TRAFFIC VIOLATION DETECTION SYSTEM

# A PROJECT REPORT

***Submitted by***

# B. NANDHAGOPAL (913120205036)

**S. KALIDASS (913120205025)**

***In Partial fulfillment for the award of the degree of***

# BACHELOR OF TECHNOLOGY

**in INFORMATION TECHNOLOGY**

# VELAMMAL COLLEGE OF ENGINEERING AND TECHNOLOGY, MADURAI-625 009

**ANNA UNIVERSITY :: CHENNAI 600 025**

**MAY 2023**

## VELAMMAL COLLEGE OF ENGINEERING AND TECHNOLOGY, VIRAGANOOR, MADURAI

**DEPARTMENT OF INFORMATION TECHNOLOGY ANNA UNIVERSITY: CHENNAI 600 025**

## BONAFIDE CERTIFICATE

Certified that this project report **“TRAFFIC VIOLATION DETECTION**” is thebonafide work of “**B.NANDHAGOPAL (913120205036), S.KALIDASS (913120205025)** who carriedout the project work under my supervision.

## HEAD OF DEPARTMENT

Dr. R. Kavitha Professor and Head

Department of Information Technology

Velammal College of Engineering and Technology, Madurai – 6250 009

## SUPERVISOR

DR.P.Umamaheswari Assistant Professor

Department of Information Technology

Velammal College of Engineering and Technology, Madurai – 6250 009

Submitted for the Project Viva-Voce Examination held on at Velammal College of

Engineering and Technology, Madurai.

INTERNAL EXAMINER EXTERNAL EXAMINER

## ABSTRACT

Traffic infractions may result from an increase in the number of vehicles in metropolitan areas. These infractions include reckless driving, exceeding the speed limit, signal jumping, hit-and-run, riding without a helmet, using a phone while driving, parking in the wrong spot, and going in the wrong direction. Tracking these behaviours becomes more difficult and time-consuming. Due to the severe property damage and subsequent accidents, people's lives are put in danger. Automatic traffic violation detection systems can be installed in high-traffic areas of metropolitan cities to prevent such unimaginable effects and to offer appropriate solutions for traffic violations. For which the system constantly enforces appropriate traffic laws and detains those who disobey them. The proposed system includes a traffic infraction monitoring system, CCTV cameras, and an alert messaging system To monitor number plates and vehicle speed, CCTV cameras include automatic number plate recognition cameras, speed cameras and red light cameras. The system comes with a user-friendly graphical interface to make it easy for the user to run the system, monitor traffic, and take enforcement action against traffic rule infractions.

# TABLE OF CONTENTS

CHAPTER NO TITLE PAGE NO ABSTRACT iii

[LIST OF TABLES v](#_TOC_250001)

[LIST OF FIGURES vi](#_TOC_250000)

1. INTRODUCTION 1
   1. Overview
   2. Software Requirements.
   3. Hardware Requirements

# LITERATURE SURVEY 3

1. **METHODOLGY 4**
   1. Proposed System
   2. Image Processing
   3. Object Detection
   4. Object Classification
   5. CNN Algorithm
   6. YOLO Algorithm

# IMPLEMENTATION 8

* 1. Development Tools
     1. Anaconda
     2. Visual Studio Code
  2. Result

# CONCLUSION 11

1. **CODE 12**

# BIBLIOGRAPHY 21

# LIST OF TABLES

## TABLE PAGE NO

1. Darknet-53 architecture 4
2. Comparison B/W YOLO v2 & v3
3. YOLOv3 Charecteristics
4. YOLOv3 Efficiency

# LIST OF FIGURES

## Figure

1. System Architecture

## PAGE NO

6

1. YOLOv3 Architecture 10
2. YOLOv3 Performance 11
3. GUI 14

# CHAPTER 1

# INTRODUCTION

**1.1. Overview**

Traffic Violation Detection System utilizes object detection algorithms and Convolutional Neural Networks (CNNs) to accurately identify and monitor traffic violations on roads and highways.

Convolutional Neural Networks (CNNs) are a type of deep learning model widely used in computer vision tasks, including image classification, object detection, and image segmentation. CNNs are specifically designed to process and analyze visual data, such as images or videos, by automatically learning hierarchical patterns and features directly from the data.

You Only Look Once (YOLO) is a popular real-time object detection algorithm. Unlike traditional object detection algorithms that use region proposal techniques, YOLO performs object detection in a single pass of the network, resulting in faster processing times while maintaining reasonable accuracy.

CNN and YOLO play crucial roles in traffic violation detection systems, working in tandem to accurately identify and classify violations. Convolutional Neural Networks (CNNs) are employed to extract meaningful features from images or video frames, enabling the system to analyze visual patterns and learn representations of objects and violations. CNNs excel in feature extraction and classification tasks, allowing them to differentiate between normal traffic scenarios and violations such as red light running, speeding, or illegal lane changes. By leveraging their ability to learn complex patterns, CNNs contribute to the accuracy of the system.

On the other hand, You Only Look Once (YOLO) plays a significant role in efficient and real-time object detection for traffic violations. YOLO's architecture enables fast detection by dividing the input image into a grid and predicting bounding boxes and class probabilities for objects within each grid cell. This single pass detection approach eliminates the need for multiple stages or proposals, allowing for real-time processing of video streams or image sequences. YOLO's ability to accurately localize objects within the image aids in identifying the location and extent of violations, providing valuable information for subsequent enforcement actions. With its real-time monitoring capabilities, YOLO enables prompt identification of violations, facilitating immediate response and intervention by authorities.

By combining CNNs for feature extraction and classification with YOLO for efficient and real-time object detection, traffic violation detection systems can effectively detect and classify various violations in traffic scenarios. These technologies contribute to improved road safety, better enforcement of traffic regulations, and timely intervention by authorities to ensure a smoother and safer traffic flow.

# Software Requirements

* + 1. **OS** : Windows 10
    2. **Language** : Python
    3. **IDE** : Visual Studio.

# Hardware Requirements

* + 1. **RAM** : Atleast 4 GB.
    2. **Storage** : Atleast 1 GB.

