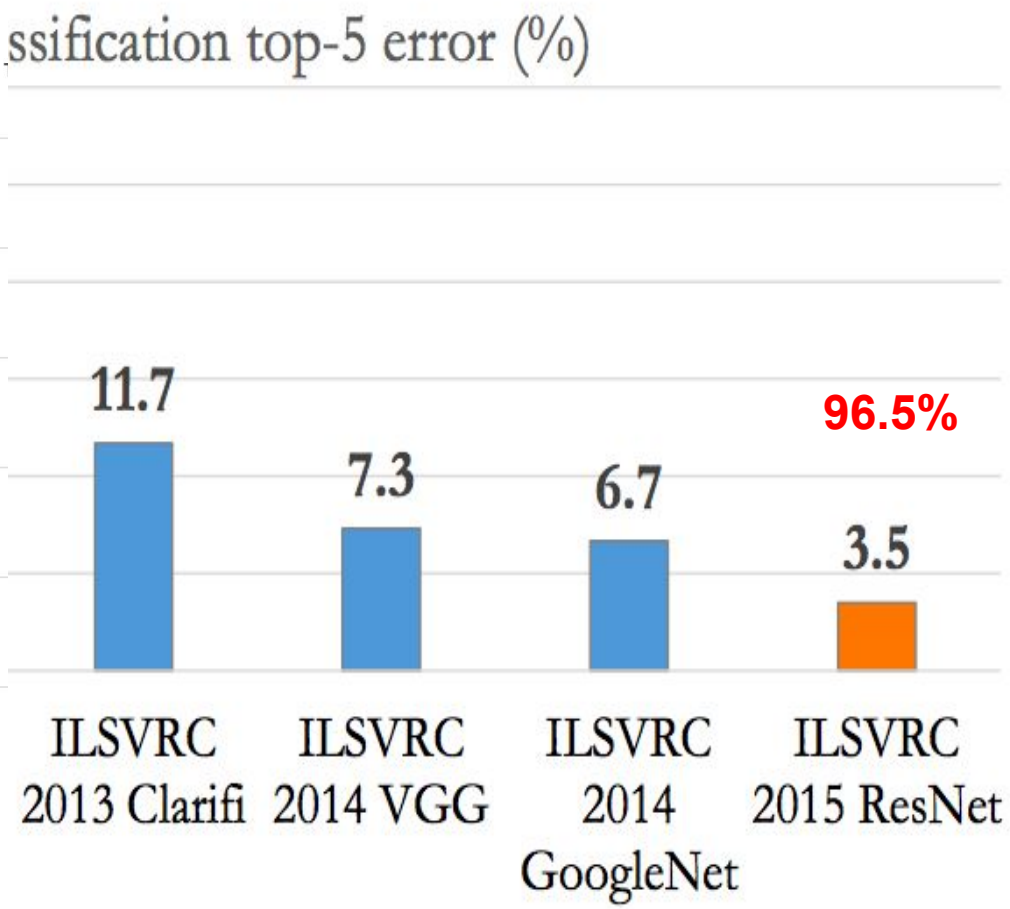
Classification: ILSVRC

RANK	METHOD	TOP 1 ACCURACY	TOP 5 ACCURACY	NUMBER OF PARAMS	EXTRA TRAINING DATA	PAPER TITLE	YEAR	PAPER	CODE
1	FixEfficientNet-L2	88.5%	98.7%	480M	✓	Fixing the train-test resolution discrepancy: FixEfficientNet	2020		
2	NoisyStudent (EfficientNet-L2)	88.4%	98.7%	480M	✓	Self-training with Noisy Student improves ImageNet classification	2020		
3	BiT-L (ResNet)	87.54%	98.46%	928M	✓	Big Transfer (BiT): General Visual Representation Learning	2019		
4	NoisyStudent (EfficientNet-L2)	87.4%	98.2%	480M	✓	Self-training with Noisy Student improves ImageNet classification	2019		
5	FixEfficientNet-B7	87.1%	98.2%	66M	✓	Fixing the train-test resolution discrepancy: FixEfficientNet	2020		

https://paperswithcode.com/sota/image-classification-on-imagenet

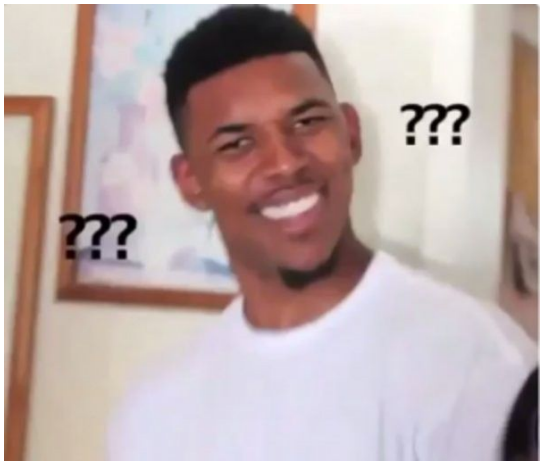
Classification: ILSVRC

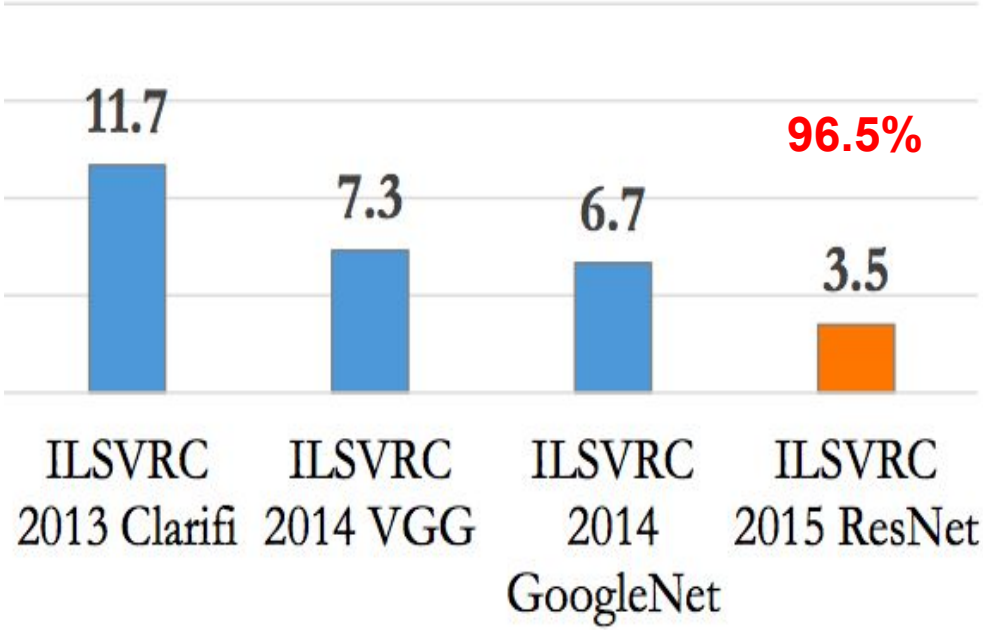
RANK	METHOD	TOP 1 ACCURACY	TOP 5 ACCURACY	NUMBER OF PARAMS
1	FixEfficientNet-L2	88.5%	98.7%	480M
2	NoisyStudent (EfficientNet-L2)	88.4%	98.7%	480M
3	BiT-L (ResNet)	87.54%	98.46%	928M
4	NoisyStudent (EfficientNet-L2)	87.4%	98.2%	480M
5	FixEfficientNet-B7	87.1%	98.2%	66M



Classification: ILSVRC

	(320x320)					
66	ResNet-152	80.62%	95.51%	Deep Residual Learning for Image Recognition	2015	
67	ResNet-200	80.6%	95.3%	Fast AutoAugment	2019	





Classification: ILSVRC

method	top-1 err.	top-5 err
VGG [41] (ILSVRC'14)	-	8.43†
GoogLeNet [44] (ILSVRC'14)	-	7.89
VGG [41] (v5)	24.4	7.1
PReLU-net [13]	21.59	5.71
BN-inception [16]	21.99	5.81
ResNet-34 B	21.84	5.71
ResNet-34 C	21.53	5.60
ResNet-50	20.74	5.25
ResNet-101	19.87	4.60
ResNet-152	**19.38**	**4.49**

Table 4. Error rates (%) of **single-model** results on the Imag validation set (except † reported on the test set).

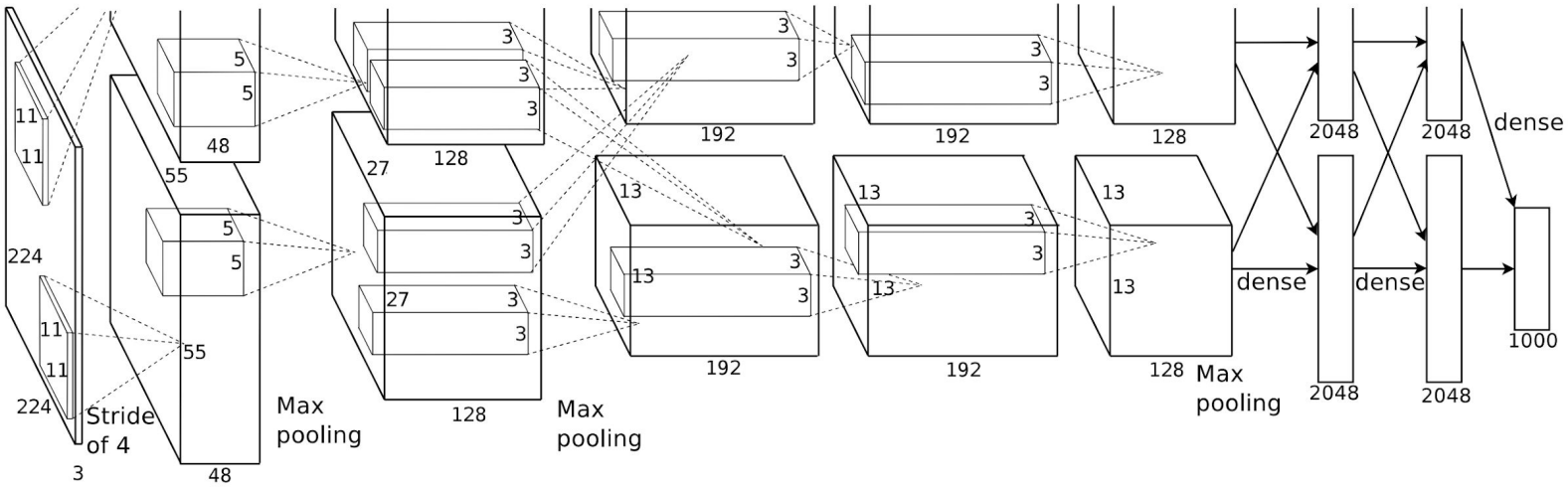
method	top-5 err. (**test**)
VGG [41] (ILSVRC'14)	7.32
GoogLeNet [44] (ILSVRC'14)	6.66
VGG [41] (v5)	6.8
PReLU-net [13]	4.94
BN-inception [16]	4.82
ResNet (ILSVRC'15)	**3.57**

Table 5. Error rates (%) of **ensembles**. The top-5 error is on the test set of ImageNet and reported by the test server.

https://arxiv.org/pdf/1512.03385.pdf

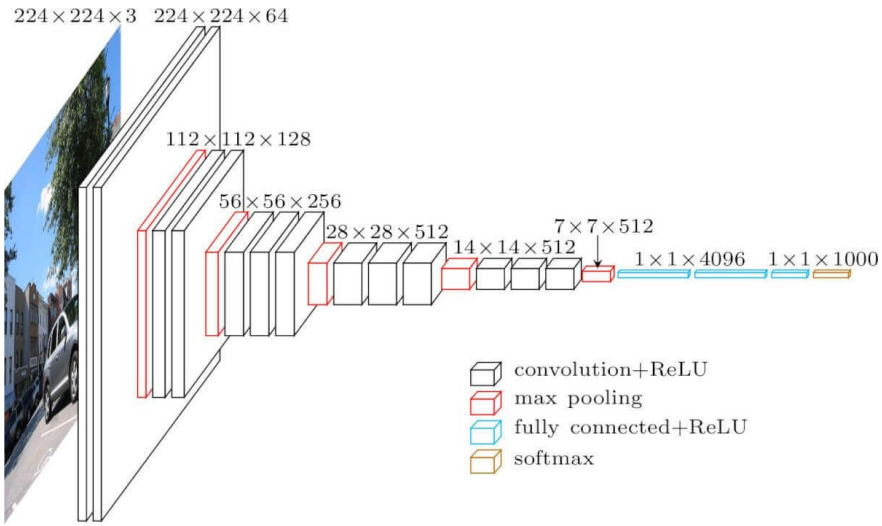
Classification: AlexNet

- Idea: Conv -> Conv (= ConvBlock) + Deeper + GPU
- Paper
- Code: available in every framework



Classification: VGG

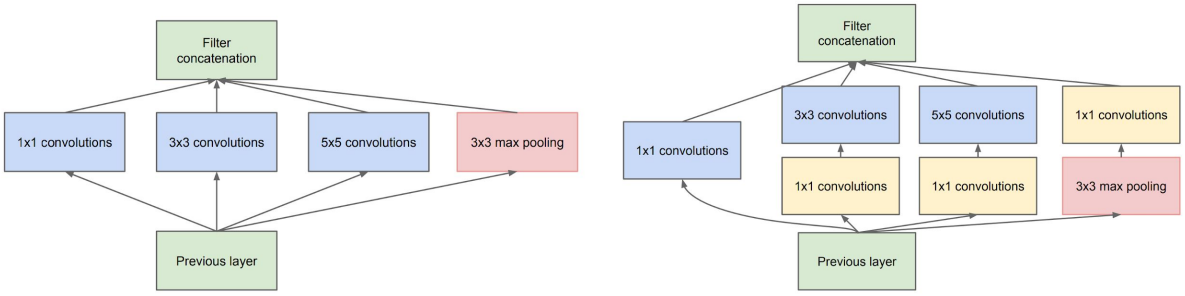
- Idea: Several conv blocks (deeper net)
- Paper
- Code: available in every framework



ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

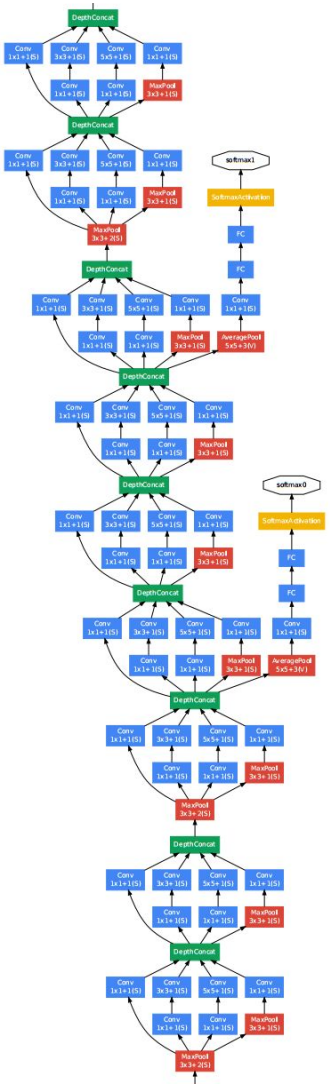
Classification: Inception (v1)

- Idea: Inception-block + 1x1 conv + deep supervision
- Paper
- Code: available in every framework



(a) Inception module, naïve version (b) Inception module with dimension reductions

Figure 2: Inception module



Classification: ResNet

- Idea: Residual connections (Shortcuts)
- Paper
- Code: available in every framework

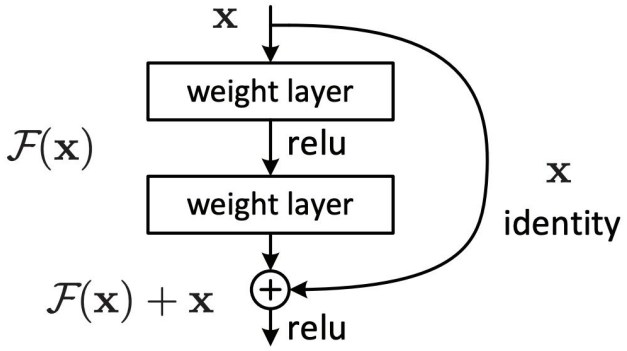
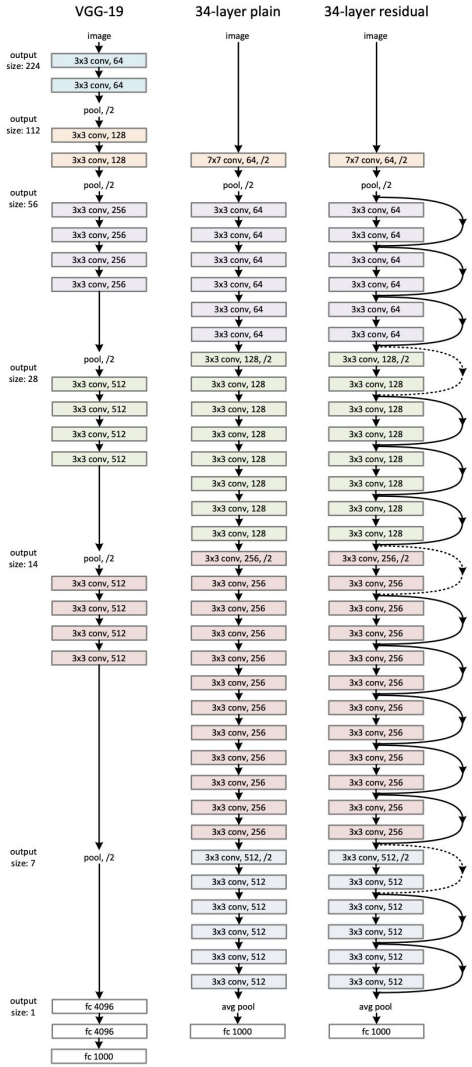
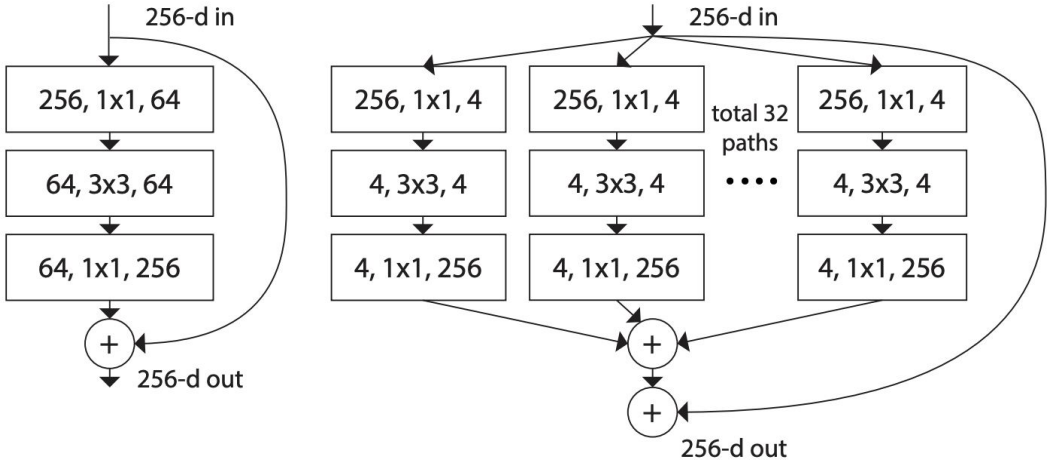


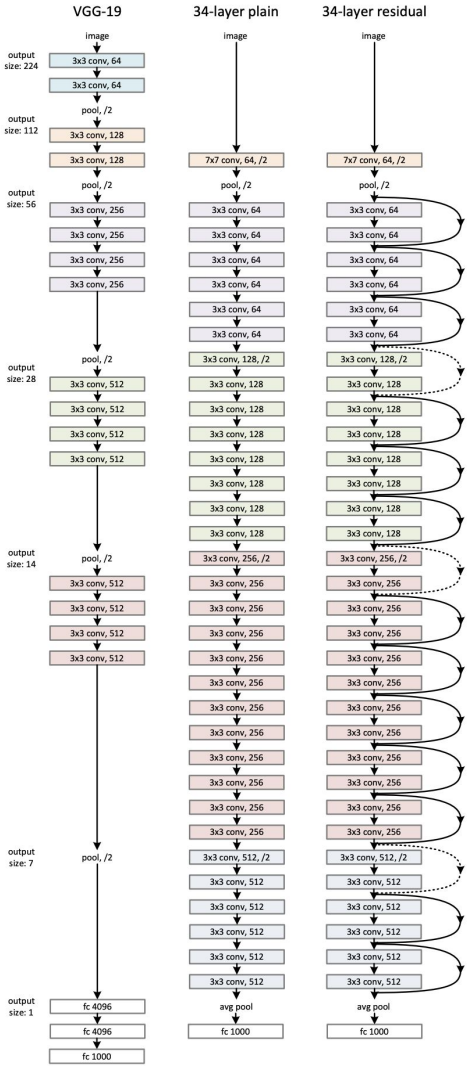
Figure 2. Residual learning: a building block.



Classification: ResNeXt

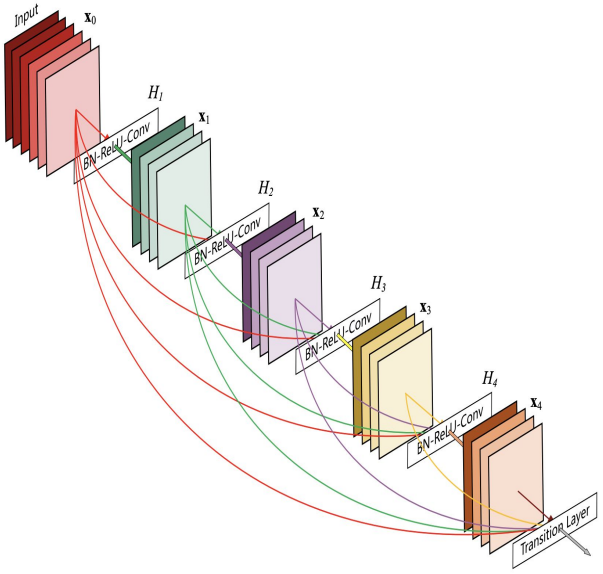
- Idea: Aggregated residual blocks
- Paper
- Code
- Bottom: Cardinality (C) = 32, d = 4

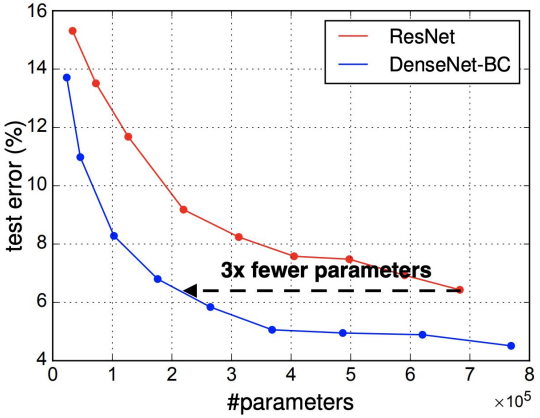


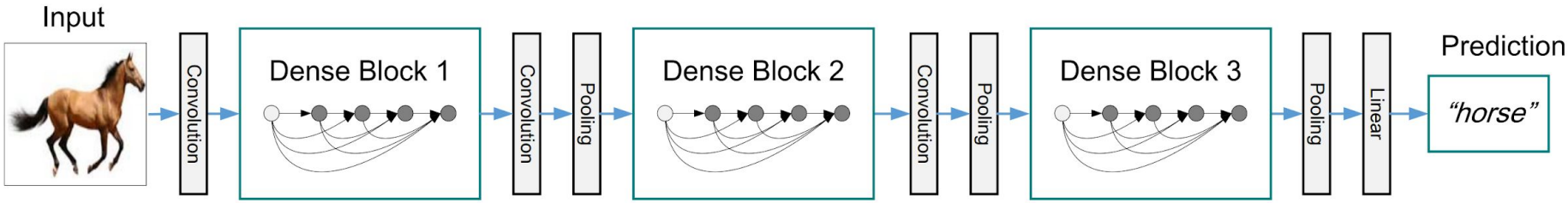


Classification: DenseNet

- Idea: Dense block
- Paper
- Code:
  torchvision / tf.models

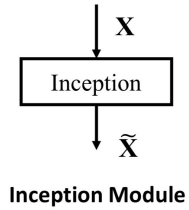




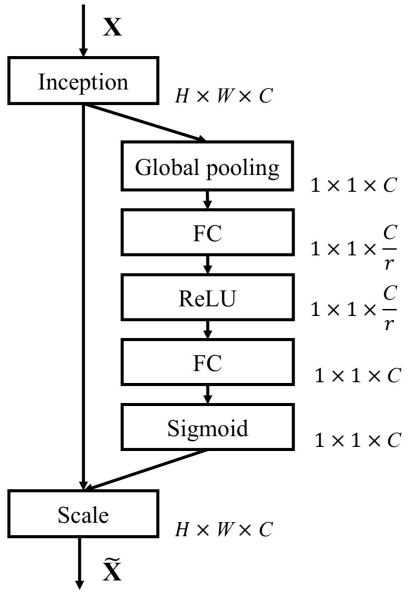


Classification: Squeeze-and-Excitation Net (SENet)

- Idea: SE-block
- Paper
- Code



Inception Module



SE-Inception Module

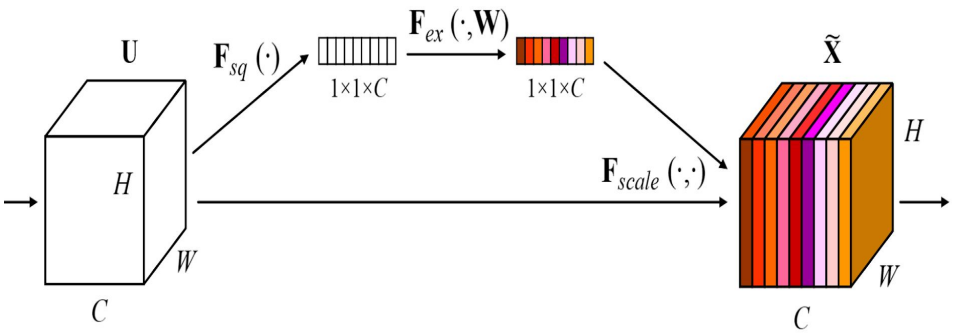
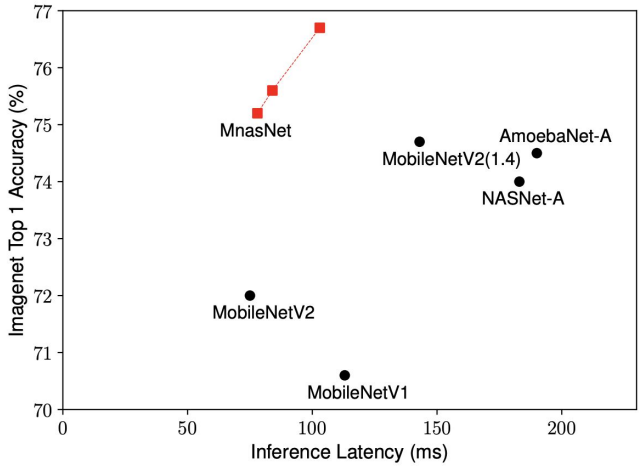
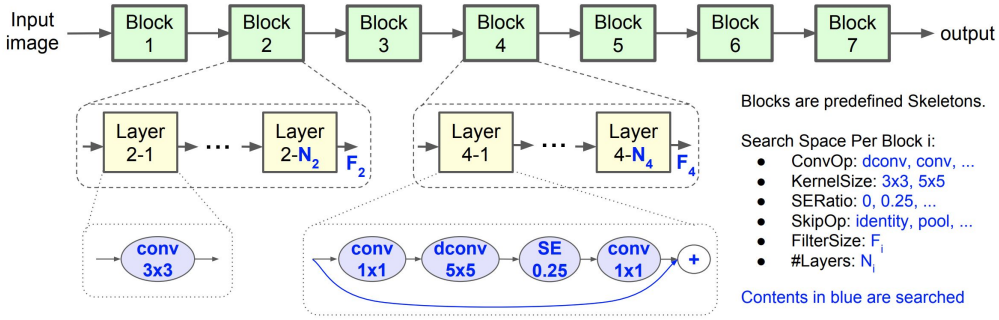


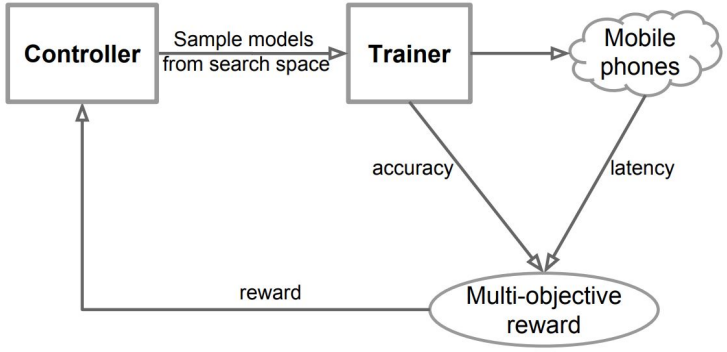
Figure 1: A Squeeze-and-Excitation block.

