HELMET WEAR DETECTION USING DEEP LEARNING

A CAPSTONE PROJECT REPORT

Submitted in partial fulfillment of the
Requirement for the award of the
Degree of

BACHELOR OF TECHNOLOGY
IN
COMPUTER SCIENCE AND ENGINEERING

by

VENKATA DURGA AVINASH BOBBALA (17BCD7069)

Under the Guidance of

DR. MALLIKHARJUNA RAO K

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING
VIT-AP UNIVERSITY
AMARAVATI- 522237

JANUARY 2021

CERTIFICATE

This is to certify that the Capstone Project work titled "**HELMET WEAR DETECTION USING DEEP LEARNING**" that is being submitted by **VENKATA DURGA AVINASH BOBBALA (17BCD7069)** is in partial fulfillment of the requirements for the award of Bachelor of Technology, is a record of bonafide work done under my guidance. The contents of this Project work, in full or in parts, have neither been taken from any other source nor have been submitted to any other Institute or University for award of any degree or diploma and the same is certified.

Dr. MALLIKHARJUNA RAO K
Guide

The thesis is satisfactory / unsatisfactory

Internal Examiner **External Examiner**

Approved by

PROGRAM CHAIR **DEAN**

B. Tech. CSE School of COMPUTER SCIENCE AND ENGINEERING

ACKNOWLEDGEMENTS

This Project itself is an acknowledgement to the inspiration, drive and technical assistance contributed by myself. This project would have never seen light of this day without the help and guidance I have received.

I express my gratitude to Dr. Mallikharjuna rao K, Department of Computer science and engineering, VIT-AP University for providing me with excellent knowledge and environment that laid the potentially strong foundation for my professional life.

I own an incalculable debt of my all staff of Department of Computer Science and Engineering for their help.

I extend my heartfelt thanks to my parents, teachers, friends for their support and help.

ABSTRACT

The continuous motorization of traffic has led to a sustained increase in global number of road accidents. The government is trying hard to make people follow driving rules to reduce accidents and fatal deaths, especially in many developing countries like India where the motorcycle is the most used vehicle. As motor cycle is less safety vehicle where we don't have any coverage so it's must to wear a helmet to reduce the accident death rate.

This project is aimed at automatic detection of whether a bike rider is wearing a helmet or not using surveillance videos. The approach first detects the bike riders using background subtraction and then it detects whether the rider is wearing a helmet or not and then the licence plate number from the bike of violators will get extracted.

The algorithms and processes involved in the development of the project are Yolov3 which uses pre-trained weights for standard object detection problems and it has weights of helmet, bikes, bike rider. The Tensor Flow object detection API is the framework for creating a deep learning network that solves object detection problems, and license plate recognizer API is used for extracting the number from the license plate of the rule violators. ORB is used for the representation of bounding boxes around the object detected.

TABLE OF CONTENTS

List of Figures

CHAPTER 1
INTRODUCTION

According to the statistics given by transport ministry about 4 people die every hour in India because they do not wear a helmet. The number of two-wheeler users who died in road accidents was 48,746. Incidentally, 73.8 per cent of them did not wear a helmet.[1]

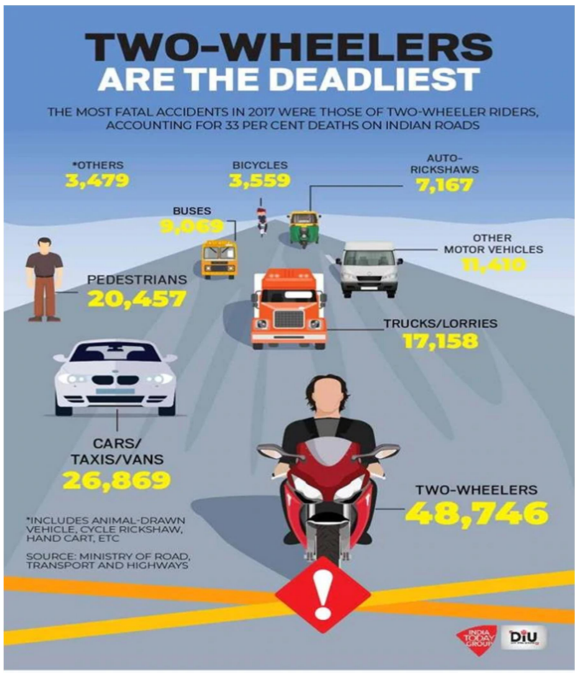

Figure 1 Road Accidents [1]
In the above figure 1 we can observe that Two-wheelers are the most affected and dangerous mode of transport compared to other motor vehicles.

Deep learning is a class of Machine learning in which a trained model works on its own using the inputs given during training period. Deep Learning algorithms build a mathematical model of sample data, known as "training data", in order to make predictions or decisions and are also used in the applications of object detection. Therefore, by training with a specific dataset, a Helmet detection model can be implemented. Using this helmet detection model helmet-less riders can be easily detected. Based on the detected classes the license plate of the rider is cropped out and saved as an image. This image is given to a license plate recognizer API model which recognizes the text and gives the License Plate number as output.

This project tries to mitigate the aforementioned problems by giving a feasible solution using the detection of continuous traffic video feed. Every vehicle on the street gets evaluated and a database of offenders will be generated real time. Hence every rider who are not wearing helmet will be prosecuted and this generates lot of awareness in the public which results in wearing a helmet while riding bikes, this method results in decrease of bike accidents.

1.1 Objectives

The following are the objectives of this project:

- The objective of this project is to develop an application for enforcing helmet wearing using CCTV cameras.
- The developed application aims to help law enforcement by police, and eventually resulting in changing risk behaviours and consequently reducing the number of accidents and its severity

1.2 Background and Literature Survey

According to the Research paper in 2016 titled Automatic Detection of Bike-riders without Helmet using Surveillance Videos in Real-time.[2] The Author from IIT Hyderabad developed the project using the algorithms like object classification, feature extraction and CNN for object detection.

Chiverton [3] proposed the use of circular arc to identify helmet in a video feed, it has very low accuracy. On the other hand, given the number of vehicles on the speed at a given instant, the computation that required is very heavy and consumes lots of resources. This method will determine any circular object around the bike rider as helmet.

Silva [4] proposed a system in which he tracks the vehicles using Kalman filter. An important advantage of this Kalman [5] tracking system is the ability to continue to track objects even if they are lightly occluded but when there were more than two or three motorcyclists appear

in a same frame, Kalman filter fails because Kalman filter mostly works well for linear state transitions (i.e tracking single objects/one object at a time). But to track multiple objects, we need non-linear functions to track them.

1.3 Organization of the Report

The remaining chapters of the project report are described as follows:

- Chapter 2 contains the proposed system, methodology, hardware and software details.
- Chapter 3 gives the cost involved in the implementation of the project.
- Chapter 4 implementation, use cases and test cases.
- Chapter 5 Results.
- Chapter 6 concludes the report.
- Chapter 7 consists of codes.
- Chapter 8 gives references.

