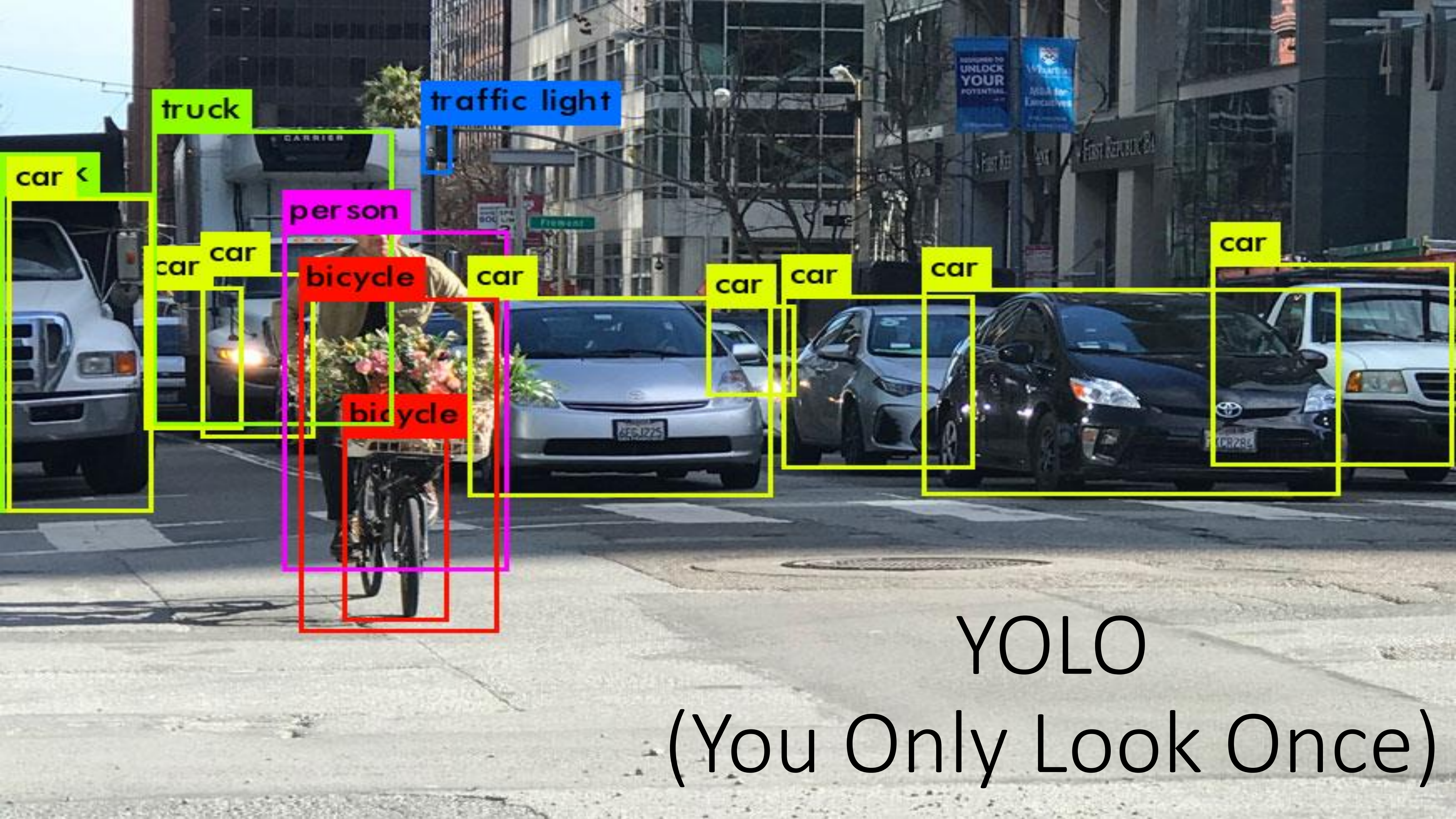


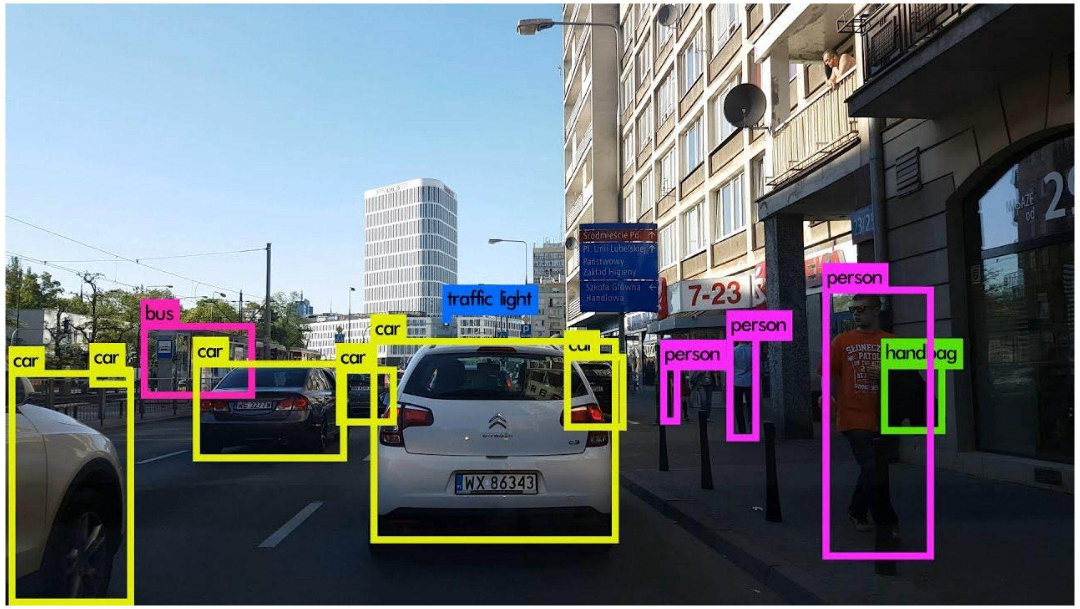
YOLO (You Only Look Once)

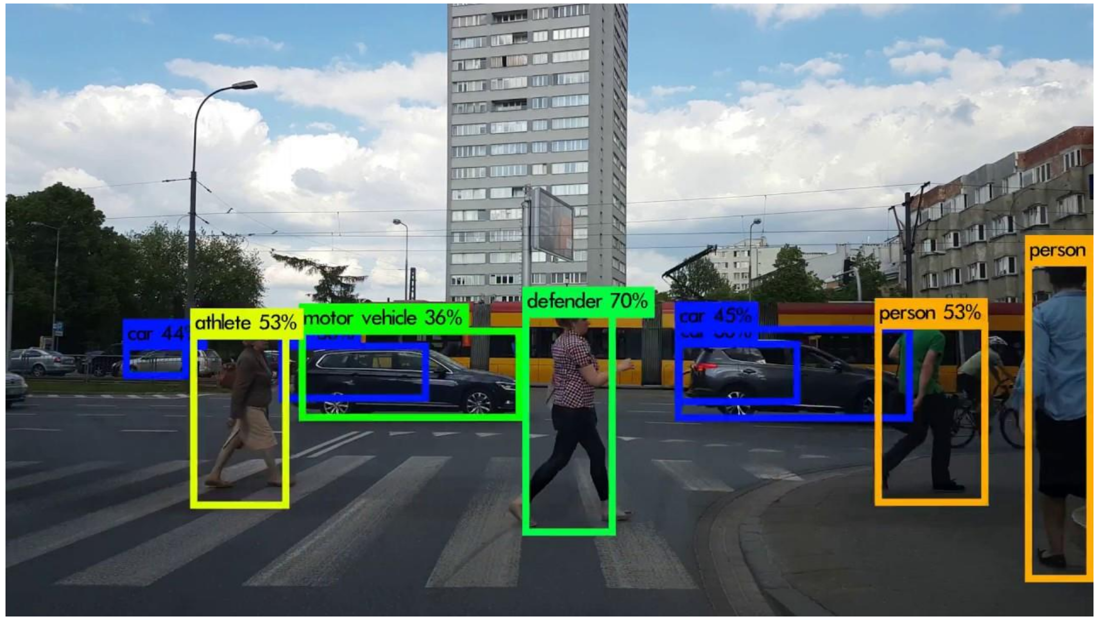
References

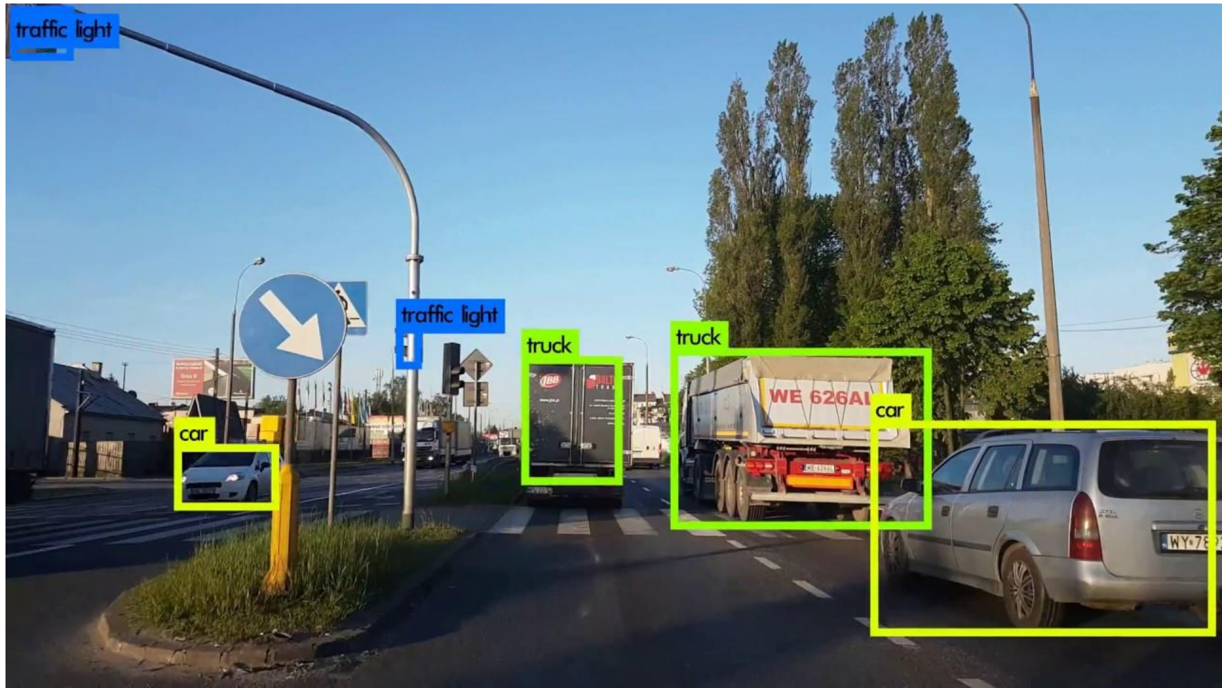
- https://blog.paperspace.com/how-to-implement-a-yolo-object-detector-in-pytorch/
- https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b
- https://heartbeat.fritz.ai/gentle-guide-on-how-yolo-object-localization-works-with-keras-part-2-65fe59ac12d
- https://medium.com/@jonathan_hui/real-time-object-detection-with-yolo-yolov2-28b1b93e2088

Outline of this Section

- Understanding the concept of YOLO
- Understanding other general concepts
- Understanding YOLO in depth
- Implementing YOLO in PyTorch



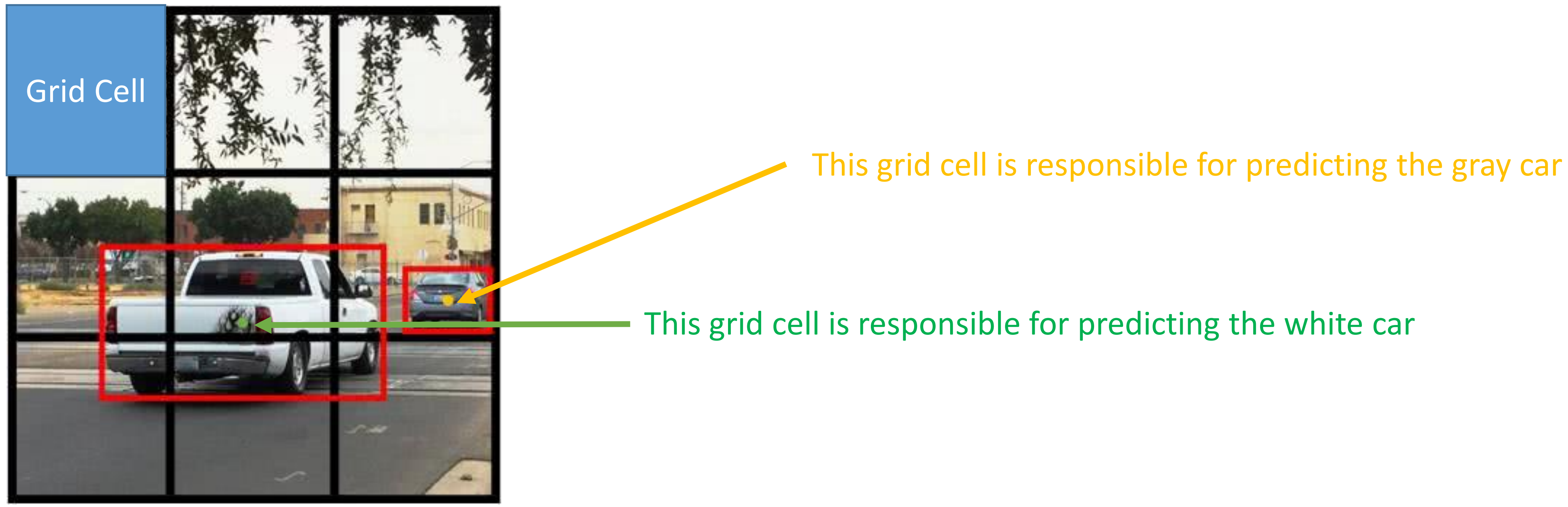




The Concept of YOLO

YOLO v2: https://www.youtube.com/watch?v=VOC3huqHrss
YOLO v3: https://www.youtube.com/watch?v=MPU2HistivI

- A model for Object Detection. It is very fast, and able to run in real-time. It has 3 versions.
- YOLO divides the input image into an **S×S** grid. Each grid cell predicts only **one** object.
- Let's look at this on a smaller scale. Divide this image into 3x3 Grid Cells (9 Grid Cells), and assign the center of the object to the grid cells it falls within. This grid cell is responsible for predicting the object.