Classification: MnasNet

- Idea: AutoML
  (Neural Architecture Search, NAS)
- Paper
- Code
- Speed ~100ms on Mobile CPU (Pixel)

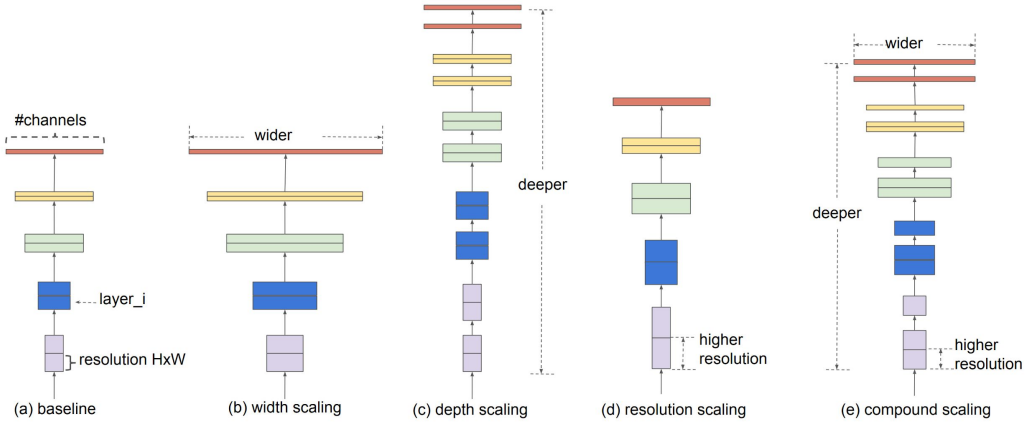


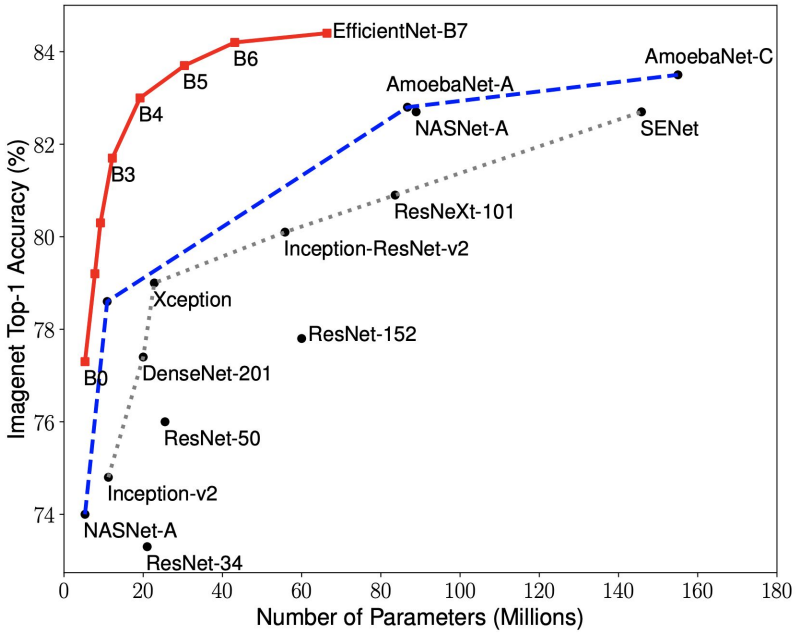




Classification: EfficientNet (2019)

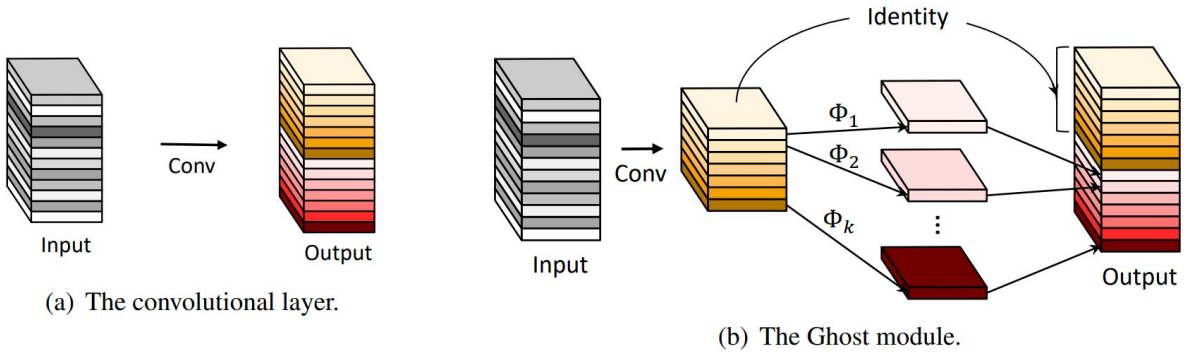
- Idea: AutoML + Compound scaling
- Paper
- Code

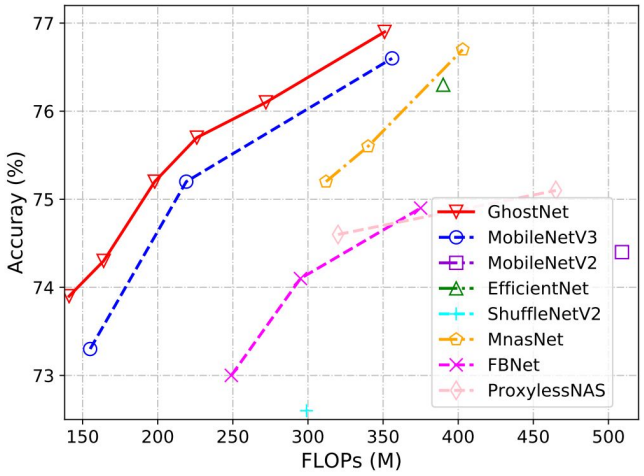


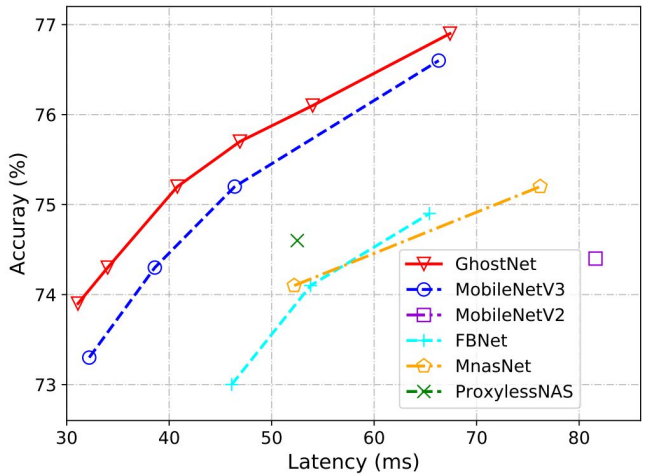


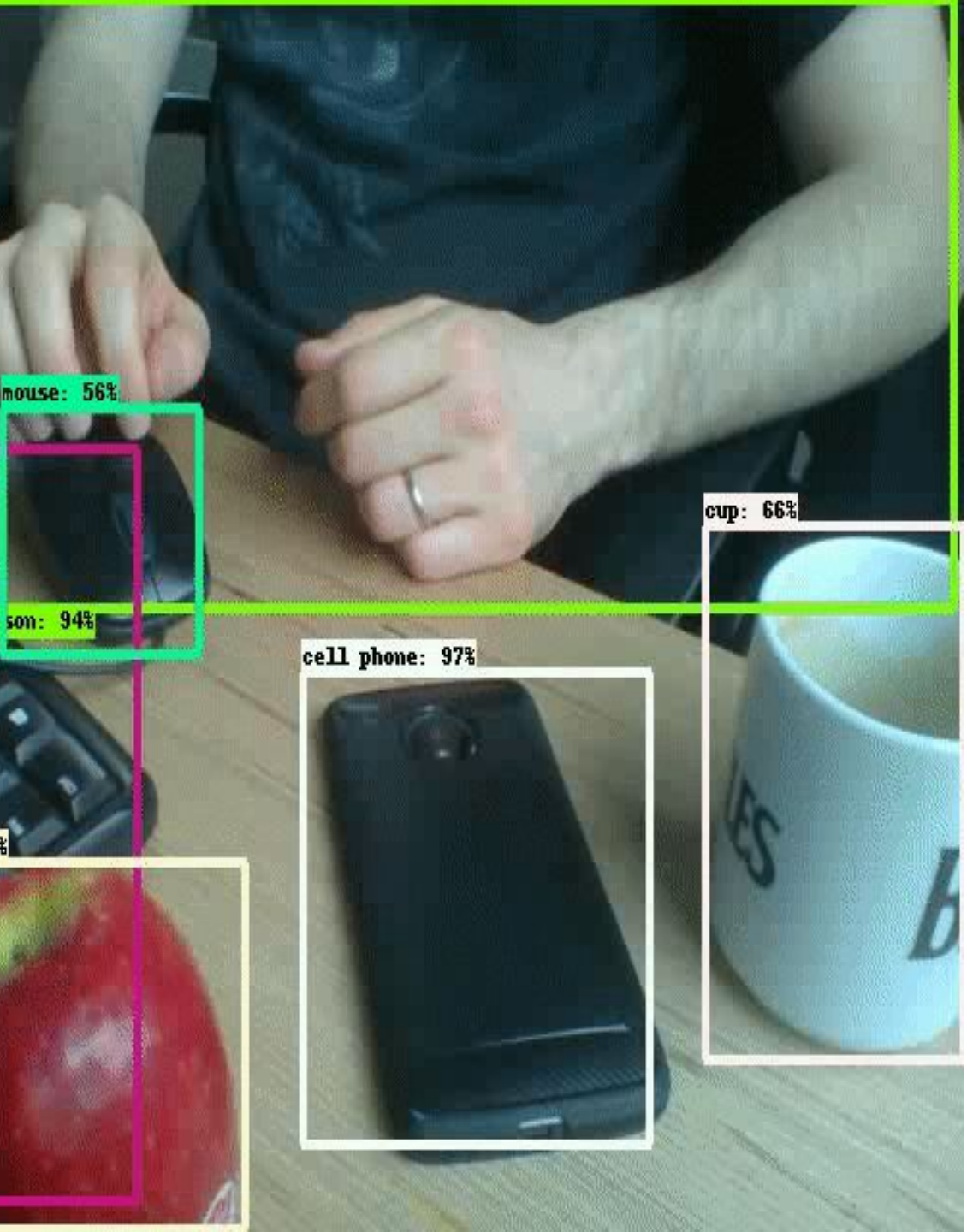
Classification: GhostNet (2019-2020)

- Idea: Ghost module
- Paper
- Code









Detection

Datasets & Metrics

Object Detection: Task statement

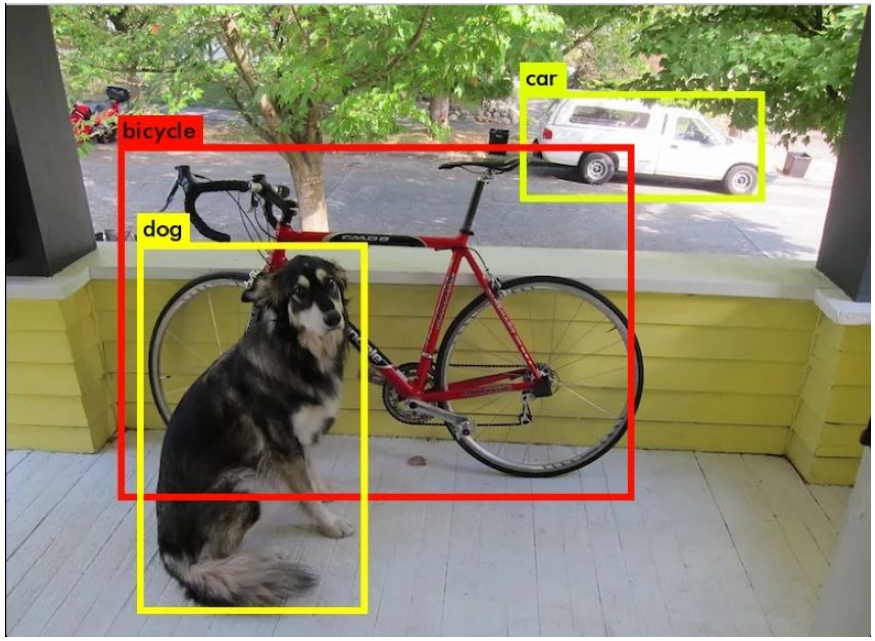
Input: Image (np.array)

Output: “bounding boxes with class labels”

Each BBox is:

- Top-Left corner: (*x_tl*, *y_tl*)
- Bottom-Right corner: (*x_br*, *y_br*)
- Class name
- Confidence (probability of this class)

⇒ 6 numbers per each box

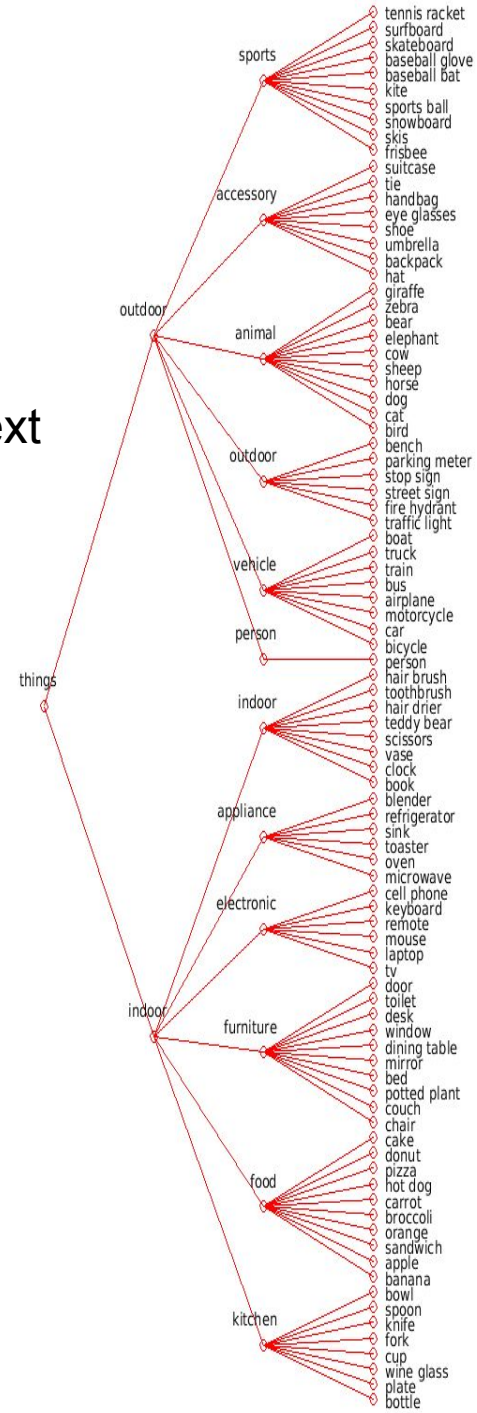


Object Detection: Data

1. PASCAL VOC 2012
2. MS COCO 2017
3. ImageNet
4. Google Open Images v6
5. Epic Kitchens

MS COCO 2017

- MicroSoft Common Objects in COntext
- 80 classes
- Train: ~120k images ~18GB
- Val: ~5k images ~1GB
- Test: ~40k images ~6GB
- Vary in size and quality
- Paper



Google Open Images v6

Table 2: Boxes.

	Train	Validation	Test	# Classes
Images	1,743,042	41,620	125,436	-
Boxes	14,610,229	303,980	937,327	600

Details:

https://storage.googleapis.com/openimages/web/factsfigures.html

Kaggle:

https://www.kaggle.com/c/open-images-object-detection-rvc-2020

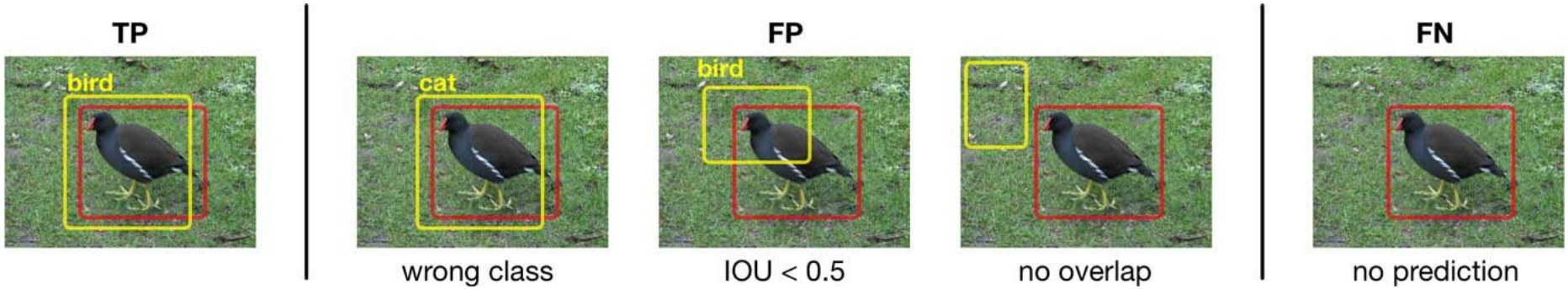
Intersection over Union (IoU, Jaccard index)

Ground truth

Prediction

$$IoU = \frac{\text{area of overlap}}{\text{area of union}}$$

Positives & Negatives



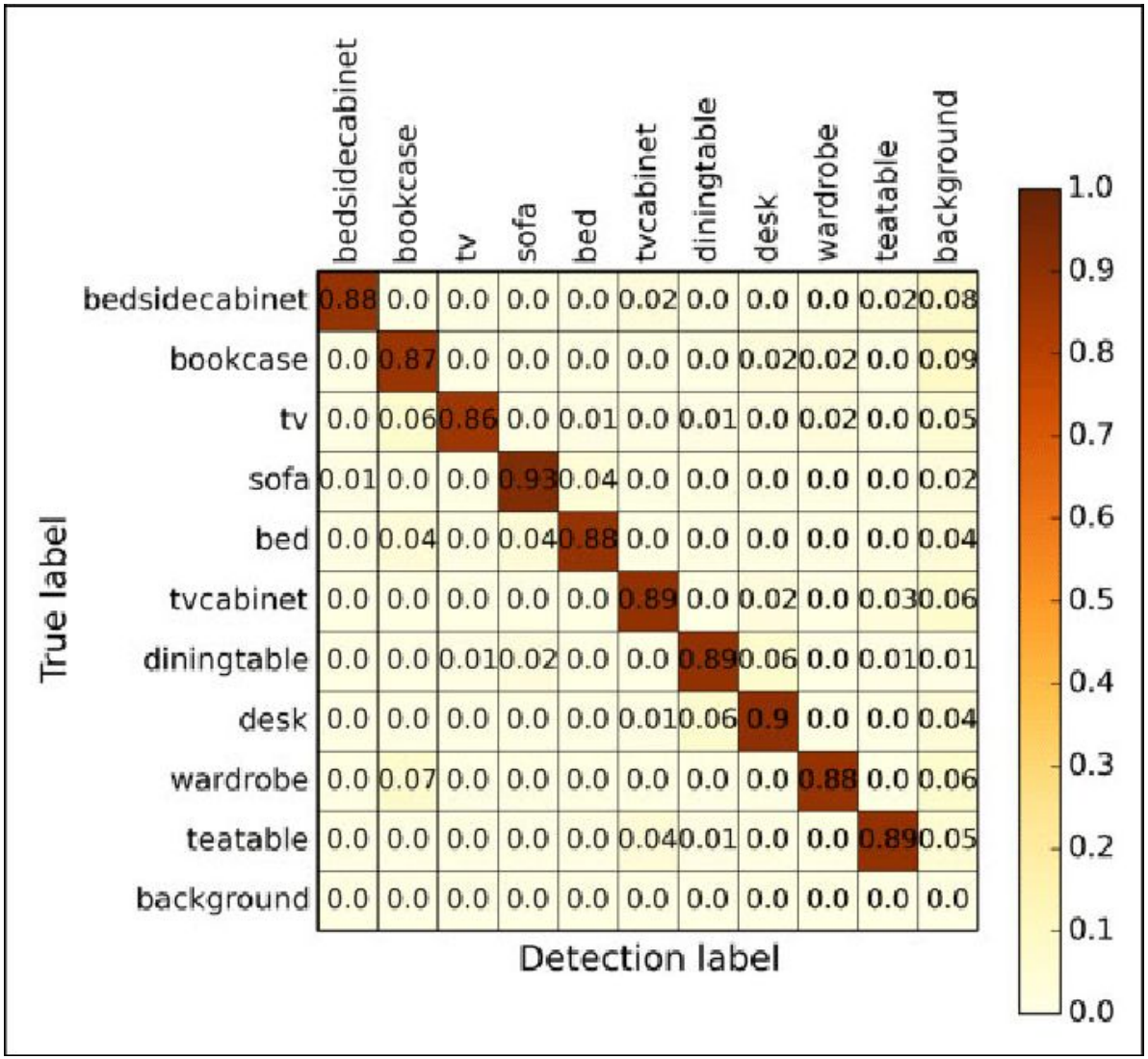
https://machinethink.net/images/object-detection/tp-fp-fn@2x.jpg

Precision, recall

$$\textbf{Precision} = \frac{True\ Positives}{True\ Positives + False\ Positives}$$

$$\textbf{Recall} = \frac{True\ Positives}{True\ Positives + False\ Negatives}$$

Confusion matrix



https://www.researchgate.net/publication/338728961/figure/fig2/AS:849968428507136@1579659673792/The-confusion-matrix-of-object-detection.png

mean Average Precision (mAP)

$$AP = \frac{1}{11} \times \left(AP_r(0) + AP_r(0.1) + \ldots + AP_r(1.0) \right)$$

Good explanation

Let’s invent our own detector

Let’s invent our own detector

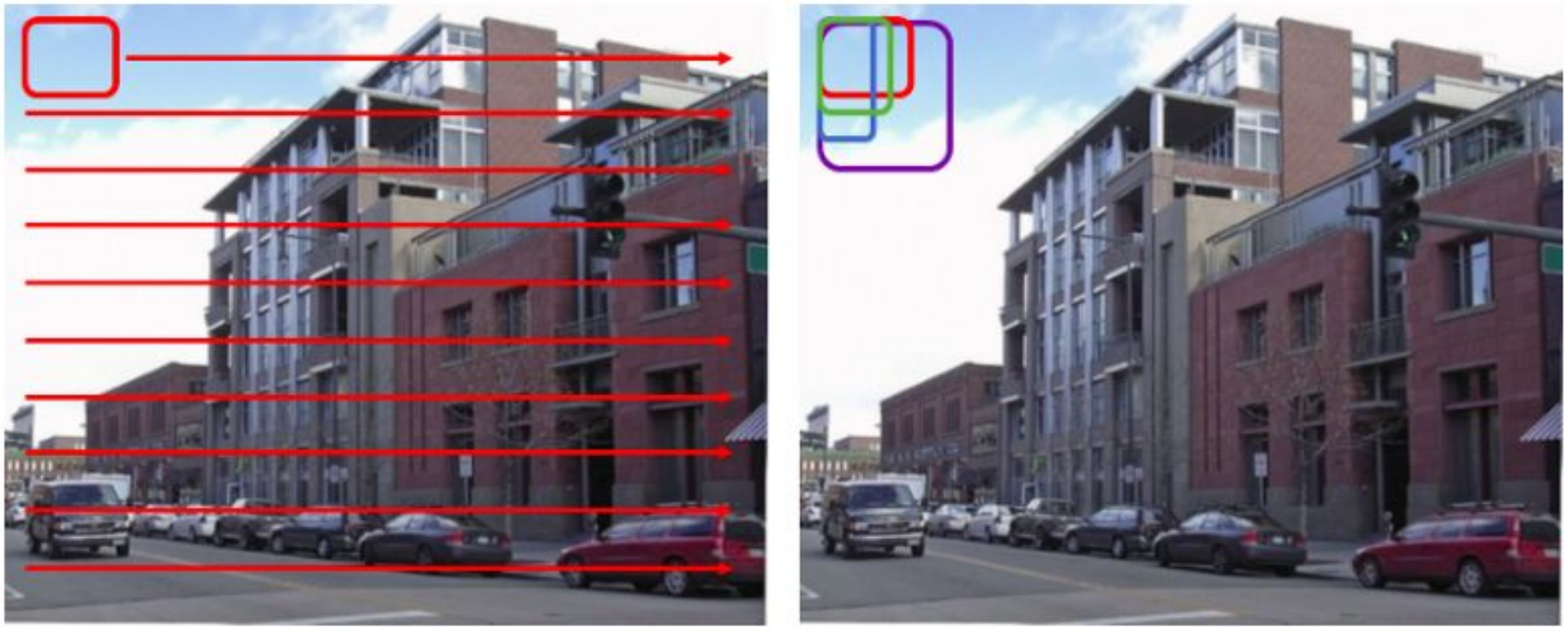


Illustration of Sliding Window (Left) with Different Aspect Ratios and Sizes (Right)

Summary

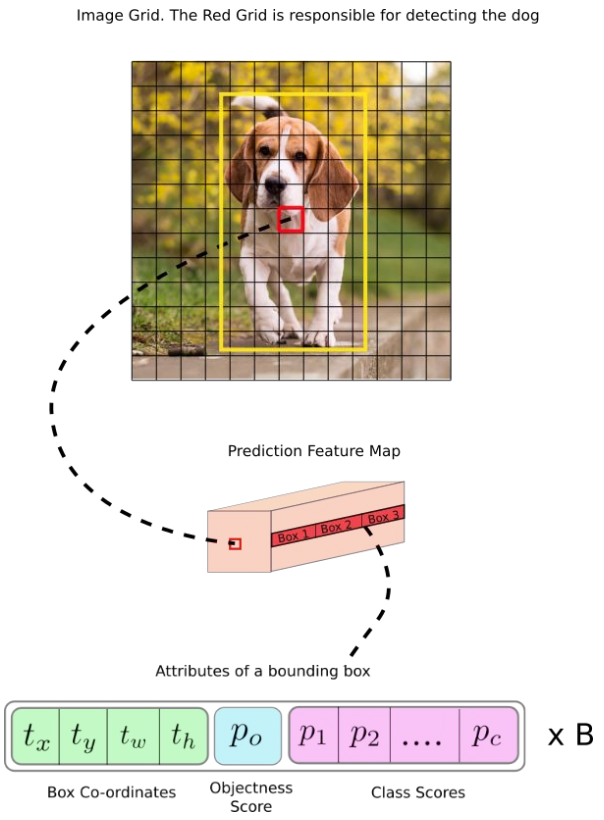
1. Detection task is not easy
2. Datasets are quite large and accessible
3. Metrics are specific, but reasonable
4. Simple object detector invented

