TRAFFIC RULES VIOLATION DETECTION SYSTEM USING COMPUTER VISION

A MINI PROJECT REPORT

Submitted by

THARUN KUMAR.S
UDAY KIRAN .V
YUVA KISHORE.K.S

BACHELOR OF ENGINEERING
IN
COMPUTER SCIENCE AND ENGINEERING

VEL TECH HIGH TECH
Dr. RANGARAJAN Dr. SAKUNTHALA ENGINEERING COLLEGE

An Autonomous Institution

Approved by AICTE-New Delhi, Affiliated to Anna University, Chennai
Accredited by NBA, New Delhi & Accredited by NAAC with "A" Grade & CGPA of 3.27

JANUARY 2023

VEL TECH HIGH TECH

Dr. RANGARAJAN Dr. SAKUNTHALA ENGINEERING COLLEGE

An Autonomous Institution

Approved by AICTE-New Delhi, Affiliated to Anna University, Chennai
Accredited by NBA, New Delhi & Accredited by NAAC with "A" Grade & CGPA of 3.27

BONAFIDE CERTIFICATE

Certified that this mini project entitled **"TRAFFIC RULES VIOLATION DETECTION SYSTEM USING COMPUTER VISION"** is the bonafide work of **"THARUN KUMAR.S (113020104100), UDAY KIRAN.V (113020104101), YUVA KISHORE.K.S (1103020104112)"** who carried out the work under my supervision.

SIGNATURE

Dr. DURGA DEVI, B.E., M.E., PhD.,
HEAD OF THE DEPARTMENT
Computer Science and Engineering

SIGNATURE

Mrs.N.SHARMILA BANU, B.TECH., M.E
ASSISTANT PROFESSOR
Computer Science and Engineering

SIGNATURE

Dr. V. R. RAVI

DEAN - SoEC

ACKNOWLEDGEMENT

We wish to express our sincere thanks to the people who extended their help during the course of our Mini Projectwork.

First of all, we would like to express our deep gratitude to our beloved and respected **FOUNDER & CHAIRMAN Col Prof. Vel. Shri. Dr. R. Rangarajan,** and **FOUNDRESS & VICE CHAIRMAN Dr. Mrs. SAKUNTHALA RANGARAJAN.**We also record our sincere thanks to our Honourable Principal, **Dr. E**. **KAMALANABAN, B.E., M.E., Ph.D.,** for his kind support for this Mini Project.

We are thankful and extremely grateful to our Dean-SoEC **Dr. V. R. RAVI** and Head of the Department **DR. S. DURGA DEVI, B.E., M.E., PhD.,** for their support and motivation feed to us for doing the Mini Project.

We would like to extend our sincere thanks to our guide **Dr. S. DURGA DEVI, B.E., M.E., PhD,** for her technical support during this project. Her stupendous encouragement enabled us to complete our Mini Project successfully.

We would like to extend our sincere thanks to our Project Coordinator **DR.M.MALATHY B.E (CSE)., M.E(CSE)., Ph.D. (ICE)** and **Mrs. P.K. SHEELA SHANTHA KUMARI B.E., M.E(CSE)** for their continuous support over this Mini Project.

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

The number of new vehicles on the road is increasing rapidly, which in turn causes highly congested roads and serving as a reason to break traffic rules by violating them. This leads to a high number of road accidents. Traffic violation detection systems using computer vision are a very efficient tool to reduce traffic violations by tracking and Penalizing. The proposed system was implemented using YOLOV3 object detection for traffic violation detections such as signal jump, vehicle speed, and the number of vehicles. Further, the system is optimized in terms of accuracy. Using the Region of interest and location of the vehicle in the duration of frames, determining signal jump. This implementation obtained an accuracy of 97.67% for vehicle count detection and an accuracy of 89.24% for speed violation detection.

CHAPTER-1.1

GENERAL INTRODUCTION

The increasing number of cars in cities can cause high volume of traffic, and implies that traffic violations become more critical nowadays in Bangladesh and also around the world. This causes severe destruction of property and more accidents that may endanger the lives of the people. To solve the alarming problem and prevent such unfathomable consequences, traffic violation detection systems are needed. For which the system enforces proper traffic regulations at all times, and apprehend those who does not comply. A traffic violation detection system must be realized in real-time as the authorities track the roads all the time. Hence, traffic enforcers will not only be at ease in implementing safe roads accurately, but also efficiently; as the traffic detection system detects violations faster than humans. This system can detect most common three types of traffic violation in real-time which are signal violation, parking violation and wrong direction violation. A user friendly graphical interface is associated with the system to make it simple for the user to operate the system, monitor traffic and take action against the violations of traffic rules.

CHAPTER-2

LITERATURE SURVEY

"Traffic Rules Violation Detection using Deep Learning," Aniruddha Tonge, S. Chandak, R. Khiste, U. Khan and L. A. Bewoor, Published on 2020

In the suggested technique, the system detects motorcycle using YOLO-based object detection, and then checks each motorcycle for particular violations, such as not wearing a helmet or crosswalk. A CNN (Convolutional neural network) based classifier is used to detect helmet violations. In order to ensure safety measures on roads of India, the identification of traffic rule violators is highly desirable but challenging job due to numerous difficulties such as occlusion, illumination, etc. In this paper we propose an end to end framework for detection of violations, notifying violators, and also storing them for analyzing and generating statistics for better decision making regarding traffic rules policy. In the proposed approach, we first detect vehicles using object detection which is performed using YOLO, and then accordingly each vehicle is checked against appropriate violations viz. not wearing a helmet, violation of crosswalks. Helmet violation is detected using a CNN (Convolutional neural network) based classifier. Crosswalk violation is detected using Instance Segmentation by Mask R-CNN architecture. After violations are detected, vehicle numbers are obtained of respective violators using OCR, and violators are notified. Thus an end to end autonomous system will help enforcing strong regulation of traffic rules.

