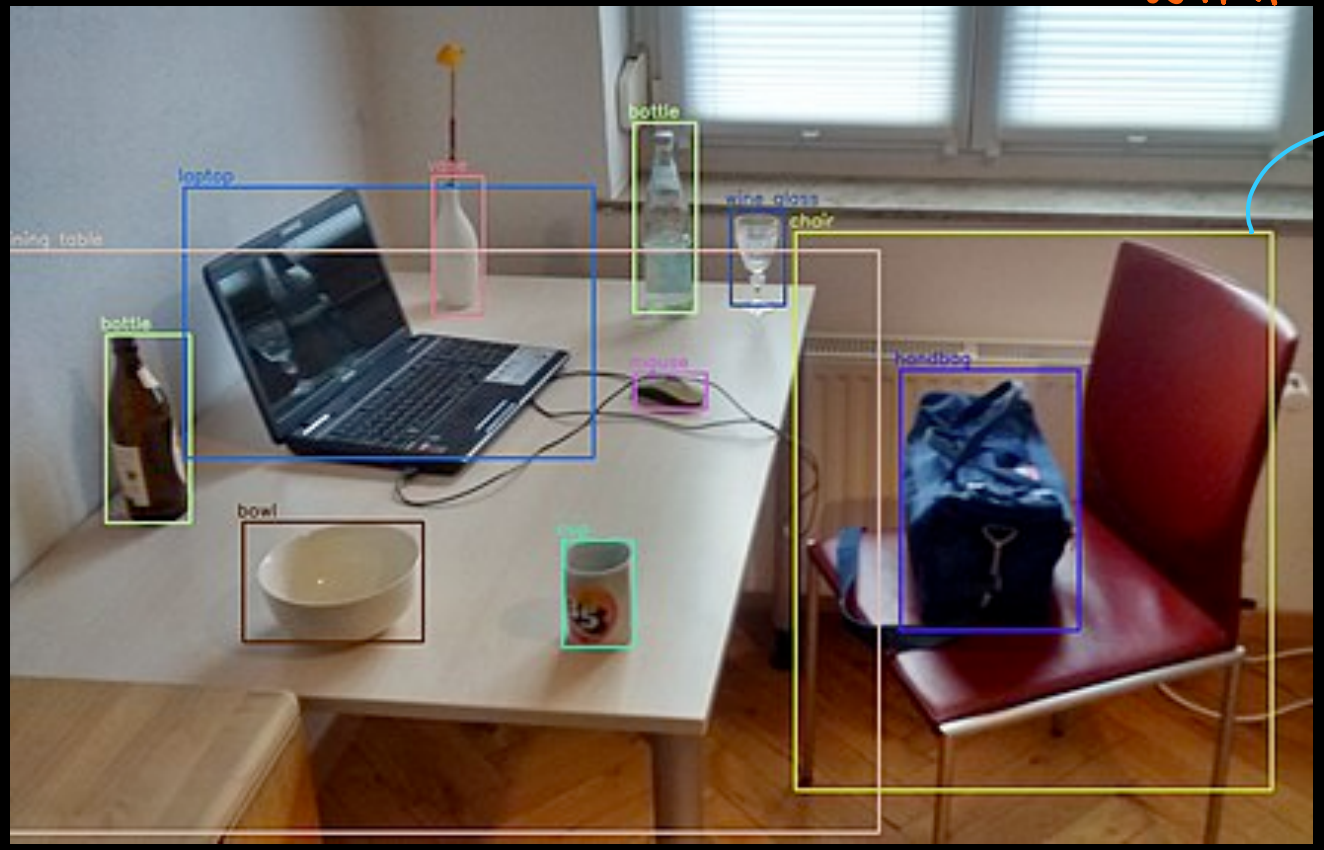
object-detection

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

APPLIED COURSE

Using Azure-API

– Pre-req: Using Web-APIs in Python

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

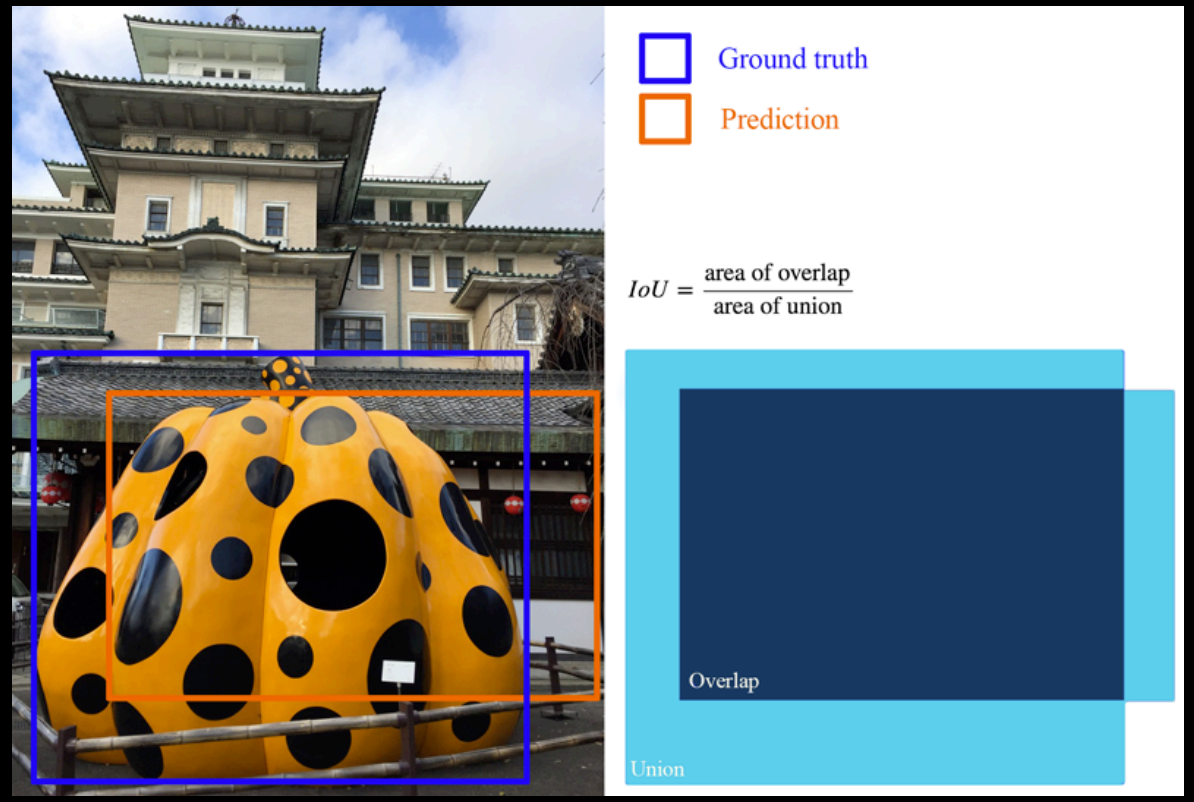
Documentation:
https://westcentralus.dev.cognitive.microsoft.com/docs/services/5adf991815e1060e6355ad44/operations/56f91f2e778daf14a499e1fa

