THANTHAI PERIYAR GOVERNMENT INSTITUTE OF TECHNOLOGY

VELLORE-02.

TRAFFIC VIOLATION PROCTORING SYSTEM: HELMET DETECTION AND LICENSE PLATE EXTRACTION

A MINI PROJECT REPORT

Submitted by

JEEVITHA S	**513120104013**
MEGASRI M	**513120104021**
SARASWATHI V	**513120104030**
SIVASANKARI S	**513120104035**

In partial fulfilment for the award of degree

of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING
ANNA UNIVERSITY,

CHENNAI – 600 025.

THANTHAI PERIYAR GOVERNMENT INSTITUTE OF TECHNOLOGY

VELLORE-02.

BONAFIDE CERTIFICATE

This is to certify that the mini project report entitled **TRAFFIC VIOLATION PROCTORING SYSTEM: HELMET DETECTION AND LICENSE PLATE EXTRACTION** Project is a bonafide record of the project work done by **JEEVITHA S[513120104013], MEGASRI M[513120104035], SARASWATHI V[513120104035], SIVASANKARI S[513120104035**] during the academic year 2020-2024 towards the partial fulfillment of the requirement of the award of B- E Degree in **COMPUTER SCIENCE AND ENGINEERING** of Anna University, Chennai-600025.

Guided by

Prof. N.JAGADEESWARI , M.E,	**Dr. RAHILA BILAL, M.E.,Ph.D.,**
Assistant Professor,	Head of the Department/CSE,
Department of CSE,	TPGIT,Vellore.
TPGIT,Vellore.	

Internal Examiner **External Examiner**

ACKNOWLEDGEMENT

Project work is a practical application of our knowledge that we gather during the course. To make a project work successful, eminent person's involvement is very much required.

We thank our beloved former principal **Dr. M. ARULARASU, M.E., Ph.D.,** and principal **Dr.J.SREERAMBABU, M.E., Ph.D., PDF., FIE** who always served as a source of inspiration and encouragement.

We express our grateful thanks to Dr. **RAHILA BILAL, M.E., Ph.D.,** who has contributed his valuable suggestions in doing this project.

We have no words to express thank to our project supervisor, Prof. **N.JAGADEESWARI , M.E,** who gave fruitful ideas and constantly guided us to finish this project.

We extend our thanks to all the Teaching and Non-Teaching Staff of the Department of Computer science and Engineering and others sole help and cooperation in successful completion of this project.

Submitted by

JEEVITHA S [513120104013],

MEGASRI M [513120104021],

SARASWATHI V [513120104030],

SIVASANKARI S [513120104035] .

ABSTRACT

Violations in traffic laws are very common in a highly populated country like India. The accidents associated with these violations cause a huge loss to life and property. Since utilization of bikes is high, mishaps associated with bikes are additionally high contrasted with different vehicles. One of the main causes of these is not using motorcycle helmets. So we propose an approach called "**TRAFFIC VIOLATION PROCTORING SYSTEM: HELMET DETECTION AND LICENSE PLATE EXTRACTION** " using deep learning which automatically detects the rider who are not wearing a helmet and extract their license plate number and save it in one folder , and also capture their images in another folder.

The proposed approach detects and classify under two classifications called with helmet and without helmet. After detecting the vehicle rider without helmet the number plate recognition process takes place for the images which are having rider without the helmet

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

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

ACRONYM	ABBREVIATION
YOLO	You Only Look Once
CNN	Convolution Neural Network
ResNet	Residual Neural Network
RGB	Red Green Blue
ML	Machine Learning

CHAPTER 1

INTRODUCTION

OBJECTIVE

The main objective of the Project on TRAFFIC VIOLATION PROCTORING SYSTEM: HELMET DETECTION AND LICENSE PLATE EXTRACTION is to decrease the danger of injuries in case of accident. It is the system that detects people not wearing helmet so, it will increase their safety while on road. Hence by automating we reduce the workload on the traffic control team and will be able to share the evidence with the team efficiently to impose fines on violators.

PROJECT OVERVIEW

In this Project Work, a Non-Helmet Rider detection system is built which attempts to satisfy the automation of detecting the traffic violation of not wearing helmet and extracting the vehicles' license plate number. The main principle involved is Object Detection using Deep Learning. The objects detected are rider, license plate and rider's head using YOLOv5 algorithm. Then resnet50 image classifier is used to detect whether the rider is wearing a helmet or not. If rider not wearing a helmet then it will capture the license plate and store it in a folder, and also will capture rider image and store it in a separate folder.

CHAPTER 2

SYSTEM ANALYSIS

EXISTING SYSTEM

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

Deep Learning approach:

Single Shot Multibox Detection The paper about SSD:

Single Shot MultiBox Detector (by C. Szegedy et al.) was released at the end of November 2016 and reached new records in terms of performance and precision for object detection tasks, scoring over 74% mAP (mean Average Precision) at 59 frames per second onstandard datasets such as PascalVOC and COCO.

Convolutional predictors for object detection:

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

PROPOSED SYSTEM

In the proposed design, the model is trained with YOLO v5. The dataset collected is annotated and the images along with the label are fed to YOLO v5 for training and at the end weight files are created. The model weight file is loaded in the PyTorch and then the frames from recorded video are fed in order to start the process of detection.

Based on the YOLO v5 weight files from training, the proposed system detects and classify under two classifications called with helmet and without helmet. After detecting the vehicle rider without helmet the number plate recognition process takes place for the images which are having rider without the helmet.

The proposed model confidence score of 84% for rider, 92% for helmet, 90% for no helmet and 82% for number plate have been achieved.

CHAPTER 3

SYSTEM SPECIFICATION

HARDWARE COMPONENTS:

- Processor : Intel processor 3.0 GHz
- RAM : 4 GB
- Hard disk : 1 TB
- Compact Disk : 650 MB
- Keyboard : Standard keyboard
- Mouse : Logitech mouse
- Monitor : 15-inch color monitor
- System type : 64 -bit operating system
- OS : Windows OS

SOFTWARE REQUIREMENTS:

- Coding Language : Python
- Libraries : Opencv, PyTorch, TorchVision
- Code editor : VS code

CHAPTER 4

SOFTWARE DESCRIPTION

Python: -

The programming style of Python is simple, clear and it also contains powerful different kinds of classes. Moreover, Python can easily combine other programming languages, such as C or C++. As a successful programming language, ithas its own advantages:

- Simple and easy to learn
- Open source
- Scalability

OpenCV: -

OpenCV (Open source computer vision) is a library of programming functions mainly aimed at real- time computer vision. The library is cross-platform and free for use under the open-source BSD license. OpenCV supports the deep leaning framework TensorFlow, Torch/PyTorch and caffe.

NUMPY: -

In Python, there is data type called array. To implement the data type of array with python, NumPy is the essential library for analysing and calculating data. They are all open source libraries. NumPy is mainly used 22 for the matrix calculation.

TensorFlow:-

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

Pandas :-

pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language. Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. Matplotlib makes easy things easy and hard things possible.

pillow

Python Imaging Library (abbreviated as PIL) (in newer versions known as Pillow) is a free and open-source additional library for the Python programming language that adds support for opening, manipulating, and saving many different image file formats.

CHAPTER 5

SYSTEM DESIGN

INTRODUCTION:

In this application, the admin, staff and student are the authentications. Admin can add staff and student details and create a timetable for the live classes. Staff can view the student and subject details. staff can take the live classes according to the timetable generated by the admin and also share study materials. Students can view the study materials sent by staff and attend live classes.

