

NUMBER PLATE RECOGNITION FOR NON-HELMET RIDERS

Submitted in partial fulfillment of the requirements for
the award of
Bachelor of Engineering degree in Computer Science and Engineering

by

N.V.V.S.K.SUBRAHMANYAM (Reg. No: 37110498)



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SCHOOL OF COMPUTING**

SATHYABAMA

**INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)**

**Accredited with Grade "A" by NAAC
JEPPIAAR NAGAR, RAJIV GANDHI SALAI,
CHENNAI – 600 119**

MARCH - 2021



SATHYABAMA



INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited with “A” grade by NAAC
Jeppiaar Nagar, Rajiv Gandhi Salai, Chennai – 600 119
www.sathyabama.ac.in

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this project report is the bonafide work **N.V.V.S.K.SUBRAHMANYAM (37110498)** who carried out the project entitled “**NUMBER PLATE RECOGNITION FOR NON-HELMET RIDERS**” under my supervision from **November 2020** to **March 2021**.

Internal Guide

Dr. A. VIJI AMUTHA MARY, M.Tech., Ph.D,

Head of the Department

Dr. S. VIGNESHWARI, M.E., Ph.D.,
Dr. L. Lakshmanan, M.E., Ph.D.,

Submitted for Viva voce Examination held on _____

Internal Examiner

External Examiner

DECLARATION

I N.V.V.S.K.SUBRAHMANYAM (37110498) hereby declare that the Project Report entitled "NUMBER PLATE RECOGNITION FOR NON-HELMET RIDERS" is done by me under the guidance of Dr. A. VIJI AMUTHA MARY, M.Tech., Ph.D, Department of Computer Science and Engineering at Sathyabama Institute of Science and Technology is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering degree in Computer Science and Engineering.

DATE:

PLACE: CHENNAI

SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to **Board of Management of SATHYABAMA** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T. Sasikala, M.E., Ph.D., Dean**, School of Computing, **Dr. S. Vigneswari, M.E., Ph.D., and Dr. L. Lakshmanan, M.E., Ph.D.**, Heads of the Department of Computer Science and Engineering for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Dr. A. VIJI AMUTHA MARY, M.Tech., Ph.D.**, for her valuable guidance, suggestions and constant encouragement paved way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the **Department of Computer Science and Engineering** who were helpful in many ways for the completion of the project.

ABSTRACT

Since, motorcycles are affordable and a daily mode of transport there has been a rapid increase in motorcycle accidents, due to the fact that most of the motorcyclists do not wear a helmet which has made it an ever-present danger everyday to travel by motorcycle. In last couple of years Government has made it a punishable offense to ride a motorcycle without helmet. The existing video surveillance based system is effective but it requires significant human assistance whose efficiency decreases with time and human biasing also comes into picture. So, automation of this process is highly desirable. In this paper, we propose an approach for automatic detection of motorcyclists without helmet using surveillance videos in real-time. The proposed approach first detects the motorcycle from the surveillance videos using background subtraction. Then it classifies between helmet and non-helmet using 1st order and 2nd order derivative edge detection algorithm and neural network. After detection if the motorcyclists are found without helmet then it will trace the vehicle number plate of the motorcyclists using (OCR) Optical Character Recognition and Neural Network. The project detects the number plate of the bike and digitally retrieve the bike plate number and also saves the frame of the picture where the rider is found non-helmet and the frame can be held as a proof against the biker. This helps government to raise challan to the rider and make sure the society that someone is watching the roads even when the roads are empty.

TABLE OF CONTENTS

ABSTRACT	v
LIST OF FIGURES	viii

CHAPTER No.	TITLE	PAGE No.
1.	INTRODUCTION	1
	1.1 WHAT IS OBJECT DETECTION	1
	1.2 TWO-STEP OBJECT DETECTION	2
	1.3 ONE-STEP OBJECT DETECTION	3
	1.4 HEATMAP BASED OBJECT DETECTION	4
2.	LITERATURE SURVEY	5
	2.1 WORKFLOW OF OBJECT DETECTION	5
	2.1.1 <i>WHAT IS FEATURE EXTRACTION</i>	5
	2.1.2 <i>WHY FEATURE EXTRACTION USEFUL</i>	5
	2.2 YOLO FOR OBJECT DETECTION	6
3.	METHODS AND TECHNIQUES USED	8
	3.1 WHAT IS YOLOV3	8
	3.2 HOW YOLO IS USEFUL	8
	3.3 YOLOV3 FUNCTION	9
	3.4 HOW TO ENCODE BOUNDING BOXES	12
	3.5 TESTING	15
	3.6 IMPLEMENTING YOLO IN PYTHON	16
4.	RESULTS	25

5.	CONCLUSION AND FUTURE WORK	29
5.1	CONCLUSION	29
5.2	FUTURE WORK	29
	 REFERENCES	 30
	 APPENDIX	
	A. SOURCE CODE	31
	B. SCREENSHOTS	36

LIST OF FIGURES

FIGURE No.	FIGURE NAME	PAGE No.
1.1	OBJECT DETECTION	1
1.2.1	TWO STEP DETECTION LAYOUT	2
1.3.1	ONE STEP DETECTION	3
1.4.1	HEATMAP DETECTION	4
2.1.1	WORKFLOW OF OD	6
2.2.1	YOLO OBJECT DETECTOR	7
3.3.1	YOLO STEP1	9
3.3.2	YOLO STEP2	10
3.4.1	YOLO FINAL DETECTION	14
3.6.1	DETECTION OF CARS	24
4.1	VIDEO SNAPSHOT	25
4.2	TRAINING WEIGHTS	26
4.3	CREATING FRAMES	27
4.4	OUTPUT FRAME	28

CHAPTER 1

INTRODUCTION

1.1 WHAT IS OBJECT DETECTION

Object Detection is a common Computer Vision problem which deals with identifying and locating object of certain classes in the image. Interpreting the object localisation can be done in various ways, including creating a bounding box around the object or marking every pixel in the image which contains the object (called segmentation).

Object detection was studied even before the breakout popularity of CNNs in Computer Vision. While CNNs are capable of automatically extracting more complex and better features, taking a glance at the conventional methods can at worst be a small detour and at best an inspiration. Object detection before Deep Learning was a several step process, starting with edge detection and feature extraction using techniques like SIFT, HOG etc. These image were then compared with existing object templates, usually at multi scale levels, to detect and localize objects present in the image.

Image classification involves assigning a class label to an image, whereas object localization involves drawing a bounding box around one or more objects in an image. Object detection is more challenging and combines these two tasks and draws a bounding box around each object of interest in the image and assigns them a class label. Together, all of these problems are referred to as object recognition.