# CHAPTER 2

**LITERATURE SURVEY**

1. **"Traffic Violation Detection and Vehicle Classification Using Deep Learning" by S. Shetkar et al. (2018):** This paper presents a system that combines deep learning techniques with image processing algorithms for traffic violation detection and vehicle classification. The authors employ a CNN for feature extraction and violation classification.
2. **"Traffic Violation Detection Using Deep Learning" by H. Shakhatreh et al. (2017):** The authors propose a deep learning-based approach for traffic violation detection. They use a convolutional neural network (CNN) to extract features from video frames and classify violations such as running red lights, speeding, and wrong-way driving.
3. **"Vehicle Tracking and Traffic Violation Detection Based on Surveillance Video" by W. Yao and Z. Yuan (2016):** This paper introduces a framework for vehicle tracking and traffic violation detection using surveillance video. The system employs background subtraction, object tracking, and rule-based violation detection algorithms.
4. **"Real-Time Traffic Violation Detection Using Surveillance Cameras" by K. Simonyan and A. Zisserman (2015):** The authors propose a method for real-time traffic violation detection using surveillance cameras. The system is capable of detecting violations such as running red lights, wrong-way driving, and illegal U-turns.
5. **"Vehicle Detection and Traffic Violation Detection System Based on Computer Vision" by Wei Liu et al. (2014):** This paper presents a system that utilizes computer vision techniques for vehicle detection and traffic violation detection. It focuses on detecting red light running and wrong-way driving.

## "Traffic Violation Detection System based on Deep Learning Techniques," by J. Gómez and D. G.

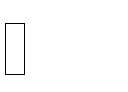
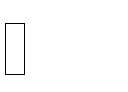
**C. Chávez, (2019):** The paper discusses an integrated multi-object tracking system that utilizes deep learning techniques for autonomous vehicles. Unfortunately, without access to the full paper, I cannot provide more specific details about its content.

1. **“Traffic Signal Violation Detection using Artificial Intelligence and Deep Learning”By Ruben J Franklin; Mohana (2020):**The paper uses YOLOV3 object detection for traffic violation detections such as signal jump, vehicle speed, and the number of vehicles.
2. **“Traffic Violation Detection via Depth and Gradient Angle Change”, by Yan-Tsung Peng; Chen- Yu Liu; He-Hao Liao; Wei-Cheng Lien; Gee-Sern Jison Hsu(2022):** The paper aims to develop an effective traffic violation detection system to detect traffic violations automatically from videos reported by the public, especially for case footage recorded by dashcams facing forward mounted on vehicles or helmets.
3. **“Vehicle Classification and Violation Detection on Traffic Light Area using BLOB and Mean-Shift Tracking Method”, By Mochamad Mobed Bachtiar; Achmad Rahman Mawardi; Adnan Rachmat Anom Besari(2020):** In this study, a vision system was built to recognize vehicles (cars and motorcycles) and to recognize violations committed by drivers at road intersections. Vehicle violation data is taken from CCTV cameras installed at road junctions. the system works for vehicle detection and traffic violation detection.
4. **“Traffic Rules Violation Detection using Machine Learning Techniques”, By P.Srinivas Reddy; T. Nishwa; R. Shiva Kiran Reddy; Ch Sadviq; K. Rithvik(2021):**In this paper the system can detect most common types of traffic violations in real-time through computer vision techniques and it also leverages good results with great accuracy.

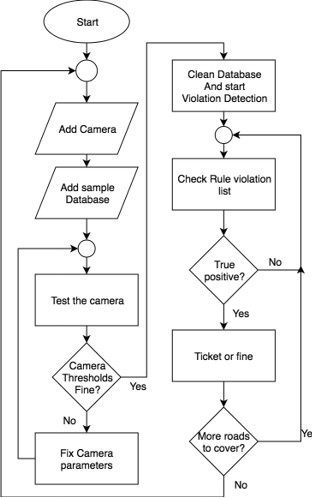
# CHAPTER 3

**METHODOLOGY**

# PROPOSED SYSTEM

 The proposed model necessitates two things in particular -Vehicle detection process  Graphical User Interface [GUI]

The system will receive the CCTV camera recordings that were made in various locations. The video will allow the identification of vehicles. The violation will be picked up each time the suggested programme evaluates the video. Signal violation is supported by the suggested programme by utilising the R-CNN algorithm. We use Tkinter because it has a very interactive graphical user interface. Police officer can take care and track the traffic footage and get the details of violation with the captured vehicle image. User can take further action like manually noting license numbers which violates traffic laws and transmit to neighbouring police stations to pursue further actions.



**Fig.1:System Architecture**

# IMAGE PROCESSING:

**GRAY SCALING AND BLURRING:**

Pre-processing the input video is essential for getting the best accuracy, and the video is blurred and grayscaled using the Gaussian blur approach.

Grey scaling has been applied to provide the best accuracy and reduce noise.

## BACKGROUND SUBTRACTION

Background subtraction has been used to subtract the current frame from the reference frame, yielding the necessary object's area.

The process is shown in Equation (1). dist(I) = saturate(|frame1(I) − frame2(I)|)

## BINARY THRESHOLD

The binarization approach has been used to clean up the input video's noise and other imperfections.

Through this procedure, holes and noises are eliminated. Figure 2 depicts the binary threshold procedure.

if frame(x, y) > threshold, dist(x, y) = MaxVal; otherwise

## o DILATION AND FIND THE CONTOUR

When we obtain the threshold image, we must dilate to fill in the gaps. The contour is then calculated to reform the better image.

# OBJECT DETECTION:

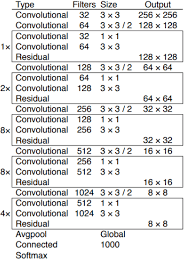
Regions with CNN features. Three-stage approach: -

* + 1. By using support vendor machine (SVM), wecan extract the objects from images.
    2. By using convolutional neural network (CNN) we can extract the features from each region of image.
    3. Classify and categorize each region using SVMs.

# OBJECT CLASSIFICATION:

The moving items are extracted from the image after preprocessing. A vehicle categorization model divides the moving items into four categories: vehicles, non-vehicles, and things with four, two, and three wheels.