CHAPTER 2

HELMET WEAR DETECTION USING DEEP LEARNING

This Chapter describes the proposed system, working methodology, software and hardware details.

2.1 Proposed System

The following block diagram (figure 2) shows the system architecture of this project.

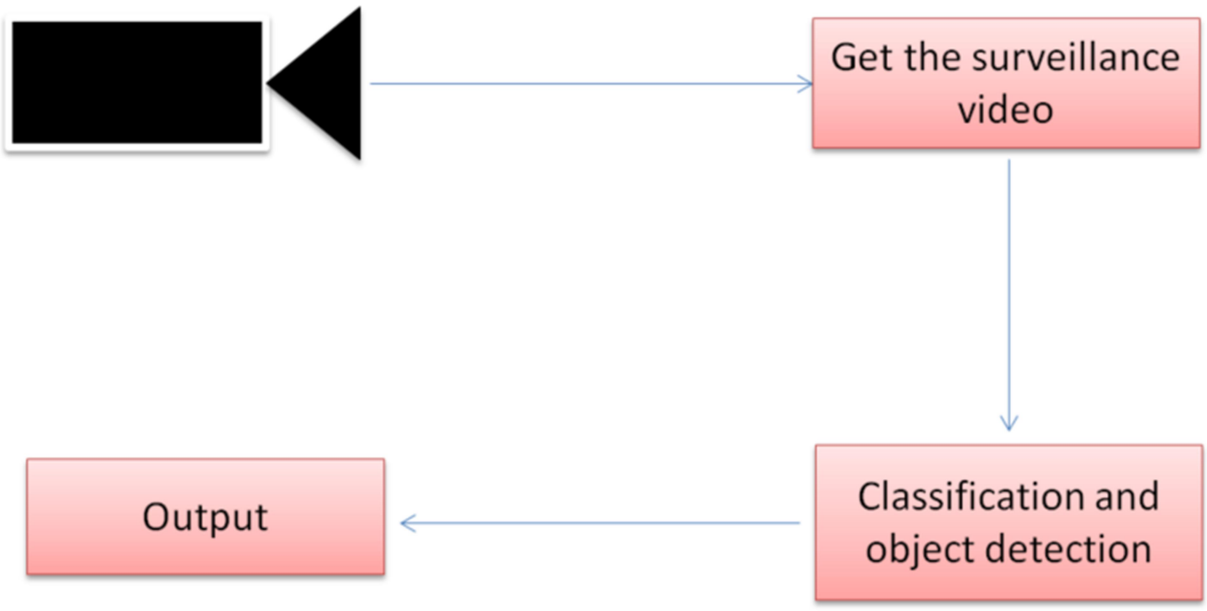

Figure 2 System Block Diagram

In the above figure 2 represents the system block diagram which explains that the video from the cctv on the streets will be considered as input and the classification and object detection takes place to differentiate whether a bike rider is wearing a helmet or not.

2.2 Working Methodology

For the real-time helmet detection, there is a need for accuracy and speed. Hence a DNN based model known as YOLO version3 was chosen. YOLO means You Only Look Once. YOLO V3 is extremely fast and accurate and is a great improvement over the previous versions of YOLO. It is based on single network evaluation unlike systems like R-CNN. Here, there are 5 objects to be detected and classified i.e., Helmet, No Helmet, Bike, Person on bike, and license plate. So for the object detection the collection of images containing each object is used for

training the custom model. Once the model has been trained, it can be used to detect these five objects.

2.2.1 Background Modeling

The first step of the methodology is background modeling and moving object detection here in this background modeling the moving objects in the surveillance video is detected and other all unwanted objects will not be considered.

2.2.2 Classification of Motorcyclists

To find bounding boxes of different objects, we used Gaussian background subtraction which uses a method to model each background pixel. The probable background colors are the ones which stay longer and are more static. On these varying pixels, we draw a rectangular bounding box. After obtaining all the objects of motorcyclists and non-motorcyclists, a CNN model is built using these images to separate the motorcyclists from other moving objects and able to distinguish between a motorcyclist and other objects.

2.2.3 Recognition of Motorcyclists without helmet

Here in this step, from the all images of motorcycles we will crop the top $1/4^{th}$ region of the biker as that is the portion where the biker wears a helmet. Then we check whether the helmet is there or not using the object detection and trained model. Then we separate the helmet motorcyclists and non-helmet motorcyclists.

2.2.4 Extraction of Number Plate

After the classification of helmet and non-helmet, number from the number plate gets extracted if the motorcyclist doesn't wear a helmet through a license plate recognizer API which performs accurately. Then using the reference of license number the rule violator gets fined.

2.3 Software and Hardware Requirements

2.3.1 Software requirements

1. Any python environment (PyCharm)

2. OpenCV, Yolov3, Tensorflow

2.3.2 Hardware requirements

1. i5 processor
2. HDD- 256GB
3. RAM- 8GB
4. LPR traffic camera

CHAPTER 3

COST ANALYSIS

The proposed solution will provide a completely free of cost system once installed (cost for only LPR cameras installed in traffic). Also, the software was build using free and open source technologies hence it has an overall software cost equal to zero. Every vehicle on the street will be evaluated and a database of the offenders will be generated real time.

CHAPTER 4

IMPLEMENTATION, USE CASES AND TEST CASES

4.1 Implementation

4.1.1 Dataset

Collection of five required objects that are bike, helmet, no helmet, biker, and license plate forms different custom datasets and used for training.

4.1.2 YOLO

The approach involves a single neural network trained end to end that takes a photograph as input and predicts bounding boxes and class labels for each bounding box directly. The technique offers lower predictive accuracy (e.g. more localization errors), although operates at 45 frames per second and up to 155 frames per second for a speed-optimized

version of the model. Conceptually, YOLO [6] divides the input image into an SxS grid, and if the center of an object falls into a grid cell, that grid cell is responsible for detecting that object. Then, each grid cell predicts B bounding boxes. Each bounding box has associated 5 predictions plus C conditional class probabilities [6]:

- x, y: Centered relative to the grid cell ($x \in [0..1]$)
- w, h: Width and height
- IOU: Interaction over union
- OBJconf: Confidence score (Objectness)
- C1, C2, ...Cn: Conditional class probabilities