Fig 1.1 object detection

1.2 TWO-STEP OBJECT DETECTION

Two-Step Object Detection involves algorithms that first identify bounding boxes which may potentially contain objects and then classify each bounding separately.

The first step requires a *Region Proposal Network*, providing a number of regions which are then passed to common DL based classification architectures. From the hierarchical grouping algorithm in RCNNs (which are extremely slow) to using CNNs and ROI pooling in Fast RCNNs and anchors in Faster RCNNs (thus speeding up the pipeline and training end-to-end), a lot of different methods and variations have been provided to these region proposal networks (RPNs).

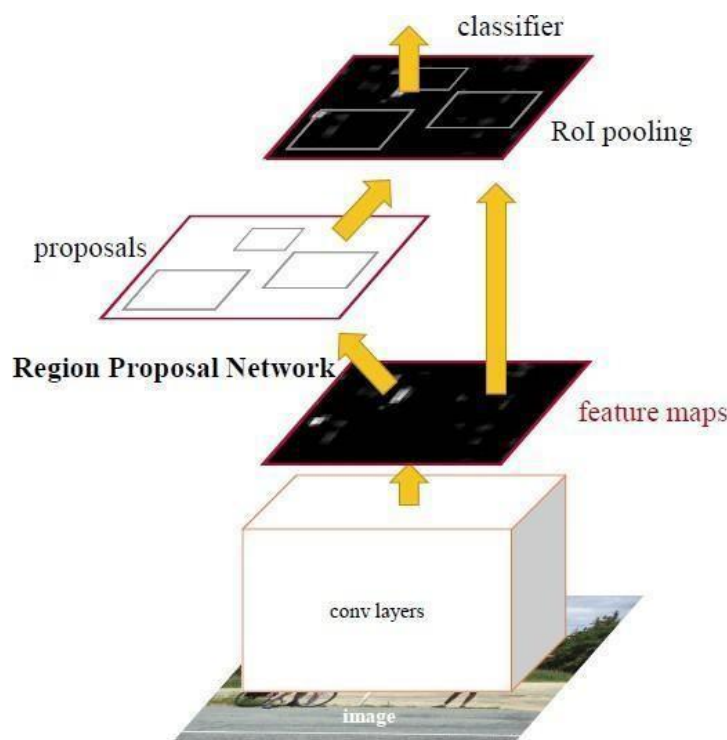


Fig 1.2.1 two step detection layout .

These algorithms are known to perform better than their one-step object detection counterparts, but are slower in comparison. With various improvements suggested over the years, the current bottleneck in the latency of Two-Step Object Detection networks is the RPN step. You can refer to this nice blog below for more details on RPN based object detection.

1.3 ONE-STEP OBJECT DETECTION

With the need of real time object detection, many one-step object detection architectures have been proposed, like YOLO, YOLOv2, YOLOv3, SSD, RetinaNet etc. which try to combine the detection and classification step.

One of the major accomplishments of these algorithms have been introducing the idea of 'regressing' the bounding box predictions. When every bounding box is represented easily with a few values (for example, xmin, xmax, ymin and ymax), it becomes easier to combine the detection and classification step and dramatically speed up the pipeline.

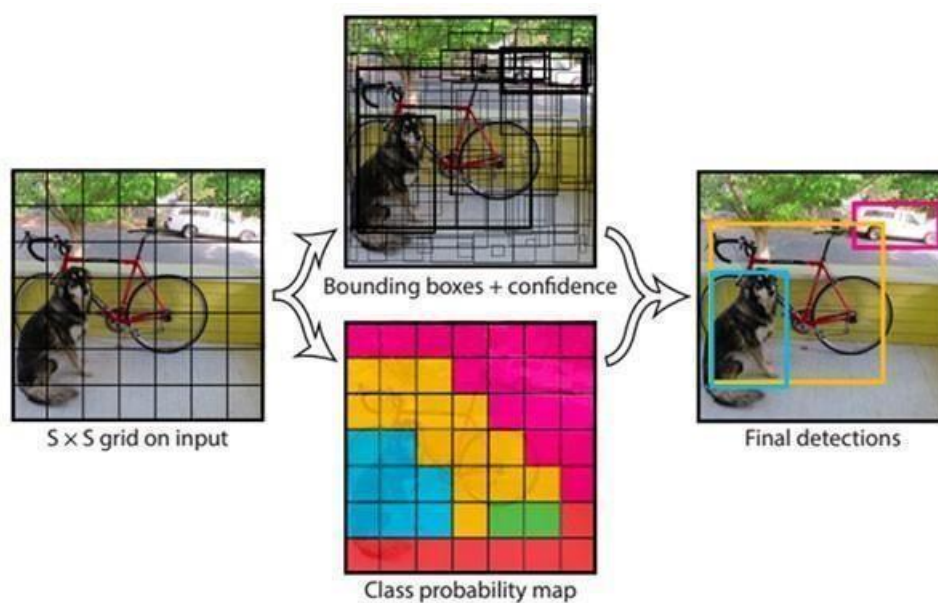


Fig 1.3.1 one step object detection layout.

For example, YOLO divided the entire image into smaller grid boxes. For each grid cell, it predicts the class probabilities and the x and y coordinates of every bounding box which passes through that grid cell. *Kinda like the image based captcha where you select all smaller grids which contain the object!!!*

These modification allow one-step detectors to run faster and also work on a global level. However, since they do not work on every bounding box separately, this can cause them to

perform worse in case of smaller objects or similar object in close vicinity. There have been multiple new architectures introduced to give more importance to lower level features too, thus trying to provide a balance.

1.4 HEATMAP BASED OBJECT DETECTION

Heatmap-based object detection can be, in some sense, considered an extension of one-shot based Object Detection. While one-shot based object detection algorithms try to directly regress the bounding box coordinates (or offsets), heatmap-based object detection provides probability distribution of bounding box corners/center.

Based on the positioning of these corner center peaks in the heatmaps, resulting bounding boxes are predicted. Since a different heatmap can be created for every class, this method also combines detection and classification. While heatmap-based object detection is currently leading new research, it is still not as fast as conventional one-shot object detection algorithms. This is due to the fact that these algorithms require more complex backbone architectures (CNNs) to get respectable accuracy.