Based on a neural network model, the following classification



**Table.1:Darknet-53 architecture**

# CONVOLUTIONAL NEURAL NETWORK(CNN) :

A Convolutional Neural Network (CNN) is a type of deep learning algorithm that is particularly well-suited for image recognition and processing tasks. It is made up of multiple layers, including convolutional layers, pooling layers, and fully connected layers.

The convolutional layers are the key component of a CNN, where filters are applied to the input image to extract features such as edges, textures, and shapes. The output of the convolutional layers is then passed through pooling layers, which are used to down-sample the feature maps, reducing the spatial dimensions while retaining the most important information. The output of the pooling layers is then passed through one or more fully connected layers, which are used to make a prediction or classify the image.

CNNs are trained using a large dataset of labeled images, where the network learns to recognize patterns and features that are associated with specific objects or classes. Once trained, a CNN can be used to classify new images, or extract features for use in other applications such as object detection or image segmentation.

CNNs have achieved state-of-the-art performance on a wide range of image recognition tasks, including object classification, object detection, and image segmentation. They are widely used in computer vision, image processing, and other related fields, and have been applied to a wide range of applications, including self-driving cars, medical imaging, and security systems.

A convolutional neural network, or CNN, is a deep learning neural network sketched for processing structured arrays of data such as portrayals.

CNN are very satisfactory at picking up on design in the input image, such as lines, gradients, circles, or even eyes and faces.

This characteristic that makes convolutional neural network so robust for computer vision. CNN can run directly on a underdone image and do not need any preprocessing.

A convolutional neural network is a feed forward neural network, seldom with up to 20.

The strength of a convolutional neural network comes from a particular kind of layer called the convolutional layer.

CNN contains many convolutional layers assembled on top of each other, each one competent of recognizing more sophisticated shapes.

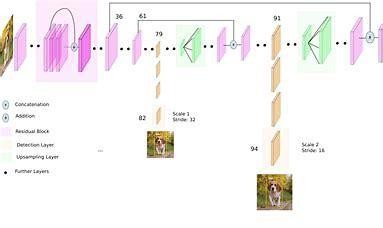
With three or four convolutional layers it is viable to recognize handwritten digits and with 25 layers it is possible to differentiate human faces.

The agenda for this sphere is to activate machines to view the world as humans do, perceive it in a alike fashion and even use the knowledge for a multitude of duty such as image and video recognition, image inspection and classification, media recreation, recommendation systems, natural language processing, etc.

# YOLO ALGORITHM:

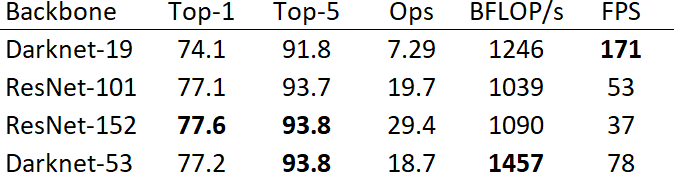
YOLOv3 (You Only Look Once, Version 3) is a real-time object detection algorithm that identifies specific objects in videos, live feeds, or images. The YOLO machine learning algorithm uses features learned by a deep convolutional neural network to detect an object.

YOLO is a Convolutional Neural Network (CNN) for performing object detection in real-time. CNNs are classifier-based systems that can process input images as structured arrays of data and recognize patterns between them (view image below). YOLO has the advantage of being much faster than other networks and still maintains accuracy. It allows the model to look at the whole image at test time, so its predictions are informed by the global context in the image. YOLO and other convolutional neural network algorithms “score” regions based on their similarities to predefined classes. High-scoring regions are noted as positive detections of whatever class they most closely identify with. For example, in a live feed of traffic, YOLO can be used to detect different kinds of vehicles depending on which regions of the video score highly in comparison to predefined classes of vehicles.



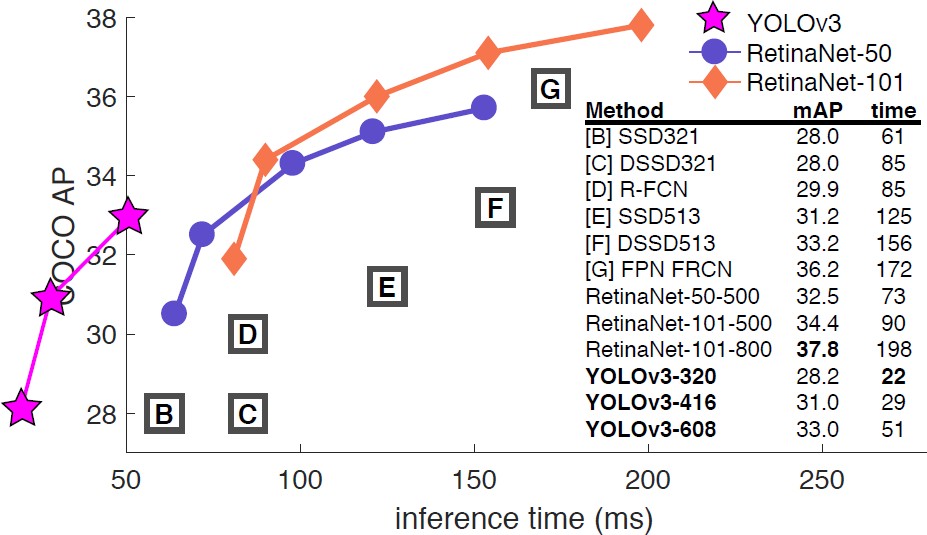
**Fig.2:YOLO v3 Architecture**

YOLOv2 was using Darknet-19 as its backbone feature extractor, while YOLOv3 now uses Darknet-53. Darknet-53 is a backbone also made by the YOLO creators Joseph Redmon and Ali Farhadi. Darknet-53 has 53 convolutional layers instead of the previous 19, making it more powerful than Darknet-19 and more efficient than competing backbones (ResNet-101 or ResNet-152).



