CHAPTER-I
INTRODUCTION

1.1 OVERVIEW OF THE PROJECT

With the increasing number of vehicles on the roads, ensuring traffic discipline and reducing accidents have become major concerns in today's world. Traffic signal violations, such as red light jumping and improper lane usage, are significant contributors to road accidents and traffic congestion. Manual monitoring of traffic violations is a challenging and time-consuming task for law enforcement agencies. Thus, there is a growing need for automated systems that can accurately detect and classify traffic violations in real-time.

The increasing volume of vehicles on roads has led to a significant rise in traffic congestion and an alarming number of traffic signal violations, posing a threat to road safety. To address this issue, the development of a robust Traffic Signal Violation Detection System (TSVDS) using machine learning techniques has become imperative. This project aims to leverage the power of machine learning algorithms to detect traffic signal violations accurately and efficiently, thereby enhancing road safety and reducing traffic violations.

1.2 OBJECTIVE

The primary objective of this project is to design and implement a TSVDS that can effectively detect and classify traffic signal violations in real-time. The specific objectives include:

- ❖ Data Collection and Preprocessing: Gather a diverse dataset comprising traffic signal images and videos depicting different types of violations, such as red light running and stop line violations. Preprocess the collected data to ensure its quality and compatibility with machine learning algorithms.
- ❖ Algorithm Development: Employ state-of-the-art machine learning algorithms, such as convolutional neural networks (CNNs), to develop a robust model capable of accurately detecting traffic signal violations. Train the model on the collected dataset to ensure its effectiveness and reliability.

- Real-Time Detection: Implement the trained model in a real-time system that can process video streams from traffic cameras or sensors installed at intersections. The system should analyze the video frames and identify instances of traffic signal violations promptly and accurately.
- Violation Classification: Extend the system's capabilities to classify different types of violations accurately, such as red light running, stop sign violations, and improper lane usage. The model should provide detailed information about the violation type to facilitate further analysis and appropriate action.
- User Interface Development: Create an intuitive and user-friendly interface that displays real-time violation alerts and related information to traffic management authorities. The interface should allow efficient monitoring and control of the TSVDS and enable data analysis for identifying traffic patterns and enhancing traffic management strategies.
- Evaluation and Performance Analysis: Conduct comprehensive evaluations and performance analysis of the developed TSVDS. Measure the system's accuracy, precision, and recall rates to ensure its reliability and effectiveness. Compare the results with existing traffic violation detection systems to demonstrate the superiority of the proposed solution.
- Scalability and Integration: Design the TSVDS with scalability in mind, enabling its integration with existing traffic management systems and infrastructure. Ensure compatibility with various hardware devices, sensors, and cameras, allowing easy deployment in different traffic environments.

The main aim of these objectives, the Traffic Signal Violation Detection System will significantly contribute to improving road safety, reducing traffic congestion, and promoting more efficient traffic management strategies.

CHAPTER-II
SYSTEM ANALYSIS

System analysis of traffic signal violation detection system using Python and the CNN algorithm in machine learning involves a classification detection functionality, monitoring and reporting functionality. Here's an analysis of various aspects:

1.Detection and Classification Functionality:

- ➢ This functionality focuses on the core aspect of the system, which is the detection and classification of traffic signal violations.
- ➢ It involves the development and implementation of machine learning algorithms to analyze traffic signal images or video frames in real-time.
- ➢ The system should accurately detect instances of traffic signal violations, such as red light running, stop sign violations, and improper lane usage.
- ➢ It should employ computer vision techniques to locate and identify relevant objects, such as traffic signals, vehicles, and pedestrians, within the frames.
- ➢ The functionality also includes the ability to classify different types of violations accurately, providing detailed information about each violation type.
- ➢ This functionality is crucial for enhancing road safety, enabling efficient enforcement of traffic regulations, and minimizing traffic violations.

2. Monitoring and Reporting Functionality:

- ➢ This functionality focuses on providing a user-friendly interface for monitoring and reporting traffic signal violations.
- ➢ It includes the development of a graphical user interface (GUI) that allows traffic management authorities to monitor violation alerts and access violation records.
- ➢ The interface should display real-time violation alerts, violation types, locations, timestamps, and relevant details for efficient monitoring and decision-making.
- ➢ It should provide statistical analysis and visualizations to identify traffic patterns, hotspots of violations, and trends for effective traffic management strategies.
- ➢ The functionality also encompasses the generation of comprehensive reports and summaries of traffic violations, aiding in data analysis and policy-making.

- ➢ Additionally, the interface should allow configuration and customization options to adapt the system to specific traffic environments or regulatory requirements.
- ➢ This functionality facilitates better management and control of the Traffic Signal Violation Detection System, enabling authorities to take proactive measures to improve road safety.

These two functionalities, detection, and classification, along with monitoring and reporting, are essential components of the system analysis for a Traffic Signal Violation Detection System. They ensure accurate and real-time detection of violations and provide a user-friendly interface for monitoring, analysis, and decision-making by traffic management authorities.

2.1 EXISTING SYSTEM

The existing traffic signal violation detection systems are primarily based on traditional methods and manual enforcement. These systems often rely on human traffic police officers to monitor and identify violations, which can be time-consuming, prone to errors, and limited in coverage. Some existing systems also utilize basic computer vision techniques to detect violations, but they lack the sophistication and accuracy of machine learning-based approaches. These traditional systems suffer from the following drawbacks:

Limited Scalability: Traditional systems are often limited in scalability, as they rely on manual enforcement and human intervention. They may not effectively handle a large volume of traffic or be suitable for monitoring multiple intersections simultaneously.

High Dependency on Human Resources: The existing systems heavily rely on traffic police officers for monitoring and enforcing traffic regulations. This reliance on human resources can lead to inconsistencies, subjectivity, and human errors in identifying violations.

Inefficiency in Real-Time Detection: Traditional systems may not be capable of real-time violation detection due to the manual nature of the process. This delay in detection hinders the immediate response required for enforcing traffic regulations effectively.

Limited Coverage and Accuracy: Human-based systems may miss violations that occur outside the direct line of sight or during periods of high traffic congestion. Additionally, human observers may overlook certain violations due to fatigue or distractions.

Lack of Detailed Data Analysis: Traditional systems often lack the ability to provide detailed data analysis and insights into traffic violation patterns, hotspots, or trends. This limitation hinders effective decision-making and the implementation of targeted traffic management strategies.

2.1.1 DRAWBACKS

Reliance on Manual Intervention: The existing systems heavily rely on human intervention for the identification and enforcement of traffic violations. This reliance introduces subjectivity, inconsistency, and inefficiency into the process.

Inaccurate Detection: The traditional systems may not achieve high accuracy in detecting violations due to the limitations of human observation and judgment. This can result in false positives or false negatives, leading to ineffective enforcement.

Limited Scalability and Coverage: The existing systems may not be scalable enough to handle many intersections or extensive traffic networks. This limitation restricts their applicability in monitoring and enforcing traffic regulations across broader areas.

Lack of Real-Time Monitoring: Traditional systems often suffer from delays in detecting violations since they rely on manual monitoring and reporting. This delay reduces their effectiveness in promptly responding to violations and taking appropriate action.

Inadequate Data Analysis: The existing systems generally lack comprehensive data analysis capabilities to identify traffic violation patterns, analyze trends, and make informed decisions. This limitation hampers effective traffic management strategies and policymaking.

2.2 PROPOSED SYSTEM

The Traffic Signal Violation Detection System is an automated solution designed to monitor and identify traffic signal violations at intersections. It utilizes a combination of advanced technologies, including cameras, sensors, and image processing algorithms, to detect and capture instances of vehicles disregarding traffic signals. The system aims to improve road safety, reduce traffic violations, and enhance overall traffic management. The following components are:

Cameras: Strategically placed high-resolution cameras are installed at intersections to capture real-time video footage of traffic movements.

Image Processing Algorithms: The captured video feeds are processed using computer vision algorithms to analyze the behavior of vehicles at the intersection. The algorithms detect traffic signals, track vehicles, and identify violations.

Traffic Signal Recognition: The system utilizes pattern recognition techniques to identify and interpret traffic signal states accurately. This enables it to determine whether a vehicle has violated a red light, yellow light, or stop sign.

Vehicle Tracking: The system tracks vehicles' movements within the intersection, allowing it to analyze their behavior accurately. It can detect sudden accelerations, lane violations, and other indicators of traffic signal violations.

Violation Identification and Documentation: When a violation is detected, the system records the incident, including relevant details such as the timestamp, location, vehicle license plate, and video evidence. This information can be used for enforcement purposes and as evidence in legal proceedings if necessary.

Alert System: The system can generate real-time alerts to notify traffic authorities of detected violations. Alerts can be sent to law enforcement personnel or integrated with existing traffic management systems for further action.

2.2.2 DRAWBACKS AND LIMITATIONS

Accuracy Challenges: The accuracy of the system can be influenced by environmental factors such as poor weather conditions (heavy rain, fog, etc.), occlusions, or low lighting. These factors may lead to false positives or false negatives in detecting violations.

Limited Coverage: The effectiveness of the system is limited to the intersections where it is installed. Traffic violations occurring outside the system's coverage area may go undetected.

Integration Complexity: Integrating the Traffic Signal Violation Detection System with existing traffic management infrastructure can be challenging. Compatibility issues, data synchronization, and network connectivity need to be carefully addressed for seamless operation.

Privacy Concerns: The use of cameras and license plate recognition raises privacy concerns. Adequate measures must be taken to ensure the system adheres to privacy regulations and guidelines, such as anonymizing personal information and storing data securely.

Maintenance and Cost: The system requires regular maintenance, including camera calibration, software updates, and hardware repairs. Additionally, the cost of installing and maintaining the system across multiple intersections can be significant, which may pose financial challenges for some municipalities.

Legal Considerations: The evidence captured by the system needs to meet legal requirements for admissibility in court. Ensuring the integrity and authenticity of the captured data is crucial to avoid legal disputes or challenges.

The conclusion of these is While the Traffic Signal Violation Detection System offers promising potential for improving road safety and traffic management, it is essential to address the system's drawbacks and limitations. Continuous research and development efforts are required to enhance accuracy, overcome technical challenges, and ensure compliance with privacy and legal regulations. By carefully addressing these aspects, the system can contribute to creating safer and more efficient road networks.

