



**VIT-AP
UNIVERSITY**

School of Computer Science & Engineering
JULY, 2022

A PROJECT ON

**“TRAFFIC SIGNAL VIOLATION DETECTION
SYSTEM”**

Submitted in partial fulfilment for the award of the degree of

**MASTER OF TECHNOLOGY IN
INTEGRATED SOFTWARE ENGINEERING**

by

GONTLA SRI LIKITHA (19MIS7092)

Under Guidance of

Dr. Anupama Namburu

CERTIFICATE

This is to certify that the Summer Project/Summer Internship work titled "**TRAFFIC SIGNAL VIOLATION DETECTION SYSTEM**" that is being submitted by **GONTLA SRI LIKITHA(19MIS7092)** is in partial fulfillment of the requirements for the award of Master of Technology (Integrated 5 Year) Software Engineering, is a record of bonafide work done under my guidance. The contents of this project work, in full or in parts, have neither been taken from any other source nor have been submitted to any other institute or university for award of any degree or diploma and the same is certified.



Signature

Dr. Anupama Namburu

Guide

The thesis is satisfactory

Approved by



PROGRAM CHAIR

M. Tech. SE



DEAN

School of Computer Science and Engineering

ABSTRACT

This project is to identify and detect the vehicles that violate the traffic signals by giving our own region of interest. So over all this project is mainly about object detection in a video using YOLOv3 algorithm.

Some types of traffic violations are like moving violations and non moving traffic violations. Driving offences which have fatalities are dangerous driving and inconsiderate driving. As the number of vehicles on the roads is increasing rapidly, this successively causes highly congested roads and serving as a reason to interrupt traffic rules by violating them. This results in a high number of road accidents. Traffic violation detection systems using computer vision are an awfully efficient tool to scale back traffic violations by tracking and penalizing.

The recent developments in vision systems have encouraged us to develop advanced computer vision applications suitable to embedded platforms. Within the embedded closed-circuit television, where memory and computing resources are limited, simple and efficient computer vision algorithms are required.

Object detection from a video in video footage which is recorded is the major task. Object detection method is used to identify required objects in videos and to cluster pixels of these objects. The detection of an object in video plays an important role in several applications specifically as video applications.

The proposed system was implemented using YOLOv3 object detection algorithm for traffic violation detections in a video like signal jump, and also detecting the number of vehicles selecting our own region.

Keywords:

Object detection, YOLOv3

ACKNOWLEDGEMENTS

It is our pleasure to express with deep sense of gratitude to **Dr. Anupama Namburu**, Associate Professor, School of Computer Science and Engineering, VIT-AP, for her constant guidance, continual encouragement, and understanding and more than all she taught us patience in our endeavour. Our association with her is not confined to academics only, but it is a great opportunity on our part of work with an intellectual and expert in the field of Computer Science and Engineering.

We would like to express our gratitude to **Dr. G. Viswanathan**, Chancellor, **Dr. Sekar Viswanathan**, VP, **Dr. Sandhya Pentareddy**, Executive Director, **Dr. S V Kota Reddy**, VC, and **Dr. Sudha S V**, Dean, School of Engineering VIT-AP, for providing with an environment to work in and for his inspiration during the tenure of the course.

In jubilant mood, we express ingeniously our whole-hearted thanks to **Dr. Reeja S R**, Program Chair and Associate Professor, all teaching staff and members working as limbs of our university for their not-self-centred enthusiasm coupled with timely encouragements showered on us with zeal, which prompted the acquirement of the requisite knowledge to finalize our course study successfully. We would like to thank our parents for their support.

It is indeed a pleasure to thank our friends who persuaded and encouraged us to take up and complete this task. At last, but not least, we express our gratitude and appreciation to all those who have helped us directly or indirectly toward the successful completion of this project.

Place: AMARAVATHI

Date: 25-07-2022

GONTLA SRI LIKITHA (19MIS7092)

CONTENTS

CHAPTERS.....	6
LIST OF FIGURES	8
LIST OF TABLES	8
LIST OF ABBREVIATIONS	8

CHAPTERS

CHAPTER	NAME	PAGE NO
1	INTRODUCTION	9
1.1	OBJECTIVE	9
1.2	BACKGROUND AND LITERATURE SURVEY	9
1.3	ORGANIZATION OF THE REPORT	10
2	FUNDAMENTALS OF TRAFFIC SIGNAL VIOLATIO DETECTION SYSTEM	12
2.1	IMAGE PROCESSING, OBJECT DETECTION AND IMAGE RECOGNITION	12
2.2	YOLOv3	12
2.3	COMPARISON ON OBJECT DETECTION MODELS	15
3	DESIGN AND WORKING OF DETECTION SYSTEM	18
3.1	PROPOSED SYSTEM	18
3.1.1	VECHILE CLASSIFICATION	18
3.1.2	VIOLATION DETECTION	18
3.2	WORKING METHODOLOGY	18
3.3	STANDARDS	20
3.4	REQUIREMENTS	21
3.5	ER DIAGRAM	22
4	IMPLEMENTATION	23
5	CONCLUSION AND FUTURE WORK	28
5.1	CONCLUSION	28

5.2	FUTURE WORK	28
6	APPENDICES	29
7	REFERENCES	49

LIST OF FIGURES

2.2 YOLOV3 ARCHITECTURE	13
2.3 OBJECT DETECTION MODEL FRAMEWORK	15
2.3 YOLO VS FASTER R-CNN IN ACCURACY AND SPEED.....	17
3.1.1 DARKNET-53 ARCHITECTURE.....	18
3.2 SYSTEM OVERVIEW.....	19
3.2 SYSTEM FUNCTION DIAGRAM	19
3.5 ER DIAGRAM	22

LIST OF TABLES

2.3 COMPARISON OF DIFFERENT MODELS	17
--	----

LIST OF ABBREVIATIONS

CNN	Convolutional Neural Network
R-CNN	Region-based Convolutional Neural Network
YOLO	You Only Look Once
ROI	Region of Interest
SVM	Support Vector Machine

Chapter 1

INTRODUCTION

The increasing number of vehicles in cities can cause high volume of traffic, and implies that traffic violations become more critical nowadays round the world. This causes severe destruction of property and more accidents that will endanger the lives of the people.

To unravel this alarming problem and forestall such unfathomable consequences, traffic violation detection systems are needed. A traffic violation detection system must be realized in real-time because the authorities track the roads all the time.

Hence, traffic enforcers won't only be comfortable in implementing safe roads accurately, but also efficiently, because the traffic detection system detects violations faster than humans.

The system that we developed can detect light violation in a video as a video surveillance application. A user-friendly graphical interface is related to the system to create it simple for the user to control the system, monitor traffic and take action against the violations of traffic rules.

1.1. OBJECTIVE

The goal of the project is to automate the traffic signal violation detection system and make it easy to monitor the traffic and take action against the violated vehicle owner in a fast and efficient way. Detecting and tracking the vehicle and their activities accurately is the main priority of the system.

Computer vision-based detection is based on the principle of vehicle classification, environment awareness, and traffic violation detection. Detection system, the proposed architecture of surveillance system with intelligent detection and tracking of multiple vehicles from the surveillance input video using YOLOv3 as an object detection algorithm.

The main objective is to detect multiple vehicle violation detection and it gives a more detailed picture of concepts and technology involved in creating a traffic violation detection system using computer vision.

1.2. BACKGROUND AND LITERATURE SURVEY

Manually checking the violation made by vehicles is difficult in real life and troublesome and there is chance of mistakes made due to feeble and problematic human memory. Earlier the traffic violation detection was done by human traffic cops at signal junctions, which highly need human resources and presence at all times. This eventually became more difficult as there is highly increase in vehicles.