## Table.2:Comparison B/W YOLO v2 & v3

Using the chart provided in the YOLOv3 paper by Redmon and Farhadi, we can see that Darknet-52 is 1.5 times faster than ResNet101. The depicted accuracy doesn’t entail any trade-off between accuracy and speed between Darknet backbones either since it is still as accurate as ResNet-152 yet two times faster. YOLOv3 is fast and accurate in terms of mean average precision (mAP) and intersection over union (IOU) values as well. It runs significantly faster than other detection methods with comparable performance (hence the name – You only look once). Moreover, you can easily trade-off between speed and accuracy simply by changing the model’s size, without the need for model retraining.



**Fig.3:YOLO v3 Performance**

# CHAPTER 4

**IMPLEMENTATION**

# Development Tools

## Anaconda:

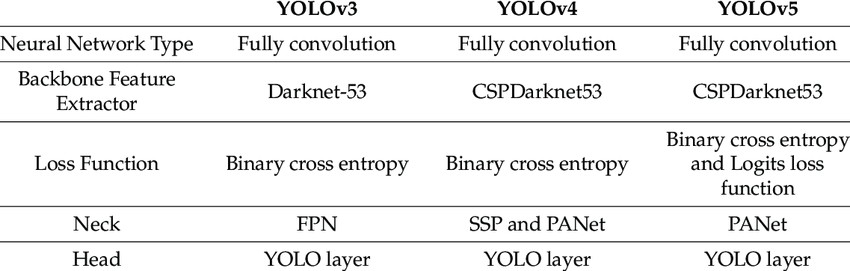
Anaconda distribution is a free and open-source platform for Python/R programming languages. It can be easily installed on any OS such as Windows, Linux, and MAC OS. It provides more than 1500 Python/R data science packages which are suitable for developing machine learning and deep learning models. Anaconda distribution provides installation of Python with various IDE's such as Jupyter Notebook, Spyder, Anaconda prompt, etc. Hence it is a very convenient packaged solution which you can easily download and install in your computer. It will automatically install Python and some basic IDEs and libraries with it.

## Visual Studio:

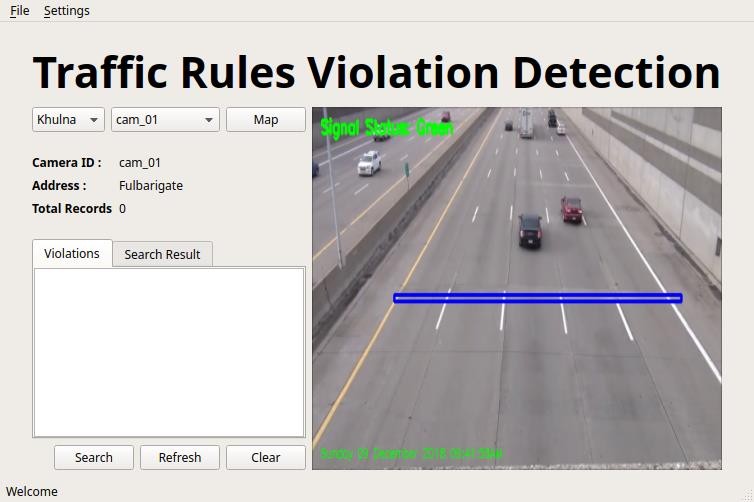
Visual Studio is a powerful developer tool that you can use to complete the entire development cycle in one place. It is a comprehensive integrated development environment (IDE) that you can use to write, edit, debug, and build code, and then deploy your app. It provides various features like Auto Code Completion, Code Inspection and Debugging, Code linting/Static Code Analysis. Visual Studio Code has a much more robust ecosystem of plugins, excellent support for remote development, and is a tool you can use repeatedly to easily write documentation and work in many languages.

# RESULT

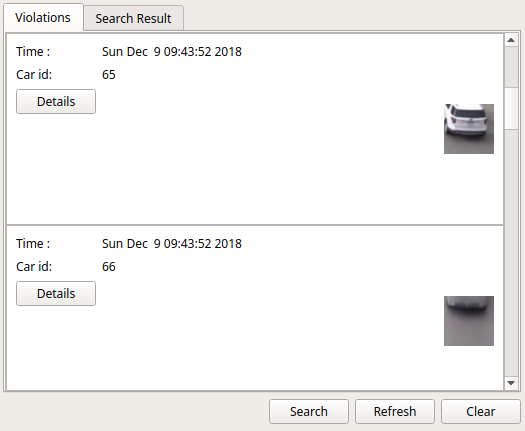
Primarily, for the start of the project usage, the administrator needs to add a camera with the menu item. In the way, the administrator can add the location of the camera, the feed file for the camera. Here the feed file is installed by the camera module over the internet. We have used Linux file sharing pattern for getting the video from the camera, where the camera will feed the given file to the server, and the server will take the feed file to process and detect violation. Also the X and Y coordinate of the camera location can be saved by the admin. This is done for future use, when we will try to use a map for locating the cameras with ease. Also the admin need to specify some rules with a JSON file for the camera. For example, the camera is used for cross road on red line violation, or is used for wrong place parking detection etc.



## Table 3. YOLOv3 Charecteristics



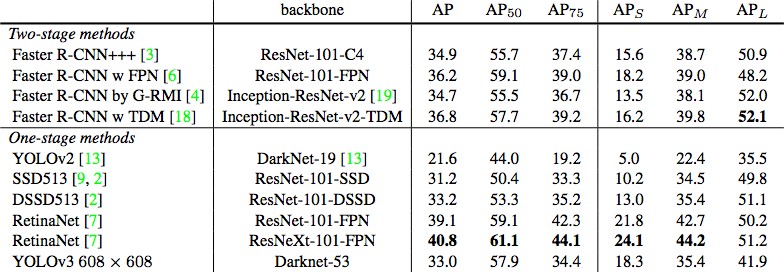
**Fig. 5. GUI**



**Fig. 6.Output**

# CHAPTER 5 CONCLUSION

Detections of traffic violation in the video surveillance is challenging as the number of vehicles on the road and traffic rules are depended on the different area of the road and timings. This paper proposes that the YOLOv3 algorithm is suitable for traffic violation detection. Results show that the detection of multiple traffic violations from a single input source is achievable. The system has an accuracy of 97.67% for vehicle count detection and an accuracy of 89.24% to detect the vehicle speed. The detection time is lower for high dense traffic flow. Thus, the system operation speed is dependent on the density of traffic.



