Prepared by:
Fida Hussain Abbas Rao ME-18064
Hammas Amir Khan ME-18063

You Only Look Once:

Unified, Real-Time Object Detection

By:

Joseph Redmon, Santosh Divvala , Ross Girshick, Ali Farhadi

A Review report submitted to the NED University of Engineering & Technology in partial fulfilment of the course requirements for the course of EE-373

Supervised by : Sir Adeel Arif

Department of Mechanical Engineering

NED University of Engineering and Technology

Karachi, Sindh.

May 2021

Abstract

***YOLO**, a real-time object detection algorithm, was presented in this paper. YOLO was well received because it ran in real-time without compromising on accuracy. The algorithm 'only looks once' at the image since it requires only a single forward propagation to detect objects. YOLO algorithm utilizes a Convolutional Neural Network to use features from the image to simultaneously predict class probabilities and multiple bounding boxes.*

Classical deep learning approaches for object detection such as R-CNN and DPM were usually very slow to be used for real-time tasks. In this report, we will demonstrate the usage of the latest YOLOv4 algorithm for vehicle detection systems which is one of the most essential techniques implemented in self-driving cars. Our pre-trained weights will predict vehicles, pedestrians, and traffic lights from the given image (or video).

List of Figures

List of Tables

Table of Content

Chapter 1: Introduction

Object detection has been attracting growing attention in the last few years due to its broad range of applications and recent technological advancements. Humans were always blessed with their ability to recognize, locate and sense the interaction between objects in real-time. However, to eliminate human error due to carelessness or labor cost for hiring, it is essential to automate the entire object detection process. On account of the recent progress of deep learning, many algorithms have been designed to allow the computer to emulate that performance accurately and swiftly. One of the algorithms is **YOLO, '***you only look once',* which is one of the most powerful real-time object detector algorithms[3]. It is named that way because unlike previous object detector algorithms, like DPM which uses a sliding window approach, or R-CNN and its improved version Faster R-CNN which uses region proposal methods[2][4][5], YOLO only needs the image (or video) to pass only once through the network to make predictions.

The significance of YOLO is enhanced by the fact that it is extremely fast when compared to other detection systems. It can process 45 frames per second. It only has 25 milliseconds of latency and a much higher mean average precision. Unlike other detection methods that use classifiers, YOLO first generates potential bounding boxes and then runs classifiers on them. We normalize the bounding boxes and image dimensions, so they lie between 0 and 1. It is characterized by a single regression problem where high quality and fine-grained image of 448x448 resolution is directly reformulated into bounding boxes and class probabilities.

Although, YOLO outperforms other detection systems in generalizability and speed, but it produces more localization errors than Fast R-CNN[1]. This is seen to have improved by combining YOLO with Fast R-CNN.

1.1 Stages of YOLO Object Detection

The steps of the YOLO detection model are shown in figure 1. It uses a convolutional neural network model that is crucial for feature extraction[8]. Firstly, the model divides the image into an SxS grid. If the midpoint of an object falls in a particular grid, then that grid is assigned to detect that object. Each grid is responsible for predicting B bounding boxes, their confidence scores, and C conditional probabilities for classes.

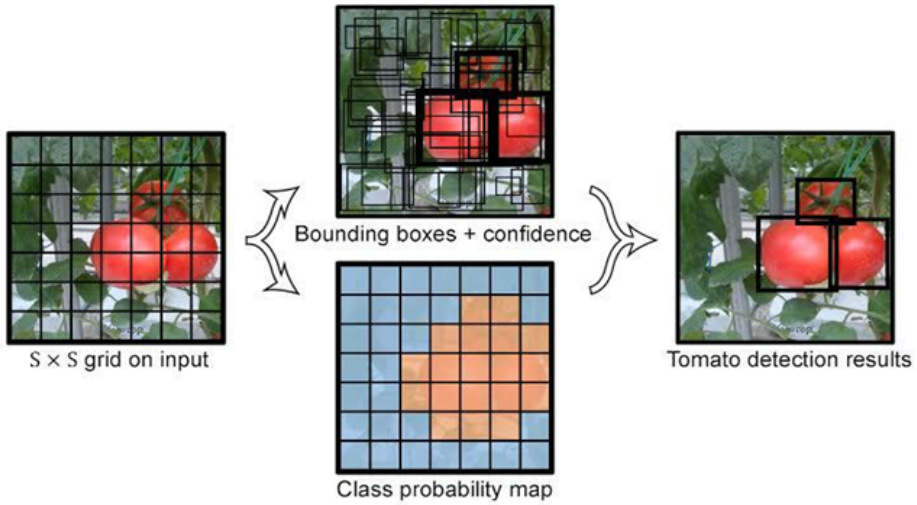


Figure 1: Stages of YOLO detection model.

