TRAFFIC VIOLATION PROCTORING SYSTEM: HELMET AND TRIPLE RIDING DETECTION

A Project report submitted in partial fulfillment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

ELECTRONICS AND COMMUNICATION ENGINEERING

Submitted by

M.Jayasree (317126512147)   P.Sai Vamsi Dheeraj (317126512158)

S.Sindhura (317126512171)   K.Lokesh (318126512L32)

Under the guidance of

Mr. N. SRINIVASA NAIDU M.Tech, AMIETE (Ph.D)

Assistant Professor, Department of ECE

PRAGNANAM BRAHMA

ANITS

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

ANIL NEERUKONDA INSTITUTE OF TECHNOLOGY AND SCIENCES
(UGC AUTONOMOUS)
(*Permanently Affiliated to AU, Approved by AICTE and Accredited by NBA & NAAC with 'A' Grade*)
Sangivalasa, bheemili mandal, visakhapatnam dist.(A.P)
2020-2021

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

ANIL NEERUKONDA INSTITUTE OF TECHNOLOGY AND SCIENCES

(*Permanently Affiliated to AU, Approved by AICTE and Accredited by NBA & NAAC with 'A' Grade*)

Sangivalasa, Bheemili Mandal, Visakhapatnam dist. (A.P)

PRAGNANAM BRAHMA

ANITS

CERTIFICATE

This is to certify that the project report entitled **"TRAFFIC VIOLATION PROCTORING SYSTEM: Helmet and Triple Riding Detection"** submitted by **M. Jayasree (317126512147), P. Sai Vamsi Dheeraj (317126512158), S. Sindhura (318126512171), K. Lokesh (318126512L32)** in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology in Electronics & Communication Engineering** of Andhra University, Visakhapatnam is a record of bonafide work carried out under my guidance and supervision.

Project Guide

[illegible] 14/7/21

Mr. N. Srinivasa Naidu M. Tech,AMIETE (Ph.D)
Assistant Professor
Department of ECE, ANITS

Assistant Professor
Department of E.C.E.
Anil Neerukonda
Institute of Technology & Sciences
S[illegible] Visakhapatnam-531 162

Head of the Department

[illegible]

Dr. V. Rajya Lakshmi M.E, Ph.D., MHRM, MIEEE, MIMIET
Professor and HOD
Department of E.C.E, ANITS

Head of the Department
Department of E C E
Anil Neerukonda Institute of Technology & Sciences
Sangivalasa - 531 162

ACKNOWLEDGEMENT

We would like to express our deep gratitude to our project guide **Mr. N. Srinivasa Naidu** Assistant Professor, M.Tech, AMIETE(Ph.D), Department of Electronics and Communication Engineering, ANITS, for his/her guidance with unsurpassed knowledge and immense encouragement. We are grateful to **Dr. V. Rajyalakshmi**, Professor & Head of the Department, Electronics and Communication Engineering, for providing us with the required facilities for the completion of the project work.

We are very much thankful to the **Principal and Management, ANITS, Sangivalasa,** for their encouragement and cooperation to carry out this work.

We express our thanks to all **teaching faculty** of Department of ECE, whose suggestions during reviews helped us in accomplishment of our project. We would like to thank **all non-teaching staff** of the Department of ECE, ANITS for providing great assistance in accomplishment of our project.

We would like to thank our parents, friends, and classmates for their encouragement throughout our project period. At last but not the least, we thank everyone for supporting us directly or indirectly in completing this project successfully.

PROJECT STUDENTS

M.Jayasree(317126512147),
P.Sai Vamsi Dheeraj(317126512158),
S.Sindhura(317126512171),
K.Lokesh(318126512L32)

ABSTRACT

Violations in traffic laws are very common in a highly populated country like India. The accidents associated with these violations cause a huge loss to life and property. Since utilization of bikes is high, mishaps associated with bikes are additionally high contrasted with different vehicles. One of the main causes of these is not using motorcycle helmets. So we propose an approach called "TRAFFIC VIOLATION PROCTORING SYSTEM:HELMET AND TRIPLE RIDING DETECTION" using deep learning which automatically sends challan or send an SMS for individuals in case of identification of bicycle riders without headgear and who are triple riding utilizing surveillance videos in real-time. The proposed approach initially recognizes motorcycle riders utilizing background subtraction and object segmentation. At that point we utilize object classifier to classify violators.

Since wearing helmet is critical while driving, our main aim is to decrease the danger of injuries in case of accident. By detecting the motorcyclists without helmets, triple riding or other violations we can therefore increase their safety while on road. Hence by automating we reduce the workload on the traffic control team and will be able to share the evidence with the team efficiently to impose fines on violators.

CONTENTS

LIST OF SYMBOLS vi

LIST OF FIGURES vii

LIST OF TABLES viii

LIST OF ABBREVATIONS ix

CHAPTER 1 INTRODUCTION 01

1.1 Project Overview 02

1.2 Literature Survey 02

1.3 Problem Definition 04

CHAPTER 2 OVERVIEW OF OBJECT DETECTION 05

2.1 Existing Methods 05

2.1.1 Single Shot Multibox Detection The paper about SSD 05

2.1.2 Region Proposals (R-CNN, Fast R-CNN, Faster R-CNN) 07

2.1.3 Resnet 07

2.1.4 Yolov3 08

CHAPTER 3 OBJECT DETECTION WITH PYTHON AND IMAGEAI 12

3.1 Creating an Image Object Detection System 12

3.2 Tuning Image Object Detector 15

3.3 Creating a Video Object Detection System 17

3.4 Tuning Video Object Detection 20

3.5 Creating Video Object Detection from Webcam 22

3.6 Closing Remarks 23

CHAPTER 4 SYSTEM DESIGN 24

4.1 Procedure 24

4.2 Flowchart of Proposed System 24

4.3 System Requirements 24

4.3.1 Major Software's and Libraries Used 24

CHAPTER 5 SYSTEM IMPLEMENTATION 28

5.1 Model for Motorcycle Detection 28

5.2 Model for Helmet Detection 30

5.2.1 Procedure for training a YOLOV3 HELMET model 30

5.3 Number plate Detection 31

5.4 Automated E-Challan Generation 32

5.5 Triple Riding Detection 33

5.5.1 Proposed Model 33

5.5.2 Traffic rule violation recognition system 34

5.5.3 YOLOv3 (You Only Look Once) 35

5.5.4 COCO dataset 36

5.5.5 Training and testing data 36

5.5.6 License plate detection 36

5.5.7 E-Challan for Triple Riding 37

CHAPTER 6 WORKFLOW & SYSTEM TESTING 38

6.1 Account Setup Gdrive 38

6.2 Detection of Motorcycle 39

6.3 Detection of Helmet 42

6.4 Detection of Triple Riding 44

6.5 Detection of Number Plate 45

6.6 E-Challan Generation 47

6.7 Reliability 50

FUTURE WORK 51

CONCLUSION 52

REFERENCES 53

LIST OF SYMBOLS

Bw	Bounding box width
Bh	Bounding box height
Bx	Position of the box at x
By	Position of the box at y
Pc	Probability that box contains the detectable object

LIST OF FIGURES

Figure no	Title	Page no
Fig. 2.1	Retina Net Detector Architecture.	08
Fig. 2.2	Overview of YOLOV3 algorithm.	09
Fig. 2.3	YoloV3 Architecture	09
Fig. 2.4	Bounding box prediction.	10
Fig. 2.5	Algorithm comparison	11
Fig. 3.1	Input image without detection and output image indicating detection.	15
Fig. 3.2	Detection of class objects with their accuracies.	17
Fig. 3.3	Complete output of the detected objects with greater accuracy	19
Fig. 4.1	Flowchart of the System	24
Fig. 5.1	Flowchart for Helmet Model	30
Fig. 5.2	Helmet dataset	31
Fig. 5.3	Working of Web API	32
Fig. 5.4	Proposed Model of triple riding detection	34
Fig. 5.5	Test image for triple riding.	35
Fig. 5.6	Test image for non-triple riding detection.	36

Fig. 5.7	License Plate detection	37
Fig. 6.1	Google Drive Login	38
Fig. 6.2	Importing libraries	39
Fig. 6.3	(a) and (b) Code for Motorcycle Detection	39
Fig. 6.4	(a) Frame-1 Case 1 (b) Frame-2 Case 2	40
Fig. 6.5	Helmet ROI cropping.	41
Fig. 6.6	Code for Helmet Detection	42
Fig. 6.7	Helmet yolov3 prediction	43
Fig. 6.8	Code for Triple Riding Detection.	44
Fig. 6.9	Output for Triple Riding Detection	45
Fig. 6.10	Code for Number Plate Recognition	45
Fig. 6.11	Database created in Spreadsheet	46
Fig. 6.12	Number plate recognition	46
Fig. 6.13	Extraction of details using Google Sheets	47
Fig. 6.14	Code for E-Challan Generation	48
Fig. 6.15	Challan for No Helmet Violation.	49
Fig. 6.16	Challan for Triple Riding Violation	49

LIST OF TABLES

Table no	Title	Page no
Table 5.1	COCO dataset objects	29
Table 5.2	ASP.NET Web API Versions	32
Table 6.1	Analysis of the model	50

LIST OF ABBREVATIONS

CNN	convolutional Neural Network
R-CNN	region based- convolutional Neural Network
LBP	Local binary pattern
HoG	Histogram of oriented gradients
HaaR	Haar casacade
ML	Machine Learning
ResNet	Residual Neural Network
COCO	Common objects in context
BGR	Blue Green Red
RGB	Red Green Blue
Collab	Collaboratory
API	Application Programming Interface
HTML	Hypertext markup language

CHAPTER 1

INTRODUCTION

All over the world around 1.35 million lives are lost each year, 50 million people are getting injured due to road accidents, according to a report titled " The Global status Revised Manuscript Received on December 05, 2019 report on road safety 2018" released by world health organization. It is very hard to imagine that this burden is unevenly borne by motorcyclists, cyclists, and pedestrians. This report noted that a comprehensive action plan must be set up in order to save lives.

Two-wheeler is a very popular mode of transportation in almost every country. However, there is a high risk involved because of less protection. When a two-wheeler meets with an accident, due of sudden deceleration, the rider is thrown away from the vehicle. If head strikes any object, motion of the head becomes zero, but with its own mass brain continues to be in motion until the object hits inner part of the skull. Sometimes this type of head injury may be fatal in nature. In such times helmet acts as life saviour. Helmet reduces the chances of skull getting decelerated, hence sets the motion of the head to almost zero. Cushion inside the helmet absorbs the impact of collision and as time passes head comes to a halt. It also spreads the impact to a larger area, thus safeguarding the head from severe injuries. More importantly it acts as a mechanical barrier between head and object to which the rider came into contact. Injuries can be minimized if a good quality full helmet is used. Traffic rules are there to bring a sense of discipline, so that the risk of deaths and injuries can be minimized significantly. However strict adherence to these laws is absent. Hence efficient and feasible techniques must be created to overcome these problems. To reduce the involved risk, it is highly desirable for bike-riders to use helmet. Worrying fact is that India ranks in top as far as road crash deaths are considered. Rapid urbanization, avoiding helmets, seat belts and other safety measures while driving are some of the reasons behind this trend according to analysis done by experts. In 2015 India signed Brasilia Declaration on Road Safety, where India committed to reduce road crash deaths to 50 percent by 2020.

Observing the usefulness of helmet, Governments have made it a punishable offense to ride a bike without helmet and have adopted manual strategies to catch the violators. However, the existing video surveillance-based methods are passive and require significant human assistance. In general, such systems are infeasible due to involvement of humans, whose efficiency decreases over long duration.

Automation of this process is highly desirable for reliable and robust monitoring of these violations as well as it also significantly reduces the amount of human resources needed.

Recent research has successfully done this work based on CNN, R-CNN, LBP, HoG, HaaR features, etc. But these works are limited with respect to efficiency, accuracy or the speed with which object detection and classification is done.

1.1 Project Overview:

In this Project Work, a Non-Helmet Rider detection system is built which attempts to satisfy the automation of detecting the traffic violation of not wearing helmet and extracting the vehicles' license plate number. The main principle involved is Object Detection using Deep Learning at three levels. The objects detected are person, motorcycle at first level using YOLOv3, helmet at second level using YOLOv3, License plate at the last level using Web API. Then the license plate registration number is extracted using Web Automation. Hence a database will be available for analysis for the police authority.

1.2 Literature Survey

In various fields, there is a necessity to detect the target object and track them effectively while handling occlusions and other included complexities. Many researchers (Almeida and Guting 2004, Hsiao-Ping Tsai 2011, Nicolas Papadakis and Aure lie Bugeau 2010) attempted for various approaches in object tracking. The nature of the techniques largely depends on the application domain. Some of the research works which made the evolution to proposed work in the field of object tracking are depicted as follows

Until very recently, most of the methods used for object detection and object classification used methods such as Haar, HOG, local binary patterns (LBP), the scale invariant feature transform (SIFT), or speeded up robust features (SURF) for feature extraction and then support vector machines (SVM), random forests, or AdaBoost for the classifier. Silva et al. [1] use methods such as histograms of oriented gradient (HOG), LBP, and the wavelet transform (WT) for feature extraction for classifying motorcyclists with helmets and without helmets. They use multiple combinations of the base features such as HOG+LBP+WT, obtaining seven possible feature sets. In [5], K. Dahiya et al. came up with helmet detection from surveillance videos where they used an SVM classifier for classifying between motorcyclist and non-motorcyclist and another SVM classifier for classifying between helmet and without helmet. For both classifiers, three widely used features - HOG, SIFT and LBP - were implemented and the performance of each was compared with that of other two features. They concluded that HOG descriptor helped in achieving the best performance.

In [6], C. Vishnu et al. proposed an approach using Convolutional Neural Networks (CNNs) for classification. In recent years, CNNs performing both automatic feature extraction and classification have outperformed previously dominant methods in many problems. Advances in graphical processing units (GPUs), along with the availability of more training data for neural networks to learn, have recently enabled unprecedented accuracy in the fields of machine vision, natural language processing, and speech recognition. Nowadays, all state-of-the-art methods for object classification, object detection, character classification, and object segmentation are based on CNNs. See for example the methods used in the ImageNet large scale visual recognition challenge [2].

Li and Shen [3] use a deep convolutional neural network and long-short term memory (LSTM) for the license plate recognition and character extraction process. They use two methods for segmentation and recognition. [4] have shown the use of CNNs for text detection and recognition provides significant improvement over existing methods.

The YOLOv3 algorithm is capable of accurate object detection (traffic participants) with near real- time performance (~ 25 fps on HD images) in the variety of the driving conditions (bright and overcast sky, snow on the streets, and driving during the night).

YOLO v3 algorithm consists of fully CNN [7] and an algorithm for post-processing outputs from neural network. CNNs are special architecture of neural networks suitable for processing grid-like data topology. The distinctive feature of CNNs which bears importance in object detection is parameter sharing. Unlike feedforward neural networks, where each weight parameter is used once, in CNN architecture each member of the kernel is used at every position of the input, which means learning one set of parameters for every location instead a separate set of parameters.

The YOLOv3 AP does indicate a trade-off between speed and accuracy for using YOLO when compared to RetinaNet, since RetinaNet training time is greater than YOLOv3. The accuracy of detecting objects with YOLOv3 can be made equal to the accuracy when using RetinaNet by having a larger dataset, which makes it an ideal option for models that can be trained with large datasets. An example of this would be common detection models like traffic detection, where plenty of data can be used to train the model since the number of images of different vehicles is plentiful. YOLOv3 may not be ideal to use with niche models where large datasets can be hard to obtain.

1.3 Problem Definition:

Road safety is the most important aspect of this automobile driven technological world. Considering the number of people taking road transport as the means to reach their destination, the number of people reaching the heavens instead of their safe home, is increasing day-to-day. The irresponsible driving of the two-wheelers or the heavy speeding of the four-wheelers is the major reason for the occurring accidents. These irresponsible drivers are making it hard for the drivers that follow the traffic rules. The current increase in the fine/challan system might control these irresponsible drivers to an extent, but this is not a permanent solution that we can rely on.

Existing system monitors the traffic violations primarily through CCTV recordings, where the traffic police must investigate the frame where the traffic violation is happening, zoom into the license plate in case rider is not wearing helmet. But this requires lot of manpower and time as the traffic violations frequently and the number of people using motorcycles is increasing day-by-day. What if there is a system, which would automatically look for traffic violation of not wearing helmet while riding motorcycle and if so, would automatically extract the vehicles’ license plate number.

CHAPTER 2

OVERVIEW OF OBJECT DETECTION

Object detection is a computer technology related to computer vision and image processing that deals with detecting instances of semantic objects of a certain class (such as humans, buildings, or cars) in digital images and videos. Well researched domains of object detection include face detection and pedestrian detection. Object detection has applications in many areas of computer vision, including image retrieval and video surveillance. It is widely used in computer vision task such as face detection, face recognition, video object co-segmentation. It is also used in tracking objects, for example tracking a ball during a football match, tracking movement of a cricket bat, tracking a person in a video.

Every object class has its own special features that helps in classifying the class – for example all circles are round. Object class detection uses these special features. For example, when looking for circles, objects that are at a distance from a point (i.e. the centre) are sought. Similarly, when looking for squares, objects that are perpendicular at corners and have equal side lengths are needed. A similar approach is used for face identification where eyes, nose, and lips can be found and features like skin Colour and Distance Between Eyes Can Be Found.

Object classification systems are used by Artificial Intelligence (AI) programs to perceive specific objects in a class as subjects of interest. The systems sort objects in images into groups where objects with similar characteristics are placed together, while others are neglected unless programmed to do otherwise.

2.1 Existing Methods

Methods for object detection generally fall into either machine learning-based approaches or deep learning-based approaches. For Machine Learning approaches, it becomes necessary to first define features using one of the methods below, then using a technique such as support vector machine (SVM) to do the classification. On the other hand, deep learning techniques that can do end to-end object detection without specifically defining features and are typically based on convolutional neural networks (CNN).

Deep Learning approach:

2.1.1 Single Shot Multibox Detection The paper about SSD:

Single Shot MultiBox Detector (by C. Szegedy et al.) was released at the end of November 2016 and reached new records in terms of performance and precision for object detection tasks, scoring over 74% mAP (mean Average Precision) at 59 frames per second on standard datasets such as PascalVOC and COCO. To better understand SSD, let's start by explaining where the name of this architecture comes from:

• **Single Shot:** this means that the tasks of object localization and classification are done in a single forward pass of the network

• **MultiBox:** this is the name of a technique for bounding box regression developed by Szegedy et al. (we will briefly cover it shortly)

• **Detector:** The network is an object detector that also classifies those detected objects The SSD object detection composes of 2 parts:
• Extract feature maps.

