**Project Report**

**On**

**Traffic Signal Violation Detection System**

**Submitted as partial fulfillment for the award of**

**BACHELOR OF TECHNOLOGY**

**DEGREE**

**Session 2022-23**

**In**

**Information Technology**

**By:**

**Chihit Gaur (1901610130036)**

**Abhishek Tyagi (1901610130017)**

**Manas (1901610130025)**

**Under the guidance of: Dr. Rakesh Arora**

|  |  |
| --- | --- |
| C:\Users\Utkarsh\Desktop\index.png | **Krishna Engineering College, Ghaziabad**  **Department of Information Technology** |
| APJAK logo | |

**AFFILIATED TO**

**DR. A.P.J. ABDUL KALAM TECHNICAL UNIVERSITY, U.P., LUCKNOW**

**(Formerly UPTU)**

|  |  |
| --- | --- |
|  |  |
|  |  |
|  | |

**STUDENT’S DECLARATION**

We hereby declare that the work being presented in this report entitled “Traffic Signal Violation Detection System” is an authentic record of our own work carried out under the supervision of “Dr. Rakesh Tyagi”.

The matter embodied in this report has not been submitted by us for the award of any other degree.

**Dated:**

**Signature of student Signature of student Signature of student**

**Chihit... Manas Abhishek..**

**Department: IT Department: IT Department: IT**

## This is to certify that the above statement made by the candidates is correct to the best of my knowledge.

|  |  |
| --- | --- |
| Signature of HOD(Prof. (Dr.) Deepak Kumar Singh)(Information Technology Department)Date............................ | Signature of Supervisor(Dr. Rakesh Tyagi)(Associate Professor)(Information Technology Department) |

## CERTIFICATE

This is to certify that Project Report entitled “Traffic Signal Violation Detection System” which is submitted by Chihit Gaur, Abhishek Tyagi and Manas in partial fulfillment of the requirement for the award of degree B. Tech. in Department of Information Technology of Dr. A.P.J. Abdul Kalam Technical University, formerly Uttar Pradesh Technical University is a record of the candidate own work carried out by them under my supervision. The matter embodied in this thesis is original and has not been submitted for the award of any other degree.

**Dr. Rakesh Tyagi**

**Date**

ACKNOWLEDGEMENT

*It gives us a great sense of pleasure to present the report of the B. Tech Project undertaken during B. Tech. Final Year. We owe special debt of gratitude to Professor Dr. Rakesh Tyagi Department of Information Technology KEC Ghaziabad for her constant support and guidance throughout the course of our work. Her sincerity, thoroughness and perseverance have been a constant source of inspiration for us. It is only her cognizant efforts that our endeavors have seen light of the day.*

*We also take the opportunity to acknowledge the contribution of Professor Deepak Kumar Singh, Head, Department of Information Technology KEC Ghaziabad for his full support and assistance during the development of the project.*

*We also do not like to miss the opportunity to acknowledge the contribution of all faculty members of the department for their kind assistance and cooperation during the development of our project. Last but not the least, we acknowledge our friends for their contribution in the completion of the project.*

*Signature:*

*Chihit Gaur*

*1901610130015*

*Date:*

*Signature:*

*Abhishek Kumar*

*1901610130002*

*Date:*

*Signature:*

*Manas*

*1901610130032*

*Date:*

ABSTRACT

***The Traffic Signal Violation Detection System project aims to enhance road safety and traffic management by developing an intelligent system capable of detecting and recording traffic signal violations. This project report provides an overview of the project's objectives, methodology, and the hardware used during its implementation.***

***The methodology involves data collection, preprocessing of video frames, object detection and tracking, traffic signal detection, violation detection, and alert generation and reporting. Diverse traffic video datasets were collected and processed to extract relevant frames. Computer vision techniques, including deep learning-based object detection models, were employed to detect and track vehicles in the frames. Traffic signal detection involved a combination of image processing techniques and pattern recognition algorithms. Violations such as red light running and stop line violations were detected by analyzing vehicle trajectories and signal states. Real-time alerts were generated to notify authorities, and comprehensive violation reports were automatically generated for further analysis.***

***The hardware used in the project included cameras or video capture devices for data collection, a powerful computing system with a dedicated Graphics Processing Unit (GPU) for real-time analysis, communication devices for transmitting alerts and reports, and power supply systems for uninterrupted operation.***

***The implementation of the Traffic Signal Violation Detection System offers several benefits, such as improved road safety, efficient traffic management, and valuable data for analysis and decision-making. Future enhancements may involve integration with surveillance cameras and machine learning algorithms to improve accuracy and handle complex traffic scenarios.***

***The successful integration of hardware components, such as cameras, computing systems, GPUs, communication devices, and power supply, played a crucial role in the real-time analysis, accurate violation detection, and effective communication with traffic management authorities.***

***Overall, the Traffic Signal Violation Detection System project demonstrates the potential of leveraging computer vision and hardware technologies to enhance road safety, reduce traffic violations, and improve traffic management at signalized intersections.***

**TABLE OF CONTENTS** Page

DECLARATION ii

CERTIFICATE iii

ACKNOWLEDGEMENTS iv

ABSTRACT v

LIST OF TABLES 13

LIST OF FIGURES 14

LIST OF ABBREVIATIONS 15

CHAPTER 1 INTRODUCTION 16

1.1. OBJECTIVES 17

CHAPTER 2 SOFTWARE REQUIREMENT SPECIFICATION 19

2.1. METHODOLOGY 24

2.2. VIOLATION DETECTION 26

2.3. USE CASE DIAGRAM 27

2.4 LIMITATIONS 29

CHAPTER 3 IMPLEMENTATION 31

3.1. COMPUTER VISION 31

3.2. SYSTEM GUI 33

3.3 SOFTWARE FLOW 37

CHAPTER 4 CONCLUSION 40

APPENDIX A................................................................................................ 43

APPENDIX B................................................................................................ 53

REFERENCES................................................................................................ 72**LIST OF TABLES**

**Table No. Title Page No.**

**1 Darknet – 53 25**

**LIST OF FIGURES**

**Figure No. Title Page No.**

**2.3 Use Case 27**

**3.2.1 User Interface View 33**

**3.2.2 Adding Footage to Interface 34**

**3.2.3 Region of Interest 35**

**3.2.4 Line Coordinates 35**

**3.2.5 Final Output Display 36**

**3.3 Overall Software Flow 39**

**LIST OF ABBREVIATIONS**

1. TVDS: Traffic Violation Detection System
2. VCA: Video Content Analysis
3. ML: Machine Learning
4. DL: Deep Learning
5. CV: Computer Vision
6. IoT: Internet of Things
7. API: Application Programming Interface
8. GUI: Graphical User Interface
9. OCR: Optical Character Recognition
10. FPS: Frames Per Second
11. ROI: Region of Interest
12. IR: Infrared
13. UI: User Interface
14. UX: User Experience

**CHAPTER 1**

**INTRODUCTION**

Traffic congestion and violations at signalized intersections pose significant challenges to road safety and traffic management. In order to address these issues, the Traffic Signal Violation Detection System project has been initiated. The project aims to develop an intelligent system that can effectively detect and record instances of traffic signal violations, such as red light running and stop line violations. By leveraging computer vision techniques, real-time alerts, and automated violation reports, the system intends to enhance road safety, reduce violations, and improve traffic management at signalized intersections.