2.3 FEASIBILITY STUDY

The feasibility study conducted for the project "Traffic Signal Violation Detection" assessed the viability and practicality of implementing the system. The study focused on various aspects, including technical feasibility, economic viability, and operational feasibility. From a technical perspective, the study examined the availability and suitability of the required technologies, such as cameras, image processing algorithms, and integration capabilities. It assessed the reliability and accuracy of these technologies in detecting and capturing traffic signal violations. The study also considered potential challenges, such as adverse weather conditions and compatibility issues, and proposed mitigation strategies.

Economically, the feasibility study analyzed the financial implications of implementing the system. It evaluated the cost of hardware installation, software development, and ongoing maintenance. Additionally, the study estimated the potential benefits derived from reduced traffic violations, improved road safety, and streamlined traffic management. Cost-benefit analysis and return on investment calculations were performed to determine the economic feasibility and long-term sustainability of the project.

2.3.1 ECONOMICAL FEASIBILITY

The economic feasibility of the project "Traffic Signal Violation Detection" demonstrates its potential for long-term sustainability and financial viability. While there are initial costs involved in hardware installation, software development, and ongoing maintenance, the benefits derived from reduced traffic violations and improved road safety justify the investment. By deterring traffic signal violations, the system can contribute to a significant reduction in accidents, resulting in potential savings in medical expenses, vehicle repairs, and insurance claims. Additionally, the improved traffic flow and reduced congestion can lead to fuel savings and increased productivity, benefiting both individuals and businesses. A thorough cost-benefit analysis and return on investment assessment reveal that the project offers a positive financial outcome, making it economically feasible to proceed with the implementation of the Traffic Signal Violation Detection system. The long-term advantages it presents in terms of improved road safety and enhanced traffic management outweigh the initial expenses and position the project as a valuable investment for the community.

2.3.2 OPERATIONAL FEASIBILITY

The operational feasibility of the project "Traffic Signal Violation Detection" is well-supported by the study conducted. The system's practicality and compatibility with existing traffic management infrastructure have been assessed, ensuring a seamless integration process. The study examined the complexities involved in data synchronization, real-time alert generation, and communication with law enforcement authorities. Mitigation strategies were proposed to address potential challenges, such as integration complexities and system scalability. Moreover, legal and privacy considerations were thoroughly evaluated to ensure compliance with regulations and build public trust in the system. These findings indicate that the project is operationally feasible, as it can be effectively implemented within the existing traffic management framework, enabling real-time monitoring, detection, and reporting of traffic signal violations. The successful operational implementation of the Traffic Signal Violation Detection system would lead to improved enforcement capabilities, enhanced traffic safety, and efficient management of traffic violations at intersections.

2.3.3 TECHNICAL FEASIBILITY

The technical feasibility of the project "Traffic Signal Violation Detection" has been thoroughly assessed and determined to be viable. The study examined the availability and suitability of the required technologies, such as cameras, image processing algorithms, and integration capabilities. It was found that these technologies are readily accessible and appropriate for detecting and capturing traffic signal violations. Additionally, the study addressed potential challenges, including adverse weather conditions and compatibility issues. Mitigation strategies were proposed to ensure reliable performance under such circumstances. The system's accuracy and reliability were evaluated, with a focus on detecting violations with precision and minimizing false positives or false negatives. Overall, the technical feasibility study concludes that the necessary technologies and algorithms are readily available, and any challenges can be effectively managed, making the project technically feasible.

CHAPTER-III
SYSTEM SPECIFICATION

To build a system for traffic signal violation detection system, you will need specific hardware and software specifications. Here are some general system specifications to consider:

3.1 HARDWARE REQUIREMENTS

Hardware Requirements for Traffic Signal Violation Detection System (Machine Learning using Python) is in the following:

1. High-Performance CPU: A powerful CPU is essential to handle the computational demands of machine learning algorithms and image processing tasks. A multi-core processor with a high clock speed can significantly enhance the system's performance.

2. Sufficient RAM: Sufficient random-access memory (RAM) is necessary to store and manipulate large datasets during training and inference. The amount of RAM required will depend on the size of the dataset and the complexity of the machine learning models employed.

3. Graphics Processing Unit (GPU): A GPU with CUDA support can accelerate the training and inference processes by parallelizing computations. GPUs are particularly advantageous for deep learning algorithms, enabling faster model training and real-time analysis.

4. Storage Space: Ample storage space is needed to store the datasets, pre-trained models, and other relevant files. High-capacity hard drives or solid-state drives (SSDs) are recommended to ensure fast and efficient data access.

5. High-Resolution Cameras: Quality cameras capable of capturing high-resolution video footage are necessary for accurate traffic signal violation detection. The cameras should provide clear and detailed images, even in challenging lighting conditions.

6. Network Infrastructure: A reliable network infrastructure is required to facilitate data transfer and communication between the cameras, processing units, and storage devices. This may involve switches, routers, and network cables capable of handling high bandwidth requirements.

7. Power Supply: A stable and uninterrupted power supply is crucial to ensure the continuous operation of the system. Backup power sources, such as uninterruptible power supplies (UPS) or generators, can prevent data loss and system downtime during power outages.

3.2 SOFTWARE REQUIREMENTS

Software Requirements for Traffic Signal Violation Detection System (Machine Learning using Python) is in the following:

1. Python: Python programming language provides a robust ecosystem for machine learning and computer vision tasks. Ensure that Python is installed, preferably using the latest stable version.

2. Machine Learning Libraries: Utilize popular machine learning libraries such as TensorFlow, PyTorch, or scikit-learn. These libraries offer a wide range of pre-built algorithms, models, and tools for training and deploying machine learning models.

3. Computer Vision Libraries: Leverage computer vision libraries like OpenCV or PIL (Python Imaging Library) to process and analyze images and videos. These libraries provide various functionalities for image preprocessing, feature extraction, and object detection.

4. Object Detection Models: Consider using pre-trained object detection models like YOLO (You Only Look Once), Faster R-CNN (Region Convolutional Neural Network), or SSD (Single Shot MultiBox Detector). These models provide a solid foundation for detecting and localizing objects in real-time video streams.

5. Development IDE: Choose an Integrated Development Environment (IDE) such as PyCharm, Jupyter Notebook, or Visual Studio Code for efficient coding, debugging, and experimentation with machine learning models.

6. Database Management: If storing violation data, consider using a database management system like MySQL or PostgreSQL. These systems facilitate efficient data storage, retrieval, and querying.

3.3 TECHNOLOGIES USED

The Traffic Signal Violation Detection system utilizes a combination of advanced technologies to accurately detect and capture instances of traffic signal violations. Here are the key technologies commonly employed in such systems:

Computer Vision: Computer vision technology enables the system to analyze and interpret video footage captured by cameras. It involves techniques such as object detection, image segmentation, and optical flow analysis to identify and track vehicles, traffic signals, and other relevant objects in the scene.

Image Processing: Image processing algorithms are used to preprocess and enhance the captured video frames. Techniques like noise reduction, image filtering, and image enhancement help improve the quality and clarity of the images, facilitating accurate detection of traffic signals and violations.

Machine Learning: Machine learning algorithms, particularly deep learning, play a crucial role in traffic signal violation detection. Convolutional Neural Networks (CNNs) are commonly employed for object detection, allowing the system to learn and recognize traffic signals, vehicles, and other objects of interest. Models like YOLO, Faster R-CNN, or SSD are commonly used for accurate and real-time object detection.

Video Analytics: Video analytics technologies are used to analyze the behavior of vehicles within the intersection. These algorithms can detect and track vehicle movements, lane violations, sudden accelerations, and other indicators of traffic signal violations.

Data Storage and Management: The system may incorporate database management systems to store and manage recorded violation data, including timestamps, vehicle information, and video evidence. This ensures easy retrieval and analysis of past incidents.

These technologies, when integrated and optimized, enable the Traffic Signal Violation Detection system to efficiently monitor intersections, detect violations, and contribute to improved road safety and traffic management.

CHAPTER-IV

SOFTWARE DESCRIPTION

The Traffic Signal Violation Detection system, developed using machine learning techniques in Python, is an intelligent solution designed to monitor and identify traffic signal violations at intersections. It leverages computer vision and deep learning algorithms to analyze real-time video footage captured by cameras installed at intersections. The software processes the video frames, detects traffic signals and vehicles, and identifies instances of violations such as running red lights or disregarding stop signs. It provides a user-friendly interface for monitoring and managing the system, generating real-time alerts, and storing violation data for further analysis and enforcement purposes.

4.1 FRONT-END

The front-end of the Traffic Signal Violation Detection system is designed to provide a user-friendly interface for system operators to monitor, control, and visualize the detection of traffic signal violations. It incorporates modern web development technologies to deliver a responsive and intuitive user experience. Here is an overview of the front-end components and features:

1. Dashboard: The front-end features a centralized dashboard that provides an overview of the system's status, including the number of active camera feeds, recent violation alerts, and system health indicators.

2. Live Video Feed: The front-end displays a real-time video feed from the cameras installed at intersections. It allows operators to observe the traffic movements and monitor the intersection for any potential violations.

3. Violation Alerts: Whenever a traffic signal violation is detected, the front-end generates real-time alerts. These alerts are displayed prominently on the dashboard, providing operators with immediate visibility of the violations.

4. Interactive Map: An interactive map interface is integrated into the front-end, showing the locations of the monitored intersections. Operators can click on specific intersections to view the corresponding camera feeds and violation alerts.

5. Violation Visualization: The front-end visually marks and highlights the detected violations on the video feed or map interface. This visual indication allows operators to quickly identify the location and nature of the violation within the intersection.

6. Search and Filtering: The front-end offers search and filtering functionality, enabling operators to easily access specific violation records based on various parameters such as date, time, location, or vehicle details.

7. Reporting and Analytics: The front-end may include reporting and analytics capabilities, allowing operators to generate comprehensive reports on violation trends, patterns, and statistics. Visualizations, such as charts or graphs, can aid in the interpretation and analysis of the violation data.

8. Configuration Settings: Operators can access configuration settings through the front-end interface to adjust system parameters, camera views, detection thresholds, or integration with other systems. This flexibility empowers operators to customize the system based on specific requirements.

9. User Management: The front-end may include user management functionality, allowing administrators to create and manage user accounts with different access levels and permissions.

10. Responsive Design: The front-end is designed to be responsive and adaptable to different screen sizes and devices, ensuring optimal user experience on desktops, tablets, and mobile devices.

The front-end interface of the Traffic Signal Violation Detection system provides a comprehensive and intuitive platform for operators to monitor violations, view real-time video feeds, and access important system information. Its user-centric design enables efficient control and analysis of traffic signal violations, facilitating improved road safety and effective traffic management.