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

$IoU \geq 0.5$

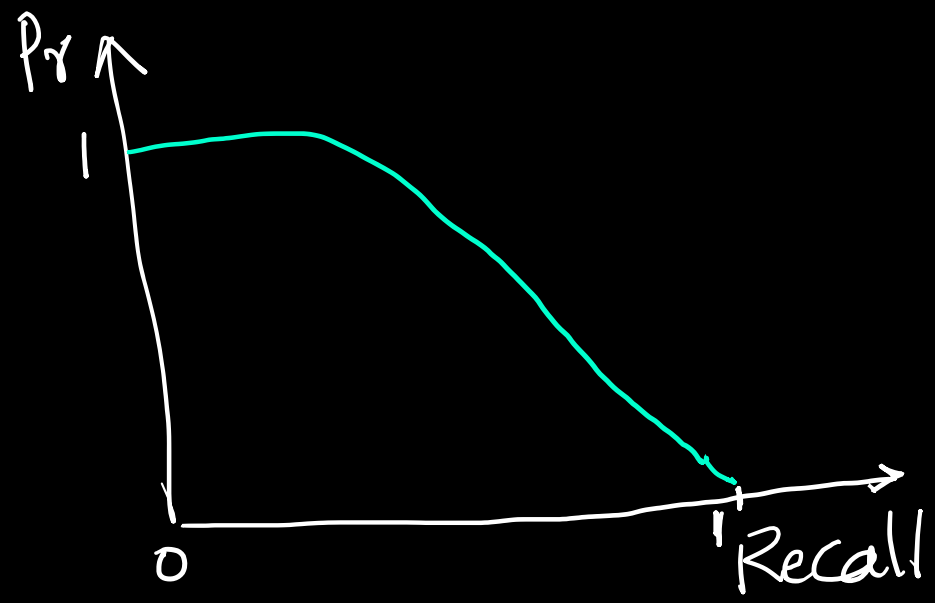
Performance - metric :

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

compute avg-precision

↓

AU - PR Curve



Performance - Metric :

mean - avg - precision (mAP) → NOT Max - A - posterion (MAP) in stats.

↓

mean across all classes

Deep-Learning - models:

input : image / video

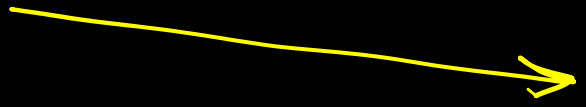
output : bounding-box-1, object-class-1

bounding-box-2, object - class-2

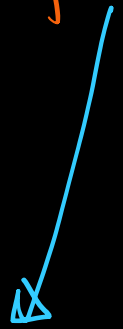
⋮

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

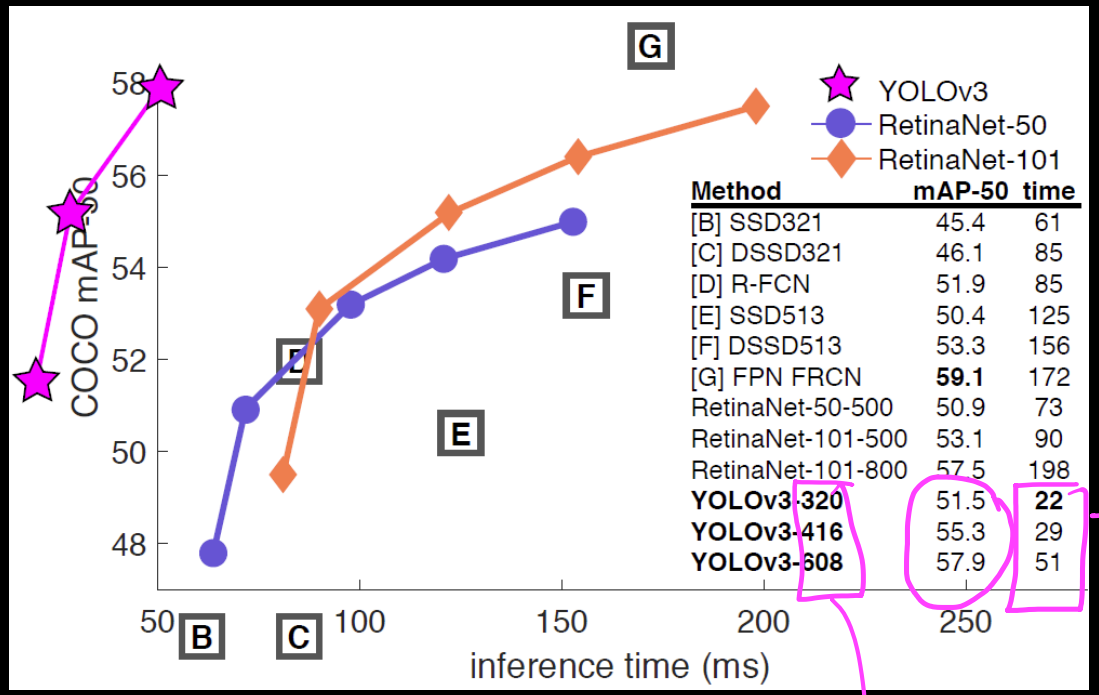
SSD

RetinaNet

YOLO V1

Yolo V2 / YOLO 9000

Yolo V3 — APR-2018



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

→ Super-fast + V. good mAP

input - image - size

APPLIED COURSE

YOLO - V3: feature-extractor (1)

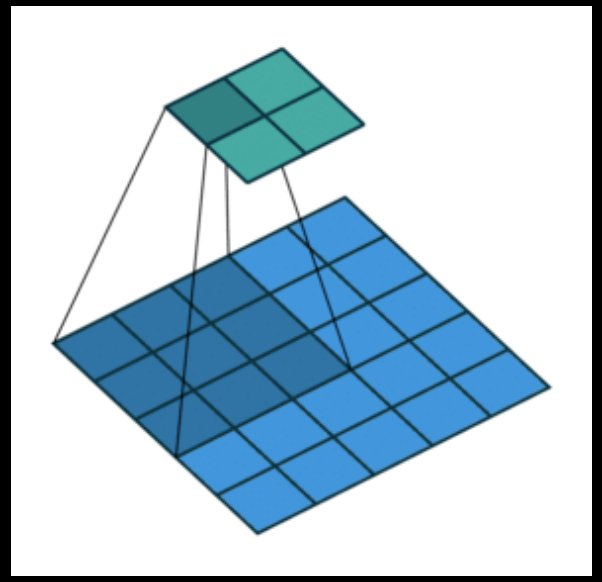
- Fully convolutional Network (FCN)
- conV - BatchNorm - LeakyReLU
- conv with stride - downsample
- no pooling
- pre-trained model