Two-stage detectors

RCNN, Fast-RCNN, Faster-RCNN

Detection: Core idea

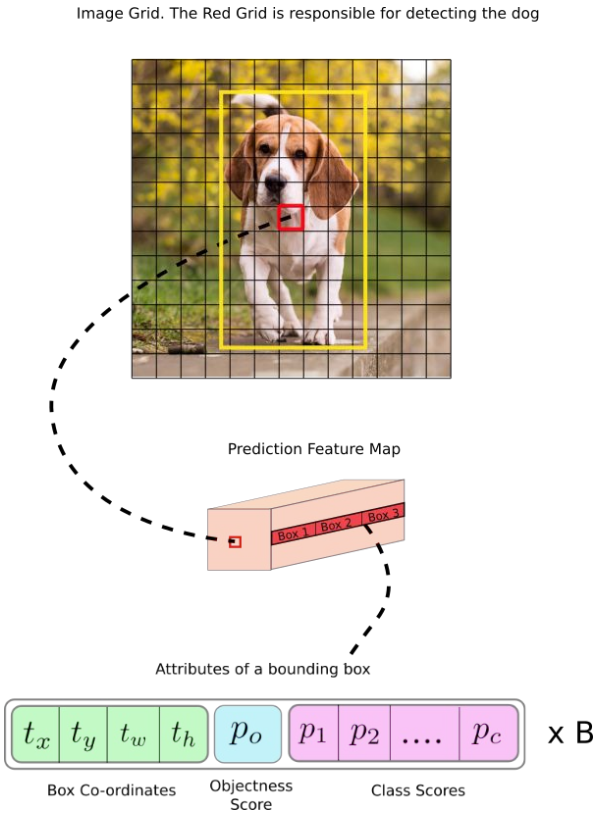
1. Make default boxes assumptions
2. Regress their corners coordinates
3. Predict class inside each box
4. Filter not confident predictions



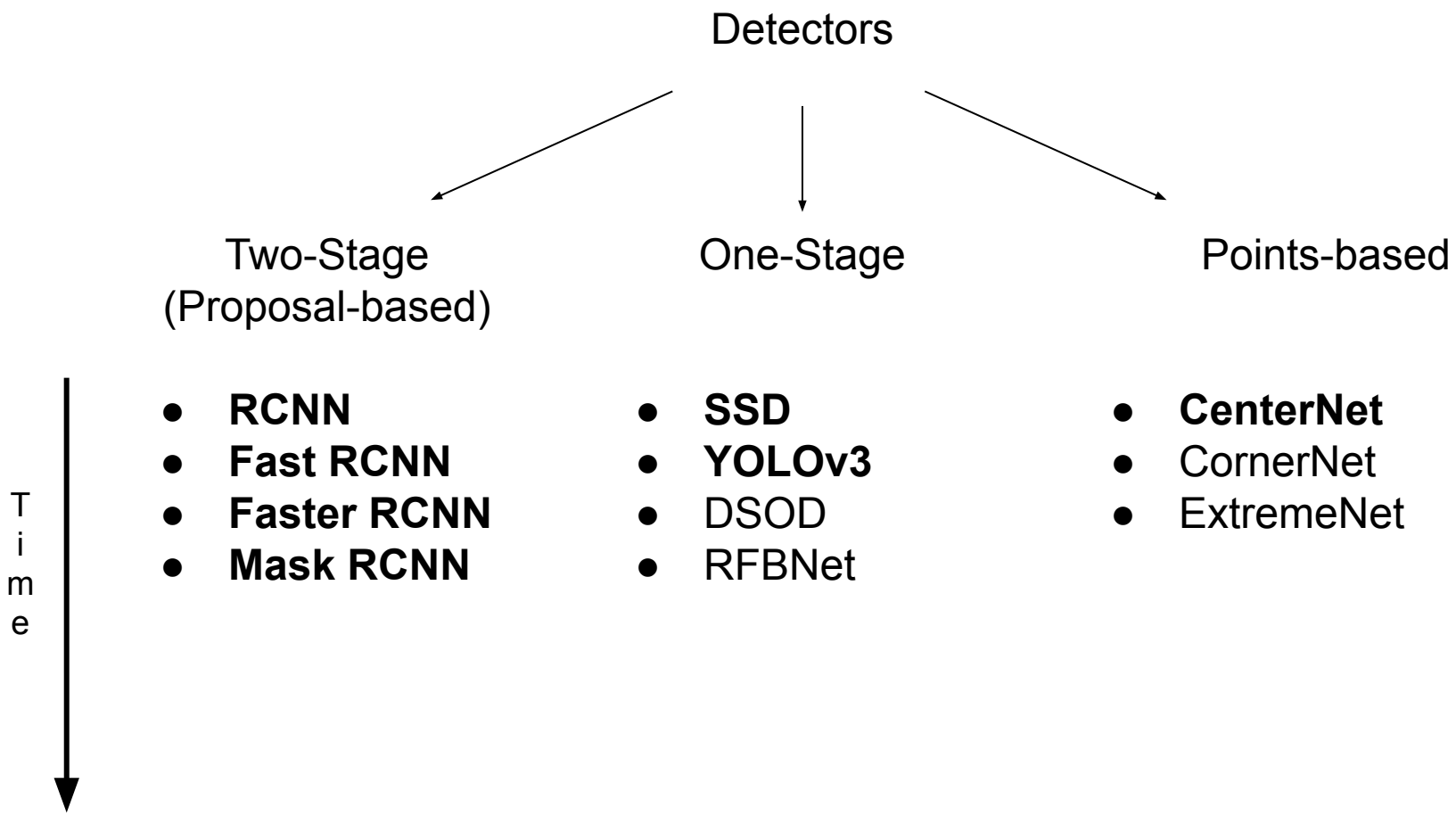
Detection: Core idea

1. Make default boxes assumptions
   “Anchor boxes”
2. **Regress** their corners coordinates
   Simultaneously
3. Predict **class** inside each box

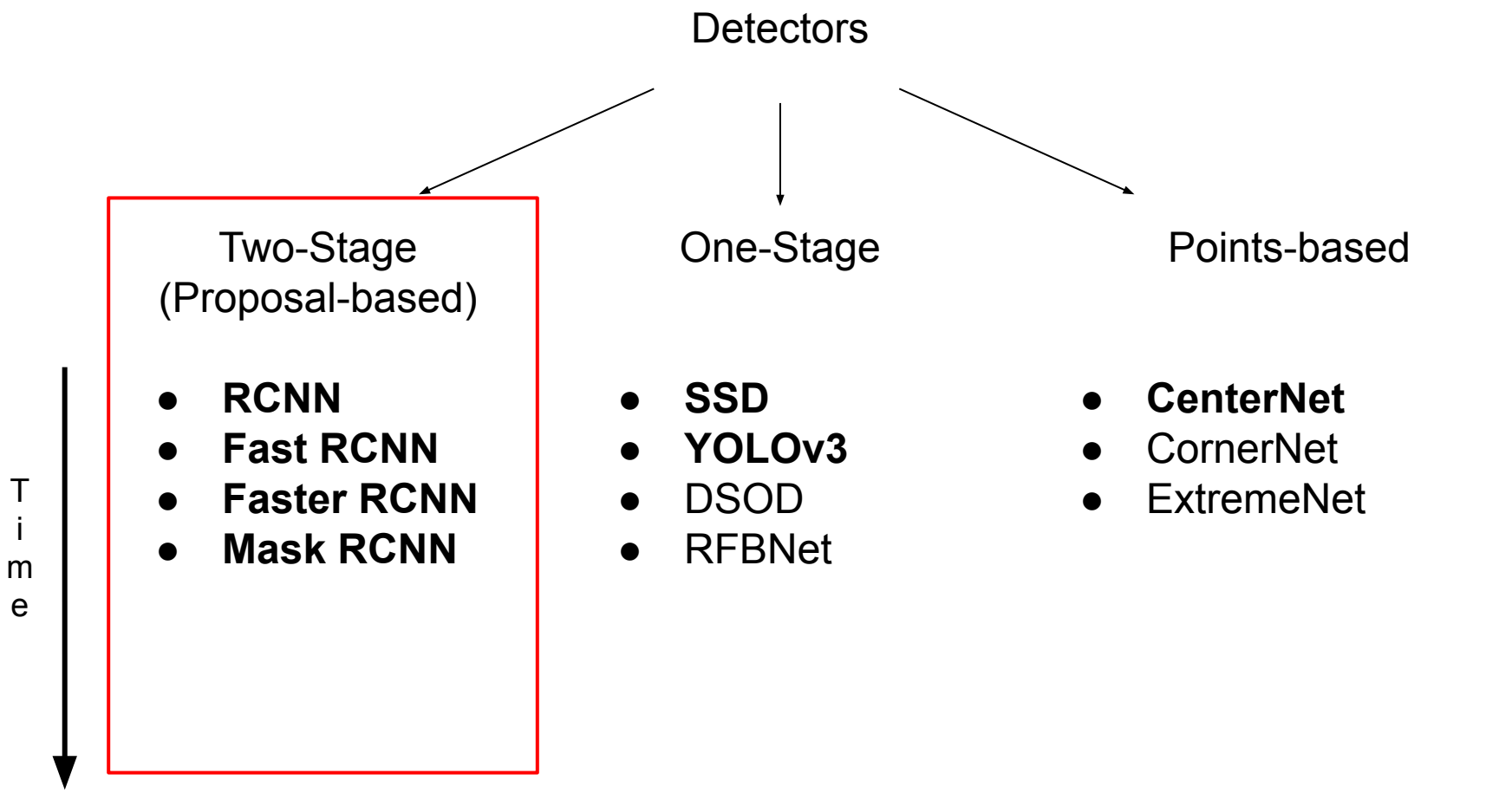
4. Filter not confident predictions
   Non-max suppression (NMS)



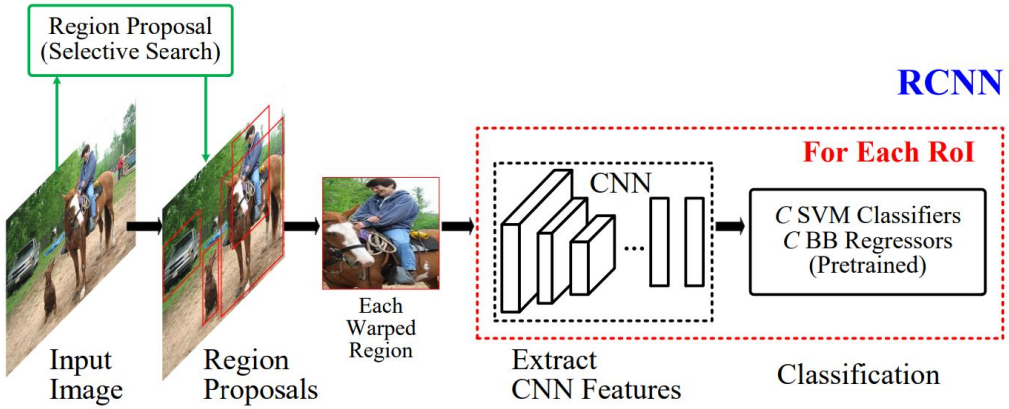
Detection: Taxonomy



Detection: Taxonomy



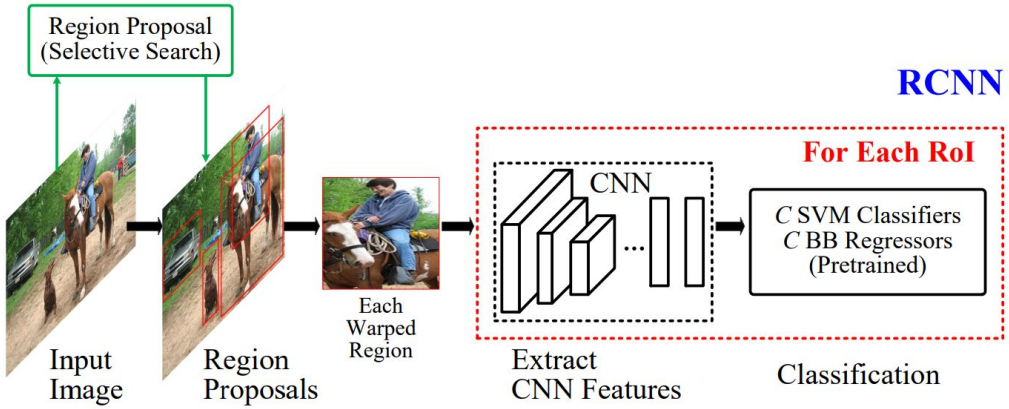
Detection: Two-stage (proposal-based)



https://arxiv.org/abs/1905.05055

Detection: Two-stage (proposal-based)

1. Generate boxes proposals
   (with separate algorithm / net)

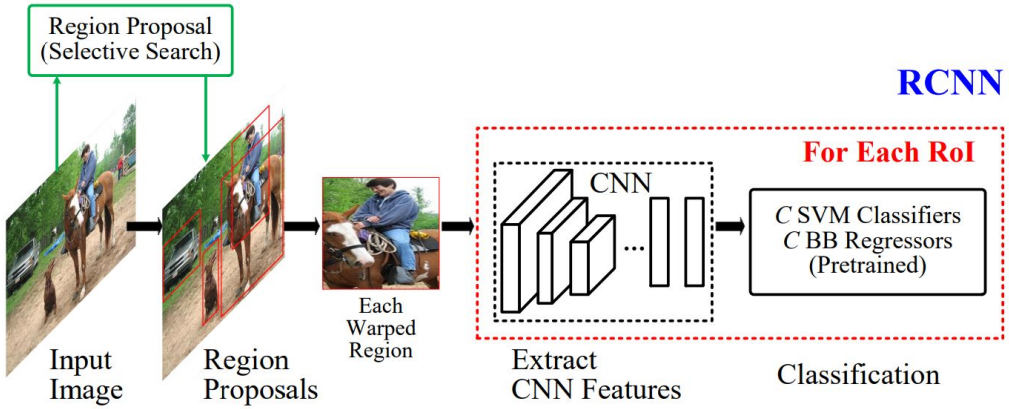


https://arxiv.org/abs/1905.05055

Detection: Two-stage (proposal-based)

1. Generate boxes proposals
   (with separate algorithm / net)

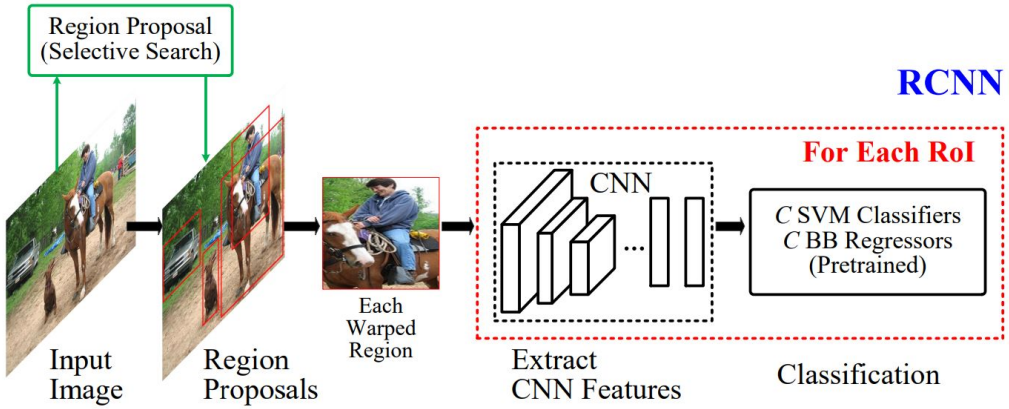
2. Warp them to one shape



https://arxiv.org/abs/1905.05055

Detection: Two-stage (proposal-based)

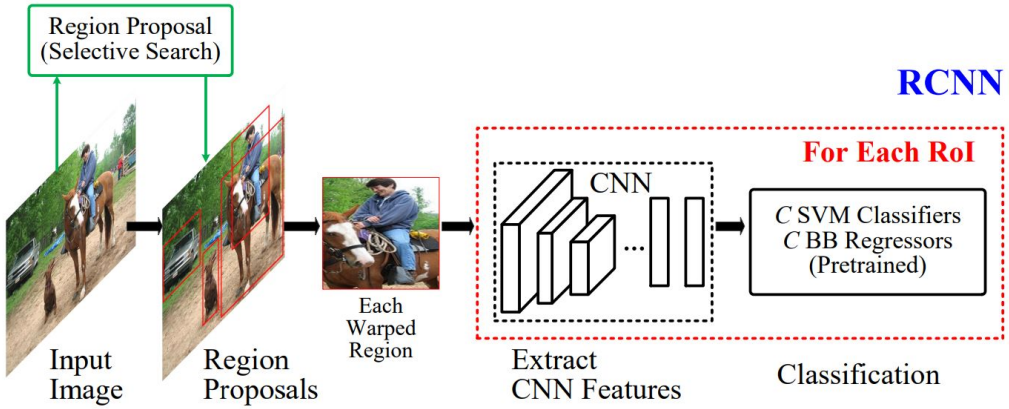
1. Generate boxes proposals (with separate algorithm / net)
2. Warp them to one shape
3. Regress and classify them (with another algorithm)



https://arxiv.org/abs/1905.05055

Detection: Two-stage (proposal-based)

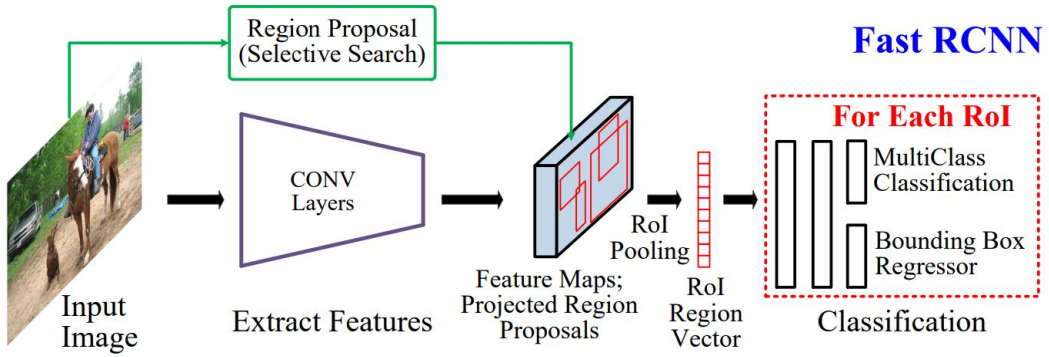
1. Generate boxes proposals (with separate algorithm / net)
2. Warp them to one shape
3. Regress and classify them (with another algorithm)
4. Post-process predictions



https://arxiv.org/abs/1905.05055

Detection: Two-stage (proposal-based)

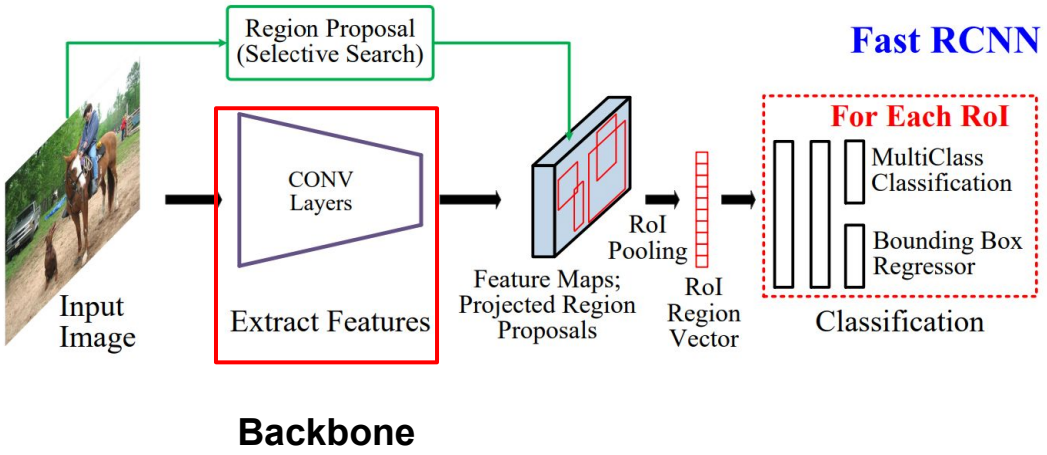
1. Generate boxes proposals
   (with separate algorithm / net)
2. Warp them to one shape
3. Regress and classify them
   (with another algorithm)
4. Post-process predictions



https://arxiv.org/abs/1905.05055

Detection: **Backbone**

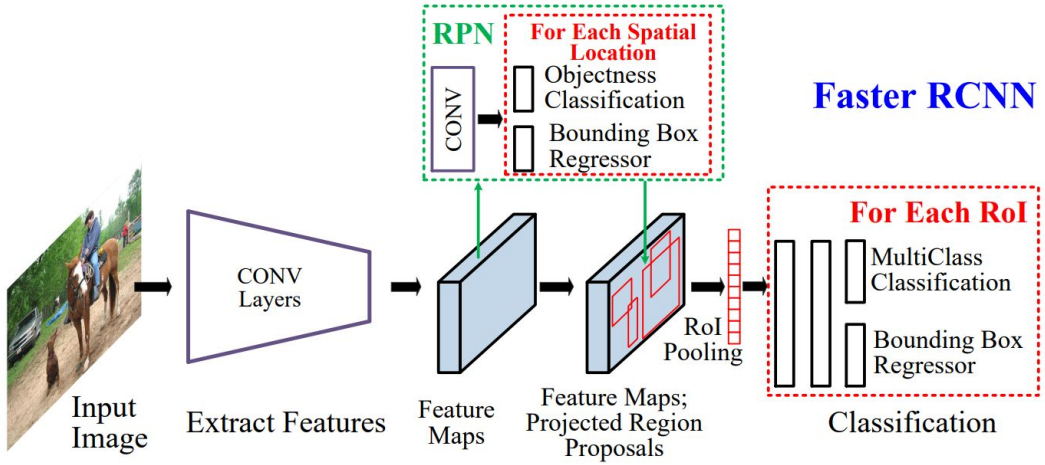
1. Generate boxes proposals (with separate algorithm / net)
2. Warp them to one shape
3. Regress and classify them (with another algorithm)
4. Post-process predictions



https://arxiv.org/abs/1905.05055

Detection: Two-stage (proposal-based)

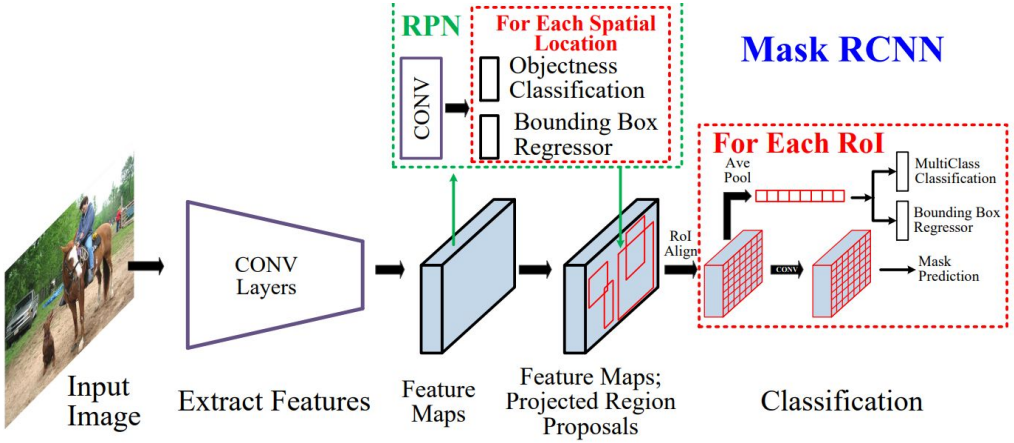
1. Generate boxes proposals (with separate algorithm / net)
2. Warp them to one shape
3. Regress and classify them (with another algorithm)
4. Post-process predictions



https://arxiv.org/abs/1905.05055

Detection: Two-stage (proposal-based)