Traffic signal violations not only increase the risk of accidents but also disrupt the smooth flow of traffic, causing congestion and delays. Traditional methods of monitoring and enforcing traffic signal adherence rely heavily on manual observation, which is time-consuming, subjective, and often prone to human errors. To overcome these limitations, the Traffic Signal Violation Detection System project focuses on harnessing the power of artificial intelligence and computer vision technology.

The system employs sophisticated computer vision algorithms to analyze video footage captured at signalized intersections. By detecting and tracking vehicles in the video frames, the system can precisely monitor their behavior in relation to traffic signals. It can identify instances where vehicles fail to comply with the signal indications, such as running a red light or crossing the stop line. Real-time alerts are generated to promptly notify traffic management authorities about violations, enabling them to take immediate action.

Furthermore, the Traffic Signal Violation Detection System automatically generates comprehensive violation reports that include crucial information such as the date, time, location, and vehicle details. These reports serve as valuable evidence for further analysis, enforcement, and decision-making processes. By providing an objective and accurate record of violations, the system helps authorities prioritize enforcement efforts, identify high-risk areas, and devise effective strategies to mitigate traffic violations.

The implementation of the Traffic Signal Violation Detection System has the potential to significantly improve road safety and traffic management. By discouraging traffic signal violations and enforcing compliance, the system aims to create a safer environment for all road users. Moreover, the availability of real-time violation alerts and automated violation reports empowers traffic management authorities to take proactive measures, optimize traffic flow, and allocate resources efficiently.

In conclusion, the Traffic Signal Violation Detection System project endeavors to leverage the advancements in computer vision technology to enhance road safety, reduce traffic violations, and improve traffic management at signalized intersections. By automating the detection and reporting of violations, the system aims to contribute to safer roads, efficient traffic flow, and a better overall commuting experience.

* 1. **Objectives**

The primary objectives of the Traffic Signal Violation Detection System project are as follows:

1. Automated Detection: Develop a system that can automatically detect various types of traffic violations, such as speeding, red light running, illegal parking, and improper lane usage.

2. Real-time Monitoring: Enable real-time monitoring of traffic violations by integrating the system with surveillance cameras or other relevant sensors.

3. Accuracy and Reliability: Ensure high accuracy and reliability in detecting traffic violations to minimize false positives and negatives, thus reducing the chances of wrongful penalties or missed violations.

4. Multilingual Support: Design the system to support multiple languages to cater to diverse populations and locations.

5. Scalability: Develop a scalable system that can handle a large volume of traffic data and adapt to increasing traffic densities in urban areas.

6. Integration with Law Enforcement: Enable seamless integration with law enforcement agencies, allowing automatic generation of violation reports and facilitating effective enforcement of traffic laws.

7. Warning and Alert Mechanisms: Implement mechanisms to issue warnings or alerts to drivers in real-time, notifying them of their violation and promoting safer driving behavior.

8. Data Analysis and Reporting: Provide comprehensive data analysis and reporting features to identify traffic patterns, hotspots for violations, and trends over time, aiding in proactive traffic management strategies.

9. User-Friendly Interface: Create an intuitive and user-friendly interface for law enforcement personnel to access and analyze violation data efficiently.

10. Compliance with Privacy Regulations: Ensure compliance with privacy regulations by employing appropriate data anonymization techniques and protecting personally identifiable information.

11. Cost-Effectiveness: Develop a cost-effective solution by leveraging existing infrastructure and optimizing hardware and software components.

12. Continuous Improvement: Establish a framework for continuous improvement and refinement of the system through user feedback, data analysis, and technological advancements.

**CHAPTER 2**

# SOFTWARE REQUIREMENT SPECIFICATION

*The project was developed using Python. This chapter describes about the software packages and libraries which were used in the project. The project uses TKINTER. Image Processing was implemented with the help OpenCV**.*

1. **The Python Programming Language**

### Python was developed by Guido van Rossum in the early 1990s. It has a lot of ad- vantages over other object-oriented programming languages like C++ and Java. The primary advantages of the language are as follows:

### Due to its popularity among scientists for numerical computation, python was adapted rapidly for machine learning and artificial intelligence. Some of the li- braries which are available are Scikit-learn, Scipy, Numpy, Tensorflow, Keras, Pytorch, Matplotlib, etc.

### It gives the user more flexibility in terms of application development as compared to MATLAB even though the latter is recommended for faster prototyping and diverse in terms of tools available.

### It is simple to program in python and learning curve is not steep. The syntax is easy to use and the use of strict indentation to improve code readability.

### It is independent of platform. Hence, the programs built and tested on Raspbian OS, could also be tested on Windows platform.

1. **OpenCV Library**

OpenCV (Open Source Computer Vision Library) is an image processing library made by Willow Garage of Intel. It was built for real-time computer vision applications. It was designed for efficient computation. It is written in C/C++ and can take advantage of multi-core processing.

OpenCL gives it support for hardware acceleration. OpenCV’s application areas include:

* Egomotion estimation
* Facial recognition system
* Motion understanding
* 2D and 3D feature toolkits
* Segmentation and recognition
* Structure from motion (SFM)
* Motion tracking
* Gesture recognition
* Human computer interaction (HCI)
* Object identification
* Stereopsis stereo vision: depth perception from 2 cameras
* Augmented reality
* Mobile robotic

OpenCV uses a statistical machine learning library that contains:

* + - * Boosting
      * Artificial neural networks
      * Decision tree learning
      * Gradient boosting trees
      * Expectation-maximization algorithm
      * k-nearest neighbor algorithm
      * Naive Bayes classifier
      * Random forest
      * Support vector machine (SVM)
      * Deep neural networks (DNN)

OpenCV has several static libraries. The following modules are available:

* + - * High-level GUI (highgui)
      * Video I/O (videoio)
      * Video Analysis (video)
      * Image Processing (imgproc)
      * 2D Features Framework (features2d
      * Core functionality (core)
      * Object Detection (objdetect)
      * Camera Calibration and 3D Reconstruction (calib3d)
      * Some other helper modules, such as FLANN and Google test wrappers and oth- ers.