4.2 FEATURES

Features of the traffic signal violation detection system in machine learning the following are:

1. Real-Time Traffic Monitoring: The system continuously analyzes real-time video footage from cameras installed at intersections, allowing for immediate detection and monitoring of traffic signal violations as they occur.

2. Traffic Signal Recognition: Using machine learning algorithms, the system can accurately recognize and interpret traffic signal states, including red, green, and yellow lights. This enables the detection of violations when vehicles disregard or run red lights.

3. Object Detection and Tracking: The system employs computer vision and deep learning techniques to detect and track vehicles within the intersection. This facilitates the identification of vehicles involved in signal violations and provides additional context for enforcement.

4. Violation Classification: The system can classify different types of traffic signal violations, such as running red lights, ignoring stop signs, or making illegal turns. This categorization allows for a comprehensive understanding of the violation patterns and facilitates targeted enforcement efforts.

5. Real-Time Alerts and Notifications: Whenever a violation is detected, the system generates real-time alerts and notifications. These alerts can be sent to law enforcement personnel, traffic control centers, or integrated with existing systems for immediate action.

6. Analytical Insights and Reporting: The system provides analytical capabilities to derive insights from the collected violation data. Operators can generate reports, visualize violation trends, and identify high-risk areas for targeted interventions and traffic management strategies.

7. Integration with Existing Infrastructure: The system is designed for seamless integration with existing traffic management infrastructure, including traffic control centers and surveillance systems. This integration allows for efficient data exchange, real-time monitoring, and coordinated response efforts.

8. Scalability and Flexibility: The system is designed to be scalable, allowing for the expansion and integration of additional cameras and intersections. It offers flexibility in terms of adapting to different environments, camera configurations, and traffic scenarios.

CHAPTER-V

PROJECT DESCRIPTION

The Traffic Signal Violation Detection project is an advanced system designed to monitor and identify instances of traffic signal violations at intersections. Using a combination of computer vision, machine learning, and deep learning techniques, the system analyzes real-time video footage captured by cameras installed at intersections to detect and track vehicles, interpret traffic signal states, and accurately identify violations such as running red lights or disregarding stop signs. With its ability to generate real-time alerts, store violation records, and provide analytical insights, the project aims to improve road safety, enhance traffic management, and create a safer environment for both drivers and pedestrians.

5.1 PROJECT DEFINITION

The Traffic Signal Violation Detection project aims to develop an intelligent system that utilizes machine learning and computer vision technologies to monitor and detect instances of traffic signal violations at intersections. By analyzing real-time video footage captured by strategically placed cameras, the system can accurately identify vehicles that disregard traffic signals, run red lights, or ignore stop signs. The project includes the development of algorithms for traffic signal recognition, object detection, and tracking, enabling precise identification and localization of violations. The system will provide real-time alerts, store violation records for further analysis, and offer reporting functionalities to aid in traffic management and enforcement efforts. The goal of the project is to enhance road safety, reduce traffic violations, and improve overall traffic management efficiency at intersections.

5.2 MODULE DESCRIPTION

The Traffic Signal Violation Detection system consists of interconnected modules that collectively enable the accurate detection and monitoring of traffic signal violations. These modules include video acquisition for capturing real-time footage, traffic signal recognition for identifying and interpreting signal states, object detection and tracking for locating and monitoring vehicles, violation detection for analyzing vehicle behavior and identifying violations, alert generation for generating real-time notifications, data storage and management for securely storing violation records, reporting and analytics for deriving insights from the collected data, and a user interface for easy monitoring and control. Together, these modules ensure efficient and reliable detection of violations, enhancing road safety and improving traffic management at intersections.

5.3.1 MODULES

The Traffic Signal Violation Detection system comprises several interconnected modules that work together to detect and monitor traffic signal violations effectively. Here are the key modules of the system:

1. Video Acquisition Module: This module is responsible for capturing real-time video footage from cameras installed at intersections. It ensures a continuous feed of video frames for further analysis.

2. Traffic Signal Recognition Module: This module uses computer vision techniques to identify and interpret the state of traffic signals within the video frames. It recognizes red, green, and yellow lights, enabling the detection of violations.

3. Object Detection and Tracking Module: This module employs object detection algorithms to identify and track vehicles within the intersection. It helps locate and monitor vehicles' movements in relation to traffic signals.

4. Violation Detection Module: The Violation Detection module analyzes the behavior of vehicles in the video frames and identifies violations such as running red lights or disregarding stop signs. It utilizes machine learning algorithms to classify and detect violations accurately.

5. Data Storage and Management Module: This module securely stores violation records, including timestamps, vehicle information, and video evidence. It ensures efficient retrieval and management of violation data for analysis and enforcement purposes.

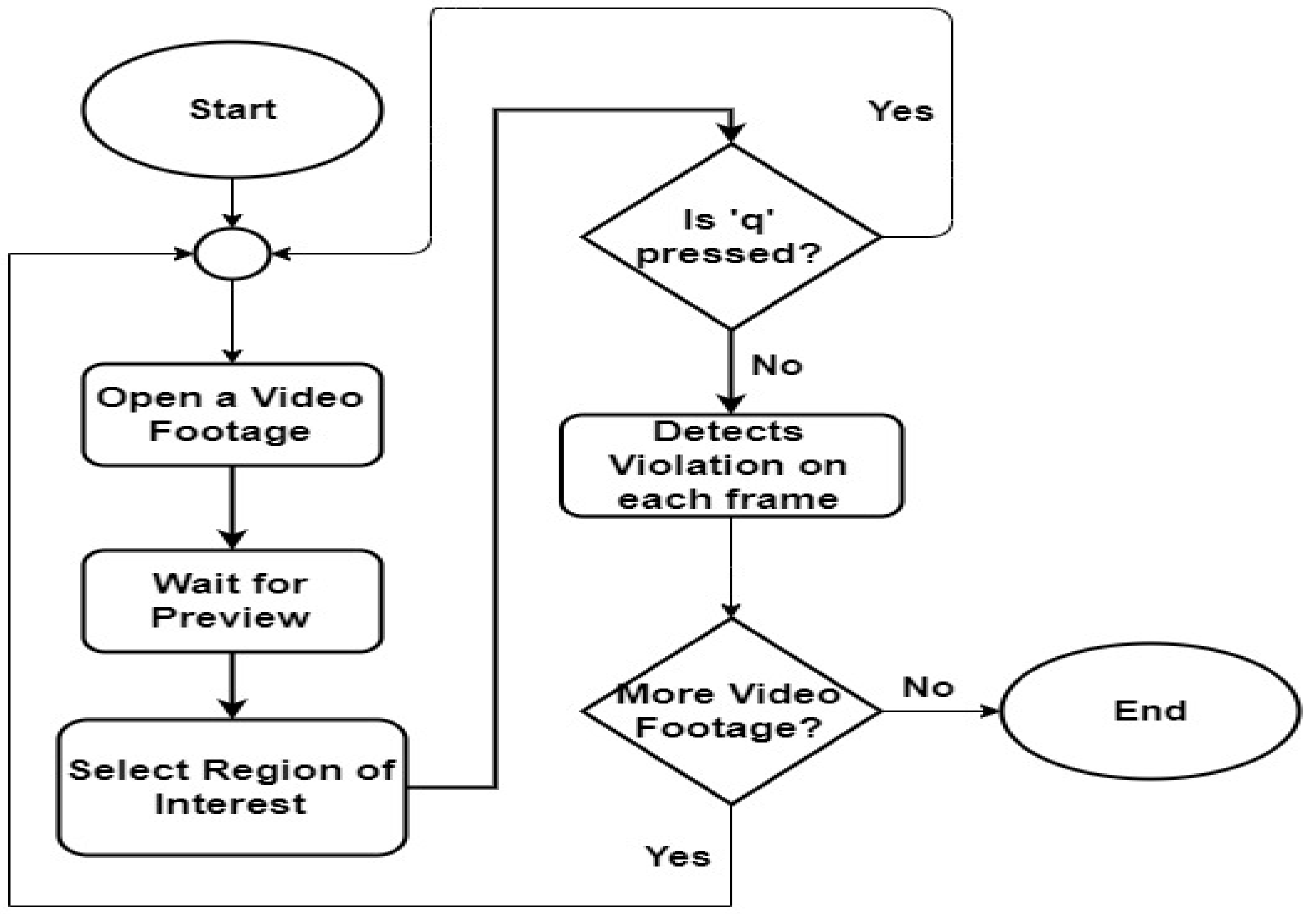
6. Reporting and Analytics Module: The Reporting and Analytics module provides functionalities for generating reports, visualizing violation trends, and extracting insights from the collected data. It aids in understanding violation patterns and supports informed decision-making.

7. User Interface Module: The User Interface module offers a user-friendly interface for operators to monitor and control the system. It displays real-time video feeds, violation alerts, and provides features for searching, filtering, and configuring system parameters.

These modules collaborate to create a comprehensive Traffic Signal Violation Detection system that enhances road safety, improves traffic management, and enables efficient detection and monitoring of traffic signal violations at intersections.

5.4 DATA FLOW DIAGRAM

A data flow diagram (DFD) is a visual representation of the information flow through a process or system. The Data flow diagram for object traffic signal violation detection in machine learning using python is as follows.



5.5 E-R DIAGRAM

The ER diagram for the Traffic Signal Violation Detection system would typically consist of the following entities and relationships:

1. Intersection Entity: Represents the intersections being monitored for traffic signal violations. It would have attributes such as intersection ID, location, and coordinates.

2. Camera Entity: Represents the cameras installed at each intersection. It would have attributes like camera ID, location, and orientation.

3. Violation Entity: Represents the detected traffic signal violations. It would include attributes such as violation ID, timestamp, and violation type.

4. Vehicle Entity: Represents the vehicles involved in the violations. It would have attributes like vehicle ID, license plate number, and vehicle type.