The confidence score for each box can be defined as:

$$OBJconf = Pr(Object) * IOU_{truth,prediction}$$

And the conditional class probability as:

$$Ci = Pr(Classi \mid Object)$$

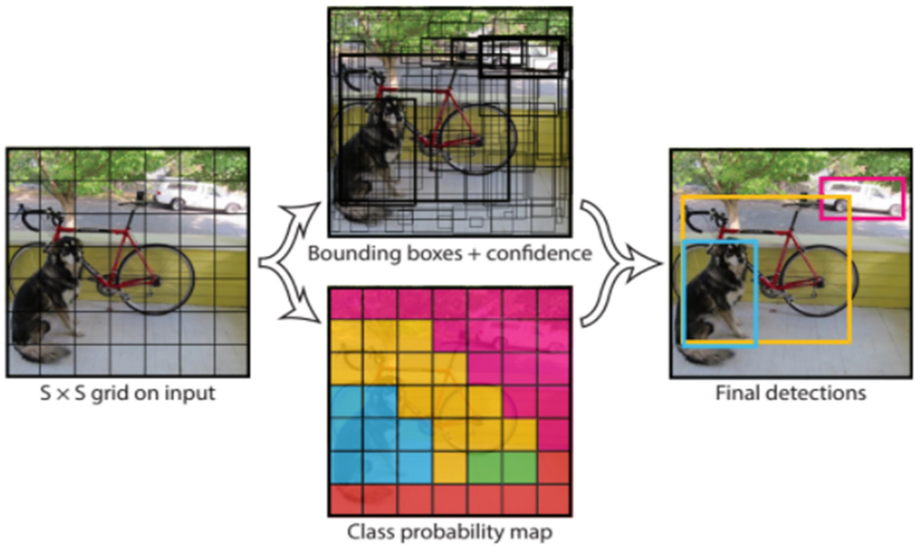

Figure 3 YoloV3 [6]

The above figure explains the method involved in Yolov3 that first the image will be drawn with S X S grids on the input and then the yolo works on detecting the objects present in the input by the bounding boxes and the bounding box having the highest confidence value will be considered as an accurate object shown as in the final detections.

- **STEPS FOR FUNCTIONING OF YOLO:**
- [7] YOLO first takes an input image:

Figure 3.1 Object detection

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

Figure 3.2 Making grids

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

- 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 car, Helmet, 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:

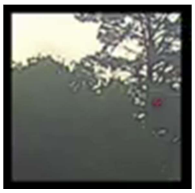

Figure 3.2.1 Grid 1

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

y =	0
	?
	?
	?
	?
	?
	?
	?

Here, '?' means that it doesn't matter what 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 ($c_2 = 1$):

Figure 3.2.2 Grid

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 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 X 3 X 8.

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

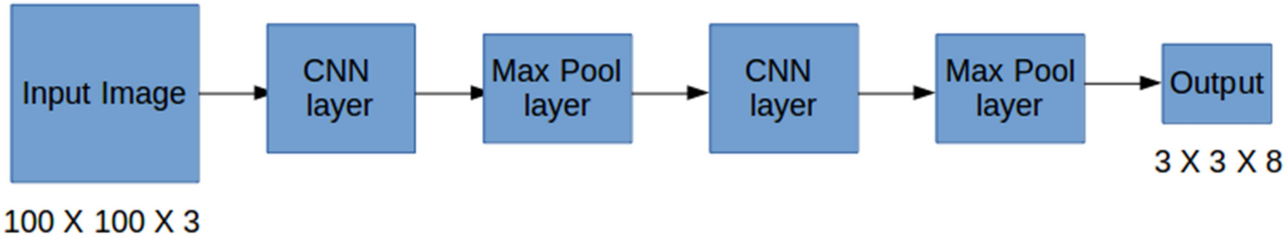

Figure 3.3 Model

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

Even if an object spans out to more than one grid, it will only be assigned to a single grid in which its mid-point is located. We can reduce the chances of multiple objects appearing in the same grid cell by increasing the more number of grids (19 X 19, for example)[7].

4.1.3 OpenCV

OpenCV is the huge open-source library for computer vision, machine learning, and image processing and now it plays a major role in real-time operation which is very important in today's systems. By using it, one can process images and videos to identify objects, faces, or even the handwriting of a human. Here we used open cv in the license plate recognition.

4.1.4 ORB Algorithm

ORB is basically a fusion of FAST key point detector and BRIEF descriptor with many modifications to enhance the performance. First it use FAST to find key points, then apply Harris corner measure to find top N points among them. It also use pyramid to produce multiscale-features and it draw the bounding boxes around the identified object.

Figure 4 Bounding boxes by ORB Algorithm

The above figure represents the working of ORB algorithm and shows the way the objects are detected and represented with the bounding boxes around them.

4.1.5 License Plate Recognition [8]

STEP1: Resize the image to required size and gray scale it.

```
img = cv2.resize(img, (620,480) )
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) #convert to grey scale
```

Resizing will help us from the major problems caused by the higher resolutions or too lower resolutions. Gray scaling is common in all image processing steps. This speeds up other following process as we no longer wanted to process the colours of objects so gray scaling speeds up the process with better accuracy.

STEP2: The images have lot of noise and unwanted objects and particles in the image which are useless for the process so we need to remove them for better accuracies. Normally using a bilateral filter (Blurring) will remove the unwanted details from an image. The code for the same is blurred

```
gray = cv2.bilateralFilter(gray, 13, 15, 15)
```

Syntax is destination_image = cv2.bilateralFilter(source_image, diameter of pixel, sigmaColor, sigmaSpace). You can increase the sigma color and sigma space from 15 to higher values to blur out more background information, but be careful that the useful part does not get blurred. This way we can avoid the program from concentrating on unwanted details later.

STEP3: The next step is very important that is edge detection and the following way is very interesting and popular way is to use the canny edge method from OpenCV. The line to do the same is shown below

```
edged = cv2.Canny(gray, 30, 200) #Perform Edge detection
```

The syntax will be *destination_image = cv2.Canny(source_image, thresholdValue 1, thresholdValue 2)*. The Threshold Vale 1 and Threshold Value 2 are the minimum and maximum threshold values. Only the edges that have an intensity gradient more than the minimum threshold value and less than the maximum threshold value will be displayed.

STEP 4: Now we can start looking for contours on our image

```
contours=cv2.findContours(edged.copy(),cv2.RETR_TREE
cv2.CHAIN_APPROX_SIMPLE)
contours = imutils.grab_contours(contours)
contours = sorted(contours,key=cv2.contourArea, reverse = True)[:10] screenCnt =
None
```

Once the counters have been detected we sort them from big to small and consider only the first 10 results ignoring the others. In our image the counter could be anything that has a closed surface but of all the obtained results the license plate number will also be there since it is also a closed surface.

To filter the license plate image among the obtained results, we will loop though all the results and check which has a rectangle shape contour with four sides and closed figure. Since a license plate would definitely be a rectangle four sided figure.

```
for c in cnts:
 # approximate the contour
 peri = cv2.arcLength(c, True)
 approx = cv2.approxPolyDP(c, 0.018 * peri, True)
# if our approximated contour has four points, then
 # we can assume that we have found our screen
 if len(approx) == 4:
```

```
screenCnt = approx
break
```

Once we have found the right counter we save it in a variable called *screenCnt* and then draw a rectangle box around it to make sure we have detected the license plate correctly.

