

5000+ Professionals follow this roadmap  to transition in data science

[Download here](#)



[Home](#)

A Practical Guide to Object Detection using the Popular YOLO Framework – Part III (with Python codes)



Pulkit Sharma — Published On December 6, 2018

[Advanced](#) [Algorithm](#) [Computer Vision](#) [Deep Learning](#) [Image](#) [Object Detection](#) [Python](#) [Supervised](#) [Technique](#) [Unstructured Data](#)

Introduction

How easy would our life be if we simply took an already designed framework, executed it, and got the desired result? Minimum effort, maximum reward. Isn't that what we strive for in any profession?

I feel incredibly lucky to be part of our machine learning community where even the top tech behemoths embrace open source technology. Of course it's important to understand and grasp concepts before implementing them, but it's always helpful when the ground work has been laid for you by top industry data scientists and researchers.

This is especially true for deep learning domains like computer vision. Not everyone has the computational resources to build a DL model from scratch. That's where predefined frameworks and pretrained models come in handy. And in this article, we will look at one such framework for object detection – YOLO. It's a supremely fast and accurate framework, as we'll see soon.



So far in our series of posts detailing object detection (links below), we've seen the various algorithms that are used, and how we can detect objects in an image and predict bounding boxes using algorithms of the R-CNN family. We have also looked at the implementation of Faster-RCNN in Python.

In part 3 here, we will learn what makes YOLO tick, why you should use it over other object detection algorithms, and the different techniques used by YOLO. Once we have understood the concept thoroughly, we will then implement it in Python. It's the ideal guide to gain invaluable knowledge and then apply it in a practical hands-on manner.

I highly recommend going through the first two parts before diving into this guide:

- [A Step-by-Step Introduction to the Basic Object Detection Algorithms \(Part 1\)](#)
- [A Practical Implementation of the Faster R-CNN Algorithm for Object Detection \(Part 2\)](#)

Table of Contents

1. What is YOLO and Why is it Useful?
2. How does the YOLO Framework Function?
3. How to Encode Bounding Boxes?
4. Intersection over Union and Non-Max Suppression
5. Anchor Boxes
6. Combining all the Above Ideas
7. Implementing YOLO in Python

What is YOLO and Why is it Useful?

The R-CNN family of techniques we saw in Part 1 primarily use regions to localize the objects within the image. The network does not look at the entire image, only at the parts of the images which have a higher chance of containing an object.

The YOLO framework (You Only Look Once) on the other hand, deals with object detection in a different way. It takes the entire image in a single instance and predicts the bounding box coordinates and class probabilities for these boxes. **The biggest advantage of using YOLO is its superb speed** – it's incredibly fast and can process 45 frames per second. YOLO also understands generalized object representation.

This is one of the best algorithms for object detection and has shown a comparatively similar performance to the R-CNN algorithms. In the upcoming sections, we will learn about different techniques used in YOLO algorithm. The following explanations are inspired by [Andrew NG's course on Object Detection](#) which helped me a lot in understanding the working of YOLO.

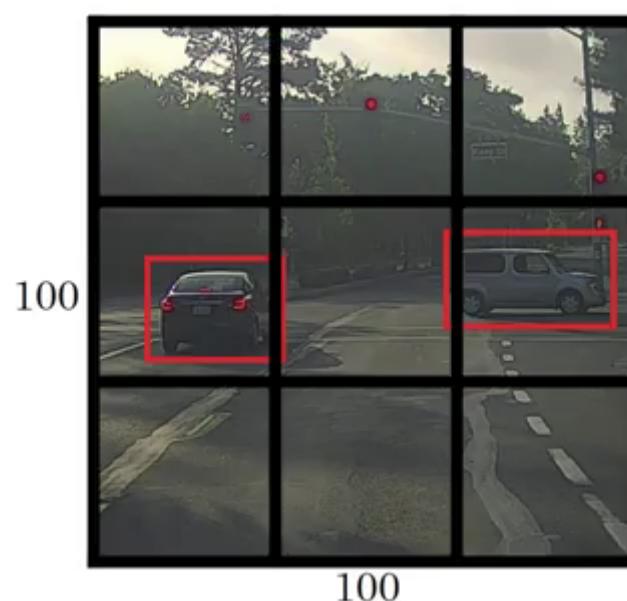
How does the YOLO Framework Function?

Now that we have grasp on why YOLO is such a useful framework, let's jump into how it actually works. In this section, I have mentioned the steps followed by YOLO for detecting objects in a given image.

- 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.

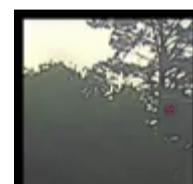
We need to pass the labelled data to the model in order to train it. Suppose we have divided the image into a grid of size 3 X 3 and there are a total of 3 classes which we want the objects to be classified into. Let's say the classes are Pedestrian, Car, and Motorcycle respectively. So, for each grid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

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

Here,

- p_c defines whether an object is present in the grid or not (it is the probability)
- b_x, b_y, b_h, b_w specify the bounding box if there is an object
- c_1, c_2, c_3 represent the classes. So, if the object is a car, c_2 will be 1 and $c_1 & c_3$ will be 0, and so on

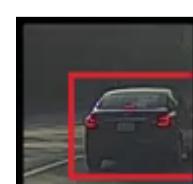
Let's say we select the first grid from the above example:



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

$y =$	0 ? ? ? ? ? ? ?
-------	--------------------------------------

Here, '?' means that it doesn't matter what $b_x, b_y, b_h, b_w, c_1, c_2$, and c_3 contain as there is no object in the grid. Let's take another grid in which we have a car ($c_2 = 1$):



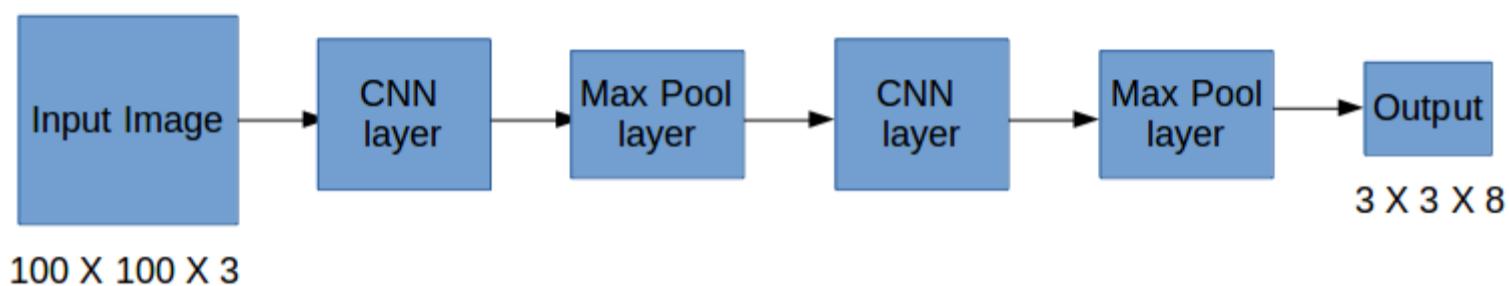
Before we write the y label for this grid, it's important to first understand how YOLO decides whether there actually is an object in the grid. In the above image, there are two objects (two cars), so YOLO will take the mid-point of these two objects and these objects will be assigned to the grid which 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, p_c will be equal to 1. b_x, b_y, b_h, b_w will be calculated relative to the particular grid cell we are dealing with. Since car is the second class, $c_2 = 1$ and $c_1 = c_3 = 0$. So, for each of the 9 grids, we will have an eight dimensional output vector. This output will have a shape of $3 \times 3 \times 8$.

So now we have an input image and its corresponding target vector. Using the above example (input image – $100 \times 100 \times 3$, output – $3 \times 3 \times 8$), 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 3×3 grid here, but generally in real-world scenarios we take larger grids (perhaps 19×19).

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 (19×19 , for example).

How to Encode Bounding Boxes?

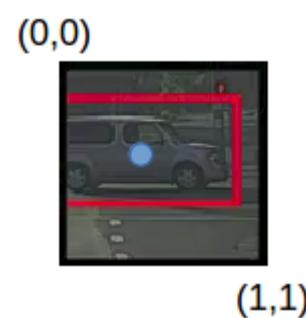
As I mentioned earlier, b_x, b_y, b_h , and b_w are calculated relative to the grid cell we are dealing with. Let's understand this concept with an example. Consider the center-right grid which contains a car:



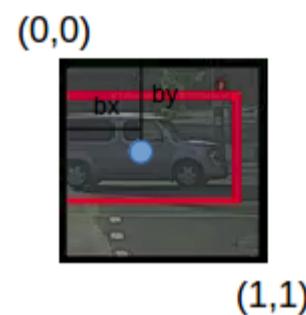
So, b_x, b_y, b_h , and b_w will be calculated relative to this grid only. The y label for this grid will be:

$y =$	1 bx by bh bw 0 1 0
-------	--

$p_c = 1$ since there is an object in this grid and since it is a car, $c_2 = 1$. Now, let's see how to decide b_x, b_y, b_h , and b_w . In YOLO, the coordinates assigned to all the grids are:



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



b_h 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, $b_h = 0.9$. b_w is the ratio of the width of the bounding box to the width of the grid cell. So, $b_w = 0.5$ (approximately). The y label for this grid will be:

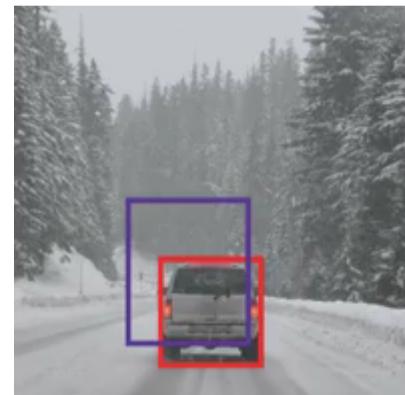
$y =$	<table border="1" style="border-collapse: collapse; text-align: center;"> <tr><td>1</td></tr> <tr><td>0.4</td></tr> <tr><td>0.3</td></tr> <tr><td>0.9</td></tr> <tr><td>0.5</td></tr> <tr><td>0</td></tr> <tr><td>1</td></tr> <tr><td>0</td></tr> </table>	1	0.4	0.3	0.9	0.5	0	1	0
1									
0.4									
0.3									
0.9									
0.5									
0									
1									
0									

Notice here that b_x and b_y will always range between 0 and 1 as the midpoint will always lie within the grid. Whereas b_h and b_w can be more than 1 in case the dimensions of the bounding box are more than the dimension of the grid.