This made to trigger the need for detection systems which are specialized systems made for detecting types of violations. A new system was needed which can be operated all the time without any interpretation. So, there is high need for traffic violation detection systems now a days.

We need traffic violation detection system in order to detect violations made by people while driving vehicles and for identification of criminal offenses like signal jump, over speed driving.

A system that works with least or no human resource requirement and which can identify multiple violations with high accurate result is required. So, this is how traffic violation detection systems using computer vision came into highlight.

Traffic violation detection with computer vision is based on computer vision technologies such as image processing, artificial intelligence, machine learning and deep learning. This detects objects in digital images or videos with instances of semantic objects and class. This makes easy for traffic police to capture the traffic violations made by people even in small lanes to big highways.

It also saves a lot of time and makes work easier for traffic police personnel to mainly focus on other work. The computer vision-based object detection is mainly based on the principle of vehicle classification, environment awareness and traffic violation detection.

So, a detection system is proposed with architecture of surveillance system with intelligent detection and tracking of vehicles from given video using YOLOv3 as an object detection algorithm.

This is made through a neural network and an object detection model in classification of the moving objects into different respective classes.

1.3. ORGANIZATION OF THE REPORT

The remaining chapters of the project are described as follows:

- Chapter 2 discusses about the fundamentals of the detection system like image processing, YOLOv3.
- Chapter 3 is for the projects proposed system, working methodology, software and hardware requirements.
- Chapter 4 discusses about the results obtained after the project was successfully implemented.
- Chapter 5 concludes the project report with conclusion and future work.
- Chapter 6 consists of codes in the project.
- Chapter 7 gives the references.

Chapter 2

FUNDAMENTALS OF TRAFFIC SIGNAL VIOLATION DETECTION SYSTEM

This Chapter describes about the fundamentals of the detection system like image processing, YOLOv3 algorithm.

2.1. IMAGE PROCESSING, OBJECT DETECTION AND IMAGE RECOGNITION

Image Processing: It is a method to transform a image into an enhanced image or to get some useful information from it by doing some operations on it. It is a process to transform the image into a digital format. The algorithms help to analyze images to get deep insights in computer vision use cases.

Object Detection: It is a computer vision method for identifying and locating instances of objects in images or videos. It can also be used to find number of objects in a particular scene and then determine and track their location precisely and accurately by labelling them.

The main difference between **image recognition** and object detection is that image recognition only assigns a label to an image whereas object detection draws a box around the object and also labels it. Object detection also predicts where the object is located and what label should be applied to it.

2.2. YOLOv3

It is version 3 of YOLO which stands for ‘You only look once’. It is a method which considers object detection as a regression problem. It is a object detection algorithm in real-time that identifies the specific objects in images, videos and live feeds. It is an algorithm that uses the features of deep convolutional neural network to detect the objects which is implemented using the Keras or OpenCV libraries of deep learning.

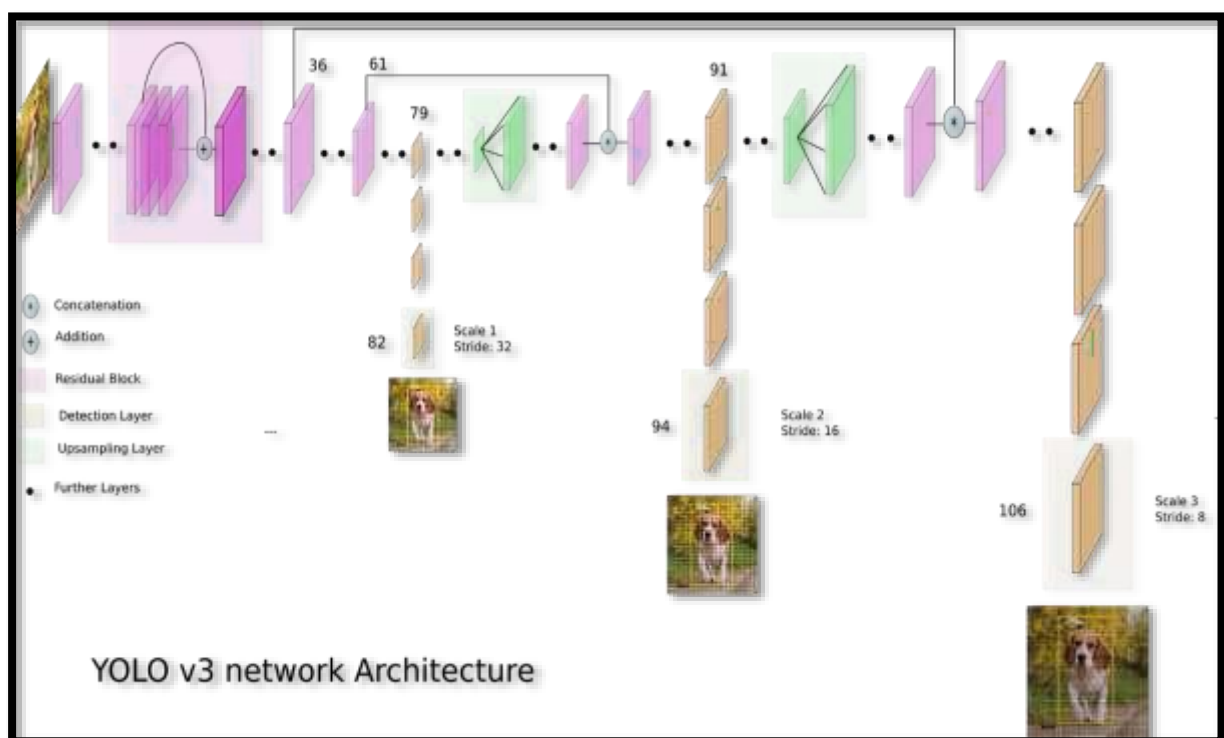
It predicts class probabilities as well as bounding box offsets directly from full images with a single feed forward convolution neural network. It eliminates region proposal generation and feature re-sampling, and encapsulates all stages in a single network.

YOLOv1 uses Darknet framework which is trained on ImageNet-1000 dataset. YOLOv2 uses Darknet-19 framework which contains 19 convolutional layers. But YOLO version 3 consists of 53 convolutional layers that are also called Darknet-53.

There are major differences between YOLOv3 and older versions occur in terms of speed, precision, accuracy, architecture and specificity of classes.

YOLOv3 can give the multi label classification since it uses a logistic classifier for each class in place of the SoftMax layer which is used in YOLOv2. As mentioned earlier YOLOv3 uses Darknet-53 framework which has fifty-three layers of convolution. These layers are more in detail and depth compared to Darknet-19 which used in YOLOv2. Darknet-53 mainly contains 3x3 and 1x1 filters along with by-pass links. The YOLOv3 is accurate version than YOLOv1, YOLOv2.

The architecture of YOLOv3 is shown here:



YOLO runs a lot more faster than RCNN, Fast RCNN and Faster RCNN version algorithms due to its simple architecture. Because unlike RCNN algorithms it is trained to do the classification as well as bounding box regression at a same time.

YOLO uses a single neural network to the entire image. This helps to divide the image into regions and gives probabilities for all regions. After that it predicts the number of

bounding boxes which covers the regions on the image and chooses the best ones according to the probabilities.

Features of YOLOv3:

1. Bounding box predictions:

It is a single network that predicts the objective score using the logistic regression where one means that complete overlap of bounding box is prior over the object. It will predict only one main bounding box for an object. There will be other bounding boxes prior to it with objectiveness score much more than threshold but lesser than the best bounding box.

2. Class prediction:

It uses independent logistic classifier method for each class instead of a normal regular softmax layer. This is made to classify the multi-label classification. Each bounding box predicts the classes in the image using the multi-label classification.