1. **Numpy**

NumPy is the essential toolkit for scientific computing in Python. It contains but is not limited to:

* + - * A highly impactful N-dimensional array object
      * helpful Fourier transform, linear algebra, and random number capabilities
      * tools for use with Fortran code and C/C++
      * sophisticated (broadcasting) functions

It can also be used as an effective multi-dimensional generic data container. Cus- tomized and less-restricted data-types can be defined. Thus, it can be easily used with a broad range of databases.

NumPy is licensed under the BSD license, enabling reuse with few restrictions.

1. **KERAS**

KERAS is a popular Python package that provides a high-level, user-friendly interface for building and training deep learning models. It is widely used in the field of artificial intelligence and has gained popularity due to its simplicity, flexibility, and compatibility with various deep learning frameworks, including TensorFlow, Theano, and CNTK.

Keras abstracts away the complexities of low-level deep learning implementation, allowing users to focus on designing and experimenting with neural networks. It offers a range of pre-defined layers, activation functions, optimizers, and loss functions, making it easy to construct deep learning models with just a few lines of code. Keras provides a modular and intuitive API, enabling users to quickly prototype and iterate on different network architectures and configurations.

1. **TensorFlow**

**TensorFlow is a widely used open-source Python package that provides a comprehensive ecosystem for developing and deploying machine learning models. It was developed by the Google Brain team and has gained immense popularity due to its versatility, scalability, and extensive community support. TensorFlow enables developers to build and train neural networks and perform various computational tasks efficiently.**

**1. Struct**

**The struct package is a built-in module in Python that provides functionalities for working with structured binary data. It allows you to interpret binary data stored in strings or files and provides functions to pack and unpack data into various binary formats. The struct package is particularly useful when working with binary data such as network protocols, file formats, and low-level data manipulation. It provides a way to handle data at the byte level, allowing you to read and write binary data in a structured manner.**

**2. PIL**

**The Python Imaging Library (PIL) is a popular open-source library that provides extensive support for image processing and manipulation tasks. PIL offers a wide range of functionalities, making it a versatile tool for working with images in various formats. It simplifies complex image processing tasks and provides a straightforward interface for common image operations.**

### 2.1 Methodology

**Vehicle Classification**

From the given video footage, moving objects are detected. An object detection model YOLOv3 is used to classify those moving objects into respective classes. YOLOv3 is the third object detection algorithm in YOLO (You Only Look Once) family. It improved the accuracy with many tricks and is more capable of detecting objects. The classifier model is built with Darknet-53 architecture. Table-1 shows how the neural network architecture is designed.

**Features:**

**1. Bounding Box Predictions:**

YOLOv3 is a single network the loss for objectiveness and classification needs to be calculated separately but from the same network. YOLOv3 predicts the objectiveness score using logistic regression where 1 means complete overlap of bounding box prior over the ground truth object. It will predict only 1 bonding box prior for one ground truth object and any error in this would incur for both classification as well as detection loss. There would also be other bounding box priors which would have objectiveness score more than the threshold but less than the best one. These errors will only incur for the detection loss and not for the classification loss.

**2. Class Prediction:**

YOLOv3 uses independent logistic classifiers for each class instead of a regular softmax layer. This is done to make the classification multi-label classification. Each box predicts the classes the bounding box may contain using multilabel classification.

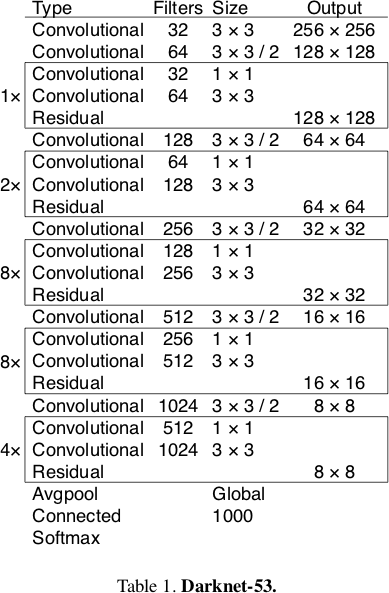
.

**3. Predictions across scales:**

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

**4. Feature Extractor:**

YOLOv3 uses a new network- Darknet-53. Darknet-53 has 53 convolutional layers, its deeper than YOLOv2 and it also has residuals or shortcut connections. Its powerful than Darknet -19 and more efficient than ResNet-101 or ResNet-152.



## 

## 2.2 Violation Detection

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

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

## Violation detection in a traffic violation detection system involves analyzing the behavior of vehicles captured by surveillance cameras or sensors to identify instances of traffic rule violations. This process typically includes the following steps:

## 1. Object Detection: The system detects and locates vehicles within the captured frames using computer vision techniques.

## 2. Behavior Analysis: The system analyzes the behavior of the detected vehicles to identify potential violations. This may involve assessing factors such as vehicle speed, direction, lane usage, or compliance with traffic signals and rules.

## 3. Rule-based or Machine Learning Models: The system applies predefined rules or machine learning models to determine if the observed behavior constitutes a violation. Rule-based approaches involve comparing the detected behavior against specific thresholds or conditions for different violations. Machine learning models can learn from labeled data to automatically recognize patterns associated with different violations.

## 4. Violation Identification: Once a violation is detected and confirmed, the system classifies and labels it based on the type of violation, such as speeding, red light running, illegal parking, or improper lane usage.

## 5. Evidence Collection: The system captures and stores relevant information and evidence related to the violation, including images, videos, timestamps, and location data. This evidence is crucial for generating detailed violation reports.

## 6. Reporting and Alerting: The system generates violation reports that include comprehensive information about the violation, such as date, time, location, vehicle details, and supporting evidence. These reports can be automatically sent to law enforcement personnel for further action or used for statistical analysis and traffic management purposes. Additionally, real-time alerts or warnings may be issued to drivers or law enforcement officers to notify them of detected violations.

## Overall, violation detection involves a combination of object detection, behavior analysis, rule-based or machine learning algorithms, and evidence collection to identify and report instances of traffic rule violations captured by the traffic violation detection system.

* 1. **Use case**

**2.3.1 TVD System data linking use case diagram**

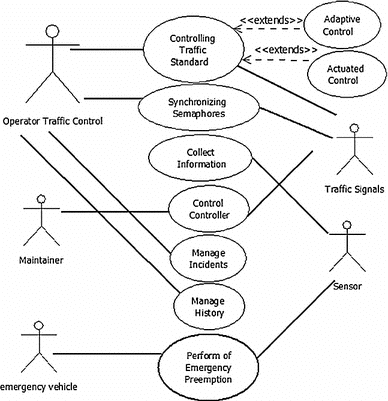


Fig-2.3: Use case diagram

A use case diagram for a traffic violation detection system project represents the various interactions between the system and its users, showcasing the functionality and the goals of the system. Here's a brief explanation of the main components typically found in a use case diagram for such a project:

1. Actors:

- Driver: The person operating the vehicle.

