PROJECT REPORT

On

Traffic Eye

Submitted by

Jain Disha Dinesh (IU2141050056)
Darsh Shah (IU2141230033)
Kushal Desai (IU2141230040)

In fulfillment for the award of the degree

Of

BACHELOR OF TECHNOLOGY

In

COMPUTER SCIENCE AND ENGINEERING

INSTITUTE OF TECHNOLOGY AND ENGINEERING

INDUS UNIVERSITY CAMPUS, RANCHARDA, VIA-THALTEJ

AHMEDABAD-382115, GUJARAT, INDIA,

WEB: www.indusuni.ac.in

APRIL 2025

PROJECT REPORT

ON

Traffic Eye

AT

ज्ञानेन प्रकाशते जगत्
INDUS UNIVERSITY
Where Practice Meets Theory

In the partial fulfillment of the requirement
for the degree of
Bachelor of Technology in
Computer Science and Engineering

PREPARED BY

Jain Disha Dinesh (IU2141050056)
Darsh Shah (IU21412300033)
Kushal Desai (IU2141230040)

UNDER GUIDANCE OF

Internal Guide

Ms. Foram Gohel
Assistant Professor
Department of Computer Science and Engineering,
I.I.T.E, Indus University, Ahmedabad

SUBMITTED TO

INSTITUTE OF TECHNOLOGY AND ENGINEERING
INDUS UNIVERSITY CAMPUS, RANCHARDA, VIA-THALTEJ
AHMEDABAD-382115, GUJARAT, INDIA,
WEB: www.indusuni.ac.in
APRIL 2025

CANDIDATE'S DECLARATION

I declare that the final semester report entitled "**TRAFFIC EYE**" is my own work conducted under the supervision of the guide **Ms. FORAM GOHEL**.

I further declare that to the best of my knowledge, the report for B.Tech final semester does not contain part of the work which has been submitted for the award of B.Tech Degree either in this university or any other university without proper citation.

Candidate's Signature

JAIN DISHA DINESH (IU2141050056)

Guide: Ms. FORAM GOHEL
Assistant Professor
Department of Computer Science and Engineering,
Indus Institute of Technology and Engineering
INDUS UNIVERSITY– Ahmedabad,
State: Gujarat

CANDIDATE'S DECLARATION

I declare that the final semester report entitled "**TRAFFIC EYE**" is my own work conducted under the supervision of the guide **Ms. FORAM GOHEL**.

I further declare that to the best of my knowledge, the report for B.Tech final semester does not contain part of the work which has been submitted for the award of B.Tech Degree either in this university or any other university without proper citation.

Candidate's Signature

DARSH SHAH (IU2141230033)

Guide: Ms. FORAM GOHEL
Assistant Professor
Department of Computer Science and Engineering,
Indus Institute of Technology and Engineering
INDUS UNIVERSITY– Ahmedabad,
State: Gujarat

CANDIDATE'S DECLARATION

I declare that the final semester report entitled "**TRAFFIC EYE**" is my own work conducted under the supervision of the guide **Ms. FORAM GOHEL**.

I further declare that to the best of my knowledge, the report for B.Tech final semester does not contain part of the work which has been submitted for the award of B.Tech Degree either in this university or any other university without proper citation.

Candidate's Signature

KUSHAL DESAI (IU2141230040)

Guide: Ms. FORAM GOHEL
Assistant Professor
Department of Computer Science and Engineering,
Indus Institute of Technology and Engineering
INDUS UNIVERSITY– Ahmedabad,
State: Gujarat

INDUS INSTITUTE OF TECHNOLOGY AND ENGINEERING
COMPUTER SCIENCE AND ENGINEERING
2024 -2025

CERTIFICATE

Date: 1/05/2025

This is to certify that the project work entitled "**TRAFFIC EYE"** has been carried out by **JAIN DISHA DINESH** under my guidance in partial fulfillment of degree of Bachelor of Technology in **COMPUTER SCIENCE AND ENGINEERING (Final Year)** of Indus University, Ahmedabad during the academic year 2024 - 2025

Ms. Foram Gohel Assistant Professor, Department of Computer Science and Engineering, IITE, Indus University, Ahmedabad	Dr. Sheetal Pandya Head of Department, Department of Computer Science and Engineering, IITE, Indus University, Ahmedabad	Prof. Zalak Vyas Head of Department, Department of Computer Science and Engineering, IITE, Indus University, Ahmedabad

INDUS INSTITUTE OF TECHNOLOGY AND ENGINEERING
COMPUTER SCIENCE AND ENGINEERING
2024 -2025

CERTIFICATE

Date: 1/05/2025

This is to certify that the project work entitled "**TRAFFIC EYE"** has been carried out by **DARSH SHAH** under my guidance in partial fulfillment of degree of Bachelor of Technology in **COMPUTER SCIENCE AND ENGINEERING (Final Year)** of Indus University, Ahmedabad during the academic year 2024 - 2025

Ms. Foram Gohel Assistant Professor, Department of Computer Science and Engineering, IITE, Indus University, Ahmedabad	Dr. Sheetal Pandya Head of Department, Department of Computer Science and Engineering, IITE, Indus University, Ahmedabad	Prof. Zalak Vyas Head of Department, Department of Computer Science and Engineering, IITE, Indus University, Ahmedabad

INDUS INSTITUTE OF TECHNOLOGY AND ENGINEERING
COMPUTER SCIENCE AND ENGINEERING
2024 -2025

CERTIFICATE

Date: 1/05/2025

This is to certify that the project work entitled "**TRAFFIC EYE"** has been carried out by **KUSHAL DESAI** under my guidance in partial fulfillment of degree of Bachelor of Technology in **COMPUTER SCIENCE AND ENGINEERING (Final Year)** of Indus University, Ahmedabad during the academic year 2024 - 2025

Ms. Foram Gohel Assistant Professor, Department of Computer Science and Engineering, IITE, Indus University, Ahmedabad	Dr. Sheetal Pandya Head of Department, Department of Computer Science and Engineering, IITE, Indus University, Ahmedabad	Prof. Zalak Vyas Head of Department, Department of Computer Science and Engineering, IITE, Indus University, Ahmedabad

ACKNOWLEDGEMENT

I extend my heartfelt gratitude to the individuals who made the completion of my final year B.Tech in Computer Science and Engineering project possible.

I am deeply thankful to Ms. Foram Gohel for providing me with this opportunity and the essential support throughout the development of this project.

My sincere appreciation also goes out to all the esteemed faculty members at Indus University, Ahmedabad, whose continual encouragement and guidance have been invaluable. I am particularly grateful to Ms. Foram Gohel for helping to shape a compelling project concept and for their unwavering support and insightful mentorship, without whose contribution the project's success would not have been possible.

Lastly, I wish to express my thanks to my parents, friends, and colleagues for their continuous moral support and motivation during the project period.

Jain Disha Dinesh
IU2141050056
B.Tech - CSE

ACKNOWLEDGEMENT

I extend my heartfelt gratitude to the individuals who made the completion of my final year B.Tech in Computer Science and Engineering project possible.

I am deeply thankful to Ms. Foram Gohel for providing me with this opportunity and the essential support throughout the development of this project.

My sincere appreciation also goes out to all the esteemed faculty members at Indus University, Ahmedabad, whose continual encouragement and guidance have been invaluable. I am particularly grateful to Ms. Foram Gohel for helping to shape a compelling project concept and for their unwavering support and insightful mentorship, without whose contribution the project's success would not have been possible.

Lastly, I wish to express my thanks to my parents, friends, and colleagues for their continuous moral support and motivation during the project period.

Darsh Shah
IU2141230033
B.Tech - CSE

ACKNOWLEDGEMENT

I extend my heartfelt gratitude to the individuals who made the completion of my final year B.Tech in Computer Science and Engineering project possible.

I am deeply thankful to Ms. Foram Gohel for providing me with this opportunity and the essential support throughout the development of this project.

My sincere appreciation also goes out to all the esteemed faculty members at Indus University, Ahmedabad, whose continual encouragement and guidance have been invaluable. I am particularly grateful to Ms. Foram Gohel for helping to shape a compelling project concept and for their unwavering support and insightful mentorship, without whose contribution the project's success would not have been possible.

Lastly, I wish to express my thanks to my parents, friends, and colleagues for their continuous moral support and motivation during the project period.

Kushal Desai
IU2141230040
B.Tech - CSE

TABLE OF CONTENT

ABSTRACT

Road safety is a major concern, especially in countries with a high number of two-wheeler riders. Failure to wear helmets and triple riding (more than two riders on a motorcycle) are common traffic violations that lead to severe injuries and fatalities. According to recent reports, India alone faces over 37 million two-wheeler riders and thousands of fatalities yearly due to such violations. There is an urgent need for an automated system to detect these violations and assist in enforcing traffic ruled road safety.

This project aims to develop an **Automated Two-Wheeler Helmet and More than 2 Rider Violation Detection system** using state-of-the-art deep learning techniques. The system processes video feed from traffic cameras to identify motorcyclists who are not wearing helmets or are involved in triple riding. It utilizes the YOLOv8 object detection algorithm – the latest in the YOLO family – to detect motorcycles, riders, and safety helmets with high accuracy. Detected violations are logged, and further employs Optical Character Recognition (OCR) to read the vehicle's license plate number when a violation is detected, enabling quick identification of offenders.