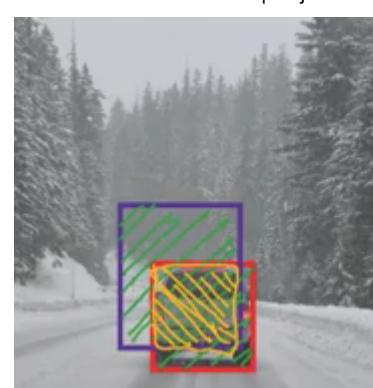
In the next section, we will look at more ideas that can potentially help us in making this algorithm's performance even better.

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:



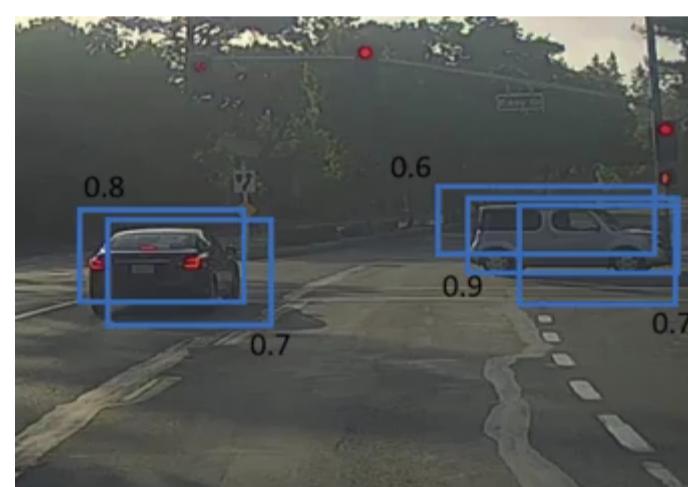
$\text{IoU} = \text{Area of the intersection} / \text{Area of the union}$, i.e.

$\text{IoU} = \text{Area of yellow box} / \text{Area of green box}$

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:



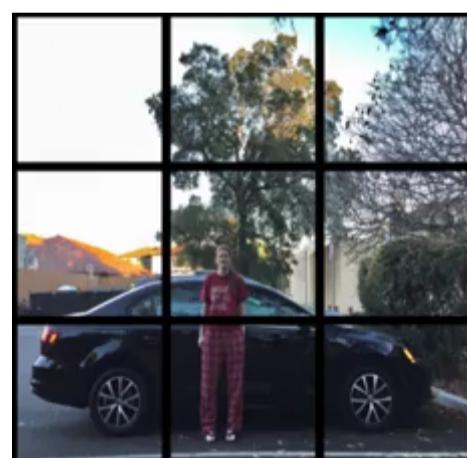
This is what Non-Max Suppression is all about. We are taking the boxes with maximum probability and suppressing the close-by boxes with non-max probabilities. Let's quickly summarize the points which we've seen in this section about the Non-Max suppression algorithm:

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

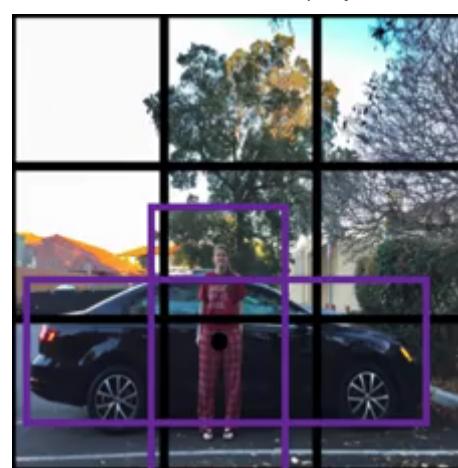
There is another method we can use to improve the perform of a YOLO algorithm – let's check it out!

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:

Anchor box 1:



Anchor box 2:



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 $3 \times 3 \times 8$ (using a 3×3 grid and 3 classes), will be $3 \times 3 \times 16$ (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 3×3 grid with two anchors per grid, and there are 3 different object classes. So the corresponding y labels will have a shape of $3 \times 3 \times 16$. 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 $3 \times 3 \times 10 \times 5 = 3 \times 3 \times 50$. This is how the training process is done – taking an image of a particular shape and mapping it with a $3 \times 3 \times 16$ 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 $3 \times 3 \times 16$ (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 19×19 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

Implementing YOLO in Python

Time to fire up our Jupyter notebooks (or your preferred IDE) and finally implement our learning in the form of code! This is what we have been building up to so far, so let's get the ball rolling.

The code we'll see in this section for implementing YOLO has been taken from [Andrew NG's GitHub repository](#) on Deep Learning. You will also need to download the [pretrained weights](#) required to run this code.

Let's first define the functions that will help us choose the boxes above a certain threshold, find the IoU, and apply Non-Max Suppression on them. Before everything else however, we'll first import the required libraries:

```
import os
import matplotlib.pyplot as plt
from matplotlib.pyplot import imshow
import scipy.io
import scipy.misc
import numpy as np
import pandas as pd
import PIL
import tensorflow as tf
from skimage.transform import resize
from keras import backend as K
from keras.layers import Input, Lambda, Conv2D
from keras.models import load_model, Model
from yolo_utils import read_classes, read_anchors, generate_colors, preprocess_image, draw_boxes, scale_boxes
from yad2k.models.keras_yolo import yolo_head, yolo_boxes_to_corners, preprocess_true_boxes, yolo_loss,
yolo_body

%matplotlib inline
```

Now, let's create a function for filtering the boxes based on their probabilities and threshold:

```
def yolo_filter_boxes(box_confidence, boxes, box_class_probs, threshold = .6):
    box_scores = box_confidence*box_class_probs
    box_classes = K.argmax(box_scores,-1)
    box_class_scores = K.max(box_scores,-1)
    filtering_mask = box_class_scores>threshold
    scores = tf.boolean_mask(box_class_scores,filtering_mask)
    boxes = tf.boolean_mask(boxes,filtering_mask)
    classes = tf.boolean_mask(box_classes,filtering_mask)

    return scores, boxes, classes
```

Next, we will define a function to calculate the IoU between two boxes:

```
def iou(box1, box2):
    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 = (yi2-yi1)*(xi2-xi1)
    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
    iou = inter_area/union_area

    return iou
```

Let's define a function for Non-Max Suppression:

```
def yolo_non_max_suppression(scores, boxes, classes, max_boxes = 10, iou_threshold = 0.5):
    max_boxes_tensor = K.variable(max_boxes, dtype='int32')
    K.get_session().run(tf.variables_initializer([max_boxes_tensor]))
    nms_indices = tf.image.non_max_suppression(boxes,scores,max_boxes,iou_threshold)
    scores = K.gather(scores,nms_indices)
    boxes = K.gather(boxes,nms_indices)
    classes = K.gather(classes,nms_indices)

    return scores, boxes, classes
```

We now have the functions that will calculate the IoU and perform Non-Max Suppression. We get the output from the CNN of shape (19,19,5,85). So, we will create a random volume of shape (19,19,5,85) and then predict the bounding boxes:

```
yolo_outputs = (tf.random_normal([19, 19, 5, 1], mean=1, stddev=4, seed = 1),
                tf.random_normal([19, 19, 5, 2], mean=1, stddev=4, seed = 1),
                tf.random_normal([19, 19, 5, 2], mean=1, stddev=4, seed = 1),
                tf.random_normal([19, 19, 5, 80], mean=1, stddev=4, seed = 1))
```

Finally, we will define a function which will take the outputs of a CNN as input and return the suppressed boxes:

```
def yolo_eval(yolo_outputs, image_shape = (720., 1280.), max_boxes=10, score_threshold=.6, iou_threshold=.5):
    box_confidence, box_xy, box_wh, box_class_probs = yolo_outputs
    boxes = yolo_boxes_to_corners(box_xy, box_wh)
    scores, boxes, classes = yolo_filter_boxes(box_confidence, boxes, box_class_probs, threshold =
score_threshold)
    boxes = scale_boxes(boxes, image_shape)
    scores, boxes, classes = yolo_non_max_suppression(scores, boxes, classes, max_boxes, iou_threshold)

    return scores, boxes, classes
```

Let's see how we can use the *yolo_eval*/function to make predictions for a random volume which we created above:

```
scores, boxes, classes = yolo_eval(yolo_outputs)
```

How does the outlook look?

```
with tf.Session() as test_b:
    print("scores[2] = " + str(scores[2].eval()))
    print("boxes[2] = " + str(boxes[2].eval()))
    print("classes[2] = " + str(classes[2].eval()))

    scores[2] = 138.79124
    boxes[2] = [1292.3297 -278.52167 3876.9893 -835.56494]
    classes[2] = 54
```

'scores' represents how likely the object will be present in the volume. 'boxes' returns the (x1, y1, x2, y2) coordinates for the detected objects. 'classes' is the class of the identified object.

