

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 and bounding boxes simultaneously.

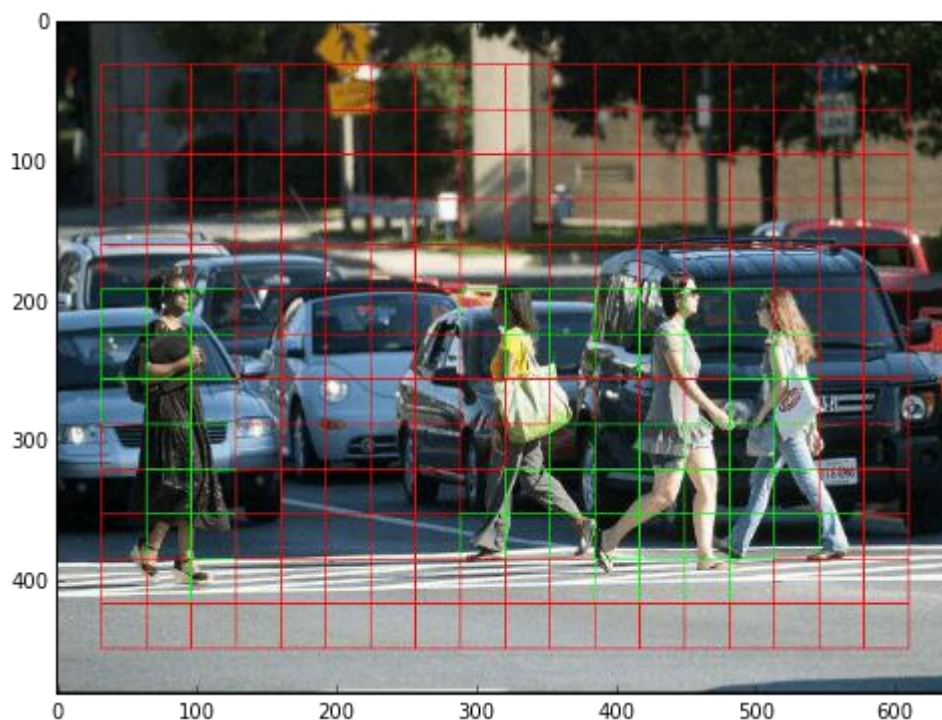
The YOLO algorithm consists of various variants. Some of the common ones include tiny YOLO and YOLOv3.

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.



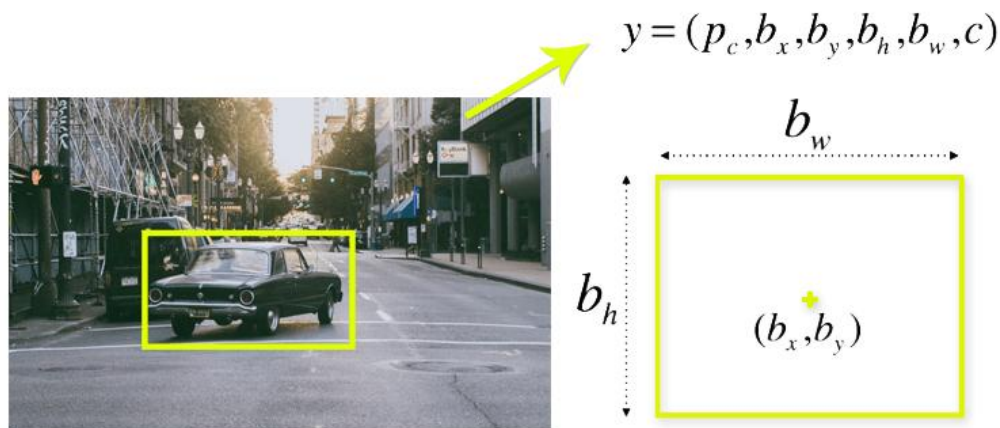
In the image above, there are many grid cells of equal dimension. Every grid cell will detect objects that appear within them. For example, if an object center appears within a certain grid cell, then this cell will be responsible for detecting it.