1.2 Understanding Convolution

It is essential to understand how convolution works before we dive further into object detection. Convolution is a mathematical operation that takes the image matrix and an FxF kernel-sized filter and convolves them both together[8]. We will try to understand this from figure 2. Consider a 5x5 image with pixel values of either 0 or 1 and a filter with a kernel size

of 3x3 from figure 2. Firstly, we will multiply the top left 3x3 part of the image with the filter and the values returned are summed. Then the kernel will slide 1 pixel to the right. NOTE: distance by which the kernel slides is called stride. Similarly, then the next 3x3 part of the image matrix is multiplied with the kernel and so on until we reach the bottom left. The shape of the matrix obtained after filtering would be :

$$o = [(I + F) / s] + 1$$

where:

- I = shape of the input image
- F = kernel size
- s = strides

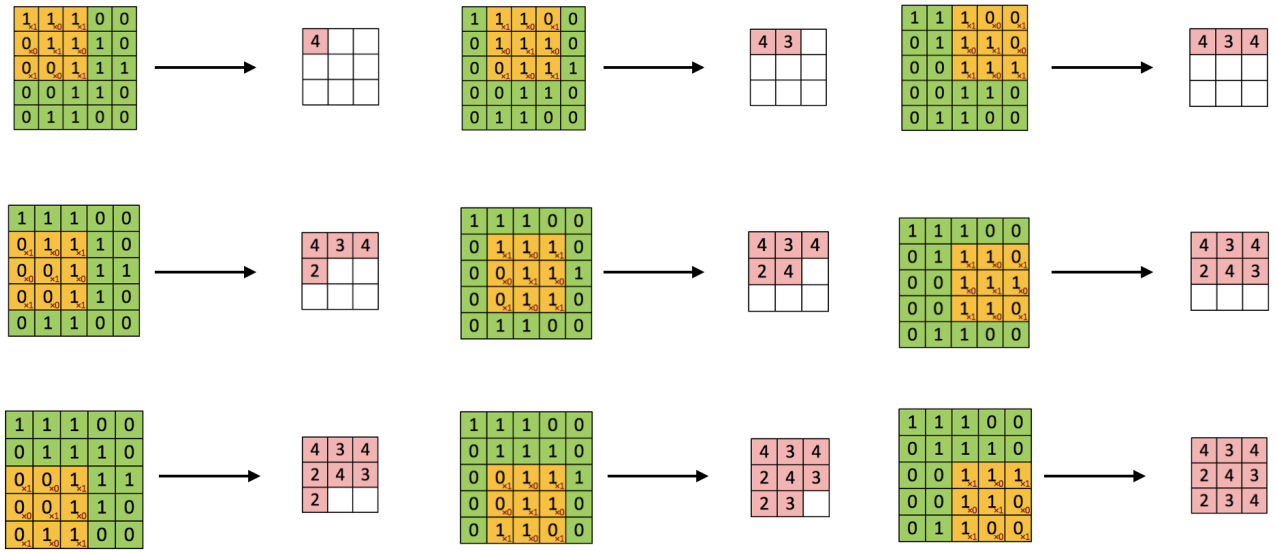


Figure 2: Computing the output of convolutions. 5 x 5 image matrix, 3 x 3 kernel size, stride =1.

1.3 Applications of Object Detection

In recent years, there has been a significant increase in the number of applications in which object detection is helpful. It is critical in different applications, such as surveillance, cancer detection, vehicle detection, face recognition, and underwater object detection[9]. Numerous techniques have been used to detect the object accurately and efficiently for different applications.