Now, let's use a pretrained YOLO algorithm on new images and see how it works:

```
sess = K.get_session()
class_names = read_classes("model_data/coco_classes.txt")
anchors = read_anchors("model_data/yolo_anchors.txt")

yolo_model = load_model("model_data/yolo.h5")
```

After loading the classes and the pretrained model, let's use the functions defined above to get the *yolo_outputs*.

```
yolo_outputs = yolo_head(yolo_model.output, anchors, len(class_names))
```

Now, we will define a function to predict the bounding boxes and save the images with these bounding boxes included:

```
def predict(sess, image_file):
    image, image_data = preprocess_image("images/" + image_file, model_image_size = (608, 608))
    out_scores, out_boxes, out_classes = sess.run([scores, boxes, classes], feed_dict={yolo_model.input: image_data, K.learning_phase(): 0})

    print('Found {} boxes for {}'.format(len(out_boxes), image_file))

    # Generate colors for drawing bounding boxes.
    colors = generate_colors(class_names)

    # Draw bounding boxes on the image file
    draw_boxes(image, out_scores, out_boxes, out_classes, class_names, colors)

    # Save the predicted bounding box on the image
    image.save(os.path.join("out", image_file), quality=90)

    # Display the results in the notebook
    output_image = scipy.misc.imread(os.path.join("out", image_file))

    plt.figure(figsize=(12,12))
    imshow(output_image)

    return out_scores, out_boxes, out_classes
```

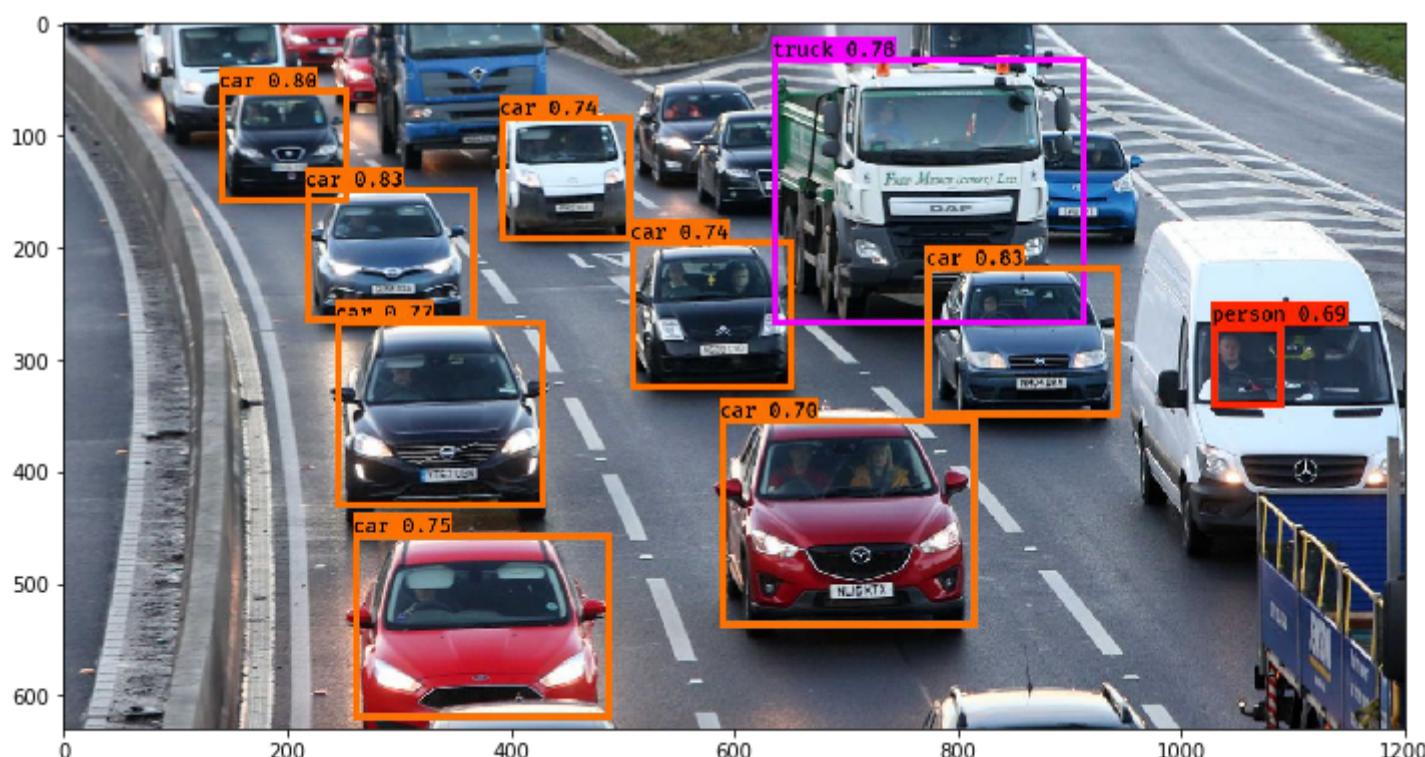
Next, we will read an image and make predictions using the *predict* function:

```
img = plt.imread('images/img.jpg')
image_shape = float(img.shape[0]), float(img.shape[1])
scores, boxes, classes = yolo_eval(yolo_outputs, image_shape)
```

Finally, let's plot the predictions:

```
out_scores, out_boxes, out_classes = predict(sess, "img.jpg")
```

```
Found 10 boxes for img.jpg
person 0.69 (1027, 270) (1091, 343)
car 0.70 (587, 352) (817, 539)
car 0.74 (390, 82) (509, 194)
car 0.74 (507, 193) (653, 327)
car 0.75 (259, 455) (490, 621)
car 0.77 (243, 265) (431, 432)
truck 0.78 (634, 30) (915, 269)
car 0.80 (139, 58) (255, 160)
car 0.83 (216, 146) (369, 267)
car 0.83 (771, 216) (945, 351)
```



Not bad! I especially like that the model correctly picked up the person in the mini-van as well.

End Notes

Here's a brief summary of what we covered and implemented in this guide:

- YOLO is a state-of-the-art object detection algorithm that is incredibly fast and accurate
- We send an input image to a CNN which outputs a 19 X 19 X 5 X 85 dimension volume.
- Here, the grid size is 19 X 19 and each grid contains 5 boxes
- We filter through all the boxes using Non-Max Suppression, keep only the accurate boxes, and also eliminate overlapping boxes

YOLO is one of my all-time favorite frameworks and I'm sure you'll see why once you implement the code on your own machine. It's a great way of getting your hands dirty with a popular computer vision algorithm. If you have any questions or feedback regarding this guide, connect with me in the comments section below.

[Computer Vision](#) [deep learning](#) [object detection](#) [python](#) [YOLO](#)



About the Author



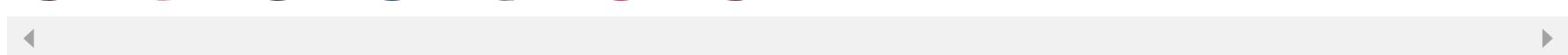
[Pulkit Sharma](#)

My research interests lies in the field of Machine Learning and Deep Learning. Possess an enthusiasm for learning new skills and technologies.

Our Top Authors



[view more](#)



Download

Analytics Vidhya App for the Latest blog/Article



[Previous Post](#)[Next Post](#)[5 Best Machine Learning GitHub Repositories & Reddit Discussions \(November 2018\)](#)[Building a Face Detection Model from Video using Deep Learning \(Python Implementation\)](#)

34 thoughts on "A Practical Guide to Object Detection using the Popular YOLO Framework – Part III (with Python codes)"



sset says:

December 06, 2018 at 11:13 am

Thanks for article. How does YOLO compare with Faster-RCNN for detection of very small objects like scratches on metal surface? My observation was - RCNN lacks an elegant way to compute anchor sizes based on dataset... also I attempted changing scales,strides,box size results are bad. How do we custom train for YOLO?

[Reply](#)

Pulkit Sharma says:

December 06, 2018 at 11:57 am

Hi, YOLO is faster in comparison to Faster-RCNN. Their accuracies are comparatively similar. YOLO does not work pretty well for small objects. In order to improve its performance on smaller objects, you can try the following things:
Increase the number of anchor boxes
Decrease the threshold for IoU
This may give better results.

[Reply](#)

Hamza Mukhtar says:

December 07, 2018 at 2:48 am

Is this Yolo implemented on GPU based? How to training Yolo for customize object? Is there any yolo code or tutorial using Tensorflow GPU base is available ? I only want to detect vehicles. Please guide me.

[Reply](#)

Pulkit Sharma says:

December 07, 2018 at 12:20 pm

Hi Hamza, Yes, I have trained this model on GPU. For customized training, you can pass your own dataset and the bounding boxes and train the model to get weights.

[Reply](#)

Hamza Mukhtar says:

December 10, 2018 at 2:56 am

Hello Dear Pulkit Sharma, Hope you will be in good health. Kindly explain the steps for the training of Yolo on GPU for vehicles only. Where is the training code file? Explain the step of passing the dataset? Explain the step of passing the bounding box?

[Reply](#)

Pulkit Sharma says:

December 10, 2018 at 11:21 am

Hi Hamza, Please refer to this [GitHub link](#).

[Reply](#)

Hamza says:

December 10, 2018 at 8:46 pm

Hi Pulkit, Link to download back-end weight is not working. kindly guide me.

[Reply](#)

Pulkit Sharma says:

December 12, 2018 at 11:47 am

Hi Hamza, The link is working fine at my end. Can you please share what is the error that you are getting?

