Wykrywanie obiektów na obrazie

Dr inż. Karol Majek

Dr inż. Karol Majek

KarolMajek.pl
DeepDrive.pl

Institute of Mathematical Machines

Fraunhofer FKIE

NASK

Currently:

- Warsaw Self-Driving Cars
- Deep Drive PL
- Cufix

WSDC.PL

Warsaw Self-Driving Cars

Warsaw, Poland

509 members · Public group

Organized by **Karol M.** and **2 others**

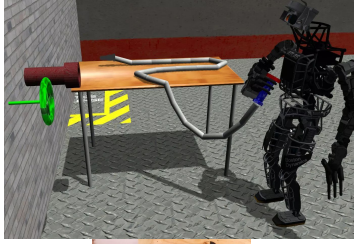
Share:

deepdrive.pl/meetupwsdc

C-ELROB 2011

Eurathlon 2013

2013



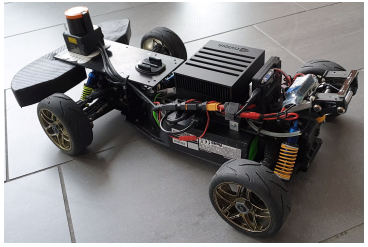
2013

2014 Eurathlon 2015

Enrich 2017

ERL 2017

Self-Racing Cars 2017/2018



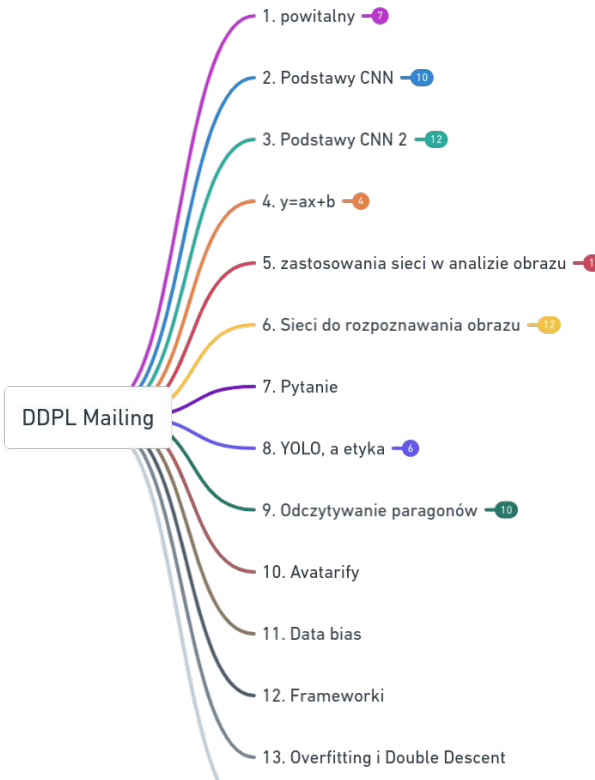
F1/10 2018/2019

ELROB 2018

Indy Autonomous 2020

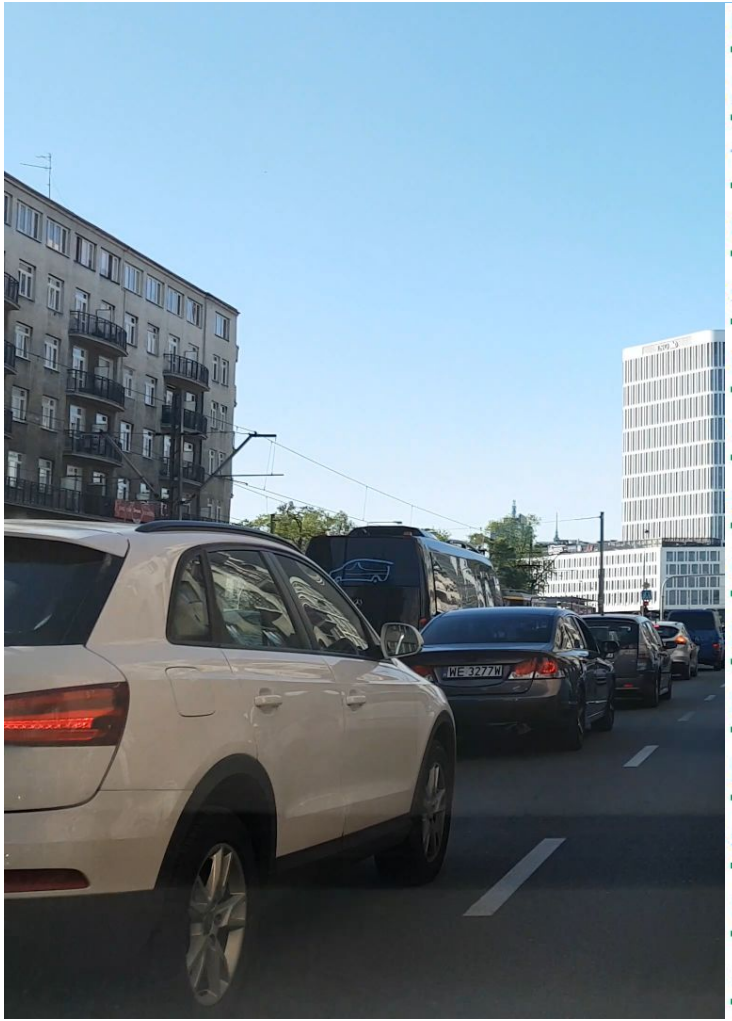
DeepDrive.pl/30

- zapis #30DevStories
- cotygodniowy mailing

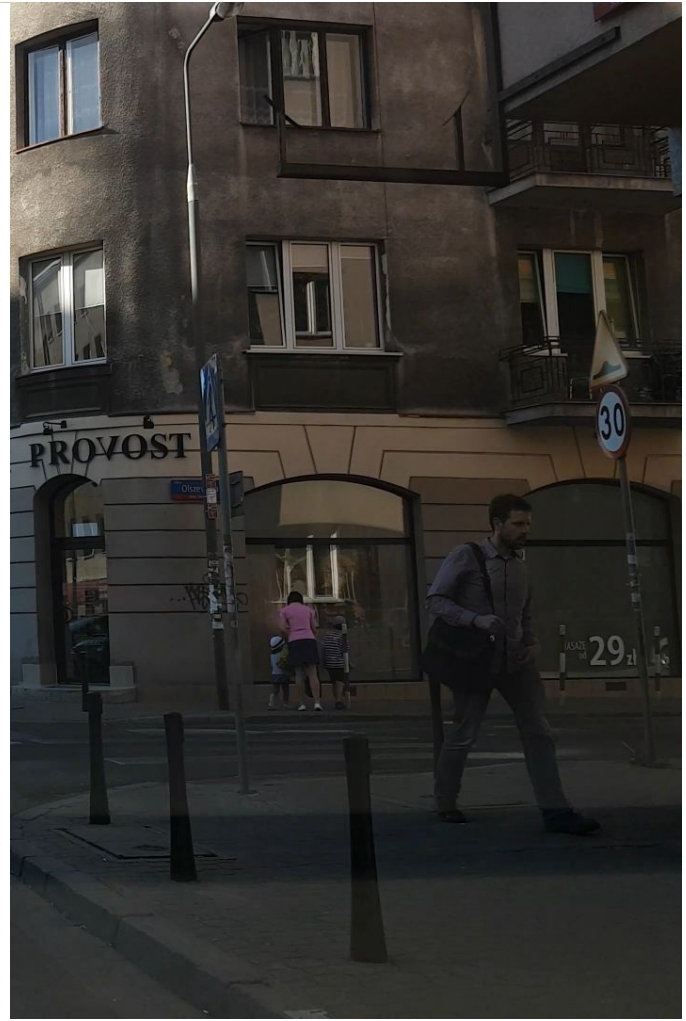


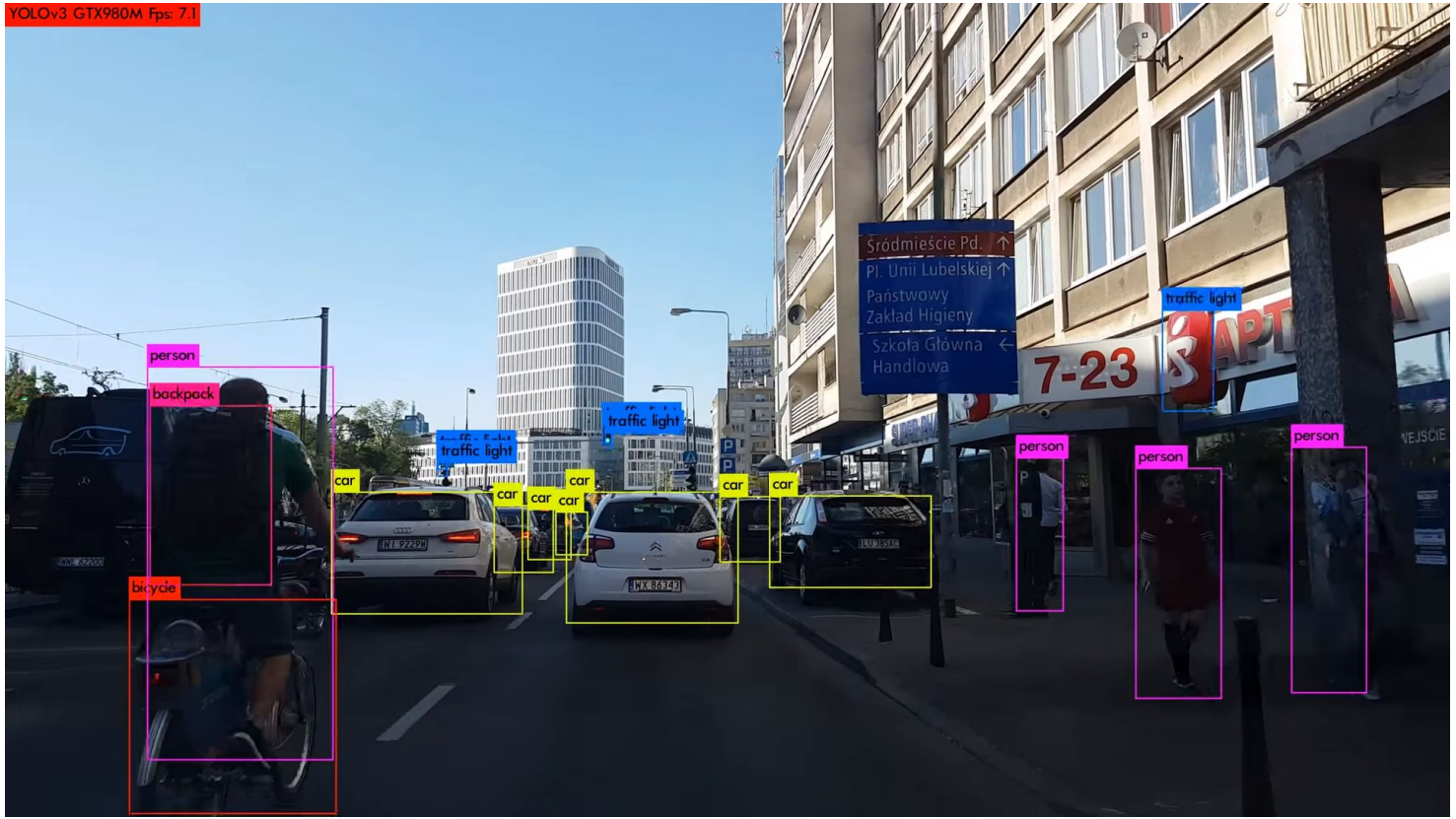
Jak rozumiemy wykrywanie obiektów?


BLOCK PROVOST
30
WE 3277W
WX 8634J



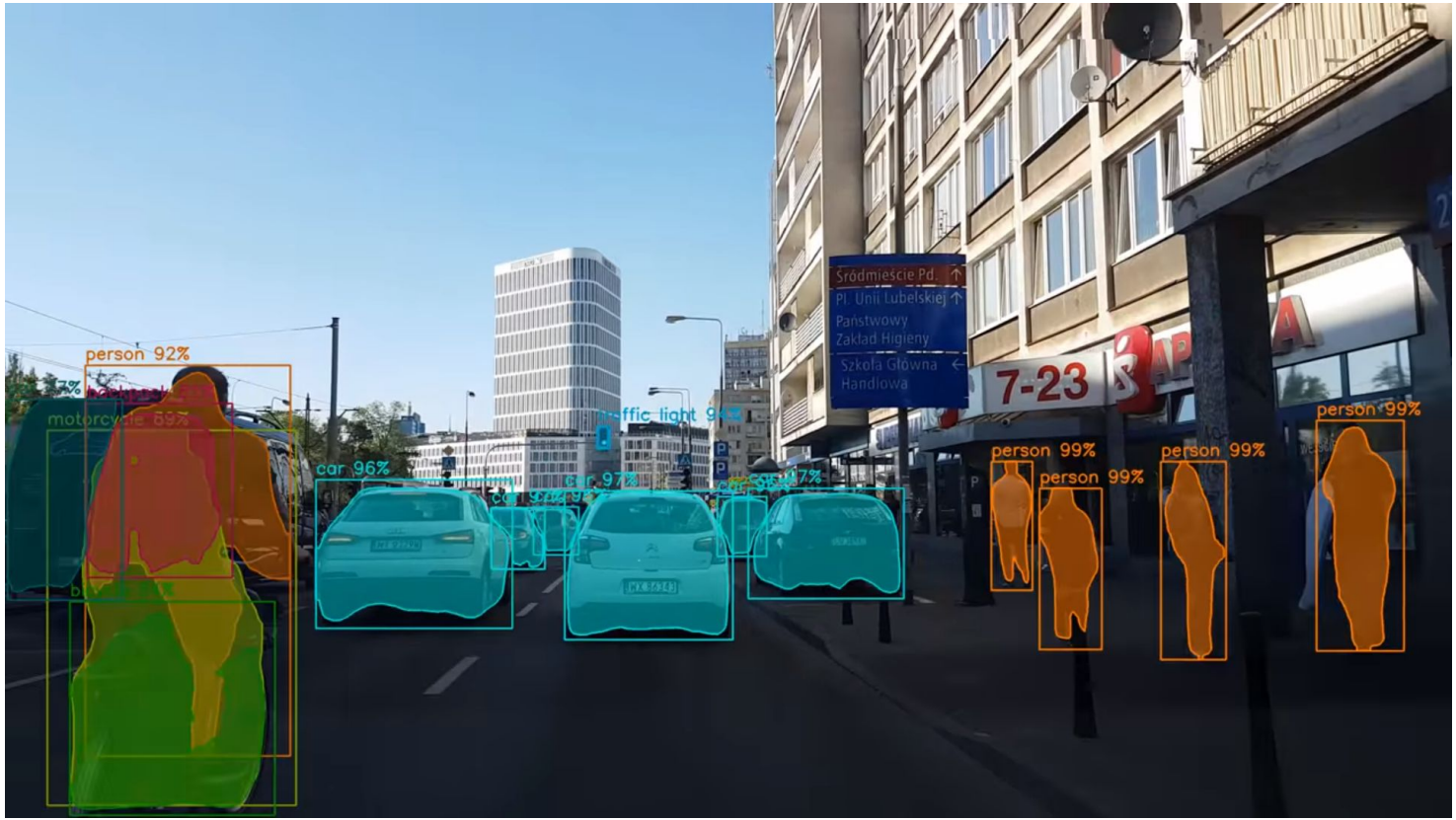
Urban Area	96%
Neighbourhood	96%
Town	95%
City	94%
Street	93%
Road	92%
Metropolitan Area	92%
Human Settlement	91%
Downtown	90%
Residential Area	84%
Building	83%
Lane	82%
Snapshot	82%
Metropolis	81%
Infrastructure	80%





Object detection

youtube.com/karolmajek



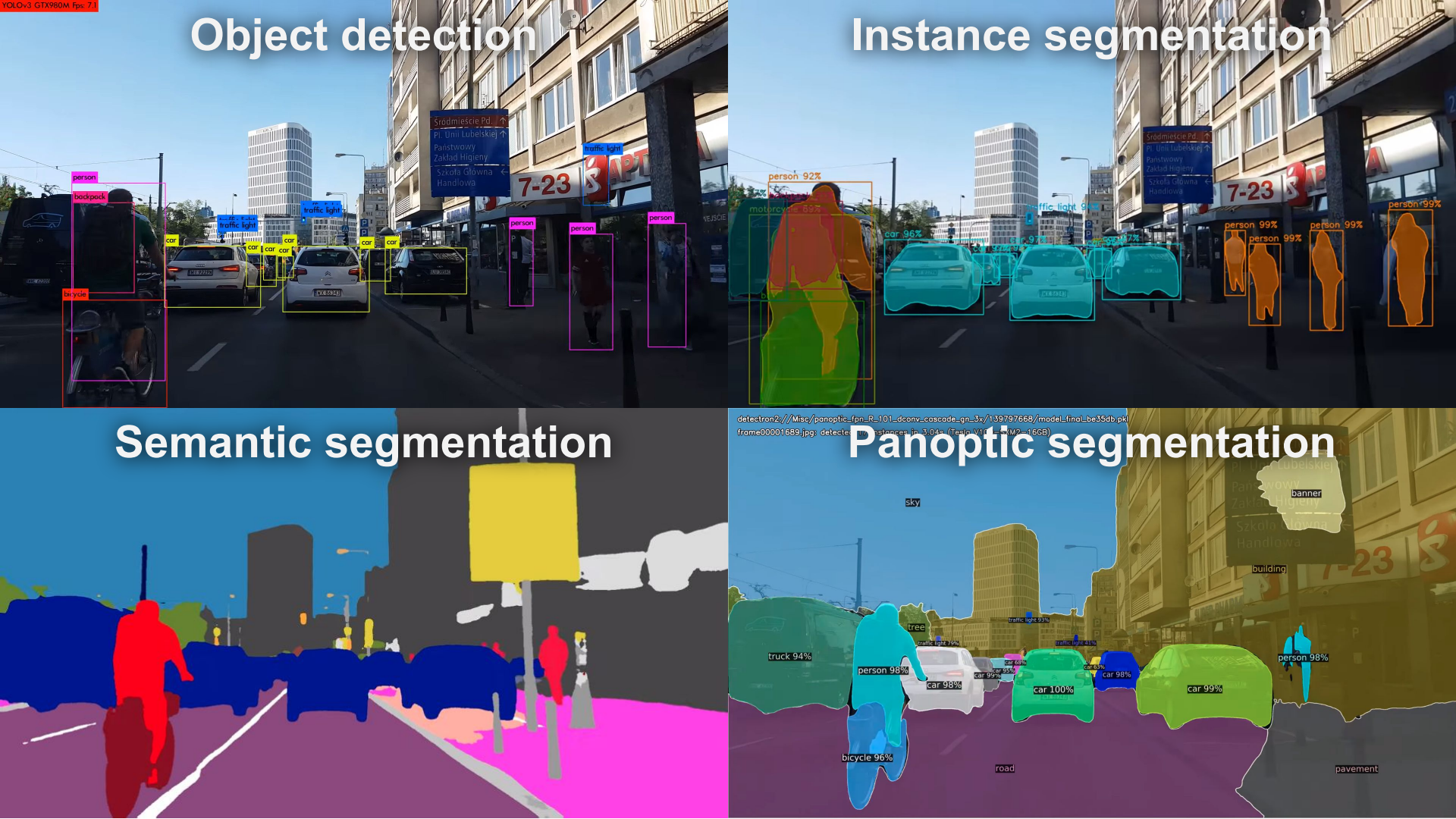
Instance segmentation

youtube.com/karolmajek

Semantic segmentation

youtube.com/karolmajek

Kirillov, Alexander, et al. "**Panoptic segmentation**."
arXiv preprint arXiv:1801.00868 (2018).


YOLOv3 GTX980M Fps: 7.1
Object detection
Instance segmentation
Semantic segmentation
Panoptic segmentation
person
backpack
bicycle
car
traffic light
sky
banner
building
tree
truck 94%
person 98%
car 98%
car 100%
car 99%
bicycle 96%
road
pavement

Annotation tools

LabelImg

LabelImg

LabelImg


Open
Open Dir
Change Save Dir
Next Image
Prev Image
Verify Image
Save
Create RectBox
Duplicate RectBox
Delete RectBox
Zoom In
83 %
Zoom Out
Fit Window
Box Labels
Edit Label
difficult
Use default label
pipes
barrier
barrier
barrier
barrier
blocked
File List

LabelImg

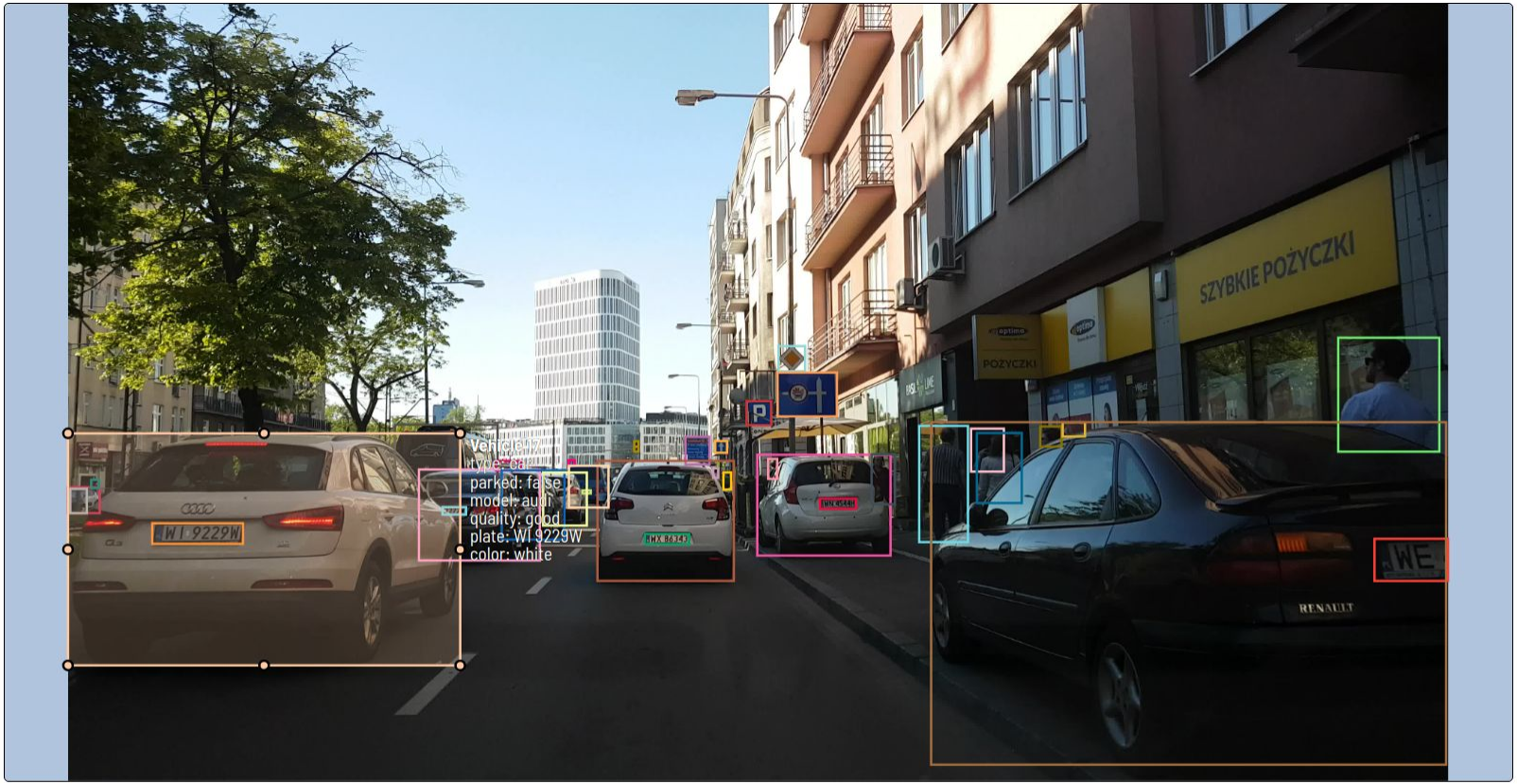
Input:

- images

Output:

- xml per image
- bounding boxes

OpenCV CVAT



Objects | Labels

vehicle 44 [box, interpolation]

Label: Vehicle

Attributes

Quality: Good / Bad

Type: Car

Model:

Color: Gray

Plate:

Parked

person 38 [box, interpolation]

Label: Person

Attributes

Quality: Good / Bad

Task: Walking

person 37 [box, interpolation]

Label: Person

Attributes

Quality

Create Shape

Merge Shapes

Group Shapes

Vehicle

Interpolation

Box

Poly Shape Size: 0-100

Open Menu | Filter: car[attr/model="mazda"] | Reset | None None | Frame 0

Fill Opacity: | Selected Fill Opacity: | Black Stroke: | Color by: Instance / Group / Label

OpenCV CVAT

Input:

- video/videos/images

Output:

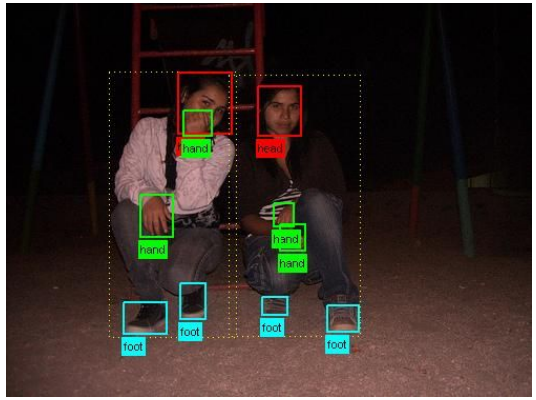
- xml per task (video/set of images)
  - bounding boxes
  - polygons
  - polylines
  - point sets

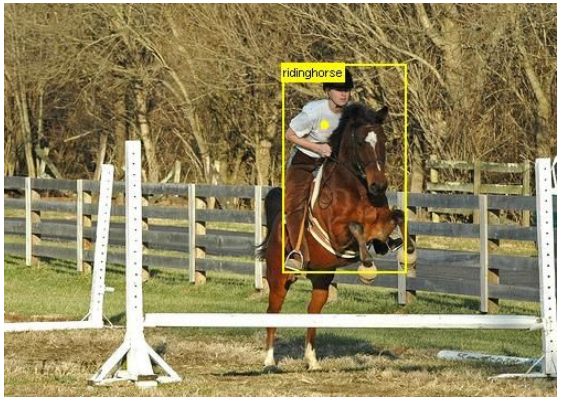
Datasets

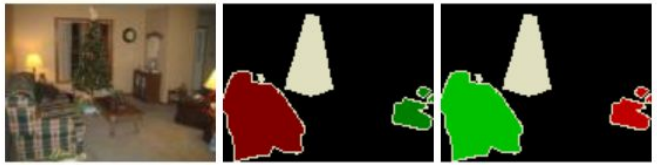
PASCAL VOC

2005-2012


pottedplantTruncOcc
pottedplantTruncOcc
chairRear


hand
head
hand
hand
hand
foot
foot
foot
foot


ridinghorse



PASCAL VOC 2005

4 classes: bicycles, cars, motorbikes, people

1 578 images: 2 209 objects

PASCAL VOC 2005

4 classes: bicycles, cars, motorbikes, people

1 578 images: 2 209 objects

PASCAL VOC 2012

20 classes

11 530 images: 27 450 objects

6 929 segmentations

Object Detection: PASCAL VOC mean Average Precision (mAP)