Grid Cell
This grid cell is responsible for predicting the gray car
This grid cell is responsible for predicting the white car

For each grid cell

- The labels we have for training are:

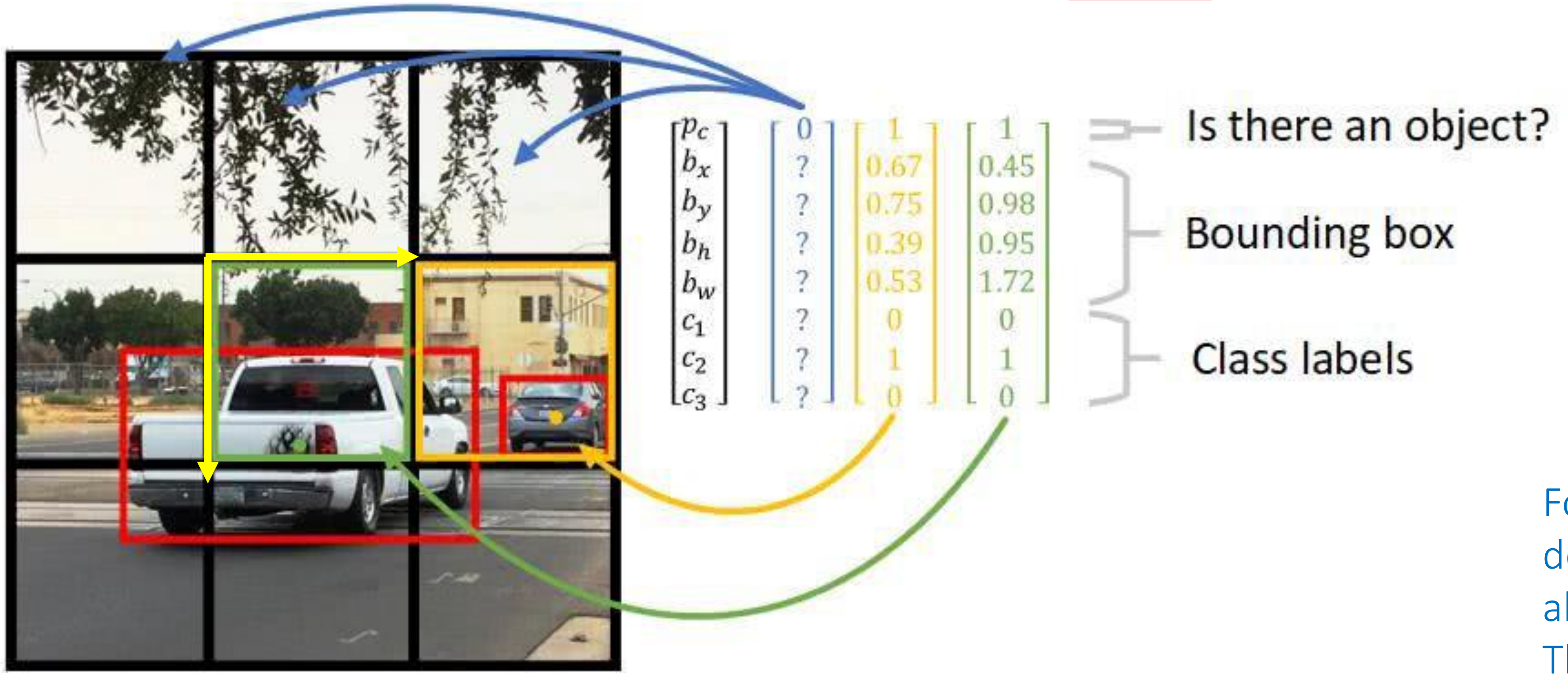
$$\boxed{3 \times 3 \times 8}$$

$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

If the grid cell contains an object

The bounding boxes of the object (if there is)

The number of classes (assume 3 for this example)

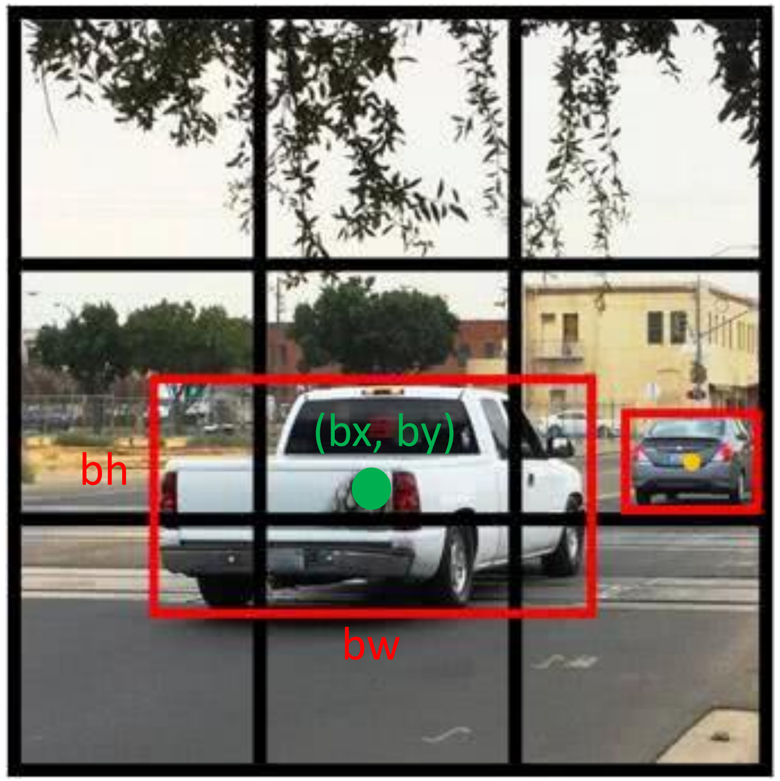


$\begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix}$ $\begin{bmatrix} 0 \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \\ ? \end{bmatrix}$ $\begin{bmatrix} 1 \\ 0.67 \\ 0.75 \\ 0.39 \\ 0.53 \\ 0 \\ 1 \\ 0 \end{bmatrix}$ $\begin{bmatrix} 1 \\ 0.45 \\ 0.98 \\ 0.95 \\ 1.72 \\ 0 \\ 1 \\ 0 \end{bmatrix}$
Is there an object?
Bounding box
Class labels

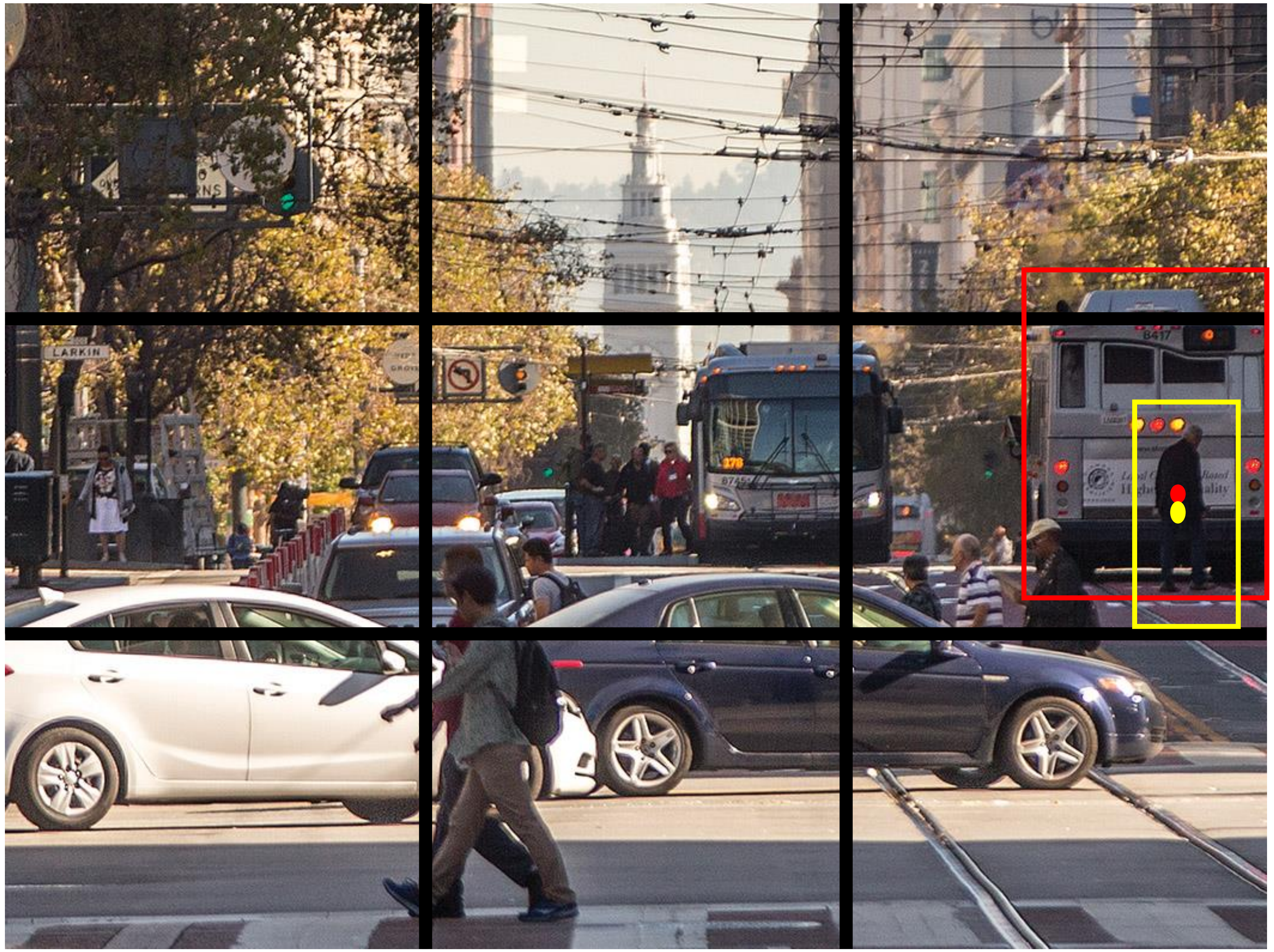
For those grid cells with no object detected, it's pc = 0 and we don't care about the rest of the other values. That's what the "?" means in the graph.

Bounding Boxes

- bx: the center of the object according to the **x** coordinate, and has a value ranging from 0~1,
- by: the center of the object according to the x coordinate, and has a value ranging from 0~1
- bh: **height** of the bounding box, the value could be greater than 1,
- bw: **width** of the bounding box, the value could be greater than 1

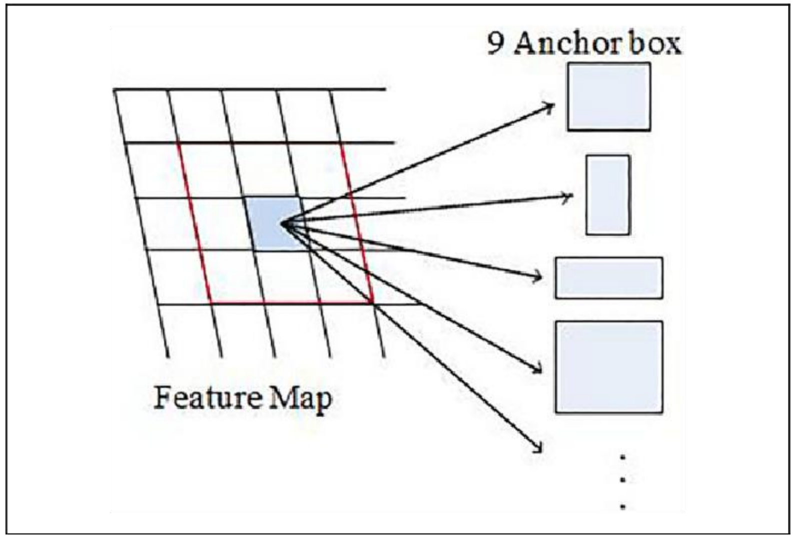


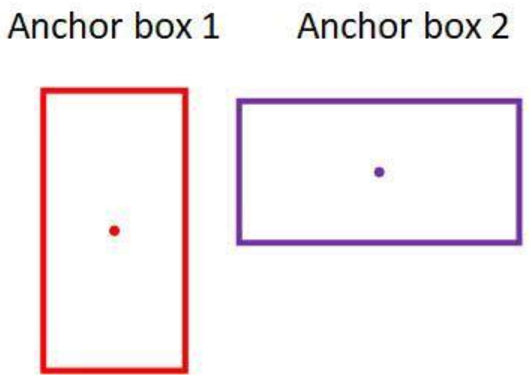
What if this happens?

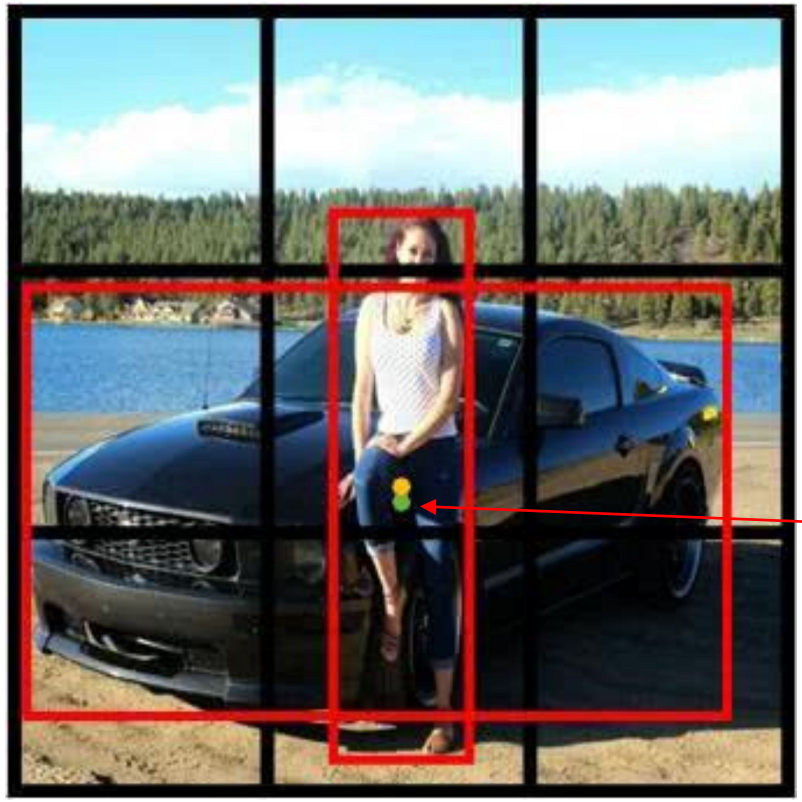


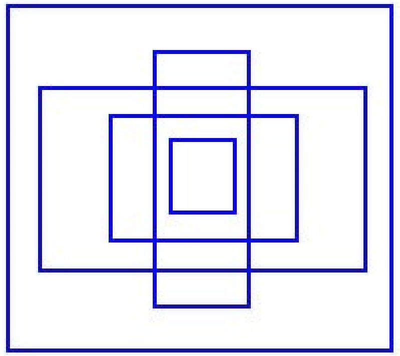
Anchor Boxes

- There is a limitation with only having grid cells.
- Say we have multiple objects in the same grid cell. For instance, there's a person standing in front of a car and their bounding box centers are so close. Shall we choose the person or the car?
- To solve the problem, we'll introduce the concept of **anchor box**. Anchor box makes it possible for the YOLO algorithm to detect multiple objects centered in one grid cell.







