How does the YOLO Framework Function?

- YOLO first takes an input image:

- The framework then divides the input image into grids (say a 3 X 3 grid):

- Image classification and localization are applied on each grid. YOLO then predicts the bounding boxes and their corresponding class probabilities for objects (if any are found, of course).

Pretty straightforward, isn't it? Let's break down each step to get a more granular understanding of what we just learned.

We need to pass the labelled data to the model in order to train it. Suppose we have divided the image into a grid of size 3 X 3 and there are a total of 3 classes which we want the objects to be classified into. Let's say the classes are Pedestrian, Car, and Motorcycle respectively. So, for each grid cell, the label y will be an eight dimensional vector:

y =	pc
	bx
	by
	bh
	bw
	c1
	c2
	c3

Here,

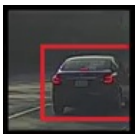
- pc defines whether an object is present in the grid or not (it is the probability)
- bx, by, bh, bw specify the bounding box if there is an object
- c1, c2, c3 represent the classes. So, if the object is a car, c2 will be 1 and c1 & c3 will be 0, and so on

Let's say we select the first grid from the above example:

Since there is no object in this grid, pc will be zero and the y label for this grid will be:

y =	0
	?
	?
	?
	?
	?
	?
	?

Here, '?' means that it doesn't matter what bx, by, bh, bw, c1, c2, and c3 contain as there is no object in the grid. Let's take another grid in which we have a car (c2 = 1):

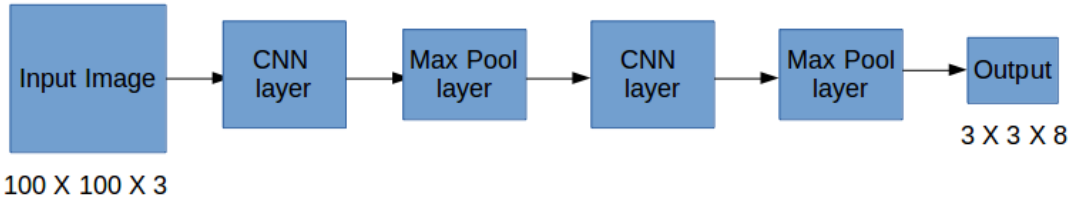


Before we write the y label for this grid, it's important to first understand how YOLO decides whether there actually is an object in the grid. In the above image, there are two objects (two cars), so YOLO will take the mid-point of these two objects and these objects will be assigned to the grid which contains the mid-point of these objects. The y label for the centre left grid with the car will be:

y =	1
	bx
	by
	bh
	bw
	0
	1
	0

Since there is an object in this grid, pc will be equal to 1. bx, by, bh, bw will be calculated relative to the particular grid cell we are dealing with. Since car is the second class, c2 = 1 and c1 and c3 = 0. So, for each of the 9 grids, we will have an eight dimensional output vector. This output will have a shape of 3 X 3 X 8.

So now we have an input image and it's corresponding target vector. Using the above example (input image – 100 X 100 X 3, output – 3 X 3 X 8), our model will be trained as follows:



We will run both forward and backward propagation to train our model. During the testing phase, we pass an image to the model and run forward propagation until we get an output y. In order to keep things simple, I have explained this using a 3 X 3 grid here, but generally in real-world scenarios we take larger grids (perhaps 19 X 19).

Even if an object spans out to more than one grid, it will only be assigned to a single grid in which its mid-point is located. We can reduce the chances of multiple objects appearing in the same grid cell by increasing the more number of grids (19 X 19, for example).

How to Encode Bounding Boxes?

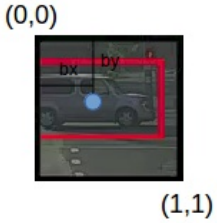
As mentioned earlier, bx, by, bh, and bw are calculated relative to the grid cell we are dealing with. Let's understand this concept with an example. Consider the center-right grid which contains a car:

So, bx, by, bh, and bw will be calculated relative to this grid only. The y label for this grid will be:

y =	1
	bx
	by
	bh
	bw
	0
	1
	0

pc = 1 since there is an object in this grid and since it is a car, c2 = 1. Now, let's see how to decide bx, by, bh, and bw. In YOLO, the coordinates assigned to all the grids are:

bx, by are the x and y coordinates of the midpoint of the object with respect to this grid. In this case, it will be (around) bx = 0.4 and by = 0.3:



bh is the ratio of the height of the bounding box (red box in the above example) to the height of the corresponding grid cell, which in our case is around 0.9. So, bh = 0.9. bw is the ratio of

the width of the bounding box to the width of the grid cell. So, bw = 0.5 (approximately). The y label for this grid will be:

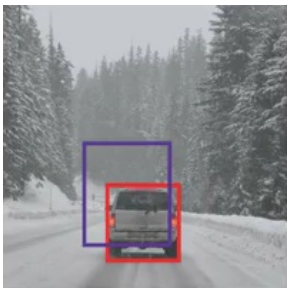
y =	1
	0.4
	0.3
	0.9
	0.5
	0
	1
	0

Notice here that bx and by will always range between 0 and 1 as the midpoint will always lie within the grid. Whereas bh and bw can be more than 1 in case the dimensions of the bounding box are more than the dimension of the grid.

