AI VIETNAM
AI Insight Course

Object Detection

Quang-Vinh Dinh
Ph.D. in Computer Science

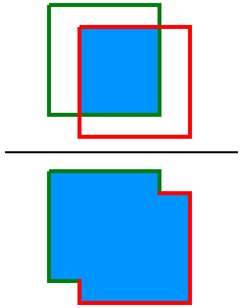
Year 2021

Object Detection Metrics

❖ Intersection Over Union (IOU)

$$IoU = \frac{B_p \cap B_{gt}}{B_p \cup B_{gt}}$$

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



True Positive (TP): A correct detection. Detection with IOU ≥ *threshold*	
False Positive (FP): A wrong detection. Detection with IOU < *threshold*	
False Negative (FN): A ground truth not detected	
True Negative (TN): Does not apply.	

threshold: depending on the metric,
usually set to 50%, 75% or 95%.

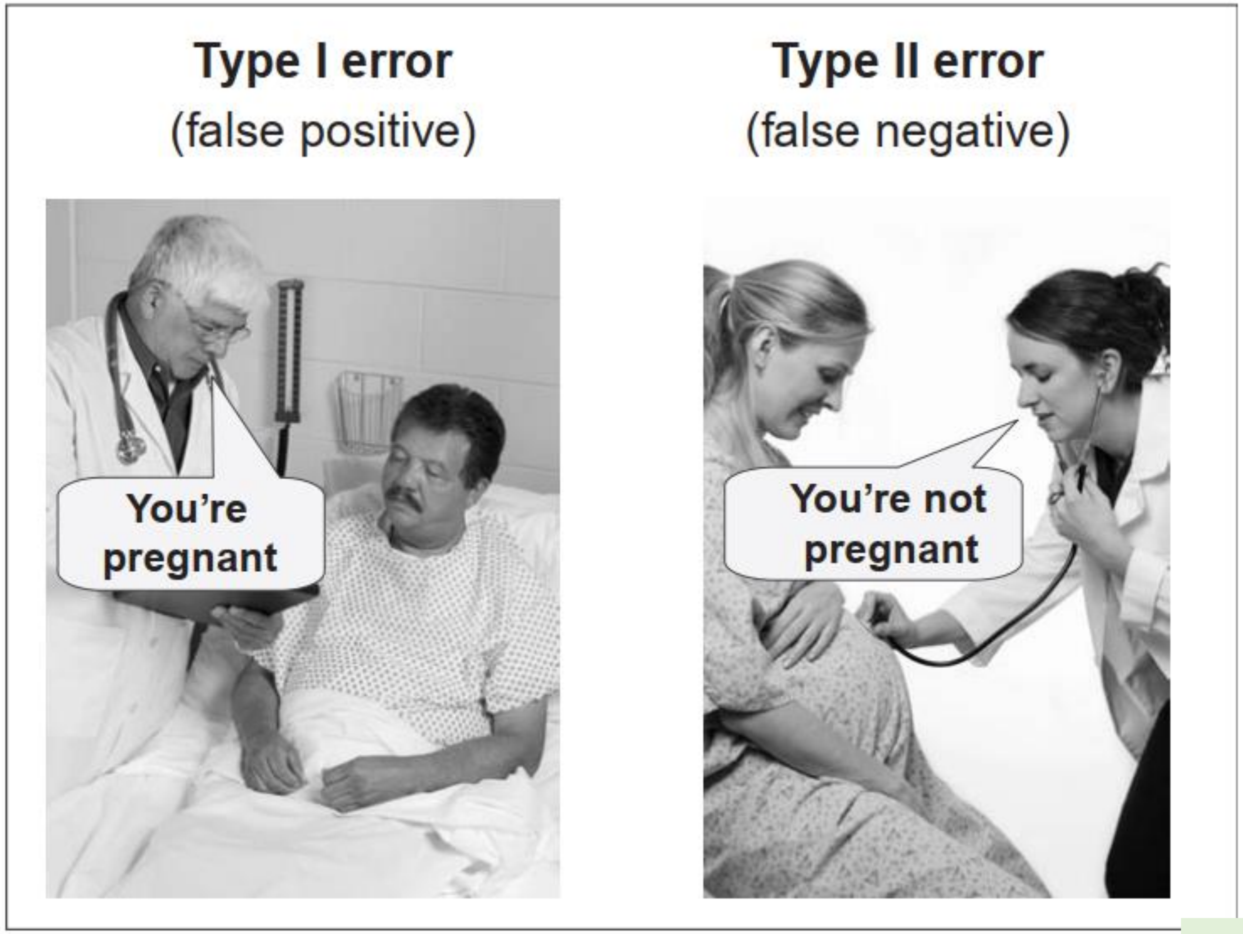
Object Detection Metrics

Confusion Matrix

		Actual Value	
		Positive	Negative
Predicted Value	Positive	TP (True Positive)	FP (False Positive)
	Negative	FN (False Negative)	TN (True Negative)

- True Positive (TP) : Observation is positive, and is predicted to be positive.
- False Negative (FN) : Observation is positive, but is predicted negative.
- True Negative (TN) : Observation is negative, and is predicted to be negative.
- False Positive (FP) : Observation is negative, but is predicted positive.

https://www.kdnuggets.com/2020/04/performance-evaluation-metrics-classification.html



The Essential Guide to Effect Sizes

Object Detection Metrics

❖ Confusion matrix

n=165	Predicted: NO	Predicted: YES
Actual: NO	50	10
Actual: YES	5	100

There are two possible predicted classes: "yes" and "no".

"yes" → have the disease,
"no" → don't have the disease.

https://www.dataschool.io/simple-guide-to-confusion-matrix-terminology/

Object Detection Metrics

❖ Confusion matrix

n=165	Predicted: NO	Predicted: YES
Actual: NO	50	10
Actual: YES	5	100

True positives (TP): We predicted yes, and they do have the disease.

True negatives (TN): We predicted no, and they don't have the disease.

False positives (FP): We predicted yes, but they don't actually have the disease. (Also known as a "Type I error.")

False negatives (FN): We predicted no, but they actually do have the disease. (Also known as a "Type II error.")

https://www.dataschool.io/simple-guide-to-confusion-matrix-terminology/

Object Detection Metrics

- **Precision**
  - **Ability of a model to identify only the relevant objects**

$$Precision = \frac{TP}{TP + FP} = \frac{TP}{all\ detections}$$

- **Recall**
  - **Ability of a model to find all the relevant cases (all ground truth bounding boxes)**

$$Recall = \frac{TP}{TP + FN} = \frac{TP}{all\ ground\ truths}$$

Object Detection Metrics

❖ Confusion matrix

n=165	Predicted: NO	Predicted: YES	
Actual: NO	TN = 50	FP = 10	60
Actual: YES	FN = 5	TP = 100	105
	55	110	

Recall: When it's actually yes, how often does it predict yes?
TP/actual yes = 100/105 = 0.95

Precision: When it predicts yes, how often is it correct?
TP/predicted yes = 100/110 = 0.91

Object Detection Metrics

❖ Example

precision = TP/(all predictions) = 2/3;

recall = TP / (all positive ground truth) = 2/5

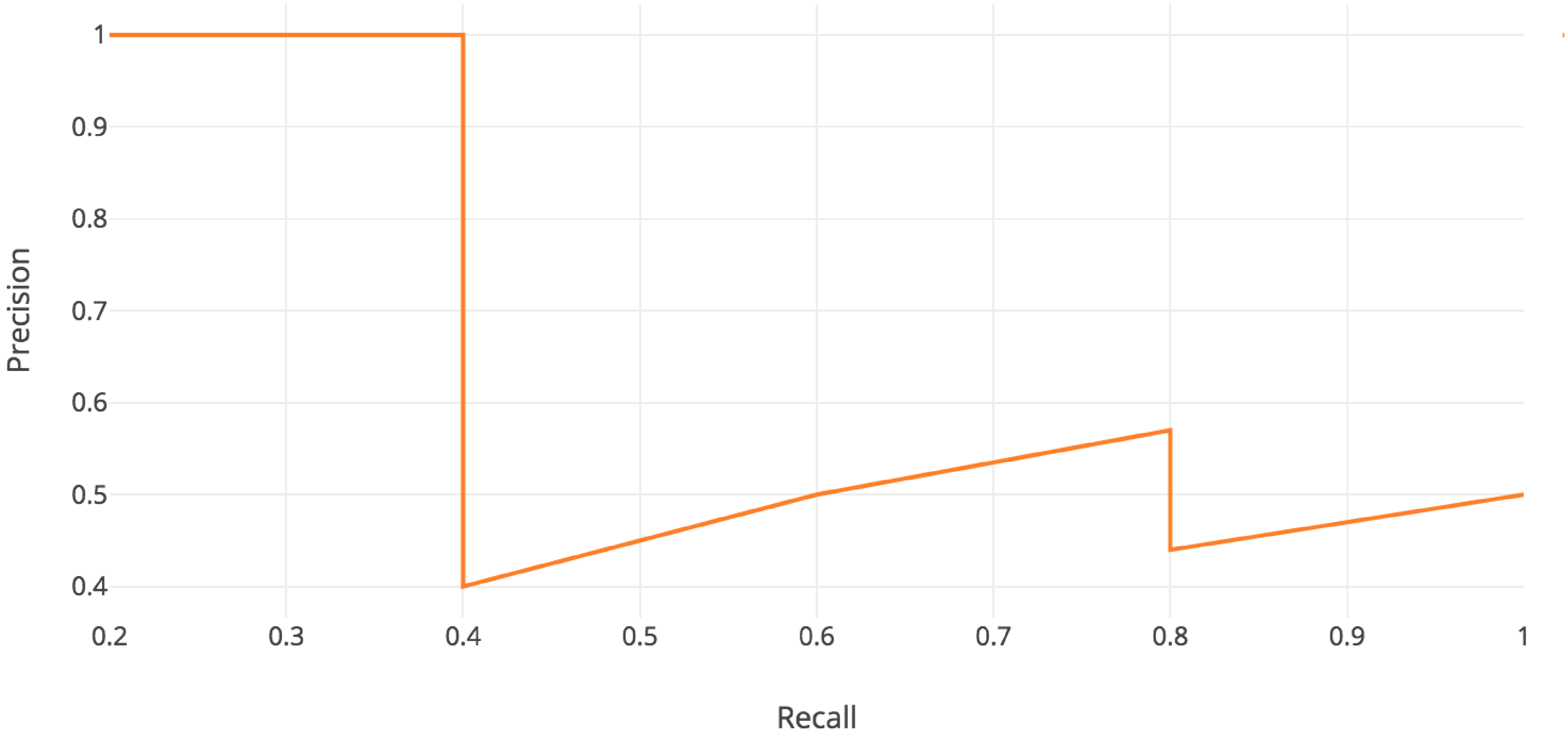
Recall values increase as we go down the prediction ranking, but precision has a zigzag pattern.