• Apply convolution filters to detect objects. Modified from SSD:

Single Shot MultiBox Detector SSD uses VGG16 to extract feature maps. Then it detects objects using the Conv4_3 layer. For illustration, we draw the Conv4_3 to be 8 × 8 spatially (it should be 38 × 38). For each cell (also called location), it makes 4 object predictions. Each prediction composes of a boundary box and 21 scores for each class (one extra class for no object), and we pick the highest score as the class for the bounded object. Conv4_3 makes a total of 38 × 38 × 4 predictions: four predictions per cell 10 regardless of the depth of the feature maps. As expected, many predictions contain no object. SSD reserves a class "0" to indicate it has no objects. Each prediction includes a boundary box and 21 scores for 21 classes (one class for no object).

Convolutional predictors for object detection:

SSD does not use a delegated region proposal network. Instead, it resolves to a very simple method. It computes both the location and class scores using small convolution filters. After extracting the feature maps, SSD applies 3 × 3 convolution filters for each cell to make predictions. (These filters compute the results just like the regular CNN filters.) Each filter outputs 25 channels: 21 scores for each class plus one boundary box. Apply a 3x3 convolution filter to make a prediction for the location and the class. 11 For example, in Conv4_3, we apply four 3 × 3 filters to map 512 input channels to 25 output channels.

2.1.2 Region Proposals (R-CNN, Fast R-CNN, Faster R-CNN):

In this, algorithms try to draw a bounding box around the object of interest to locate it within the image. Also, you might not necessarily draw just one bounding box in an object detection case, there could be many bounding boxes representing different objects of interest within the image and you would not know how many beforehand. The major reason why you cannot proceed with this problem by building a standard convolutional network followed by a fully connected layer is that, the length of the output layer is variable — not constant, this is because the number of occurrences of the objects of interest is not fixed. A naive approach to solve this problem would be to take different regions of interest from the image and use a CNN to classify the presence of the object within that region. The problem with this approach is that the objects of interest might have different spatial locations within the image and different aspect ratios. Hence, you would have to select a huge number of regions and this could computationally blow up. Therefore, algorithms like R-CNN, YOLO etc have been developed to find these occurrences and find them fast.

2.1.3 Resnet:

To train the network model in a more effective manner, we herein adopt the same strategy as that used for DSSD (the performance of the residual network is better than that of the VGG network). The goal is to improve accuracy. However, the first implemented for the modification was the replacement of the VGG network which is used in the original SSD with ResNet. We will also add a series of convolution feature layers at the end of the underlying network. These feature layers will gradually be reduced in size that allowed prediction of the detection results on multiple scales. When the input size is given as 300 and 320, although the ResNet–101 layer is deeper than the VGG–16 layer, it is experimentally known that it replaces the SSD's underlying convolution network with a residual network, and it does not improve its accuracy but rather decreases it.

RetinaNet, a one-stage detector, by using focal loss, lower loss is contributed by "easy" negative samples so that the loss is focusing on "hard" samples, which improves the prediction accuracy. With ResNet+FPN as backbone for feature extraction, plus two task-specific subnetworks for classification and bounding box regression, forming the RetinaNet, which achieves state-of-the-art performance, outperforms Faster R-CNN, the well-known two-stage detectors**.** ResNet is used for deep feature extraction.

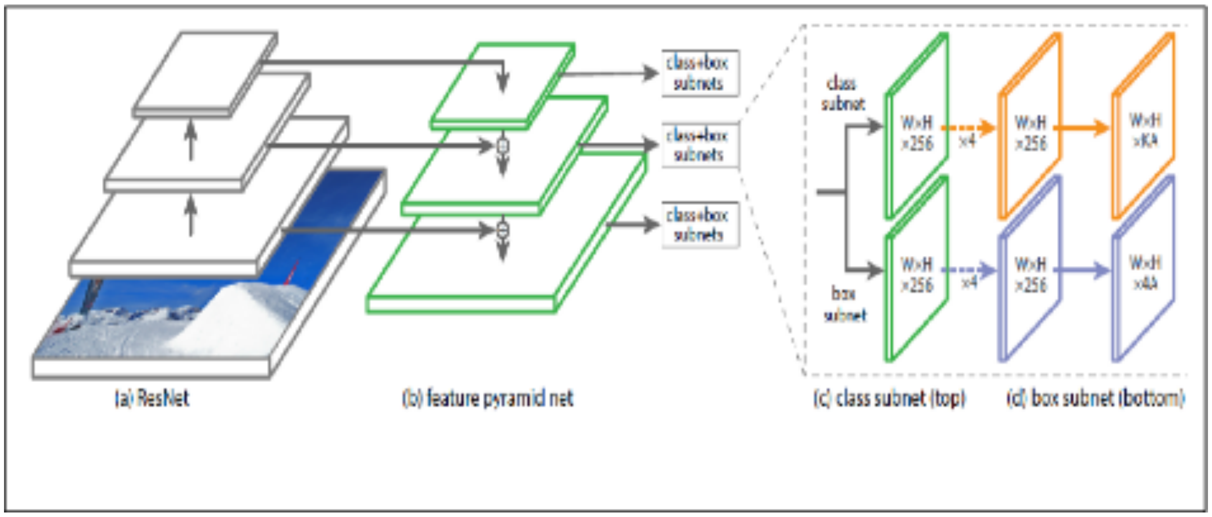


Figure 2.1. Retina Net Detector Architecture

2.1.4 Yolov3:

YOLOv3 (You Only Look Once, Version 3) is a real-time object detection algorithm that identifies specific objects in videos, live feeds, or images. Versions 1-3 of YOLO were created by Joseph Redmon and Ali Farhadi. The first version of YOLO was created in 2016, and version 3, which is discussed extensively in this article, was made two years later in 2018. YOLO is implemented using the Keras or OpenCV deep learning libraries.

YOLO v3 algorithm consists of fully CNN and an algorithm for post-processing outputs from neural network. CNNs are special architecture of neural networks suitable for processing grid-like data topology. The distinctive feature of CNNs which bears importance in object detection is parameter sharing. Unlike feedforward neural networks, where each weight parameter is used once, in CNN architecture each member of the kernel is used at every position of the input, which means learning one set of parameters for every location instead a separate set of parameters. This feature plays important role in capturing whole scene on the road.

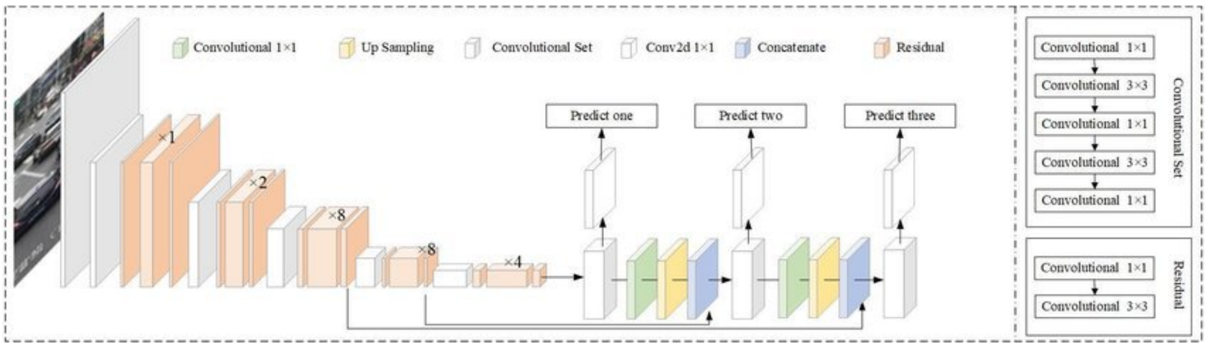


Figure 2.2. Overview of YOLOV3 algorithm.

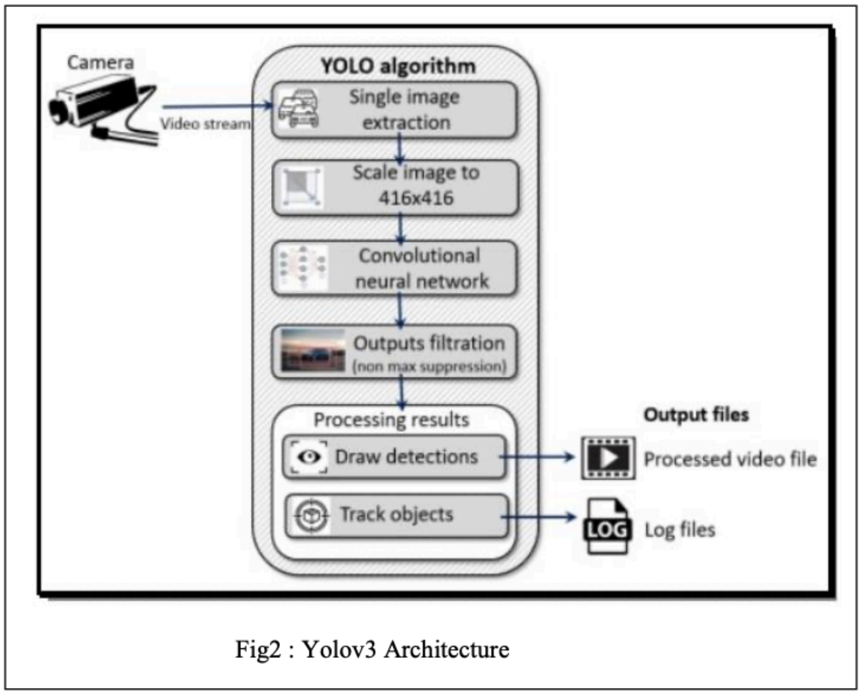


Figure 2.3. YoloV3 Architecture

This algorithm starts with extraction single image from video stream, in a next step extracted image is resized and that represent input to Yolo network. YOLO v3 neural network consist of 106 layers. Besides using convolutional layers, its architecture also contains residual layers, up sampling layers, and skip (shortcut) connections.

CNN takes an image as an input and returns tensor (see Fig 3) which represents:

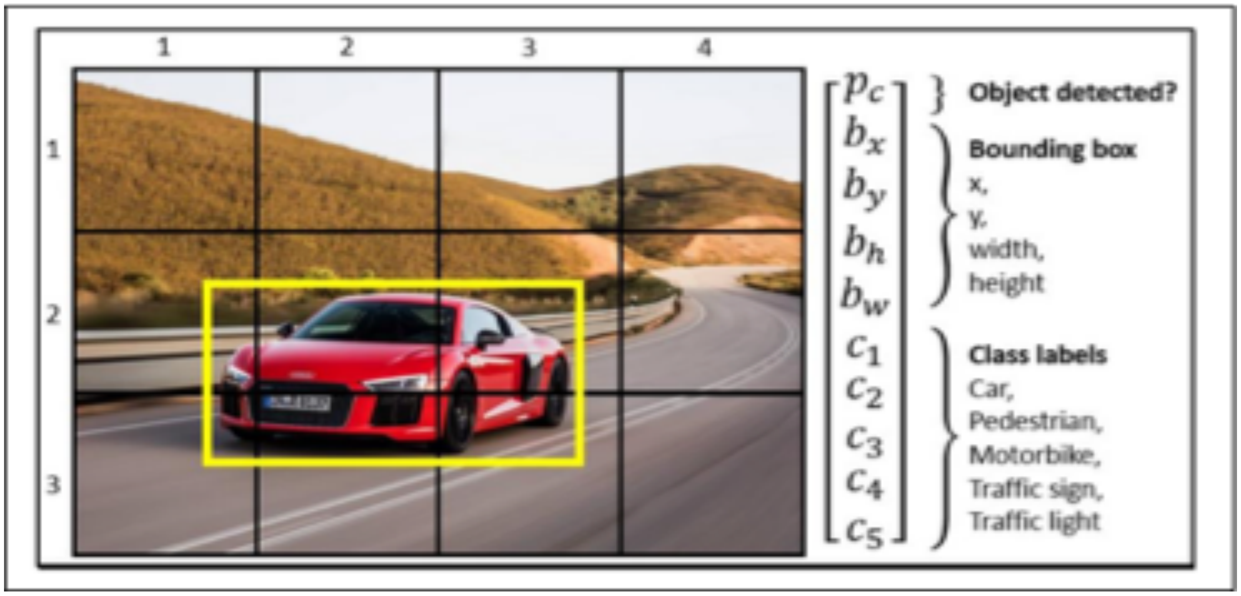


Figure 2.4. Bounding box prediction.

Coordinates and positions of predicted bounding boxes which should contain objects, A probability that each bounding box contains object,

Probabilities that each object inside its bounding box belongs to a specific class.

The detection is done on the three separate layers. Object detection done at 3 different scales addresses the issue of older YOLO neural network architectures, the detection of the small objects. Output tensors from those detection layers have the same widths and heights as their inputs, but depth is defined as: the number of bounding box properties such as width (bw), height (bh), x and y position of the box (bx, by) inside the image, 1 is the probability that box contains the detectable object (pc) and class probabilities for each of the classes (c1, c2, ..., c5).

That sum is multiplied by 3, because each of the cells inside the grid can predict 3 bounding boxes. As the output from the network, we get 10 647 bounding box predictions.

This network has an ability to simultaneously detect multiple objects on the single input image. Features are learned during the network training process when the network analyses the whole input image and does the predictions. In that way, the network has knowledge about the whole scenery and objects environment, which helps the network to perform better and achieve higher precision results comparing to the methods which use the sliding window approach. The concept of breaking down the images to grid cells is unique in YOLO, as compared to other object detection solutions. Predictions whose pc is lower than

0.5 are ignored and that way, most of the false predictions are filtered out. Remaining bounding boxes are usually prediction of the same object inside the image. They are filtered out using the non max suppression algorithm.

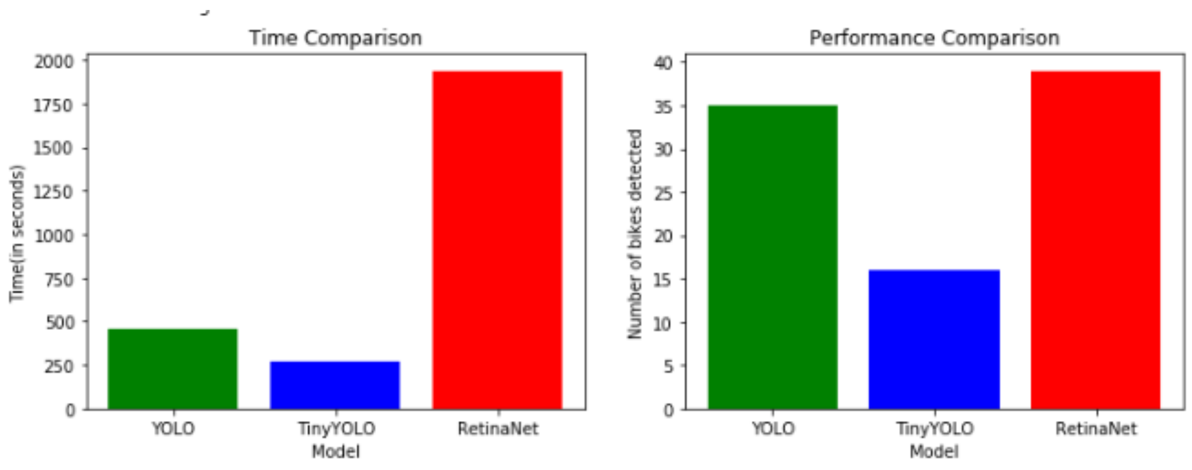


Figure 2.5. Algorithm comparison

CHAPTER 3

OBJECT DETECTION WITH PYTHON AND IMAGEAI

ImageAI is a Python library to enable ML practitioners to build an object detection system with only a few lines of code.

Before we start, we need to install some of the dependencies that we will need to run ImageAI properly. These dependencies are:

Numpy

pip install numpy==1.16.1

TensorFlow

pip install tensorflow==1.14.0

TensorFlow GPU

pip install tensorflow-gpu==1.14.0

Keras

pip install keras==2.2.4

OpenCV

pip install opencv-python

After installing all of those libraries, then we can start to install ImageAI library by typing the following command in your prompt:

pip install imageai –upgrade

Next, we are ready to build our object detection system for image and for video.

3.1 Creating an Image Object Detection System

To start creating an image object detection system, first let's import the libraries that we're going

to use and also set our current working directory.

```
from imageai.Detection

importObjectDetection
import oscurrent_directory = os.getcwd()
```

As we want to implement an object detection in an easy and quick way, we will use a pretrained model specific for object detection that has been trained on COCO dataset.

There are three different pretrained models that you can choose with ImageAI: RetinaNet, YOLOv3, and tinyYOLOv3. You can download the model of your choice.

After you've downloaded the model, place the h5 file in the same directory as your Python script. Your working directory should now has the following structure.

```
root_folder/
  python_script.py
  pretrain_model.h5
```

Back to our Python script, we now can instantiate the ObjectDetection class that we have imported before. Depending on the model that you have downloaded before, we need to call a proper method from ObjectDetection class. Below is the code implementation for that.

```
detector = ObjectDetection()# if you use RetinaNet model, call the following method
detector.setModelTypeAsRetinaNet()# if you use YOLOv3 model, call the following method
detector.setModelTypeAsYOLOv3()# if you use tinyYOLOv3 model, call the following
method
detector.setModelTypeAsTinyYOLOv3()
```

Next, we can start to load the model by first specifying the path to our model. Since our model is in the same directory as our Python script, here I show you how to load the RetinaNet model. If you use YOLOv3 or tinyYOLOv3, you need to change the file name of your h5 file accordingly.

```
detector.setModelPath(os.path.join(current_directory                    ,
"resnet50_coco_best_v2.0.1.h5"))detector.loadModel()
```

Finally, we can start to create an image object detection system. To do this, we need to specify two things: First, the directory and the filename of our input image and second, the directory and the filename of the output image.

In the following code implementation, the input image will be an image called *'traffic.jpg'* that is located in the same directory as the Python script. Meanwhile, the detection result will be saved in a file called *'traffic_detected.jpg'* in the same directory.

```
Detections = detector.detectObjectsFromImage(
input_image = os.path.join(current_directory, "traffic.jpg"),
output_image_path = os.path.join(current_directory , "traffic_detected.jpg")
)
for eachObject in detections:
   print(
       eachObject["name"] , " : ",
       eachObject["percentage_probability"], " : ",
       eachObject["box_points"] )
   print("--------------------------------")
```

And that's the code that we need to instantiate our image object detection system. Below is the full code implementation of the steps that we have covered above:

```
from imageai.Detection import ObjectDetection

import os

current_directory = os.getcwd()

detector = ObjectDetection()

detector.setModelTypeAsRetinaNet()

detector.setModelPath(os.path.join(current_directory , "resnet50_coco_best_v2.0.1.h5"))

detector.loadModel()

detections = detector.detectObjectsFromImage(input_image=os.path.join(current_directory,
"traffic.jpg"),

                                   output_image_path=os.path.join(current_directory ,
"traffic_detected.jpg"))

for eachObject in detections:

   print(eachObject["name"] , " : ", eachObject["percentage_probability"], " : ",
eachObject["box_points"] )

   print("--------------------------------")
```

If you run the complete code above, you'll get more or less similar result on the image of your choice.