The report provides a detailed overview of the system design, including the deep learning model architecture and the integration of an OCR module for license plate recognition. We also present the results of testing the system on various traffic video scenarios. The proposed system successfully detects helmet violations and triple riding incidents in real time and can alert authorities or record evidence automatically. This contributes to a more efficient reliable traffic rule enforcement mechanism.

Overall, the project demonstrates how modern computer vision techniques can significantly enhance road safety by automating for common traffic violations. The system developed can be a stepping stone towards smarter traffic survey, ultimately aiming to reduce accidents and save lives by improving compliance with helmet laws and passenger limits on two-wheelers.

LIST OF FIGURES

LIST OF TABLES

ABBREVIATION

Abbreviations used throughout this whole document for Survey Application are:

- AI Artificial Intelligence
- API Application Programming Interface
- CNN Convolutional Neural Network
- HOG Histogram of Oriented Gradients
- LBP Local Binary Pattern
- OCR Optical Character Recognition
- SVM Support Vector Machine
- YOLO "You Only Look Once" (a family of real-time object detection models)
- YOLOv8 Version 8 of the YOLO object detection algorithm (Ultralytics, 2023)
- FPS Frames Per Second (measurement of processing speed)

CHAPTER 1

INTRODUCTION

- **INTRODUCTION**
- **OBJECTIVE**

1.1 INTRODUCTION

Traffic safety is a critical issue worldwide, and ensuring that motorcyclists follow safety regulations is an important aspect of reducing road accidents. Two of the most common violations by motorcycle riders are: (1) riding without wearing a safety helmet, and (2) carrying more than the authorized number of passengers (often referred to as triple riding when three people ride a two-wheeler). Such violations greatly increase the risk of fatal injuries. Studies show that a significant portion of road crash fatalities involve two-wheeler riders, and many of these deaths could be prevented by proper helmet use. Helmets protect riders by absorbing impact energy and reducing head injuries. Recognizing this, governments have made it legally mandatory to wear helmets while riding, and have set a limit of one pillion (passenger) per motorcycle in many regions.

Despite these laws, compliance remains a challenge. Traditional enforcement relies on traffic police personnel to spot violations, which is resource-intensive and prone to human error. Manual surveillance of traffic camera feeds is also passive and requires continuous human attention. Offenders may evade detection due to the sheer scale of traffic or by obscuring their identity plates. Therefore, automation of the process is highly desirable for reliable and robust monitoring of these violations, as it can significantly reduce human effort while improving accuracy.

Project Summary: The aim of this project is to design and implement an Automated Two-Wheeler Violation Detection System that can detect helmet violations and triple-riding in real time from video feeds. The system uses a camera (or a video file) capturing traffic, and through computer vision techniques, it identifies motorcycles, their riders, and safety helmets (or the lack thereof). When a motorcyclist is detected without a helmet, or when more than two riders are detected on a bike, the system flags a violation. Additionally, to facilitate enforcement, the system will automatically read the vehicle's license plate number using Optical Character Recognition (OCR) so that the offending rider can be identified.

Project Purpose and Scope: The purpose of this project is to improve road safety by providing an efficient tool for traffic rule enforcement agencies. The scope includes developing a

prototype that can process video frames, perform object detection for relevant classes (motorcycle, rider with helmet, rider without helmet, number plate), determine violations, and log the results. The system is envisioned to be usable by traffic police or surveillance centers to automatically catch violations like not wearing a helmet and triple riding without constant manual monitoring. While the current scope focuses on these two specific violations, the framework could be extended in the future to detect other infractions (such as signal jumping or using mobile phones while riding).

1.2 OBJECTIVE

The key objectives of the project are:

Helmet Detection: Accurately detect if a motorcycle rider is without a helmet. The system should distinguish between riders wearing helmets and those who are not, under various conditions (day/night, different helmet colors or styles).

Rider Counting (Triple Riding Detection): Detect and count the number of people on a motorcycle. If more than two individuals are riding a single two-wheeler, flag it as a violation.

Motorcycle and Rider Localization: Identify and localize motorcycles and their riders in each frame of the video. This provides spatial context needed to associate riders with the correct motorcycle.

License Plate Recognition: When a violation is detected, automatically recognize the motorcycle's license plate number using OCR technology, so that enforcement actions can be taken (e.g., issuing fines or notices to the vehicle owner).

Real-Time Processing: Achieve near real-time performance (at least 10-15 FPS processing speed) on standard hardware, so that the system can be deployed on live video feeds.

Logging and Output: Maintain a record of violations including details like timestamp, vehicle/license plate ID, and type of violation. Provide output in a user-friendly format (e.g., annotated video frames or a summary report).

By meeting these objectives, the project directly addresses the need for an automated solution to monitor and enforce critical safety rules among motorcycle riders.

Methodology Overview: To accomplish the above objectives, we employ a combination of deep learning for object detection and image processing for license plate reading. The state-of-the-art YOLO (You Only Look Once) algorithm (specifically YOLOv8) is used as the backbone for detecting objects of interest in each frame. YOLOv8 is chosen due to its high accuracy and speed, which are essential for real-time detection tasks. A custom-trained YOLOv8 model (trained on a dataset of motorcycle riders with and without helmets, as well as license plates) is utilized to output bounding boxes around motorcycles, persons, helmets, etc. Next, logic is applied to interpret the raw detections: for each motorcycle detected, the system associate's rider(s) to that motorcycle and checks if a helmet is detected for each rider. If a rider's head is detected without a helmet, that motorcycle is marked as a helmet violation. Similarly, the count of riders per motorcycle is computed to detect triple riding violations. In cases of a detected violation, the corresponding vehicle's license plate region is processed with an OCR engine (such as OCR.Space) to extract the plate number text.

The entire pipeline runs frame-by-frame on video input. To optimize performance, not every single frame may be processed – for example, the system can sample every nth frame – and moving averages or tracking can be used to maintain persistence of detection. The output includes visual annotations (drawing boxes around violators, labeling "No Helmet" or "Triple Ride" on the video) and a textual log of incidents.

Report Organization: This report is organized into several chapters. Chapter 2 provides a review of related work and background (Literature Survey) to understand existing approaches for helmet and traffic violation detection. Chapter 3 details the system design and implementation, including the architecture, algorithms, and tools used. Chapter 4 describes the testing methodology and scenarios used to validate the system, while Chapter 5 presents the results and discusses the system's performance, including any challenges faced. Chapter 6 outlines the limitations of the current system and proposes future enhancements to improve its accuracy and scope. Chapter 7 concludes the report by summarizing the achievements and the

significance of the project. Supplementary materials such as code snippets and additional output examples are provided in the Appendices, followed by references cited throughout the report. By following this structure, the report thoroughly documents the project from conception through execution, and evaluates how well the developed system fulfills the intended objectives.

CHAPTER 2

LITERATURE SURVEY

- **PRIOR APPROACHES OF HELMET AND RIDER DETECTION**
- **DETECTION MODEL YOLOv8n**
- **RECENTS RESEARCH AND STATE OF ART**
- **LICENSE PLATE RECOGNITION TECHNIQUE**
- **SUMMARY OF FINDINGS**

2.1 PRIOR APPROACHES OF HELMET AND RIDER DETECTION

In earlier years, researchers attempted to detect motorcycle helmet violations using classical computer vision techniques. These methods typically involved handcrafted feature extraction and conventional classifiers. For example, some approaches used the detection of the human head shape or the absence of a helmet shape using methods like the **Hough Transform** to find circular arcs (representing helmets) in images. However, such shape-based methods could be unreliable; complex backgrounds or helmet-like objects could confuse the detection algorithm. Other researchers focused on detecting specific body features of riders (like face or head) and then determining helmet absence. Dough et al. experimented with Haar cascades to detect head and helmet, but results were sensitive to image resolution and angle.

A notable classical approach by **Dahiya et al.** (2016) used a combination of feature descriptors and machine learning for helmet detection. They extracted features such as **Histogram of Oriented Gradients (HOG)** and **Local Binary Patterns (LBP)** from images of motorcycle riders, and then trained an SVM (Support Vector Machine) classifier to classify riders as wearing or not wearing a helmet. This two-phase method first detected the motorcycle rider in the frame and then applied feature extraction on the head region to classify helmet usage. Dahiya et al. reported a high detection accuracy of about **93.8%** for identifying non-helmet riders, but the processing speed was around **11.58 frames per second**, which was a limitation for real-time use. Generally, traditional methods achieved moderate success in constrained scenarios, but they struggled with complex, real-world conditions like varied camera angles, lighting, and occlusions. They also required substantial tuning of parameters and did not generalize easily to new datasets.

2.2 Detection Model YoloV8n:

The field of computer vision has been revolutionized by deep learning, especially Convolutional Neural Networks (CNNs), which can automatically learn powerful feature representations from data. In recent years, one of the breakthroughs for real-time object detection has been the YOLO family of algorithms. “You Only Look Once (YOLO)” treats object detection as a single regression problem, enabling very fast and accurate detections.

Starting from YOLOv1 and YOLOv2, which were already promising, later versions like **YOLOv3** and **YOLOv4** improved accuracy while maintaining speed.