Rank	Correct?	Precision	Recall
1	True	1.0	0.2
2	True	1.0	0.4
3	False	0.67	0.4
4	False	0.5	0.4
5	False	0.4	0.4
6	True	0.5	0.6
7	True	0.57	0.8
8	False	0.5	0.8
9	False	0.44	0.8
10	True	0.5	1.0

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

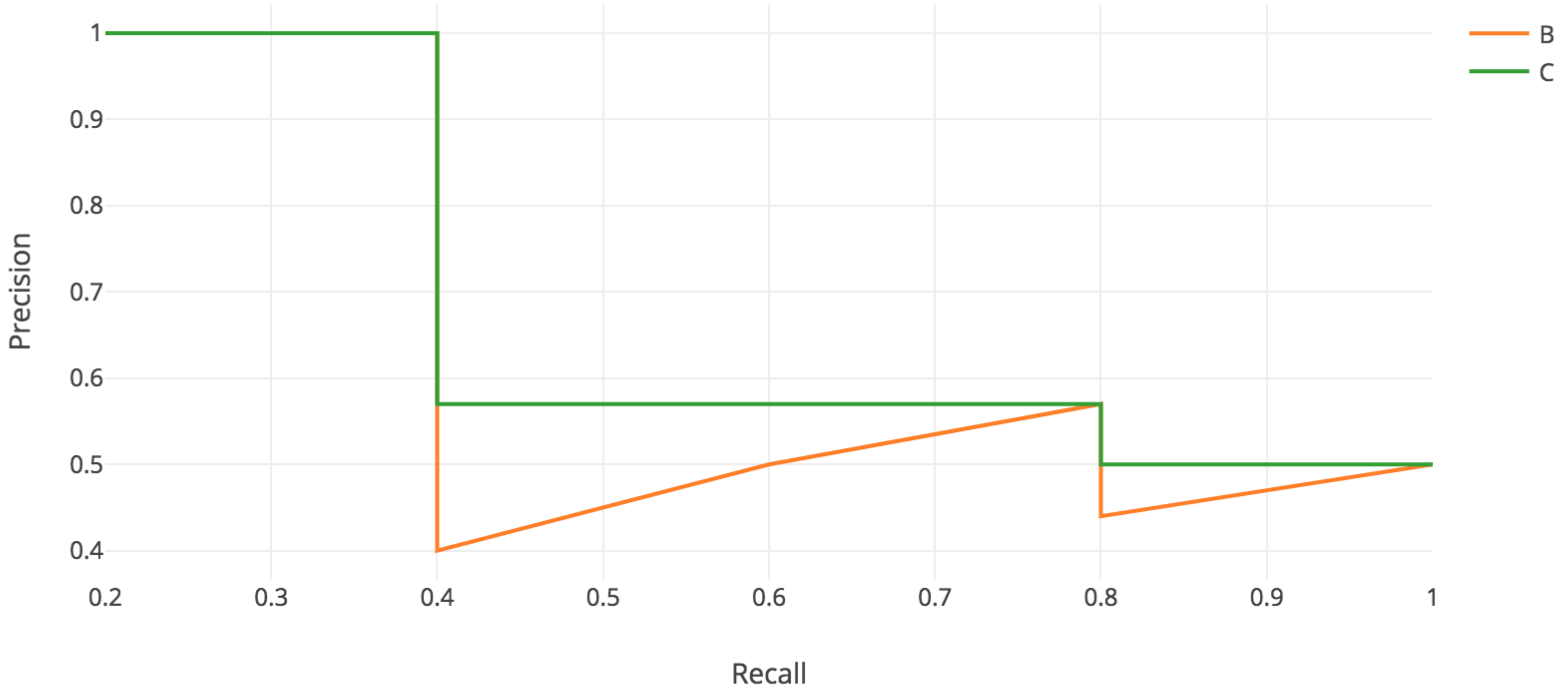
Object Detection Metrics

❖ **Example**



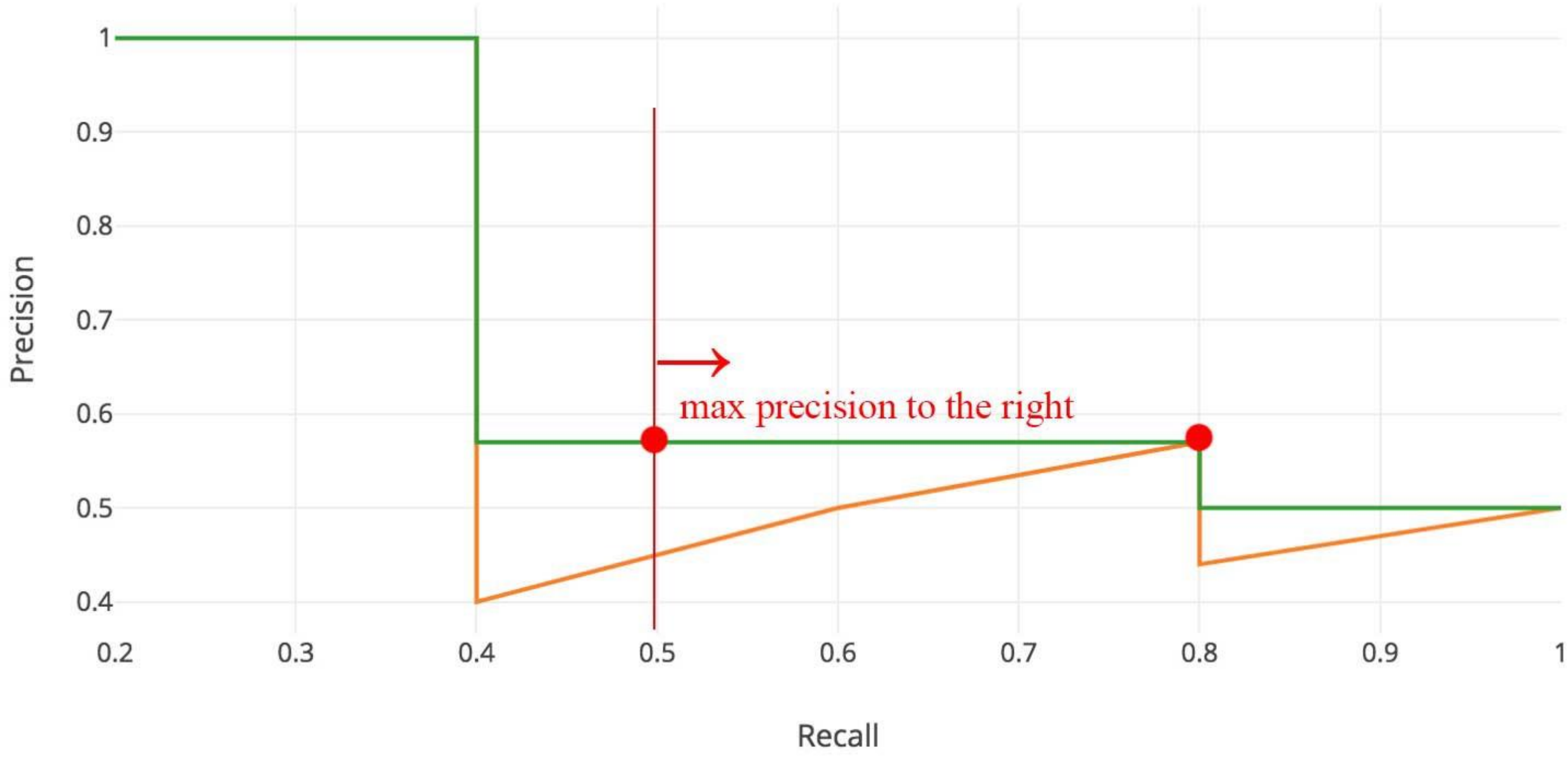
Object Detection Metrics

- ❖ **Example**
  - ❖ **Smooth out the zigzag pattern**



Object Detection Metrics

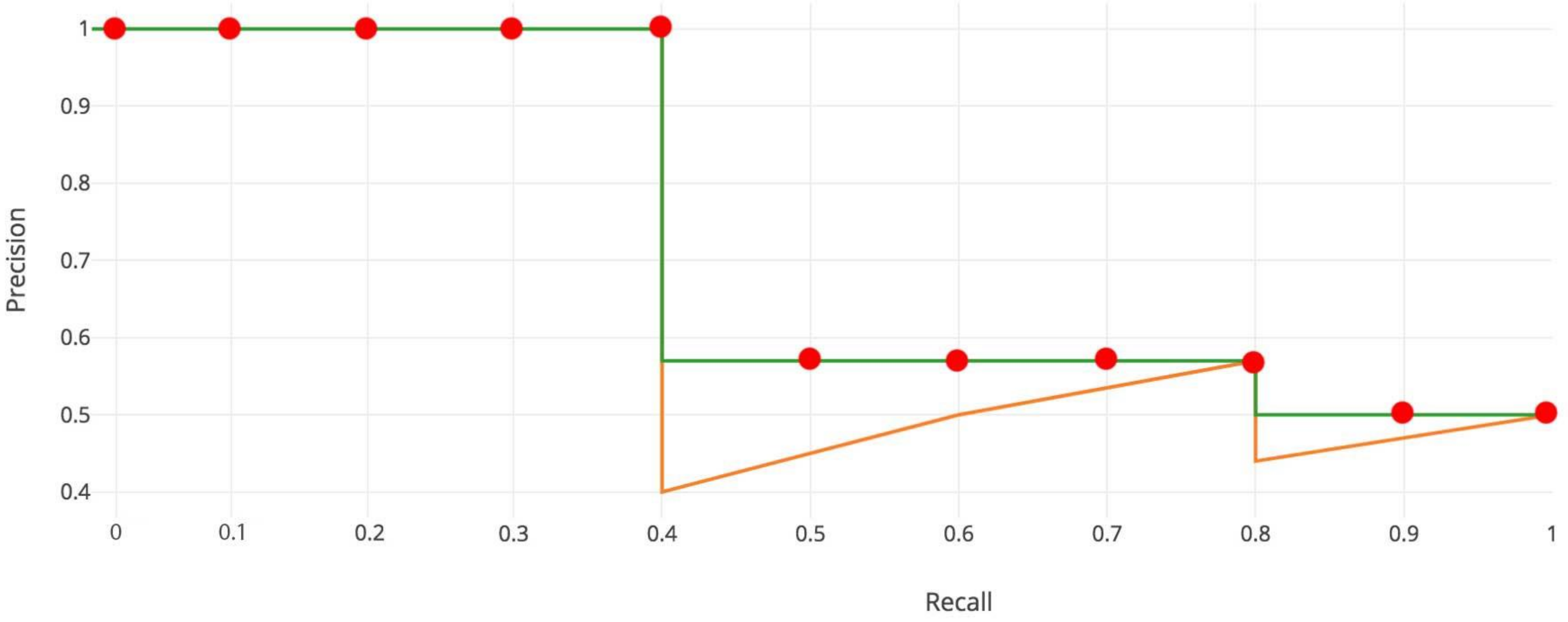
- ❖ **Example**
  - ❖ **Smooth out the zigzag pattern**



Object Detection Metrics

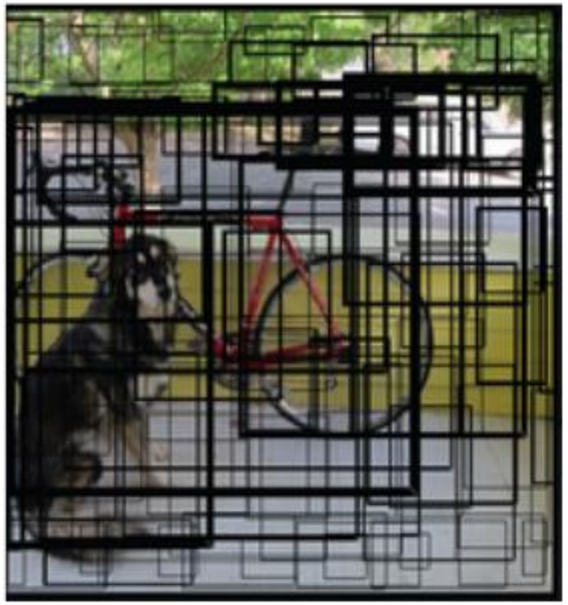
- ❖ **Example**
  - ❖ **Pascal VOC2008**

$$AP = \frac{1}{11} \sum_{r \in \{0.0,\ldots,1.0\}} AP_r$$
$$= \frac{1}{11} \sum_{r \in \{0.0,\ldots,1.0\}} p_{interp}(r)$$