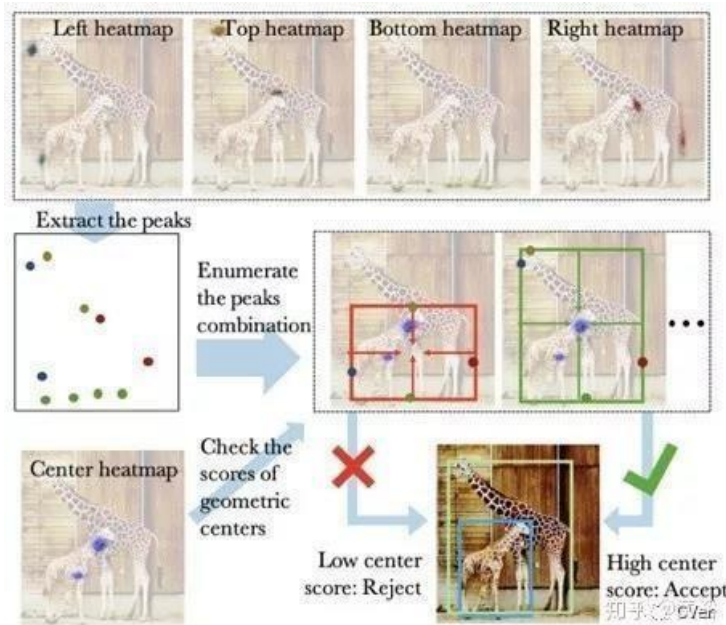


Fig 1.4.1 heatmap object detection layout.

CHAPTER 2

LITERATURE SURVEY

2.1 WORK FLOW OF OBJECT DETECTION

Every Object Detection Algorithm has a different way of working, but they all work on the same principle.

Feature Extraction: They extract features from the input images at hands and use these features to determine the class of the image. Be it through MatLab, Open CV, Viola Jones or Deep Learning.

2.1.1 WHAT IS FEATURE EXTRATION

Feature extraction is a part of the dimensionality reduction process, in which, an initial set of the raw data is divided and reduced to more manageable groups. So when you want to process it will be easier. The most important characteristic of these large data sets is that they have a large number of variables. These variables require a lot of computing resources to process them. So Feature extraction helps to get the best feature from those big data sets by select and combine variables into features, thus, effectively reducing the amount of data. These features are easy to process, but still able to describe the actual data set with the accuracy and originality.

2.1.2 WHY FEATURE EXTRACTION IS USEFUL

The technique of extracting the features is useful when you have a large data set and need to reduce the number of resources without losing any important or relevant information. Feature extraction helps to reduce the amount of redundant data from the data set.

In the end, the reduction of the data helps to build the model with less machine's efforts and also increase the speed of learning and generalization steps in the machine learning process.

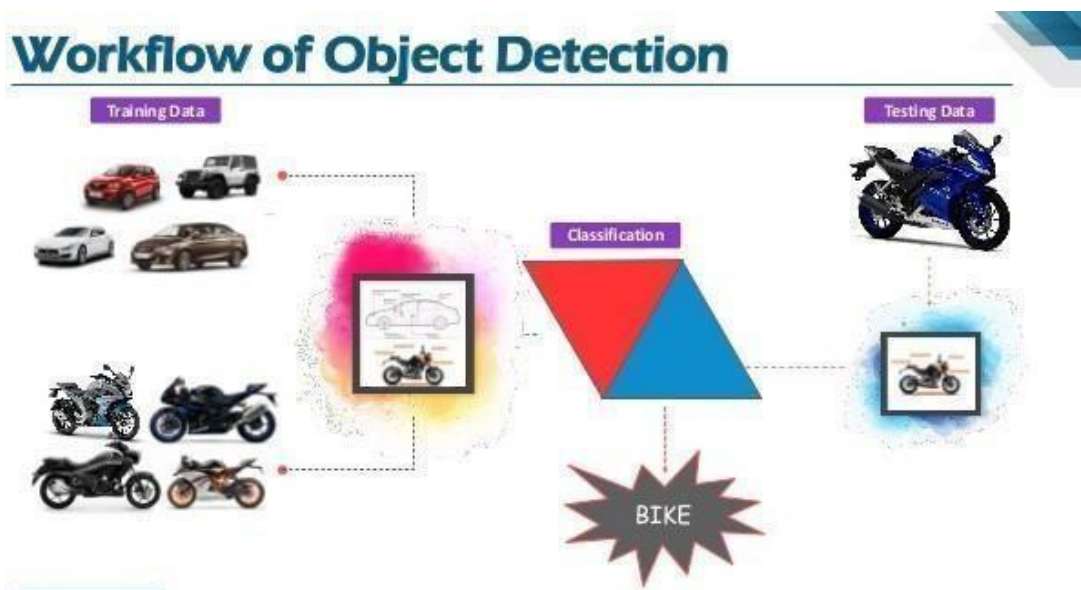


Fig 2.1.1 workflow of object detection

2.2 YOLO FOR OBJECT DETECTION

Object detection is a computer vision task that involves both localizing one or more objects within an image and classifying each object in the image.

It is a challenging computer vision task that requires both successful object localization in order to locate and draw a bounding box around each object in an image, and object classification to predict the correct class of object that was localized.

The “*You Only Look Once*,” or YOLO, family of models are a series of end-to-end deep learning models designed for fast object detection, developed by Joseph Redmon, et al. and first described in the 2015 paper titled “You Only Look Once: Unified, Real-Time Object Detection.”

The approach involves a single deep convolutional neural network (originally a version of GoogLeNet, later updated and called DarkNet based on VGG) that splits the input into a grid of cells and each cell directly predicts a bounding box and object classification. The result is a large number of candidate bounding boxes that are consolidated into a final prediction by a post-processing step.

There are three main variations of the approach, at the time of writing; they are YOLOv1, YOLOv2, and YOLOv3. The first version proposed the general architecture, whereas the second version refined the design and made use of predefined anchor boxes to improve bounding box proposal, and version three further refined the model architecture and training process.

