YOLO v3 theory explained

In this tutorial I will explain you what is YOLO model and how it works in details.

**Tutorial content:**

1. What is Yolo?
2. Prerequisites
3. A Fully Convolutional Neural Network
4. Interpreting the output
5. Anchor Boxes and Predictions
6. Dimensions of the Bounding Box
7. Objectness Score and Class Confidences
8. Prediction across different scales
9. Output Processing (Filtering with a threshold on class scores)
10. Implementation

# 1. What is Yolo?

“You Only Look Once” is an algorithm that uses convolutional neural networks for object detection. You only look once, or YOLO, is one of the faster object detection algorithms out there. Though it is not the most accurate object detection algorithm, but it is a very good choice when we need real-time detection, without loss of too much accuracy.

In comparison to recognition algorithms, a detection algorithm does not only predict class labels but detects locations of objects as well. So, It not only classifies the image into a category, but it can also detect multiple Objects within an Image. This Algorithm applies a single Neural network to the Full Image. It means that this network divides the image into regions and predicts bounding boxes and probabilities for each region. These bounding boxes are weighted by the predicted probabilities.

Last time when I was working with object detection I made a [CS:GO aim bot tutorial](https://pylessons.com/Tensorflow-CSGO-aimbot-tutorial/), but it was quite slow (~10 FPS). So now it’s time to make it work faster. One of the biggest takeaways from this experience has been realizing that the best way to go about learning object detection is to implement the algorithms by yourself, from scratch. This is exactly what we’ll do in this tutorial. But before we get out hands dirty with code, we must understand how YOLO works.

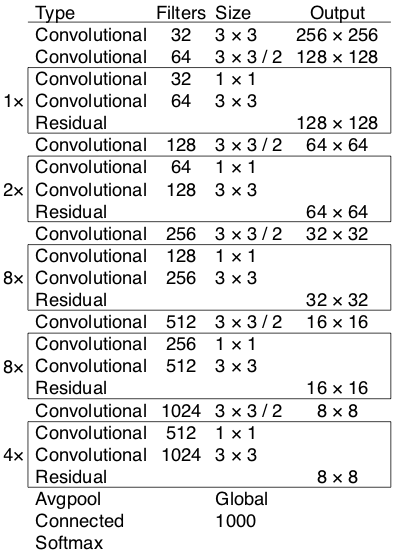
# 2. Prerequisites

To fully understand this tutorial:

* You should understand how convolutional neural networks work. This also includes knowledge of Residual Blocks, skip connections, and Upsampling.
* What is object detection, bounding box regression, IoU and non-maximum suppression.
* You should be able to create simple neural networks with ease.

# 3. A Fully Convolutional Neural Network

YOLO makes use of only convolutional layers, making it a fully convolutional network (FCN). In YOLO v3 paper, the authors present new, deeper architecture of feature extractor called Darknet-53. As it’s name suggests, it contains of 53 convolutional layers, each followed by batch normalization layer and Leaky ReLU activation. 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.



Darknet-53 model

YOLO is invariant to the size of the input image. However, in practice, we might want to stick to a constant input size due to various problems that only show their heads when we are implementing the algorithm.

A big one among these problems is that if we want to process our images in batches (images in batches can be processed in parallel by the GPU, leading to speed boosts), we need to have all images of fixed height and width. This is needed to concatenate multiple images into a large batch.

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**.

# 4. Interpreting the output

First things to know:

* The input is a batch of images of shape (m, 416, 416, 3)
* The output is a list of bounding boxes along with the recognized classes. Each bounding box is represented by 6 numbers (𝑝𝑐,𝑏𝑥,𝑏𝑦,𝑏ℎ,𝑏𝑤,𝑐)(pc,bx,by,bh,bw,c). If we expand 𝑐c into an 80-dimensional vector, each bounding box is then represented by 85 numbers.

The same way as in all object detectors the features learned by the convolutional layers are passed onto a classifier/regressor which makes the detection prediction (coordinates of the bounding boxes, the class label.. etc).

