ABSTRACT

In this advancing world, traffic rule violations have become a main issue for many of the developed and developing nations. The number of vehicles is increasing rapidly, parallelly the number of traffic rule violations are increasing. After watching this traffic rules violation, this issue has now consistently been focused in many nations. Despite the job for the traffic rules executives or supervisors has become a challenge to identify and supervise the roads and pedestrians all the time. This emerged the need for building a prototype which can reduce the human intervention and make this job done easily. The primary goal of this prototype is to control the traffic rule violation precisely and economically.

Analysing the statistics of road traffic violations in India it can be concluded that there's a need for creation of a system that automatically detects traffic violations to improve safety on roads of the country. Therefore, the development of such algorithm for management of traffic violations on roadsides, crosswalks is in need and will be in demand soon.

The algorithm proceeds in multiple steps. They are vehicle detection at zebra crossings, car detection and pedestrian detection. For detections, I use Image Processing and Object Detection tools. The algorithm shows productive results in the detection of the violations of traffic rules.

ACRONYMS

IP	:	Image Processing
YOLO	:	You Only Look Once
R-CNN	:	Region based Convolutional Neural Networks
GUI	:	Graphical User Interface
OpenCV	:	Open-source Computer Vision library

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

1.1 INTRODUCTION

In this advancing world, traffic rule violations have become a main issue for many of the developed and developing nations. The number of vehicles is increasing rapidly, parallelly the number of traffic rule violations are increasing. After watching this traffic rules violation, this issue has now consistently been focused in many nations. the system, monitor traffic and act against the violations of traffic rules.

Despite the job for the traffic rules executives or supervisors has become a challenge to identify and supervise the roads and pedestrians all the time. This emerged the need for building a prototype which can reduce the human intervention and make this job done easily. The primary goal of this prototype is to control the traffic rule violation precisely and economically. According to reports, in 2015, there were about 5 lakh road accidents in India, which resulted in the death of 1.5 lakh people and about 5 lakh people were severely injured.

Taking apart the estimations of road insignificant criminal offenses in India it will in general be assumed that there's a necessity for development of a system that normally distinguishes negligible criminal offenses to improve prosperity on roads of the country. Henceforth, the improvement of such computation for the chiefs of criminal traffic offenses on roadside, crosswalks is up the creek without a paddle and will be famous soon.

This algorithm works in two stages. Toward the starting stage, it is important to recognize three classifications of items by the picture/video caught on the intersections. Zebra crossing, the moving vehicles and the third is the people on foot passing on the zebra crossing. After every one of the articles are found on each edge of the video or picture, at that point the subsequent stage starts. It is important to cross check the video arrangement outlines with the assigned articles at one another. Whenever found on a similar casing, the vehicle and the person on foot are at the zebra crossing, it implies the vehicle has abused the traffic guidelines.

1.2 OBJECTIVES

The goals of the project are

- To automate the traffic signal violation detection system
- To make it easy for the traffic police department to monitor the traffic
- To act against the violated vehicle owner in a fast and efficient way.
- Detecting and tracking the vehicle and their activities accurately is the main priority of the system.

1.3 METHODOLOGY

- First the video footage from the roadside is sent to the system.
- Vehicles are detected from the footage.
- Tracking the activity of vehicles, system determines if there is any violation or not. Figure 1 shows how the system works.
- The Graphical User Interface (GUI) makes the system interactive for the user to use.
- User can monitor the traffic footage and get the alert of violation with the detected bounding box of vehicle.
- User can take further action using the GUI.

CHAPTER 2

LITERATURE REVIEW

Many researchers have done research on Traffic violations detection and few of them are described in this chapter.

[1] M. Takatoo et. al. (1989) - This paper portrays a Traffic Flow Measurement System, TMS-II, in light of the vehicle group idea. The author states that “TMS-II is a framework which focuses on an ongoing and exact estimation of spatially dispersed traffic information, like vehicle areas, vehicle speeds, gridlock, line tail focuses, and so forth, through picture preparing”.

Presented in [2] is the implementation of real-time traffic violation detection in a monitoring stream which utilized simultaneous video stream from different cameras using parallel computing techniques. The author states that “This system proposes a detection algorithm which can discover various types of violations taking place on the roadways as well as in the parking lots. To achieve real-time analysis, parallel computing techniques are used in the implementation. An optimization scheme as well as a well-design data structure is proposed to improve the performance of the parallel implementation. Both real data and synthetic data are applied in the experiments. Experimental results demonstrate that the proposed system can discover all the violations from the high-throughput traffic monitoring stream in real-time”.

In [3], they used video-based traffic detection through an improved background-updating algorithm, thereafter, track the moving vehicles by feature based tracking method. The author states that “It can detect traffic violations, such as running red lights, speeding, and vehicle retrogress in real time. In this paper, we propose an improved background-updating algorithm by using wavelet transform on dynamic background, and then track moving vehicles by feature-based tracking method”.

This project is inspired by above project, but it is implemented using a self-developed approach. Conventionally vehicle detection is referred as an object detection problem. To detect moving vehicle objects from the road, YOLOv3 model is used which uses Darknet-53. After detecting vehicles, violation conditions are checked.

CHAPTER 3

IMPLEMENTATION

In this chapter, detailed description of algorithm, software and hardware requirements, flowchart of framework of system is given.

3.1 ALGORITHM

In our project, is done based on the analysis of YOLO (You Only Look Once) algorithm and data sets namely object detection, test_files, traffic_violation and car_identification.

3.1.1. YOLO (YOU ONLY LOOK ONCE) ALGORITHM

There are a couple of various algorithms for object identification, and they can be parted into two gatherings:

1. Algorithm reliant upon portrayal. They are executed in two stages. First and foremost, they select areas of interest in an image. Second, they orchestrate these regions using convolutional neural associations. This plan can be moderate since we should run assumptions for each picked area. A for the most part known representation of this kind of calculation is the Region-based convolutional neural association (RCNN) and its cousins Fast-RCNN, Faster-RCNN and the uttermost down the line development to the family: Mask-RCNN. Another model is RetinaNet.

2. Algorithm subject to backslide – as opposed to picking interesting bits of an image, they anticipate classes and bobbing encloses for the whole picture one run of the calculation. The two most mainstream models from this social event are the YOLO (You Only Look Once) family algorithm and SSD (Single Shot Multi-Box Detector). They are routinely used for continuous article area as, all things considered, they trade a hint of precision for tremendous overhauls in speed.