-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 (b_w), Height (b_h), Class (for example, person, car, traffic light, etc.)- This is represented by the letter c ., Bounding box center (b_x, b_y)

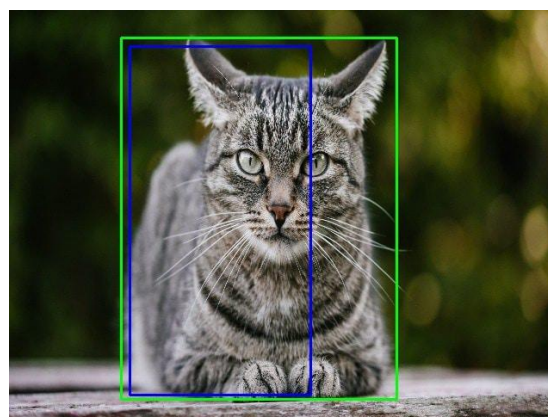
The following image shows an example of a bounding box. The bounding box has been represented by a yellow outline.



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):

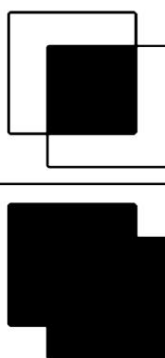
Intersection over union (IOU) is a phenomenon in object detection that describes how boxes overlap. YOLO uses IOU to provide an output box that surrounds the objects perfectly. Each grid cell is responsible for predicting the bounding boxes and their confidence scores. The IOU is equal to 1 if the predicted bounding box is the same as the real box. This mechanism eliminates bounding boxes that are not equal to the real box.



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.

The You Only Look Once (YOLO) algorithm, unsurprisingly, looks at each frame only once. This means that it forward propagates the entire frame in a single pass to the classifier network. YOLOv3 utilizes a single fully connected classifier network to compute confidence values and predict anchor boxes. It employs non-maximal suppression to classify and recognize objects.

The YOLO algorithm divides each frame into $M \times M$ cells and N bounding boxes are predicted for each cell ($M \times M \times N$). The bounding boxes are predicted by the network. The initial version of YOLO predicted objects by only using bounding boxes, but this technique detected and identified only one object per frame. YOLOv3 makes use of Anchor Boxes as well for detecting multiple objects in every frame. An anchor box is a hand labeled bounding box assigned to each object. Anchor boxes are also referred to as ground truth boxes. For each anchor box, the Intersection Over Union (IOU) for every cell is calculated. Bounding boxes and Anchor Boxes are the two parameters for the calculation of IOU. The formula for IOU is given as area of overlap over area of union. Only objects with IOU over 0.5 (50%) are detected.

$$IoU = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$


I have also done a project using YOLO algorithm for object detection. Following is a real time output of obtained from the same:



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.

YOLO in python:

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
```

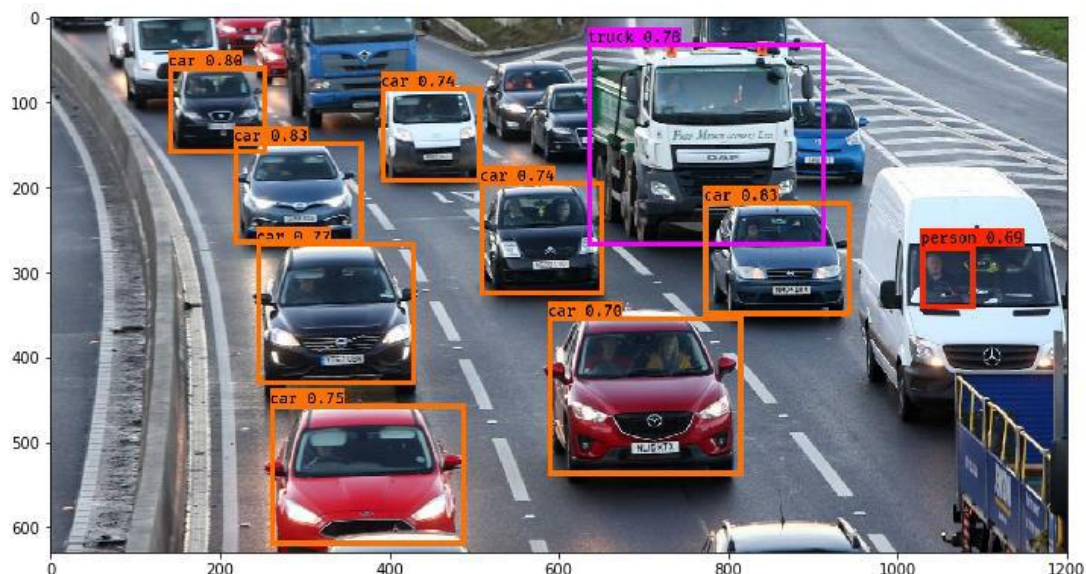
Next, we will read an image and make predictions using the *predict* function:

```
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)
```



SUMMARY

- 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.