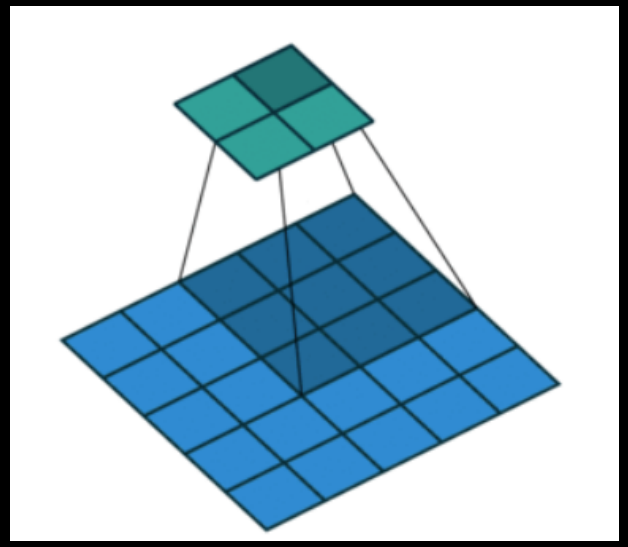
→ stride-2-conv

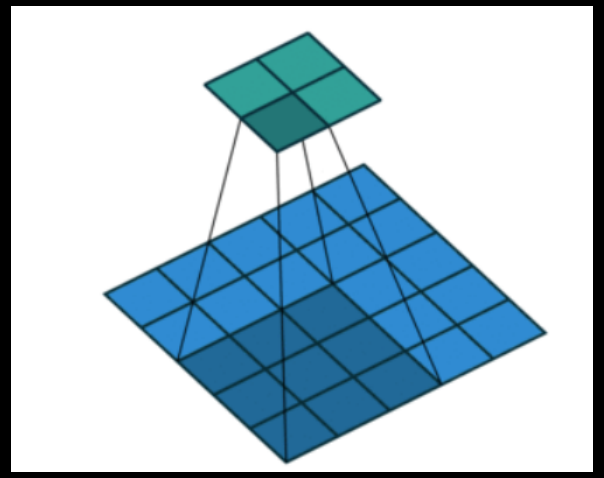
	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3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

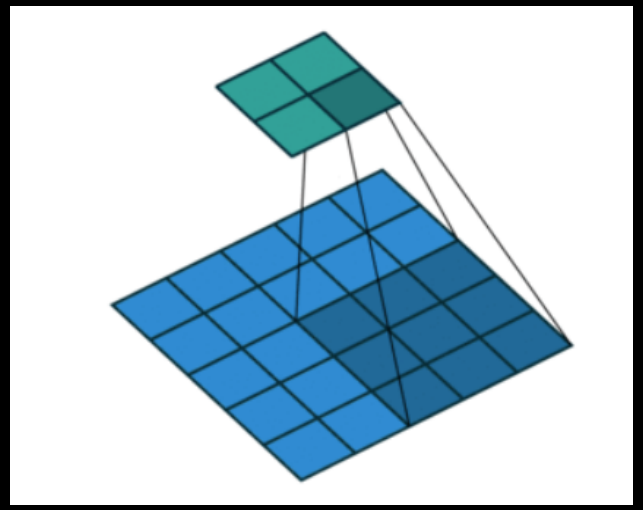
Darknet-53 model

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









Stride-2 convolutions

5x5 —3x3 kernels→ 2x2

https://github.com/vdumoulin/conv_arithmetic

a.k.a → feature-extractor

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

used in YOLOV2

78 → V.good

Slower than Darknet-19

more op/sec

input = images :

416 x 416 x 3

feature-extracted : 13 x 13 x 1024

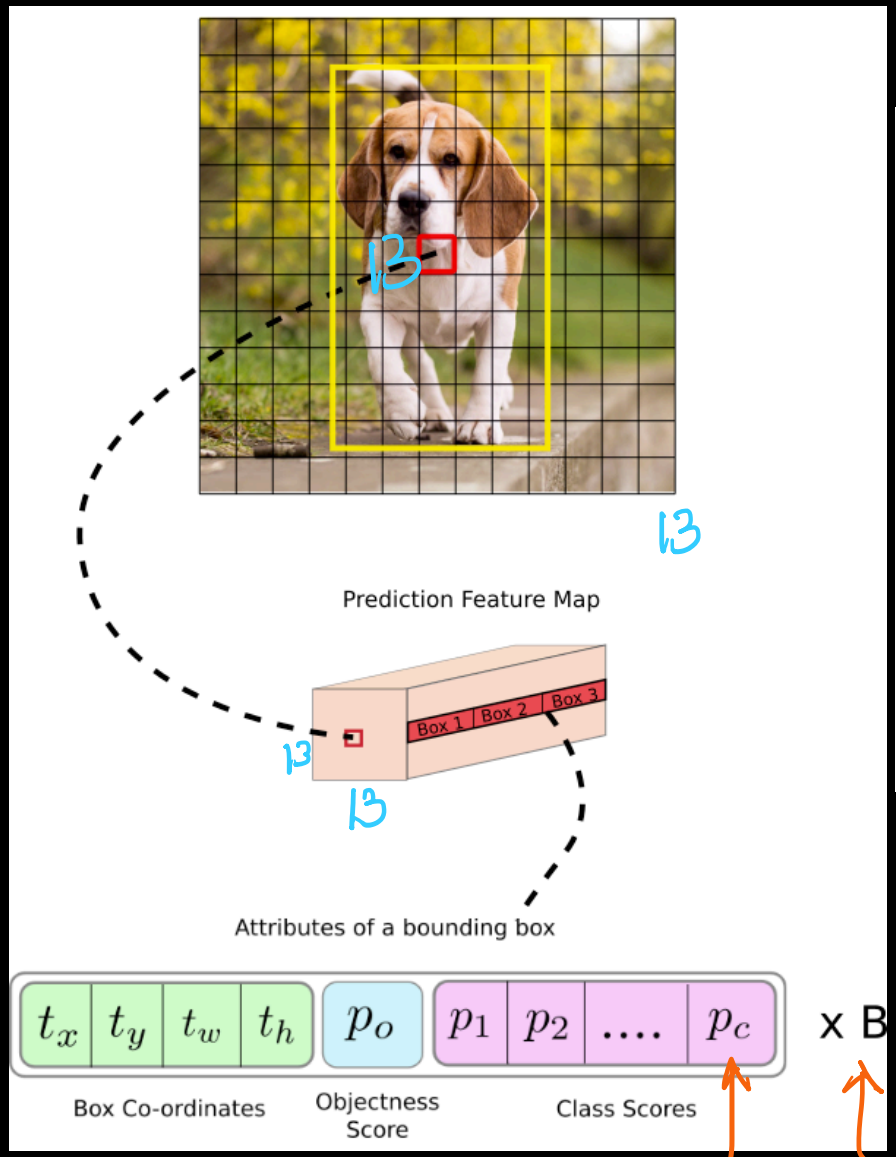
	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3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