To grasp the YOLO algorithm, it is essential to set up the thing is being expected. Ultimately, we intend to anticipate a class of an article and the bouncing box deciding item region.

Each bounding box can be portrayed utilizing four descriptors:

1. center of a bounding box (**bxby**)
2. width (**bw**)
3. height (**bh**)
4. value **c**is corresponding to a class of an object (such as: car, traffic lights, etc.).

Advantages of YOLO

- The greatest benefit of utilizing YOLO is its amazing rate – it's staggeringly quick and can deal with 45 frames each second.
- YOLO likewise comprehends summed up object portrayal.
- This is probably the best algorithm for object identification and has shown a relatively comparative execution to the R-CNN algorithms.
- Unlike quicker RCNN, it's prepared to do characterization and bounding box relapse simultaneously.

Disadvantages of YOLO

- The limit of YOLO algorithm is that it battles with little items inside the picture, for instance it may experience issues in distinguishing a herd of birds. This is because of the spatial limitations of the algorithm.
- Comparatively low review and more restriction mistake contrasted with Faster R_CNN.
- Struggles to identify close objects in light of the fact that every lattice can propose just 2 bounding boxes

METHODOLOGY FOR YOLO ALGORITHM

Every Algorithm has its own method of performing its task. The Object detection algorithm based on YOLO is shown in Fig. 3.1

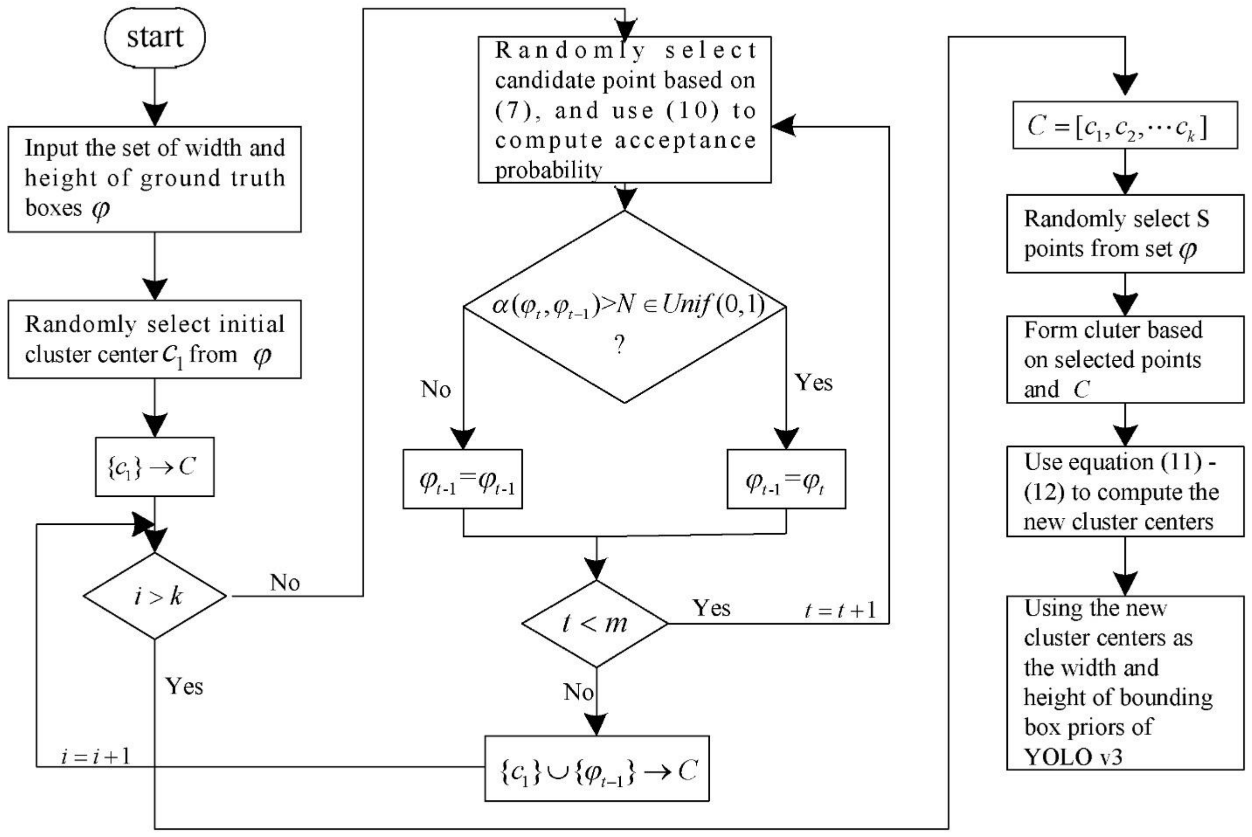

start
Input the set of width and height of ground truth boxes φ
Randomly select initial cluster center c_1 from φ
$\{c_1\} \rightarrow C$
$i > k$
No
Yes
$i = i + 1$
Randomly select candidate point based on (7), and use (10) to compute acceptance probability
$\alpha(\varphi_t, \varphi_{t-1}) > N \in Unif(0,1)$?
No
Yes
$\varphi_{t-1} = \varphi_{t-1}$
$\varphi_{t-1} = \varphi_t$
$t < m$
Yes
$t = t + 1$
No
$\{c_1\} \cup \{\varphi_{t-1}\} \rightarrow C$
$C = [c_1, c_2, \cdots c_k]$
Randomly select S points from set φ
Form cluter based on selected points and C
Use equation (11) - (12) to compute the new cluster centers
Using the new cluster centers as the width and height of bounding box priors of YOLO v3

Fig. 3.1 YOLO Algorithm

3.2 SOFTWARE REQUIREMENT SPECIFICATIONS (SRS)

A Software Requirements Specification (SRS) is an archive that depicts the idea of an undertaking, programming, or application. In basic words, the SRS archive is a manual of a venture gave it is set up before you launch a task/application. This archive is additionally known by the names SRS report, programming record. A product record is basically ready for an undertaking, programming or any sort of use.

There are a bunch of rules to be followed while setting up the product requirement specification archive. This incorporates the reason, scope, practical and non-functional prerequisites, programming and equipment necessities of the undertaking. Likewise, it additionally contains the data about natural conditions required, wellbeing and security prerequisites, programming quality ascribes of the undertaking, and so forth.