# CHAPTER 6 CODE

**MainProcessor.py**

from Database import Database

from processor.TrafficProcessor import TrafficProcessor

from processor.violation\_detection import DirectionViolationDetection class MainProcessor:

def init (self, camera\_id): self.cam\_id = camera\_id

self.cam\_violation\_count, self.cam\_location, self.cam\_feed = Database.get\_instance().get\_cam\_details(camera\_id)

if camera\_id == 'cam\_01' or camera\_id == 'cam\_03': self.processor = TrafficProcessor() self.processor.zone1 = (100, 150)

self.processor.zone2 = (450, 145)

self.processor.thres = 30

elif camera\_id == 'cam\_02': self.processor = TrafficProcessor() self.processor.zone1 = (100, 150)

self.processor.zone2 = (450, 145)

self.processor.thres = 6 self.processor.dynamic = True

elif camera\_id == 'cam\_04':

self.processor = DirectionViolationDetection(self.cam\_feed)

def getProcessedImage(self, frame=None, cap=None): if self.cam\_id in ['cam\_01', 'cam\_02', 'cam\_03']:

dicti = self.processor.cross\_violation(frame)

elif self.cam\_id == 'cam\_04':

dicti = self.processor.feedCap(frame) return dicti

def setLight(self, color): self.processor.light = color

def getLight(self):

return self.processor.light

# TrafficProcessor.py

import datetime

import cv2 import imutils

# import winsound class TrafficProcessor:

def init (self): self.firstFrame = None self.light = "Green" self.cnt = 0 self.dynamic = False self.min\_area = 500

self.duration = 200 # millisecond self.freq = 900 # Hz

# cam specific properties self.zone1 = (100, 150)

self.zone2 = (450, 145)

self.thres = 30 self.dynamic = False

def cross\_violation(self, frame): text = ""

isCar = False

cropped\_cars = [] # list for taking all violated car's snap

# resize the frame, convert it to grayscale, and blur it self.frame = imutils.resize(frame, width=500)

self.gray = cv2.cvtColor(self.frame, cv2.COLOR\_BGR2GRAY) self.gray = cv2.GaussianBlur(self.gray, (21, 21), 0)

# if the first frame is None, initialize it if self.firstFrame is None:

self.firstFrame = self.gray

pack = {'frame': self.frame, 'reference': self.firstFrame, 'list\_of\_cars': cropped\_cars, 'cnt': self.cnt} return pack

# compute the absolute difference between the current frame and # first frame

self.frameDelta = cv2.absdiff(self.firstFrame, self.gray)

self.thresh = cv2.threshold(self.frameDelta, self.thres, 255, cv2.THRESH\_BINARY)[1] # dilate the thresholded image to fill in holes, then find contours

# on thresholded image

self.thresh = cv2.dilate(self.thresh, None, iterations=2)

cnts = cv2.findContours(self.thresh.copy(), cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

# cnts = cnts[0] if imutils.is\_cv2() else cnts[1] cnts = imutils.grab\_contours(cnts)

# loop over the contours for c in cnts:

# if the contour is too small, ignore it if cv2.contourArea(c) < self.min\_area:

continue

# compute the bounding box for the contour, draw it on the frame, # and update the text

(x, y, w, h) = cv2.boundingRect(c)

if (self.zone1[0] < (x + w / 2) < self.zone2[0] and (y + h / 2) < self.zone1[1] + 100 and ( y + h / 2) > self.zone2[1] - 100):

isCar = True

if self.light == "Red" and self.zone1[0] < (x + w / 2) < self.zone2[0] and self.zone1[1] > (y + h / 2) > \ self.zone2[1]:

# winsound.Beep(self.freq, self.duration) rcar = self.frame[y:y + h, x:x + w]

rcar = cv2.resize(rcar, (0, 0), fx=4, fy=4) cropped\_cars.append(rcar)

cv2.imwrite('reported\_car/car\_' + str(self.cnt) + ".jpg", rcar) self.cnt += 1

text = "<Violation>"

cv2.rectangle(self.frame, (x, y), (x + w, y + h), (255, 255, 0), 2) if self.dynamic or not isCar:

self.firstFrame = self.gray

# draw the text and timestamp on the frame if self.light == "Green":

color = (0, 255, 0) else:

color = (0, 0, 255)

cv2.rectangle(self.frame, self.zone1, self.zone2, (255, 0, 0), 2)

cv2.putText(self.frame, "Signal Status: {}".format(self.light), (10, 20),

cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, color, 2)

cv2.putText(self.frame, "{}".format(text), (10, 50),

cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, color, 2)

cv2.putText(self.frame, datetime.datetime.now().strftime("%A %d %B %Y %I:%M:%S%p"), (10, self.frame.shape[0] - 10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.35, color, 1)

pack = {'frame': self.frame, 'reference': self.firstFrame, 'list\_of\_cars': cropped\_cars, 'cnt': self.cnt} return pack

# Voilation\_detection.py:

import cv2 import imutils

import numpy as np import time

from processor import Vehicle

class DirectionViolationDetection: def init (self, vid\_file):

self.cnt\_up = 0

self.cnt\_down = 0

self.zone1 = (100, 200)

self.zone2 = (450, 100)

self.cap = cv2.VideoCapture(vid\_file)

self.w = self.cap.get(3) self.h = self.cap.get(4)

self.frameArea = self.h \* self.w self.areaTH = self.frameArea / 200 print('Area Threshold', self.areaTH)

self.line\_up = int(2 \* (self.h / 5)) self.line\_down = int(3 \* (self.h / 5))

self.up\_limit = int(1 \* (self.h / 5)) self.down\_limit = int(4 \* (self.h / 5))

self.line\_down\_color = (255, 0, 0)

self.line\_up\_color = (0, 0, 255) self.pt1 = [0, self.line\_down] self.pt2 = [self.w, self.line\_down]

self.pts\_L1 = np.array([self.pt1, self.pt2], np.int32) self.pts\_L1 = self.pts\_L1.reshape((-1, 1, 2)) self.pt3 = [0, self.line\_up]

self.pt4 = [self.w, self.line\_up]