Figure 5 License plate detected
In the above figure the license plate is detected from the input image and shows the 97% accuracy for the detection of number plate.

STEP5: The Final step is to read the license plate number from the segmented image with the characters on it.

```
#Read the number plate
text = pytesseract.image_to_string(Cropped, config='--psm 11')
print("the bike number is:",text)
```

```
Creating...frame.jpg

respond.status_code 201

[OrderedDict([('processing_time', 111.368), ('results', [OrderedDict([('box',

the bike number is AP27BB6346
```

Figure 6 Extraction of number
The above figure is the screenshot of the output where the license plate number from the figure 4 is extracted and given as output so that the number of the violator gets noted.

4.2 Use cases

4.2.1 Use case of helmet wear detection

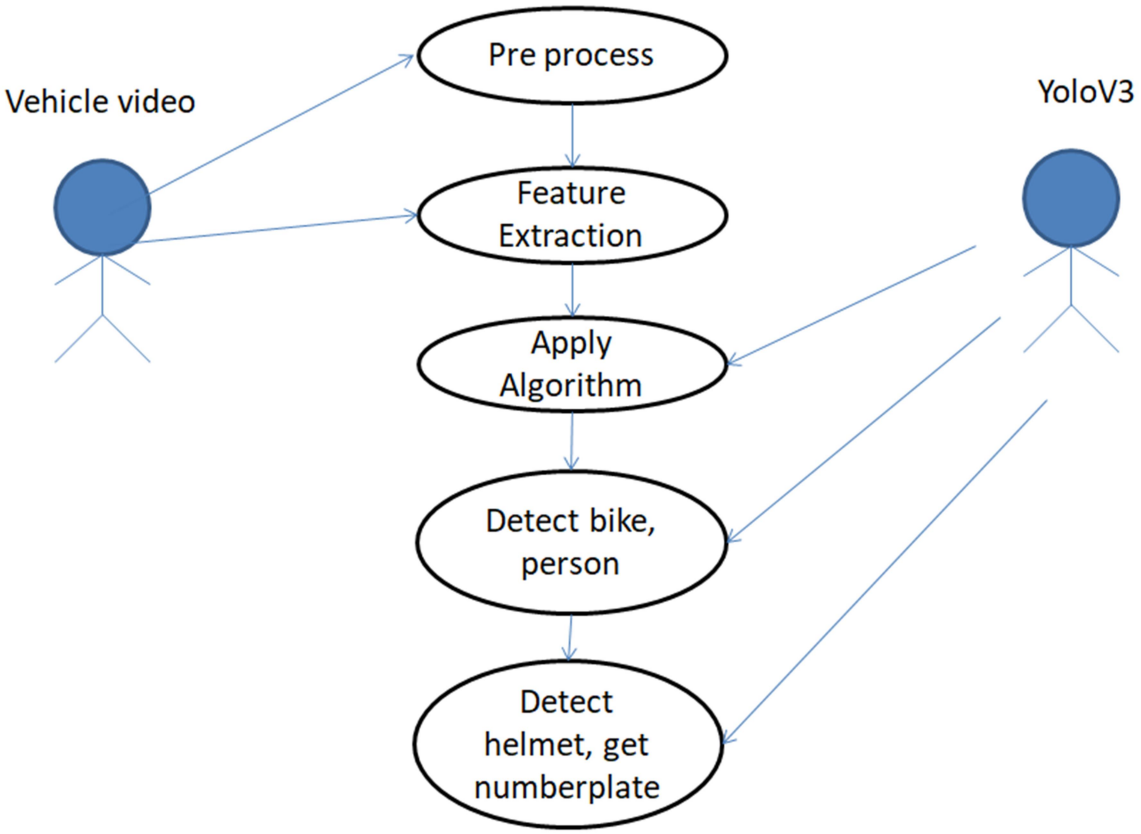

Figure 7 Use case of helmet wear detection

The use case diagram of helmet wear detection is that the video from the surveillance video is considered as input and first the background subtraction and preprocessing takes place and then the feature extraction of the video. Next by applying yolo to the video we can able to detect the trained objects i.e., bike and person and later on it checks whether the helmet is present or not and if the helmet is not present then the license plate number gets extracted.

4.2.2 Use case for pre-process

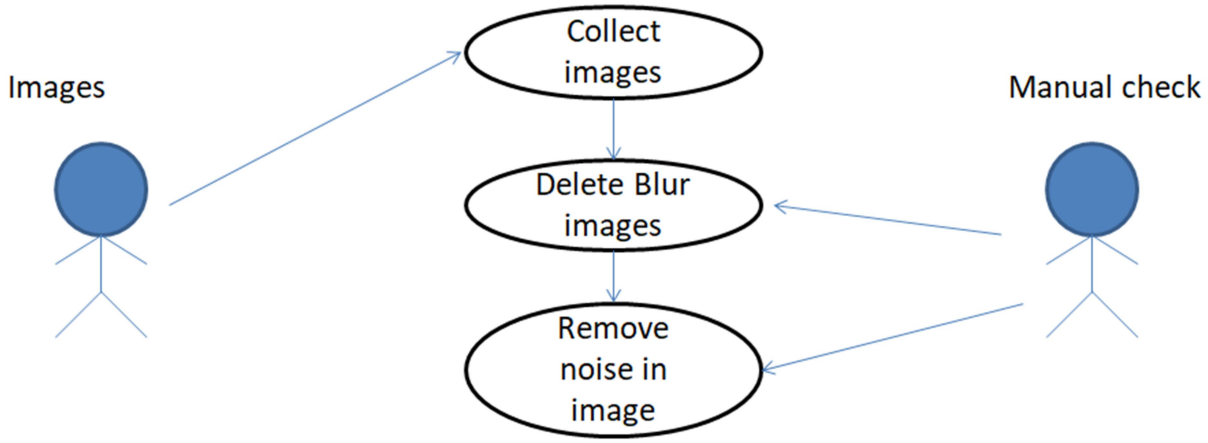

Figure 8 Use case for preprocessing

The use case for preprocessing represents the different set of images captured from the video will perform in iterations for the object detection so then it erase the blur images and remove noise in the image by turning it into greyscale.