1. Generate boxes proposals (with separate algorithm / net)
2. Warp them to one shape
3. Regress and classify them (with another algorithm)
4. Post-process predictions



https://arxiv.org/abs/1905.05055

Summary

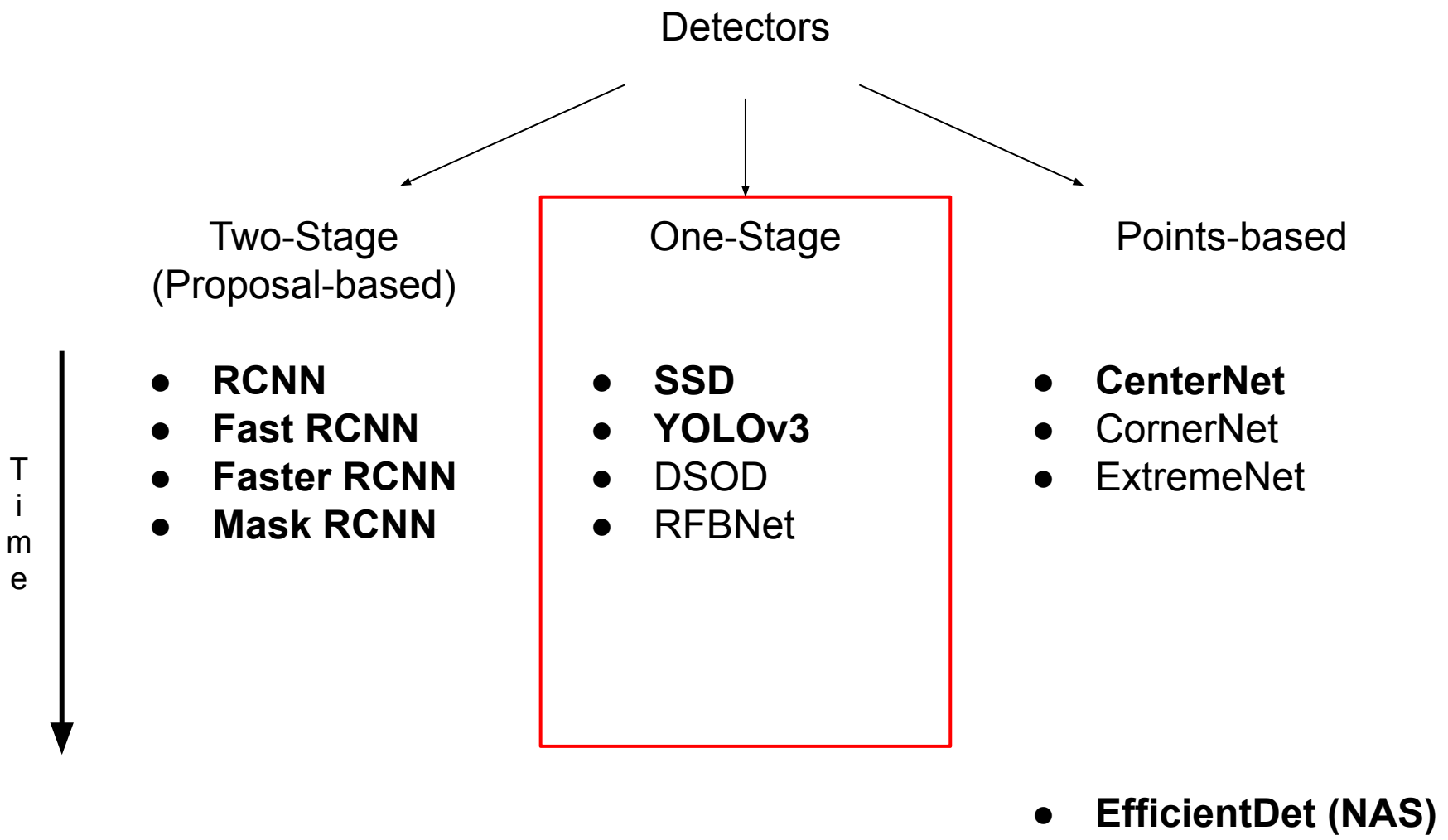
1. Core ideas

2. Two-stage detectors

One-stage detectors

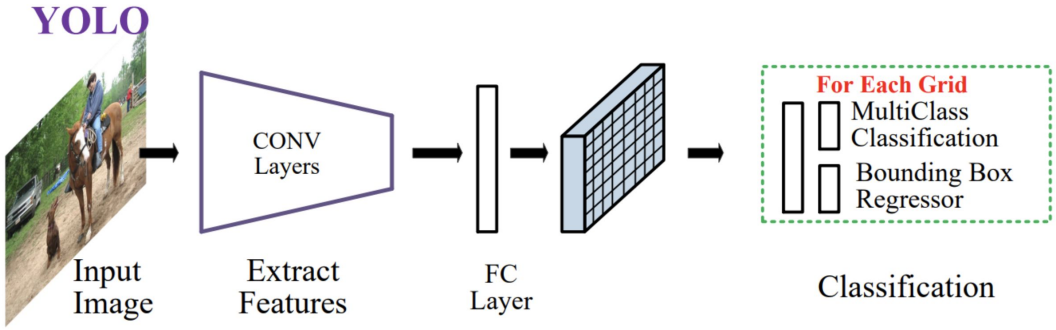
SSD, YOLO

Detection: Taxonomy



Detection: You Only Look Once

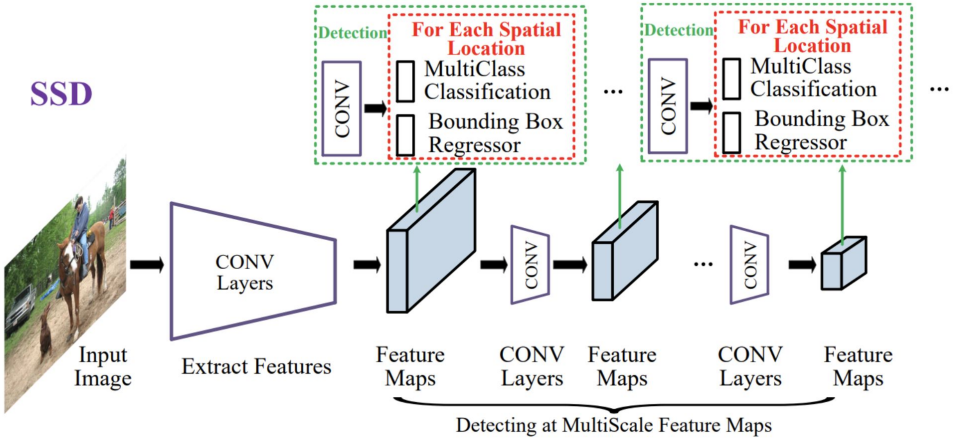
1. Get features from a **Backbone Network**
2. Predict feature map (“grid”)
3. Predict boxes from grid
4. Post-process predictions



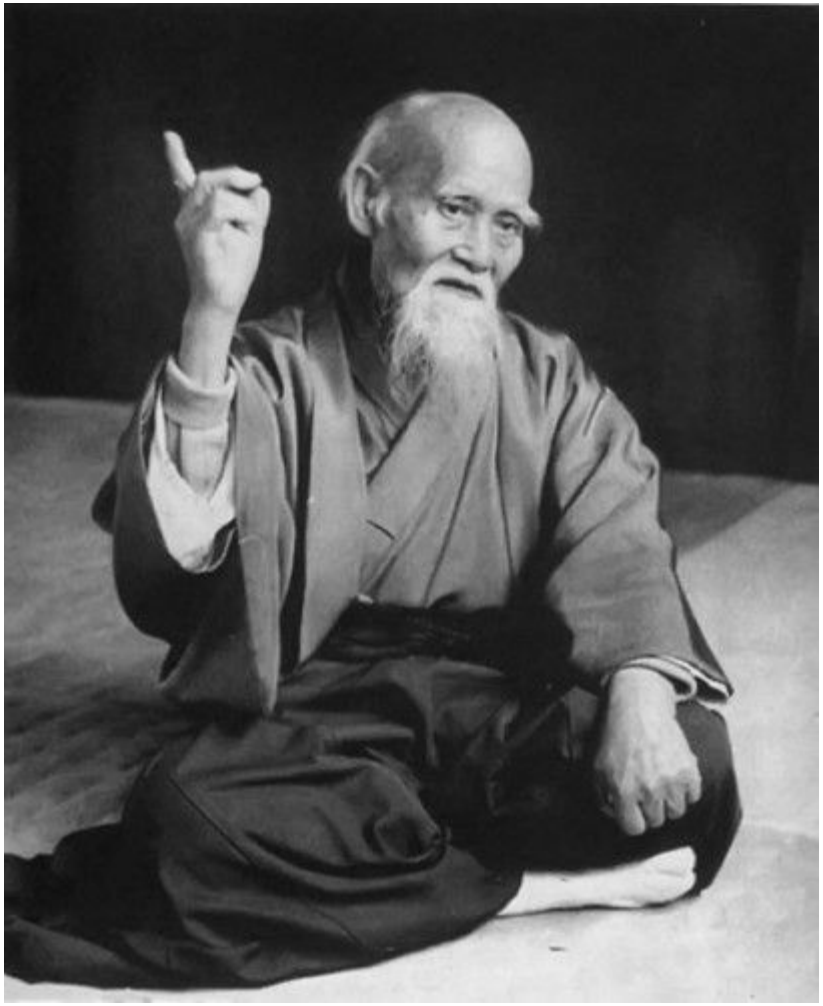
https://arxiv.org/abs/1905.05055

Detection: Single-Shot Multibox Detector (SSD)

1. Get features from a **Backbone Network**
2. Predict feature map (“grid”)
3. Predict boxes from grid
4. Post-process predictions



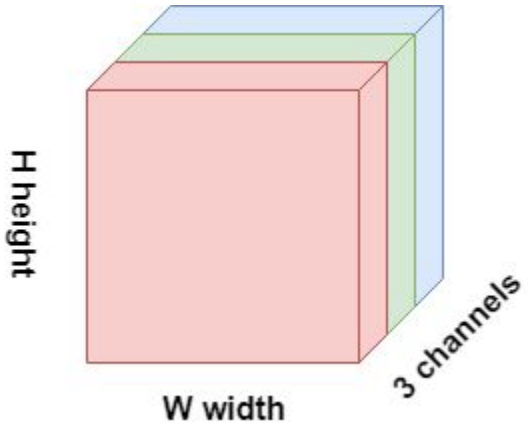
https://arxiv.org/abs/1905.05055



“Hone one thing,
but to the ideal”

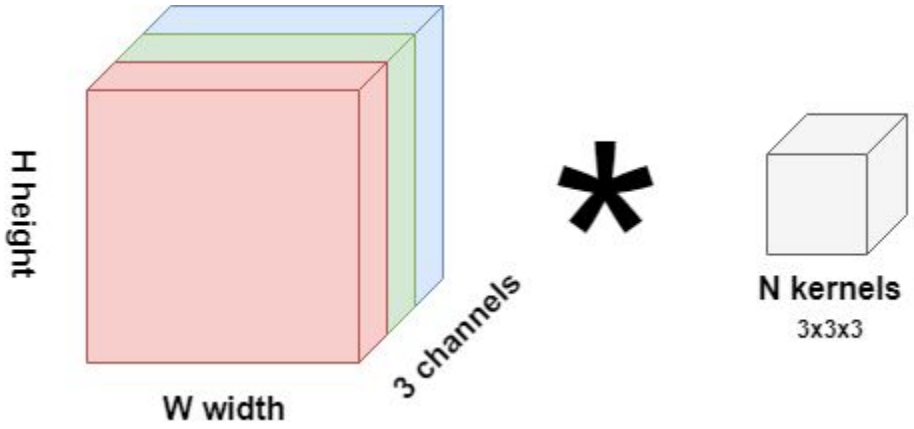
Some clever person

Convolutions: recap



image

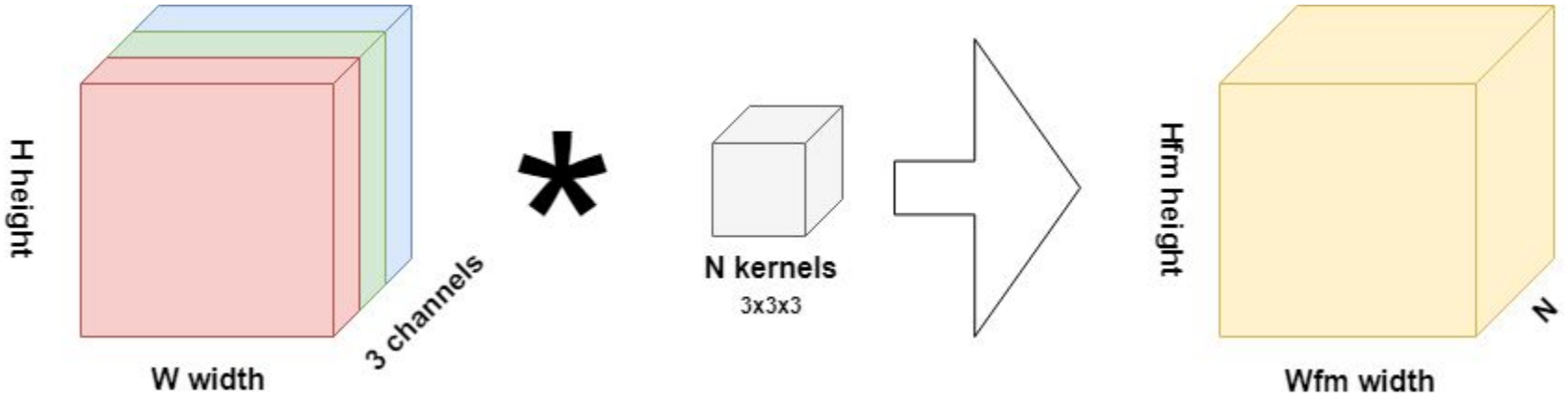
Convolutions: recap



image

conv kernels
(filters)

Convolutions: recap

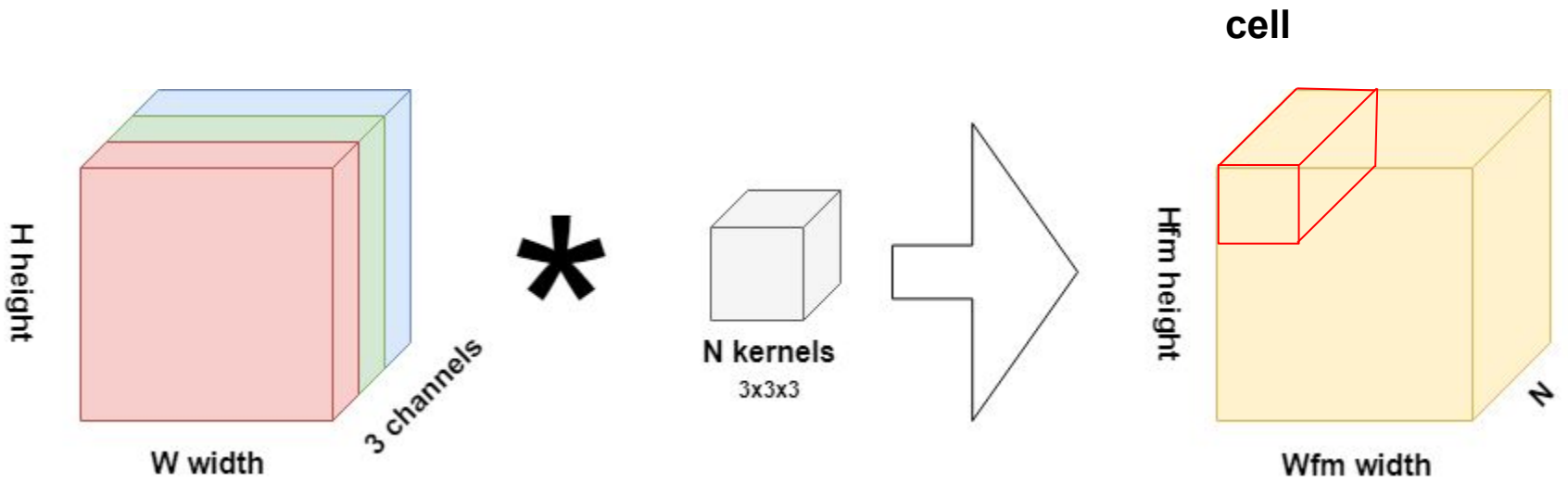


image

conv kernels
(filters)

feature map

Convolutions: recap



image

conv kernels
(filters)

feature map

SSD: up-to-bottom explanation

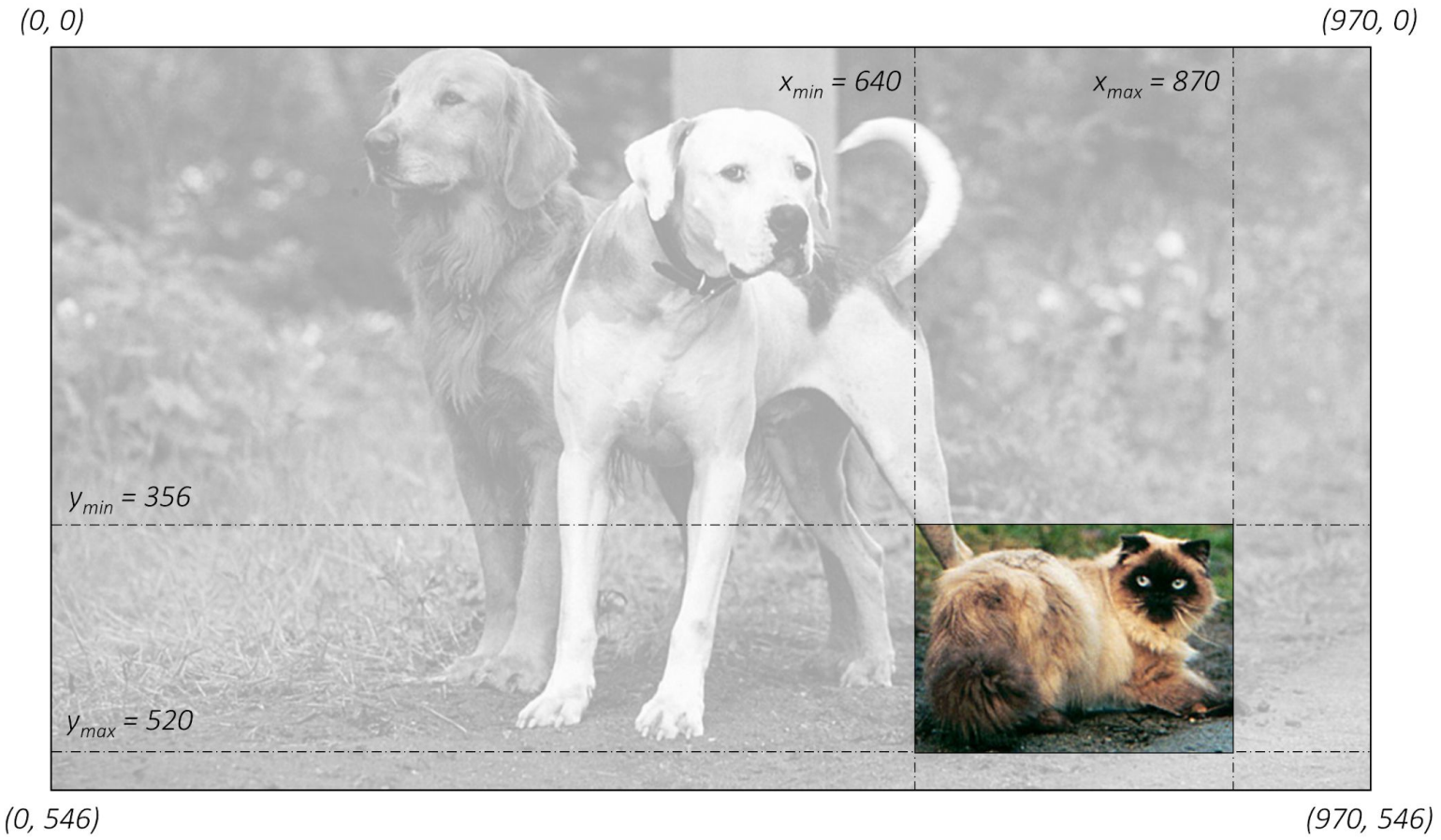
SSD pipeline

1. Input: image (h,w,3) → Backbone
2. Backbone → feature map
3. Feature map → boxes locations and classes inside of them

Detection task

1. How many boxes?
2. What should be the size of an input image?
3. How to train? (~ how to make it differentiable?)

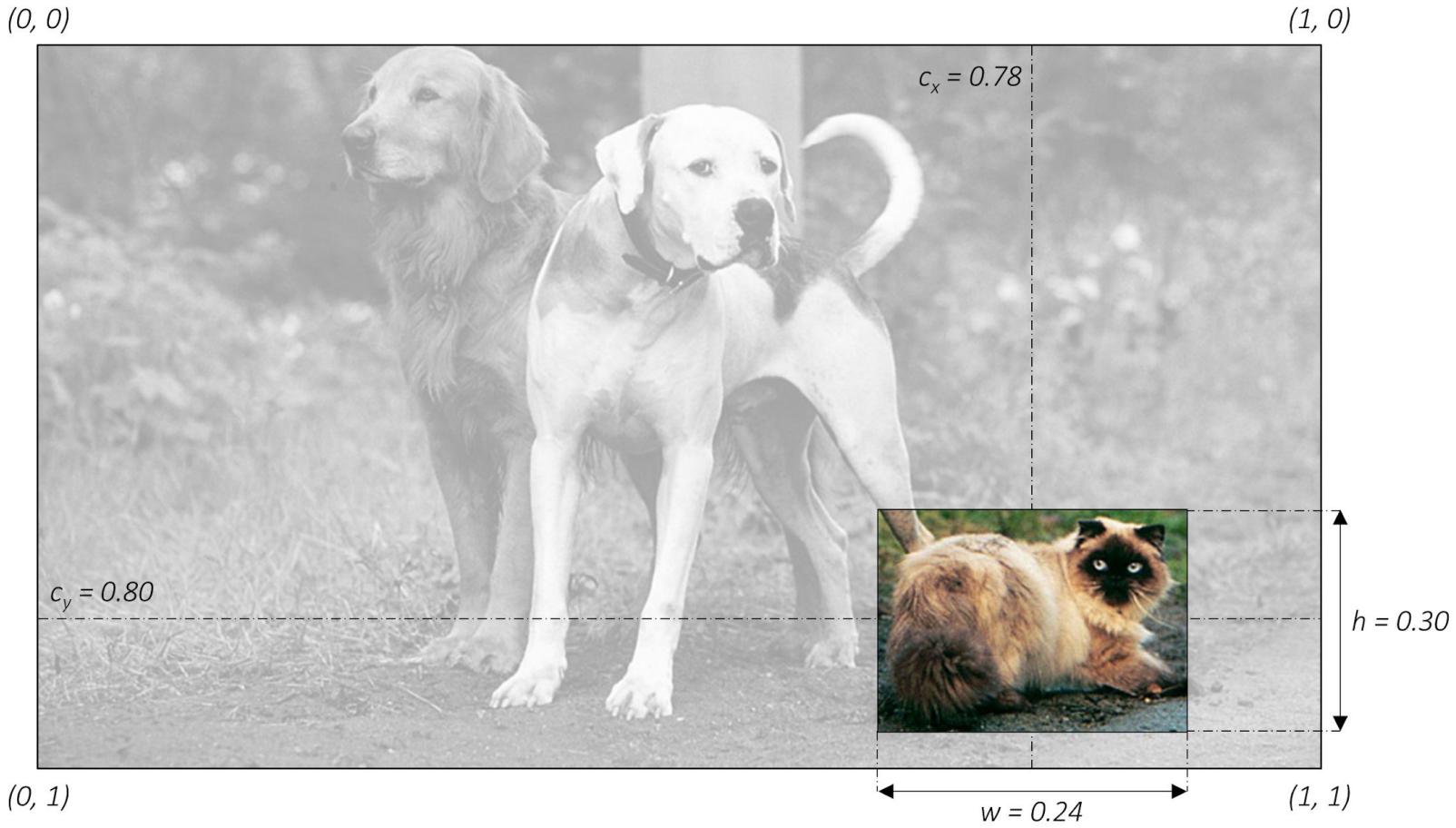
SSD: tlrb format



Boundary Coordinates (x_{min}, y_{min}, x_{max}, y_{max}) = (640, 356, 870, 520)

https://github.com/sgrvinod/a-PyTorch-Tutorial-to-Object-Detection

SSD: xywh format

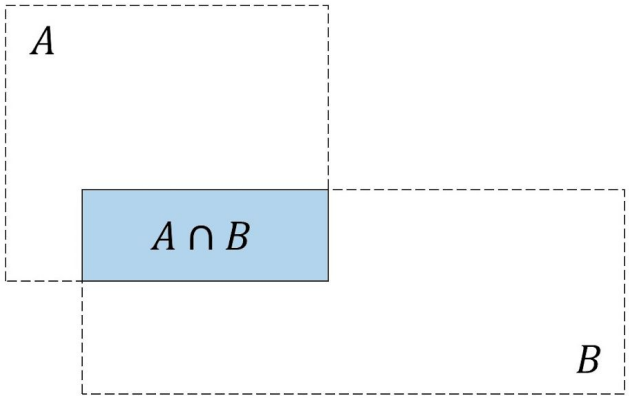


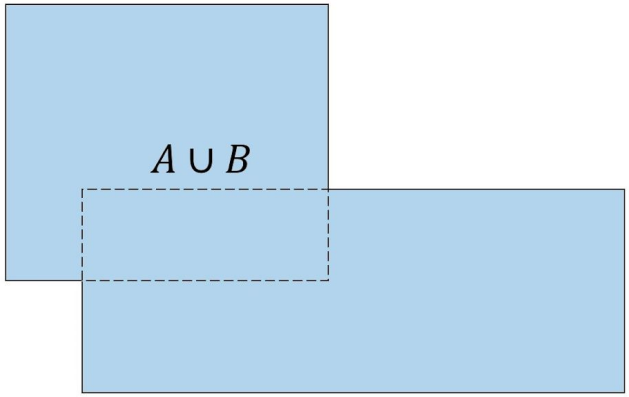
Center-Size Coordinates $(c_x, c_y, w, h) = (0.78, 0.8, 0.24, 0.30)$

https://github.com/sgrvinod/a-PyTorch-Tutorial-to-Object-Detection

SSD: Intersection over Union

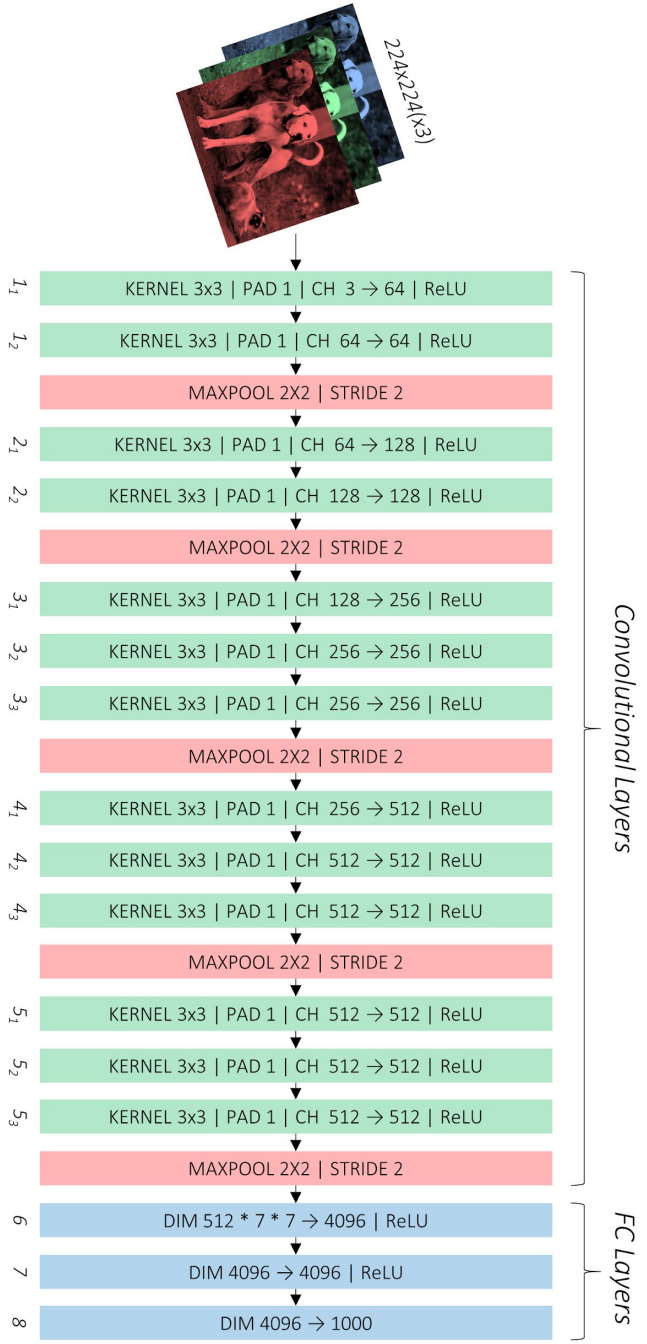
$$Jaccard\ Index = \frac{A \cap B}{A \cup B} =$$