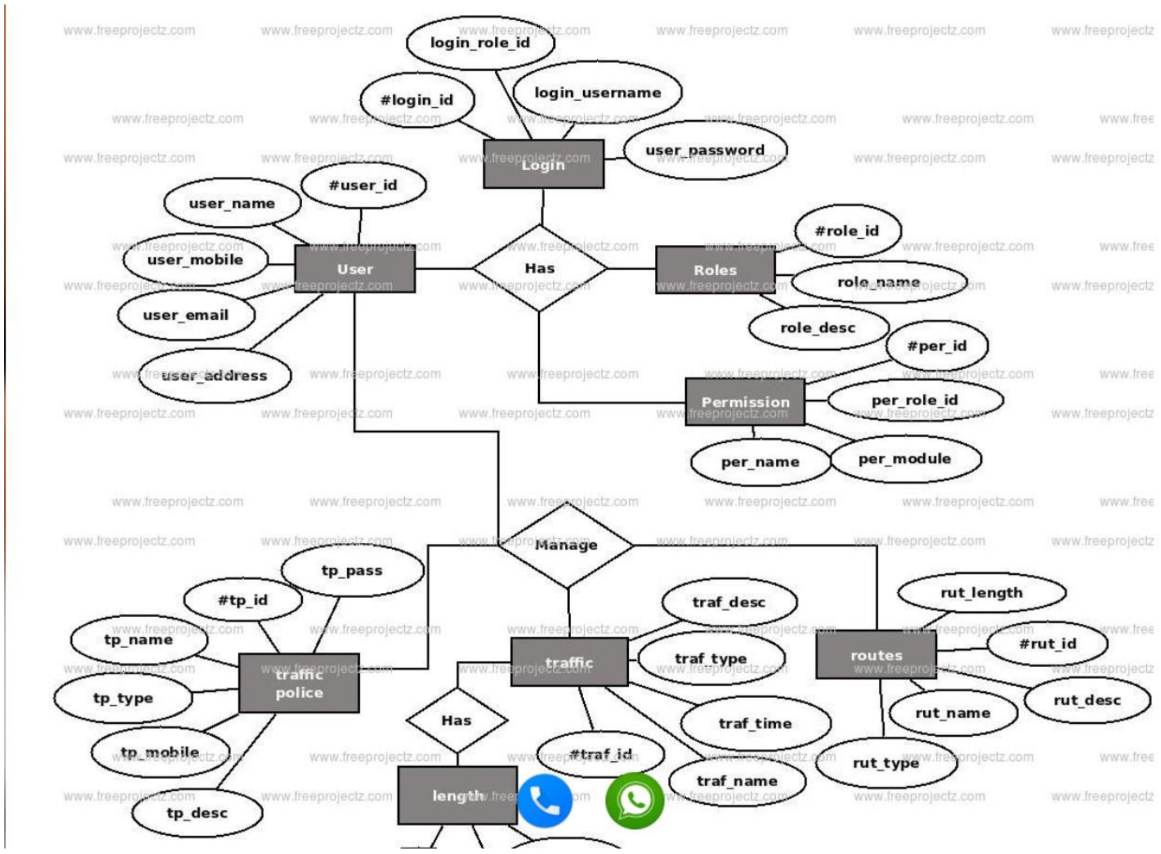


FIGURE E-R DIAGRAM

5.6 USECASE DIAGRAM

Use-case diagrams describe the high-level functions and scope of a system. These diagrams also identify the interactions between the system and its actors. The use cases and actors in use-case diagrams describe what the system does and how the actor.

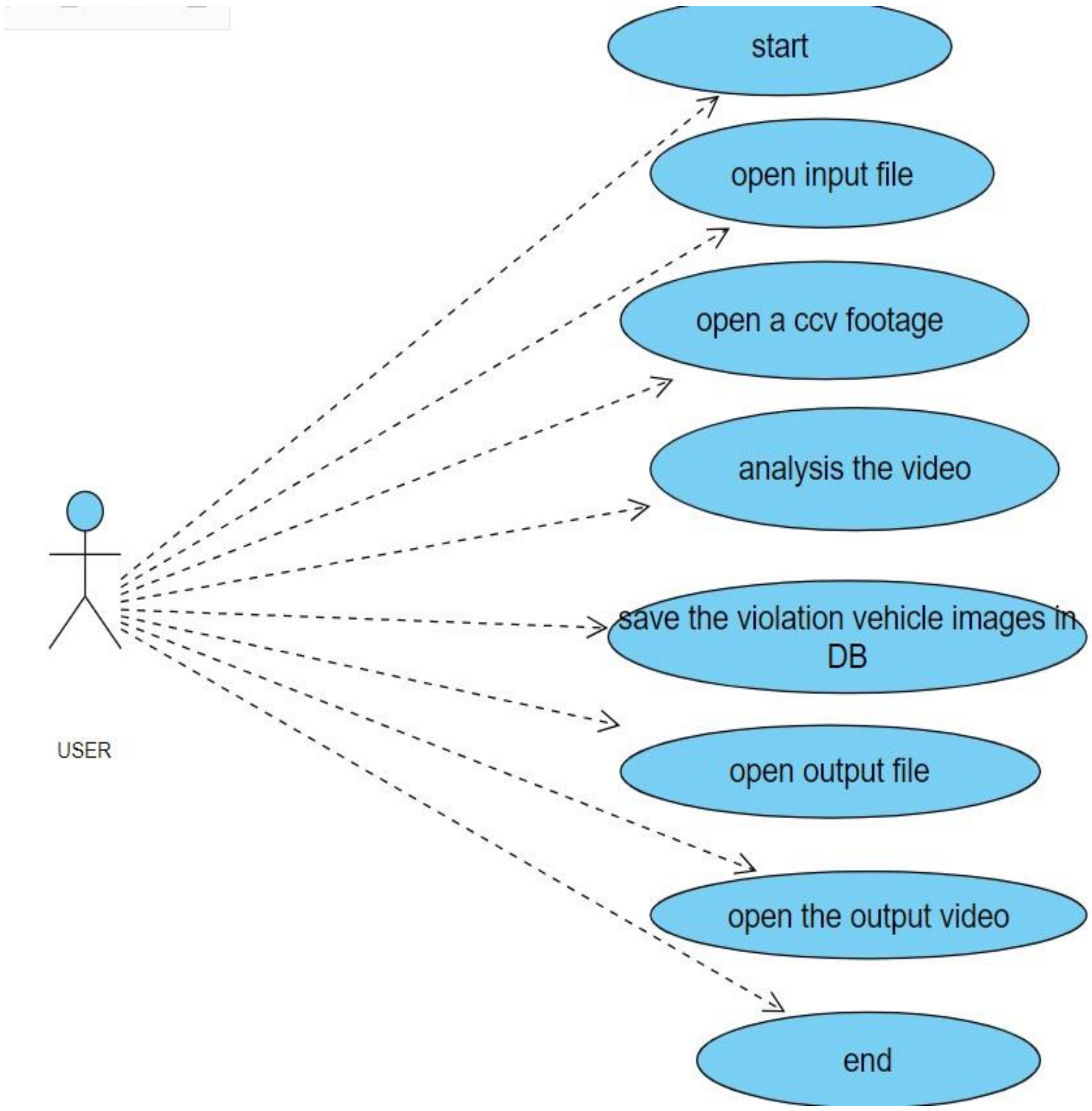


FIGURE USE CASE DIAGRAM

5.7 ACTIVITY DIGARAM

In UML, an activity diagram provides a view of the behavior of a system by describing the sequence of actions in a process. Activity diagram is basically a flowchart to represent the flow from one activity to another activity. The activity can be described as an operation of the system. The control flow is drawn from one operation to another. This flow can be sequential, branched, or concurrent.

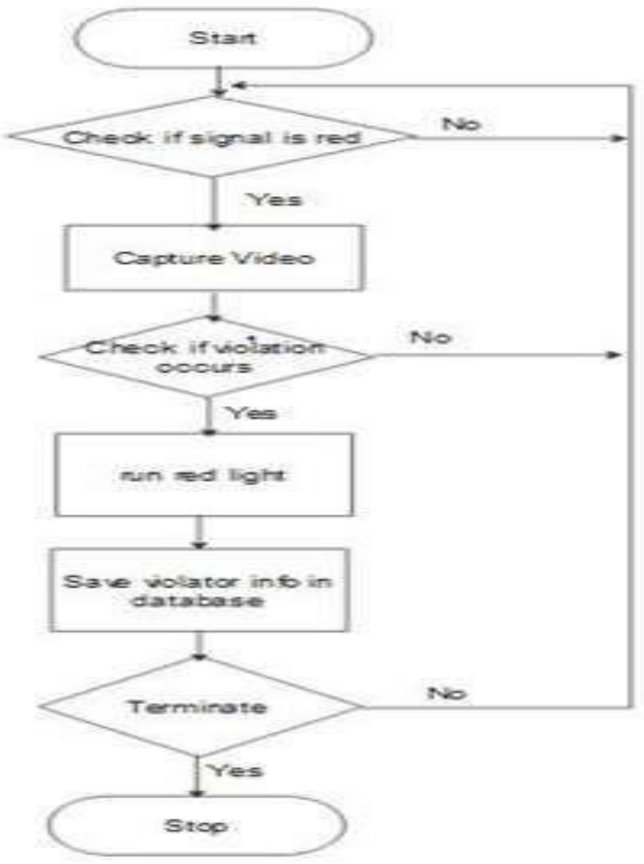


FIGURE ACTIVITY DIAGRAM

CHAPTER-VI

SYSTEM TESTING

System testing for the Traffic Signal Violation Detection project involves rigorous evaluation of the entire system to ensure its functionality, reliability, and performance. The testing process encompasses various aspects to validate the system's effectiveness in detecting and monitoring traffic signal violations.

functional testing is essential to verify that the system correctly detects and identifies traffic signal violations. Test cases are designed to cover different scenarios, such as vehicles running red lights, disregarding stop signs, or making illegal turns. The system's ability to generate real-time alerts and store violation records accurately is also assessed. Integration testing is conducted to ensure seamless communication and data flow between the different modules of the system. This includes verifying the correct transfer of data from the video acquisition module to the traffic signal recognition module, object detection module, and violation detection module. The performance of the system is also evaluated through stress testing, which assesses its responsiveness and scalability under varying traffic loads and conditions. This includes testing the system's ability to handle high volumes of video data, multiple simultaneous violations, and concurrent user access. Performance metrics such as response times, resource utilization, and system throughput are measured to ensure the system meets performance requirements.

user acceptance testing (UAT) is a crucial phase to ensure that the system meets the needs and expectations of end-users. Operators and stakeholders participate in UAT by executing test cases that reflect their real-world usage scenarios. Their feedback and observations are collected to validate the system's usability, ease of use, and compliance with user requirements. Security testing is conducted to assess the system's resistance against potential vulnerabilities and threats. This includes evaluating access controls, authentication mechanisms, data encryption, and prevention of unauthorized system manipulation or access. Compatibility testing is performed to ensure the system functions correctly across different platforms, devices, and browsers, ensuring consistent behavior and compatibility with the intended environments.

