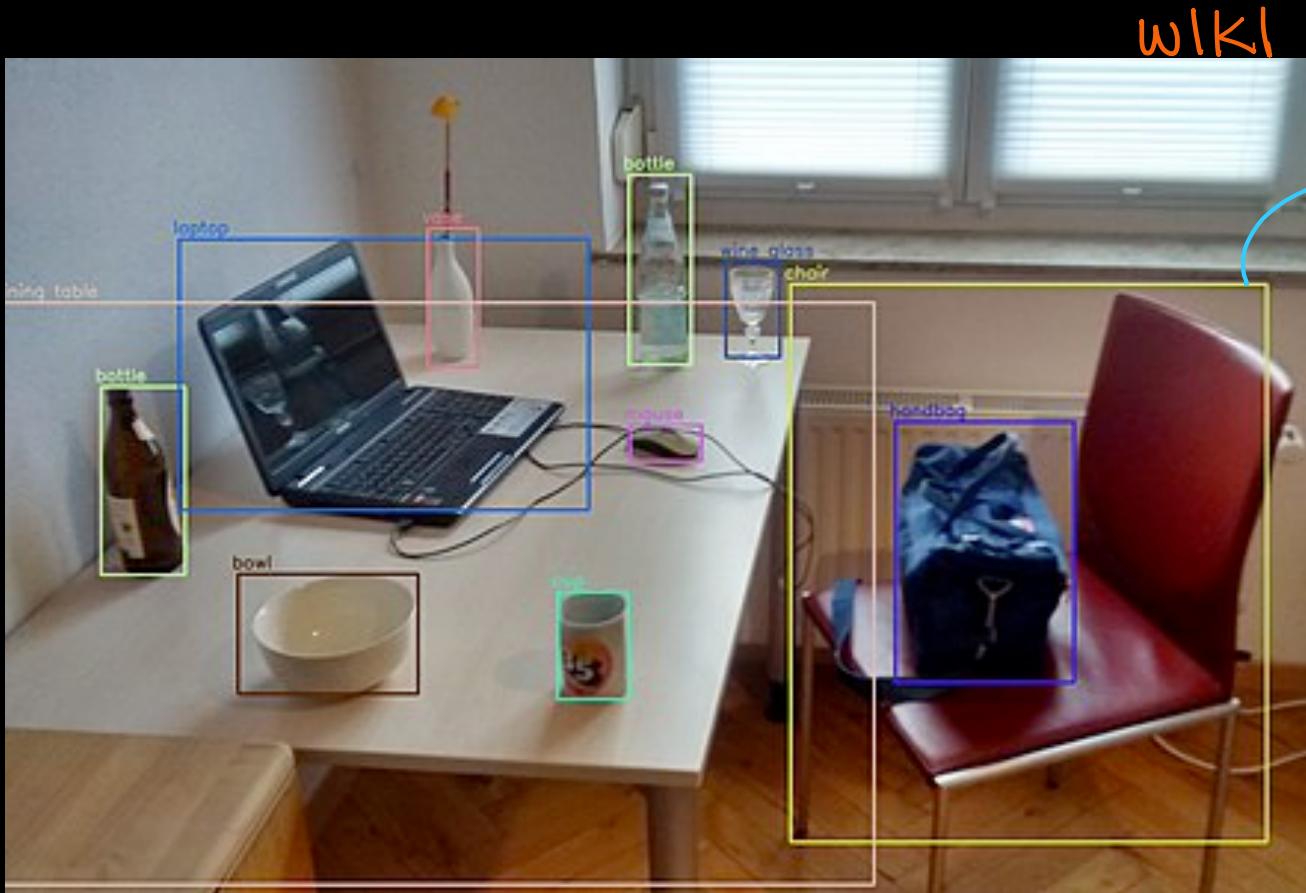


Object-detection

Agenda

1. Problem formulation
2. Azure API-based (+code)
3. Overview of Deep-Learning based models
4. YOLO v3 dive - deep
 - a. Theory
 - b. C-code

Problem-formulation:



WIKI

bounding boxes
(NOT pixels)

multiple-objects in single image

Using Azure-API

- Pre-req: Using Web-APIs in Python

Overview:

<https://docs.microsoft.com/en-us/azure/cognitive-services/computer-vision/concept-object-detection>

Documentation:

<https://westcentralus.dev.cognitive.microsoft.com/docs/services/5adf991815e1060e6355ad44/operations/56f91f2e778daf14a499e1fa>

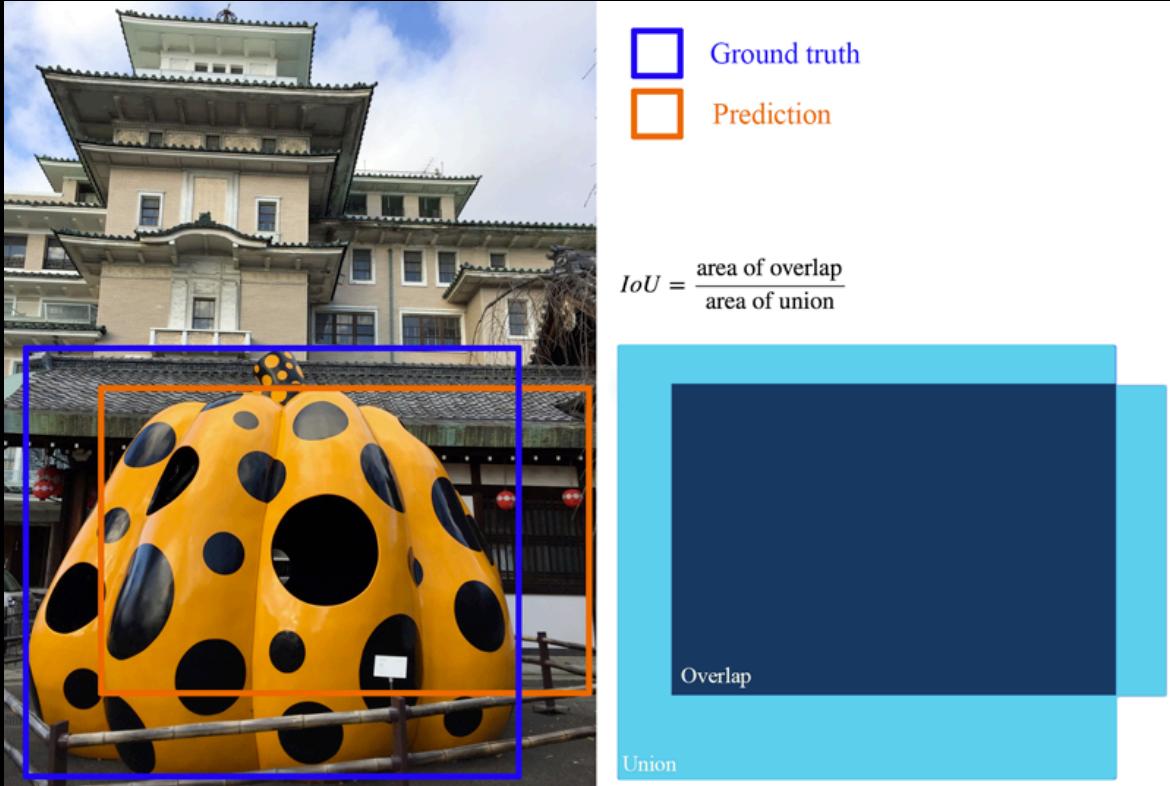
Python code:

<https://docs.microsoft.com/en-us/azure/cognitive-services/computer-vision/quickstarts/python-analyze>

```
print(json.dumps(response.json()))
```

(no 'objects' in
visualFeatures)

Performance - Metric :



https://medium.com/@jonathan_hui/map-mean-average-precision-for-object-detection-45c121a31173