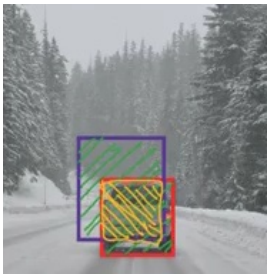
In the next section, we will look at more ideas that can potentially help us in making this algorithm's performance even better.

Intersection over Union and Non-Max Suppression

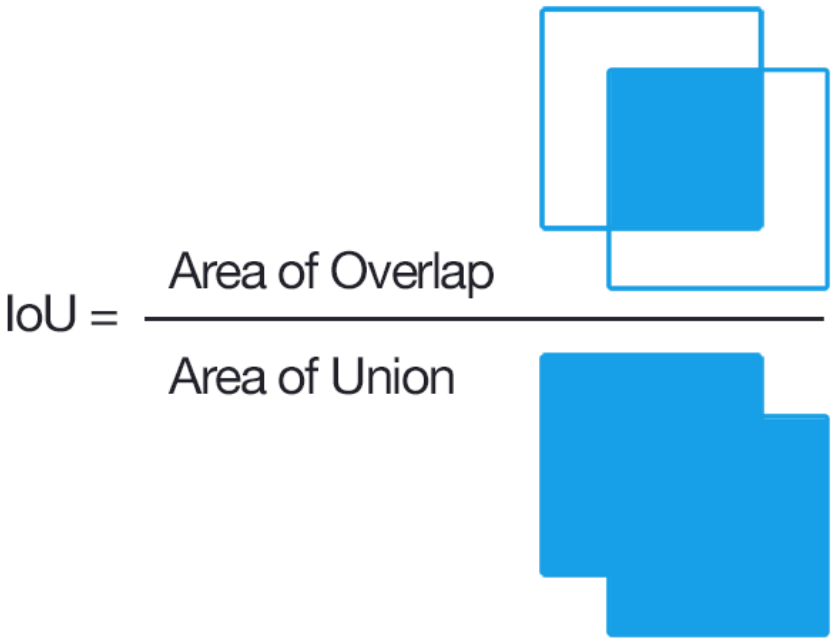
Here's some food for thought – how can we decide whether the predicted bounding box is giving us a good outcome (or a bad one)? This is where Intersection over Union comes into the picture. It calculates the intersection over union of the actual bounding box and the predicted bonding box. Consider the actual and predicted bounding boxes for a car as shown below:



Here, the red box is the actual bounding box and the blue box is the predicted one. How can we decide whether it is a good prediction or not? IoU, or Intersection over Union, will calculate the area of the intersection over union of these two boxes. That area will be:



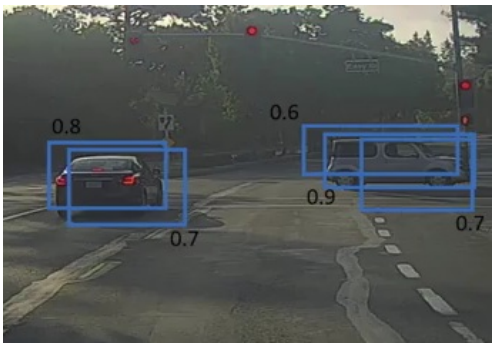
IoU = Area of the intersection / Area of the union



If IoU is greater than 0.5, we can say that the prediction is good enough. 0.5 is an arbitrary threshold we have taken here, but it can be changed according to your specific problem. Intuitively, the more you increase the threshold, the better the predictions become.

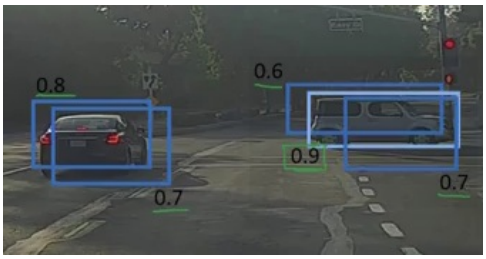
There is one more technique that can improve the output of YOLO significantly – Non-Max Suppression.