HARDWARE REQUIREMENTS:

Processor: Min. Core i3 processor
RAM: 2GB (Min.) or 8GB (Recommended)
Hard Disk Space: 50GB+

SOFTWARE REQUIREMENTS

Operating System: Windows 7 or later versions of windows
Libraries used for GUI: TKinter
Business Tier: Python (with dependencies - OpenCV, TensorFlow, Numpy etc.,)

3.3 SYSTEM OVERVIEW

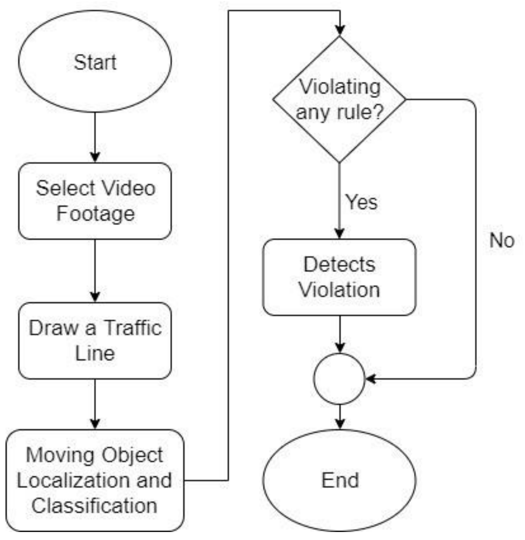


Fig. 3.2. System Overview.

The System consists of two main components -

- Vehicle detection model
- A graphical user interface (GUI)

First the video footage from the roadside is sent to the system. Vehicles are detected from the footage. Tracking the activity of vehicles, system determines if there is any violation or not. Figure 3.2 shows how the system works.

The Graphical User Interface (GUI) makes the system interactive for the user to use. User can monitor the traffic footage and get the alert of violation with the detected bounding box of vehicle. User can take further action using the GUI.

3.4 METHODOLOGY

3.4.1 Vehicle Classification

From the given video footage, moving objects are detected. An object detection model YOLOv3 is used to classify those moving objects into respective classes. YOLOv3 is the third item discovery algorithm in YOLO (You Only Look Once) family. It improved the exactness with numerous stunts and is more equipped for distinguishing objects. The classifier model is built with *Darknet-53* architecture. Table-1 shows how the neural network architecture is designed.

Features:

1. **Bounding Box Predictions:**

   YOLOv3 is a solitary organization the misfortune for objectiveness and grouping should be determined independently yet from a similar organization. YOLOv3 predicts the objectiveness score utilizing calculated relapse where 1 method complete cover of bounding box earlier over the ground truth object. It will anticipate just 1 bounding box earlier for one ground truth object and any blunder in this would bring about for both orders just as location misfortune. There would likewise be other bounding box priors which would have objectiveness score more than the limit however not exactly the best one. These blunders will just cause for the recognition misfortune and not for the grouping misfortune.

2. **Class Prediction:**

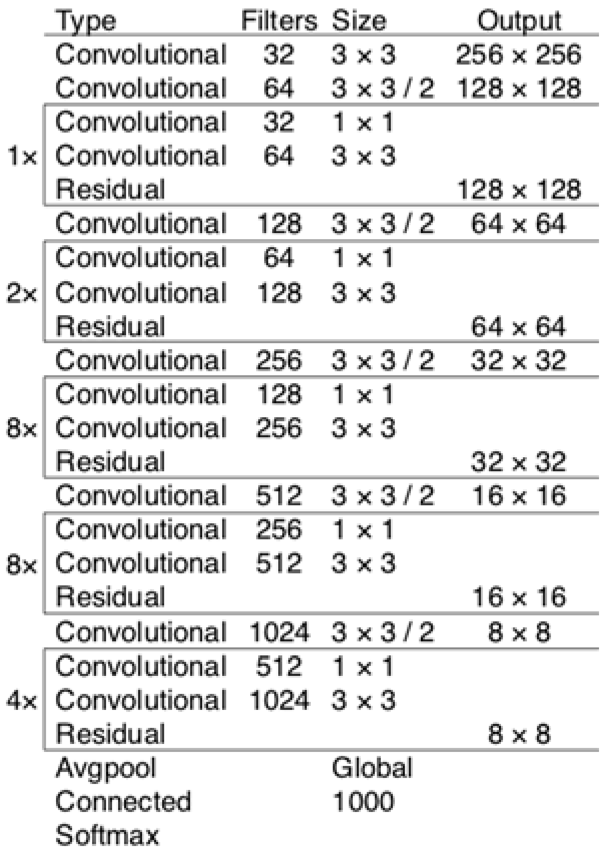
   YOLOv3 utilizes autonomous calculated classifiers for each class rather than a customary SoftMax layer. This is done to make the order multi-name characterization. Each crate predicts the classes the bouncing box may contain utilizing multilabel arrangement.

3. **Predictions across scales:**

   To support detection a varying scales YOLOv3 predicts boxes at 3 different scales. Then features are extracted from each scale by using a method similar to that of feature pyramid networks. YOLOv3 gains the ability to better predict at varying scales using the above method. The bounding box priors generated using dimension clusters are divided into 3 scales, so that there are 3 bounding box priors per scale and thus total 9 bounding box priors.

4. **Feature Extractor:**

   YOLOv3 uses a new network- **Darknet-53.** Darknet-53 has 53 convolutional layers, its more profound than YOLOv2 and it likewise has residuals or easy route associations. It's incredible than Darknet - 19 and more effective than ResNet-101 or ResNet-152.



	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Fig. 3.3. Darknet-53.

3.4.2 Violation Detection

The vehicles are detected using YOLOv3 model. After detecting the vehicles, violation cases are checked. A traffic line is drawn over the road in the preview of the given video footage by the user. The line specifies that the traffic light is red. Violation happens if any vehicle crosses the traffic line in red state.

The detected objects have a green bounding box. If any vehicle passes the traffic light in red state, violation happens. After detecting violation, the bounding box around the vehicle becomes red.

3.5 IMPLEMENTATION

Computer Vision

OpenCV is an open-source computer vision and machine learning software library which is used in this project for image processing purpose. Tensorflow is used for implementing the vehicle classifier with *darknet-53*.