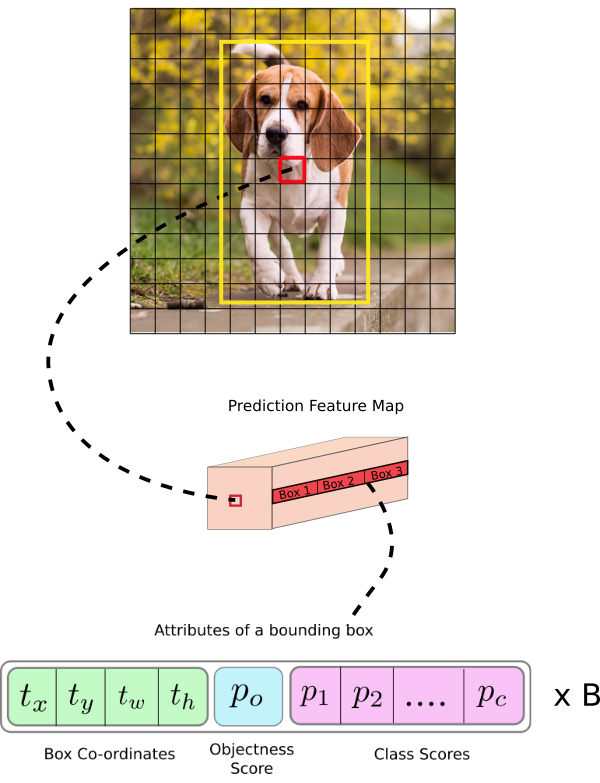
In YOLO, the prediction is done by using a convolutional layer which uses 1 x 1 convolutions. So, the first thing to notice is our output is a feature map. Since we have used 1 x 1 convolutions, the size of the prediction map is exactly the size of the feature map before it. In YOLO v3, the way you interpret this prediction map is that each cell can predict a fixed number of bounding boxes.

For example if we have (B x (5 + C)) entries in the feature map. B represents the number of bounding boxes each cell can predict. According to the paper, each of these B bounding boxes may specialize in detecting a certain kind of object. Each of the bounding boxes have 5 + C attributes, which describe the center coordinates, the dimensions, the objectness score and C class confidences for each bounding box. YOLO v3 predicts 3 bounding boxes for every cell.

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.

This has to do with how YOLO is trained, where only one bounding box is responsible for detecting any given object. First, we must ascertain which of the cells this bounding box belongs to.

To do that, we divide the input image into a grid of dimensions equal to that of the final feature map. Take a look at 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.



Then, 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).

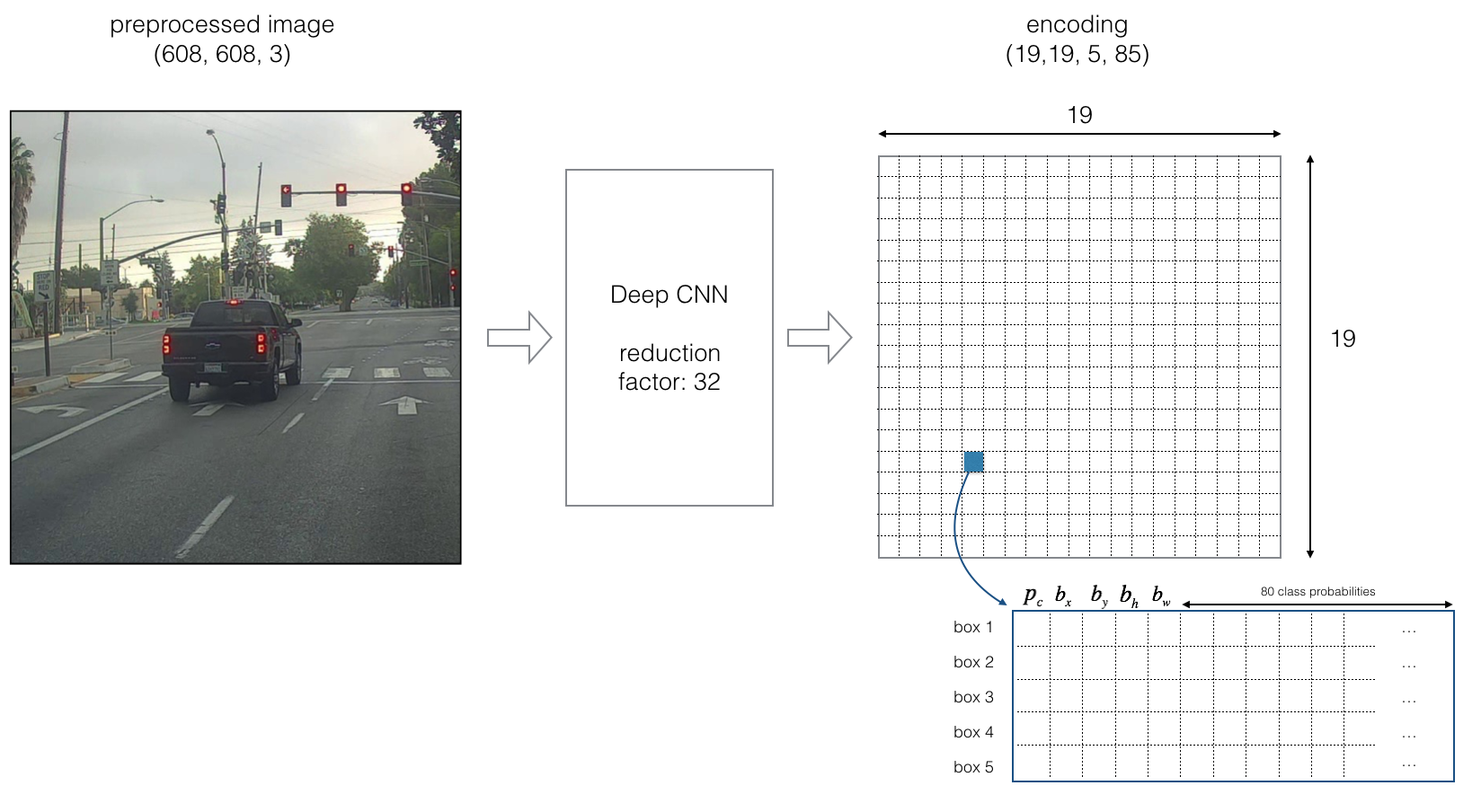
Now, the red cell is the 7th cell in the 7th row on the grid. We now assign the 7th cell in the 7th row on the feature map (corresponding cell on the feature map) as the one responsible for detecting the dog.