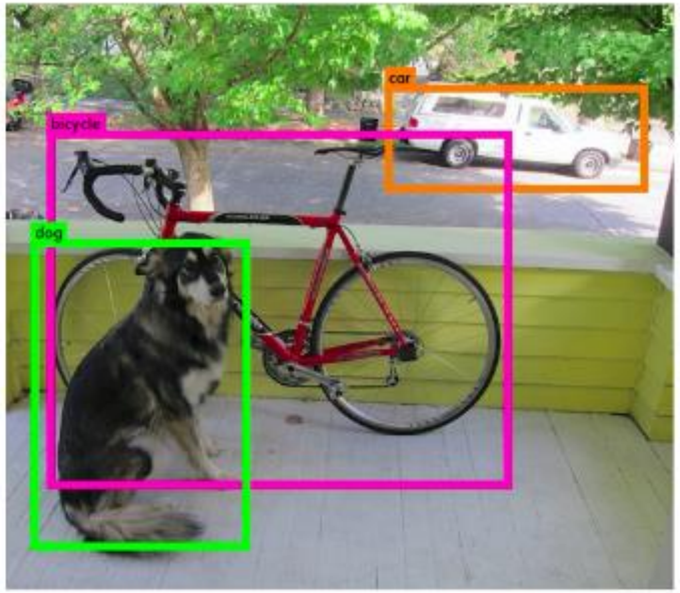


Non-max Suppression

❖ Motivation



Multiple Bounding Boxes

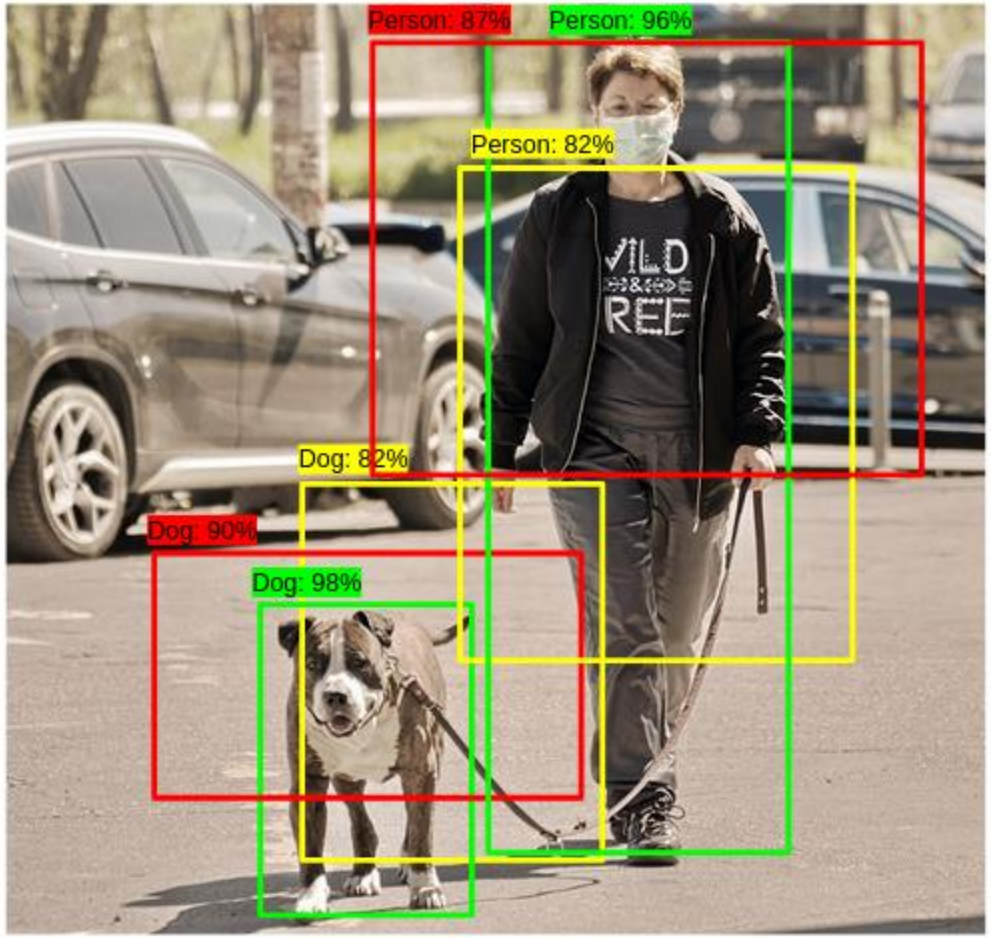


Final Bounding Boxes

https://pjreddie.com/darknet/yolov1/

Non-max Suppression

- ❖ Confidence score
- ❖ IoU



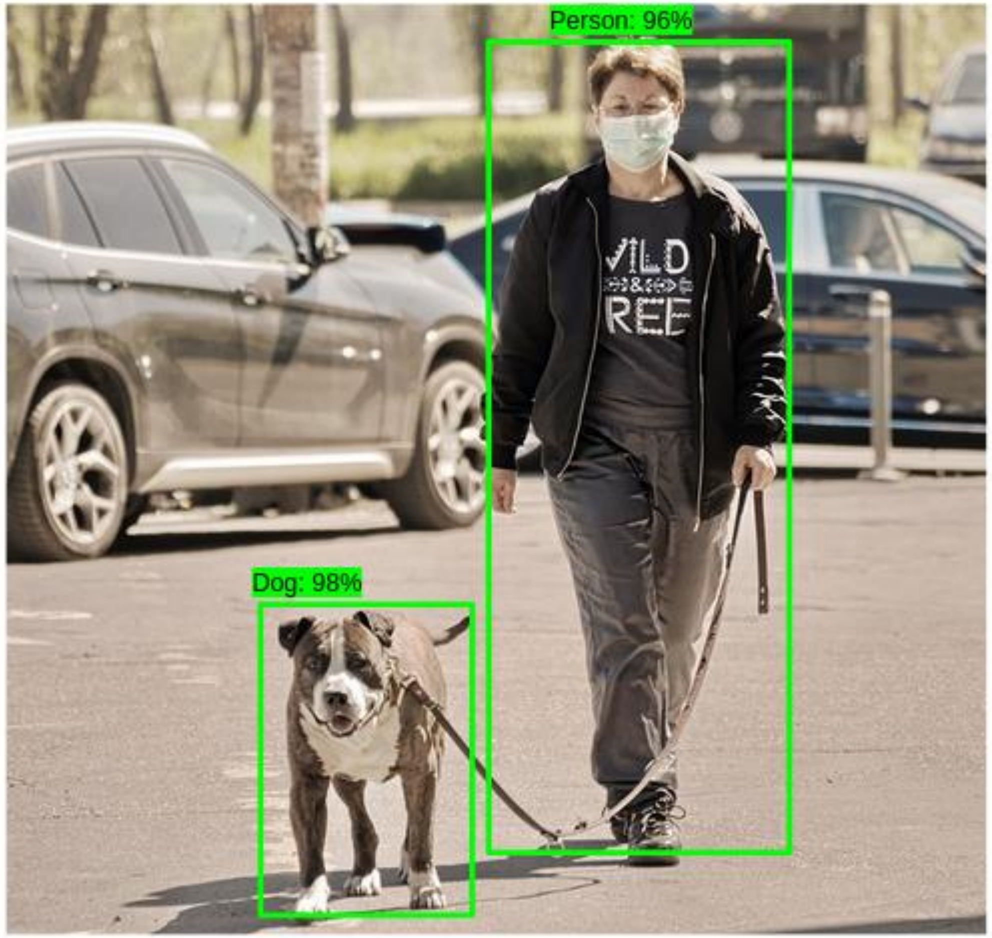
https://www.analyticsvidhya.com/blog/2020/08/selecting-the-right-bounding-box-using-non-max-suppression-with-implementation/

Non-max Suppression

❖ Procedure

Select the bounding box with the highest confidence score

Remove all the other boxes with high overlap



Non-max Suppression

❖ Procedure

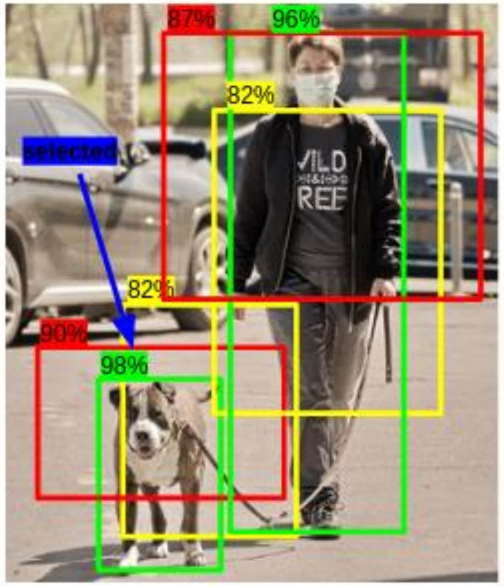
Step 1: Select the box with highest objectiveness score

Step 2: Then, compare the overlap (intersection over union) of this box with other boxes

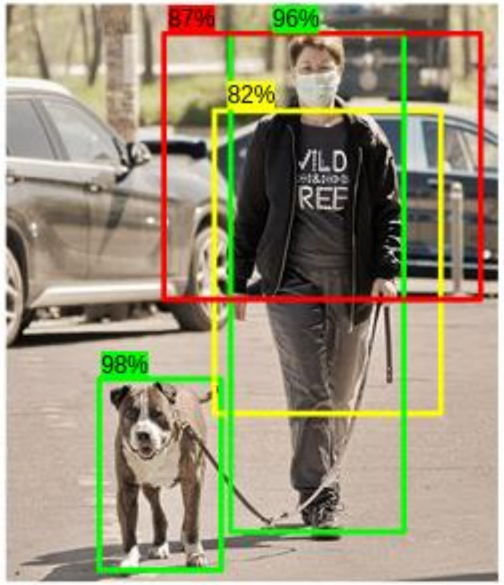
Step 3: Remove the bounding boxes with overlap (intersection over union) >50%

Step 4: Then, move to the next highest objectiveness score

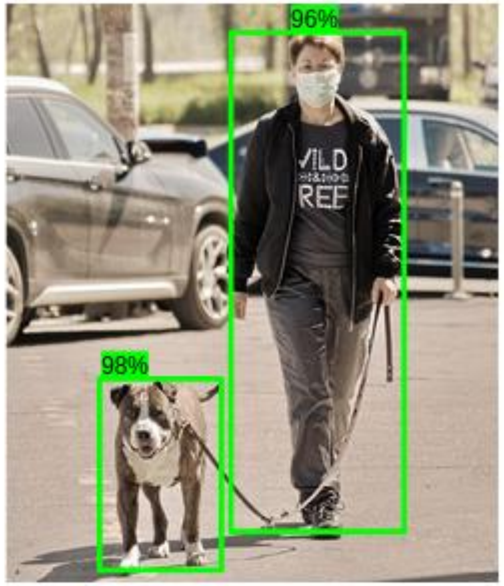
Step 5: Finally, repeat steps 2-4



Step 1: Selecting Bounding box with highest score



Step 3: Delete Bounding box with high overlap



Step 5: Final Output

VOC2007 Dataset

- **20 categories**

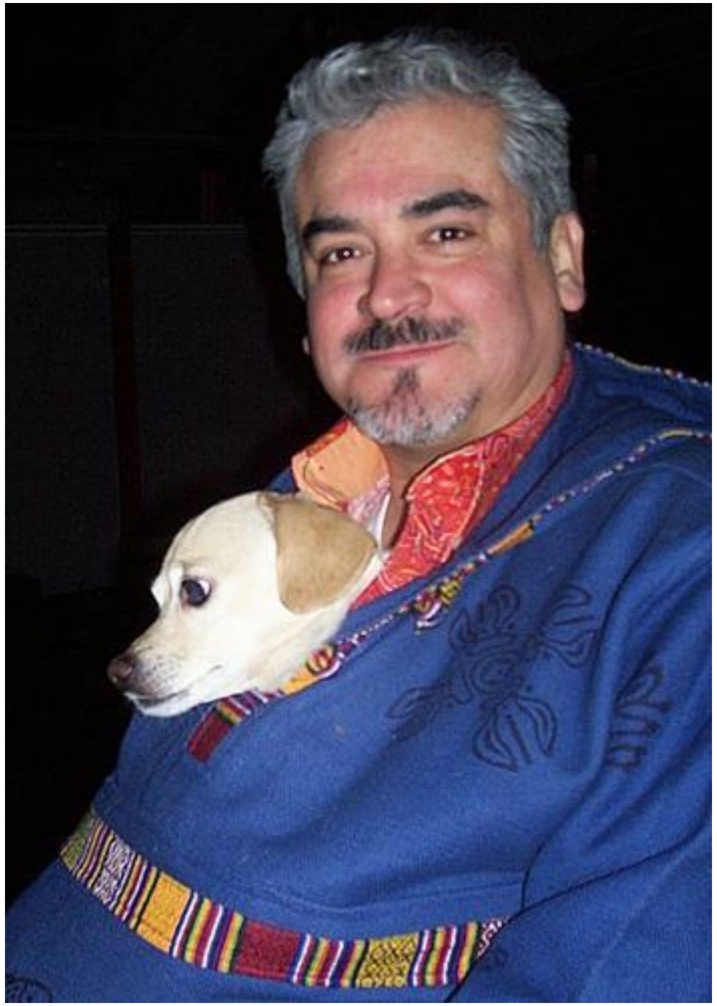
Person: person
Animal: bird, cat, cow, dog, horse, sheep
Vehicle: aeroplane, bicycle, boat, bus, car, motorbike, train
Indoor: bottle, chair, dining table, potted plant, sofa, tv/monitor