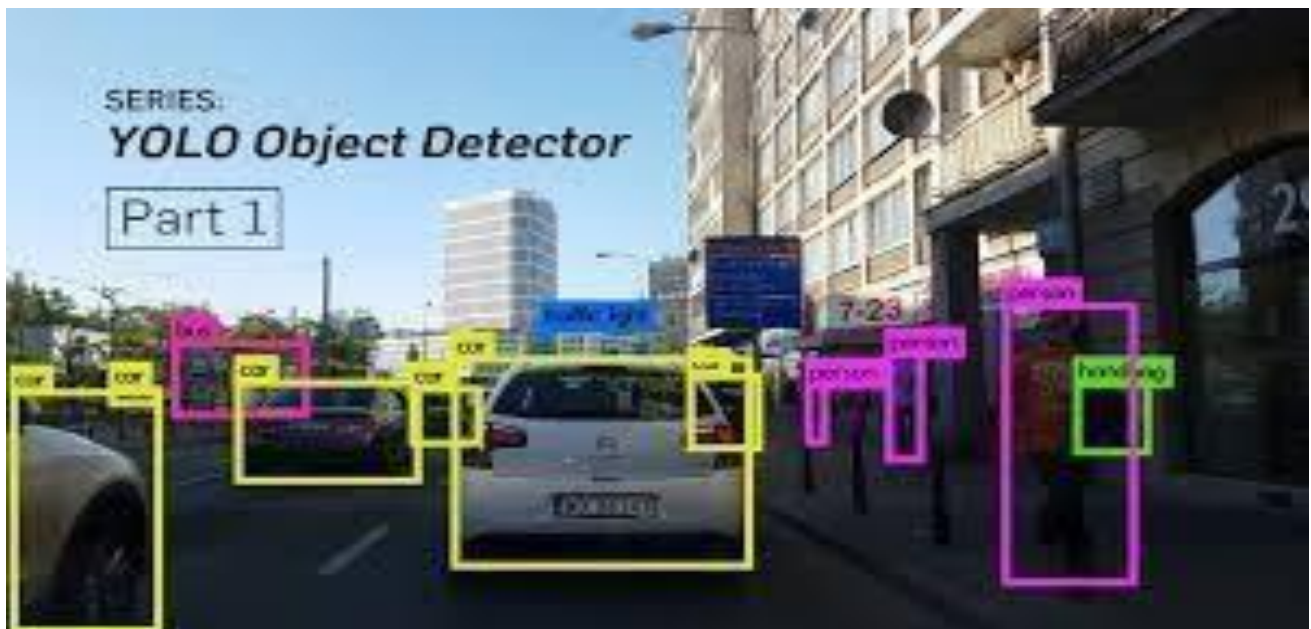


Fig 2.2.1 yolo object detector

CHAPTER 3

METHODS AND TECHNIQUES USED

3.1 WHAT IS YOLOV3

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.

3.2 HOW YOLO IS 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.

3.3 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:

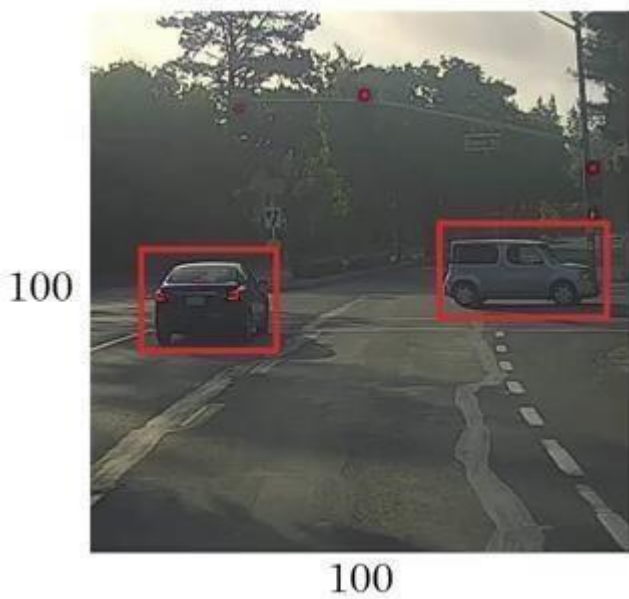


Fig 3.3.1 yolo step1

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

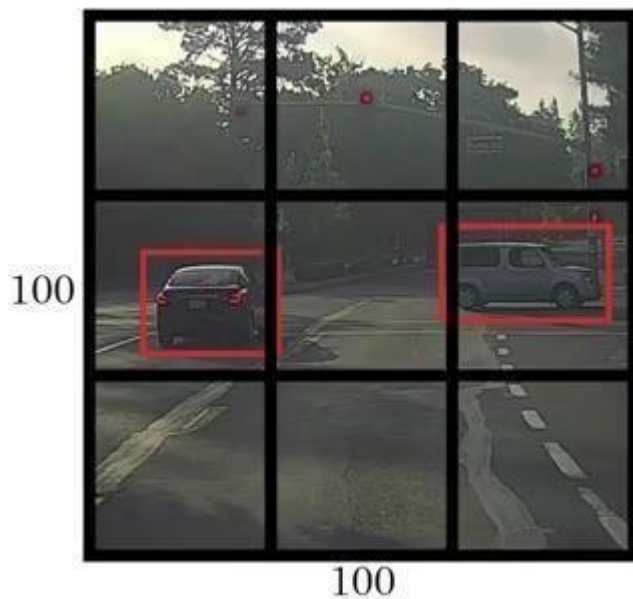


Fig 3.3.2 yolo step2

- 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

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

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

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:

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 and $c_3 = 0$. 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

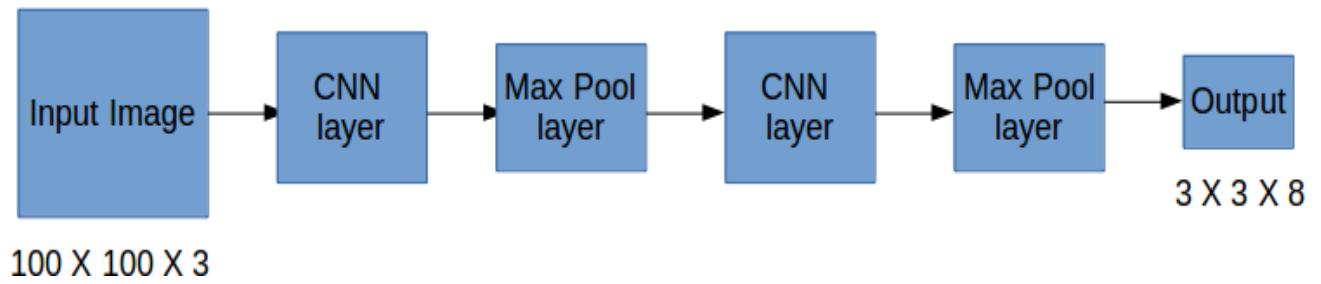


Fig 3.3.4 yolo process

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).

3.4 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

$p_c = 1$ 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

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.