[Reply](#)

Hamza Mukhtar says:
December 13, 2018 at 8:32 pm

Dear Pulkit, Link to download pretrained weights for backend, which is a OneDrive link, not have said file.

[Reply](#)

Pulkit Sharma says:
December 14, 2018 at 11:01 am

Hi Hamza, Once you open the link, there will be an option to download the zip file on the top right side.

[Reply](#)

Priya Khare says:
December 17, 2018 at 2:34 pm

Hi PulKit, Very nicely explained article. In your Training paragraph , you have said that the output vector for 5 anchor boxes and 5 classes will be 3X3X25, shouldnt it be 3X3X50 instead , 50 comprising of [object(y/N)(1),coordinates of boundary box(4),5 classes(1 for each class)] X no. of anchor boxes . i.e 10 X 5. = 50 Pls correct me if wrong

[Reply](#)

Arijit Sengupta says:
December 17, 2018 at 8:04 pm

I followed the instructions as given but there are a lot of errors in classification. It is identifying objects but not correctly. Any reason why this might be happening?

[Reply](#)

Pulkit Sharma says:
December 18, 2018 at 11:51 am

Hi Arijit, Can you share some of the results? You can try to train your own model as well instead of using the trained weights.

[Reply](#)

Pulkit Sharma says:
December 18, 2018 at 11:55 am

Hi Priya, That's correct. I have updated it in the article. Thanks for pointing it out

[Reply](#)

Valli Kumar says:
December 20, 2018 at 12:58 pm

Hello Pulkit Sharma , How do you rate YOLO working for Face(face also being a type of object) detection, especially in video frames from surveillance feeds. ? Kindly also comment on the speed(real time performance) of this.What type of h/w platform would suit. ? How do you compare its performance with that of Voila Jones frame work (keeping human face as the object,in mind) regards, Thank you.

[Reply](#)

Ganesh says:
December 24, 2018 at 3:52 pm

Hi Pulkit, The article explains the concept with ease. Thanks for the nice article. I want to implement the YOLO v2 on a customized dataset. The input dimension is greater than the required size of YOLOv2(400x400 pixels). My doubt is should I resize the images and annotated them or is it fine to train the model with original annotations and resized images?

[Reply](#)

Pulkit Sharma says:
December 24, 2018 at 3:54 pm

Hi Ganesh, If you are resizing the image, annotations should also be resized accordingly. If you use the original annotations, you will not get desired results.

[Reply](#)

Pulkit Sharma says:
January 23, 2019 at 11:36 am

Hi Valli Kumar, I have not yet tested YOLO for detecting faces. For object detection it is faster than most of the other object detection techniques so, I hope it will also work good for face detection. I am currently working on the same project. First I will try

different RNN techniques for face detection and then will try YOLO as well. Then only we can compare it with the other techniques. I will share the results as soon as I am done with this project.

[Reply](#)



Muhammed Hassan M says:

January 25, 2019 at 2:21 pm

how to train our own dataset with yolo v3

[Reply](#)



Pulkit Sharma says:

January 28, 2019 at 11:40 am

Hi Muhammed, You can refer to [this article](#) to learn how to train yolo on custom dataset.

[Reply](#)



Rak says:

January 29, 2019 at 3:06 pm

I can't find any zip file of pretrained weights for backend too. So, what should I do.

[Reply](#)



Pulkit Sharma says:

January 29, 2019 at 4:16 pm

Hi Rak, The pretrained weights can be found in [this link](#). I have mentioned it in the article as well.

[Reply](#)



Shree says:

February 11, 2019 at 1:50 am

I'm not able to load yolo.h5. When I do so the kernel crashes. What do I do in this case?

[Reply](#)



Pulkit Sharma says:

February 11, 2019 at 11:29 am

Hi Shree, What is the configuration of your system? Does it have a GPU? If not, try to use Google colab which will give you access to free GPU.

[Reply](#)



Rak says:

February 25, 2019 at 12:55 pm

Hi PULKIT, I load model using my own custom pre-train instead of yolo.h5. My model has only 2 classes. I run following your code and I got error because of "yolo_head" function in keras_yolo.py . the error look ==> Traceback (most recent call last): File "C:\Python36\Food-Diary_Project\YOLO_Food-Diary_new\YOLO_food_prediction_new.py", line 87, in yolo_outputs = yolo_head(yolo_model.output, anchors, len(class_names)) File "C:\Python36\Food-Diary_Project\YOLO_Food-Diary_new\yad2k\models\keras_yolo.py", line 109, in yolo_head conv_index = K.cast(conv_index, K.dtype(feats)) File "C:\Python36\lib\site-packages\keras\backend\tensorflow_backend.py", line 649, in dtype return x.dtype.base_dtype.name AttributeError: 'list' object has no attribute 'dtype' How can I solve this issue?

[Reply](#)



Muhammad Hasnain Ahsan says:

May 31, 2019 at 11:09 am

Hi Pulkit, in IoU section, you said that box with IoU greater than 0.5 will be ignored in this case, isn't it inverse to what you just said?

[Reply](#)



Pulkit Sharma says:

May 31, 2019 at 5:10 pm

Hi Ahsan, Firstly, we select the bounding box with the maximum probability. Now, we calculate the IoU of other bounding boxes with the box that we have selected. If this IoU is more, than we can infer that both the boxes are for the same object and can be removed. This is exactly what have been explained in the article as well.

[Reply](#)



Aditya says.
June 27, 2019 at 1:43 pm

the link shows that the site cannot be reached.

[Reply](#)



Pulkit Sharma says:
June 28, 2019 at 4:48 pm

Hi Aditya, Can you specify which link is not working for you? I just checked and all the links are working fine at my end.

[Reply](#)



Kunal Jain says:
July 09, 2019 at 7:44 pm

```
Hello Pulkit, I am preparing my model using celeb dataset for gender recognition. Firstly I prepare my training dataset then model but during model I am getting error i.e Negative dimension size caused by subtracting 2 from 1 for 'MaxPool_32' (op: 'MaxPool') with input shapes: [?,50,1,16]. my code for model is-- model = Sequential() model.add(Convolution2D(16, 3, 3,input_shape=(50,50,1),border_mode='same',subsample=(1,1))) model.add(LeakyReLU(alpha=0.1)) model.add(MaxPooling2D(pool_size=(2, 2))) model.add(Convolution2D(32,3,3 ,border_mode='same')) model.add(LeakyReLU(alpha=0.1)) model.add(MaxPooling2D(pool_size=(2, 2),border_mode='valid')) model.add(Convolution2D(64,3,3 ,border_mode='same')) model.add(LeakyReLU(alpha=0.1)) model.add(MaxPooling2D(pool_size=(2, 2),border_mode='valid')) model.add(Convolution2D(128,3,3 ,border_mode='same')) model.add(LeakyReLU(alpha=0.1)) model.add(MaxPooling2D(pool_size=(2, 2),border_mode='valid')) model.add(Convolution2D(256,3,3 ,border_mode='same')) model.add(LeakyReLU(alpha=0.1)) model.add(MaxPooling2D(pool_size=(2, 2),border_mode='valid')) model.add(Convolution2D(512,3,3 ,border_mode='same')) model.add(LeakyReLU(alpha=0.1)) model.add(MaxPooling2D(pool_size=(2, 2),border_mode='valid')) model.add(Convolution2D(1024,3,3 ,border_mode='same')) model.add(LeakyReLU(alpha=0.1)) model.add(Convolution2D(1024,3,3 ,border_mode='same')) model.add(LeakyReLU(alpha=0.1)) model.add(Flatten()) model.add(Dense(256)) model.add(Dense(4096)) model.add(LeakyReLU(alpha=0.1)) model.add(Dense(1470)) model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy']) model.fit(X, y, batch_size=32, nb_epoch=10, validation_split=0.1)
```

[Reply](#)



Pulkit Sharma says:
July 10, 2019 at 1:24 pm

Hi Kunal, You can either try to increase the size of the images or reduce the max pool size.

[Reply](#)



kalpesh says:
July 13, 2019 at 2:31 pm
can we detect the only person using this technique

[Reply](#)



Pulkit Sharma says:
July 15, 2019 at 5:59 pm

Hi Kalpesh, Yes! You can. For this case, you first have to train your own model. So, you can train the model for detecting just the persons in the image. To do so, you can just give the true labels (bounding boxes for object detection case) for the persons in the image and train the model. Once the model is trained, you can test the model by predicting on new images and the model should generate the bounding boxes for all the persons in those new images.