One of the most common problems with object detection algorithms is that rather than detecting an object just once, they might detect it multiple times. Consider the below image:



Here, the cars are identified more than once. The Non-Max Suppression technique cleans up this up so that we get only a single detection per object. Let's see how this approach works.

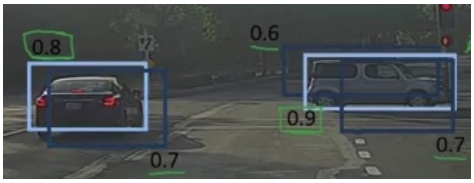
1. It first looks at the probabilities associated with each detection and takes the largest one. In the above image, 0.9 is the highest probability, so the box with 0.9 probability will be selected first:



2. Now, it looks at all the other boxes in the image. The boxes which have high IoU with the current box are suppressed. So, the boxes with 0.6 and 0.7 probabilities will be suppressed in our example:

3. After the boxes have been suppressed, it selects the next box from all the boxes with the highest probability, which is 0.8 in our case:

4. Again it will look at the IoU of this box with the remaining boxes and compress the boxes with a high IoU:



5. We repeat these steps until all the boxes have either been selected or compressed and we get the final bounding boxes:

This is what Non-Max Suppression is all about. We are taking the boxes with maximum probability and suppressing the close-by boxes with non-max probabilities. Let's quickly summarize the points which we've seen in this section about the Non-Max suppression algorithm:

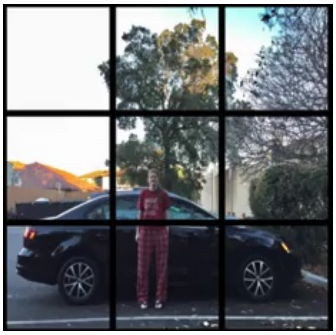
1. Discard all the boxes having probabilities less than or equal to a pre-defined threshold (say, 0.5)
2. For the remaining boxes:
    1. Pick the box with the highest probability and take that as the output prediction
    2. Discard any other box which has IoU greater than the threshold with the output box from the above step
3. Repeat step 2 until all the boxes are either taken as the output prediction or discarded

There is another method we can use to improve the perform of a YOLO algorithm – let's check it out!

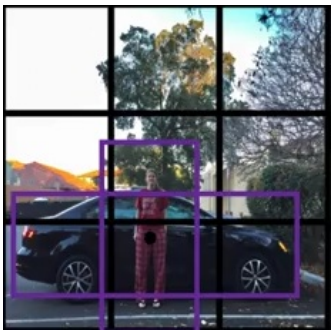
Anchor Boxes

We have seen that each grid can only identify one object. But what if there are multiple objects in a single grid? That can so often be the case in reality. And that leads us to the concept of

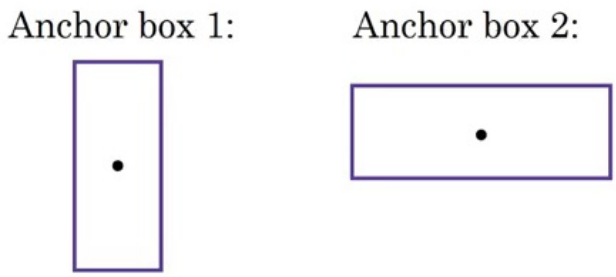
anchor boxes. Consider the following image, divided into a 3 X 3 grid:



Remember how we assigned an object to a grid? We took the midpoint of the object and based on its location, assigned the object to the corresponding grid. In the above example, the midpoint of both the objects lies in the same grid. This is how the actual bounding boxes for the objects will be:



We will only be getting one of the two boxes, either for the car or for the person. But if we use anchor boxes, we might be able to output both boxes. How do we go about doing this? First, we pre-define two different shapes called anchor boxes or anchor box shapes. Now, for each grid, instead of having one output, we will have two outputs. We can always increase the number of anchor boxes as well. I have taken two here to make the concept easy to understand:



This is how the y label for YOLO without anchor boxes looks like:

y =	pc
	bx
	by
	bh
	bw
	c1
	c2
	c3

What do you think the y label will be if we have 2 anchor boxes? I want you to take a moment to ponder this before reading further. Got it? The y label will be:

y =	pc
	bx
	by
	bh
	bw
	c1
	c2
	c3
	pc
	bx
	by
	bh
	bw
	c1
	c2
	c3

The first 8 rows belong to anchor box 1 and the remaining 8 belongs to anchor box 2. The objects are assigned to the anchor boxes based on the similarity of the bounding boxes and the anchor box shape. Since the shape of anchor box 1 is similar to the bounding box for the person, the latter will be assigned to anchor box 1 and the car will be assigned to anchor box 2. The output in this case, instead of 3 X 3 X 8 (using a 3 X 3 grid and 3 classes), will be 3 X 3 X 16 (since we are using 2 anchors).