3. Feature extractor:

YOLO version 3, originally, consists of 53 convolutional layers that are also called Darknet-53. But for detection tasks the original architecture is added with 53 more layers which gives us 106 layers of architecture for YOLO version 3. So you will see the process of loading architecture that has 106 layers. The detections for objects are mainly made at three layers 82, 94 and 106.

4. Predictions across scales:

YOLOv3 predicts at three different scales which precisely given by down sampling all the dimensions of the given image by 32, 16 and 18 respectively. First detection is made at 82nd layer as it will down sampled by the network till 81 layers such that the 81st layer have stride of 32. Then other detections are made at 94 and 106 layers as shown in the architecture.

5. Better at detecting smaller objects:

The detection at different layers helps to identify the issues of detection of small objects which is a major issue in YOLOv2. Usually the 13*13 layer is for detecting the large objects in images or videos, similarly the 52*52 layer is for detecting smaller objects and as well as 26*26 layer is for detecting the medium objects.

2.3. COMPARISON ON OBJECT DETECTION MODELS

There are various types of object detection algorithms available today. And there are mainly two types of region base CNN models: 1. Two-stage models, 2. Single-stage models. Further the two-stage models are divided into R-CNN, Fast R-CNN and Faster R-CNN while single-stage models are divided into You Only Look Once version 1 (YOLOv1), YOLOv2 and YOLOv3. These models are based on convolutional neural networks (CNN) which act as backbone of object detection techniques. Object detection system assumes bounding frames, one for every object detected.

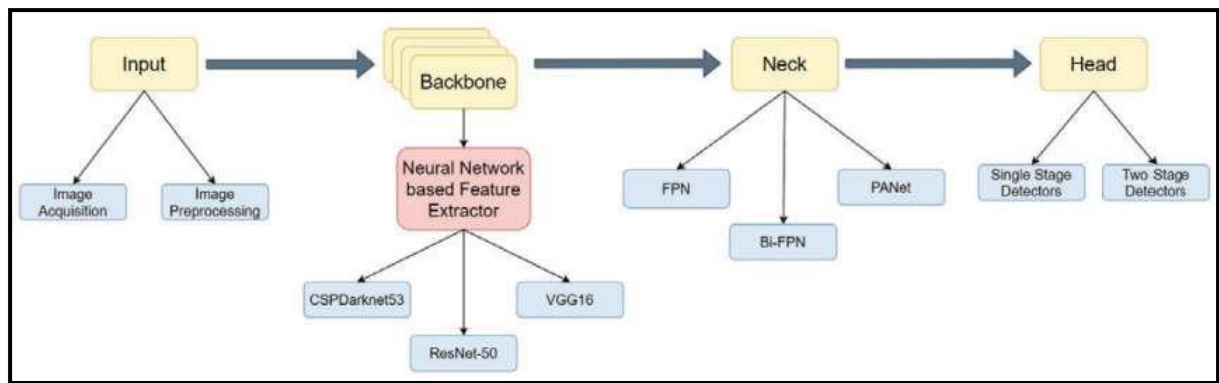


Fig: Object Detection Model Framework

Two-stage models: They classify regions into various parts which have the object in them. Then second stage is detection of false decisions that refine the boundary boxes. In these detectors image is divided based on region of interest and have a high accuracy for localization and recognition of object. These models are achieve very high precision but they are slow to respond or make progress when compared with single stage detection models.

- **R-CNN:** In R-CNN each region is first categorized the labelled and then is provided with a boundary box. Now a pretrained R-CNN is used previous to output layer. Support vector machine (SVM) which is used for classification of object is trained using features that are extracted from the proposed regions.
 - Drawbacks of this model are: it is not flexible since it is based on selective search algorithm, due to its slow speed it can't be used in real time processing and the training need a lot of time since various stages are involved in it. This method is not used in practical applications as computation time is 40 to 50 seconds.

- **Fast R-CNN:** In this model the whole image is taken as input. Several CNN layers are used for processing of image and get convolutional feature map as the output. It has less computational time when compared with R-CNN and is improved in terms of accuracy of detection of object. For the classification it used softmax and not SVM as in R-CNN. ROI makes this method faster.
 - Drawbacks of this model are: it is complicated, calculation time is high and computation time is 2 seconds.
- **Faster R-CNN:** The selective search which is a disadvantage in fast R-CNN is discarded and region proposal network is used for decreasing the number of proposed regions. The scores of boundary boxes help to know the degree of similarity of detected objects such that it is both highly accurate and effective in cost.
 - Drawbacks of this model are: Since there are many operations which are aligned in a sequence from in a pipeline for other previous operations it becomes high time consuming for detection of various objects.

Single-stage models: They require only a single pass through the CNN layer and predicts all the bounding boxes at a once and have a feed forward type of neural networks. They are very much faster since there they have less computational cost and high performance.

- **YOLOv1:** The object detection is performed as a regression problem. A single neural network predicts the bounding box and class probabilities from the entire image in just one evaluation. Here both the bounding boxes and the class probabilities of those boxes are predicted simultaneously. It has 24 CNN with two fully connected layers.
 - Drawbacks of this model are: It cannot determine objects which are of low resolution in the image, cannot predict more than one box for a region and there is a chance of missing out some objects in a image and is less flexible.
- **YOLOv2:** It is based on Darknet-19 framework which has nineteen convolutional layers and five max pooling layers. It predicts detections for the object classes with training dataset which is a combination of COCO dataset and ImageNet dataset. It traverses tree until it reaches a certain threshold and move the path with high level of confidence score.
- **YOLOv3:** YOLOv3 is a model which calculates the objectness value for of the bounding box via logistic regression. It has a better predictor and feature extractor compared to other models. Logistic regression is used in place of softmax function which finds only one class for a single object at a time. The multilabel approach here

gives three predictions for each location which gives a better accuracy of prediction. It gives decent frames per second and takes low computational time, so it is most optimal object detection technique and most used one at present.

- ❖ The following graph shows the accuracy and speed comparison of YOLOv3 and faster R-CNN

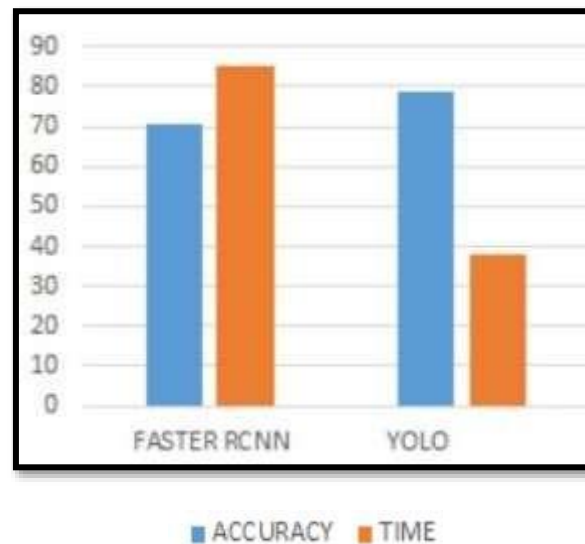


Fig: Accuracy and Speed comparison of YOLOv3 with faster R-CNN

- ❖ The table below will give comparisons between different object detection models in terms of mAP, FPS, time and Backbone.