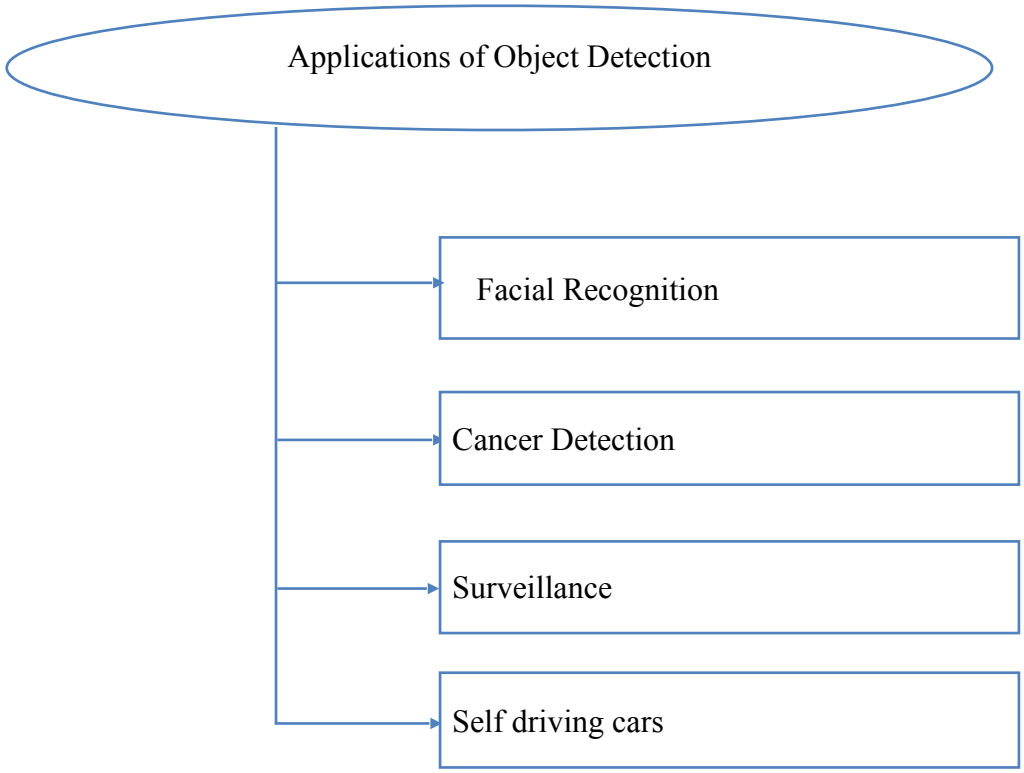


Figure 3: Applications of Object Detection

Chapter 2: Background Subtopics

** we will try to understand why the authors of this paper felt the need to design this algorithm*

Object detection is one of the classical problems in computer vision, where we basically recognize the object and its position inside the given image[3]. To understand the concept of object detection completely, it is essential to comprehend image classification and image localization in the first place.

- *Image Classification* aims at recognizing the object and assigning a label to an image. It will basically answer 'What is in the picture.' Every image has only one label set for it[7].
- *Image Localization* refers to finding the position of an object in the image. Hence the question changes to 'What is in the image and where is it'[6].

Image Classification | Image Localization | Object Detection

Figure 4: Difference between classification, localization and Object Detection

However, to be able to use the above concepts for real-world tasks such as self-driving cars, we need to detect multiple objects and their respective positions in an image. Object detection allows us just to do that.

2.1 Disadvantages of Object Detection algorithms prior to YOLO

Over the years, many popular algorithms have been designed for object detection. Classical algorithms such as R-CNN and Fast R-CNN were based on classification. We basically select the target areas in the object and then use Convolutional Neural Network to classify those regions[2][4]. It is a relatively simple and straightforward technique and has proven to be highly accurate. However, there are some downsides to it as well, leading to the designing of the YOLO algorithm. Those downsides are:

1. Fast R-CNN mistakes background for an object because it can't see the larger context [1][2].
2. Algorithms such as R-CNN and DPM fail to generalize to unseen data. Generalization is a crucial feature for deep learning-based detection algorithms that are supposed to perform well across different datasets. While training the model, we tend to obtain training and testing data from the same distribution; however, in a real-world case, the test data can diverge from what the model has been trained on[4][5].
3. It is a comparatively slower method; hence, we can't use it for real-time object detection applications such as in self-driving cars, automated CCTV surveillance, etc. [2][4][5].

For the above reasons, the authors of this paper have come up with the YOLO algorithm. It is based on the regression technique, where we pass the image to the algorithm only once and it predicts the classes and bounding boxes. YOLO is an extremely fast algorithm, and our base model processes images at 45 frames per second, making it ideal for real-time applications. Moreover, YOLO also generalizes well to unseen data making it suitable for applications that rely on fast, robust object detection. Lastly, it could be seen from Table 1 below that YOLO is making three times fewer false positive detections than Fast R-CNN [1].

Table 1: Error Analysis of Fast R-CNN and YOLO: the table shows errors in different areas and their respective correctness percentages.

FAST R-CNN		**YOLO**	
Error	*Percentage %*	*Error*	*Percentage %*
Background	13.6	Background	4.75
Simulation	4.3	Simulation	6.75
Localization	8.6	Localization	19.0
Other	1.9	Other	4
Correctness (No Error)	71.6	Correctness (No Error)	65.5

Chapter 3: Methodology

We must first collect training images in order to get the object detector off the ground. To achieve a highly accurate model, we must ensure that images for each class are evenly distributed. The following step would be to annotate the images as the model needs to be fed with labeled training data. Around each object, we draw a bounding box we want the model to see and label each class with the object class. Furthermore, pass the input images to the pre-trained CNN, and it will output the vector of bounding boxes and class predictions[8].