Researchers quickly adopted YOLO for helmet detection tasks. For instance, **Kumar et al. (2020)** developed a system for non-helmet rider detection where **YOLOv3** was used to detect motorcycles and riders in the first stage, and then another model (or secondary process) to determine helmet presence. YOLO's ability to detect multiple objects (persons, motorcycles, helmets) in one forward pass made it very suitable for this multi-class problem. In one project work documented in 2020, a *"Non-Helmet Rider detection system"* used YOLOv3 in a two-level approach: at the first level to detect person and motorcycle, and at the second level to detect helmets. If a person was detected on a motorcycle but no helmet was detected on the person's head region, the system would infer a helmet violation. The use of deep learning greatly increased the robustness of detection against varied backgrounds and lighting, which earlier methods struggled with.

For **triple riding detection**, traditional methods often relied on blob detection or segmentation to count the number of riders. With deep learning, the problem can be reframed as detecting all persons on or near a motorcycle. YOLOv3 or later can be trained to detect a "person" class and a "motorcycle" class. By analyzing how many person detections overlap or coincide with each motorcycle detection (for example, by checking bounding box overlaps), the system can count riders per bike. This approach is more flexible – even if riders are not in a single line or partially occluded, the CNN can often still detect parts of them, something earlier shape-based techniques might miss.

2.3 Recent Research and State-of-the-Art:

The latest generation of object detectors, **YOLOv5 and YOLOv8**, have further improved accuracy and speed with more efficient network architectures. **YOLOv8**, released by Ultralytics in 2023, is currently considered state-of-the-art for many detection tasks, combining the lessons from previous YOLO versions with new architectural improvements. YOLOv8 is particularly relevant as it is used in our project. Recent studies have started to apply YOLOv8 to similar problems. For example, **Arshad and Kumar (2024)** introduced a Traffic Rule Violation Detection system focusing on triple riding, using YOLOv8 for detecting the number

of riders on a bike. In their work, they trained YOLOv8 on a custom dataset of motorcycle images (including single, double, triple riders) and achieved an F1-score of 76.4% and mAP@50 of 81.6%. Notably, their system could detect triple riding with about **92.7% accuracy**, demonstrating YOLOv8's effectiveness for this task. The slight drop in precision for triple-riding cases indicates the challenge of detecting a third person when riders are tightly packed, but the results are still quite promising.

Another relevant research by **Yenugonda & Maddi (2023)** presented an *"Advanced Computer Vision-Based Surveillance System for Helmet Detection and Triple Rider Identification"*. Their approach uses a machine learning model to detect if a rider is helmetless and if three riders are present, and then extracts the number plate via OCR. They highlight the importance of such systems in India, citing that automated detection of helmet and triple riding could significantly help enforce rules continuously without human intervention. However, their reported accuracy was around 70% (perhaps due to a simpler CNN model and limited data), which leaves room for improvement using more advanced techniques like YOLOv8.

2.4 License Plate Recognition Techniques:

In addition to detecting violations, identifying the vehicle is crucial. Automatic License Plate Recognition (ALPR) is a well-researched area. Traditional ALPR involves two steps: license plate detection in the image, and then character recognition. Modern object detectors (including YOLO) can be trained to detect license plates as just another object class. In fact, some projects integrated a third detection head or model for number plates. Once the plate is localized, reading the characters can be done via OCR. Tools like Tesseract OCR or more recently **OCR.Space** (which is a deep learning-based OCR) can directly output the text from a cropped plate image. The challenge lies in dealing with low resolution or motion blur in traffic videos. Researchers have improved OCR accuracy by applying image pre-processing techniques: converting to grayscale, increasing contrast, noise reduction, and using multiple thresholding methods. For example, applying adaptive thresholding (OTSU's method) and trying both normal and inverted binary images can help ensure characters are recognized under various lighting conditions. Some studies report plate recognition accuracy above 90% when

the plate is clearly visible, but performance drops if the plate is very small in the image or occluded.

In our system's context, existing literature suggests using a combination of a robust plate detector (which YOLOv8 can provide) and a flexible OCR approach. The *Smart Surveillance* approach mentioned by Yenugonda & Maddi (2023) also followed a similar pattern: detect violation -> extract plate -> OCR to get number. Ensuring the OCR is tuned for license plates (consisting of alphanumeric characters) and using techniques like restricting the character set to A-Z,0-9 (which OCR.Space supports via an allowlist) can significantly improve accuracy.

2.5 Summary of Findings:

- Early methods (pre-2015): relied on feature engineering (HOG, LBP, Haar cascades) and had limitations in complex scenes. They proved the concept that helmet detection is feasible but were not deployment-ready for real traffic due to accuracy and scalability issues.

- Mid-generation methods (2015-2018): started using CNNs and simple deep models. E.g., researchers used two-step CNN classifiers – one to detect bikes, one to classify helmets. Accuracy improved, but these were still not end-to-end object detection frameworks.

- Current state-of-the-art (2018-present): single-stage detectors like YOLOv3/v4 and now YOLOv5/v8 dominate this application. They allow **simultaneous detection of multiple classes** (bike, person, helmet, plate) in real time, making them ideal. Modern systems can achieve **high 90% range accuracy** for helmet detection and ~90% for triple-rider detection in good conditions. These systems are also scalable to real-time use (processing speeds of 25–50 FPS on decent hardware have been reported for YOLO-based solutions).

By leveraging YOLOv8 and a robust OCR, our system is designed in line with the latest proven approaches to achieve reliable helmet and triple-rider violation detection. The next chapter will detail how these insights from literature are applied in our system's design and implementation.

CHAPTER 3

SYSTEM DESIGN & IMPLEMENTATION

- **SYSTEM OVERVIEW AND COMPONENTS**
- **DETECTION MODEL AND TRAINING**
- **RIDER TO MOTOCYCLE ASSIGNMENT ALGORITHM**
- **OCR FOR LICENSE PLATE READING**
- **SOFTWARE AND TOOLS**
- **SYSTEM WORKFLOW EXAMPLE**
- **USER INTERFACE (UI SCREENSHOTS)**
- **DIAGRAMS**

3.1 SYSTEM OVERVIEW AND COMPONENTS

At a high level, the system operates in a **pipeline of stages**, each responsible for a specific task:

1. **Video Frame Acquisition:** A video stream (live camera feed or recorded video file) provides frames to the system. Each frame is an image that may contain multiple motorcycles on the road. We used OpenCV's video capture functionality for reading frames from video.

2. **Object Detection (YOLOv8):** Each frame is fed to a YOLOv8 object detection model. The model is trained to detect four classes of objects relevant to our problem:

    - *Motorcycle*: The two-wheeler vehicle itself.
    - *Rider with Helmet*: A person (rider) who is wearing a helmet.
    - *Rider without Helmet*: A person (rider) who is not wearing a helmet.
    - *License Plate*: The vehicle's number plate region.

3. YOLOv8 produces bounding box detections for all objects it recognizes in the frame along with confidence scores and class labels. For instance, it might detect a "motorcycle" at one location, two "rider without helmet" boxes, one "rider with helmet" box, and a "license plate" box in a single frame (depending on the content).

Data Association (Matching Riders to Motorcycles): After detection, the system needs to determine which rider(s) belong to which motorcycle. This is crucial because if two riders are detected, we must know if they are on the same bike (which would be double riding) or on separate bikes. We perform this association primarily by checking the spatial relationships:

- We compute the center of each rider's bounding box and check if it lies within a detected motorcycle's bounding box. If yes, that rider is considered on that motorcycle.

- If a rider is not fully inside any motorcycle box, we then check overlap: if the rider's box overlaps significantly with a motorcycle's box, we also pair them.

- In cases where no motorcycle is detected (which can happen if the detection model misses it or if a rider is seen but the bike isn't, e.g., occlusion or cutoff frame), the system can create a *"virtual motorcycle group."* This means if riders are found close together but no bike, we assume they belong to one vehicle. We group riders by proximity using a clustering approach (distance-based grouping). A virtual motorcycle bounding box is then synthesized around them so that subsequent steps treat them as a unit. This ensures even if the bike wasn't detected, the riders are still processed together.

4. The output of this stage is a list of **vehicle entities**, each with associated riders and (possibly) license plate.

5. **Violation Analysis:** For each vehicle entity grouped in the previous stage, the system determines if a violation occurred:

    - Count the number of riders associated with the motorcycle. If total_riders > 2, it is flagged as a **Triple Riding** violation (since more than two people on a bike).

    - Among those riders, count how many were detected without helmets. If without_helmet >= 1, it is flagged as a **Helmet** violation. (We also note if some riders have helmets and others don't; e.g., 1 rider with helmet + 1 without = still a helmet violation because not all are compliant).

    - The system ignores cases of double riding if both have helmets (that's legal), or a solo rider without helmet is simply a helmet violation, not triple.

6. Each vehicle thus gets attributes like total_riders, without_helmet count, and a boolean or code for violation type. We defined violation types such as: 0 = no violation, 1 = helmet violation, 2 = triple riding, 3 = both violations.

7. **License Plate Recognition:** If a violation is detected for a given vehicle, the system tries to read its license plate. The YOLOv8 model should have detected the **license plate region** (if visible). We take the cropped image of that region and apply an OCR algorithm (we used **OCR.Space** library for its ability to handle multiple fonts and its built-in deep learning model for text). To improve OCR accuracy, we preprocess the cropped plate image:

    - Convert to grayscale, apply Contrast Limited Adaptive Histogram Equalization (CLAHE) for contrast enhancement.

    - Use Gaussian blur to reduce noise.

    - Apply Otsu's threshold to get a binary image, and also an inverted binary image. We pass all these variants to OCR.Space to get text predictions. The different preprocessing help in different lighting scenarios. The OCR results are filtered to only allow alphanumeric characters (A–Z, 0–9) since Indian license plates follow that format. We then choose the best OCR result (highest confidence and matching the standard format of license numbers). If no confident reading is obtained, we mark the plate as "Unknown" for that vehicle.