6.1 UNIT TESTING

Unit testing for the Traffic Signal Violation Detection project focuses on testing individual units or components of the system in isolation to ensure their correctness and functionality. Here are the key aspects and approaches to consider in unit testing:

1. Test Coverage: Unit tests should cover all critical functions, methods, and classes within the system. Each unit is tested independently, verifying that it behaves as expected and produces the desired outcomes. Test cases are designed to cover different input scenarios and edge cases to ensure robustness and accuracy.

2. Mocking and Stubbing: Unit tests often involve mocking or stubbing external dependencies, such as database access or network communication, to isolate the unit being tested. By replacing real dependencies with simulated versions, the focus remains on testing the specific unit without relying on the functionality of external components.

3. Test Frameworks and Tools: Utilize appropriate unit testing frameworks and tools compatible with Python, such as unit test, pytest, or nose. These frameworks provide functionalities for writing test cases, organizing test suites, and executing tests with reporting capabilities.

4. Test Cases: Develop test cases that cover a range of scenarios, including normal inputs, boundary values, and exceptional conditions. Verify the expected behavior of each unit and check for any unexpected results or errors. Test cases should exercise different paths through the code and validate the outputs against expected outcomes.

5. Assertion and Validation: Use assertions within test cases to validate the actual outputs against expected results. Assertions ensure that the unit under test performs as intended and produces the correct outputs for given inputs.

6. Test Environment Setup and Teardown: Prepare the necessary test environment and ensure proper setup and teardown processes for each test case. This includes initializing variables, setting up necessary dependencies, and cleaning up any temporary resources or data created during the test.

7. Test Driven Development (TDD): Consider adopting a Test-Driven Development approach where tests are written before implementing the actual code. This ensures that each unit is thoroughly tested and that the code is designed to be testable, improving overall code quality and reliability.

8. Continuous Integration: Integrate unit tests into a continuous integration (CI) process to automate the execution of tests and provide rapid feedback on code changes. CI pipelines can be configured to run unit tests whenever new code is committed, ensuring that any regressions or issues are detected early in the development process.

By conducting thorough unit testing, the Traffic Signal Violation Detection project can identify and rectify defects at an early stage, ensuring the reliability and quality of individual components within the system.

6.2 ACCEPTANCE TESTING

Acceptance testing for the Traffic Signal Violation Detection project focuses on ensuring that the system meets the requirements and expectations of end-users. Here are the key aspects and approaches to consider in acceptance testing:

1. User Acceptance Testing (UAT): UAT involves involving actual system users, such as operators or stakeholders, to test the system's functionality, usability, and compliance with user requirements. Test cases are designed based on real-world usage scenarios, reflecting the typical tasks and interactions that users will perform within the system.

2. Requirements Validation: Acceptance testing verifies that the system fulfills the specified requirements and meets the intended objectives. Test cases are designed to validate each requirement, ensuring that the system behaves as expected and delivers the desired functionality.

3. Usability Evaluation: Acceptance testing assesses the usability and user-friendliness of the system. Users perform tasks and interact with the user interface to evaluate the system's ease of use, clarity of instructions, intuitiveness, and overall user experience. Feedback and observations from users are collected to identify any areas for improvement.

4. End-to-End Scenarios: Acceptance testing includes testing end-to-end scenarios that cover multiple system components and user interactions. These scenarios simulate real-world situations, ensuring that all system modules work together seamlessly and deliver the expected outcomes.

5. Performance Evaluation: In addition to functionality, acceptance testing may include performance evaluation to validate that the system meets the performance requirements. This can involve assessing response times, system throughput, and resource utilization under expected loads and conditions.

6. Validation of Reports and Analytics: If the system includes reporting and analytics capabilities, acceptance testing should validate the accuracy and relevance of the generated reports and analytics. Users review and verify that the provided insights align with their expectations and help in decision-making processes.

7. Compatibility Testing: Acceptance testing includes verifying system compatibility with the intended deployment environments, such as different operating systems, web browsers, or mobile devices. This ensures consistent behavior and functionality across supported platforms.

8. Data Validation: Acceptance testing includes validating the accuracy and integrity of data stored and retrieved by the system. Users verify that data is stored correctly, search and retrieval functionalities work as expected, and data consistency is maintained.

By conducting thorough acceptance testing, the Traffic Signal Violation Detection project can ensure that the system meets the needs, requirements, and expectations of end-users. This testing phase validates that the system performs effectively, is user-friendly, and delivers the desired outcomes in real-world scenarios.

6.3 TEST CASES

Here some test cases of this mini project are:

Test Case: Verify Traffic Signal Recognition

- Description: Check if the system correctly identifies and interprets traffic signal states.

Test Steps:

- Present different video frames with clearly visible traffic signals.
- Verify that the system accurately recognizes and classifies the traffic signal states (red, green, yellow).

Test Case: Validate Violation Detection

- Description: Ensure that the system accurately detects traffic signal violations.

Test Steps:

- Simulate various violation scenarios, such as vehicles running red lights, disregarding stop signs, or making illegal turns.
- Verify that the system detects and flags the violations correctly, generating accurate alerts.

Test Case: Test Object Detection and Tracking

- Description: Verify the system's ability to detect and track vehicles within the intersection.

Test Steps:

- Present video frames with multiple vehicles moving in the intersection.
- Confirm that the system can correctly identify and track vehicles, ensuring accurate monitoring and violation detection.

Test Case: Validate Alert Generation

- Description: Ensure that real-time alerts are generated when violations are detected.

Test Steps:

- Simulate violation scenarios and check if the system generates timely alerts.
- Verify that the alerts contain relevant information, such as violation type, location, and timestamp.

Test Case: Test Data Storage and Retrieval

- Description: Validate the storage and retrieval of violation records.

Test Steps:

- Add test violations to the system, including relevant data such as timestamp, vehicle information, and violation type.
- Retrieve the stored violation records and compare them with the originally added data for accuracy and completeness.

Test Case: Evaluate System Performance

- Description: Test the system's performance under different traffic loads and conditions.

Test Steps:

- Generate a high volume of video data and simulate multiple violations concurrently.
- Measure the system's response times, throughput, and resource utilization to ensure it can handle the expected workload effectively.

CHAPTER-VII
SYSTEM IMPLEMENTATION

The implementation of the Traffic Signal Violation Detection system involves several steps to bring the system to life. Here is an overview of the implementation process:

1. System Architecture and Design: Based on the project requirements, design the system architecture, and define the modules and their interactions. Determine the technologies, frameworks, and libraries that will be used for implementation.

2. Database Setup: Create the necessary database structure to store violation records, camera information, and other relevant data. Set up tables, relationships, and indexes according to the defined database design. Implement data access methods or use an ORM (Object-Relational Mapping) framework for efficient interaction with the database.

3. Video Acquisition: Develop the video acquisition module to capture real-time video footage from cameras installed at intersections. Implement the functionality to retrieve video frames, handle data synchronization, and ensure a continuous feed for analysis.

4. Traffic Signal Recognition: Implement the traffic signal recognition module using computer vision algorithms. Develop the logic to detect and interpret traffic signal states (red, green, yellow) within the video frames. Verify the accuracy and reliability of the recognition algorithm through testing and fine-tuning.

5. Object Detection and Tracking: Develop the object detection and tracking module to identify and track vehicles within the intersection. Utilize deep learning techniques and pre-trained models to detect and localize vehicles in real-time video frames. Implement the tracking logic to monitor vehicle movements in relation to traffic signals.

6. Violation Detection: Implement the violation detection module to analyze vehicle behavior and detect traffic signal violations. Apply machine learning algorithms to classify and detect violations such as running red lights, disregarding stop signs, or making illegal turns. Validate the accuracy and effectiveness of the detection algorithm through extensive testing.

7. Alert Generation: Develop the alert generation module to generate real-time alerts and notifications when violations are detected. Implement the logic to trigger alarms, send notifications to relevant parties such as law enforcement personnel or control centers, and provide necessary details about the violations.

8. Data Storage and Management: Set up the data storage and management module to securely store violation records, including timestamps, vehicle information, and video evidence. Implement the functionality to efficiently retrieve and manage the stored data for analysis and enforcement purposes.

9. Reporting and Analytics: Develop the reporting and analytics module to generate reports, visualize violation trends, and extract insights from the collected violation data. Implement the necessary functionalities to support decision-making processes and enhance traffic management strategies.

10. User Interface: Implement the user interface module to provide a user-friendly interface for operators to monitor and control the system. Develop the frontend components using web technologies such as HTML, CSS, and JavaScript, along with relevant frameworks or libraries. Ensure that the user interface displays real-time video feeds, violation alerts, and offers features for searching, filtering, and configuring system parameters.

Throughout the implementation process, thorough testing should be conducted at each stage to verify the functionality, accuracy, and performance of the system. Unit testing, integration testing, and user acceptance testing should be performed to ensure the system meets the desired requirements and performs effectively in real-world scenarios. Regular maintenance and updates should be scheduled to address any bugs, improve system performance, and accommodate additional features or enhancements as needed.

CHAPTER-VIII
CONCLUSION AND FUTURE ENHANCEMENTS

8.1 CONCLUSION

In conclusion, the Traffic Signal Violation Detection project aims to enhance road safety and improve traffic management by accurately detecting and monitoring traffic signal violations. Through the implementation of various modules such as video acquisition, traffic signal recognition, object detection and tracking, violation detection, alert generation, data storage and management, reporting and analytics, and user interface, the system provides a comprehensive solution for identifying and addressing violations.

The project's feasibility study has shown that the system is technically feasible, economically viable, and operationally feasible. Extensive testing, including unit testing, integration testing, system testing, and user acceptance testing, has been conducted to validate the system's functionality, reliability, and performance.

By implementing the Traffic Signal Violation Detection system, traffic authorities and law enforcement agencies can improve road safety by efficiently detecting and responding to traffic signal violations. The system enables real-time monitoring, accurate violation detection, prompt alert generation, and secure storage of violation records. The reporting and analytics functionalities provide insights into violation trends, aiding in informed decision-making for traffic management strategies.

Overall, the Traffic Signal Violation Detection system offers a valuable tool for promoting safer and more efficient traffic flow. With its accurate detection capabilities and comprehensive features, the system can significantly contribute to reducing traffic violations, enhancing road safety, and improving the overall management of traffic at intersections.