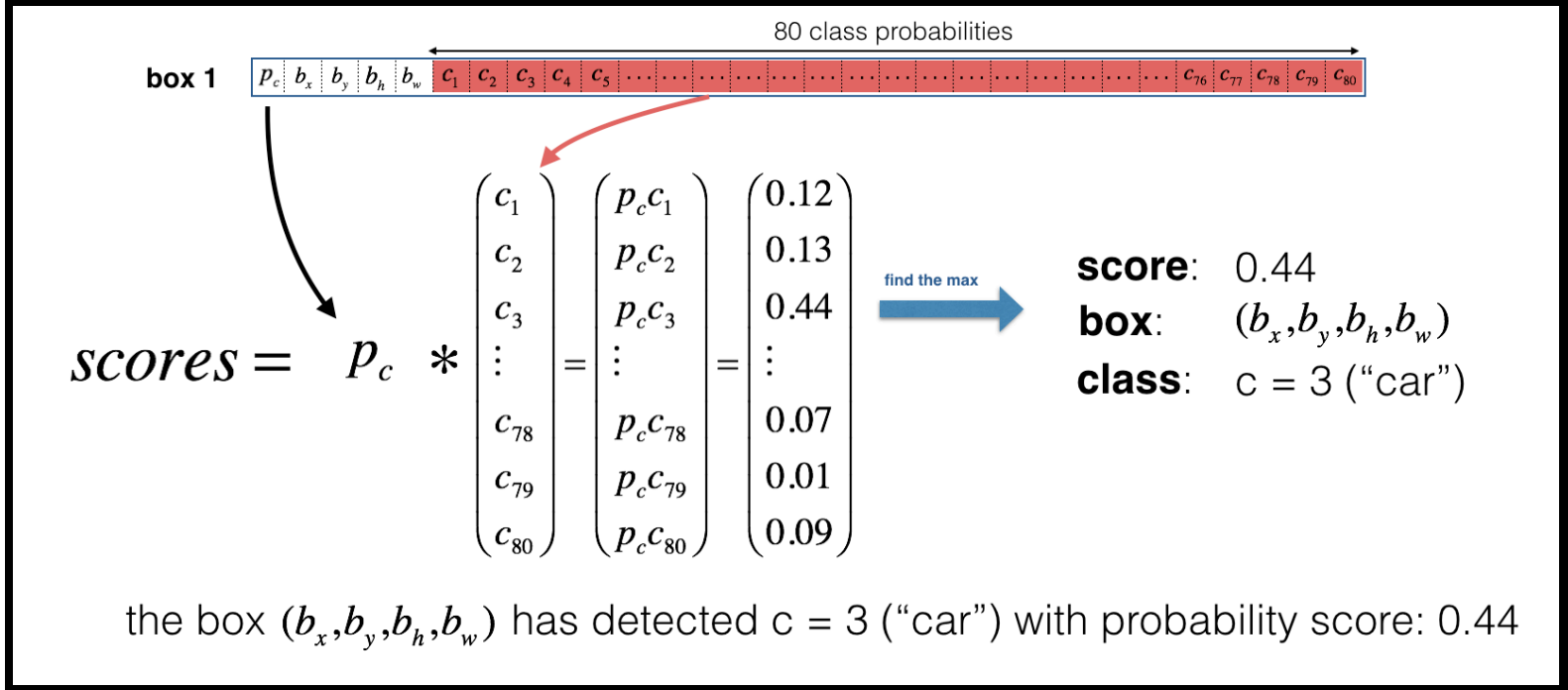
Darknet-53 model

/32

② Bounding-boxes & output

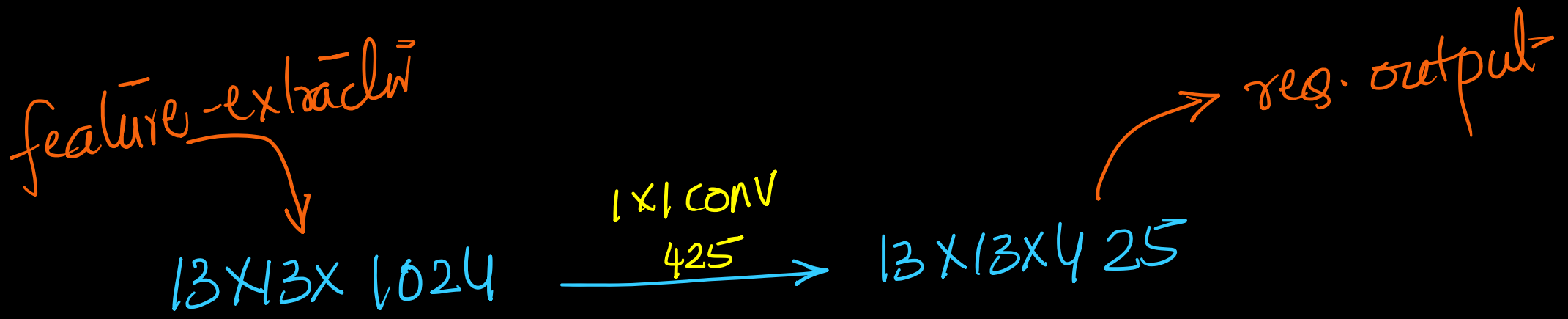
$13 \times 13 \times (B \cdot (5+80)) \rightarrow 425$ if $B=5$