Prediction is
correct if

$IoU \geq 0.5$

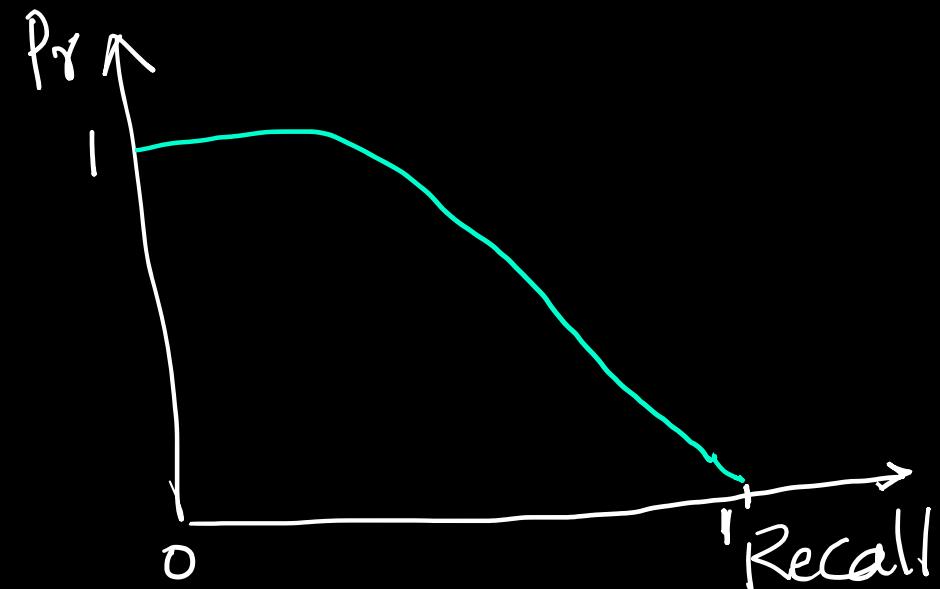
Performance - Metric :

Now, for each class (chair, person, ...),

compute avg-precision



AU-PR Curve



Performance - Metric :

mean - avg - precision (mAP) → NOT Max - A - posterior
(MAP)



mean across all classes

Deep-Learning - Models:

input : image / video

output : bounding-box-1 , object-class-1
 bounding-box-2 , object-class-2
 :
 :

Dataset : COCO <http://cocodataset.org/#home> [80-class-data]

Trade off: Speed VS mAP

→ medical diagnosis

↙
self-driving cars

real-time face-detection

R-CNN

fast R-CNN

faster R-CNN

SSD

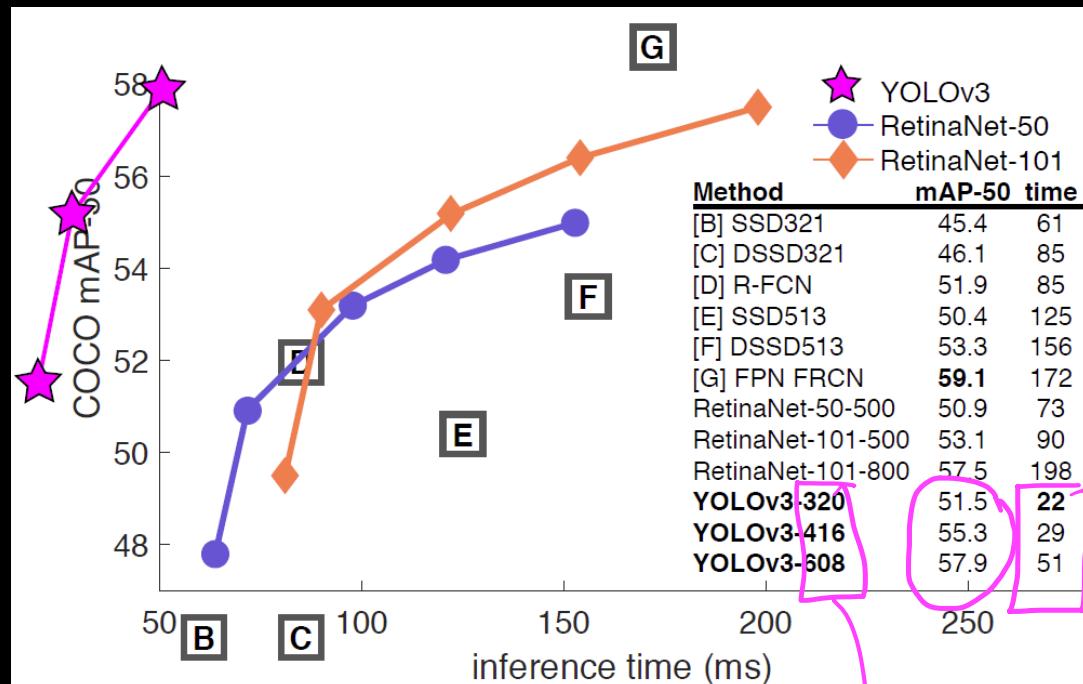
RetinaNet

YOLO V1

YOLO V2 / YOLO 9000

YOLO V3

APR-2018



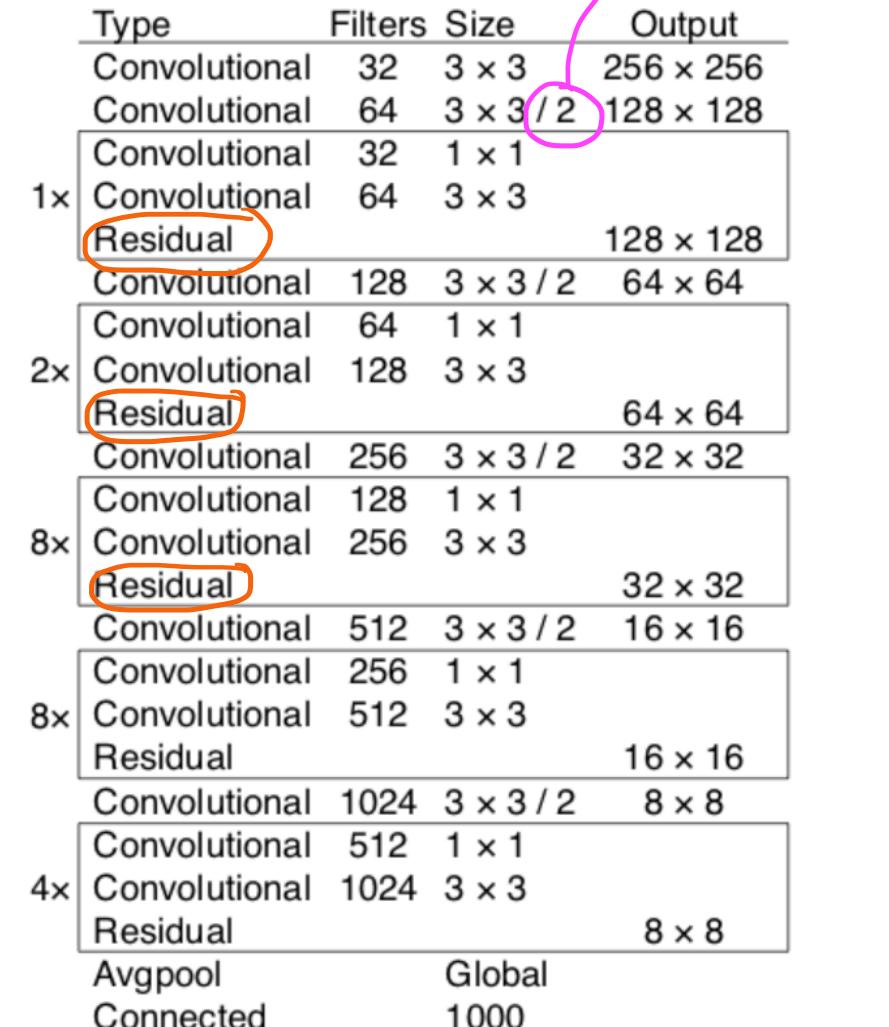
input - image size ↗

Super-fast + v-good mAP

YOLO - V3: feature-extractor

①

- fully convolutional Network (FCN)
- CONV - BatchNorm - LeakyReLU
- CONV with stride - downsample
- NO pooling
- pre-trained model