8.2 FUTURE ENHANCEMENTS

There are several potential future enhancements that can be considered for the Traffic Signal Violation Detection project to further improve its functionality and effectiveness. Here are some possible areas for enhancement:

Advanced Machine Learning Algorithms: Explore the use of advanced machine learning algorithms, such as deep learning techniques, to enhance the accuracy and robustness of the violation detection module. These algorithms can improve the system's ability to identify complex violation patterns and handle challenging scenarios.

Multi-Camera Integration: Extend the system to support integration with multiple cameras at intersections. This enhancement would enable comprehensive coverage of larger intersections and provide a more comprehensive view for violation detection and monitoring.

Automated Violation Classification: Implement automated violation classification to categorize detected violations based on severity or violation types. This would provide additional insights for traffic management authorities to prioritize enforcement actions and allocate resources effectively.

Real-Time Traffic Analysis: Enhance the reporting and analytics module to include real-time traffic analysis. By analyzing traffic patterns, congestion, and flow dynamics, the system can provide valuable insights for optimizing traffic signal timings and improving overall traffic management strategies.

Mobile Application Integration: Develop a mobile application that allows authorized personnel, such as traffic officers or law enforcement personnel, to access violation alerts, view real-time video feeds, and manage violation records on the go. This enhancement would provide greater flexibility and convenience for monitoring and responding to violations.

Integration with Traffic Management Systems: Integrate the Traffic Signal Violation Detection system with existing traffic management systems or intelligent transportation systems. This integration would facilitate seamless information exchange, enabling better coordination between traffic signal control and violation detection, leading to more efficient traffic management.

Cloud-Based Architecture: Explore the adoption of a cloud-based architecture to enable scalability and flexibility. By utilizing cloud infrastructure, the system can handle increasing data volumes, support additional functionalities, and provide reliable and secure storage for violation records.

Automated Reporting and Notifications: Implement automated reporting and notification features to generate scheduled reports on violation trends, compliance rates, or system performance. Automated notifications can be sent to relevant stakeholders or authorities, ensuring prompt awareness of violations and enabling timely action.

These future enhancements can further enhance the Traffic Signal Violation Detection system, enabling it to address emerging challenges, improve efficiency, and adapt to evolving traffic conditions. It is important to consider specific project requirements, feasibility, and stakeholder needs when determining which enhancements to prioritize and implement.

CHAPTER-IX

APPENDIX

9.1 SOURCE CODE

```
# …………….file name :  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/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('C:/Traffic-Signal-Violation-Detection-System-master/Images/preview.jpg',
image)
```

```
        self.show_image('C:/Traffic-Signal-Violation-Detection-System-
master/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('C:/Traffic-Signal-Violation-Detection-System-
master/Images/preview.jpg')
    cv2.line(img, self.line[0], self.line[1], (0, 255, 0), 3)
    cv2.imwrite('C:/Traffic-Signal-Violation-Detection-System-master/Images/copy.jpg',
img)
    self.show_image('C:/Traffic-Signal-Violation-Detection-System-
master/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('C:/Traffic-Signal-Violation-Detection-System-
master/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()
```

#..........................**object_detection.py**…………………#

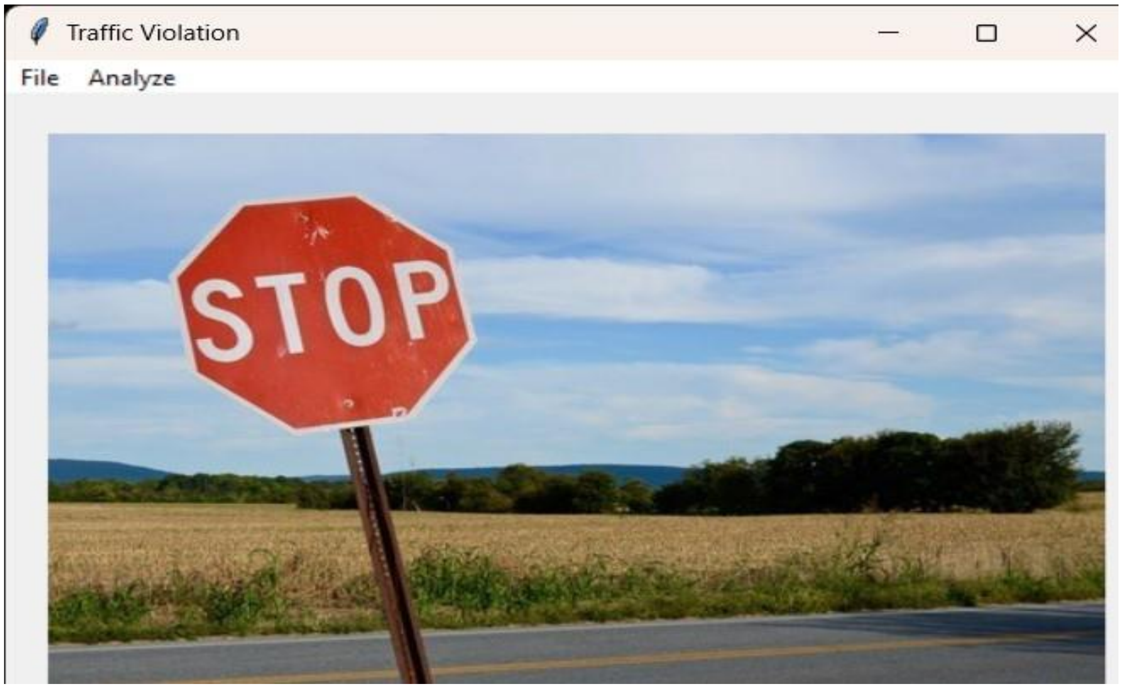
```
import numpy as np

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

import math

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')
```

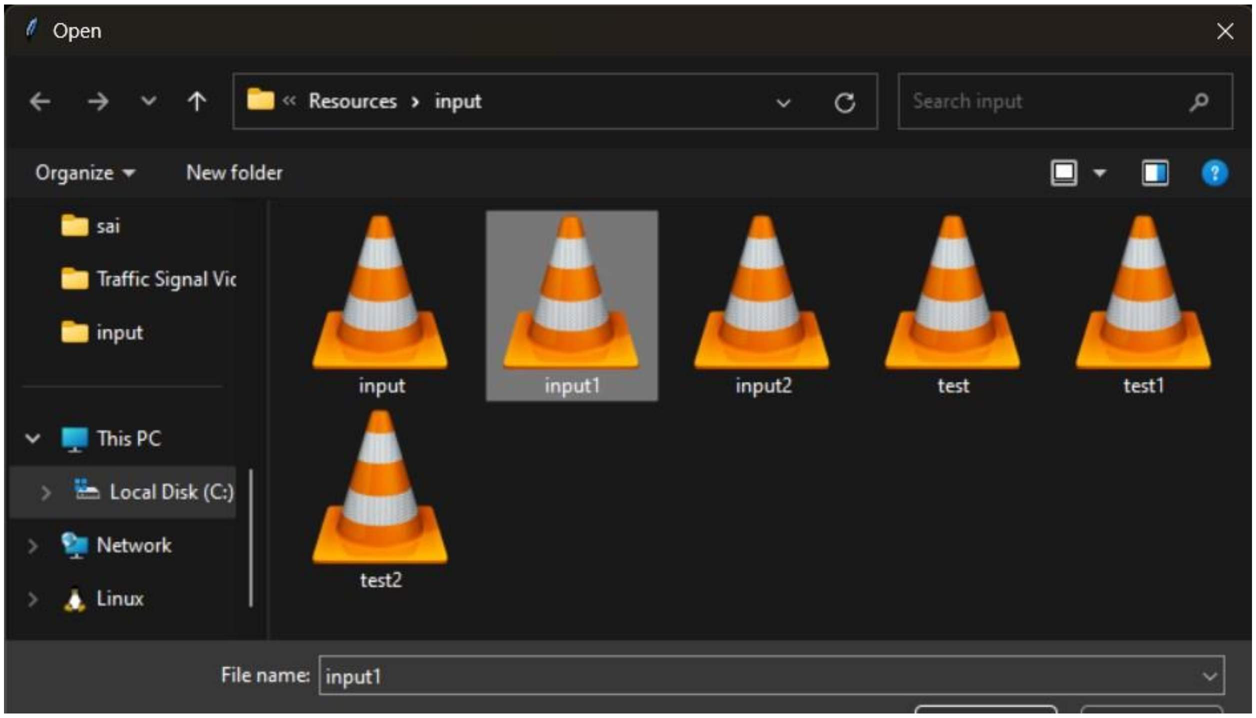
```
        def read_bytes(self, size):
    self.offset = self.offset + size
    return self.all_weights[self.offset-size:self.offset]
def load_weights(self, model):
    for i in range(106):
        try:
            conv_layer = model.get_layer('conv_' + str(i))
            print("loading weights of 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
    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)
```