“Traffic Signal Violation Detection using Artificial Intelligence and Deep Learning”, Ruben.J Franklin and Mohana, Published on 2020

The number of new vehicles on the road is increasing rapidly, which in turn causes highly congested roads and serving as a reason to break traffic rules by violating them. This leads to a high number of road accidents. Traffic violation detection systems using computer vision are a very efficient tool to reduce traffic violations by tracking and Penalizing. The proposed system was implemented using YOLOV3 object detection for traffic violation detections such as signal jump, vehicle speed, and the number of vehicles. Further, the system is optimized in terms of accuracy. Using the Region of interest and location of the vehicle in the duration of frames, determining signal jump. This implementation obtained an accuracy of 97.67% for vehicle count

detection and an accuracy of 89.24% for speed violation detection.

"Applications for traffic surveillance use object detection algorithms like convolution neural networks (CNN)". Chetan Kumar.B Published on [2020]

The single object detection has been performed by using the concepts of convolution layers. A neural network consists of several different layers such as the input layer, at least one hidden layer, and an output layer. The dataset used for single object detection is the on-road vehicle dataset. This dataset consists of three classes of images which are Heavy, Auto and Light. The dataset consists of images of varying illuminations. The performance metrics has been calculated for the day dataset, evening dataset and night dataset. Multiple object detection has been performed using the You Only Look Once (YOLOv3) algorithm. This approach encompasses a single deep convolution neural network dividing the input into a cell grid and each cell predicts a boundary box and classifies object directly.

"Performance Evaluation of Background Modeling Methods for Object Detection and Tracking," Mohana, HV Ravish Aradhya Published on 2020

Background modeling is a method of detecting moving objects from the difference between current frame and a reference frame. Benefits of background modeling and object detection are uncountable, and have enormous applications in the area of artificial intelligence, video surveillance, medicine and some of security related applications. Popular background modeling methods are Gaussian Mixture Model (GMM), Adaptive Gaussian Mixture Model (AGMM), Kernel Density Estimator (KDE), Codebook, Visual Background Extractor(VIBE), Self-Organizing Background Subtraction (SOBS), all these models suffers from illumination changes and dynamic background. In this paper, GMM and SOBS algorithms are implemented using MATLAB software for object detection and tracking, and evaluated results using performance parameters such as recall, specificity, False positive Rate (FPR), False Negative Rate (FNR), precision, Percentage of Wrong Classification (PWC), F-measure. Obtained results shows that 9.71% improvement in specificity of GMM and 2.97% improvement in specificity of SOBS.

“Background Modeling techniques for foreground detection and Tracking using Gaussian Mixture model” R.K. Meghana, Yojan Chitkara, Apoorva S., Mohana Published on 2019

Background Modeling and Foreground detection in sports has been achieved by cleverly developing a model of a background from a video by deducing knowledge from frames and comparing this model to every subsequent frame and subtracting the background region from it, hence leaving the foreground detected. This output from GMM background subtraction is fed into the feature extraction algorithm, which segregates the players based on teams. By extracting information of primary colors from each frame, the design of the algorithm based on the color of preference is done. Tracking algorithms Kalman and extended Kalman Filters help to predict and correct the location of players and in correctly estimating their trajectory on the field. Challenges such as shadowing, occlusions and illumination changes are addressed. The designed algorithms are tested against a set of performance parameters for the following datasets (Norway and FIFA) using MATLAB (2017b) and the inferences are respectively made. Object detection, motion detection and Kalman filter algorithms are implemented and the observed results are 100%, 84% and 100% accuracy respectively. With the results quantification and performance analysis, it is observed that with the decrease in contrast between player jerseys a decrease in detection accuracy occurs and with players crowded regions on the field and occluded players a decrease in tracking accuracy was observed.

CHAPTER-3

HARDWARE AND SOFTWARE

3.1 HARDWARE:

Motor Driver:

L293D is a dual H-bridge motor driver integrated circuit (IC). Motor drivers act as current amplifiers since they take a low current control signal and provide a higher-current signal. This higher current signal is used to drive the motors. L293D contains two inbuilt H-bridge driver circuits. In its common mode of operation, two DC motors can be driven simultaneously, both in forward and reverse direction. The motor operations of two motors can be controlled by input logic at pins 2 & 7 and 10 & 15. Input logic 00 or 11 will stop the corresponding motor. Logic 01 and 10 will rotate it in clockwise and anticlockwise directions, respectively.

LCD:

LCD (Liquid Crystal Display) screen is an electronic display module and find a wide range of applications. A 16x2 LCD display is very basic module and is very commonly used in various devices and circuits. These modules are preferred over seven segments and other multi segment LEDs. The reasons being: LCDs are economical; easily programmable; have no limitation of displaying special & even custom characters (unlike in seven segments), animations and so on.