Models	mAP-50	FPS	Time(s)	Backbone
R-CNN	-	0.03	40-45	Pascal VOC 2007
Fast R-CNN	39.9	0.5	2	VGG-16
Faster R-CNN	42.7	7	1.1	VGG-16
Mask R-CNN	36.1	8.6	1.16	ResNet-101 FPN
YOLOv1	37.4	45	0.02	Pascal VOC 2007
SSD300	41.2	46	0.61	ResNet-101-SSD
SSD512	46.5	19	0.125	ResNet-101-SSD
YOLOv2 288x288	44.0	91	0.02	Darknet-19
YOLOv2 480x480	46.9	59	-	Darknet-19
YOLOv2 544x544	47.9	40	0.03	Darknet-19
YOLOv3 320x320	51.5	45	0.22	Darknet-53
YOLOv3 416x416	55.3	35	0.29	Darknet-53
YOLOv3 608x608	57.9	20	0.51	Darknet-53

Table: Comparison of different models

Chapter 3

DESIGN AND WORKING OF DETECTION SYSTEM

This Chapter describes the proposed system of the detection system, the working methodology, and software and hardware requirements.

3.1. PROPOSED SYSTEM

The system is mainly divided into two parts:

1. Vehicle classification
2. Violation detection

3.1.1. VEHICLE CLASSIFICATION

From the given video footage, the vehicles which violate the traffic signals are detected and images of those vehicles are classified. The object detection algorithm - YOLOv3 is the third version in YOLO (You Only Look Once) algorithms is used for this vehicle classification. It is mainly used to improve the accuracy and is more capable for detecting the objects that violate the system. Some of the features of algorithm that made it unique are: Bounding box predictions, Class prediction, Predictions across scales, Feature extractor.

The classifier model that we built is with Darknet-53 architecture which is as shown below.

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

Fig: Darknet-53 architecture

The vehicles which violate the traffic signal i.e. the region of interest that we give in the input video data are detected using YOLOv3 algorithm. After the image classification and detection of the vehicles the violation cases are checked.

A traffic line i.e region of interest is drawn on the road in the input video data by the user. This line tells that the traffic light is red and the driver need to stop. The violation happens if the vehicle crosses the region of interest in red state.

The identified vehicle objects are shown with a green bounding box. If any object passes the region of interest in red state, it is shown in bounding box with a different colour since the violation happened.

3.2. WORKING METHODOLOGY

The following diagram (figure) shows the system overview of this project:

The system mainly contains two components: Vehicle detection model and a graphical user interface i.e. GUI.

At first user selects the video footage of the vehicles and it is sent to the system. Then vehicles are detected from the given input video footage. The user now will draw the line which is region of interest. After tracking the activity of vehicles, system determines whether any violation

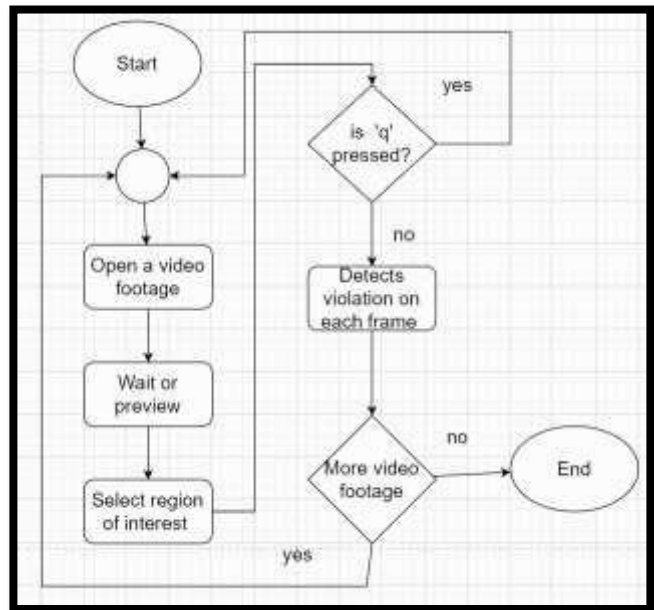


Fig: System overview

is happened or not. If system detects any objects that violate the signal then it will capture that vehicle and store it in the database folder.

The GUI makes the system more interactive and helps the user to use it easily. User can view the traffic footage and can get the output with an alert of violation as a bounding box around the violated vehicle. User can also take further action using the GUI.

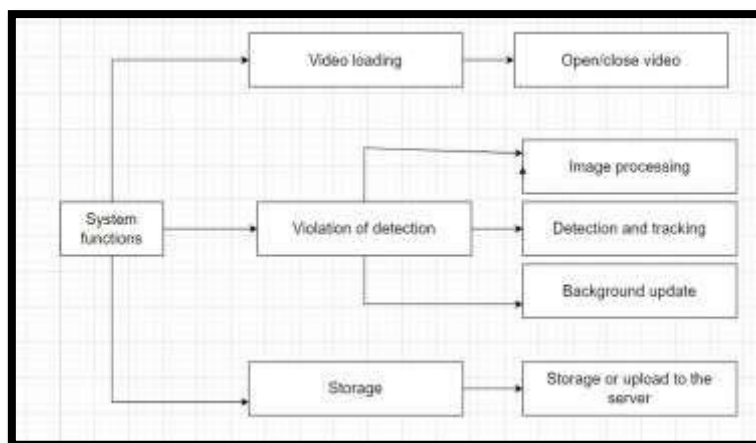


Fig: System Function Diagram

System can load the video footage then open or close video. The violation detection is divided as image pre-processing, detection and tracking, background update in the database.

3.3. STANDARDS

Various standards used in this project are as follows:

- **TensorFlow:** It is an end-to-end open-source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML powered applications.
- **Numpy:** It is a library in python programming language, adding support for large, multi-dimensional array and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.
- **Pandas:** Pandas allows us to analyze big data and make conclusions based on statistical theories. Pandas can clean messy data sets, and make them readable and relevant. Relevant data is very important in data science. Pandas are also able to delete rows that are not relevant, or contains wrong values, like empty or NULL values. This is called cleaning the data.
- **Keras:** Keras is a minimalist Python library for deep learning that can run on top of Theano or TensorFlow and also powerful and easy-to-use free open-source Python library for developing and evaluating deep learning models.
It is part of the TensorFlow library and allows you to define and train neural network models in just a few lines of code.
- **Struct:** The module struct is used to convert the native data types of python into string of bytes and vice versa. We don't have to install it. It's a built-in module available in Python3.
- **OpenCV:** OpenCV (Open-Source Computer Vision Library) is an open-source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products.
- **Tkinter:** Tkinter is the standard GUI library for Python. Python when combined with Tkinter provides a fast and easy way to create GUI applications. Tkinter provides a powerful object-oriented interface to the Tk GUI toolkit.
- **Imageio:** Imageio is a Python library that provides an easy interface to read and write a wide range of image data, including animated images, volumetric data, and scientific formats.