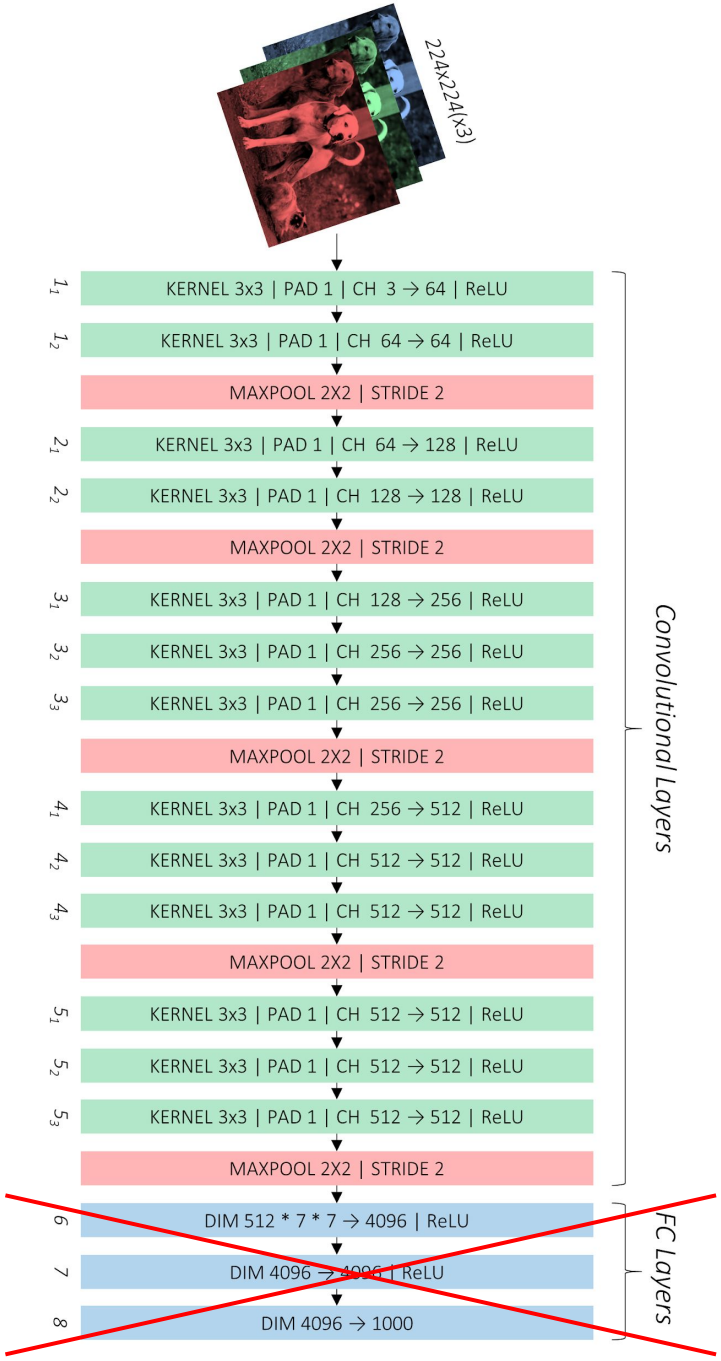
https://github.com/sgrvinod/a-PyTorch-Tutorial-to-Object-Detection

SSD: VGG



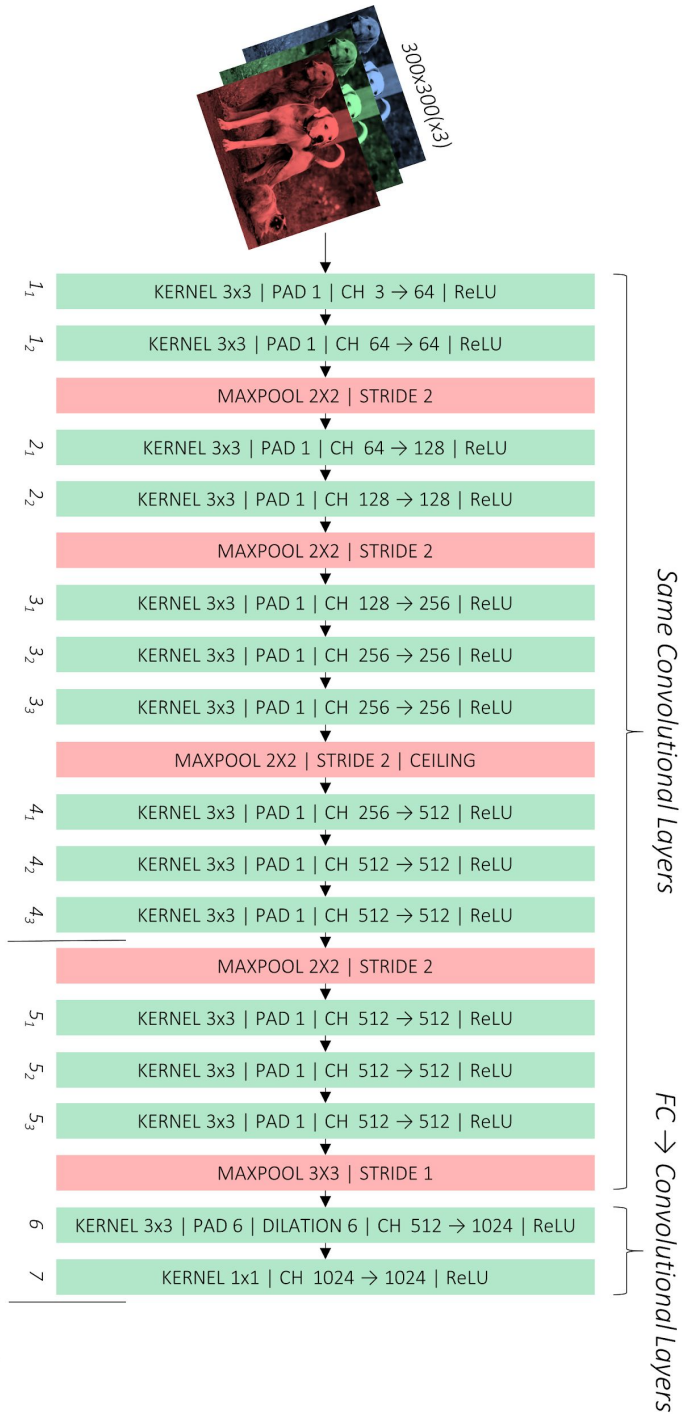
https://github.com/sgrvinod/a-PyTorch-Tutorial-to-Object-Detection

SSD: VGG



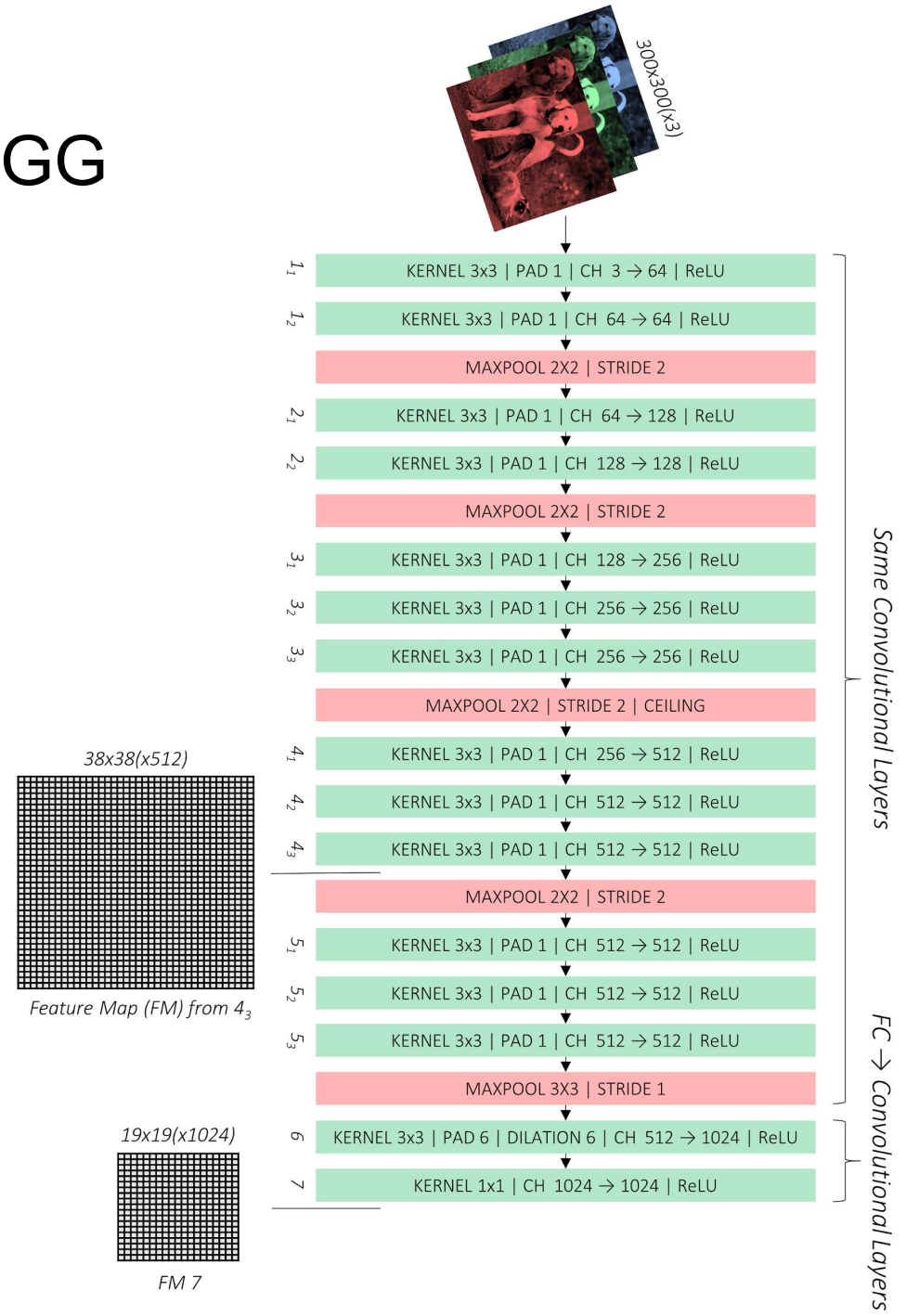
https://github.com/sgrvinod/a-PyTorch-Tutorial-to-Object-Detection

SSD



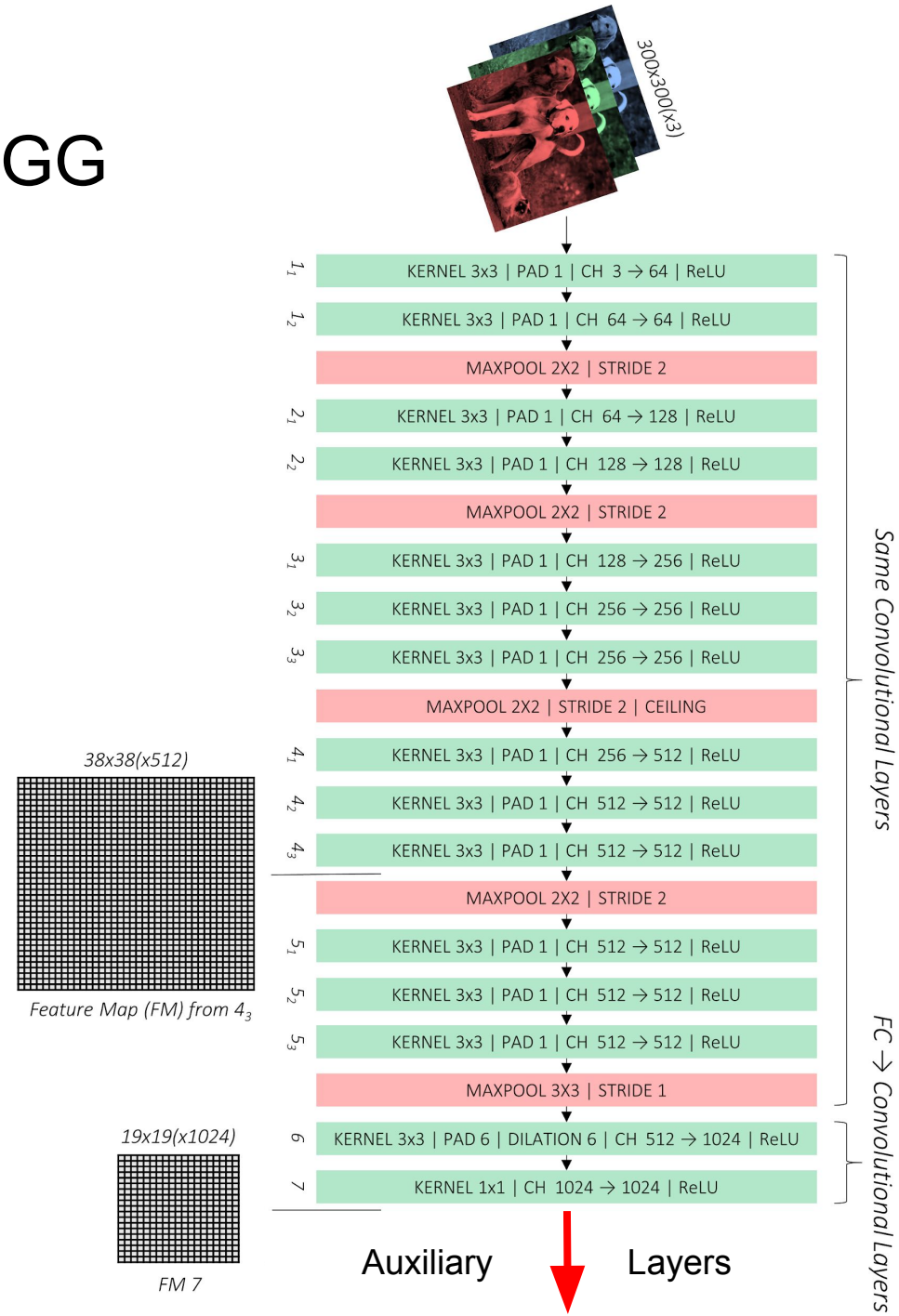
https://github.com/sgrvinod/a-PyTorch-Tutorial-to-Object-Detection

SSD: VGG



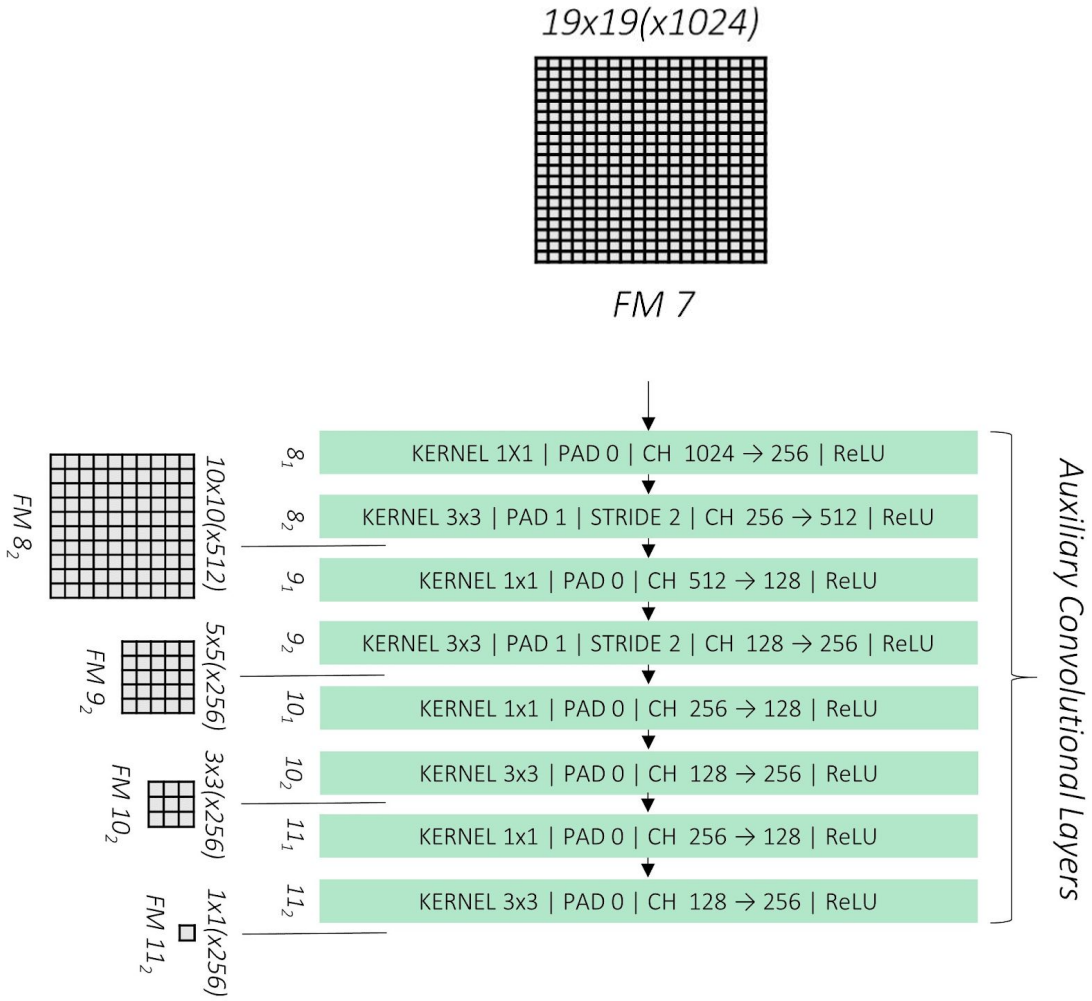
https://github.com/sgrvinod/a-PyTorch-Tutorial-to-Object-Detection

SSD: VGG



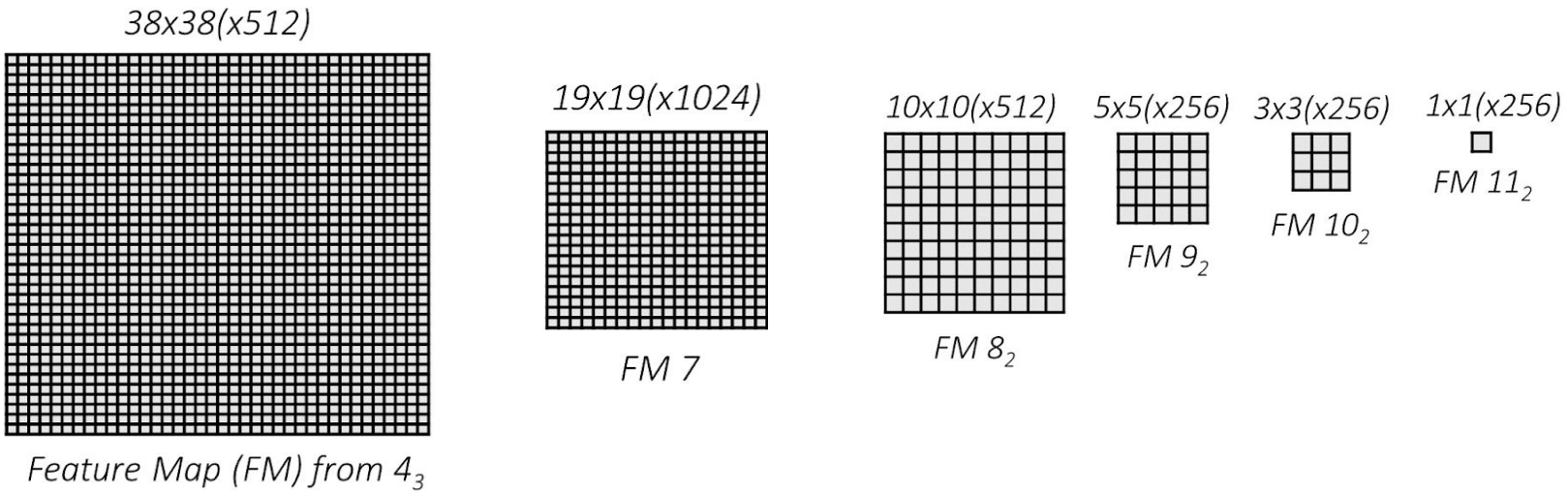
https://github.com/sgrvinod/a-PyTorch-Tutorial-to-Object-Detection

SSD: auxiliary (“extra”) layers



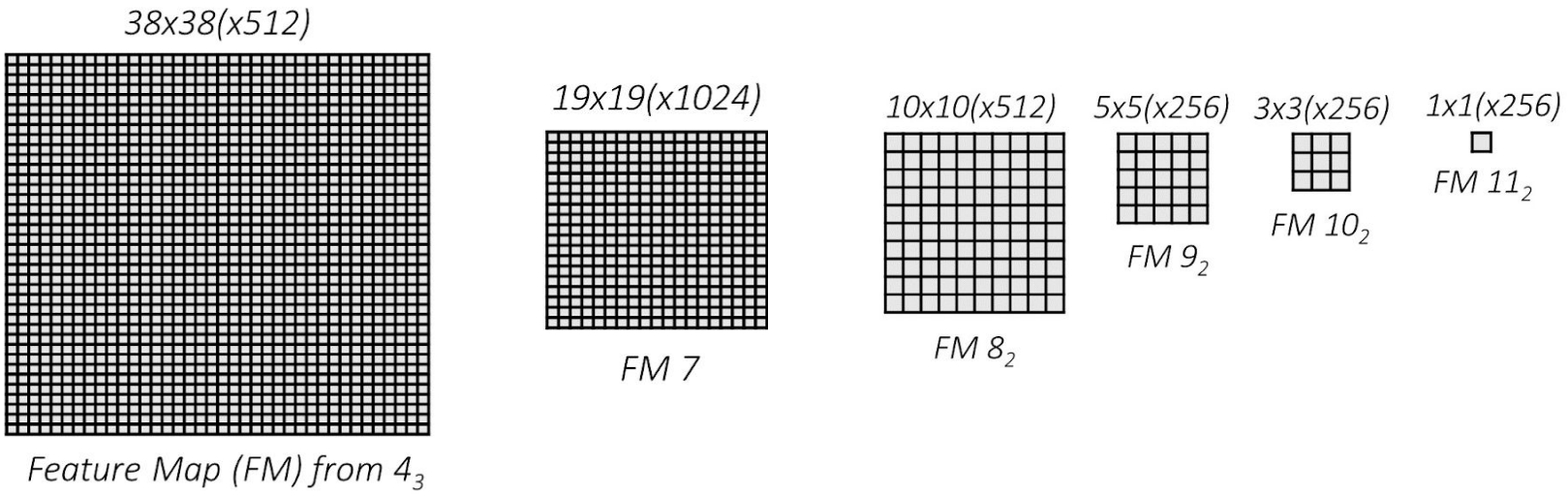
https://github.com/sgrvinod/a-PyTorch-Tutorial-to-Object-Detection

SSD: feature maps



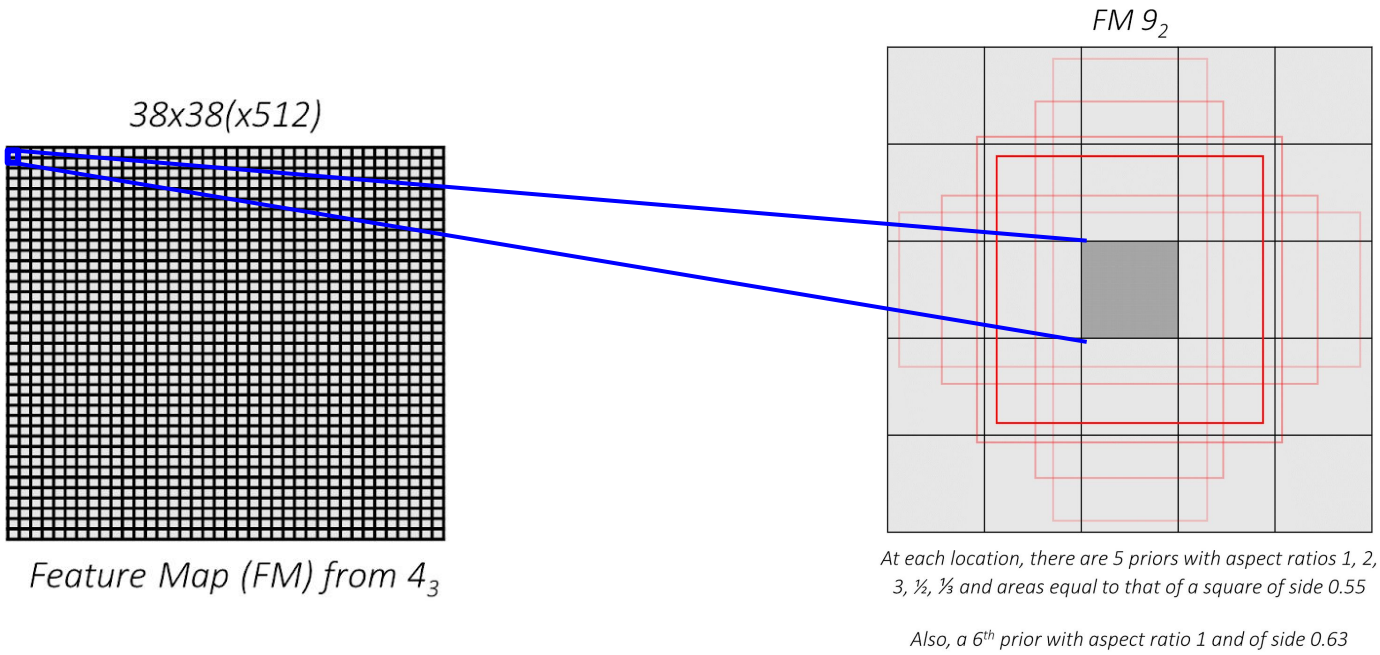
https://github.com/sgrvinod/a-PyTorch-Tutorial-to-Object-Detection

SSD



Each of these feature maps' cells has 4-6 "default boxes"