- Law Enforcement Officer: The authorized personnel responsible for enforcing traffic laws.

- System Administrator: The individual responsible for managing and maintaining the traffic violation detection system.

2. Use Cases:

- Detect Speeding: The system detects instances of vehicles exceeding the speed limit.

- Detect Red Light Violation: The system identifies vehicles that run red lights at intersections.

- Detect Illegal Parking: The system detects vehicles parked in prohibited areas or violating parking regulations.

- Generate Violation Report: The system generates a detailed report of the detected violation, including relevant information such as date, time, location, and evidence (e.g., images or videos).

- Notify Law Enforcement: The system alerts the law enforcement officers about the violation and provides them with the violation report.

- Issue Warning/Alert: The system issues warnings or alerts to drivers in real-time, notifying them of their violation and promoting safer driving behavior.

- Manage User Accounts: The system allows the system administrator to create, modify, and manage user accounts, including law enforcement officers' and administrators' accounts.

- Generate Traffic Analysis Reports: The system generates reports analyzing traffic patterns, violation hotspots, and trends over time for proactive traffic management.

3. Relationships:

- Association: It represents the relationship between an actor and a use case, indicating their interaction.

- Include: It shows that one use case includes the functionality of another use case. For example, "Generate Violation Report" may include "Detect Speeding" or "Detect Red Light Violation."

- Extend: It represents optional or alternative functionality that can be added to a use case. For instance, "Issue Warning/Alert" may extend from "Detect Speeding" or "Detect Red Light Violation."

The use case diagram provides a high-level overview of the interactions and functionalities of the traffic violation detection system, helping stakeholders understand the system's behavior and requirements. It serves as a starting point for more detailed analysis and design phases of the project.

* 1. **Limitations**

While a traffic violation detection system can provide numerous benefits, it also has certain limitations. Here are a few limitations commonly associated with such projects:

1. Accuracy and Reliability: The system's accuracy and reliability in detecting traffic violations may not be 100%, leading to potential false positives or false negatives. Factors such as poor visibility, adverse weather conditions, or system errors can impact the accuracy of violation detection.

2. Limited Coverage: The system's coverage is dependent on the placement and availability of surveillance cameras or other detection sensors. Areas without proper camera coverage may result in undetected violations.

3. Privacy Concerns: The use of surveillance cameras and data collection for violation detection raises privacy concerns. It is crucial to handle collected data responsibly, ensuring compliance with privacy regulations and protecting personally identifiable information.

4. Dependence on Infrastructure: The system relies on proper functioning and maintenance of the surveillance cameras, communication networks, and other infrastructure components. Any infrastructure-related issues can affect the system's performance and effectiveness.

5. Cost and Resource Intensiveness: Implementing and maintaining a traffic violation detection system can be financially demanding. It requires investment in hardware, software, infrastructure, and ongoing maintenance, which may pose challenges for budget-constrained organizations or regions.

6. Adapting to Changing Regulations: Traffic laws and regulations can evolve over time, requiring the system to be updated and adjusted accordingly. Keeping up with changing regulations and ensuring compliance can be a continuous effort.

# CHAPTER 3

# IMPLEMENTATION

# Computer Vision

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

OpenCV (Open Source Computer Vision Library) is a widely used open-source library for computer vision and image processing tasks. It provides a comprehensive set of functions and algorithms that can be utilized in various stages of a traffic violation detection system project. Here's a brief overview of how OpenCV can be used in the creation of such a project:

1. Data Acquisition and Preprocessing:

- OpenCV enables the capture and preprocessing of video feeds or images from surveillance cameras or other sources. It provides functions for accessing video streams, reading image files, and performing basic image preprocessing tasks such as resizing, cropping, and filtering.

2. Object Detection and Tracking:

- OpenCV offers various algorithms and techniques for object detection and tracking. These include popular methods such as Haar cascades, Histogram of Oriented Gradients (HOG), and Deep Learning-based approaches (e.g., YOLO and SSD). These algorithms can be used to detect and track vehicles or other objects of interest in the captured video frames.

3. Violation Recognition:

- OpenCV can be utilized to implement algorithms for recognizing different types of traffic violations. For example, it can be used to estimate vehicle speeds based on frame-to-frame motion analysis or optical flow. OpenCV's feature extraction and matching techniques can aid in detecting red light violations or improper lane usage.

4. Image Processing and Analysis:

- OpenCV provides a wide range of image processing functions that can be employed to enhance and analyze captured frames. These functions include image filtering, morphological operations, edge detection, and contour analysis. These techniques can help in preprocessing, object segmentation, and feature extraction for traffic violation detection.

5. Visualization and Reporting:

- OpenCV allows the visualization of processed frames and detected violations by overlaying graphical annotations, bounding boxes, or text labels. It facilitates the generation of informative visual reports or real-time displays to aid in monitoring and analysis.

6. Integration with Machine Learning:

- OpenCV can be combined with machine learning frameworks (e.g., TensorFlow or PyTorch) to train custom object detection or recognition models specifically tailored for traffic violation detection. OpenCV provides functionalities to load and utilize pre-trained models and integrate them seamlessly into the traffic violation detection system.

Overall, OpenCV's rich set of functions and algorithms enable developers to implement various computer vision tasks necessary for a traffic violation detection system. It simplifies the process of data acquisition, preprocessing, object detection, violation recognition, and visualization, contributing to the efficient and effective creation of such systems.

# Graphical User Interface(GUI)

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



Fig-3.2.1: Initial user interface view.

Primarily, for the start of the project usage, the administrator needs to open a video footage using ‘Open’ item that can be found under ‘File’ (Figure-2). The administrator can open any video footage from the storage files (Figure-3).