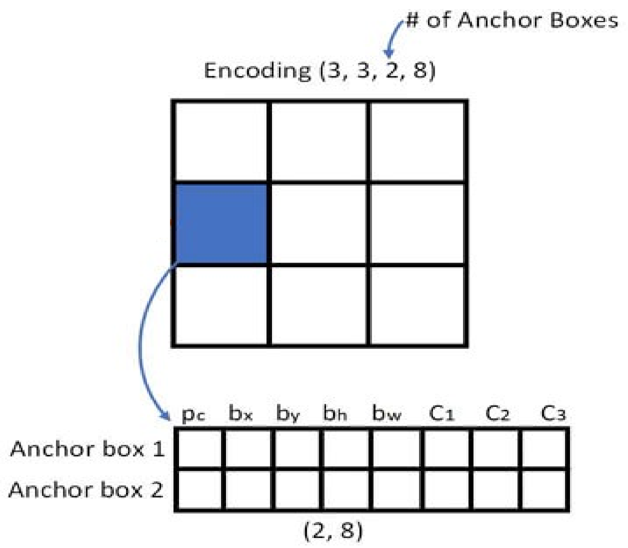


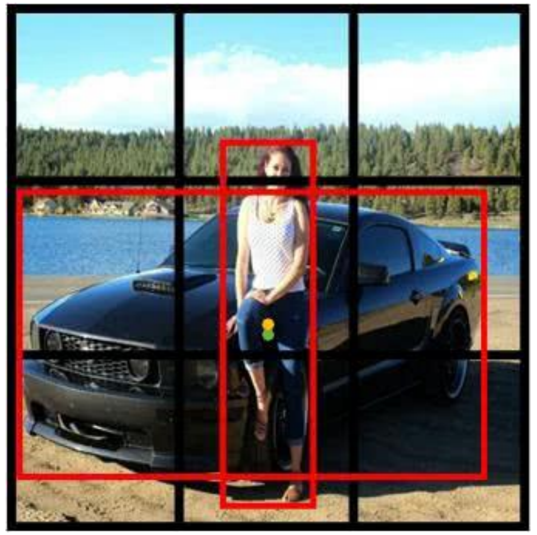
Notice that, in the image above, both the car and the pedestrian are centered in the middle grid cell.

- Each grid cell now has two anchor boxes, where each anchor box acts like a container. Meaning now each grid cell can predict up to 2 objects.
- Choosing anchor boxes with two different shapes is becausse when we make a decision as to which object is put in which anchor box, we look at their shapes, noting how similar one object's bounding box shape is to the shape of the anchor box. For the above example, the person will be associated with the tall anchor box since their shape is more similar.
- As a result, the output of one grid cell will be extended to contain information for two anchor boxes.
- For example, the center grid cell in the image now has 8 x 2 output labels in total, as shown below → 3x3x16

The General Formula:
(N x N) x [num_anchors x (5 + num_classes)]

N = 3, num_anchors = 2, num_classes = 3 →
3x3x[2 x (5 + 3)] = 3x3x16





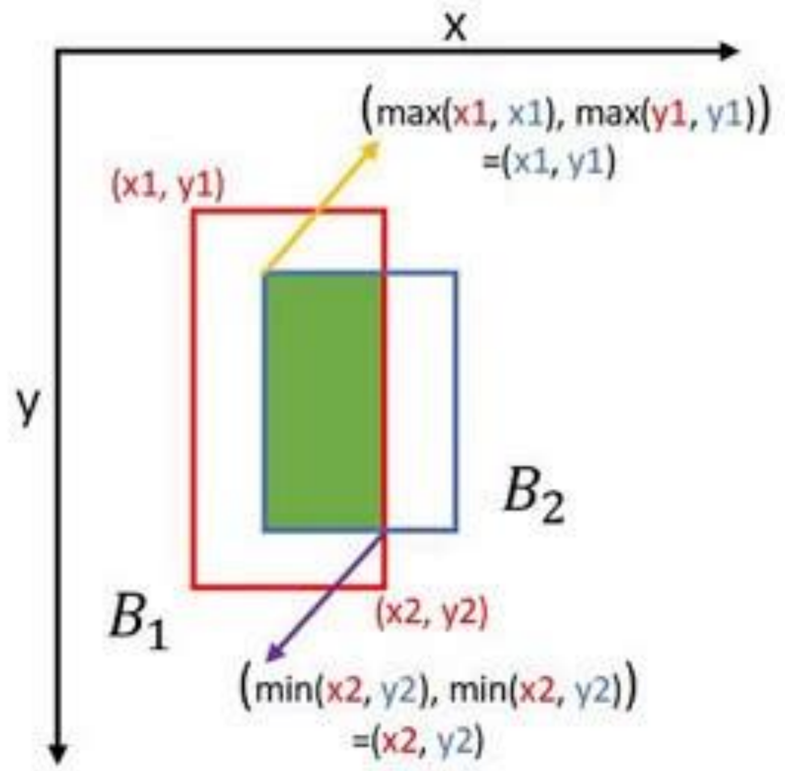
$$y = \begin{bmatrix} p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \\ p_c \\ b_x \\ b_y \\ b_h \\ b_w \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 1 \\ 0 \\ 0 \\ 1 \\ b_x \\ b_y \\ b_h \\ b_w \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

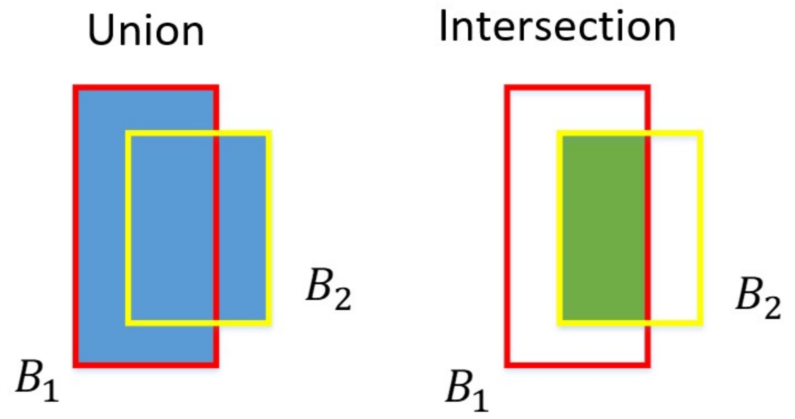
Anchor box 1
Pedestrian

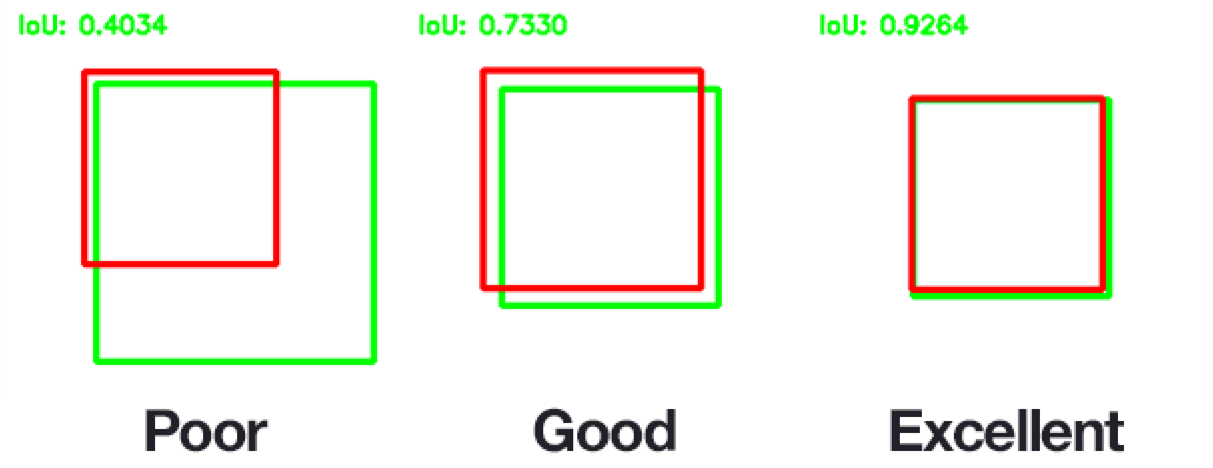
Anchor box 2
Car

Evaluation metric

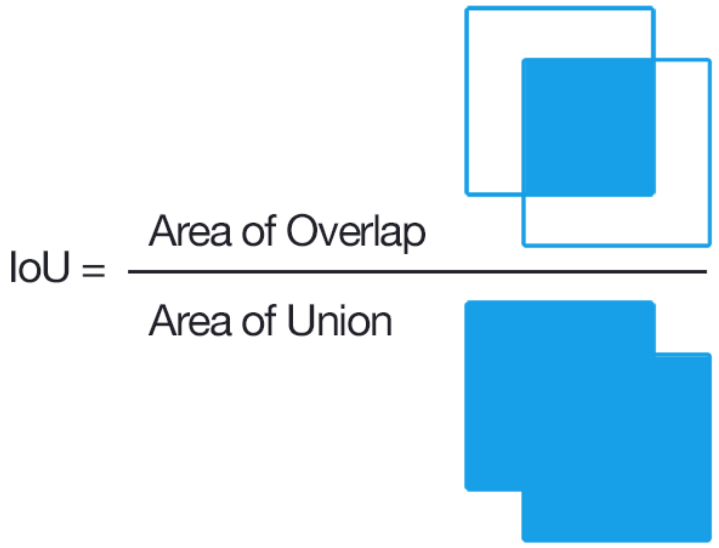
- Instead of defining a box by its center point, width and height, let’s define it using its two corners (upper left and lower right): (x1, y1, x2, y2)
- To compute the intersection of two boxes, we start off by finding the intersection area’s two corners.