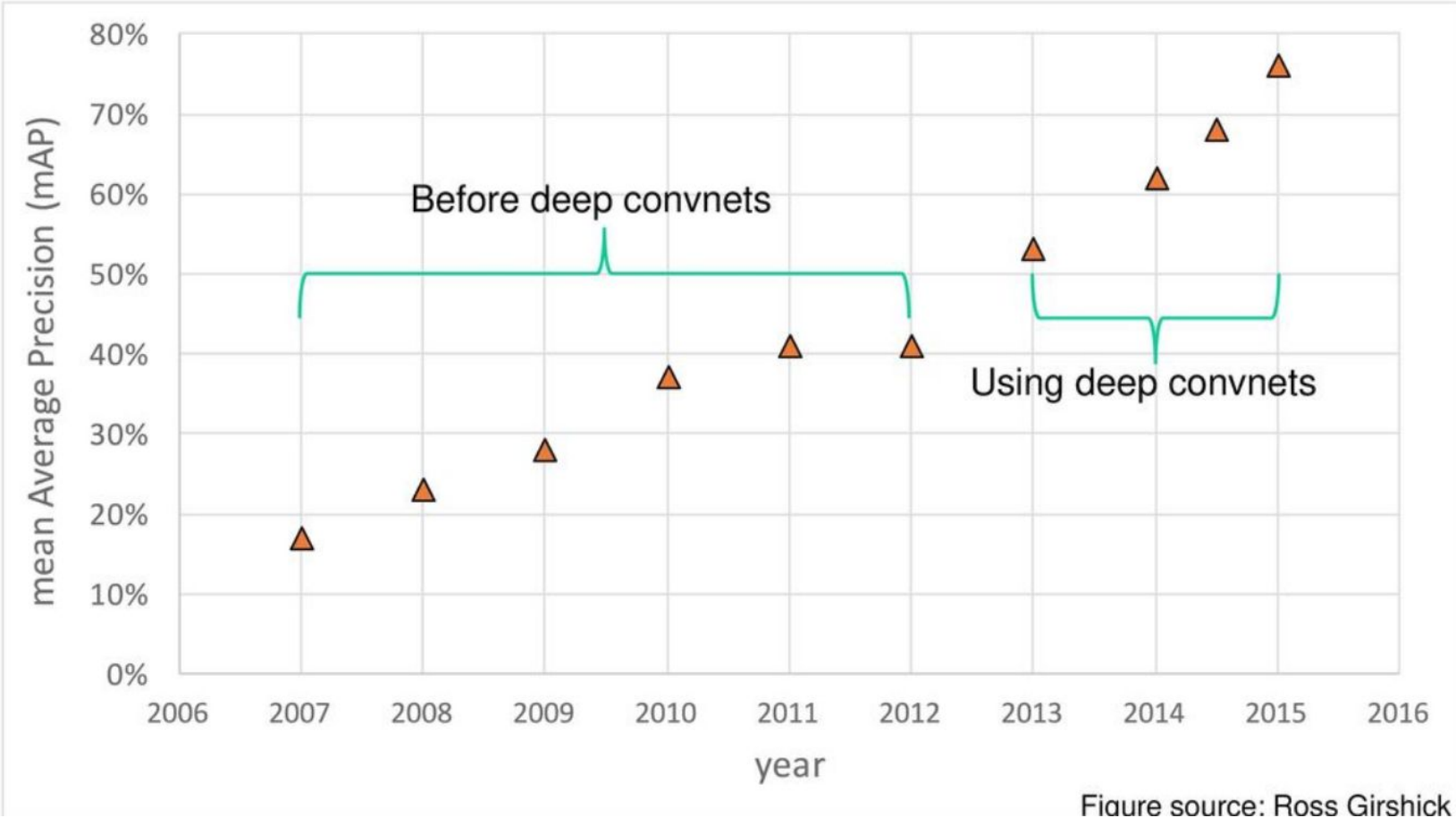
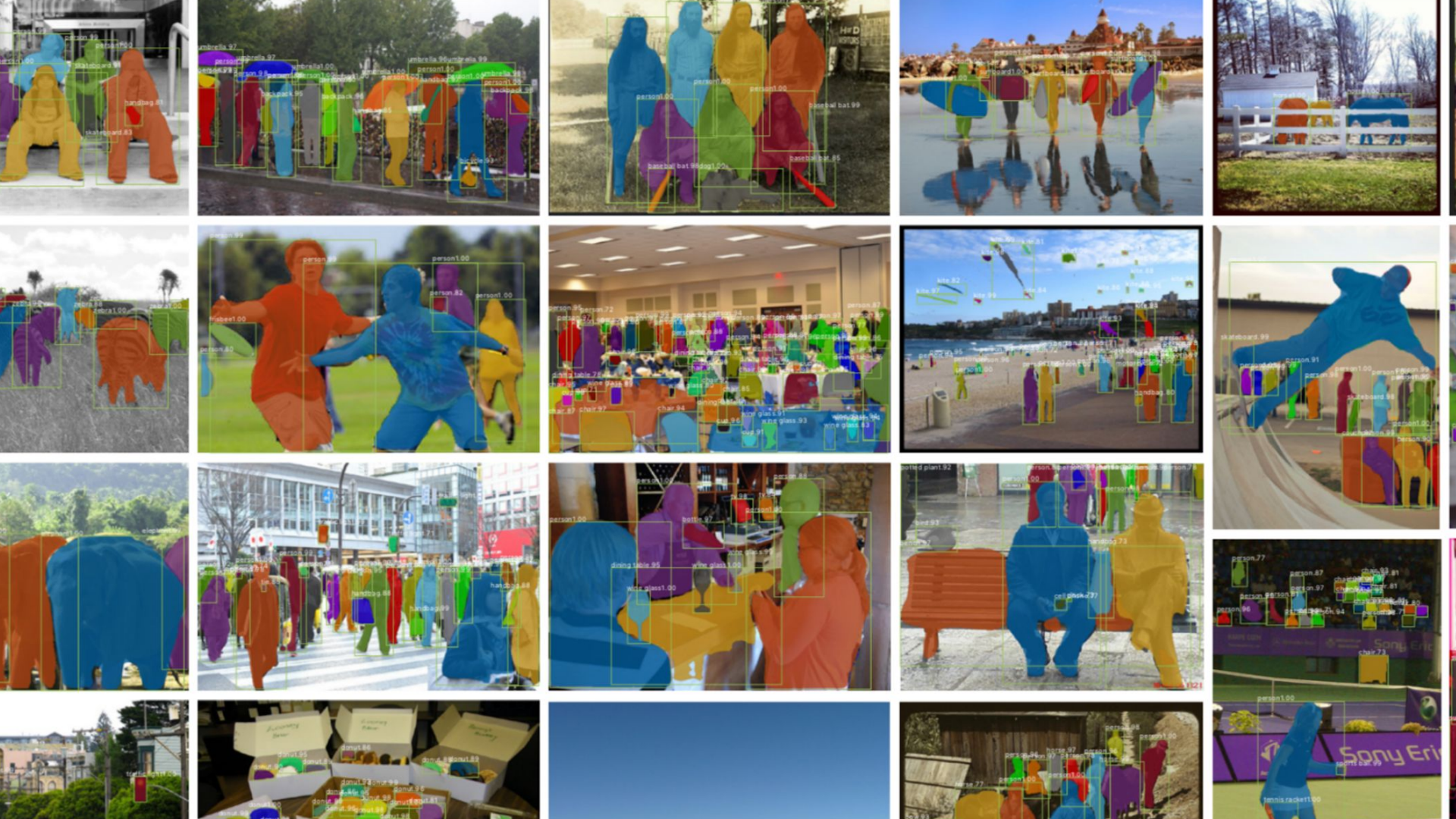


Figure source: Ross Girshick

MS COCO

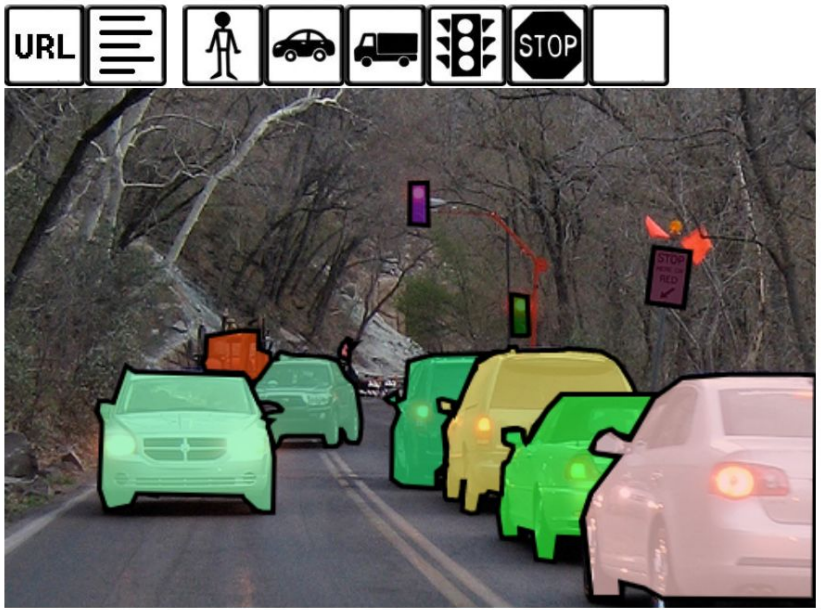
2015



>200K labeled images

80 object categories

250 000 person instances labeled with keypoints



COCO Keypoint Detection Task

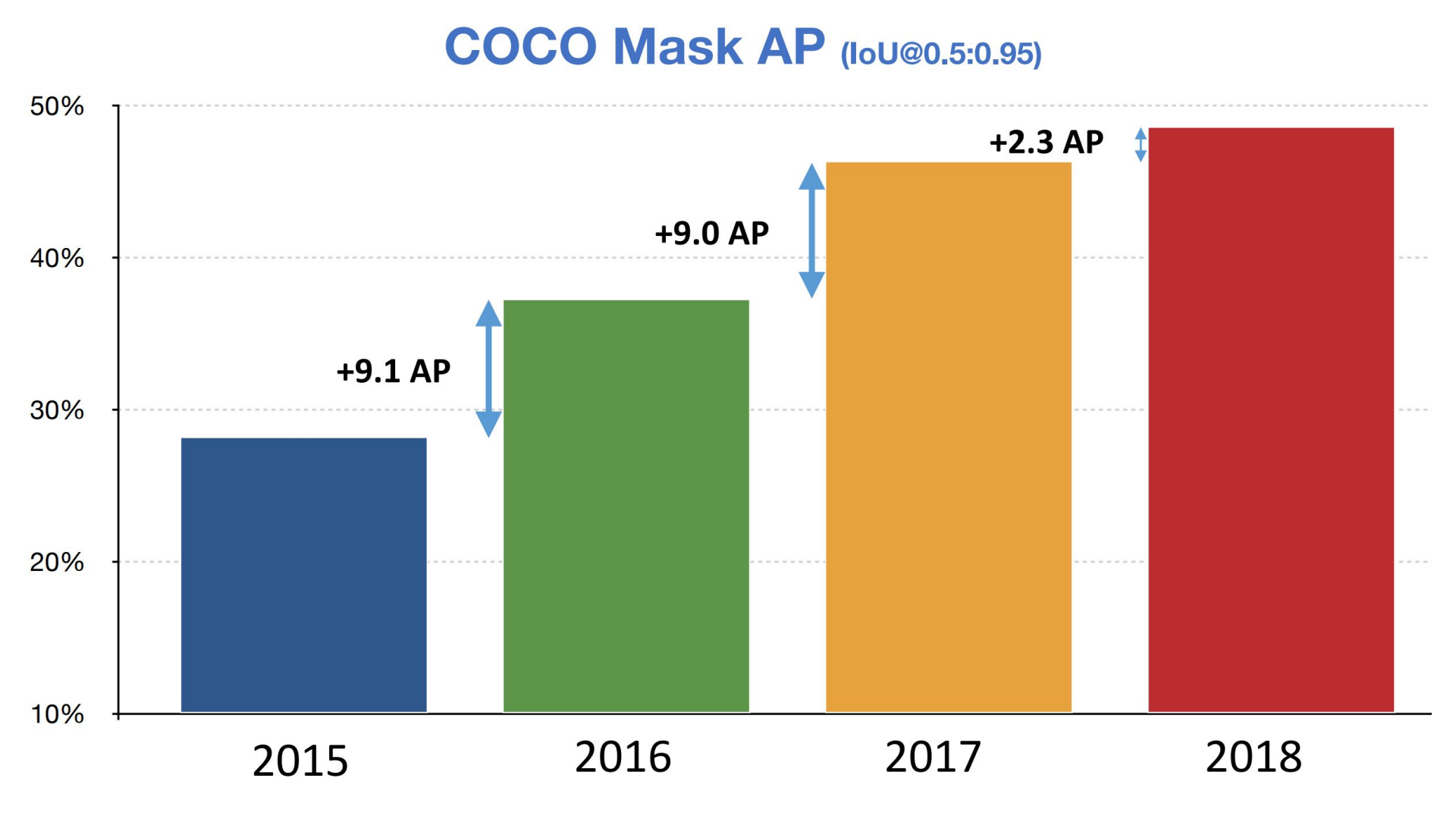
COCO Object Detection Task



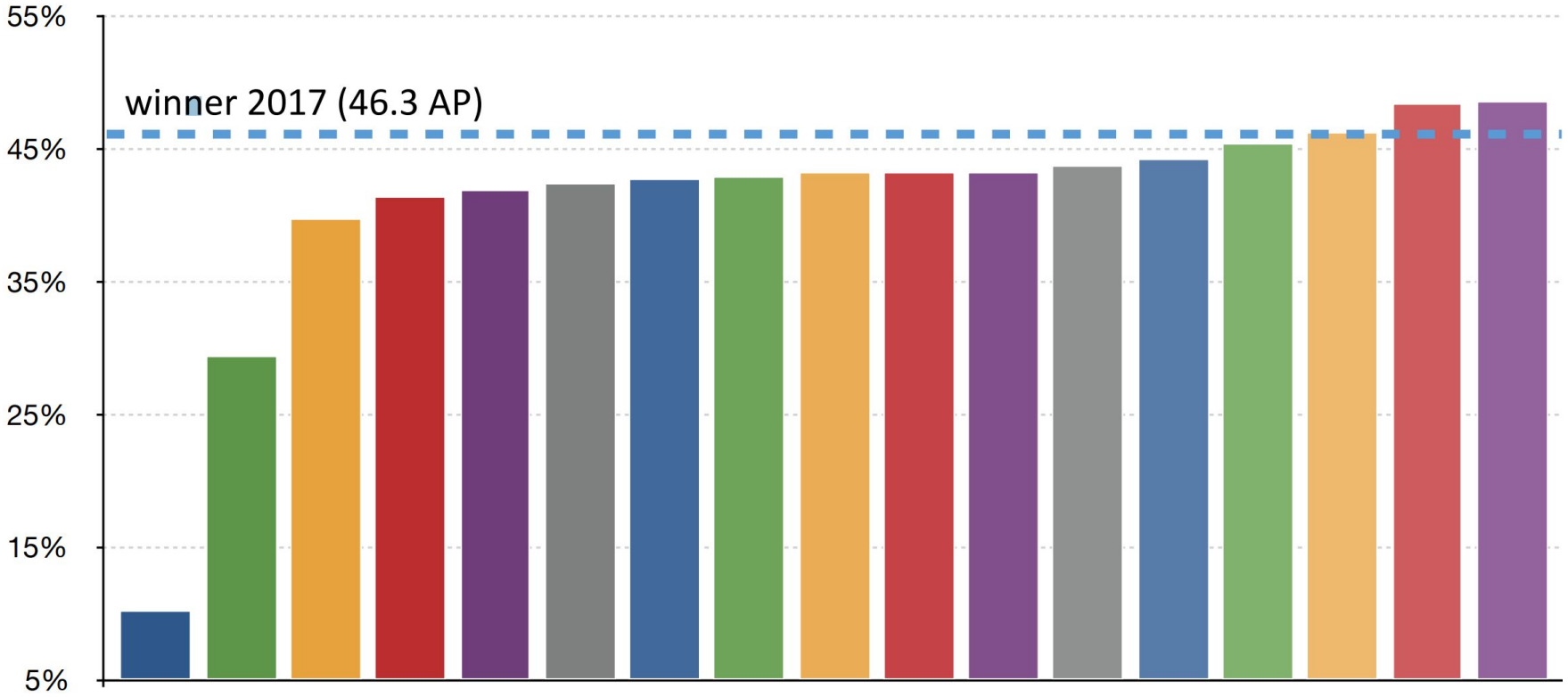
2020 - only the instance segmentation (no bboxes)

COCO Stuff Segmentation Task

COCO Panoptic Segmentation Task


COCO Mask AP (IoU@0.5:0.95)
50%
40%
30%
20%
10%
+9.1 AP
+9.0 AP
+2.3 AP
2015
2016
2017
2018

COCO AP (IoU@0.5:0.95)



17 teams joined the competition

3 teams achieved same or better performance than last year's winner

Mapillary Vistas

2017

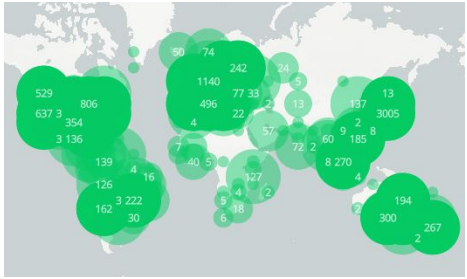
18k train / 2k val / 5k test

65 object classes

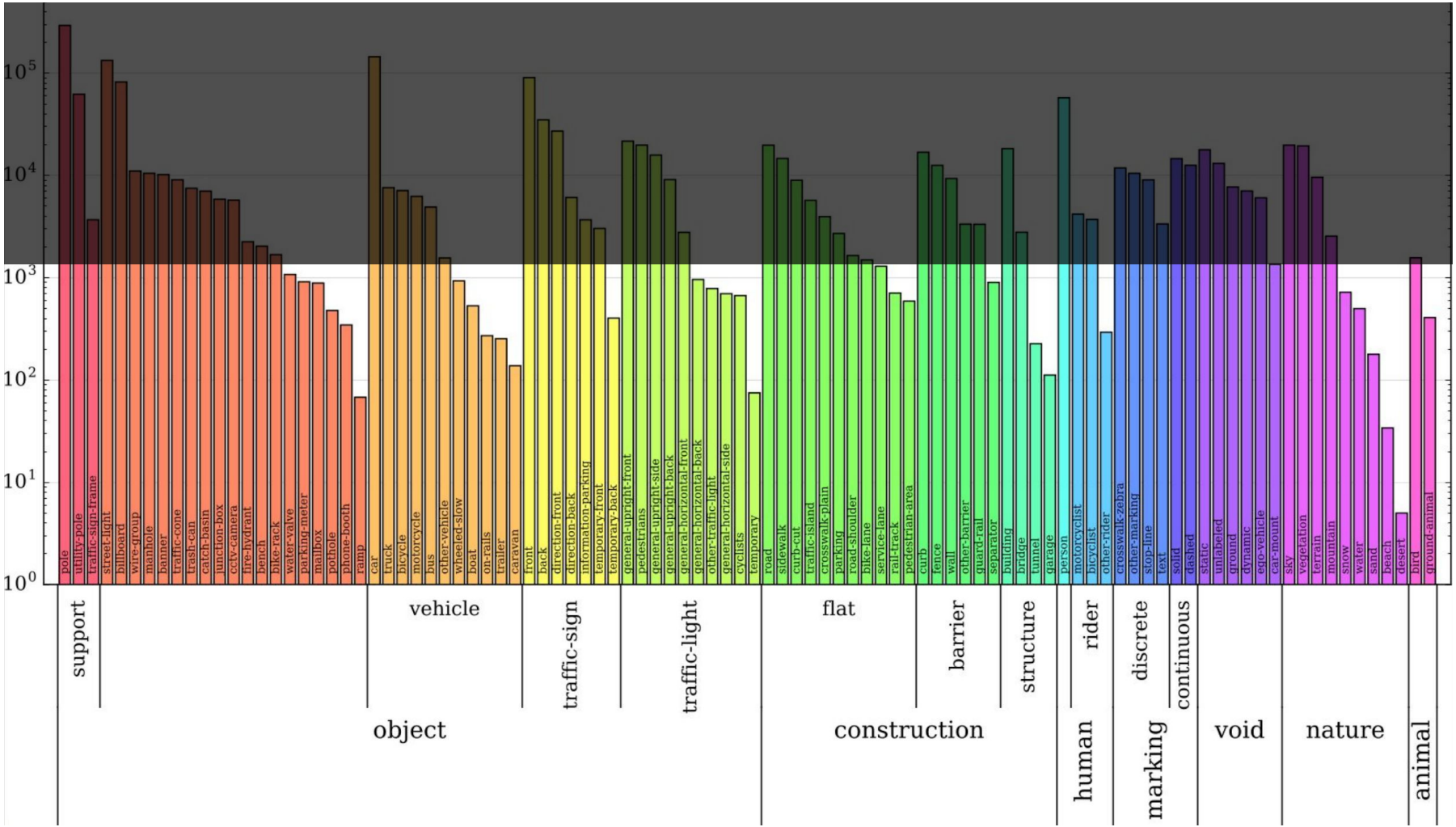
6 continents

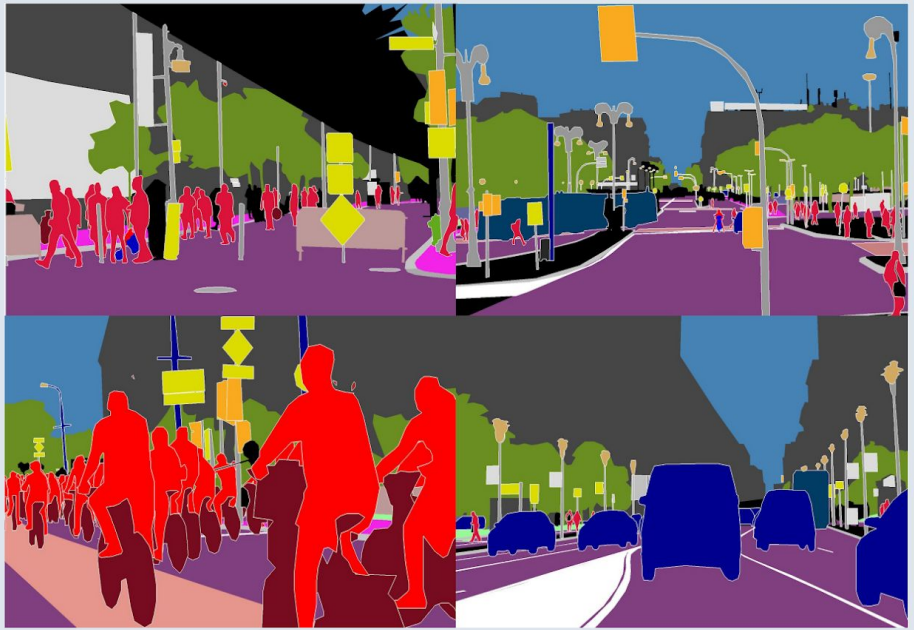
Roads, sidewalks, off-road

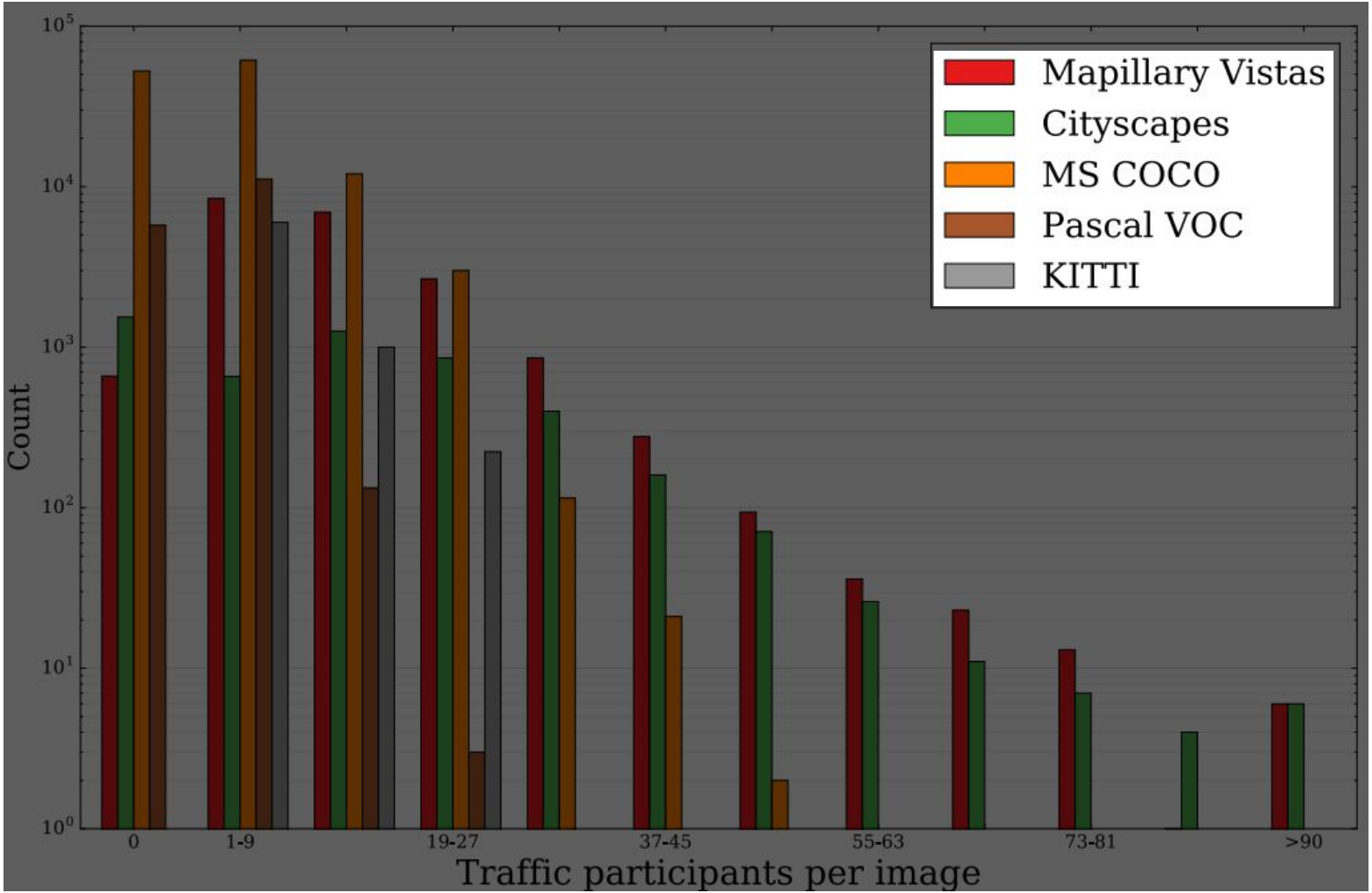
Variety of camera sensors

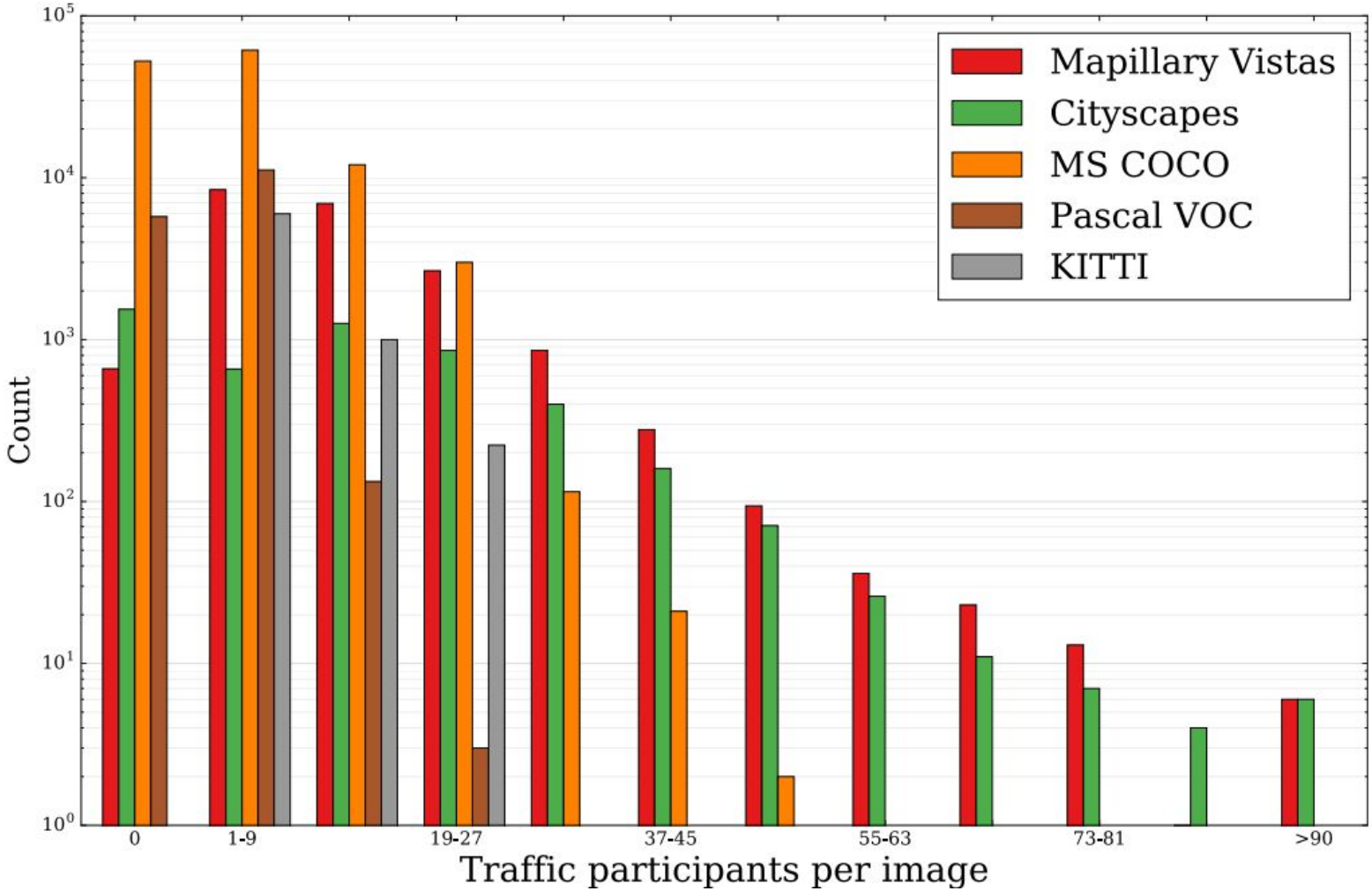


10^5
10^4
10^3
10^2
10^1
10^0
pole
utility-pole
traffic-sign-frame
street-light
billboard
wire-group
manhole
banner
traffic-cone
trash-can
catch-basin
junction-box
cctv-camera
fire-hydrant
bench
bike-rack
water-valve
parking-meter
mailbox
pothole
phone-booth
ramp
car
truck
bicycle
motorcycle
bus
other-vehicle
wheeled-slow
boat
on-rails
trailer
caravan
front
back
direction-front
direction-back
information-parking
temporary-front
temporary-back
general-upright-front
pedestrians
general-upright-side
general-upright-back
general-horizontal-front
general-horizontal-back
other-traffic-light
general-horizontal-side
cyclists
temporary
road
sidewalk
curb-cut
traffic-island
crosswalk-plain
parking
road-shoulder
bike-lane
service-lane
rail-track
pedestrian-area
curb
fence
wall
other-barrier
guard-rail
separator
building
bridge
tunnel
garage
person
motorcyclist
bicyclist
other-rider
crosswalk-zebra
other-marking
stop-line
text
solid
dashed
static
unlabeled
ground
dynamic
ego-vehicle
car-mount
sky
vegetation
terrain
mountain
snow
water
sand
beach
desert
bird
ground-animal
support
vehicle
traffic-sign
traffic-light
flat
barrier
structure
human
rider
discrete
continuous
void
nature
animal
object
construction
marking


10^5
10^4
10^3
10^2
10^1
10^0
pole
utility-pole
traffic-sign-frame
street-light
billboard
wire-group
manhole
banner
traffic-cone
trash-can
catch-basin
junction-box
cctv-camera
fire-hydrant
bench
bike-rack
water-valve
parking-meter
mailbox
pothole
phone-booth
ramp
car
truck
bicycle
motorcycle
bus
other-vehicle
wheeled-slow
boat
on-rails
trailer
caravan
front
back
direction-front
direction-back
information-parking
temporary-front
temporary-back
general-upright-front
pedestrians
general-upright-side
general-upright-back
general-horizontal-front
general-horizontal-back
other-traffic-light
general-horizontal-side
cyclists
temporary
road
sidewalk
curb-cut
traffic-island
crosswalk-plain
parking
road-shoulder
bike-lane
service-lane
rail-track
pedestrian-area
curb
fence
wall
other-barrier
guard-rail
separator
building
bridge
tunnel
garage
person
motorcyclist
bicyclist
other-rider
crosswalk-zebra
other-marking
stop-line
text
solid
dashed
static
unlabeled
ground
dynamic
ego-vehicle
car-mount
sky
vegetation
terrain
mountain
snow
water
sand
beach
desert
bird
ground-animal
support
vehicle
traffic-sign
traffic-light
flat
barrier
structure
human
rider
discrete
continuous
void
nature
animal
object
construction
marking



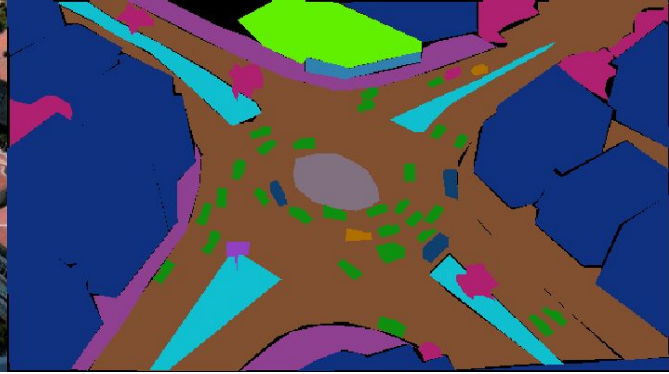

Mapillary Vistas
Cityscapes
MS COCO
Pascal VOC
KITTI
Count
10^5
10^4
10^3
10^2
10^1
10^0
0
1-9
19-27
37-45
55-63
73-81
>90
Traffic participants per image

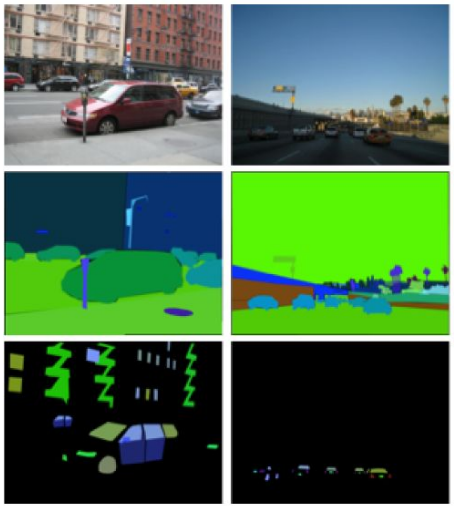

Mapillary Vistas
Cityscapes
MS COCO
Pascal VOC
KITTI
Count
10^5
10^4
10^3
10^2
10^1
10^0
0
1-9
19-27
37-45
55-63
73-81
>90
Traffic participants per image

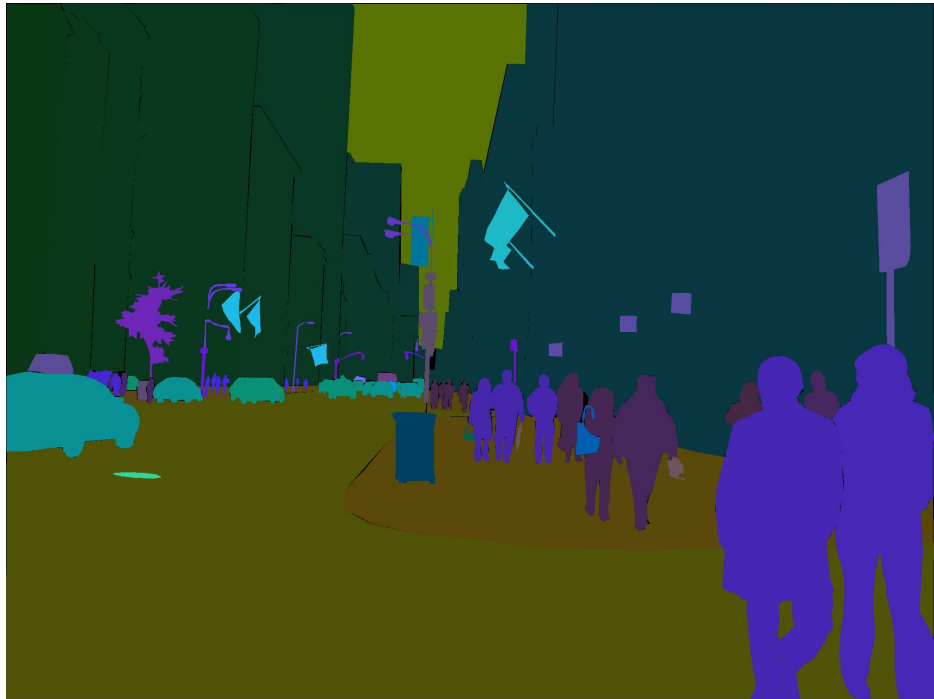
ADE20K

2017

http://groups.csail.mit.edu/vision/datasets/ADE20K/







ADE20K

20k/2k/3k images in train/val/test sets

270 classes with >100 instances

68 classes with >1000 instances

Zhou, Bolei, et al. "Scene parsing through ade20k dataset."
Proceedings of the IEEE conference on computer vision and pattern recognition. 2017.