self.pts\_L2 = np.array([self.pt3, self.pt4], np.int32) self.pts\_L2 = self.pts\_L2.reshape((-1, 1, 2))

self.pt5 = [0, self.up\_limit] self.pt6 = [self.w, self.up\_limit]

self.pts\_L3 = np.array([self.pt5, self.pt6], np.int32) self.pts\_L3 = self.pts\_L3.reshape((-1, 1, 2))

self.pt7 = [0, self.down\_limit] self.pt8 = [self.w, self.down\_limit]

self.pts\_L4 = np.array([self.pt7, self.pt8], np.int32) self.pts\_L4 = self.pts\_L4.reshape((-1, 1, 2))

self.fgbg = cv2.createBackgroundSubtractorMOG2()

self.kernelOp = np.ones((3, 3), np.uint8) self.kernelOp2 = np.ones((5, 5), np.uint8) self.kernelCl = np.ones((11, 11), np.uint8)

self.font = cv2.FONT\_HERSHEY\_SIMPLEX self.vehicles = []

self.max\_p\_age = 5

self.pid = 1

def feedCap(self, frame): retDict = {

'image\_threshold': None, 'image\_threshold\_2': None, 'mask\_image': None, 'mask\_image\_2': None, 'frame': None, 'list\_of\_cars': []

}

for i in self.vehicles: i.age\_one()

fgmask = self.fgbg.apply(frame) fgmask2 = self.fgbg.apply(frame)

\_, imBin = cv2.threshold(fgmask, 200, 255, cv2.THRESH\_BINARY)

\_, imBin2 = cv2.threshold(fgmask2, 200, 255, cv2.THRESH\_BINARY) # Opening (erode->dilate) to remove noise

mask = cv2.morphologyEx(imBin, cv2.MORPH\_OPEN, self.kernelOp) mask2 = cv2.morphologyEx(imBin2, cv2.MORPH\_OPEN, self.kernelOp) # Closing (dilate->erode) to join white region

mask = cv2.morphologyEx(mask, cv2.MORPH\_CLOSE, self.kernelCl) mask2 = cv2.morphologyEx(mask2, cv2.MORPH\_CLOSE, self.kernelCl) # cv2.imshow('Image Threshold', cv2.resize(fgmask, (400, 300)))

# cv2.imshow('Image Threshold2', cv2.resize(fgmask2, (400, 300)))

# cv2.imshow('Masked Image', cv2.resize(mask, (400, 300)))

# cv2.imshow('Masked Image2', cv2.resize(mask2, (400, 300)))

retDict['image\_threshold'] = cv2.resize(fgmask, (400, 300))

retDict['image\_threshold\_2'] = cv2.resize(fgmask2, (400, 300))

retDict['mask\_image'] = cv2.resize(mask, (400, 300))

retDict['mask\_image\_2'] = cv2.resize(mask2, (400, 300))

# cv2.rectangle(frame, zone1, zone2, (255, 0, 0), 2)

contours0 = cv2.findContours(mask, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_NONE) contours0 = imutils.grab\_contours(contours0)

for cnt in contours0:

# cv2.drawContours(frame, cnt, -1, (0,255,0), 3, 8) area = cv2.contourArea(cnt)

# print area," ",areaTH

if self.areaTH < area < 20000: M = cv2.moments(cnt)

cx = int(M['m10'] / M['m00'])

cy = int(M['m01'] / M['m00'])

x, y, w, h = cv2.boundingRect(cnt)

# the object is near the one which already detect before new = True

for i in self.vehicles:

if abs(x - i.getX()) <= w and abs(y - i.getY()) <= h: new = False

i.updateCoords(cx, cy) # Update the coordinates in the object and reset age if i.going\_UP(self.line\_down, self.line\_up):

self.cnt\_up += 1

print("ID:", i.getId(), 'crossed going up at', time.strftime("%c"))

# cv2.putText(frame,str(i.getId()), (x, y-2), cv2.FONT\_HERSHEY\_SIMPLEX, 5, 255) elif i.going\_DOWN(self.line\_down, self.line\_up):

roi = frame[y:y + h, x:x + w]

# cv2.imshow('Region of Interest', roi) retDict['list\_of\_cars'] = roi

print("Area equal to ::::", area) self.cnt\_down += 1

print("ID:", i.getId(), 'crossed going down at', time.strftime("%c"))

# cv2.putText(frame,str(i.getId()), (x, y-2), cv2.FONT\_HERSHEY\_SIMPLEX, 5, 255) break

if i.getState() == '1':

if i.getDir() == 'down' and i.getY() > self.down\_limit: i.setDone()

elif i.getDir() == 'up' and i.getY() < self.up\_limit: i.setDone()

if i.timedOut():

# Remove from the list person index = self.vehicles.index(i) self.vehicles.pop(index)

del i if new:

p = Vehicle.MyVehicle(self.pid, cx, cy, self.max\_p\_age) self.vehicles.append(p)

self.pid += 1

cv2.circle(frame, (cx, cy), 5, (0, 0, 255), -1)

img = cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2) # cv2.drawContours(frame, cnt, -1, (0,255,0), 3)

# cv2.imshow('Image', cv2.resize(img, (400, 300))) str\_up = 'UP: ' + str(self.cnt\_up)

str\_down = 'DOWN: ' + str(self.cnt\_down)

frame = cv2.polylines(frame, [self.pts\_L1], False, self.line\_down\_color, thickness=2) frame = cv2.polylines(frame, [self.pts\_L2], False, self.line\_up\_color, thickness=2) frame = cv2.polylines(frame, [self.pts\_L3], False, (255, 255, 255), thickness=1)

frame = cv2.polylines(frame, [self.pts\_L4], False, (255, 255, 255), thickness=1) # cv2.putText(frame, str\_up, (10,40),self.font,2,(255,255,255),2,cv2.LINE\_AA) # cv2.putText(frame, str\_up, (10,40),self.font,2,(0,0,255),1,cv2.LINE\_AA)

# cv2.putText(frame, str\_down, (10,90),self.font,2,(255,255,255),2,cv2.LINE\_AA) # cv2.putText(frame, str\_down, (10,90),self.font,2,(255,0,0),1,cv2.LINE\_AA)