Now, this cell can predict three bounding boxes. Which one will be assigned to the dog’s ground truth label? In order to understand that, we must wrap out head around the concept of anchors. (Note that the cell we’re talking about here is a cell on the prediction feature map. We divide the input image into a grid just to determine which cell of the prediction feature map is responsible for prediction).

## For better understanding we can analyse another example with more images:

Now we will use 5 anchor boxes. So we can think of the YOLO architecture as the following: IMAGE (m, 608, 608, 3) -> DEEP CNN -> ENCODING (m, 19, 19, 5, 85). Someone may ask where this 19 number came from ? Answer: same way as before 608/32=19608/32=19

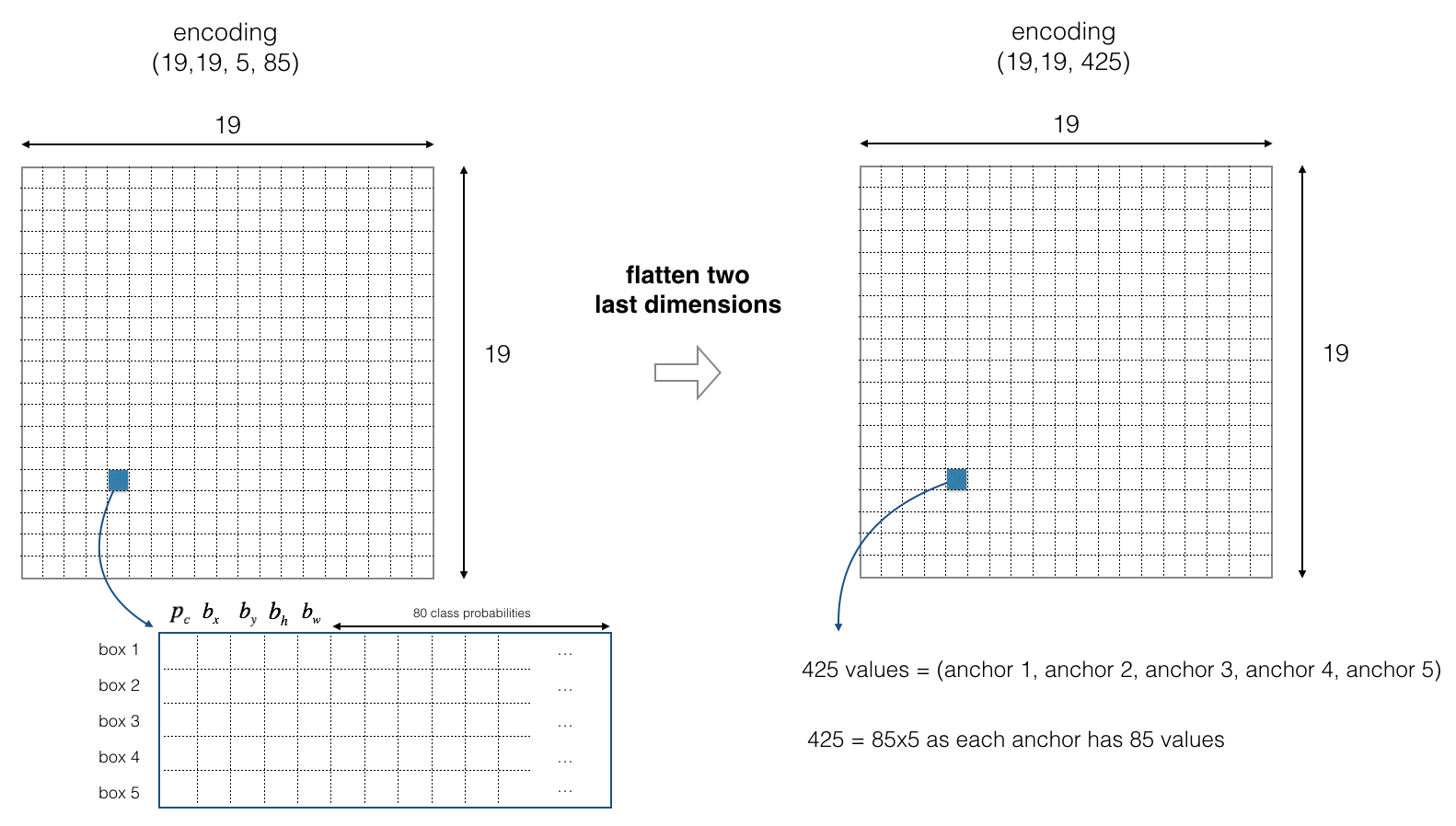
Lets look in greater detail at what this encoding represents:



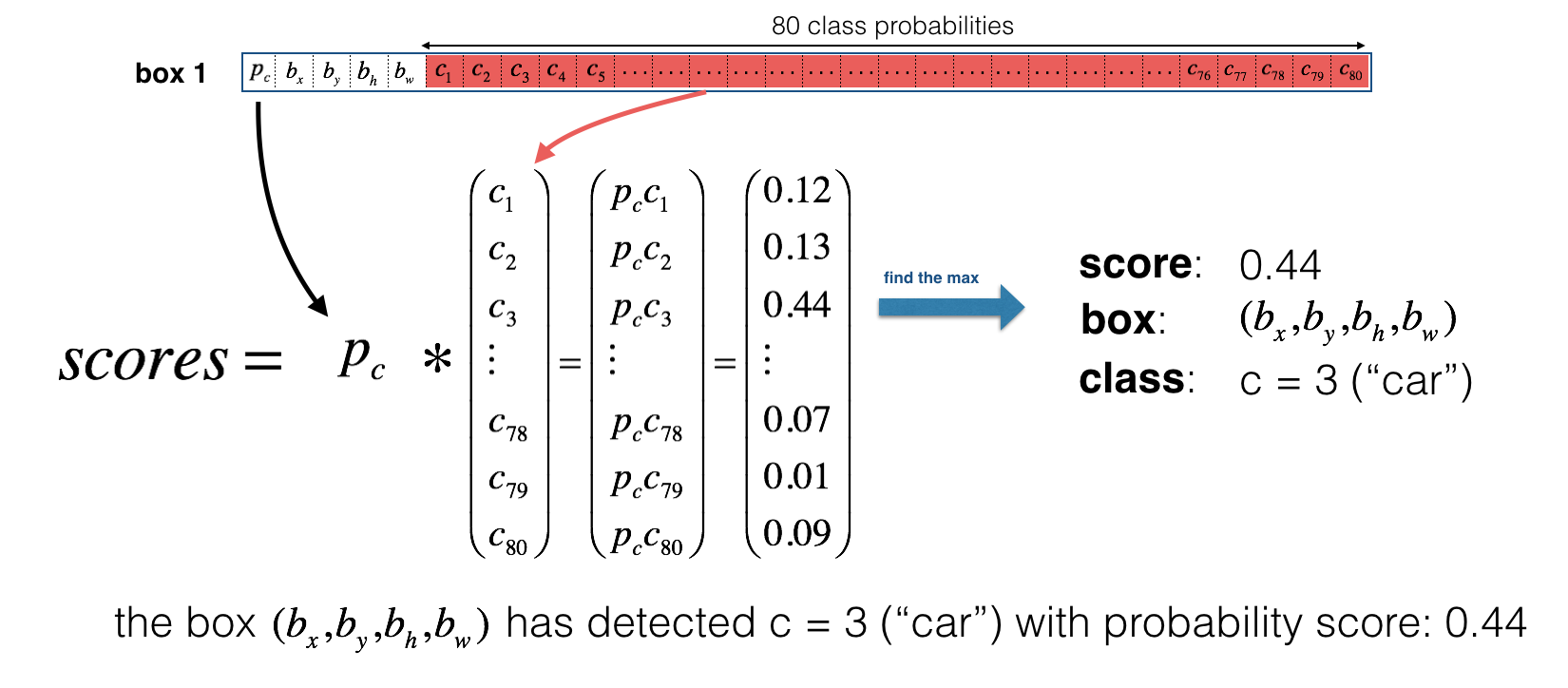
If the center/midpoint of an object falls into a grid cell, that grid cell is responsible for detecting that object.

Since we are using 5 anchor boxes, each of the 19x19 cells thus encodes information about 5 boxes. Anchor boxes are defined only by their width and height.