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 bounding 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:

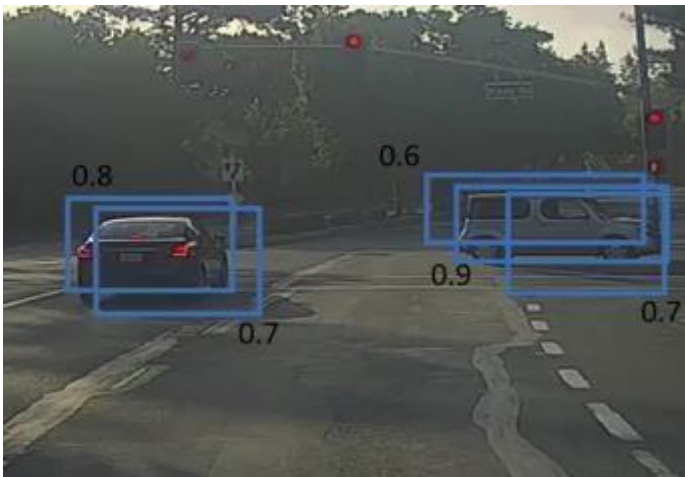
$$\text{IoU} = \text{Area of the intersection} / \text{Area of the union, i.e.}$$

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

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:



3.4.1 yolo final detection

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:
3. Repeat step 2 until all the boxes are either taken as the output prediction or discarded

3.5 TESTING

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

Finally, the Non-Max Suppression technique will be applied on the predicted boxes to

obtain a single prediction per object.

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

- Takes an input image of shape (608, 608, 3)
- Passes this image to a convolutional neural network (CNN), which returns a (19, 19, 5, 85) dimensional output
- The last two dimensions of the above output are flattened to get an output volume of (19, 19, 425):
 - o Here, each cell of a 19 X 19 grid returns 425 numbers
 - o $425 = 5 * 85$, where 5 is the number of anchor boxes per grid
 - o $85 = 5 + 80$, 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

3.6 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_bo  
xes, 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):  
  
    x11 =  
  
    max(box1[0],box2[0]) y11  
  
    = 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, i  
ou_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:

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)

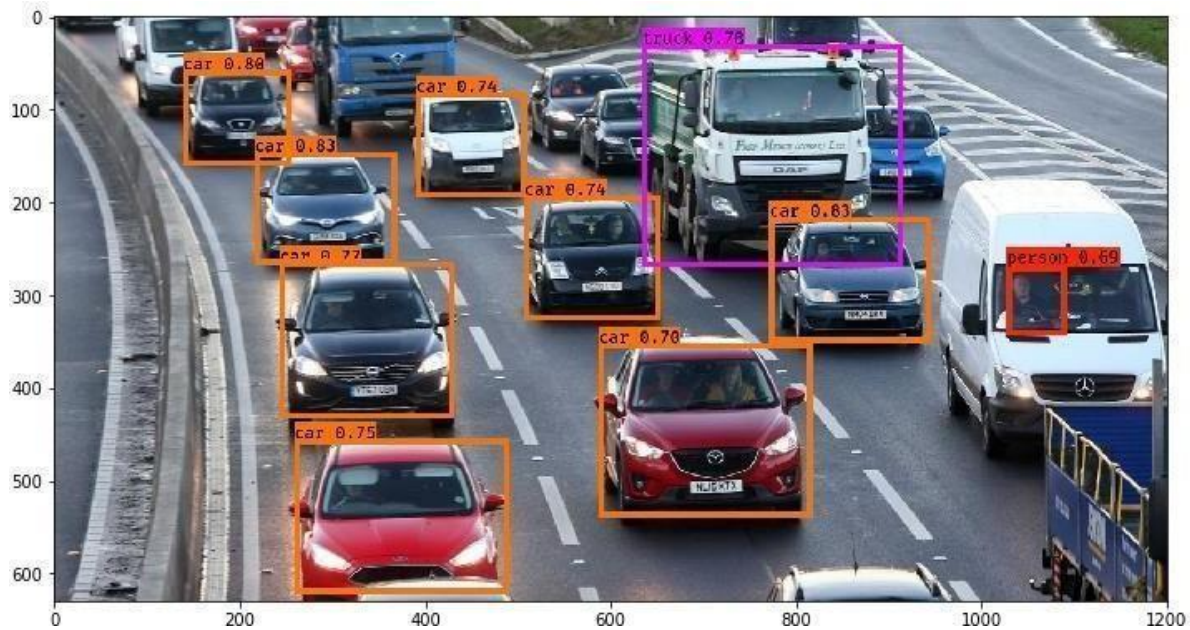


Fig 3.6.1 detection of cars

CHAPTER 4

RESULTS

At first we take a testing video and name it bike1 this is the file name that also coded in the code. The testing video consists of a non-helmet bike rider riding a bike . The main theme is that to detect the non-helmet rider's bike and recognize the number plate and retrieve the number digitally and save the frame of the video where it finds the non-helmet rider.

The video is divided into frames/images for every second where these are used to detect for the trained weights. The weights that are defined in the code are the trained weights.

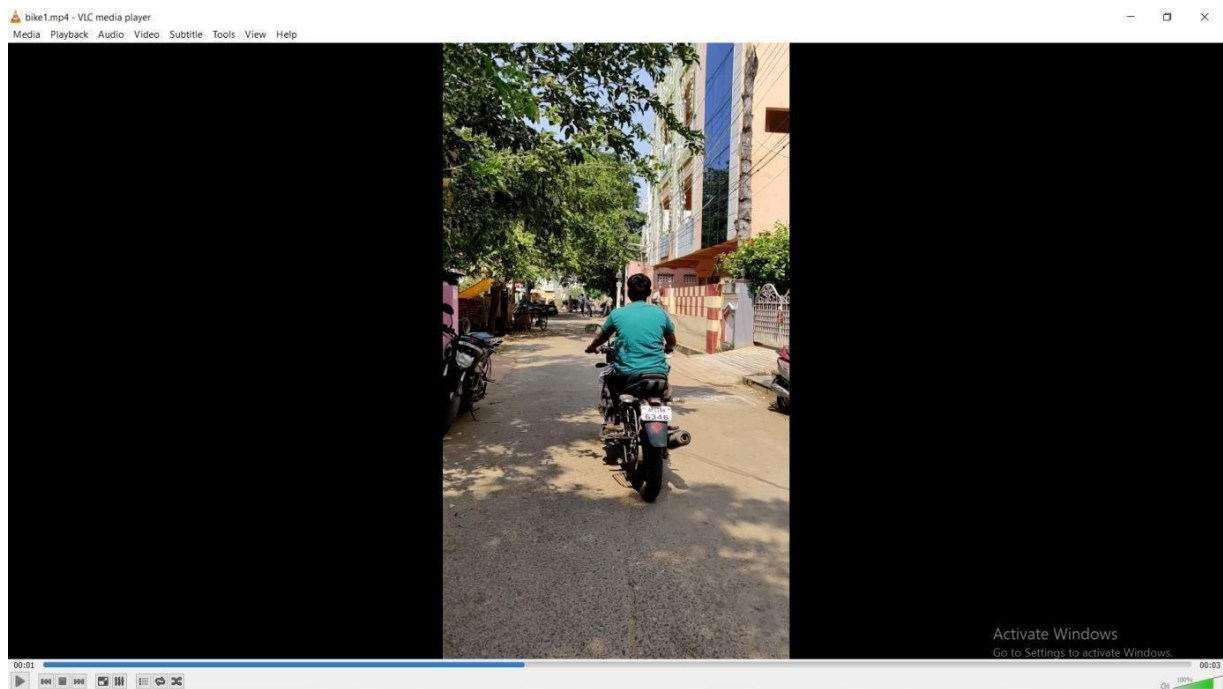


Fig 4.1 video snapshot

The trained weights are default in the base code so no need to train the objects again for the code sake. The objects here are detected by the trained weights.