Open Images v6


storage.googleapis.com/openimages/web/visualizer/index.html?set=train&type=segmentation&r=false&c=%2Fm%2F07cmd
Open Images Dataset V6
Explore
Description
Download
Extended
Subset
Type: Segmentation
Category:
Tank
Random category
Options
For clarity, we show the masks of the current category only.
We display the top 500 images ranked by estimated quality.
This image is no longer available in Flickr

storage.googleapis.com/openimages/web/visualizer/index.html?set=train&type=segmentation&r=false&c=%2Fm%2F0gxl3

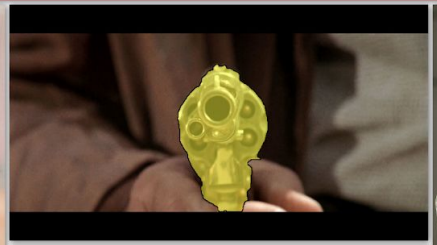
Open Images Dataset V6

Explore | Description | Do

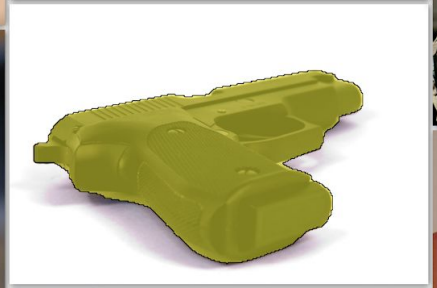
Subset | Type: Segmentation | Category: Handgun | Random category | Options

For clarity, we show the masks of the current category only.
We display the top 500 images ranked by estimated quality.













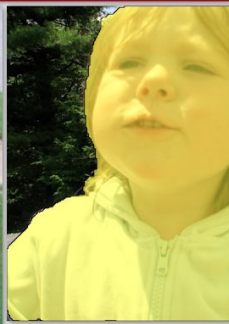


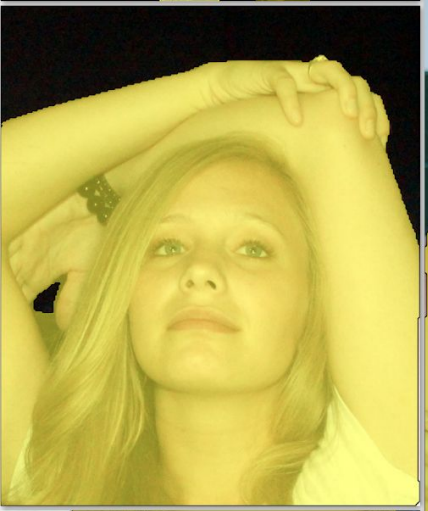
Open Images Dataset V6

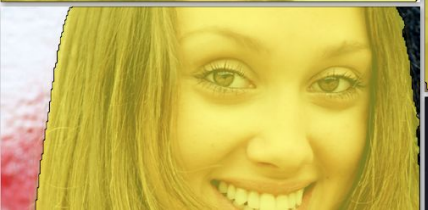
Explore | Description | Do

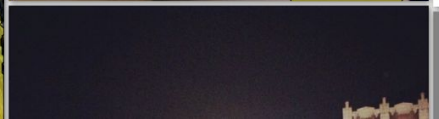
Subset | Type: Segmentation | Category: Girl | Random category | Options

For clarity we show the masks of the current category only.
We display the top 500 images ranked by estimated quality.










Building (Group)
Billboard
Billboard
LG
Billboard
Billboard
Billboard
Billboard
Human face
Human face
Human face
Billboard
Billboard
Billboard
swatch
Billboard
Billboard
Human face
Foot Lock
maxell
Man
Man
Human face
Human face
Human face
Car
Car
Car
Car
Car
Car
Wheel

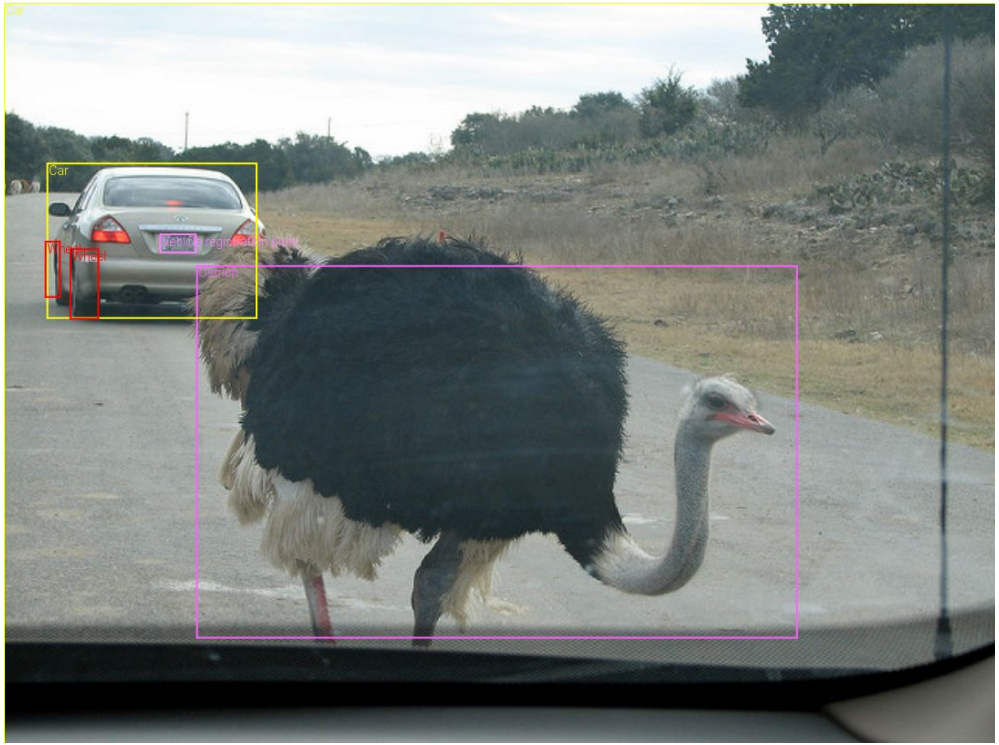
Open Images v6

1.7M images

(with bounding boxes)

600 categories

(for bounding boxes)



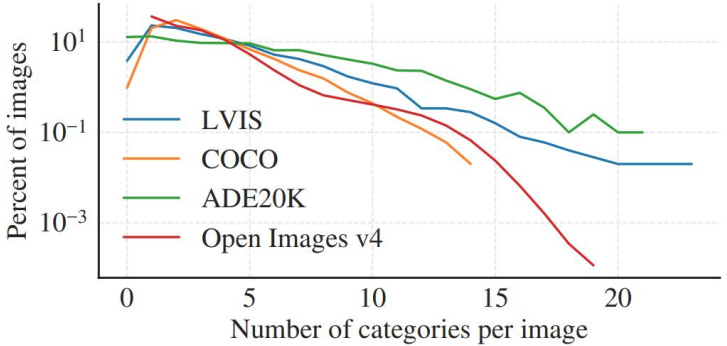
LVIS
2019

LVIS: A DATASET FOR LARGE VOCABULARY INSTANCE SEGMENTATION

A new dataset for long tail object recognition.

PAPER

Figure 1. **Example annotations.** We present **LVIS**, a new dataset for benchmarking Large Vocabulary Instance Segmentation in the 1000+ category regime with a challenging long tail of rare objects.



(a) Distribution of category count per image. LVIS has a heavier tail than COCO and Open Images training set. ADE20K is the most uniform.

LVIS Dataset

v1.0

Training set

- 1,270,141 instances (1 GB)
- 100,170 images (18 GB)

Validation set

- 244,707 instances (192 MB)
- 19,809 images (1 GB)

Test Dev

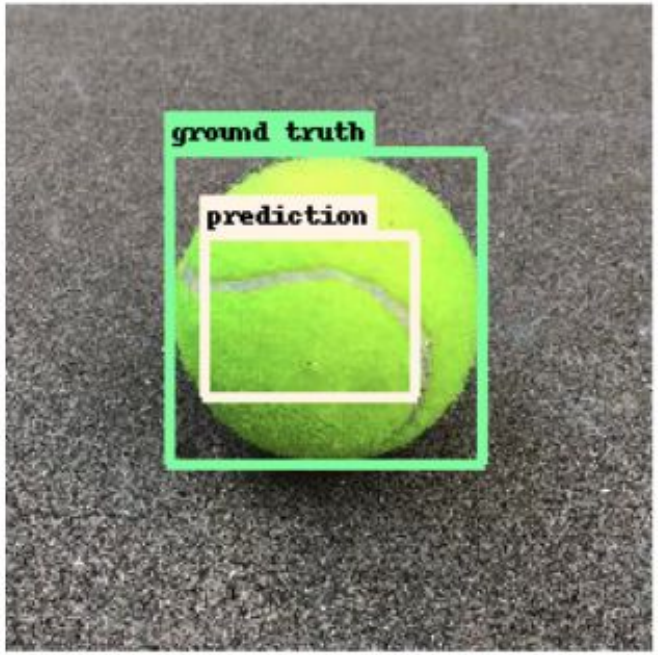
- info (4 MB)
- 19,822 images (6 GB)

Test Challenge

- info (4 MB)
- 19,822 images (6 GB)

Metrics

Metryka?



? = < >

$$IOU(a,b) = \frac{Area(a) \bigcap Area(b)}{Area(a) \bigcup Area(b)}$$

IoU: 35%

IoU: 55%

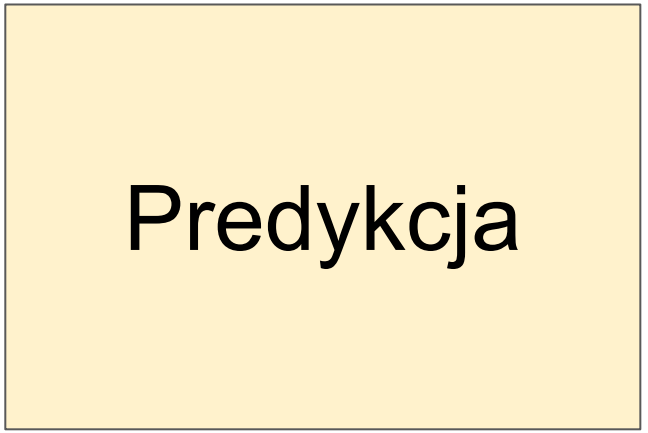
IoU: 75%

IoU: 95%

$$IOU(a,b) = \frac{Area(a) \bigcap Area(b)}{Area(a) \bigcup Area(b)}$$

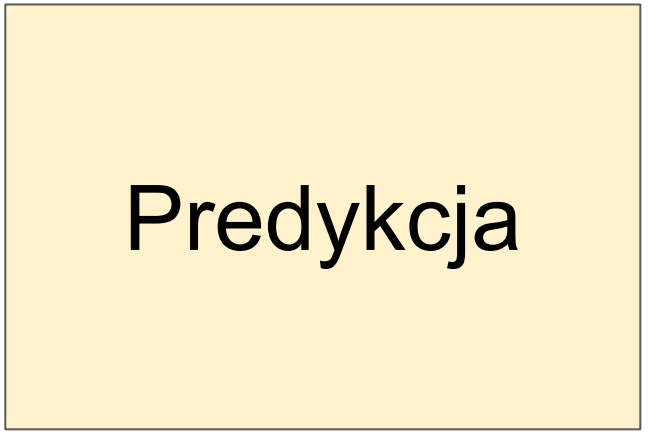
https://apple.github.io/turicreate/docs/userguide/object_detection/advanced-usage.html

$$IoU = \frac{|A \cap B|}{|A \cup B|}$$



IoU=0

$$IoU = \frac{|A \cap B|}{|A \cup B|}$$



Czy takie etykiety (zaznaczone przez ludzi) powinniśmy nazywać Ground Truth? #BIAS

IoU=0

$$IoU = \frac{|A \cap B|}{|A \cup B|}$$

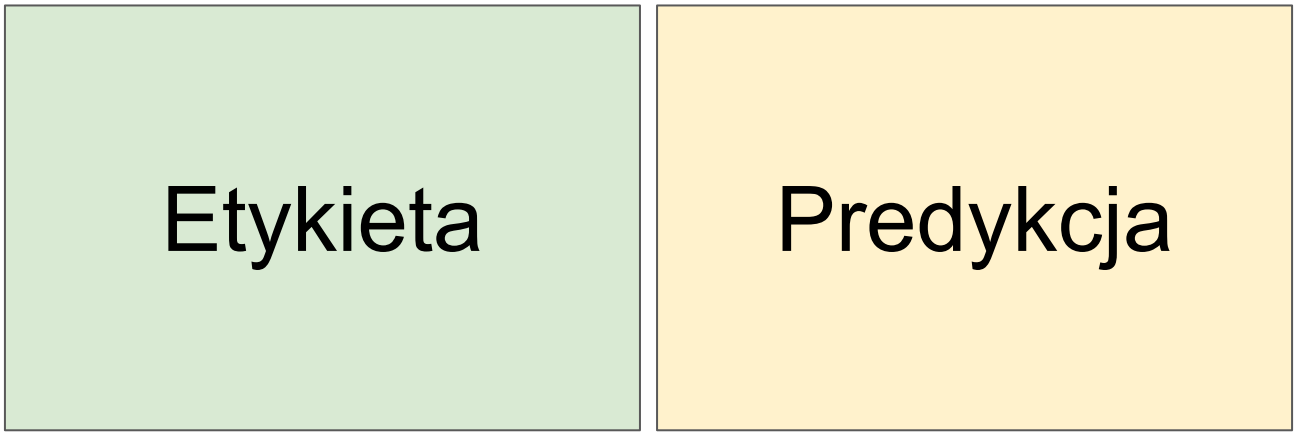
Predykcja

Etykieta

Czy takie etykiety (zaznaczone przez ludzi) powinniśmy nazywać Ground Truth? #BIAS

IoU=0

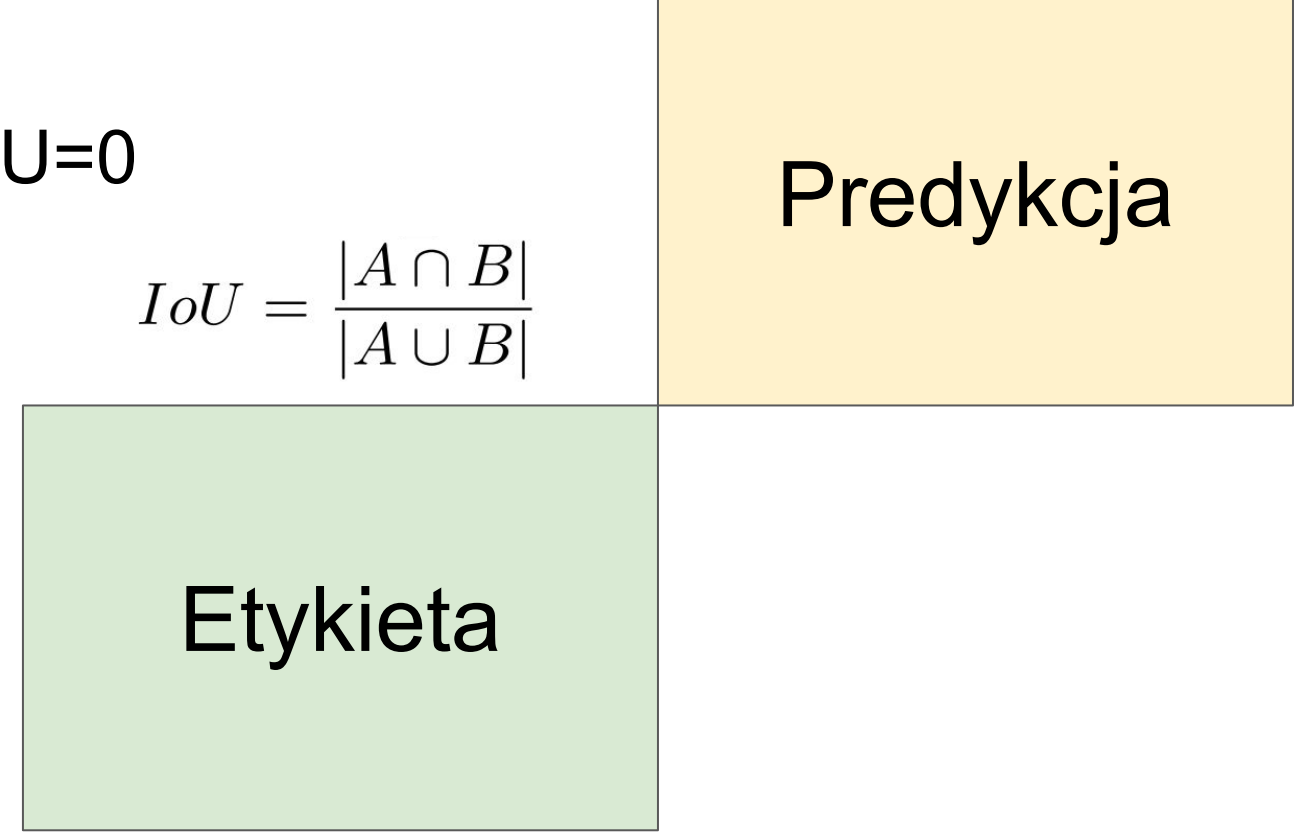
$$IoU = \frac{|A \cap B|}{|A \cup B|}$$



Czy takie etykiety (zaznaczone przez ludzi) powinniśmy nazywać Ground Truth? #BIAS

IoU=0

$$IoU = \frac{|A \cap B|}{|A \cup B|}$$



Czy takie etykiety (zaznaczone przez ludzi) powinniśmy nazywać Ground Truth? #BIAS

IoU=0

$$IoU = \frac{|A \cap B|}{|A \cup B|}$$

Czy takie etykiety (zaznaczone przez ludzi) powinniśmy nazywać Ground Truth? #BIAS

GIoU - Generalized IoU

Algorithm 1: Generalized Intersection over Union

input : Two arbitrary convex shapes: $A, B \subseteq \mathbb{S} \in \mathbb{R}^n$

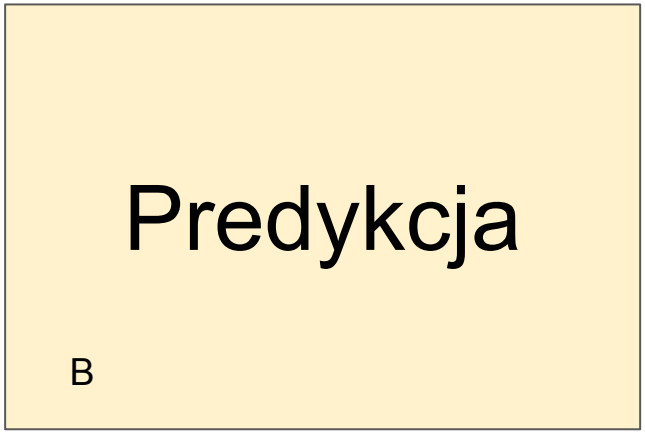
output: $GIoU$

1 For A and B, find the smallest enclosing convex object C, where $C \subseteq \mathbb{S} \in \mathbb{R}^n$

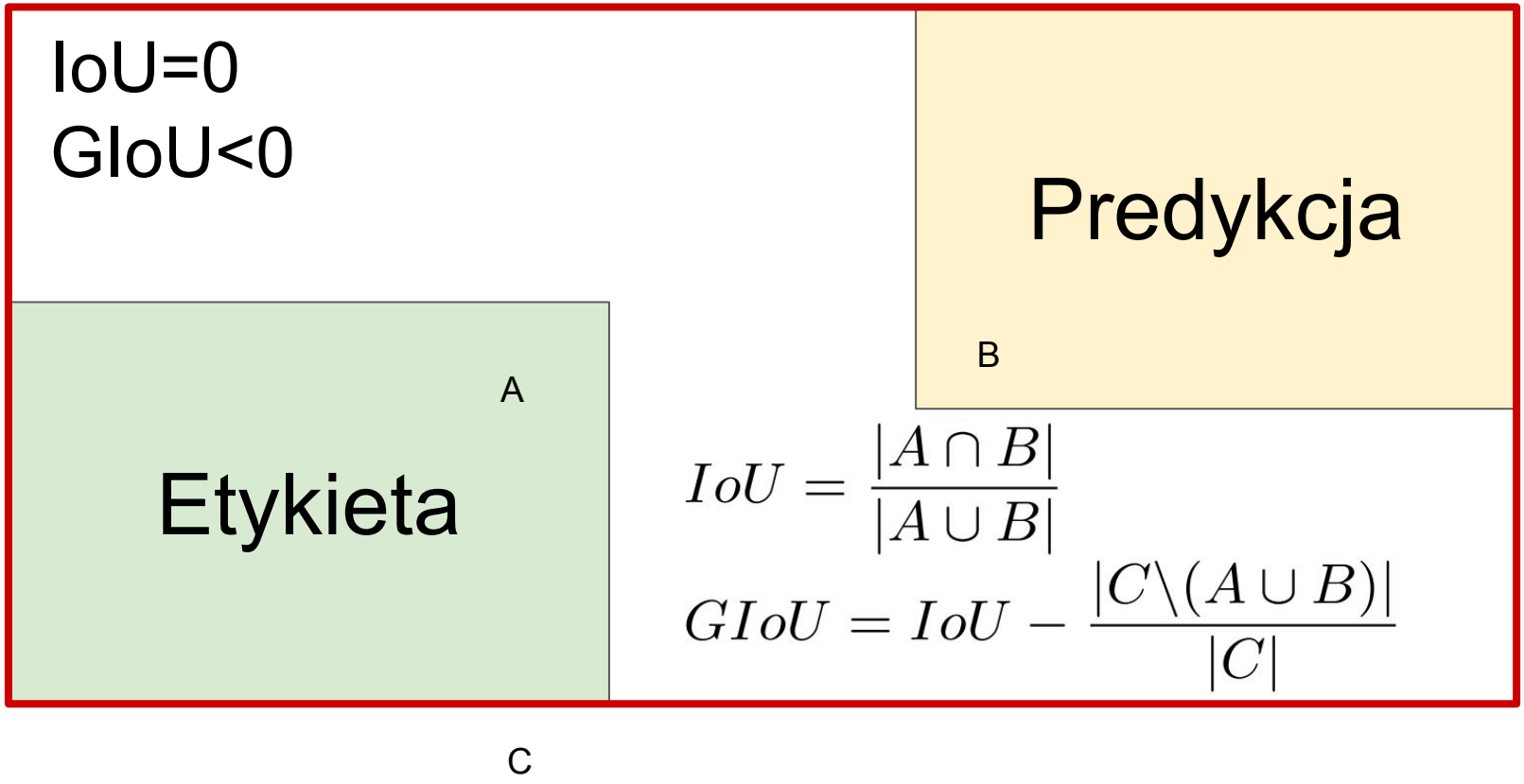
2 $$IoU = \frac{|A \cap B|}{|A \cup B|}$$

3 $$GIoU = IoU - \frac{|C \backslash (A \cup B)|}{|C|}$$

IoU=0



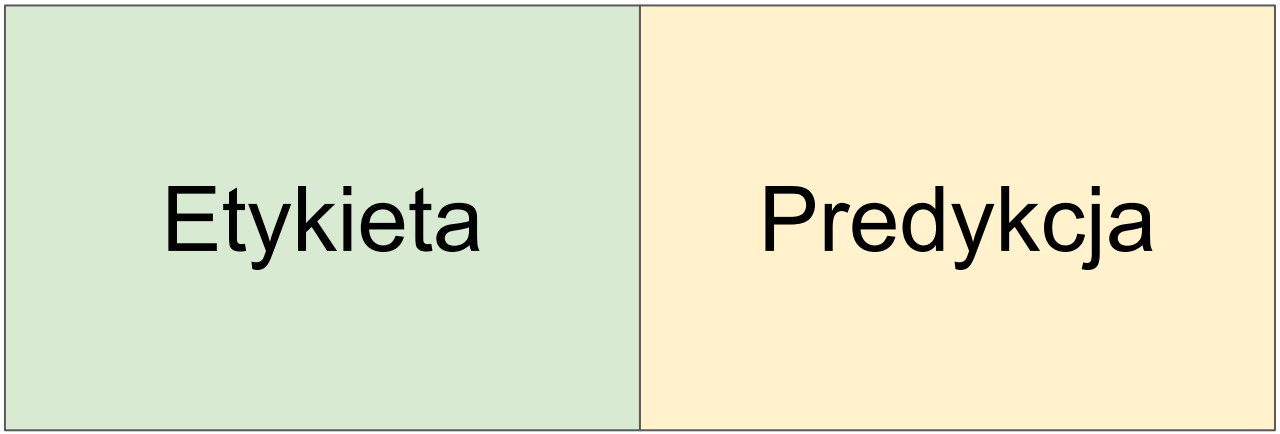
$$IoU = \frac{|A \cap B|}{|A \cup B|}$$

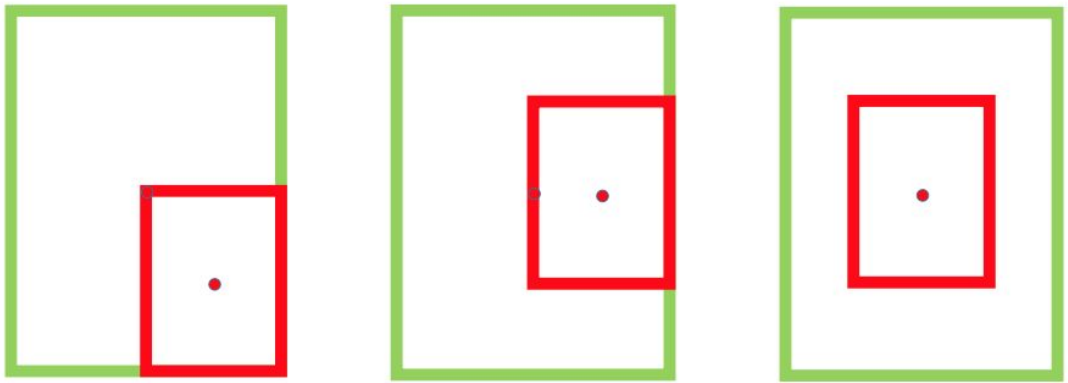


$$IoU = \frac{|A \cap B|}{|A \cup B|}$$

$$GIoU = IoU - \frac{|C \backslash (A \cup B)|}{|C|}$$

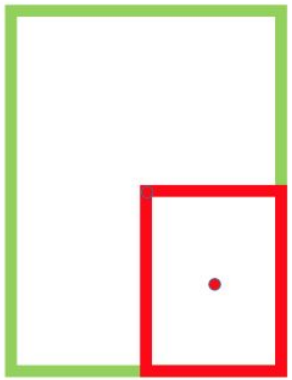
IoU=0
GIoU=0



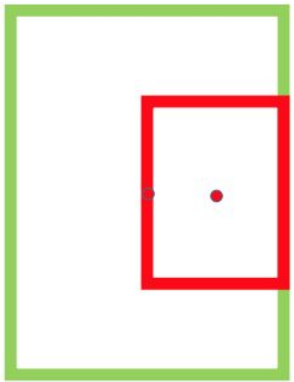


Green and red denote target box and predicted box respectively.