3.1 Network Architecture

We will implement a 24 layer convolutional network and max-pooling followed by two fully connected layers in the end[13]. The convolutional layers will extract the features from the image while the following fully connected layers predict the output probabilities and bounding box coordinates. We'll use Linear activation in the last layer, while we'll use Leaky RELU activation in the remaining layers.We employ dropout and significant data augmentation to avoid overfitting. Co-adaptation between layers is prevented by a dropout layer with rate =0.5 after the first connected layer [14]. We will use the sum-squared error loss function, and the parameters of the CNN will be determined by minimizing the loss function.

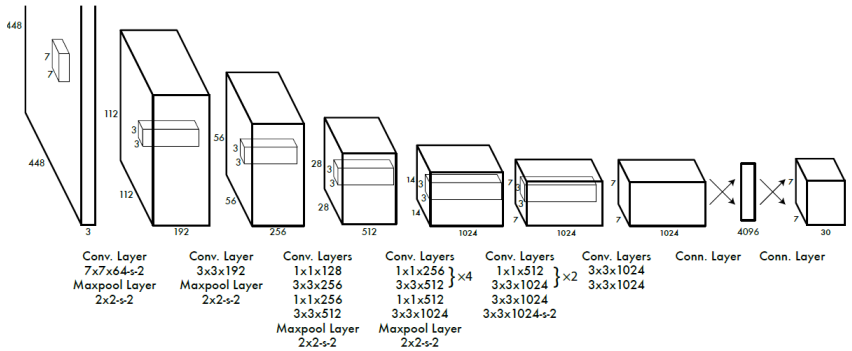


Figure 5: YOLO architecture

3.2 Working of YOLO algorithm:

The final output of the CNN will be a vector of size SxSx(c+5). Understanding how YOLO encodes the output is very important. First, the input image is divided into grids of SxS dimensions.

Figure 6: SxS grid cells. How an image is divided into grid cells?

Then, a grid cell is responsible for detecting an object that appears within them. For instance, if an object's center appears within a particular grid cell, then that cell is responsible for detecting it. Every grid cell will predict "B" bounding boxes and "c" class probabilities. The bounding box is a rectangular outline that highlights an object in an image. Every bounding box in an image will have the following components:

1. Probability of object in that bounding box (pc)
2. Width of the box (bw)
3. Height of the box (bh)
4. Center of the bounding box (bx, by)
5. The class of the object such as car, traffic lights, etc. (c)

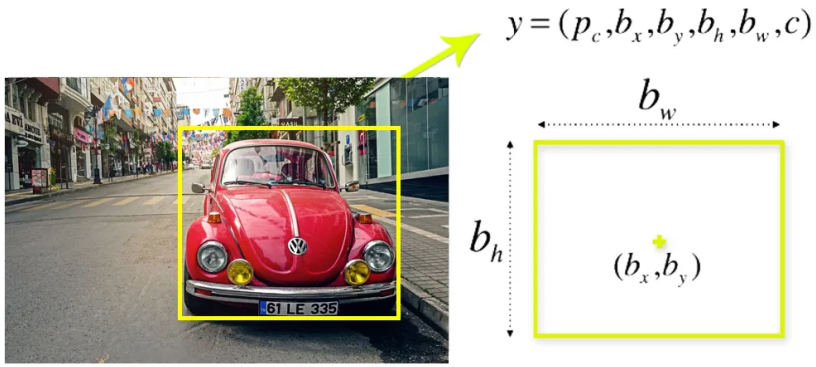


Figure 7: Example of a bounding box. The bounding box has been represented by a yellow box.

The majority of these grid cells will be empty. As a result, we predict the value of pc to remove boxes with low object probability. We will only keep those boxes that have a likelihood of more than a certain threshold. Next, we will take element wise product of the remaining bounding box's "pc" with their respective class scores. Lastly, we will remove the overlapping boxes by a process called non-max suppression.

3.2.1 Non-Max Suppression:

As the object in the image can be of different shapes and sizes, the object detection algorithm creates multiple bounding boxes to capture each of these. However, we must only have a single bounding box for each object. Non-Max suppression is a technique that allows us to select the best bounding box by suppressing the less likely bounding boxes and keeping the best one only.

3.2.2 How does Non-Max Suppression work?

NMS takes two things into account in order to select the best bounding boxes

1. objectiveness score (element-wise multiplication of pc with class probabilities)
2. Intersection over union of the bounding boxes (detail in the following paragraph)