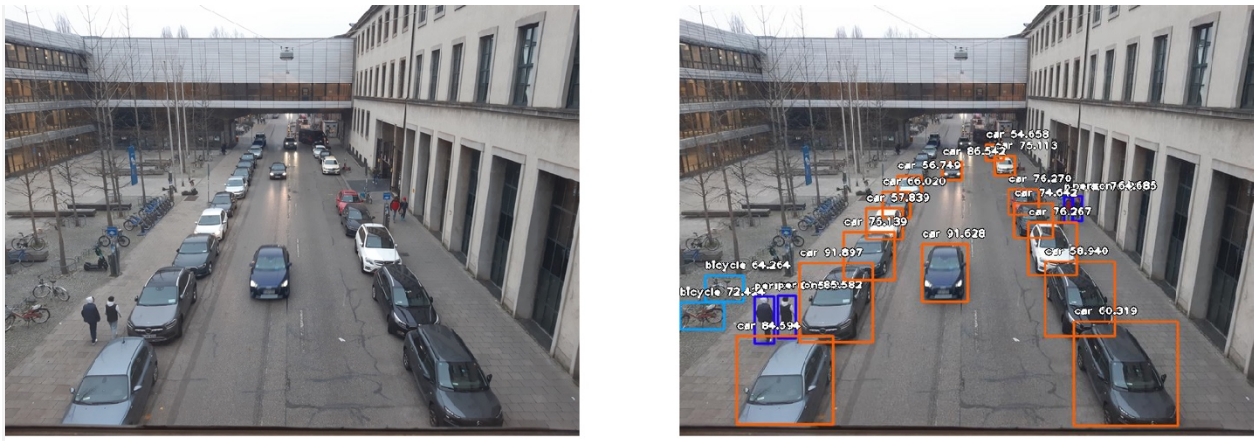


Figure 3.1. Input image without detection and output image indicating detection.

3.2 Tuning Image Object Detector

If you use the default value that we have seen in the implementation above, you might think that somehow the result of the object detection is overcrowded, with several bounding boxes overlapping one another.In order to reduce the clutter in the prediction result, you can tune the object detector such that it only shows the object that really matters for you.Let’s use the image above as an example. Let’s say I want the object detector to predict only the people and the bicycle. To do this, we need to instantiate Custom Objects method. Then, we pass the name of the objects that we want the system to detect as the argument.

```
custom = detector.CustomObjects(person=True, bicycle=True)
```

Next, we can start to build the object detection system with detectCustomObjectsFromImage method. We pass our custom variable, the path and name of our input image, as well as the path and name of our output image.

```
detections = detector.detectCustomObjectsFromImage(
custom_objects = custom,
input_image = os.path.join(current_directory, "traffic.jpg"),
output_image_path = os.path.join(current_directory , "traffic_detected.jpg")
)
```

Moreover, we can also further remove the clutter by ignoring the predictions that have probability values below a certain threshold value. Let’s say that you want to ignore the predictions with probability value below 70%. You can do that by adding the minimum_percentage_probability argument in the detectCustomObjectsFromImage method.

```
detections = detector.detectCustomObjectsFromImage(
custom_objects = custom,
input_image = os.path.join(current_directory, "traffic.jpg"),
output_image_path = os.path.join(current_directory , "traffic_detected.jpg"),
minimum_percentage_probability = 70
)
```

Below is the complete code implementation when we want to detect only people and bicycle from our image, and we also only want to show the detection where the probability value is above 70%.

```
from imageai.Detection import ObjectDetection

import os

current_directory = os.getcwd()

detector = ObjectDetection()

detector.setModelTypeAsRetinaNet()

detector.setModelPath(os.path.join(current_directory , "resnet50_coco_best_v2.0.1.h5"))

detector.loadModel()

custom = detector.CustomObjects(person=True, bicycle=True)

detections = detector.detectCustomObjectsFromImage(

            custom_objects = custom,

            input_image = os.path.join(current_directory, "traffic.jpg"),

            output_image_path = os.path.join(current_directory , "traffic_detected.jpg"),

            minimum_percentage_probability = 70

)

for eachObject in detections:

   print(eachObject["name"] , " : ", eachObject["percentage_probability"], " : ",
eachObject["box_points"] )
```

```
    print("--------------------------------")
```

If you run the code above, you'll get more or less the result like this:

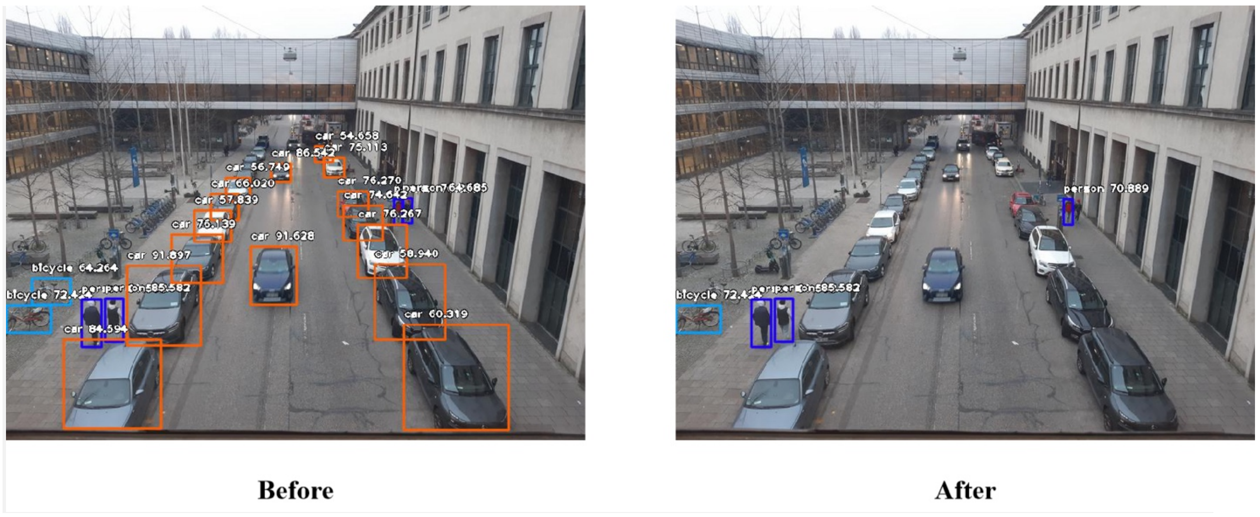


Figure 3.2. Detection of class objects with their accuracies.

And that's it! With just a few lines of code now you can implement an object detection system for your own image. However, what if you want to detect objects in a video or even from your webcam instead of an image? Let's find out how to create a similar object detection for video with ImageAI in the next section.

3.3 Creating a Video Object Detection System

Believe it or not, the code to create a video object detection system with ImageAI is pretty much similar with the image object detection system we've built before. All we need to do is changing 3 lines of code.

The first one is the library that we should import. So instead of `ObjectDetection` , we need to import `VideoObjectDetection` .

```
from imageai.Detection import VideoObjectDetection
```

The second change that we should apply is the step where we instantiate object detection class. Instead of using `ObjectDetection()` , we should use `VideoObjectDetection()` .

```
detector = VideoObjectDetection()
```

Next, the third or the final change that we should apply is when we create the object detection system. In our previous code, we use detectObjectsFromImage method. If we want to detect objects from a video, we need to use detectObjectsFromVideo instead.

```
detections = detector.detectObjectsFromVideo(
 input_file_path=os.path.join(current_directory, "footage.mp4"),
 output_file_path=os.path.join(current_directory,"footage_detected")
)
```

As you can see, the argument that we need to pass into this method is still the same as before. First, we need to specify the path to our video directory and the filename of our video. Second, we also need to specify the path and the filename of the output video.

Below is the entire code implementation to create a video object detection system.

```
from imageai.Detection import VideoObjectDetection
import os
current_directory = os.getcwd()
detector = VideoObjectDetection()
detector.setModelTypeAsRetinaNet()
detector.setModelPath(os.path.join(current_directory , "resnet50_coco_best_v2.0.1.h5"))
detector.loadModel()
detections =
detector.detectObjectsFromVideo(input_file_path=os.path.join(current_directory,
"footage.mp4"),
output_file_path=os.path.join(current_directory , "footage_detected"),
frames_per_second=20, log_progress=True)
```

Now if you run the code implementation above, you’ll get more or less similar result as below.

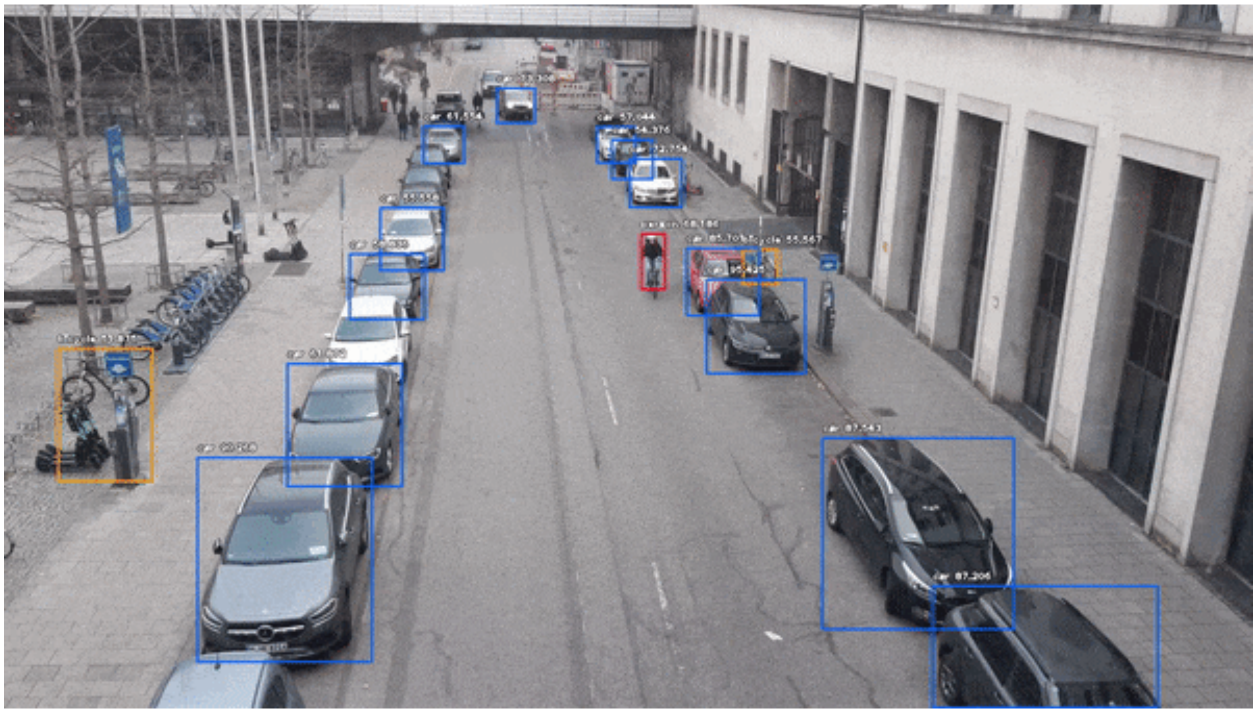


Figure 3.3. Complete output of the detected objects with greater accuracy.

Note: if you somehow don’t get the same color format in your output video, i.e you get the output video that is in BGR format instead of RGB format, you can use the code below to convert the output video back to RGB format.

All you need to do is specifying the path and the filename of the video that you want to convert as well as the converted video.

```
import cv2

import os

current_directory = os.getcwd()

# Change the path and filename of your input video with its extension

video_path = os.path.join(current_directory , 'footage_detected.avi' )

vs = cv2.VideoCapture(video_path)

output_video = None

while True:


    (frame_exists, frame) = vs.read()
    if not frame_exists:

        break

else:

         new_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

         key = cv2.waitKey(1) & 0xFF

# Write the both outputs video to a local folders

    if output_video is None:

        fourcc1 = cv2.VideoWriter_fourcc(*"MJPG")

        # Change the path and filename of your output video with .avi extension

        output_video = cv2.VideoWriter(os.path.join(current_directory ,
'footage_detected_corrected.avi' ), fourcc1, 25,(new_frame.shape[1], new_frame.shape[0]),
True)

      elif output_video is not None:

        output_video.write(new_frame)

               if key == ord("q"):

        break

vs.release()

output_video.release()

cv2.destroyAllWindows()
```

3.4 Tuning Video Object Detection

Same as our image object detection system, we can also fine tune our video object detection system a little bit. We have the option to pick which objects that we want to detect and to select the threshold for the probability value that should be displayed.

Let's say we want to only detect people and bicycle for our video object detection system. Also, we only want to show the detections that have the probability value above 70%. We can do so by first instantiating a variable to store the objects that we want to observe.

```
custom = detector.CustomObjects(person=True, bicycle=True)
```

Next, we can create our video object detection system with our custom objects. To do so, we call the detectCustomObjectsFromVideo method. For the arguments of this method, we pass our custom variable, path and filename of our input and output video, frames per second, as well as the minimum threshold for probability value.

```
detections = detector.detectCustomObjectsFromVideo(
custom_objects = custom,
input_file_path=os.path.join(execution_path, "footage.mp4"),
output_file_path=os.path.join(execution_path, "footage_detected"),
frames_per_second=20, log_progress=True,
minimum_percentage_probability = 70)
```

Below is the complete code implementation if you want to only detect people and bicycle which has a probability value above 70%.

```
from imageai.Detection import VideoObjectDetection

import os

current_directory = os.getcwd()

detector = VideoObjectDetection()

detector.setModelTypeAsRetinaNet()

detector.setModelPath(os.path.join(current_directory , "resnet50_coco_best_v2.0.1.h5"))

detector.loadModel()

custom = detector.CustomObjects(person=True, bicycle=True)

detector = detector.detectCustomObjectsFromVideo(

    custom_objects = custom,

    input_file_path=os.path.join(current_directory, "footage.mp4"),

    output_file_path=os.path.join(current_directory, "footage_detected"),

    frames_per_second=20, log_progress=True,

    minimum_percentage_probability = 70)
```

3.5 Creating Video Object Detection from Webcam

Now what if you want to create an object detection system with your camera feeds as your input? Creating this system with ImageAI is also very straightforward. The code itself is very much similar with our video object detection system. You only need to add one line of code and also change one line of code to do this.

First, we need to create a variable to instantiate the OpenCV library to capture the frame directly from our webcam.

```
camera = cv2.VideoCapture(0)
```

Finally, we need to change the argument in detectObjectsFromVideo method. Instead of specifying the input file path, we need to specify our camera input, which is the camera variable that we've created above.

```
detections = detector.detectObjectsFromVideo(
        camera_input = camera,
        output_file_path = os.path.join(current_directory,"camera_detected_video"),
        frames_per_second=20, log_progress=True)
```

Below is the complete code implementation to create an object detection system directly from your webcam.

```
from imageai.Detection import VideoObjectDetection

import os

import cv2

current_directory = os.getcwd()

camera = cv2.VideoCapture(0)

detector = VideoObjectDetection()
detector.setModelTypeAsRetinaNet()

detector.setModelPath(os.path.join(current_directory , "resnet50_coco_best_v2.0.1.h5"))
detector.loadModel()

detections = detector.detectObjectsFromVideo(
            camera_input = camera,
            output_file_path = os.path.join(current_directory, "camera_detected_video"),
            frames_per_second = 20, log_progress=True)
```

Again, if you somehow get the a different color format in your output video, you can use the aforementioned code to convert the output from BGR to RGB format.

3.6 Closing Remarks

Now you already know how to create a quick and easy object detection system with ImageAI. As you have seen, ImageAI library enables us to build an object detection system without having to deal with the complexity behind object detection model like ResNet or YOLO.

Note that with the pretrained model supported by ImageAI, the object detector can detect 80 different objects. These objects are:

person, bicycle, car, motorcycle, airplane, bus, train, truck, boat, traffic light, fire hydrant, stop_sign, parking meter, bench, bird, cat, dog, horse, sheep, cow, elephant, bear,zebra, giraffe, backpack, umbrella, handbag, tie, suitcase, frisbee, skis, snowboard, sports ball, kite, baseball bat, baseball glove, skateboard, surfboard, tennis racket, bottle, wine glass, cup, fork, knife, spoon, bowl, banana, apple, sandwich, orange, broccoli, carrot, hot dog, pizza, donot, cake, chair, couch, potted plant, bed, dining table, toilet, tv, laptop, mouse, remote, keyboard, cell phone, microwave, oven, toaster, sink, refrigerator, book, clock, vase, scissors, teddy bear, hair dryer, toothbrush.

CHAPTER 4

SYSTEM DESIGN

4.1 Procedure

I. Install TensorFlow-GPU and all required libraries
II. Set up Object Detection directory structure and Google collab Environment
III. Gather and label pictures
IV. Generate training data
V. Create label map and configure training
VI. Train object detector
VII. Test it out.
VIII. Extract Licence plate number
IX. Generate E-challan

4.2 Flowchart of Proposed System

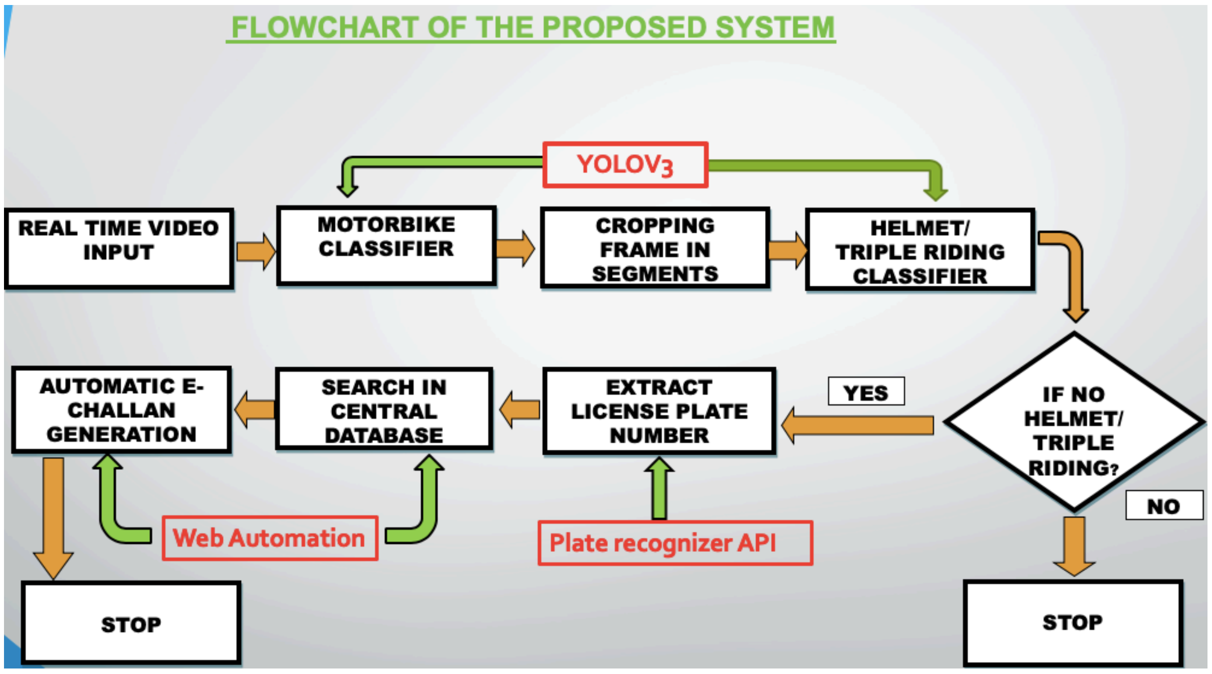

FLOWCHART OF THE PROPOSED SYSTEM
YOLOV3
REAL TIME VIDEO INPUT
MOTORBIKE CLASSIFIER
CROPPING FRAME IN SEGMENTS
HELMET/ TRIPLE RIDING CLASSIFIER
IF NO HELMET/ TRIPLE RIDING?
YES
NO
EXTRACT LICENSE PLATE NUMBER
SEARCH IN CENTRAL DATABASE
AUTOMATIC E-CHALLAN GENERATION
Web Automation
Plate recognizer API
STOP
STOP

Figure 4.1. Flowchart of the System.