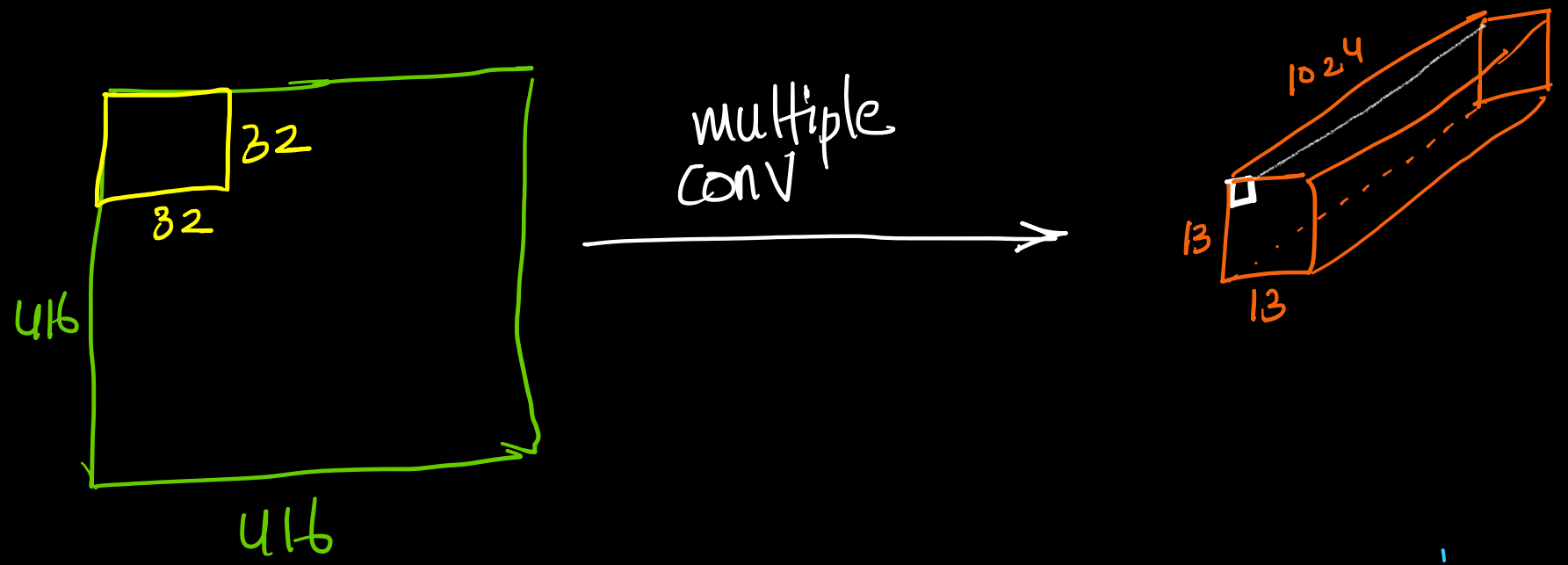


https://pylessons.com/YOLOv3-introduction/

APPLIED COURSE



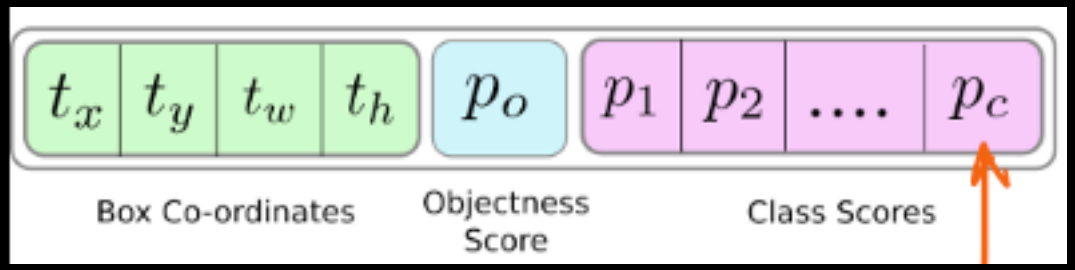
Refer: https://www.appliedaicourse.com/lecture/11/applied-machine-learning-online-course/3412/inception-network/8/module-8-neural-networks-computer-vision-and-deep-learning



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

effective receptive field

bounding box representation



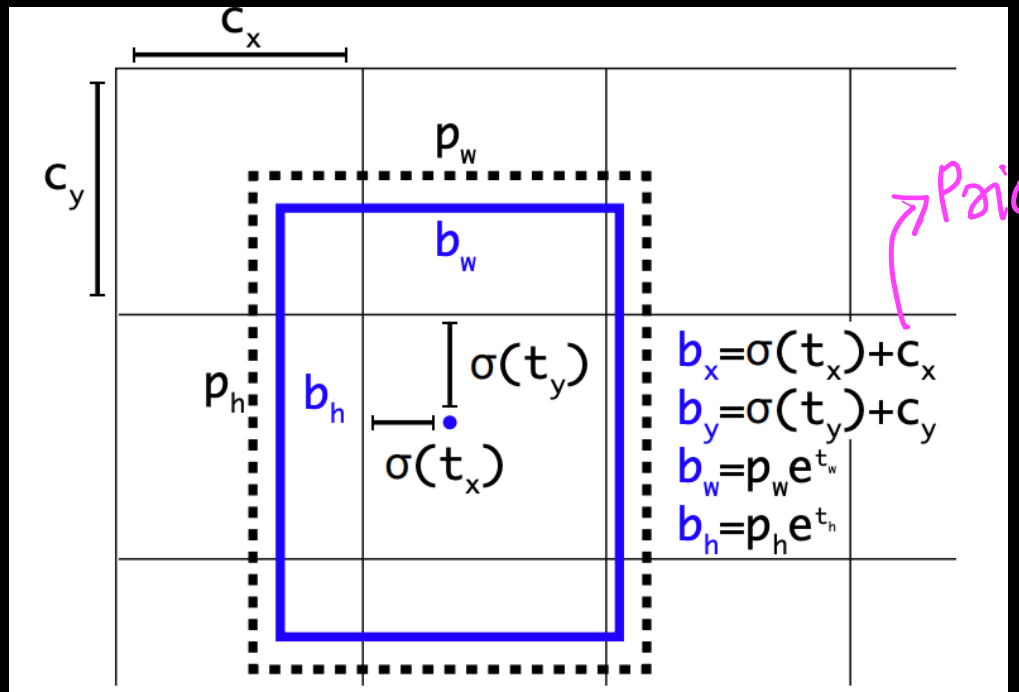


Figure 2. **Bounding boxes with dimension priors and location prediction.** We predict the width and height of the box as offsets from cluster centroids. We predict the center coordinates of the box relative to the location of filter application using a sigmoid function. This figure blatantly self-plagiarized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

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

ANCHOR-boxes:

[Cx, Cy, Pw, Ph]

APPLIED COURSE

ANCHOR - boxes [Cx, Cy, Pw, Ph]

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

→ K=5 ⇒ 5 anchor-boxes

↓

5 bounding boxes per cell

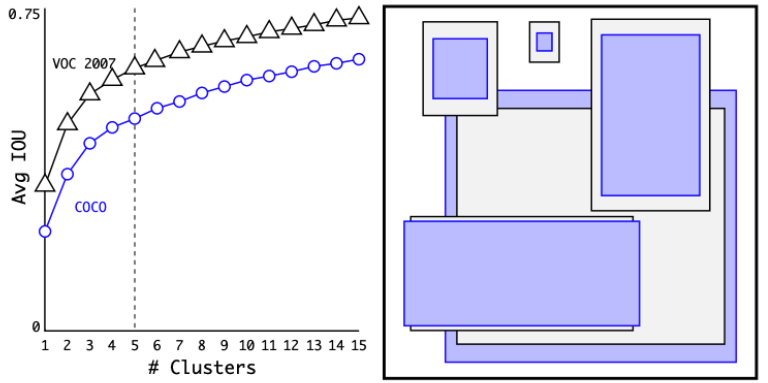
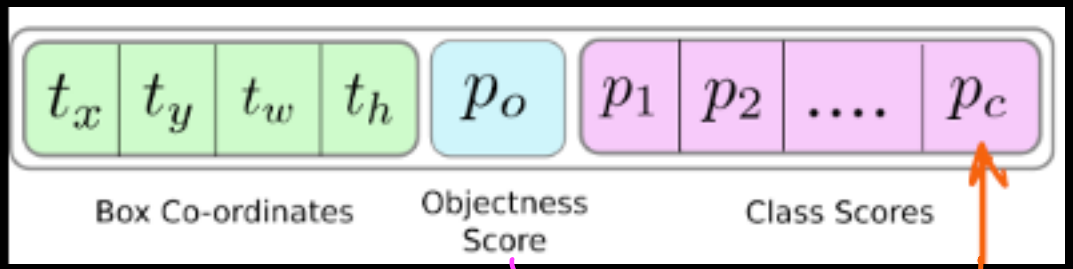