3.4. REQUIREMENTS

The software requirements for this project are as follows:

- **Operating system** - Windows 7 is used as the operating system as it is stable and supports more features and is more user friendly
- **Development tools & Programming language** - PYTHON is used to write the whole code and visual studio code has been used for development

Software	Minimum System Requirement
Operating System	Windows 7 (or) Later
Runtime Server	Visual Studio Code

The hardware requirements for this project are as follows:

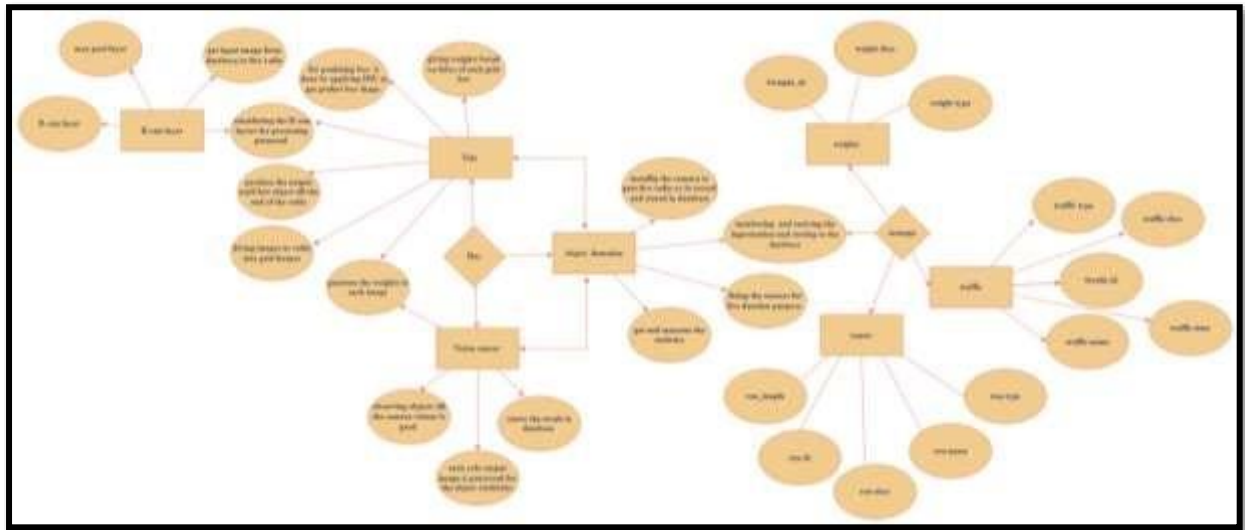
- **INTELCOREI32NDGENERATION** is the least that can be used as a processor because it is fast than other processors can provide reliable and stable and we can run our pc for long time. By using this processor, we can keep on developing our project without any worries.
- **RAM 1 GB** is used as it will provide fast reading and writing capabilities and will in turn support in processing.

Hardware	Minimum System Requirement
Processor	Intel i3
Memory	128 MB RAM
Disk	10 GIGA BYTE

The interface requirements for this project are as follows:

- Easy to navigate& Less graphics
- Display error messages and relevant dialogue boxes
- Provide high security such that no to be modified by irrelevant users
- It must provide all options to all users.

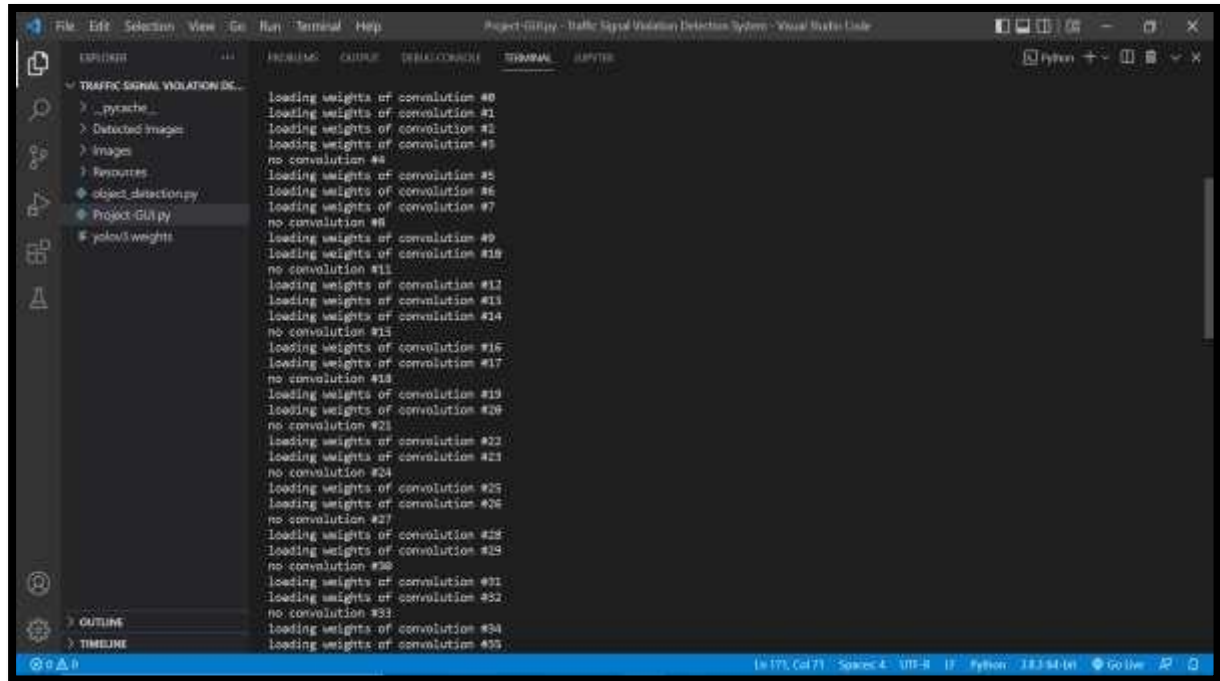
3.5. ER DIAGRAM



Chapter 4

IMPLEMENTATION

Loading weights of convolution layers:

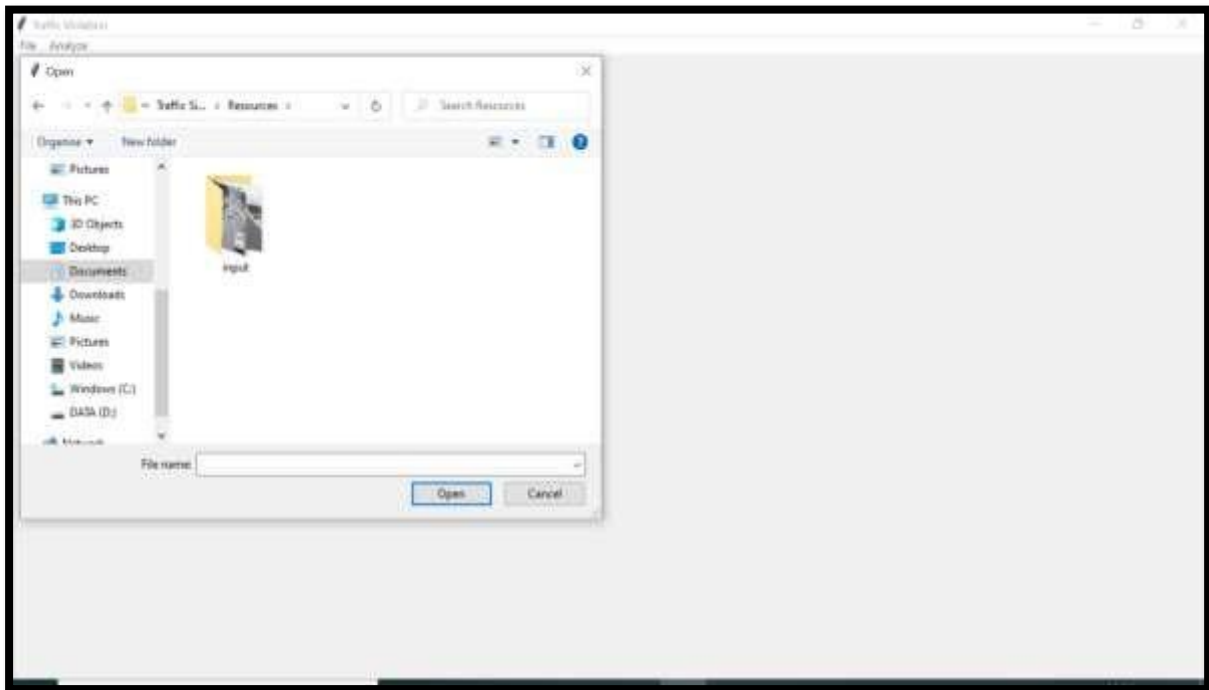


```
loading weights of convolution #0
loading weights of convolution #1
loading weights of convolution #2
loading weights of convolution #3
no convolution #4
loading weights of convolution #5
loading weights of convolution #6
loading weights of convolution #7
no convolution #8
loading weights of convolution #9
loading weights of convolution #10
no convolution #11
loading weights of convolution #12
loading weights of convolution #13
loading weights of convolution #14
no convolution #15
loading weights of convolution #16
loading weights of convolution #17
no convolution #18
loading weights of convolution #19
loading weights of convolution #20
no convolution #21
loading weights of convolution #22
loading weights of convolution #23
no convolution #24
loading weights of convolution #25
loading weights of convolution #26
no convolution #27
loading weights of convolution #28
loading weights of convolution #29
no convolution #30
loading weights of convolution #31
loading weights of convolution #32
no convolution #33
loading weights of convolution #34
loading weights of convolution #35
```

This is the GUI home page with have ‘File’ and ‘Analyse’ options. Further the file have ‘Open’ and ‘Exit’ options and analyse have ‘Region of interest’ option.