Level Shifter:

The MAX232 IC is used to convert the TTL/CMOS logic levels to RS232 logic levels during serial communication of microcontrollers with PC. The controller operates at TTL logic level (0-5V) whereas the serial communication in PC works on RS232 standards (-25 V to + 25V). This makes it difficult to establish a direct link between them to communicate with each other. The intermediate link is provided through MAX232. It is a

dual driver/receiver that includes a capacitive voltage generator to supply RS232 voltage levels from a single 5V supply. Each receiver converts RS232 inputs to 5V TTL/CMOS levels. These receivers (R1 & R2) can accept ±30V inputs. The drivers (T1 & T2), also called transmitters, convert the TTL/CMOS input level into RS232 level

Buzzer:

A buzzer or beeper is an audio signaling device, which may be mechanical, electromechanical, or piezoelectric. Typical uses of buzzers and beepers include alarm devices, timers and confirmation of user input such as a mouse click or keystroke. IR Sensor: IR Sensors work by using a specific light sensor to detect a select light wavelength in the Infra-Red (IR) spectrum. By using an LED which produces light at the same wavelength as what the sensor is looking for, you can look at the intensity of the received light. When an object is close to the sensor, the light from the LED bounces off the object and into the light sensor. This results in a large jump in the intensity, which we already know can be detected using a threshold.

3.2 SOFTWARE

Open CV is a field of AI that helps systems and computers to obtain meaningful and useful information from images, visual inputs and videos. Open CV is an open-source computer vision and Python library which is generally used for the purpose of image processing. Tensor Flow is used for classification of the vehicles with the help of darknet-53.

CHAPTER-4

EXISTING SYSTEM

Traffic Rules Violation Detection System The designed set of rules was correctly capable of stumbling on the type of violation particular on this challenge that's denying traffic signals, parking in no parking zone, and wrong direction driving. The convergence of detection for the three kinds of traffic violations mentioned is dissimilar since they each have a different threshold condition. The system provides detection for all three violations but detects signal violation and parking violation better than direction violation. Further, the system can process one data at a time. Additionally, this system runtime is somewhat sluggish and can be improved through using a pc with high-speed processor specs or GPU Future research about the software of the designed set of rules for different superior image processing strategies. Because this will improve the runtime of the system by neglecting different useless steps done in a background distinction approach. A computer vision algorithm may be achieved rather to offer extra intelligence within the machine. We plan to implement the number plate detection with OCR support to make this system more robust.

Automatic Traffic Rule Violation Detection and Number Plate Recognition The proposed model includes an automated system that uses IR sensors and a camera-based on Adriano to capture video. The project presents Automatic Number Plate Recognition (ANPR) techniques and other image manipulation techniques for plate localization and character recognition which makes it faster and easier to identify the number plates. After recognizing the vehicle number from the number plate the SMS based module is used to notify the vehicle owners about their traffic rule violation. An additional SMS is sent to the Regional Transport Office (RTO) for tracking the report status.” This project was designed to detect the traffic signal violation and also to report the victim. It was able to detect “Signal violation detection”, “Reporting the victim”. The system could not process more than one data at a time and also the runtime was slow. Project was sensitive to vibration and fast-changing targets due to the long shutter time

CHAPTER-5

PROPOSED SYSTEM

To design and develop traffic rules violation detection system using Machine Learning. 4. WORKING MODEL A PC is used in the recognition system to capture the car registration number plate. Under poor environmental conditions, as shown in the following point, car licence plate images are illegible when taken by the system:

1. Overexposure, reflection, or shadows result in poor lighting and low contrast.
2. Unfavorable weather conditions, such as rain or smog.
3. Images that is hazy.
4. Lowering the image's illumination. The system will recognize the vehicle's license plate and convert the photos to gray scale images.

The gray scale photos are then converted to binary images, which only include the numbers '0' and '1'. Following the binary graphics, the system will segment the automobile licenseplate's personality. The character and number will be segmented for each separate figure. After that, all of the characters and numbers will be converted to binary form in terms of the matrix and recognized by the neural network. After that, image cropping and recognition come next.

1. Take a picture with your webcam.
2. . Change the image's scale to a smaller size.
3. Determine the location of the number plate.
4. Segmentation.
5. Identification by number.
6. Save the file in the specified format.

1) Take a picture with your webcam: After taking a picture with your webcam. Save the captured image to a picture document for farther processing.
2) Convert the picture to binary format: Determine the opacity of the image. Calculate the images correct threshold value. Using the computed threshold, convert the image to a binary picture., Recognition in context, super pixel stuff segmentation, 330K images (>200K labeled), 1.5 million object instances, 80 object categories, 91 stuff categories, 5 captions per image, 250,000 people with key points.