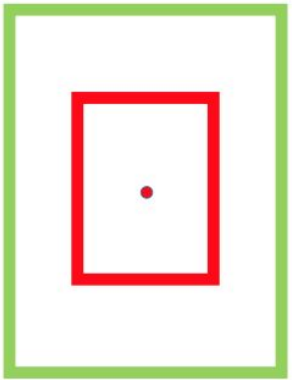
Zheng, Zhaohui, et al. "Distance-IoU Loss: Faster and Better Learning for Bounding Box Regression." *AAAI*. 2020.



$\mathcal{L}_{IoU} = 0.75$
$\mathcal{L}_{GIoU} = 0.75$



$\mathcal{L}_{IoU} = 0.75$
$\mathcal{L}_{GIoU} = 0.75$

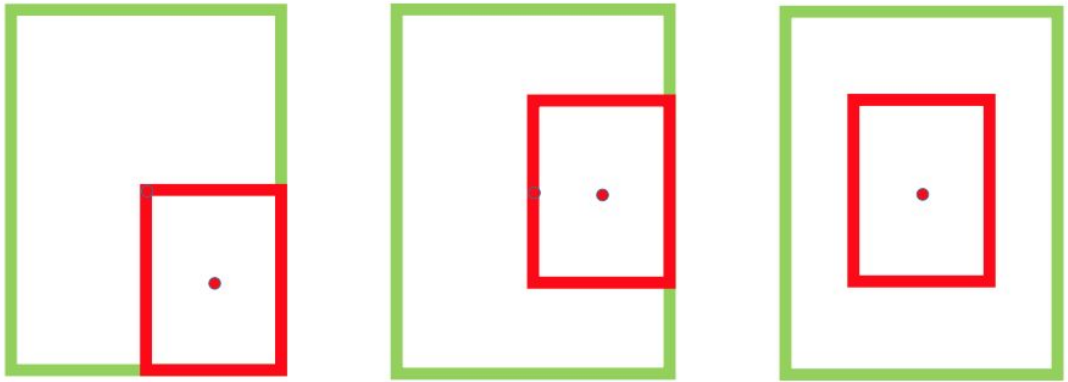


$\mathcal{L}_{IoU} = 0.75$
$\mathcal{L}_{GIoU} = 0.75$

Green and red denote target box and predicted box respectively.

Zheng, Zhaohui, et al. "Distance-IoU Loss: Faster and Better Learning for Bounding Box Regression." *AAAI*. 2020.

Distance IoU



$\mathcal{L}_{IoU} = 0.75$ $\mathcal{L}_{IoU} = 0.75$ $\mathcal{L}_{IoU} = 0.75$

$\mathcal{L}_{GIoU} = 0.75$ $\mathcal{L}_{GIoU} = 0.75$ $\mathcal{L}_{GIoU} = 0.75$

$\mathcal{L}_{DIoU} = 0.81$ $\mathcal{L}_{DIoU} = 0.77$ $\mathcal{L}_{DIoU} = 0.75$

Figure 2: GIoU loss degrades to IoU loss for these cases, while our DIoU loss is still distinguishable. Green and red denote target box and predicted box respectively.

Zheng, Zhaohui, et al. "Distance-IoU Loss: Faster and Better Learning for Bounding Box Regression." *AAAI*. 2020.

Distance IoU

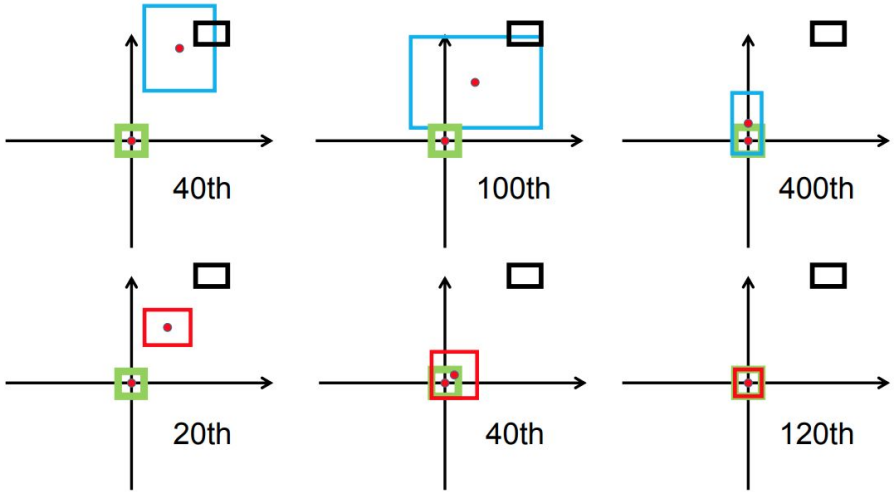


Figure 1: Bounding box regression steps by GIoU loss (first row) and DIoU loss (second row). Green and **black** denote target box and **anchor** box, respectively. Blue and red denote predicted boxes for GIoU loss and DIoU loss, respectively. GIoU loss generally increases the size of predicted box to overlap with target box, while DIoU loss directly minimizes normalized distance of central points.

Zheng, Zhaohui, et al. "Distance-IoU Loss: Faster and Better Learning for Bounding Box Regression." *AAAI*. 2020.

Distance IoU

$$\mathcal{R}_{DIoU} = \frac{\rho^2(\mathbf{b}, \mathbf{b}^{gt})}{c^2}, \tag{6}$$

where $\mathbf{b}$ and $\mathbf{b}^{gt}$ denote the central points of B and B^{gt}, $\rho(\cdot)$ is the Euclidean distance, and c is the diagonal length of the smallest enclosing box covering the two boxes. And then the DIoU loss function can be defined as

$$\mathcal{L}_{DIoU} = 1 - IoU + \frac{\rho^2(\mathbf{b}, \mathbf{b}^{gt})}{c^2}. \tag{7}$$

As shown in Fig. 5, the penalty term of DIoU loss directly minimizes the distance between two central points, while GIoU loss aims to reduce the area of $C - B \cup B^{gt}$.

Zheng, Zhaohui, et al. "Distance-IoU Loss: Faster and Better Learning for Bounding Box Regression." *AAAI*. 2020.

CIoU: Complete IoU

- overlap area
- central point distance
- **aspect ratio**

$$v = \frac{4}{\pi^2}(arctan\frac{w^{gt}}{h^{gt}} - arctan\frac{w}{h})^2.$$

Then the loss function can be defined as

$$\mathcal{L}_{CIoU} = 1 - IoU + \frac{\rho^2(\mathbf{b}, \mathbf{b}^{gt})}{c^2} + \alpha v$$

And the trade-off parameter α is defined as

$$\alpha = \frac{v}{(1 - IoU) + v},$$

Zheng, Zhaohui, et al. "Distance-IoU Loss: Faster and Better Learning for Bounding Box Regression." *AAAI*. 2020.

CIoU: Complete IoU

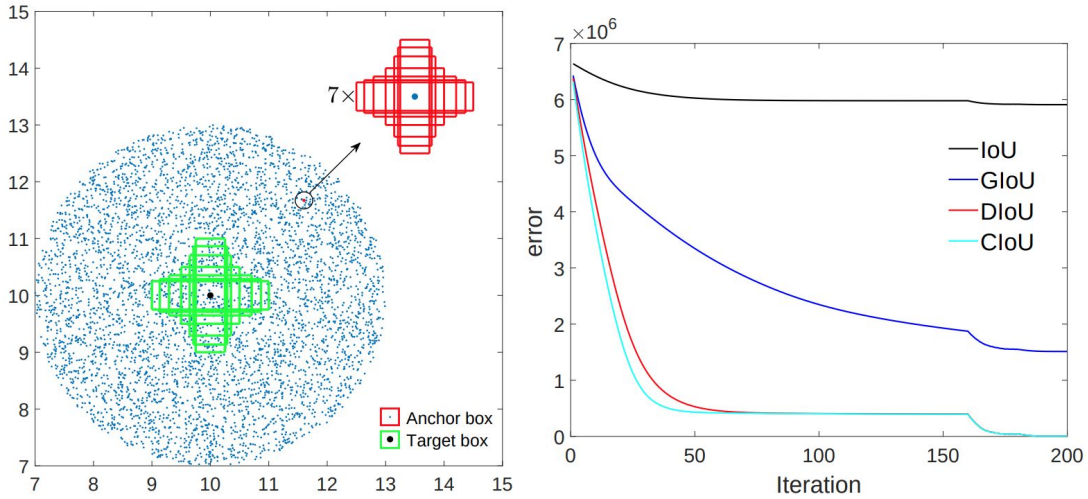
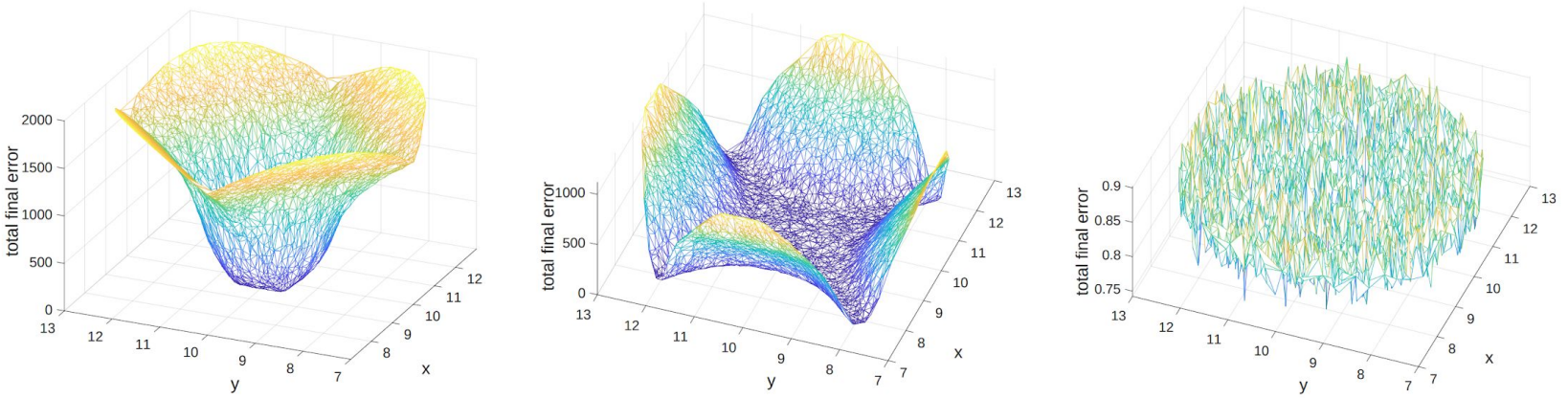


Figure 3: Simulation experiments: (a) 1,715,000 regression cases are adopted by considering different distances, scales and aspect ratios, (b) regression error sum (i.e., $\sum_n \mathbf{E}(t, n)$) curves of different loss functions at iteration t.

Zheng, Zhaohui, et al. "Distance-IoU Loss: Faster and Better Learning for Bounding Box Regression." *AAAI*. 2020.

CIoU: Complete IoU



(a) $\mathcal{L}_{IoU}$ (b) $\mathcal{L}_{GIoU}$ (c) $\mathcal{L}_{DIoU}$

Figure 4: Visualization of regression errors of IoU, GIoU and DIoU losses at the final iteration T, i.e.. $\mathbf{E}(T, n)$ for every coordinate n. We note that the basins in (a) and (b) correspond to good regression cases. One can see that IoU loss has large errors for non-overlapping cases, GIoU loss has large errors for horizontal and vertical cases, and our DIoU loss leads to very small regression errors everywhere.

Zheng, Zhaohui, et al. "Distance-IoU Loss: Faster and Better Learning for Bounding Box Regression." *AAAI*. 2020.

Historia i 2020

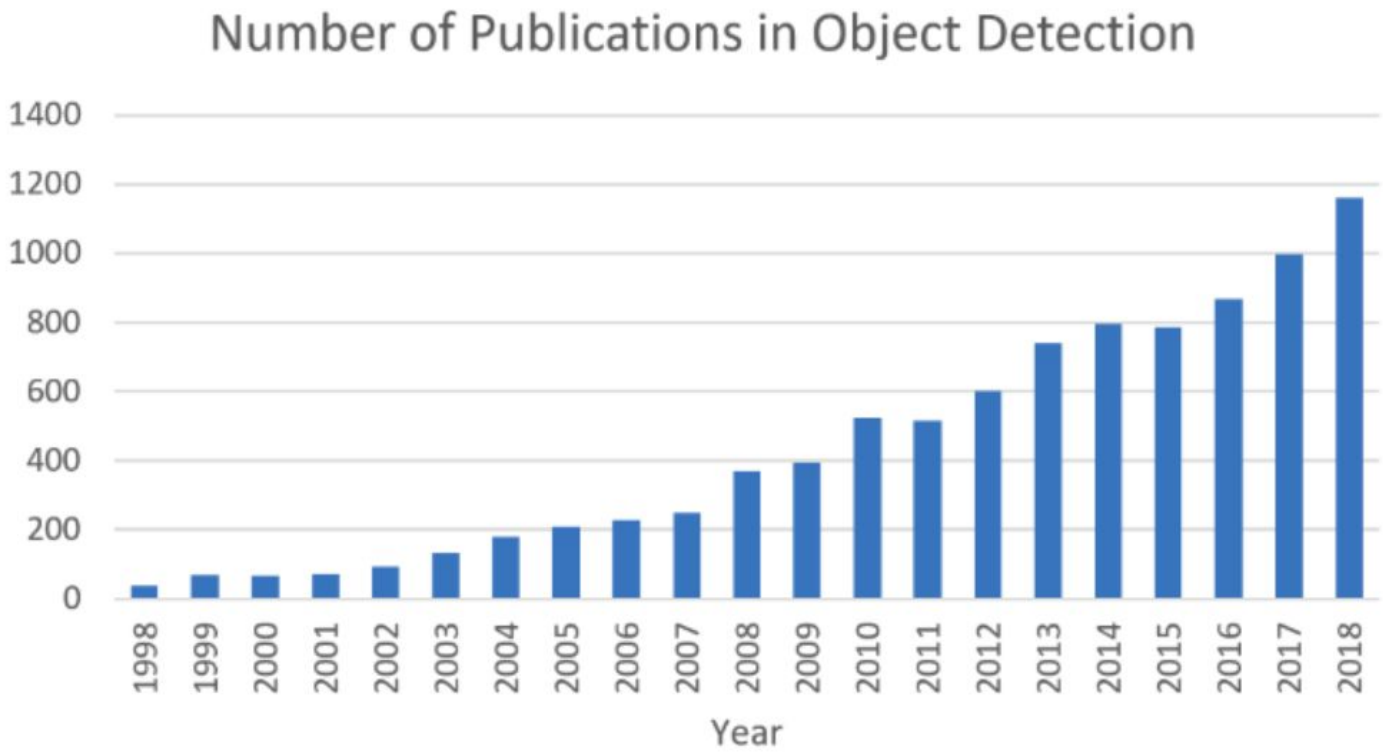


Fig. 1. The increasing number of publications in object detection from 1998 to 2018. (Data from Google scholar advanced search: *allintitle: "object detection" AND "detecting objects"*.)

Object Detection in 20 Years: A Survey

Zou, Z., Shi, Z., Guo, Y., & Ye, J. (2019). Object Detection in 20 Years: A Survey. arXiv preprint arXiv:1905.05055.

arxiv.org/abs/1905.05055

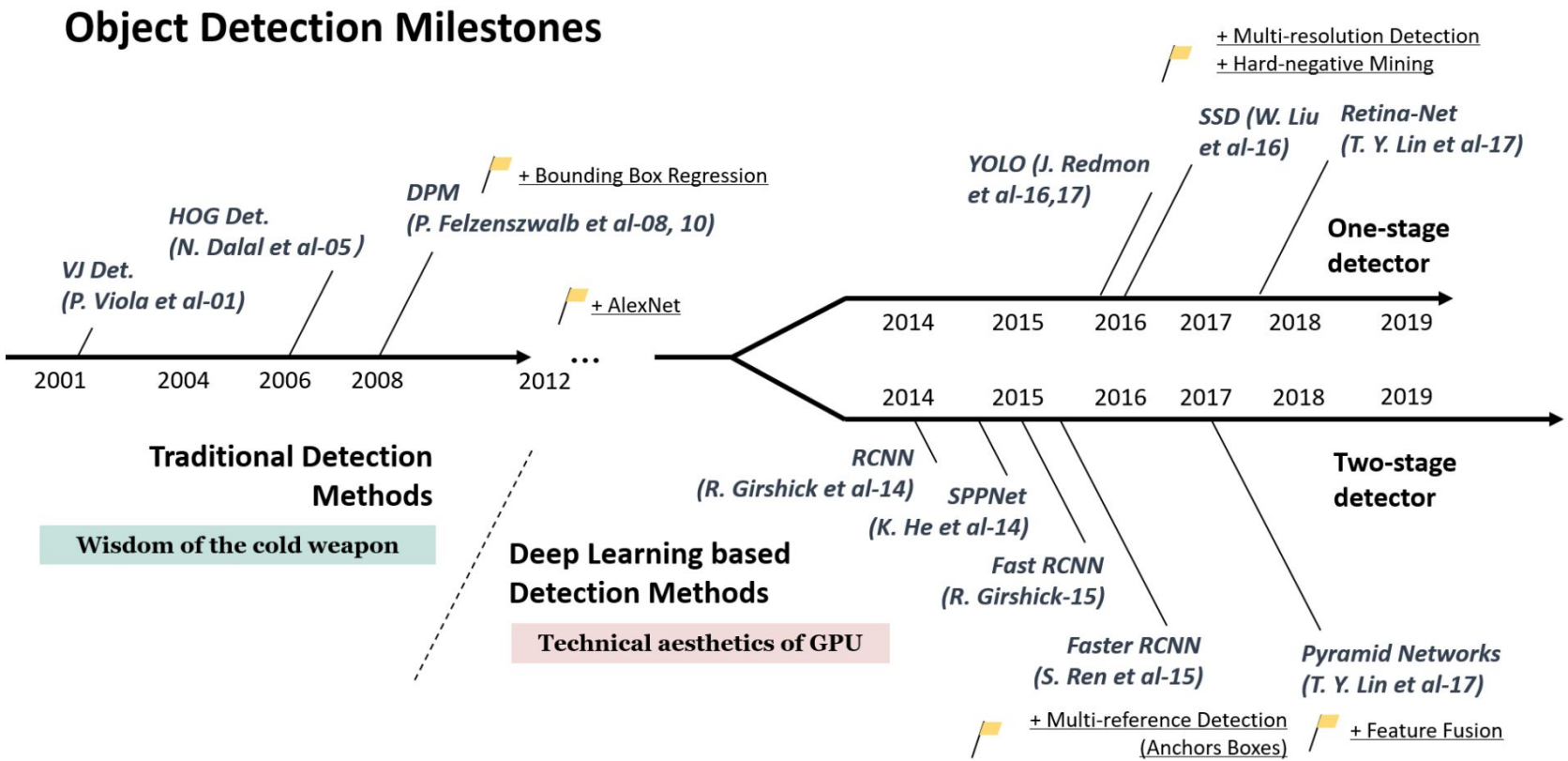


Fig. 2. A road map of object detection. Milestone detectors in this figure: VJ Det. [10, 11], HOG Det. [12], DPM [13–15], RCNN [16], SPPNet [17], Fast RCNN [18], Faster RCNN [19], YOLO [20], SSD [21], Pyramid Networks [22], Retina-Net [23].

Zou, Zhengxia, et al. "Object detection in 20 years: A survey." *arXiv preprint arXiv:1905.05055* (2019).

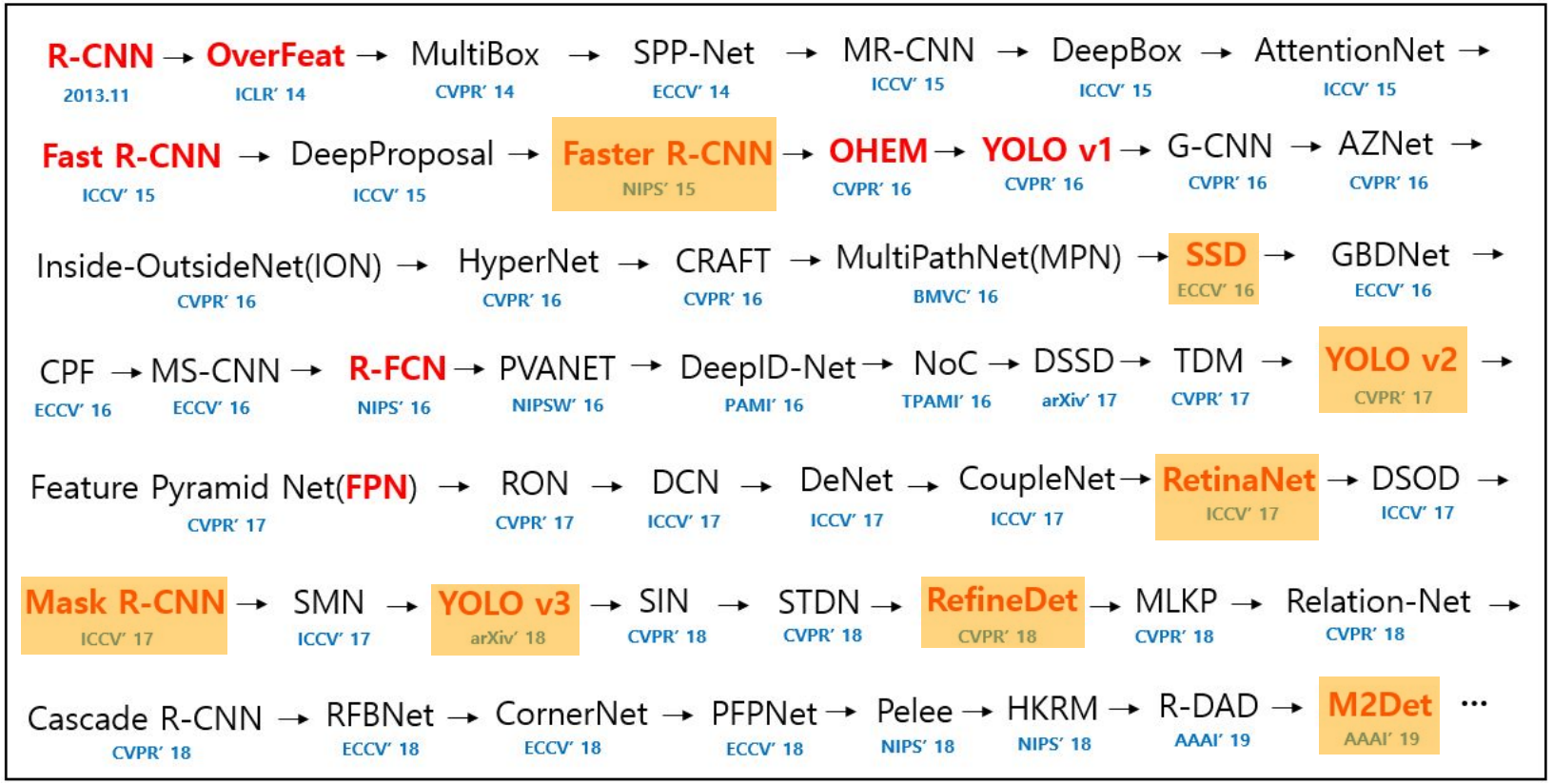
Object detection architectures

Two stage:

- 2014 R-CNN
- 2015 Fast R-CNN
- 2015 Faster R-CNN
- 2016 R-FCN

Single stage:

- 2016 YOLO
- 2016 SSD
- 2017 YOLOv2
- 2017 SSD MobileNet v1
- 2017 RetinaNet
- 2018 SSD MobileNet v2
- 2018 YOLOv3
- 2018 CornerNet
- 2019 FCOS



youtube.com/karolmajek

Selective Search for Object Recognition (2012)

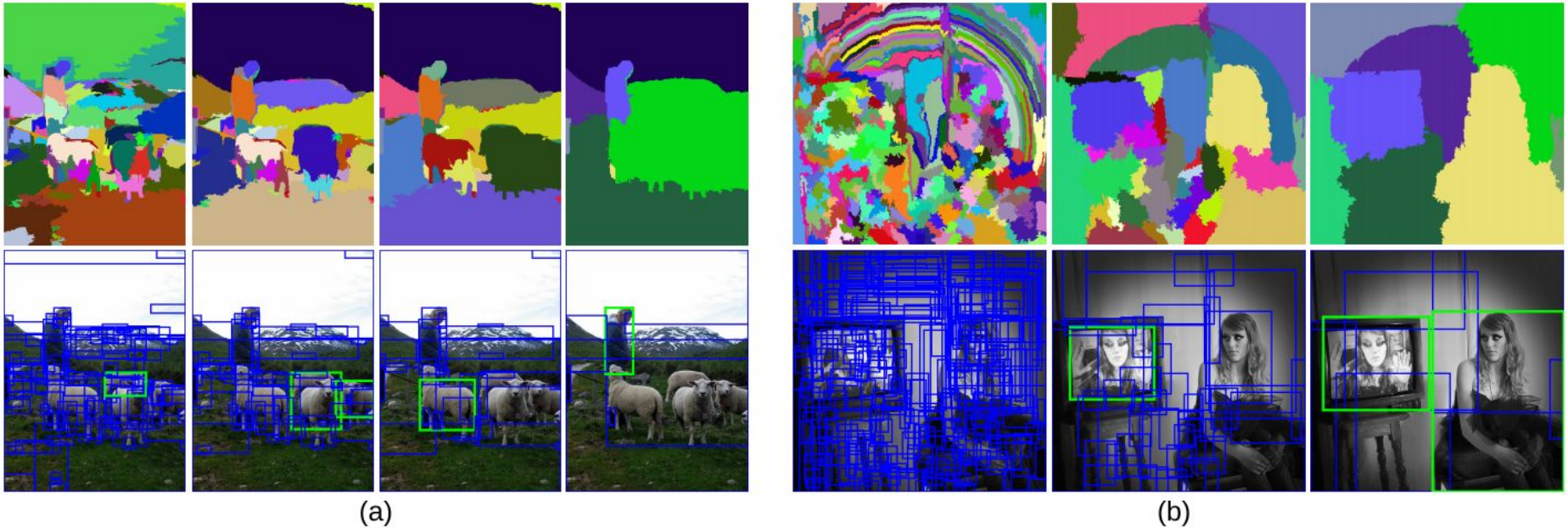
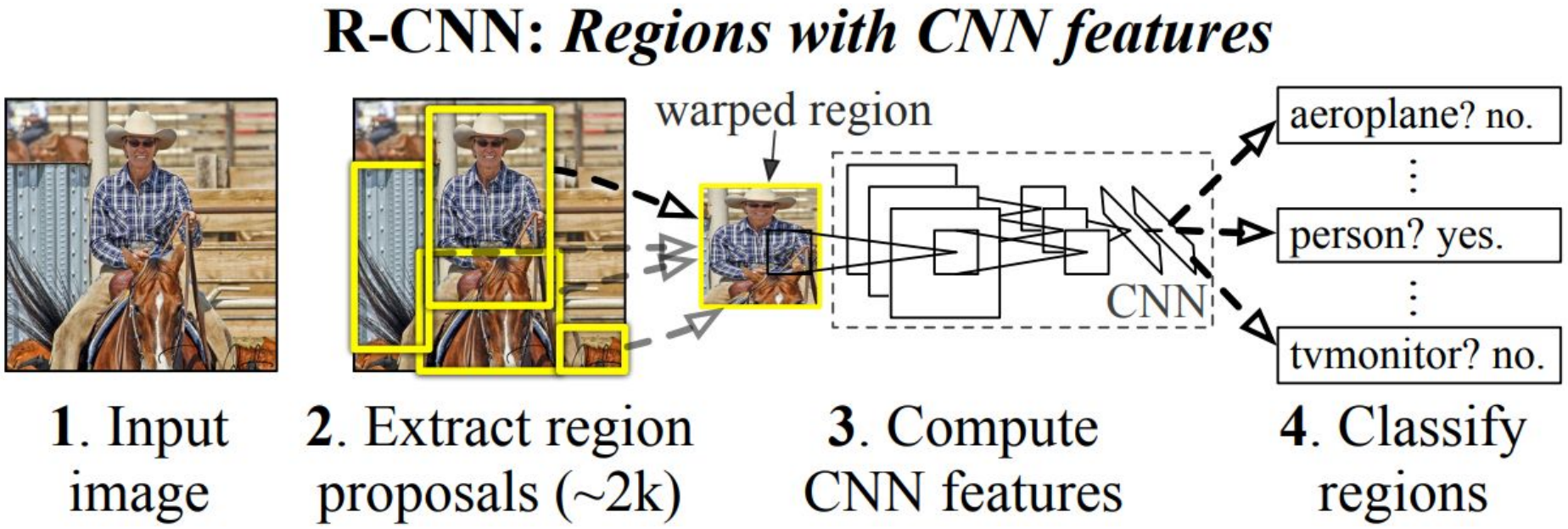
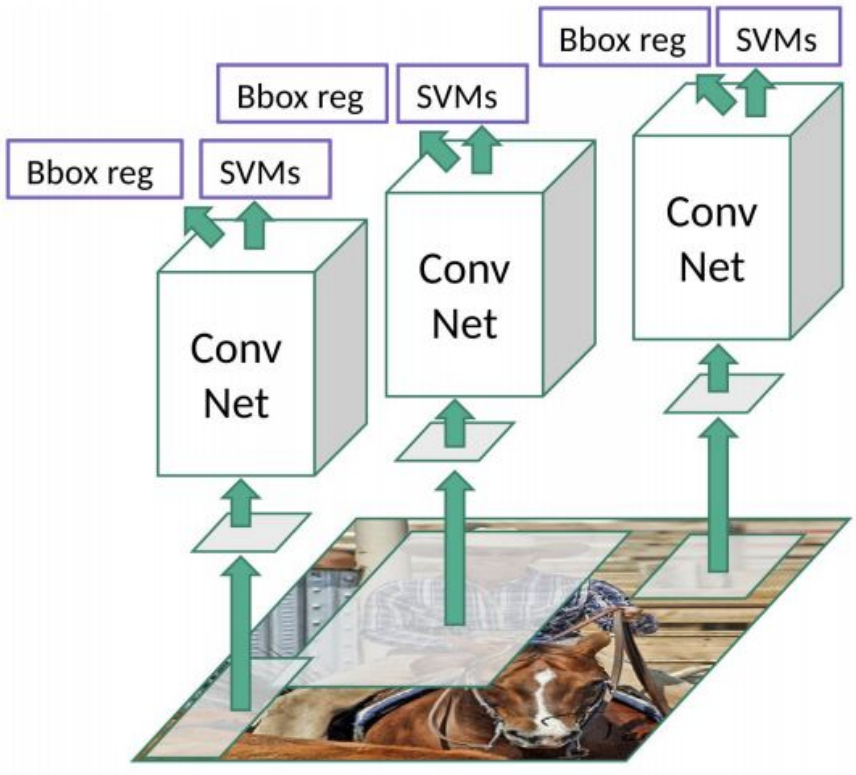


Figure 2: Two examples of our selective search showing the necessity of different scales. On the left we find many objects at different scales. On the right we necessarily find the objects at different scales as the girl is contained by the tv.