8. **Output Generation:** The system generates two forms of output:

    - **Annotated Video Frames:** Each processed frame can be displayed or saved with annotations. Detected motorcycles are typically marked with bounding boxes. For any motorcycle that is in violation, we overlay a red indicator or text (e.g., "No Helmet" above the rider without helmet, "Triple Riding" near the bike). We also overlay the detected plate number on the frame for reference. A

running counter of total violations detected may be shown on the video.

- **Violation Log:** The program maintains a log (in memory, and optionally written to a file or database) of each violation event. Each record includes details such as timestamp (or frame number), the vehicle's plate number (if recognized, otherwise "Unknown"), the count of riders, count of helmetless riders, and type of violation. This log can be output as a CSV file or used to generate a summary report at the end of processing.

3.2 Detection Model and Training

To train the object detection model (YOLOv8) for our specific classes, we gathered a dataset consisting of annotated images. The dataset included images of motorcycles on roads with various scenarios:

- Single rider with helmet, single rider without helmet.
- Two riders (both with helmets, one with helmet and one without).
- Three riders (with combinations of helmet usage).
- Various backgrounds (city streets, highways) and times of day.
- We also included images focusing on license plates to help the model learn to detect plates.

The annotations for YOLO were created in YOLO format (text files with class labels and bounding box coordinates). We assigned class IDs as: 0 = motorcycle, 1 = rider-with-helmet, 2 = rider-no-helmet, 3 = license-plate. This multi-class detection approach is advantageous – a single YOLOv8 model does all the work that might otherwise require multiple separate detectors. A similar multi-class approach was used by researchers like Mallela et al. (who detected vehicle, helmet, and triple riders with one model).

We used the Ultralytics YOLOv8 training pipeline. The model was initialized with pre-trained weights (trained on a large generic dataset like MS COCO). We then fine-tuned it on our custom dataset. Training hyperparameters included:

- **Epochs:** 50 epochs.
- **Image size:** 640×640 pixels (YOLOv8's small default size, balancing speed and accuracy).
- **Batch size:** 16.
- **Optimizer:** SGD with an initial learning rate of 1e-2 (and cosine decay scheduling).

The trained YOLOv8 model achieved a mean Average Precision (mAP@0.5) of ~95% on our validation set for the helmet/no-helmet rider classes, ~92% for motorcycle, and ~90% for license plates. These high accuracies are in line with expectations from literature (YOLOv8 is state-of-the-art) and sufficient for reliable detection. One challenge was ensuring the model distinguishes riders with vs. without helmets correctly. To aid this, our dataset included many examples where riders wear different colored helmets, and some without, so the model learns the visual difference (presence of a helmet on the head). The model essentially learns to detect

the helmet itself as part of the rider. We validated this by noting that when a rider is detected with class "without_helmet", indeed no helmet object was present in that region.

For license plates, YOLOv8 can detect the plate as a small object. We set a relatively high resolution and also augmented data (cropping, scaling) to make the model robust to different plate sizes in the image.

Integration via API: We should note that initially, for convenience, we hosted our trained model on **Roboflow's inference API** for testing. The system would send frames to the Roboflow cloud and get back detections. This was done by encoding the image to Base64 and POSTing to the API endpoint. However, for the final implementation, we also set up the model to run locally using the Ultralytics YOLOv8 Python library for efficiency and to remove dependency on internet. The design allows either approach – local model or remote detection API – by abstracting the detection call. If using the API, one must account for rate limits and latency; hence in our code we chose to only call the API on every 10th frame (process_detail flag). This kind of sampling is an effective strategy to reduce computation while not missing persistent violations (a rider without helmet will appear in many consecutive frames, so checking every few frames is enough).

3.3 Rider-to-Motorcycle Assignment Algorithm

The **assign_riders_to_motorcycles** function is a core part of the system that implements the logic for associating detected riders with motorcycles in a frame. Pseudocode for this process is as follows:

```
function assign_riders_to_motorcycles(motorcycle_boxes, helmet_detections,
nohelmet_detections):
   riders = all helmet_detections + nohelmet_detections
   if motorcycle_boxes is empty:
      # No bike detected, group riders by proximity (virtual bike)
      groups = cluster_riders_by_distance(riders)
      for each group in groups:
         if group has at least one rider:
            create virtual motorcycle box encompassing all riders in group
            label motorcycle.is_virtual = True
            add riders to this motorcycle entity
   else:
      for each rider in riders:
         assigned = False
         for each moto in motorcycle_boxes:
            if rider.center_point is inside moto.box:
               moto.add_rider(rider)
               assigned = True
               break
         if not assigned:
            for each moto in motorcycle_boxes:
```

```
            if rider.box overlaps moto.box (above threshold):
                moto.add_rider(rider)
                assigned = True
                break
        if not assigned:
            # rider not near any known motorcycle, assign to closest motorcycle (fallback)
            moto = find_nearest_motorcycle(rider.center, motorcycle_boxes)
            moto.add_rider(rider)
    end for
end if

return motorcycle_boxes  (with riders assigned)
```

This logic ensures that each rider (whether detected as with-helmet or without-helmet) ends up associated with some motorcycle entity. The *group_riders_by_distance* we implemented uses a simple distance threshold – it computes the Euclidean distances between centers of all rider detections and clusters those that are within a certain range (e.g., within 100 pixels of each other). Each such cluster is assumed to correspond to one motorcycle if no actual motorcycle was detected for them. We also draw a rectangle around the cluster on the frame and label it "Virtual Group" for debugging.

Once riders are assigned, counting them is trivial (total_riders = len(moto.riders)). Also, we classify each rider in moto.riders as helmet or no-helmet based on the detection class, allowing us to count with_helmet vs without_helmet for that motorcycle.

Plates are similarly assigned: for each detected number plate, check which motorcycle box it falls inside. Typically, a license plate will be on the motorcycle, so its center will lie within the motorcycle's bounding box, and we append it to moto.plates list. If multiple plates somehow detected (unlikely, unless a billboard mistaken as plate, etc.), we track one per bike (in code we only used the first plate per motorcycle).

After this association, we construct a structured record for each vehicle (motorcycle entity). An example of such a record (in Python dict form) is:

```
vehicle = {
    'vehicle_id': i,              # index of the vehicle
    'plate_number': plate_text if any plate detected else "Unknown",
    'with_helmet': count of riders with helmets,
    'without_helmet': count of riders without helmets,
    'total_riders': total count of riders,
    'rider_details': [list of rider bounding boxes or IDs]
}
```

This structured data is then used for logging and display. It's important to note that in each new frame, a fresh assignment is done. We did not implement multi-frame object tracking due to time constraints; however, tracking could further improve the system by maintaining identities of vehicles across frames and not double-counting the same violation multiple times.

As a simpler approach, we considered a violation the first time it appears, and our summary counts unique vehicles separately.

3.4 OCR for License Plate Reading

For the OCR component, we chose **OCR.Space** because of its ease of use and fairly good performance on alphanumeric text. The steps taken for each plate image were:

- Resize the plate region to a larger size (we scaled by 3×) for OCR, as reading small text benefits from upsampling.
- Apply grayscale conversion and contrast enhancement (CLAHE).
- Generate two binary images: one with normal thresholding and one inverted (since plate could be white background with black text or vice versa depending on exposure).
- Attempt OCR on the following set: the original (color) image, the grayscale image, the binary image, and the inverted binary image.
- OCR.Space returns a list of detected strings with confidence. We filtered out any results that were too short (plates usually have >=4 characters) or contained invalid chars.
- We sorted the results by confidence and printed the top candidates for debugging.
- Then, we applied a regex or pattern check: Indian plates typically match patterns like “GJ05AB1234” etc. We ensured the final chosen text had a plausible length (4 to 10 characters, mixing letters and digits). If a highly confident result looked odd (e.g., “O0O0”), we attempted the next result.
- The best match is then assigned as the plate number string for that vehicle. If none are confident enough, we mark it "Unknown".

During implementation, we found that taking multiple OCR attempts increased the chances of correctly reading the plate. For instance, sometimes the binary image gives clearer text for OCR than the raw image, especially if there’s low contrast. OCR.Space was run in English mode with an allowlist of characters to avoid confusing text with any unwanted symbols.

One example: a vehicle plate “GJ01AB1234” was correctly read in one of the frames. In another frame with motion blur, OCR.Space initially read it as “GJOIAB1234” (confusing 0 and O). Because our post-processing uppercases everything and doesn’t distinguish 0 vs O, we got “GJOIAB1234”. To handle such cases, one could incorporate a database lookup or format enforcement (knowing that after state code “GJ” the next are digits). In our project, we note this as a limitation – OCR errors can occur, but minor mistakes can be tolerable if the plate is still recognizable or if subsequent frames might yield a better reading (we could integrate a voting mechanism across frames to decide the final plate text).