4.3 System Requirements

4.3.1 Major Software's and Libraries Used

1. **Google Collab**
   Google Collab is a free Jupyter notebook environment that runs entirely in the cloud. Most importantly, it does not require a setup and the notebooks that you create can be simultaneously edited by your team members - just the way you edit documents in Google Docs. Collab supports many popular machine learning libraries which can be easily loaded in your notebook.
   As a programmer, we can perform the following using Google Collab:
   - Write and execute code in Python
   - Document your code that supports mathematical equations
   - Import/Save notebooks from/to Google Drive
   - Integrate PyTorch, TensorFlow, Keras, OpenCV
   - Free Cloud service with free GPU
2. **Python:** The programming style of Python is simple, clear and it also contains powerful different kinds of classes. Moreover, Python can easily combine other programming languages, such as C or C++. As a successful programming language, it has its own advantages:
   - Simple and easy to learn
   - Open source
   - Scalability
3. **OpenCV:** OpenCV (Open source computer vision) is a library of programming functions mainly aimed at real- time computer vision. The library is cross-platform and free for use under the open-source BSD license. OpenCV supports the deep leaning framework TensorFlow, Torch/PyTorch and caffe.
4. **NumPy:** In Python, there is data type called array. To implement the data type of array with python, NumPy is the essential library for analysing and calculating data. They are all open source libraries. NumPy is mainly used 22 for the matrix calculation
5. **Pandas , Matplotlib**: pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool,built on top of the Python programming language. Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. Matplotlib makes easy things easy and hard things possible.
6. **Pillow:** Python Imaging Library (abbreviated as PIL) (in newer versions known as Pillow) is a free and open-source additional library for the Python programming

language that adds support for opening, manipulating, and saving many different image file formats.

The **Python Imaging Library** adds image processing capabilities to your Python interpreter. This library provides extensive file format support, an efficient internal representation, and powerful image processing capabilities. The core image library is designed for fast access to data stored in a few basic pixel formats. It should provide a solid foundation for a general image processing tool.

7. **TensorFlow:** TensorFlow, an open-source software library for dataflow programming across a range of tasks. It is a symbolic math library and is also used for machine learning applications such as neural networks. It is used for both research and production at Google. Its flexible architecture allows for the easy deployment of computation across a variety of platforms (CPUs, GPUs, TPUs), and from desktops to clusters of servers to mobile and edge devices. TensorFlow computations are expressed as stateful dataflow graphs. The name TensorFlow derives from the operations that such neural networks perform on multidimensional data arrays. These arrays are referred to as "tensors".

Why TensorFlow for Object Detection:

- It allows Deep Learning.
- Known as the second-generation machine learning system, it performs numerical computations through data flow graphs.
- It is open source and free.
- It is reliable (and without major bugs). 26
- It is backed by Google and a good community.
- It is a skill recognized by many employers.
- It is easy to implement.
- With capability of running on CPUs and GPUs, it can be deployed in broad range of products of Google such as Speech Recognition, Google Photos, Gmail and even Search.

ImageAI: is a python library built to empower developers, researchers and students to build applications and systems with self-contained Deep Learning and Computer Vision capabilities using simple and few lines of code.

Keras: Keras is an open source neural network library written in Python. It can run on top of TensorFlow, Microsoft Cognitive Toolkit, or Theano Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular, and extensible Keras.This is done via the **keras preprocessing image.ImageDataGenerator** class.

This class allows you to configure random transformations and normalization operations to be done on your image data during training instantiate generators of augmented image batches (and their labels) via .flow(data, labels) These generators can then be used with the Keras model method that accepts datainputs, fit_generator, evaluate_generator and predict_generator.

Selenium Web driver: Selenium is an umbrella project for a range of tools and libraries that enable and support the automation of web browsers. Selenium supports automation of all the major browsers in the market using *WebDriver*. WebDriver is an API and protocol that defines a language-neutral interface for controlling the behavior of web browsers.

Web scraping is a technique which could help us transform HTML unstructured data into structed data in spreadsheet.

Refers to both the language bindings and the implementations of the individual browser controlling code. This is commonly referred to as just *WebDriver*. Selenium WebDriver is aW3C Recommendation.

- WebDriver is designed as a simple and more concise programming interface.
- WebDriver is a compact object-oriented API.
- It drives the browser effectively.

CHAPTER 5

SYSTEM IMPLEMENTATION

One important element of deep learning and machine learning at large is dataset. A good dataset will contribute to a model with good precision and recall. In the realm of object detection in images or motion pictures.

- **For Motorcycle detection**: we used trained model with COCO Dataset with accuracy of 99%.
- **For Helmet Detection:** We created our own Yolov3 Model with our own dataset with 1000+ images of helmet and non-helmet riders.
- **For License plate:** We used API for extraction and web automation for getting details for E- Challan Generation.

5.1 Model for Motorcycle Detection:

Coco Dataset for Motorcycle Detection: COCO is a large-scale object detection, segmentation, and captioning dataset. This version contains images, bounding boxes " and labels for the 2017 version. Coco defines 80 classes.

COCO stands for Common Object Context. The COCO dataset contains the images which are captured in our daily life scenes. COCO provides multi-object labeling, segmentation mask annotations, image captioning, key-point detection and panoptic segmentation annotations with a total of 80 categories, making it a very versatile and multi-purpose dataset.

As we have mentioned above that the COCO dataset contains a total of 80 categories, we import the required libraries that are required to access the COCO dataset. The input contains many objects like Buses, Cars, and Motorcycles etc. The libraries imported will helps divide the required objects from all other objects and the detected objects will be given certain IDs to access them later. Coco.GetObjectIds is a function used to get the IDs for the differentiated objects.

The 80 categories are as follows:

Table 5.1. COCO dataset objects

Person	Bicycle	Car	Motorcycle	Airplane
Bus	Train	Truck	Boat	Traffic light
Fire hydrant	Stop sign	Parking meter	Bench	Bird
Cat	Dog	Sheep	Cow	Elephant
Bear	Zebra	Giraffe	Backpack	Umbrella
Handbag	Tie	Suitcase	Frisbee	Skis
Snowboard	Sports ball	Kite	Baseball bat	Baseball glove
Skateboard	Surfboard	Tennis racket	Bottle	Wine glass
Cup	Knife	Spoon	Bowl	Banana
Apple	Sandwich	Orange	Broccoli	Carrot
Hotdog	Pizza	donot	Cake	Chair
Couch	Potted plant	Bed	Dining table	Toilet
Tv	Laptop	Mouse	Remote	Keyboard
Cell phone	Microwave	Toaster	Sink	Refrigerator
Book	Clock	Vase	Scissors	Teddy bear
Hair dryer	Toothbrush	Fork	Horse	oven

Now the question is what if you want to create custom dataset object detection.

Steps for creating a custom dataset for object detection:

- Annotate data
- Convert annotation files to COCO dataset format
- Train an instance segmentation model with mmdetection framework

Framework: Darknet is an open source neural network framework written in C and CUDA. YOLO first takes an input image.The framework then divides the input image into grids (say a 3 X 3 grid) Image classification and localization are applied on each grid. YOLO then predicts the bounding boxes and their corresponding class probabilities for objects (if any are found, of course). It is fast, easy to install, and supports CPU and GPU computation.

In our case, the framework is done in such a way that the motorcycle which is detected will be divided into three frames.

One contains the complete motorcycle image for future reference, the second contains the upper part of the motorcycle image for helmet detection and the final part contains the lower part for license plate recognition.

5.2 Model for Helmet Detection:

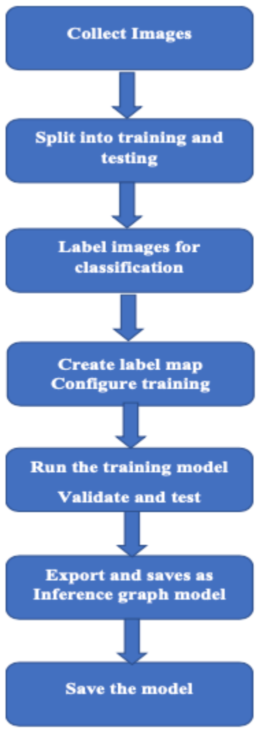
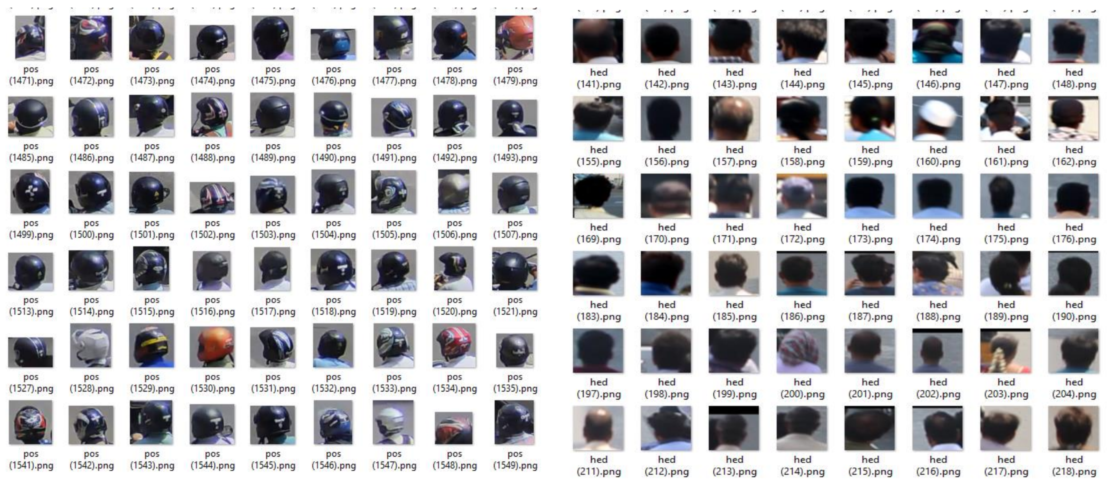

Collect Images
Split into training and testing
Label images for classification
Create label map Configure training
Run the training model Validate and test
Export and saves as Inference graph model
Save the model

Figure 5.1. Flowchart for Helmet Model

5.2.1 Procedure for training a YOLOV3 HELMET model

Gathering images (Creating data set): To detect a bike rider with helmet or without helmet. We need bunch of images of bike-riders with helmet, bike-rider without helmet and bike license plate. In this project, we used 1000+ images.



Helmet Images Non Helmet Images

Figure 5.2. Helmet dataset

Label Images:

Label the all images with the help of **LableImg tool**. In this project, **Helmet class** was created with the help of LableImg tool. Create .xml file corresponding to each image with the above following categories of classes

Now that our dataset labels are in the required format, we need to create a train-test split. I chose to create a test set containing 10% of the images in the dataset. Configuring YOLO with your dataset. Now that we have created our train and test sets, we need to make some changes to train the YOLO model on the dataset.

Training:

Now that our dataset is ready to use, we can begin training. Before we start, compile the darknet repository with the make command. To compile with specific options, such as GPU, CUDNN and OPENCV. This will create a darknet executable. Trained weights for model. You can set other parameters (learning rate, momentum, weight decay etc by editing the corresponding lines). Finally, model is ready to use.

5.3 Number plate Detection

Platerecognizer: is an open source Automatic License Plate Recognition library written in C++ with bindings in C#, Java, Node.js, and Python. The library analyses images and video streams to identify license plates. The output is the text representation of any license plate characters.

The ASP.NET Web API is an extensible framework for building HTTP based services that can be accessed in different applications on different platforms such as web, windows, mobile etc. It works more or less the same way as ASP.NET MVC web application except that it sends data as a response instead of html view. It is like a web service or WCF service but the exception is that it only supports HTTP protocol.

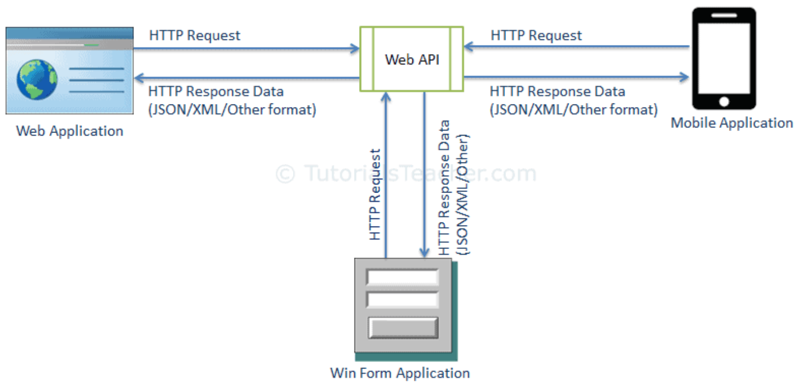
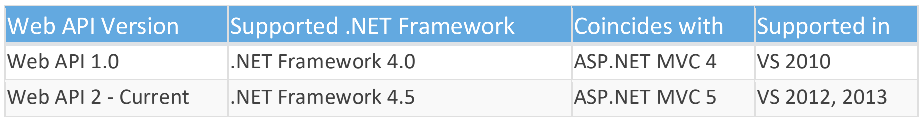


Figure 5.3. Working of Web API



Web API Version	Supported .NET Framework	Coincides with	Supported in
Web API 1.0	.NET Framework 4.0	ASP.NET MVC 4	VS 2010
Web API 2 - Current	.NET Framework 4.5	ASP.NET MVC 5	VS 2012, 2013

Table 5.2. ASP.NET Web API Versions

Table Plate Recognizer Snapshot API! You can use our API to access our API endpoints, which can read license plates from images. Plate Recognizer provides accurate, fast, developer-friendly Automatic License Plate Recognition (ALPR) software that works in all environments.

The software can be used in many ways:

1. Recognize license plates from camera streams. The results are browsable, searchable, and can trigger alerts. The data repository can be in the cloud or stored entirely within your on-site network.
2. Recognize license plates from camera streams and send the results to your own application.
3. Integrate license plate recognition into your application directly in-code.

5.4Automated E-Challan Generation

Further this Detected LP Numbers are injected to **RTO Database** for extracting the further details of the Violator. Now, we do **Automation of web Browser,** to get the details of offender from **RTO Database.**

1. **Selenium** is an umbrella project for a range of tools and libraries that enable and support the automation of web browsers. Selenium supports automation of all the major browsers in the market using ***WebDriver.***
2. WebDriver is an API and protocol that defines a language-neutral interface for controlling the behavior of web browsers. **Web scraping** is a technique which could help us transform HTML unstructured data into structed data in spreadsheet.
3. With **Pillow Image Library,** with extracted details saved in excel sheet an automatic E-challan is generated with details Including Date and time & further it can be sent through message, mail or post

5.5 Triple Riding Detection

5.5.1 Proposed Model

The system consists of a main subsystem: setting up the environment, training, testing the model and getting the accurate coordinates of the vehicle that comes along with the system. The Proposed system of triple riding is illustrated in Fig. 5.4. The imposing challans or fines with an automated push of the data to the respective user account comes as an extension to the system.

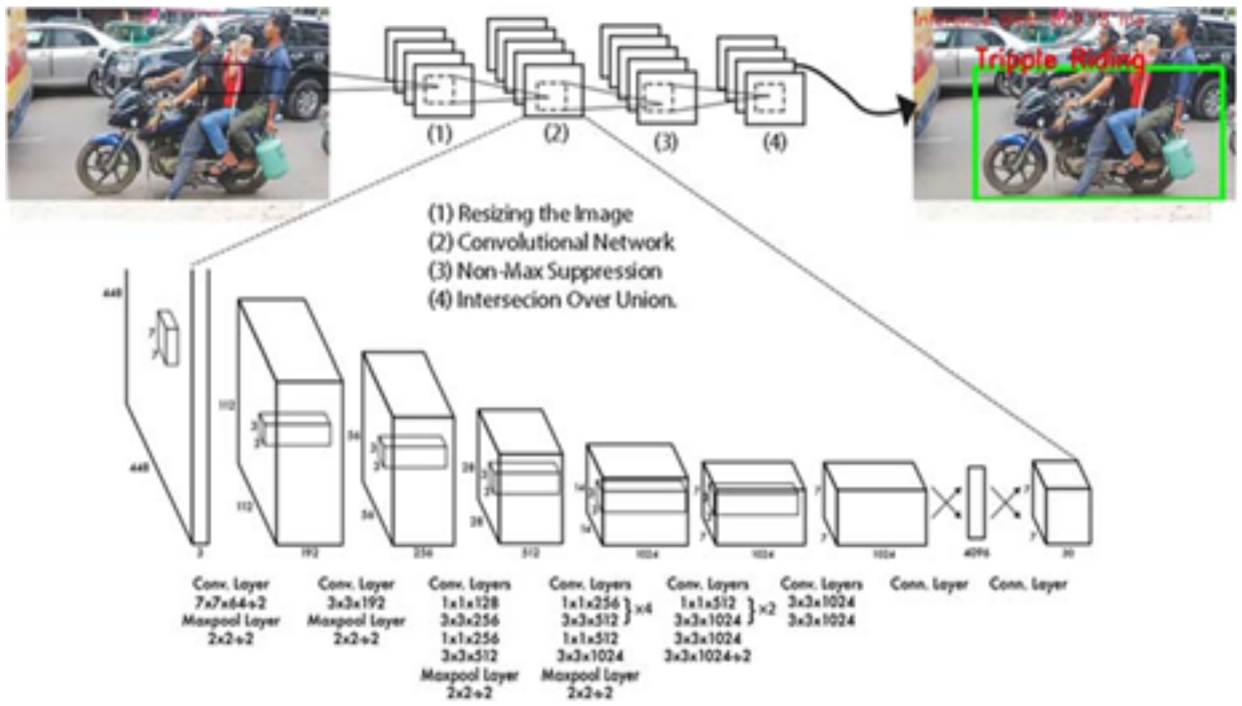

Tripple Riding
(1)
(2)
(3)
(4)
(1) Resizing the Image
(2) Convolutional Network
(3) Non-Max Suppression
(4) Intersecion Over Union.
Conv. Layer 7x7x64-s-2 Maxpool Layer 2x2-s-2
Conv. Layer 3x3x192 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x128 3x3x256 1x1x256 3x3x512 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x256 3x3x512 }x4 1x1x512 3x3x1024 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x512 3x3x1024 }x2 3x3x1024 3x3x1024-s-2
Conv. Layers 3x3x1024 3x3x1024
Conn. Layer
Conn. Layer

Figure 5.4. Proposed Model of triple riding detection

5.5.2. Traffic rule violation recognition system

The system mainly uses the deconvolutional approach of deep learning along with the deep learning framework Darknet and the object detection algorithm YOLOv3 performs the various functionalities like detection, recognition, and Identification, followed by classification. The subsystem having the support of the image, video and live feed as the input, enables the system to process all kinds of data. Having the image and video input assessment as an extension, the live feed of the camera modules deployed at the junctions processes every single frame. Every single frame is subjected to the detection of various objects. In our prototype model, the various objects that were subjected to the process of the training are the bike and the person. The two objects bike, and person are the most important aspects of the focus.