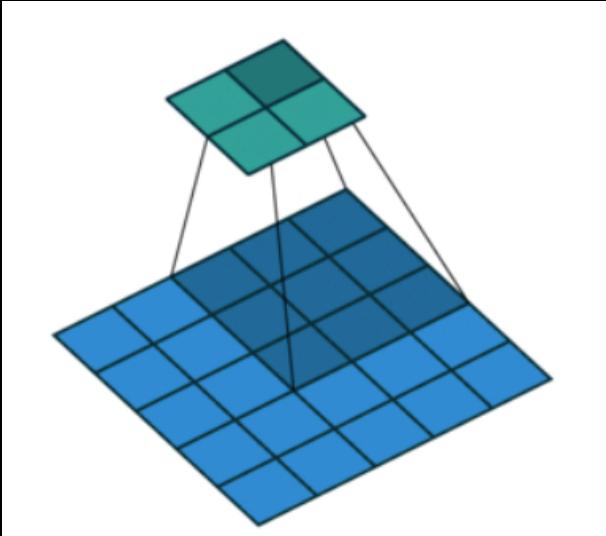
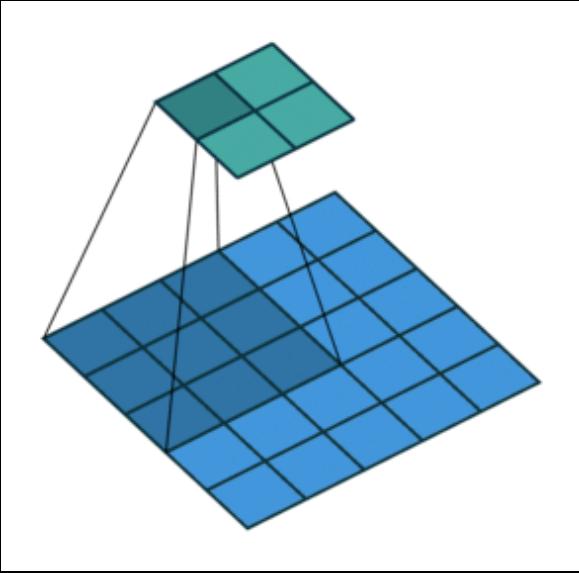


The diagram illustrates the YOLO-v3 feature-extractor architecture. It shows a sequence of layers grouped by their stride. Residual connections are indicated by orange circles around the 'Residual' labels. Stride-2 convolutions are highlighted with pink circles around the '3 x 3 / 2' entries in the 'Size' column.

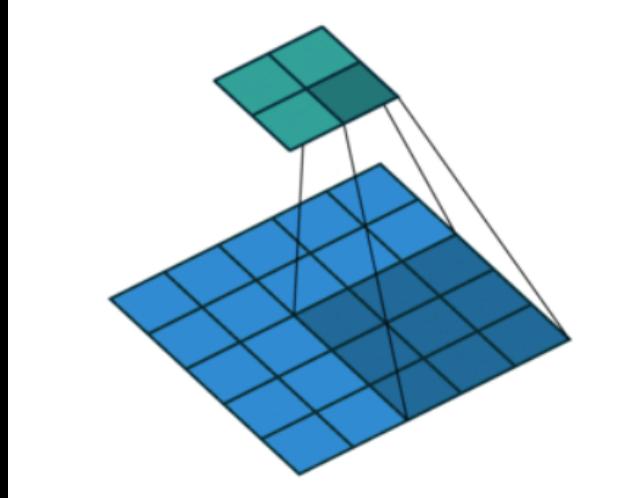
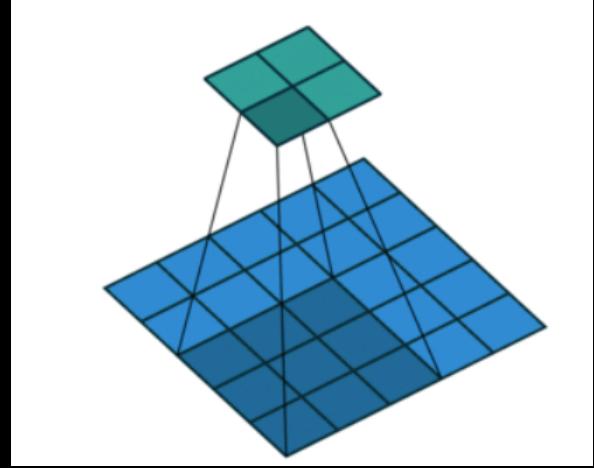
	Type	Filters	Size	Output
	Convolutional	32	3 x 3	256 x 256
	Convolutional	64	3 x 3 / 2	128 x 128
1x	Convolutional	32	1 x 1	
	Convolutional	64	3 x 3	
	Residual			128 x 128
	Convolutional	128	3 x 3 / 2	64 x 64
	Convolutional	64	1 x 1	
2x	Convolutional	128	3 x 3	
	Residual			64 x 64
	Convolutional	256	3 x 3 / 2	32 x 32
	Convolutional	128	1 x 1	
8x	Convolutional	256	3 x 3	
	Residual			32 x 32
	Convolutional	512	3 x 3 / 2	16 x 16
	Convolutional	256	1 x 1	
8x	Convolutional	512	3 x 3	
	Residual			16 x 16
	Convolutional	1024	3 x 3 / 2	8 x 8
	Convolutional	512	1 x 1	
4x	Convolutional	1024	3 x 3	
	Residual			8 x 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Darknet-53 model

<https://pjreddie.com/media/files/papers/YOLOv3.pdf>



Stride - 2
convolutions



5x5 $\xrightarrow{3 \times 3 \text{ kernels}} 2 \times 2$

https://github.com/vdumoulin/conv_arithmetic

a.k.a \rightarrow feature-extractor

Used
in
YOLOv2

Backbone	Top-1	Top-5	Bn Ops	BFLOP/s	FPS
Darknet-19 [15]	74.1	91.8	7.29	1246	171
ResNet-101[5]	77.1	93.7	19.7	1039	53
ResNet-152 [5]	77.6	93.8	29.4	1090	37
Darknet-53	77.2	93.8	18.7	1457	78 \rightarrow V.good

Slowest (Darknet-19)
More Ops/sec

input - Images :