For simplicity, we will flatten the last two dimensions of the shape (19, 19, 5, 85) encoding. So the output of the Deep CNN is (19, 19, 425):

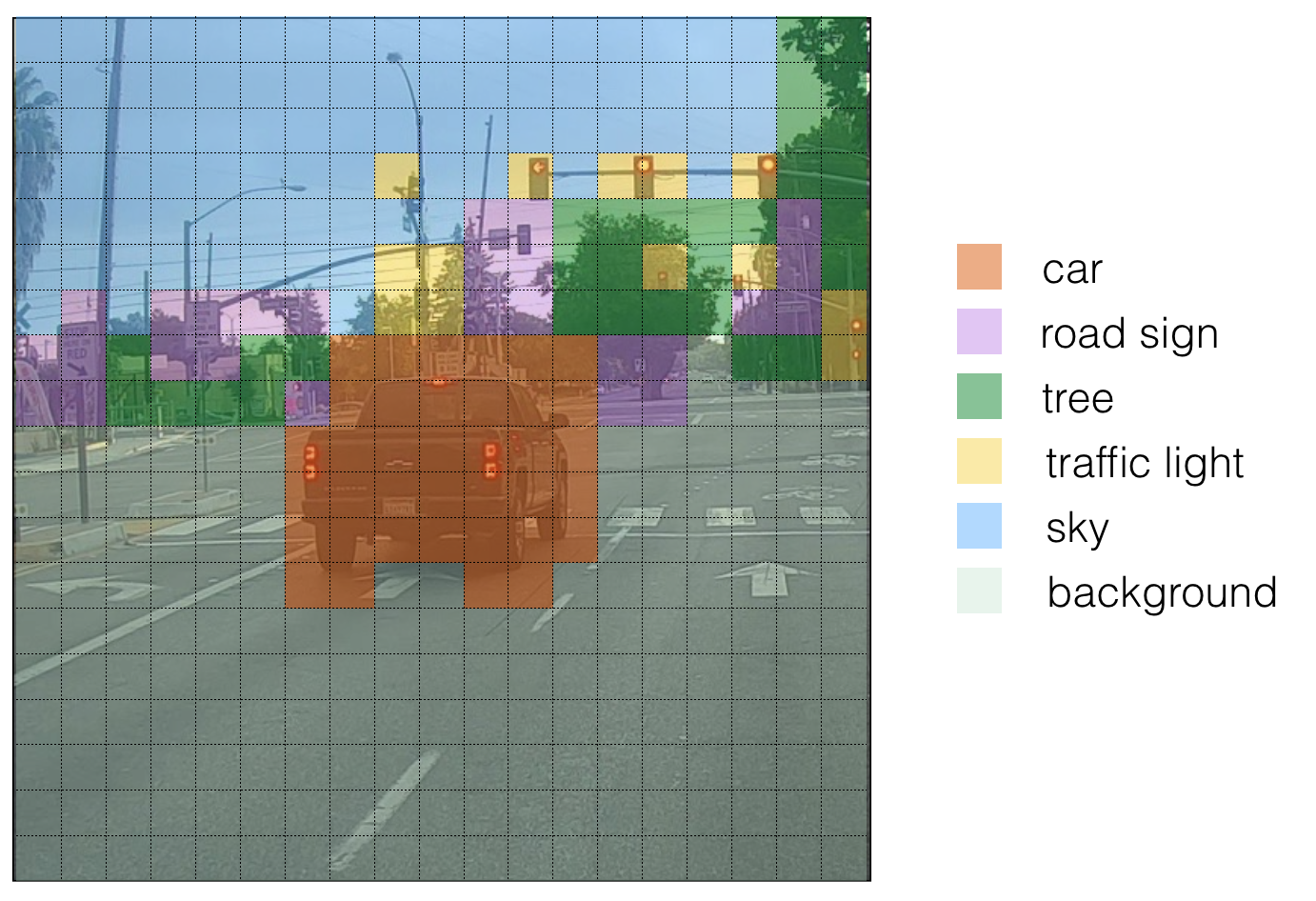


Now, for each box (of each cell) we will compute the following element wise product and extract a probability that the box contains a certain class.



Here’s one way to visualize what YOLO is predicting on an image:

* For each of the 19x19 grid cells, find the maximum of the probability scores (taking a max across both the 5 anchor boxes and across different classes).
* Color that grid cell according to what object that grid cell considers the most likely. Doing this results in this picture:



This above visualization isn’t a core part of the YOLO algorithm itself for making predictions; it’s just a nice way of visualizing an intermediate result of the algorithm.

Another way to visualize YOLO’s output is to plot the bounding boxes that it outputs. Doing that results in a visualization like this:



In the figure above, we plotted only boxes that the model had assigned a high probability to, but this is still too many boxes. We’d like to filter the algorithm’s output down to a much smaller number of detected objects. To do so, we’ll use non-max suppression, more about that we’ll talk in 9th this tutorial part.

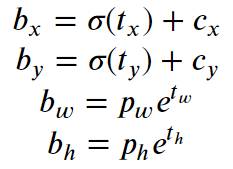
# 5. Anchor Boxes and Predictions

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 predict log-space transforms, or simply offsets to pre-defined default bounding boxes called 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.

Anchors are sort of bounding box priors, that were calculated on the COCO dataset using k-means clustering. We are going to predict the width and height of the box as offsets from cluster centroids. The center coordinates of the box relative to the location of filter application are predicted using a sigmoid function.