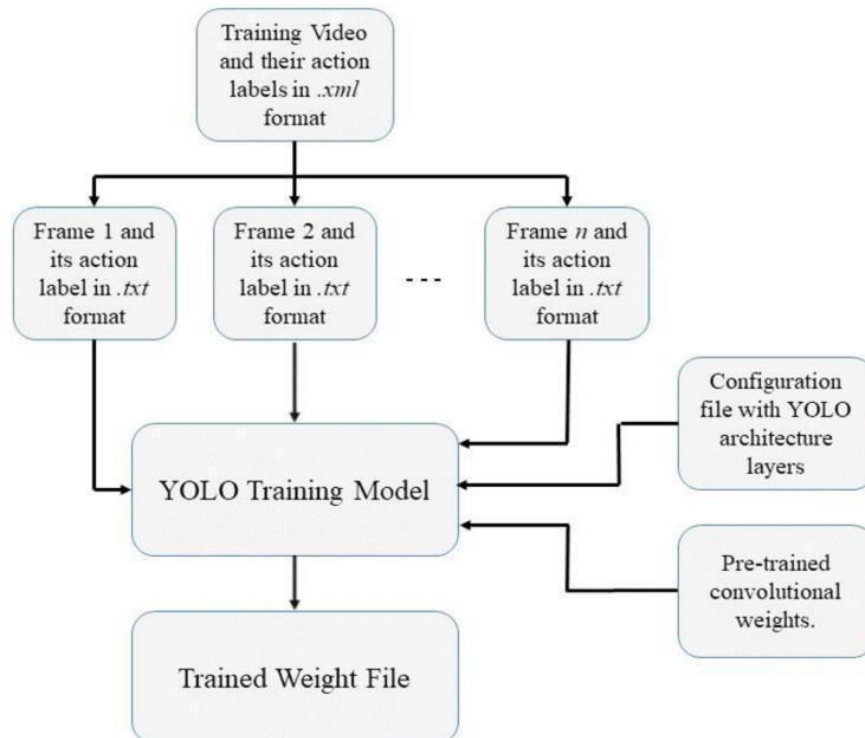


Fig 4.2 training weights

Some regular images of bikes, helmets and bike riders with and without helmet are given as input to YOLOv3 model to train for the custom classes. The weights generated after training are used to load the model. Once this is done, an image is given as input. The model detects all the five classes trained. From this we obtain the information regarding person riding motorbike.

If the person is not wearing a helmet, then we can easily extract the other class information of the rider. This can be used to extract the license plate. Once the helmetless rider is detected, the associated person class is detected. This is done by finding whether the co-ordinates of the no helmet class lie inside the person class or not.

Some regular images of bikes, helmets and bike riders with and without helmet are given as input to YOLOv3 model to train for the custom classes. The weights generated after training are used to load the model. From the images, the above five classes are detected. The model then detects all the five classes from images when given as input.

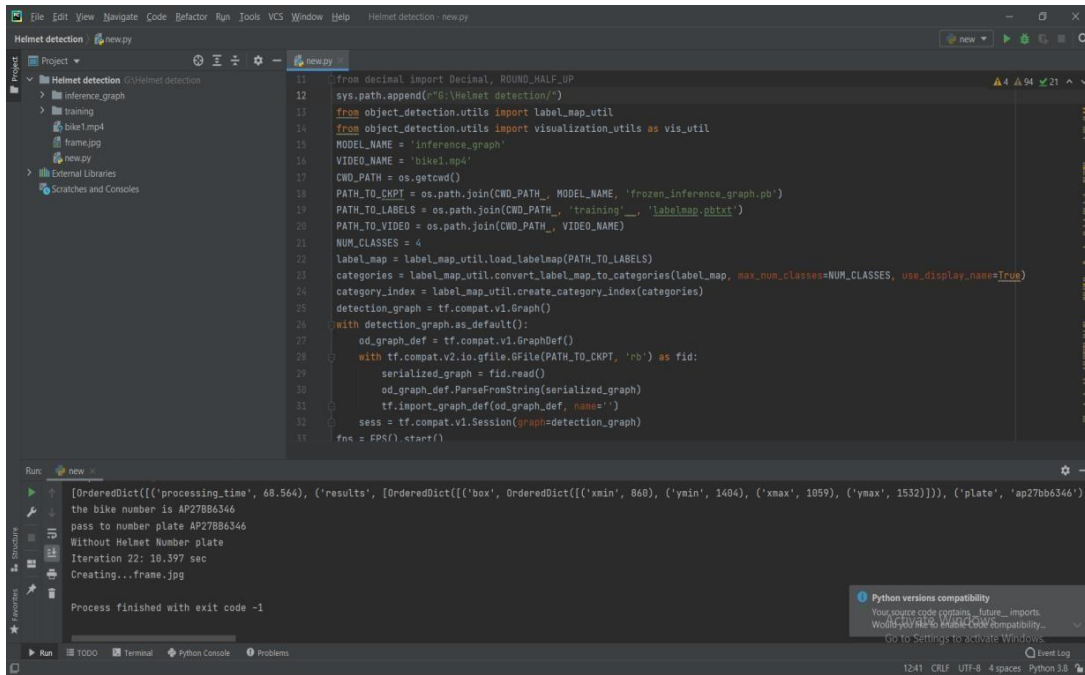


Fig 4.3 creating frames

We provide video of our own by each frame (i.e as images) which results in detecting the classes of above mentioned .The trained model can detect the classes without helmets, this can be used to differentiate the classes and obtain persons and helmets count . If the person on two wheeler without helmet is identified then it moves to next step .