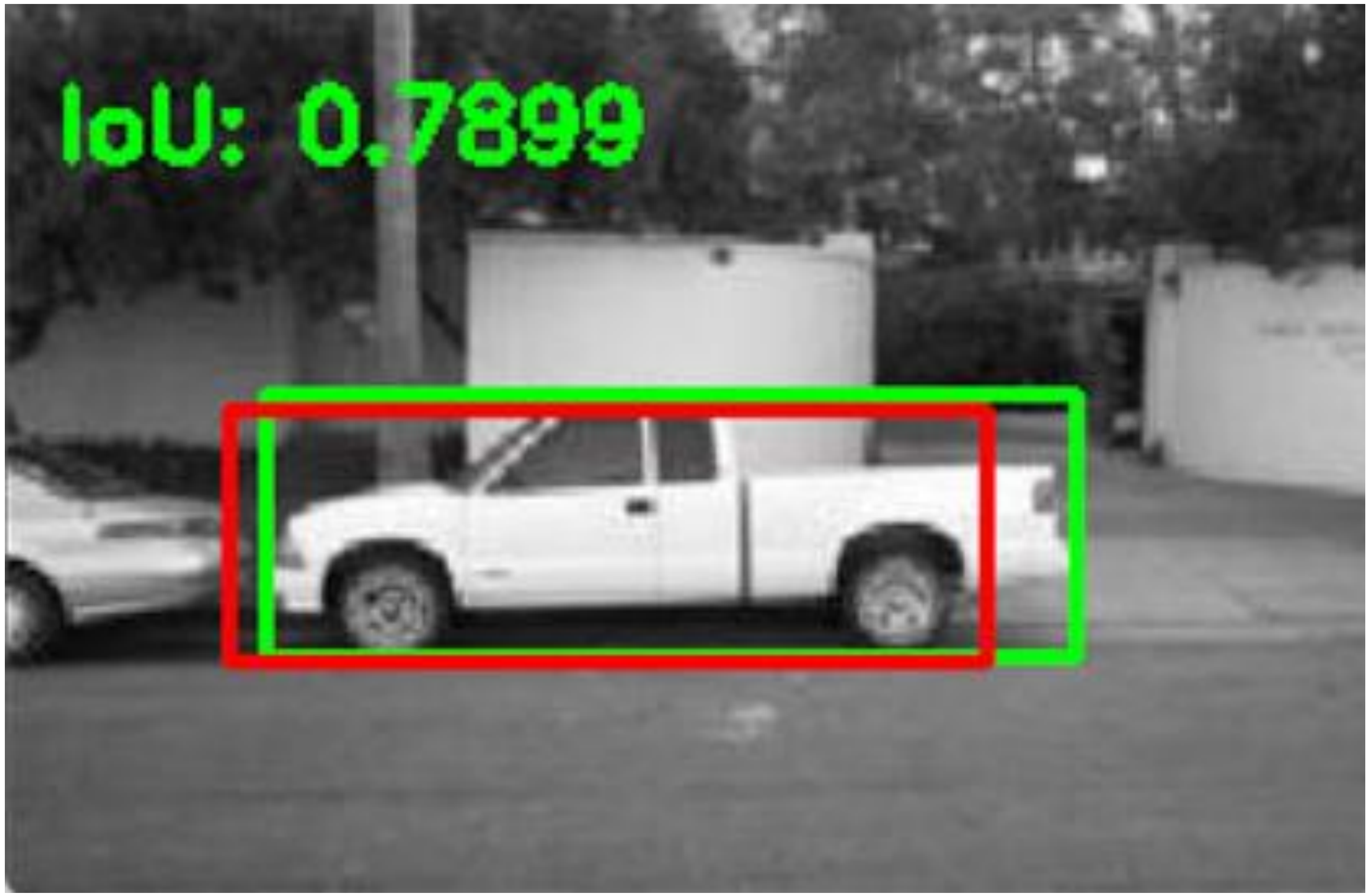


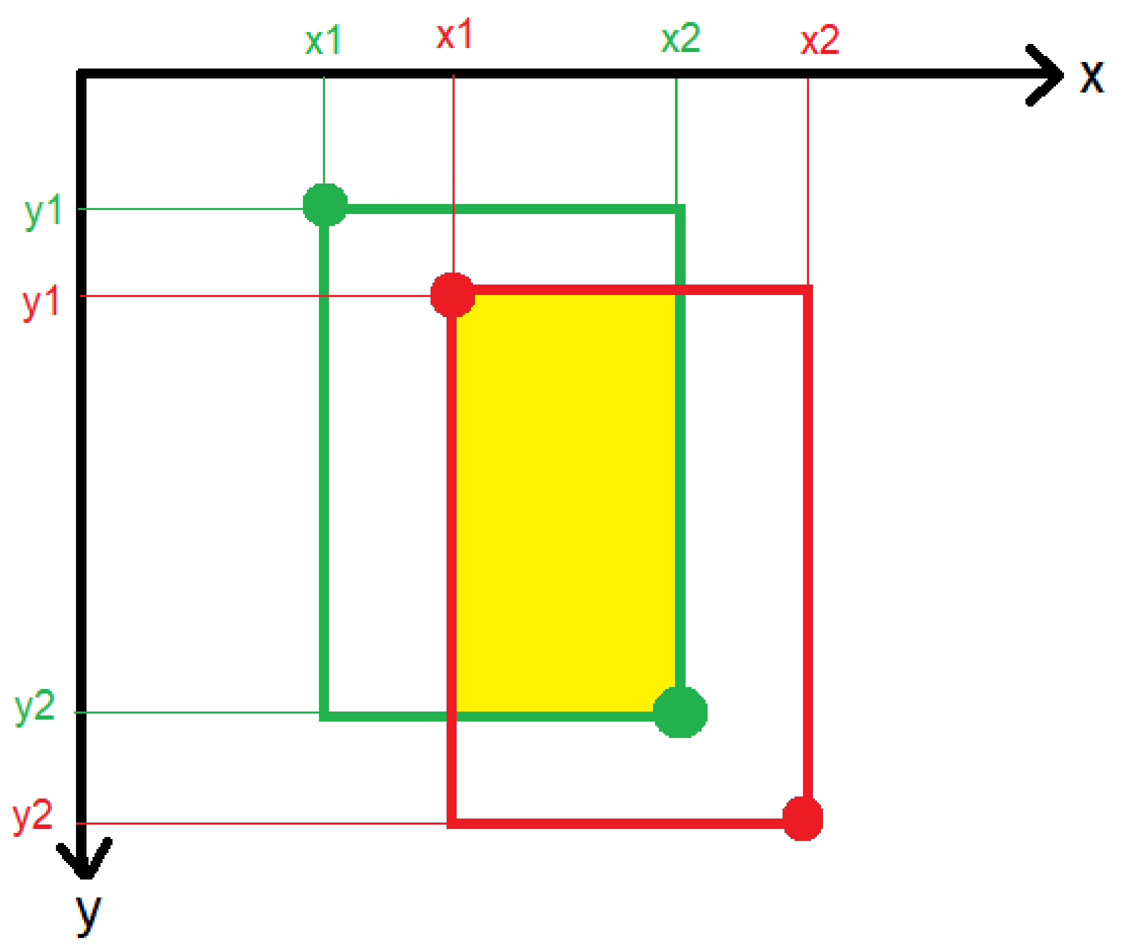
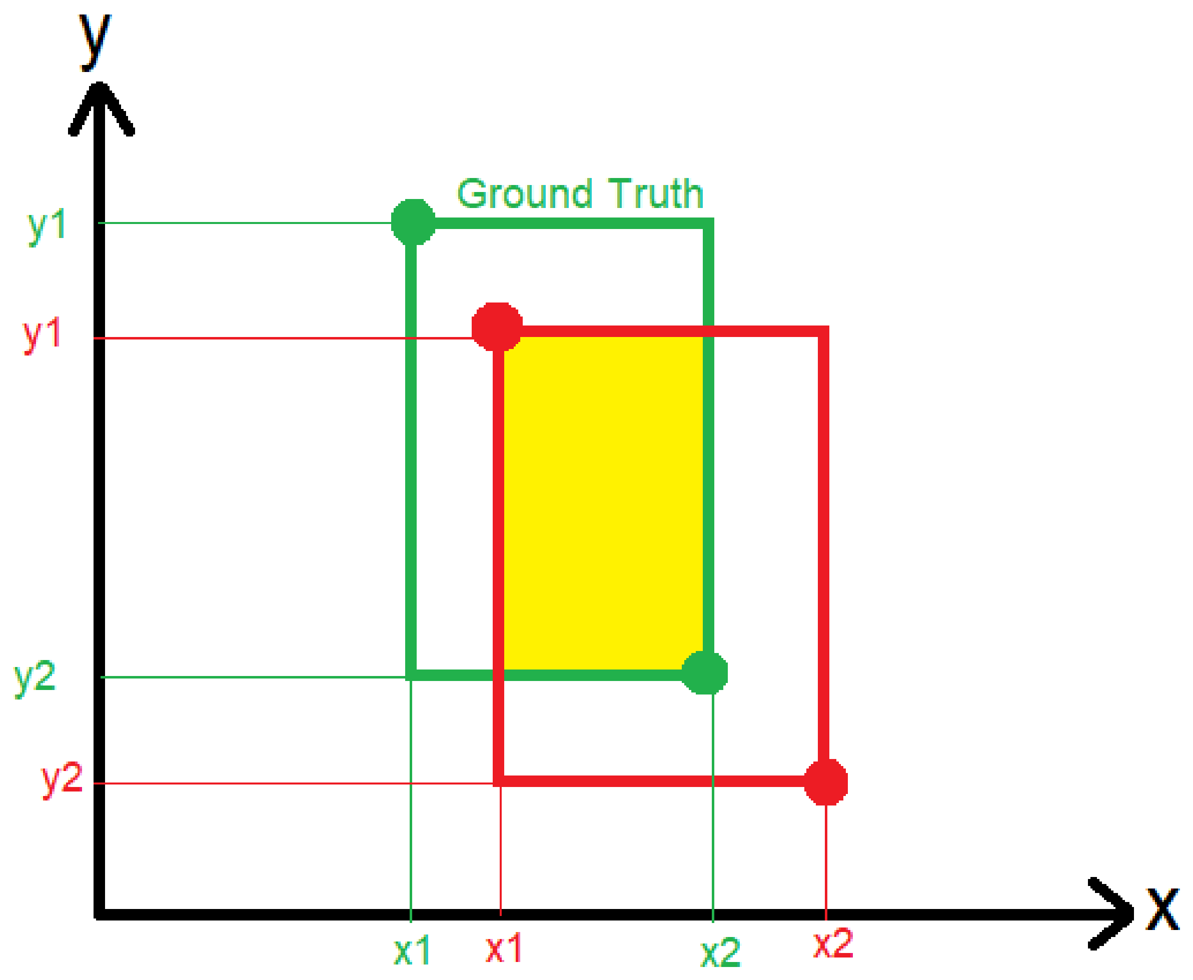




Example

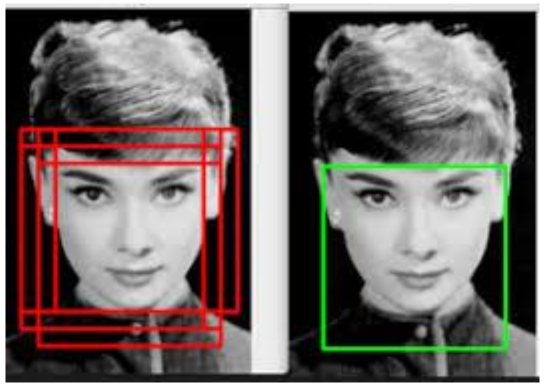


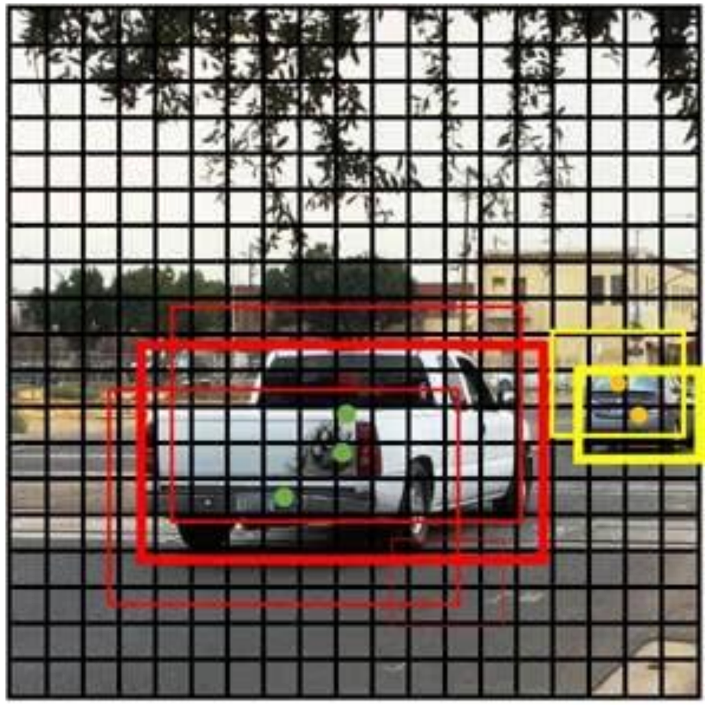



x1
x1
x2
x2
x
y1
y1
y2
y2
y

y
Ground Truth
y1
y1
y2
y2
x1
x1
x2
x2
x

Non-max Suppression

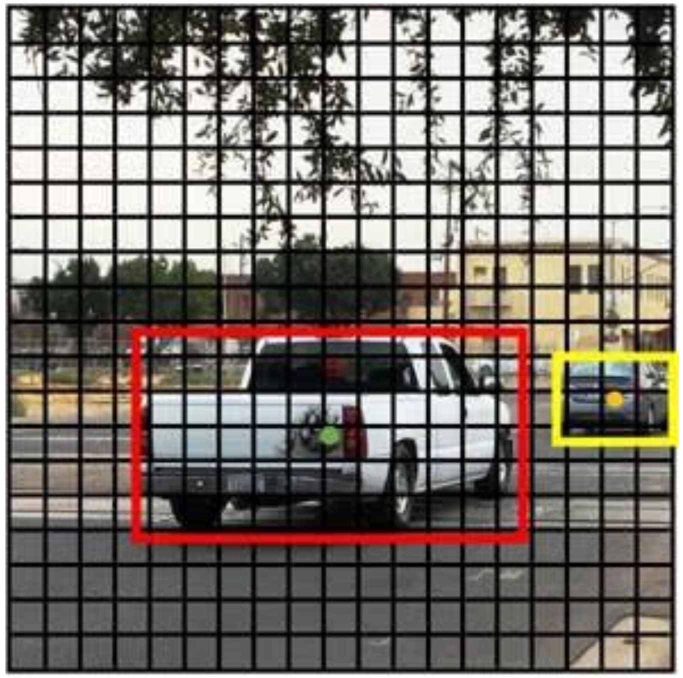
- Non-max suppression is a common algorithm used for cleaning up when multiple grid cells are predicted the same object.
- For the example below, the model outputs three predictions for the truck in the center. There are three bounding boxes, but we only need one. The thicker the predicted bounding box, the more confident the prediction is — that means a higher pc value.





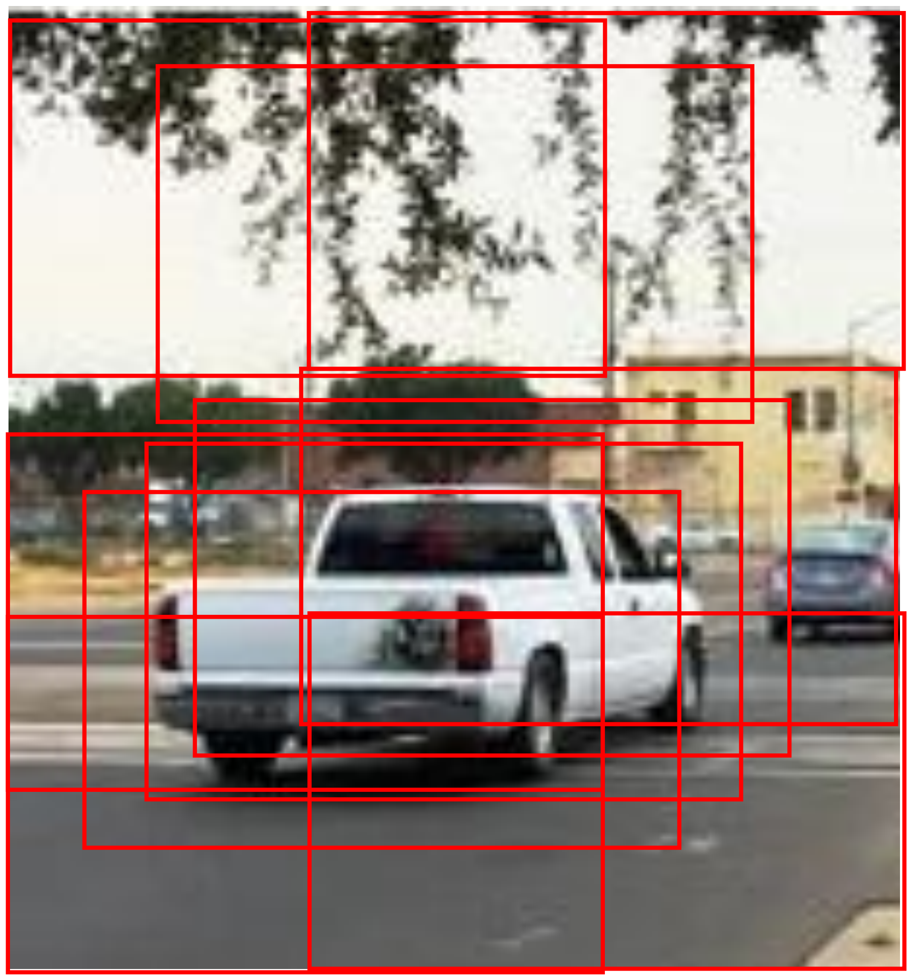
Non-max suppression in 3 steps:

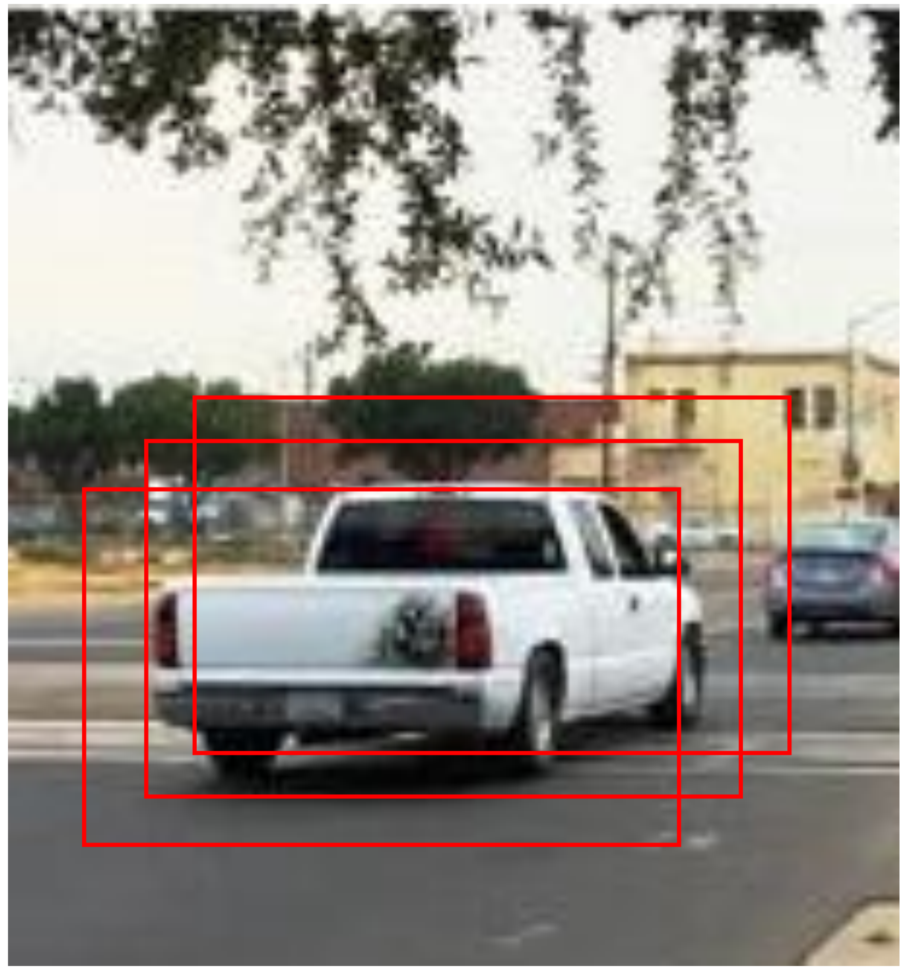
1. Discard all boxes with pc less or equal to 0.6.
2. Pick the box with the largest pc output as a prediction.
3. Discard any remaining box with IoU greater than or equal to 0.5.