VOC2007 Dataset

- 20 categories

VOC2007 Dataset

❖ **Example**




```xml
<annotation>
    <folder>VOC2007</folder>
    <filename>000001.jpg</filename>
    <source>
        <database>The VOC2007 Database</database>
        <annotation>PASCAL VOC2007</annotation>
        <image>flickr</image>
        <flickrid>341012865</flickrid>
    </source>
    <owner>
        <flickrid>Fried Camels</flickrid>
        <name>Jinky the Fruit Bat</name>
    </owner>
    <size>
        <width>353</width>
        <height>500</height>
        <depth>3</depth>
    </size>
    <segmented>0</segmented>
    <object>
        <name>dog</name>
        <pose>Left</pose>
        <truncated>1</truncated>
        <difficult>0</difficult>
        <bndbox>
            <xmin>48</xmin>
            <ymin>240</ymin>
            <xmax>195</xmax>
            <ymax>371</ymax>
        </bndbox>
    </object>
    <object>
        <name>person</name>
        <pose>Left</pose>
        <truncated>1</truncated>
        <difficult>0</difficult>
        <bndbox>
            <xmin>8</xmin>
            <ymin>12</ymin>
            <xmax>352</xmax>
            <ymax>498</ymax>
        </bndbox>
    </object>
</annotation>
```

XML File Processing

❖ ElementTree

```xml
<?xml version="1.0"?>
<data>
    <country name="Liechtenstein">
        <rank>1</rank>
        <year>2008</year>
        <gdppc>141100</gdppc>
        <neighbor name="Austria" direction="E"/>
        <neighbor name="Switzerland" direction="W"/>
    </country>
    <country name="Singapore">
        <rank>4</rank>
        <year>2011</year>
        <gdppc>59900</gdppc>
        <neighbor name="Malaysia" direction="N"/>
    </country>
    <country name="Panama">
        <rank>68</rank>
        <year>2011</year>
        <gdppc>13600</gdppc>
        <neighbor name="Costa Rica" direction="W"/>
        <neighbor name="Colombia" direction="E"/>
    </country>
</data>
```

```python
import xml.etree.ElementTree as ET
tree = ET.parse('country_data.xml')
root = tree.getroot()
```

```python
for neighbor in root.iter('neighbor'):
    print(neighbor.attrib)
```

```
{'name': 'Austria', 'direction': 'E'}
{'name': 'Switzerland', 'direction': 'W'}
{'name': 'Malaysia', 'direction': 'N'}
{'name': 'Costa Rica', 'direction': 'W'}
{'name': 'Colombia', 'direction': 'E'}
```

XML File Processing

❖ ElementTree

```xml
<?xml version="1.0"?>
<data>
    <country name="Liechtenstein">
        <rank>1</rank>
        <year>2008</year>
        <gdppc>141100</gdppc>
        <neighbor name="Austria" direction="E"/>
        <neighbor name="Switzerland" direction="W"/>
    </country>
    <country name="Singapore">
        <rank>4</rank>
        <year>2011</year>
        <gdppc>59900</gdppc>
        <neighbor name="Malaysia" direction="N"/>
    </country>
    <country name="Panama">
        <rank>68</rank>
        <year>2011</year>
        <gdppc>13600</gdppc>
        <neighbor name="Costa Rica" direction="W"/>
        <neighbor name="Colombia" direction="E"/>
    </country>
</data>
```

```python
for neighbor in root.iter('neighbor'):
    print(neighbor.attrib['name'])
```

```
Austria
Switzerland
Malaysia
Costa Rica
Colombia
```

```python
for rank in root.iter('rank'):
    print(rank.text)
```

```
1
4
68
```

```python
for country in root.iter('country'):
    rank = country.find('rank').text
    print(rank)
```

```
1
4
68
```

xml_processing.ipynb

VOC2007 Dataset

❖ Data Processing

'aeroplane': 0	'bus': 5	'diningtable': 10	'pottedplant': 15
'bicycle': 1	'car': 6	'dog': 11	'sheep': 16
'bird': 2	'cat': 7	'horse': 12	'sofa': 17
'boat': 3	'chair': 8	'motorbike': 13	'train': 18
'bottle': 4	'cow': 9	'person': 14	'tv/monitor': 19

VOC2007 Dataset

❖ Data Processing

Name

- Annotations
- ImageSets
- JPEGImages
- SegmentationClass
- SegmentationObject

Name

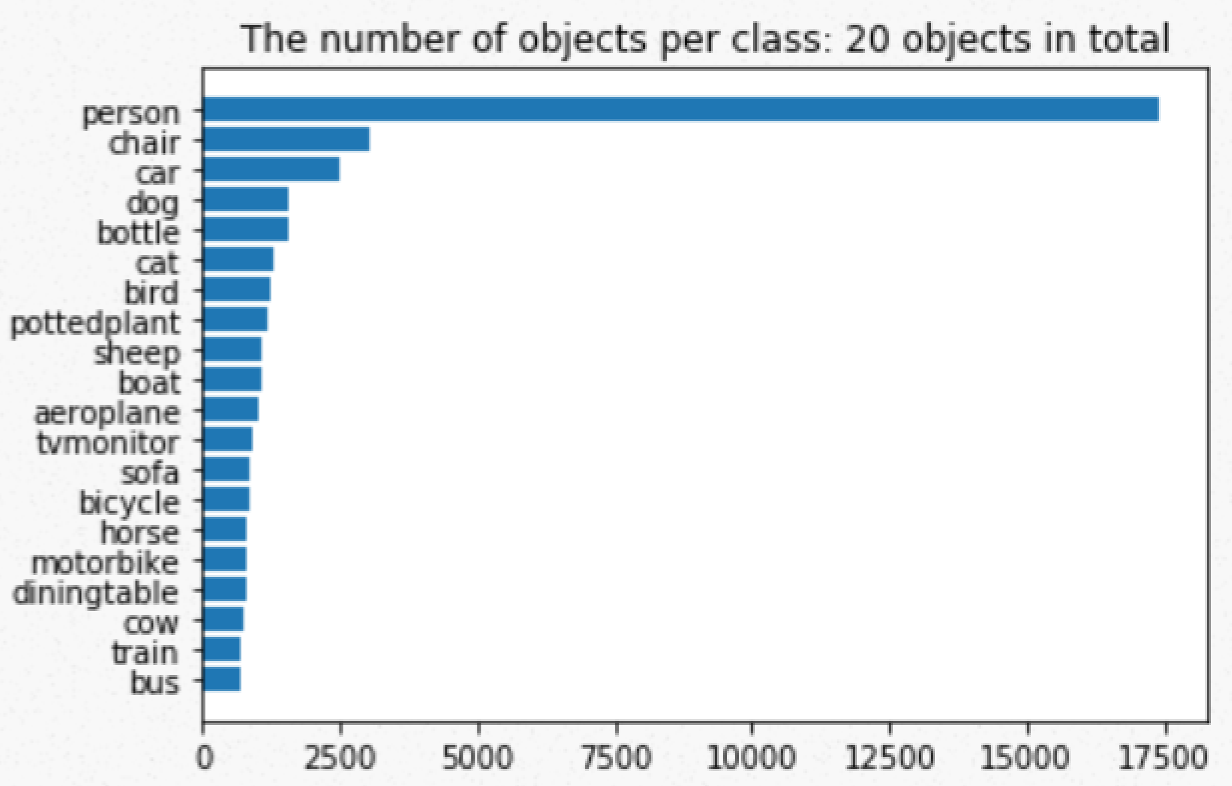
- 000001.xml
- 000002.xml
- 000003.xml
- 000004.xml
- 000005.xml
- 000006.xml
- 000007.xml
- 000008.xml
- 000009.xml
- 000010.xml

Name

- 000001.jpg
- 000002.jpg
- 000003.jpg
- 000004.jpg
- 000005.jpg
- 000006.jpg
- 000007.jpg
- 000008.jpg
- 000009.jpg
- 000010.jpg

VOC2007 Dataset

❖ **Statistics**



	train		val		trainval	
	Images	**Objects**	**Images**	**Objects**	**Images**	**Objects**
Aeroplane	112	151	126	155	238	306
Bicycle	116	176	127	177	243	353
Bird	180	243	150	243	330	486
Boat	81	140	100	150	181	290
Bottle	139	253	105	252	244	505
Bus	97	115	89	114	186	229
Car	376	625	337	625	713	1250
Cat	163	186	174	190	337	376
Chair	224	400	221	398	445	798
Cow	69	136	72	123	141	259
Diningtable	97	103	103	112	200	215
Dog	203	253	218	257	421	510
Horse	139	182	148	180	287	362
Motorbike	120	167	125	172	245	339
Person	1025	2358	983	2332	2008	4690
Pottedplant	133	248	112	266	245	514
Sheep	48	130	48	127	96	257
Sofa	111	124	118	124	229	248
Train	127	145	134	152	261	297
Tvmonitor	128	166	128	158	256	324
Total	2501	6301	2510	6307	5011	12608

VOC2007 Dataset

❖ Data Processing

```python
import matplotlib.pyplot as plt
import cv2
import numpy as np

# read an image
image = cv2.imread('VOCdevkit/VOC2007/JPEGImages/000026.jpg')
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
print(image.shape)

# draw bounding boxes
color = (255, 0, 0)
thickness = 2
image = cv2.rectangle(image, (90,125), (337,212),  color, thickness)

# plot image
fig = plt.figure()
plt.imshow(image/255.0)
```

```
(333, 500, 3)
```