SYSTEM ARCHITECTURE:

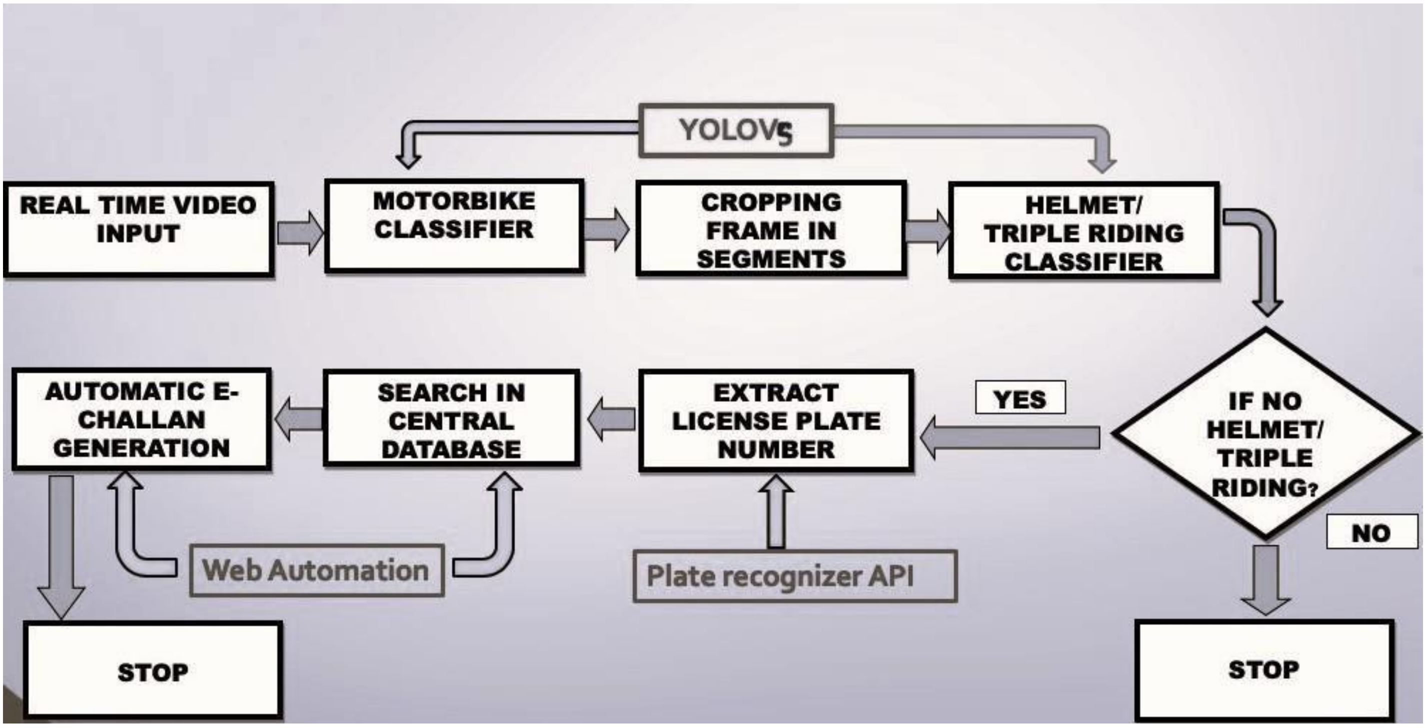


Fig: 5.1 Architecture diagram

USE-CASE DIAGRAM:

A use case diagram at its simplest is a representation of a user's interaction with the system that shows the relationship between the user and the different use cases. A use case diagram can identify the different types of users of a system and the different use cases and will often be accompanied by other types of diagrams as well.

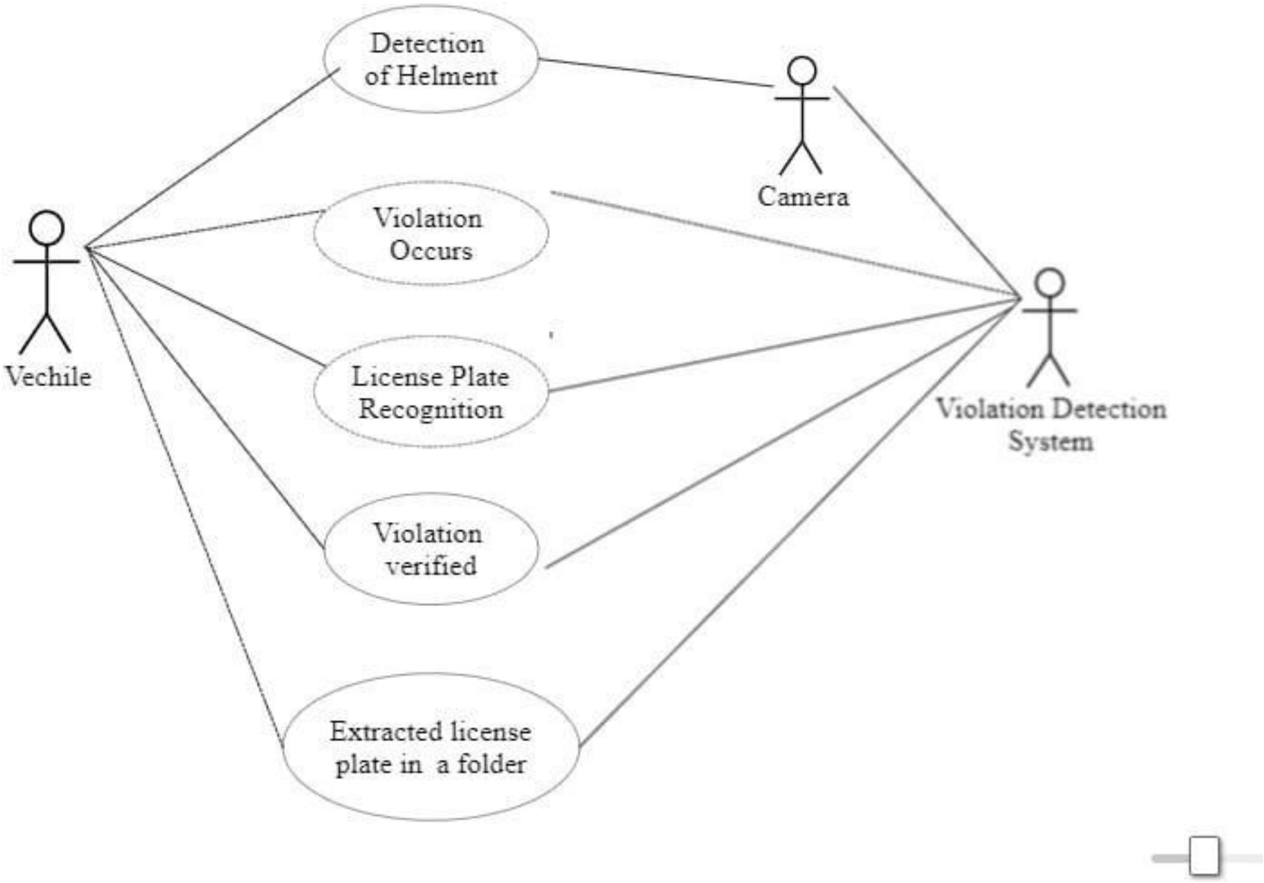


Fig 5.2 Usecase diagram

Class Diagram:

A class diagram in the Unified Modelling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.