4.2.3 Use case for bike detection

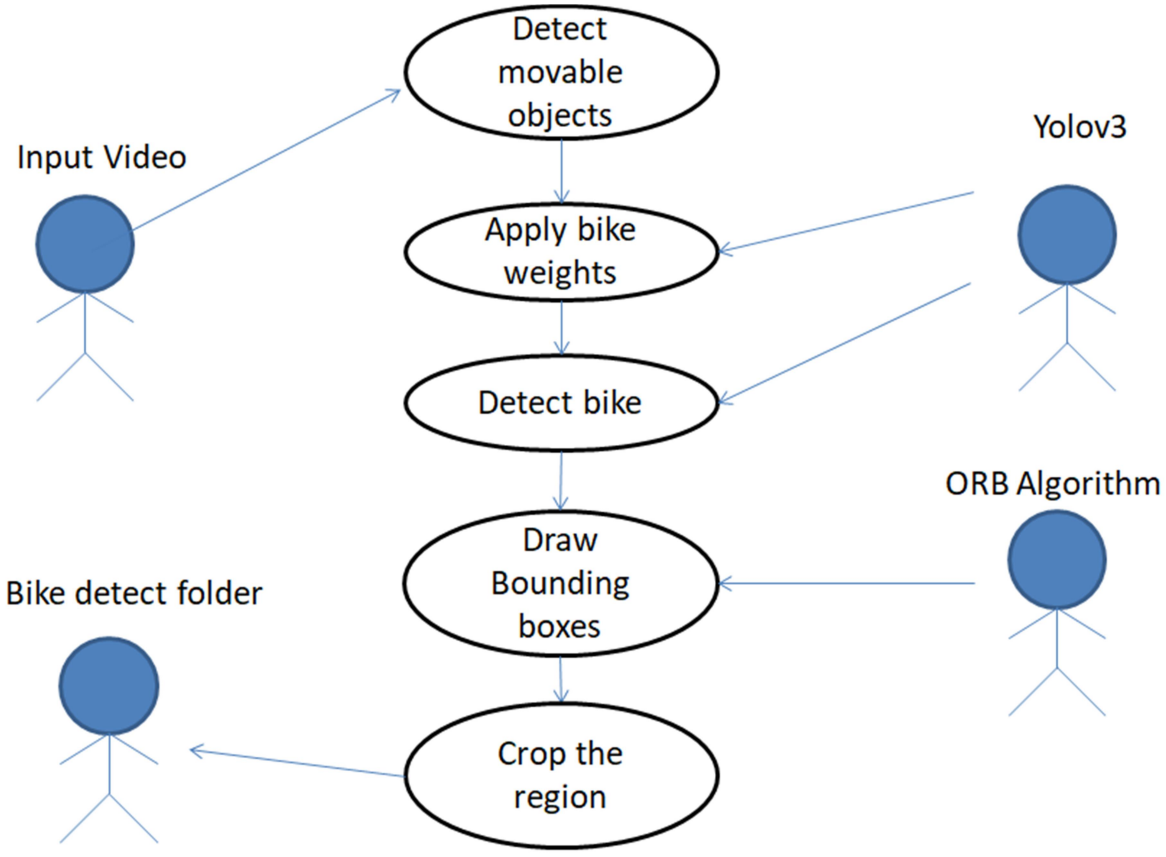

Figure 9 Use case for bike detection

The use case for bike detection is that the surveillance video is considered as input and all the movable objects get detected initially and as we trained yolo with bike weights the bike gets detected and to represent the object the ORB algorithm is used for the representation of bounding boxes around the object and the final image is captured and saved to the folder.

4.2.4 Use case for biker detection

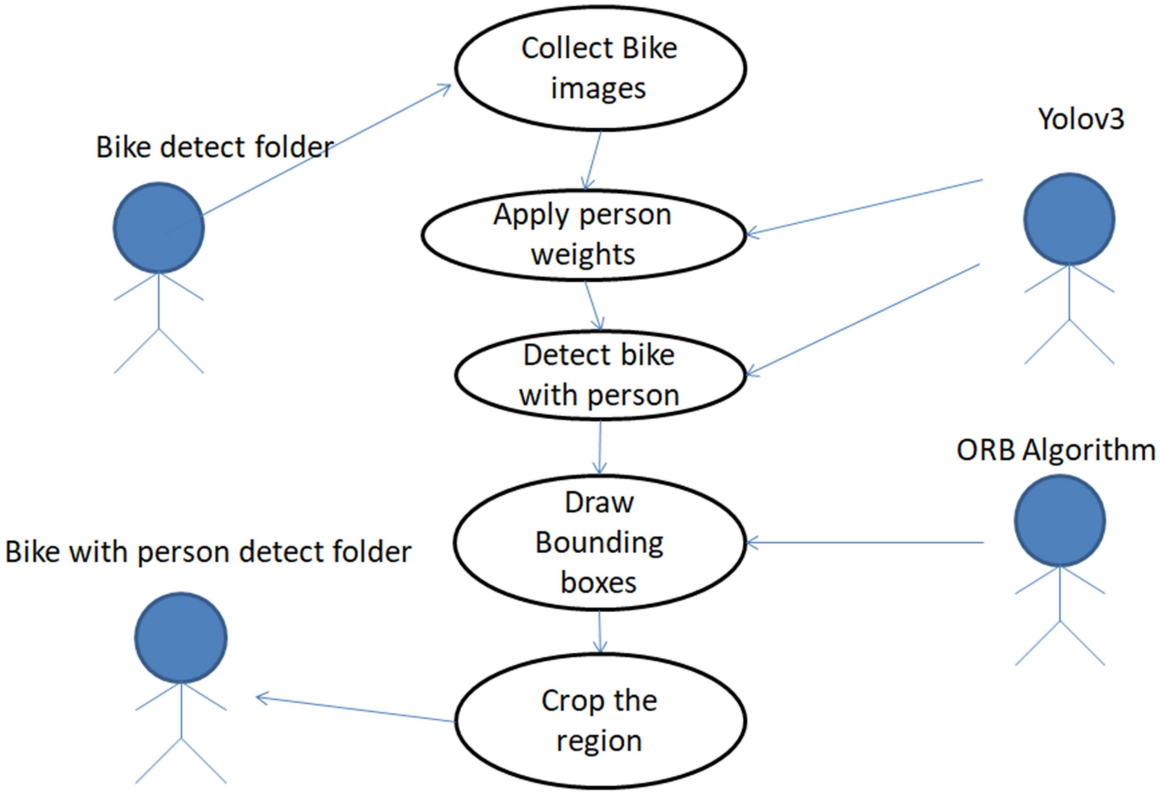

Figure 10 Use case for biker detection

The use case for biker detection is that the bikes image from the folder is considered as input and yolo applies weights of the person and if the person is located on bikes then the ORB algorithm draw the bounding box around the objects found by yolo and the region is taken and saved to a folder.