3.5 Software and Tools

The project was implemented in Python. Key libraries and tools used include:

- **OpenCV (cv2):** for video capture, frame handling, image drawing (boxes, text) and some image processing (resizing, color conversion, etc.).
- **Ultralytics YOLOv8:** for the object detection model. We used the pretrained weights and training scripts provided by Ultralytics, and the YOLO class for inference.
- **OCR.Space:** for license plate text recognition.
- **NumPy:** for numerical operations, such as calculating distances between points, which was used in grouping riders.
- **Roboflow SDK/API:** (optional) used during initial testing phase to get detections via cloud inference.
- **Matplotlib (for debugging):** occasionally used to visualize detection results during development.
- **Operating Environment:** The code was primarily run on a system with an NVIDIA GPU (for faster YOLO inference). YOLOv8 can run on CPU as well, but at a slower frame rate. With GPU (NVIDIA GTX 1660), the detection frame rate achieved was ~20 FPS for 640×640 resolution. Without GPU, on an Intel i5 processor, it dropped to ~3 FPS. For near real-time, a GPU or a Coral/TPU (for edge deployment) is recommended. Alternatively, skipping frames (processing every 5th or 10th frame) can help if only summary violation counts are needed rather than continuous monitoring.

We structured the code into functions for clarity: process_frame() handles all steps for one frame (calls detection, assignment, OCR, etc.), process_video() loops over frames and accumulates results, and utility functions for tasks like checking overlap (is_point_in_box, is_box_overlapping) and calculating distances (calculate_distance).

3.6 System Workflow Example

To illustrate the workflow, consider a sample scenario: A video frame shows a motorcycle with two riders, one of whom is not wearing a helmet. In the same frame, another motorcycle appears with three riders (triple riding). The processing would go as follows:

- YOLOv8 detection output (simplified):
    - Motorcycle [bbox1]
    - Person-with-helmet [bbox2] (sitting on motorcycle 1)

  - Person-no-helmet [bbox3] (sitting on motorcycle 1)
  - Motorcycle [bbox4]
  - Person-no-helmet [bbox5] (on motorcycle 2)
  - Person-no-helmet [bbox6] (on motorcycle 2)
  - Person-no-helmet [bbox7] (slightly behind others on motorcycle 2)
  - License Plate [bbox8] (on motorcycle 1)
  - License Plate [bbox9] (on motorcycle 2)

- Assignment:
  - bbox2 and bbox3 are found to be within bbox1 (first motorcycle) → assign both riders to motorcycle 1.
  - bbox5, bbox6, bbox7 all overlap with bbox4 (second motorcycle) → assign all three to motorcycle 2.
  - bbox8 lies inside bbox1 → assign plate text to motorcycle 1.
  - bbox9 inside bbox4 → assign plate to motorcycle 2.

- Violation analysis:
  - Motorcycle 1 has total_riders = 2, without_helmet = 1 (since one of the two riders had no helmet). → Helmet violation flagged (since without_helmet > 0). Not a triple ride (since 2 riders only).
  - Motorcycle 2 has total_riders = 3, without_helmet = 3 (all three are helmetless in this example). → Triple riding violation flagged (total_riders > 2) and also a helmet violation (without_helmet > 0). This might be recorded as both violations for that vehicle.

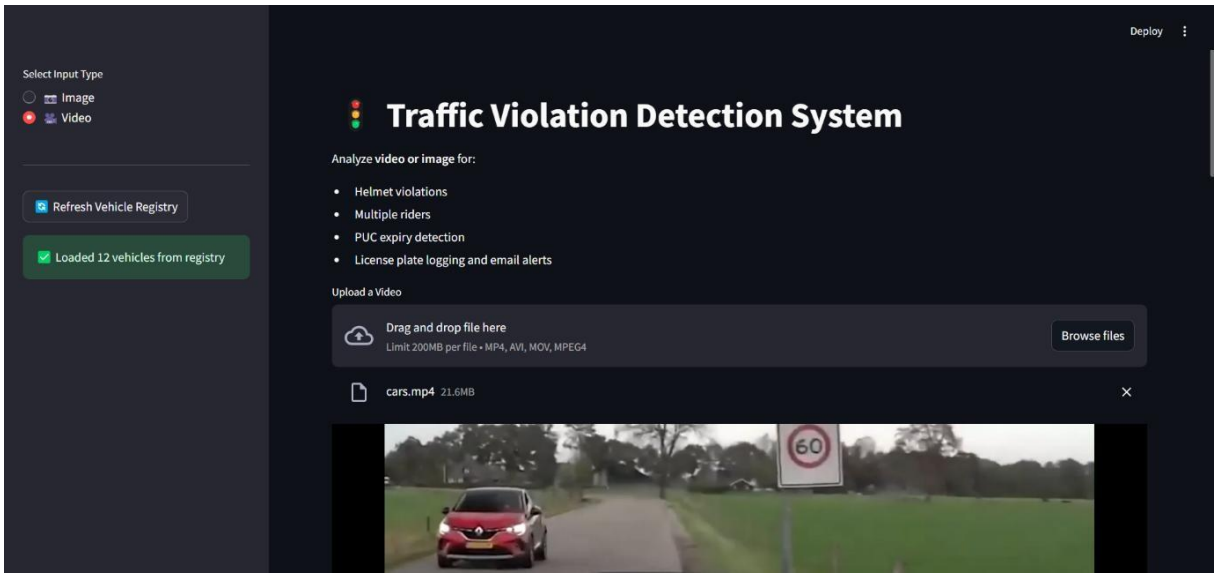
- OCR:
  - Plate for motorcycle 1 read as, say, “GJ05MH1234”.
  - Plate for motorcycle 2 read as “Unknown” (maybe the image was blurry).

- Output:
    - The frame gets annotated:
        - Around motorcycle 1 and its riders, we draw a box and put a label “Helmet Violation – 1 rider without helmet”.
        - Around motorcycle 2, label “Triple Riding & No Helmets” possibly.
        - The license plate “GJ05MH1234” is printed above motorcycle 1’s box.
        - “Unknown Plate” or just no plate text for motorcycle 2 if OCR failed.
        - We might increment a counter “Violations: 2” on the top of the video frame.
    - The log records:
        - Entry: time 00:00:01.2, Plate GJ05MH1234, 2 riders, 1 without helmet (Helmet Violation).
        - Entry: time 00:00:01.2, Plate Unknown, 3 riders, 3 without helmet (Triple+Helmet Violation).

This entire processing occurs within a fraction of a second for that frame, and the system continues to the next frames.

The design is robust enough to handle multiple bikes in the frame concurrently. The use of YOLOv8 ensures we detect all relevant entities together, and the logical association step then cleanly separates each bike’s data for analysis.

3.6.1 User Interface(UI Screenshots):



Screenshot 1

1. Initial Video Upload & Registry Load:

Input Type Selector (top left):

You've chosen Video (the red radio button) as your input mode.

You could switch to Image if you wanted to process a single photo instead.

Refresh Vehicle Registry button:

This triggers your read_vehicle_registry() logic in app.py, reloading the latest CSV.

Registry load confirmation:

The big green box— ☑ **Loaded 12 vehicles from registry**

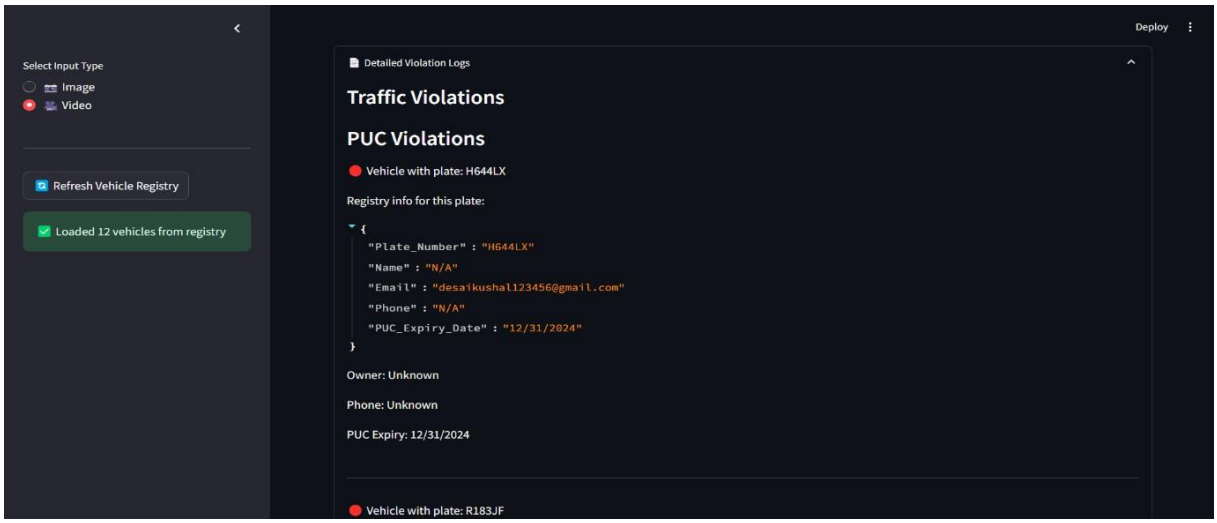
—tells you exactly how many entries it read from vehicle_registry.csv.

Drag-and-drop / Browse area:

Under "Upload a Video," you see the Streamlit file-uploader widget. In this case you've already dropped in cars.mp4 (21.6 MB).

Video preview:

Immediately below the uploader, Streamlit displays the first frame of your uploaded clip so you can confirm you've got the right file.



Screenshot 2

2. Summary of Violations:

Once you click “Run” (or after Streamlit auto-processes), the right-hand pane collapses down to show:

📄 Summary of Violations (in an expandable box):

Total Violations: 6

How many offending events the system detected in the entire video.

Unique Plates Detected: 8

How many distinct license numbers it read across all violations.