The detection process of the YOLO model makes use of the target regression as a regression problem for the spatially separate target box and confidence.

1. The YOLO first divides the image into convolutions of size NxN like 13x13, and the size of each of the NxN cells depends on the size of the input.

2. Each cell of these NxN cells is responsible for predicting the number of bounding boxes in the input image.

3. For every box in the input, the deconvolutional network predicts the confidence that the bounding box contains the object and the probability of the enclosed object being from one of the classes mentioned in the configuration file.

4. After that, it applies the Non-Max Suppression to remove the bounding boxes with less confidence value.

5. The processed images are subject to the Intersection over union function developed to find out the relation between the persons detected in the frame along with the motorbike. Thereby, marking the rectangular bounding box with triple riding as the label.

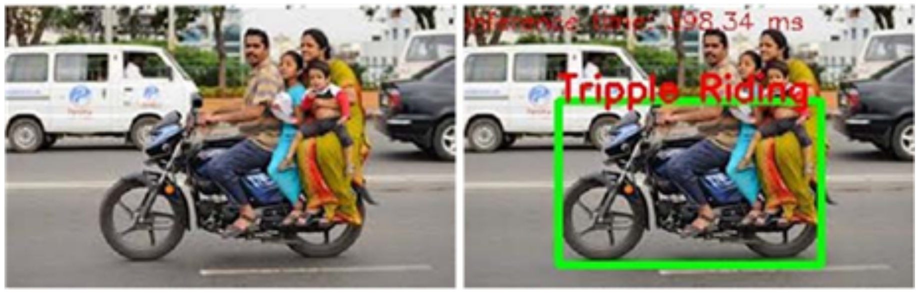


Figure 5.5. Test image for triple riding.

5.5.3 YOLOv3 (You Only Look Once):

YOLO is an incremental approach, in this the third version of the YOLO is used. This comes with an incremental improvement from the initial YOLO. The YOLO algorithm forwards the image as a whole only once through the network. On the other hand, The Single Shot Detector (SSD) also takes the entire image at the same time, but YOLOv3 performs much faster, while achieving a better accuracy.

The YOLOv3 system has been trained on the COCO dataset which was available on the internet, capable of detecting various classes. The following figures depict the working of the model, Figs. 2 is the test images for the triple riding whereas Fig. 3 is the test image for the non-triple riding.

Figure 5.6. Test image for non-triple riding detection

5.5.4 COCO dataset:

Microsoft Common Objects in context (MS COCO) dataset has 91 common object groups. This dataset has 2,500,000 labeled instances in 328,000 images. This dataset is very extensively used for training because it has images of everyday scenes in their natural environment. Objects have been labeled using per-instance segmentation to help in achieving accurate object localization. This dataset contains images of 91 object types. While MS COCO has fewer categories than ImageNet and SUN, it has more instances per category.

5.5.5 Training and testing data:

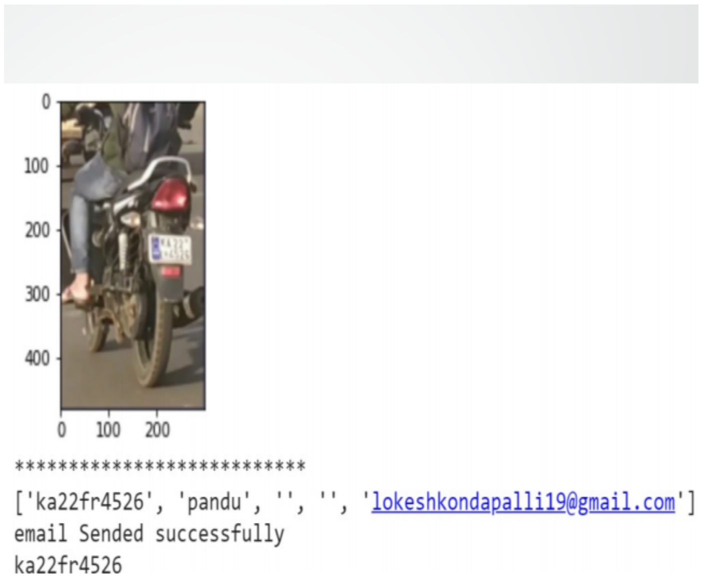
For training purposes, COCO dataset is being used as it provides us 2,500,000 labeled instances in 3,28,000 images. The alternative would have been to collect data and label it manually, using COCO dataset helps us avoid this work. DPMv5-P is the performance reported in VOC release 5. DPMv5-C uses the same implementation but is trained using MS COCO dataset. For testing purposes, 1000 unseen images were used. Figure 4 shows a sample of detecting triple riding on which the model was tested on.

5.5.6 License plate detection:

If the offender is found with triple riding, there is no need for this step. However, if the offender is found, then the motorcycle image is given as input to the license plate detection phase Fig.5. Where the image is passed to this library to detect the License Plate Number and Return it and Store it for Further use.

Plate Recognizer: It is an open source Automatic License Plate Recognition library. The library analyzes images and video streams to identify license plates. The output is the text representation of any license plate characters. Further

These Detected LP Numbers are injected to the Database for extracting the further details of the Violator.



```
**************************
['ka22fr4526', 'pandu', '', '', 'lokeshkondapalli19@gmail.com']
email Sended successfully
ka22fr4526
```

Figure 5.7**.** License Plate detection

The system is tested on images that have triple riding as well as images that do not have triple riding to check how accurate our system is. The system detects triple riders and puts a bounding box on them and labels them as triple riders. The accuracy of our model is 80% with an F1 score of 0.847 and also the precision value of our system is 0.8. Implementation of this kind of system will increase general awareness and hence reduce accidents.

5.5.7 E-Challan for Triple Riding:

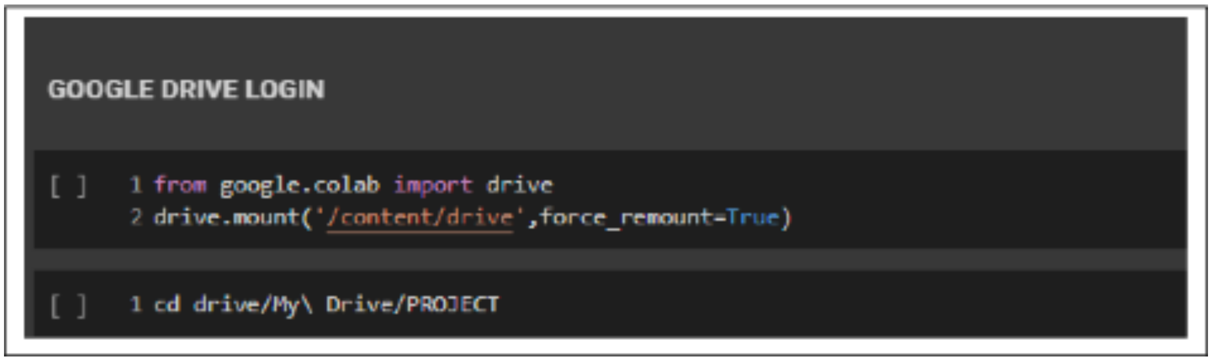
After detection of the license plate number, the system generates E-challan to the offenders by accessing the database for their details. It sends the message through E-mail of the offender.

CHAPTER 6

WORKFLOW & SYSTEM TESTING

6.1 Account Setup Gdrive

InGooglecollab Editor, First, upload all required data into google drive and now sync Gdrive with collab editor.


GOOGLE DRIVE LOGIN

```
[ ]  from google.colab import drive
     drive.mount('/content/drive',force_remount=True)
```

```
[ ]  cd drive/My\ Drive/PROJECT
```

Figure 6.1. Google Drive Login

Create **PROJECT** Folder:

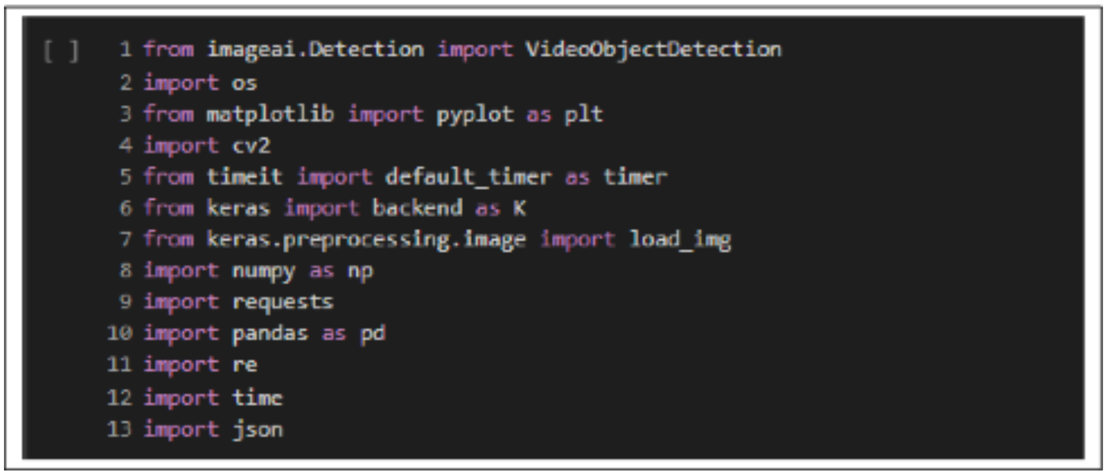
- Motorcycle_Detection_Model File
- Helmet Detection Model File
- Final Outputs (Folder)

  Subfolders

  - Full Frame Image
  - Bikes Image
  - Rider Image

- Challan (Folder)
- Chrome WebDriver.exe

Importing Required Libraries

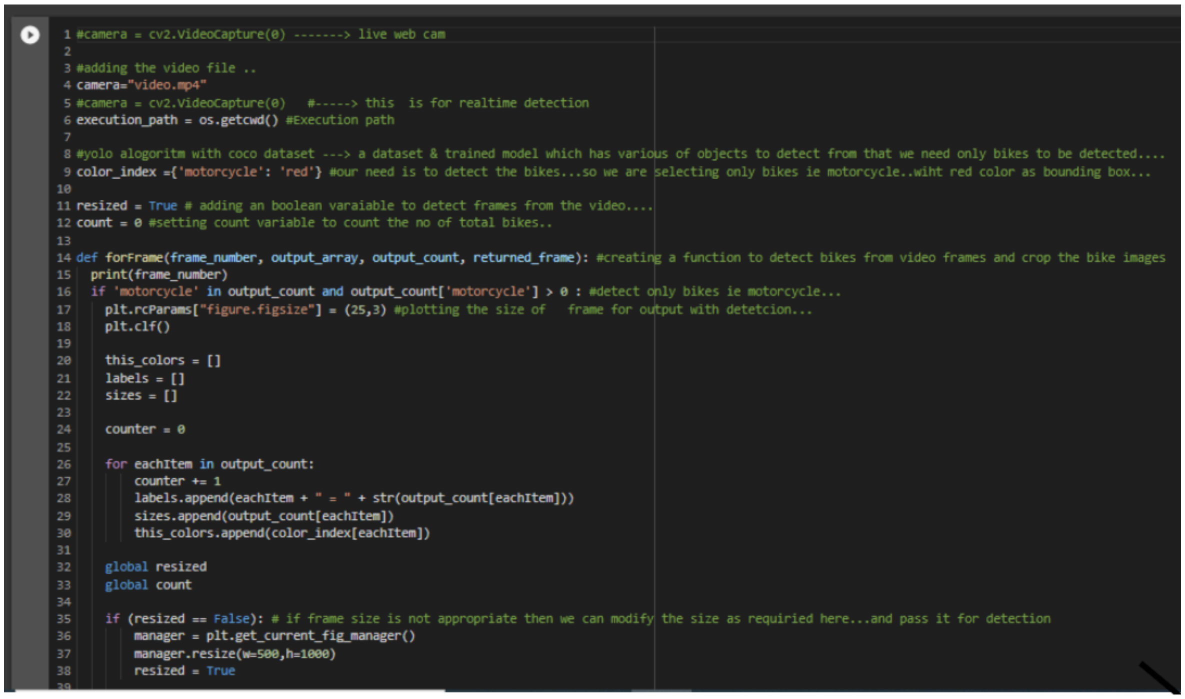


```python
from imageai.Detection import VideoObjectDetection
import os
from matplotlib import pyplot as plt
import cv2
from timeit import default_timer as timer
from keras import backend as K
from keras.preprocessing.image import load_img
import numpy as np
import requests
import pandas as pd
import re
import time
import json
```

Figure 6.2. Importing libraries

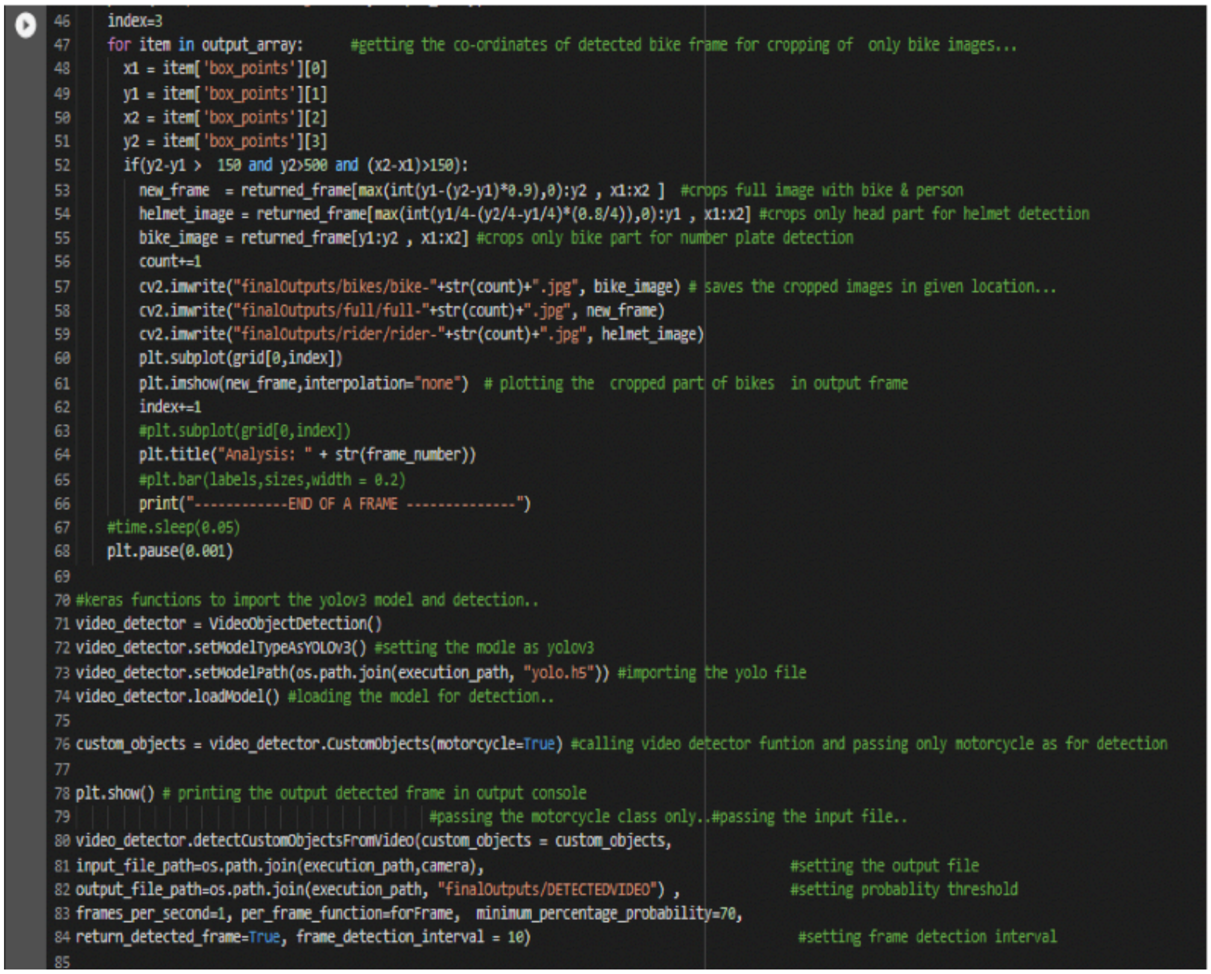
- In this section we explain different processing steps. initial phase, frames are collected at regular intervals from video file and passed into detection model for processing.
- All these techniques are subjected to predefined conditions and constraints, especially the license plate number extraction part. Since, this work takes video as its input, the speed of execution is crucial. We have used above said methodologies to build a holistic system for both helmet detection and license plate number extraction.