So, for each grid, we can detect two or more objects based on the number of anchors. Let's combine all the ideas we have covered so far and integrate them into the YOLO framework.

Combining the Ideas

In this section, we will first see how a YOLO model is trained and then how the predictions can be made for a new and previously unseen image.

Training

The input for training our model will obviously be images and their corresponding y labels. Let's see an image and make its y label:

Consider the scenario where we are using a 3 X 3 grid with two anchors per grid, and there are 3 different object classes. So the corresponding y labels will have a shape of 3 X 3 X 16. Now, suppose if we use 5 anchor boxes per grid and the number of classes has been increased to 5. So the target will be 3 X 3 X 10 X 5 = 3 X 3 X 50. This is how the training process is done – taking an image of a particular shape and mapping it with a 3 X 3 X 16 target (this may change as per the grid size, number of anchor boxes and the number of classes).

Testing

The new image will be divided into the same number of grids which we have chosen during the training period. For each grid, the model will predict an output of shape 3 X 3 X 16 (assuming this is the shape of the target during training time). The 16 values in this prediction will be in the same format as that of the training label. The first 8 values will correspond to anchor box 1, where the first value will be the probability of an object in that grid. Values 2-5 will be the bounding box coordinates for that object, and the last three values will tell us which class the object belongs to. The next 8 values will be for anchor box 2 and in the same format, i.e., first the probability, then the bounding box coordinates, and finally the classes.

Finally, the Non-Max Suppression technique will be applied on the predicted boxes to obtain a single prediction per object.

That brings us to the end of the theoretical aspect of understanding how the YOLO algorithm works, starting from training the model and then generating prediction boxes for the objects. Below are the exact dimensions and steps that the YOLO algorithm follows:

- Takes an input image of shape (608, 608, 3)
- Passes this image to a convolutional neural network (CNN), which returns a (19, 19, 5, 85) dimensional output
- The last two dimensions of the above output are flattened to get an output volume of (19, 19, 425):
    - Here, each cell of a 19 X 19 grid returns 425 numbers
    - 425 = 5 * 85, where 5 is the number of anchor boxes per grid
    - 85 = 5 + 80, where 5 is (pc, bx, by, bh, bw) and 80 is the number of classes we want to detect
- Finally, we do the IoU and Non-Max Suppression to avoid selecting overlapping boxes

Loss function

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

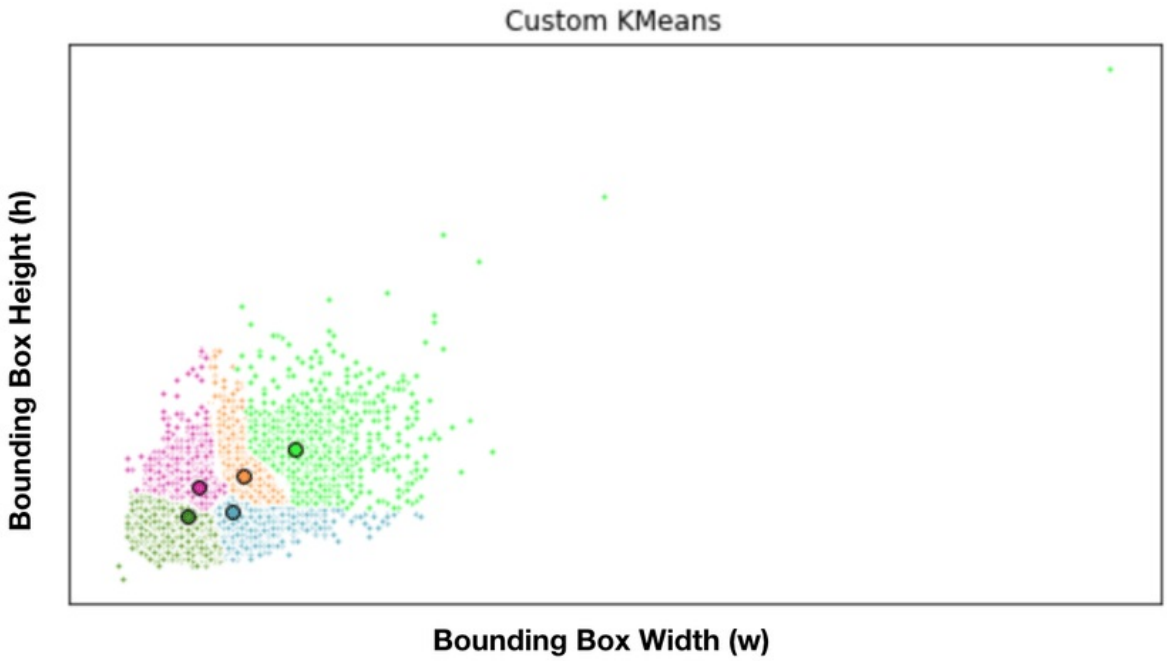
Leaky relu is used as activation function for all layers except final layer.