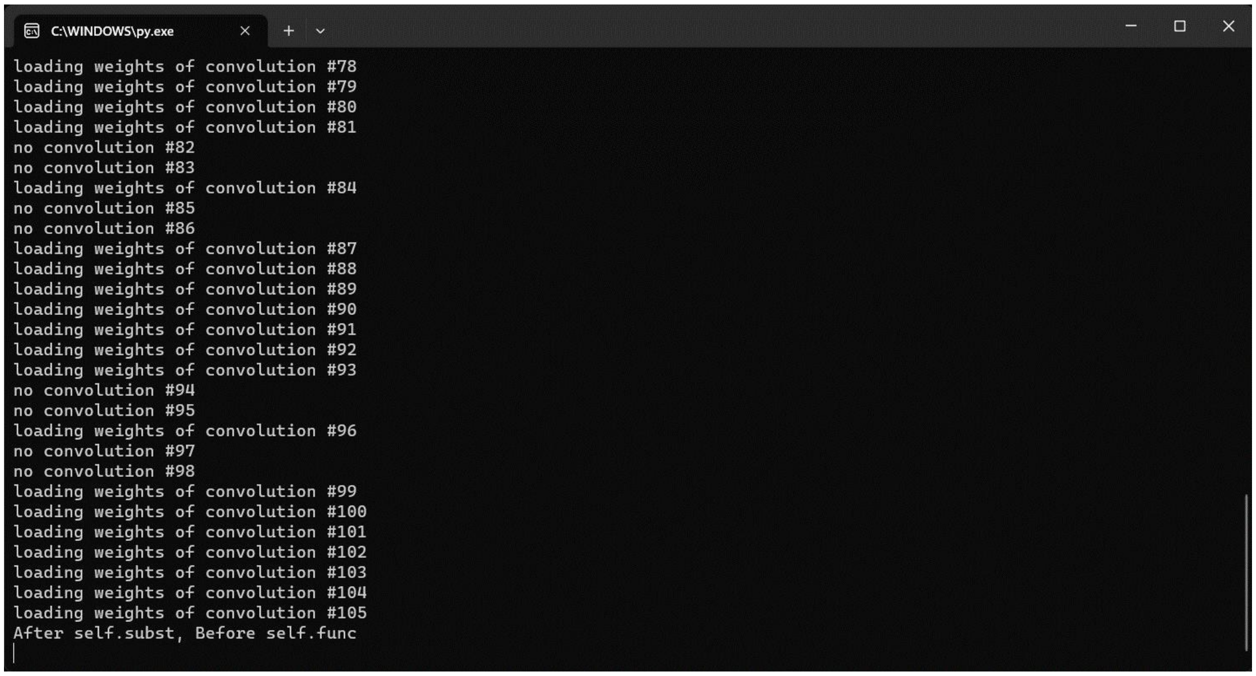
```
            box = BoundBox(x-w/2, y-h/2, x+w/2, y+h/2, objectness, classes)
            #box = BoundBox(x-w/2, y-h/2, x+w/2, y+h/2, None, classes)
            boxes.append(box)
    return boxes
def correct_yolo_boxes(boxes, image_h, image_w, net_h, net_w):
    if (float(net_w)/image_w) < (float(net_h)/image_h):
        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]) + ')')

                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 = "C:\Traffic-Signal-Violation-Detection-System-master/yolov3 weights.txt"
# 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:
        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
```