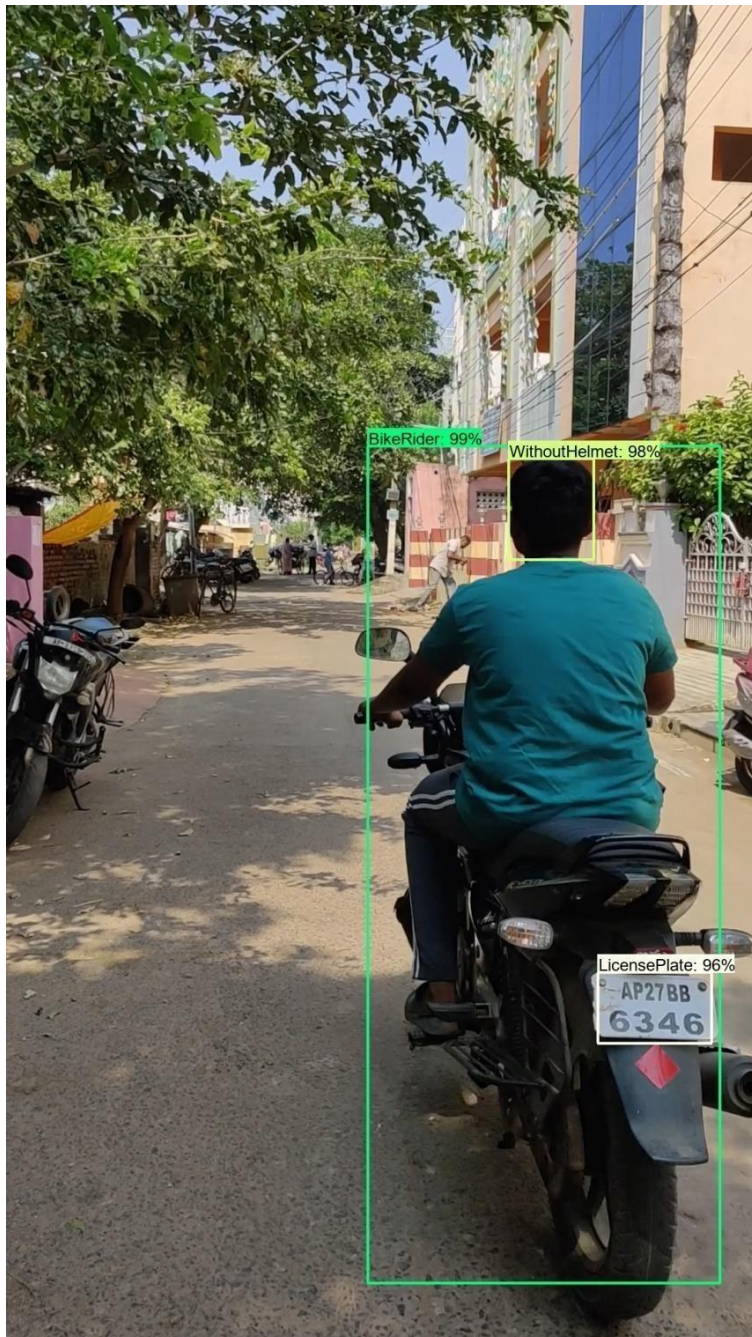


Fig output Frame

If the person is not wearing a helmet, then we can easily extract the other class information of the rider. This can be used to extract the license plate. Once the helmetless rider is detected, the associated person class is detected. This is done by finding whether the co- ordinates of the no helmet class lie inside the person class or not.

5.1 CONCLUSION

In the project, we have described a framework for automatic detection of motorcycle riders without helmet from CCTV video and automatic retrieval of vehicle license number plate for such motorcyclists. The use of yolov3 and transfer learning has helped in achieving good accuracy for detection of motorcyclists not wearing helmets. The accuracy obtained was 98.72%. But, only detection of such motorcyclists is not sufficient for taking action against them. So, the system also recognizes the number plates of their motorcycles and stores them. The stored number plates can be then used by Transport Office to get information about the motorcyclists from their database of licensed vehicles. Concerned motorcyclists can then be penalized for breach of law.

5.2 FUTURE WORK

The proposed system was able to cope up with certain challenges while detecting the motorcyclists with and without helmet such as poor quality of the image, brightness, slight changes in angle, etc. The future of this project lies in the ideas of the implements that it focused on this project. The idea can also be pointed in sense of cars which violate the road rules also can be used in parking lots where some people disorderly place their cars , even damage other cars around them ,these projects helps in detecting those people.

The detection and number plate recognition also help in lifting the tolls where they create many traffic if any misfunction of server. These detect the number plates and automatically detect the amount for the toll pays. The future work of this project lies in the ways of using it .

REFERENCES

- [1] K. Dahiya, D. Singh and C.K.Mohan, "*Automatic detection of bike riders Without helmet using surveillance videos in real-time*", Proceeding of International Joint Conference Neural Networks (IJCNN), Vancouver, Canada, 24-2 July 2016, pp.3046-3051.
- [2] R. V. Silva, T. Aires, and V. Rodrigo, "*Helmet Detection on Motorcyclists Using image descriptors and classifiers*", Proceeding of Graphics, Patterns and Images (SIBGRAPI), Rio de Janeiro, Brazil, 27-30 August 2014
- [3] Pathasu Doughmala, Katanyoo Klubsuwan, "*Half and Full Helmet Detection in Thailand using Haar Like Feature and Circle Hough Transform on Image Processing*" Proceeding of IEEE International Conference on Computer and Information Technology, Thailand, Bangkok, pg. 611-614, 2016.
- [4] J. Chiverton, "*Helmet presence classification with motorcycle detection and Tracking*", IET Intelligence Transport Systems (ITS), Volume 6, Issue no. 3, pp. 259-269, 2012.
- [5] C.C. Chiu, M.Y. Ku, and H. -T. Chen, "*Motorcycle detection and tracking system with occlusion segmentation*" Proceeding of International Workshop on Image Analysis for Multimedia Interactive Services, Santorini, Greece, 6-8 June 2007, pp. 32-32.
- [6] C.Y.Wen, S.H.Chiu, J.J.Liaw, and C.P. Lu, "*The safety helmet detection for ATM's surveillance system via the modified Hough transform*" Proceedings of IEEE 37th Annual International Carnahan Conference on Security Technology, pp. - 364-369, 2003.
- [7] Romuere Silva, Kelson Aires, Rodrigo Veras, Thiago Santos, Kalyf Lima and Andre Soares, "*Automatic motorcycle detection on public roads*" CLEIELECTRONIC JOURNAL, Volume 16, Number 3, Paper 04, December 2013.