[Reply](#)



sonia says:
July 25, 2019 at 6:10 pm
Did you fix it? I am getting same error

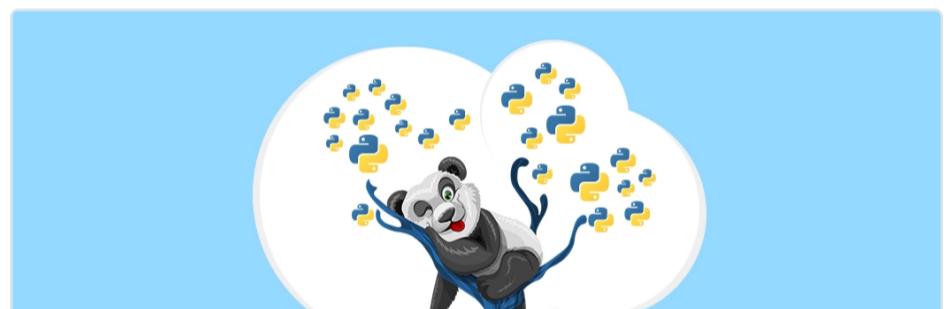
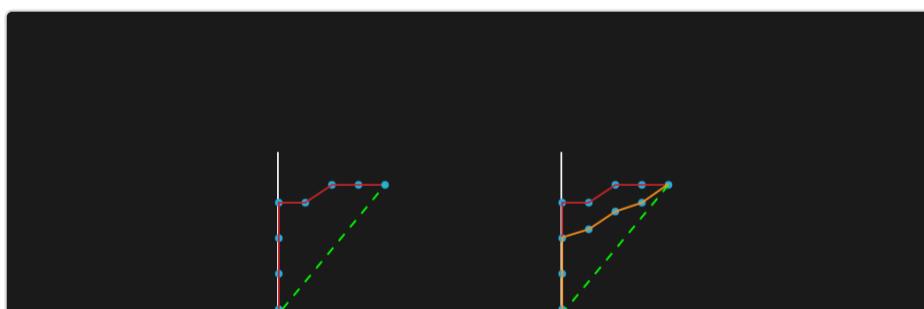
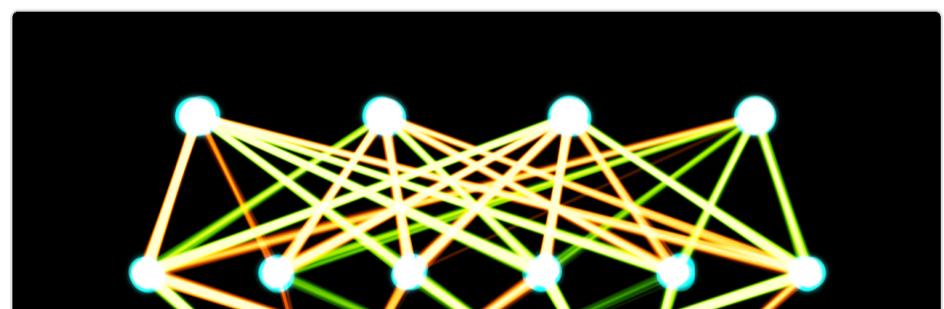
[Reply](#)

Leave a Reply

Your email address will not be published. Required fields are marked *

Notify me of follow-up comments by email. Notify me of new posts by email.

Top Resources

[Python Tutorial: Working with CSV file for Data Science](#) Harika Bonthu - AUG 21, 2021[Joins in Pandas: Master the Different Types of Joins in..](#) Abhishek Sharma - FEB 27, 2020[AUC-ROC Curve in Machine Learning Clearly Explained](#) Aniruddha Bhandari - JUN 16, 2020[Boost Model Accuracy of Imbalanced COVID-19 Mortality Prediction Using GAN-based..](#) Bala Gangadhar Thilak Adiboina - OCT 07, 2020[Analytics Vidhya](#)[About Us](#)[Our Team](#)[Data Scientists](#)[Blog](#)[Hackathon](#)

[Download App](#)[Careers](#)
[Contact us](#)
Companies[Post Jobs](#)
[Trainings](#)
[Hiring Hackathons](#)
[Advertising](#)[Discussions](#)
[Apply Jobs](#)
Visit us

© Copyright 2013-2022 Analytics Vidhya.

[Privacy Policy](#) [Terms of Use](#) [Refund Policy](#)

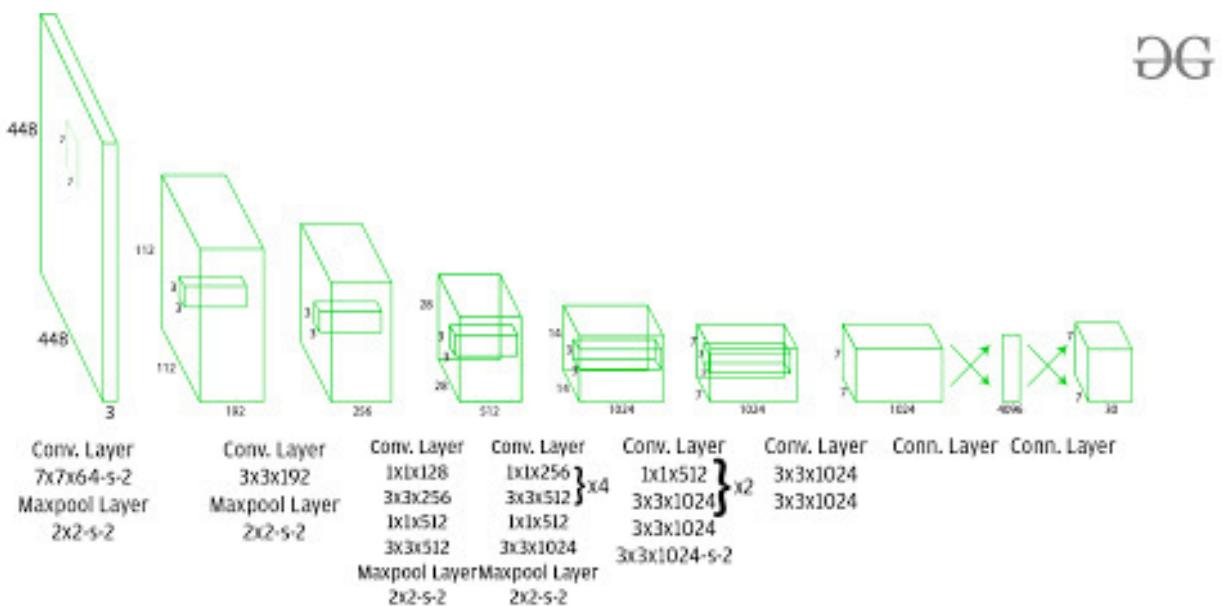
YOLO : You Only Look Once – Real Time

 GeeksforGeeks

Data Structures Algorithms Interview Preparation Topic-wise Practice C++ Java Python

YOLO was proposed by Joseph Redmond *et al.* in 2015. It was proposed to deal with the problems faced by the object recognition models at that time, Fast R-CNN is one of the state-of-the-art models at that time but it has its own challenges such as this network cannot be used in real-time, because it takes 2-3 seconds to predict an image and therefore cannot be used in real-time. Whereas, in YOLO we have to look only once in the network i.e. only one forward pass is required through the network to make the final predictions.

Architecture:



This architecture takes an image as input and resizes it to 448*448 by keeping the aspect ratio same and performing padding. This image is then passed in the CNN network. This model has 24 convolution layers, 4 max-pooling layers followed by 2 fully connected layers. For the reduction of the number of layers (Channels), we use

Start Your Coding Journey Now!

[Login](#)
[Register](#)

fully connected layer and reshaping it to size $(7, 7, 30)$.

This architecture uses Leaky ReLU as its activation function in whole architecture except the last layer where it uses linear activation function. The definition of Leaky ReLU can be found [here](#). Batch normalization also helps to regularize the model. Dropout technique is also used to prevent overfitting.

Training:

This model is trained on the *ImageNet-1000* dataset. The model is trained over a week and achieve top-5 accuracy of *88%* on *ImageNet 2012* validation which is comparable to GoogLeNet (2014 ILSVRC winner), the state of the art model at that time. Fast YOLO uses fewer layers (*9 instead of 24*) and fewer filters. Except this, the fast YOLO have all parameters similar to YOLO. YOLO uses sum-squared error loss function which is easy to optimize. However, this function gives equal weight to the classification and localization task. The loss function defined in YOLO as follows:

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

where,

$\mathbb{1}_i^{\text{obj}}$ denotes if object is present in cell i .

$\mathbb{1}_{ij}^{\text{obj}}$ denotes j^{th} bounding box responsible for prediction of object in the cell i .

λ_{coord} and λ_{noobj} are regularization parameter required to balance the loss function.

In this model, we take $\lambda_{\text{coord}} = 5$ and $\lambda_{\text{noobj}} = .5$

The first two parts of the above loss equation represent localization mean-squared

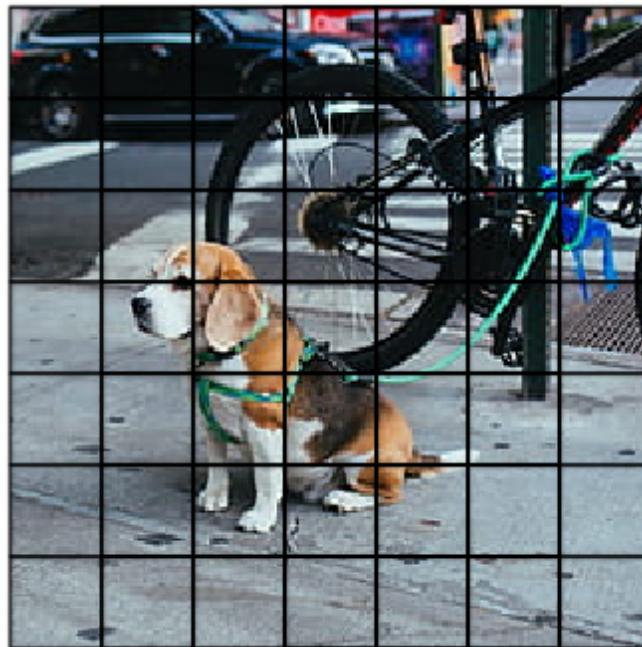
Start Your Coding Journey Now!

[Login](#)
[Register](#)

second term calculates the square root of the difference between height and width of the bounding box. In the second term, we take the square root of width and height because our loss function should be able to consider the deviation in terms of the size of the bounding box. For small bounding boxes, the little deviation should be more important as compared to large bounding boxes.