# cv2.imshow('Frame', cv2.resize(frame, (400, 300))) time.sleep(0.04)

# cv2.imshow('Backgroud Subtraction', fgmask) retDict['frame'] = cv2.resize(frame, (400, 300)) return retDict

# Abort and exit with 'Q' or ESC # k = cv2.waitKey(10) & 0xff

# if k == 27:

# break

# cap.release()

# cv2.destroyAllWindows()

# MainWindow.py:

import time

import cv2 import qdarkstyle

from PyQt5 import QtCore, QtWidgets from PyQt5.QtCore import QTimer

from PyQt5.QtGui import QImage, QPixmap

from PyQt5.QtWidgets import QMainWindow, QStatusBar, QListWidget, QAction, qApp, QMenu from PyQt5.uic import loadUi

from Archive import ArchiveWindow from Database import Database

from ViolationItem import ViolationItem

from add\_windows.AddCamera import AddCamera from add\_windows.AddCar import AddCar

from add\_windows.AddRule import AddRule

from add\_windows.AddViolation import AddViolation

from processor.MainProcessor import MainProcessor

class MainWindow(QMainWindow): def init (self):

super(MainWindow, self). init () loadUi("./UI/MainWindow.ui", self)

self.live\_preview.setScaledContents(True) from PyQt5.QtWidgets import QSizePolicy

self.live\_preview.setSizePolicy(QSizePolicy.Ignored, QSizePolicy.Ignored) self.cam\_clear\_gaurd = False

self.statusBar = QStatusBar() self.setStatusBar(self.statusBar) self.statusBar.showMessage("Welcome")

self.search\_button.clicked.connect(self.search) self.clear\_button.clicked.connect(self.clear) self.refresh\_button.clicked.connect(self.refresh)

self.database = Database.get\_instance() self.database.delete\_all\_cars() self.database.delete\_all\_violations()

cam\_groups = self.database.get\_cam\_group\_list() self.camera\_group.clear() self.camera\_group.addItems(name for name in cam\_groups) self.camera\_group.setCurrentIndex(0)

self.camera\_group.currentIndexChanged.connect(self.camGroupChanged)

cams = self.database.get\_cam\_list(self.camera\_group.currentText()) self.cam\_selector.clear()

self.cam\_selector.addItems(name for name, location, feed in cams) self.cam\_selector.setCurrentIndex(0) self.cam\_selector.currentIndexChanged.connect(self.camChanged)

self.processor = MainProcessor(self.cam\_selector.currentText()) self.log\_tabwidget.clear()

self.violation\_list = QListWidget(self) self.search\_result = QListWidget(self)

self.log\_tabwidget.addTab(self.violation\_list, "Violations") self.log\_tabwidget.addTab(self.search\_result, "Search Result")

self.feed = None self.vs = None self.updateCamInfo()

self.updateLog()

self.initMenu()

self.timer = QTimer(self) self.timer.timeout.connect(self.update\_image) self.timer.start(50)

# trafficLightTimer = QTimer(self)

# trafficLightTimer.timeout.connect(self.toggleLight) # trafficLightTimer.start(5000)

def toggleLight(self):

self.processor.setLight('Green' if self.processor.getLight() == 'Red' else 'Red')

def initMenu(self):

menubar = self.menuBar()

fileMenu = menubar.addMenu('&File') # File menu

## add record manually

addRec = QMenu("Add Record", self)

act = QAction('Add Car', self) act.setStatusTip('Add Car Manually') act.triggered.connect(self.addCar) addRec.addAction(act)

act = QAction('Add Rule', self) act.setStatusTip('Add Rule Manually') act.triggered.connect(self.addRule) addRec.addAction(act)

act = QAction('Add Violation', self) act.setStatusTip('Add Violation Manually') act.triggered.connect(self.addViolation) addRec.addAction(act)

act = QAction('Add Camera', self) act.setStatusTip('Add Camera Manually') act.triggered.connect(self.addCamera) addRec.addAction(act)

fileMenu.addMenu(addRec)

# check archive record ( Create window and add button to restore them) act = QAction('&Archives', self)

act.setStatusTip('Show Archived Records') act.triggered.connect(self.showArch) fileMenu.addAction(act)

settingsMenu = menubar.addMenu('&Settings') themeMenu = QMenu("Themes", self) settingsMenu.addMenu(themeMenu)

act = QAction('Dark', self) act.setStatusTip('Dark Theme')

act.triggered.connect(lambda: qApp.setStyleSheet(qdarkstyle.load\_stylesheet\_pyqt5())) themeMenu.addAction(act)

act = QAction('White', self) act.setStatusTip('White Theme')

act.triggered.connect(lambda: qApp.setStyleSheet(qdarkstyle.load\_stylesheet\_pyqt5())) themeMenu.addAction(act)

## Add Exit fileMenu.addSeparator() act = QAction('&Exit', self) act.setShortcut('Ctrl+Q')

act.setStatusTip('Exit application') act.triggered.connect(qApp.quit) fileMenu.addAction(act)

def keyReleaseEvent(self, event):

if event.key() == QtCore.Qt.Key\_G: self.processor.setLight("Green")

elif event.key() == QtCore.Qt.Key\_R: self.processor.setLight("Red")

elif event.key() == QtCore.Qt.Key\_S: self.toggleLight()

def addCamera(self):

addWin = AddCamera(parent=self) addWin.show()

def addCar(self):

addWin = AddCar(parent=self) addWin.show()

def addViolation(self): pass

addWin = AddViolation(parent=self) addWin.show()

def addRule(self):

addWin = AddRule(parent=self) addWin.show()

def showArch(self):

addWin = ArchiveWindow(parent=self) addWin.show()

def updateSearch(self): pass

def update\_image(self):