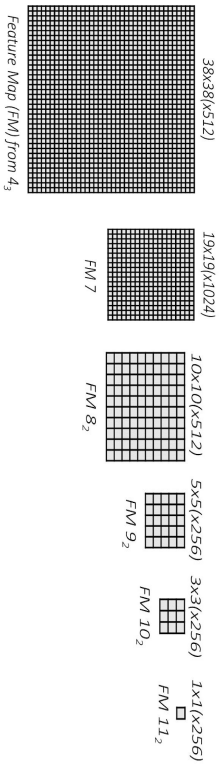
SSD: “default boxes” of one cell



Each of these feature maps’ cells has 4-6 “default boxes”

https://github.com/sgrvinod/a-PyTorch-Tutorial-to-Object-Detection

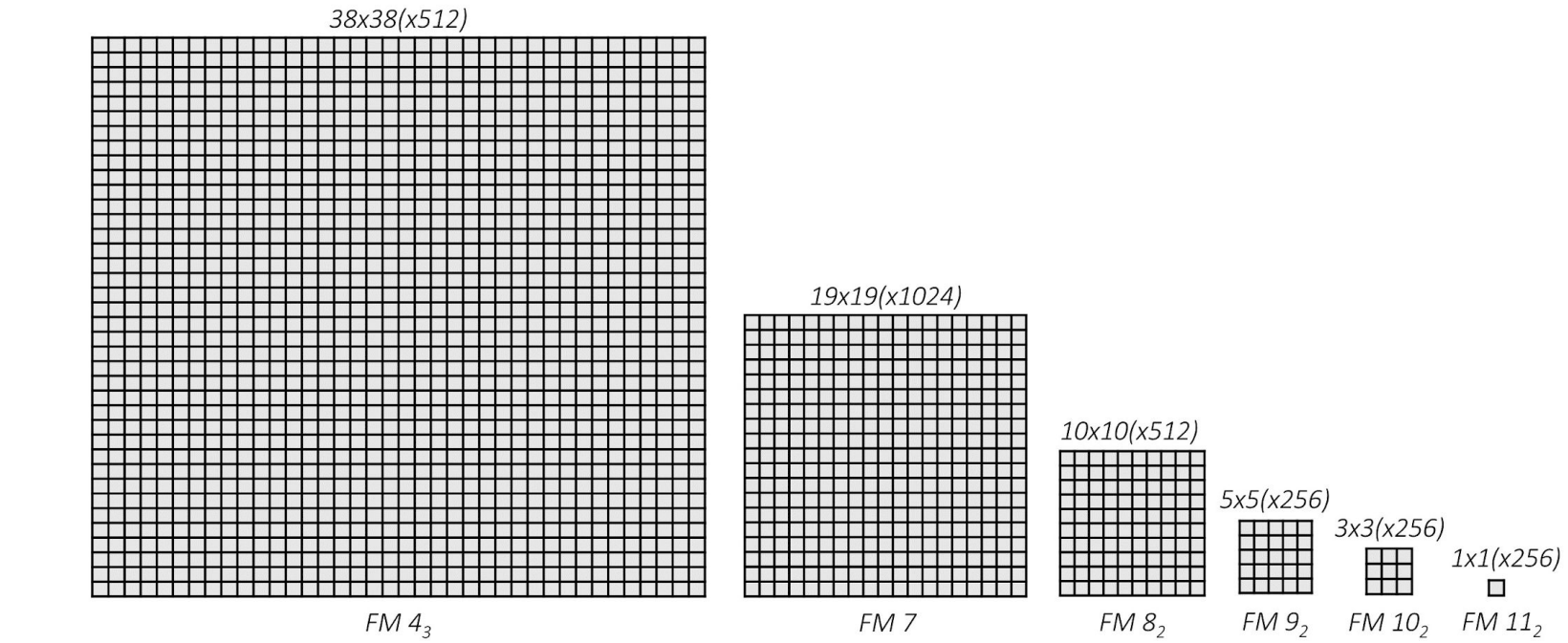
SSD: number of default boxes



Feature Map From	Feature Map Dimensions	Prior Scale	Aspect Ratios	Number of Priors per Position	Total Number of Priors on this Feature Map
`conv4_3`	38, 38	0.1	1:1, 2:1, 1:2 + an extra prior	4	5776
`conv7`	19, 19	0.2	1:1, 2:1, 1:2, 3:1, 1:3 + an extra prior	6	2166
`conv8_2`	10, 10	0.375	1:1, 2:1, 1:2, 3:1, 1:3 + an extra prior	6	600
`conv9_2`	5, 5	0.55	1:1, 2:1, 1:2, 3:1, 1:3 + an extra prior	6	150
`conv10_2`	3, 3	0.725	1:1, 2:1, 1:2 + an extra prior	4	36
`conv11_2`	1, 1	0.9	1:1, 2:1, 1:2 + an extra prior	4	4
Grand Total	–	–	–	–	**8732 priors**

https://github.com/sgrvinod/a-PyTorch-Tutorial-to-Object-Detection

SSD: prediction convolutions



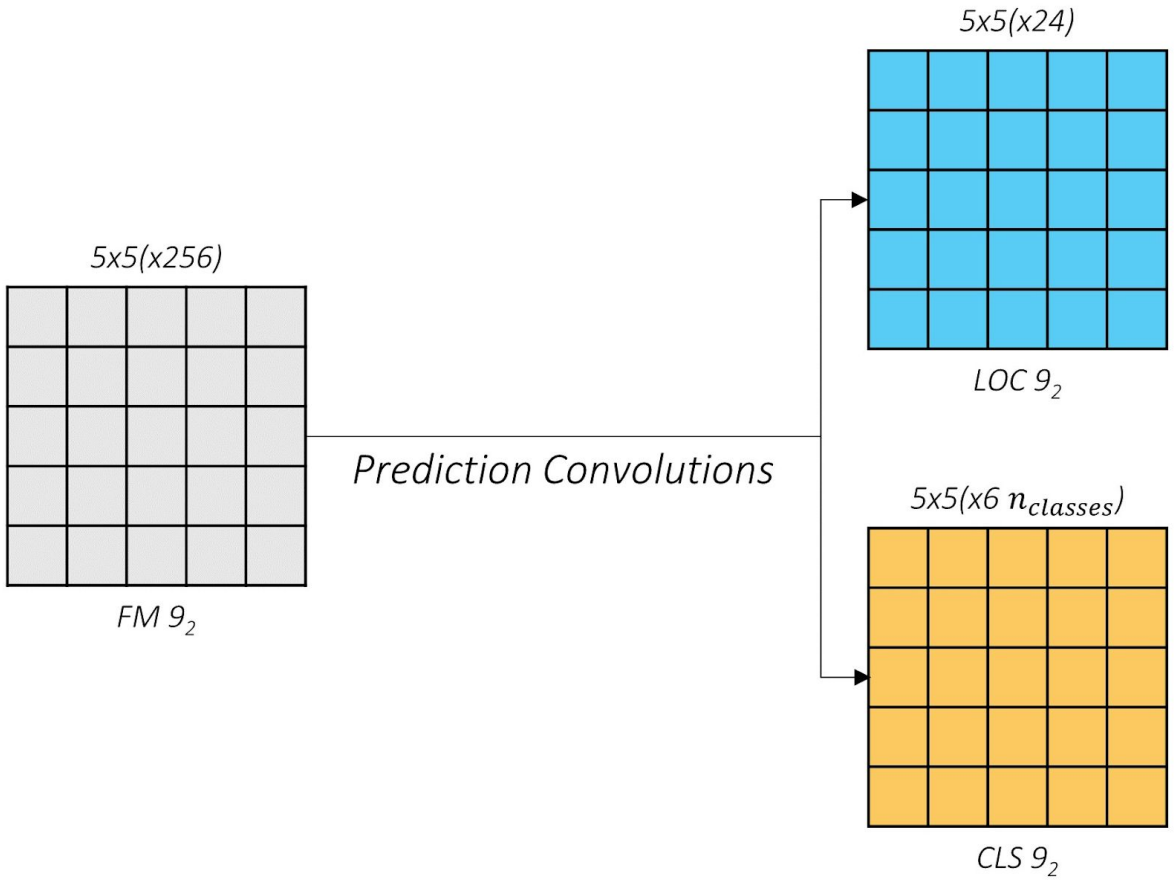
A convolutional layer for each feature map predicting localization boxes in the form of offsets (g_{c_x}, g_{c_y}, g_w, g_h) from each prior at each location in the feature map.

A convolutional layer for each feature map predicting class scores for each prior at each location in the feature map.

This indicates the type of object in the corresponding localization box. If the class predicted is 'background', there is no object.

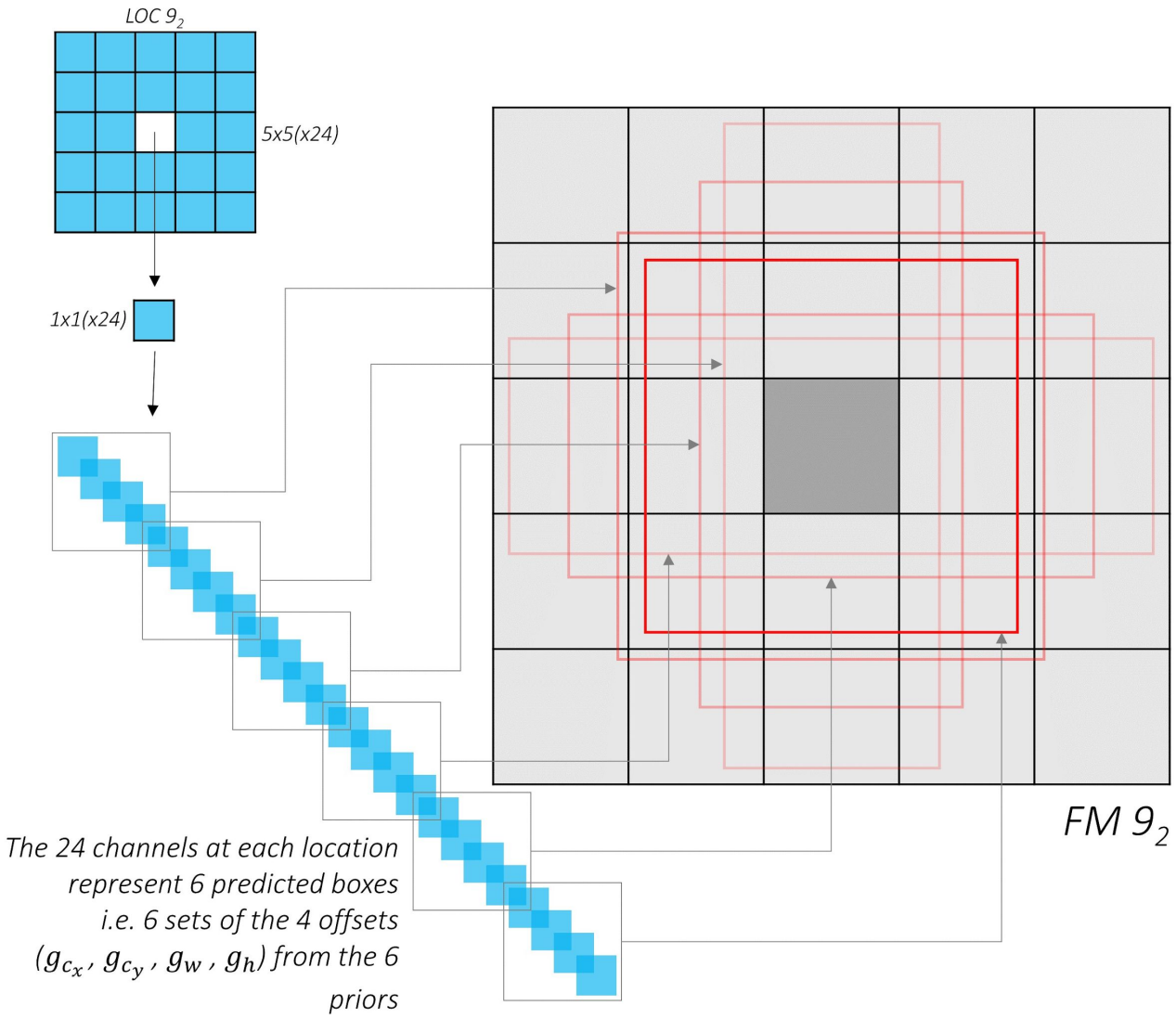
https://github.com/sgrvinod/a-PyTorch-Tutorial-to-Object-Detection

SSD: prediction scheme



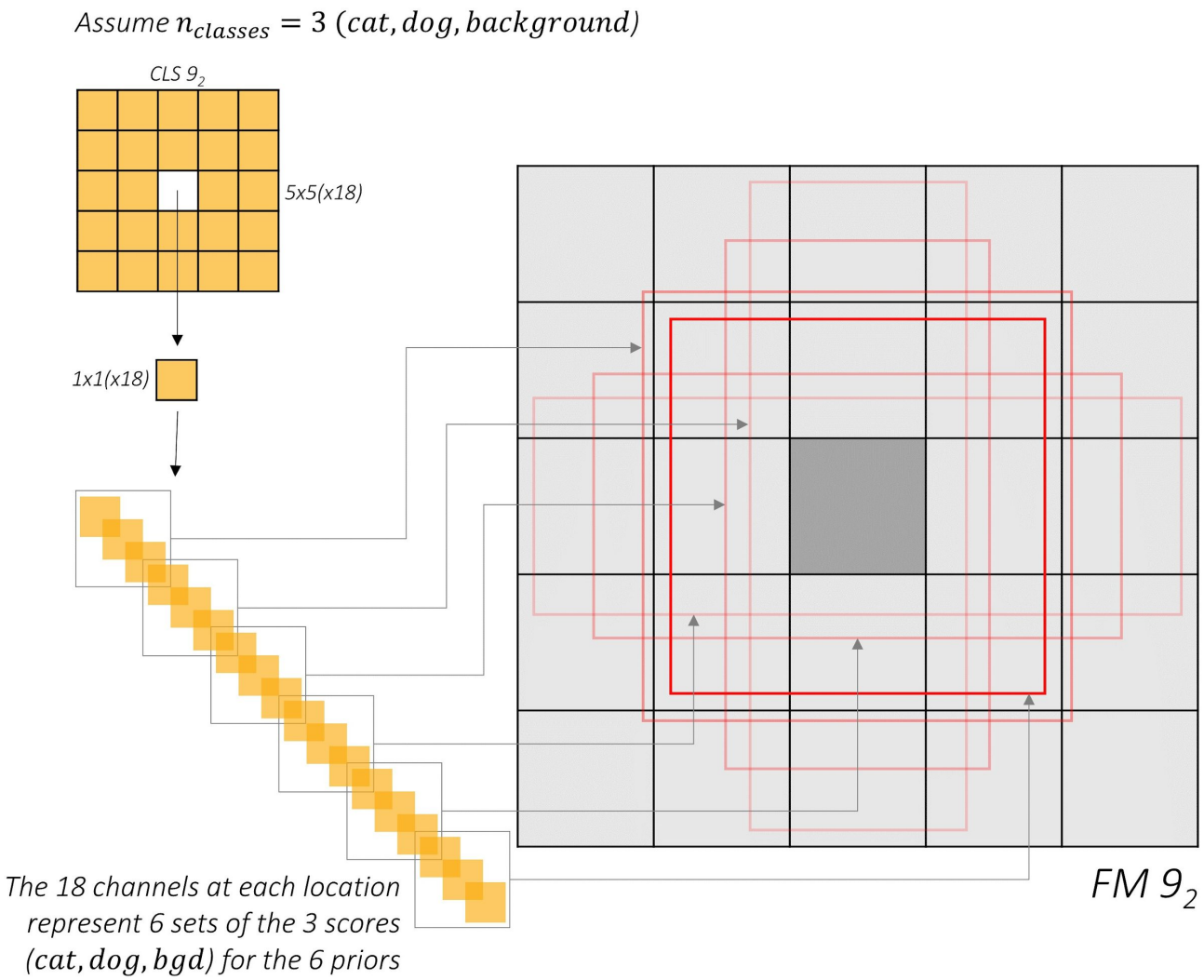
https://github.com/sgrvinod/a-PyTorch-Tutorial-to-Object-Detection

SSD: location coordinates channels



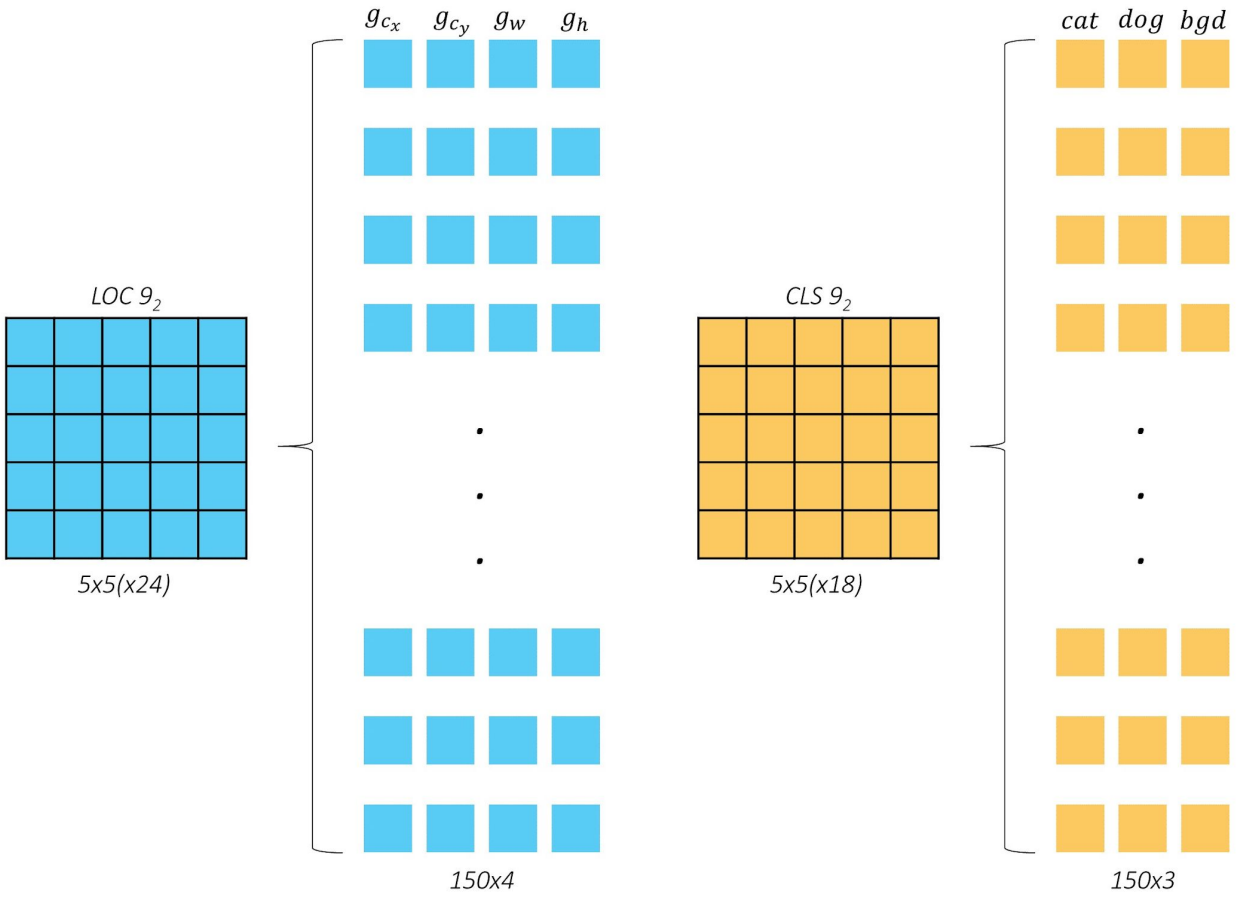
https://github.com/sgrvinod/a-PyTorch-Tutorial-to-Object-Detection

SSD: classification score channels



https://github.com/sgrvinod/a-PyTorch-Tutorial-to-Object-Detection

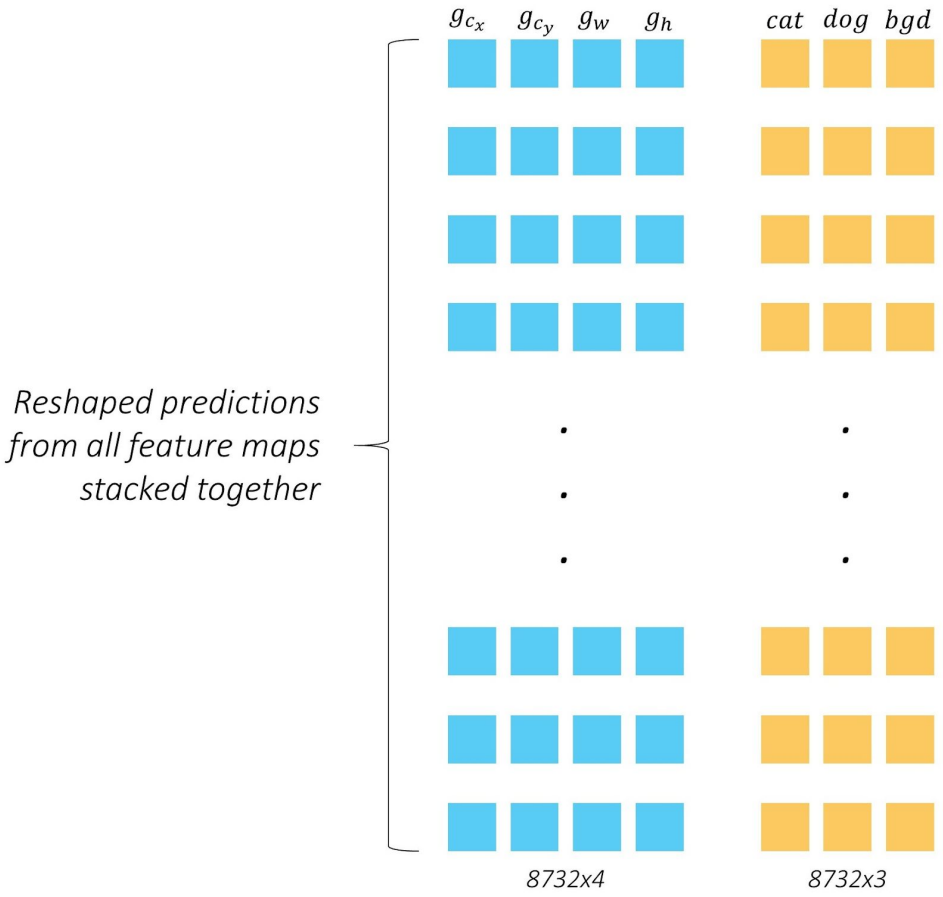
SSD: boxes of FM 9



Reshape predictions from FM 9_2 to represent offsets and class scores for the 150 predicted boxes

https://github.com/sgrvinod/a-PyTorch-Tutorial-to-Object-Detection

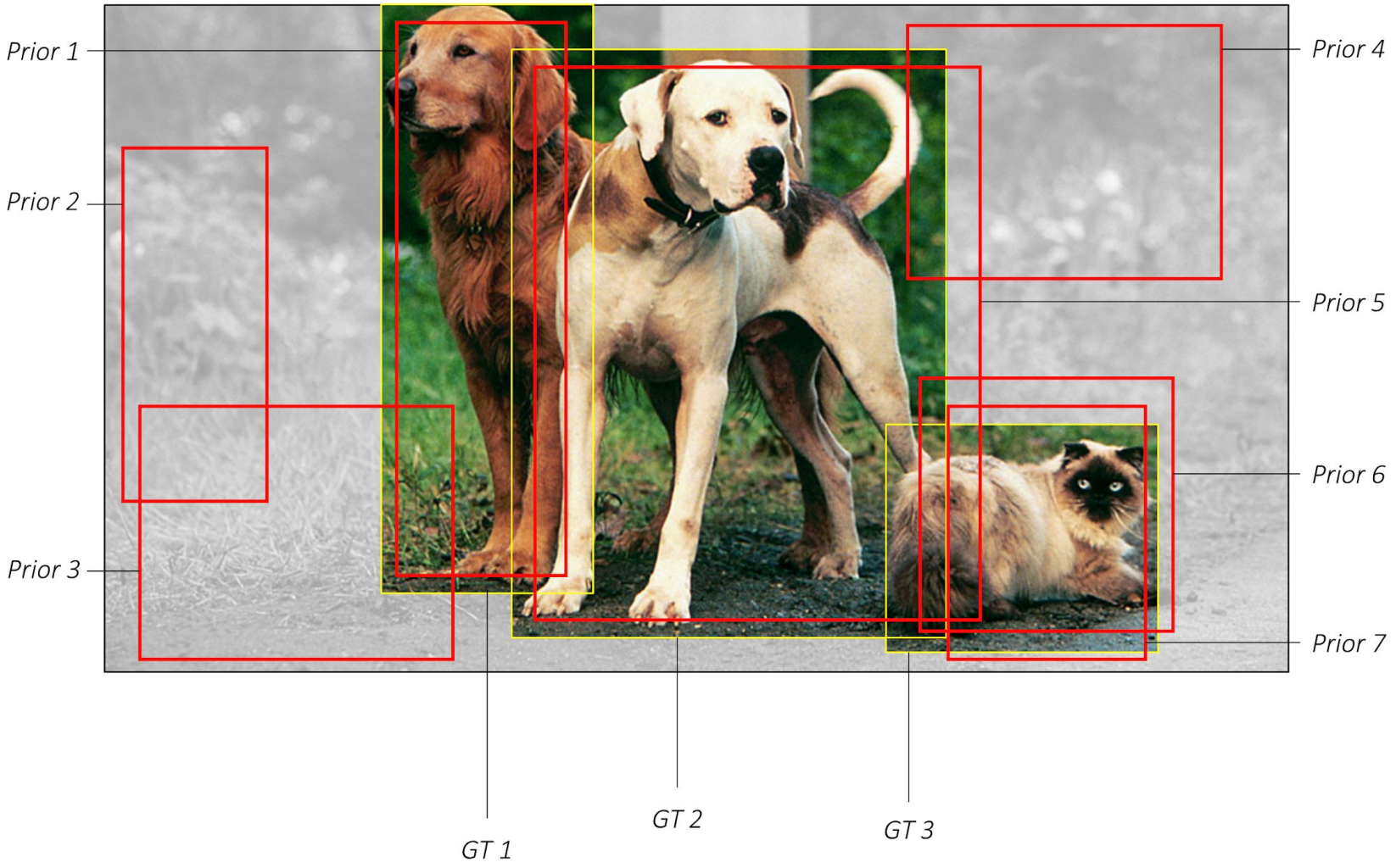
SSD: all predicted boxes



A total of 8732 predicted boxes

https://github.com/sgrvinod/a-PyTorch-Tutorial-to-Object-Detection

SSD: predictions



https://github.com/sgrvinod/a-PyTorch-Tutorial-to-Object-Detection

SSD: matching

IoU	*GT 1*	*GT 2*	*GT 3*
Prior 1	0.75	0.07	0
Prior 2	0	0	0
Prior 3	0.05	0	0
Prior 4	0	0.03	0
Prior 5	0.05	0.88	0.06
Prior 6	0	0.03	0.65
Prior 7	0	0	0.68

Match each prior to the ground truth that has the best Jaccard overlap.

	Matched GT	*Type*	*Label*	*Coordinates*
Prior 1	GT 1	Positive	*dog*	of GT 1
Prior 2	GT 1	Negative	*bgd*	-
Prior 3	GT 1	Negative	*bgd*	-
Prior 4	GT 2	Negative	*bgd*	-
Prior 5	GT 2	Positive	*dog*	of GT 2
Prior 6	GT 3	Positive	*cat*	of GT 3
Prior 7	GT 3	Positive	*cat*	of GT 3

If a prior has a Jaccard overlap greater than 0.5 with its matched ground truth object, then it really contains this object, and is a positive match.

Otherwise, it is a negative match, and is assigned a 'background' label.