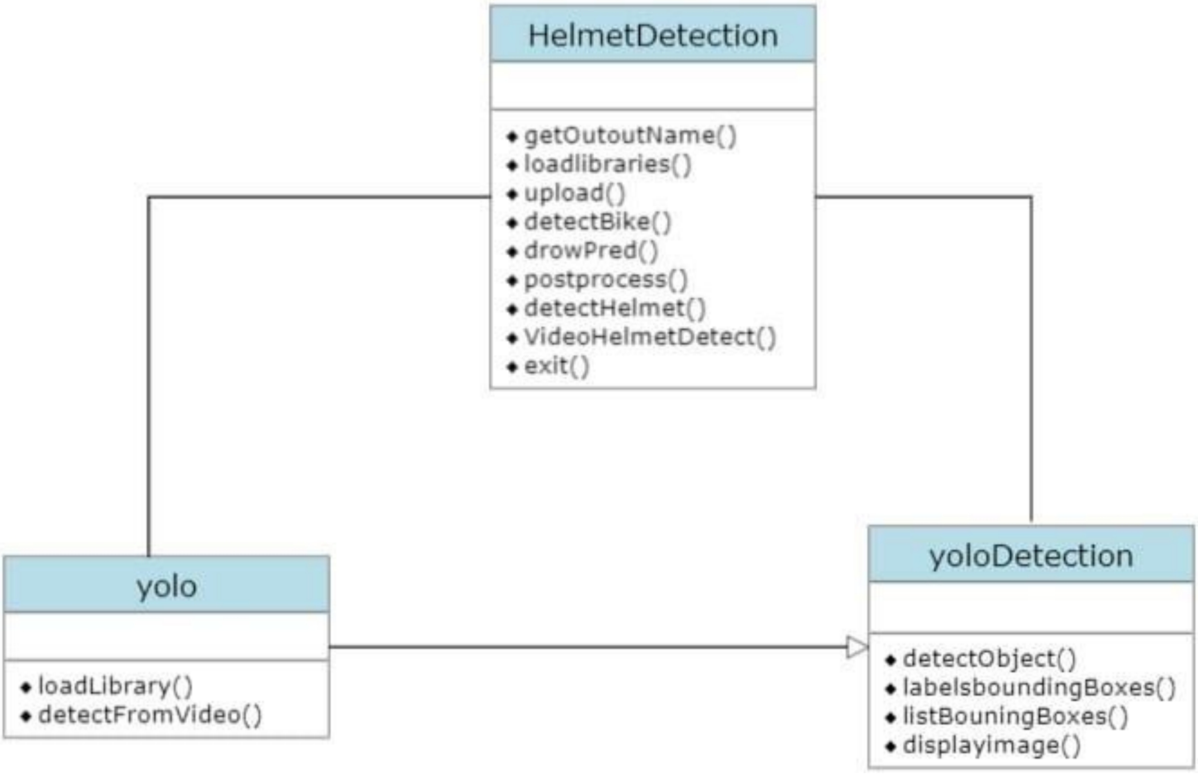


Fig 5.3 Class diagram

Activity Diagram:

An activity diagram is a behavioural diagram i.e., it depicts the behaviour of a system. activity diagram portrays the control flow from a start point to a finish point showing the various decision paths that exist while the activity.

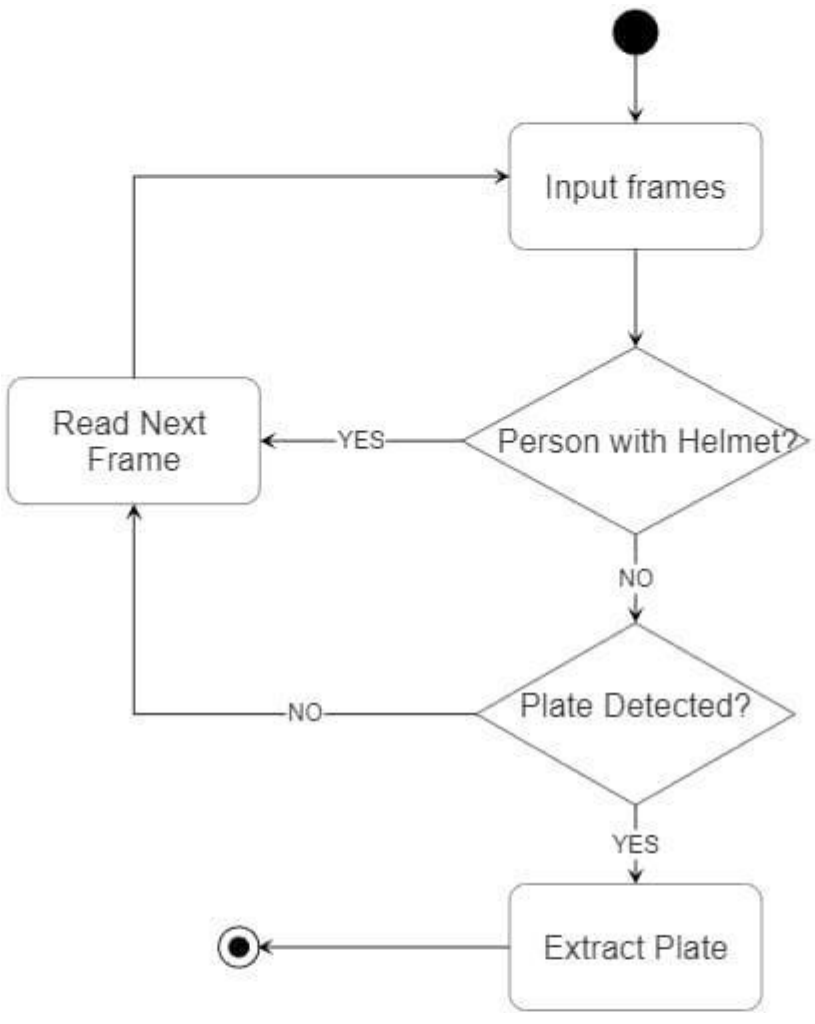


Fig 5.4 Activity diagram

Sequence Diagram:

A sequence diagram shows object interactions arranged in time sequence. It depicts the objects and classes involved in the scenario and the sequence of messages exchanged between the objects needed to carry out the functionality of the scenario.

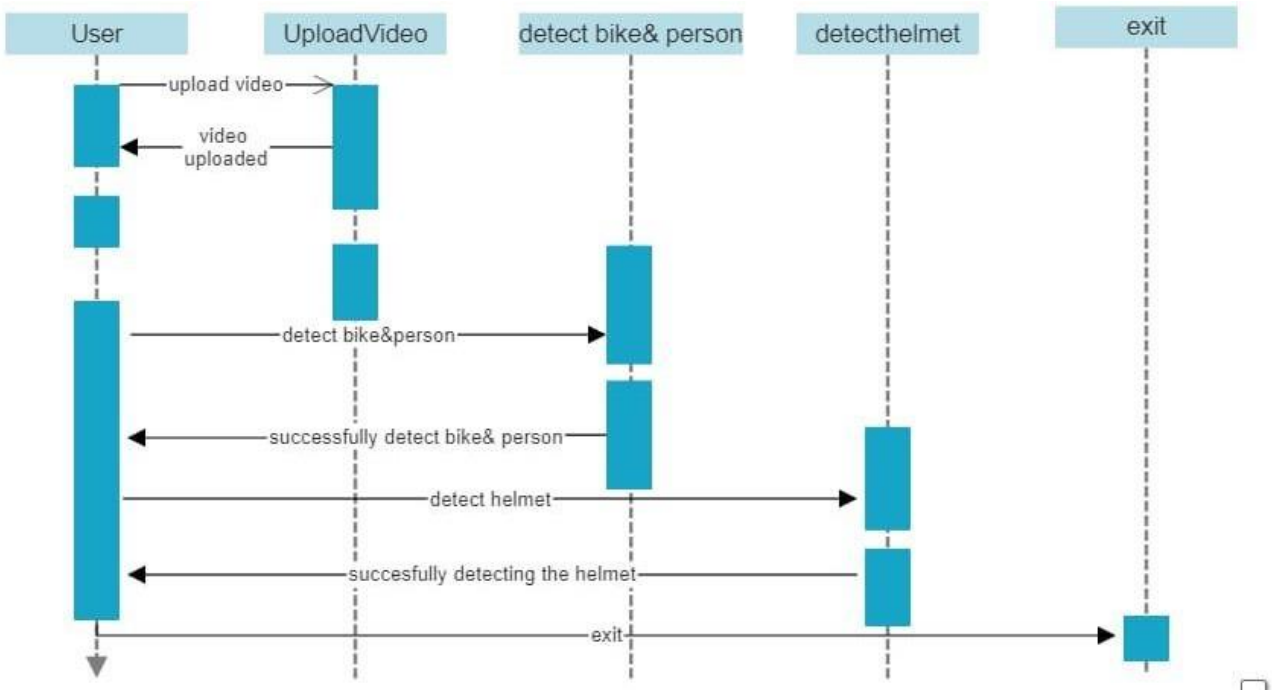


FIG 5.5 SEQUENCE DIAGRAM

CHAPTER – 6

PROJECT DESCRIPTION

In this Project Work, a Non-Helmet Rider detection system is built which attempts to satisfy the automation of detecting the traffic violation of not wearing helmet and extracting the vehicles license plate number. The main principle involved is objection detection using deep learning at three levels. The object detected are person, motor cycles at first level using YOLOv5, helmet at second level using YOLOv5, license plate at the last level using Web API. Then the license plate registration number is extracted using Resnet50 classifier.