CHAPTER-6

OVERALL ARCHITECTURE

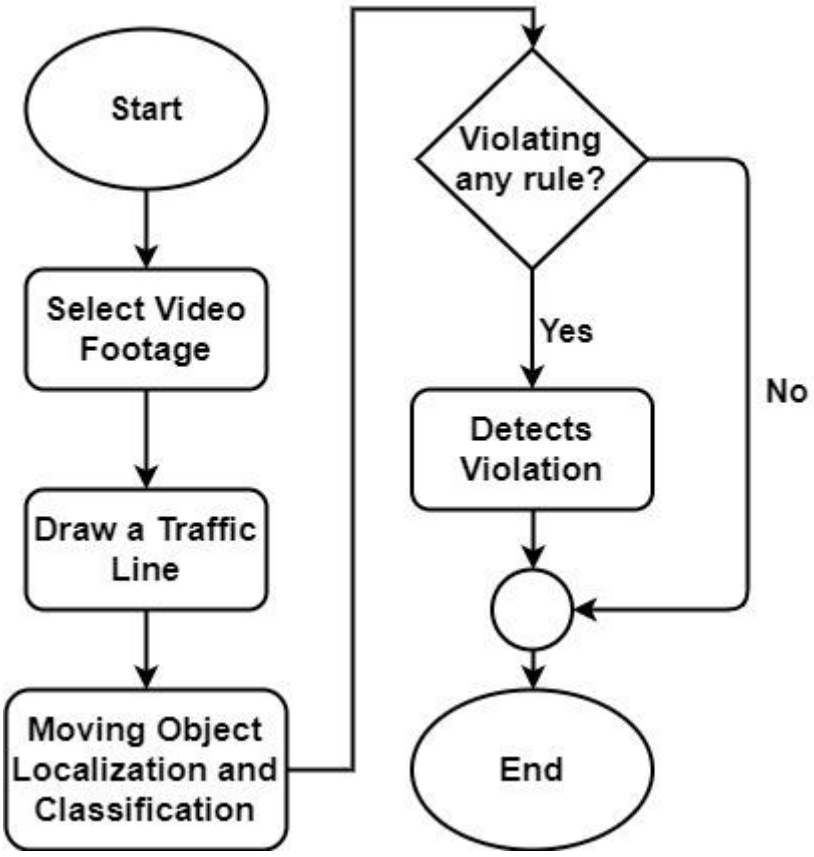

Fig. 6.1 Flow chart of violation detection

CHAPTER-7

MODULES

MODULE 1- Vehicle Classification

- **Class Prediction**
- **Predictions across scales**
- **Feature Extractor**

MODULE 2- Violation Detection

CHAPTER-7.1

MODULE EXPLANATION

MODULE 1 - Vehicle Classification:

` A video footage is given as input to the system and the moving objects are detected.A detection model YOLO version3 is used to categorize the moving vehicles into particular classes. YOLOv3 is the third version algorithm in YOLO family which is used for detection of objects. It maintains the integrity of data with the use of several techniques and it is able to identify the objects more accurately. Darknet-53 architecture is used to build the classifier model.

Features of vehicle classification are

- Class Prediction
- Predictions across scales
- Feature Extractor

It is a single network for real-time object detection and categorization needs to be evaluated individually but from the same convolutional region. This algorithm anticipates the fair score by means of logistic regression. Here 1 indicates the overall overlapping of bounding box on the object. It predicts only 1 bounding box prior for one ground truth object and any misconception in this process lead to both categorization and identification

Class Prediction:

This algorithm makes use of unconventional logistic classifiers for every class in place of a conventional Soft max layer. This process is performed to convert into a multi-label classification. Every box identifies the classes that it may contain with the help of the multi label classification.

Predictions across scales:

For the identification of different scales, it detects boxes at three different scales. Then features can be extracted from each scale by using a method which is similar to that of feature pyramid networks. YOLOv3 gains the ability to predict at different scales using the

above method. The bounding box priors which are generated using dimension clusters can be divided into 3 scales, so that there are three bounding box priors per scale. Thus there will be 9 bounding box priors as a total.

Feature Extractor:

It uses a new network called Darknet-53 which consists of 53 convolutional layers, and it has more features when compared to YOLO version2. It is more powerful than Darknet -19. And its efficiency is higher than ResNet-101/ResNet-152. 2. Violation Detection The vehicles are detected using the model YOLOv3. After detection of vehicles, violation cases are checked. A traffic line is drawn by the user over the road in the preview of the given video footage. This line indicates that the traffic signal is red in that area. The objects are detected with a bounding box of green color around them. If any of the moving vehicles go ahead of the traffic line indicating the red signal, then the traffic violation happens. The color of the bounding box around the vehicle becomes red after the violation detection.

MODULE 2 - Violation Detection:

The vehicles are detected using the model YOLOv3. After detection of vehicles, violation cases are checked. A traffic line is drawn by the user over the road in the preview of the given video footage. This line indicates that the traffic signal is red in that area. The objects are detected with a bounding box of green color around them. If any of the moving vehicles go ahead of the traffic line indicating the red signal, then the traffic violation happens. The color of the bounding box around the vehicle becomes red after the violation detection.

.

CHAPTER-8

ALGORITHM

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. The YOLO machine learning algorithm uses features learned by a deep convolutional neural network to detect an object. Versions 1-3 of YOLO were created by Joseph Redmon and Ali Farhadi, and the third version of the YOLO machine learning algorithm is a more accurate version of the original ML algorithm. 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. YOLOv3 is an improved version of YOLO and YOLOv2. YOLO is implemented using the Keras or Open CV deep learning libraries.