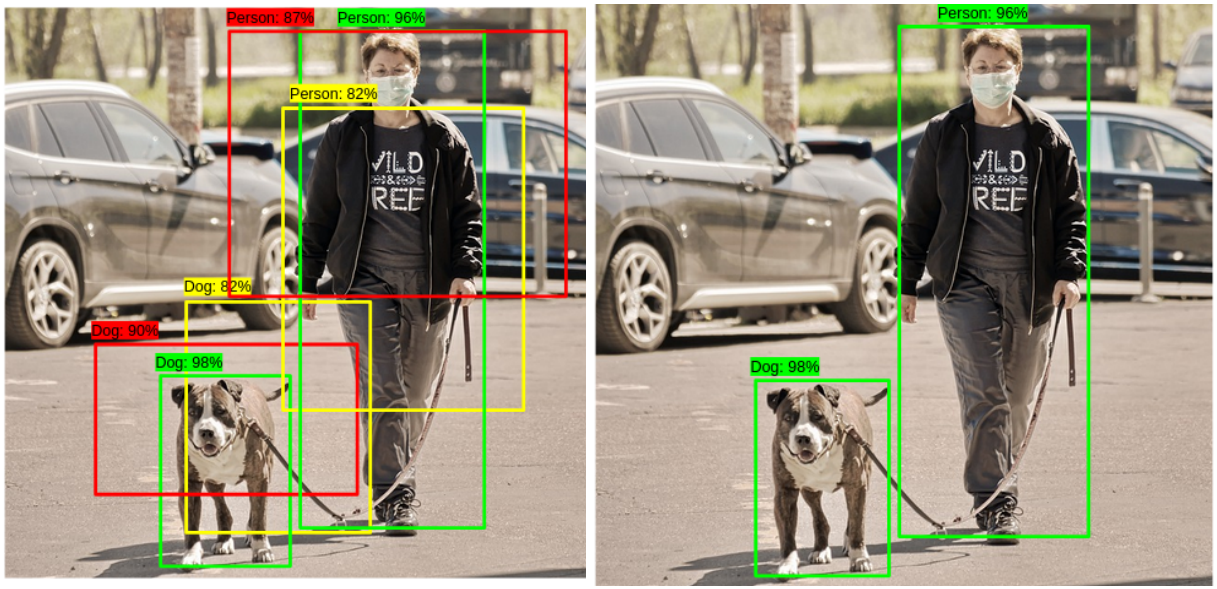


Figure 8: Removal of unwanted bounding boxes

How can we get rid of the unwanted bounding boxes?

Firstly, non-max suppression will select the bounding box with the highest objectiveness score and then compute the IoU of all the other bounding boxes with the chosen bounding box. It will then remove the boxes with high IoU. The exact process will continue iteratively for the remaining boxes until there is no more reduction of boxes. This type of filtering makes sure that only one bounding box is returned per object detected.

3.2.3 Intersection over Union (IoU):

Intersection over Union (IoU) is a concept in object detection that illustrates how boxes overlap. Given two boxes as input, it calculates the ratio of the intersection over union of the two boxes.

For example, for two boxes A and B, IoU is calculated as :