Graphical User Interface (GUI)

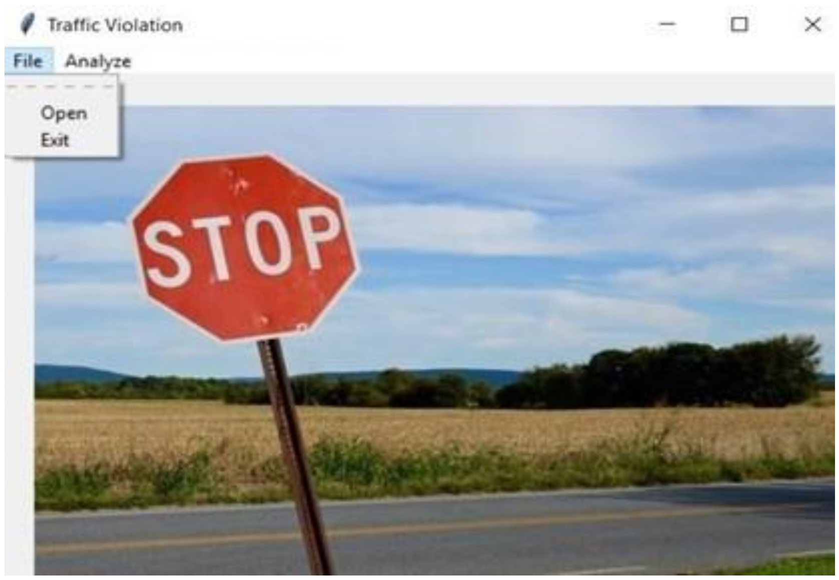


Fig. 3.4. Initial user interface view.

The graphical user interface has all the options needed for the software. The software serves administration and other debugging purposes. We don’t need to edit code for any management. For example, if we need to open any video footage, we can do it with the Open item (Figure-2).

Primarily, for the start of the project usage, the administrator needs to open a video footage using ‘Open’ item that can be found under ‘File’ (Fig. 3.2). The administrator can open any video footage from the storage files (Fig. 3.3).

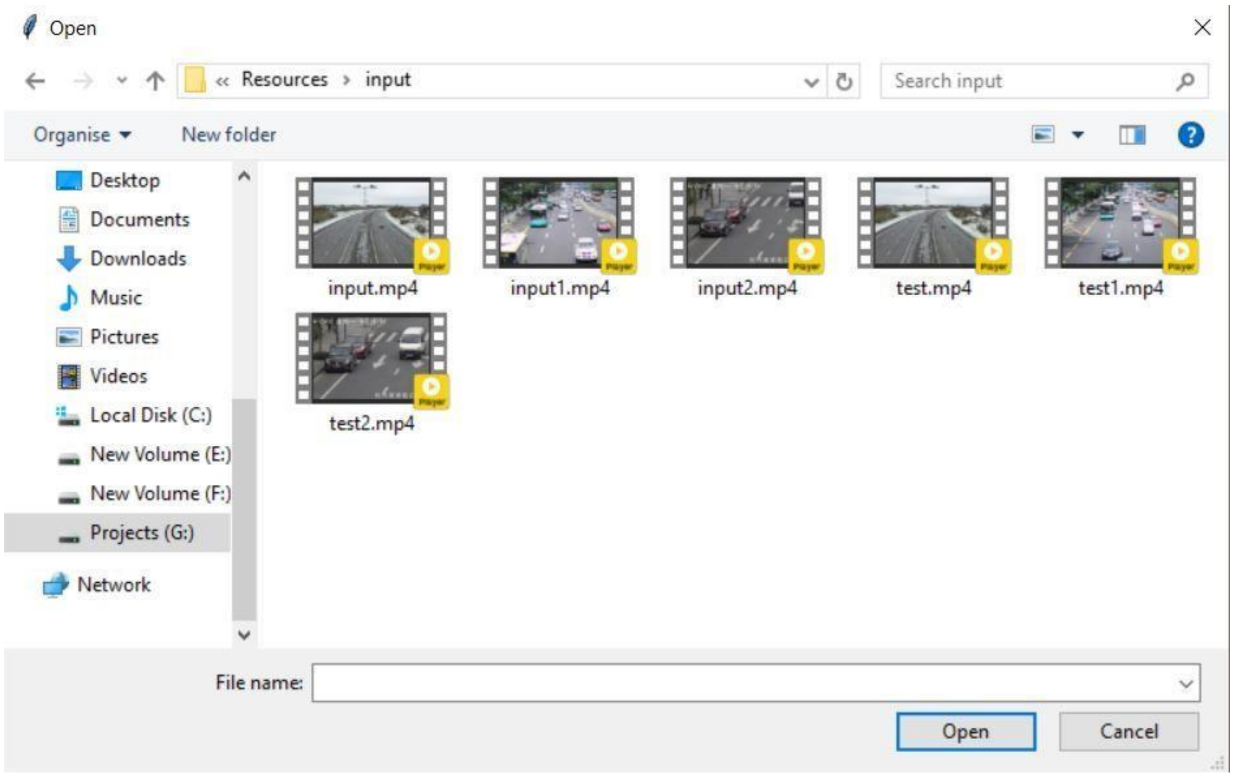


Fig. 3.5. Opening a video footage from storage.

After opening a video footage from storage, the system will get a preview of the footage. The preview contains a frame from the given video footage. The preview is used to identify roads and draw a traffic line over the road. The traffic line drawn by administrator will act as a traffic signal line. To enable the line drawing feature, we need to select ‘Region of interest’ item from the ‘Analyze’ option (Figure-4). After that administrator will need to select two points to draw a line that specifies traffic signal.

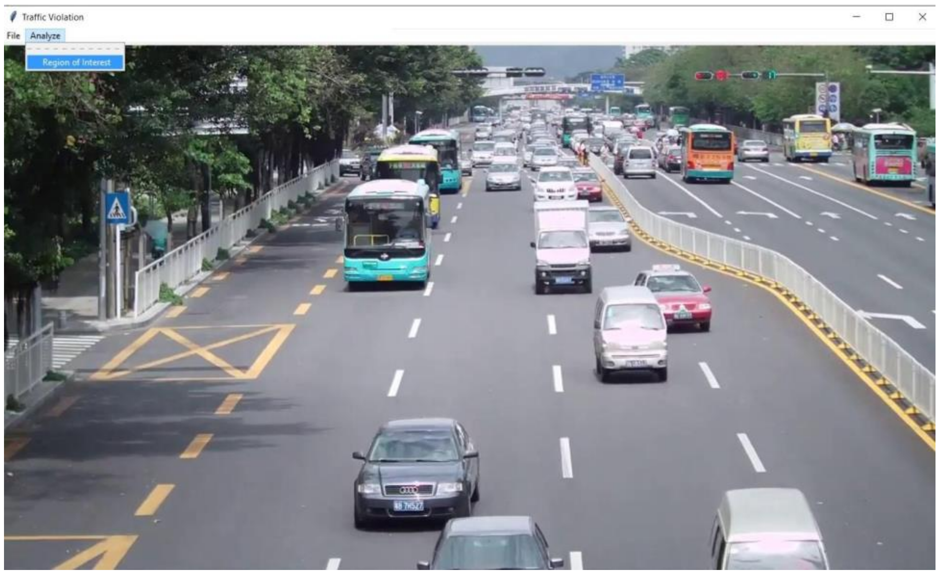
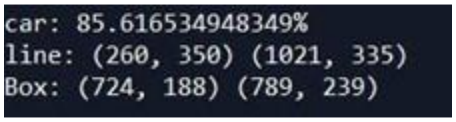