Hence a database will be available for analysis for the police authority. The proposed approach initially recognizes motorcycle riders utilizing background subtraction and object segmentation. At that point we utilize object classifier to classify violators.

YOLOv5:

YOLOv5 (You Only Look Once, Version 3) is a real-time object detection algorithm that identifies specific objects in videos, live feeds, or images. YOLO is implemented using the Keras or OpenCV deep learning libraries. YOLO v5 algorithm consists of fully CNN and an algorithm for post-processing outputs from neural network. CNNs are special architecture of neural networks suitable for processing grid-like data topology. The distinctive feature of CNNs which bears importance in object detection is parameter sharing. This feature plays important role in capturing whole scene on the road.

CHAPTER-7

TESTING

Unit Testing

It is the testing of an individual unit or group of related units. It is done by the programmer to test that the implementation is producing expected output against given input and it falls under white box testing. Unit testing is done in order to check registrationwhether the user properly registered into the cloud. It is done in order to check whether a file is properly uploaded into the cloud. And an encryption and decryption are checked with unit testing if it is converted properly. Then deduplication is checked with unit testing.

Integration Testing

All the modules should be integrated into a single module and it should be checked thatit is still working still by integration testing.

System Testing

It is done to ensure that by putting the software in different environments and check thatit still works. System Testing is done by uploading same file in this cloud checking whetherany duplicate file exists.

Software Testing

It is the process of evaluating a software item to detect differences between given input and expected output. Testing assesses the quality of the product. In other words, software testing is a verification and validation process.

Verification

Verification is the process to make sure the product satisfies the conditions imposed at the start of the development phase. In other words, to make sure the product behaves the way we want it to.

Validation

Validation is the process to make sure the product satisfies the specified requirements at the end of the development phase. In other words, to make sure the product is built as per customer requirements.

Black Box Testing

Black box testing is a testing which ignores internal mechanism of system and focuses on output generated against any input and execution of system. It is done for validation. It is done to check encryption and decryption after uploading a file into the cloud.

White Box Testing

It is done for verification and it is a testing that takes into account the internal mechanism of the system. It is done by checking content verification. It will verify that whether same content exists in the cloud.

CHAPTER 8

CONCLUSION

CONCLUSION:

A Non-Helmet Rider Detection system is developed where a video file is taken as input. If the motorcycle rider in the video footage is not wearing helmet while riding the motorcycle, then the license plate number of that motorcycle is extracted and displayed for above cases separately. Object detection principle with YOLO architecture is used for motorcycle, person, helmet and license plate detection. ResNet classifier is used to classify whether the rider is wearing a helmet or not. If the rider is not wearing a helmet then his license plate is extracted and stored in a folder and the rider picture also captured and stored in separate folder. All the objectives of the project is achieved satisfactorily.

FUTURE WORK:

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

CHAPTER 9

APPENDICES

SOURCE CODE

MAIN CODE:

```
from my_functions import *

source = 'test_video.MOV'

save_video = True
show_video=True
save_img=False
fourcc = cv2.VideoWriter_fourcc(*'XVID')
out = cv2.VideoWriter('output.avi', fourcc, 20.0, frame_size)

cap = cv2.VideoCapture(source)
while(cap.isOpened()):
    ret, frame = cap.read()
    if ret == True:
            frame = cv2.resize(frame, frame_size)  # resizing image
            orifinal_frame = frame.copy()
            frame, results = object_detection(frame)


            rider_list = []
            head_list = []
            number_list = []


            for result in results:
                x1,y1,x2,y2,cnf, clas = result
                if clas == 0:
                    rider_list.append(result)
```

```
            elif clas == 1:
                head_list.append(result)
            elif clas == 2:
                number_list.append(result)

        for rdr in rider_list:
            time_stamp = str(time.time())
            x1r, y1r, x2r, y2r, cnfr, clasr = rdr
            for hd in head_list:
                x1h, y1h, x2h, y2h, cnfh, clash = hd
                if inside_box([x1r,y1r,x2r,y2r], [x1h,y1h,x2h,y2h]):
                    try:
                        head_img = orifinal_frame[y1h:y2h, x1h:x2h]
                        helmet_present = img_classify(head_img)
                    except:
                        helmet_present[0] = None

                    if  helmet_present[0] == True:  # if helmet present
                        frame = cv2.rectangle(frame, (x1h, y1h), (x2h, y2h),
(0,255,0), 1)
        frame = cv2.putText(frame, f'{round(helmet_present[1],1)}', (x1h, y1h+40),
          cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,0,255), 1, cv2.LINE_AA)

                   elif helmet_present[0] == None: # Poor prediction
          frame = cv2.rectangle(frame, (x1h, y1h), (x2h, y2h), (0, 255, 255), 1)
         frame = cv2.putText(frame, f'{round(helmet_present[1],1)}', (x1h, y1h),
          cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,0,255), 1, cv2.LINE_AA)
```

```
elif helmet_present[0] == False:
frame = cv2.rectangle(frame, (x1h, y1h), (x2h, y2h), (0, 0, 255), 1)
frame = cv2.putText(frame, f'{round(helmet_present[1],1)}', (x1h, y1h+40),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,0,255), 1, cv2.LINE_AA)
try:
cv2.imwrite(f'riders_pictures/{time_stamp}.jpg',
frame[y1r:y2r, x1r:x2r])
except:
print('could not save rider')


for num in number_list:
x1_num, y1_num, x2_num, y2_num, conf_num, clas_num = num
if inside_box([x1r,y1r,x2r,y2r], [x1_num, y1_num, x2_num, y2_num]):
try:
num_img = orifinal_frame[y1_num:y2_num,
x1_num:x2_num]
cv2.imwrite(f'number_plates/{time_stamp}_
{conf_num}.jpg', num_img)
except:
print('could not save number plate')


if save_video: # save video
    out.write(frame)
if save_img: #save img
    cv2.imwrite('saved_frame.jpg', frame)
if show_video: # show video
    frame = cv2.resize(frame, (900, 450)) # resizing to fit in screen
    cv2.imshow('Frame', frame)
```

```
            if cv2.waitKey(1) & 0xFF == ord('q'):
                break
        else:
            break

    cap.release()
    cv2.destroyAllWindows()
    print('Execution completed')
```