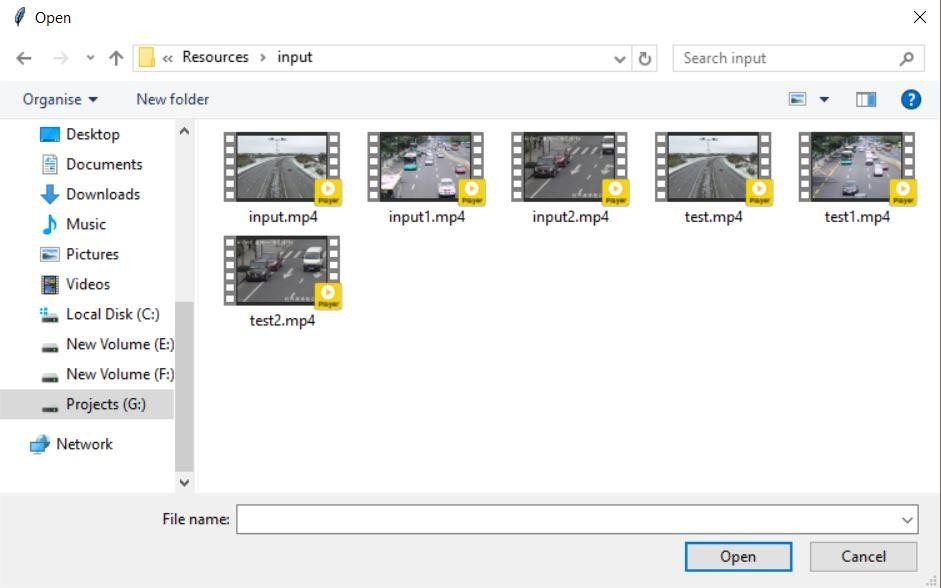


Fig-3.2.2: Opening a video footage from storage.

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



Fig-3.2.3: Region of Interest (Drawing signal line)

Selecting the region of interest will start violation detection system. The coordinates of the line drawn will be shown on console (Figure-5). The violation detection system will start immediately after the line is drawn. At first the weights will be loaded. Then the system will detect objects and check for violations. The output will be shown frame by frame from the GUI (Figure-6).

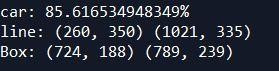


Fig-3.2.4: Line Coordinates (from console)

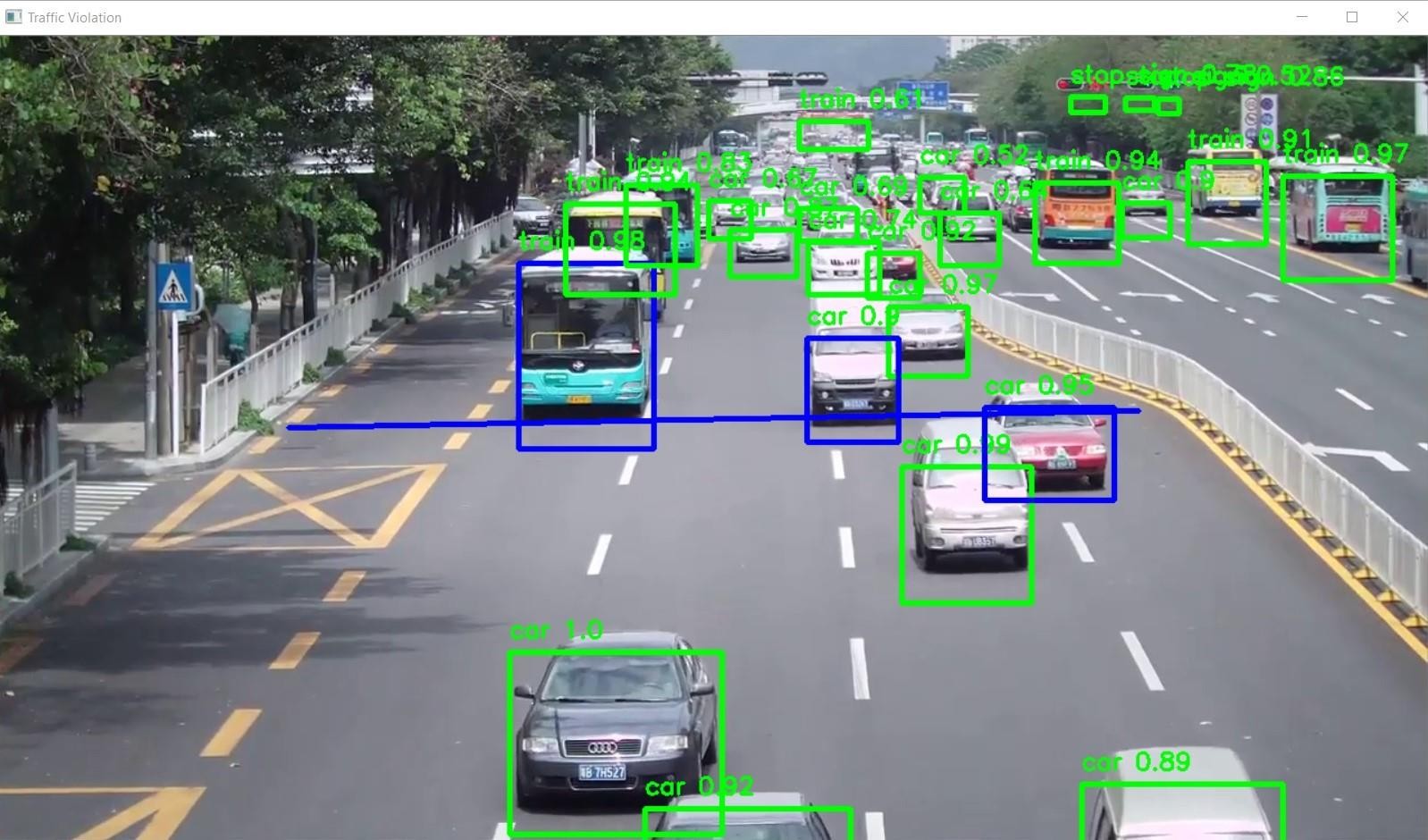


Figure-3.2.5: Final Output (on each frame)

The system will show output until the last frame of the footage. In background a ‘output.mp4’ will be generated. The file will be in ‘output’ folder of ‘Resources’. The process will be immediately terminated by clicking ‘q’.

After processing a video footage, the administrator can add another video footage from the initial file manager (Figure-2). If the work is complete the administrator can quit using ‘Exit’ item from File option.

**Library used for GUI:**

* Tkinter
  1. **The overall flow of the software:**

The overall flow of software in a traffic violation detection system project typically involves several stages and components. Here's a brief overview of the general flow:

1. Data Acquisition:

- Capture Data: Surveillance cameras or other sensors capture traffic-related data, such as video feeds, images, or sensor readings, from the road network.

- Data Preprocessing: Raw data is processed and preprocessed to extract relevant information, filter noise, and enhance the quality of the data for further analysis.

2. Violation Detection:

- Object Detection and Tracking: Using computer vision techniques, objects of interest (e.g., vehicles) are detected and tracked in the captured data.