Fig. 3.6. Region of Interest (Drawing signal line)

Selecting the region of interest will start violation detection system. The coordinates of the line drawn will be shown on console (Fig. 3.5). The violation detection system will start immediately after the line is drawn. At first the weights will be loaded. Then the system will detect objects and check for violations. The output will be shown frame by frame from the GUI (Fig. 3.6).



```
car: 85.616534948349%
line: (260, 350) (1021, 335)
Box: (724, 188) (789, 239)
```

Fig. 3.7. Line Coordinates (from console)

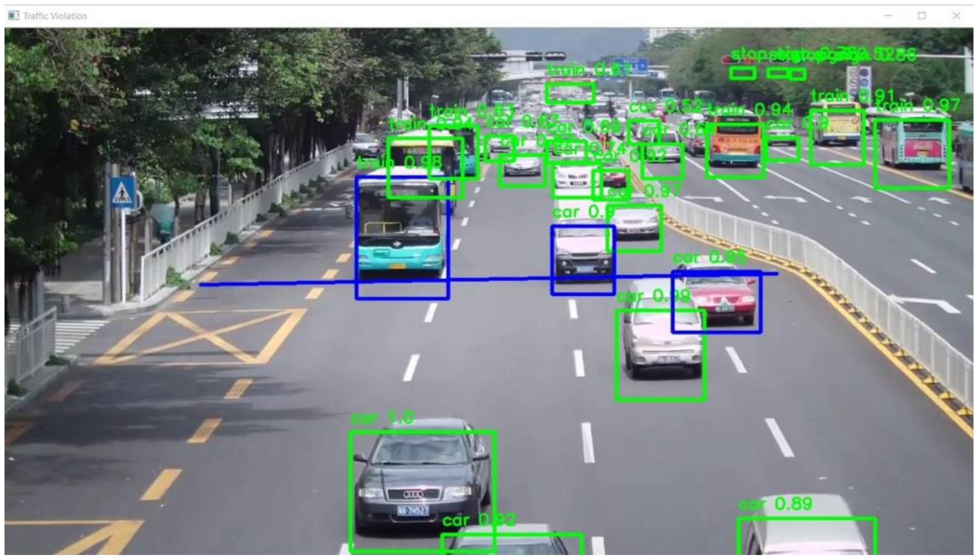


Fig. 3.8. Final Output (on each frame)

The system will show output until the last frame of the footage. In background a ‘output.mp4’ will be generated. The file will be in ‘output’ folder of ‘Resources’. The process will be immediately terminated by clicking ‘q’.
After processing a video footage, the administrator can add another video footage from the initial file manager. If the work is complete the administrator can quit using ‘Exit’ item from File option.

3.6 FLOW DIAGRAM

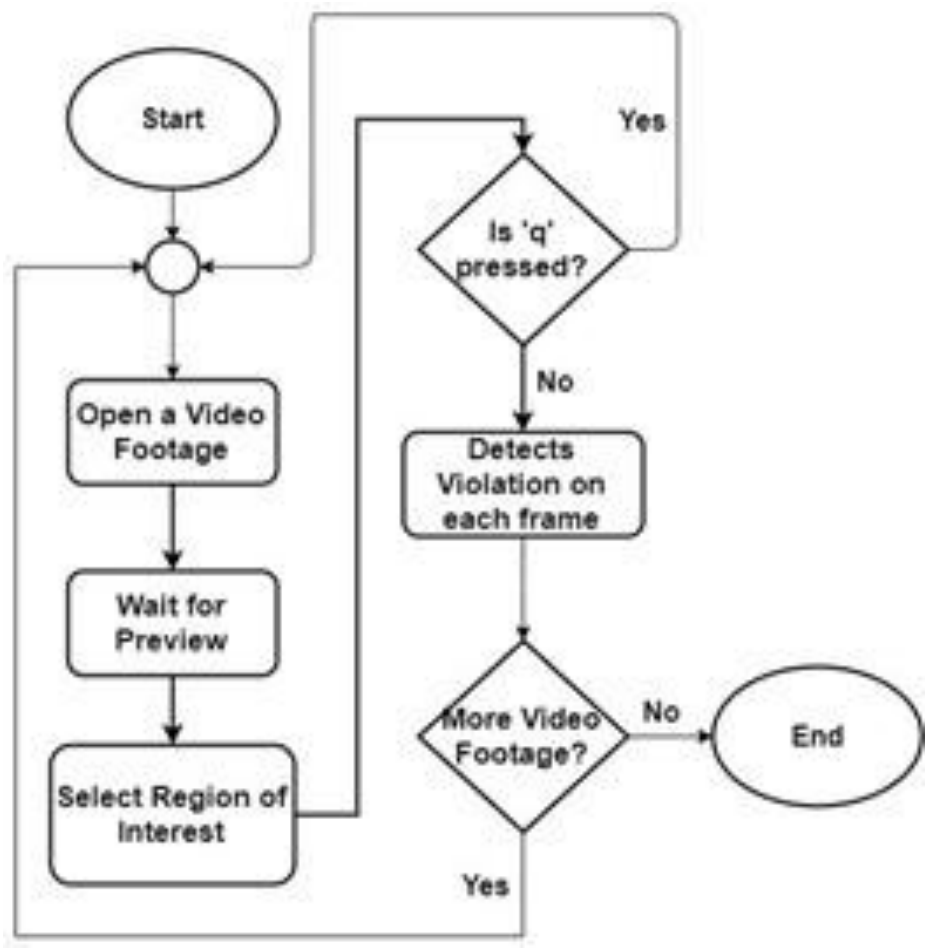


Fig. 3.9. Flow Diagram of the Software

CHAPTER 4

EXPERIMENTATION AND RESULTS

In this chapter, implementation of algorithm is described. Results will be explained in terms of tables and graphs etc.