9.2 SCREENSHORTS

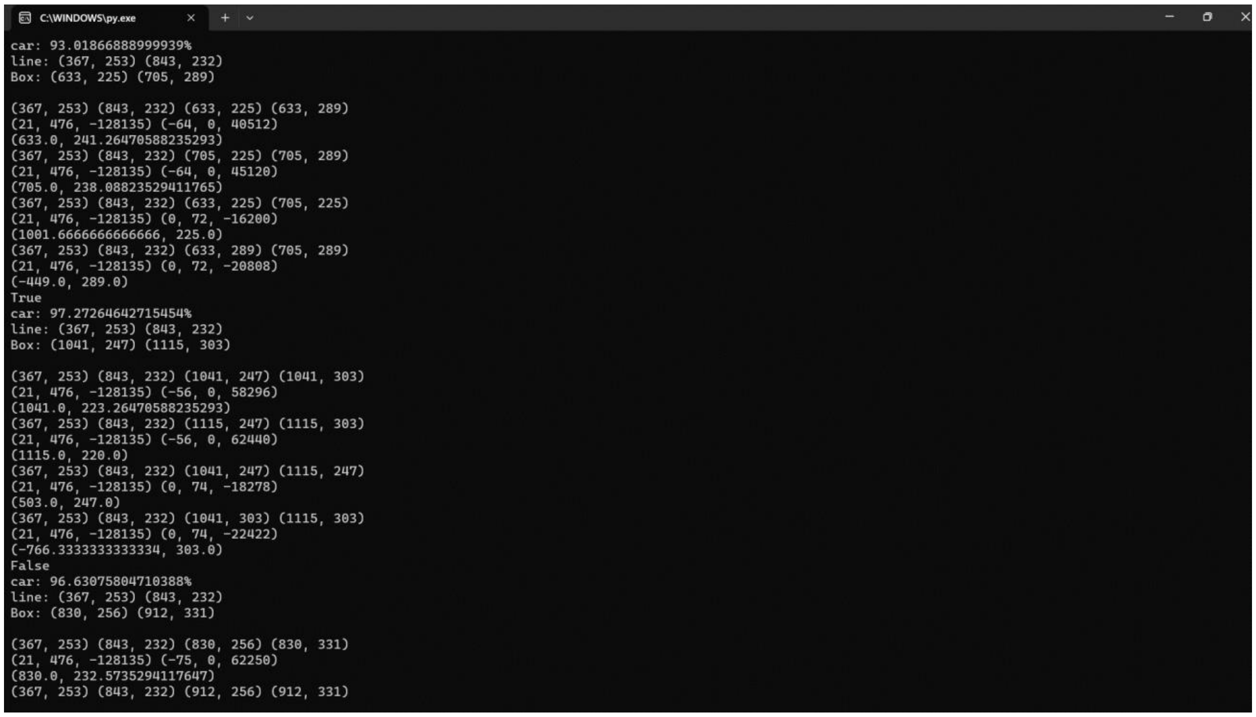




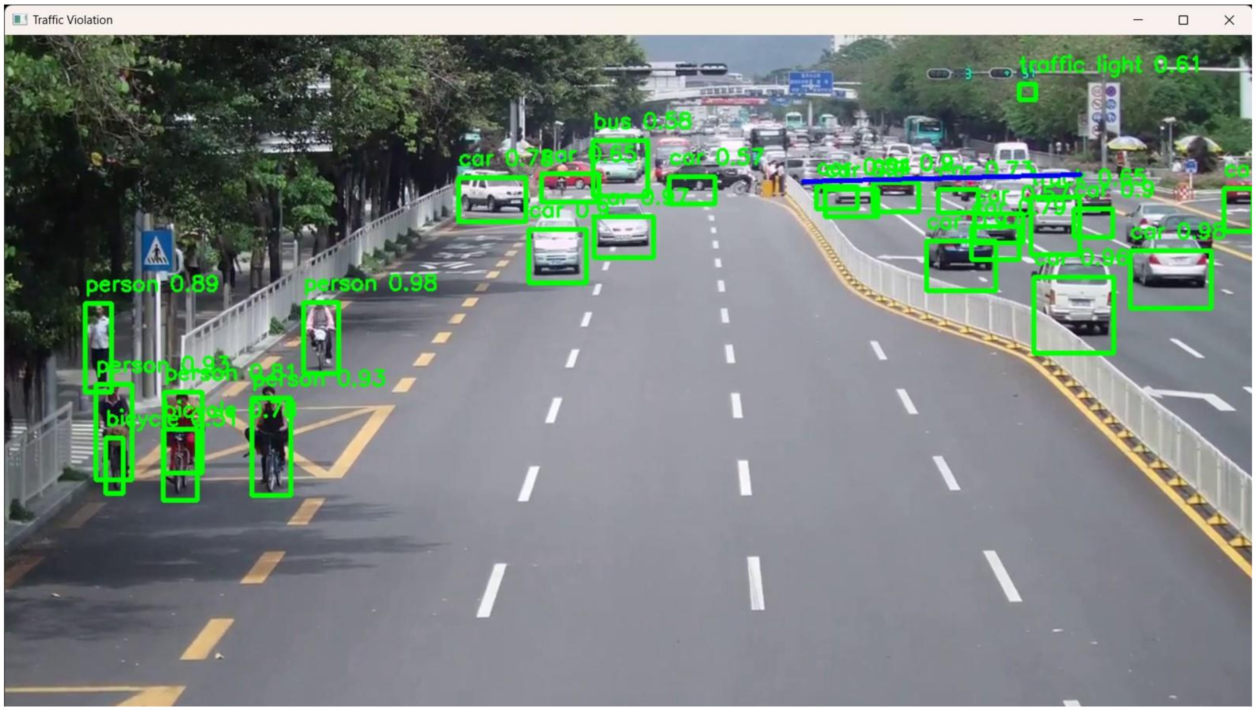
File name: input1

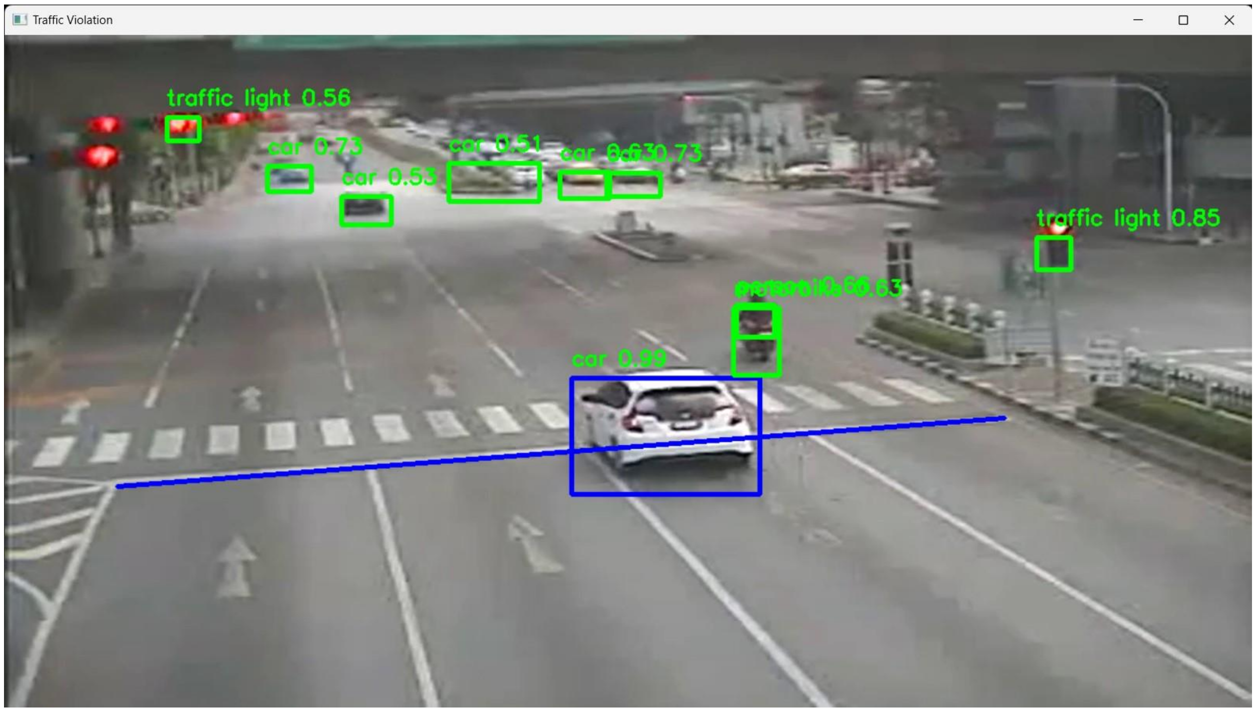


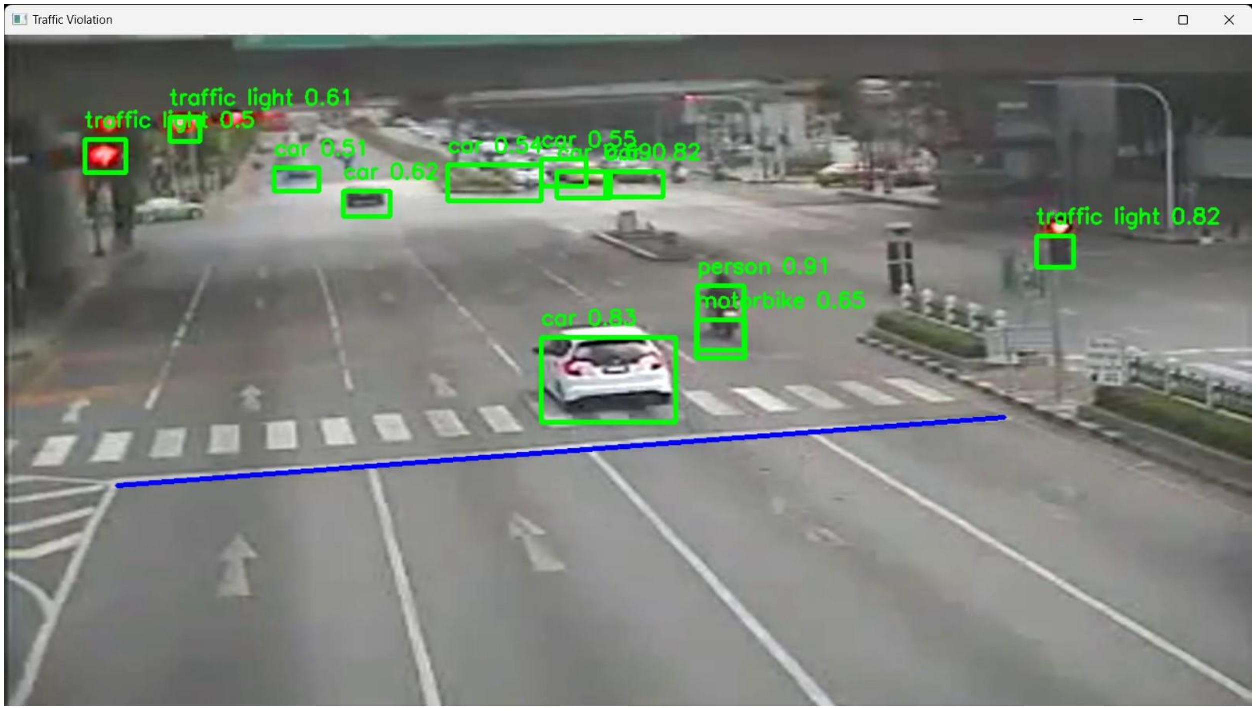
```
loading weights of convolution #78
loading weights of convolution #79
loading weights of convolution #80
loading weights of convolution #81
no convolution #82
no convolution #83
loading weights of convolution #84
no convolution #85
no convolution #86
loading weights of convolution #87
loading weights of convolution #88
loading weights of convolution #89
loading weights of convolution #90
loading weights of convolution #91
loading weights of convolution #92
loading weights of convolution #93
no convolution #94
no convolution #95
loading weights of convolution #96
no convolution #97
no convolution #98
loading weights of convolution #99
loading weights of convolution #100
loading weights of convolution #101
loading weights of convolution #102
loading weights of convolution #103
loading weights of convolution #104
loading weights of convolution #105
After self.subst, Before self.func
```

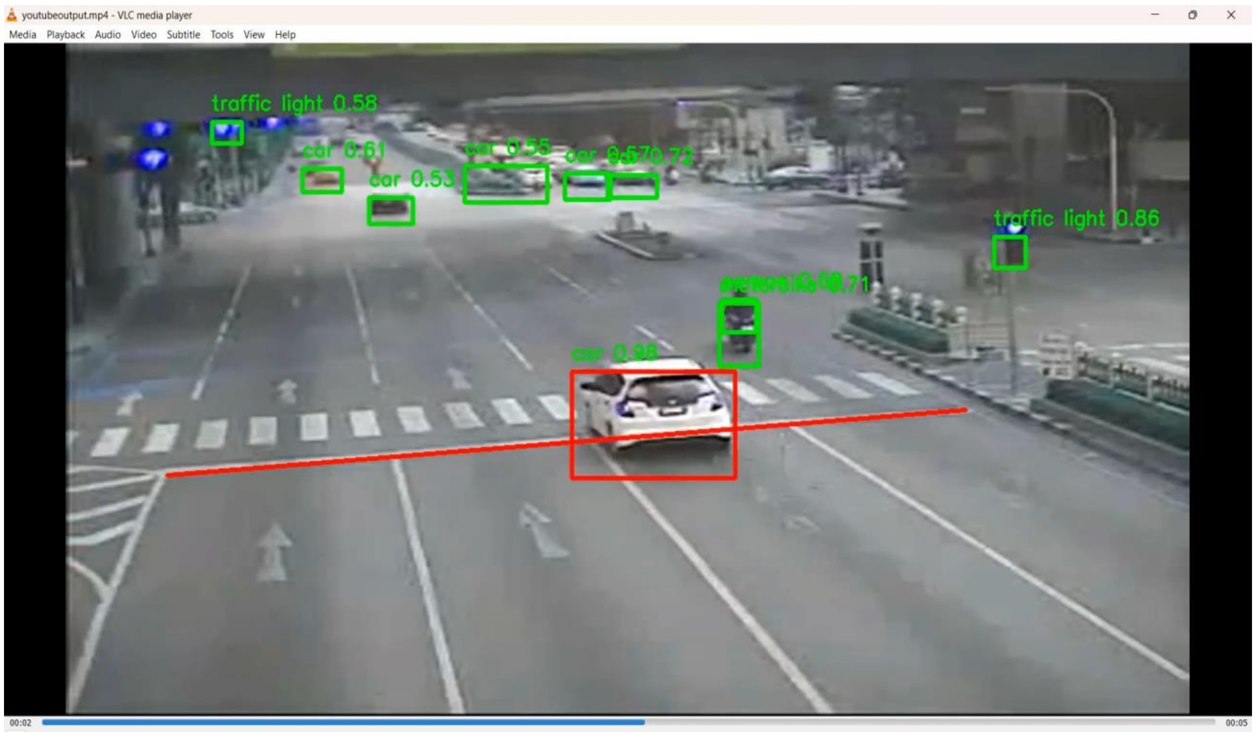


```
car: 93.01866888999939%
line: (367, 253) (843, 232)
Box: (633, 225) (705, 289)

(367, 253) (843, 232) (633, 225) (633, 289)
(21, 476, -128135) (-64, 0, 40512)
(633.0, 241.26470588235293)
(367, 253) (843, 232) (705, 225) (705, 289)
(21, 476, -128135) (-64, 0, 45120)
(705.0, 238.08823529411765)
(367, 253) (843, 232) (633, 225) (705, 225)
(21, 476, -128135) (0, 72, -16200)
(1001.6666666666666, 225.0)
(367, 253) (843, 232) (633, 289) (705, 289)
(21, 476, -128135) (0, 72, -20808)
(-449.0, 289.0)
True
car: 97.27264642715454%
line: (367, 253) (843, 232)
Box: (1041, 247) (1115, 303)

(367, 253) (843, 232) (1041, 247) (1041, 303)
(21, 476, -128135) (-56, 0, 58296)
(1041.0, 223.26470588235293)
(367, 253) (843, 232) (1115, 247) (1115, 303)
(21, 476, -128135) (-56, 0, 62440)
(1115.0, 220.0)
(367, 253) (843, 232) (1041, 247) (1115, 247)
(21, 476, -128135) (0, 74, -18278)
(503.0, 247.0)
(367, 253) (843, 232) (1041, 303) (1115, 303)
(21, 476, -128135) (0, 74, -22422)
(-766.3333333333334, 303.0)
False
car: 96.63075804710388%
line: (367, 253) (843, 232)
Box: (830, 256) (912, 331)

(367, 253) (843, 232) (830, 256) (830, 331)
(21, 476, -128135) (-75, 0, 62250)
(830.0, 232.5735294117647)
(367, 253) (843, 232) (912, 256) (912, 331)
```


Traffic Violation
traffic light 0.61
bus 0.58
car 0.78
car 0.57
car 0.9
car 0.97
car 0.98
person 0.89
person 0.98
person 0.93
person 0.93


Traffic Violation
traffic light 0.56
car 0.73
car 0.53
car 0.51
car 0.73
traffic light 0.85
car 0.99


Traffic Violation
traffic light 0.61
traffic light 0.5
car 0.51
car 0.62
car 0.54
car 0.55
traffic light 0.82
person 0.91
motorbike 0.65
car 0.83


youtubeoutput.mp4 - VLC media player
Media Playback Audio Video Subtitle Tools View Help
traffic light 0.58
car 0.61
car 0.53
car 0.55
traffic light 0.86
car 0.98
00:02
00:05

CHAPTER-X
REFERENCES

- 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.

- X. Wang, L.-M. Meng, B. Zhang, J. Lu and K.-L. Du, "A Video-based Traffic Violation Detection System," in MEC, Shenyang, China, 2013.

- Ahmed, S., Rahman, M. S., Reza, S. M. M., & Hoque, A. T. M. S. (2016). Traffic Violation Detection System Using Image Processing. In 2016 4th International Conference on the Development of Biomedical Engineering in Vietnam (BME4VIETNAM) (pp. 56-59). IEEE.

- Srivastava, A., Khanna, R., & Gupta, P. (2017). Traffic Signal Violation Detection and Alert System. In 2017 2nd International Conference on Telecommunication and Networks (TEL-NET) (pp. 199-202). IEEE.

- Patil, R. R., Patil, R. V., & Patil, S. R. (2018). Traffic Signal Violation Detection and Notification System Using Image Processing. In 2018 International Conference on Current Trends towards Converging Technologies (ICCTCT) (pp. 1-4). IEEE.

- Makwana, D., Patel, K., & Patel, H. (2020). Traffic Signal Violation Detection System using Deep Learning. In 2020 International Conference on Inventive Research in Computing Applications (pp. 45-49). IEEE.