4.2.5 Use case for helmet or no helmet

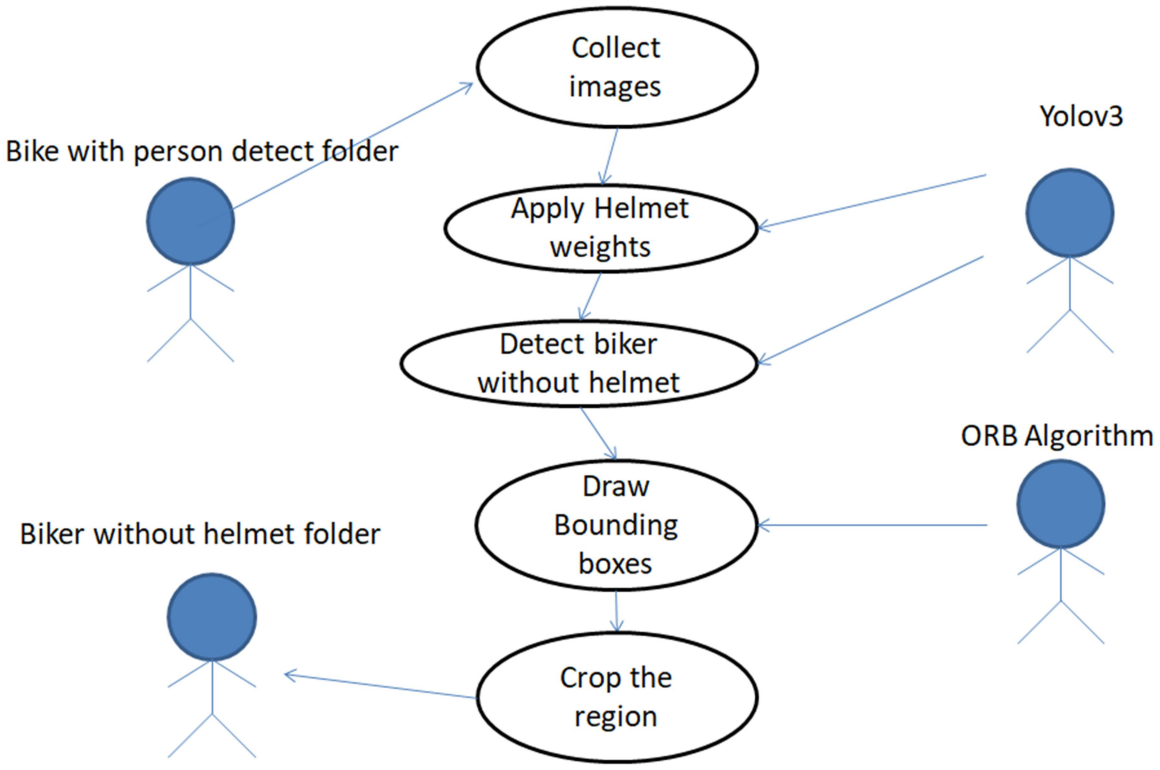

Figure 11 Use case for helmet detection

The use case for helmet detection is that the images from the folder is considered as input and the helmet is present at top 1/3 of the image so using the yolo weights the helmet are detected in the image and the biker without helmet, the ORB algorithm is responsible for the representation of bounding boxes around the objects detected and the images of bikers without helmet will be saved to folder.

4.2.6 Use case for number plate extraction

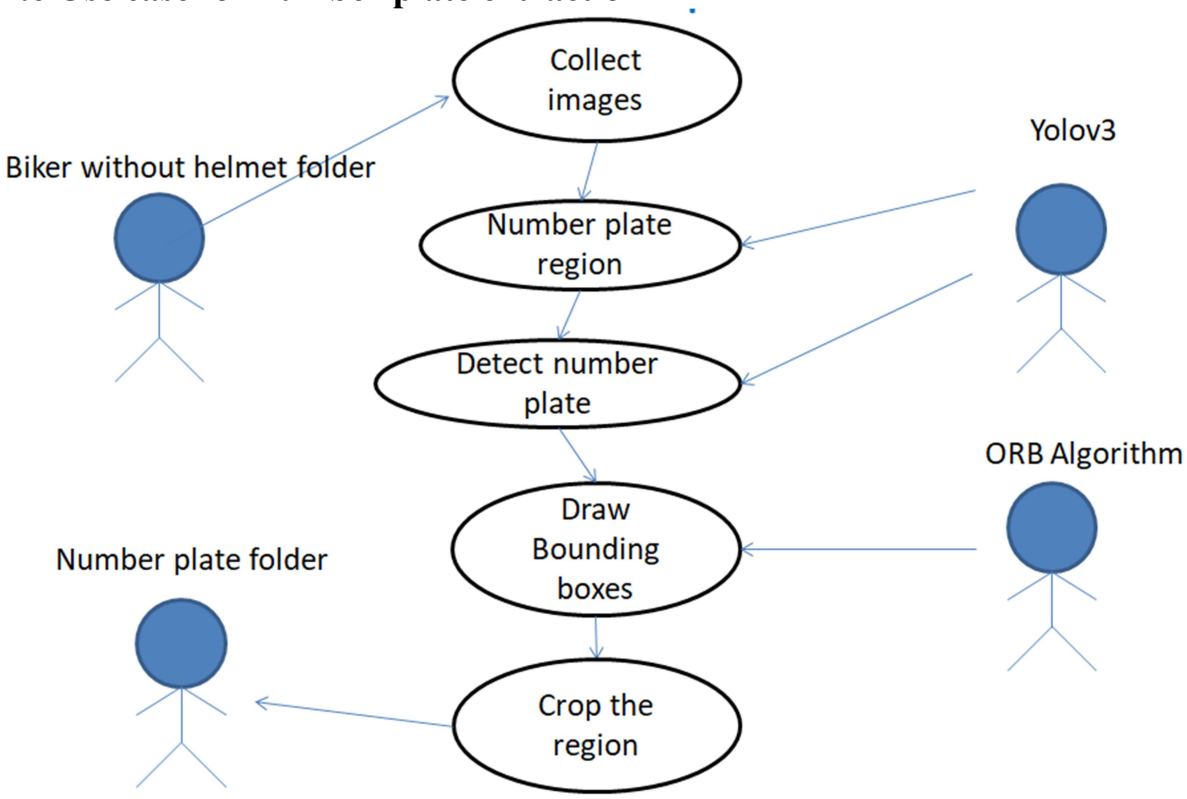

Figure 12 Use case for number plate extraction

The use case for number plate extraction is that the images from the folder is considered as input and the bike number plate will be detected by the yolo and bounding boxes were drawn by the ORB algorithm and the region was cropped and send to the license plate recognizer API by that the number will get extracted.

4.3 Test cases

4.3.1 Test case-1

Test Case Id	Tc1
Test case Name	Bike-Rider Detection
Input:	Surveillance video
Desc:	Step1: Input the video Step2: Detect moving Objects Step3: Detect the bike-rider
Expected output:	Detect the bike rider
Pass/Fail	Pass

Figure 13 Test case-1

The test case is bike rider detection, here we consider video as an input and it first detects the moving objects in the video and the test case is passed as it detects the bike-rider in my video represented in Figure 18 with accuracy 99%.