Figure 2: Clustering box dimensions on VOC and COCO. We run k-means clustering on the dimensions of bounding boxes to get good priors for our model. The left image shows the average IOU we get with various choices for k. We find that $k = 5$ gives a good tradeoff for recall vs. complexity of the model. 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 (box contains C_i object)

$= p_o \times p_i$

Is there any object at all

e.g: person C_{10}, woman C_{13} [softmax doesn't work]

APPLIED COURSE

Loss:

$\lambda_{coord} = 5$

$\lambda_{noobj} = 0.5$

$\sum$ all bounding boxes containing an object

λ_{coord} · squared-loss on $\hat{t}_w, \hat{t}_h, t_x \& t_y$

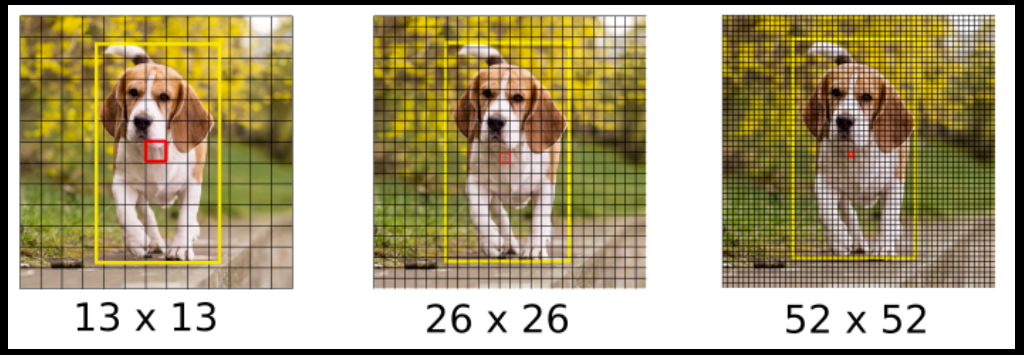
+ log-loss for p_o

+ log-loss for each p_i

+ $\sum$ noobj boxes

λ_{noobj} · log-loss for p_o

④ Multi-scale Prediction

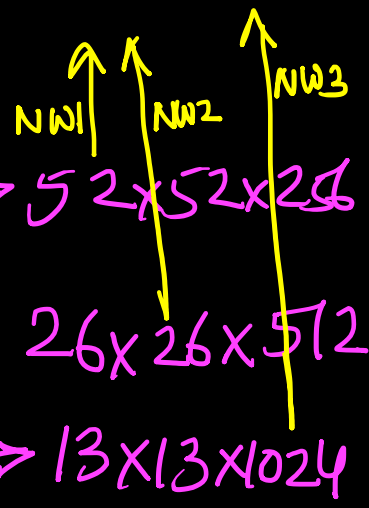


→ detect smaller objects

→ ideas from feature pyramid networks (FPN)

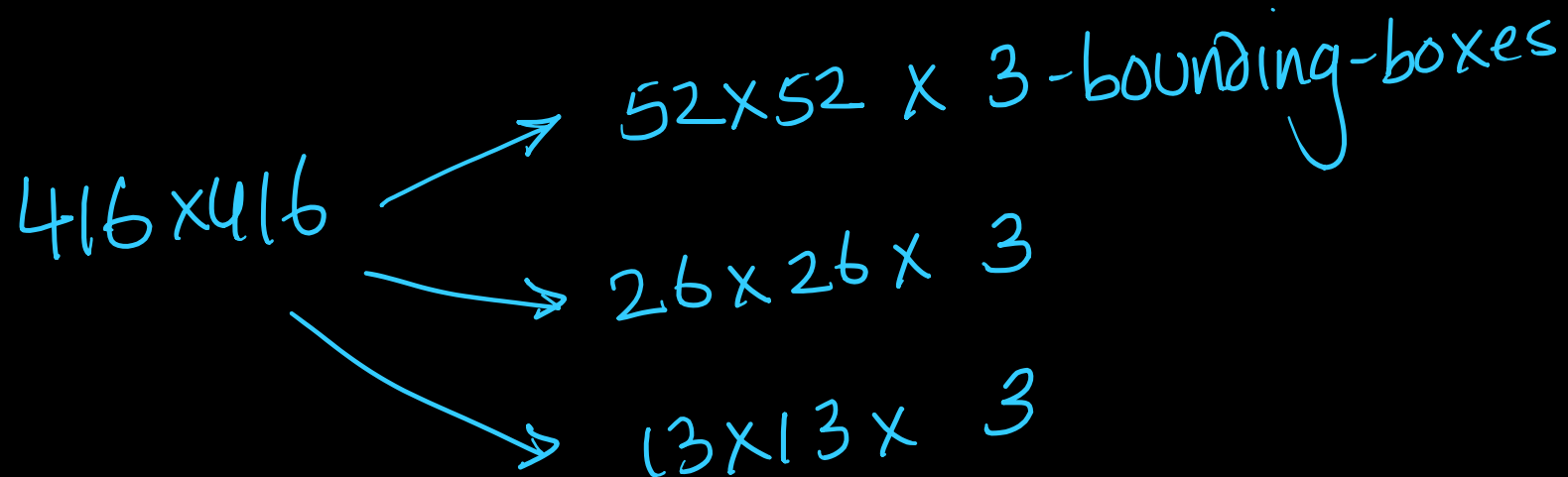
	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3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