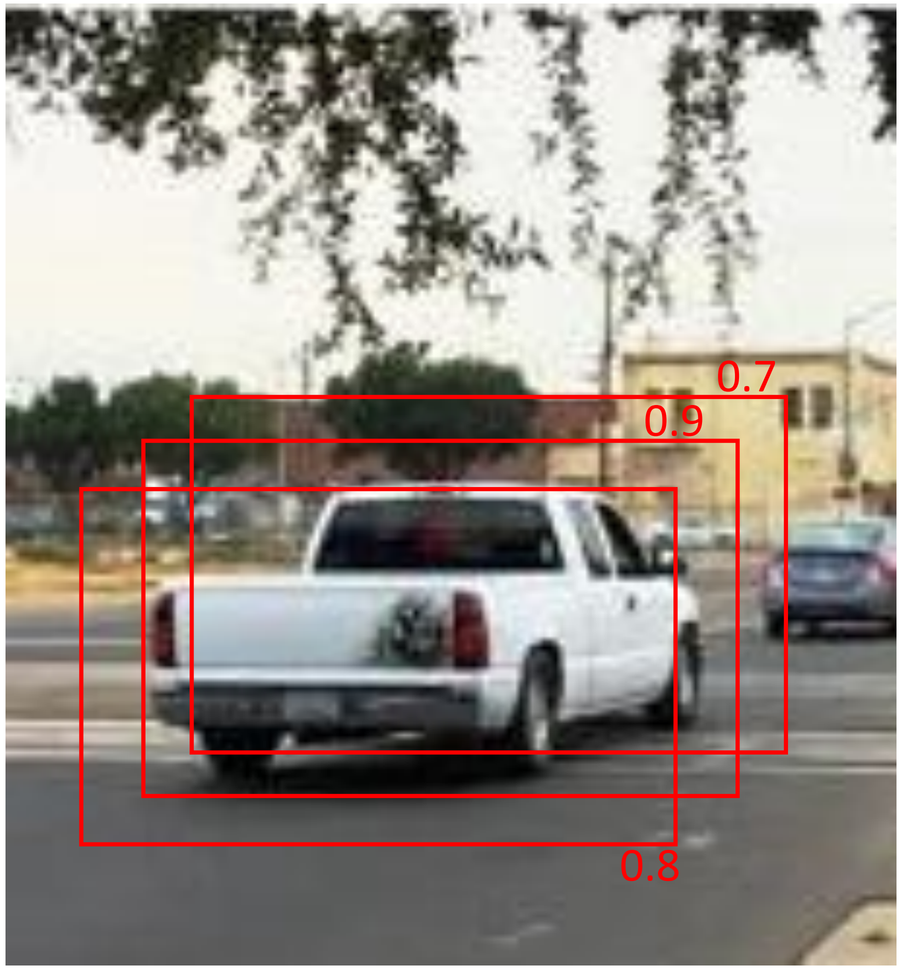
We'll see the step-by-step in the next slide!

1.Discard all boxes with pc less or equal to 0.6.

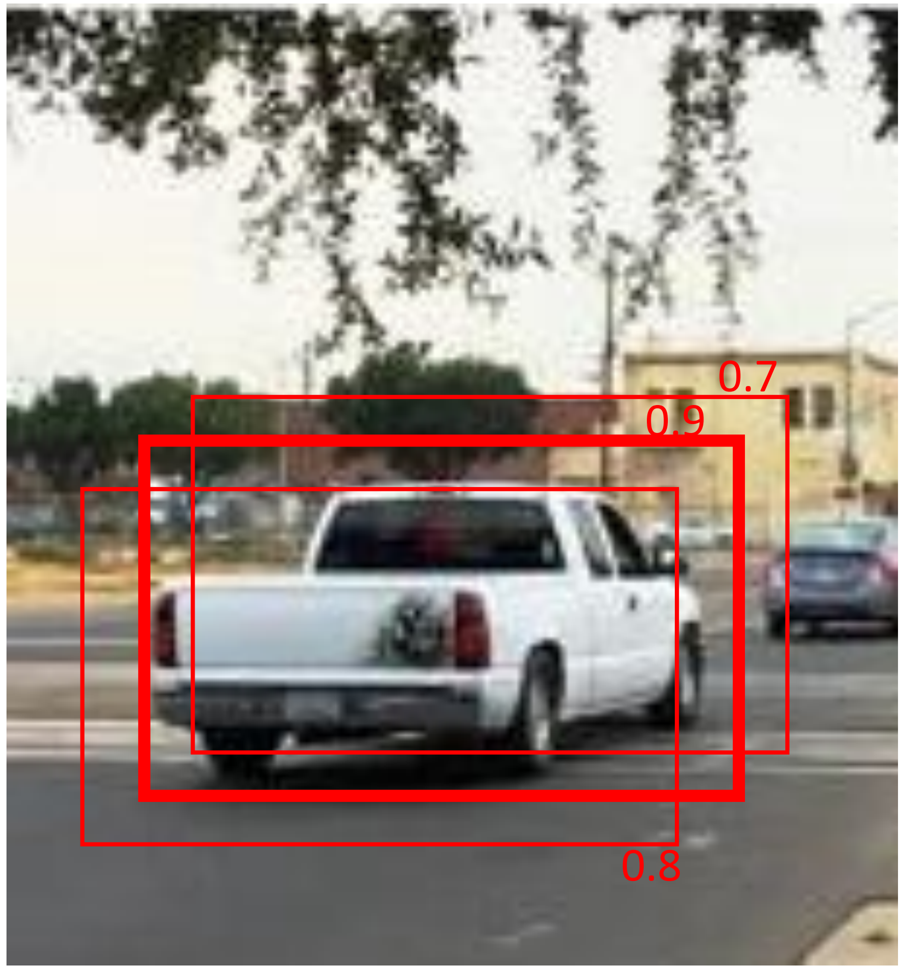




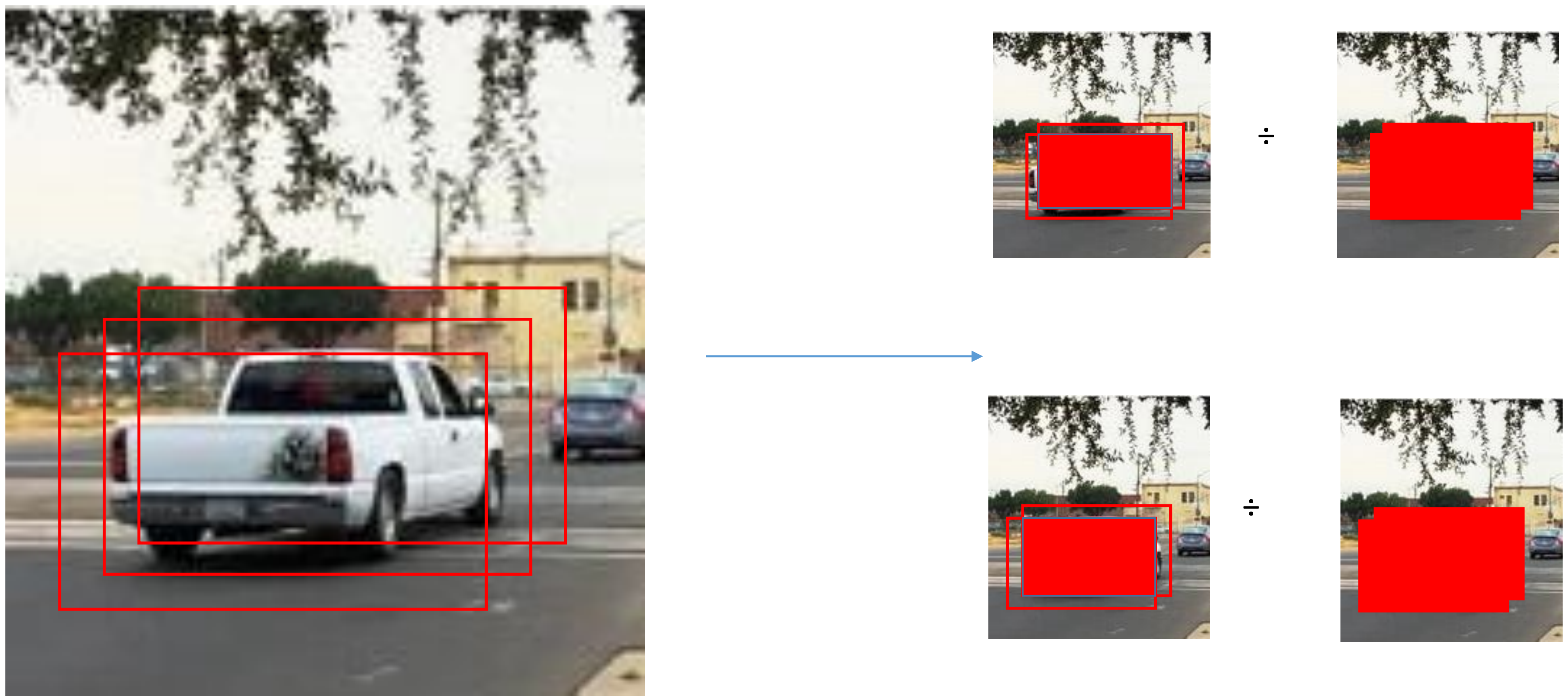
2. Pick the box with the largest pc output as a prediction.



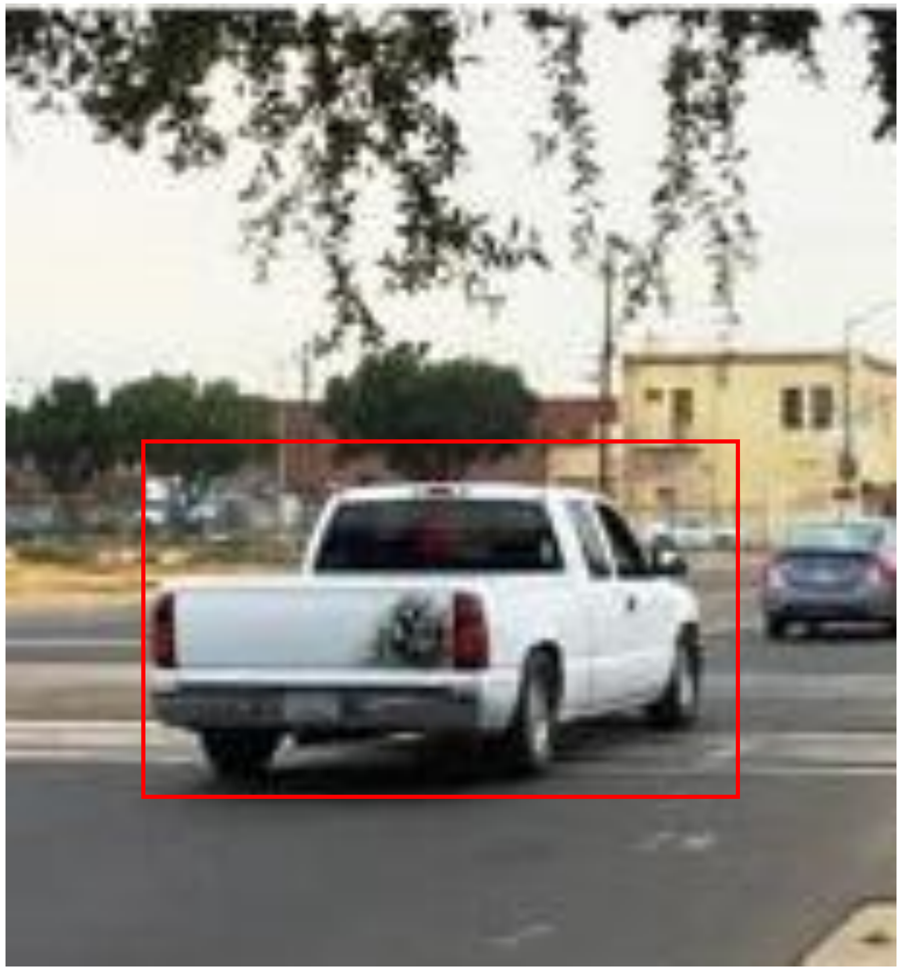
2. Pick the box with the largest pc output as a prediction.