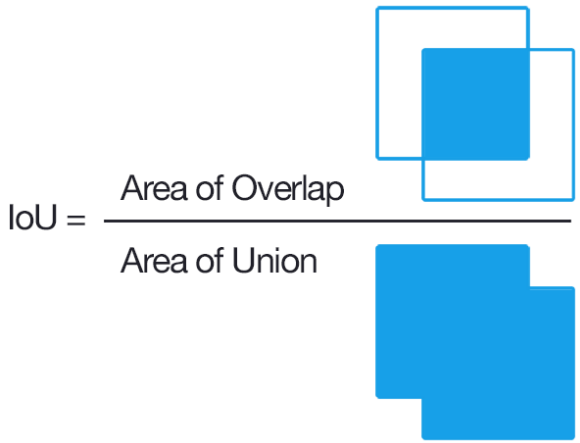


Figure 9: Intersection over Union

3.3 Summary of YOLO algorithm

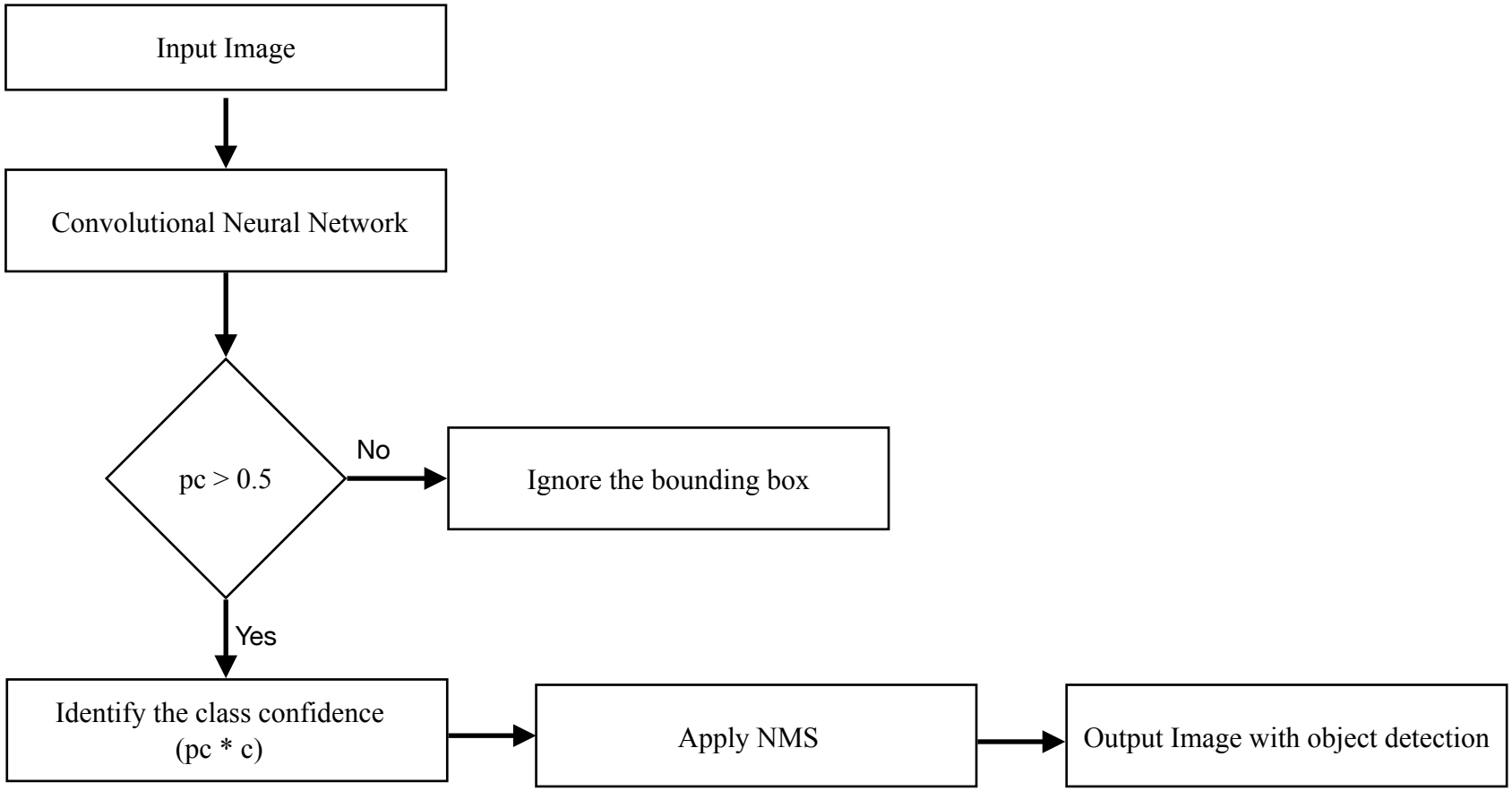


Figure 10: Summary of YOLO algorithm

Chapter 4: Results

Firstly, we will compare YOLO with other real-time detection systems. We investigate the errors caused by YOLO and Fast R-CNN, one of the most performant versions of R-CNN, using VOC 2007 better to understand the difference between YOLO and R-CNN variants[2]. Based on the varied error profiles, we show that YOLO may be utilized to rescore Fast R-CNN detections and reduce the errors from background false positives, resulting in a considerable performance improvement[11].

YOLO shows promising results on individual basis but it performs more accurately when combined with other detection methods such as R-CNN. For example, the combination of Fast R-CNN and YOLO produces an improvement of 2.5% when compared to their performances individually.

Table 2: MAP and per-class average precision of YOLO compared with other algorithms

VOC 2012 Test	mAP	Aero	Bike	Bird	Boat	Bottle	Bus	Car	Cat
FAST R-CNN+YOLO	70.7	83.4	78.5	73.5	55.8	43.4	79.1	73.1	89.4
FASTER R-CNN	70.4	84.9	79.8	74.3	53.9	49.8	77.5	75.9	88.5
FAST R-CNN	68.4	82.3	78.4	70.8	52.3	38.7	77.8	71.6	89.3
YOLO	57.9	77.0	67.2	57.7	38.3	22.7	68.3	55.9	81.4
R-CNN	49.6	68.1	63.8	46.1	29.4	27.9	56.6	57.0	65.9

Table 2 above shows the results on VOC 2012 test set, and it shows that YOLO scores 57.9% on mAP. Other detection methods are also listed for a better comparison. Analysis of the table highlights the fact that YOLO does not perform well in terms of small objects, like boat and bottle, but shows applaudable results in other categories.

Figure 11 shows the precision recall curve obtained on artwork datasets for various detectors[10]. It shows the purple curve of precision for YOLO which is a clear evidence of its high position on the leaderboard. Higher precision-recall means YOLO generalizes better to unseen domains when compared to other detectors making it ideal for applications that rely on quick and reliable object detection[10]. Therefore, YOLO is a quick and accurate object detection method suitable for computer vision applications. Moreover, it maintains real-time performance including the time to fetch images from the camera and display the detection.