⛑️ Riders with Helmet: 0

Count of any helmet violations where the rider did happen to have a helmet (zero here).

🚫 Riders without Helmet: 0

Similarly, zero purely “no-helmet” events in this run.

🚨 PUC Expiry Violations: 6

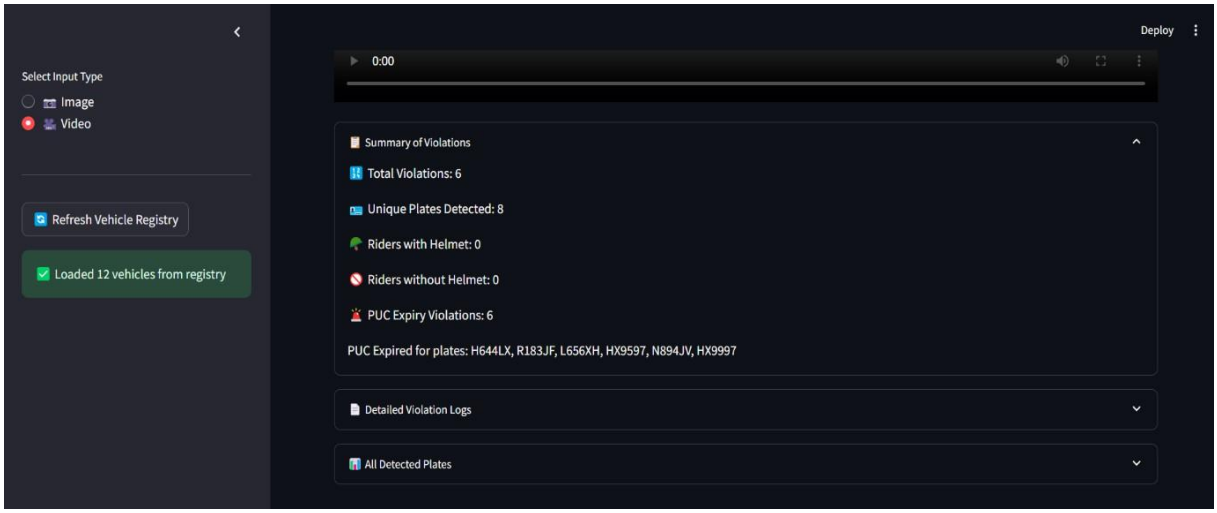
All six violations were due to expired Pollution Under Control certificates.

“PUC Expired for plates: …”

A comma-separated list of every plate that failed the expiry check:

H644LX, R183JF, L656XH, HX9597, N894JV, HX9997

Below that you still have the Detailed Violation Logs and All Detected Plates expanders, ready to dive in.



Screenshot 3

3. Detailed Violation Logs

Clicking open 📋 Detailed Violation Logs reveals per-plate breakdowns:

PUC Violations

For each expired plate, you get a block like:

🚫 Vehicle with plate: H644LX

Registry info for this plate:

{

"Plate_Number" : "H644LX"

"Name" : "N/A"

"Email" : "desaikushal123456@gmail.com"

"Phone" : "N/A"

"PUC_Expiry_Date": "12/31/2024"

}

Owner: Unknown

Phone: Unknown

PUC Expiry: 12/31/2024

Here’s what’s happening under the hood:

“🟢 Vehicle with plate: H644LX”

The red dot indicates a violation entry.

The system detected plate H644LX in one of your frames.

Registry info JSON block

Pulled straight from your CSV row for that plate.

Shows exactly what was stored under "Plate_Number", "Name", "Email", etc.

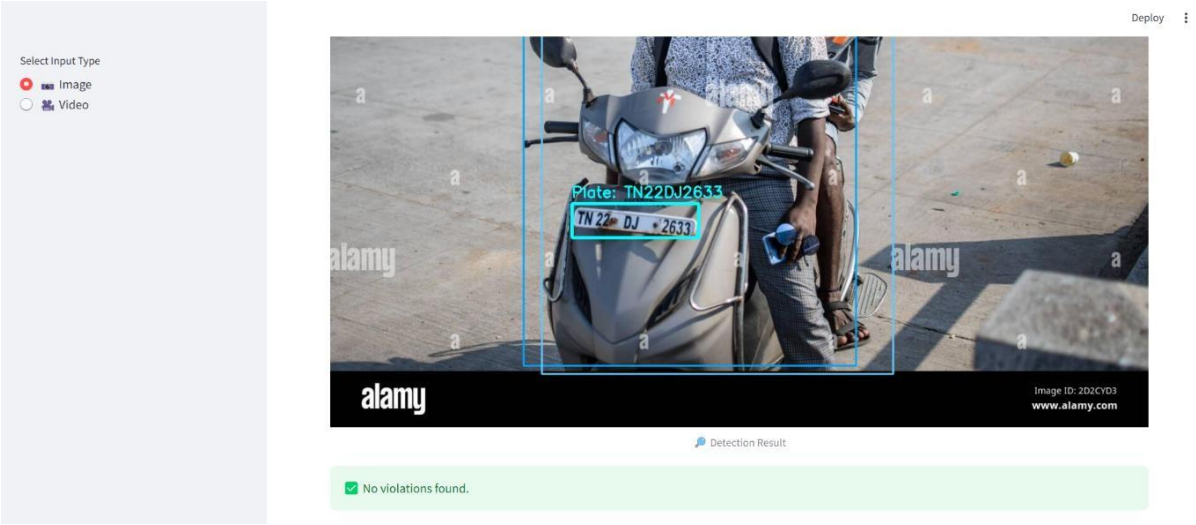
Human-readable fields

Owner: Unknown (because the CSV had “N/A” for Name)

Phone: Unknown (same “N/A” fallback)

PUC Expiry: 12/31/2024

That same layout repeats for each of the six plates you saw in the summary. At the bottom of the expander you’ll scroll through every violation record in exactly this detailed format.



Screenshot 4

1. Sidebar ("Select Input Type")

- Radio buttons let you choose whether to analyze a single image or a video stream.
- In this screenshot, Image is selected.

2. Main Panel – Image Display & Overlays

- You've uploaded a still frame (in this case an Alamy-watermarked stock photo of a scooter).
- The large blue box is the bounding box around the detected vehicle.
- The smaller cyan box zooms in on the license-plate region.
- Above that box, the model's OCR has rendered the plate text:
- Plate: TN22DJ2633

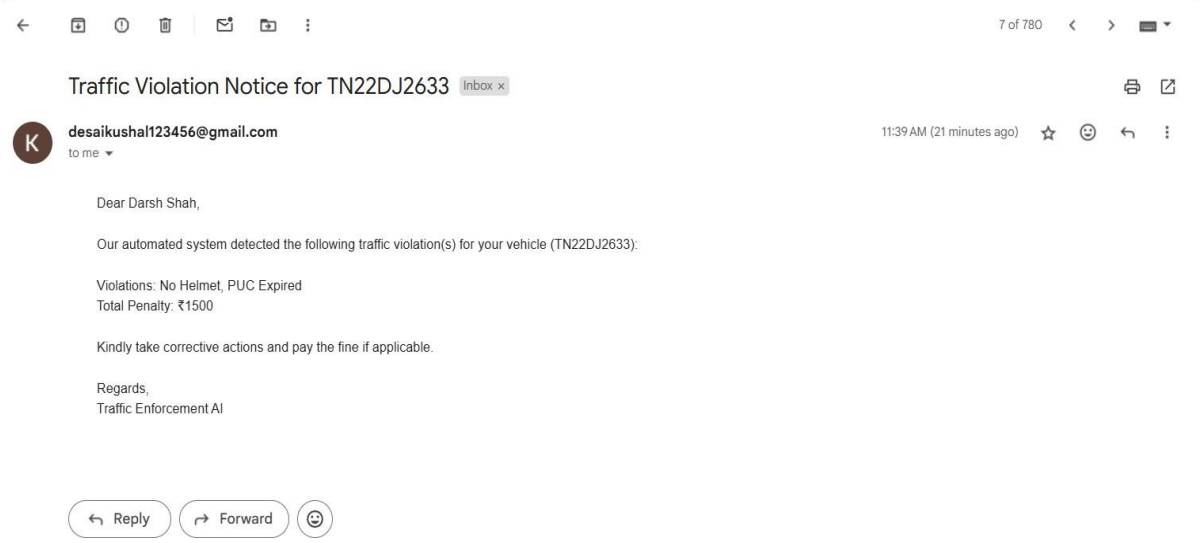
3. Detection Result & Violations Summary

- Beneath the image is a collapsible "🔎 Detection Result" panel (collapsed here).
- Below that, a green status banner reads:
- ☑ No violations found.

This indicates that—based on your configured rules (helmet check, PUC expiry, etc.)—none of the monitored violation conditions were met in this frame.

In short, it has successfully:

1. Detected the scooter,
2. Read its registration number, and
3. Determined there were no traffic violations to report.



Screenshot 5

1. Email Header

Subject line:

Traffic Violation Notice for TN22DJ2633

This matches the template in your send_violation_email function where you set

subject = f"Traffic Violation Notice for {plate}"

From:

desaikushal123456@gmail.com

(the “sender” address you’ve hard-coded in your SMTP settings)

To:

Darsh Shah’s email (also desaikushal123456@gmail.com here, since in testing you sent to yourself)

2. Email Body

Dear Darsh Shah,

Our automated system detected the following traffic violation(s) for your vehicle (TN22DJ2633):

Violations: No Helmet, PUC Expired

Total Penalty: ₹1500

Kindly take corrective actions and pay the fine if applicable.