4.1 EXPERIMENTAL WORK

In our project, Analysis of algorithms is done based on the data sets namely object detection, test_files, traffic_violation and car_identification. The algorithm used in the project is

- YOLOv3 (You Only Look Once)

4.1.1 IMPLEMENTATION OF YOLO ALGORITHM IN PYTHON

Project-GUI.py

```
from tkinter import *
from PIL import Image, ImageTk
from tkinter import filedialog
import object_detection as od
import imageio
import cv2

class Window(Frame):
    def __init__(self, master=None):
        Frame.__init__(self, master)

        self.master = master
        self.pos = []
        self.line = []
        self.rect = []
        self.master.title("GUI")
        self.pack(fill=BOTH, expand=1)
        self.counter = 0

        menu = Menu(self.master)
        self.master.config(menu=menu)

        file = Menu(menu)
        file.add_command(label="Open", command=self.open_file)
        file.add_command(label="Exit", command=self.client_exit)
        menu.add_cascade(label="File", menu=file)

        analyze = Menu(menu)
        analyze.add_command(label="Region of Interest", command=self.regionOfInterest)
        menu.add_cascade(label="Analyze", menu=analyze)

        self.filename = "Images/home.jpg"
        self.imgSize = Image.open(self.filename)
        self.tkimage =  ImageTk.PhotoImage(self.imgSize)
        self.w, self.h = (1366, 768)

        self.canvas = Canvas(master = root, width = self.w, height = self.h)
        self.canvas.create_image(20, 20, image=self.tkimage, anchor='nw')
        self.canvas.pack()

    def open_file(self):
        self.filename = filedialog.askopenfilename()

        cap = cv2.VideoCapture(self.filename)

        reader = imageio.get_reader(self.filename)
        fps = reader.get_meta_data()['fps']
    ret, image = cap.read()
    cv2.imwrite('E:\Traffic Violation\Images\preview.jpg', image)

    self.show_image('E:\Traffic Violation\Images\preview.jpg')



def show_image(self, frame):
    self.imgSize = Image.open(frame)
    self.tkimage =  ImageTk.PhotoImage(self.imgSize)
    self.w, self.h = (1366, 768)

    self.canvas.destroy()

    self.canvas = Canvas(master = root, width = self.w, height = self.h)
    self.canvas.create_image(0, 0, image=self.tkimage, anchor='nw')
    self.canvas.pack()

def regionOfInterest(self):
    root.config(cursor="plus")
    self.canvas.bind("<Button-1>", self.imgClick)

def client_exit(self):
    exit()

def imgClick(self, event):

    if self.counter < 2:
        x = int(self.canvas.canvasx(event.x))
        y = int(self.canvas.canvasy(event.y))
        self.line.append((x, y))
        self.pos.append(self.canvas.create_line(x - 5, y, x + 5, y, fill="red", tags="crosshair"))
        self.pos.append(self.canvas.create_line(x, y - 5, x, y + 5, fill="red", tags="crosshair"))
        self.counter += 1

    # elif self.counter < 4:
    #     x = int(self.canvas.canvasx(event.x))
    #     y = int(self.canvas.canvasy(event.y))
    #     self.rect.append((x, y))
    #     self.pos.append(self.canvas.create_line(x - 5, y, x + 5, y, fill="red", tags="crosshair"))
    #     self.pos.append(self.canvas.create_line(x, y - 5, x, y + 5, fill="red", tags="crosshair"))
    #     self.counter += 1

    if self.counter == 2:
        #unbinding action with mouse-click
        self.canvas.unbind("<Button-1>")
        root.config(cursor="arrow")
        self.counter = 0

        #show created virtual line
        print(self.line)
        print(self.rect)
        img = cv2.imread('E:\Traffic Violation\Images\preview.jpg')
        cv2.line(img, self.line[0], self.line[1], (0, 255, 0), 3)
        cv2.imwrite('E:\Traffic Violation\Images\copy.jpg', img)
        self.show_image('E:\Traffic Violation\Images\copy.jpg')

        ## for demonstration
        # (rxmin, rymin) = self.rect[0]
        # (rxmax, rymax) = self.rect[1]

        # tf = False
        # tf |= self.intersection(self.line[0], self.line[1], (rxmin, rymin), (rxmin, rymax))
        # print(tf)
        # tf |= self.intersection(self.line[0], self.line[1], (rxmax, rymin), (rxmax, rymax))
```

```
        # print(tf)
        # tf |= self.intersection(self.line[0], self.line[1], (rxmin, rymin), (rxmax, rymin))
        # print(tf)
        # tf |= self.intersection(self.line[0], self.line[1], (rxmin, rymax), (rxmax, rymax))
        # print(tf)

        # cv2.line(img, self.line[0], self.line[1], (0, 255, 0), 3)

        # if tf:
        #     cv2.rectangle(img, (rxmin,rymin), (rxmax,rymax), (255,0,0), 3)
        # else:
        #     cv2.rectangle(img, (rxmin,rymin), (rxmax,rymax), (0,255,0), 3)

        # cv2.imshow('traffic violation', img)

        #image processing
        self.main_process()
        print("Executed Successfully!!!")

        #clearing things
        self.line.clear()
        self.rect.clear()
        for i in self.pos:
            self.canvas.delete(i)

    def intersection(self, p, q, r, t):
        print(p, q, r, t)
        (x1, y1) = p
        (x2, y2) = q

        (x3, y3) = r
        (x4, y4) = t
```

```
a1 = y1-y2
b1 = x2-x1
c1 = x1*y2-x2*y1

a2 = y3-y4
b2 = x4-x3
c2 = x3*y4-x4*y3

if(a1*b2-a2*b1 == 0):
    return False
print((a1, b1, c1), (a2, b2, c2))
x = (b1*c2 - b2*c1) / (a1*b2 - a2*b1)
y = (a2*c1 - a1*c2) / (a1*b2 - a2*b1)
print((x, y))

if x1 > x2:
    tmp = x1
    x1 = x2
    x2 = tmp
if y1 > y2:
    tmp = y1
    y1 = y2
    y2 = tmp
if x3 > x4:
    tmp = x3
    x3 = x4
    x4 = tmp
if y3 > y4:
    tmp = y3
    y3 = y4
    y4 = tmp
```

```
        if x >= x1 and x <= x2 and y >= y1 and y <= y2 and x >= x3 and x <= x4 and y >= y3 and y <= y4:
            return True
        else:
            return False

    def main_process(self):

        video_src = self.filename

        cap = cv2.VideoCapture(video_src)

        reader = imageio.get_reader(video_src)
        fps = reader.get_meta_data()['fps']
        writer = imageio.get_writer('E:\Traffic Violation\Resources\output\output.mp4', fps = fps)

        j = 1
        while True:
            ret, image = cap.read()

            if (type(image) == type(None)):
                writer.close()
                break

            image_h, image_w, _ = image.shape
            new_image = od.preprocess_input(image, od.net_h, od.net_w)

            # run the prediction
            yolos = od.yolov3.predict(new_image)
            boxes = []
            for i in range(len(yolos)):
                # decode the output of the network
                boxes  +=  od.decode_netout(yolos[i][0],  od.anchors[i],  od.obj_thresh,  od.nms_thresh,
od.net_h, od.net_w)

            # correct the sizes of the bounding boxes
            od.correct_yolo_boxes(boxes, image_h, image_w, od.net_h, od.net_w)

            # suppress non-maximal boxes
            od.do_nms(boxes, od.nms_thresh)

            # draw bounding boxes on the image using labels
            image2 = od.draw_boxes(image, boxes, self.line, od.labels, od.obj_thresh, j)

            writer.append_data(image2)

            # cv2.imwrite('E:\Traffic Violation\Images\frame'+str(j)+'.jpg', image2)
            # self.show_image('E:\Traffic Violation\Images\frame'+str(j)+'.jpg')

            cv2.imshow('Traffic Violation', image2)

            print(j)

            if cv2.waitKey(10) & 0xFF == ord('q'):
                writer.close()
                break

            j = j+1

        cv2.destroyAllWindows()

root = Tk()
```

```
app = Window(root)
root.geometry("%dx%d"%(535, 380))
root.title("Traffic Violation")

root.mainloop()
```