Darknet-53 model



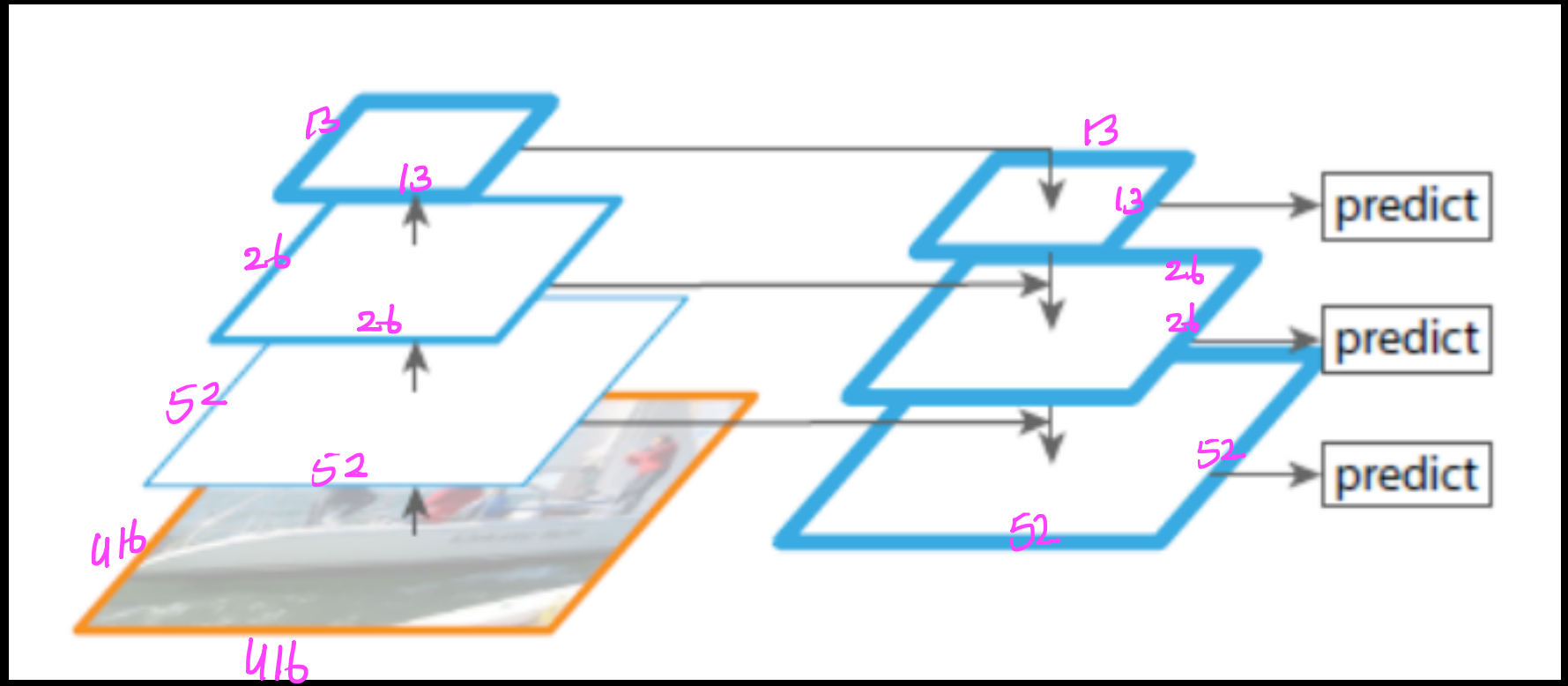
APPLIED COURSE

(5) Combining boxes from various scales



10,647 boxes

FPN:



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

Darknet

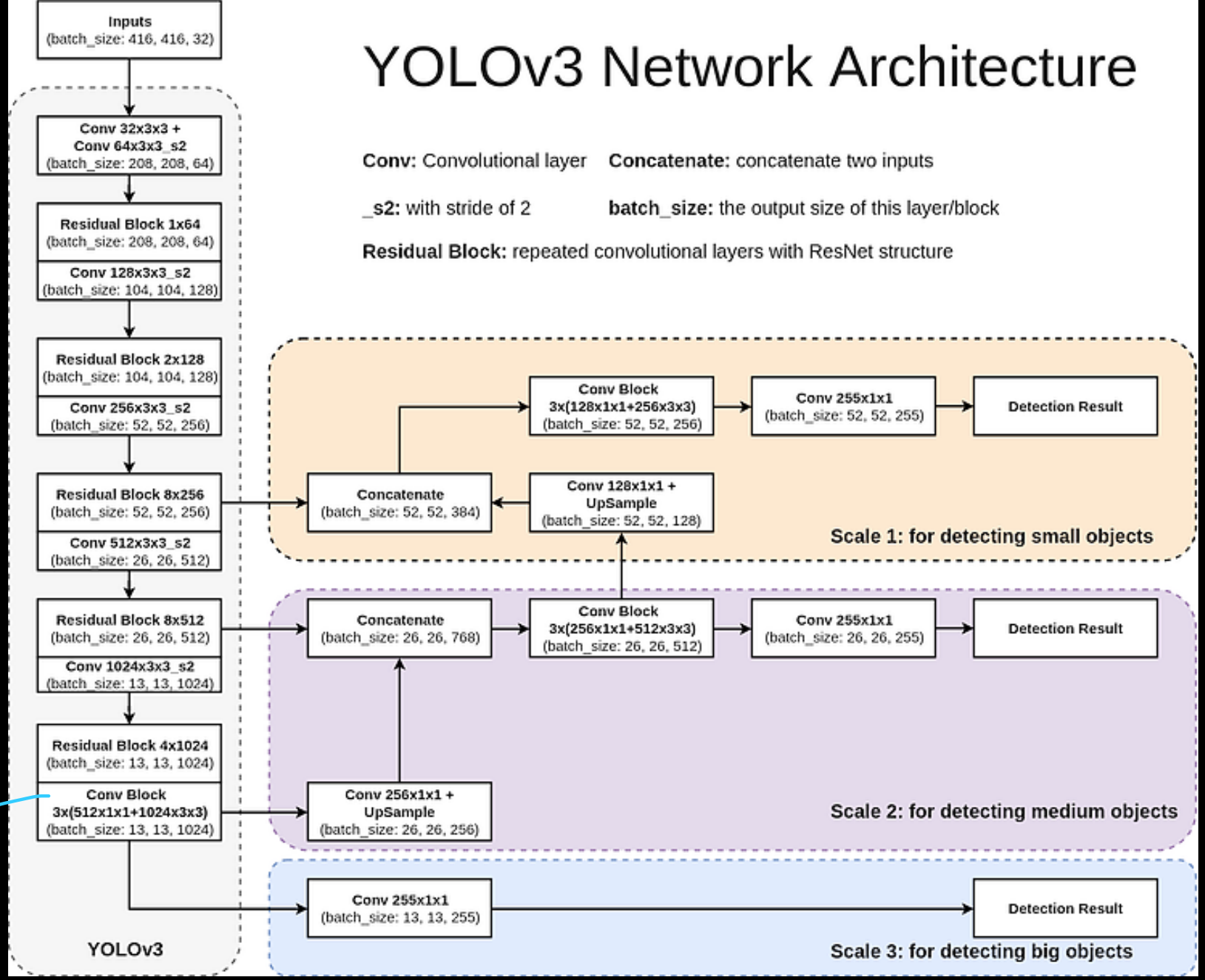
FPN with lateral connections

YOLOv3 Network Architecture

Conv: Convolutional layer **Concatenate:** concatenate two inputs

_s2: with stride of 2 **batch_size:** the output size of this layer/block

Residual Block: repeated convolutional layers with ResNet structure



More conv

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

```python
def make_yolov3_model():
    input_image = Input(shape=(None, None, 3))
    # Layer  0 => 4
    x = _conv_block(input_image, [{'filter': 32, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 0},
                                  {'filter': 64, 'kernel': 3, 'stride': 2, 'bnorm': True, 'leaky': True, 'layer_idx': 1},
                                  {'filter': 32, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 2},
                                  {'filter': 64, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 3}])
    # Layer  5 => 8
    x = _conv_block(x, [{'filter': 128, 'kernel': 3, 'stride': 2, 'bnorm': True, 'leaky': True, 'layer_idx': 5},
                        {'filter':  64, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 6},
                        {'filter': 128, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 7}])
    # Layer  9 => 11
    x = _conv_block(x, [{'filter':  64, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 9},
                        {'filter': 128, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 10}])
    # Layer 12 => 15
    x = _conv_block(x, [{'filter': 256, 'kernel': 3, 'stride': 2, 'bnorm': True, 'leaky': True, 'layer_idx': 12},
                        {'filter': 128, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 13},
                        {'filter': 256, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True, 'layer_idx': 14}])
```