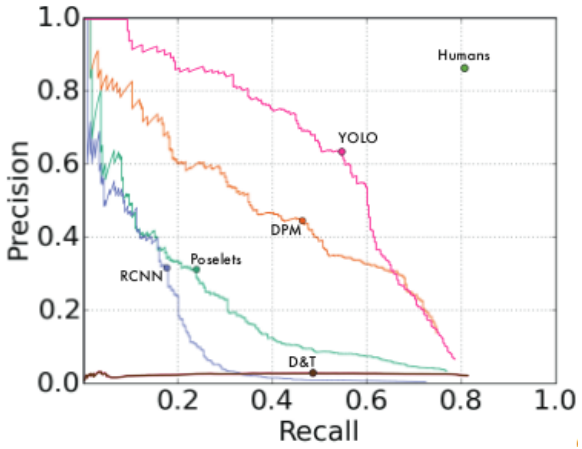


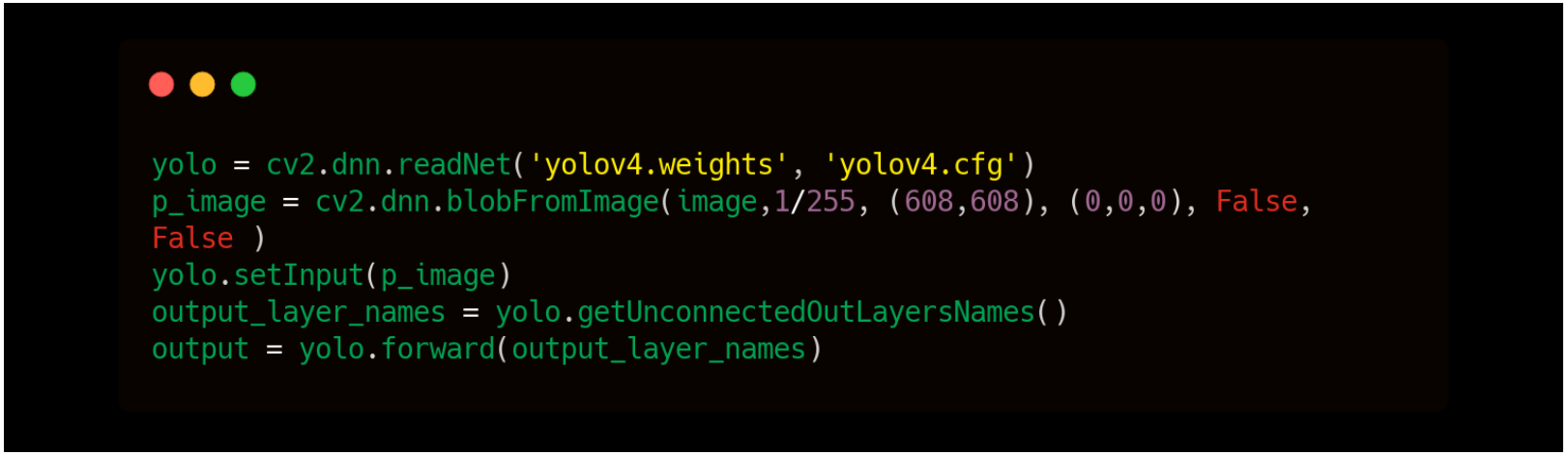
Figure 11: Precision Recall curve

Chapter 5: Simulation

There are several implementations of the YOLO algorithm, and perhaps the quickest and simplest among them is OpenCV. Before diving into the code, we'll install and import all the necessary libraries. After that, we will download the weights of the pre-trained network('yolov4.weights') and the network configuration('yolov4.cfg'). Once we have set up the environment, we will begin coding.

5.1 Step1 - Getting bounding boxes

First, we'll use the function "cv2.dnn.ReadNet()" to load the model. We can either use this model for object detection in an image or in real-time object detection. A neural network's input image must be in a certain format known as a blob. Therefore, we'll create a four-dimensional blob from the image. This process will normalize the image pixel values using a scale factor of 1/255 to a range of 0 and 1 and resize the image to (608,608) without cropping. This output blob is then fed to the neural network using '.set_input' function and we'll obtain a list of predicted bounding boxes after running a forward pass as the output.



```
yolo = cv2.dnn.readNet('yolov4.weights', 'yolov4.cfg')
p_image = cv2.dnn.blobFromImage(image,1/255, (608,608), (0,0,0), False,
False )
yolo.setInput(p_image)
output_layer_names = yolo.getUnconnectedOutLayersNames()
output = yolo.forward(output_layer_names)
```