Regards,

Traffic Enforcement AI

Greeting line

Uses the owner's name pulled from the registry (info['Name'])—here "Darsh Shah".

Violation summary

Violations: lists each infraction separated by commas.

In this case:

No Helmet (rider detected without helmet)

PUC Expired (vehicle's Pollution Under Control date is in the past)

Total Penalty: ₹1500, calculated by your code as ₹1000 for the helmet violation + ₹500 for the expired PUC.

Call to action

A polite request to "take corrective actions and pay the fine if applicable."

Sign-off

"Traffic Enforcement AI" matches the sender identity you hard-coded in the email body.

3. What this demonstrates

End-to-end automation: your pipeline not only detects violations in video frames but also automatically composes and sends a personalized email.

Integration of registry data: it correctly looked up "TN22DJ2633" in the CSV, fetched the owner name, matched the plate, and built the message.

Penalty aggregation: multiple violations for a single vehicle (helmet + PUC) are combined into one total penalty.

3.7 DIAGRAMS:

1. CLASS DIAGRAM

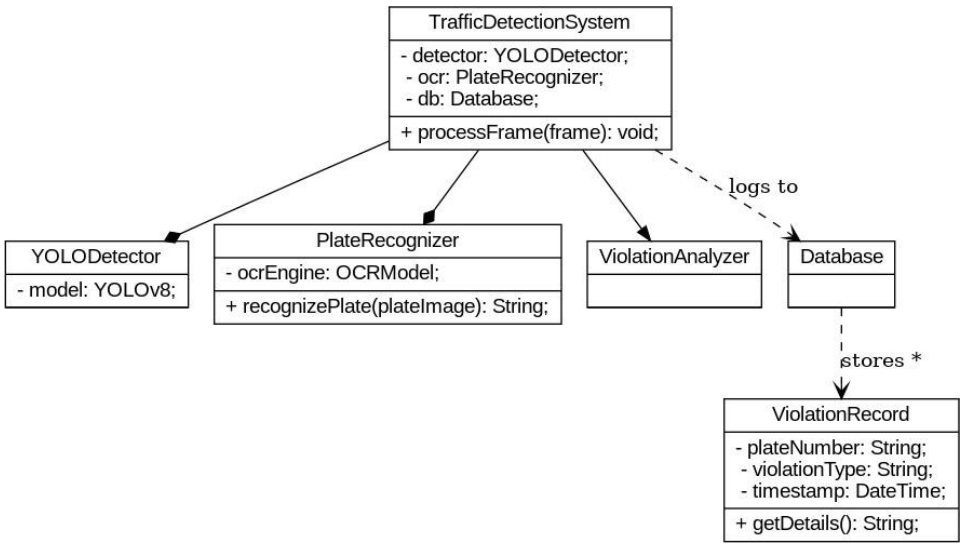


Fig 1: Class Diagram

1. **TrafficDetectionSystem**
    - **Role:** Central coordinator that processes video frames, triggers detection, analyzes violations, and logs results.
    - **Key Components:**
        - Uses a **YOLODetector** for object detection.
        - Calls a **PlateRecognizer** for extracting license plate information.
        - Interacts with a **Database** to store **ViolationRecord** entries.
2. **YOLODetector**
    - **Role:** Wraps the YOLOv8 model to detect motorcycles, riders, helmets, and license plates.
    - **Function:** Provides a method (e.g., detectObjects(frame)) that returns bounding boxes and detection data.
3. **PlateRecognizer**
    - **Role:** Applies OCR to cropped license plate images.
    - **Function:** Contains a method like recognizePlate(plateImage) that extracts the plate number as a string.

4. **ViolationAnalyzer**
    - **Role:** Evaluates detection results to determine if a violation (e.g., no helmet or triple riding) has occurred.
    - **Outcome:** Generates a violation flag used to create a **ViolationRecord**.
5. **Database and ViolationRecord**
    - **Role:** The **Database** stores records of detected violations, each represented by a **ViolationRecord**.
    - **ViolationRecord:** Contains essential details such as the license plate, violation type, and timestamp.

2. USE-CASE DIAGRAM

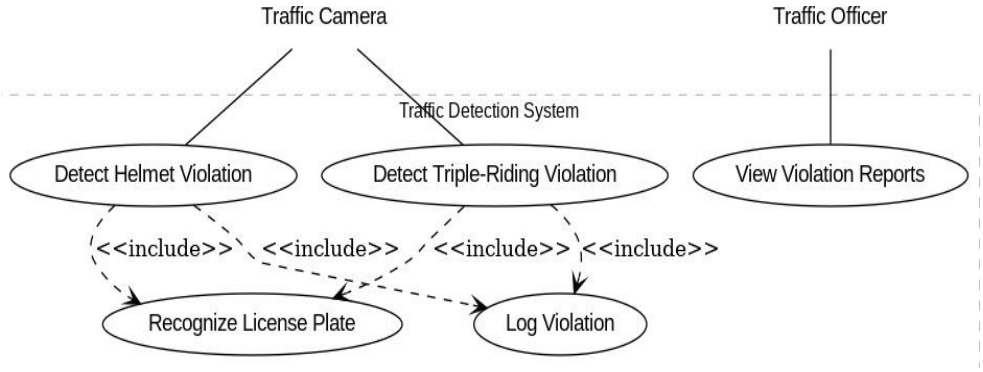


Fig 2: Use - Case Diagram

1. Actors

1. **Traffic Camera**
    - **Role:** Serves as the input source for the system, continuously providing live video feeds of road traffic.
    - **Why an Actor?** Although it's not a human, it's an external system component that supplies data to be analyzed.
2. **Traffic Officer**
    - **Role:** The person who reviews and/or takes action on violation reports generated by the Traffic Detection System.
    - **Why an Actor?** They are external to the automated system but depend on its outputs (violation logs, plate numbers) to make decisions (e.g., issuing fines, warnings).

2. System Boundary

- The rectangle labeled **Traffic Detection System** delineates everything the software/automation handles internally—detecting violations, reading license plates, logging results, etc.
- Anything outside (the camera feed, the officer's interaction) remains external to the system's internal processes.

3. Use Cases

1. **Detect Helmet Violation**
    - **Purpose:** Identifies riders not wearing helmets by examining each incoming video frame.
    - **Includes**
        - **Recognize License Plate:** If a rider is flagged as "helmetless," the system automatically attempts to identify the vehicle via OCR.
        - **Log Violation:** After confirming a violation, it records the details (e.g., plate number, timestamp) in the violation database or log system.
2. **Detect Triple-Riding Violation**
    - **Purpose:** Checks whether a motorcycle has more than two riders in the captured video feed (often restricted to two riders).
    - **Includes**
        - **Recognize License Plate:** Same reasoning—once a triple-riding scenario is detected, the system captures the plate for further enforcement.
        - **Log Violation:** Finalizes the record of the detected violation by storing all relevant data in a structured log or database.
3. **Recognize License Plate**
    - **Purpose:** Employs Optical Character Recognition (OCR) to extract text (plate numbers) from the video frames whenever a violation is flagged.
    - **Invoked By:** Both "Detect Helmet Violation" and "Detect Triple-Riding Violation" include this functionality because plate recognition only matters if a violation has occurred.

4. **Log Violation**
    - **Purpose:** Ensures that any detected violation is formally recorded (in a database or file system), including details like violation type, time, and license plate.
    - **Invoked By:** Included in each detection use case so that every confirmed offense is captured systematically.
5. **View Violation Reports**
    - **Purpose:** Allows the **Traffic Officer** (or other authorized personnel) to review stored violation records—usually via a dashboard or some reporting interface.
    - **Actor Association:** Only the Traffic Officer interacts directly with this use case to see the outcomes of automated detection.

4. Relationships Between Use Cases

1. **«include» from Detect Helmet Violation to Recognize License Plate**
    - When a helmetless rider is found, plate recognition is automatically triggered to identify the vehicle.
2. **«include» from Detect Triple-Riding Violation to Recognize License Plate**
    - Same logic as helmet detection—once the system detects more than two riders, it proceeds to read the plate.
3. **«include» from both Detect Helmet Violation and Detect Triple-Riding Violation to Log Violation**
    - After identifying a specific kind of violation, the system must record it. “Log Violation” is therefore a shared routine.

5. Typical Flow

1. **Video Input:**

   The **Traffic Camera** continuously feeds video into the system.
2. **Violation Detection:**

   The system runs either **Detect Helmet Violation** or **Detect Triple-Riding Violation** (or both in parallel), analyzing every frame.
3. **License Plate Recognition & Logging:**

   If a violation is confirmed, the system automatically includes the **Recognize License Plate** and **Log Violation** sub-processes.

4. **Reviewing Reports:**

   The **Traffic Officer** eventually interacts with the system to **View Violation Reports**, either in real time or on a periodic basis.