SSD: coordinate regression loss

$$L_{loc} = \frac{1}{n_{positives}} \left(\sum_{positives} Smooth\ L_1\ Loss \right)$$

https://github.com/sgrvinod/a-PyTorch-Tutorial-to-Object-Detection

SSD: coordinate regression loss

$$L_{loc} = \frac{1}{n_{positives}} \left(\sum_{positives} Smooth\ L_1\ Loss \right)$$

$$L_{\text{loc}}(t\ \ , v) = \sum_{i \in \{\text{x,y,w,h}\}} \text{smooth}_{L_1}(t_i\ - v_i),$$

in which

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

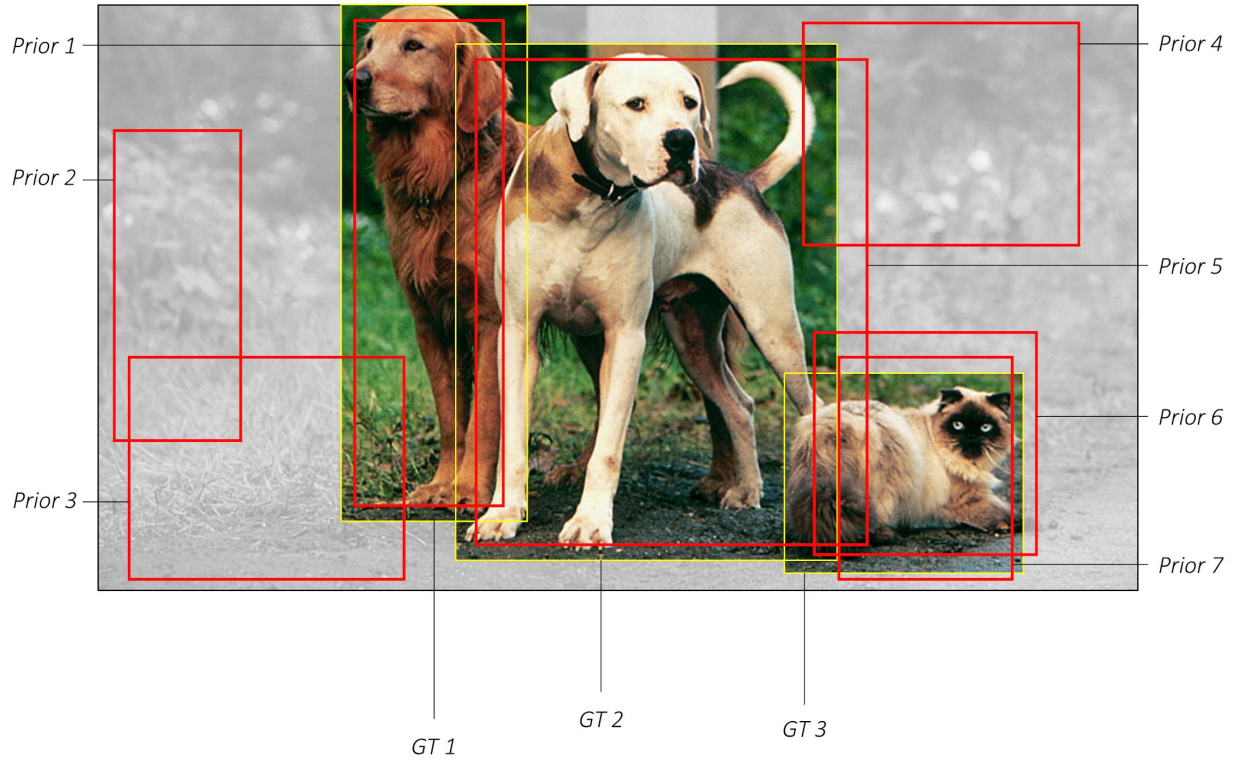
https://github.com/sgrvinod/a-PyTorch-Tutorial-to-Object-Detection

SSD: classification loss + hard negative mining

$$L_{conf} = \frac{1}{n_{positives}} \left(\sum_{positives} CE\ Loss + \sum_{hard\ negatives} CE\ Loss \right)$$

https://github.com/sgrvinod/a-PyTorch-Tutorial-to-Object-Detection

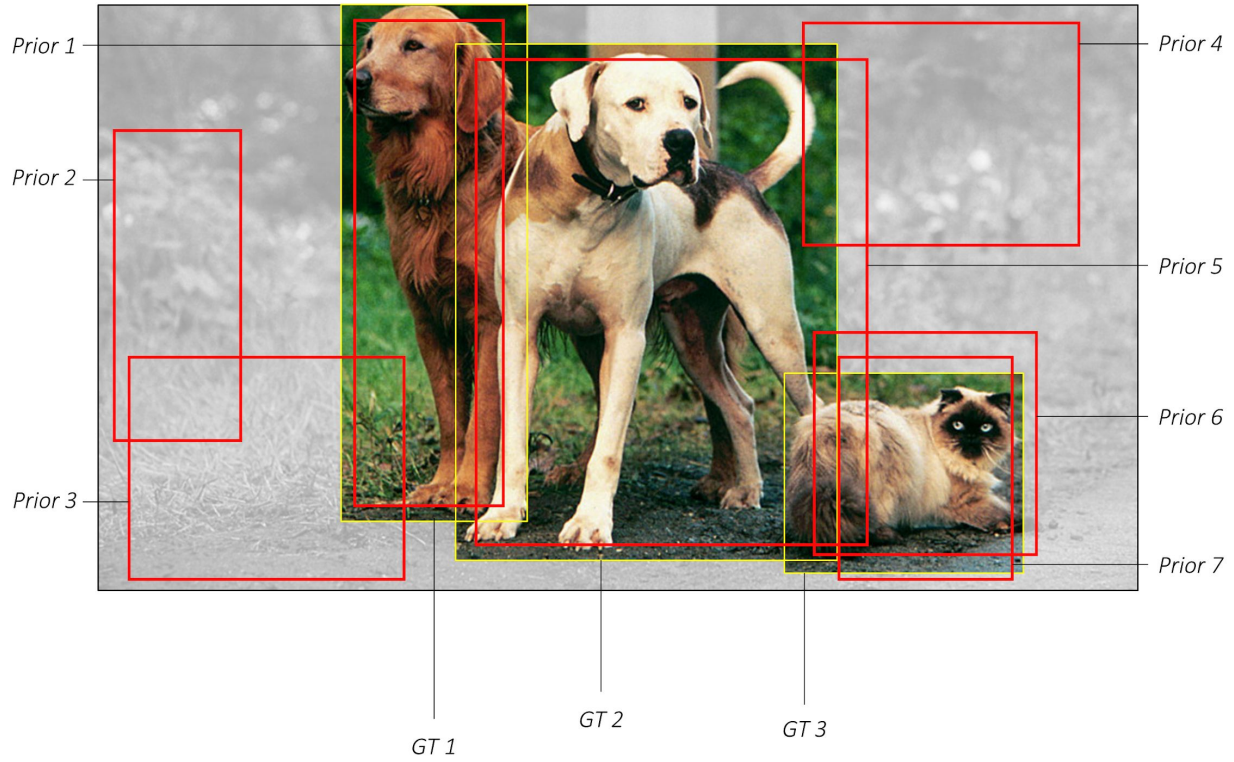
SSD: classification loss + hard negative mining



$$L_{conf} = \frac{1}{n_{positives}} \left(\sum_{positives} CE\ Loss + \sum_{hard\ negatives} CE\ Loss \right)$$

https://github.com/sgrvinod/a-PyTorch-Tutorial-to-Object-Detection

SSD: classification loss + hard negative mining



$$L_{conf} = \frac{1}{n_{positives}} \left(\sum_{positives} CE\ Loss + \sum_{hard\ negatives} CE\ Loss \right)$$

N_hard_negatives = 3 * N_positives

https://github.com/sgrvinod/a-PyTorch-Tutorial-to-Object-Detection

SSD: full multibox loss

$$L_{loc} = \frac{1}{n_{positives}} \left(\sum_{positives} Smooth\ L_1\ Loss \right)$$

$$L_{conf} = \frac{1}{n_{positives}} \left(\sum_{positives} CE\ Loss + \sum_{hard\ negatives} CE\ Loss \right)$$

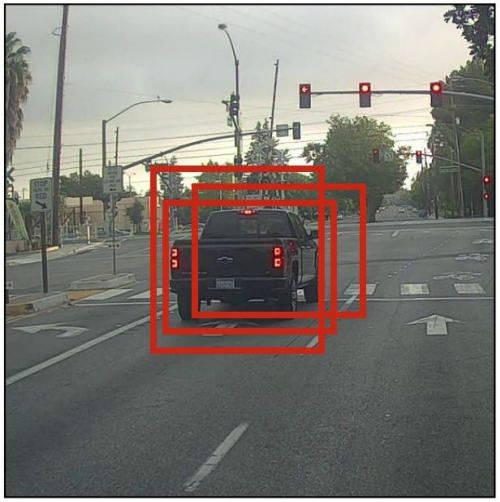
N_hard_negatives = 3 * N_positives

$$L = L_{conf} + \alpha \cdot L_{loc}$$

https://github.com/sgrvinod/a-PyTorch-Tutorial-to-Object-Detection

Detection: non-max suppression

Before non-max suppression

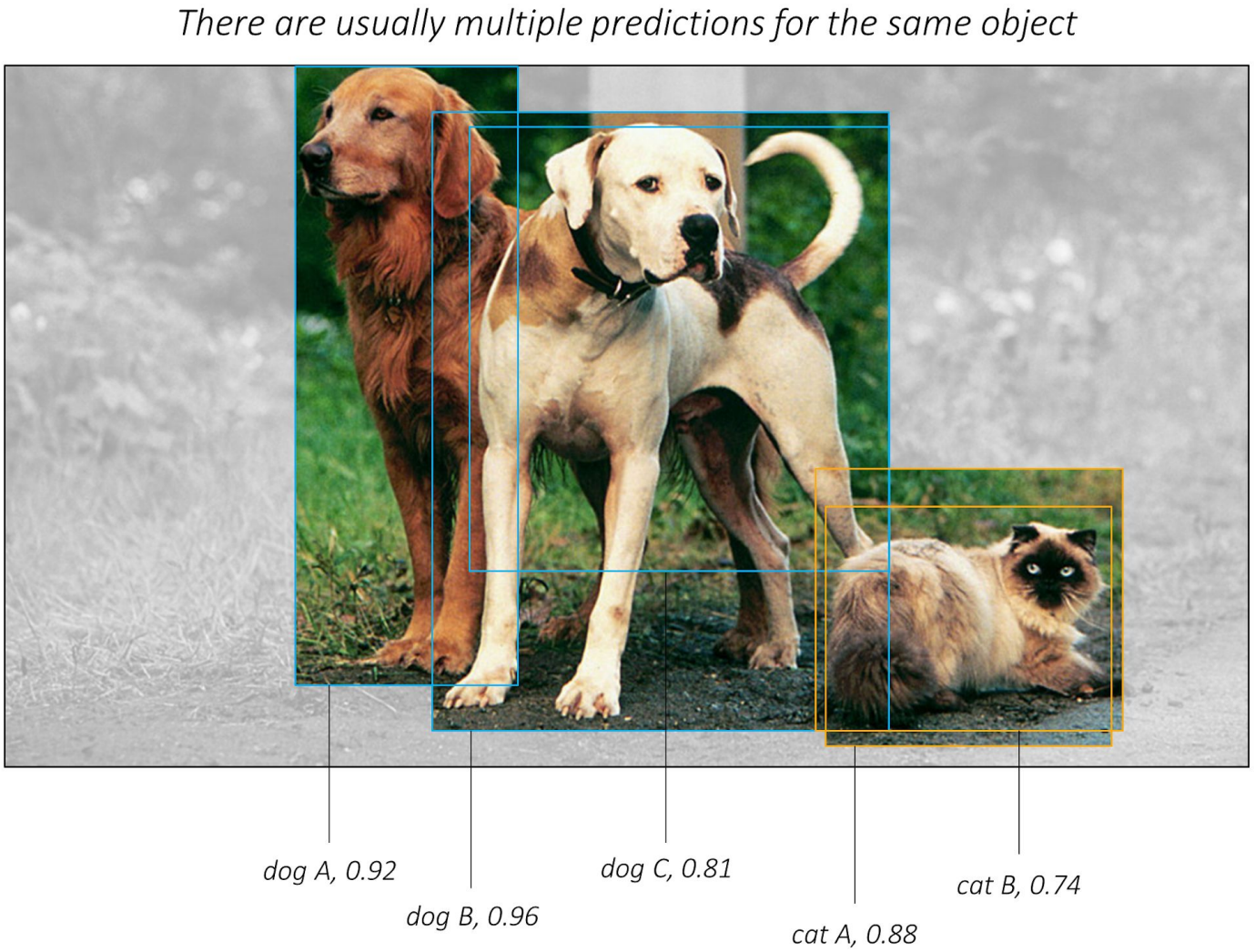


Non-Max Suppression

After non-max suppression

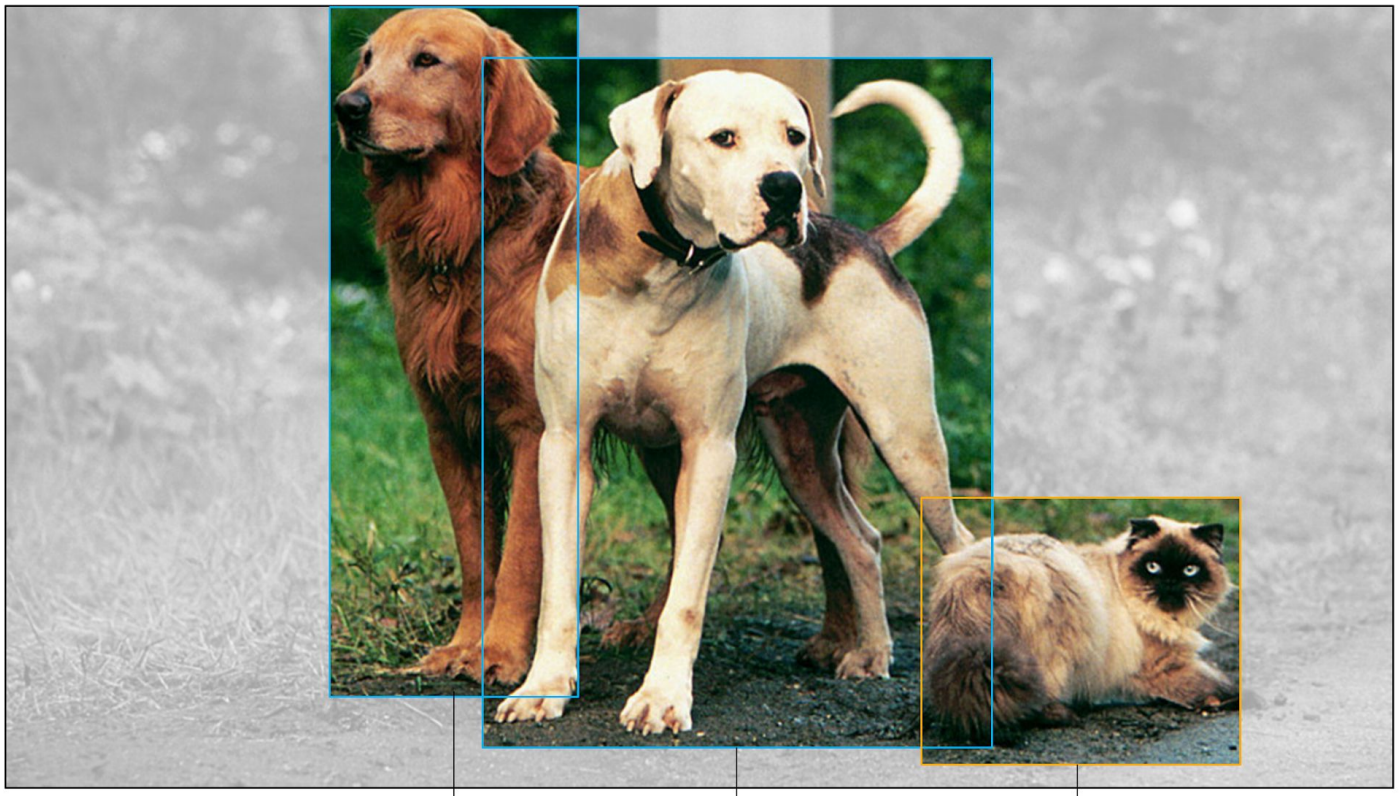
For each class separately!

Detection: non-max suppression



https://github.com/sgrvinod/a-PyTorch-Tutorial-to-Object-Detection

Detection: non-max suppression



https://github.com/sgrvinod/a-PyTorch-Tutorial-to-Object-Detection

Summary

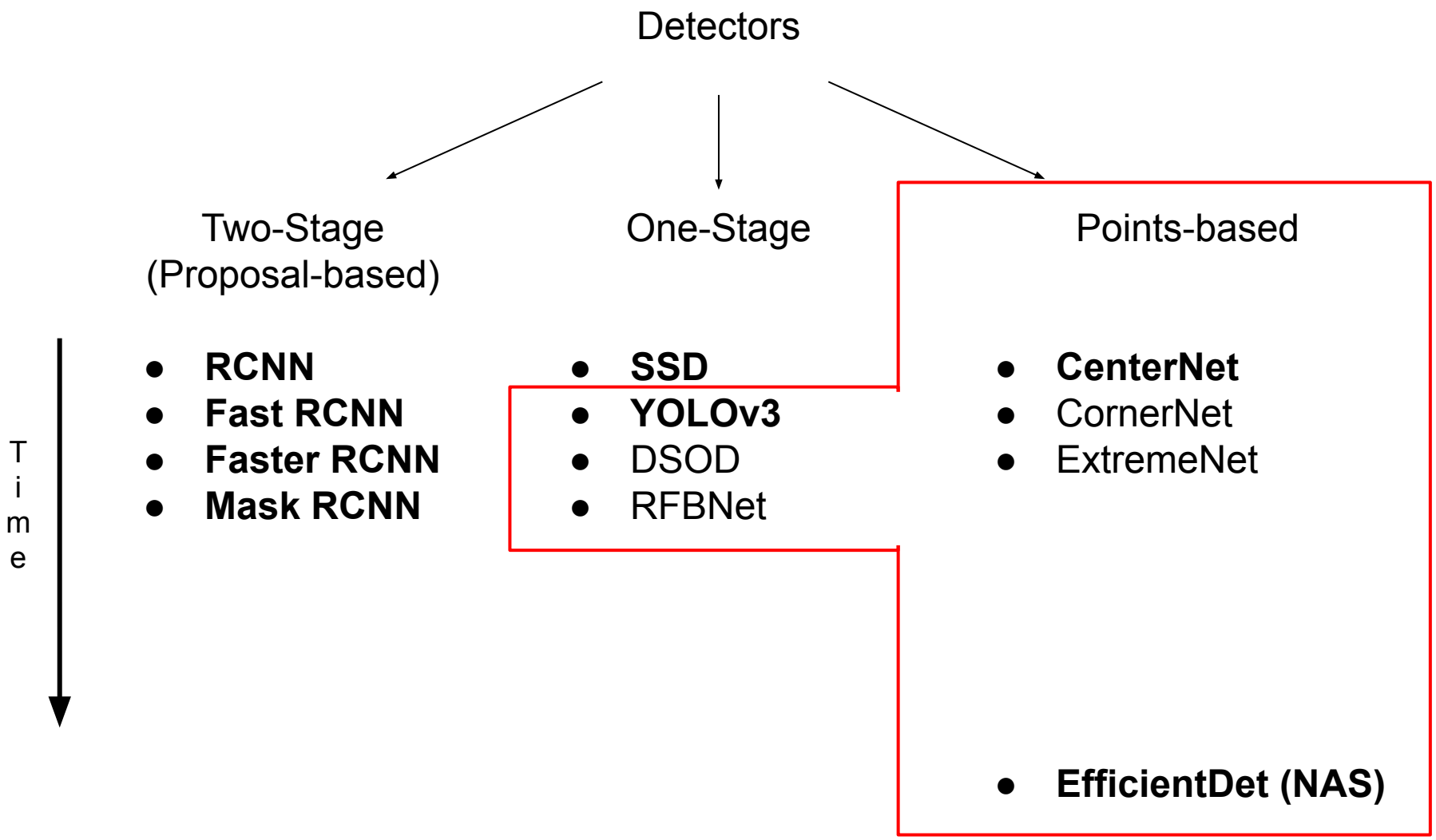
1. One-stage detectors

2. In-depth SSD architecture

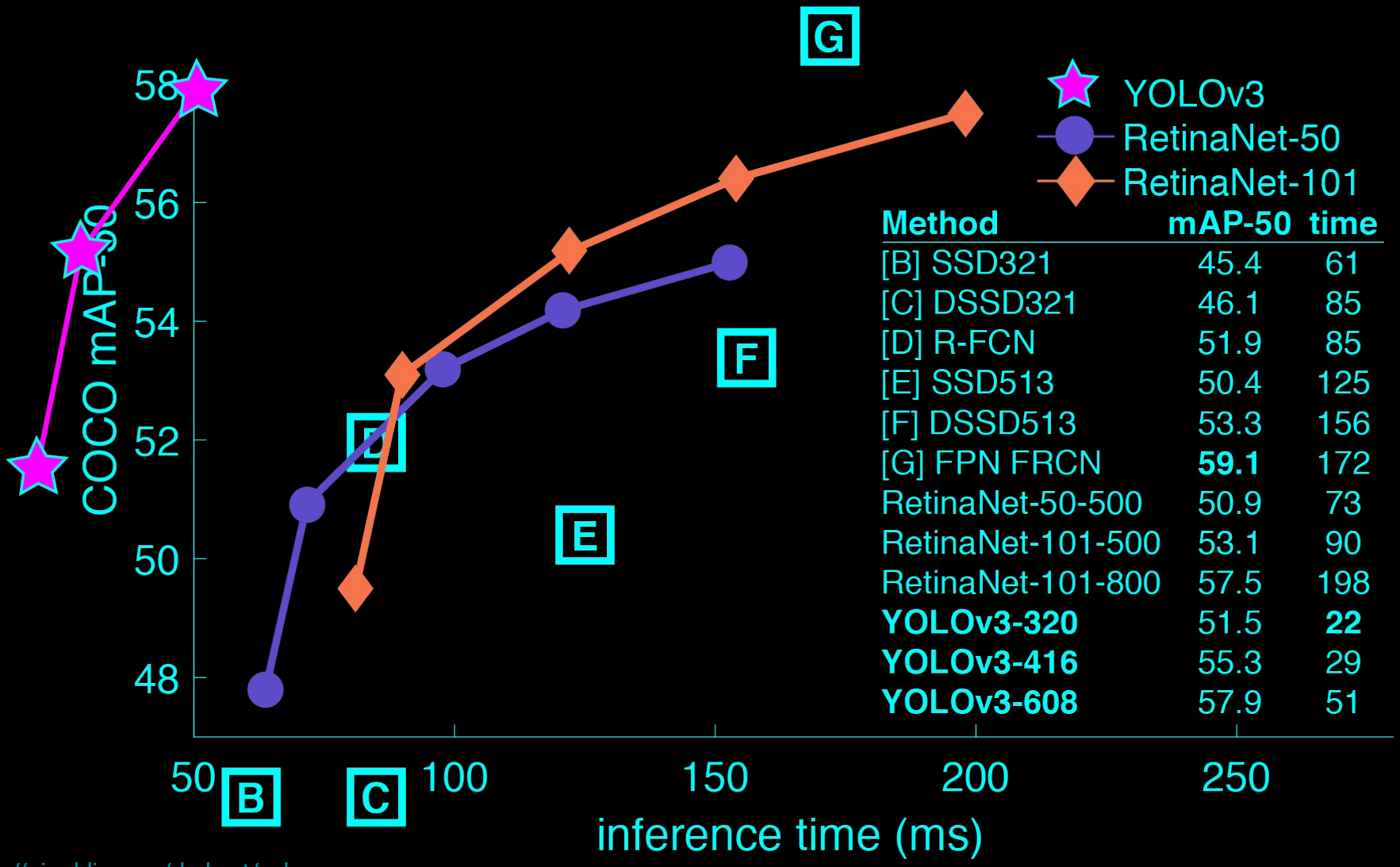
Modern approaches

State-of-the-Art

Detection: Taxonomy



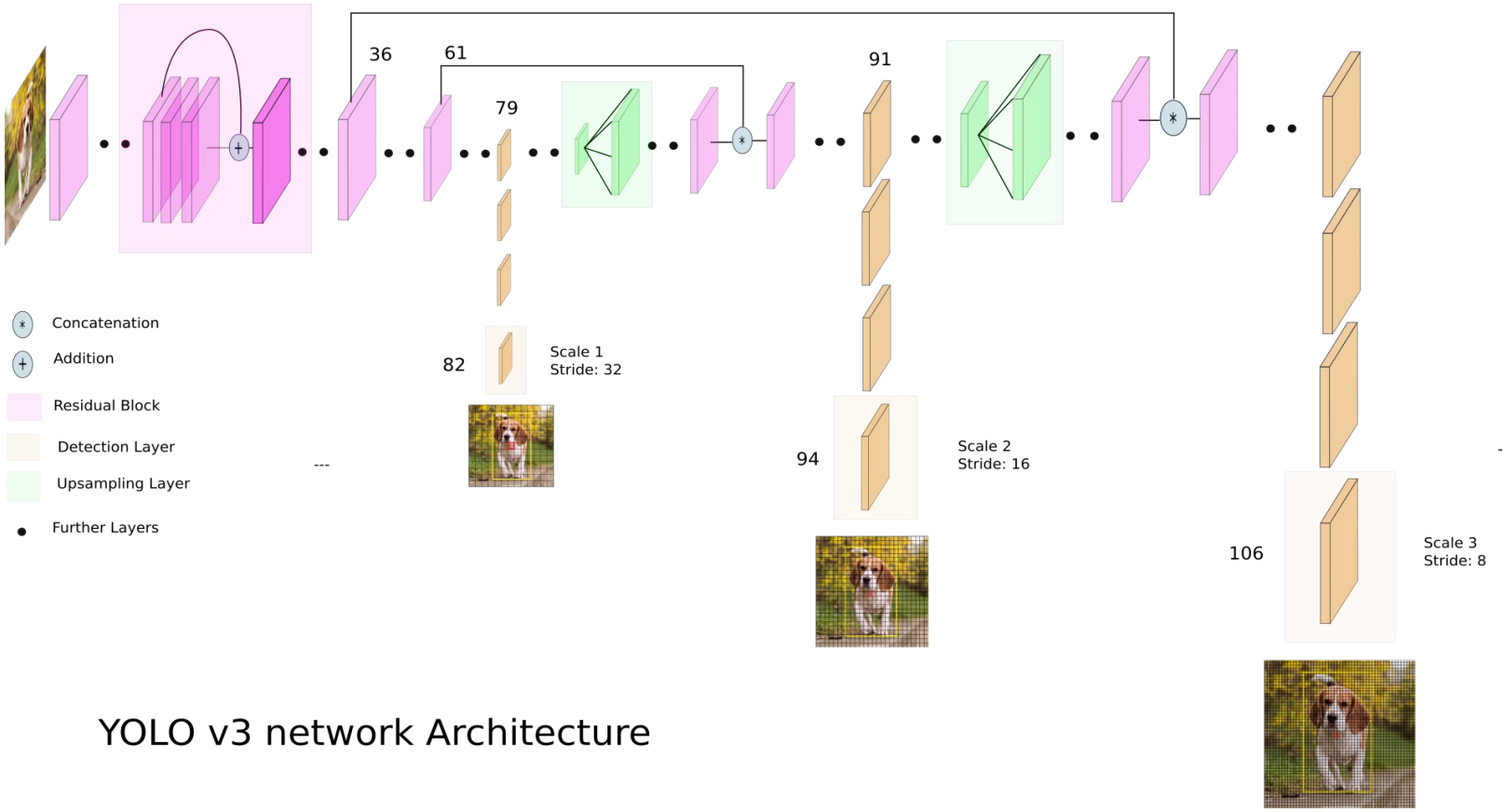
Incremental improvement: YOLOv3 (2018)



Method	mAP-50	time
[B] SSD321	45.4	61
[C] DSSD321	46.1	85
[D] R-FCN	51.9	85
[E] SSD513	50.4	125
[F] DSSD513	53.3	156
[G] FPN FRCN	**59.1**	172
RetinaNet-50-500	50.9	73
RetinaNet-101-500	53.1	90
RetinaNet-101-800	57.5	198
YOLOv3-320	51.5	**22**
YOLOv3-416	55.3	29
YOLOv3-608	57.9	51

https://pjreddie.com/darknet/yolo

Incremental improvement: YOLOv3 (2018)



YOLO v3 network Architecture

Optimal detection: YOLOv4 (2020)

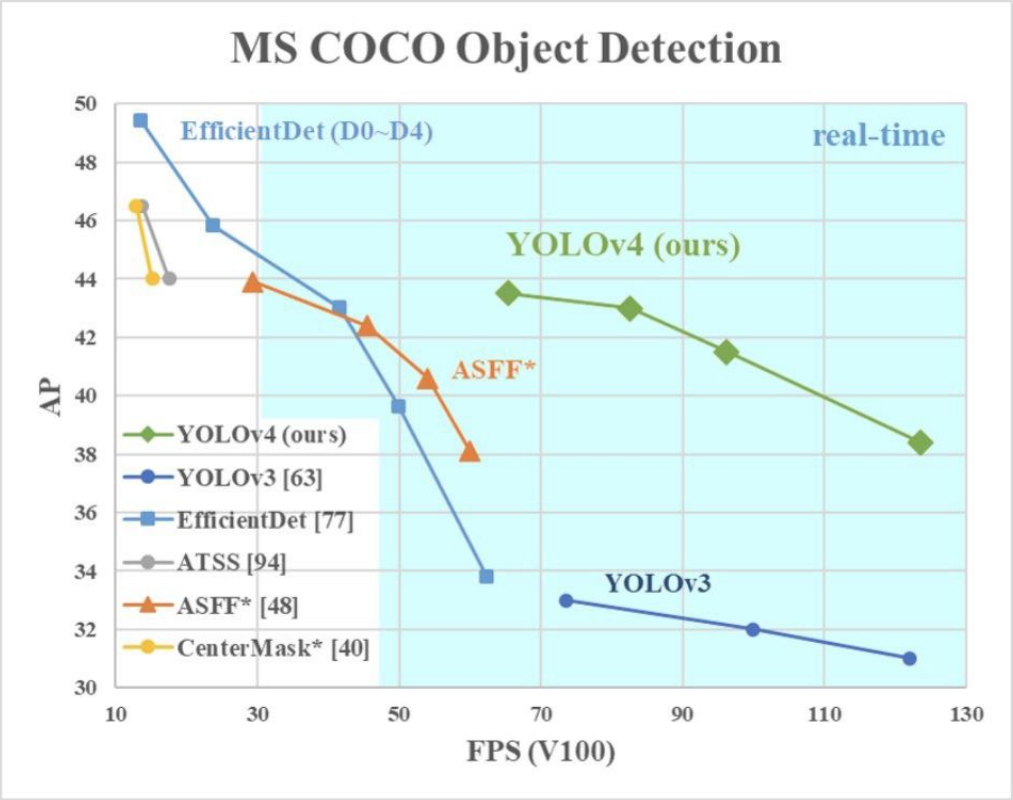


Figure 1: Comparison of the proposed YOLOv4 and other state-of-the-art object detectors. YOLOv4 runs twice faster than EfficientDet with comparable performance. Improves YOLOv3's AP and FPS by 10% and 12%, respectively.

https://habr.com/ru/post/503200/

Optimal detection: YOLOv4 (2020)

3.4. YOLOv4

In this section, we shall elaborate the details of YOLOv4.