There are three terms in classification loss, the first term calculates the sum-squared error between the predicted confidence score that whether the object present or not and the ground truth for each bounding box in each cell. Similarly, The second term calculates the mean-squared sum of cells that do not contain any bounding box, and a regularization parameter is used to make this loss small. The third term calculates the sum-squared error of the classes belongs to these grid cells.

Detection: This architecture divides the image into a grid of $S \times S$ size. If the centre of the bounding box of the object is in that grid, then this grid is responsible for detecting that object. Each grid predicts bounding boxes with their confidence score. Each confidence score shows how accurate it is that the bounding box predicted contains an object and how precise it predicts the bounding box coordinates wrt. ground truth prediction.



YOLO Image (divided into $S \times S$ grid)

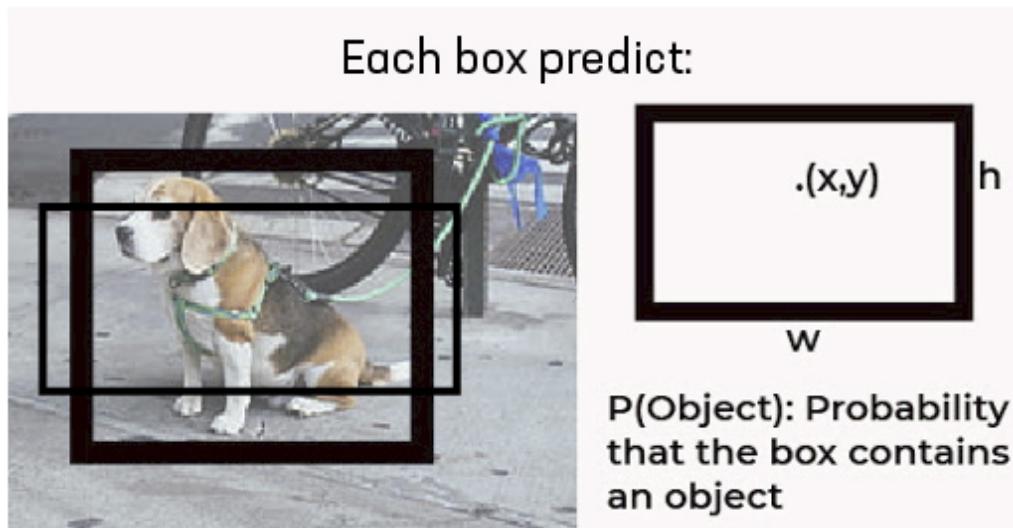


At test time we multiply the conditional class probabilities and the individual box confidence predictions. We define our confidence score as follows :

Start Your Coding Journey Now!

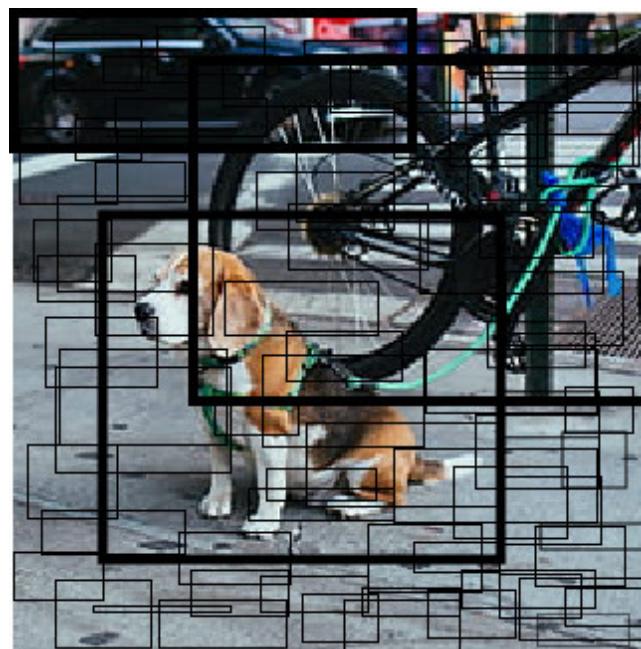
[Login](#)
[Register](#)

Note, the confidence score should be 0 when there is no object exists in the grid. If there is an object present in the image the confidence score should be equal to IoU between ground truth and predicted boxes. Each bounding box consists of 5 predictions: (x, y, w, h) and confidence score. The (x, y) coordinates represent the centre of the box relative to the bounds of the grid cell. The h, w coordinates represents height, width of bounding box relative to (x, y) . The confidence score represents the presence of an object in the bounding box.



YOLO single Grid Bounding box-Box

This results in combination of bounding boxes from each grid like this.



YOLO bounding box Combination

Start Your Coding Journey Now!

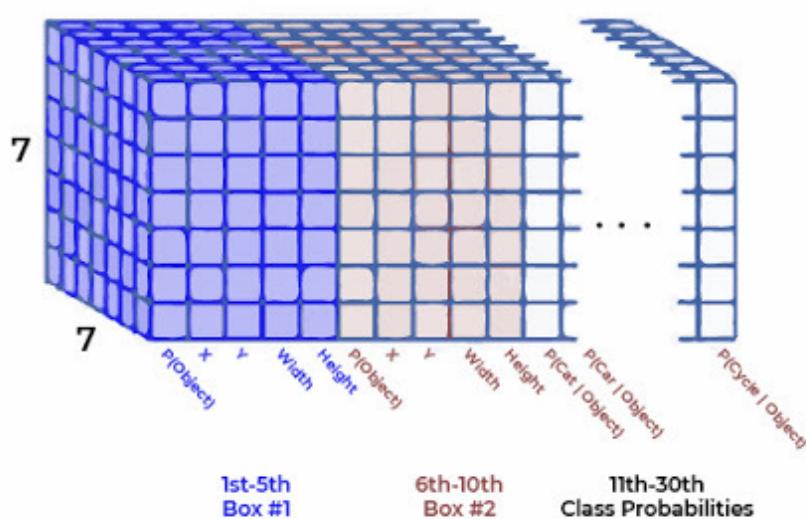
[Login](#)

[Register](#)



YOLO conditional probability map

This probability were conditional based on the presence of an object in grid cell. Regardless the number of boxes each grid cell predicts only one set of class probabilities. These prediction are encoded in the 3D tensor of size $S * S * (5*B + C)$. Now, we multiply the conditional class probabilities and the individual box confidence predictions,

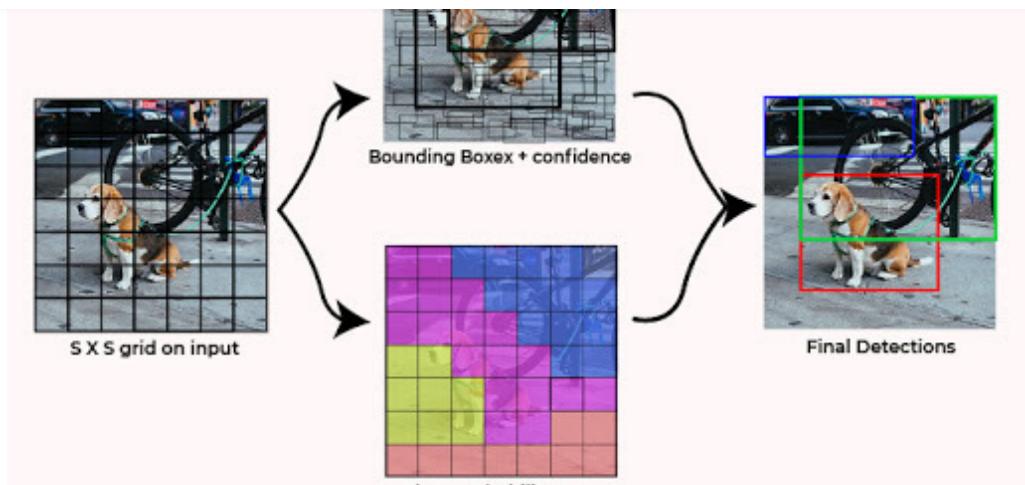


YOLO output feature map



$$(Class_i|Object) * P_r(Object) * IOU_{pred}^{truth} = P_r(Class_i) * IOU_{pred}^{truth}$$

Start Your Coding Journey Now!

[Login](#)
[Register](#)


YOLO test Result

which gives us class-specific confidence scores for each box. These scores encode both the probability of that class appearing in the box and how well the predicted box fits the object. Then after we apply non-maximal suppression for suppressing the non max outputs (when a number of boxes are predicted for the same object). And at last , our final predictions are generated.

YOLO is very fast at the test time because it uses only a single CNN architecture to predict results and class is defined in such a way that it treats classification as a regression problem.

Results: The simple YOLO has a mAP (mean average precision) of *63.4%* when trained on VOC in 2007 and 2012, the Fast YOLO which is almost *3x* faster in result generation has mAP of *52%*. This is lower than the best Fast R-CNN model achieved (*71% mAP*) and also the R-CNN achieved (*66% mAP*). However, it beats other real-time detectors such as (*DPMv5 33% mAP*) on accuracy.

Benefits of YOLO:

- Process frames at the rate of *45 fps* (larger network) to *150 fps* (smaller network) which is better than real-time.
- The network is able to generalize the image better.

Disadvantages of YOLO:

- Comparatively low recall and more localization error compared to Faster R_CNN. Struggles to detect close objects because each grid can propose only 2 bounding boxes.
- Struggles to detect small objects.

Start Your Coding Journey Now!

[Login](#)
[Register](#)

different alignments.