CHAPTER-9

IMPLEMENTATION AND OUTPUT

```
import
numpy
as np
        from keras.layers import Conv2D, Input, BatchNormalization, LeakyReLU,
        ZeroPadding2D, UpSampling2D
        from keras.layers.merge import add, concatenate
        from keras.models import Model
        import struct
        import cv2
        class WeightReader:
          def __init__(self, weight_file):
            with open(weight_file, 'rb') as w_f:
              major,    = struct.unpack('i', w_f.read(4))
              minor,    = struct.unpack('i', w_f.read(4))
              revision, = struct.unpack('i', w_f.read(4))
              if (major*10 + minor) >= 2 and major < 1000 and minor < 1000:
                w_f.read(8)
              else:
                w_f.read(4)
              transpose = (major > 1000) or (minor > 1000)

              binary = w_f.read()
            self.offset = 0
            self.all_weights = np.frombuffer(binary, dtype='float32')

          def read_bytes(self, size):
            self.offset = self.offset + size
            return self.all_weights[self.offset-size:self.offset]
          def load_weights(self, model):
            for i in range(106):
```

```
            try:
                conv_layer = model.get_layer('conv_' + str(i))
                print("loading weights of convolution #" + str(i))
                if i not in [81, 93, 105]:
                    norm_layer = model.get_layer('bnorm_' + str(i))
                    size = np.prod(norm_layer.get_weights()[0].shape)
                    beta  = self.read_bytes(size) # bias
                    gamma = self.read_bytes(size) # scale
                    mean  = self.read_bytes(size) # mean
                    var   = self.read_bytes(size) # variance
                    weights = norm_layer.set_weights([gamma, beta, mean, var])
                if len(conv_layer.get_weights()) > 1:
                    bias   = self.read_bytes(np.prod(conv_layer.get_weights()[1].shape))
                    kernel = self.read_bytes(np.prod(conv_layer.get_weights()[0].shape))

                    kernel = kernel.reshape(list(reversed(conv_layer.get_weights()[0].shape)))
                    kernel = kernel.transpose([2,3,1,0])
                    conv_layer.set_weights([kernel, bias])
                else:
                    kernel = self.read_bytes(np.prod(conv_layer.get_weights()[0].shape))
    return add([skip_connection, x]) if skip else x
def _interval_overlap(interval_a, interval_b):
    x1, x2 = interval_a
    x3, x4 = interval_b
    if x3 < x1:
        if x4 < x1:
            return 0
        else:
            return min(x2,x4) - x1
    else:
        if x2 < x3:
             return 0
        else:
            return min(x2,x4) - x3
```

```
def _sigmoid(x):
    return 1. / (1. + np.exp(-x))
def bbox_iou(box1, box2):
    intersect_w = _interval_overlap([box1.xmin, box1.xmax], [box2.xmin, box2.xmax])
    intersect_h = _interval_overlap([box1.ymin, box1.ymax], [box2.ymin, box2.ymax])

    intersect = intersect_w * intersect_h
    w1, h1 = box1.xmax-box1.xmin, box1.ymax-box1.ymin
    w2, h2 = box2.xmax-box2.xmin, box2.ymax-box2.ymin

    union = w1*h1 + w2*h2 - intersect

    return float(intersect) / union
def make_yolov3_model():
    input_image = Input(shape=(None, None, 3))
    # Layer  0 => 4
    x = _conv_block(input_image, [{'filter': 32, 'kernel': 3, 'stride': 1, 'bnorm': True,
'leaky': True, 'layer_idx': 0},
                                  {'filter': 64, 'kernel': 3, 'stride': 2, 'bnorm': True, 'leaky': True,
'layer_idx': 1},
                                  {'filter': 32, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True,
'layer_idx': 2},
                                  {'filter': 64, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True,
'layer_idx': 3}])
    # Layer  5 => 8
    x = _conv_block(x, [{'filter': 128, 'kernel': 3, 'stride': 2, 'bnorm': True, 'leaky': True,
'layer_idx': 5},
                        {'filter':  64, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True,
'layer_idx': 6},
                        {'filter': 128, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True,
'layer_idx': 7}])
    # Layer  9 => 11
    x = _conv_block(x, [{'filter':  64, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True,
'layer_idx': 9},
```

```
                {'filter': 128, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True,
'layer_idx': 10}])
   # Layer 12 => 15
   x = _conv_block(x, [{'filter': 256, 'kernel': 3, 'stride': 2, 'bnorm': True, 'leaky': True,
'layer_idx': 12},
                {'filter': 128, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True,
'layer_idx': 13},
                {'filter': 256, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True,
'layer_idx': 14}])
   # Layer 16 => 36
   for i in range(7):
      x = _conv_block(x, [{'filter': 128, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky':
True, 'layer_idx': 16+i*3},
                   {'filter': 256, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True,
'layer_idx': 17+i*3}])

   skip_36 = x

   # Layer 37 => 40
   x = _conv_block(x, [{'filter': 512, 'kernel': 3, 'stride': 2, 'bnorm': True, 'leaky': True,
'layer_idx': 37},
                {'filter': 256, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True,
'layer_idx': 38},
                {'filter': 512, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True,
'layer_idx': 39}])
   # Layer 41 => 61
   for i in range(7):
      x = _conv_block(x, [{'filter': 256, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky':
True, 'layer_idx': 41+i*3},
                   {'filter': 512, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True,