VOC2007Dataset.ipynb

VOC2007 Dataset

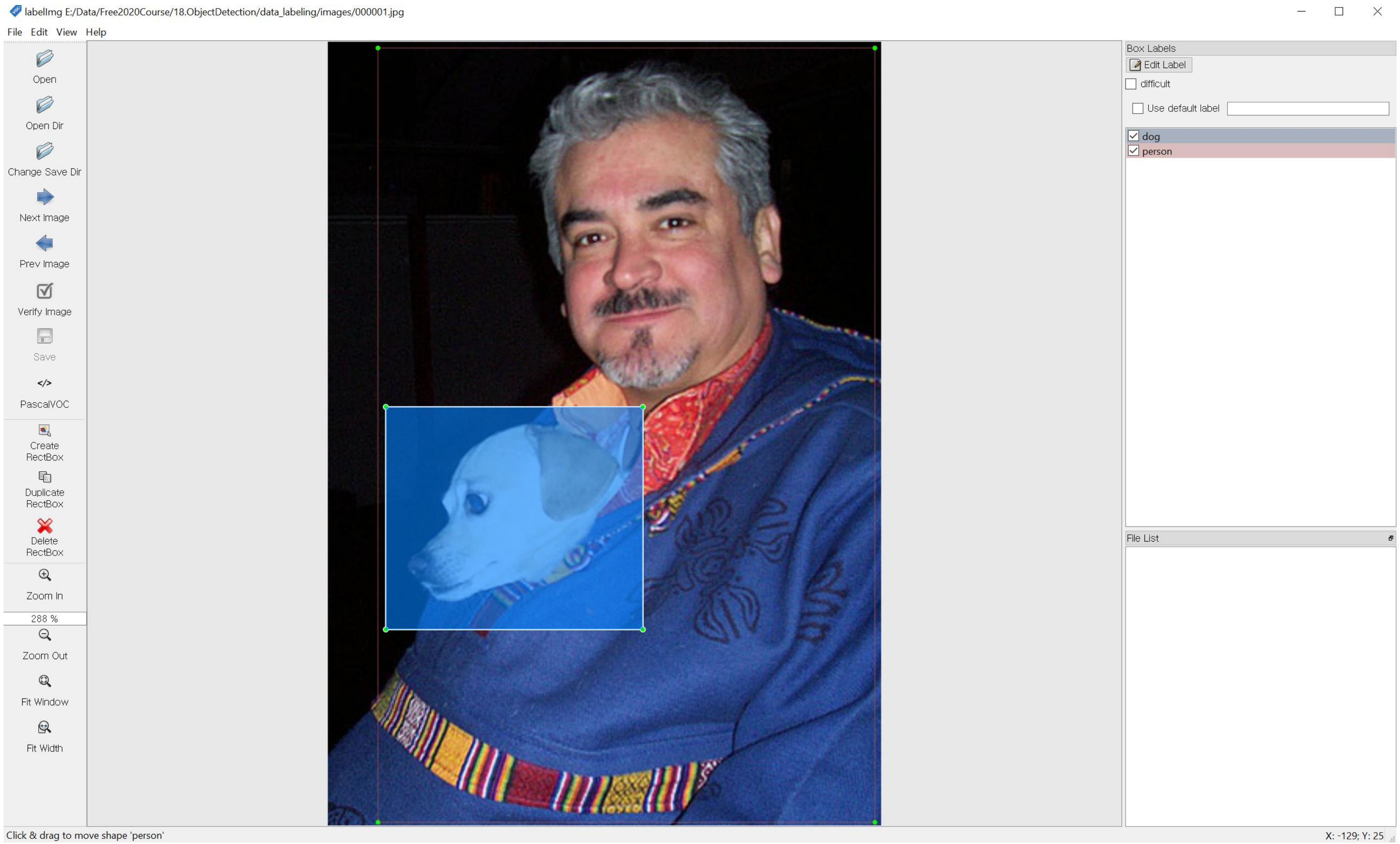
❖ Data Processing

```
VOCdevkit/VOC2007/JPEGImages/000153.jpg 237,147,358,191,6
VOCdevkit/VOC2007/JPEGImages/000154.jpg 59,76,367,266,3
VOCdevkit/VOC2007/JPEGImages/000159.jpg 234,48,286,124,14 1,16,498,333,6
VOCdevkit/VOC2007/JPEGImages/000161.jpg 104,34,446,390,6 68,195,121,288,6
VOCdevkit/VOC2007/JPEGImages/000162.jpg 306,227,380,299,19 196,143,309,369,14
VOCdevkit/VOC2007/JPEGImages/000163.jpg 52,22,308,328,14 26,108,456,396,13
VOCdevkit/VOC2007/JPEGImages/000164.jpg 114,154,369,348,13 292,49,446,370,14
VOCdevkit/VOC2007/JPEGImages/000171.jpg 1,290,128,407,11 94,21,375,491,14
VOCdevkit/VOC2007/JPEGImages/000173.jpg 106,64,270,297,14 109,64,288,464,12
VOCdevkit/VOC2007/JPEGImages/000174.jpg 143,5,426,333,14
VOCdevkit/VOC2007/JPEGImages/000187.jpg 1,95,240,336,19
VOCdevkit/VOC2007/JPEGImages/000189.jpg 65,39,459,346,2
VOCdevkit/VOC2007/JPEGImages/000192.jpg 116,64,356,375,14
VOCdevkit/VOC2007/JPEGImages/000193.jpg 80,4,500,375,14 1,29,227,375,14
VOCdevkit/VOC2007/JPEGImages/000194.jpg 86,36,239,224,12 115,19,203,136,14 279,77,298,132,14
VOCdevkit/VOC2007/JPEGImages/000198.jpg 160,134,286,239,18
```