PYTHON PROGRAMMING FOUNDATION

- ✓ Self-Paced
- ✓ Lifetime Access
- ✓ Premium Lectures

[Enrol Now](#)

Like 1

[Previous](#)
[Next](#)

RECOMMENDED ARTICLES

Page : [1](#) [2](#) [3](#)

01 [YOLO v2 - Object Detection](#)
01, May 20

02 [Real-Time Edge Detection using OpenCV in Python | Canny edge detection method](#)
13, Dec 16

03 [Real time object color detection using OpenCV](#)
14, Oct 21

04 [Object Detection with Detection Transformer \(DETR\) by Facebook](#)
24, Jun 20

05 [Python | Real time weather detection using Tkinter](#)
26, Dec 18

06 [OpenCV | Real Time Road Lane Detection](#)
12, Dec 19

07 [Multiple Color Detection in Real-Time using Python-OpenCV](#)
24, Apr 20

08 [Difference between YOLO and SSD](#)
14, Jul 21



Start Your Coding Journey Now!

[Login](#)[Register](#)

Article Contributed By :

**pawangfg**

@pawangfg

Vote for difficulty

[Easy](#)[Normal](#)[Medium](#)[Hard](#)[Expert](#)

Improved By : surinderdawra388, kk9826225, jatingrg2399

Article Tags : Machine Learning, Python

Practice Tags : Machine Learning

[Improve Article](#)[Report Issue](#)

Writing code in comment? Please use [ide.geeksforgeeks.org](#), generate link and share the link here.

[Load Comments](#)**GeeksforGeeks**

A-143, 9th Floor, Sovereign Corporate Tower,
Sector-136, Noida, Uttar Pradesh - 201305

feedback@geeksforgeeks.org



Start Your Coding Journey Now!

[Login](#)
[Register](#)

Careers	Data Structures
In Media	SDE Cheat Sheet
Contact Us	Machine learning
Privacy Policy	CS Subjects
Copyright Policy	Video Tutorials
	Courses

News

Top News
 Technology
 Work & Career
 Business
 Finance
 Lifestyle
 Knowledge

Languages

Python
 Java
 CPP
 Golang
 C#
 SQL
 Kotlin

Web Development

Web Tutorials
 Django Tutorial
 HTML
 JavaScript
 Bootstrap
 ReactJS
 NodeJS

Contribute

Write an Article
 Improve an Article
 Pick Topics to Write
 Write Interview Experience
 Internships
 Video Internship

@geeksforgeeks , Some rights reserved





EngEd Community

Section's Engineering Education (EngEd) Program fosters a community of university students in Computer Science related fields of study to research and share topics that are relevant to engineers in the modern technology landscape. You can find more information and program guidelines in the GitHub repository. If you're currently enrolled in a Computer Science related field of study and are interested in participating in the program, please complete this form .

Introduction to YOLO Algorithm for Object Detection

April 15, 2021

YOLO is an algorithm that uses neural networks to provide real-time object detection. This algorithm is popular because of its speed and accuracy. It has been used in various applications to detect traffic signals, people, parking meters, and animals.

This article introduces readers to the YOLO algorithm for object detection and explains how it works. It also highlights some of its real-life applications.

Introduction to object detection

Object detection is a phenomenon in [computer vision](#) that involves the detection of various objects in digital images or videos. Some of the



This phenomenon seeks to answer two basic questions:

1. *What is the object?* This question seeks to identify the object in a specific image.
2. *Where is it?* This question seeks to establish the exact location of the object within the image.

Object detection consists of various approaches such as [fast R-CNN](#), [Retina-Net](#), and [Single-Shot MultiBox Detector \(SSD\)](#). Although these approaches have solved the challenges of data limitation and modeling in object detection, they are not able to detect objects in a single algorithm run. **YOLO algorithm** has gained popularity because of its superior performance over the aforementioned object detection techniques.

What is YOLO?

YOLO is an abbreviation for the term ‘You Only Look Once’. This is an algorithm that detects and recognizes various objects in a picture (in real-time). Object detection in YOLO is done as a regression problem and provides the class probabilities of the detected images.

YOLO algorithm employs convolutional neural networks (CNN) to detect objects in real-time. As the name suggests, the algorithm requires only a single forward propagation through a neural network to detect objects.

This means that prediction in the entire image is done in a single algorithm run. The CNN is used to predict various class probabilities



The YOLO algorithm consists of various variants. Some of the common ones include tiny YOLO and YOLOv3.

Why the YOLO algorithm is important

YOLO algorithm is important because of the following reasons:

Speed: This algorithm improves the speed of detection because it can predict objects in real-time.

High accuracy: YOLO is a predictive technique that provides accurate results with minimal background errors.

Learning capabilities: The algorithm has excellent learning capabilities that enable it to learn the representations of objects and apply them in object detection.

How the YOLO algorithm works

YOLO algorithm works using the following three techniques:

Residual blocks

Bounding box regression

Intersection Over Union (IOU)

Residual blocks

First, the image is divided into various grids. Each grid has a dimension of $S \times S$. The following image shows how an input image is divided into grids.

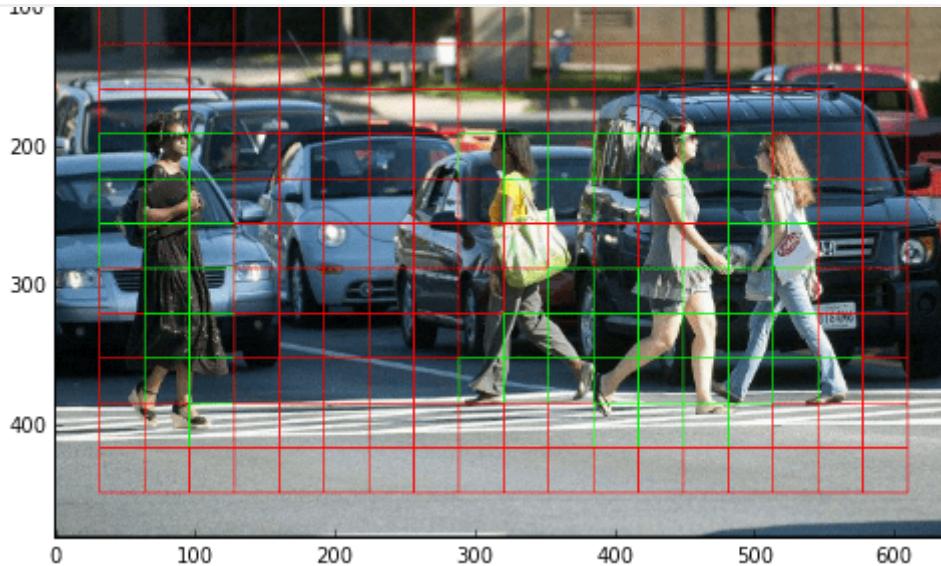


Image Source

In the image above, there are many grid cells of equal dimension. Every grid cell will detect objects that appear within them. For example, if an object center appears within a certain grid cell, then this cell will be responsible for detecting it.

Bounding box regression

A bounding box is an outline that highlights an object in an image.

Every bounding box in the image consists of the following attributes:

Width (bw)

Height (bh)

Class (for example, person, car, traffic light, etc.)- This is represented by the letter c.

bounding box has been represented by a yellow outline.

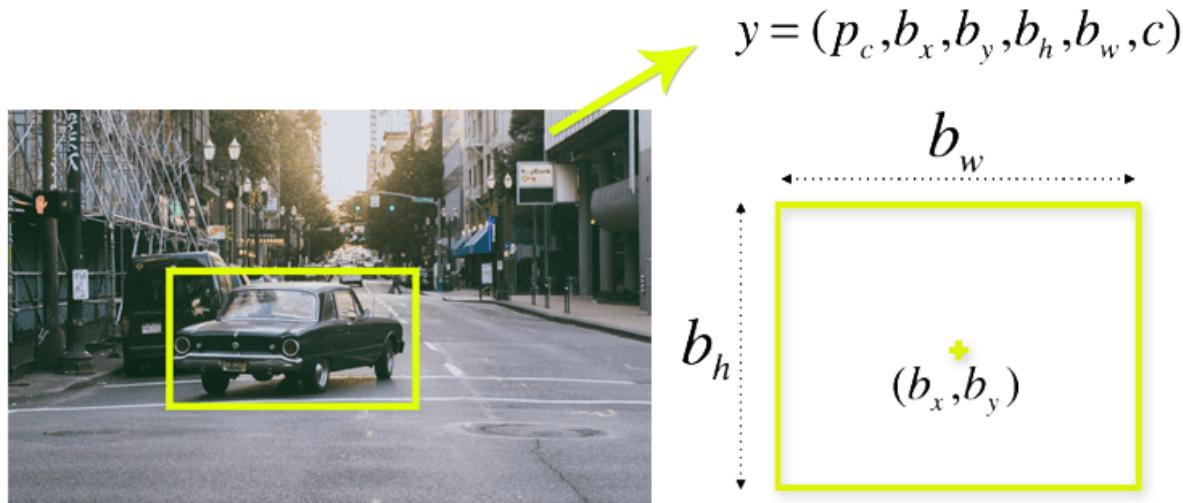


Image Source

YOLO uses a single bounding box regression to predict the height, width, center, and class of objects. In the image above, represents the probability of an object appearing in the bounding box.

Intersection over union (IOU)

Intersection over union (IOU) is a phenomenon in object detection that describes how boxes overlap. YOLO uses IOU to provide an output box that surrounds the objects perfectly.

Each grid cell is responsible for predicting the bounding boxes and their confidence scores. The IOU is equal to 1 if the predicted bounding box is the same as the real box. This mechanism eliminates bounding boxes that are not equal to the real box.