- Violation Recognition: Algorithms analyze the tracked objects to identify various types of traffic violations, such as speeding, red light running, or illegal parking. This involves applying specific rules or machine learning models to determine violations based on predefined criteria.

3. Violation Verification:

- Verification Algorithms: Detected violations are subjected to verification algorithms to minimize false positives. Additional checks, such as analyzing multiple frames or using speed estimation algorithms, can be performed to increase accuracy.

4. Violation Reporting:

- Generation of Violation Reports: Validated violations are recorded, and comprehensive reports are generated, including information such as date, time, location, vehicle details, violation type, and supporting evidence (images, videos, etc.).

- Database Storage: Violation data is stored in a database for future reference, analysis, and retrieval.

5. Alert and Notification:

- Real-time Alerts: Depending on the system's capabilities, real-time alerts or warnings can be issued to drivers or law enforcement officers to notify them of detected violations. This may involve audio-visual alerts or messages sent to connected devices.

6. Data Analysis and Reporting:

- Traffic Pattern Analysis: Collected violation data is analyzed to identify traffic patterns, violation hotspots, and trends over time. Statistical analysis, visualization techniques, or machine learning algorithms may be employed to derive meaningful insights.

- Reporting: Reports are generated based on the analysis, providing stakeholders with detailed information about traffic violations, their frequency, geographical distribution, and other relevant metrics.

7. System Management and Maintenance:

- User Management: System administrators oversee user accounts, access rights, and privileges.

- System Maintenance: Regular maintenance and updates are performed to ensure the system's smooth operation, including software updates, hardware maintenance, and data backups.

- Continuous Improvement: User feedback, system performance analysis, and technological advancements are considered for ongoing improvements, bug fixes, and feature enhancements.

It's important to note that the specific flow and components may vary based on the design choices, system architecture, and the technologies used in a particular traffic violation detection system project.

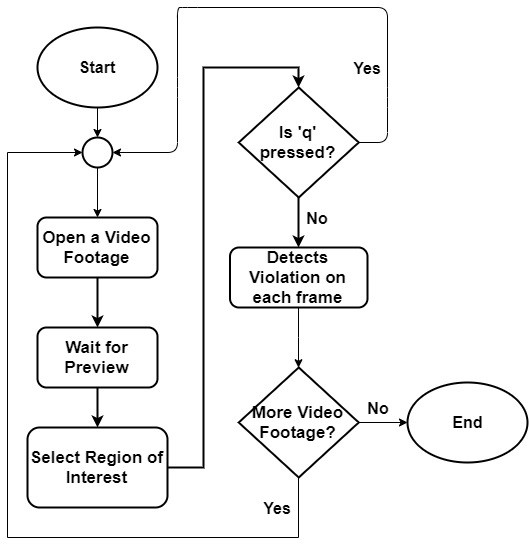


Fig-3.3: Overall flow of the software.

**CHAPTER 4**

# CONCLUSION

The designed algorithm was effectively able to detect the type of violation specified on this project which are denying traffic signal. The convergence of detection for the traffic violation mentioned is dissimilar, since it has a different threshold condition. The system provides detection for traffic signal violation. Further, the system is able to process one data at a time. Also, the program runtime is somewhat slow, and can be improved by using a computer with high speed processor specifications or GPU.

Future research about the application of the designed algorithm for other advanced image processing techniques. Since, this may improve the program runtime of the system by neglecting other unnecessary steps done in a background difference method. A computer vision algorithm may be done instead to provide more intelligence in the system. Our future plan is to implement the number plate detection with OCR support to make this system more robust.

The Traffic Violation Detection System is an innovative solution that can significantly improve road safety. By detecting and alerting drivers of traffic violations, it can help reduce the number of accidents caused by reckless driving. The system can also display the penalty incurred and reason for license suspension, which can serve as a deterrent for repeat offenders. As technology continues to advance, it is important to explore new ways to leverage it for the benefit of society. The Traffic Violation Detection System is a prime example of how technology can be used to make our roads safer and more efficient.

**Future Directions**

Future directions for traffic violation detection system projects may include advancements in technology, data analysis, and system integration. Here are some potential future directions:

1. Enhanced Accuracy and Real-time Detection: Continued research and development to improve the accuracy and real-time detection capabilities of the system, reducing false positives and negatives. This can involve the use of advanced computer vision algorithms, machine learning techniques, and the integration of more powerful hardware.

2. Integration with Connected and Autonomous Vehicles: As connected and autonomous vehicles become more prevalent, future traffic violation detection systems can integrate with these vehicles' systems to improve violation detection and enhance overall traffic safety. This may involve direct communication between the system and the vehicles, enabling proactive violation prevention and enforcement.

3. Advanced Sensor Technologies: Exploration and integration of advanced sensor technologies such as LiDAR, radar, or infrared sensors to complement or enhance the existing camera-based detection systems. These sensors can provide additional data for more accurate and comprehensive violation detection, particularly in challenging weather or lighting conditions.

4. Behavioral Analysis and Predictive Analytics: Expanding the capabilities of traffic violation detection systems to analyze driver behavior and predict potential violations before they occur. This can involve analyzing patterns, driver actions, and contextual information to anticipate risky behaviors and take proactive measures.

5. Integration with Smart City Infrastructure: Integration with smart city infrastructure, including traffic management systems, intelligent transportation systems, and centralized control centers. This integration allows for seamless information exchange, improved traffic flow management, and efficient enforcement of traffic regulations.

6. Multimodal Violation Detection: Incorporating multimodal data sources, such as audio data or vehicle sensor data, to enhance violation detection. This can enable the detection of violations like horn misuse, aggressive driving behavior, or lane departure, providing a more comprehensive understanding of traffic violations.

7. Data-driven Traffic Management: Leveraging the collected violation data to gain insights into traffic patterns, identify high-risk areas, and optimize traffic management strategies. Data analysis techniques, such as machine learning and predictive modeling, can be employed to develop proactive traffic management approaches, including dynamic speed limits, optimized signal timings, and adaptive traffic control systems.

8. Privacy Protection and Ethical Considerations: Paying careful attention to privacy concerns and ensuring compliance with ethical guidelines and regulations. Implementing privacy-preserving techniques, secure data handling protocols, and anonymization methods to protect personal information while maintaining system effectiveness.

9. Collaborative Systems and Citizen Engagement: Exploring the involvement of citizens as active participants in the traffic violation detection process. This can include the integration of citizen reporting platforms, crowd-sourced data collection, and citizen engagement initiatives to enhance traffic safety and promote community involvement.

10. Continuous Improvement and Adaptation: Establishing mechanisms for continuous system improvement, including regular updates, feedback collection, and adaptation to changing traffic regulations and technological advancements. Embracing an iterative development approach to address emerging challenges and incorporate user feedback for system enhancement.