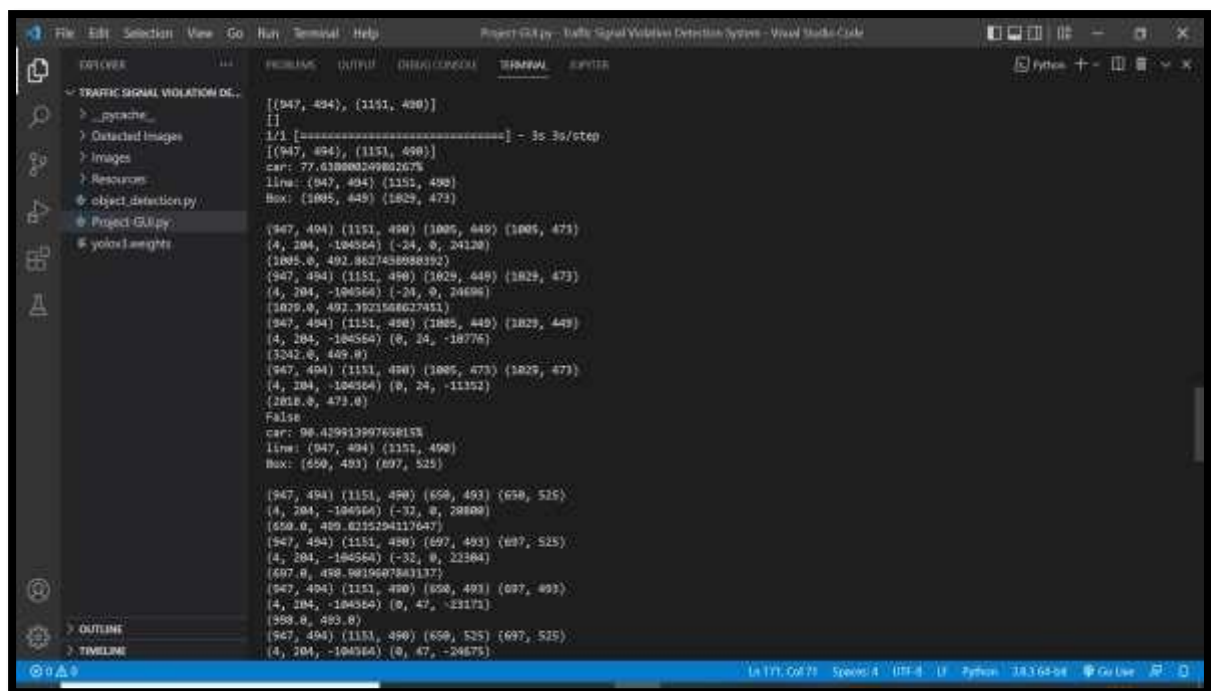
Selecting a video footage from folder:



Drawing region of interest after selecting video:



Execution after drawing region of interest:



```
[[947, 494], [1151, 490]]
1/1 [-----] - 3s 3s/step
[[947, 494], [1151, 490]]
car: 77.61888324980267%
line: (947, 494) (1151, 490)
Box: (1895, 649) (1829, 473)

(947, 494) (1151, 490) (1895, 649) (1829, 473)
(4, 204, -104364) (-24, 0, 24120)
(1895, 0, 492.8627438888392)
(947, 494) (1151, 490) (1829, 449) (1829, 473)
(4, 204, -104364) (-24, 0, 24696)
(1829, 0, 492.3921588627451)
(947, 494) (1151, 490) (1895, 449) (1829, 449)
(4, 204, -104364) (0, 24, -18776)
(1829, 0, 449, 0)
(947, 494) (1151, 490) (1895, 473) (1829, 473)
(4, 204, -104364) (0, 24, -11352)
(1829, 0, 473, 0)
False
car: 98.42991399765815%
line: (947, 494) (1151, 490)
Box: (650, 493) (697, 525)

(947, 494) (1151, 490) (650, 493) (697, 525)
(4, 204, -104364) (-32, 0, 28800)
(650, 0, 492.6215294117647)
(947, 494) (1151, 490) (697, 493) (697, 525)
(4, 204, -104364) (-32, 0, 22384)
(697, 0, 492.9810407843137)
(947, 494) (1151, 490) (650, 493) (697, 493)
(4, 204, -104364) (0, 47, -23173)
(697, 0, 493, 0)
(947, 494) (1151, 490) (650, 525) (697, 525)
(4, 204, -104364) (0, 47, -24675)
```

All the vehicles are detected, labelled and localized. The blue colour line here is region of interest.



Detection of vehicles which violates the rule by crossing region of interest is shown in with blue boundary box. Detection of vehicle crossing during a red light in the image and locate the exact location. It generated the regions where there might be an object based on the input digital image. Second, predicts the class of the object, the bounding box. The blue line indicates it the signal line where the vehicle must stop during the red light.



After successfully executing entire video the region of interest turns into green from blue and now we can exit from the system.



The below figure shows the generated data concerning the region of interest and the boundary boxes and after executing successfully it will display as below:

The screenshot shows the Project GUI in a Jupyter Notebook environment. The interface includes a file explorer on the left, a main output area, and a status bar at the bottom.

File Explorer (Left Panel):

- TRAFFIC SIGNAL VIOLATION DE...
 - pycache_
 - Detected images
 - Images
 - Resources
 - object_detection.py
 - Project GUI.py
 - yolo3.weights

Main Output Area:

```

line: [955, 478] (1132, 471)
Box: (1019, 469) (1015, 508)

[955, 478] (1132, 471) (1019, 469) (1015, 508)
(7, 177, -91291) (-38, 0, 35741)
[1019, 0, 475, 469] (1015, 508) (1019, 469)
[955, 478] (1132, 471) (1015, 469) (1015, 508)
(7, 177, -91291) (-38, 0, 41145)
[1015, 0, 474, 465] (1015, 508) (1015, 508)
[955, 478] (1132, 471) (1019, 469) (1015, 469)
(7, 177, -91291) (0, 36, -16884)
[1132, 471, 4285734287, 469, 0]
[955, 478] (1132, 471) (1019, 508) (1015, 508)
(7, 177, -91291) (0, 36, -16288)
[1015, 4285734287, 1142, 508, 0]
True
car: 79.884489786479105
line: [955, 478] (1132, 471)
Box: (1019, 469) (1134, 527)

[955, 478] (1132, 471) (1019, 469) (1015, 508)
(7, 177, -91291) (-38, 0, 35528)
[1019, 0, 472, 465] (1015, 508) (1019, 469)
[955, 478] (1132, 471) (1134, 469) (1134, 527)
(7, 177, -91291) (-38, 0, 40024)
[1134, 0, 470, 469] (1015, 508) (1134, 469)
[955, 478] (1132, 471) (1019, 469) (1134, 469)
(7, 177, -91291) (0, 36, -17676)
[1015, 4285734287, 1142, 491, 0]
[955, 478] (1132, 471) (1019, 508) (1134, 527)
(7, 177, -91291) (0, 36, -18972)
[-184, 0, 527, 0]
False
31
Executed Successfully!!!
PS C:\Users\arave\Documents\Traffic Signal Violation Detection System>

```

Status Bar (Bottom): PS C:\Users\arave\Documents\Traffic Signal Violation Detection System

Chapter 5

CONCLUSION AND FUTURE WORK

5.1. CONCLUSION

The detection of vehicles which violates traffic signals in a video footage is a challenging task as the vehicles that travel on road and traffic rules for them are depended on the different regions of the roads and their timings. In this project we proposed that YOLO algorithm with version 3 is much suitable for traffic signal violation detection.