The following formula describes how the network output is transformed to obtain bounding box predictions:



Here 𝑏𝑥, 𝑏𝑦, 𝑏𝑤, 𝑏ℎ are the x,y center coordinates, width and height of our prediction. 𝑡𝑥, 𝑡𝑦, 𝑡𝑤, 𝑡ℎ is what the network outputs. 𝑐𝑥 and 𝑐𝑦 are the top-left coordinates of the grid. 𝑝𝑤 and 𝑝ℎ are anchors dimensions for the box.

## Center Coordinates

We are running our center coordinates prediction through a sigmoid function. This forces the value of the output to be between 0 and 1. Normally, YOLO doesn’t predict the absolute coordinates of the bounding box’s center. It predicts offsets which are:

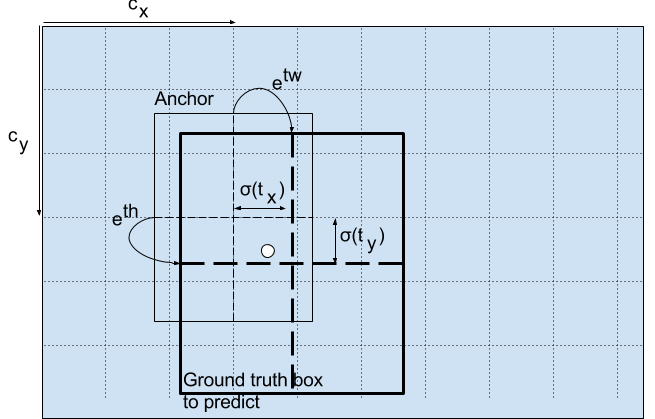
* Relative to the top left corner of the grid cell which is predicting the object.
* Normalized by the dimensions of the cell from the feature map, which is, 1.

For example, consider the case of our above dog image. If the prediction coordinates 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 and y coordinates are greater than one, for example (1.2, 0.7). This means that center lies at (7.2, 6.7). 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. So to solve 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.

# 6. Dimensions of the Bounding Box

The dimensions of the bounding box are predicted by applying a log-space transformation to the output and then multiplying with an anchor.



Here predictions, 𝑏𝑤 and 𝑏ℎ, are normalized by the height and width of the image. (Training labels are chosen this way). So, if the predictions 𝑏𝑥bx and 𝑏𝑦by 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\*0.3, 13\*0.8).

# 7. Objectness Score and Class Confidences

Fir of all object score represents the probability that an object is contained inside a bounding box. It should be nearly 1 for the red and the neighboring grids, whereas almost 0 for, say, the grid at the corners.

The objectness score is also passed through a sigmoid, as it is to be interpreted as a probability.

Talking about Class confidences, they represent the probabilities of the detected object belonging to a particular class (Dog, cat, person, car, bicycle etc). In older YOLO versions it was used to softmax the class scores.

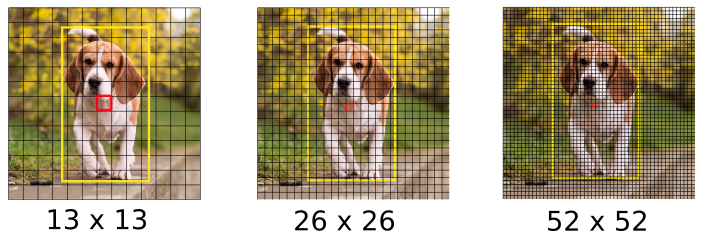
In YOLO authors have decided using sigmoid instead. The reason is that Softmaxing class scores assume that the classes are mutually exclusive. In simple words, if an object belongs to one class, then it cannot belong to another class. This is true for COCO database, which we will implement first.

# 8. 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.

The network downsamples the input image until the first detection layer, where a detection is made using feature maps of a layer with stride 32. Further, layers are upsampled by a factor of 2 and concatenated with feature maps of a previous layers having identical feature map sizes. Another detection is now made at layer with stride 16. The same upsampling procedure is repeated, and a final detection is made at the layer of stride 8.

At each scale, each cell predicts 3 bounding boxes using 3 anchors, making the total number of anchors used 9. (The anchors are different for different scales)



YOLO authors report that this helps get better at detecting small objects, a frequent complaint with the earlier versions of YOLO. Upsampling can help the network learn fine-grained features which are instrumental for detecting small object

# 9. Output Processing (Filtering with a threshold on class scores)

For an image of size 416 x 416, YOLO predicts ((52 x 52) + (26 x 26) + 13 x 13)) x 3 = 10647 bounding boxes. However, in case of our image, there’s only one object, a dog. So, you may ask how do we reduce the detections from 10647 to 1?

First, we filter boxes based on their objectness score. Generally, boxes having scores below a threshold (for example below 0.5) are ignored. Next, 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, so NMS is used to remove multiple detections.