416 × 416 × 3

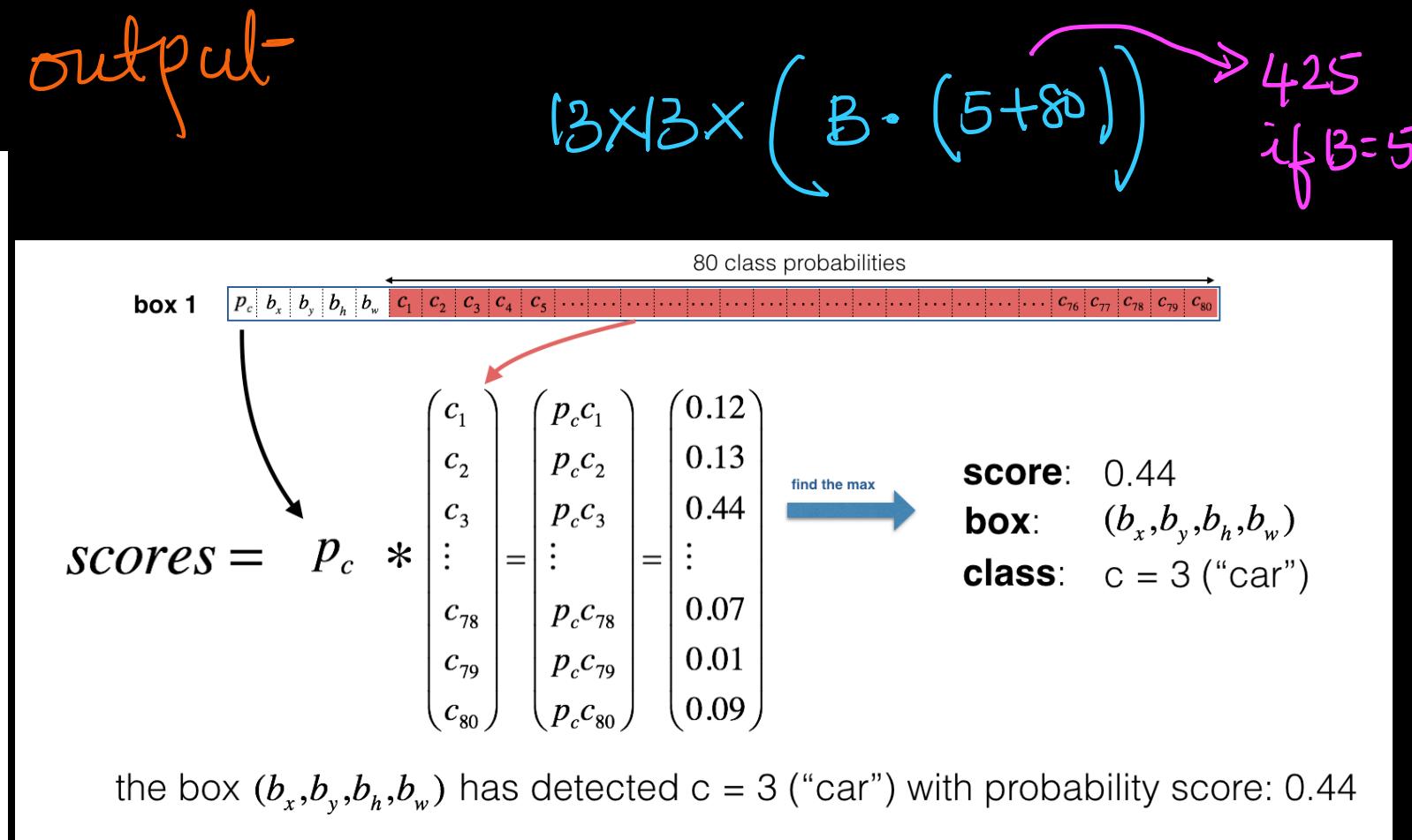
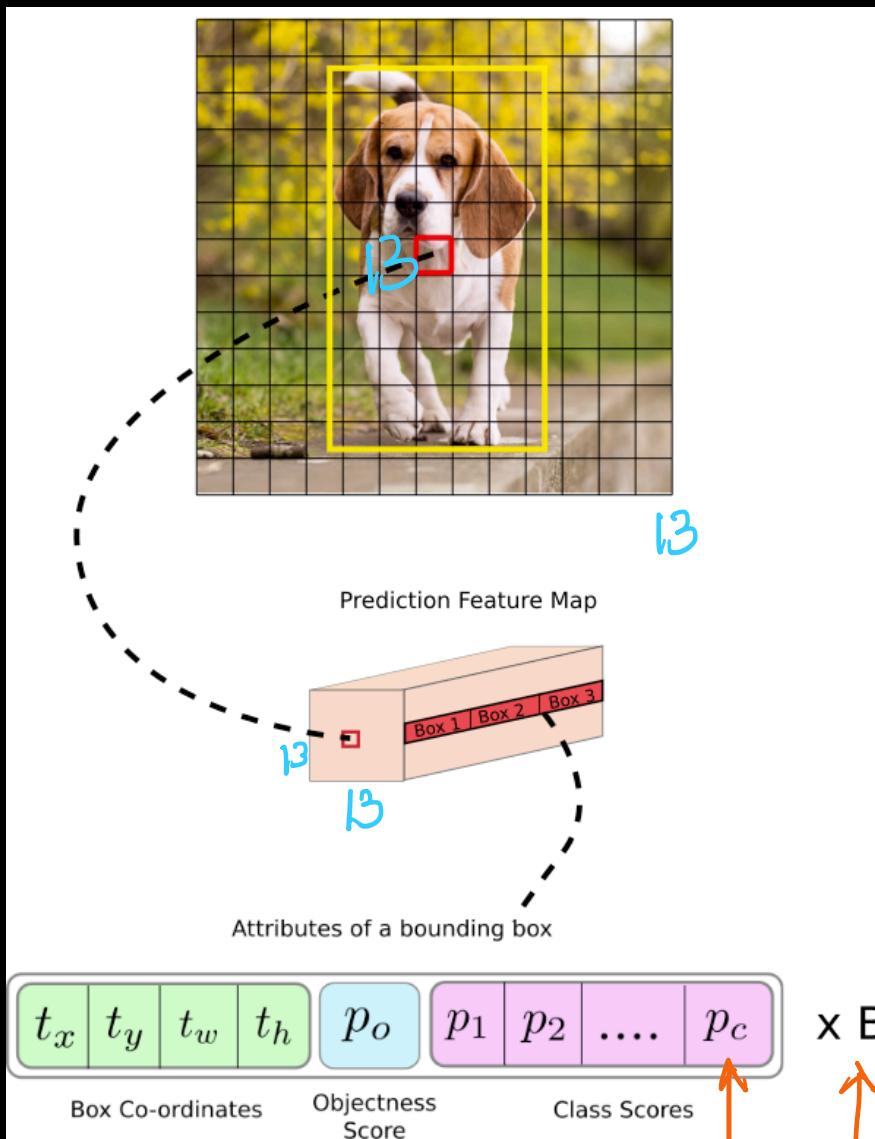
feature-extracted : 13 × 13 × 1024

Type	Filters	Size	Output
Convolutional	32	3 × 3	256 × 256
Convolutional	64	3 × 3 / 2	128 × 128
1x Convolutional	32	1 × 1	
Convolutional	64	3 × 3	
Residual			128 × 128
Convolutional	128	3 × 3 / 2	64 × 64
2x Convolutional	64	1 × 1	
Convolutional	128	3 × 3	
Residual			64 × 64
Convolutional	256	3 × 3 / 2	32 × 32
8x Convolutional	128	1 × 1	
Convolutional	256	3 × 3	
Residual			32 × 32
Convolutional	512	3 × 3 / 2	16 × 16
8x Convolutional	256	1 × 1	
Convolutional	512	3 × 3	
Residual			16 × 16
Convolutional	1024	3 × 3 / 2	8 × 8
4x Convolutional	512	1 × 1	
Convolutional	1024	3 × 3	
Residual			8 × 8
Avgpool		Global	
Connected		1000	
Softmax			