APPENDIX

A. SOURCE CODE

```
from __future__ import absolute_import, division, print_function

import os
import cv2
import numpy as np
import tensorflow as tf
import sys
from imutils.video import FPS
import imutils
import time

from decimal import Decimal, ROUND_HALF_UP
sys.path.append(r"G:\Helmet detection/")
from object_detection.utils import label_map_util
from object_detection.utils import visualization_utils as vis_util
MODEL_NAME = 'inference_graph'
VIDEO_NAME = 'bike1.mp4'
CWD_PATH = os.getcwd()
PATH_TO_CKPT = os.path.join(CWD_PATH, MODEL_NAME, 'frozen_inference_graph.pb')
PATH_TO_LABELS = os.path.join(CWD_PATH, 'training', 'labelmap.pbtxt')
PATH_TO_VIDEO = os.path.join(CWD_PATH, VIDEO_NAME)
NUM_CLASSES = 4

label_map = label_map_util.load_labelmap(PATH_TO_LABELS)
categories = label_map_util.convert_label_map_to_categories(label_map,
max_num_classes=NUM_CLASSES, use_display_name=True)
category_index = label_map_util.create_category_index(categories)
detection_graph = tf.compat.v1.Graph()
with detection_graph.as_default():
    od_graph_def = tf.compat.v1.GraphDef()
    with tf.compat.v2.io.gfile.GFile(PATH_TO_CKPT, 'rb') as fid:
```

```
serialized_graph = fid.read()
od_graph_def.ParseFromString(serialized_graph)
tf.import_graph_def(od_graph_def, name='')
sess = tf.compat.v1.Session(graph=detection_graph)
```

```

fps = FPS().start()
video = cv2.VideoCapture(PATH_TO_VIDEO)
import json
from collections import OrderedDict
from glob import glob
import cv2
import requests
def platemain():
    regions = ['in']
    result= []
    path='frame.jpg'
    with open(path,'rb') as fp:
        response =
        requests.post(
            'https://api.platerecognizer.com/v1/plate-reader/',
            files=dict(upload=fp),
            data=dict(regions=regions),
            headers={'Authorization': 'Token ' +
'fe801a314498e5fd43a6069099e65b7bc5ff9c3d'})
        print("respond.status_code",response.status_code)
        result.append(response.json(object_pairs_hook=OrderedDict))
    print(result)
    time.sleep(1)
    im=cv2.imread(path)
    resp_dict = json.loads(json.dumps(result, indent=2))
    if resp_dict[0]['results']:
        num=resp_dict[0]['results'][0]['plate']
        boxs=resp_dict[0]['results'][0]['box']
        xmins,ymins,ymaxs,xmaxs=boxs['xmin'],boxs['ymin'],boxs['ymax'],boxs['xmax']
    #
    cv2.imshow("image",im) #
        cv2.waitKey(0)
    img = cv2.imread(path,cv2.IMREAD_GRAYSCALE)
    #  cv2.imshow("Gray Image",img)

```

```
# cv2.waitKey(0)
edges = cv2.Canny(img,100,200)
# cv2.imshow("Edge
Image",edges)
```

```

# cv2.waitKey(0)
cv2.rectangle(im, (xmins, ymins), (xmaxs, ymaxs), (255,0,0), 2)
cv2.rectangle(edges, (xmins, ymins), (xmaxs, ymaxs), (255,0,0), 2)
# cv2.imshow("Box
Edges",edges) # cv2.waitKey(0)
# cv2.imshow("Box On
Original",im) # cv2.waitKey(0)
font = cv2.FONT_HERSHEY_SIMPLEX
cv2.putText(im,num,(xmins, ymins-10), font, 1,(255,0,0),2,cv2.LINE_AA)
# cv2.imshow("Number",im)
# cv2.waitKey(0)
cv2.destroyAllWindows()
print("the bike number is {}".format(str(num).upper()))
return str(num).upper()

```

```

def resize(w, h, w_box, h_box, pil_image):
    f1 = 1.0*w_box/w # 1.0 forces float division in Python2
    f2 = 1.0*h_box/h
    factor = min([f1, f2])
    width = int(w*factor)
    height =
    int(h*factor)
    return pil_image.resize((width, height))

```

```

j=0
s=[]
with detection_graph.as_default():
    with tf.compat.v1.Session(graph=detection_graph) as sess:
        nmp=[]
        st=[]
        while True:
            start_time = time.time()
            j+=1
            ret, image_np = video.read()

```

```
if ret==True :
```

```
    image_np_expanded = np.expand_dims(image_np, axis=0)
```

```

image_tensor = detection_graph.get_tensor_by_name('image_tensor:0')
boxes = detection_graph.get_tensor_by_name('detection_boxes:0')
scores = detection_graph.get_tensor_by_name('detection_scores:0')
classes = detection_graph.get_tensor_by_name('detection_classes:0')
num_detections = detection_graph.get_tensor_by_name('num_detections:0')
(boxes, scores, classes, num_detections) = sess.run(
    [boxes, scores, classes, num_detections],
    feed_dict={image_tensor: image_np_expanded})
# scores=(scores[0]*10).astype(int)
cl=(classes[0].astype(int))[:7]
sc=(scores[0][:7])*100
sc=list(map(int,sc))
# print(cl,sc)
d={1:0,2:0,3:0,4:0}
    for i in
        range(len(cl)):
            if(sc[i]>d[cl[i]]):
                d[cl[i]]=sc[i]
# Visualization of the results of a detection.
vis_util.visualize_boxes_and_labels_on_image_array(
    image_np,
    np.squeeze(boxes),
    np.squeeze(classes).astype(np.int32),
    np.squeeze(scores),
    category_index,
    use_normalized_coordinates=True,
    line_thickness=4,
    min_score_thresh=0.85)
data = {}
if(d[3]>80):
    if(d[4]<90):
        print("Without Helmet: No Number plate")
if (d[4] > 90):
    name = 'frame.jpg'

```

```
print('Creating...' + name)
cv2.imwrite(name, image_np)
```

```

number = platemain()
if len(number) not in [8, 9, 10]: continue
print("pass to number plate", number)
if (d[3] > 90):
    print("Without Helmet Number plate ")
    if number not in nmp:
        nmp.append(number)
        st.append("Without Helmet")
elif (d[2] > 90):
    print("With Helmet Number pLtae")
    if number not in nmp:
        nmp.append(number)
        st.append("With Helmet")
elif (d[1] < 90):
    print("Vehicle with number plate")
    if number not in nmp:
        nmp.append(number)
        st.append('Other Vehicle')
# print(nmp, st)
print('Iteration %d: %.3f sec'%(j, time.time()-start_time))
cv2.imshow('object detection', cv2.resize(image_np, (800,600)))
#cv2.waitKey()
if cv2.waitKey(1) & 0xFF == ord('q'):
    break
cv2.destroyAllWindows()

```

B. SCREENSHOTS