YOLOv4 consists of:

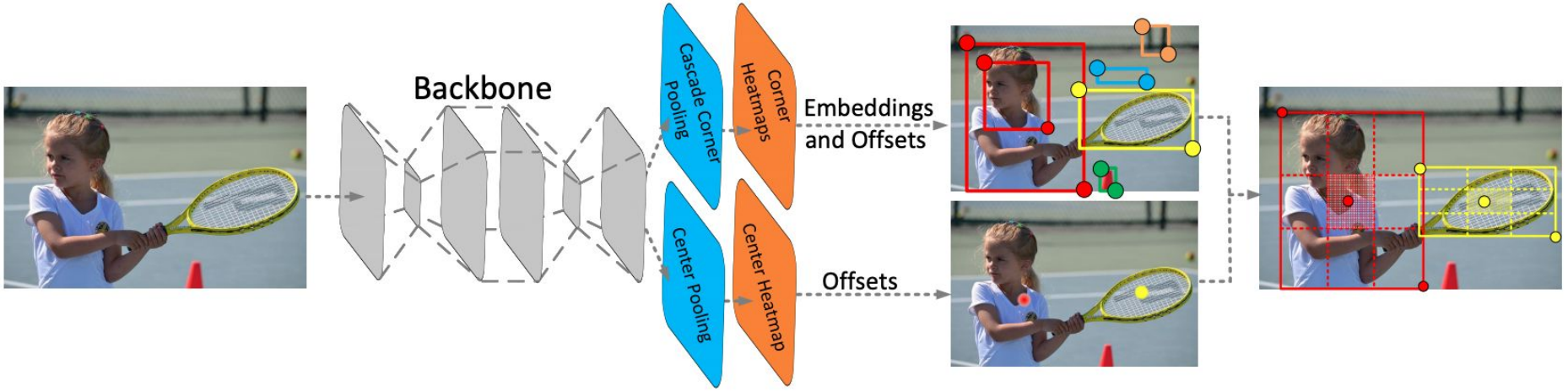
- Backbone: CSPDarknet53 [81]
- Neck: SPP [25], PAN [49]
- Head: YOLOv3 [63]

YOLO v4 uses:

- Bag of Freebies (BoF) for backbone: CutMix and Mosaic data augmentation, DropBlock regularization, Class label smoothing
- Bag of Specials (BoS) for backbone: Mish activation, Cross-stage partial connections (CSP), Multi-input weighted residual connections (MiWRC)
- Bag of Freebies (BoF) for detector: CIoU-loss, CmBN, DropBlock regularization, Mosaic data augmentation, Self-Adversarial Training, Eliminate grid sensitivity, Using multiple anchors for a single ground truth, Cosine annealing scheduler [52], Optimal hyper-parameters, Random training shapes
- Bag of Specials (BoS) for detector: Mish activation, SPP-block, SAM-block, PAN path-aggregation block, DIoU-NMS

https://habr.com/ru/post/503200/

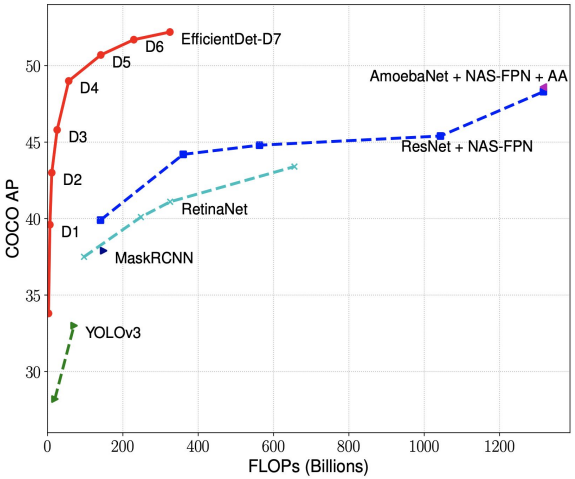
Objects as Points: CenterNet (2019)

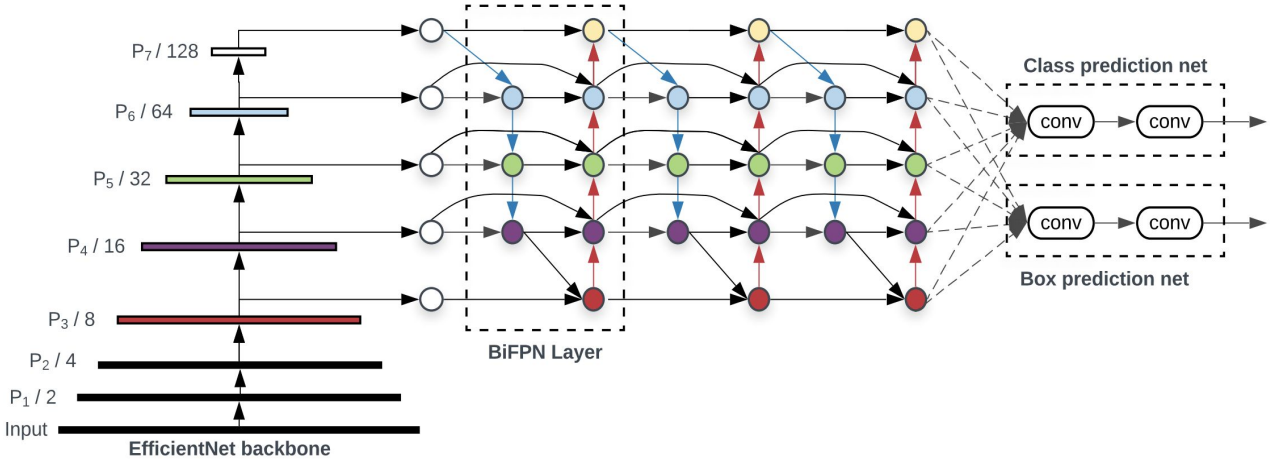


keypoint heatmap [C] | local offset [2] | object size [2] | 3D size [3] | depth [1] | orientation [8] | joint locations [$k \times 2$] | joint heatmap [k] | joint offset [2]

https://arxiv.org/abs/1904.07850

NAS in Detection: EfficientDet (2019-2020)





Idea: EfficientNet + BiFPN

https://github.com/google/automl/tree/master/efficientdet

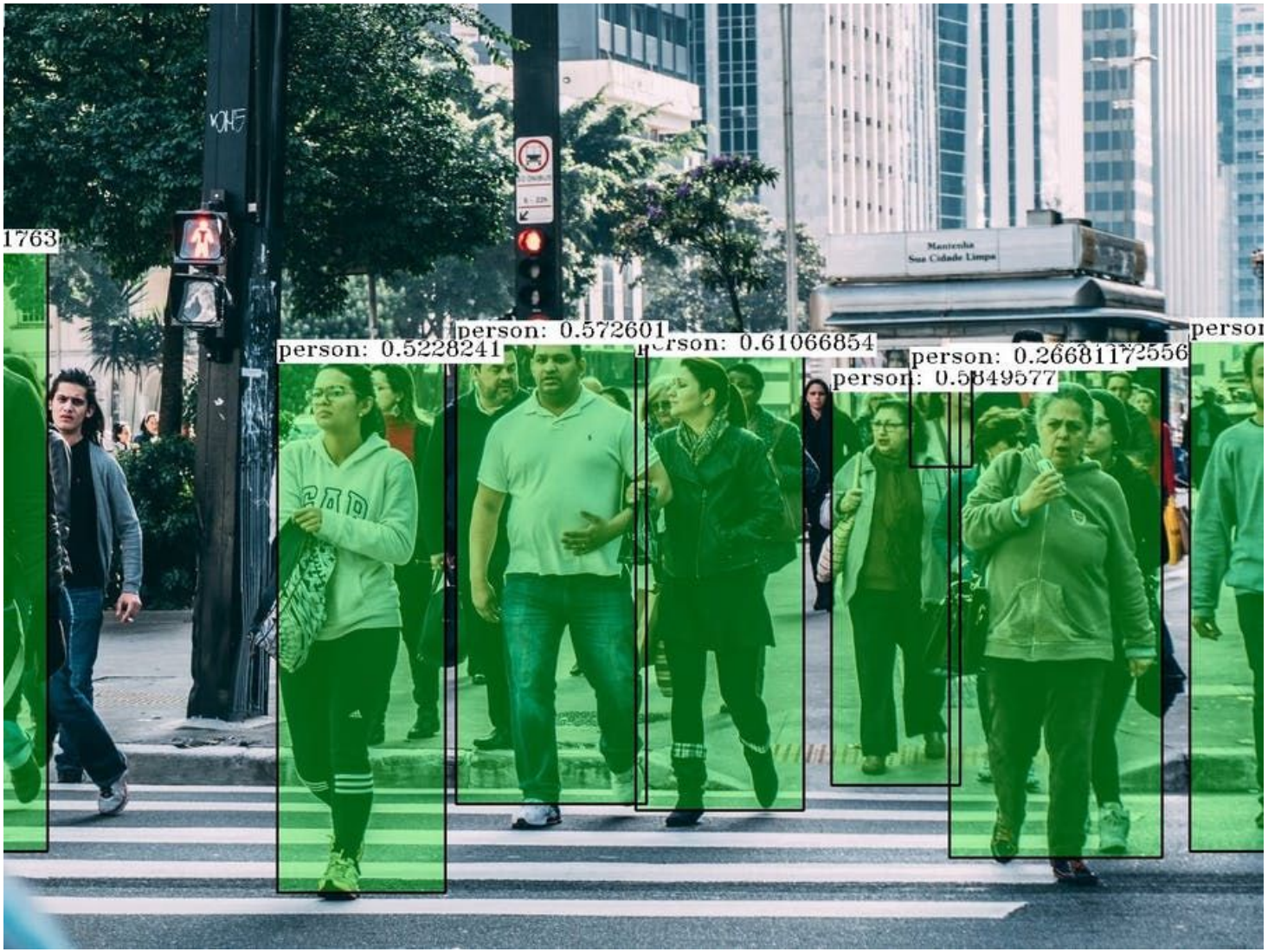
Modern tasks

Specific detection applications

In real life we need to

1. Detect **people**
2. Detect **faces**
3. Detector **text**
4. Do **tracking** through the video (time), not only image
5. Recognize **actions**: proxy, end-to-end
6. Detect in **3D**: self-driving, robotics, VR/AR

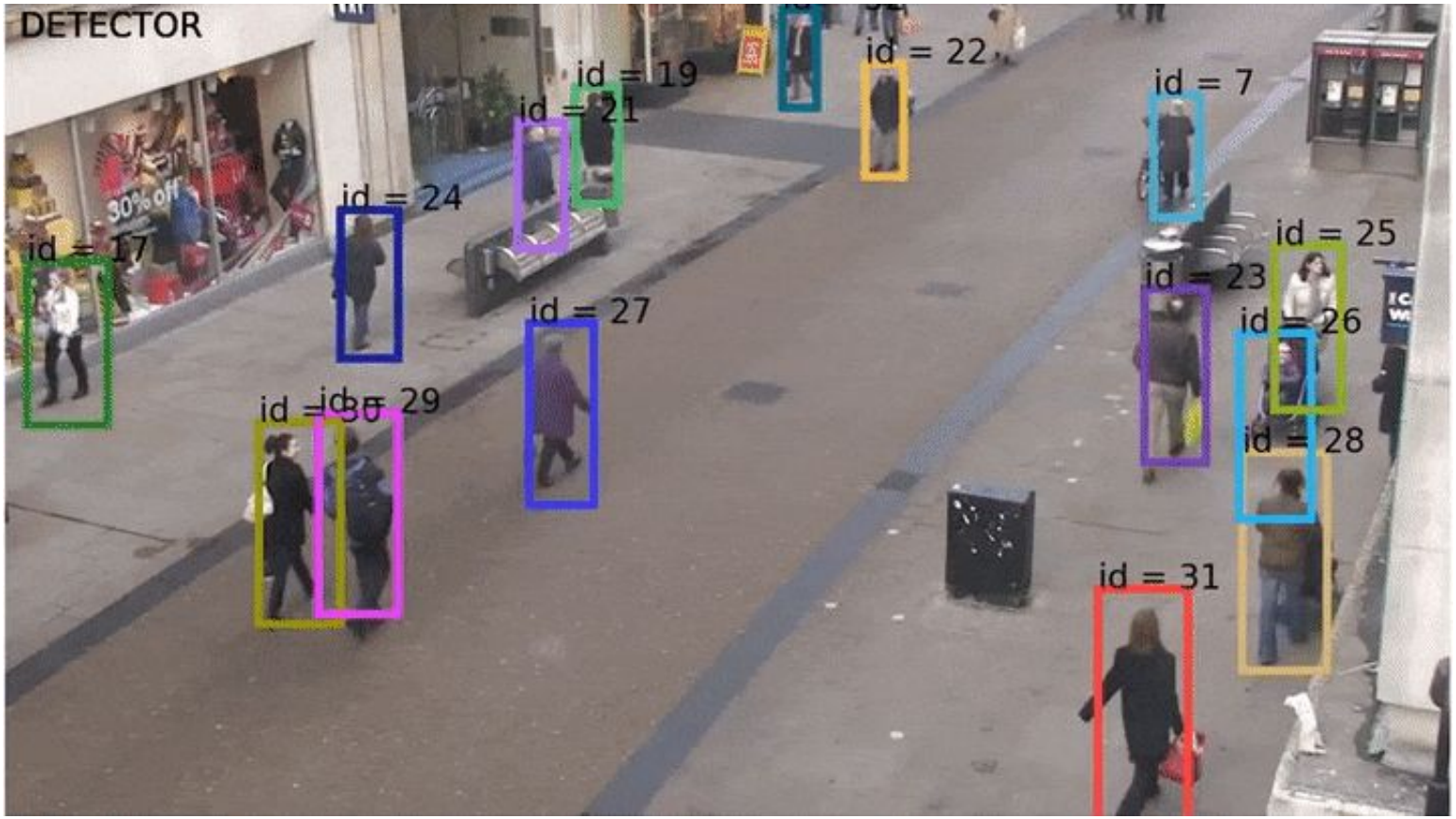
In real life we need to: detect people



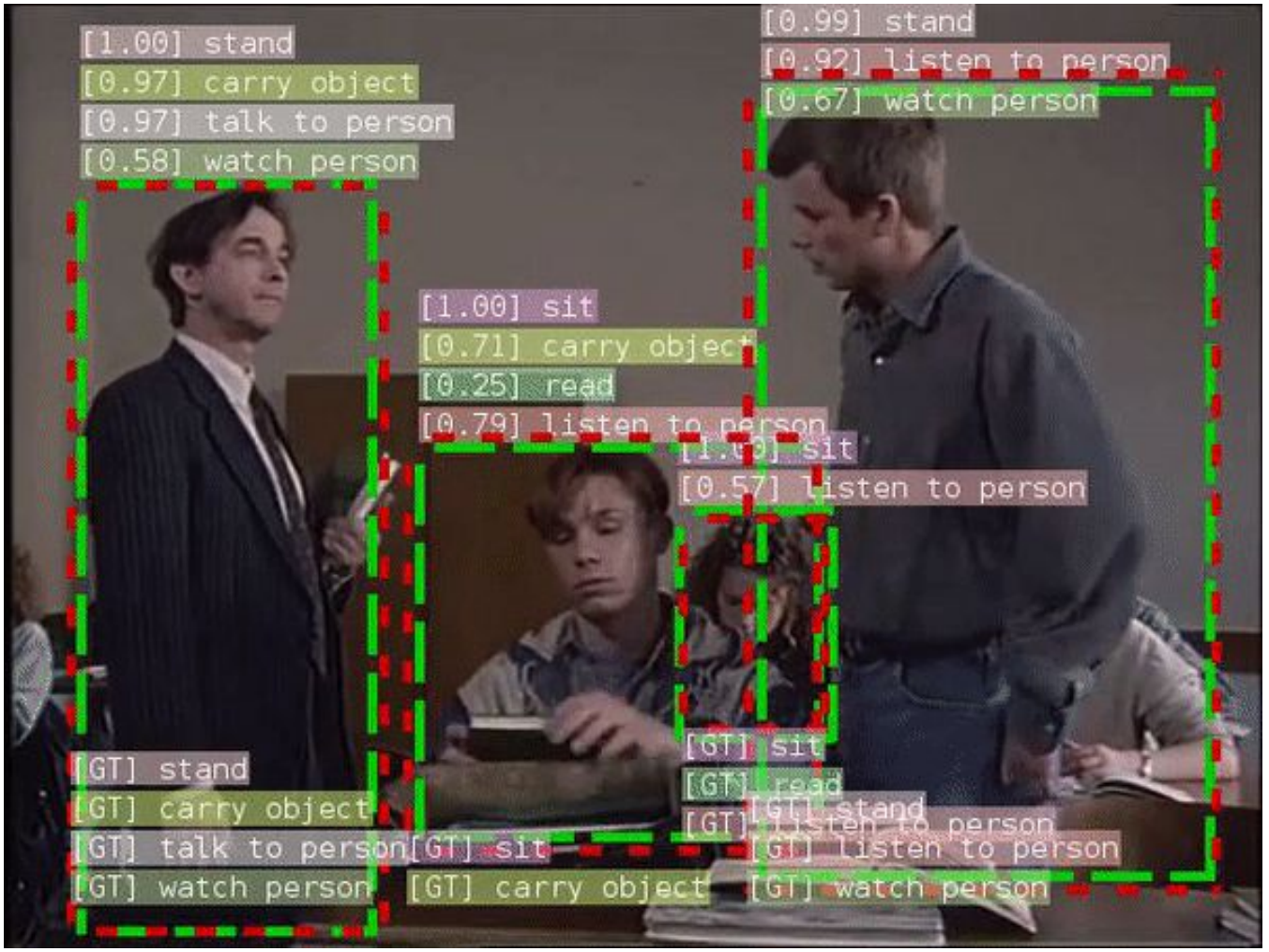
In real life we need to: detect faces

In real life we need to: detect text

In real life we need to: do tracking

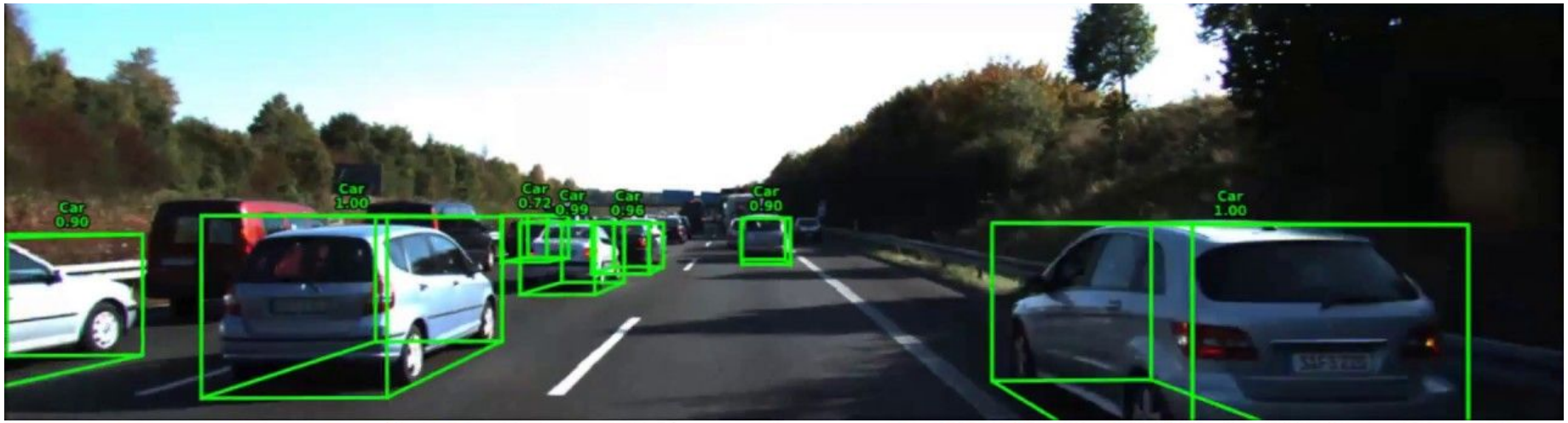


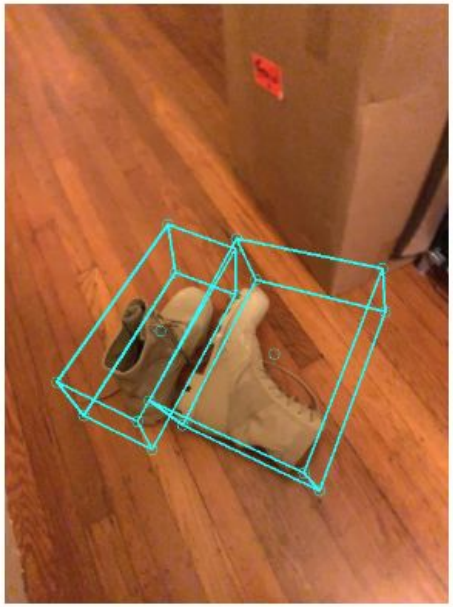
In real life we need to: recognize actions

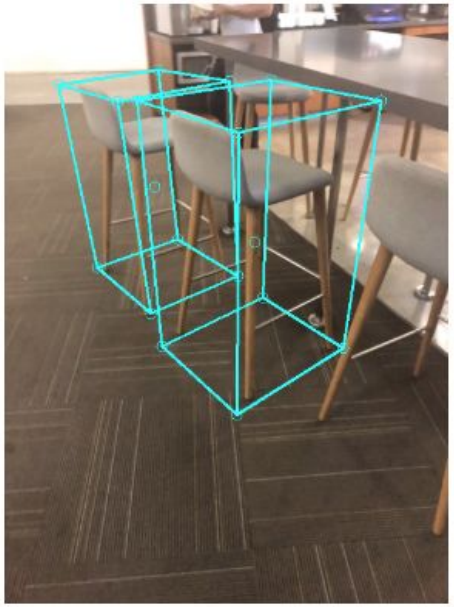


https://github.com/facebookresearch/SlowFast

In real life we need to: detect in 3D

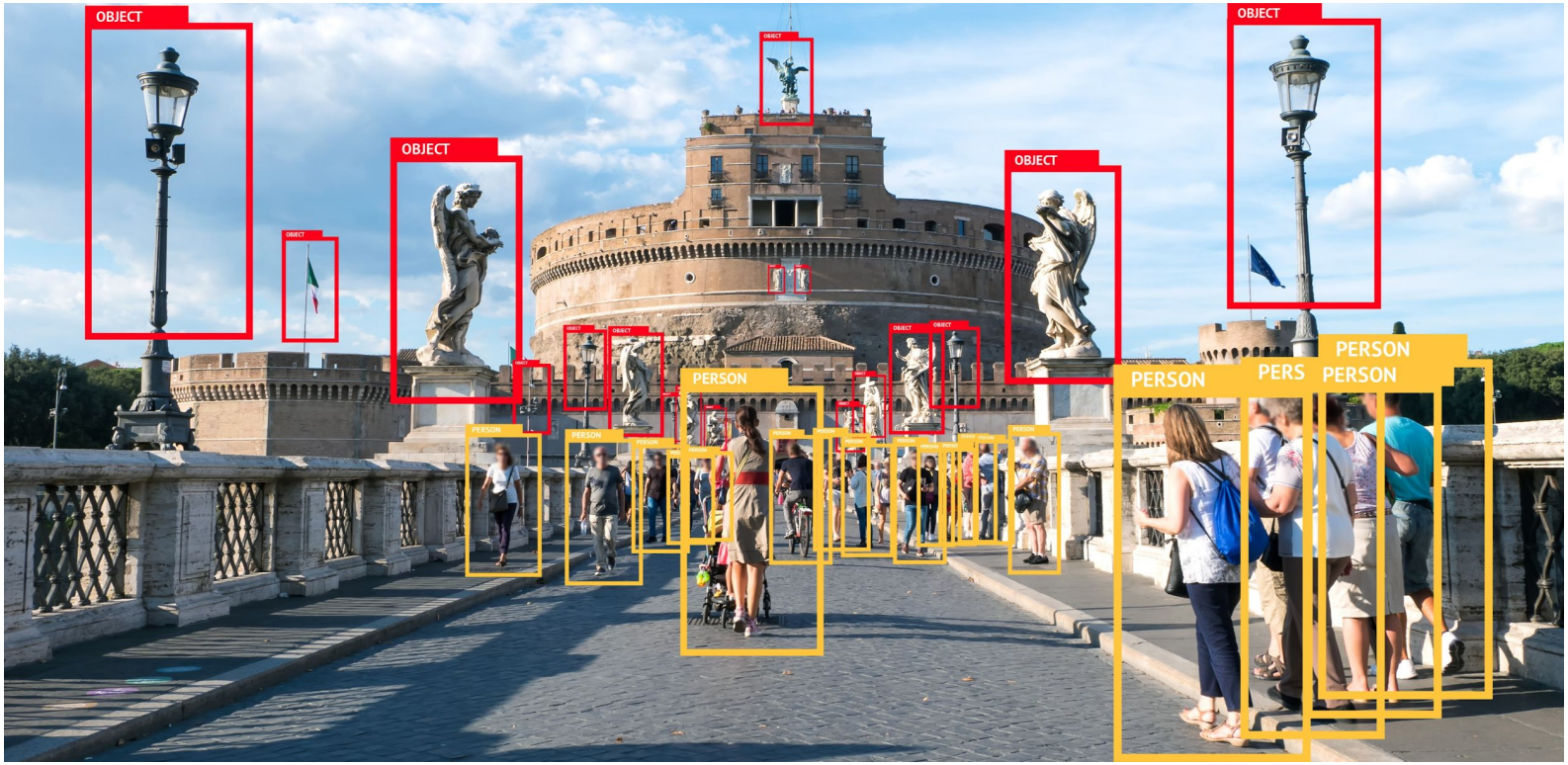






https://google.github.io/mediapipe/solutions/objectron

Solutions to these tasks



https://habr.com/ru/company/mipt/blog/458190/

Lecture summary

1. Classification recap
2. Object Detection basics
3. Two-stage (proposal-based) methods
4. One-stage (single-shot) methods
5. Modern approaches & tasks