Darknet-53 model

/32

② Bounding-boxes & output



<https://pylessons.com/YOLOv3-introduction/>

feature-extractin

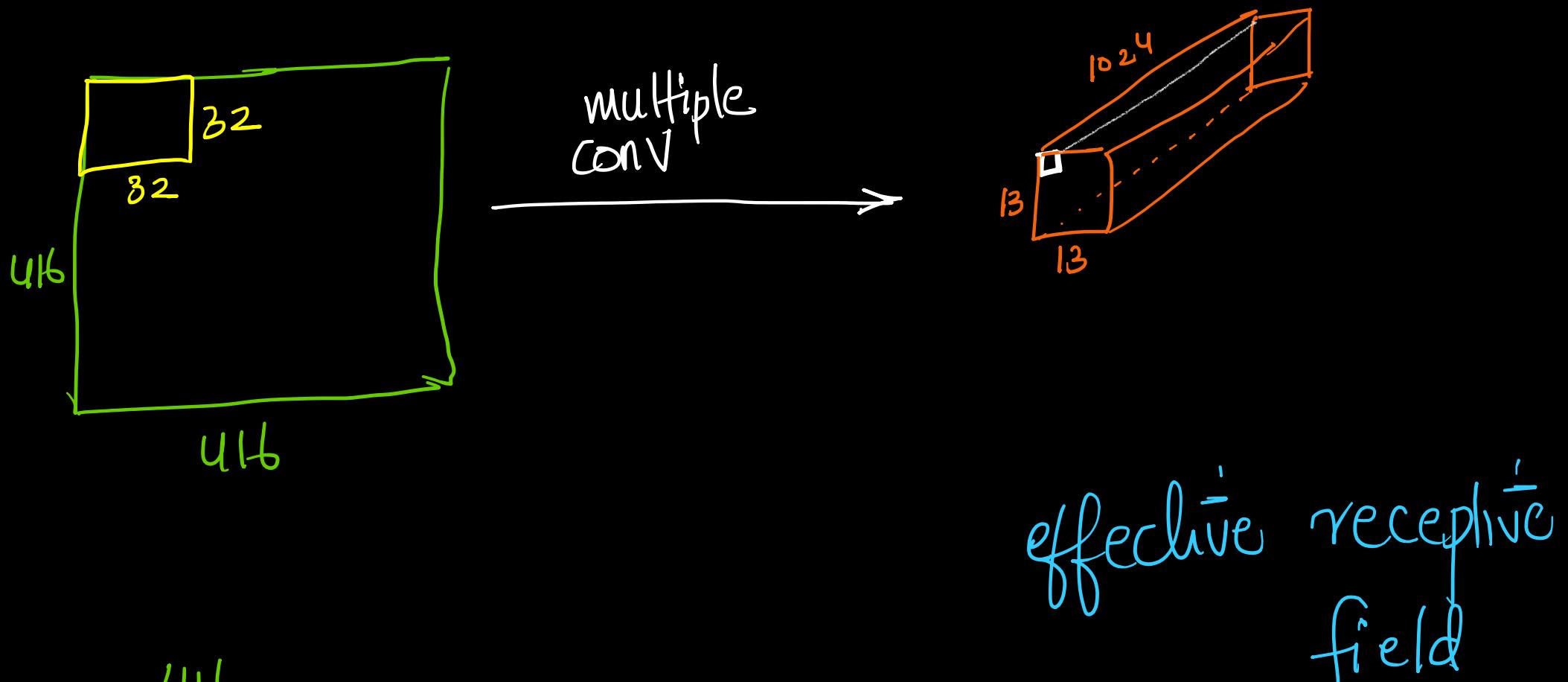
$13 \times 13 \times 1024$

$1 \times 1 \text{ conv}$
425

$13 \times 13 \times 425$

res. output

Refer: <https://www.appliedaicourse.com/lecture/11/applied-machine-learning-online-course/3412/inception-network/8/module-8-neural-networks-computer-vision-and-deep-learning>



$$\frac{416}{13} = 32$$

bounding box representation

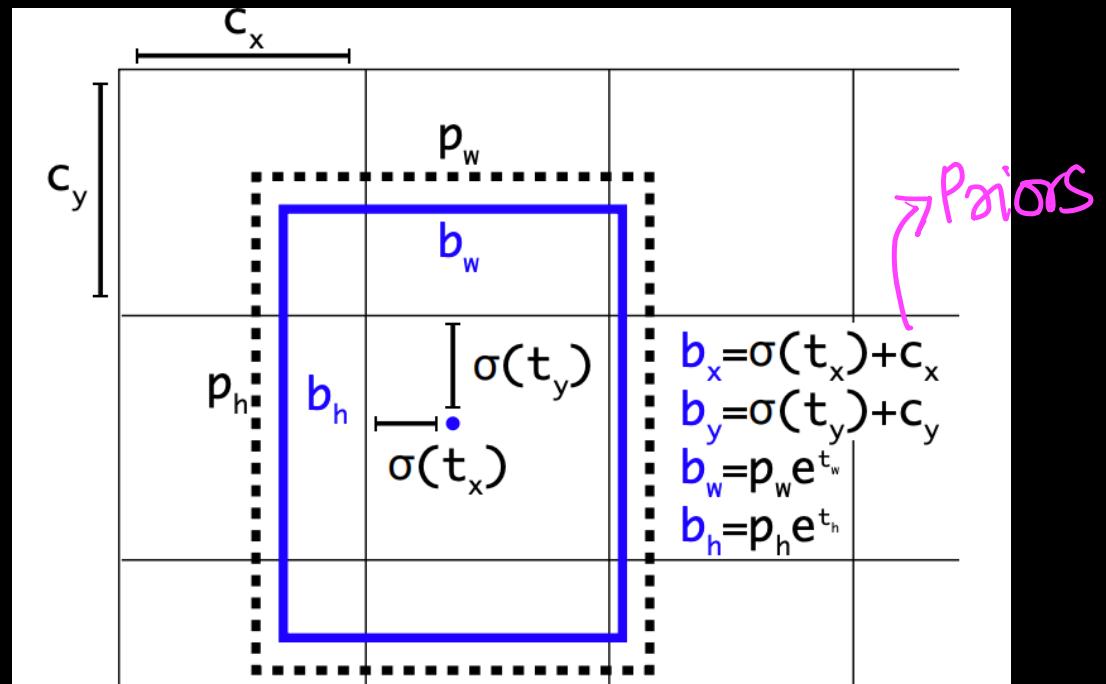
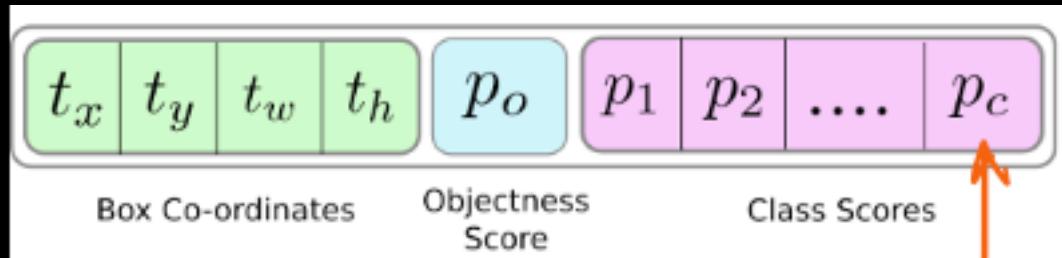


Figure 2. **Bounding boxes with dimension priors and location prediction.** We predict the width and height of the box as offsets from cluster centroids. We predict the center coordinates of the box relative to the location of filter application using a sigmoid function. This figure blatantly self-plagiarized from [15].

ANCHOR-boxes:

$$[c_x, c_y, p_w, p_h]$$

<https://pjreddie.com/media/files/papers/YOLOv3.pdf>

ANCHOR-boxes $\{x_c, y_c, P_w, P_h\}$

→ K-means clustering on train data's bounding boxes.

→ $K=5 \Rightarrow$ 5 anchor-boxes



5 bounding boxes
per cell

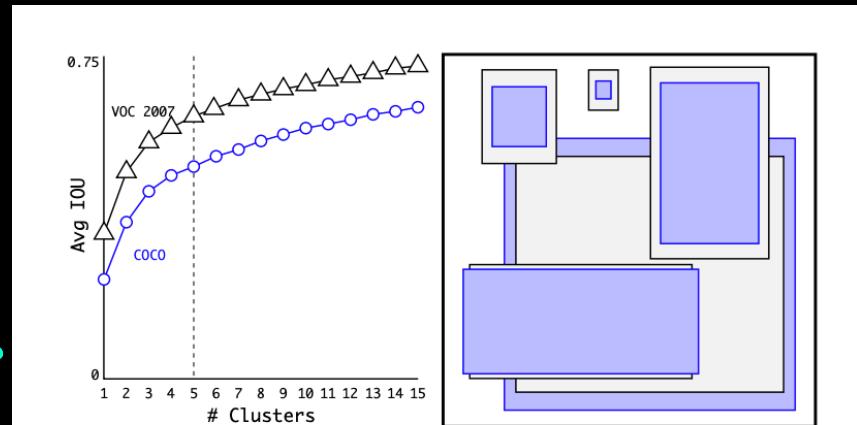
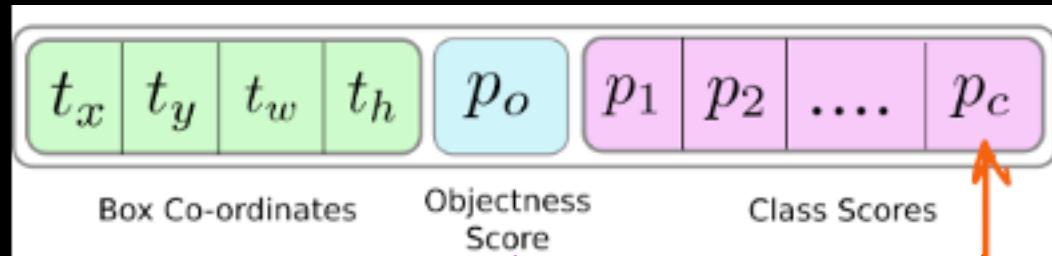


Figure 2: Clustering box dimensions on VOC and COCO. We run k-means clustering on the dimensions of bounding boxes to get good priors for our model. The left image shows the average IOU we get with various choices for k . We find that $k = 5$ gives a good tradeoff for recall vs. complexity of the model. The right image shows the relative centroids for VOC and COCO. Both sets of priors favor thinner, taller boxes while COCO has greater variation in size than VOC.