2. Discard any remaining box with IoU greater than or equal to 0.5.

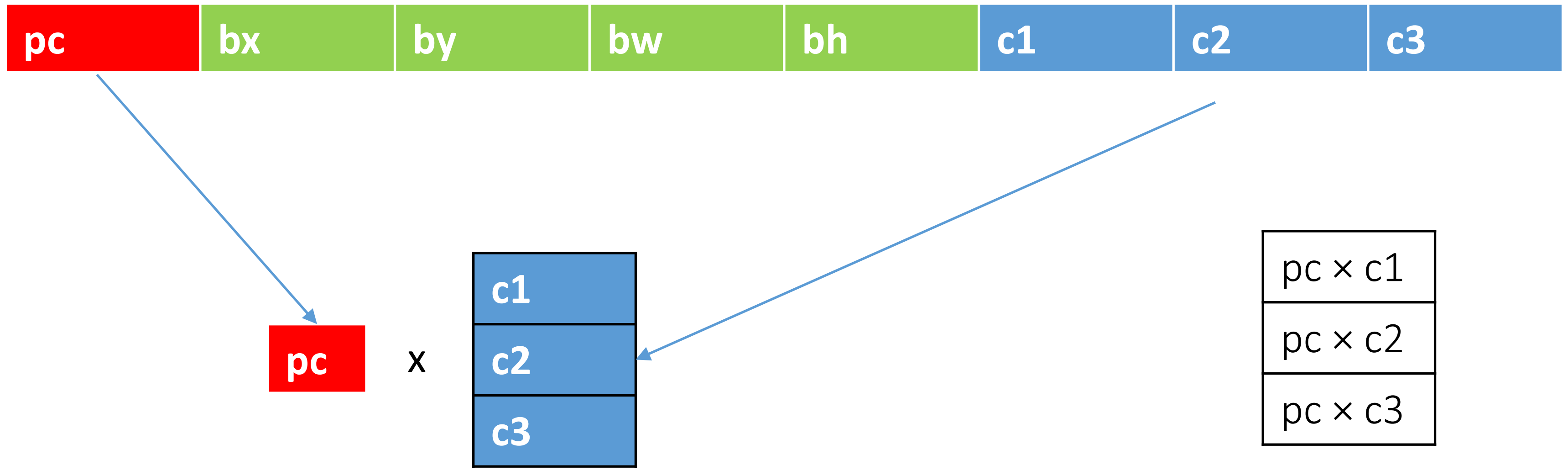


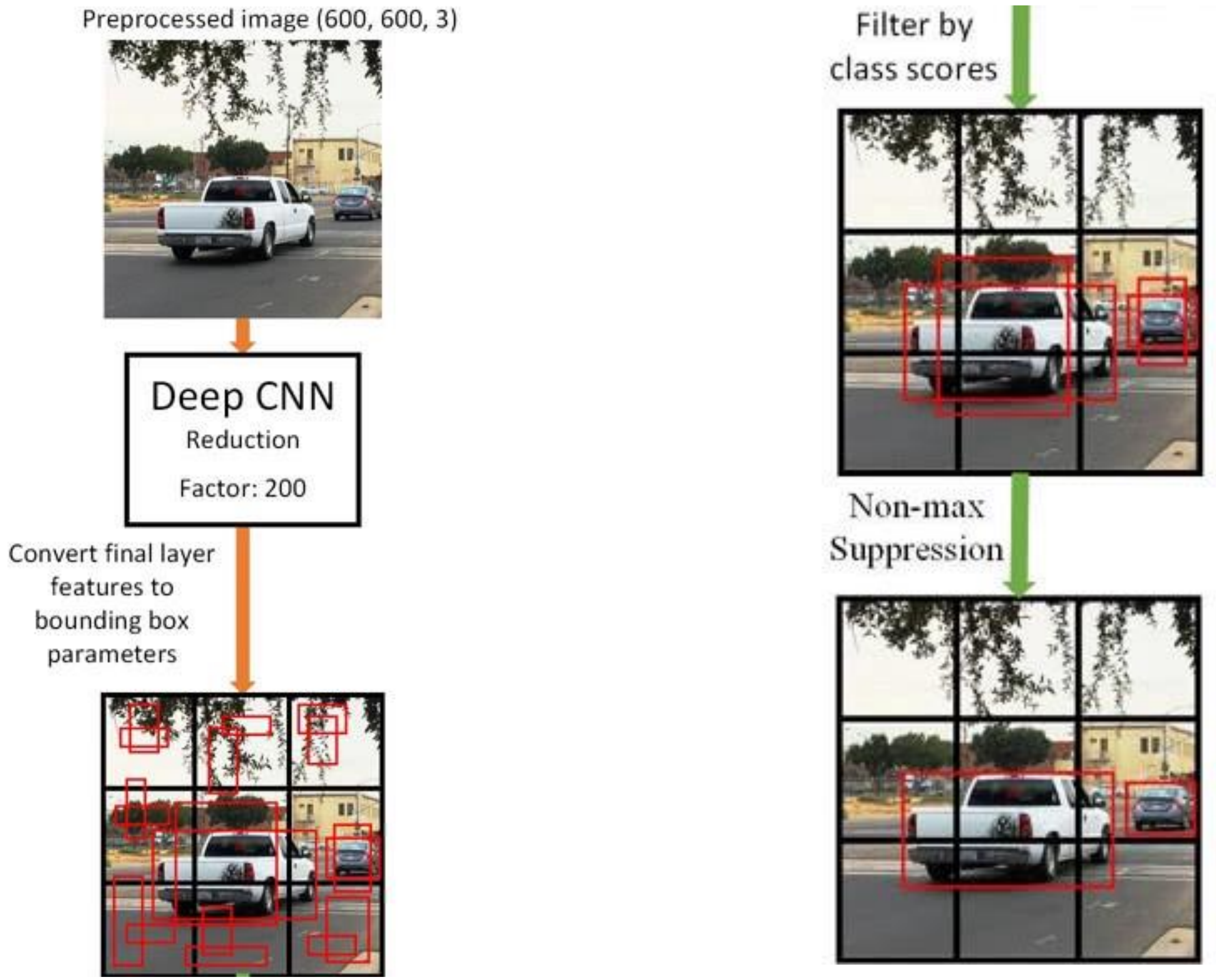
We’re Left with this



Discard all boxes with pc less or equal to 0.6… How?

- Given an image, the YOLO model will generate an output matrix of shape (3, 3, 2, 8). Which means each of the grid cells will have two predictions, even for those grid cells that don't have any object inside.
- Now we need to **filter with a threshold by "class scores"**.
- The class scores are computed by multiplying pc with the individual class output (C1, C2, C3).



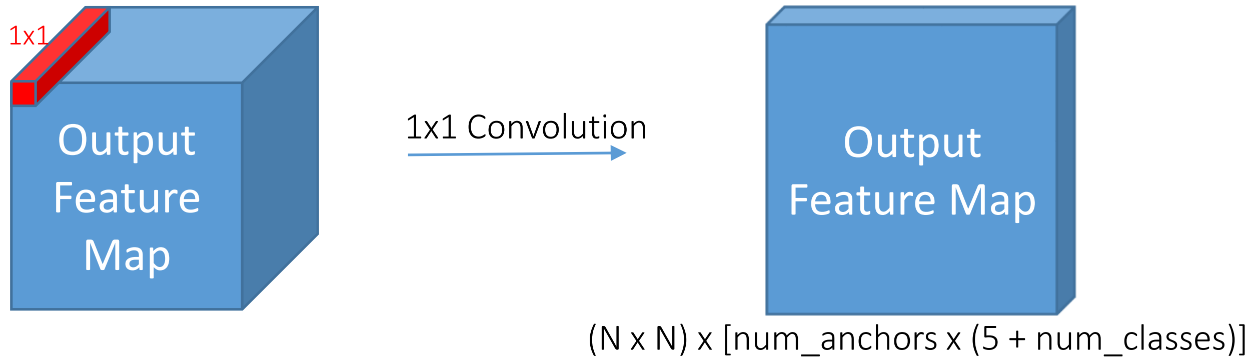

Preprocessed image (600, 600, 3)
Deep CNN
Reduction
Factor: 200
Convert final layer features to bounding box parameters
Filter by class scores
Non-max Suppression

YOLO in Depth

- YOLO makes use of only convolutional layers, making it a fully convolutional network (FCN). It has layers with skip connections and upsampling layers. No form of pooling is used, and a convolutional layer with stride 2 is used to downsample the feature maps. This helps in preventing loss of low-level features often attributed to pooling.
- The network downsamples the image by a factor called the **stride** of the network. For example, if the stride of the network is 32, then an input image of size 416 x 416 will yield an output of size 13 x 13. Generally, ***stride* of any layer in the network is equal to the factor by which the output of the layer is smaller than the input image to the network.**

In YOLO, the prediction is done by using a convolutional layer which uses 1 x 1 convolutions, and the output is a feature map. **Depth-wise, we have (B x (5 + C)) entries in the feature map. We expect each cell of the feature map to predict an object through one of it's bounding boxes if the center of the object falls in the receptive field of that cell.**

YOLO:
- num_anchors (bounding boxes) = 3 for each scale
- we have 3 scales
- we have 80 classes



Let us consider an example below, where the input image is 416 x 416, and stride of the network is 32. As pointed earlier, the dimensions of the feature map will be 13 x 13. We then divide the input image into 13 x 13 cells.

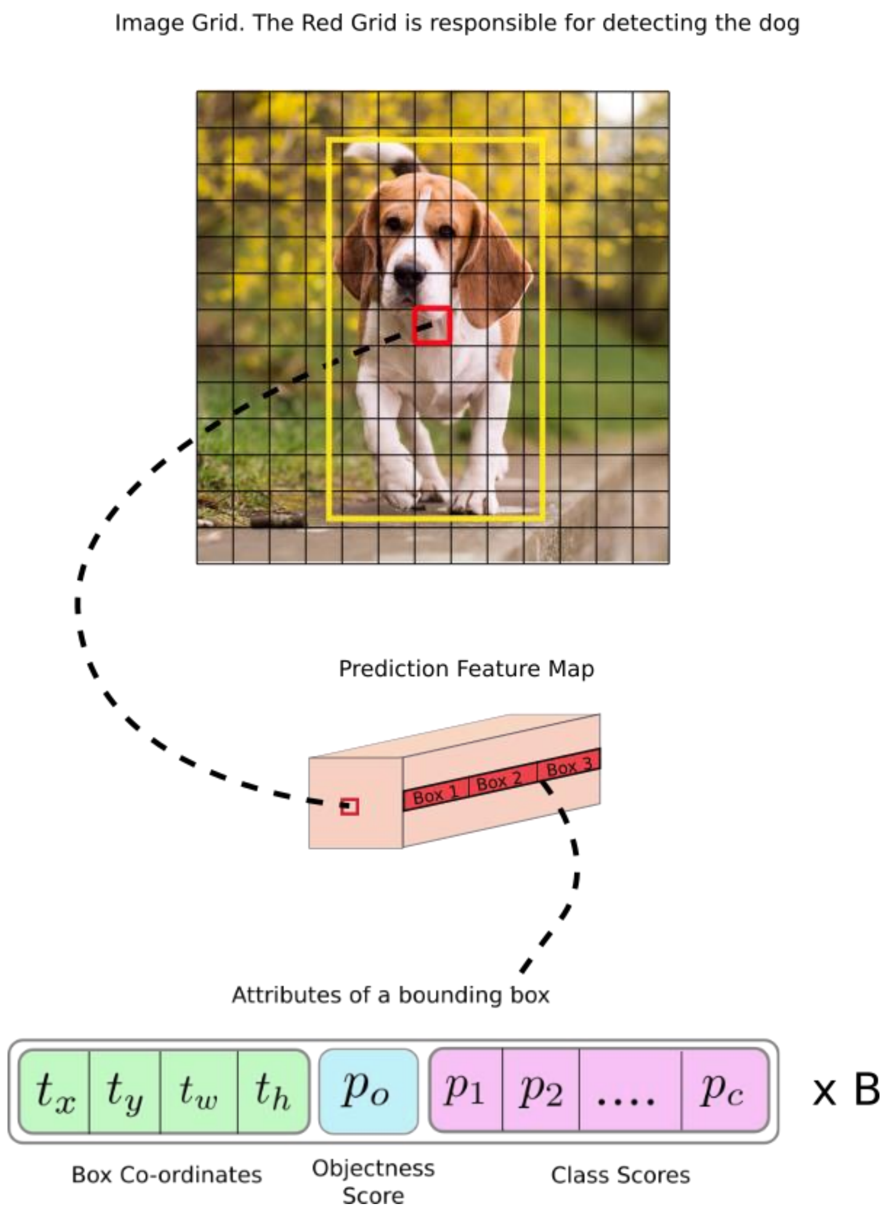


Image Grid. The Red Grid is responsible for detecting the dog
Prediction Feature Map
Box 1
Box 2
Box 3
Attributes of a bounding box
t_x t_y t_w t_h p_o p_1 p_2 .... p_c
x B
Box Co-ordinates
Objectness Score
Class Scores

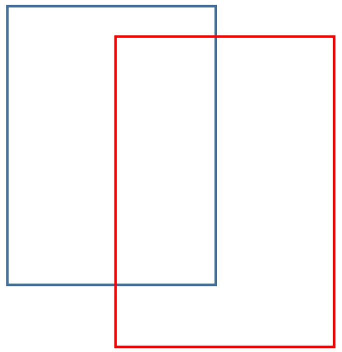
The cell (*on the input image*) containing the center of the ground truth box of an object is chosen to be the one responsible for predicting the object. In the image, it is the cell which marked red, which contains the center of the ground truth box (marked yellow).

$$\frac{input\ size}{stride}$$

Now, the red cell is the 7th cell in the 7th row on the grid. We now assign the 7th cell in the 7th row **on the feature map** (corresponding cell on the feature map) as the one responsible for detecting the dog. This cell predicts three bounding boxes (three anchors)

Prediction of the Anchor Boxes

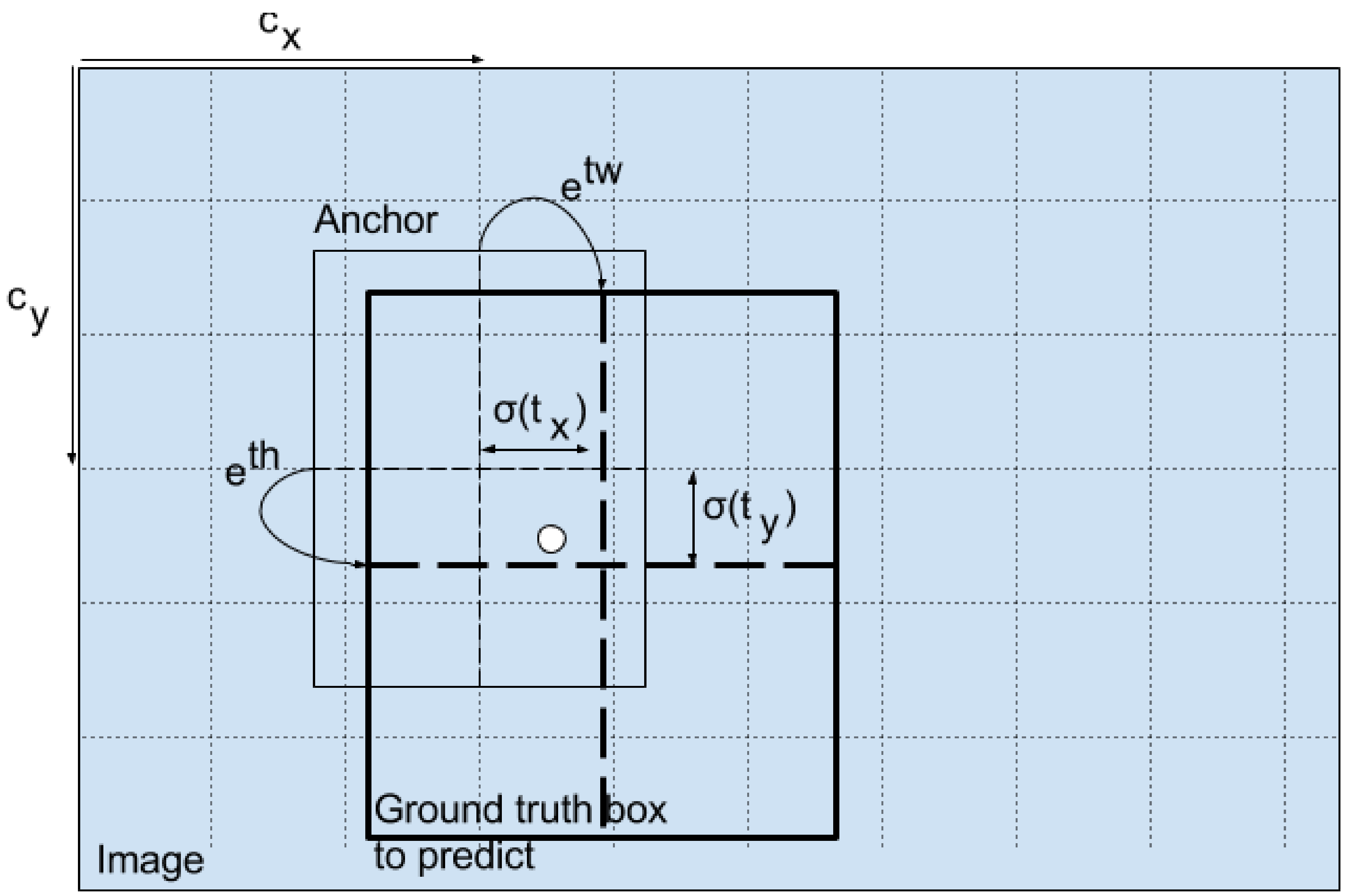
- It might make sense to predict the width and the height of the bounding box, but in practice, that leads to unstable gradients during training. Instead, most of the modern object detectors **offsets (which is how much should we move the predicted bounding box in order to get the desired bounding box)** to pre-defined default bounding boxes (anchors).
- Then, these transforms are applied to the anchor boxes to obtain the prediction. YOLO v3 has three anchors, which result in prediction of three bounding boxes per cell.



tx, ty, tw, th is what the network outputs. *cx* and *cy* are the top-left co-ordinates of the grid. *pw* and *ph* are anchors dimensions for the box.