These future directions aim to enhance the accuracy, efficiency, and effectiveness of traffic violation detection systems, ultimately contributing to improved traffic safety and management.

**APPENDIX A**

**GUI Coding Data**

from tkinter import \*

from pil import Image, ImageTk

from tkinter import filedialog

import object\_detection as od

import imageio

import cv2

class Window(Frame):

def \_\_init\_\_(self, master=None):

Frame.\_\_init\_\_(self, master)

self.master = master

self.pos = []

self.line = []

self.rect = []

self.master.title("GUI")

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

self.counter = 0

menu = Menu(self.master)

self.master.config(menu=menu)

file = Menu(menu)

file.add\_command(label="Open", command=self.open\_file)

file.add\_command(label="Exit", command=self.client\_exit)

menu.add\_cascade(label="File", menu=file)

analyze = Menu(menu)

analyze.add\_command(label="Region of Interest", command=self.regionOfInterest)

menu.add\_cascade(label="Analyze", menu=analyze)

self.filename = "Images/home.jpg"

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

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

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

self.canvas = Canvas(master = root, width = self.w, height = self.h)

self.canvas.create\_image(20, 20, image=self.tkimage, anchor='nw')

self.canvas.pack()

def open\_file(self):

self.filename = filedialog.askopenfilename()

cap = cv2.VideoCapture(self.filename)

reader = imageio.get\_reader(self.filename)

fps = reader.get\_meta\_data()['fps']

ret, image = cap.read()

cv2.imwrite('G:/Traffic Violation Detection/Traffic Signal Violation Detection System/Images/preview.jpg', image)

self.show\_image('G:/Traffic Violation Detection/Traffic Signal Violation Detection System/Images/preview.jpg')

def show\_image(self, frame):

self.imgSize = Image.open(frame)

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

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

self.canvas.destroy()

self.canvas = Canvas(master = root, width = self.w, height = self.h)

self.canvas.create\_image(0, 0, image=self.tkimage, anchor='nw')

self.canvas.pack()

def regionOfInterest(self):

root.config(cursor="plus")

self.canvas.bind("<Button-1>", self.imgClick)

def client\_exit(self):

exit()

def imgClick(self, event):

if self.counter < 2:

x = int(self.canvas.canvasx(event.x))

y = int(self.canvas.canvasy(event.y))

self.line.append((x, y))

self.pos.append(self.canvas.create\_line(x - 5, y, x + 5, y, fill="red", tags="crosshair"))

self.pos.append(self.canvas.create\_line(x, y - 5, x, y + 5, fill="red", tags="crosshair"))

self.counter += 1

# elif self.counter < 4:

# x = int(self.canvas.canvasx(event.x))

# y = int(self.canvas.canvasy(event.y))

# self.rect.append((x, y))

# self.pos.append(self.canvas.create\_line(x - 5, y, x + 5, y, fill="red", tags="crosshair"))

# self.pos.append(self.canvas.create\_line(x, y - 5, x, y + 5, fill="red", tags="crosshair"))

# self.counter += 1

if self.counter == 2:

#unbinding action with mouse-click

self.canvas.unbind("<Button-1>")

root.config(cursor="arrow")

self.counter = 0

#show created virtual line

print(self.line)

print(self.rect)

img = cv2.imread('G:/Traffic Violation Detection/Traffic Signal Violation Detection System/Images/preview.jpg')

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

cv2.imwrite('G:/Traffic Violation Detection/Traffic Signal Violation Detection System/Images/copy.jpg', img)

self.show\_image('G:/Traffic Violation Detection/Traffic Signal Violation Detection System/Images/copy.jpg')

## for demonstration

# (rxmin, rymin) = self.rect[0]

# (rxmax, rymax) = self.rect[1]

# tf = False

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

# print(tf)

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

# print(tf)

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

# print(tf)

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

# print(tf)

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

# if tf:

# cv2.rectangle(img, (rxmin,rymin), (rxmax,rymax), (255,0,0), 3)

# else:

# cv2.rectangle(img, (rxmin,rymin), (rxmax,rymax), (0,255,0), 3)

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

#image processing

self.main\_process()

print("Executed Successfully!!!")

#clearing things

self.line.clear()

self.rect.clear()

for i in self.pos:

self.canvas.delete(i)

def intersection(self, p, q, r, t):

print(p, q, r, t)

(x1, y1) = p

(x2, y2) = q

(x3, y3) = r

(x4, y4) = t

a1 = y1-y2

b1 = x2-x1

c1 = x1\*y2-x2\*y1

a2 = y3-y4

b2 = x4-x3

c2 = x3\*y4-x4\*y3

if(a1\*b2-a2\*b1 == 0):

return False

print((a1, b1, c1), (a2, b2, c2))

x = (b1\*c2 - b2\*c1) / (a1\*b2 - a2\*b1)

y = (a2\*c1 - a1\*c2) / (a1\*b2 - a2\*b1)

print((x, y))

if x1 > x2:

tmp = x1

x1 = x2

x2 = tmp

if y1 > y2:

tmp = y1

y1 = y2

y2 = tmp

if x3 > x4:

tmp = x3

x3 = x4

x4 = tmp

if y3 > y4:

tmp = y3

y3 = y4

y4 = tmp

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

return True

else:

return False

def main\_process(self):

video\_src = self.filename

cap = cv2.VideoCapture(video\_src)

reader = imageio.get\_reader(video\_src)

fps = reader.get\_meta\_data()['fps']

writer = imageio.get\_writer('G:/Traffic Violation Detection/Traffic Signal Violation Detection System/Resources/output/output.mp4', fps = fps)

j = 1

while True:

ret, image = cap.read()

if (type(image) == type(None)):

writer.close()

break

image\_h, image\_w, \_ = image.shape

new\_image = od.preprocess\_input(image, od.net\_h, od.net\_w)

# run the prediction

yolos = od.yolov3.predict(new\_image)

boxes = []

for i in range(len(yolos)):

# decode the output of the network

boxes += od.decode\_netout(yolos[i][0], od.anchors[i], od.obj\_thresh, od.nms\_thresh, od.net\_h, od.net\_w)

# correct the sizes of the bounding boxes

od.correct\_yolo\_boxes(boxes, image\_h, image\_w, od.net\_h, od.net\_w)

# suppress non-maximal boxes

od.do\_nms(boxes, od.nms\_thresh)

# draw bounding boxes on the image using labels

image2 = od.draw\_boxes(image, boxes, self.line, od.labels, od.obj\_thresh, j)

writer.append\_data(image2)