6.2 Detection of Motorcycle



```python
#camera = cv2.VideoCapture(0) -------> live web cam

#adding the video file ..
camera="video.mp4"
#camera = cv2.VideoCapture(0)   #-----> this  is for realtime detection
execution_path = os.getcwd() #Execution path

#yolo alogoritm with coco dataset ---> a dataset & trained model which has various of objects to detect from that we need only bikes to be detected....
color_index ={'motorcycle': 'red'} #our need is to detect the bikes...so we are selecting only bikes ie motorcycle..wiht red color as bounding box...

resized = True # adding an boolean varaiable to detect frames from the video....
count = 0 #setting count variable to count the no of total bikes..

def forFrame(frame_number, output_array, output_count, returned_frame): #creating a function to detect bikes from video frames and crop the bike images
  print(frame_number)
  if 'motorcycle' in output_count and output_count['motorcycle'] > 0 : #detect only bikes ie motorcycle...
    plt.rcParams["figure.figsize"] = (25,3) #plotting the size of   frame for output with detetcion...
    plt.clf()

    this_colors = []
    labels = []
    sizes = []

    counter = 0

    for eachItem in output_count:
        counter += 1
        labels.append(eachItem + " = " + str(output_count[eachItem]))
        sizes.append(output_count[eachItem])
        this_colors.append(color_index[eachItem])

    global resized
    global count

    if (resized == False): # if frame size is not appropriate then we can modify the size as requiried here...and pass it for detection
        manager = plt.get_current_fig_manager()
        manager.resize(w=500,h=1000)
        resized = True
```

Figure 6.3. (a) Code for Motorcycle Detection



```
    index=3
    for item in output_array:      #getting the co-ordinates of detected bike frame for cropping of  only bike images...
      x1 = item['box_points'][0]
      y1 = item['box_points'][1]
      x2 = item['box_points'][2]
      y2 = item['box_points'][3]
      if(y2-y1 >  150 and y2>500 and (x2-x1)>150):
        new_frame  = returned_frame[max(int(y1-(y2-y1)*0.9),0):y2 , x1:x2 ]  #crops full image with bike & person
        helmet_image = returned_frame[max(int(y1/4-(y2/4-y1/4)*(0.8/4)),0):y1 , x1:x2] #crops only head part for helmet detection
        bike_image = returned_frame[y1:y2 , x1:x2] #crops only bike part for number plate detection
        count+=1
        cv2.imwrite("finalOutputs/bikes/bike-"+str(count)+".jpg", bike_image) # saves the cropped images in given location...
        cv2.imwrite("finalOutputs/full/full-"+str(count)+".jpg", new_frame)
        cv2.imwrite("finalOutputs/rider/rider-"+str(count)+".jpg", helmet_image)
        plt.subplot(grid[0,index])
        plt.imshow(new_frame,interpolation="none")  # plotting the  cropped part of bikes  in output frame
        index+=1
        #plt.subplot(grid[0,index])
        plt.title("Analysis: " + str(frame_number))
        #plt.bar(labels,sizes,width = 0.2)
        print("-----------END OF A FRAME -------------")
    #time.sleep(0.05)
    plt.pause(0.001)

#keras functions to import the yolov3 model and detection..
video_detector = VideoObjectDetection()
video_detector.setModelTypeAsYOLOv3() #setting the modle as yolov3
video_detector.setModelPath(os.path.join(execution_path, "yolo.h5")) #importing the yolo file
video_detector.loadModel() #loading the model for detection..

custom_objects = video_detector.CustomObjects(motorcycle=True) #calling video detector funtion and passing only motorcycle as for detection

plt.show() # printing the output detected frame in output console
                                        #passing the motorcycle class only..#passing the input file..
video_detector.detectCustomObjectsFromVideo(custom_objects = custom_objects,
input_file_path=os.path.join(execution_path,camera),                          #setting the output file
output_file_path=os.path.join(execution_path, "finalOutputs/DETECTEDVIDEO") ,  #setting probablity threshold
frames_per_second=1, per_frame_function=forFrame,  minimum_percentage_probability=70,
return_detected_frame=True, frame_detection_interval = 10)                      #setting frame detection interval
```

Figure 6.3. (b) Continuation of Code for Motorcycle Detection

The frame chosen is given as input to YOLOv3 Motorcycle detection model, where the classes to be detected are "Motorcycle". At the output, image with required class detection along with confidence of detection through bounding box and probability value is obtained as shown in the Fig 6.4(a) and Fig 6.4(b). Here frame with 'motorcycle' classes detected.

Rear View Front View

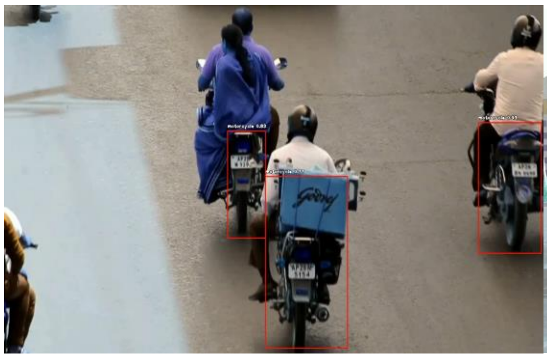


Figure 6.4. (a) Frame-1 Case 1

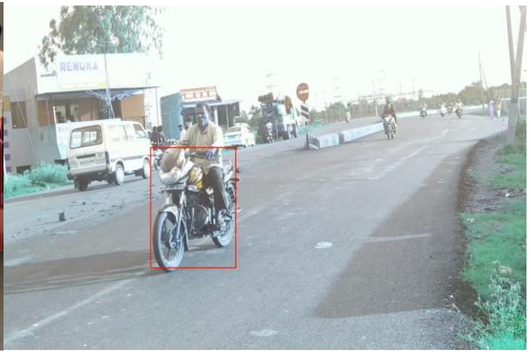


Figure 6.4. (b) Frame-2 Case 2

The details of these extracted images which is stored in a dictionary which can be later used for further processing.

Output for each object: [{'name': 'motorcycle', 'percentage_probability':89.4, 'box_points': [104, 84, 265, 400]}

With the help of functions given by Image AI library, only the detected objects are extracted as shown below, and stored as separate images and named with class name and image number in order.

We crop these detected frames in 3 formats:

1. Full Image with motorbike and rider
2. Bike Image
3. Rider Image

For example, it will be saved as Full-1, Full-2, etc. || Bike-1, Bike-2...etc. || Rider-1, Rider-2...etc.

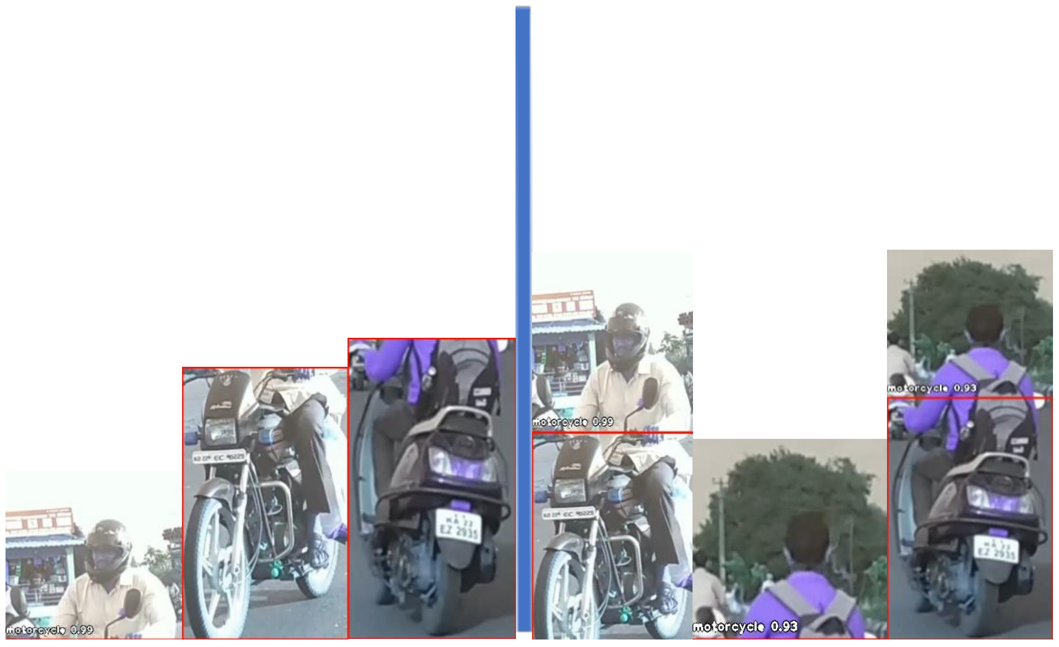
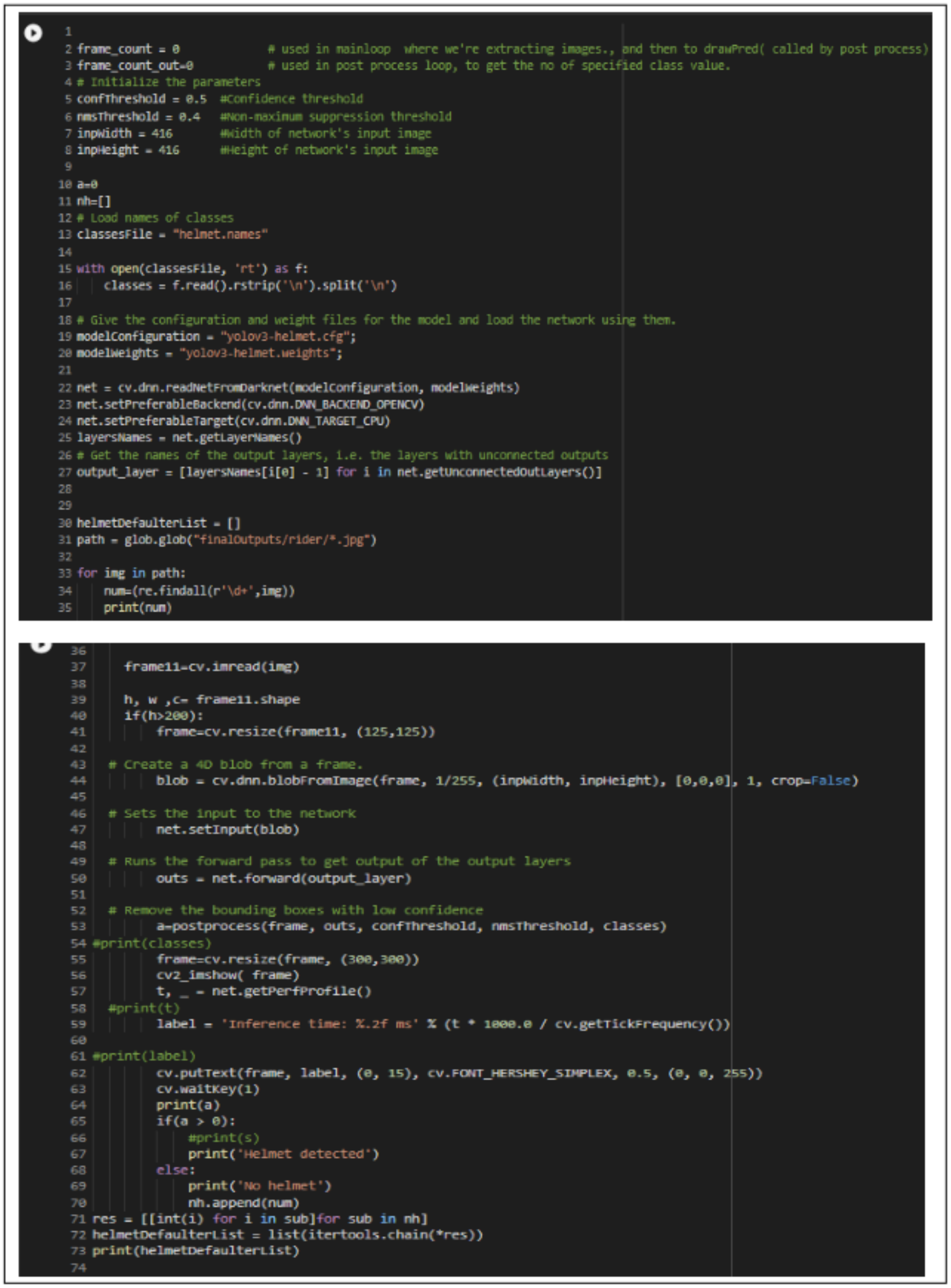


Figure 6.5. Helmet ROI cropping.



```python

frame_count = 0            # used in mainloop  where we're extracting images., and then to drawPred( called by post process)
frame_count_out=0          # used in post process loop, to get the no of specified class value.
# Initialize the parameters
confThreshold = 0.5  #Confidence threshold
nmsThreshold = 0.4   #Non-maximum suppression threshold
inpWidth = 416       #Width of network's input image
inpHeight = 416      #Height of network's input image

a=0
nh=[]
# Load names of classes
classesFile = "helmet.names"

with open(classesFile, 'rt') as f:
    classes = f.read().rstrip('\n').split('\n')

# Give the configuration and weight files for the model and load the network using them.
modelConfiguration = "yolov3-helmet.cfg";
modelWeights = "yolov3-helmet.weights";

net = cv.dnn.readNetFromDarknet(modelConfiguration, modelWeights)
net.setPreferableBackend(cv.dnn.DNN_BACKEND_OPENCV)
net.setPreferableTarget(cv.dnn.DNN_TARGET_CPU)
layersNames = net.getLayerNames()
# Get the names of the output layers, i.e. the layers with unconnected outputs
output_layer = [layersNames[i[0] - 1] for i in net.getUnconnectedOutLayers()]


helmetDefaulterList = []
path = glob.glob("finalOutputs/rider/*.jpg")

for img in path:
    num=(re.findall(r'\d+',img))
    print(num)

    frame11=cv.imread(img)

    h, w ,c= frame11.shape
    if(h>200):
        frame=cv.resize(frame11, (125,125))

  # Create a 4D blob from a frame.
        blob = cv.dnn.blobFromImage(frame, 1/255, (inpWidth, inpHeight), [0,0,0], 1, crop=False)

  # Sets the input to the network
        net.setInput(blob)

  # Runs the forward pass to get output of the output layers
        outs = net.forward(output_layer)

  # Remove the bounding boxes with low confidence
        a=postprocess(frame, outs, confThreshold, nmsThreshold, classes)
#print(classes)
        frame=cv.resize(frame, (300,300))
        cv2_imshow( frame)
        t, _ = net.getPerfProfile()
  #print(t)
        label = 'Inference time: %.2f ms' % (t * 1000.0 / cv.getTickFrequency())

#print(label)
        cv.putText(frame, label, (0, 15), cv.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255))
        cv.waitKey(1)
        print(a)
        if(a > 0):
            #print(s)
            print('Helmet detected')
        else:
            print('No helmet')
            nh.append(num)
res = [[int(i) for i in sub]for sub in nh]
helmetDefaulterList = list(itertools.chain(*res))
print(helmetDefaulterList)

```

Figure 6.6. Code for Helmet Detection.

Once the Motorcycle class is obtained, the Rider images is given as input to Helmet detection model. While testing the helmet detection model, some false detections were observed. So, the person image was cropped to get only top one-fourth portion of image, as shown in Fig. 2 (Rider.jpg). This ensures that false detection cases are eliminated as well as avoid cases leading to wrong results when the rider is holding helmet in hand while riding or keeping it on motorcycle while riding instead of wearing.

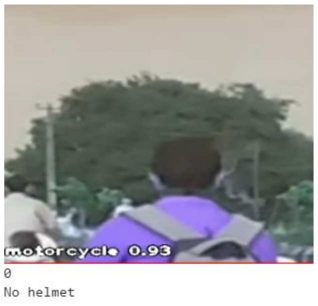
Now two cases Arise:

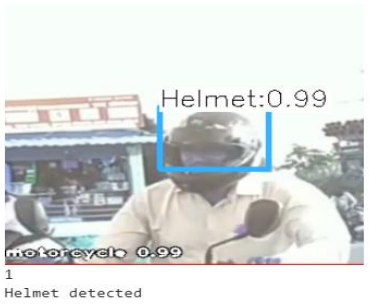
Case 1: When the motorcycle rider is wearing helmet

Case 2: When the motorcycle rider is not wearing helmet

Helmet Detection Model:

Rider-1.jpg Rider-2.jpg



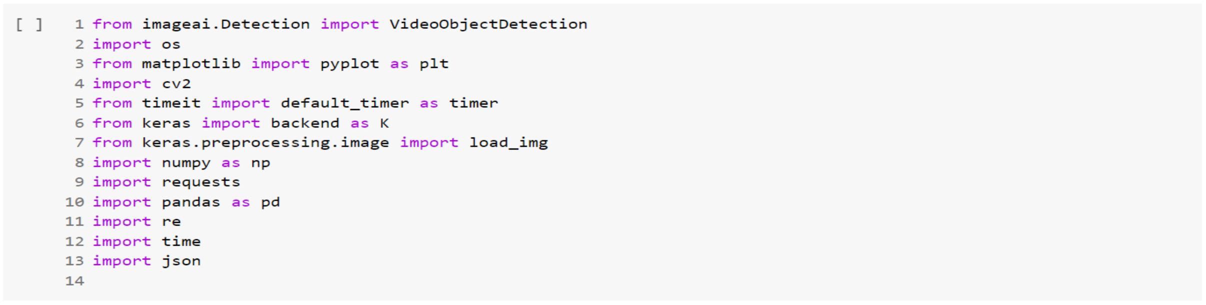


Case-1 Case-2

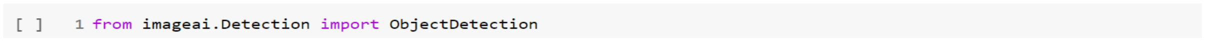
Figure 6.7. Helmet yolov3 prediction

After applying cropped image to helmet detection model, output is as shown. The bounding box around helmet along with the detection probability is displayed as shown in (Rider-2.jpg). As the rider wearing helmet in Case 2, no further processing is necessary. Since in Case 1, rider is not wearing helmet, no bounding box is created.