③ Per-class Sigmoids



$P(\text{box contains } C_i \text{ object})$

$$= p_o \times p_i$$

Is there any
object at all

e.g.: person , woman
 C_{10} C_{13} [Softmax
does not work]

LOSS:

$$\lambda_{\text{coord}} = 5$$

$$\lambda_{\text{noobj}} = 0.5$$

$\lambda_{\text{coord}} \cdot \text{squared-loss on}$
 f_{tw}, f_{h}, t_x, t_y

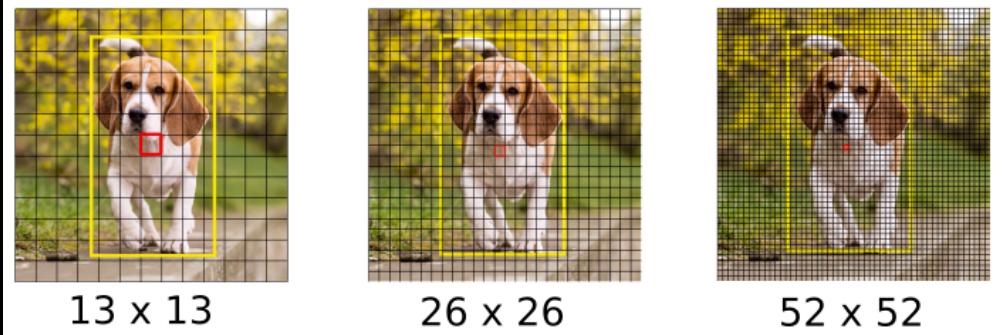
+ log-loss for p_o

all bounding
boxes
containing an object

+ log-loss for each p_i

$\sum_{\text{noobj boxes}}$ $\lambda_{\text{noobj}} \cdot \text{log-loss}$ for p_o

④ Multi-scale Prediction

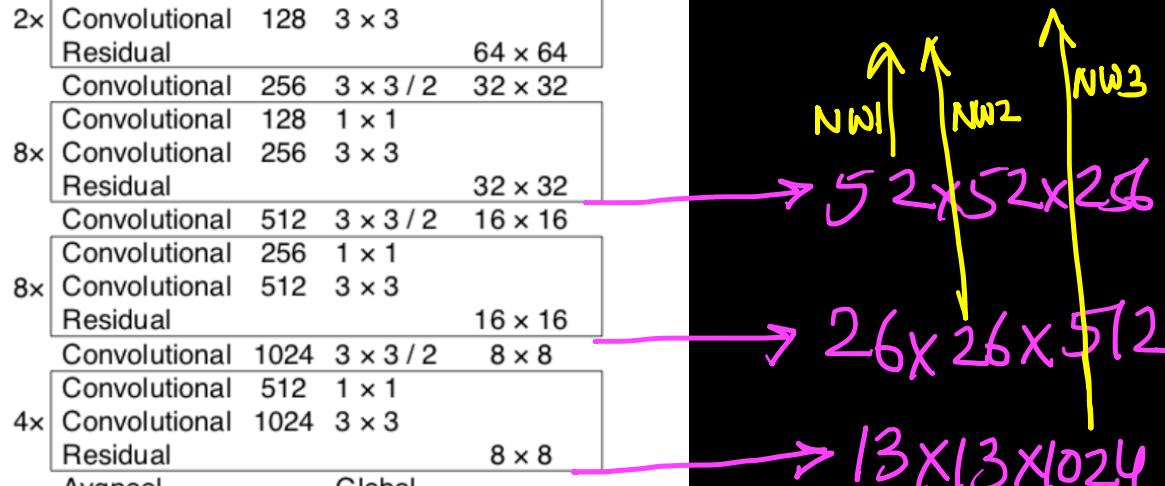


→ detect smaller objects

→ ideas from feature
pyramid networks (FPN)

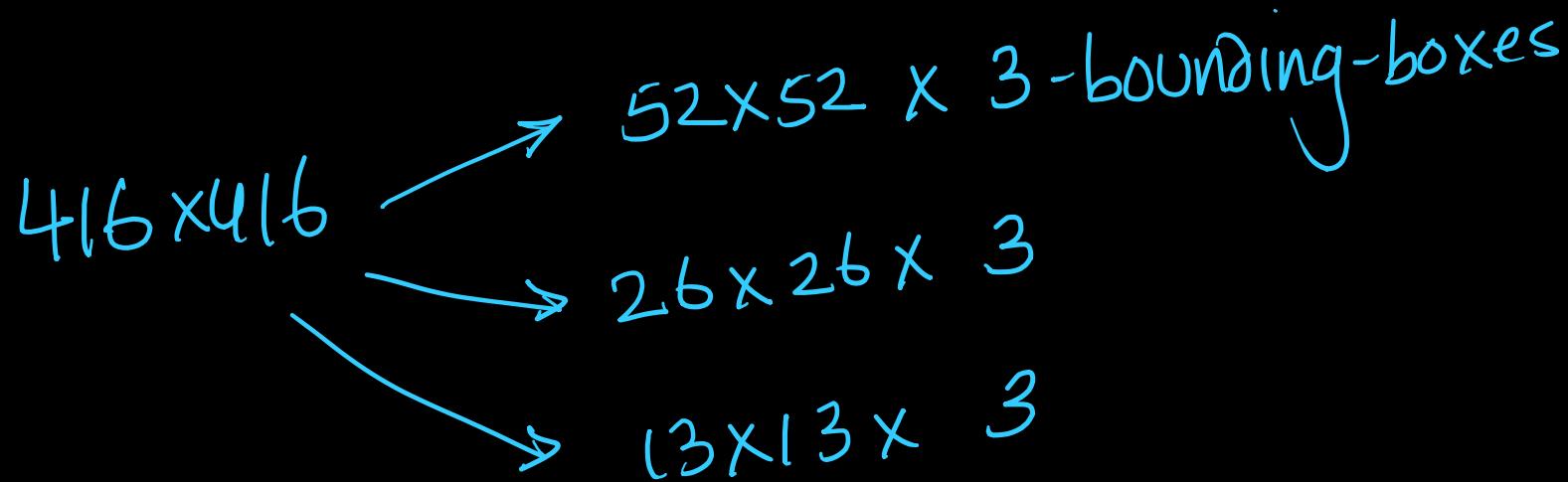
Type	Filters	Size	Output
Convolutional	32	3×3	256 x 256
Convolutional	64	$3 \times 3 / 2$	128 x 128
Convolutional	32	1×1	
1x Convolutional	64	3×3	128 x 128
Residual			
Convolutional	128	$3 \times 3 / 2$	64 x 64
Convolutional	64	1×1	
2x Convolutional	128	3×3	64 x 64
Residual			
Convolutional	256	$3 \times 3 / 2$	32 x 32
Convolutional	128	1×1	
8x Convolutional	256	3×3	32 x 32
Residual			
Convolutional	512	$3 \times 3 / 2$	16 x 16
Convolutional	256	1×1	
8x Convolutional	512	3×3	16 x 16
Residual			
Convolutional	1024	$3 \times 3 / 2$	8 x 8
Convolutional	512	1×1	
4x Convolutional	1024	3×3	8 x 8
Residual			
Avgpool		Global	
Connected		1000	
Softmax			