'layer_idx': 42+i*3}])

   skip_61 = x
```

```
    # Layer 62 => 65
    x = _conv_block(x, [{'filter': 1024, 'kernel': 3, 'stride': 2, 'bnorm': True, 'leaky': True,
'layer_idx': 62},
                        {'filter':  512, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True,
'layer_idx': 63},
                        {'filter': 1024, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True,
'layer_idx': 64}])
    # Layer 66 => 74
    for i in range(3):
        x = _conv_block(x, [{'filter':  512, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky':
True, 'layer_idx': 66+i*3},
                            {'filter': 1024, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True,
'layer_idx': 67+i*3}])

    # Layer 75 => 79
    x = _conv_block(x, [{'filter':  512, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True,
'layer_idx': 75},
                        {'filter': 1024, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True,
'layer_idx': 76},
                        {'filter':  512, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True,
'layer_idx': 77},
                        {'filter': 1024, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True,
'layer_idx': 78},
                        {'filter':  512, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True,
'layer_idx': 79}], skip=False)
    # Layer 80 => 82
    yolo_82 = _conv_block(x, [{'filter': 1024, 'kernel': 3, 'stride': 1, 'bnorm': True,
'leaky': True,  'layer_idx': 80},
                              {'filter':  255, 'kernel': 1, 'stride': 1, 'bnorm': False, 'leaky': False,
'layer_idx': 81}], skip=False)
    # Layer 83 => 86
    x = _conv_block(x, [{'filter': 256, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True,
'layer_idx': 84}], skip=False)
    x = UpSampling2D(2)(x)
```

```
    x = concatenate([x, skip_61])
    # Layer 87 => 91
    x = _conv_block(x, [{'filter': 256, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True,
'layer_idx': 87},
                        {'filter': 512, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True,
'layer_idx': 88},
                        {'filter': 256, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True,
'layer_idx': 89},
                        {'filter': 512, 'kernel': 3, 'stride': 1, 'bnorm': True, 'leaky': True,
'layer_idx': 90},
                        {'filter': 256, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True,
'layer_idx': 91}], skip=False)
    # Layer 92 => 94
    yolo_94 = _conv_block(x, [{'filter': 512, 'kernel': 3, 'stride': 1, 'bnorm': True,  'leaky':
True,  'layer_idx': 92},
                              {'filter': 255, 'kernel': 1, 'stride': 1, 'bnorm': False, 'leaky': False,
'layer_idx': 93}], skip=False)
    # Layer 95 => 98
    x = _conv_block(x, [{'filter': 128, 'kernel': 1, 'stride': 1, 'bnorm': True, 'leaky': True,
'layer_idx': 96}], skip=False)
    x = UpSampling2D(2)(x)
    x = concatenate([x, skip_36])
    # Layer 99 => 106
    yolo_106 = _conv_block(x, [{'filter': 128, 'kernel': 1, 'stride': 1, 'bnorm': True,
'leaky': True,  'layer_idx': 99},
                               {'filter': 256, 'kernel': 3, 'stride': 1, 'bnorm': True,  'leaky': True,
'layer_idx': 100},
                               {'filter': 128, 'kernel': 1, 'stride': 1, 'bnorm': True,  'leaky': True,
'layer_idx': 101},
                               {'filter': 256, 'kernel': 3, 'stride': 1, 'bnorm': True,  'leaky': True,
'layer_idx': 102},
                               {'filter': 128, 'kernel': 1, 'stride': 1, 'bnorm': True,  'leaky': True,
'layer_idx': 103},
                               {'filter': 256, 'kernel': 3, 'stride': 1, 'bnorm': True,  'leaky': True,
```

```
'layer_idx': 104},
                        {'filter': 255, 'kernel': 1, 'stride': 1, 'bnorm': False, 'leaky': False,
'layer_idx': 105}], skip=False)
    model = Model(input_image, [yolo_82, yolo_94, yolo_106])
    return model
def preprocess_input(image, net_h, net_w):
    new_h, new_w, _ = image.shape
    # determine the new size of the image
    if (float(net_w)/new_w) < (float(net_h)/new_h):
        new_h = (new_h * net_w)/new_w
        new_w = net_w
    else:
        new_w = (new_w * net_h)/new_h
        new_h = net_h
    # resize the image to the new size
    resized = cv2.resize(image[:,:,::-1]/255., (int(new_w), int(new_h)))
    # embed the image into the standard letter box
    new_image = np.ones((net_h, net_w, 3)) * 0.5
    new_image[int((net_h-new_h)//2):int((net_h+new_h)//2), int((net_w-
new_w)//2):int((net_w+new_w)//2), :] = resized
    new_image = np.expand_dims(new_image, 0)
    return new_image
def decode_netout(netout, anchors, obj_thresh, nms_thresh, net_h, net_w):
    grid_h, grid_w = netout.shape[:2]
net_h, net_w = 416, 416
obj_thresh, nms_thresh = 0.5, 0.45
anchors = [[116,90,  156,198,  373,326],  [30,61, 62,45,  59,119], [10,13,  16,30,
33,23]]
labels = ["person", "bicycle", "car", "motorbike", "aeroplane", "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", "donut",
"cake", \
        "chair", "sofa", "pottedplant", "bed", "diningtable", "toilet", "tvmonitor",
"laptop", "mouse", \
        "remote", "keyboard", "cell phone", "microwave", "oven", "toaster", "sink",
"refrigerator", \
        "book", "clock", "vase", "scissors", "teddy bear", "hair drier", "toothbrush"]
# make the yolov3 model to predict 80 classes on COCO
yolov3 = make_yolov3_model()
# load the weights trained on COCO into the model
weight_reader = WeightReader(weights_path)
weight_reader.load_weights(yolov3)
# my defined functions
def intersection(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
```