MY_FUNCTION CODE:

```
import cv2
import torch
import torch.backends.cudnn as cudnn
from models.experimental import attempt_load
from utils.general import non_max_suppression
from torchvision import models
from torchvision import transforms
from PIL import Image
import time

yolov5_weight_file = 'rider_helmet_number_small.pt' # ... may need full path
helmet_classifier_weight = 'helment_no_helmet98.6.pth'
conf_set=0.35
frame_size=(800, 480)
head_classification_threshold= 3.0 # make this value lower if want to detect non
helmet more aggresively;
```

```
# 1024, 576 # cs=4.1
# 928, 544
# 800, 480 # cs=3.9

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = attempt_load(yolov5_weight_file, map_location=device)
cudnn.benchmark = True
names = model.module.names if hasattr(model, 'module') else model.names

### Image classification
# labels = ['helmet', 'no helmet']
model2 = torch.load(helmet_classifier_weight, map_location=device)  # ... may
need full path
model2.eval()

transform = transforms.Compose([
            transforms.Resize(144),
            # transforms.CenterCrop(142),
            transforms.ToTensor(),
            transforms.Normalize([0.5], [0.5])
        ])


def img_classify(frame):
    # print('Head size: ',frame.shape[:-1])

    if frame.shape[0]<46 : # skiping small size heads < ----------------- you can adjust
this value
        return [None, 0]
```

```
    frame = transform(Image.fromarray(frame))
    frame = frame.unsqueeze(0)
    prediction = model2(frame)
    result_idx = torch.argmax(prediction).item()
    prediction_conf = sorted(prediction[0])

    cs = (prediction_conf[-1]-prediction_conf[-2]).item() # confident score
    # print(cs)
    # provide a threshold value of classification prediction as cs
    if cs > head_classification_threshold: #< --- Classification confident score. Need
to adjust, this value
        return [True, cs] if result_idx == 0 else [False, cs]
    else:
        return [None, cs]


def object_detection(frame):
    img = torch.from_numpy(frame)
    img = img.permute(2, 0, 1).float().to(device)
    img /= 255.0
    if img.ndimension() == 3:
        img = img.unsqueeze(0)

    pred = model(img, augment=False)[0]
    pred = non_max_suppression(pred, conf_set, 0.30) # prediction, conf, iou

    detection_result = []
    for i, det in enumerate(pred):
        if len(det):
            for d in det: # d = (x1, y1, x2, y2, conf, cls)
                x1 = int(d[0].item())
```

```
                y1 = int(d[1].item())
                x2 = int(d[2].item())
                y2 = int(d[3].item())
                conf = round(d[4].item(), 2)
                c = int(d[5].item())
                detected_name = names[c]

                print(f'Detected: {detected_name} conf: {conf}  bbox: 
x1:{x1}    y1:{y1}    x2:{x2}    y2:{y2}')
                detection_result.append([x1, y1, x2, y2, conf, c])
                frame = cv2.rectangle(frame, (x1, y1), (x2, y2), (255,0,0), 1) 
# box
                if c!=1: # if it is not head bbox, then write use putText
                    frame = cv2.putText(frame, f'{names[c]} {str(conf)}', 
(x1, y1), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,0,255), 1, cv2.LINE_AA)

    return (frame, detection_result)


def inside_box(big_box, small_box):
    x1 = small_box[0] - big_box[0]
    y1 = small_box[1] - big_box[1]
    x2 = big_box[2] - small_box[2]
    y2 = big_box[3] - small_box[3]
    return not bool(min([x1, y1, x2, y2, 0]))
```