VOC2007Dataset-v1.ipynb

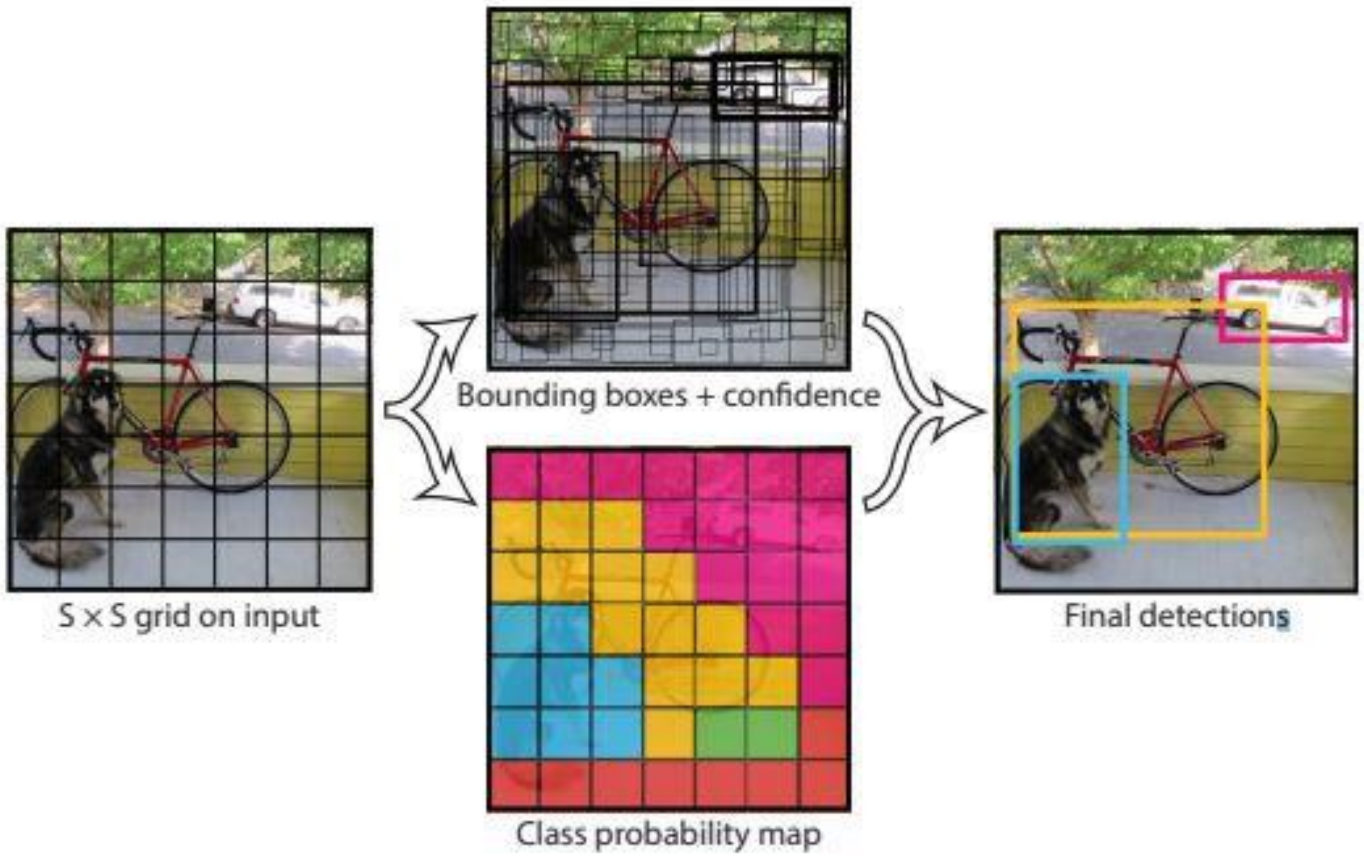
VOC2007 Dataset

❖ Data Labelling



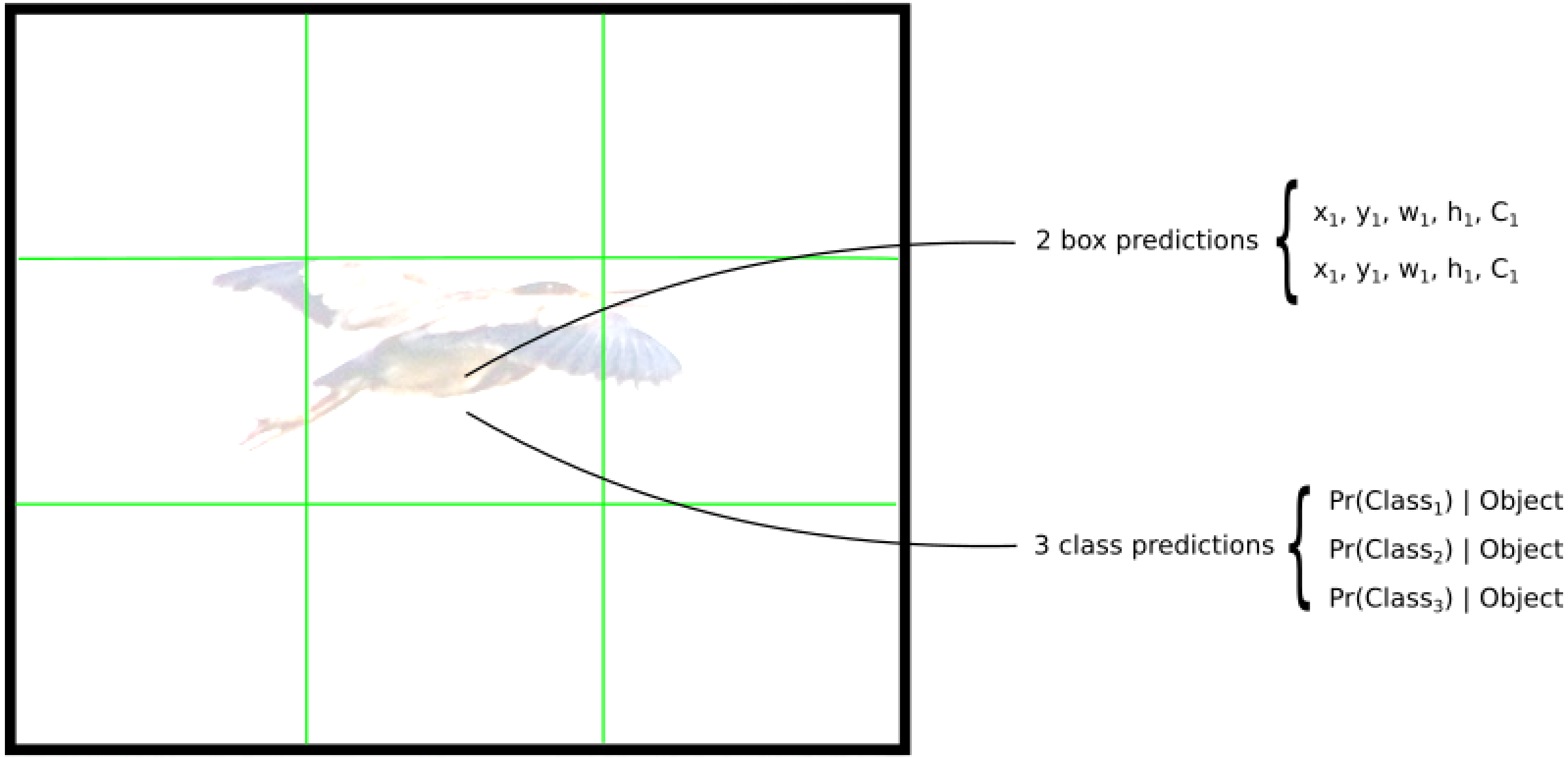
Yolo v1

❖ **The input image is divided into an S x S grid of cells**



Yolo v1

❖ **Each grid cell predicts B bounding boxes as well as C class probabilities.**



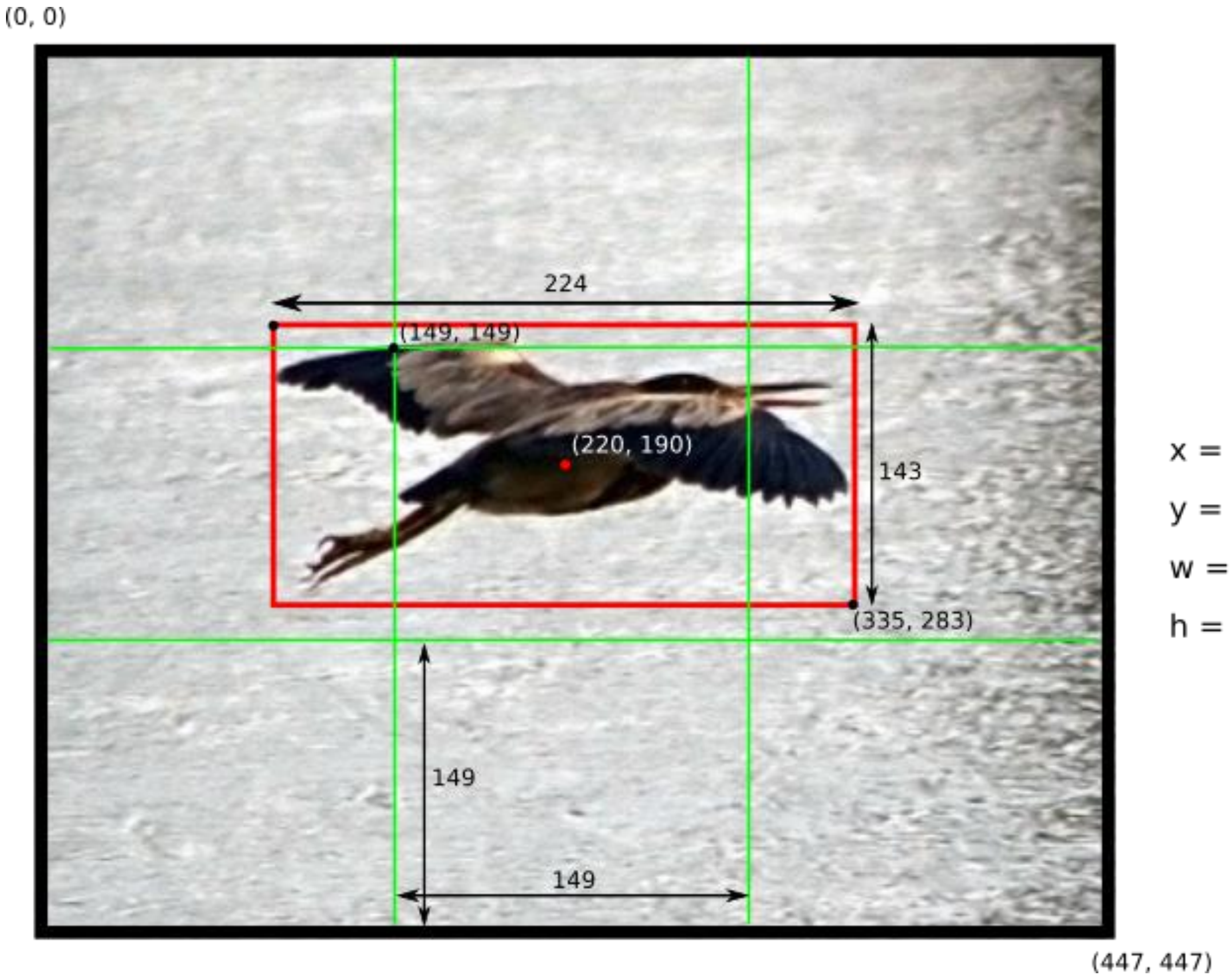
https://hackernoon.com/understanding-yolo-f5a74bbc7967

Yolo v1

❖ The bounding box prediction has 5 components: (x, y, w, h, confidence)

The (x, y) coordinates represent the center of the box, relative to the grid cell location.

These coordinates are normalized to fall between 0 and 1. The (w, h) box dimensions are also normalized to [0, 1], relative to the image size.



x = (220-149) / 149 = 0.48

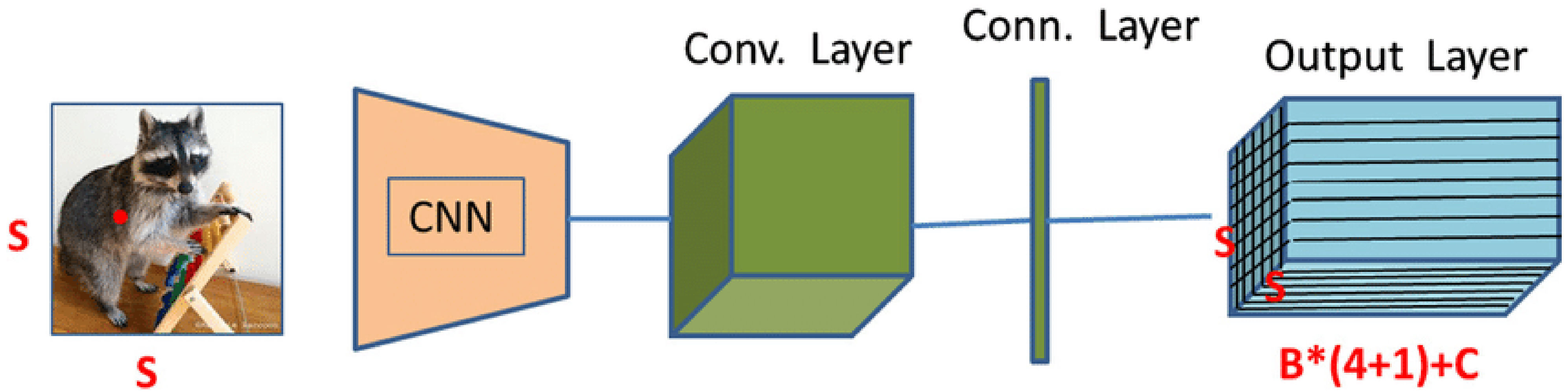
y = (190-149) / 149 = 0.28

w = 224 / 448 = 0.50

h = 143 / 448 = 0.32

Yolo v1

❖ Network output



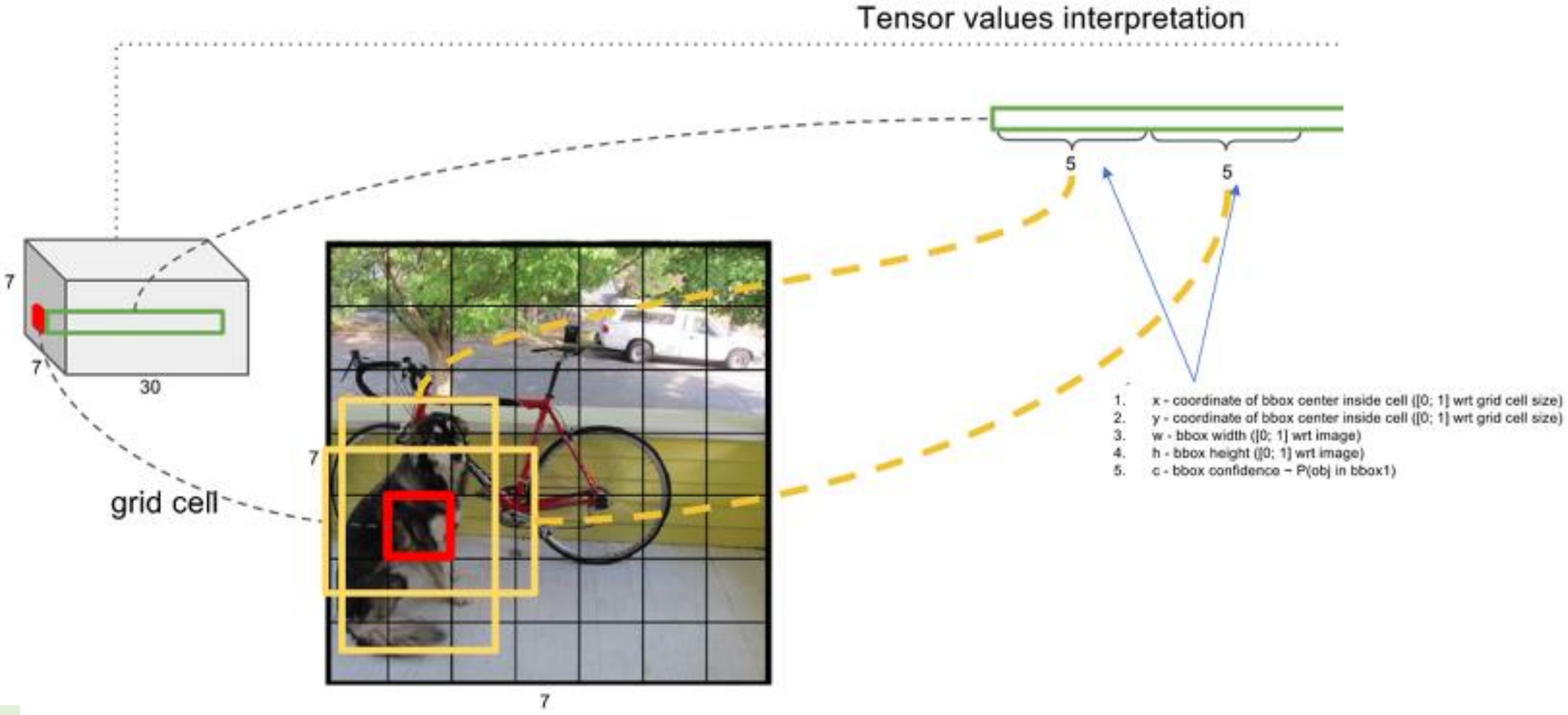
The network only predicts one set of class probabilities per cell, regardless of the number of boxes B.

That makes S x S x C class probabilities in total.

S*S*(B*5 +C) tensor

Yolo v1

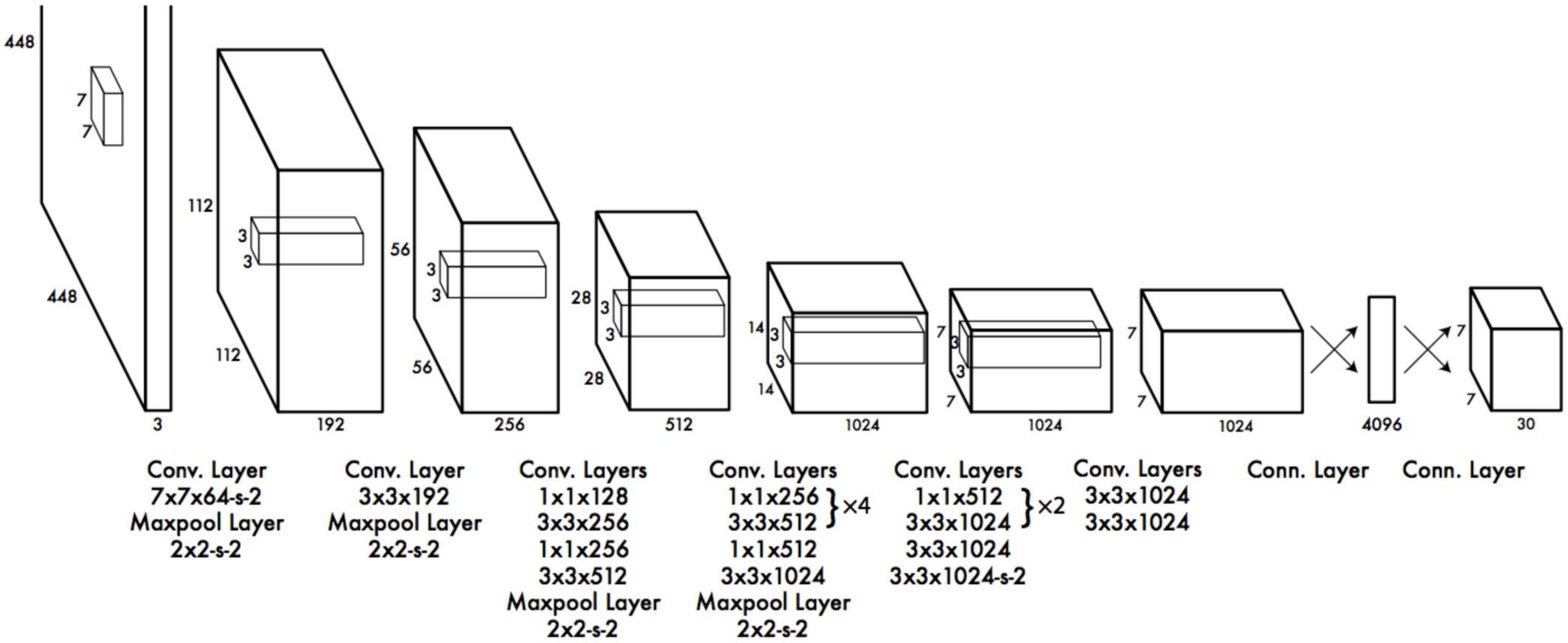
❖ Network output



Yolo v1

❖ Network architecture

http://localhost:8888/notebooks/Yolov1_architecture.ipynb



Designed for use in the Pascal VOC dataset (S=7, B=2 and C=20). The size of the output (7x7x(2*5+20)).