4.3.2 Test case-2

Test case Id	Tc2
Test case Name	Detect whether bike-rider wearing helmet
Input	Surveillance Video
Desc:	Step1: Input the video Step2: Detect moving Objects Step3: Detect the bike-rider Step4: Detect whether bike-rider wearing helmet or not.
Expected output	Bounding box showing Helmet
Pass/Fail	Pass

Figure 14 Test case-2

The test case is bike rider helmet detection, here we consider video as an input and it first detects the moving objects, then bike rider and detects whether the bike rider wearing a helmet or not in

the video and the test case is passed as it detects the bike-rider is not wearing a helmet in my video represented in Figure 18 with accuracy 99%.

4.3.3 Test case-3

Test case Id	Tc3
Test case Name	Detect Number plate
Input	Video
Desc:	Step1: Detect the number plate of bike.
Expected output	Bounding box showing the number plate
Pass/Fail	Pass

Figure 15 Test case – 3

The test case is number plate detection, here we consider video as an input and it first detects the moving objects, then bike rider and detects whether the bike rider wearing a helmet or not and if not the license plate of the violator is detected in the video and the test case is passed as it detects the number plate of a bike-rider not wearing a helmet in my video represented in Figure 18 with accuracy 96%.

4.3.4 Test case-4

Test case Id	Tc4
Test case Name	Extraction of number
Input	Result image stored
Desc:	Step1: Input the image Step2: Get number from plate recognition API
Expected output	Extract the number
Pass/Fail	Pass

Figure 16 Test case-4

The test case is Extraction of number from license plate of bike rider without helmet. The test case is passed as the number from the license plate is extracted shown in the figure 19.

4.4 Sequence Diagram

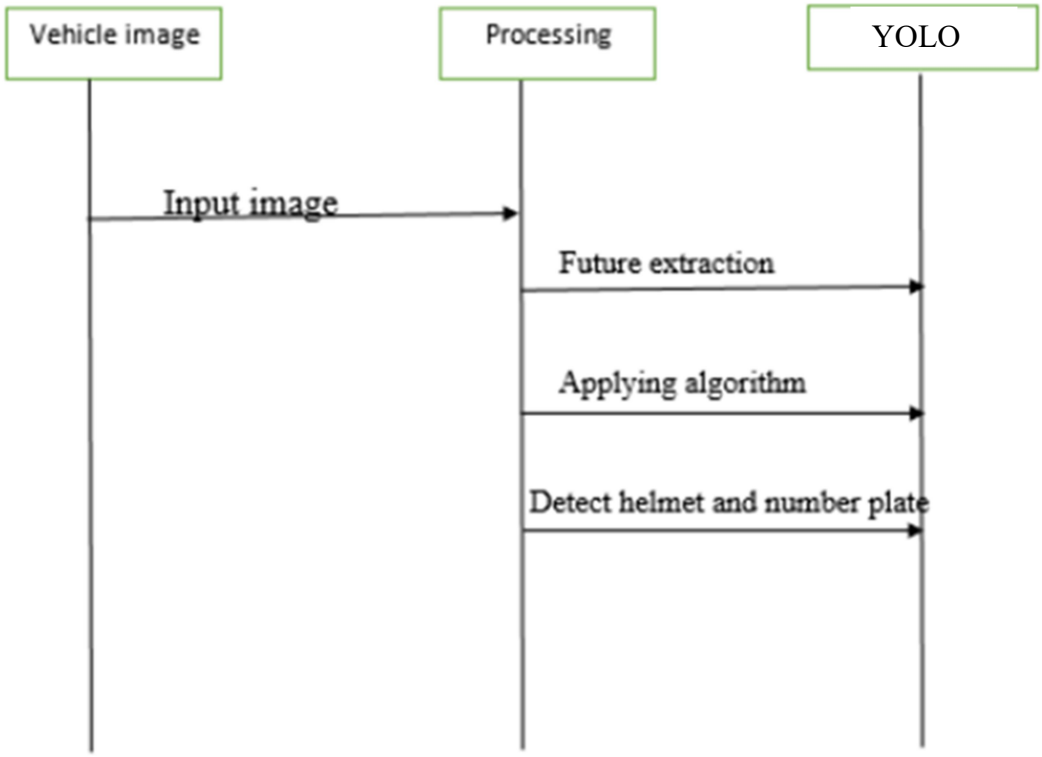

Figure 17 Sequence diagram

The figure 16 represents the sequence diagram of the helmet wear detection project where in the diagram the video and iterations of images are considered as input and the processing takes place like future extraction and YOLO is applied for the object detection and if the biker is not wearing a helmet then the number from the license plate is extracted.

CHAPTER 5

RESULT

Given in the below figure we can observe that the license number from the number plate got extracted when the bike rider is not wearing a helmet by this way the government is able to catch the violators and fine them, making them not to repeat this type of violations without any manual support. This project saves a lot of manual effort and time.