Notice we are running our center coordinates prediction through a sigmoid function. This forces the value of the output to be between 0 and 1.

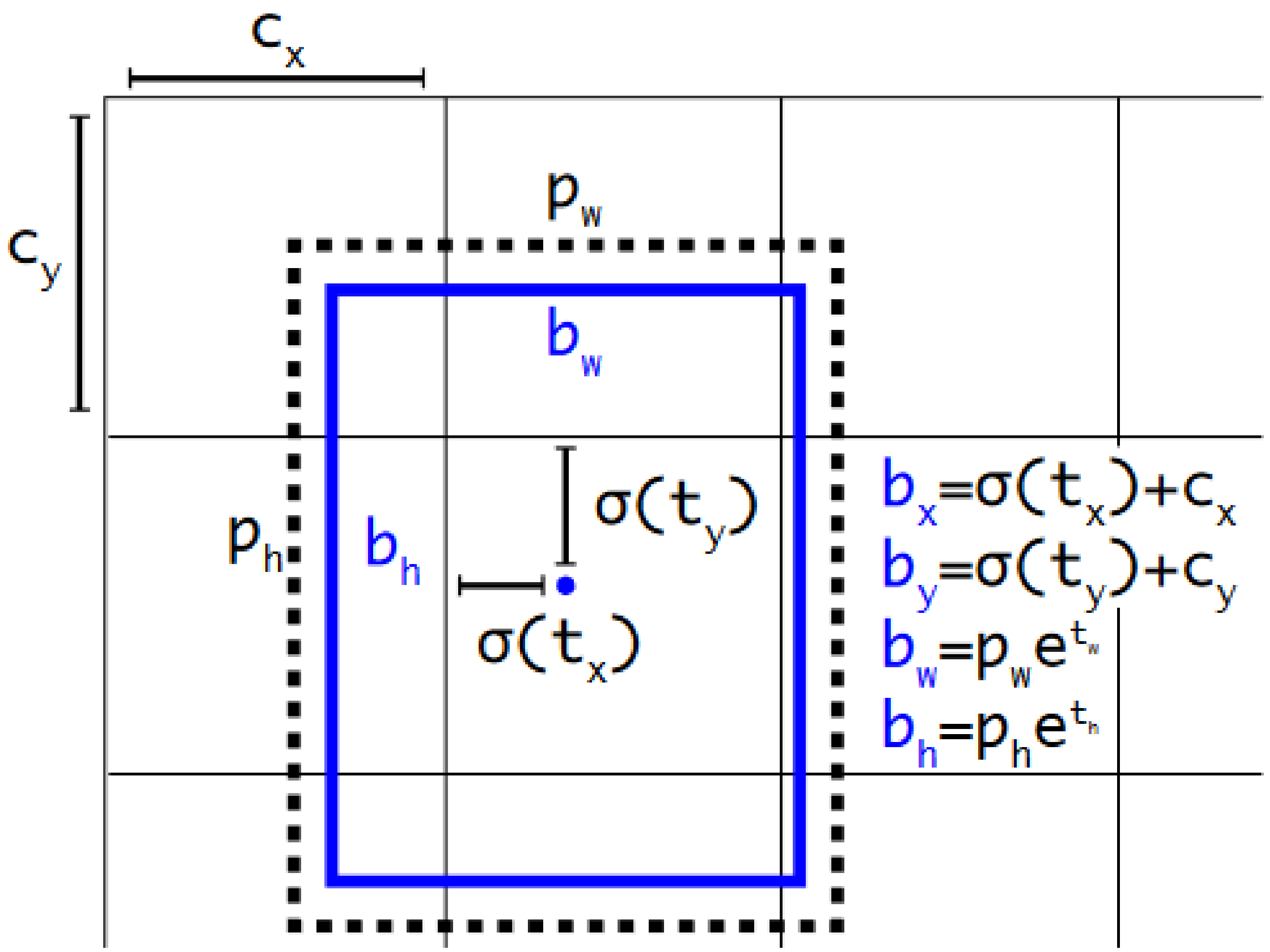
$$b_x = \sigma(t_x) + c_x$$
$$b_y = \sigma(t_y) + c_y$$
$$b_w = p_w e^{t_w}$$
$$b_h = p_h e^{t_h}$$



$$b_x = \sigma(t_x) + c_x$$
$$b_y = \sigma(t_y) + c_y$$
$$b_w = p_w e^{t_w}$$
$$b_h = p_h e^{t_h}$$

The resultant predictions, *bw* and *bh*, are normalised by the height and width of the image. (Training labels are chosen this way). So, if the predictions *bx* and *by* for the box containing the dog are (0.3, 0.8), then the actual width and height on 13 x 13 feature map is (13 x 0.3, 13 x 0.8).

Just another representation.....



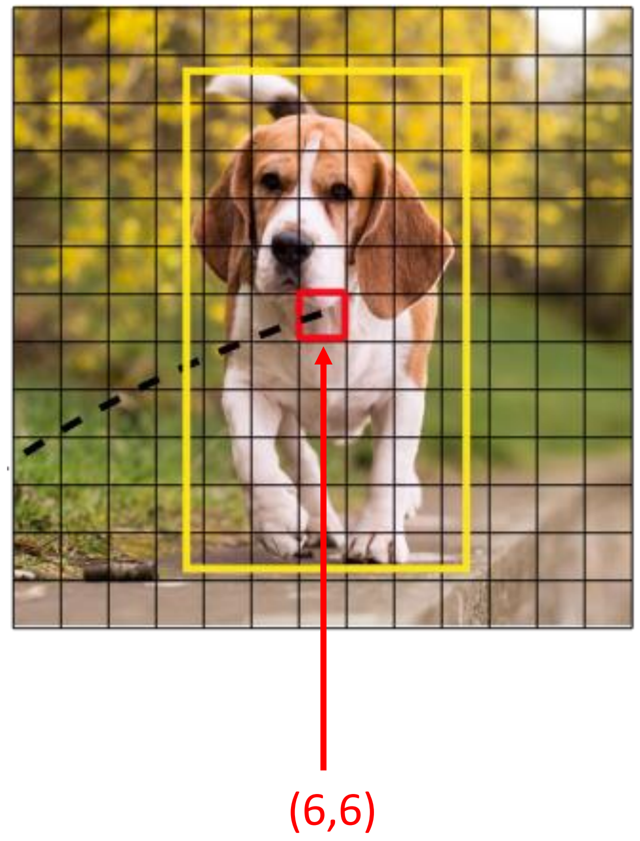
$$b_x = \sigma(t_x) + c_x$$
$$b_y = \sigma(t_y) + c_y$$
$$b_w = p_w e^{t_w}$$
$$b_h = p_h e^{t_h}$$

Why are we passing the output through a Sigmoid?

To get values between 0-1 ….. 😒 😒 😒 Seriously!!!!

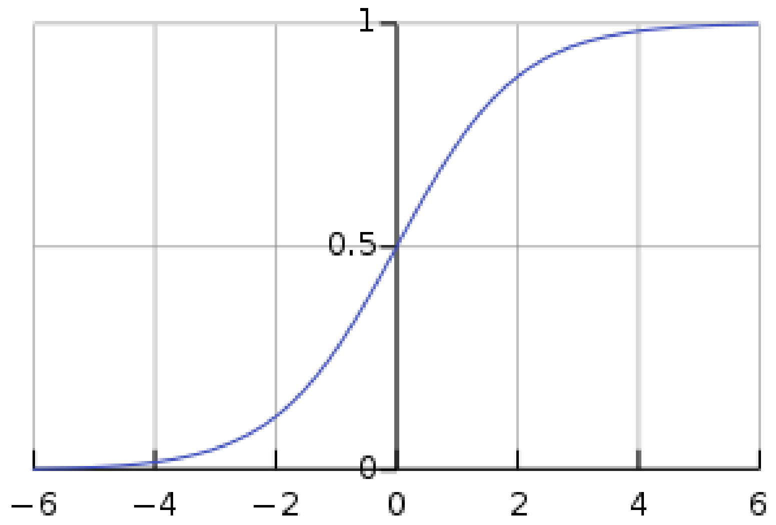
For example, consider the case of our dog image. If the prediction for center is (0.4, 0.7), then this means that the center lies at (6.4, 6.7) on the 13 x 13 feature map. (Since the top-left co-ordinates of the red cell are (6,6)).

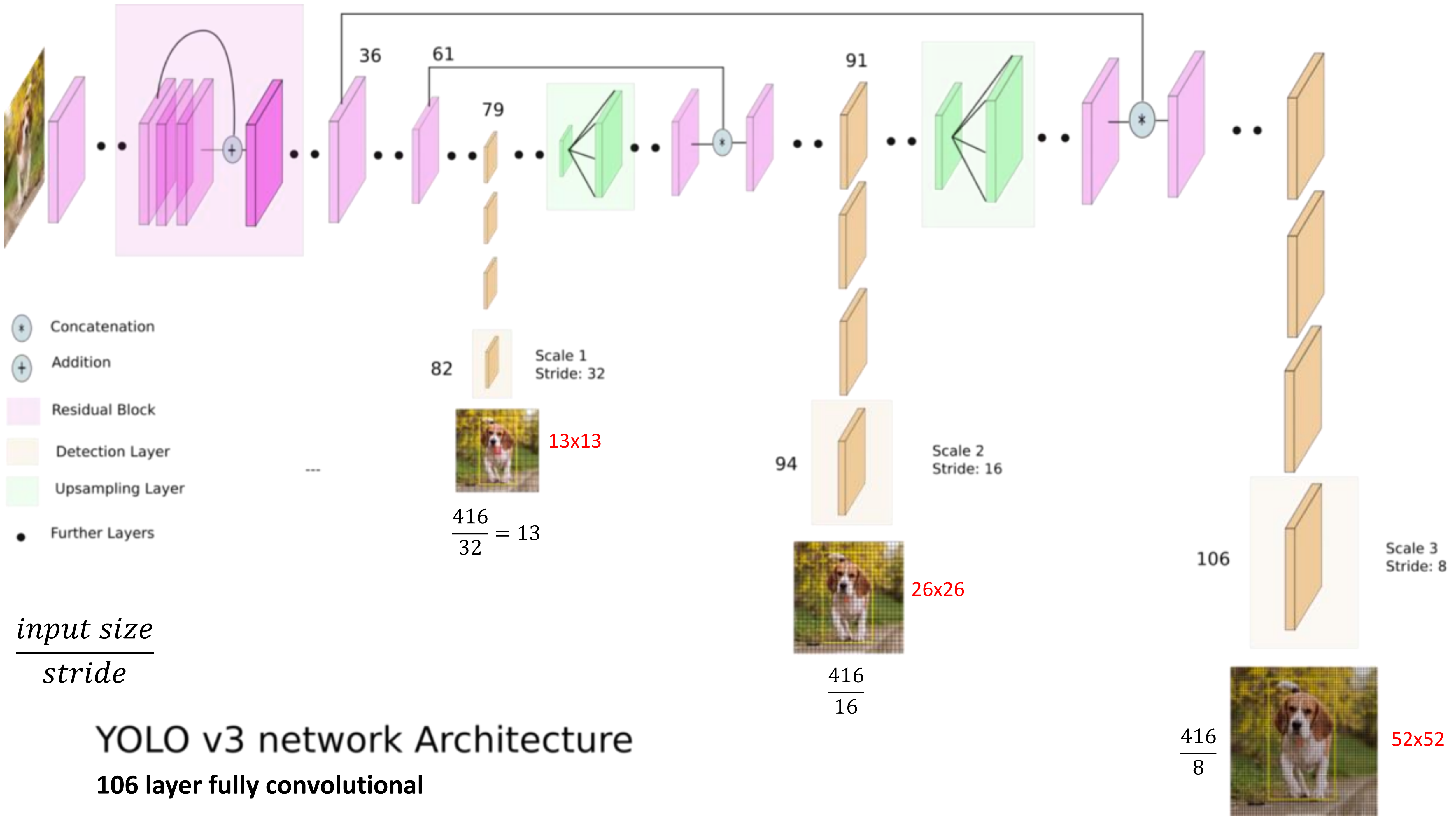
- But wait, what happens if the predicted x,y co-ordinates are greater than one, say (1.2, 0.7). This means center lies at (7.2, 6.7). Notice the center now lies in cell just right to our red cell, or the 8th cell in the 7th row. **This breaks theory behind YOLO** because if we postulate that the red box is responsible for predicting the dog, the center of the dog must lie in the red cell, and not in the one beside it.
- Therefore, to remedy this problem, the output is passed through a sigmoid function, which squashes the output in a range from 0 to 1, effectively keeping the center in the grid which is predicting.