4.2 RESULTS AND DISCUSSION

In this chapter, detailed output analysis of YOLO algorithm is described.

4.2.1 TRAINING YOLOv3

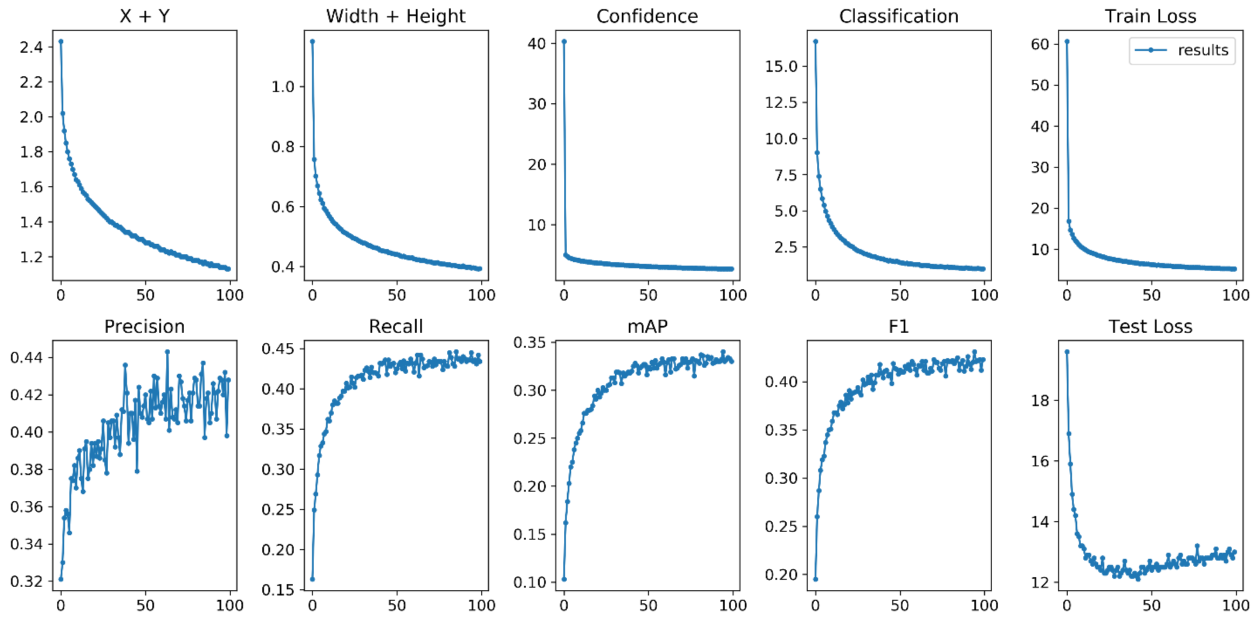


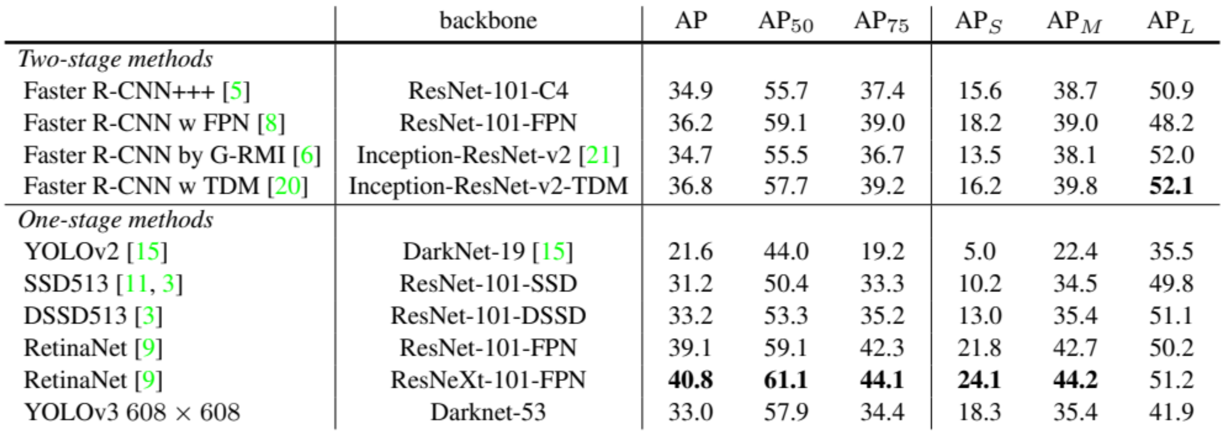
Fig. 4.1. Training YOLO algorithm

4.2.2 ANALYSIS OF ALGORITHM

YOLOv3 Performance

- YOLOv3 predicts 10x occasions more bounding box than YOLOv2. YOLOv3 utilizes nine anchor boxes.
- This model distinguishes different articles in an edge along their bounding boxes.