Fig 9.1 Image Detection

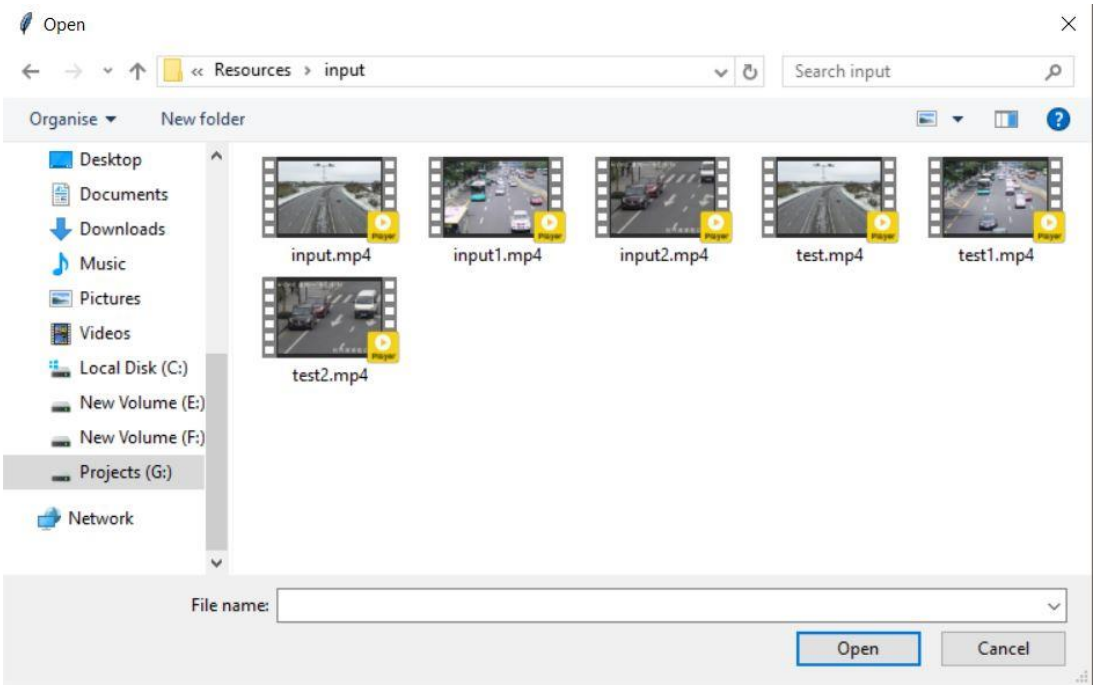

Fig 9.2 Image Saver

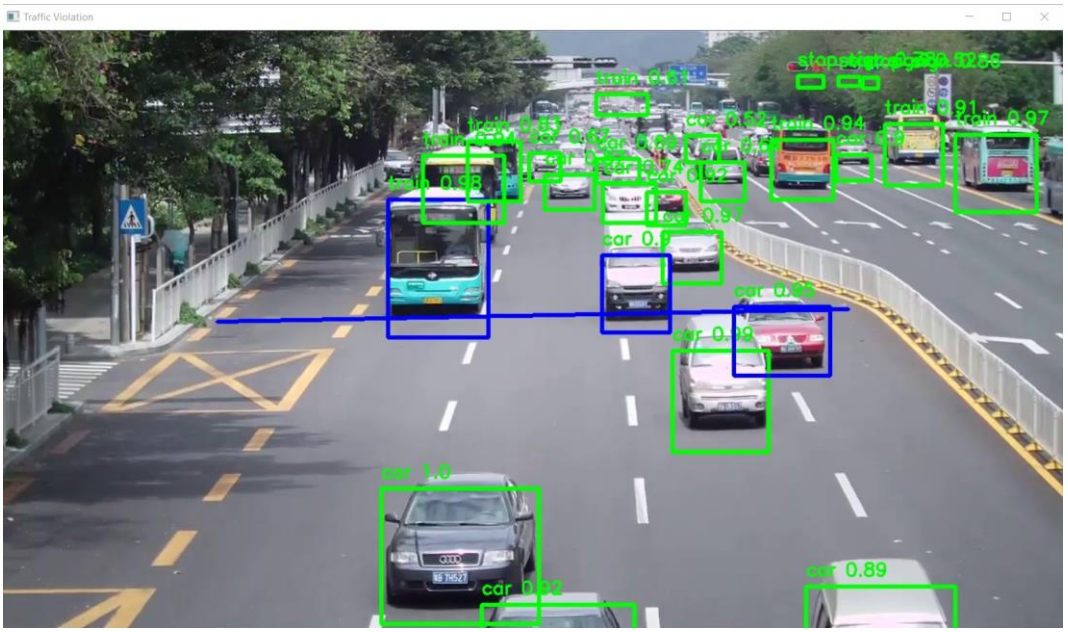

Fig 9.3 Violation Detection Frame

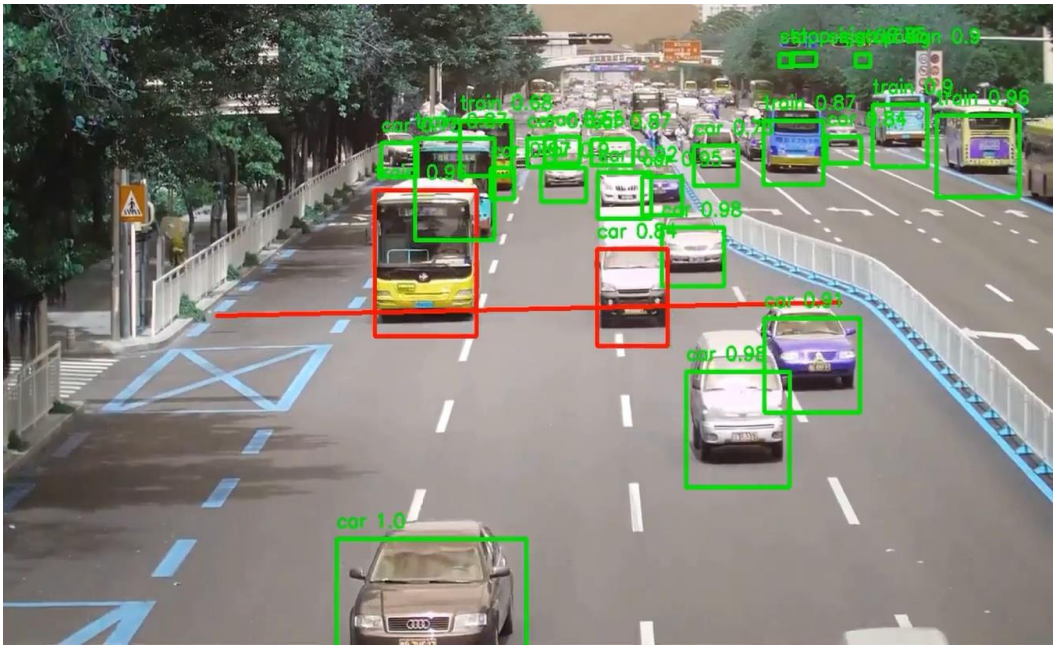

Fig 9.4 Violation Detection Frame Red

CHAPTER-10

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-11

FUTURE SCOPE

With the increasing growth in traffic density all over the world, it possesses a great challenge to traffic management. Emphasis should be that large area is covered and the high volume of traffic monitoring and detection from a single input source using parallel computation. Further enhancements are required to reduce computational time at high traffic volume.

CHAPTER-12

REFERENCES

1) "Traffic Rules Violation Detection using Deep Learning," Aniruddha Tonge, S. Chandak, R. Khiste, U. Khan and L. A. Bewoor, Published on 2020.
2) "Traffic Signal Violation Detection using Artificial Intelligence and Deep Learning", Ruben.J Franklin and Mohana, Published on 2020.
3) "Applications for traffic surveillance use object detection algorithms like convolution neural networks (CNN)". Chetan Kumar.B Published on [2020].
4) "Performance Evaluation of Background Modeling Methods for Object Detection and Tracking," Mohana, HV Ravish Aradhya Published on 2020.
5) "Background Modeling techniques for foreground detection and Tracking using Gaussian Mixture model" R.K. Meghana, Yojan Chitkara, Apoorva S., Mohana Published on 2019.
6) P. Arora, A. Dhar, T. Singh and A. Mishra, "Abandoned object identification and detection system for railways in India", *2017* International Conference on Emerging Trends in Computing and Communication Technologies *(ICETCCT)*, pp. 1-5, 2017.