Specifically, we’ll carry out these steps:

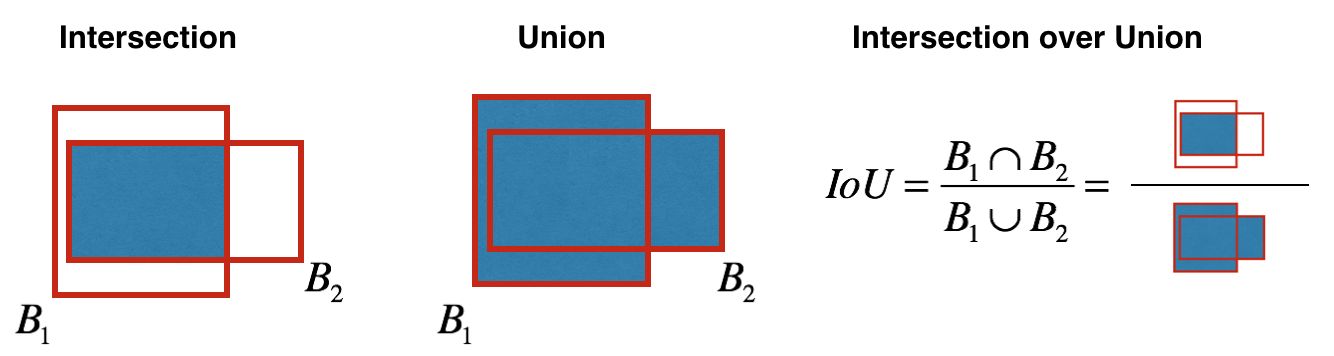
* Get rid of boxes with a low score (meaning, the box is not very confident about detecting a class)
* Select only one box when several boxes overlap with each other and detect the same object (Non-max suppression).

For better understanding I will use same example with car I mentioned above. At first we are going to apply a first filter by thresholding. We would like to get rid of any box for which the class “score” is less than a chosen threshold.

The model gives us a total of 19x19x5x85 numbers, with each box described by 85 numbers. It’ll be convenient to rearrange the (19,19,5,85) (or (19,19,425)) dimensional tensor into the following variables:

* box\_confidence: tensor of shape (19×19,5,1) containing 𝑝𝑐 (confidence probability that there’s some object) for each of the 5 boxes predicted in each of the 19x19 cells.
* boxes: tensor of shape (19×19,5,4) containing (𝑏𝑥,𝑏𝑦,𝑏ℎ,𝑏𝑤) for each of the 5 boxes per cell.
* box\_class\_probs: tensor of shape (19×19,5,80) containing the detection probabilities (𝑐1,𝑐2,…𝑐80) for each of the 80 classes for each of the 5 boxes per cell.

Even after filtering by thresholding over the classes scores, we still end up a lot of overlapping boxes. A second filter for selecting the right boxes is called NMS. NMS uses the very important function called “Intersection over Union”, or IoU.



## Implementing IoU:

1. So we define a box using its two corners (upper left and lower right): (x1, y1, x2, y2) rather than the midpoint and height/width.
2. To calculate the area of a rectangle we need to multiply its height (y2 — y1) by its width (x2 — x1)
3. We also need to find the coordinates (xi1, yi1, xi2, yi2) of the intersection of two boxes:  
   3.1. xi1 = maximum of the x1 coordinates of the two boxes  
   3.2. yi1 = maximum of the y1 coordinates of the two boxes  
   3.3. xi2 = minimum of the x2 coordinates of the two boxes  
   3.4. yi2 = minimum of the y2 coordinates of the two boxes