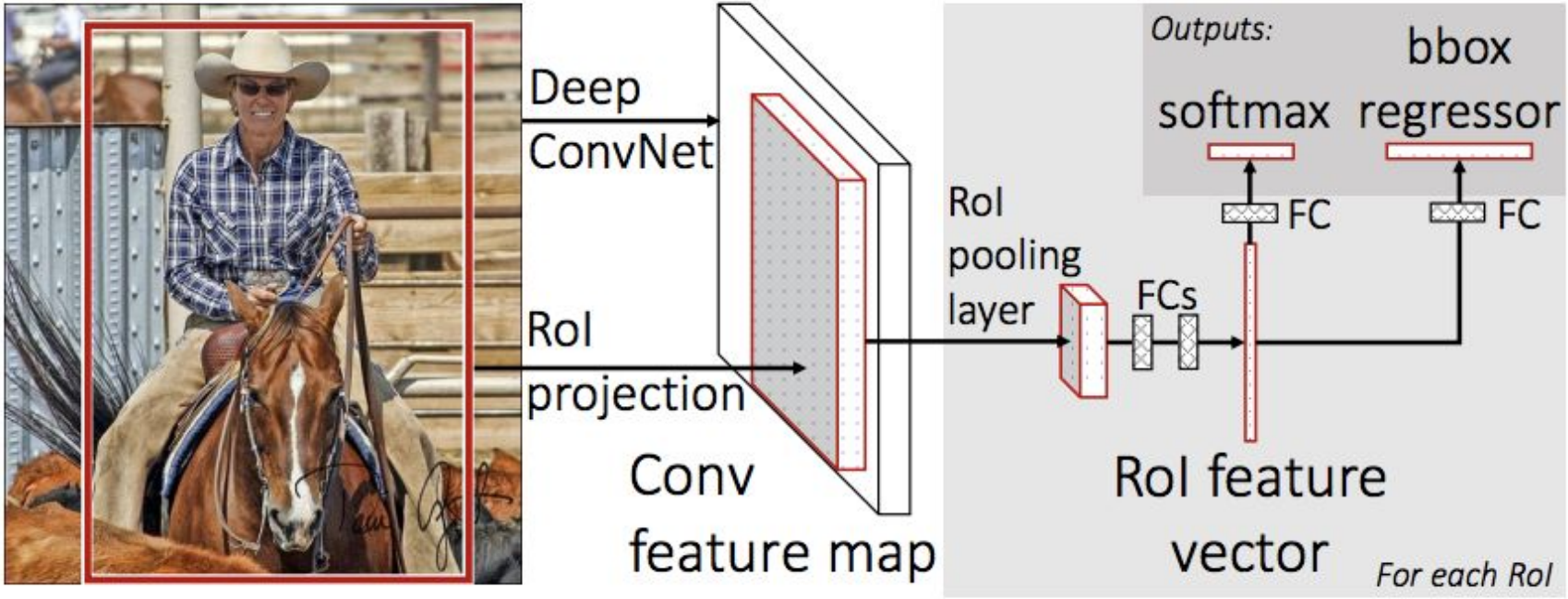
RCNN (2013)


R-CNN: Regions with CNN features
warped region
CNN
aeroplane? no.
person? yes.
tvmonitor? no.
1. Input image
2. Extract region proposals (~2k)
3. Compute CNN features
4. Classify regions

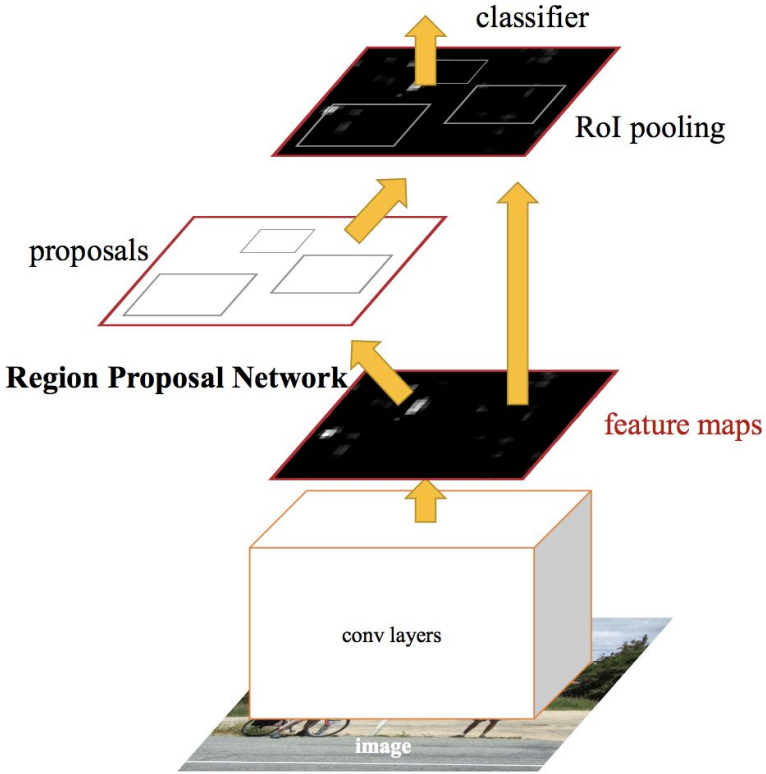
RCNN (2013)


Bbox reg
SVMs
Bbox reg
SVMs
Bbox reg
SVMs
Conv Net
Conv Net
Conv Net

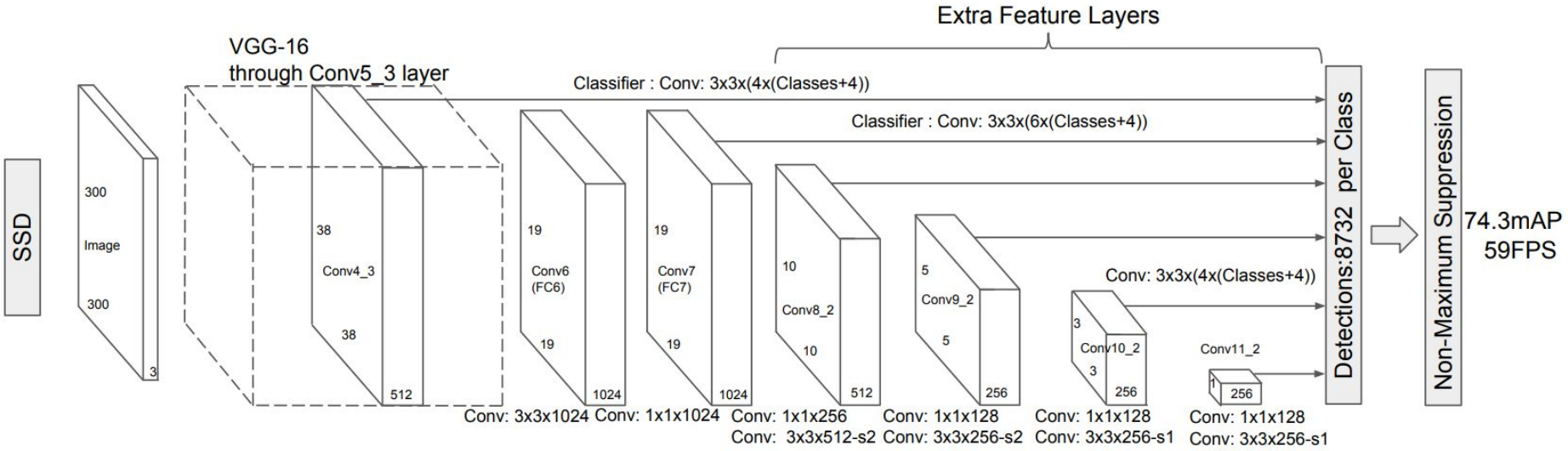
Fast RCNN (2015)

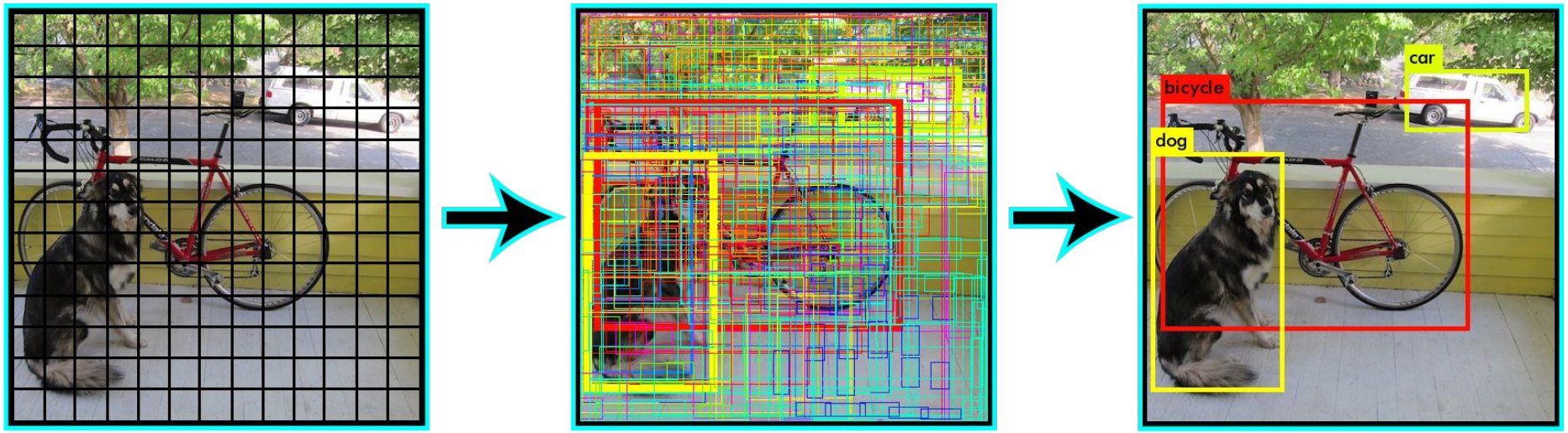


Faster RCNN (2016)

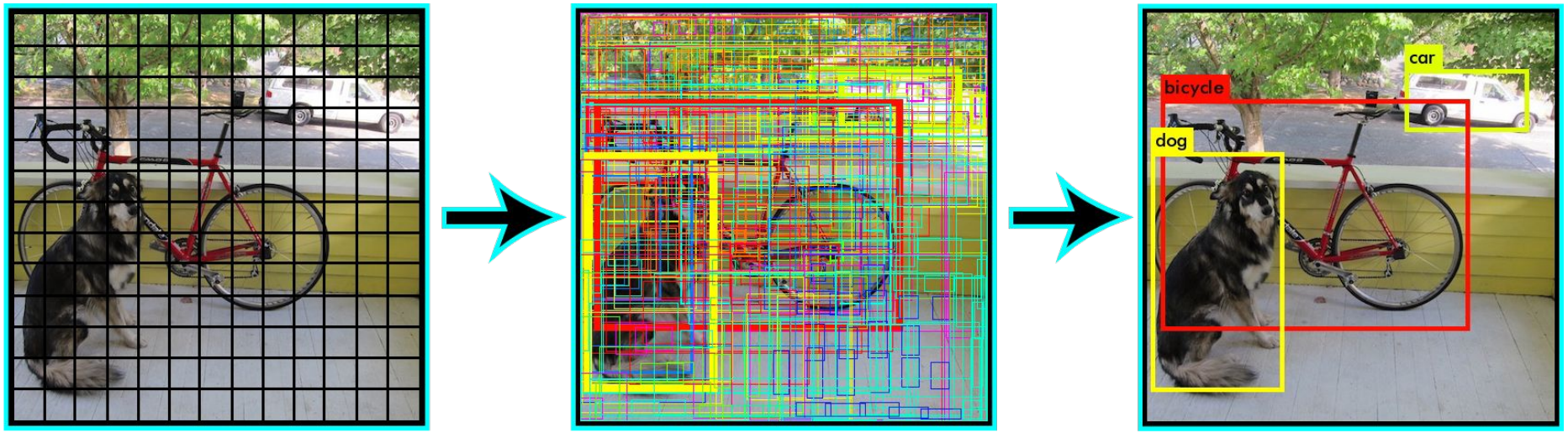


SSD: Single Shot MultiBox Detector (2016)





YOLO (2016, v2 2016, v3 2018, v4, v5 2020)



YOLO - You Only Look Once
(2016, v2 2016, v3 2018, v4, v5 2020)

YOLO - Bounding box regression loss

YOLOv1:

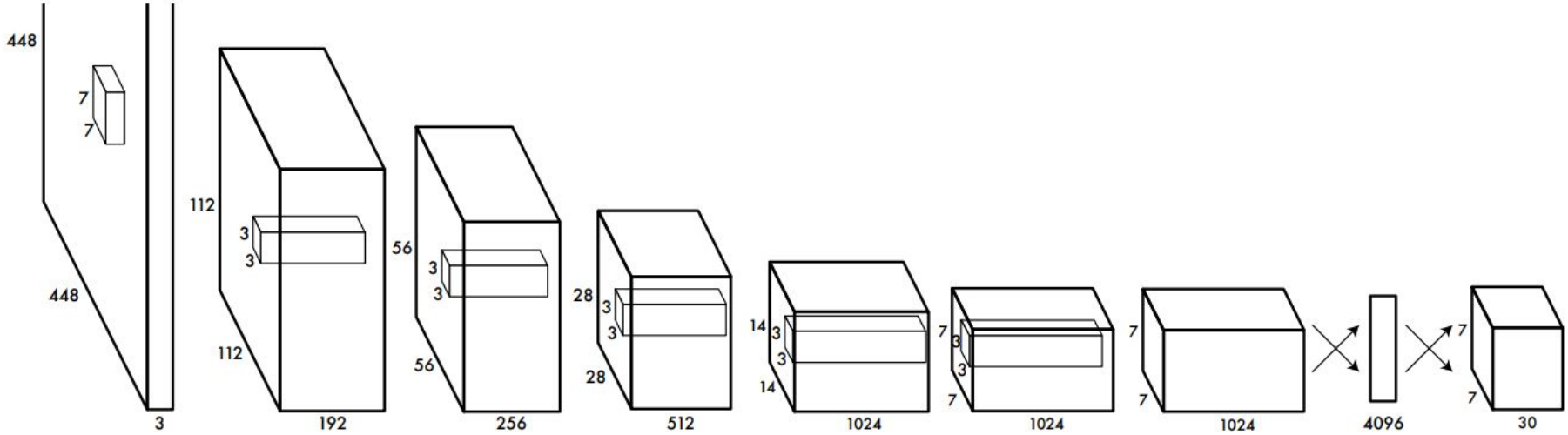
- MSE, błędy lokalizacji bez względu na wielkość boxa

YOLOv2,3:

- MSE, nieliniowa reprezentacja, by skompensować błąd, offsety

YOLOv4:

- CIoU Loss



Conv. Layer
7x7x64-s-2
Maxpool Layer
2x2-s-2

Conv. Layer
3x3x192
Maxpool Layer
2x2-s-2

Conv. Layers
1x1x128
3x3x256
1x1x256
3x3x512
Maxpool Layer
2x2-s-2

Conv. Layers
1x1x256
3x3x512
} ×4
1x1x512
3x3x1024
Maxpool Layer
2x2-s-2

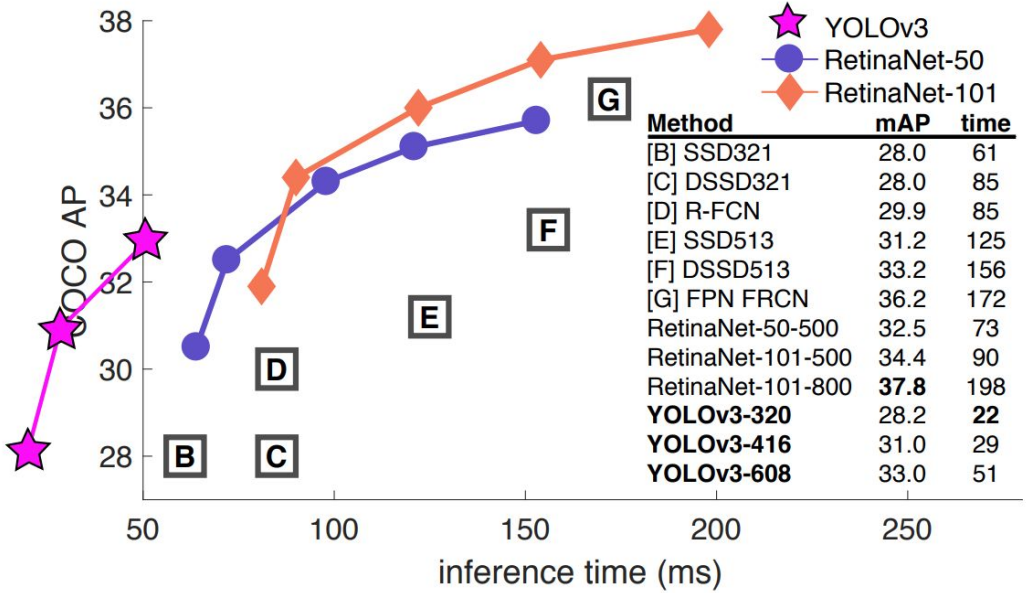
Conv. Layers
1x1x512
3x3x1024
} ×2
3x3x1024
3x3x1024-s-2

Conv. Layers
3x3x1024
3x3x1024

Conn. Layer

Conn. Layer

YOLOv3 (2018)



Method	mAP	time
[B] SSD321	28.0	61
[C] DSSD321	28.0	85
[D] R-FCN	29.9	85
[E] SSD513	31.2	125
[F] DSSD513	33.2	156
[G] FPN FRCN	36.2	172
RetinaNet-50-500	32.5	73
RetinaNet-101-500	34.4	90
RetinaNet-101-800	**37.8**	198
YOLOv3-320	28.2	**22**
YOLOv3-416	31.0	29
YOLOv3-608	33.0	51

Figure 1. We adapt this figure from the Focal Loss paper [9]. YOLOv3 runs significantly faster than other detection methods with comparable performance. Times from either an M40 or Titan X, they are basically the same GPU.

Redmon, Joseph, and Ali Farhadi. "Yolov3: An incremental improvement." *arXiv preprint arXiv:1804.02767* (2018).

Hikvision Markets Uyghur Ethnicity Analytics


Joe Redmon
@pjreddie
“We shouldn’t have to think about the societal impact of our work because it’s hard and other people can do it for us” is a really bad argument.
Roger Grosse @RogerGrosse · Feb 20
Replying to @kevin_zakka and @hardmaru
To be clear, I don't think this is a positive step. Societal impacts of AI is a tough field, and there are researchers and organizations that study it professionally. Most authors do not have expertise in the area and won't do good enough scholarship to say something meaningful.
5:05 PM · Feb 20, 2020 · Twitter Web App
401 Retweets
25 Quote Tweets
1.5K Likes
Joe Redmon @pjreddie · Feb 20
Replying to @pjreddie
I stopped doing CV research because I saw the impact my work was having. I loved the work but the military applications and privacy concerns eventually became impossible to ignore.

YOLOv4 (2020)

YOLOv4 consists of:

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

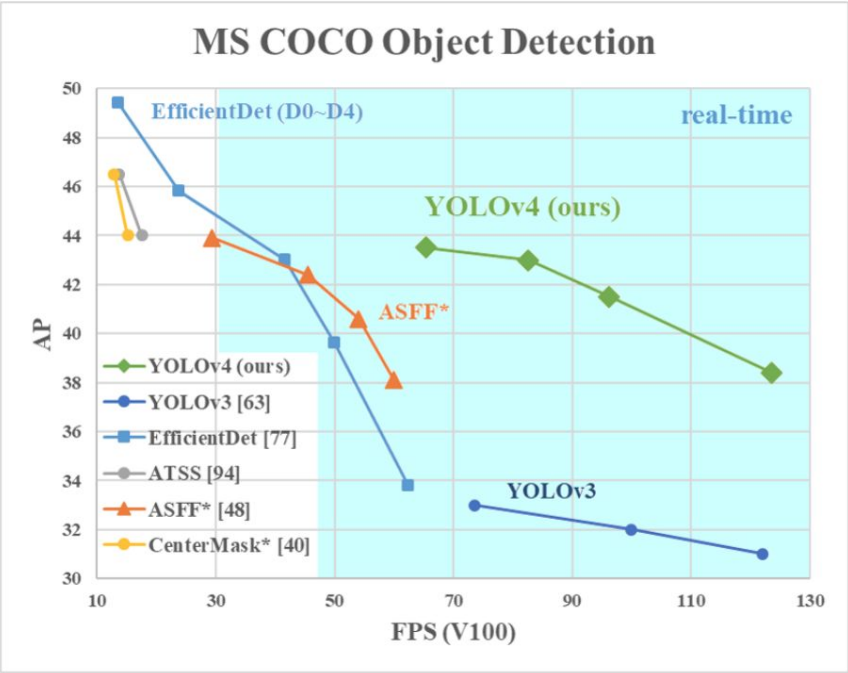
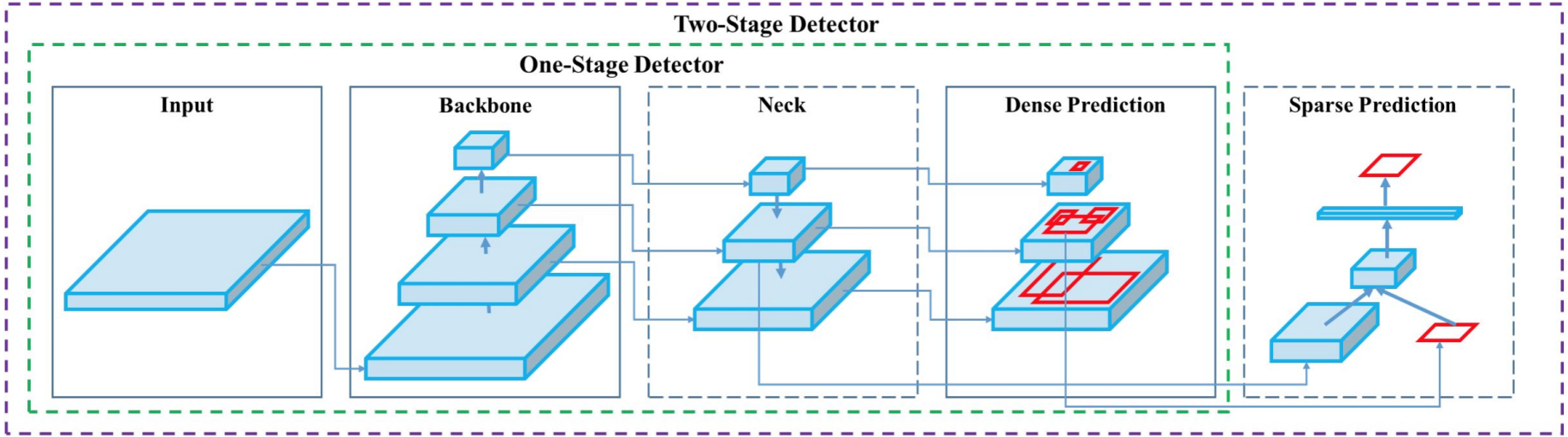


Figure 1: Comparison of the proposed YOLOv4 and other state-of-the-art object detectors. YOLOv4 runs twice faster than EfficientDet with comparable performance. Improves YOLOv3's AP and FPS by 10% and 12%, respectively.

Bochkovskiy, Alexey, Chien-Yao Wang, and Hong-Yuan Mark Liao. "YOLOv4: Optimal Speed and Accuracy of Object Detection." *arXiv preprint arXiv:2004.10934* (2020).



Input: { Image, Patches, Image Pyramid, … }

Backbone: { VGG16 [68], ResNet-50 [26], ResNeXt-101 [86], Darknet53 [63], … }

Neck: { FPN [44], PANet [49], Bi-FPN [77], … }

Head:

Dense Prediction: { RPN [64], YOLO [61, 62, 63], SSD [50], RetinaNet [45], FCOS [78], … }

Sparse Prediction: { Faster R-CNN [64], R-FCN [9], … }

Figure 2: Object detector.

YOLOv4 (2020)

Table 2: Influence of BoF and Mish on the CSPResNeXt-50 classifier accuracy.

MixUp	CutMix	Mosaic	Bluring	Label Smoothing	Swish	Mish	Top-1	Top-5
							77.9%	94.0%
✓							77.2%	**94.0%**
	✓						**78.0%**	**94.3%**
		✓					**78.1%**	**94.5%**
			✓				77.5%	93.8%
				✓			**78.1%**	**94.4%**
					✓		64.5%	86.0%
						✓	**78.9%**	**94.5%**
	✓	✓		✓			**78.5%**	**94.8%**
	✓	✓		✓		✓	**79.8%**	**95.2%**

Table 3: Influence of BoF and Mish on the CSPDarknet-53 classifier accuracy.

MixUp	CutMix	Mosaic	Bluring	Label Smoothing	Swish	Mish	Top-1	Top-5
							77.2%	93.6%
	✓	✓		✓			**77.8%**	**94.4%**
	✓	✓		✓		✓	**78.7%**	**94.8%**

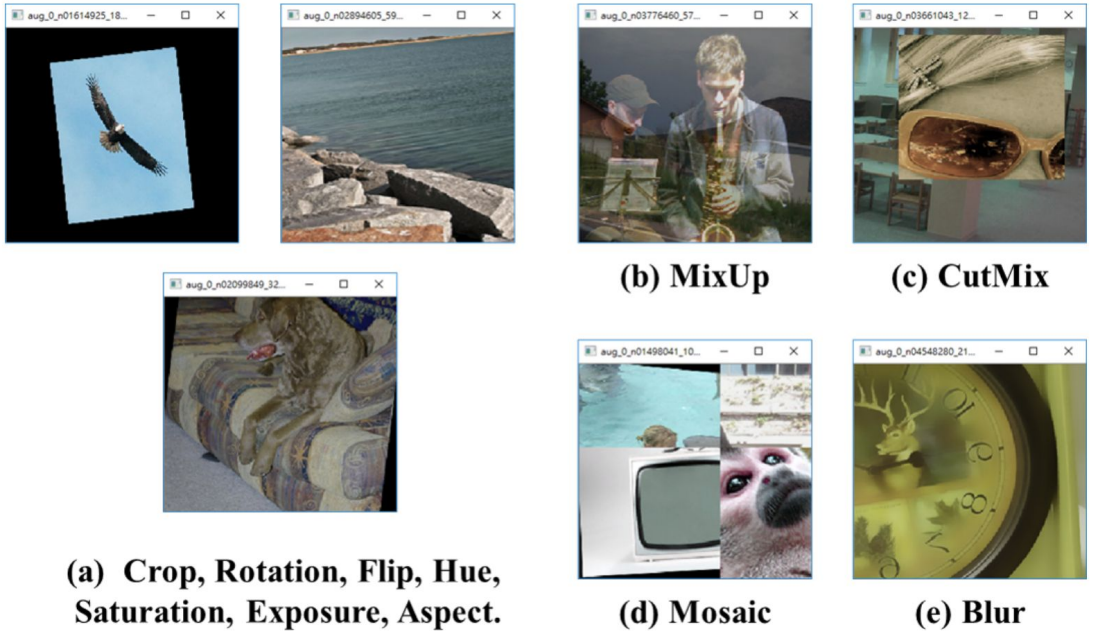


Figure 7: Various method of data augmentation.

Table 9: Comparison of the speed and accuracy of different object detectors on the MS COCO dataset (test-dev 2017). (Real-time detectors with FPS 30 or higher are highlighted here. We compare the results with batch=1 without using tensorRT.)