The final output shown that the detection of many traffic violations from a single input source is also possible. The detection time is much lower for the high denser traffic flow. Thus, the detection system operation speed depends on the density of the traffic.

5.2. FUTURE WORK

The system runtime is a little slow, and can be improved using a computer with much higher speed processor specification. Future research can be done for the system by other advanced image processing techniques. Further improvements are needed to decrease computational time of the detection system at high the traffic volume roads. This project can be developed in future to implement a number plate detection of detected vehicles in the video footage. The region of interest is a straight line in this project but it can be changed to other different shapes based on different scenarios for example, for a restricted area a person can enter from any of the sites in such cases a straight line can't be used. This system can be linked to a hosted database or server which can automatically detect it in a video without any manual work and we can develop a web application instead of GUI.

Chapter 6

APPENDICES

CODE:

ObjectDetection.py:

```
import numpy as np

from keras.layers import Conv2D, Input, BatchNormalization, LeakyReLU, ZeroPadding2D,
UpSampling2D

from keras.layers 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))
                kernel = kernel.reshape(list(reversed(conv_layer.get_weights()[0].shape)))
                kernel = kernel.transpose([2,3,1,0])
                conv_layer.set_weights([kernel])
        except ValueError:
            print("no convolution #" + str(i))

def reset(self):

```

```

self.offset = 0

class BoundBox:
    def __init__(self, xmin, ymin, xmax, ymax, objness = None, classes = None):
        self.xmin = xmin
        self.ymin = ymin
        self.xmax = xmax
        self.ymax = ymax

        self.objness = objness
        self.classes = classes
self.label = -1
        self.score = -1

    def get_label(self):
        if self.label == -1:
            self.label = np.argmax(self.classes)
        return self.label

    def get_score(self):
        if self.score == -1:
            self.score = self.classes[self.get_label()]
        return self.score

def _conv_block(inp, convs, skip=True):
    x = inp
    count = 0
    for conv in convs:
        if count == (len(convs) - 2) and skip:
            skip_connection = x

```

```

        count += 1

        if conv['stride'] > 1: x = ZeroPadding2D(((1,0),(1,0)))(x) # peculiar padding as darknet
        prefer left and top

        x = Conv2D(conv['filter'],
                    conv['kernel'],
                    strides=conv['stride'],
                    padding='valid' if conv['stride'] > 1 else 'same', # peculiar padding as darknet
        prefer left and top
                    name='conv_' + str(conv['layer_idx']),
                    use_bias=False if conv['bnorm'] else True)(x)

        if conv['bnorm']: x = BatchNormalization(epsilon=0.001, name='bnorm_' +
        str(conv['layer_idx'))(x)

        if conv['leaky']: x = LeakyReLU(alpha=0.1, name='leaky_' + str(conv['layer_idx']))(x)

        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),
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]
    nb_box = 3
    netout = netout.reshape((grid_h, grid_w, nb_box, -1))
    nb_class = netout.shape[-1] - 5
    boxes = []
    netout[..., :2] = _sigmoid(netout[..., :2])
    netout[..., 4:] = _sigmoid(netout[..., 4:])
    netout[..., 5:] = netout[..., 4][..., np.newaxis] * netout[..., 5:]
    netout[..., 5:] *= netout[..., 5:] > obj_thresh
    for i in range(grid_h*grid_w):
        row = i / grid_w
        col = i % grid_w
        for b in range(nb_box):
            # 4th element is objectness score
            objectness = netout[int(row)][int(col)][b][4]
            #objectness = netout[..., :4]
            if(objectness.all() <= obj_thresh): continue
            # first 4 elements are x, y, w, and h
            x, y, w, h = netout[int(row)][int(col)][b][:4]

```

```

    x = (col + x) / grid_w # center position, unit: image width
    y = (row + y) / grid_h # center position, unit: image height
    w = anchors[2 * b + 0] * np.exp(w) / net_w # unit: image width
    h = anchors[2 * b + 1] * np.exp(h) / net_h # unit: image height
    # last elements are class probabilities
    classes = netout[int(row)][col][b][5:]
    box = BoundBox(x-w/2, y-h/2, x+w/2, y+h/2, objectness, classes)
    #box = BoundBox(x-w/2, y-h/2, x+w/2, y+h/2, None, classes)
    boxes.append(box)

return boxes

def correct_yolo_boxes(boxes, image_h, image_w, net_h, net_w):
    if (float(net_w)/image_w) < (float(net_h)/image_h):
        new_w = net_w
        new_h = (image_h*net_w)/image_w
    else:
        new_h = net_h
        new_w = (image_w*net_h)/image_h
    for i in range(len(boxes)):
        x_offset, x_scale = (net_w - new_w)/2./net_w, float(new_w)/net_w
        y_offset, y_scale = (net_h - new_h)/2./net_h, float(new_h)/net_h
        boxes[i].xmin = int((boxes[i].xmin - x_offset) / x_scale * image_w)
        boxes[i].xmax = int((boxes[i].xmax - x_offset) / x_scale * image_w)
        boxes[i].ymin = int((boxes[i].ymin - y_offset) / y_scale * image_h)
        boxes[i].ymax = int((boxes[i].ymax - y_offset) / y_scale * image_h)

def do_nms(boxes, nms_thresh):
    if len(boxes) > 0:
        nb_class = len(boxes[0].classes)
    else:

```

```

return

for c in range(nb_class):
    sorted_indices = np.argsort([-box.classes[c] for box in boxes])
    for i in range(len(sorted_indices)):
        index_i = sorted_indices[i]
        if boxes[index_i].classes[c] == 0: continue
        for j in range(i+1, len(sorted_indices)):
            index_j = sorted_indices[j]
            if bbox_iou(boxes[index_i], boxes[index_j]) >= nms_thresh:
                boxes[index_j].classes[c] = 0

def draw_boxes(image, boxes, line, labels, obj_thresh, dcnt):
    print(line)    for box in boxes:
        label_str = "
        label = -1
        for i in range(len(labels)):
            if box.classes[i] > obj_thresh:
                label_str += labels[i]
                label = i
            print(labels[i] + ': ' + str(box.classes[i]*100) + '%')
            print('line: (' + str(line[0][0]) + ', ' + str(line[0][1]) + ') (' + str(line[1][0]) + ', ' +
str(line[1][1]) + ')')
            print('Box: (' + str(box.xmin) + ', ' + str(box.ymin) + ') (' + str(box.xmax) + ', ' +
str(box.ymax) + ')')
            print()
        if label >= 0:
            tf = False
            (rxmin, rymin) = (box.xmin, box.ymin)
            (rxmax, rymax) = (box.xmax, box.ymax)
            tf = False