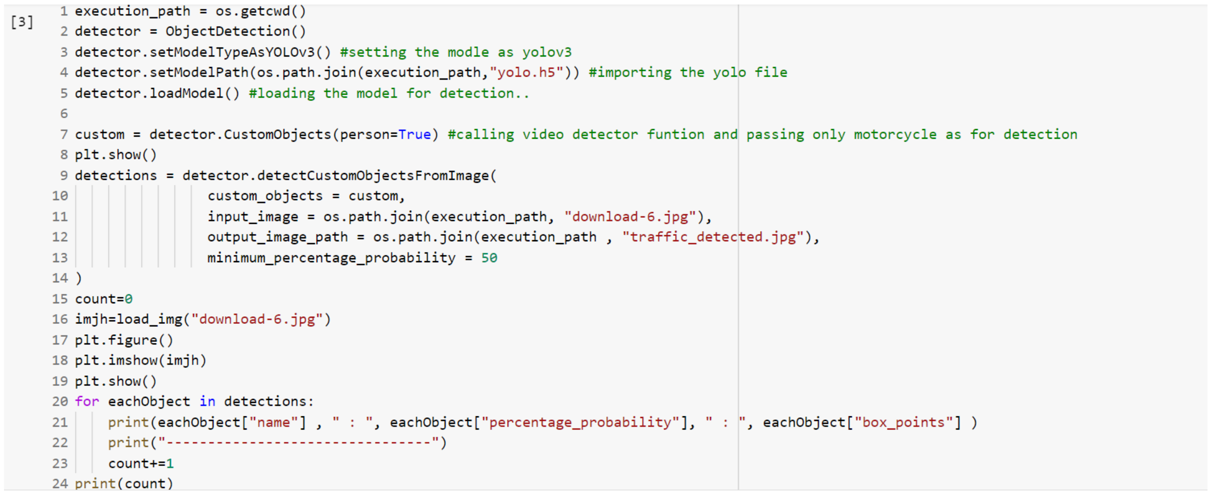
6.4 Detection of Triple Riding



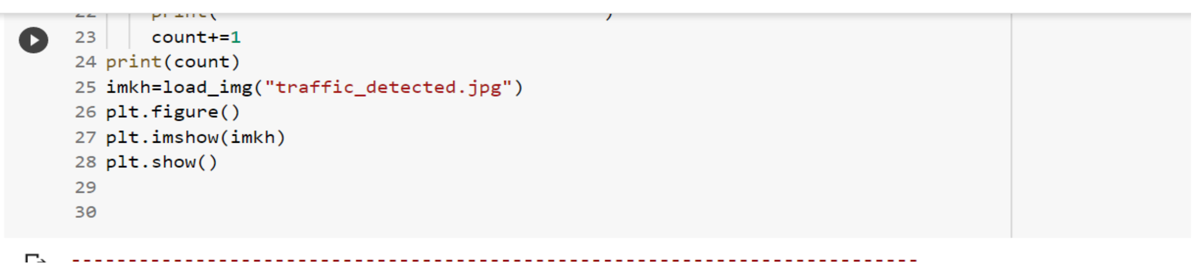
```
[ ]
from imageai.Detection import VideoObjectDetection
import os
from matplotlib import pyplot as plt
import cv2
from timeit import default_timer as timer
from keras import backend as K
from keras.preprocessing.image import load_img
import numpy as np
import requests
import pandas as pd
import re
import time
import json

```



```
[ ]
from imageai.Detection import ObjectDetection
```

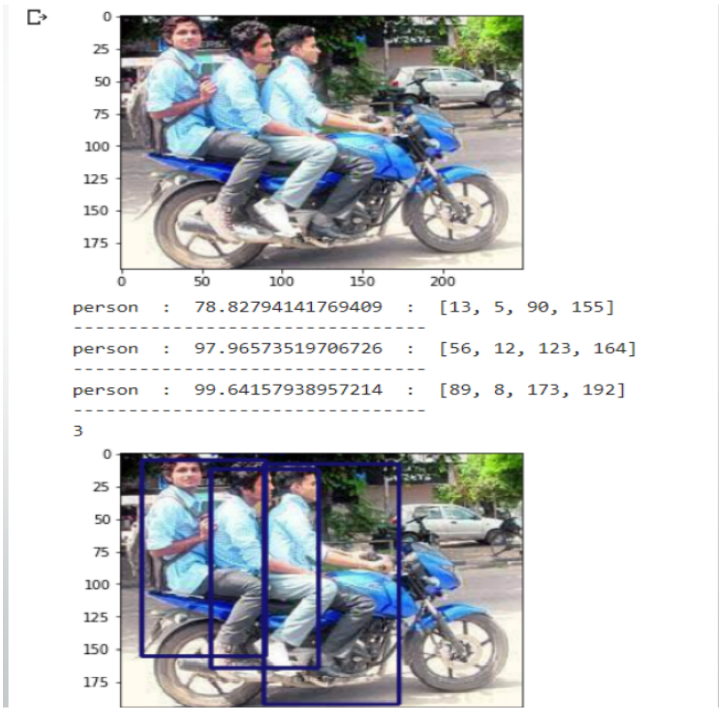


```
[3]
execution_path = os.getcwd()
detector = ObjectDetection()
detector.setModelTypeAsYOLOv3() #setting the modle as yolov3
detector.setModelPath(os.path.join(execution_path,"yolo.h5")) #importing the yolo file
detector.loadModel() #loading the model for detection..

custom = detector.CustomObjects(person=True) #calling video detector funtion and passing only motorcycle as for detection
plt.show()
detections = detector.detectCustomObjectsFromImage(
                custom_objects = custom,
                input_image = os.path.join(execution_path, "download-6.jpg"),
                output_image_path = os.path.join(execution_path , "traffic_detected.jpg"),
                minimum_percentage_probability = 50
)
count=0
imjh=load_img("download-6.jpg")
plt.figure()
plt.imshow(imjh)
plt.show()
for eachObject in detections:
    print(eachObject["name"] , " : ", eachObject["percentage_probability"], " : ", eachObject["box_points"] )
    print("--------------------------------")
    count+=1
print(count)
```



```
    count+=1
print(count)
imkh=load_img("traffic_detected.jpg")
plt.figure()
plt.imshow(imkh)
plt.show()


--------------------------------------------------------------------------------
```

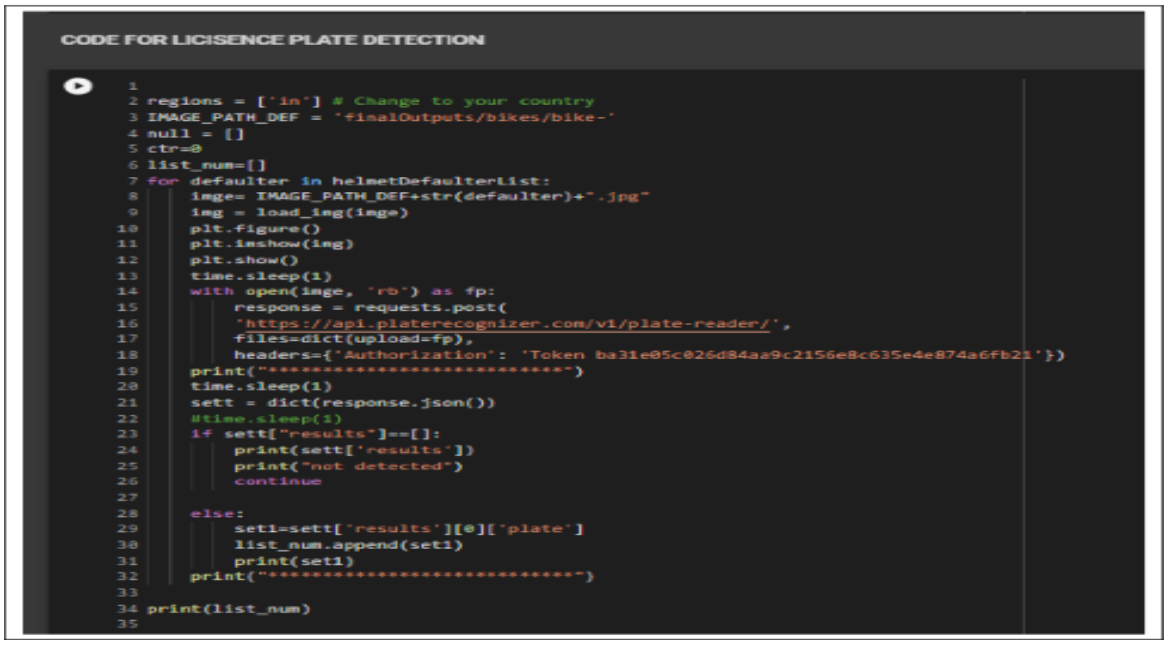
Figure 6.8. Code for Triple Riding Detection.



```
person  :  78.82794141769409  :  [13, 5, 90, 155]
------------------------------
person  :  97.96573519706726  :  [56, 12, 123, 164]
------------------------------
person  :  99.64157938957214  :  [89, 8, 173, 192]
------------------------------
3
```

Figure 6.9. Output for Triple Riding Detection.

6.5 Detection of Number Plate



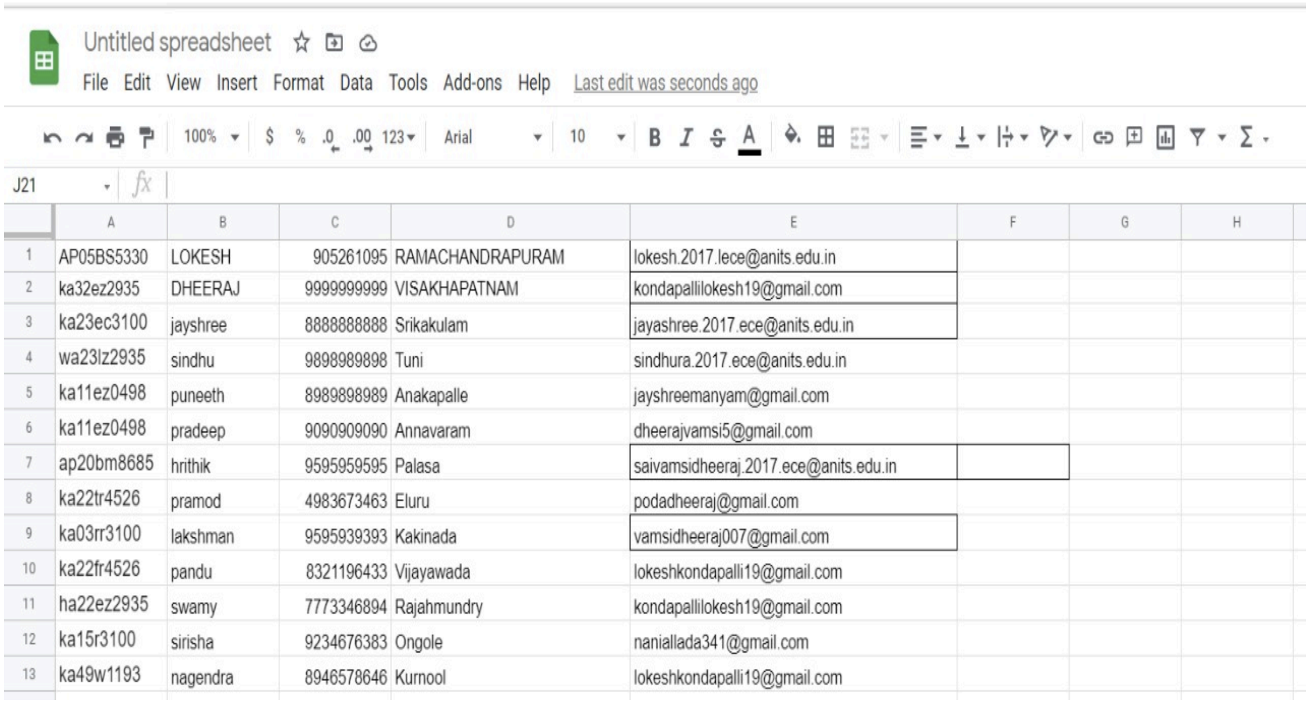
CODE FOR LICISENCE PLATE DETECTION

```python

regions = ['in'] # Change to your country
IMAGE_PATH_DEF = 'finalOutputs/bikes/bike-'
null = []
ctr=0
list_num=[]
for defaulter in helmetDefaulterList:
    imge= IMAGE_PATH_DEF+str(defaulter)+".jpg"
    img = load_img(imge)
    plt.figure()
    plt.imshow(img)
    plt.show()
    time.sleep(1)
    with open(imge, 'rb') as fp:
        response = requests.post(
        'https://api.platerecognizer.com/v1/plate-reader/',
        files=dict(upload=fp),
        headers={'Authorization': 'Token ba31e05c026d84aa9c2156e8c635e4e874a6fb21'})
    print("****************************")
    time.sleep(1)
    sett = dict(response.json())
    #time.sleep(1)
    if sett["results"]==[]:
        print(sett['results'])
        print("not detected")
        continue

    else:
        set1=sett['results'][0]['plate']
        list_num.append(set1)
        print(set1)
    print("******************************")

print(list_num)

```

Figure 6.10. Code for Number Plate Recognition

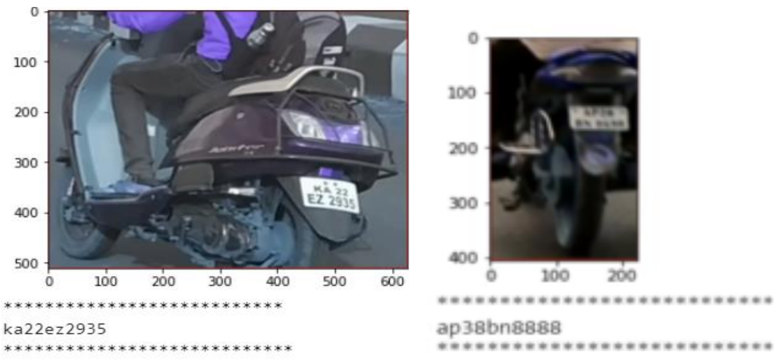
If the helmet is found, there is no need for this step. However, if the helmet is not found, then the motorcycle image is given as input to license plate detection phase. Where the image is passed to this library to detect the License Plate Number and Return it and Store it for Further use.

Platerecognizer is an open source Automatic License Plate Recognition library. The library analyzes images and video streams to identify license plates. The output is the text representation of any license plate characters. Further this Detected LP Numbers are injected to **Database** for extracting the further details of the Violator.


Untitled spreadsheet
File Edit View Insert Format Data Tools Add-ons Help Last edit was seconds ago
100% $ % .0 .00 123 Arial 10
J21

	A	B	C	D	E	F	G	H
1	AP05BS5330	LOKESH	905261095	RAMACHANDRAPURAM	lokesh.2017.lece@anits.edu.in			
2	ka32ez2935	DHEERAJ	9999999999	VISAKHAPATNAM	kondapallilokesh19@gmail.com			
3	ka23ec3100	jayshree	8888888888	Srikakulam	jayashree.2017.ece@anits.edu.in			
4	wa23lz2935	sindhu	9898989898	Tuni	sindhura.2017.ece@anits.edu.in			
5	ka11ez0498	puneeth	8989898989	Anakapalle	jayshreemanyam@gmail.com			
6	ka11ez0498	pradeep	9090909090	Annavaram	dheerajvamsi5@gmail.com			
7	ap20bm8685	hrithik	9595959595	Palasa	saivamsidheeraj.2017.ece@anits.edu.in			
8	ka22tr4526	pramod	4983673463	Eluru	podadheeraj@gmail.com			
9	ka03rr3100	lakshman	9595939393	Kakinada	vamsidheeraj007@gmail.com			
10	ka22fr4526	pandu	8321196433	Vijayawada	lokeshkondapalli19@gmail.com			
11	ha22ez2935	swamy	7773346894	Rajahmundry	kondapallilokesh19@gmail.com			
12	ka15r3100	sirisha	9234676383	Ongole	naniallada341@gmail.com			
13	ka49w1193	nagendra	8946578646	Kurnool	lokeshkondapalli19@gmail.com			

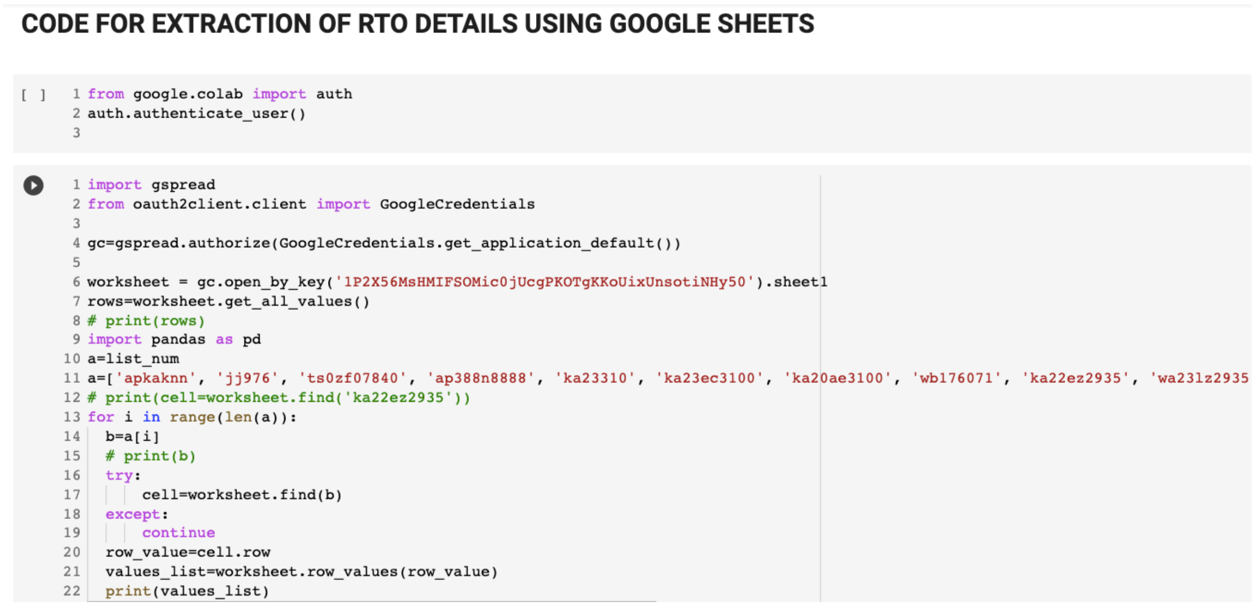
Figure 6.11. Database created in Spreadsheet.



```
****************************
ka22ez2935
****************************
```

```
****************************
ap38bn8888
****************************
```

Figure 6.12. Number plate recognition


CODE FOR EXTRACTION OF RTO DETAILS USING GOOGLE SHEETS

```
from google.colab import auth
auth.authenticate_user()

```

```
import gspread
from oauth2client.client import GoogleCredentials

gc=gspread.authorize(GoogleCredentials.get_application_default())

worksheet = gc.open_by_key('1P2X56MsHMIFSOMic0jUcgPKOTgKKoUixUnsotiNHy50').sheet1
rows=worksheet.get_all_values()
# print(rows)
import pandas as pd
a=list_num
a=['apkaknn', 'jj976', 'ts0zf07840', 'ap388n8888', 'ka23310', 'ka23ec3100', 'ka20ae3100', 'wb176071', 'ka22ez2935', 'wa231z2935
# print(cell=worksheet.find('ka22ez2935'))
for i in range(len(a)):
  b=a[i]
  # print(b)
  try:
    cell=worksheet.find(b)
  except:
    continue
  row_value=cell.row
  values_list=worksheet.row_values(row_value)
  print(values_list)
```

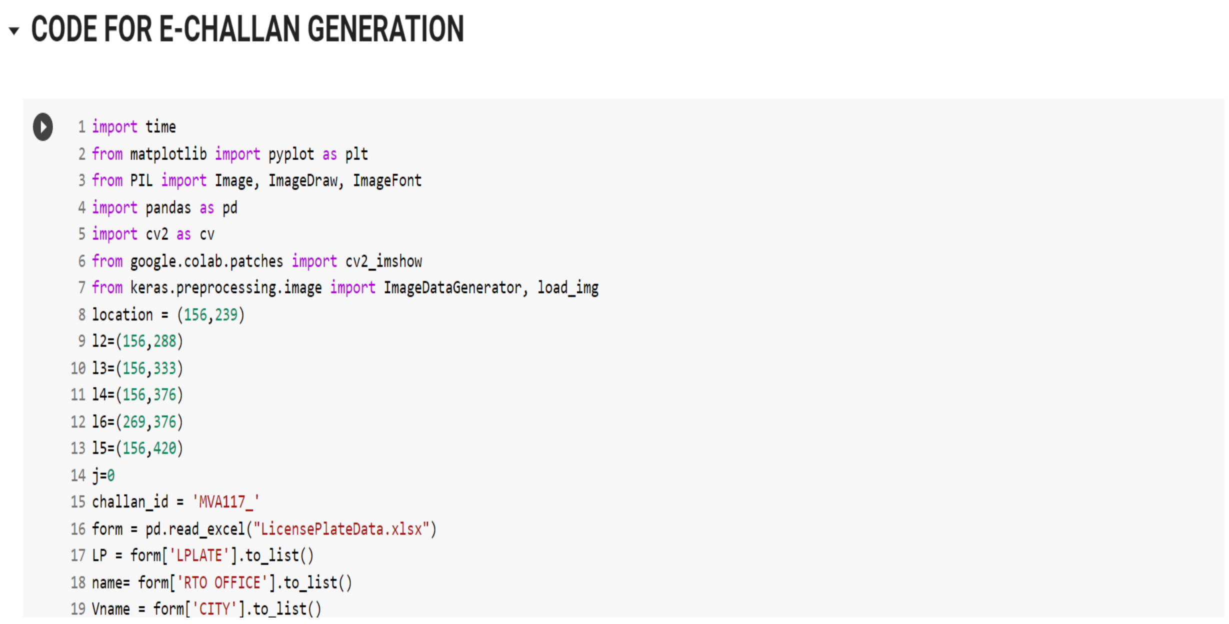
Figure 6.13. Extraction of details using Google Sheets.