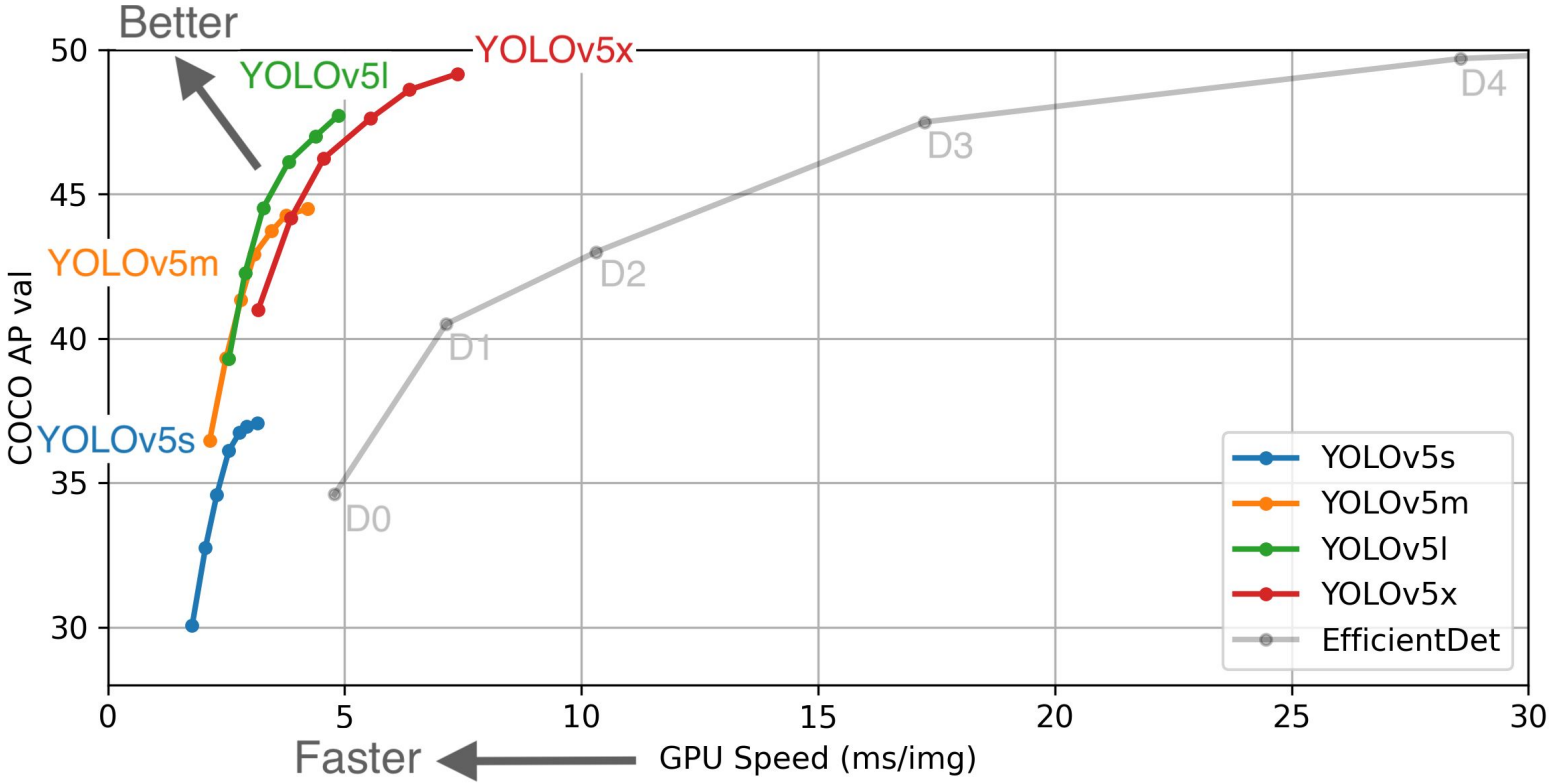
Method	Backbone	Size	FPS	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
YOLOv4: Optimal Speed and Accuracy of Object Detection									
YOLOv4	CSPDarknet-53	416	54 (P)	41.2%	62.8%	44.3%	20.4%	44.4%	56.0%
YOLOv4	CSPDarknet-53	512	43 (P)	43.0%	64.9%	46.5%	24.3%	46.1%	55.2%
YOLOv4	CSPDarknet-53	608	33 (P)	**43.5%**	**65.7%**	**47.3%**	**26.7%**	**46.7%**	53.3%
CenterMask: Real-Time Anchor-Free Instance Segmentation [40]									
CenterMask-Lite	MobileNetV2-FPN	600×	50.0 (P)	30.2%	-	-	14.2%	31.9%	40.9%
CenterMask-Lite	VoVNet-19-FPN	600×	43.5 (P)	35.9%	-	-	19.6%	38.0%	45.9%
CenterMask-Lite	VoVNet-39-FPN	600×	35.7 (P)	40.7%	-	-	22.4%	43.2%	**53.5%**
Enriched Feature Guided Refinement Network for Object Detection [57]									
EFGRNet	VGG-16	320	47.6 (P)	33.2%	53.4%	35.4%	13.4%	37.1%	47.9%
EFGRNet	VG-G16	512	25.7 (P)	37.5%	58.8%	40.4%	19.7%	41.6%	49.4%
EFGRNet	ResNet-101	512	21.7 (P)	39.0%	58.8%	42.3%	17.8%	43.6%	54.5%
Hierarchical Shot Detector [3]									
HSD	VGG-16	320	40 (P)	33.5%	53.2%	36.1%	15.0%	35.0%	47.8%
HSD	VGG-16	512	23.3 (P)	38.8%	58.2%	42.5%	21.8%	41.9%	50.2%
HSD	ResNet-101	512	20.8 (P)	40.2%	59.4%	44.0%	20.0%	44.4%	54.9%
HSD	ResNeXt-101	512	15.2 (P)	41.9%	61.1%	46.2%	21.8%	46.6%	57.0%
HSD	ResNet-101	768	10.9 (P)	42.3%	61.2%	46.9%	22.8%	47.3%	55.9%
Dynamic anchor feature selection for single-shot object detection [41]									
DAFS	VGG16	512	35 (P)	33.8%	52.9%	36.9%	14.6%	37.0%	47.7%

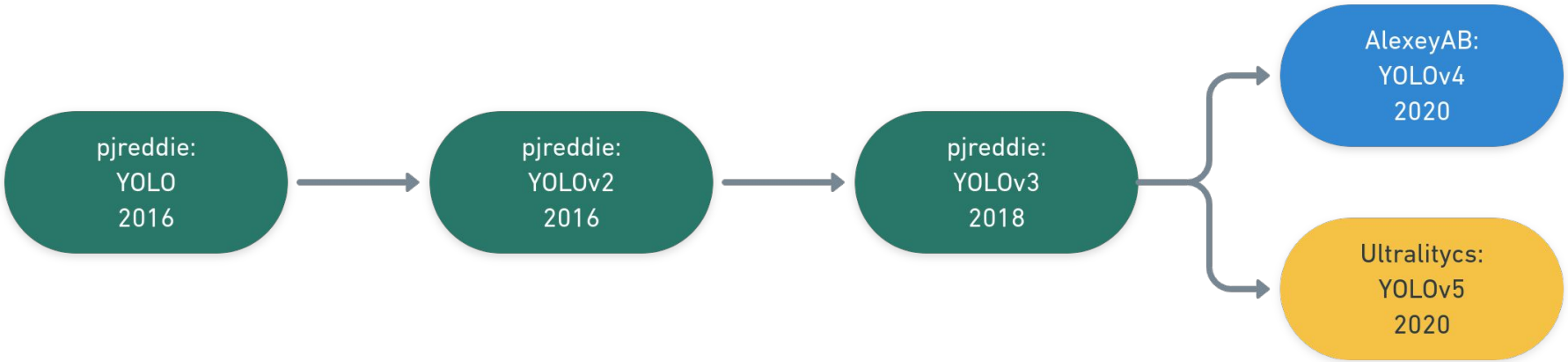
Soft Anchor-Point Object Detection [101]									
SAPD	ResNet-50	-	14.9 (P)	41.7%	61.9%	44.6%	24.1%	44.6%	51.6%
SAPD	ResNet-50-DCN	-	12.4 (P)	44.3%	64.4%	47.7%	25.5%	47.3%	57.0%
SAPD	ResNet-101-DCN	-	9.1 (P)	46.0%	65.9%	49.6%	26.3%	49.2%	59.6%
Region proposal by guided anchoring [82]									
RetinaNet	ResNet-50	-	10.8 (P)	37.1%	56.9%	40.0%	20.1%	40.1%	48.0%
Faster R-CNN	ResNet-50	-	9.4 (P)	39.8%	59.2%	43.5%	21.8%	42.6%	50.7%
RepPoints: Point set representation for object detection [87]									
RPDet	ResNet-101	-	10 (P)	41.0%	62.9%	44.3%	23.6%	44.1%	51.7%
RPDet	ResNet-101-DCN	-	8 (P)	45.0%	66.1%	49.0%	26.6%	48.6%	57.5%
Libra R-CNN: Towards balanced learning for object detection [58]									
Libra R-CNN	ResNet-101	-	9.5 (P)	41.1%	62.1%	44.7%	23.4%	43.7%	52.5%
FreeAnchor: Learning to match anchors for visual object detection [96]									
FreeAnchor	ResNet-101	-	9.1 (P)	43.1%	62.2%	46.4%	24.5%	46.1%	54.8%
RetinaMask: Learning to Predict Masks Improves State-of-The-Art Single-Shot Detection for Free [14]									
RetinaMask	ResNet-50-FPN	800×	8.1 (P)	39.4%	58.6%	42.3%	21.9%	42.0%	51.0%
RetinaMask	ResNet-101-FPN	800×	6.9 (P)	41.4%	60.8%	44.6%	23.0%	44.5%	53.5%
RetinaMask	ResNet-101-FPN-GN	800×	6.5 (P)	41.7%	61.7%	45.0%	23.5%	44.7%	52.8%
RetinaMask	ResNeXt-101-FPN-GN	800×	4.3 (P)	42.6%	62.5%	46.0%	24.8%	45.6%	53.8%
Cascade R-CNN: Delving into high quality object detection [2]									
Cascade R-CNN	ResNet-101	-	8 (P)	42.8%	62.1%	46.3%	23.7%	45.5%	55.2%
Centernet: Object detection with keypoint triplets [13]									
Centernet	Hourglass-52	-	4.4 (P)	41.6%	59.4%	44.2%	22.5%	43.1%	54.1%
Centernet	Hourglass-104	-	3.3 (P)	44.9%	62.4%	48.1%	25.6%	47.4%	57.4%
Scale-Aware Trident Networks for Object Detection [42]									
TridentNet	ResNet-101	-	2.7 (P)	42.7%	63.6%	46.5%	23.9%	46.6%	56.6%
TridentNet	ResNet-101-DCN	-	1.3 (P)	46.8%	67.6%	51.5%	28.0%	51.2%	60.5%

Table 10: Comparison of the speed and accuracy of different object detectors on the MS COCO dataset (test-dev 2017). (Real-time detectors with FPS 30 or higher are highlighted here. We compare the results with batch=1 without using tensorRT.)

Method	Backbone	Size	FPS	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
YOLOv4: Optimal Speed and Accuracy of Object Detection									
YOLOv4	CSPDarknet-53	416	96 (V)	41.2%	62.8%	44.3%	20.4%	44.4%	56.0%
YOLOv4	CSPDarknet-53	512	83 (V)	43.0%	64.9%	46.5%	24.3%	46.1%	55.2%
YOLOv4	CSPDarknet-53	608	62 (V)	**43.5%**	**65.7%**	47.3%	**26.7%**	46.7%	53.3%
EfficientDet: Scalable and Efficient Object Detection [77]									
EfficientDet-D0	Efficient-B0	512	62.5 (V)	33.8%	52.2%	35.8%	12.0%	38.3%	51.2%
EfficientDet-D1	Efficient-B1	640	50.0 (V)	39.6%	58.6%	42.3%	17.9%	44.3%	56.0%
EfficientDet-D2	Efficient-B2	768	41.7 (V)	43.0%	62.3%	46.2%	22.5%	**47.0%**	**58.4%**
EfficientDet-D3	Efficient-B3	896	23.8 (V)	45.8%	65.0%	49.3%	26.6%	49.4%	59.8%
Learning Spatial Fusion for Single-Shot Object Detection [48]									
YOLOv3 + ASFF*	Darknet-53	320	60 (V)	38.1%	57.4%	42.1%	16.1%	41.6%	53.6%
YOLOv3 + ASFF*	Darknet-53	416	54 (V)	40.6%	60.6%	45.1%	20.3%	44.2%	54.1%
YOLOv3 + ASFF*	Darknet-53	608×	45.5 (V)	42.4%	63.0%	**47.4%**	25.5%	45.7%	52.3%
YOLOv3 + ASFF*	Darknet-53	800×	29.4 (V)	43.9%	64.1%	49.2%	27.0%	46.6%	53.4%
HarDNet: A Low Memory Traffic Network [4]									
RFBNet	HarDNet68	512	41.5 (V)	33.9%	54.3%	36.2%	14.7%	36.6%	50.5%
RFBNet	HarDNet85	512	37.1 (V)	36.8%	57.1%	39.5%	16.9%	40.5%	52.9%
Focal Loss for Dense Object Detection [45]									
RetinaNet	ResNet-50	640	37 (V)	37.0%	-	-	-	-	-
RetinaNet	ResNet-101	640	29.4 (V)	37.9%	-	-	-	-	-
RetinaNet	ResNet-50	1024	19.6 (V)	40.1%	-	-	-	-	-
RetinaNet	ResNet-101	1024	15.4 (V)	41.1%	-	-	-	-	-
SM-NAS: Structural-to-Modular Neural Architecture Search for Object Detection [88]									
SM-NAS: E2	-	800×600	25.3 (V)	40.0%	58.2%	43.4%	21.1%	42.4%	51.7%
SM-NAS: E3	-	800×600	19.7 (V)	42.8%	61.2%	46.5%	23.5%	45.5%	55.6%
SM-NAS: E5	-	1333×800	9.3 (V)	45.9%	64.6%	49.6%	27.1%	49.0%	58.0%
NAS-FPN: Learning scalable feature pyramid architecture for object detection [17]									
NAS-FPN	ResNet-50	640	24.4 (V)	39.9%	-	-	-	-	-
NAS-FPN	ResNet-50	1024	12.7 (V)	44.2%	-	-	-	-	-
Bridging the Gap Between Anchor-based and Anchor-free Detection via Adaptive Training Sample Selection [94]									
ATSS	ResNet-101	800×	17.5 (V)	43.6%	62.1%	47.4%	26.1%	47.0%	53.6%
ATSS	ResNet-101-DCN	800×	13.7 (V)	46.3%	64.7%	50.4%	27.7%	49.8%	58.4%
RDSNet: A New Deep Architecture for Reciprocal Object Detection and Instance Segmentation [83]									
RDSNet	ResNet-101	600	16.8 (V)	36.0%	55.2%	38.7%	17.4%	39.6%	49.7%
RDSNet	ResNet-101	800	10.9 (V)	38.1%	58.5%	40.8%	21.2%	41.5%	48.2%
CenterMask: Real-Time Anchor-Free Instance Segmentation [40]									
CenterMask	ResNet-101-FPN	800×	15.2 (V)	44.0%	-	-	25.8%	46.8%	54.9%
CenterMask	VoVNet-99-FPN	800×	12.9 (V)	46.5%	-	-	28.7%	48.9%	57.2%

YOLOv5 - Ultralytics (2020)




pjreddie:
YOLO
2016
pjreddie:
YOLOv2
2016
pjreddie:
YOLOv3
2018
AlexeyAB:
YOLOv4
2020
Ultralitycs:
YOLOv5
2020

Joe Redmon
@pjreddie

Doesn’t matter what I think! At this point @alexeyab84 has the canonical version of darknet and yolo, he’s put a ton of work into it and everyone uses it, not mine haha.

Перевести твит

>  **Sarim Zafar** @1Sarim · 24 апр.
> В ответ @ak92501
> @pjreddie do you approve?

6:38 AM · 25 апр. 2020 г. · Twitter Web App

YOLOv5 - Ultralytics (2020)

CSPDarknet53s-PASPP-Mish: (~YOLOv4)

```
cd53s-paspp-mish 45.0% AP @ 608x608
Model Summary: 212 layers, 6.43092e+07 parameters, 6.43092e+07 gradients
Speed: 8.7/1.6/10.3 ms inference/NMS/total per 608x608 image at batch-size 16
```

YOLOv5l:

```
yolov5l 44.2% AP @ 736x736
Model Summary: 231 layers, 6.17556e+07 parameters, 6.17556e+07 gradients
Speed: 11.3/2.2/13.5 ms inference/NMS/total per 736x736 image at batch-size 16
```

YOLOv5 - Ultralytics (2020)

CSPDarknet53s-PASPP-Mish: (~YOLOv4)

```
cd53s-paspp-mish 45.0% AP @ 608x608
Model Summary: 212 layers, 6.43092e+07 parameters, 6.43092e+07 gradients
Speed: 8.7/1.6/10.3 ms inference/NMS/total per 608x608 image at batch-size 16
```

YOLOv5l:

```
yolov5l 44.2% AP @ 736x736
Model Summary: 231 layers, 6.17556e+07 parameters, 6.17556e+07 gradients
Speed: 11.3/2.2/13.5 ms inference/NMS/total per 736x736 image at batch-size 16
```

YOLOv5 - Ultralytics (2020)

CSPDarknet53s-PASPP-Mish: (~YOLOv4)

```
cd53s-paspp-mish 45.0% AP @ 608x608
Model Summary: 212 layers, 6.43092e+07 parameters, 6.43092e+07 gradients
Speed: 8.7/1.6/10.3 ms inference/NMS/total per 608x608 image at batch-size 16
```

YOLOv5l:

```
yolov5l 44.2% AP @ 736x736
Model Summary: 231 layers, 6.17556e+07 parameters, 6.17556e+07 gradients
Speed: 11.3/2.2/13.5 ms inference/NMS/total per 736x736 image at batch-size 16
```

YOLOv5 - Ultralytics (2020)

YOLOv3-SPP:

```
yolov3-spp: 45.5% AP @736x736
Model Summary: 225 layers, 6.29987e+07 parameters, 6.29987e+07 gradients
Speed: 10.4/2.1/12.6 ms inference/NMS/total per 736x736 image at batch-size 16
```

YOLOv5l:

```
yolov5l 44.2% AP @ 736x736
Model Summary: 231 layers, 6.17556e+07 parameters, 6.17556e+07 gradients
Speed: 11.3/2.2/13.5 ms inference/NMS/total per 736x736 image at batch-size 16
```

YOLOv5 - Ultralytics (2020)

YOLOv3-SPP:

```
yolov3-spp: 45.5% AP @736x736
Model Summary: 225 layers, 6.29987e+07 parameters, 6.29987e+07 gradients
Speed: 10.4/2.1/12.6 ms inference/NMS/total per 736x736 image at batch-size 16
```

YOLOv5l:

```
yolov5l 44.2% AP @ 736x736
Model Summary: 231 layers, 6.17556e+07 parameters, 6.17556e+07 gradients
Speed: 11.3/2.2/13.5 ms inference/NMS/total per 736x736 image at batch-size 16
```

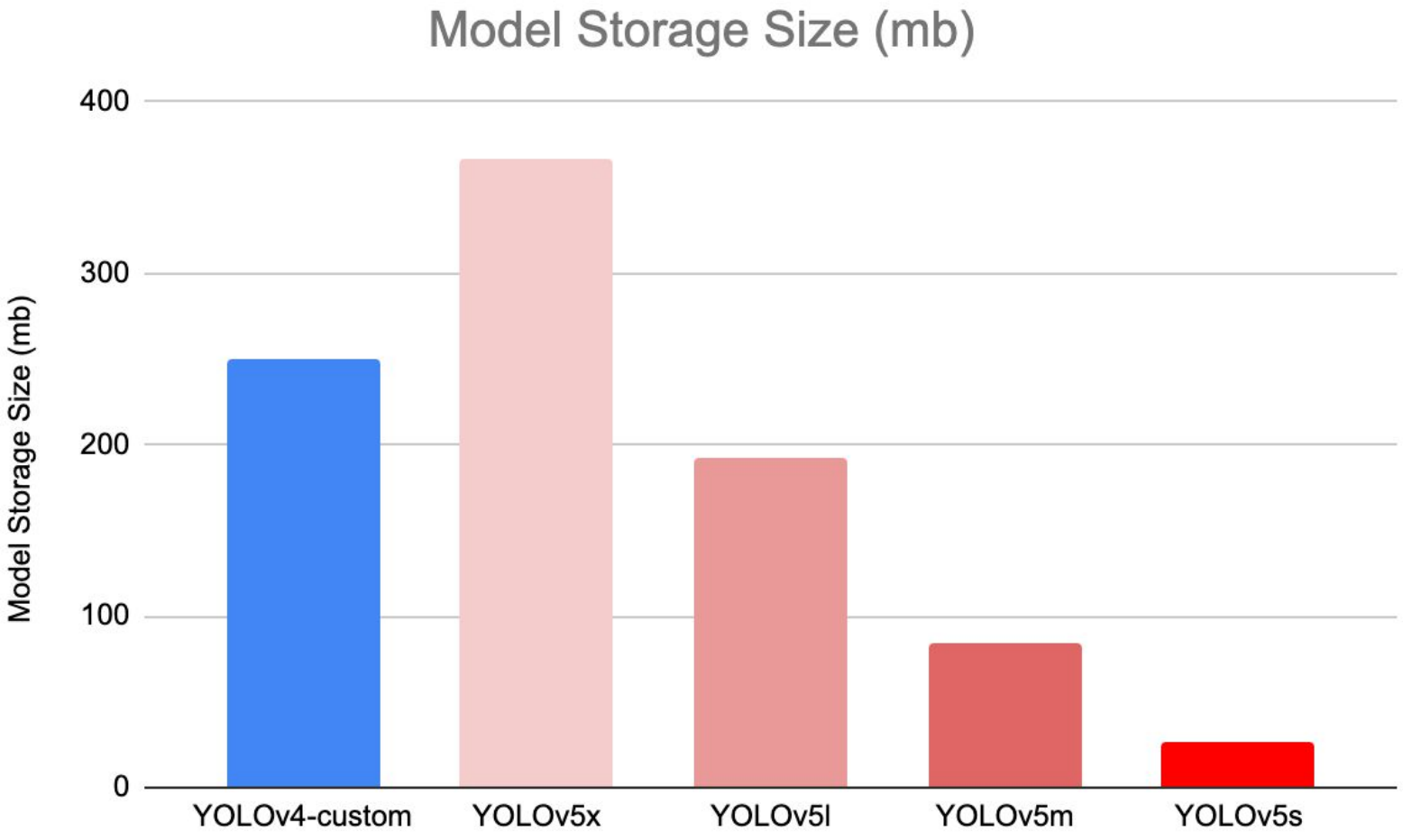
- weights size: YOLOv4s `245 MB` vs YOLOv5l `192 MB` vs YOLOv5x `366 MB`

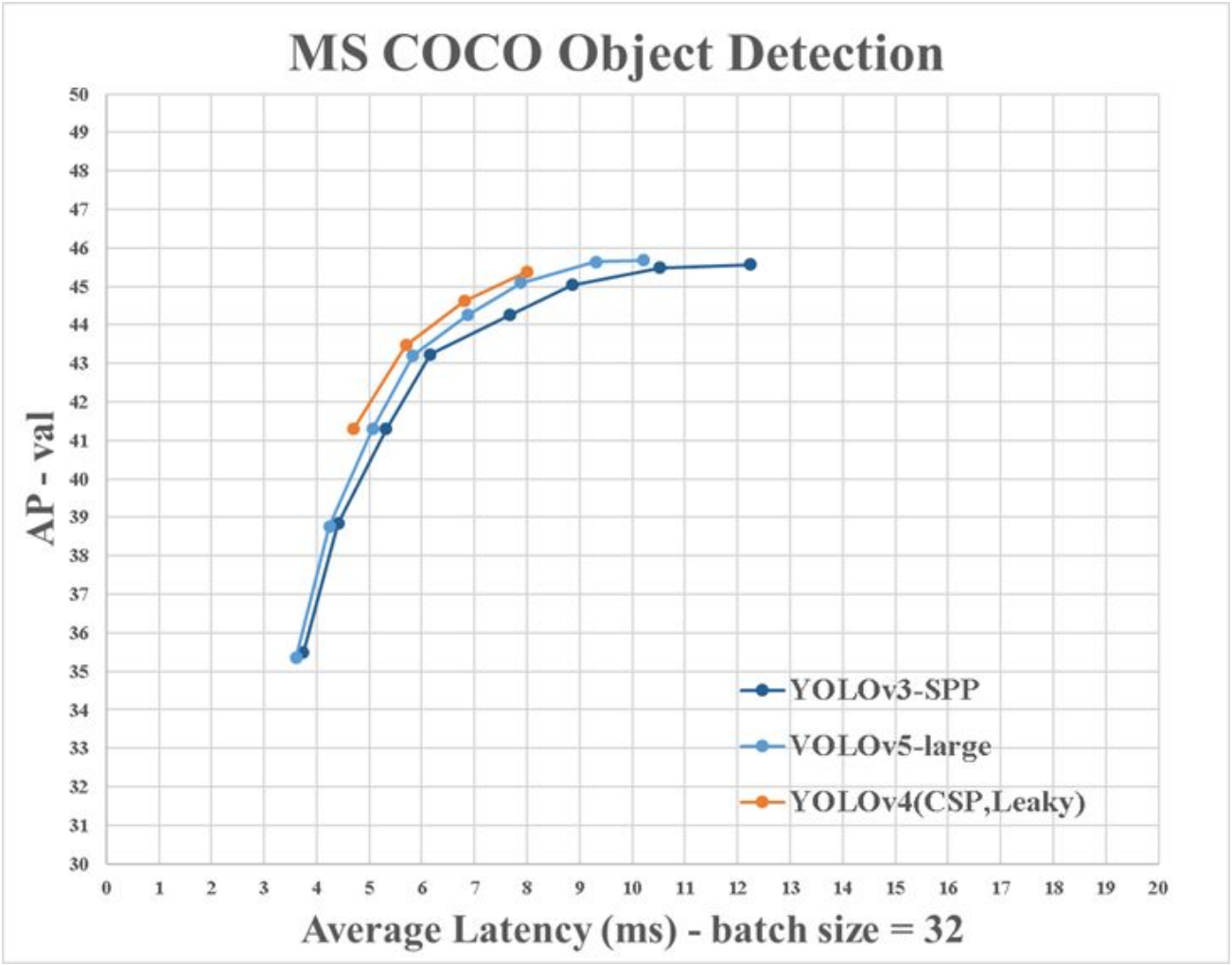
- weights size: YOLOv4s `245 MB` vs YOLOv5l `192 MB` vs YOLOv5x `366 MB`
- test-dev accuracy on MSCOCO: YOLOv4s-608 `45% AP` vs YOLOv5l-736 `44.2% AP` (**YOLOv4 is more accurate**)

- weights size: YOLOv4s `245 MB` vs YOLOv5l `192 MB` vs YOLOv5x `366 MB`
- test-dev accuracy on MSCOCO: YOLOv4s-608 `45% AP` vs YOLOv5l-736 `44.2% AP` (**YOLOv4 is more accurate**)
- speed with batch=16: YOLOv4s-608 `10.3ms` vs YOLOv5l-736 `13.5ms` (**YOLOv4 is faster**)

- weights size: YOLOv4s `245 MB` vs YOLOv5l `192 MB` vs YOLOv5x `366 MB`
- test-dev accuracy on MSCOCO: YOLOv4s-608 `45% AP` vs YOLOv5l-736 `44.2% AP` (**YOLOv4 is more accurate**)
- speed with batch=16: YOLOv4s-608 `10.3ms` vs YOLOv5l-736 `13.5ms` (**YOLOv4 is faster**)
- roboflow.ai shared the Latency-Accuracy chart withultralytics-YOLOv5 which are measured with batch=32 and then divided by 32, while latency must be measured with batch=1, because the higher batch - the higher latency, latency of 1 sample can't be less than latency of the whole batch, so **real latency of YOLOv5 can be up to `~1 second`** with high batch-size=32-64

- weights size: YOLOv4s `245 MB` vs YOLOv5l `192 MB` vs YOLOv5x `366 MB`
- test-dev accuracy on MSCOCO: YOLOv4s-608 `45% AP` vs YOLOv5l-736 `44.2% AP` (**YOLOv4 is more accurate**)
- speed with batch=16: YOLOv4s-608 `10.3ms` vs YOLOv5l-736 `13.5ms` (**YOLOv4 is faster**)
- roboflow.ai shared the Latency-Accuracy chart withultralytics-YOLOv5 which are measured with batch=32 and then divided by 32, while latency must be measured with batch=1, because the higher batch - the higher latency, latency of 1 sample can't be less than latency of the whole batch, so **real latency of YOLOv5 can be up to `~1 second`** with high batch-size=32-64
- they stated 140 FPS for YOLOv5 (s/m/l/x ???) (what batch-size ???) while **YOLOv4 achieves ~400 FPS** just with batch=4 by using OpenCV-dnn or TensorRT on GPU RTX 2080ti (table above)


Model Storage Size (mb)
Model Storage Size (mb)
400
300
200
100
0
YOLOv4-custom
YOLOv5x
YOLOv5l
YOLOv5m
YOLOv5s


MS COCO Object Detection
AP - val
Average Latency (ms) - batch size = 32
YOLOv3-SPP
VOLOv5-large
YOLOv4(CSP,Leaky)

Facebook DETR

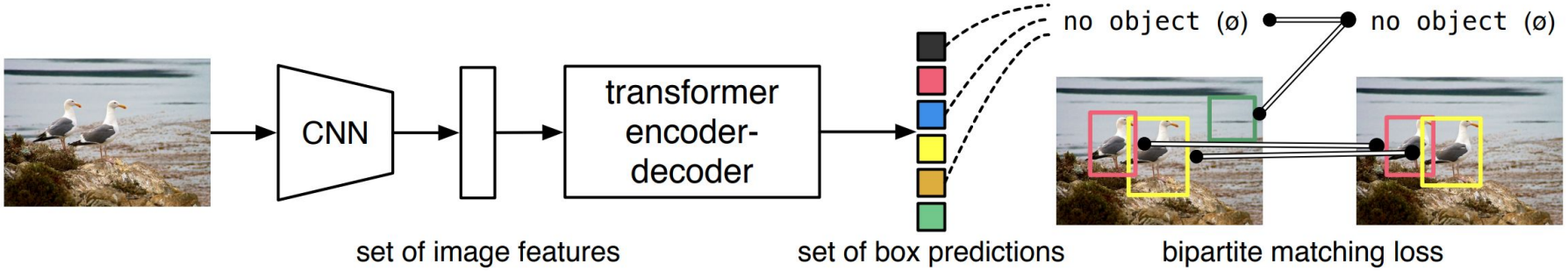


Fig. 1: DETR directly predicts (in parallel) the final set of detections by combining a common CNN with a transformer architecture. During training, bipartite matching uniquely assigns predictions with ground truth boxes. Prediction with no match should yield a "`no object`" ($\varnothing$) class prediction.

Facebook DETR

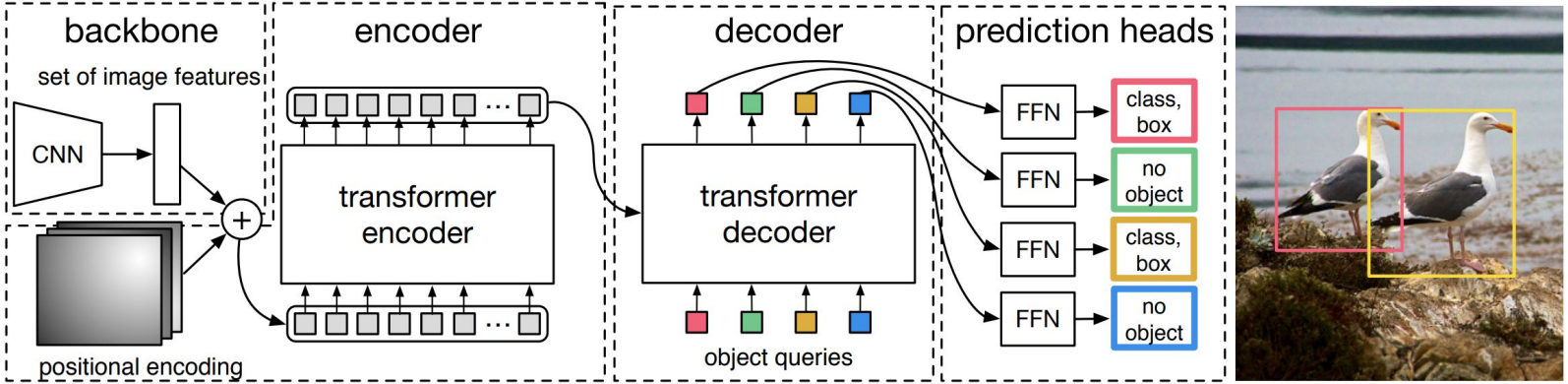


Fig. 2: DETR uses a conventional CNN backbone to learn a 2D representation of an input image. The model flattens it and supplements it with a positional encoding before passing it into a transformer encoder. A transformer decoder then takes as input a small fixed number of learned positional embeddings, which we call *object queries*, and additionally attends to the encoder output. We pass each output embedding of the decoder to a shared feed forward network (FFN) that predicts either a detection (class and bounding box) or a "`no object`" class.