Darknet-53 model



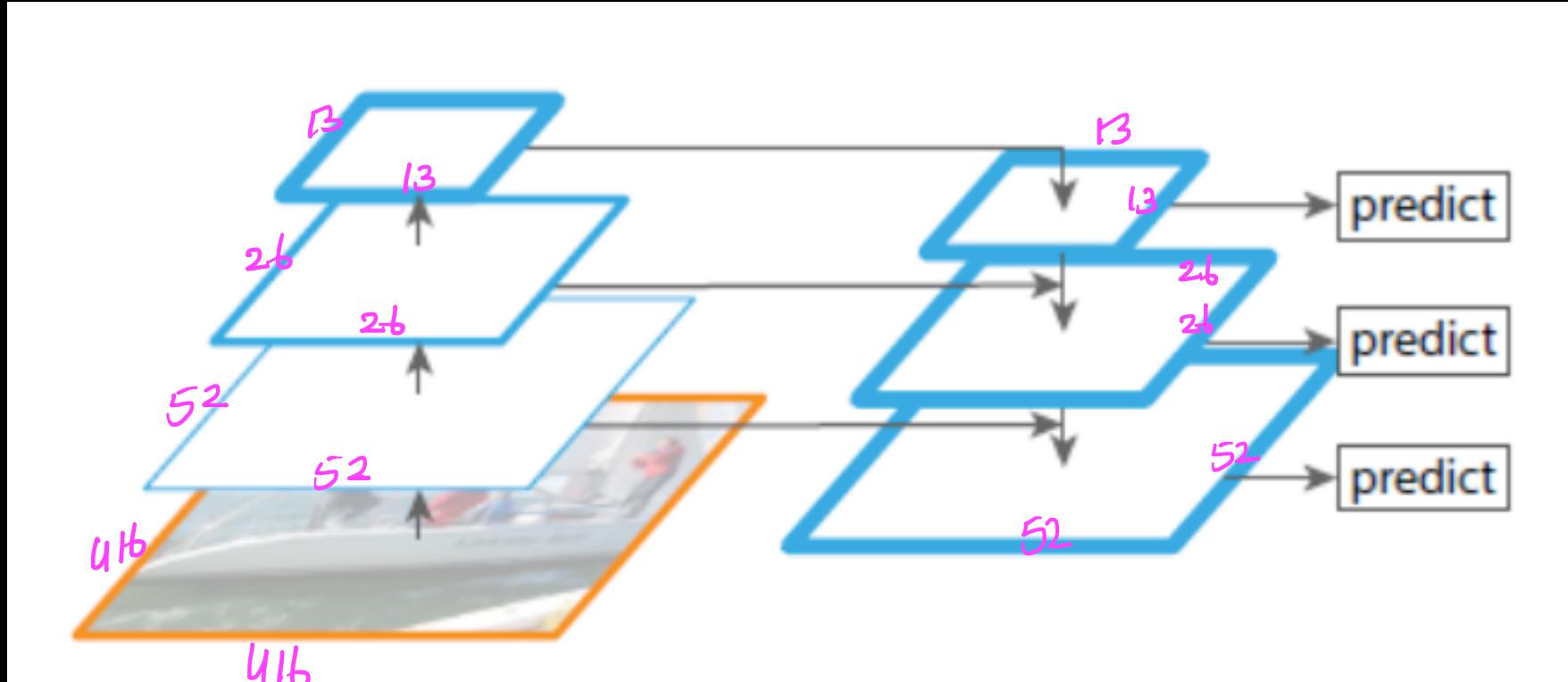
⑤

Combining boxes from various scales



10,647 boxes

FPN:



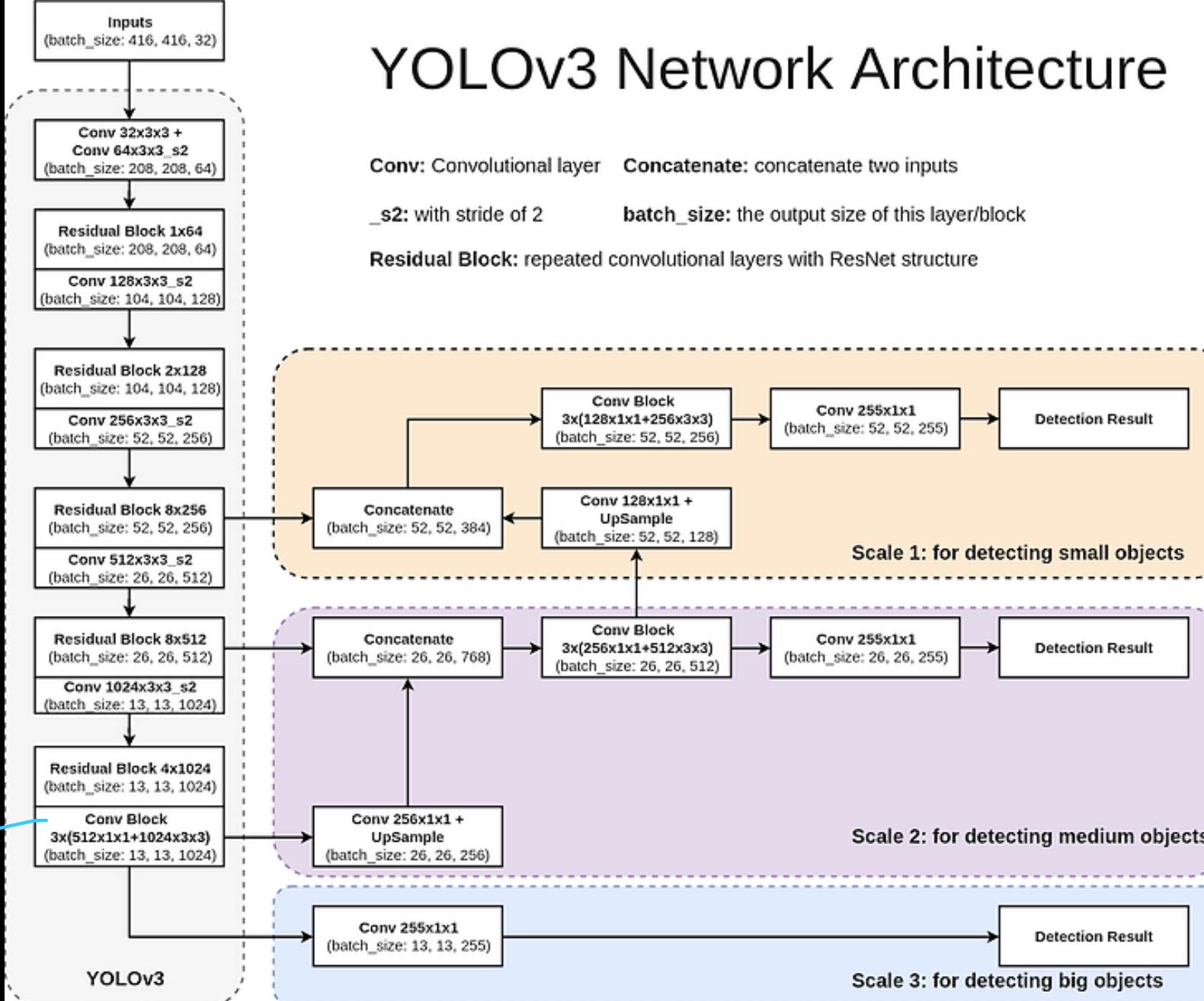
<https://towardsdatascience.com/review-fpn-feature-pyramid-network-object-detection-262fc7482610>

()

Darknet-

()