Yolo v1

❖ Loss function

$$\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} (x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2$$

$\mathbb{1}$ obj is defined as follows:

1, If an object is present in grid cell i and the jth bounding box predictor is “responsible” for that prediction

0, otherwise

(x, y) are the predicted bounding box position and (x̂, ŷ) hat are the actual position from the training data.

Yolo v1

❖ Loss function

$$\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} \left(\sqrt{w_i} - \sqrt{\hat{w}_i}\right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i}\right)^2$$

Our error metric should reflect that small deviations in large boxes matter less than in small boxes. To partially address this we predict the square root of the bounding box width and height instead of the width and height directly.

Yolo v1

❖ Loss function

$$\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{obj}(C_i-\hat{C}_i)^2+\lambda_{noobj}\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{noobj}(C_i-\hat{C}_i)^2$$

The loss associated with the confidence score for each bounding box predictor.

C is the confidence score and Ĉ is the intersection over union of the predicted bounding box with the ground truth.

𝟙 obj is equal to one when there is an object in the cell, and 0 otherwise.

𝟙 noobj is the opposite.

Yolo v1

❖ Loss function

$$\sum_{i=0}^{S^2} \mathbb{1}_i^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2$$

A normal sum-squared error for classification, except for the $\mathbb{1}$ obj term.

This term is used because so we don’t penalize classification error when no object is present on the cell

Yolo v1

❖ Loss function

$$\lambda \textbf{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$

$$+ \lambda \textbf{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} \left[(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2 \right]$$

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{obj} \left(C_i - \hat{C}_i \right)^2$$

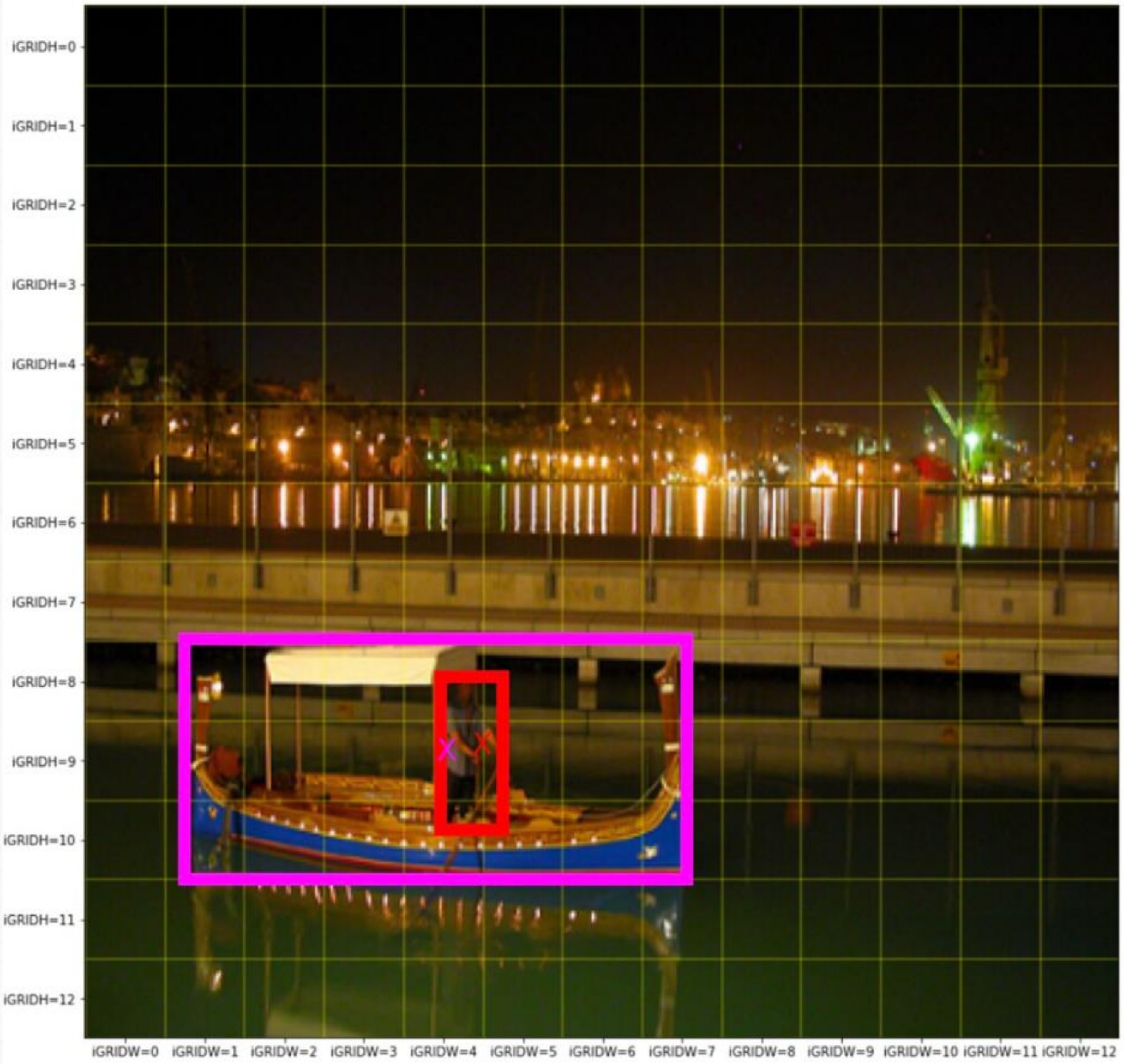
$$+ \lambda noobj \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{noobj} \left(C_i - \hat{C}_i \right)^2$$

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2$$

Yolo v2

❖ Anchor box

Multiple objects of various shapes within the same neighborhood



YOLO's Anchor box requires users to predefine two hyperparameters:

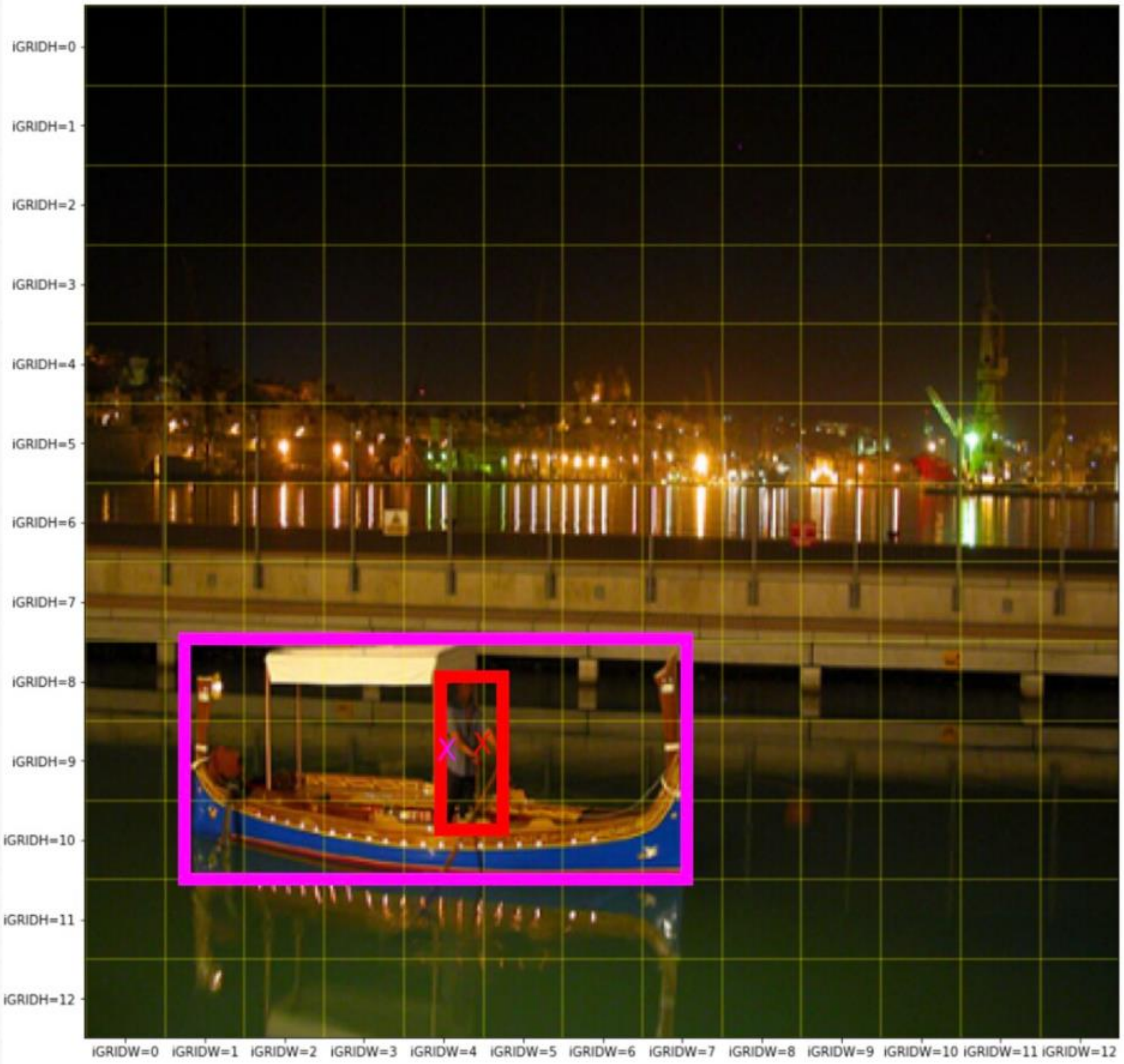
(1) The number of anchor boxs

(2) Their shapes

Yolo v2

❖ Anchor box

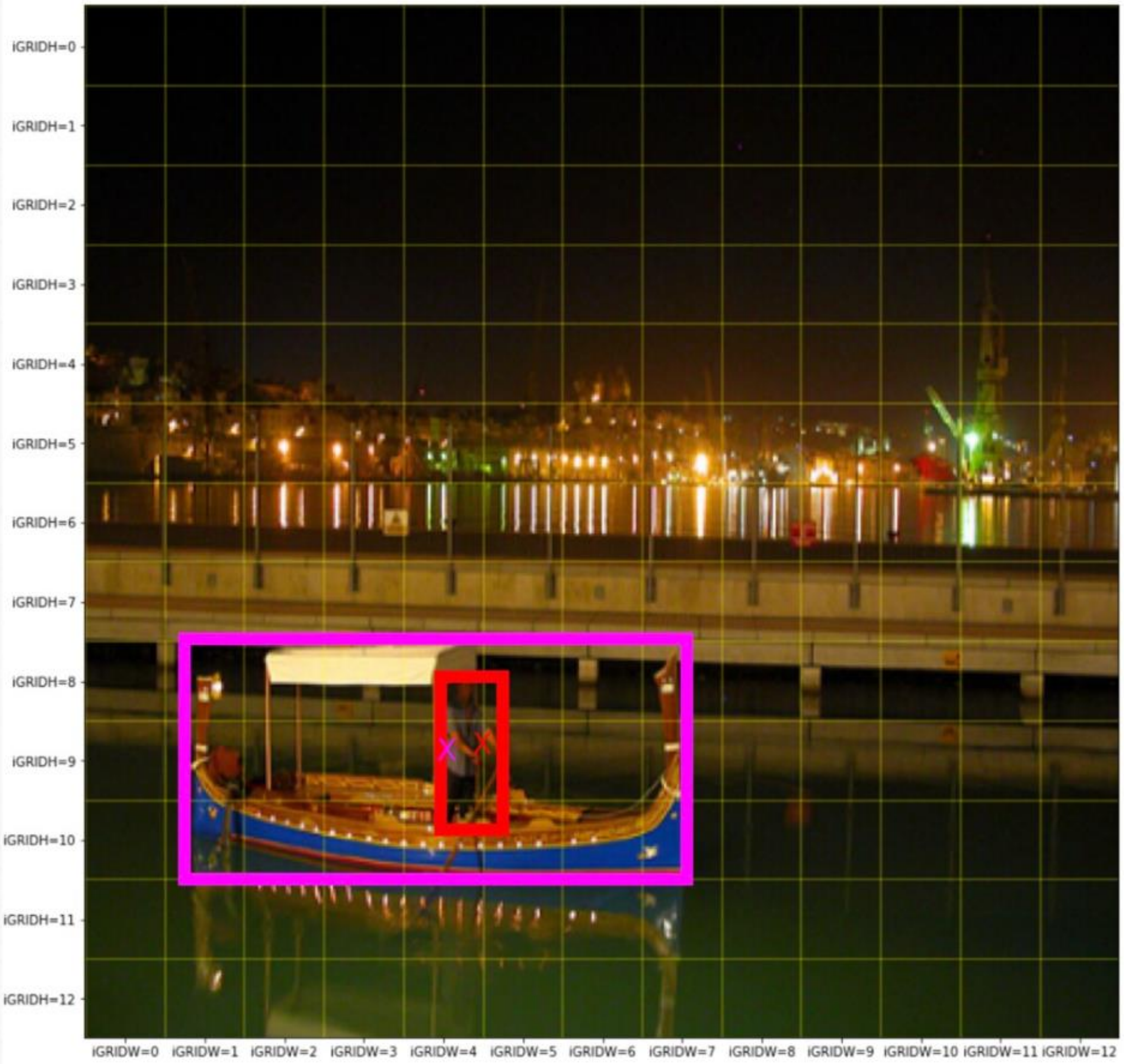
The more anchor boxes, the more objects YOLO can detect in a close neighborhood with the cost of more parameters in deep learning model.