\_, frame = self.vs.read()

packet = self.processor.getProcessedImage(frame)

cars\_violated = packet['list\_of\_cars'] # list of cropped images of violated cars if len(cars\_violated) > 0:

for c in cars\_violated:

carId = self.database.get\_max\_car\_id() + 1 car\_img = 'car\_' + str(carId) + '.png' cv2.imwrite('car\_images/' + car\_img, c)

self.database.insert\_into\_cars(car\_id=carId, car\_img=car\_img)

self.database.insert\_into\_violations(camera=self.cam\_selector.currentText(), car=carId, rule='1', time=time.time())

self.updateLog()

qimg = self.toQImage(packet['frame']) self.live\_preview.setPixmap(QPixmap.fromImage(qimg))

def updateCamInfo(self):

count, location, self.feed = self.database.get\_cam\_details(self.cam\_selector.currentText()) self.feed = 'videos/' + self.feed

self.processor = MainProcessor(self.cam\_selector.currentText()) self.vs = cv2.VideoCapture(self.feed) self.cam\_id.setText(self.cam\_selector.currentText()) self.address.setText(location) self.total\_records.setText(str(count))

def updateLog(self): self.violation\_list.clear()

rows = self.database.get\_violations\_from\_cam(str(self.cam\_selector.currentText())) for row in rows:

listWidget = ViolationItem() listWidget.setData(row)

listWidgetItem = QtWidgets.QListWidgetItem(self.violation\_list) listWidgetItem.setSizeHint(listWidget.sizeHint()) self.violation\_list.addItem(listWidgetItem) self.violation\_list.setItemWidget(listWidgetItem, listWidget)

@QtCore.pyqtSlot() def refresh(self):

self.updateCamInfo() self.updateLog()

@QtCore.pyqtSlot() def search(self):

from SearchWindow import SearchWindow

searchWindow = SearchWindow(self.search\_result, parent=self) searchWindow.show()

@QtCore.pyqtSlot() def clear(self):

qm = QtWidgets.QMessageBox

prompt = qm.question(self, '', "Are you sure to reset all the values?", qm.Yes | qm.No) if prompt == qm.Yes:

self.database.clear\_cam\_log() self.updateLog()

else:

pass

def toQImage(self, raw\_img): from numpy import copy img = copy(raw\_img)

qformat = QImage.Format\_Indexed8 if len(img.shape) == 3:

if img.shape[2] == 4:

qformat = QImage.Format\_RGBA8888 else:

qformat = QImage.Format\_RGB888

outImg = QImage(img.tobytes(), img.shape[1], img.shape[0], img.strides[0], qformat) outImg = outImg.rgbSwapped()

return outImg

@QtCore.pyqtSlot() def camChanged(self):

if not self.cam\_clear\_gaurd: self.updateCamInfo() self.updateLog()

@QtCore.pyqtSlot()

def camGroupChanged(self):

cams = self.database.get\_cam\_list(self.camera\_group.currentText()) self.cam\_clear\_gaurd = True

self.cam\_selector.clear()

self.cam\_selector.addItems(name for name, location, feed in cams) self.cam\_selector.setCurrentIndex(0)

# self.cam\_selector.currentIndexChanged.connect(self.camChanged) self.cam\_clear\_gaurd = False

self.updateCamInfo()

# Main.py:

import sys

# import qdarkstyle

from PyQt5.QtWidgets import QApplication from MainWindow import MainWindow

def main():

app = QApplication(sys.argv) main\_window = MainWindow()

# app.setStyleSheet(qdarkstyle.load\_stylesheet\_pyqt5()) main\_window.show()

sys.exit(app.exec\_())

if name == ' main ': main()

# CHAPTER 7

**BIBLIOGRAPHY**

1. S. Wahyudi et al., "Automatic Traffic Violation Detection Based on Image Processing Techniques," 2016 International Conference on Frontiers of Information Technology (FIT), Islamabad, Pakistan, 2016, pp. 165-170. doi: 10.1109/FIT.2016.036
2. S. M. Alkobaisi and N. Bouguechal, "Traffic Violation Detection System using Computer Vision Techniques," 2018 International Conference on Intelligent Systems (IS), Fes, Morocco, 2018, pp. 1-6. doi: 10.1109/IS.2018.8710614
3. M. A. Hossain and R. Begum, "Traffic Violation Detection and Recognition System: A Survey," 2018 International Conference on Innovations in Science, Engineering and Technology (ICISET),

Dhaka, Bangladesh, 2018, pp. 1-6. doi: 10.1109/ICISET.2018.8697755

1. H. G. Ali et al., "Automatic Traffic Violation Detection System Using Image Processing Techniques," 2019 4th International Conference on Intelligent Transportation Engineering (ICITE),

Cairo, Egypt, 2019, pp. 1-6. doi: 10.1109/ICITE.2019.8720277

1. A. S. Zulkifli et al., "Deep Learning Approach for Traffic Violation Detection," 2018 3rd International Conference on Computer Applications & Information Security (ICCAIS), Riyadh,

Saudi Arabia, 2018, pp. 1-6. doi: 10.1109/CAIS.2018.8559869

1. P. Vasantharaja, S. Jayapal, and S. P. Shantharajah, "Traffic Violation Detection using Machine Learning Techniques," 2017 IEEE International Conference on Intelligent Techniques in Control, Optimization and Signal Processing (INCOS), Tamil Nadu, India, 2017, pp. 187-192. doi: 10.1109/INCOS.2017.8525941
2. J. Gómez and D. G. C. Chávez, "Traffic Violation Detection System based on Deep Learning Techniques," 2019 IEEE International Autumn Meeting on Power, Electronics and Computing (ROPEC), Ixtapa, Mexico, 2019, pp. 1-6. doi: 10.1109/ROPEC48539.2019.8954433
3. L. A. Zadeh, "Traffic violation detection using fuzzy logic," 2016 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), Vancouver, BC, Canada, 2016, pp. 1417-1422. doi: 10.1109/FUZZ-IEEE.2016.7737870
4. C. L. Chen et al., "Real-time traffic violation detection using deep learning," 2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC), Budapest, Hungary, 2016, pp. 000391-000396. doi: 10.1109/SMC.2016.7844454
5. A. Ahmed, M. Islam, and S. K. Das, "Traffic violation detection using deep learning and computer vision techniques," 2017 6th International Conference on Informatics, Electronics & Vision (ICIEV), Dhaka, Bangladesh, 2017, pp. 1-6. doi: 10.1109/ICIEV.2017.8338490