Class Confidences

- Class confidences represent the probabilities of the detected object belonging to a particular class (Dog, cat, banana, car etc). Before v3, YOLO used to softmax the class scores.
- However, that design choice has been dropped in v3, and authors used sigmoid instead. The reason is that Softmaxing class scores assume that the classes are mutually exclusive. In simple words, if an object belongs to one class, then it's guaranteed it cannot belong to another class.
- Each class score is predicted using logistic regression and a threshold is used to predict multiple labels for an object. Classes with scores higher than this threshold are assigned to the box.





$$\frac{\textit{input size}}{\textit{stride}}$$

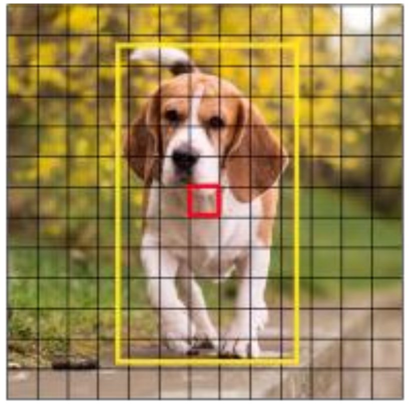
YOLO v3 network Architecture

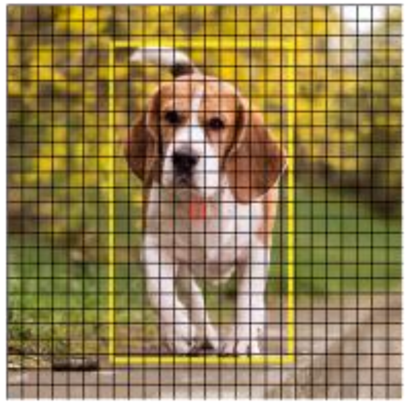
106 layer fully convolutional

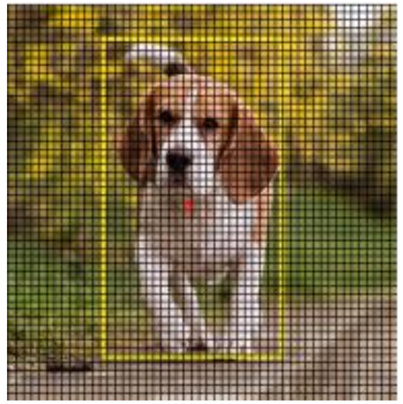
Prediction across different scales

- YOLO v3 makes prediction across 3 different scales. The detection layer is used make detection at feature maps of three different sizes, having **strides 32, 16, 8** respectively. This means, with an input of 416 x 416, we make detections on scales:

13 x 13, 26 x 26 and 52 x 52.







Test your understanding……

- What are the total numbers of bounding boxes that YOLO predicts?

$$(13 \times 13 \times 3) + (26 \times 26 \times 3) + (52 \times 52 \times 3)$$

$$=$$

$$\boxed{10{,}647}$$

Why are we predicting for 3 different scales?

The authors report that this helps YOLO v3 get better at detecting small objects, a frequent complaint with the earlier versions of YOLO.

Detections

- The first detection is made by the 82nd layer. If we have an image of 416 x 416, the resultant feature map would be of size 13 x 13. One detection is made here using the 1 x 1 detection kernel, giving us a detection feature map of 13 x 13 x 255.
- Then, the feature map from layer 79 is subjected to a few convolutional layers before being up sampled by 2x to dimensions of 26 x 26. This feature map is then depth concatenated with the feature map from layer 61. Then the combined feature maps is again subjected a few 1 x 1 convolutional layers to fuse the features from the earlier layer (61). Then, the second detection is made by the 94th layer, yielding a detection feature map of 26 x 26 x 255.
- A similar procedure is followed again, where the feature map from layer 91 is subjected to few convolutional layers before being depth concatenated with a feature map from layer 36. Like before, a few 1 x 1 convolutional layers follow to fuse the information from the previous layer (36). The third and final detection is made at the 106th layer, yielding feature map of size 52 x 52 x 255.

13 x 13 x 255 + 26 x 26 x 255 + 52 x 52 x 255

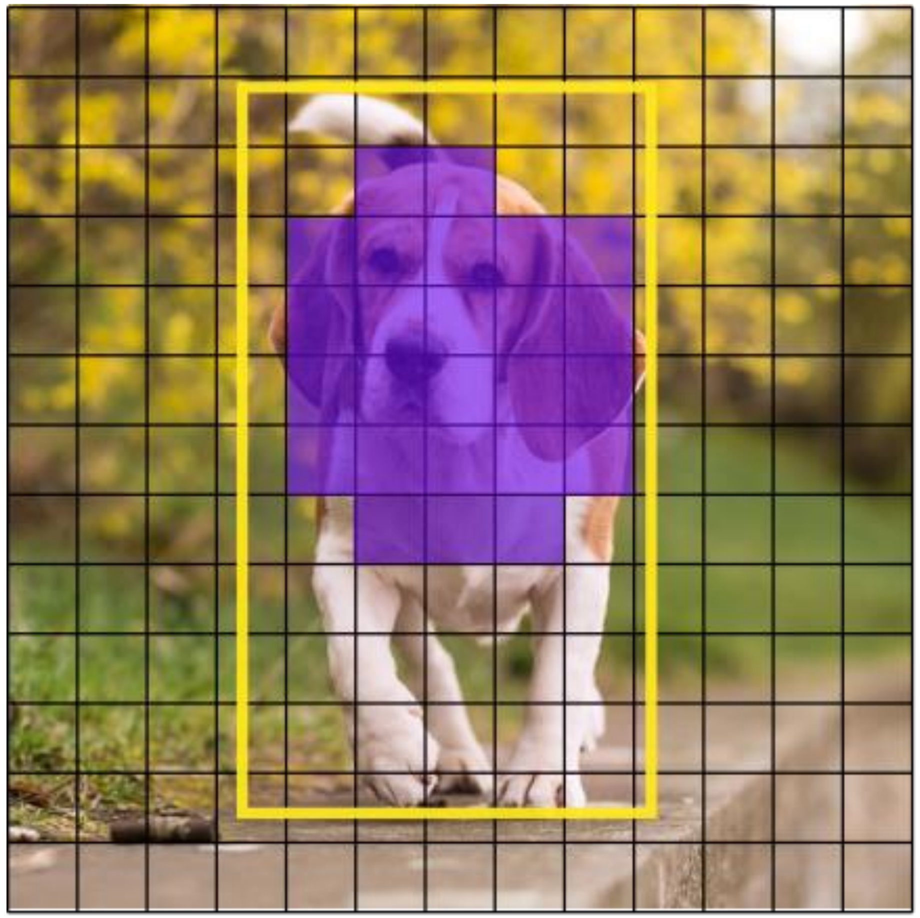
Again....We wont keep all bounding boxes

- **Thresholding by Object Confidence**

First, we filter boxes based on their objectness score. Generally, boxes having scores below a threshold are ignored.

- **Non-maximum Suppression**

NMS intends to cure the problem of multiple detections of the same image. For example, all the 3 bounding boxes of the red grid cell may detect a box or the adjacent cells may detect the same object.



Multiple Grids may detect the same object
NMS is used to remove multiple detections

Choice of anchor boxes

- YOLO v3, in total uses 9 anchor boxes. Three for each scale. It also uses K-Means clustering to generate 9 anchors.
- The anchors are then arranged in descending order of a dimension. The three biggest anchors are assigned for the first scale , the next three for the second scale, and the last three for the third.

At each scale, every grid can predict 3 anchors.
There are 3 scales.

YOLO Loss Function

- First, let’s have a look at the loss function YOLO v2.

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

Center Coordinates

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

Width/Height of Bounding Box

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

Object (ideally 1)

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

No Object (ideally 0)

j bounding box in cell i responsible for that prediction

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2 \qquad (3)$$

Classes

Only if object occurs in cell i →penalze

Squared Error

$$
\begin{aligned}
&\lambda_{\textbf{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} (x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \\
&+ \lambda_{\textbf{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(\sqrt{w_i} - \sqrt{\hat{w}_i}\right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i}\right)^2 \\
&\boxed{+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i\right)^2} \\
&+ \boxed{\lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i\right)^2} \\
&+ \boxed{\sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2} \qquad (3)
\end{aligned}
$$

Penalizes the objectness score prediction for bounding boxes responsible for predicting objects (the scores for these should ideally be 1)

Penalizes for bounding boxes having no objects, (the scores should ideally be zero)

Penalises the class prediction for the bounding box which predicts the objects.

- To compute the loss for the true positive, we only want one of them to be **responsible** for the object. For this purpose, we select the one with the highest IoU (intersection over union) with the ground truth.

To summarize, YOLO loss function is composed of:

- the **classification loss**.
- the **localization loss** (errors between the predicted boundary box and the ground truth).
- the **confidence loss** (the objectness of the box).

Still didn't get it?
No problem! We'll go
through every term again

Classification loss

- If *an object is detected*, the classification loss at each cell is the squared error of the class conditional probabilities for each class:

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

where

$\mathbb{1}_i^{obj} = 1$ if an object appears in cell i, otherwise 0.

We don't penalize classification error when no object is present on the cell

$\hat{p}_i(c)$ denotes the conditional class probability for class c in cell i.

Localization loss

- The localization loss measures the errors in the predicted boundary box locations and sizes. We only count the box responsible for detecting the object.

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

where

$\mathbb{1}_{ij}^{obj} = 1$ if the j th boundary box in cell i is responsible for detecting the object, otherwise 0.

λ_{coord} increase the weight for the loss in the boundary box coordinates.

To put more emphasis on the boundary box accuracy, we multiply the loss by λ*coord* (default: 5).

We do not want to weight absolute errors in large boxes and small boxes equally. i.e. a 2-pixel error in a large box is the same for a small box. To partially address this, YOLO predicts the square root of the bounding box width and height instead of the width and height.

“”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.””

Confidence loss

- If *an object is detected in the box*, the confidence loss (measuring the objectness of the box) is:

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

where

$\hat{C}_i$ is the box confidence score of the box j in cell i.

$\mathbb{1}_{ij}^{obj} = 1$ if the j th boundary box in cell i is responsible for detecting the object, otherwise 0.

If *an object is not detected in the box*, the confidence loss is:

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

where

$\mathbb{1}_{ij}^{noobj}$ is the complement of $\mathbb{1}_{ij}^{obj}$.

$\hat{C}_i$ is the box confidence score of the box j in cell i.

λ_{noobj} weights down the loss when detecting background.

Most boxes do not contain any objects. This causes a class imbalance problem, i.e. we train the model to detect background more frequently than detecting objects. To remedy this, we weight this loss down by a factor λ*noobj* (default: 0.5).

Together, we have……

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

Confidence score loss for each bounding box predictor.
C is the confidence score
𝟙obj = 1 when there is an object in the cell, and 0 otherwise.
𝟙noobj is the opposite of ***𝟙obj***

The difference in YOLOv2 and YOLOv3 loss

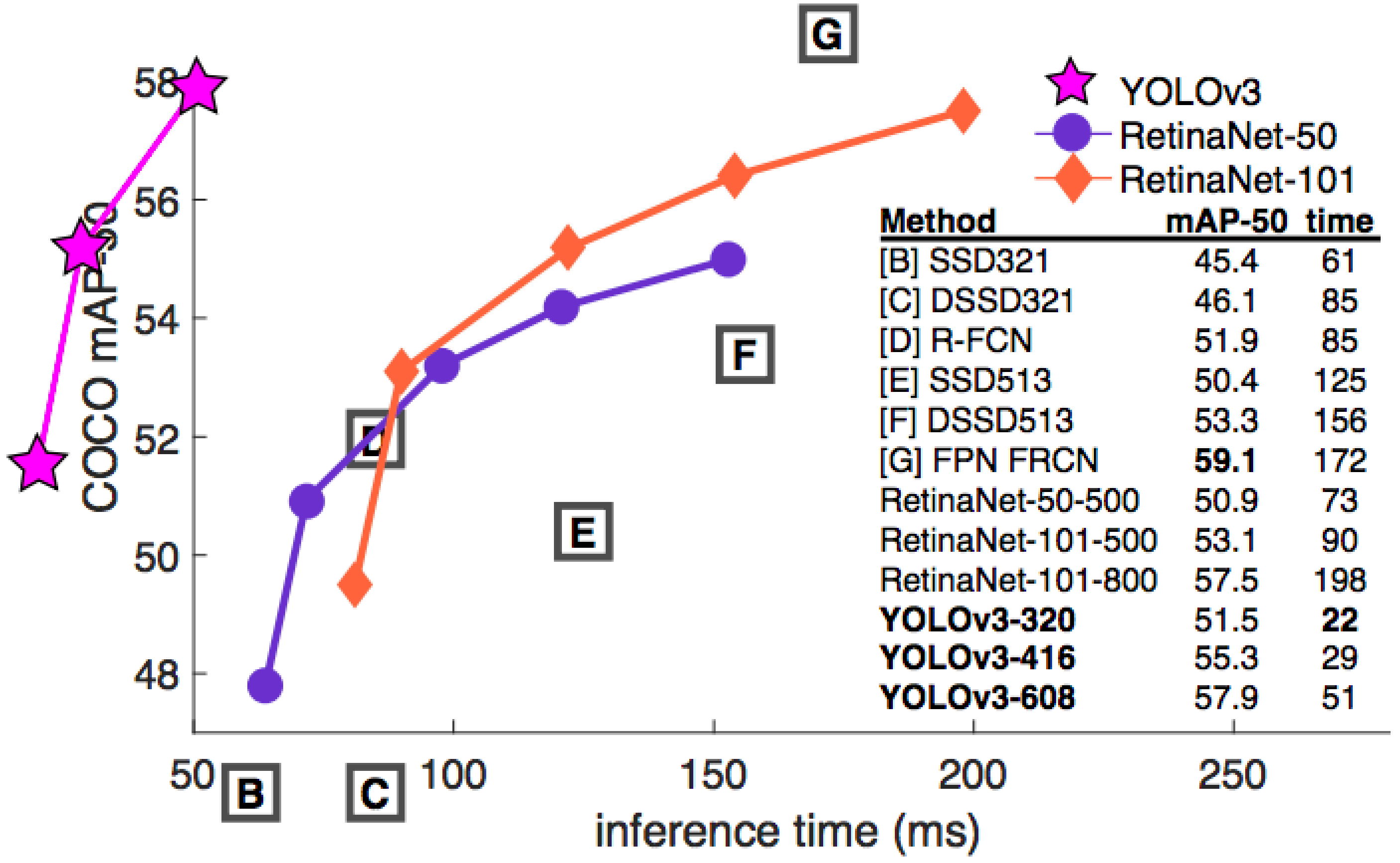
- The last three terms in YOLO v2 are the squared errors, whereas in YOLO v3, they've been replaced by cross-entropy error terms. In other words, **object confidence and class predictions in YOLO v3 are now predicted through logistic regression.**
- While we are training the detector, for each ground truth box, we assign a bounding box, whose anchor has the maximum overlap with the ground truth box.

We assign 1 bounding box for each ground truth object. If a bounding box is not assigned to a Ground Truth object, it occurs no loss for coordinate or class predictions (classification and localization loss), only objectnes/confidence (if an object is present or not).

Classification Loss: Instead of using mean square error in calculating the classification loss, YOLOv3 uses binary cross-entropy loss for each label. This also reduces the computation complexity by avoiding the softmax function.

Results of YOLOv3

YOLO v3 performs at par with other state of art detectors like RetinaNet, while being considerably faster, at **COCO mAP 50 benchmark.** It is also better than SSD and it's variants.



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

YOLOv3's COCO AP metric is on par with SSD but 3x faster. But YOLOv3's AP is still behind RetinaNet. In particular, AP@IoU=.75 drops significantly comparing with RetinaNet which suggests YOLOv3 has higher localization error.

YOLO looses out on COCO benchmarks with a higher value of IoU used to reject a detection. The **50** in COCO 50 benchmark is a measure of how well do the predicted bounding boxes align the the ground truth boxes of the object. 50 here corresponds to 0.5 IoU. If the IoU between the prediction and the ground truth box is less than 0.5, the prediction is classified as a mislocalisation and marked a false positive.