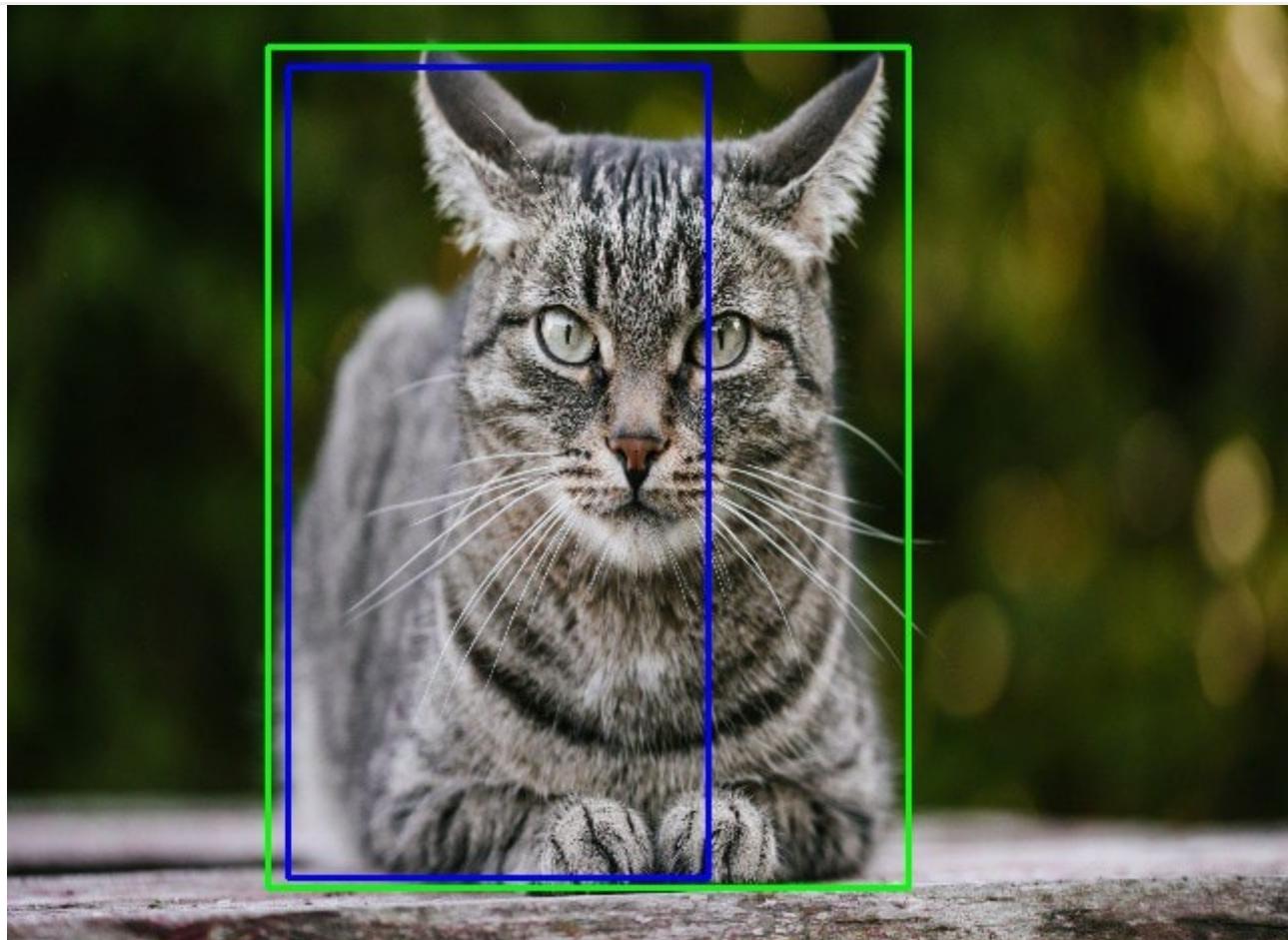


Image Source

In the image above, there are two bounding boxes, one in green and the other one in blue. The blue box is the predicted box while the green box is the real box. YOLO ensures that the two bounding boxes are equal.

Combination of the three techniques

The following image shows how the three techniques are applied to produce the final detection results.

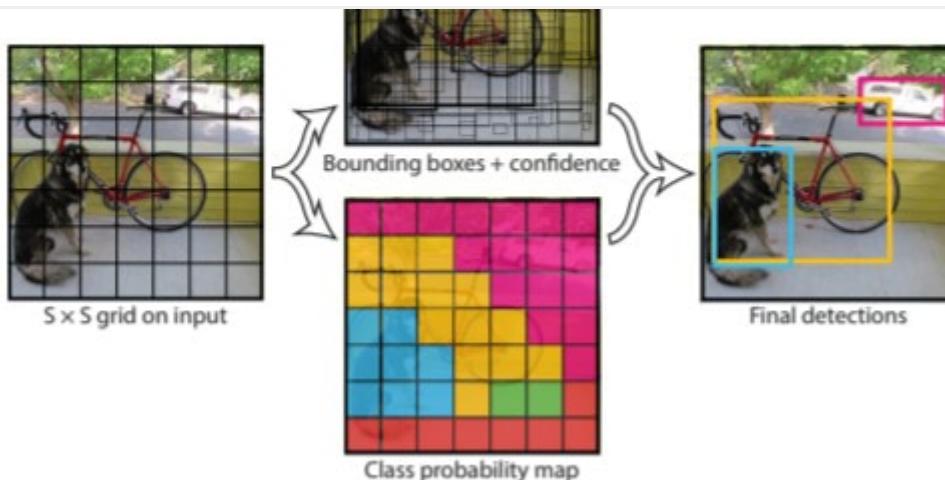


Image Source

First, the image is divided into grid cells. Each grid cell forecasts B bounding boxes and provides their confidence scores. The cells predict the class probabilities to establish the class of each object.

For example, we can notice at least three classes of objects: a car, a dog, and a bicycle. All the predictions are made simultaneously using a single convolutional neural network.

Intersection over union ensures that the predicted bounding boxes are equal to the real boxes of the objects. This phenomenon eliminates unnecessary bounding boxes that do not meet the characteristics of the objects (like height and width). The final detection will consist of unique bounding boxes that fit the objects perfectly.

For example, the car is surrounded by the pink bounding box while the bicycle is surrounded by the yellow bounding box. The dog has been highlighted using the blue bounding box.



YOLO algorithm can be applied in the following fields:

Autonomous driving: YOLO algorithm can be used in autonomous cars to detect objects around cars such as vehicles, people, and parking signals. Object detection in autonomous cars is done to avoid collision since no human driver is controlling the car.

Wildlife: This algorithm is used to detect various types of animals in forests. This type of detection is used by wildlife rangers and journalists to identify animals in videos (both recorded and real-time) and images. Some of the animals that can be detected include giraffes, elephants, and bears.

Security: YOLO can also be used in security systems to enforce security in an area. Let's assume that people have been restricted from passing through a certain area for security reasons. If someone passes through the restricted area, the YOLO algorithm will detect him/her, which will require the security personnel to take further action.

Conclusion

This article has provided an overview of the YOLO algorithm and how it is used in object detection. This technique provides improved detection results compared to other object detection techniques such as Fast R-CNN and Retina-Net.

To summarize:

We have gained an overview of object detection and the YOLO algorithm.



We have learned how the YOLO algorithm works. We have also gained an understanding of the main techniques used by YOLO to detect objects.

We have learned the real-life applications of YOLO.

Happy learning!

Peer Review Contributions by: [Onesmus Mbaabu](#)

Did you find this article helpful?



102

11 Comments

Sort By Best ▾

The EngEd community is subject to Section's [moderation policy](#).

Write your comment...

[LOGIN](#) [SIGNUP](#)



Dz 1 day ago

Sorry, this seems really inaccurate

[Reply](#) [Share](#)

0 0

Ivan Lo 24 days ago

well written

[Reply](#) [Share](#)

0 0

B Benya Adeyanju Jamiu 1 month ago

Very educative and inspiring...Good Job

[Reply](#) [Share](#)

0 0



How can I find detected objects in which grid? (grid size)

[Reply](#) [Share](#)

0 0

M

Muhammad Junaid 2 months ago

Very informative article you have written. Thanks much

[Reply](#) [Share](#)

0 0

M

Maroua 2 months ago

Thank you so much

[Reply](#) [Share](#)

0 0

B

Bruce 2 months ago

Really good blog post indeed. If your blog styles are like this, I'm definitely interested to follow you

[Reply](#) [Share](#)

0 0

D

Daryl 3 months ago

Simple and Concise, great blog

[Reply](#) [Share](#)

0 0

R

Raj 3 months ago

Great article...

[Reply](#) [Share](#)

0 0

Y

Yang 4 months ago

Thank you so much! Your article is really easy to understand what is YOLO, and I completely get the perfect overview! Much obliged!

[Reply](#) [Share](#)

0 0

p

palani 4 months ago

Thanks for updates, This blog is simple and easy to understand.

[Reply](#) [Share](#)

0 0



Grace Karimi

Grace is an undergraduate student at Maseno University. Her interests are data science, emerging technologies, and mobile development. During her free time, she prefers dancing or swimming.

[View author's full profile](#) →

Join our Slack community

[Add to Slack](#)

Company

About

Careers WE'RE HIRING

Legals

Resources

Blog

Edge Content Library

Solution Briefs



Support

Docs

Community Slack

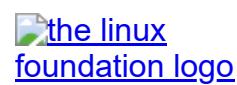
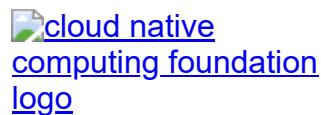
Help & Support

Release Notes

Platform Status

Contact Us

Section supports many open source projects including:



© 2022 Section

[Privacy Policy](#) [Terms of Service](#)