Figure 12: Simulation Step1 - Getting bounding boxes

5.2 Step2 - Filtering bounding boxes

The output bounding boxes will now go through the filtering process to removes the ones with the low confidence score. Hence, we need to iterate over the neural network's output, and then we'll iterate over each detection using a nested loop. We will discard any object that has confidence less than the specified threshold(i.e., 0.7). The boxes with confidence equal to or greater than the threshold are then subjected to NMS(Non-max Suppression). We will take advantage of OpenCV's builtin implementation of NMS("cv2.dnn.NMSboxes"). It will compare all the bounding boxes with the one which has the highest confidence score and discard the ones that have IOU greater than the specified threshold(i.e., 0.4). An IoU of 1 means that the two bounding boxes are completely similar, while an IoU of 0 means that they're not even intersecting.

```
for i in output:
    for detection in i:
        score = detection[5:]
        class_id = np.argmax(score)
        confidence = score[class_id]

        if confidence > 0.7:
            centre_x = int(detection[0] * width)
            centre_y = int(detection[1] * hight)
            w = int(detection[2] * width)
            h = int(detection[3] * hight)
            x = int(centre_x - w/2)
            y = int(centre_y - h/2)

indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.7, 0.4)
```

Figure 13: Simulation Step2 - Filtering bounding boxes

5.3 Step3 - Drawing bounding boxes

Finally, we'll draw the boxes and put text on the image. We begin by looping over the filtered detection from NMS. After that, we'll extract the location of the object:

- Coordinates of the center point (x, y).
- Width of the object(w).
- Height of the object(h).

```
def draw_boxes():
  for i in indexes.flatten():
      x,y,w,h = boxes[i]
      label = str(classes[class_ids[i]])
      prob = str(round(confidences[i], 2))
      cv2.rectangle(image, (x,y), (x+w, y+h), c, 5)
      cv2.putText(image, label + ' ' + prob, (x,y-5), font,2,(0, 0, 0),3)
```

Figure 14: Simulation Step3 - Drawing bounding boxes

Lastly, we'll draw the bounding boxes around the object with their class labels and confidence score image using the cv2.rectangle and cv2.putText methods.

After running YOLO on Figure 4, we can see the model was successfully able to detect all the vehicles on the road with confidence of greater than 0.8.

Conclusion

To sum up, this research paper contains many advantages and limitations regarding one of the most popular object detection methods, YOLO. The significance and stages of detection had been mentioned in detail. Its limitations were identified which has to be addressed in the future such as eliminating background errors, better detection of smaller objects, and a higher speed. This method can be brought into use more often in vast applications such as deep learning, self-driving cars, and real-time objected detection. With certain improvements, it can become the most effective and widely used method.

References

[1] D. Hoiem, Y. Chodpathumwan, and Q. Dai. Diagnosing error in object detectors. In Computer Vision–ECCV 2012, pages 340–353. Springer, 2012.

[2] R. B. Girshick. Fast R-CNN. CoRR, abs/1504.08083, 2015.

[3] C. P. Papageorgiou, M. Oren, and T. Poggio. A general framework for object detection. In Computer vision, 1998. sixth international conference on, pages 555–562. IEEE, 1998.

[4] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich fea- ture hierarchies for accurate object detection and semantic segmentation. In Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on, pages 580–587.IEEE, 2014.

[5] P.F.Felzenszwalb,R.B.Girshick,D.McAllester,andD.Ra- manan. Object detection with discriminatively trained part based models. IEEE Transactions on Pattern Analysis and Machine Intelligence, 32(9):1627–1645, 2010.

[6] M. B. Blaschko and C. H. Lampert. Learning to localize objects with structured output regression. In Computer Vision-ECCV 2008, pages 2–15. Springer, 2008.

[7] M Manoj Kirshma, M Neelima, Harshali Mane and Venu Gopala Rao Matcha. Image classification using Deep learning. International Journal of Engineering and Technology 7(2.7):614, 2018

[8] Vincent Dumoulin, Francesco Visin. A guide to convolution arithmetic for deep learning. Université de Montréal, 2016.

[9] Abdul Vahab, Maruti S Naik, Prasanna G Raikar, Prasad S R. Applications of Object Detection System. Anjuman Institute of Technology and Management, Bhatkal, 2019.

[10] H. Cai, Q. Wu, T. Corradi, and P. Hall. The cross- depiction problem: Computer vision algorithms for recog- nising objects in artwork and in photographs. arXiv preprint arXiv:1505.00110, 2015.

[11] D. Hoiem, Y. Chodpathumwan, and Q. Dai. Diagnosing error in object detectors. In Computer Vision–ECCV 2012, pages 340–353. Springer, 2012.

[12] S. Ren, K. He, R. B. Girshick, X. Zhang, and J. Sun. Object detection networks on convolutional feature maps. CoRR, abs/1504.06066, 2015.

[13] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. CoRR, abs/1409.4842, 2014.

[14] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov. Improving neural networks by pre- venting co-adaptation of feature detectors. arXiv preprint arXiv:1207.0580, 2012.