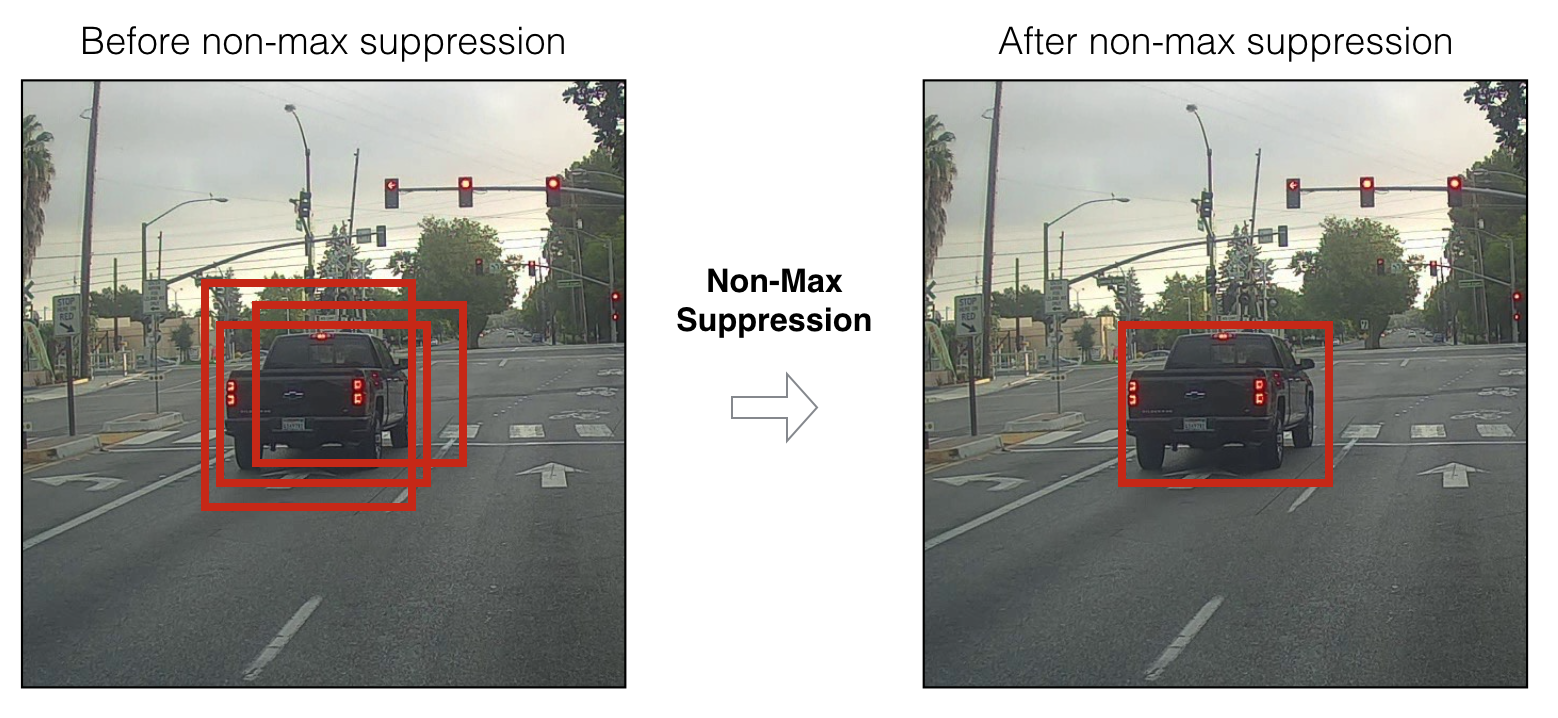
**Arguments:**  
box1 — first box, list object with coordinates (x1, y1, x2, y2)  
box2 — second box, list object with coordinates (x1, y1, x2, y2)

xi1 **=** max(box1[0], box2[0])  
yi1 **=** max(box1[1], box2[1])  
xi2 **=** min(box1[2], box2[2])  
yi2 **=** min(box1[3], box2[3])  
inter\_area **=** (xi2 **-** xi1)**\***(yi2 **-** yi1)# Formula: Union(A,B) = A + B - Inter(A,B)box1\_area **=** (box1[3] **-** box1[1])**\***(box1[2]**-** box1[0])  
box2\_area **=** (box2[3] **-** box2[1])**\***(box2[2]**-** box2[0])  
union\_area **=** (box1\_area **+** box2\_area) **-** inter\_area# compute the IoUIoU **=** inter\_area **/** union\_area

Now, to implement non-max suppression, the key steps are:

1. Select the box that has the highest score.
2. Compute its overlap with all other boxes, and remove boxes that overlap it more than iou\_threshold.
3. Go back to step 1 and iterate until there’s no more boxes with a lower score than the current selected box.

These steps will remove all boxes that have a large overlap with the selected boxes. Only the “best” boxes remain:



# 10. Implementation

YOLO can only detect objects belonging to the classes present in the dataset used to train the network. I will be using the official weight file for our detector. These weights have been obtained by training the network on COCO dataset, and therefore we can detect 80 object categories.

Here one more time I will come back to my example with car, to summarise how our model should work:

1. Input image (608, 608, 3)
2. The input image goes through a CNN, resulting in a (19,19,5,85) dimensional output.
3. After flattening the last two dimensions, the output is a volume of shape (19, 19, 425):  
   3.1. Each cell in a 19x19 grid over the input image gives 425 numbers.  
   3.2. 425 = 5 x 85 because each cell contains predictions for 5 boxes, corresponding to 5 anchor boxes.  
   3.3. 85 = 5 + 80 where 5 is because (𝑝𝑐,𝑏𝑥,𝑏𝑦,𝑏ℎ,𝑏𝑤) has 5 numbers, and and 80 is the number of classes we’d like to detect
4. We then select only few boxes based on:  
   4.1. Score-thresholding: throw away boxes that have detected a class with a score less than the threshold  
   5.1. Non-max suppression: Compute the Intersection over Union and avoid selecting overlapping boxes
5. This gives us YOLO’s final output.

# ****Conslusion****

That’s it for the theory part. I explained enough about the YOLO algorithm to understand how it works. In the next part, I will implement various layers required to run YOLO with TensorFlow.