FPN with lateral-
connections



<https://www.tejashwi.io/training-yolov3-using-custom-dataset-on-google-colab/>

```

32 def make_yolov3_model():
33     input_image = Input(shape=(None, None, 3))
34     # Layer 0 => 4
35     x = _conv_block(input_image, [{"filter": 32, "kernel": 3, "stride": 1, "bnorm": True, "leaky": True, "layer_idx": 0},
36                                 {"filter": 64, "kernel": 3, "stride": 2, "bnorm": True, "leaky": True, "layer_idx": 1},
37                                 {"filter": 32, "kernel": 1, "stride": 1, "bnorm": True, "leaky": True, "layer_idx": 2},
38                                 {"filter": 64, "kernel": 3, "stride": 1, "bnorm": True, "leaky": True, "layer_idx": 3}])
39     # Layer 5 => 8
40     x = _conv_block(x, [{"filter": 128, "kernel": 3, "stride": 2, "bnorm": True, "leaky": True, "layer_idx": 5},
41                          {"filter": 64, "kernel": 1, "stride": 1, "bnorm": True, "leaky": True, "layer_idx": 6},
42                          {"filter": 128, "kernel": 3, "stride": 1, "bnorm": True, "leaky": True, "layer_idx": 7}])
43     # Layer 9 => 11
44     x = _conv_block(x, [{"filter": 64, "kernel": 1, "stride": 1, "bnorm": True, "leaky": True, "layer_idx": 9},
45                          {"filter": 128, "kernel": 3, "stride": 1, "bnorm": True, "leaky": True, "layer_idx": 10}])
46     # Layer 12 => 15
47     x = _conv_block(x, [{"filter": 256, "kernel": 3, "stride": 2, "bnorm": True, "leaky": True, "layer_idx": 12},
48                          {"filter": 128, "kernel": 1, "stride": 1, "bnorm": True, "leaky": True, "layer_idx": 13},
49                          {"filter": 256, "kernel": 3, "stride": 1, "bnorm": True, "leaky": True, "layer_idx": 14}])

```

]- repeat
2x

Type	Filters	Size	Output
0	Convolutional	32	256×256
1	Convolutional	64	128×128
2	Convolutional	32	1×1
3	Convolutional	64	3×3
	Residual		128×128
5	Convolutional	128	$3 \times 3 / 2$
6	Convolutional	64	1×1
7	2x	Convolutional	128
	3 \times 3		64×64
12	Convolutional	256	$3 \times 3 / 2$
B	Convolutional	128	1×1
8x	Convolutional	256	3×3
H	Residual		32×32
	Convolutional	512	$3 \times 3 / 2$
	Convolutional	256	1×1
8x	Convolutional	512	3×3
	Residual		16×16
	Convolutional	1024	$3 \times 3 / 2$
	Convolutional	512	1×1
4x	Convolutional	1024	3×3
	Residual		8×8
	Avgpool	Global	
	Connected	1000	
	Softmax		

512x512x3
26x26x512
13x13x1024

<https://machinelearningmastery.com/how-to-perform-object-detection-with-yolov3-in-keras/>

Type	Filters	Size	Output
Convolutional	32	3×3	256×256
Convolutional	64	$3 \times 3 / 2$	128×128
1x	Convolutional	32	1×1
Convolutional	64	3×3	128×128
Residual			128×128
Convolutional	128	$3 \times 3 / 2$	64×64
Convolutional	64	1×1	
2x	Convolutional	128	3×3
Convolutional	Residual		64×64
Convolutional	256	$3 \times 3 / 2$	32×32
Convolutional	128	1×1	
8x	Convolutional	256	3×3
Convolutional	Residual		32×32
Convolutional	512	$3 \times 3 / 2$	16×16
Convolutional	256	1×1	
8x	Convolutional	512	3×3
Convolutional	Residual		16×16
Convolutional	1024	$3 \times 3 / 2$	8×8
Convolutional	512	1×1	
4x	Convolutional	1024	3×3
Convolutional	Residual		8×8
Avgpool		Global	
Connected		1000	
Softmax			

Darknet-53 model

$512 \times 512 \times 256$

$26 \times 26 \times 512$

$13 \times 13 \times 1024$

+ more times

```
# Layer 16 => 36
for i in range(7):
    x = _conv_block(x, [{"filter": 128, "kernel": 1, "stride": 1, "bnorm": True, "leaky": True, "layer_idx": 16+i*3},
                         {"filter": 256, "kernel": 3, "stride": 1, "bnorm": True, "leaky": True, "layer_idx": 17+i*3}])
skip_36 = x
# Layer 37 => 40
x = _conv_block(x, [{"filter": 512, "kernel": 3, "stride": 2, "bnorm": True, "leaky": True, "layer_idx": 37},
                     {"filter": 256, "kernel": 1, "stride": 1, "bnorm": True, "leaky": True, "layer_idx": 38},
                     {"filter": 512, "kernel": 3, "stride": 1, "bnorm": True, "leaky": True, "layer_idx": 39}])
```

<https://machinelearningmastery.com/how-to-perform-object-detection-with-yolov3-in-keras/>

```

# Layer 66 => 74
for i in range(3):
    x = _conv_block(x, [{"filter": 512, "kernel": 1, "stride": 1, "bnorm": True, "leaky": True, "layer_idx": 66+i*3},
                        {"filter": 1024, "kernel": 3, "stride": 1, "bnorm": True, "leaky": True, "layer_idx": 67+i*3}])
# Layer 75 => 79
x = _conv_block(x, [{"filter": 512, "kernel": 1, "stride": 1, "bnorm": True, "leaky": True, "layer_idx": 75},
                     {"filter": 1024, "kernel": 3, "stride": 1, "bnorm": True, "leaky": True, "layer_idx": 76},
                     {"filter": 512, "kernel": 1, "stride": 1, "bnorm": True, "leaky": True, "layer_idx": 77},
                     {"filter": 1024, "kernel": 3, "stride": 1, "bnorm": True, "leaky": True, "layer_idx": 78},
                     {"filter": 512, "kernel": 1, "stride": 1, "bnorm": True, "leaky": True, "layer_idx": 79}], skip=False)
# Layer 80 => 82
yolo_82 = _conv_block(x, [{"filter": 1024, "kernel": 3, "stride": 1, "bnorm": True, "leaky": True, "layer_idx": 80},
                           {"filter": 255, "kernel": 1, "stride": 1, "bnorm": False, "leaky": False, "layer_idx": 81}], skip=False)

```

Type	Filters	Size	Output
Convolutional	32	3×3	256×256
Convolutional	64	$3 \times 3 / 2$	128×128
1x	32	1×1	
Convolutional	64	3×3	
	Residual		128×128
	Convolutional	$128 / 2$	64×64
2x	64	1×1	
Convolutional	128	3×3	
	Residual		64×64
	Convolutional	$256 / 2$	32×32
8x	128	1×1	
Convolutional	256	3×3	
	Residual		32×32
	Convolutional	$512 / 2$	16×16
8x	256	1×1	
Convolutional	512	3×3	
	Residual		16×16
	Convolutional	$1024 / 2$	8×8
4x	512	1×1	
Convolutional	1024	3×3	
	Residual		8×8
	Avgpool	Global	
	Connected	1000	
	Softmax		

Darknet-53 model

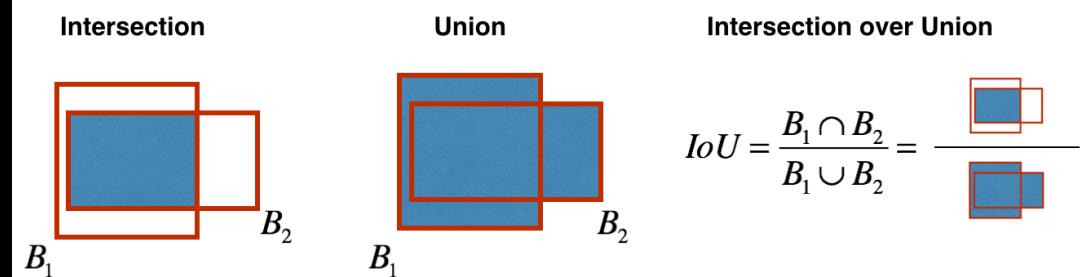
<https://machinelearningmastery.com/how-to-perform-object-detection-with-yolov3-in-keras/>



a) Filter: boxes with $\max_i(p_o \cdot p_i) < 0.5$ are ignored

b) Non-max-Suppression (NMS)

PICK THE BOX
WITH MAX IoU



Code :

<https://pjreddie.com/darknet/yolo/> → fast C-implementation

Keras-implementation : <https://github.com/qwwweee/keras-yolo3>
↳ not as-fast

<https://machinelearningmastery.com/how-to-perform-object-detection-with-yolov3-in-keras/>