# cv2.imwrite('E:/Virtual Traffic Light Violation Detection System/Images/frame'+str(j)+'.jpg', image2)

# self.show\_image('E:/Virtual Traffic Light Violation Detection System/Images/frame'+str(j)+'.jpg')

cv2.imshow('Traffic Violation', image2)

print(j)

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

writer.close()

break

j = j+1

cv2.destroyAllWindows()

root = Tk()

app = Window(root)

root.geometry("%dx%d"%(535, 380))

root.title("Traffic Violation")

root.mainloop()

**APPENDIX B**

**Object Detection Coding Data**

import numpy as np

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

from keras.layers.merge import add, concatenate

from keras.models import Model

import struct

import cv2

class WeightReader:

def \_\_init\_\_(self, weight\_file):

with open(weight\_file, 'rb') as w\_f:

major, = struct.unpack('i', w\_f.read(4))

minor, = struct.unpack('i', w\_f.read(4))

revision, = struct.unpack('i', w\_f.read(4))

if (major\*10 + minor) >= 2 and major < 1000 and minor < 1000:

w\_f.read(8)

else:

w\_f.read(4)

transpose = (major > 1000) or (minor > 1000)

binary = w\_f.read()

self.offset = 0

self.all\_weights = np.frombuffer(binary, dtype='float32')

def read\_bytes(self, size):

self.offset = self.offset + size

return self.all\_weights[self.offset-size:self.offset]

def load\_weights(self, model):

for i in range(106):

try:

conv\_layer = model.get\_layer('conv\_' + str(i))

print("loading weights of convolution #" + str(i))

if i not in [81, 93, 105]:

norm\_layer = model.get\_layer('bnorm\_' + str(i))

size = np.prod(norm\_layer.get\_weights()[0].shape)

beta = self.read\_bytes(size) # bias

gamma = self.read\_bytes(size) # scale

mean = self.read\_bytes(size) # mean

var = self.read\_bytes(size) # variance

weights = norm\_layer.set\_weights([gamma, beta, mean, var])

if len(conv\_layer.get\_weights()) > 1:

bias = self.read\_bytes(np.prod(conv\_layer.get\_weights()[1].shape))

kernel = self.read\_bytes(np.prod(conv\_layer.get\_weights()[0].shape))

kernel = kernel.reshape(list(reversed(conv\_layer.get\_weights()[0].shape)))

kernel = kernel.transpose([2,3,1,0])

conv\_layer.set\_weights([kernel, bias])

else:

kernel = self.read\_bytes(np.prod(conv\_layer.get\_weights()[0].shape))

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), int((net\_w-new\_w)//2):int((net\_w+new\_w)//2), :] = resized

new\_image = np.expand\_dims(new\_image, 0)

return new\_image

def decode\_netout(netout, anchors, obj\_thresh, nms\_thresh, net\_h, net\_w):

grid\_h, grid\_w = netout.shape[:2]

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\_w

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("G:/Traffic Violation Detection/Traffic Signal Violation Detection System/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 = "G:/Traffic Violation Detection/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

**References**

1. G. Ou, Y. Gao and Y. Liu, "Real Time Vehicular Traffic Violation Detection in Traffic Monitoring System," in 2012 IEEE/WIC/ACM, Beijing, China , 2012.
2. X. Wang, L.-M. Meng, B. Zhang, J. Lu and K.-L.Du, "A Video-based Traffic ViolationDetection System," in MEC, Shenyang, China, 2013.
3. Joseph Redmon and Ali Farhadi, "YOLOv3: An Incremental Improvement".
4. Monitoring System,” in 2012 IEEE/WIC/ACM, Beijing, China 2012.
5. W,J, Horry. ;Leach, M.F- Driver-Initiated Distractions: Examining Strategic Adaptation for In-Vehicles task Initiation. Acid. Anal. Prev. 2009,41,115-122.
6. Y, Dong.; Hu, Z.; Chimera, K.; Murayama, N. Driver Inattention Monitoring System for Intelligent Vehicles: A Review. IEEE Intel. Transp. Syst. 2011, 12, 596–614.
7. W.J, Horry.; Leach, M.F Garbed, A. Assessing the Awareness of Performance Decrements in Distracted Drivers. Acid. Anal. Prev. 2008, 41, 675–682.
8. M, Eriksson.; Panicle polos, P.N. Driver Fatigue: A Vision-Based Approach to Automatic Diagnosis. Transp. Res. Part C Emerge. Technol. 2001, 9, 399–413. Sensors 2014, 14 22126.
9. J.C, McCall.; Trivedi, M.M. Video-Based Lane Estimation and Tracking for Driver Assistance: Survey, System, and Evaluation. IEEE INTAL. Transp. Syst. 2006, 7, 20–37.
10. U, Handan.; Kalie, T.; C.; Werner and, M.; Snellen, W.V. An Image Processing System for Driver Assistance. Image Vis. Comput. 2000, 18, 367–376.
11. L.M, Bergama.; Nuevo, J.; Sotelo, M.A.; Berea, R.; Lopez, M.E. Real-Time system for monitoring driver vigilance. IEEE Intell. Transp. Syst. 2006, 7, 63–77.
12. A, Escalera.; Armingol, J.M.; Mata, M. Traffic sign recognition and analysis for intelligent vehicles. Image Vis. Comput. 2003, 12, 247–258.
13. A, Broggi.; Cerri, P.; Ghidoni, S.; Grisleri, P.; Jung, H.G. A New Approach to Urban Pedestrian Detection for Automatic Braking. IEEE Intell. Transp. Syst. 2009, 10, 594–605.
14. F, Jiménez.; Naranjo, J.E.; Gómez, O. Autonomous maneuvering systems for collision avoidance on single carriageway roads. Sensors 2012, 12, 16498–16521.
15. C.G, Keller.; Dang, T.; Fritz, H.; Jools, A.; Rabe, C.; Gavrila, D.M. Active pedestrian safety by automatic braking and evasive steering. IEEE Intell. Transp. Syst. 2011, 12, 1292–1304
16. "Traffic Violation Detection System" by S. S. Kulkarni and S. S. Kulkarni, published in the Journal of Emerging Technologies and Innovative Research (JETIR) in March 2017.
17. "A System for Traffic Violation Detection" by Nourdine Aliane, Javier Fernandez, Mario Mata, and Sergio Bemposta, published in the journal Sensors in 2014.
18. "A video-based traffic violation detection system" by Xiaoyu Zhang, published on ResearchGate in 2022.
19. "YOLOv3: An Incremental Improvement" by Joseph Redmon and Ali Farhadi, published on arXiv in April 2018.
20. "YOLOv3: An Incremental Improvement" by Joseph Redmon and Ali Farhadi, published in the journal Computer Science in 2018.

## 