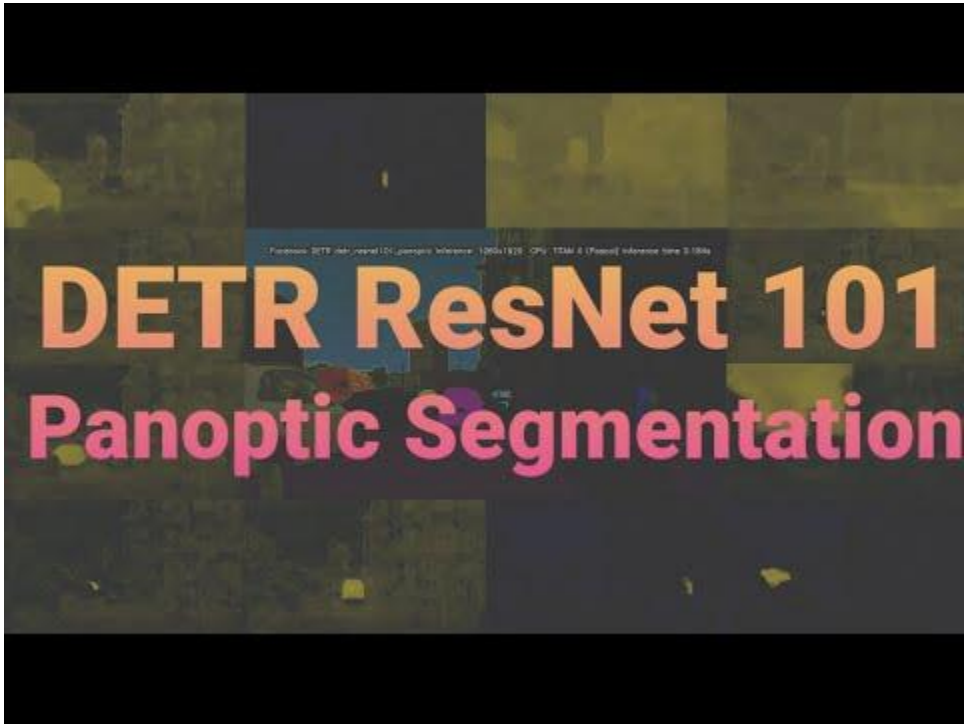
Facebook DETR

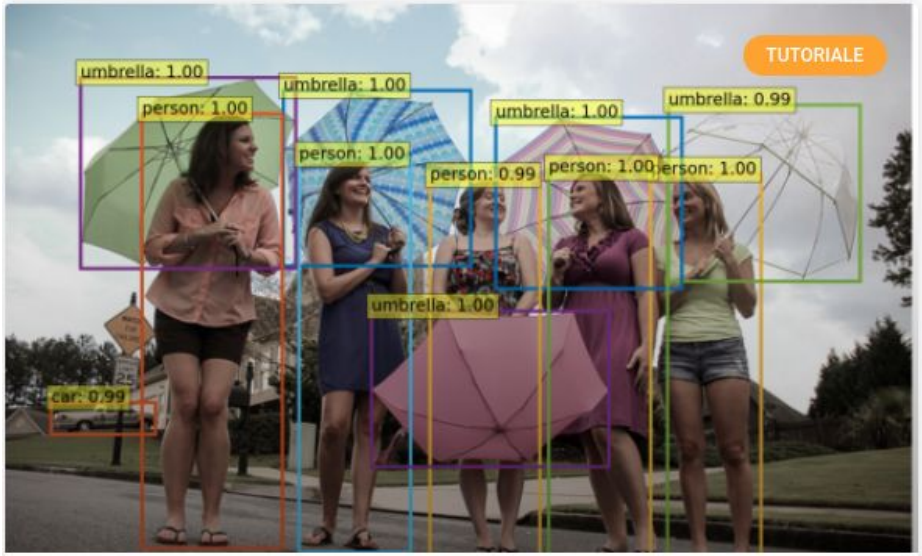
Table 1: Comparison with Faster R-CNN with a ResNet-50 and ResNet-101 backbones on the COCO validation set. The top section shows results for Faster R-CNN models in Detectron2 [50], the middle section shows results for Faster R-CNN models with GIoU [38], random crops train-time augmentation, and the long `9x` training schedule. DETR models achieve comparable results to heavily tuned Faster R-CNN baselines, having lower AP_S but greatly improved AP_L. We use torchscript Faster R-CNN and DETR models to measure FLOPS and FPS. Results without R101 in the name correspond to ResNet-50.

Model	GFLOPS/FPS	#params	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Faster RCNN-DC5	320/16	166M	39.0	60.5	42.3	21.4	43.5	52.5
Faster RCNN-FPN	180/26	42M	40.2	61.0	43.8	24.2	43.5	52.0
Faster RCNN-R101-FPN	246/20	60M	42.0	62.5	45.9	25.2	45.6	54.6
Faster RCNN-DC5+	320/16	166M	41.1	61.4	44.3	22.9	45.9	55.0
Faster RCNN-FPN+	180/26	42M	42.0	62.1	45.5	26.6	45.4	53.4
Faster RCNN-R101-FPN+	246/20	60M	44.0	63.9	**47.8**	**27.2**	48.1	56.0
DETR	86/28	41M	42.0	62.4	44.2	20.5	45.8	61.1
DETR-DC5	187/12	41M	43.3	63.1	45.9	22.5	47.3	61.1
DETR-R101	152/20	60M	43.5	63.8	46.4	21.9	48.0	61.8
DETR-DC5-R101	253/10	60M	**44.9**	**64.7**	47.7	23.7	**49.5**	**62.3**

Facebook DETR

Facebook DETR





Jak uruchomić DETR do wykrywania obiektów?

Pierwszą część, dotyczącą segmentacji instancji, możesz przeczytać tutaj: Ten post dotyczyć będzie wykorzystania modelu DETR do wykrywania obiektów i jest on kontynuacją wpisu dotyczącego segmentacji

[CZYTAJ]

Aleksandra Kos • 21 lipca, 2020

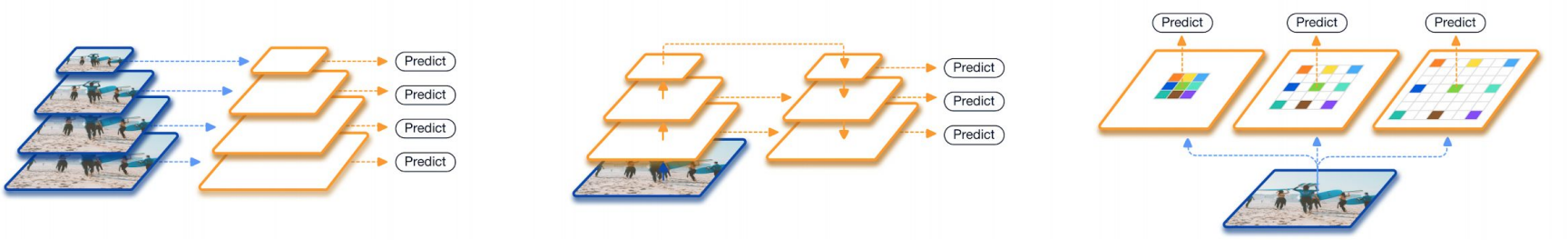
Jak uruchomić DETR do segmentacji instancji?

Zacznijmy od kwestii organizacyjnych. Jeśli chcesz, możesz na bieżąco pracować w notebooku udostępnionym na colabie. Co robi nasz program? Przede wszystkim wykonuje inferencję na pojedynczej

[CZYTAJ]

Aleksandra Kos • 21 lipca, 2020

TridentNet (2019)



(a) Image Pyramid (b) Feature Pyramid (c) Trident Network

Figure 1: (a) Using multiple images of several scales as input, the image pyramid methods perform feature extraction and object detection independently for each scale. (b) The feature pyramid methods utilize the features from different layers of CNNs for different scales, which is computational friendly. This figure takes FPN [25] as an example. (c) Our proposed Trident Network generates scale-aware feature maps efficiently by trident blocks with different receptive fields.

TridentNet (2019)

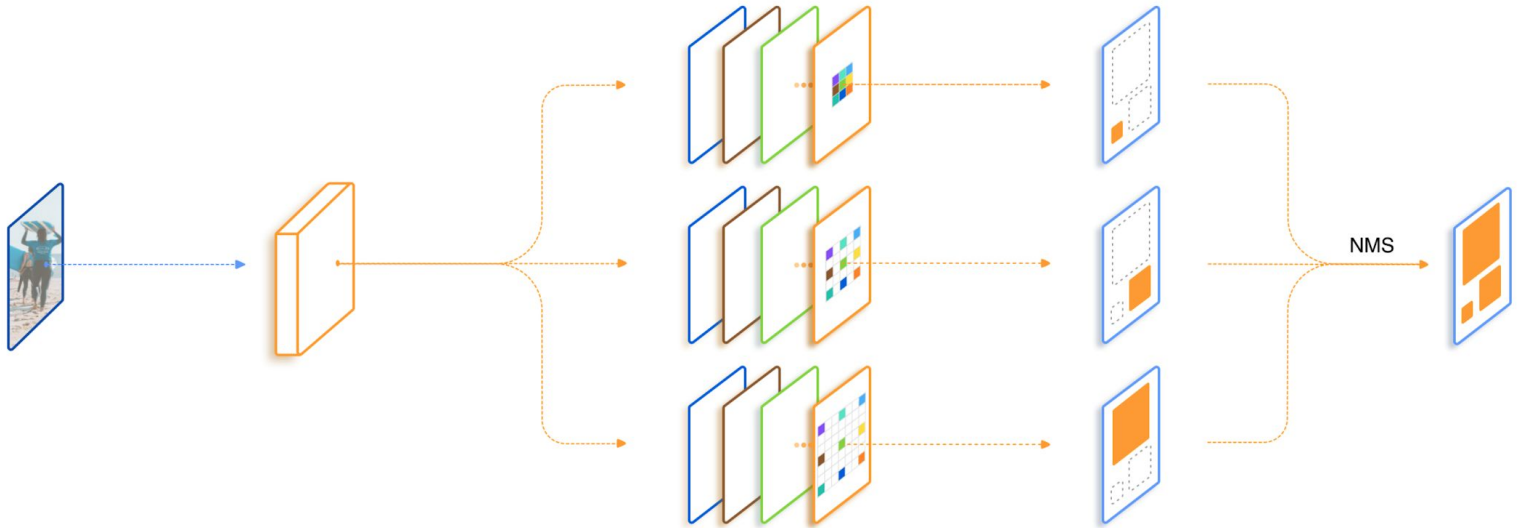


Figure 2: Illustration of the proposed TridentNet. The multiple branches in trident blocks share the same parameters with different dilation rates to generate scale-specific feature maps. Objects of specified scales are sampled for each branch during training. The final proposals or detections from multiple branches will be combined using Non-maximum Suppression (NMS). Here we only show the backbone network of TridentNet. The RPN and Fast R-CNN heads are shared among branches and ignored for simplicity.

TridentNet (2019)

Method	Backbone	AP	AP_{50}	AP_{75}	AP_s	AP_m	AP_l
Cascade R-CNN [5]	ResNet-101-FPN	42.8	62.1	46.3	23.7	45.5	55.2
DCNv2 [44]	ResNet-101-DeformableV2	46.0	67.9	50.8	27.8	49.1	59.5
DCR [9]	ResNet-101-FPN-Deformable	43.1	66.1	47.3	25.8	45.9	55.3
SNIP [38]	ResNet-101-Deformable	44.4	66.2	44.9	27.3	47.4	56.9
SNIPER [39]	ResNet-101-Deformable	46.1	67.0	51.6	29.6	48.9	58.1
TridentNet	ResNet-101	42.7	63.6	46.5	23.9	46.6	56.6
TridentNet*	ResNet-101-Deformable	46.8	67.6	51.5	28.0	51.2	**60.5**
TridentNet* + Image Pyramid	ResNet-101-Deformable	**48.4**	**69.7**	**53.5**	**31.8**	**51.3**	60.3

Table 7: Comparisons of single-model results for different object detection methods evaluated on the COCO *test-dev* set.

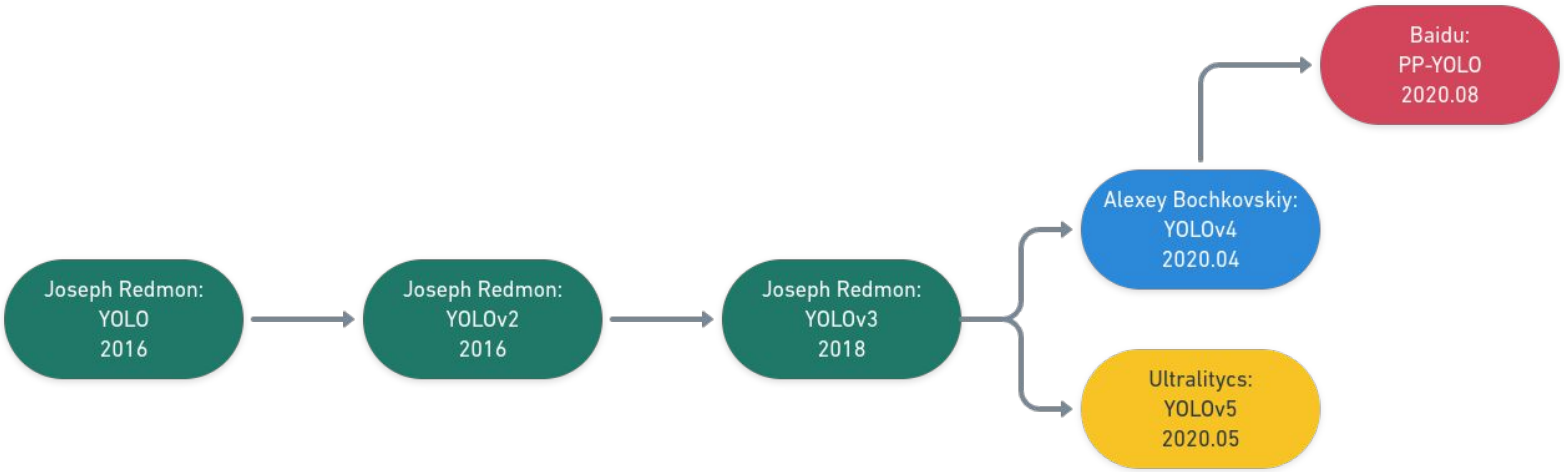
Detection Leaderboard

BBOX: Dev Std15 Chal15 Chal16 Chal17

SEGM: Dev Std15 Chal15 Chal16 Chal17 Chal18 Chal19 Chal20

Copy to Clipboard | Export to CSV | Search:

	AP	AP^{50}	AP^{75}	AP^{S}	AP^{M}	AP^{L}	AR^{1}	AR^{10}	AR^{100}	AR^{S}	AR^{M}	AR^{L}	date
Noah CV Lab (Huawei)	0.588	0.766	0.649	0.407	0.616	0.720	0.418	0.700	0.747	0.591	0.780	0.875	2020-06-11
mmdet	0.578	0.770	0.637	0.399	0.605	0.706	0.414	0.690	0.736	0.577	0.768	0.861	2019-10-04
DeepAR(ETRIxKAIST_AIM)	0.553	0.746	0.609	0.378	0.583	0.668	0.403	0.674	0.724	0.555	0.755	0.859	2019-10-04
DetectoRS	0.550	0.736	0.604	0.377	0.578	0.669	0.401	0.678	0.730	0.565	0.761	0.860	2020-05-28
KiwiDet2	0.547	0.728	0.597	0.362	0.576	0.685	0.401	0.680	0.733	0.552	0.768	0.878	2020-05-29
360 AI Research	0.546	0.737	0.600	0.369	0.583	0.674	0.398	0.665	0.716	0.534	0.749	0.862	2020-04-01


Joseph Redmon:
YOLO
2016
Joseph Redmon:
YOLOv2
2016
Joseph Redmon:
YOLOv3
2018
Alexey Bochkovskiy:
YOLOv4
2020.04
Ultralitycs:
YOLOv5
2020.05
Baidu:
PP-YOLO
2020.08

PP-YOLO (2020)

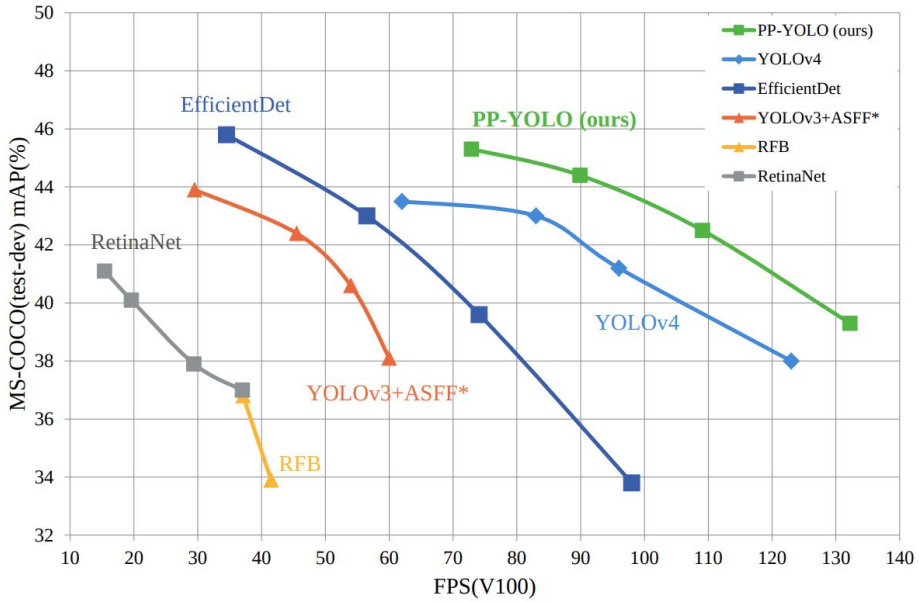


Figure 1. Comparison of the proposed PP-YOLO and other state-of-the-art object detectors. PP-YOLO runs faster than YOLOv4 and improves mAP from 43.5% to 45.2%.

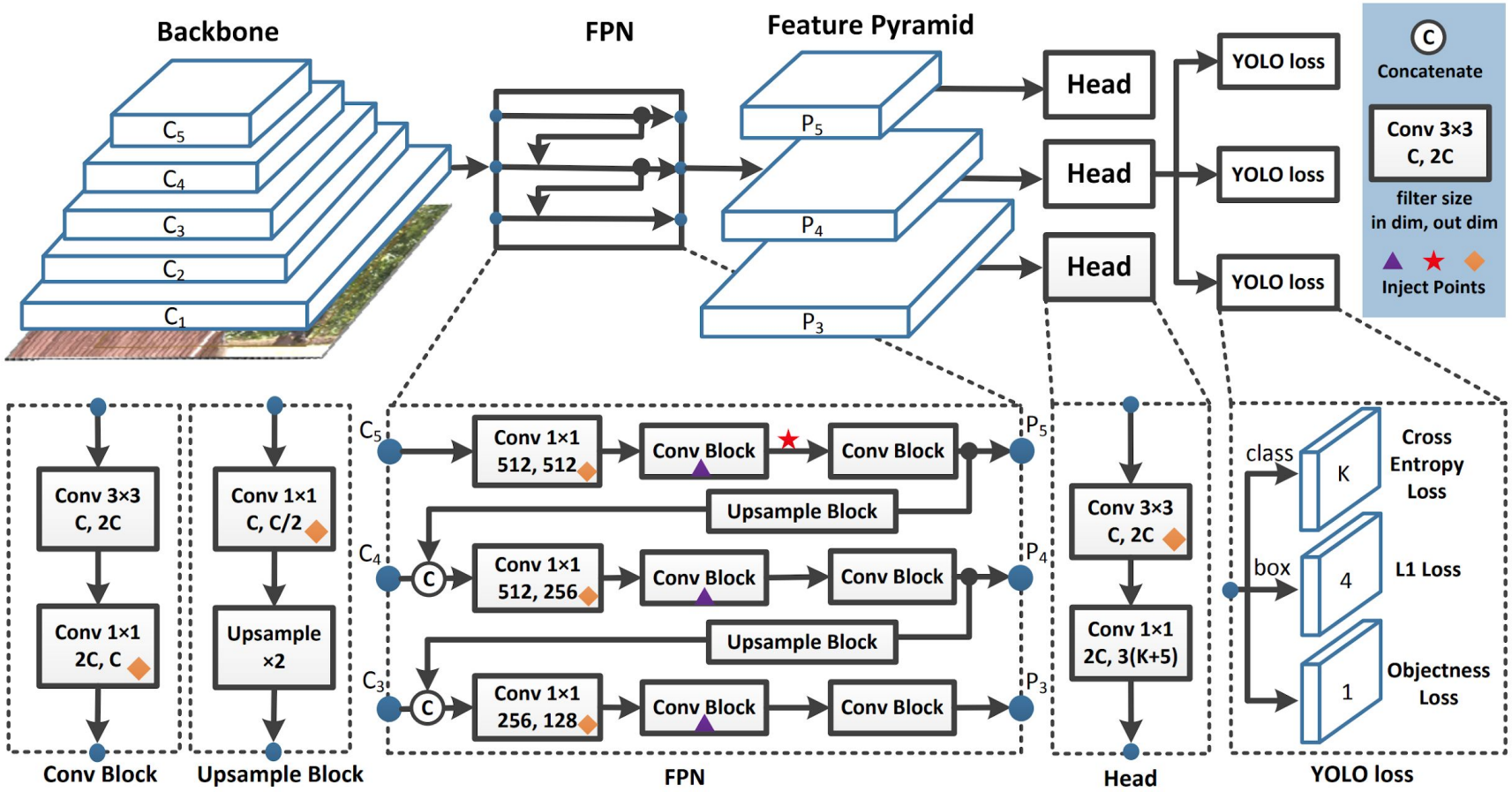
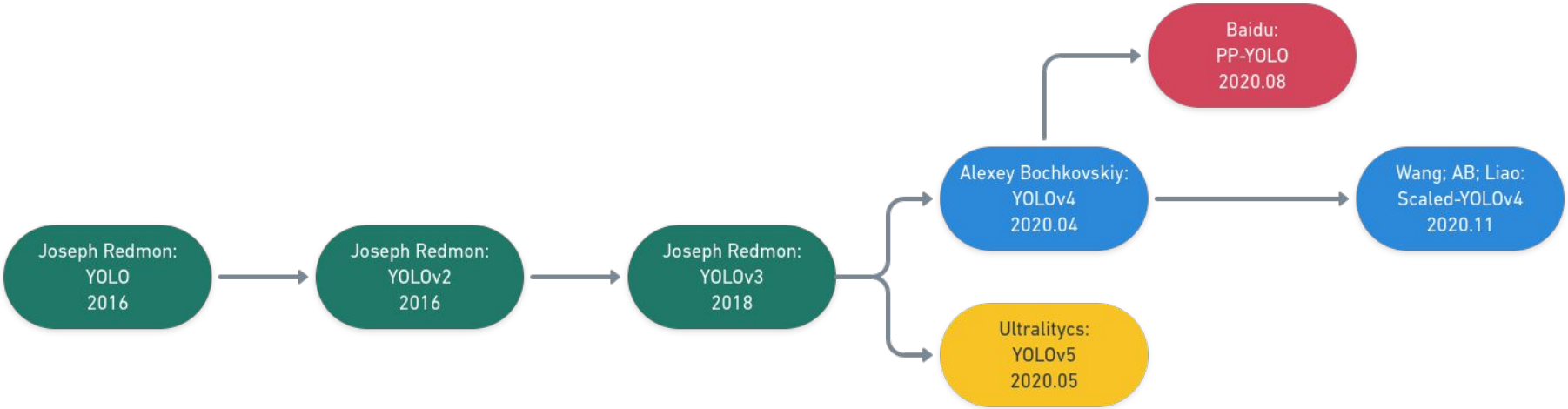


Figure 2. The network architecture of YOLOv3 and inject points for PP-YOLO. Activation layers are omitted for brevity. Details are described in Section 3.1 and Section 3.2.

Method	Backbone	Size	FPS (V100)		AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
			w/o TRT	with TRT						
RetinaNet [22]	ResNet-50	640	37	-	37.0%	-	-	-	-	-
RetinaNet [22]	ResNet-101	640	29.4	-	37.9%	-	-	-	-	-
RetinaNet [22]	ResNet-50	1024	19.6	-	40.1%	-	-	-	-	-
RetinaNet [22]	ResNet-101	1024	15.4	-	41.1%	-	-	-	-	-
EfficientDet-D0 [35]	Efficient-B0	512	98.0^+	-	33.8%	52.2%	35.8%	12.0%	38.3%	51.2%
EfficientDet-D1 [35]	Efficient-B1	640	74.1^+	-	39.6%	58.6%	42.3%	17.9%	44.3%	56.0%
EfficientDet-D2 [35]	Efficient-B2	768	56.5^+	-	43.0%	62.3%	46.2%	22.5%	47.0%	58.4%
EfficientDet-D2 [35]	Efficient-B3	896	34.5^+	-	45.8%	65.0%	49.3%	26.6%	49.4%	59.8%
RFBNet[3]	HarDNet68	512	41.5	-	33.9%	54.3%	36.2%	14.7%	36.6%	50.5%
RFBNet[3]	HarDNet85	512	37.1	-	36.8%	57.1%	39.5%	16.9%	40.5%	52.9%
YOLOv3 + ASFF* [26]	Darknet-53	320	60	-	38.1%	57.4%	42.1%	16.1%	41.6%	53.6%
YOLOv3 + ASFF* [26]	Darknet-53	416	54	-	40.6%	60.6%	45.1%	20.3%	44.2%	54.1%
YOLOv3 + ASFF* [26]	Darknet-53	608	45.5	-	42.4%	63.0%	47.4%	25.5%	45.7%	52.3%
YOLOv3 + ASFF* [26]	Darknet-53	800	29.4	-	43.9%	64.1%	49.2%	27.0%	46.6%	53.4%
YOLOv4 [1]	CSPDarknet-53	416	96	164.0^*	41.2%	62.8%	44.3%	20.4%	44.4%	56.0%
YOLOv4 [1]	CSPDarknet-53	512	83	138.4^*	43.0%	64.9%	46.5%	24.3%	46.1%	55.2%
YOLOv4 [1]	CSPDarknet-53	608	62	105.5^*	43.5%	65.7%	47.3%	26.7%	46.7%	53.3%
PP-YOLO	ResNet50-vd-dcn	320	132.2	242.2	39.3%	59.3%	42.7%	16.7%	41.4%	57.8%
PP-YOLO	ResNet50-vd-dcn	416	109.6	215.4	42.5%	62.8%	46.5%	21.2%	45.2%	58.2%
PP-YOLO	ResNet50-vd-dcn	512	89.9	188.4	44.4%	64.6%	48.8%	24.4%	47.1%	58.2%
PP-YOLO	ResNet50-vd-dcn	608	72.9	155.6	45.2%	65.2%	49.9%	26.3%	47.8%	57.2%

Table 2. Comparison of the speed and accuracy of different object detectors on the MS-COCO (test-dev 2017). We compare the results with batch size = 1, without tensorRT (w/o TRT) or with tensorRT(with TRT). Results marked by "+" are updated results from the corresponding official code base, which are higher than the results in original paper. Results marked by "*" are test in our environment using official code and model, which are slightly higher than results reported in official code-base.


Joseph Redmon:
YOLO
2016
Joseph Redmon:
YOLOv2
2016
Joseph Redmon:
YOLOv3
2018
Alexey Bochkovskiy:
YOLOv4
2020.04
Baidu:
PP-YOLO
2020.08
Wang; AB; Liao:
Scaled-YOLOv4
2020.11
Ultralitycs:
YOLOv5
2020.05

Scaled YOLOv4

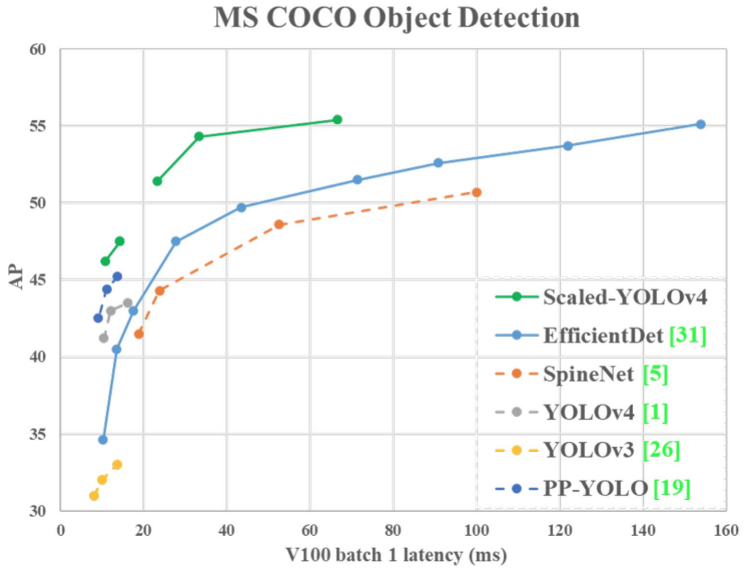


Figure 1: Comparison of the proposed YOLOv4 and other state-of-the-art object detectors. The dashed line means only latency of model inference, while the solid line include model inference and post-processing.