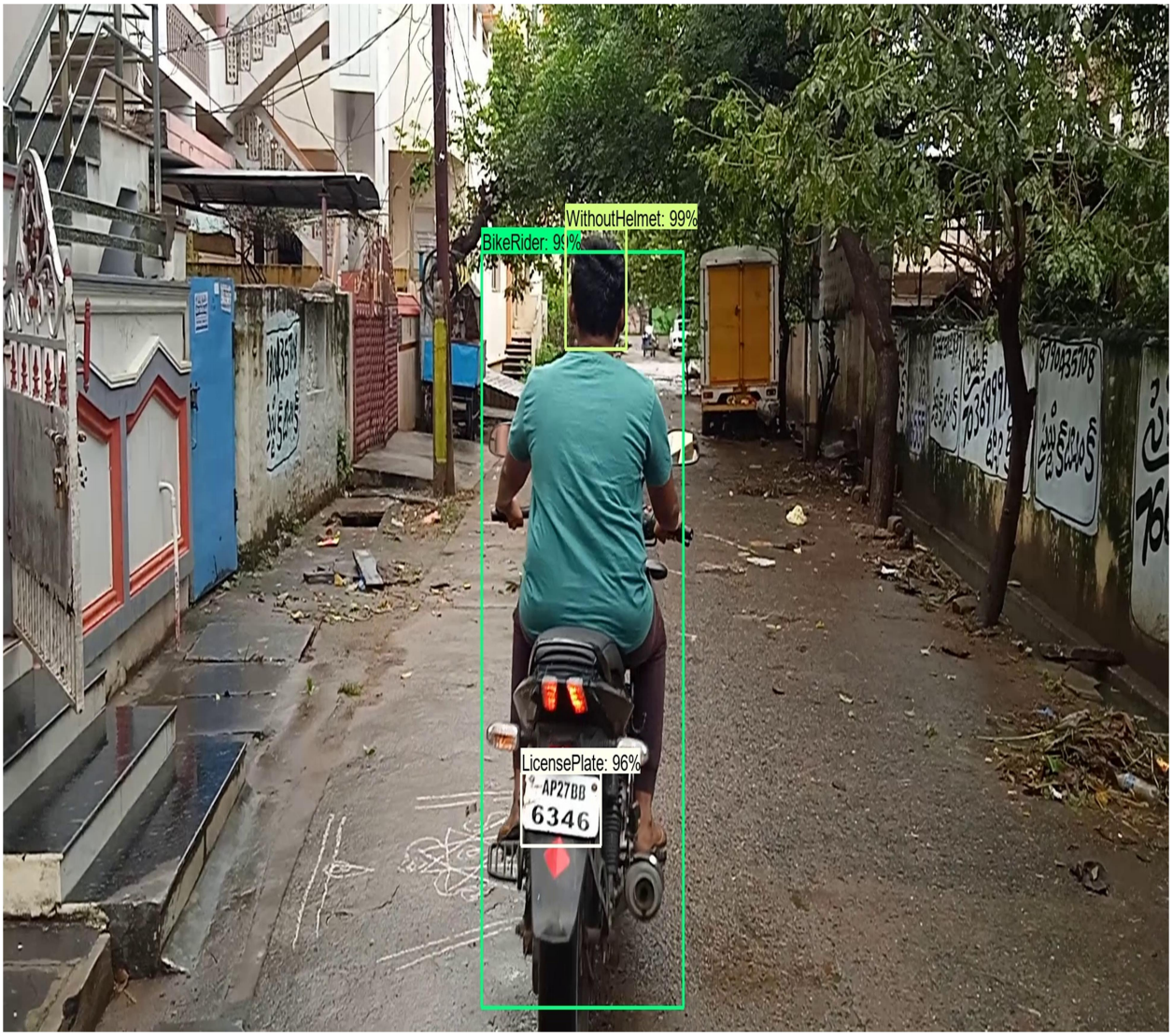

Figure 18 Output

In the figure 17 we can observe that the objects like Bike rider is detected with 99% accuracy and he is without helmet that is detected with 99% accuracy and the license plate of the rider is

detected with 96% accuracy and the number from the license plate is extracted as shown in figure 19.

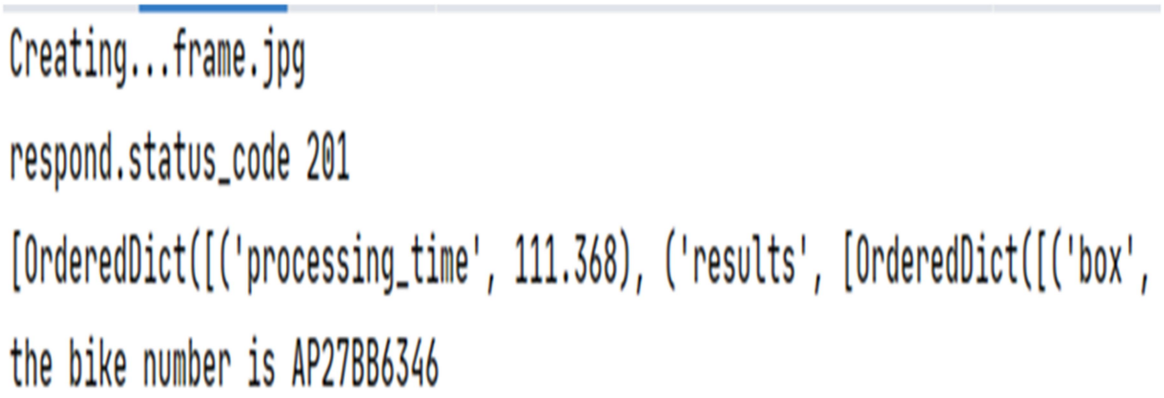

Figure 19 License plate number

The above figure represents the number extracted from the output image shown in figure 18.

CHAPTER 6

CONCLUSION AND FUTURE WORK

This project aims to give an idea about the number of traffic offenders in an area. It generates a data of all the bike rides driving without wearing a helmet. Use of Open and free technologies like tensor flow, opencv and Yolov3 makes the software relatively less expensive.

Lot can be done in this area. There is a large scope which could be ventured and new designs or system could be made to improve the conditions and efficiency of the detection and by using AI.

CHAPTER 7

APPENDIX

```
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"D:/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 = 'accurate.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, '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] > 90):
                    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 (d[4] > 80):
                    # name = 'frame.jpg'
                    # print('Creating...' + name)
                    # cv2.imwrite(name, image_np)
                    # number = platemain()
                    if len(number) not in [5, 6, 7]:
                        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 plate")
                        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
        for m in range(len(nmp)):
            data = {'Status': st[m],
                    'Number_Plate': nmp[m]
                    }
            print(data)

            m = json.dumps(data)
            with open("D:\helmet detection\data1.json", "w") as
f:
                f.write(m)
cv2.destroyAllWindows()
```

REFERENCES

[1] www.indiatoday.in/diu/story/two-wheeler-death-road-accidents-helmets-states-india-1602794-2019-09-24

[2] C. Vishnu, Dinesh Singh, C. Krishna Mohan and Sobhan Babu Visual Intelligence and Learning Group (VIGIL), Indian Institute of Technology Hyderabad. ”Detection of Motorcyclists without Helmet in Videos using Convolutional Neural Network”.

[3] J. Chiverton, “Helmet presence classification with motorcycle detection and tracking,” IET Intelligent Transport Systems (ITS), vol. 6, no. 3, pp. 259–269, 2012.

[4] R. Silva, K. Aires, T. Santos, K. Abdala, R. Veras, and A. Soares, “Automatic detection of motorcyclists without helmet,” in Proc. Latin American Computing Conf. (CLEI), Puerto Azul, Venezuela, 4–6 October 2013.

[5] R. E. Kalman, “A new approach to linear filtering and prediction problems,” Journal of Basic Engineering, vol. 82, no. 1, pp. 35–45, 1960.

[6] You Only Look Once: Unified, Real-Time Object Detection**,** Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi**.**

[7] www.analyticsvidhya.com/blog/2018/12/practical-guide-object-detection-yolo-framewor-python/

[8] License Plate Recognition using OpenCV Python by Praveen.

BIODATA

Photo

Name : VENKATA DURGA AVINASH BOBBALA
Mobile Number : 8618935214
E-mail : avinash.bobbala@vitap.ac.in
Permanaent Address : 35-1-47/4, Devudu cheruvu, Ongole

NOTE: Its **MANDATORY** for a student to attach all the PPT's, Sample Materials, Specification Sheets, Programming Codes and a 5-10 minutes demo Video of the Project Digitally In CD . Stick the Compact Disk (CD) in the final page of the Thesis after binding it.