1 obj i denotes objectness which indicates whether an object is present or not in the cell. 1 obj ij means that jth bounding box in the ith cell is the one which predicts the object. 1st term in the lost function caters for the offset x and y predicted from the cell. 2nd term denotes the error in width and height of the bounding box predicted. The square root is used to minimize the skew that can be caused by large width and height to the loss function. 3rd and 4th terms cater to the loss of network predicting if an object is present in the bounding box. 5th term denotes the loss in predicted class probabilities.

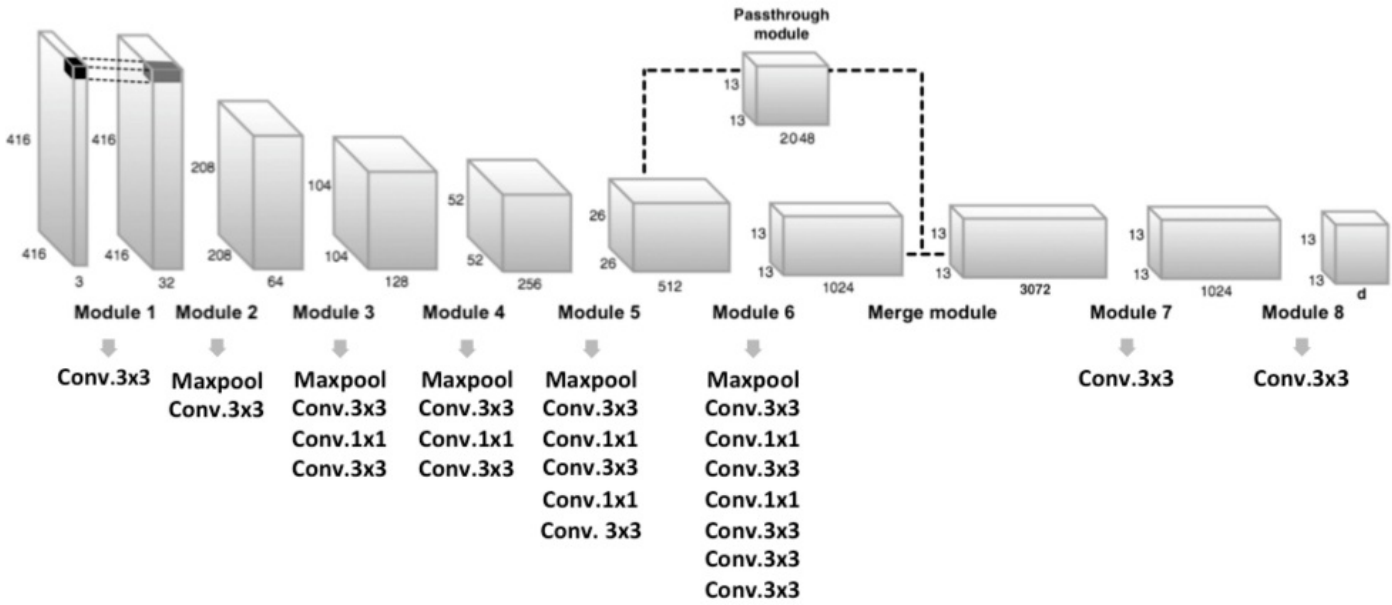
Since the objects in an image are pretty limited the number of cells which don't have object far outweigh the number not having objects and hence the classification loss from the cells which dont have any object can skew the loss and can result in divergence while training. To deal with this different weights are given λcoord = 5 and λnoobj = .5, this giving more weight to the cells having the object and its bounding box prediction loss.

Anchor boxes

The anchor boxes sizes and shapes are determined by running a K-means clustering of all ground truth boxes and a k is picked based on experiments. k=5 is picked up as a good tradeoff between speed and accuracy. Also now each bounding box can have a different object which means a max of k objects can be predicted per cell compared to 1 for YOLOv1.



Architecture



Skip connection is added for better feature representations.