7) M R Nehashree and Mohana Pallavi Raj S, "Simulation and Performance Analysis of Feature Extraction and Matching Algorithms for Image Processing Applications", International Conference on Intelligent Sustainable Systems (ICISS-2019), pp. 628-632.
8) A. Raghunandan, P. Raghav Mohana and H. V. R. Aradhya, "Object Detection Algorithms for Video Surveillance Applications", 2018 International Conference on Communication and Signal Processing (ICCSP), pp. 0563-0568, 2018.
9) A. Mangawati, M. Leesan Mohana and H. V. R. Aradhya, "Object Tracking Algorithms for Video Surveillance Applications", 2018 International Conference on Communication and Signal Processing (ICCSP), pp. 0667-0671, 2018.
10) Peng Xu et al., "Algorithms and system for segmentation and structure analysis in soccer video", IEEE International Conference on Multimedia and Expo 2001. ICME 2001., *pp. 721-724, 2001*.
11) V. P. Korakoppa, Mohana and H. V. R. Aradhya, "An area efficient FPGA implementation of moving object detection and face detection using adaptive threshold method", International Conference on Recent Trends in Electronics Information &

Communication Technology (RTEICT), *pp. 1606-1611, 2017.*

12) Anders Thirsgaard Rasmussen and Henrik Møller Rasmussen, "Tracking People in Sports using Video Analysis", Kongens Lyngby, 2008.

13) Evan Cheshire, Cibele Halasz and Jose Krause Perin, "Player Tracking and Analysis of Basketball Plays", European Conference of Computer Vision, Winter 2013/2014.

14) P. L. Mazzeo et al., "Visual Players Detection and Tracking in Soccer Matches", 2008 IEEE Fifth International Conference on Advanced Video and Signal Based Surveillance, pp. 326-333, 2008

15) Young Xu et al., "Background modelling methods for video analysis : A review and comparative study", The College of Computer Science and Technology, 2016.