```

```

tf |= intersection(line[0], line[1], (rxmin, rymin), (rxmin, rymax))
tf |= intersection(line[0], line[1], (rxmax, rymin), (rxmax, rymax))
tf |= intersection(line[0], line[1], (rxmin, rymin), (rxmax, rymin))
tf |= intersection(line[0], line[1], (rxmin, rymax), (rxmax, rymax))
print(tf)
cv2.line(image, line[0], line[1], (255, 0, 0), 3)
if tf:
    cv2.rectangle(image, (box.xmin,box.ymin), (box.xmax,box.ymax), (255,0,0), 3)
    cimg = image[box.ymin:box.ymax, box.xmin:box.xmax]
    cv2.imshow("violation", cimg)
    cv2.waitKey(5)
    cv2.imwrite("Detected Images/violation_"+str(dcnt)+".jpg", cimg)
    dcnt = dcnt+1
else:
    cv2.rectangle(image, (box.xmin,box.ymin), (box.xmax,box.ymax), (0,255,0), 3)
cv2.putText(image,
            label_str + ' ' + str(round(box.get_score(), 2)),
            (box.xmin, box.ymin - 13),
            cv2.FONT_HERSHEY_SIMPLEX,
            1e-3 * image.shape[0],
            (0,255,0), 2)

return image
weights_path = "yolov3.weights"
# set some parameters
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

```

Project-GUI.py:

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

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

        self.master = master

        self.pos = []
        self.line = []
        self.rect = []

        self.master.title("GUI")

        self.pack(fill=BOTH, expand=1)

        self.counter = 0

    menu = Menu(self.master)

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

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

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

    self.filename = "Images/home1.jpg"

    self.imgSize = Image.open(self.filename)

    self.tkimage = ImageTk.PhotoImage(self.imgSize)
```

```

self.w, self.h = (1366, 768)

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

def open_file(self):
    self.filename = filedialog.askopenfilename()
    cap = cv2.VideoCapture(self.filename)
    reader = imageio.get_reader(self.filename)
    fps = reader.get_meta_data()['fps']
    ret, image = cap.read()
    cv2.imwrite('Images/preview.jpg', image)
    self.show_image('Images/preview.jpg')

def show_image(self, frame):
    self.imgSize = Image.open(frame)
    self.tkimage = ImageTk.PhotoImage(self.imgSize)
    self.w, self.h = (1366, 768)
    self.canvas.destroy()
    self.canvas = Canvas(master = root, width = self.w, height = self.h)
    self.canvas.create_image(0, 0, image=self.tkimage, anchor='nw')
    self.canvas.pack()

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

def client_exit(self):
    exit()

```

```

def imgClick(self, event):
    if self.counter < 2:
        x = int(self.canvas.canvasx(event.x))
        y = int(self.canvas.canvasy(event.y))
        self.line.append((x, y))
        self.pos.append(self.canvas.create_line(x - 5, y, x + 5, y, fill="red", tags="crosshair"))
        self.pos.append(self.canvas.create_line(x, y - 5, x, y + 5, fill="red", tags="crosshair"))
        self.counter += 1
    if self.counter == 2:
        #unbinding action with mouse-click
        self.canvas.unbind("<Button-1>")
        root.config(cursor="arrow")
        self.counter = 0
        #show created virtual line
        print(self.line)
        print(self.rect)
        img = cv2.imread('Images/preview.jpg')
        cv2.line(img, self.line[0], self.line[1], (0, 255, 0), 3)
        cv2.imwrite('Images/copy.jpg', img)
        self.show_image('Images/copy.jpg')
        #image processing
        self.main_process()
        print("Executed Successfully!!!")
        #clearing things
        self.line.clear()
        self.rect.clear()
        for i in self.pos:
            self.canvas.delete(i)

```

```

def intersection(self, p, q, r, t):
    print(p, q, r, t)
    (x1, y1) = p
    (x2, y2) = q
    (x3, y3) = r
    (x4, y4) = t
    a1 = y1-y2
    b1 = x2-x1
    c1 = x1*y2-x2*y1
    a2 = y3-y4
    b2 = x4-x3
    c2 = x3*y4-x4*y3
    if(a1*b2-a2*b1 == 0):
        return False
    print((a1, b1, c1), (a2, b2, c2))
    x = (b1*c2 - b2*c1) / (a1*b2 - a2*b1)
    y = (a2*c1 - a1*c2) / (a1*b2 - a2*b1)
    print((x, y))
    if x1 > x2:
        tmp = x1
        x1 = x2
        x2 = tmp
    if y1 > y2:
        tmp = y1
        y1 = y2
        y2 = tmp
    if x3 > x4:
        tmp = x3
        x3 = x4
        x4 = tmp

```

```

if y3 > y4:
    tmp = y3
    y3 = y4
    y4 = tmp

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


def main_process(self):
    video_src = self.filename
    cap = cv2.VideoCapture(video_src)
    reader = imageio.get_reader(video_src)
    fps = reader.get_meta_data()['fps']
    writer = imageio.get_writer('Resources/output/output.mp4', fps = fps)
    j = 1
    while True:
        ret, image = cap.read()
        if (type(image) == type(None)):
            writer.close()
            break
        image_h, image_w, _ = image.shape
        new_image = od.preprocess_input(image, od.net_h, od.net_w)
        # run the prediction
        yolos = od.yolov3.predict(new_image)
        boxes = []
        for i in range(len(yolos)):
            # decode the output of the network
            boxes += od.decode_netout(yolos[i][0], od.anchors[i], od.obj_thresh,
od.nms_thresh, od.net_h, od.net_w)

```

```

        # correct the sizes of the bounding boxes

        od.correct_yolo_boxes(boxes, image_h, image_w, od.net_h, od.net_w)                #
suppress non-maximal boxes

        od.do_nms(boxes, od.nms_thresh)

        # draw bounding boxes on the image using labels

        image2 = od.draw_boxes(image, boxes, self.line, od.labels, od.obj_thresh, j)

        writer.append_data(image2)

        cv2.imshow('Traffic Violation', image2)

        print(j)

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

            writer.close()

            break

        j = j+1

    cv2.destroyAllWindows()

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

```


Chapter 7

REFERENCES

- He, X., & Zheng, Z. (2019, November). A Driving Warning Method based on YOLOV3 and Neural Network. In 2019 IEEE International Conference on Service Operations and Logistics, and Informatics (SOLI) (pp. 116-121). IEEE.
- Wang, X., Meng, L. M., Zhang, B., Lu, J., & Du, K. L. (2013, December). A video-based traffic violation detection system. In Proceedings 2013 International Conference on Mechatronic Sciences, Electric Engineering and Computer (MEC) (pp. 1191-1194). IEEE.
- Masurekar, O., Jadhav, O., Kulkarni, P., & Patil, S. (2020). Real time object detection using YOLOv3. International Research Journal of Engineering and Technology (IRJET), 7(03), 3764-3768.
- Adarsh, P., Rathi, P., & Kumar, M. (2020, March). YOLO v3-Tiny: Object Detection and Recognition using one stage improved model. In 2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS) (pp. 687-694). IEEE.
- Abhiraj Biswas et. al., Classification of Objects in Video Records using Neural Network “Framework,” International conference SmartSystemsandInventiveTechnology,2018.
- Shetty, A. K., Saha, I., Sanghvi, R. M., Save, S. A., & Patel, Y. J. (2021, April). A review: Object detection models. In 2021 6th International Conference for Convergence in Technology (I2CT) (pp. 1-8). IEEE.
- Adarsh, P., Rathi, P., & Kumar, M. (2020, March). YOLO v3-Tiny: Object Detection and Recognition using one stage improved model. In 2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS) (pp. 687-694). IEEE.