The working pattern and performance of YOLOv3 algorithm are shown in the below figures.



	backbone	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Two-stage methods							
Faster R-CNN+++ [5]	ResNet-101-C4	34.9	55.7	37.4	15.6	38.7	50.9
Faster R-CNN w FPN [8]	ResNet-101-FPN	36.2	59.1	39.0	18.2	39.0	48.2
Faster R-CNN by G-RMI [6]	Inception-ResNet-v2 [21]	34.7	55.5	36.7	13.5	38.1	52.0
Faster R-CNN w TDM [20]	Inception-ResNet-v2-TDM	36.8	57.7	39.2	16.2	39.8	**52.1**
One-stage methods							
YOLOv2 [15]	DarkNet-19 [15]	21.6	44.0	19.2	5.0	22.4	35.5
SSD513 [11, 3]	ResNet-101-SSD	31.2	50.4	33.3	10.2	34.5	49.8
DSSD513 [3]	ResNet-101-DSSD	33.2	53.3	35.2	13.0	35.4	51.1
RetinaNet [9]	ResNet-101-FPN	39.1	59.1	42.3	21.8	42.7	50.2
RetinaNet [9]	ResNeXt-101-FPN	**40.8**	**61.1**	**44.1**	**24.1**	**44.2**	51.2
YOLOv3 608 × 608	Darknet-53	33.0	57.9	34.4	18.3	35.4	41.9

Fig. 4.2. Object Detection algorithms and their working strengths

As shown above, compared with RetinaNet, YOLOv3 got comparable mAP@0.5 with much faster inference time.

- For overall mAP, YOLOv3 performance is dropped significantly.
- Nevertheless, YOLOv3–608 got 33.0% mAP in 51ms inference time while RetinaNet-101–50–500 only got 32.5% mAP in 73ms inference time.
- And YOLOv3 is on par with SSD variants with 3× faster.
- YOLOv3 is much better than SSD and has similar performance as DSSD.

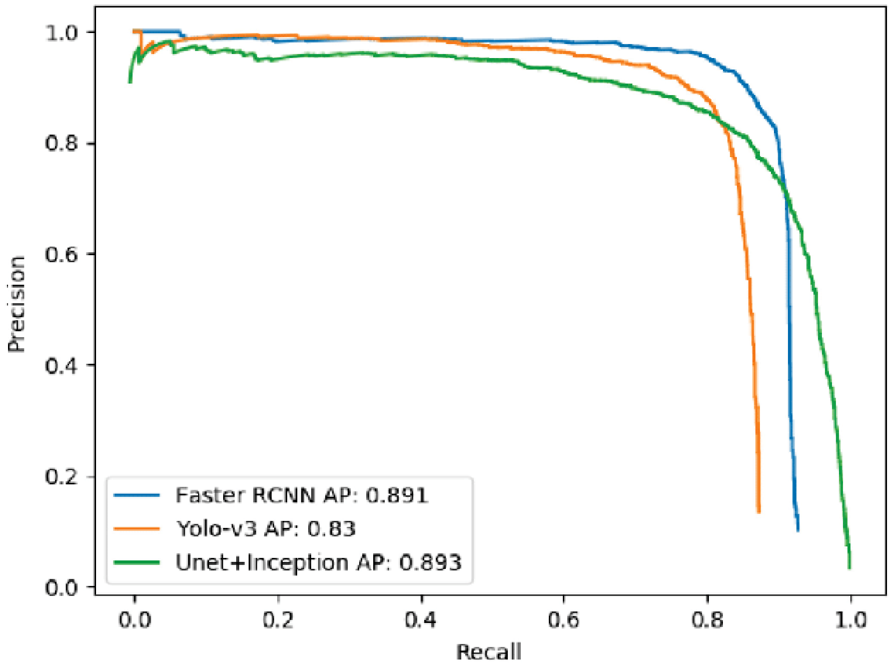


Fig. 4.3. Performance comparison of R-CNN, YOLOv3, and Inception algorithms

CHAPTER 5

CONCLUSION AND FUTURE SCOPE

5.1 CONCLUSIONS

The planned algorithm was successfully ready to identify the kind of violations determined on this venture which are denying traffic light. The assembly of recognition for the criminal traffic offense referenced is divergent since it has an alternate limit condition. The framework gives recognition to traffic light violations. Further, the framework can handle each information in turn. Likewise, the program runtime is fairly lethargic, and can be improved by utilizing a PC with fast processor determinations or GPU. The algorithm continues in different advances. They are vehicle discovery at zebra intersections, vehicles identification and walker recognition. The algorithm shows useful outcomes in the identification of the violations of traffic rules.

5.2 FUTURE SCOPE

Future exploration about the utilization of the planned algorithm for other progressed picture preparing methods. Since, this may improve the program runtime of the framework by dismissing other pointless advances done in a foundation distinction strategy. A PC vision algorithm might be done rather to give more insight in the framework. Our future plan is to implement the number plate detection with OCR support to make this system more robust.

REFERENCES

[1] M. Takatoo, T. Kitamura, Y. Okuyama, Y. Kobayashi, K. Kikuchi, H. Nakanishi & T. Shibata, Traffic flow measuring system using image processing. Proc. SPIE, 1989, Vol. 1197, 172-180.

[2] G. Ou, Y. Gao, and Y. Liu, "Real Time Vehicular Traffic Violation Detection in Traffic Monitoring System," in 2012 IEEE/WIC/ACM, Beijing, China, 2012.

[3] X. Wang, L.-M. Meng, B. Zhang, J. Lu and K.-L.Du, "A Video-based Traffic Violation Detection System," in MEC, Shenyang, China, 2013.

[4] Abdullah Asım, et al, "A Vehicle Detection Approach using Deep Learning Methodologies."

[5] Aaron Christian P. Uy, Rhen Anjerome Bedruz, Ana Riza Quiros, argel Bandala, Elmer P. Dadios, "Machine vision for traffic violation detection system through genetic algorithm", 2015 International Conference on Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment and Management (HNICEM), 2015