Now, to get the details of offender the model searches from created Database.

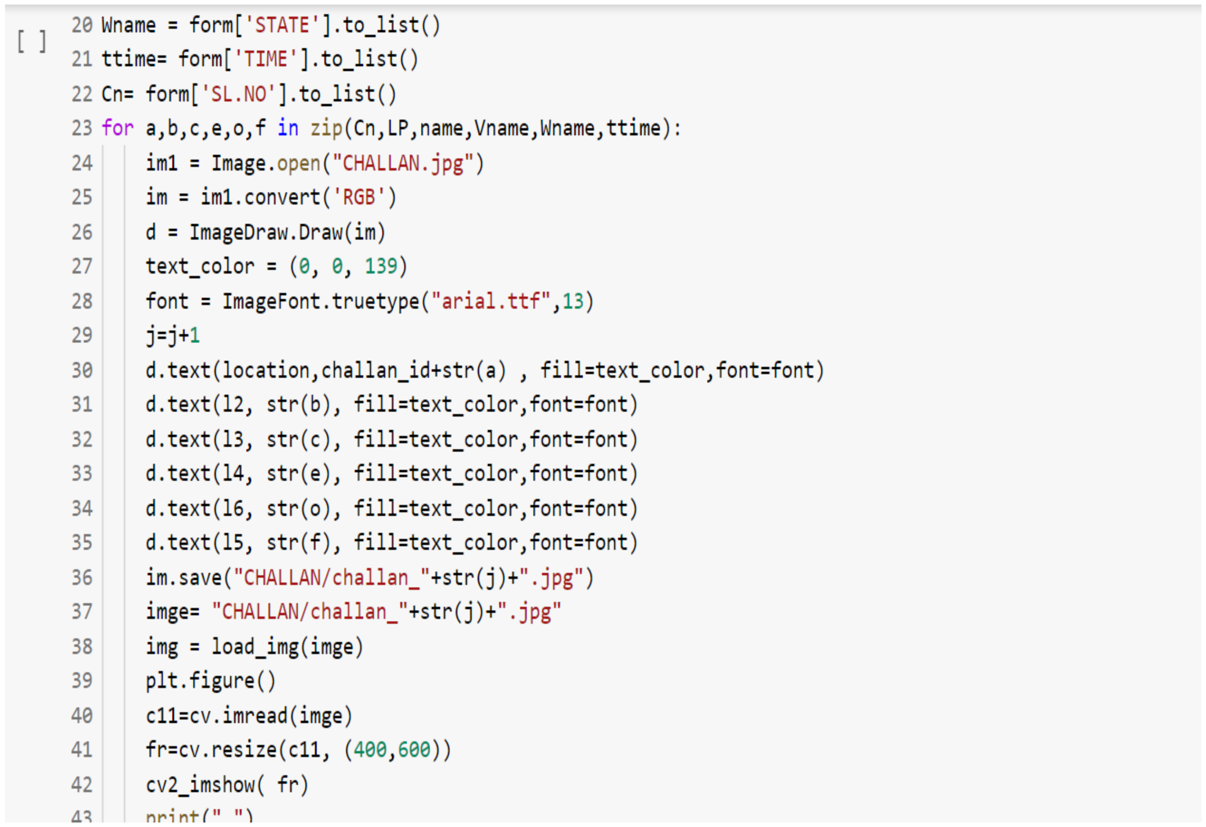
Selenium supports automation of all the major browsers in the market using *WebDriver*. WebDriver is an API and protocol that defines a language-neutral interface for controlling the behaviour of web browsers.

Web scraping is a technique which could help us transform HTML unstructured data into structed data in spreadsheet.

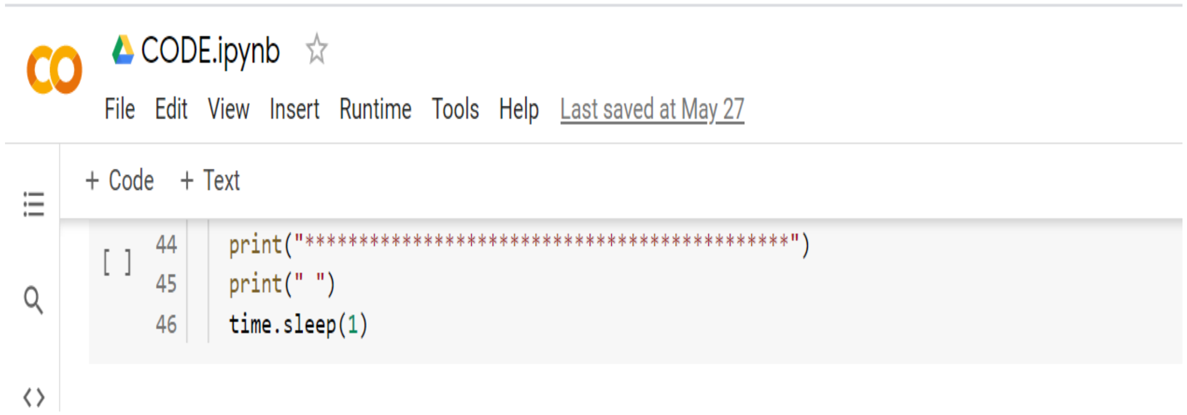
6.6 E-Challan Generation


▾ CODE FOR E-CHALLAN GENERATION

```
import time
from matplotlib import pyplot as plt
from PIL import Image, ImageDraw, ImageFont
import pandas as pd
import cv2 as cv
from google.colab.patches import cv2_imshow
from keras.preprocessing.image import ImageDataGenerator, load_img
location = (156,239)
l2=(156,288)
l3=(156,333)
l4=(156,376)
l6=(269,376)
l5=(156,420)
j=0
challan_id = 'MVA117_'
form = pd.read_excel("LicensePlateData.xlsx")
LP = form['LPLATE'].to_list()
name= form['RTO OFFICE'].to_list()
Vname = form['CITY'].to_list()
```



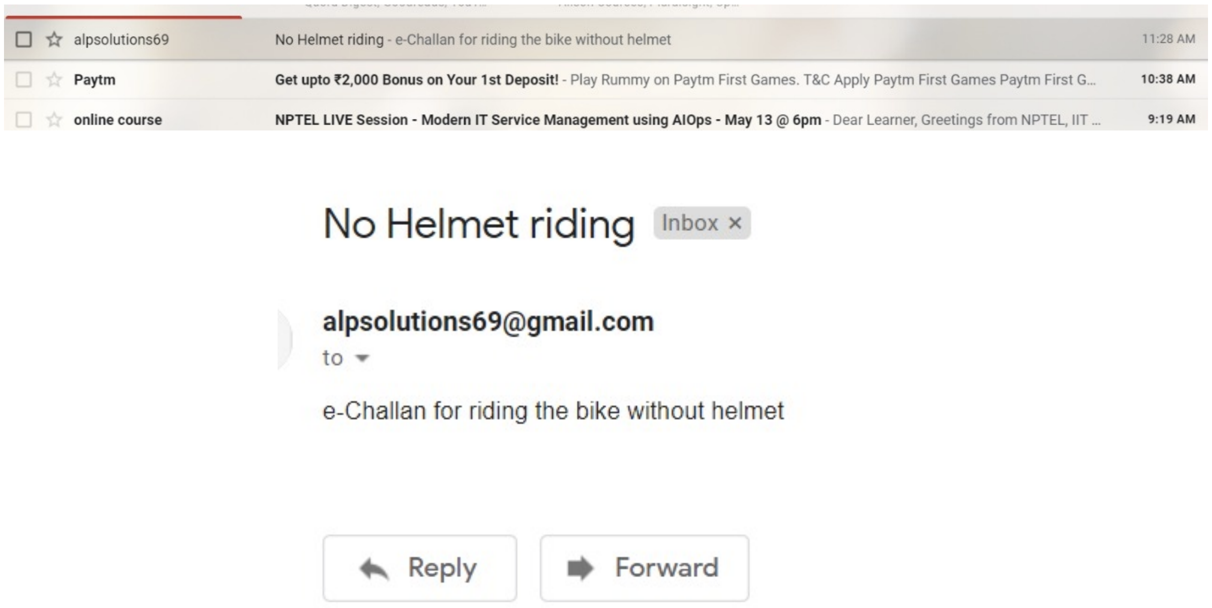
```
Wname = form['STATE'].to_list()
ttime= form['TIME'].to_list()
Cn= form['SL.NO'].to_list()
for a,b,c,e,o,f in zip(Cn,LP,name,Vname,Wname,ttime):
    im1 = Image.open("CHALLAN.jpg")
    im = im1.convert('RGB')
    d = ImageDraw.Draw(im)
    text_color = (0, 0, 139)
    font = ImageFont.truetype("arial.ttf",13)
    j=j+1
    d.text(location,challan_id+str(a) , fill=text_color,font=font)
    d.text(l2, str(b), fill=text_color,font=font)
    d.text(l3, str(c), fill=text_color,font=font)
    d.text(l4, str(e), fill=text_color,font=font)
    d.text(l6, str(o), fill=text_color,font=font)
    d.text(l5, str(f), fill=text_color,font=font)
    im.save("CHALLAN/challan_"+str(j)+".jpg")
    imge= "CHALLAN/challan_"+str(j)+".jpg"
    img = load_img(imge)
    plt.figure()
    c11=cv.imread(imge)
    fr=cv.resize(c11, (400,600))
    cv2_imshow( fr)
    print(" ")
```


CODE.ipynb
File Edit View Insert Runtime Tools Help Last saved at May 27
+ Code + Text

```
    print("*********************************************")
    print(" ")
    time.sleep(1)
```

Figure 6.14. Code for E-Challan Generation.

With **Pillow Image Library,** with extracted details saved in excel sheet an automatic E-challan is generated with details Including Date and time & further it can be sent through message, mail or post.



alpsolutions69 — No Helmet riding - e-Challan for riding the bike without helmet — 11:28 AM

Paytm — Get upto ₹2,000 Bonus on Your 1st Deposit! - Play Rummy on Paytm First Games. T&C Apply Paytm First Games Paytm First G... — 10:38 AM

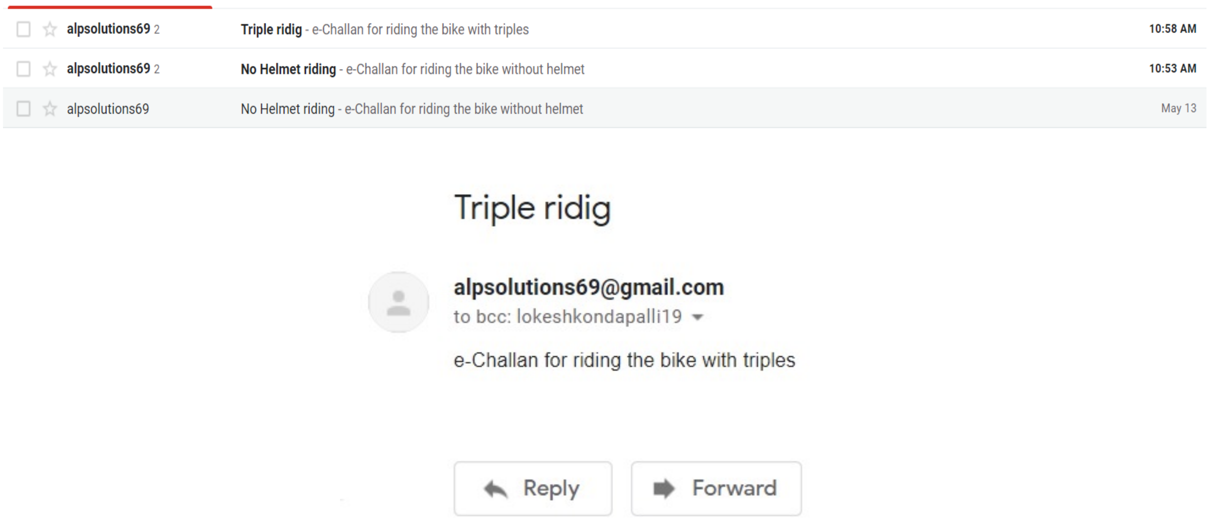
online course — NPTEL LIVE Session - Modern IT Service Management using AIOps - May 13 @ 6pm - Dear Learner, Greetings from NPTEL, IIT ... — 9:19 AM

No Helmet riding

alpsolutions69@gmail.com
to

e-Challan for riding the bike without helmet

Figure 6.15. Challan for No Helmet Violation



alpsolutions69 2 — Triple ridig - e-Challan for riding the bike with triples — 10:58 AM

alpsolutions69 2 — No Helmet riding - e-Challan for riding the bike without helmet — 10:53 AM

alpsolutions69 — No Helmet riding - e-Challan for riding the bike without helmet — May 13

Triple ridig

alpsolutions69@gmail.com
to bcc: lokeshkondapalli19

e-Challan for riding the bike with triples

Figure 6.16. Challan for Triple Riding Violation.

6.7 RELIABILITY

The system developed is analysed in terms of its efficiency of producing the output with accurate detection of objects.

Table 6.1. Analysis of the model

Model	Efficiency
Bike Detection	95%
Helmet Detection	70%
Triple Riding	80%
License Plate Recognition	70%

Here in the Table2, the helmet is detected with 70% accuracy because during the detection process if the model encounters obstacles like the signboards, turbans, caps, etc, in the processed frame, it is treated as the helmet. As for triple riding, the accuracy is 80%, the same obligations as helmet detection is observed in this case. The license plate identification is purely based on the resolution of the camera. The camera that we used has a 720p definition and we were able to detect the characters accurately on the license plate. But in some of the cases, one or two characters were misread. Ideally, we recommend a resolution of 1080p especially for license number recognition and, 720p in case of a helmet and triple riding.

About the statistics of the project, the detection process is rapid. The video taken has a duration of 54sec and a total frame of 1360. The model took around 1min 8sec to extract the video. Out of the total frames 23 frames are used for the implementation. Detection of bikes took 220 sec and the helmet detection took 17 sec for 8 frames. Triple Riding Detection processed 3 images in 8 sec. At last, license plate recognition and sending of the challan took approximately 2 sec per image. The total time taken for this whole process is 5 min 5sec. These are purely the observations taken for the prototype developed. The duration varies for real-time surveillance videos. The objective with which the system is proposed has achieved satisfactory results.

FUTURE WORK:

The system implemented is a prototype. It can be expanded to process the day-to-day traffic video by attaining the permissions of the required authorities. A large database is created to maintain the records of the violators and their payment of the challans being monitored every few minutes. Also, the identification of the license plate becomes the core part of this project. So, a camera of high resolution is recommended to maintain precision and accuracy. For sending the challan directly to offender's mobile numbers, the subscriptions for SMS are required, as of now it is sent through mail ids, but the motto to send the challan to their mails as well as through SMS along with their violation photo, time and date. Our system is developed to process the above-mentioned future implementations.

CONCLUSION:

A Non-Helmet Rider and Triple Riding Detection system is developed where a video file is taken as input. If the motorcycle rider in the video footage is not wearing helmet while riding the motorcycle, or riding with three members, then the license plate number of that motorcycle is extracted and displayed for above cases separately. Object detection principle with YOLO architecture is used for motorcycle, person, helmet and license plate detection. Google Spreadsheet is used for license plate number extraction if rider is not wearing helmet or triple riding. The characters are extracted from LP so that it can be used for other purposes. All the objectives of the project is achieved satisfactorily.

REFERENCES:

[1] H. Li and C. Shen, “Reading car license plates using deep convolutional neural networks and LSTMs”, arXiv preprint arXiv:1601.05610, 2016.

[2] C. Vishnu, D. Singh, C. K. Mohan, and S. Babu, "Detection of motorcyclists without helmet in videos using convolutional neural network," 2017 International Joint Conference on Neural Networks (IJCNN).

[3] How to Create a Simple Object Detection System with Python and ImageAI-by Ruben Winastwan.

[4] Wang H, Yu Y, Cai Y, Chen L, Chen X (2018) A vehicle recognition algorithm based on deep transfer learning with a multiple feature subspace distribution. Sensors 18(12):4109.

[5] Satya (2018) Deep learning based object detection using YOLOv3 with OpenCV (Python/C++). https:// www.learnopencv.com/deep-learning-based-object-detection-using-yolov3-with-opencv-python-c/.

[6] Raj KD, Chairat A, Timtong V, Dailey MN, Ekpanyapong M (2018) Helmet violation processing using deep learning. In: 2018 International workshop on advanced image technology. IEEE, IWAIT, pp 1–4.

[7] Redmon (2018) Darknet: open-source neural networks in c. http://pjreddie.com/darknet/.

[8] Redmon J, Divvala S, Girshick R, Farhadi A (2016) You only look once: unified, real-time object detection. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 779–788.

[9] Desai M, Khandelwal S, Singh L, Gite S (2016) Automatic helmet detection on public roads. Int J Eng Trends Technol (IJETT) 35:185–188.

[10] E-challan generation – how to send emails using python – smtplib by johan godinho.

[11] Gokhale M, Wagh R, Chaudhari P, Khairnar S, Jadhav S (2018) Iot based e-tracking system for waste management. In: 2018 Fourth international conference on computing communication control and automation (ICCUBEA). IEEE, pp 1–6.

[12] Girish G. Desai, Prashant P. Bartakke, Real-Time Implementation Of Indian License Plate Recognition System IEEE Xplore 2019.