Yolo v2

❖ Anchor box

- An anchor box specializes small tall rectangle bounding box
- Another anchor box specializes large flat rectangle bounding box



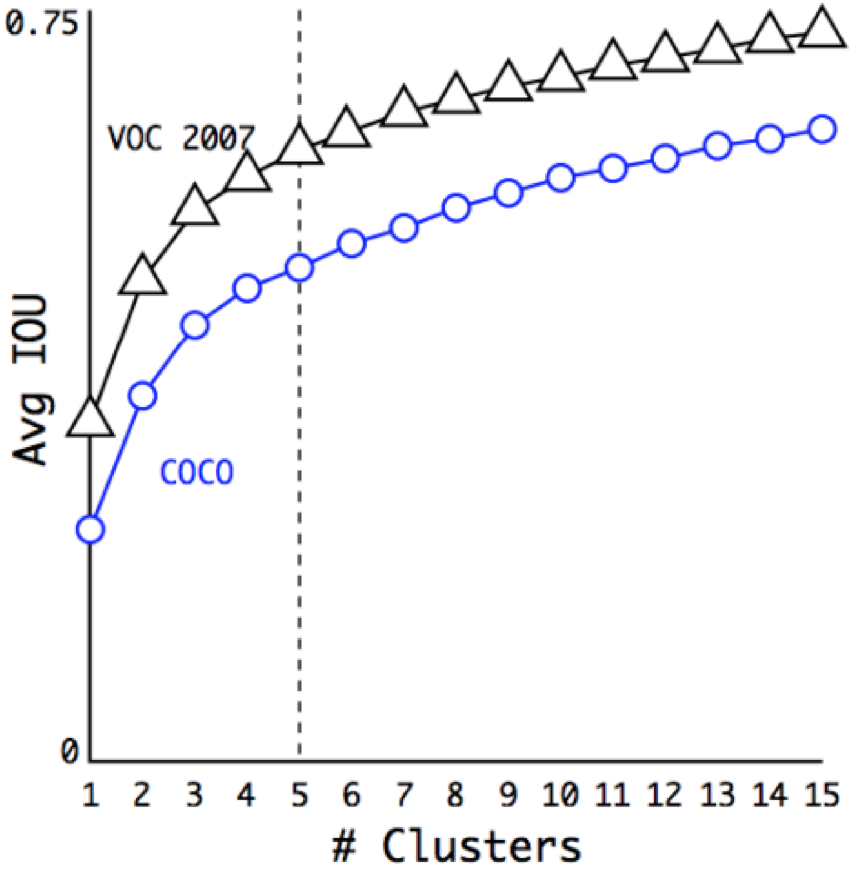
Yolo v2

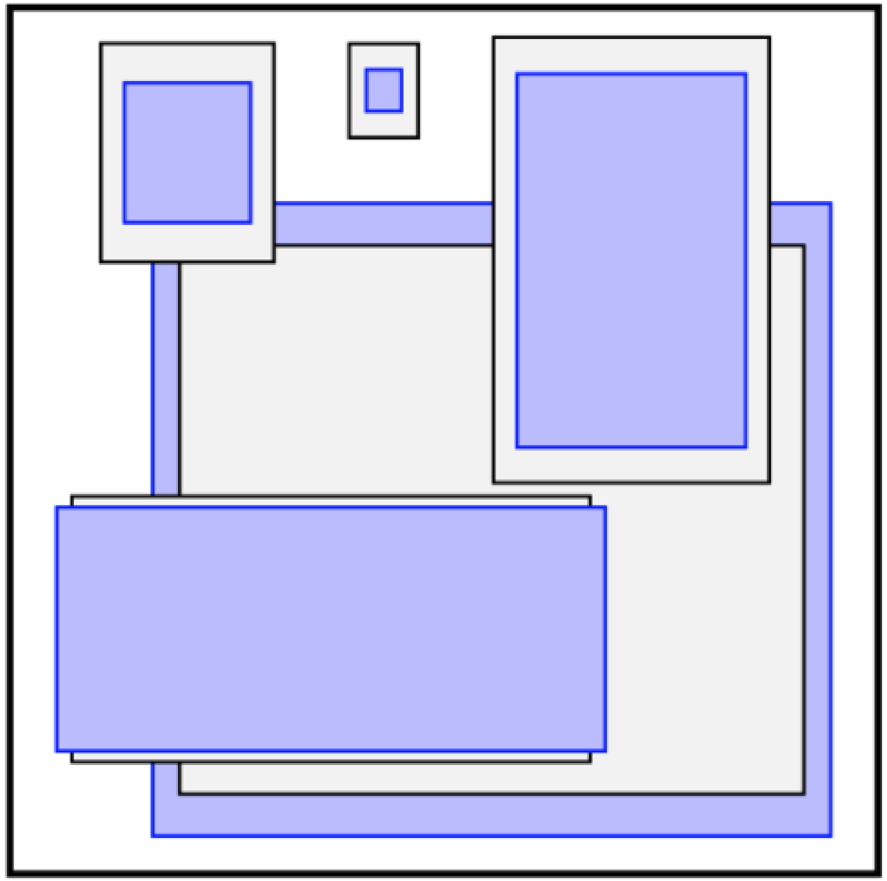
❖ Improvement

	YOLO								YOLOv2
batch norm?		✓	✓	✓	✓	✓	✓	✓	✓
hi-res classifier?			✓	✓	✓	✓	✓	✓	✓
convolutional?				✓	✓	✓	✓	✓	✓
anchor boxes?				✓	✓				
new network?					✓	✓	✓	✓	✓
dimension priors?						✓	✓	✓	✓
location prediction?						✓	✓	✓	✓
passthrough?							✓	✓	✓
multi-scale?								✓	✓
hi-res detector?									✓
VOC2007 mAP	63.4	65.8	69.5	69.2	69.6	74.4	75.4	76.8	**78.6**

Yolo v2

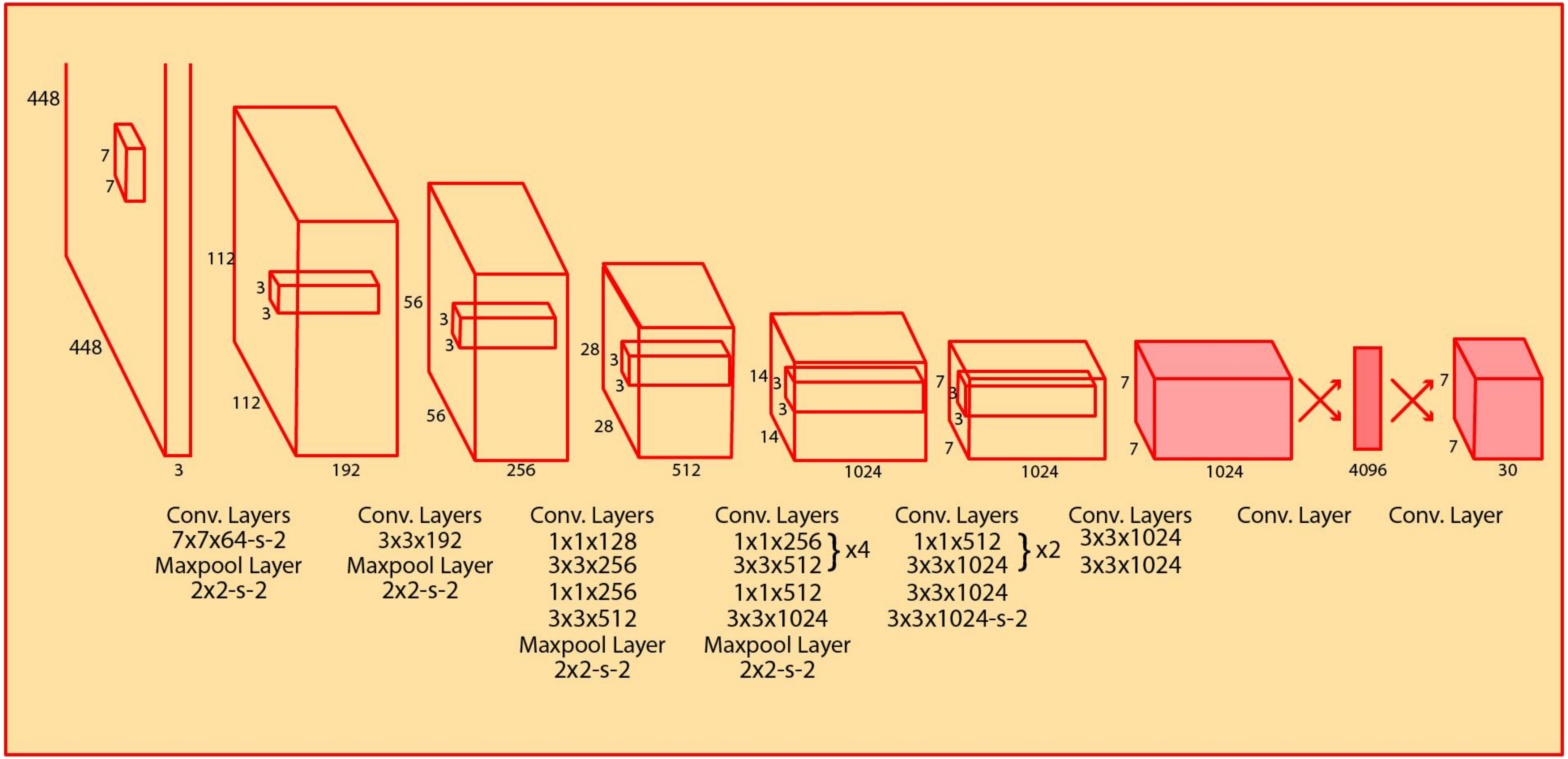
❖ Priors





Yolo v2

❖ Yolov2 architecture



Yolo v2

❖ Improvement

	YOLO								YOLOv2
batch norm?		✓	✓	✓	✓	✓	✓	✓	✓
hi-res classifier?			✓	✓	✓	✓	✓	✓	✓
convolutional?				✓	✓	✓	✓	✓	✓
anchor boxes?				✓	✓				
new network?					✓	✓	✓	✓	✓
dimension priors?						✓	✓	✓	✓
location prediction?						✓	✓	✓	✓
passthrough?							✓	✓	✓
multi-scale?								✓	✓
hi-res detector?									✓
VOC2007 mAP	63.4	65.8	69.5	69.2	69.6	74.4	75.4	76.8	**78.6**

Thank you!