Scaled YOLOv4

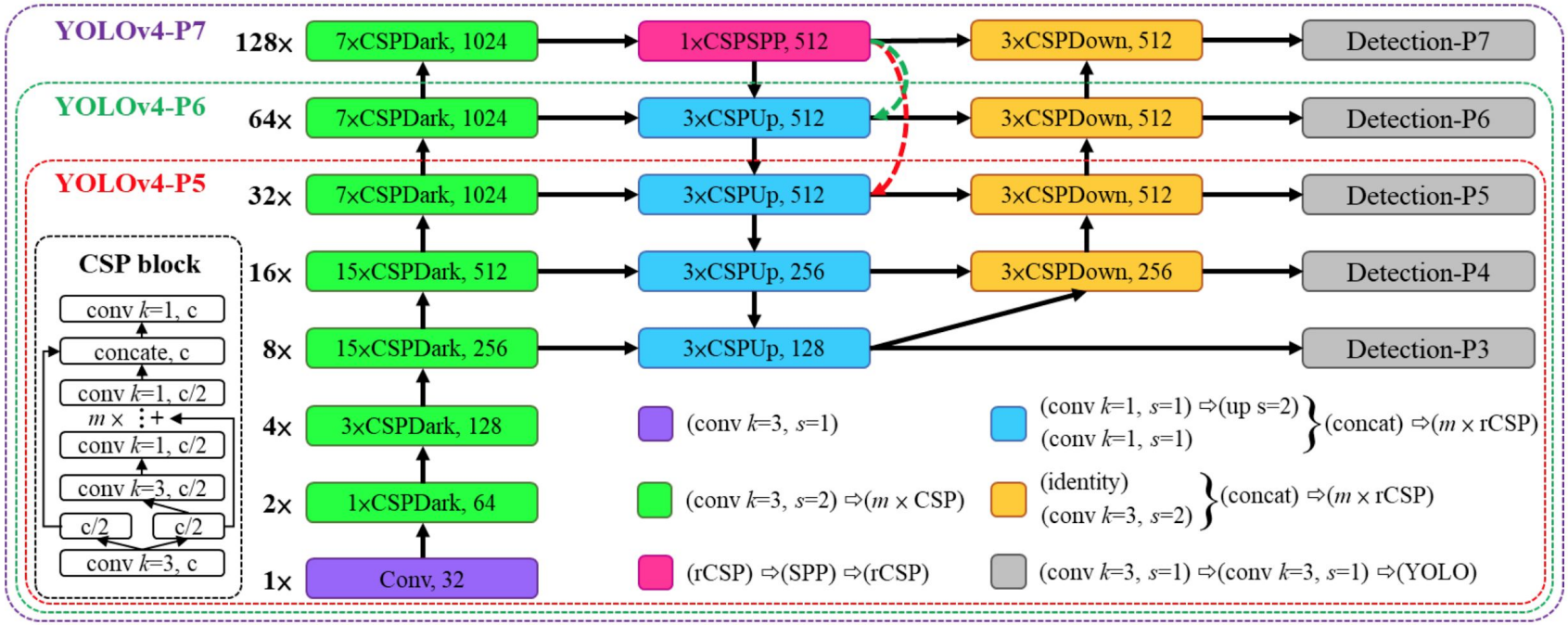
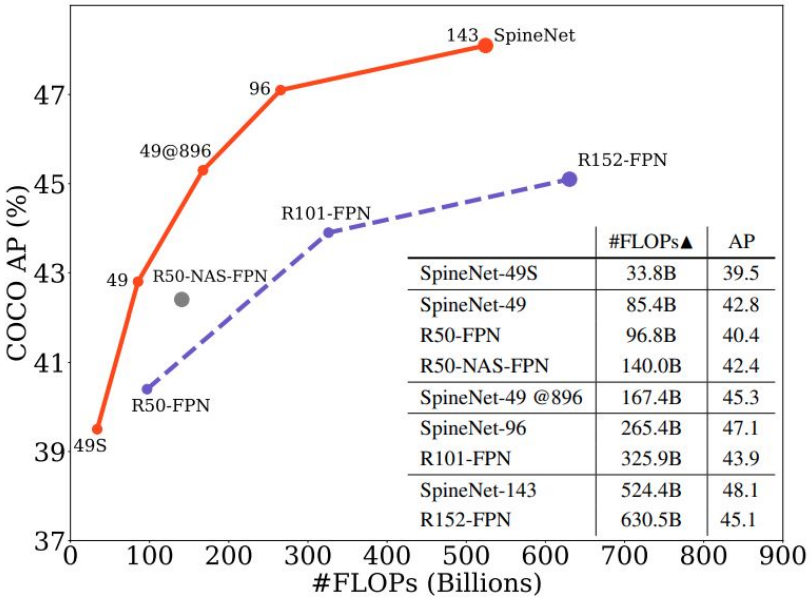


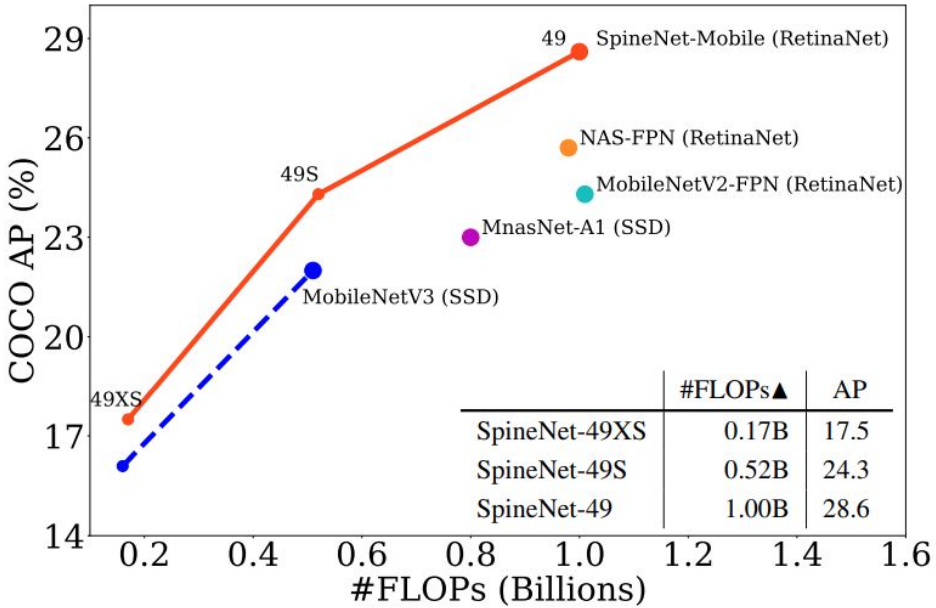
Figure 4: Architecture of YOLOv4-large, including YOLOv4-P5, YOLOv4-P6, and YOLOv4-P7. The dashed arrow means replace the corresponding CSPUp block by CSPSPP block.

SpineNet (2020)



	#FLOPs▲	AP
SpineNet-49S	33.8B	39.5
SpineNet-49	85.4B	42.8
R50-FPN	96.8B	40.4
R50-NAS-FPN	140.0B	42.4
SpineNet-49 @896	167.4B	45.3
SpineNet-96	265.4B	47.1
R101-FPN	325.9B	43.9
SpineNet-143	524.4B	48.1
R152-FPN	630.5B	45.1

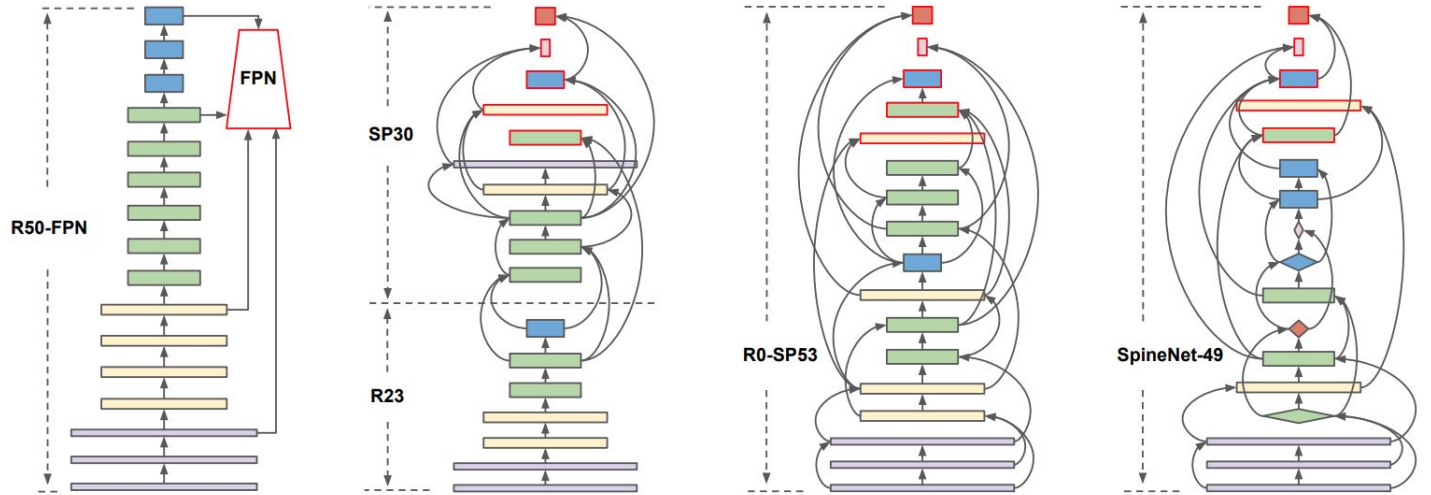
Figure 1: The comparison of RetinaNet models adopting SpineNet, ResNet-FPN, and NAS-FPN backbones. Details of training setup is described in Section 5 and controlled experiments can be found in Table 2, 3.



	#FLOPs▲	AP
SpineNet-49XS	0.17B	17.5
SpineNet-49S	0.52B	24.3
SpineNet-49	1.00B	28.6

Figure 2: A comparison of mobile-size SpineNet models and other prior art of detectors for mobile-size object detection. Details are in Table 9.

SpineNet (2020)



(a) R50-FPN @37.8% AP (b) R23-SP30 @39.6% AP (c) R0-SP53 @40.7% AP (d) SpineNet-49 @40.8% AP

Figure 4: **Building scale-permuted network by permuting ResNet.** From (a) to (d), the computation gradually shifts from ResNet-FPN to scale-permuted networks. (a) The R50-FPN model, spending most computation in ResNet-50 followed by a FPN, achieves 37.8% AP; (b) R23-SP30, investing 7 blocks in a ResNet and 10 blocks in a scale-permuted network, achieves 39.6% AP; (c) R0-SP53, investing all blocks in a scale-permuted network, achieves 40.7% AP; (d) The SpineNet-49 architecture achieves 40.8% AP with 10% fewer FLOPs (85.4B *vs*. 95.2B) by learning additional block adjustments. Rectangle block represent bottleneck block and diamond block represent residual block. Output blocks are indicated by red border.

backbone model	resolution	#FLOPs▲	#Params	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
SpineNet-49S	640×640	**33.8B**	**11.9M**	**39.5**	**59.3**	**43.1**	**20.9**	**42.2**	**54.3**
SpineNet-49	640×640	**85.4B**	**28.5M**	**42.8**	**62.3**	**46.1**	**23.7**	**45.2**	**57.3**
R50-FPN	640×640	96.8B	34.0M	40.4	59.9	43.6	22.7	43.5	57.0
R50-NAS-FPN	640×640	140.0B	60.3M	42.4	61.8	46.1	25.1	46.7	57.8
SpineNet-49	896×896	**167.4B**	**28.5M**	**45.3**	**65.1**	**49.1**	**27.0**	**47.9**	**57.7**
SpineNet-96	1024×1024	**265.4B**	**43.0M**	**47.1**	**67.1**	**51.1**	**29.1**	**50.2**	**59.0**
R101-FPN	1024×1024	325.9B	53.1M	43.9	63.6	47.6	26.8	47.6	57.0
SpineNet-143	1280×1280	**524.4B**	**66.9M**	**48.1**	**67.6**	**52.0**	**30.2**	**51.1**	**59.9**
R152-FPN	1280×1280	630.5B	68.7M	45.1	64.6	48.7	28.4	48.8	58.2
R50-FPN†	640×640	96.8B	34.0M	42.3	61.9	45.9	23.9	46.1	58.5
SpineNet-49S†	640×640	**33.8B**	**12.0M**	**41.5**	**60.5**	**44.6**	**23.3**	**45.0**	**58.0**
SpineNet-49†	640×640	**85.4B**	**28.5M**	**44.3**	**63.8**	**47.6**	**25.9**	**47.7**	**61.1**
SpineNet-49†	896×896	**167.4B**	**28.5M**	**46.7**	**66.3**	**50.6**	**29.1**	**50.1**	**61.7**
SpineNet-96†	1024×1024	**265.4B**	**43.0M**	**48.6**	**68.4**	**52.5**	**32.0**	**52.3**	**62.0**
SpineNet-143†	1280×1280	**524.4B**	**66.9M**	**50.7**	**70.4**	**54.9**	**33.6**	**53.9**	**62.1**
SpineNet-190†	1280×1280	**1885.0B**	**163.6M**	**52.1**	**71.8**	**56.5**	**35.4**	**55.0**	**63.6**

Table 2: **One-stage object detection results on COCO `test-dev`.** We compare employing different backbones with RetinaNet on single model without test-time augmentation. By default we apply protocol B with multi-scale training and ReLU activation to train all models in this table, as described in Section 5.1. Models marked by dagger (†) are trained with protocol C by applying stochastic depth and swish activation for a longer training schedule. FLOPs is represented by Multi-Adds.

Gotowe modele, frameworki

- TensorFlow Object Detection API (TF1, TF2)
- Darknet YOLOv4 (C++)
  - BMW-YOLOv4-Training-Automation
- Facebook Detectron2 (PyTorch)
- Facebook DETR (PyTorch)
- Tencent NCNN

TensorFlow 1 Object Detection API - Model ZOO

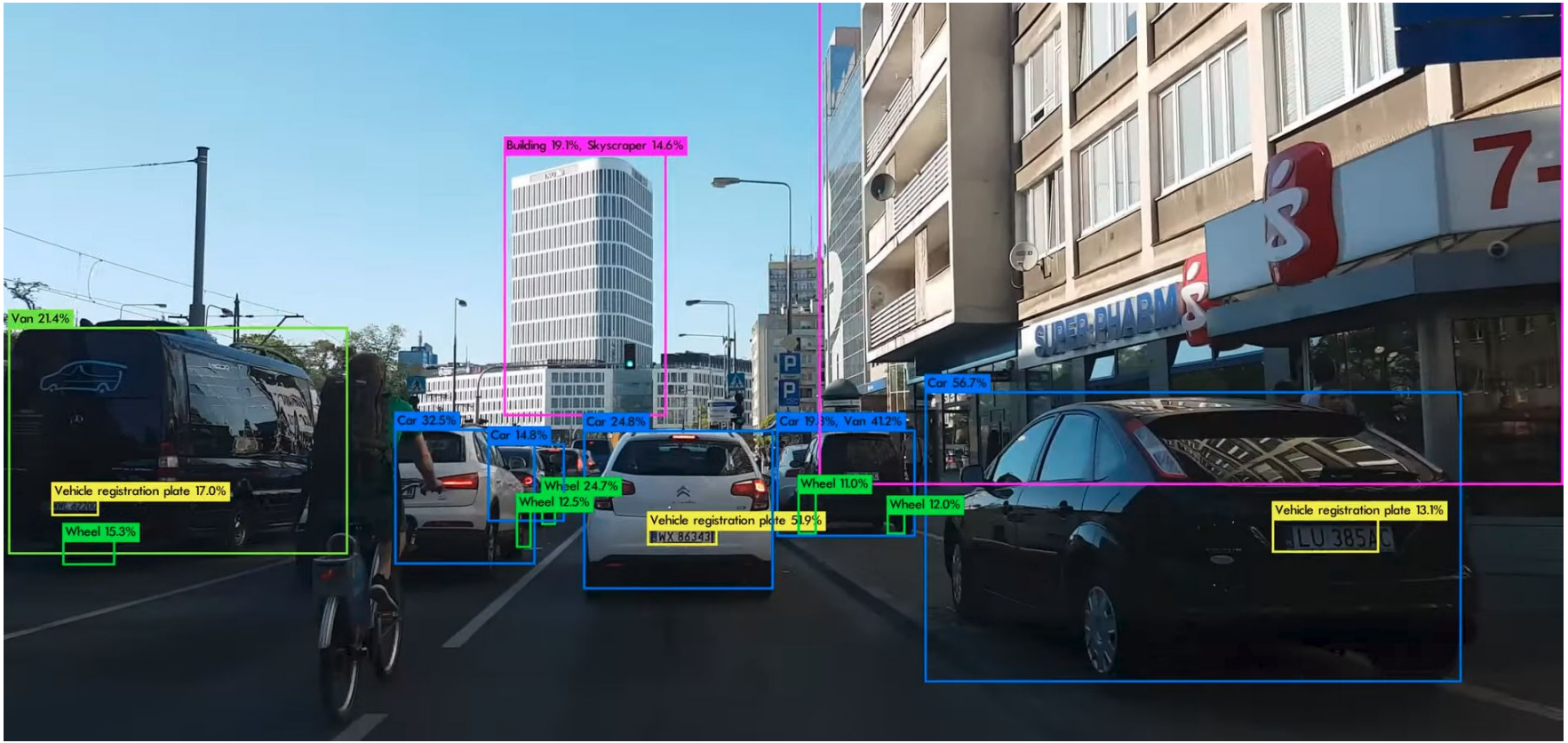
COCO-trained models

Model name	Speed (ms)	COCO mAP[^1]	Outputs
ssd_mobilenet_v1_coco	30	21	Boxes
ssd_mobilenet_v1_0.75_depth_coco ☆	26	18	Boxes
ssd_mobilenet_v1_quantized_coco ☆	29	18	Boxes
ssd_mobilenet_v1_0.75_depth_quantized_coco ☆	29	16	Boxes
ssd_mobilenet_v1_ppn_coco ☆	26	20	Boxes
ssd_mobilenet_v1_fpn_coco ☆	56	32	Boxes
ssd_resnet_50_fpn_coco ☆	76	35	Boxes
ssd_mobilenet_v2_coco	31	22	Boxes
ssd_mobilenet_v2_quantized_coco	29	22	Boxes
ssdlite_mobilenet_v2_coco	27	22	Boxes
ssd_inception_v2_coco	42	24	Boxes
faster_rcnn_inception_v2_coco	58	28	Boxes
faster_rcnn_resnet50_coco	89	30	Boxes
faster_rcnn_resnet50_lowproposals_coco	64		Boxes

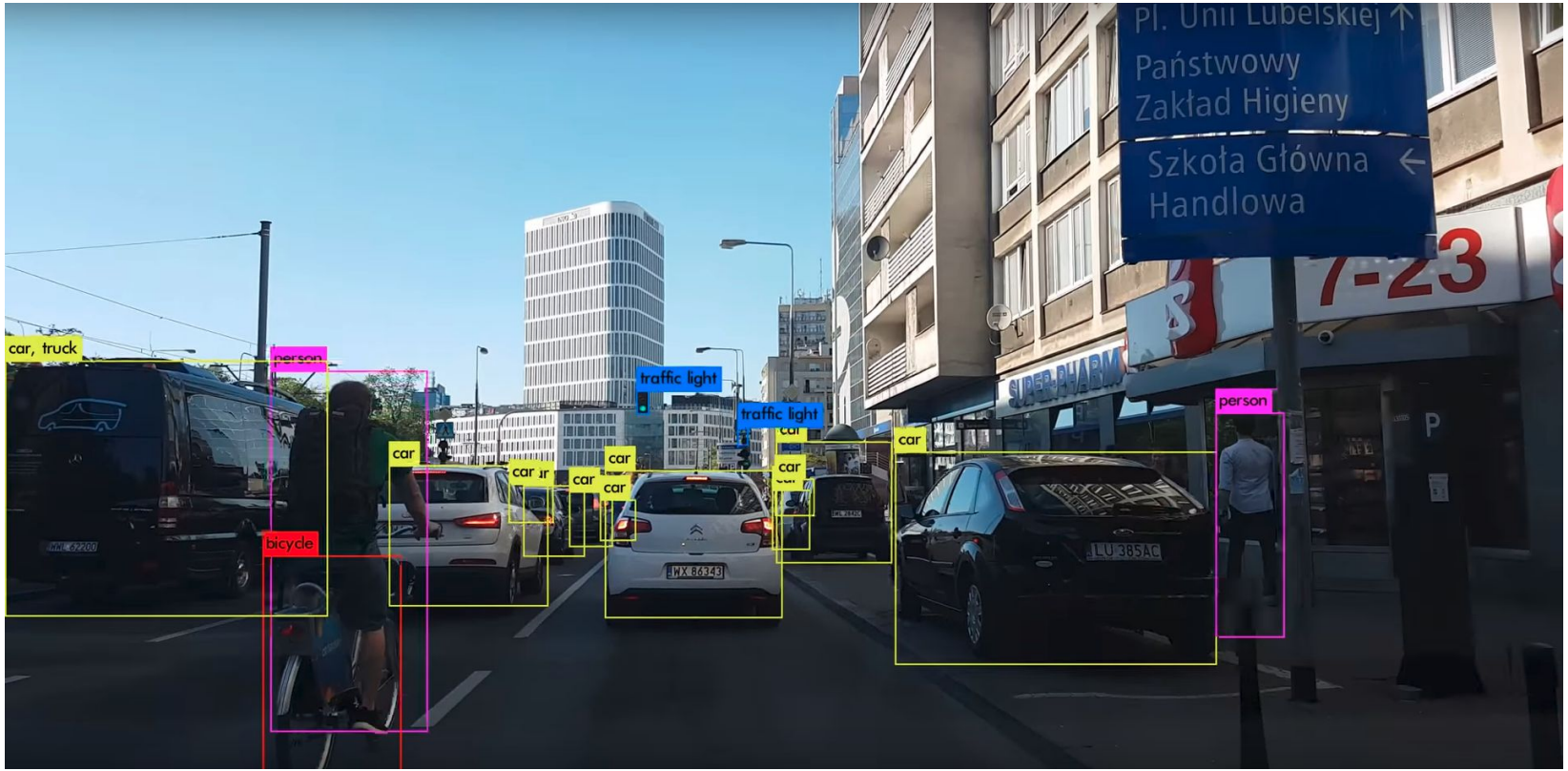
TensorFlow 2 Object Detection API - Model ZOO

Model name	Speed (ms)	COCO mAP	Outputs
CenterNet HourGlass104 512x512	70	41.9	Boxes
CenterNet HourGlass104 Keypoints 512x512	76	40.0/61.4	Boxes/Keypoints
CenterNet HourGlass104 1024x1024	197	44.5	Boxes
CenterNet HourGlass104 Keypoints 1024x1024	211	42.8/64.5	Boxes/Keypoints
CenterNet Resnet50 V1 FPN 512x512	27	31.2	Boxes
CenterNet Resnet50 V1 FPN Keypoints 512x512	30	29.3/50.7	Boxes/Keypoints
CenterNet Resnet101 V1 FPN 512x512	34	34.2	Boxes
CenterNet Resnet50 V2 512x512	27	29.5	Boxes
CenterNet Resnet50 V2 Keypoints 512x512	30	27.6/48.2	Boxes/Keypoints
EfficientDet D0 512x512	39	33.6	Boxes
EfficientDet D1 640x640	54	38.4	Boxes
EfficientDet D2 768x768	67	41.8	Boxes
EfficientDet D3 896x896	95	45.4	Boxes
EfficientDet D4 1024x1024	133	48.5	Boxes
EfficientDet D5 1280x1280	222	49.7	Boxes
EfficientDet D6 1280x1280	268	50.5	Boxes
EfficientDet D7 1536x1536	325	51.2	Boxes
SSD MobileNet v2 320x320	19	20.2	Boxes
SSD MobileNet V1 FPN 640x640	48	29.1	Boxes
SSD MobileNet V2 FPNLite 320x320	22	22.2	Boxes
SSD MobileNet V2 FPNLite 640x640	39	28.2	Boxes
SSD ResNet50 V1 FPN 640x640 (RetinaNet50)	46	34.3	Boxes
SSD ResNet50 V1 FPN 1024x1024 (RetinaNet50)	87	38.3	Boxes
SSD ResNet101 V1 FPN 640x640 (RetinaNet101)	57	35.6	Boxes
SSD ResNet101 V1 FPN 1024x1024 (RetinaNet101)	104	39.5	Boxes
SSD ResNet152 V1 FPN 640x640 (RetinaNet152)	80	35.4	Boxes
SSD ResNet152 V1 FPN 1024x1024 (RetinaNet152)	111	39.6	Boxes
Faster R-CNN ResNet50 V1 640x640	53	29.3	Boxes
Faster R-CNN ResNet50 V1 1024x1024	65	31.0	Boxes
Faster R-CNN ResNet50 V1 800x1333	65	31.6	Boxes
Faster R-CNN ResNet101 V1 640x640	55	31.8	Boxes
Faster R-CNN ResNet101 V1 1024x1024	72	37.1	Boxes
Faster R-CNN ResNet101 V1 800x1333	77	36.6	Boxes
Faster R-CNN ResNet152 V1 640x640	64	32.4	Boxes
Faster R-CNN ResNet152 V1 1024x1024	85	37.6	Boxes
Faster R-CNN ResNet152 V1 800x1333	101	37.4	Boxes
Faster R-CNN Inception ResNet V2 640x640	206	37.7	Boxes
Faster R-CNN Inception ResNet V2 1024x1024	236	38.7	Boxes
Mask R-CNN Inception ResNet V2 1024x1024	301	39.0/34.6	Boxes/Masks
ExtremeNet	--	--	Boxes

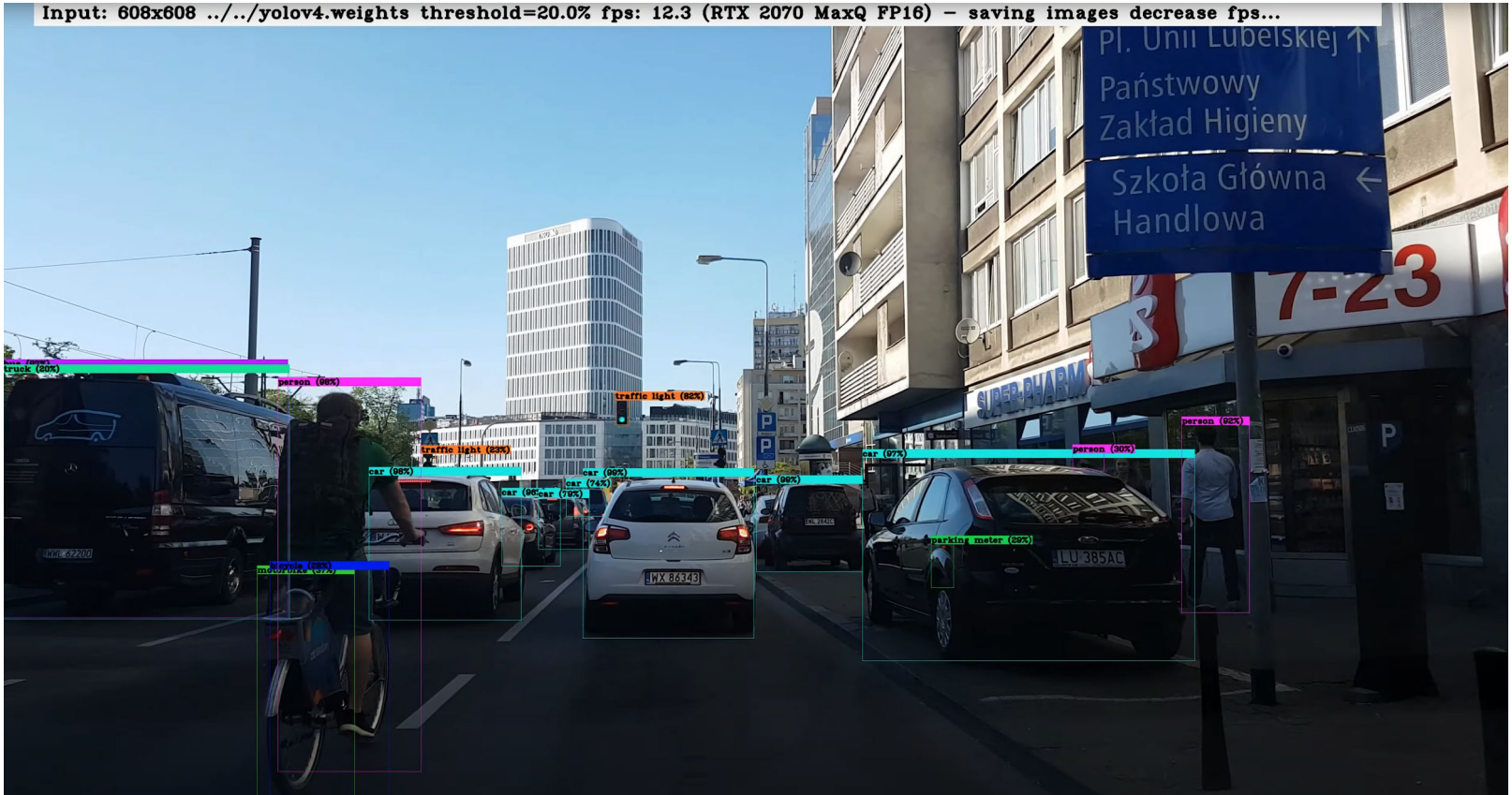
Darknet YOLOv3 (Open Images)



Darknet YOLOv3 (COCO)



Darknet YOLOv4 (COCO)



Facebook Detectron2

Tencent NCNN

VGG AlexNet GoogleNet Inception
ResNet DenseNet SENet FPN
SqueezeNet MobileNetV1/V2/V3
ShuffleNetV1/V2 MNasNet
MTCNN, RetinaFace
VGG-SSD, MobileNet-SSD,
SqueezeNet-SSD
MobileNetV2,3-SSDLite
Faster-RCNN R-FCN
YOLOV2 YOLOV3 YOLOV4
FCN PSPNet UNet YOLACT
SimplePose

Current building status matrix

System	CPU (32bit)	CPU (64bit)	GPU (32bit)	GPU (64bit)
Linux (GCC)	build passing	build passing	—	build passing
Linux (Clang)	build passing	build passing	—	build passing
Linux (ARM)	build passing	build passing	—	—
Linux (MIPS)	build passing	build passing	—	—
Linux (RISC-V)	—	build passing	—	—
Windows (VS2017)	—	build passed	—	build unknown
Windows (VS2019)	build passing	build passing	—	build passing
MacOS	—	build passing	—	build passing
Android	build passing	build passing	build passing	build passing
Android-x86	build passing	build passing	build passing	build passing
iOS	build passing	build passing	—	build passing
iOS Simulator	build passing	build passing	—	—
WebAssembly	—	build passing	—	—
RISC-V GCC/Newlib	build passing	build passing	—	—

tkDNN

[Research only]

Platform	Network	FP32, B=1	FP32, B=4	FP16, B=1	FP16, B=4	INT8, B=1	INT8, B=4
RTX 2080Ti	yolo4 320	118.59	237.31	207.81	443.32	262.37	530.93
RTX 2080Ti	yolo4 416	104.81	162.86	169.06	293.78	206.93	353.26
RTX 2080Ti	yolo4 512	92.98	132.43	140.36	215.17	165.35	254.96
RTX 2080Ti	yolo4 608	63.77	81.53	111.39	152.89	127.79	184.72
AGX Xavier	yolo4 320	26.78	32.05	57.14	79.05	73.15	97.56
AGX Xavier	yolo4 416	19.96	21.52	41.01	49.00	50.81	60.61
AGX Xavier	yolo4 512	16.58	16.98	31.12	33.84	37.82	41.28
AGX Xavier	yolo4 608	9.45	10.13	21.92	23.36	27.05	28.93

Dziękuję za uwagę!