YOLO CODE:

```
# YOLOv5 YOLO-specific modules
import argparse
import logging
import sys
from copy import deepcopy
sys.path.append('./')  # to run '$ python *.py' files in subdirectories
logger = logging.getLogger(__name__)
from models.common import *
from models.experimental import *
from utils.autoanchor import check_anchor_order
from utils.general import make_divisible, check_file, set_logging
from utils.torch_utils import time_synchronized, fuse_conv_and_bn, model_info,
scale_img, initialize_weights, \
    select_device, copy_attr

try:
    import thop  # for FLOPS computation
except ImportError:
    thop = None
class Detect(nn.Module):
    stride = None  # strides computed during build
    export = False # onnx export

    def___init__(self, nc=80, anchors=(), ch=()):  # detection layer
        super(Detect, self).__init__()
        self.nc = nc  # number of classes
        self.no = nc + 5 # number of outputs per anchor
        self.nl = len(anchors)  # number of detection layers
        self.na = len(anchors[0]) // 2  # number of anchors
```

```
        self.grid = [torch.zeros(1)] * self.nl  # init grid
        a = torch.tensor(anchors).float().view(self.nl, -1, 2)
        self.register_buffer('anchors', a) # shape(nl,na,2)
        self.register_buffer('anchor_grid', a.clone().view(self.nl, 1, -1, 1, 1, 2))  #
shape(nl,1,na,1,1,2)
        self.m = nn.ModuleList(nn.Conv2d(x, self.no * self.na, 1) for x in ch)  # output
conv

    def forward(self, x):
        # x = x.copy()  # for profiling
        z = [] # inference output
        self.training |= self.export
        for i in range(self.nl):
            x[i] = self.m[i](x[i])  # conv
            bs, _, ny, nx = x[i].shape  # x(bs,255,20,20) to x(bs,3,20,20,85)
     x[i] = x[i].view(bs, self.na, self.no, ny, nx).permute(0, 1, 3, 4, 2).contiguous()

            if not self.training:  # inference
                if self.grid[i].shape[2:4] != x[i].shape[2:4]:
                    self.grid[i] = self._make_grid(nx, ny).to(x[i].device)

                y = x[i].sigmoid()
                y[..., 0:2] = (y[..., 0:2] * 2. - 0.5 + self.grid[i]) * self.stride[i]  # xy
                y[..., 2:4] = (y[..., 2:4] * 2) ** 2 * self.anchor_grid[i]  # wh
                z.append(y.view(bs, -1, self.no))

        return x if self.training else (torch.cat(z, 1), x)

    @staticmethod
    def _make_grid(nx=20, ny=20):
```

```
        yv, xv = torch.meshgrid([torch.arange(ny), torch.arange(nx)])
        return torch.stack((xv, yv), 2).view((1, 1, ny, nx, 2)).float()


class Model(nn.Module):
    def__init__(self, cfg='yolov5s.yaml', ch=3, nc=None, anchors=None):  # model,
input channels, number of classes
        super(Model, self).__init__()
        if isinstance(cfg, dict):
            self.yaml = cfg  # model dict
        else:  # is *.yaml
            import yaml # for torch hub
            self.yaml_file = Path(cfg).name
            with open(cfg) as f:
                self.yaml = yaml.safe_load(f)  # model dict

        # Define model
        ch = self.yaml['ch'] = self.yaml.get('ch', ch)  # input channels
        if nc and nc != self.yaml['nc']:
            logger.info(f"Overriding model.yaml nc={self.yaml['nc']} with nc={nc}")
            self.yaml['nc'] = nc # override yaml value
        if anchors:
            logger.info(f'Overriding model.yaml anchors with anchors={anchors}')
            self.yaml['anchors'] = round(anchors)  # override yaml value
        self.model, self.save = parse_model(deepcopy(self.yaml), ch=[ch])  # model,
savelist
        self.names = [str(i) for i in range(self.yaml['nc'])]  # default names
        # logger.info([x.shape for x in self.forward(torch.zeros(1, ch, 64, 64))])

        # Build strides, anchors
```

```
        m = self.model[-1]  # Detect()
        if isinstance(m, Detect):
            s = 256  # 2x min stride
    m.stride = torch.tensor([s / x.shape[-2] for x in self.forward(torch.zeros(1, ch, s,
                                                s))]) # forward
            m.anchors /= m.stride.view(-1, 1, 1)
            check_anchor_order(m)
            self.stride = m.stride
            self._initialize_biases()  # only run once
            # logger.info('Strides: %s' % m.stride.tolist())
# Init weights, biases
        initialize_weights(self)
        self.info()
        logger.info('')

    def forward(self, x, augment=False, profile=False):
        if augment:
            img_size = x.shape[-2:]  # height, width
            s = [1, 0.83, 0.67] # scales
            f = [None, 3, None]  # flips (2-ud, 3-lr)
            y = [] # outputs
            for si, fi in zip(s, f):
                xi = scale_img(x.flip(fi) if fi else x, si, gs=int(self.stride.max()))
                yi = self.forward_once(xi)[0] # forward
                # cv2.imwrite(f'img_{si}.jpg', 255 * xi[0].cpu().numpy().transpose((1, 2,
0))[:, :, ::-1])  # save
                yi[..., :4] /= si  # de-scale
                if fi == 2:
                    yi[..., 1] = img_size[0] - yi[..., 1]  # de-flip ud
                elif fi == 3:
```

```
                yi[..., 0] = img_size[1] - yi[..., 0]  # de-flip lr
            y.append(yi)
        return torch.cat(y, 1), None  # augmented inference, train
    else:
        return self.forward_once(x, profile)  # single-scale inference, train

def forward_once(self, x, profile=False):
    y, dt = [], [] # outputs
    for m in self.model:
        if m.f != -1:  # if not from previous layer
            x = y[m.f] if isinstance(m.f, int) else [x if j == -1 else y[j] for j in m.f]  #
from earlier layers

        if profile:
            o = thop.profile(m, inputs=(x,), verbose=False)[0] / 1E9 * 2 if thop else 0
# FLOPS
            t = time_synchronized()
            for _ in range(10):
                _ = m(x)
            dt.append((time_synchronized() - t) * 100)
            logger.info('%10.1f%10.0f%10.1fms %-40s' % (o, m.np, dt[-1], m.type))

        x = m(x)  # run
        y.append(x if m.i in self.save else None)  # save output

    if profile:
        logger.info('%.1fms total' % sum(dt))
    return x

def _initialize_biases(self, cf=None):  # initialize biases into Detect(), cf is class
```

```
frequency
        # https://arxiv.org/abs/1708.02002 section 3.3
        # cf = torch.bincount(torch.tensor(np.concatenate(dataset.labels, 0)[:,
0]).long(), minlength=nc) + 1.
        m = self.model[-1] # Detect() module
        for mi, s in zip(m.m, m.stride):  # from
            b = mi.bias.view(m.na, -1)  # conv.bias(255) to (3,85)
            b.data[:, 4] += math.log(8 / (640 / s) ** 2) # obj (8 objects per 640 image)
            b.data[:, 5:] += math.log(0.6 / (m.nc - 0.99)) if cf is None else torch.log(cf /
cf.sum())  # cls
            mi.bias = torch.nn.Parameter(b.view(-1), requires_grad=True)

    def _print_biases(self):
        m = self.model[-1]  # Detect() module
        for mi in m.m:  # from
            b = mi.bias.detach().view(m.na, -1).T # conv.bias(255) to (3,85)
            logger.info(('%6g Conv2d.bias:' + '%10.3g' * 6) % (mi.weight.shape[1],
*b[:5].mean(1).tolist(), b[5:].mean()))

    # def _print_weights(self):
    #     for m in self.model.modules():
    #         if type(m) is Bottleneck:
    #             logger.info('%10.3g' % (m.w.detach().sigmoid() * 2))  # shortcut
weights

    def fuse(self):  # fuse model Conv2d() + BatchNorm2d() layers
        logger.info('Fusing layers... ')
        for m in self.model.modules():
            if type(m) is Conv and hasattr(m, 'bn'):
                m.conv = fuse_conv_and_bn(m.conv, m.bn)  # update conv
```

```
            delattr(m, 'bn')  # remove batchnorm
            m.forward = m.fuseforward  # update forward
        self.info()
        return self

    def nms(self, mode=True): # add or remove NMS module
        present = type(self.model[-1]) is NMS  # last layer is NMS
        if mode and not present:
            logger.info('Adding NMS... ')
            m = NMS() # module
            m.f = -1  # from
            m.i = self.model[-1].i + 1 # index
            self.model.add_module(name='%s' % m.i, module=m)  # add
            self.eval()
        elif not mode and present:
            logger.info('Removing NMS... ')
            self.model = self.model[:-1]  # remove
        return self

    def autoshape(self):  # add autoShape module
        logger.info('Adding autoShape... ')
        m = autoShape(self)  # wrap model
        copy_attr(m, self, include=('yaml', 'nc', 'hyp', 'names', 'stride'), exclude=())   #
copy attributes
        return m

    def info(self, verbose=False, img_size=640):  # print model information
        model_info(self, verbose, img_size)
```

```
def parse_model(d, ch):  # model_dict, input_channels(3)
    logger.info('\n%3s%18s%3s%10s    %-40s%-30s' % ('', 'from', 'n', 'params', 
'module', 'arguments'))
    anchors, nc, gd, gw = d['anchors'], d['nc'], d['depth_multiple'], d['width_multiple']
    na = (len(anchors[0]) // 2) if isinstance(anchors, list) else anchors  # number of 
anchors
    no = na * (nc + 5)  # number of outputs = anchors * (classes + 5)

    layers, save, c2 = [], [], ch[-1]  # layers, savelist, ch out
    for i, (f, n, m, args) in enumerate(d['backbone'] + d['head']):  # from, number, 
module, args
        m = eval(m) if isinstance(m, str) else m  # eval strings
        for j, a in enumerate(args):
            try:
                args[j] = eval(a) if isinstance(a, str) else a  # eval strings
            except:
                pass

        n = max(round(n * gd), 1) if n > 1 else n  # depth gain
        if m in [Conv, GhostConv, Bottleneck, GhostBottleneck, SPP, DWConv, 
MixConv2d, Focus, CrossConv, BottleneckCSP,
                 C3, C3TR]:
            c1, c2 = ch[f], args[0]
            if c2 != no:  # if not output
                c2 = make_divisible(c2 * gw, 8)

            args = [c1, c2, *args[1:]]
            if m in [BottleneckCSP, C3, C3TR]:
                args.insert(2, n)  # number of repeats
                n = 1
```

```
        elif m is nn.BatchNorm2d:
            args = [ch[f]]
        elif m is Concat:
            c2 = sum([ch[x] for x in f])
        elif m is Detect:
            args.append([ch[x] for x in f])
            if isinstance(args[1], int):  # number of anchors
                args[1] = [list(range(args[1] * 2))] * len(f)
        elif m is Contract:
            c2 = ch[f] * args[0] ** 2
        elif m is Expand:
            c2 = ch[f] // args[0] ** 2
        else:
            c2 = ch[f]

        m_ = nn.Sequential(*[m(*args) for _ in range(n)]) if n > 1 else m(*args)  #
module
        t = str(m)[8:-2].replace('__main__.', '')  # module type
        np = sum([x.numel() for x in m_.parameters()])  # number params
        m_.i, m_.f, m_.type, m_.np = i, f, t, np  # attach index, 'from' index, type,
number params
        logger.info('%3s%18s%3s%10.0f %-40s%-30s' % (i, f, n, np, t, args)) # print
        save.extend(x % i for x in ([f] if isinstance(f, int) else f) if x != -1) # append to
savelist
        layers.append(m_)
        if i == 0:
            ch = []
        ch.append(c2)
    return nn.Sequential(*layers), sorted(save)
```

```
if __name__== '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('--cfg', type=str, default='yolov5s.yaml', help='model.yaml')
    parser.add_argument('--device', default='', help='cuda device, i.e. 0 or 0,1,2,3 or
cpu')
    opt = parser.parse_args()
    opt.cfg = check_file(opt.cfg)  # check file
    set_logging()
    device = select_device(opt.device)

    # Create model
    model = Model(opt.cfg).to(device)
    model.train()

    # Profile
    # img = torch.rand(8 if torch.cuda.is_available() else 1, 3, 320, 320).to(device)
    # y = model(img, profile=True)

    # Tensorboard (not working https://github.com/ultralytics/yolov5/issues/2898)
    # from torch.utils.tensorboard import SummaryWriter
    # tb_writer = SummaryWriter('.')
    # logger.info("Run 'tensorboard --logdir=models' to view tensorboard at
http://localhost:6006/")
    # tb_writer.add_graph(torch.jit.trace(model, img, strict=False), [])  # add model
graph
    # tb_writer.add_image('test', img[0], dataformats='CWH')  # add model to
tensorboard
```