repeat 2x

	Type	Filters	Size	Output
0	Convolutional	32	3 × 3	256 × 256
1	Convolutional	64	3 × 3 / 2	128 × 128
2	Convolutional	32	1 × 1	
3	1× Convolutional	64	3 × 3	
	Residual			128 × 128
5	Convolutional	128	3 × 3 / 2	64 × 64
6	Convolutional	64	1 × 1	
7	2× Convolutional	128	3 × 3	
	Residual			64 × 64
12	Convolutional	256	3 × 3 / 2	32 × 32
13	Convolutional	128	1 × 1	
14	8× Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
	Convolutional	256	1 × 1	
	8× Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
	Convolutional	512	1 × 1	
	4× Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

512x512x256

26x26x512

13x13x1024

Darknet-53 model

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

APPLIED COURSE

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

Darknet-53 model

512x512x256

26x26x512

13x13x1024

7 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}])
```

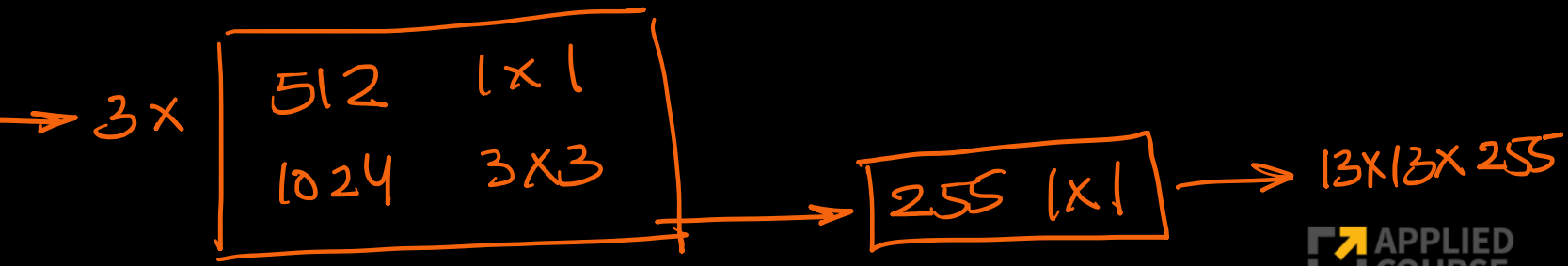
https://machinelearningmastery.com/how-to-perform-object-detection-with-yolov3-in-keras/

```
# 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)
```

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

Darknet-53 model

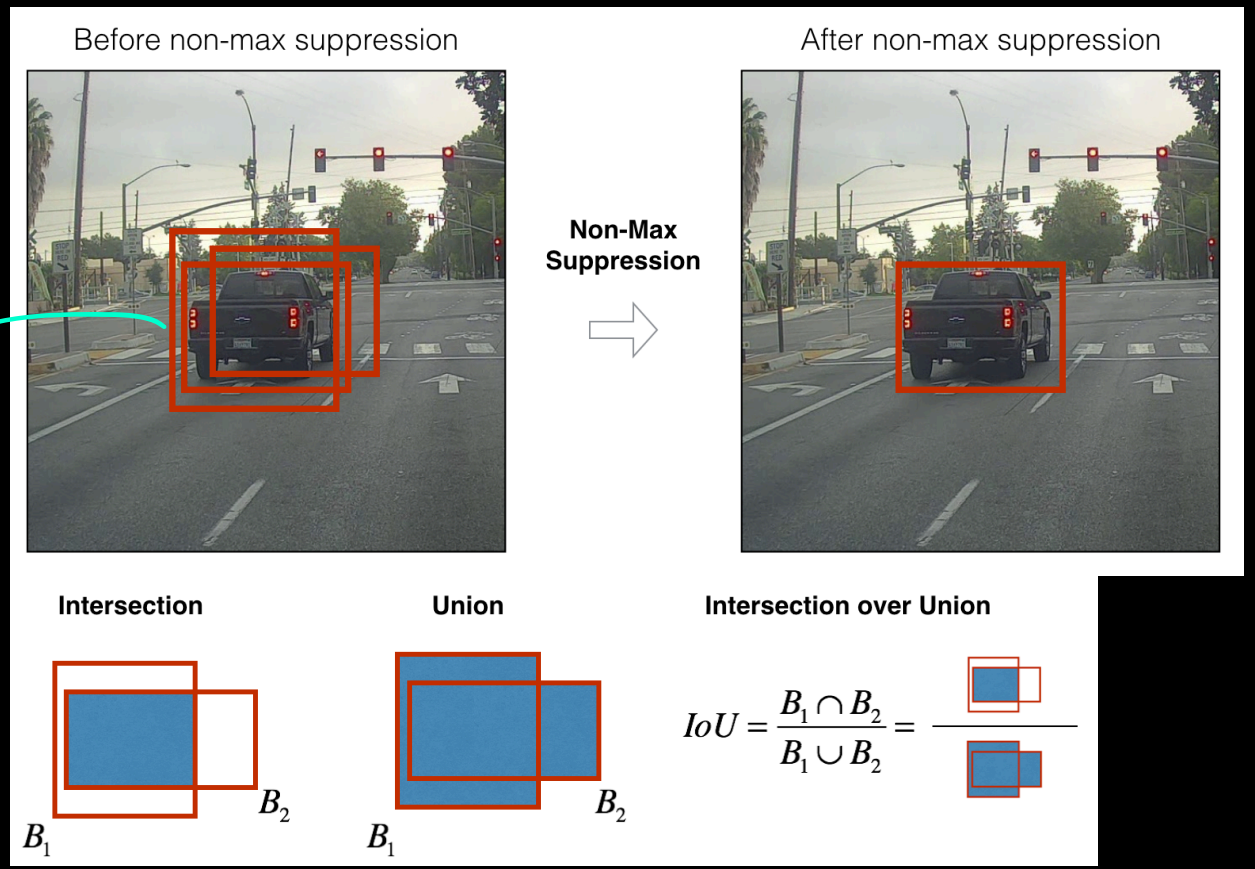


APPLIED COURSE

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

ⓑ Non-max-Suppression (NMS)

pick the box with max IoU



Code :

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

Keras- implementation : https://github.com/qqwweee/keras-yolo3

↳ not as-fast

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