EXPORT CODE :

```
import argparse
import sys
import time
sys.path.append('./')  # to run '$ python *.py' files in subdirectories
import torch
import torch.nn as nn
from torch.utils.mobile_optimizer import optimize_for_mobile
import models
from models.experimental import attempt_load
from utils.activations import Hardswish, SiLU
from utils.general import colorstr, check_img_size, check_requirements, file_size,
set_logging
from utils.torch_utils import select_device

if __name__== '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('--weights', type=str, default='./yolov5s.pt', help='weights
path')
    parser.add_argument('--img-size', nargs='+', type=int, default=[640, 640],
help='image size') # height, width
    parser.add_argument('--batch-size', type=int, default=1, help='batch size')
    parser.add_argument('--grid', action='store_true', help='export Detect() layer
grid')
    parser.add_argument('--device', default='cpu', help='cuda device, i.e. 0 or 0,1,2,3
or cpu')
    parser.add_argument('--dynamic', action='store_true', help='dynamic ONNX
axes') # ONNX-only
    parser.add_argument('--simplify', action='store_true', help='simplify ONNX
model')  # ONNX-only
```

```
    opt = parser.parse_args()
    opt.img_size *= 2 if len(opt.img_size) == 1 else 1  # expand
    print(opt)
    set_logging()
    t = time.time()

    # Load PyTorch model
    device = select_device(opt.device)
    model = attempt_load(opt.weights, map_location=device)  # load FP32 model
    labels = model.names

    # Checks
    gs = int(max(model.stride))  # grid size (max stride)
    opt.img_size = [check_img_size(x, gs) for x in opt.img_size]  # verify img_size
are gs-multiples

    # Input
    img = torch.zeros(opt.batch_size, 3, *opt.img_size).to(device)  # image
size(1,3,320,192) iDetection

    # Update model
    for k, m in model.named_modules():
        m._non_persistent_buffers_set = set()  # pytorch 1.6.0 compatibility
        if isinstance(m, models.common.Conv):  # assign export-friendly activations
            if isinstance(m.act, nn.Hardswish):
                m.act = Hardswish()
            elif isinstance(m.act, nn.SiLU):
                m.act = SiLU()
        # elif isinstance(m, models.yolo.Detect):
        #     m.forward = m.forward_export  # assign forward (optional)
```

```
    model.model[-1].export = not opt.grid  # set Detect() layer grid export
    for _ in range(2):
        y = model(img) # dry runs
    print(f"\n{colorstr('PyTorch:')} starting from {opt.weights}
({file_size(opt.weights):.1f} MB)")

    # TorchScript export -____________________-______________________________________________-
_____________________-
    prefix = colorstr('TorchScript:')
    try:
        print(f'\n{prefix} starting export with torch {torch.__version__}. ')
        f = opt.weights.replace('.pt', '.torchscript.pt')  # filename
        ts = torch.jit.trace(model, img, strict=False)
        ts = optimize_for_mobile(ts) #
https://pytorch.org/tutorials/recipes/script_optimized.html
        ts.save(f)
        print(f'{prefix} export success, saved as {f} ({file_size(f):.1f} MB)')
    except Exception as e:
        print(f'{prefix} export failure: {e}')

    # ONNX export -__________________________________________________________________________-
-------------------------
    prefix = colorstr('ONNX:')
    try:
        import onnx

        print(f'{prefix} starting export with onnx {onnx.__version__}...')
        f = opt.weights.replace('.pt', '.onnx')  # filename
        torch.onnx.export(model, img, f, verbose=False, opset_version=12,
input_names=['images'],
```

```
                    dynamic_axes={'images': {0: 'batch', 2: 'height', 3: 'width'},
                                  'output': {0: 'batch', 2: 'y', 3: 'x'}} if opt.dynamic else
None)

        # Checks
        model_onnx = onnx.load(f) # load onnx model
        onnx.checker.check_model(model_onnx)  # check onnx model
        # print(onnx.helper.printable_graph(model_onnx.graph))  # print

        # Simplify
        if opt.simplify:
            try:
                check_requirements(['onnx-simplifier'])
                import onnxsim

                print(f'{prefix} simplifying with onnx-simplifier
{onnxsim.__version__}...')
                model_onnx, check = onnxsim.simplify(model_onnx,
                                                     dynamic_input_shape=opt.dynamic,
                                                     input_shapes={'images': list(img.shape)} if
opt.dynamic else None)
                assert check, 'assert check failed'
                onnx.save(model_onnx, f)
            except Exception as e:
                print(f'{prefix} simplifier failure: {e}')
        print(f'{prefix} export success, saved as {f} ({file_size(f):.1f} MB)')
    except Exception as e:
        print(f'{prefix} export failure: {e}')

    # CoreML export
```

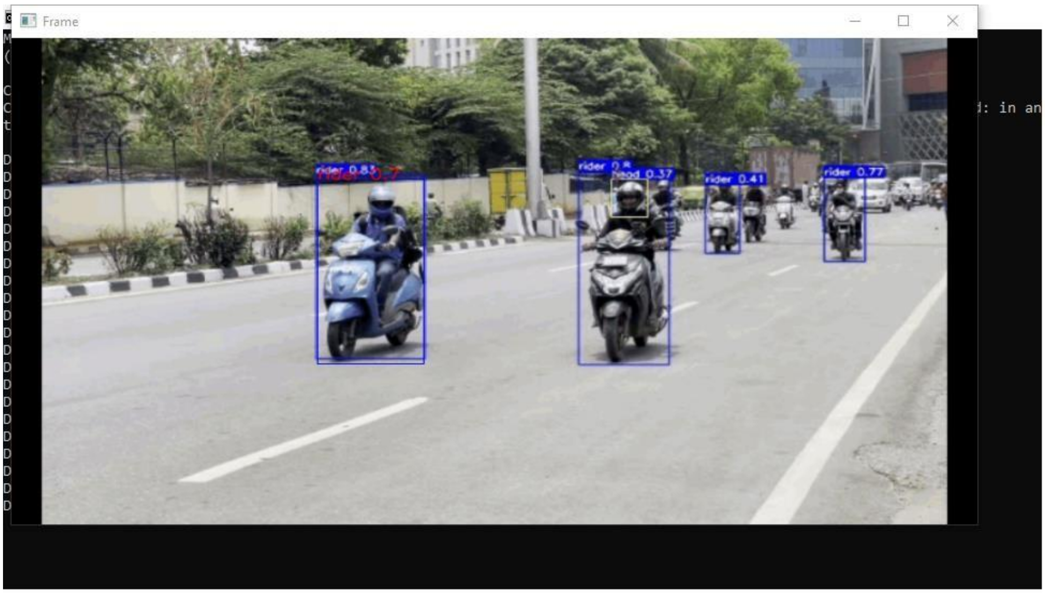
```
    prefix = colorstr('CoreML:')
    try:
        import coremltools as ct

        print(f'{prefix} starting export with coremltools {ct.__version__}...')
        # convert model from torchscript and apply pixel scaling as per detect.py
        model = ct.convert(ts, inputs=[ct.ImageType(name='image', shape=img.shape,
scale=1 / 255.0, bias=[0, 0, 0])])
        f = opt.weights.replace('.pt', '.mlmodel')  # filename
        model.save(f)
        print(f'{prefix} export success, saved as {f} ({file_size(f):.1f} MB)')
    except Exception as e:
        print(f'{prefix} export failure: {e}')

    # Finish
 print(f'\nExport complete ({time.time() - t:.2f}s). Visualize with
https://github.com/lutzroeder/netron.')
```

OUTPUT SCREENSHOTS

PROGRAM EXECUTION:

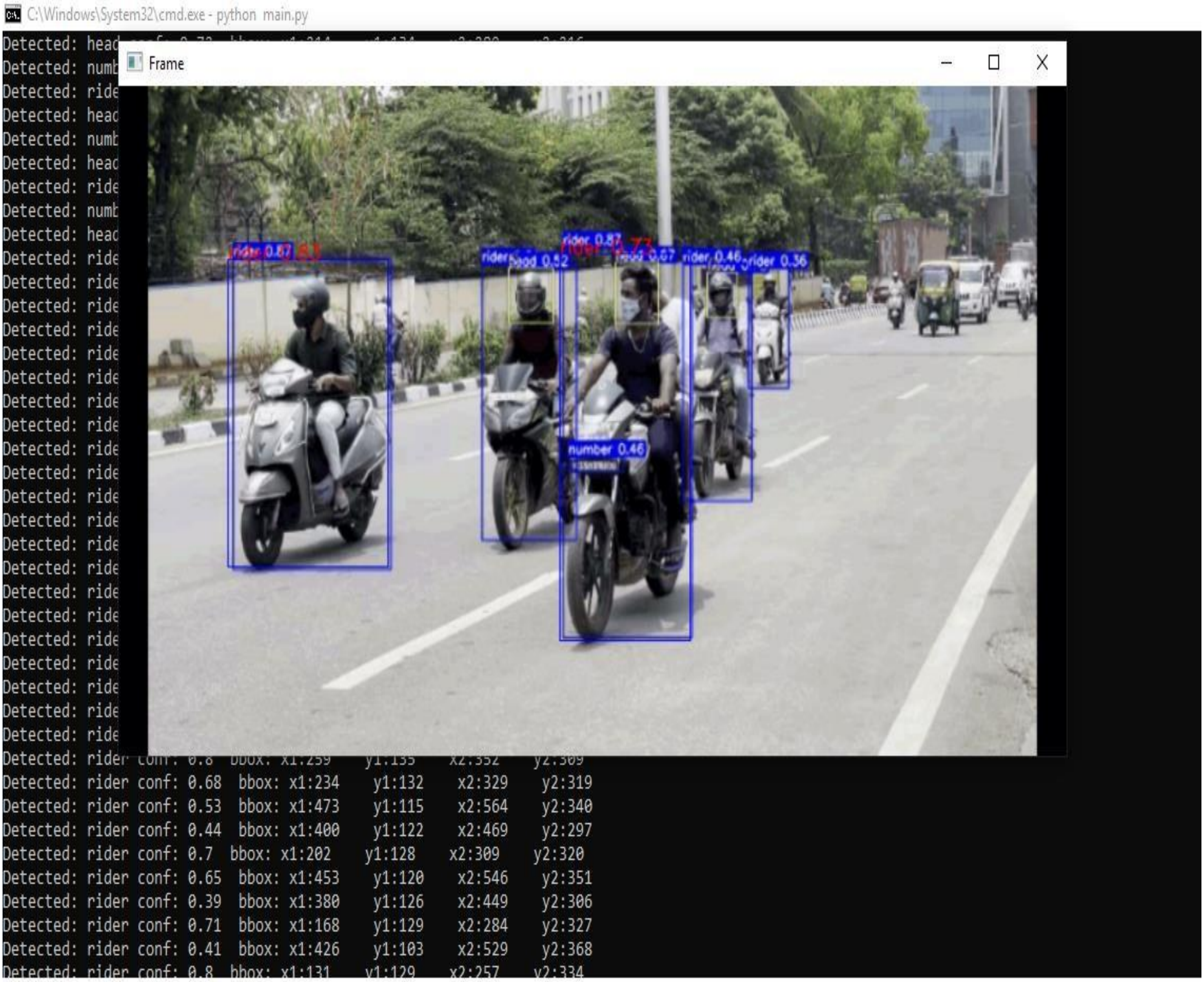
```
Detected: rider conf: 0.8  bbox: x1:131    y1:129    x2:257    y2:334
Detected: rider conf: 0.79  bbox: x1:406    y1:125    x2:509    y2:377
Detected: rider conf: 0.83  bbox: x1:92    y1:124    x2:227    y2:344
Detected: rider conf: 0.73  bbox: x1:372    y1:120    x2:482    y2:397
Detected: rider conf: 0.84  bbox: x1:52    y1:127    x2:196    y2:354
Detected: rider conf: 0.75  bbox: x1:336    y1:117    x2:457    y2:409
Detected: number conf: 0.42  bbox: x1:337    y1:257    x2:414    y2:280
Detected: rider conf: 0.82  bbox: x1:29    y1:124    x2:149    y2:368
Detected: rider conf: 0.71  bbox: x1:300    y1:115    x2:423    y2:432
Detected: number conf: 0.45  bbox: x1:302    y1:272    x2:377    y2:293
Detected: number conf: 0.65  bbox: x1:253    y1:288    x2:332    y2:313
Detected: rider conf: 0.63  bbox: x1:257    y1:107    x2:403    y2:469
Detected: rider conf: 0.61  bbox: x1:22    y1:124    x2:116    y2:371
Detected: rider conf: 0.57  bbox: x1:184    y1:115    x2:285    y2:374
Detected: number conf: 0.67  bbox: x1:197    y1:294    x2:281    y2:326
Detected: rider conf: 0.6  bbox: x1:205    y1:117    x2:367    y2:480
Detected: rider conf: 0.41  bbox: x1:127    y1:113    x2:231    y2:401
Detected: rider conf: 0.73  bbox: x1:130    y1:114    x2:332    y2:480
Detected: number conf: 0.58  bbox: x1:132    y1:316    x2:212    y2:351
Detected: rider conf: 0.77  bbox: x1:39    y1:113    x2:294    y2:481
Detected: number conf: 0.74  bbox: x1:58    y1:338    x2:139    y2:377
Detected: head conf: 0.71  bbox: x1:45    y1:122    x2:105    y2:191
Detected: rider conf: 0.7  bbox: x1:351    y1:120    x2:439    y2:352
Detected: head conf: 0.52  bbox: x1:156    y1:122    x2:224    y2:201
Detected: rider conf: 0.8  bbox: x1:314    y1:116    x2:418    y2:361
Detected: rider conf: 0.57  bbox: x1:18    y1:112    x2:232    y2:486
Detected: rider conf: 0.81  bbox: x1:280    y1:119    x2:388    y2:367
Detected: rider conf: 0.81  bbox: x1:235    y1:111    x2:357    y2:393
Detected: number conf: 0.46  bbox: x1:233    y1:254    x2:302    y2:274
Detected: rider conf: 0.8  bbox: x1:185    y1:116    x2:322    y2:412
Detected: number conf: 0.55  bbox: x1:187    y1:261    x2:256    y2:285
Detected: rider conf: 0.82  bbox: x1:127    y1:107    x2:280    y2:443
Detected: number conf: 0.72  bbox: x1:133    y1:275    x2:196    y2:297
Detected: rider conf: 0.81  bbox: x1:56    y1:116    x2:240    y2:463
Detected: number conf: 0.74  bbox: x1:60    y1:291    x2:139    y2:316
Detected: head conf: 0.53  bbox: x1:137    y1:130    x2:192    y2:198
Detected: rider conf: 0.82  bbox: x1:25    y1:115    x2:183    y2:476
Detected: head conf: 0.6  bbox: x1:66    y1:134    x2:129    y2:203
Detected: rider conf: 0.53  bbox: x1:118    y1:133    x2:198    y2:323
Detected: rider conf: 0.78  bbox: x1:53    y1:121    x2:162    y2:333
Detected: rider conf: 0.59  bbox: x1:18    y1:131    x2:123    y2:348
Execution completed
```

RECORDED VIDEO SHOWN UP:



RIDER WITH HELMET IS DETECTED:

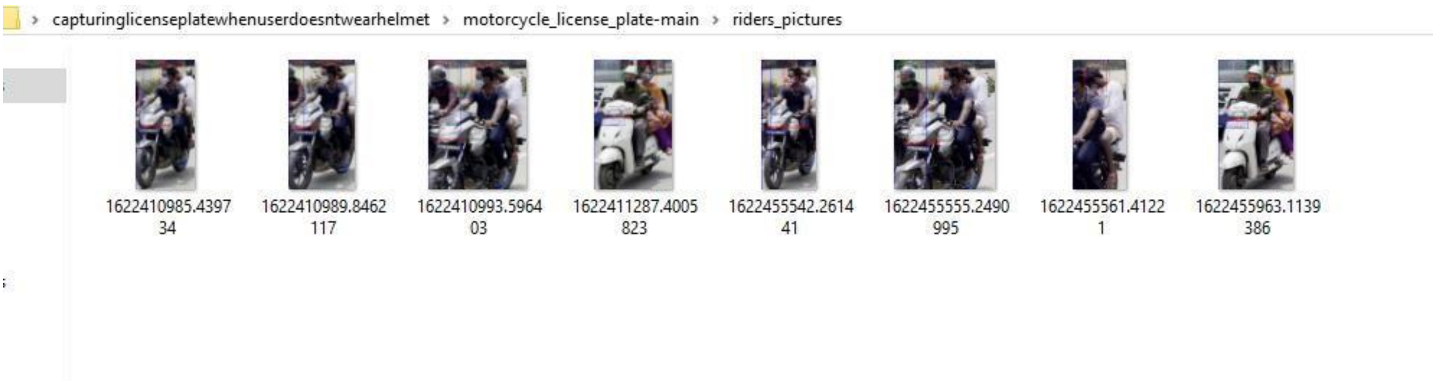
RIDER WITHOUT HELMET IS DETECTED :



RIDERS PICTURE :

RIDER LICENSE PLATE EXTRACTED :

CAPTURED RIDERS PICTURE FOLDER:



CAPTURED LICENSE PLATE FOLDER :

Chapter 10

REFERENCES

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

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

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

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

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

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

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

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

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

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

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

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