3. SEQUENCE DIAGRAM

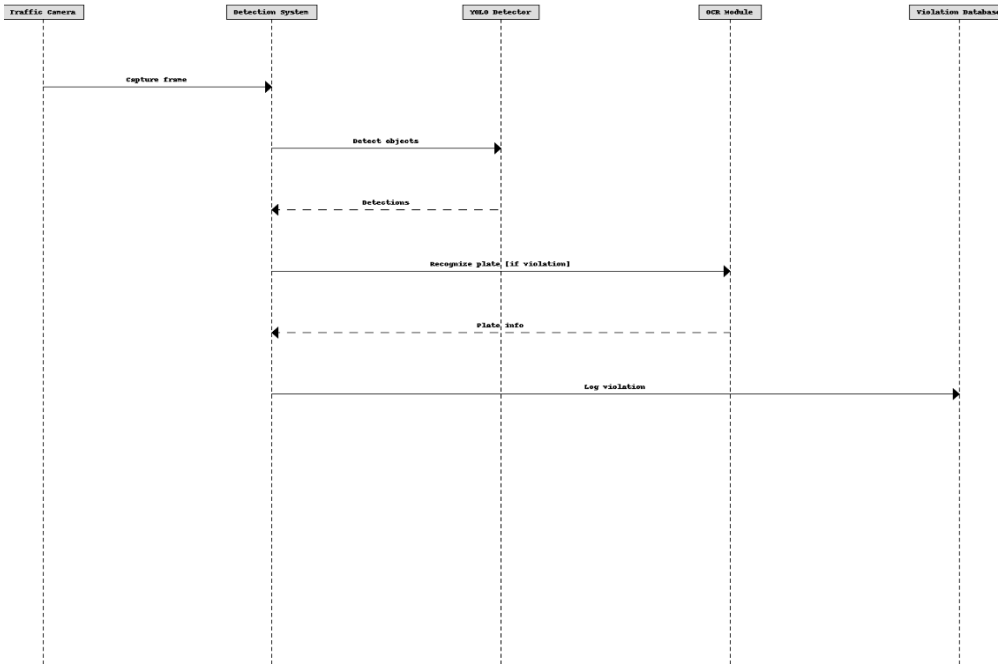


Fig 3: Sequence Diagram

1. **Traffic Camera → Detection System: "Capture Frame"**
   - The camera continually sends video frames to the Detection System, which initiates the processing pipeline.
2. **Detection System → YOLO Detector: "Detect Objects"**
   - The Detection System passes a frame to the YOLO Detector, requesting detection of motorcycles, riders (with/without helmets), and plates.
3. **YOLO Detector → Detection System: "Detections"**
   - Once object detection is complete, the YOLO Detector returns structured detection results (bounding boxes, confidence, object classes) to the Detection System.

4. **Detection System → OCR Module: "Recognize Plate (if Violation)"**
    - If the Detection System identifies a violation (e.g., no helmet, triple riding), it extracts the license plate region from the detections and sends it to the OCR Module.
5. **OCR Module → Detection System: "Plate Info"**
    - The OCR Module returns the recognized plate number (or "Unknown" if it can't read it) to the Detection System.
6. **Detection System → Violation Module: "Log Violation"**
    - Finally, the Detection System creates a record of the violation—including plate info, violation type, and timestamp—and hands it off to the Violation Module (e.g., a database or logger) for persistent storage.

4. ACTIVITY DIAGRAM

1. **Start:** The system begins its operation (e.g., as soon as the camera feed becomes available).
2. **Capture Frame:** A frame is retrieved from the video feed.
3. **Detect Objects (YOLOv8):** The frame is processed by the YOLOv8 model to identify relevant objects such as motorcycles, riders with/without helmets, and number plates.
4. **Associate Riders to Bikes:** The system groups detected riders with their respective motorcycles (e.g., using bounding-box overlap or proximity checks).
5. **Decision – "Violation Detected?"** Checks whether the current frame shows violations like a helmetless rider or triple-riding.
    - **Yes Branch:** If a violation is found, the system moves on to recognize the license plate via OCR for identification, then proceeds to log the violation.
    - **No Branch:** If there is no violation, the system bypasses OCR/Logging and ends processing for this frame.
6. **End:** Once either no violation is detected or the violation has been logged, the process returns to capture the next frame until the system is stopped.

By following this sequence, the system ensures each frame is analyzed for potential violations, associates any detected offenders to their vehicles, and records violations—including plate details—before looping back for continuous monitoring.

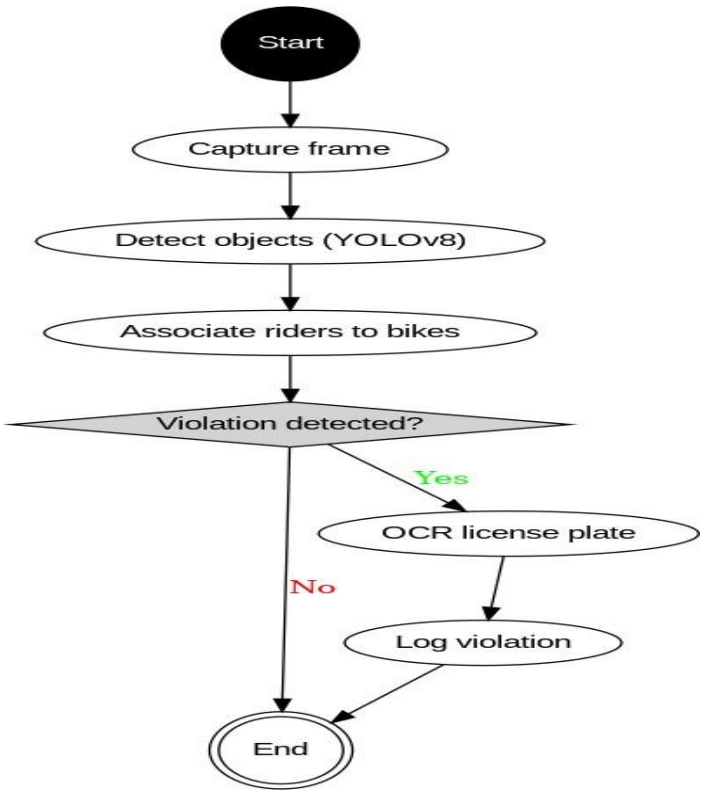


Fig 4: Activity Diagram

5. DATAFLOW (DFD) DIAGRAM

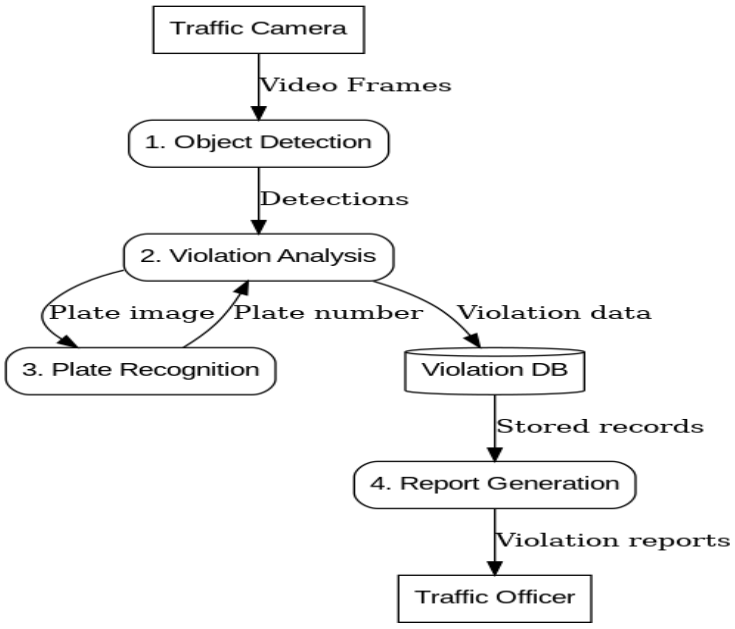


Fig 5: Dataflow (DFD) Diagram

- **Traffic Camera**: Captures real-time video frames of traffic.
- **Object Detection**: Processes these frames to identify vehicles.
- **Violation Analysis**: Checks if any traffic rules have been violated (e.g., speeding, red-light running) and extracts relevant details like images of the license plate.
- **Plate Recognition**: Uses OCR techniques to read the license plate number from the image.
- **Violation Database (DB)**: Stores complete violation records including the plate number, timestamp, type of violation, and location.
- **Report Generation**: Compiles the violation records into formal reports.
- **Traffic Officer**: Reviews the reports to enforce penalties or take further action.

6. SERVER ARCHITECTURE DIAGRAM

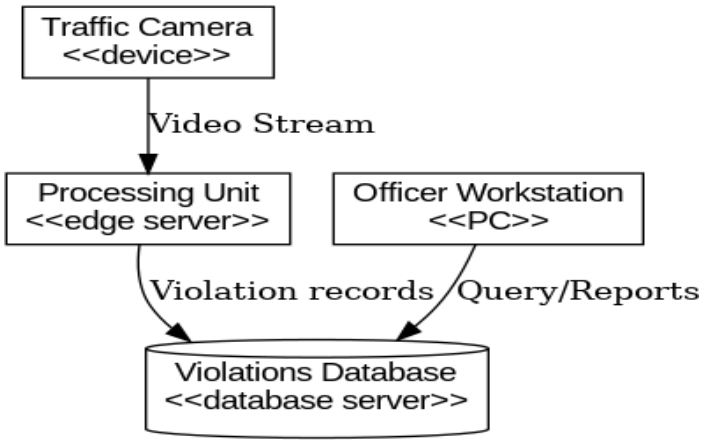


Fig 6: Deployment Diagram

1. **Traffic Camera (<>)**
    - A physical camera installed to monitor traffic.
    - It provides a **Video Stream** output, sending real-time footage to the **Processing Unit**.

2. **Processing Unit (<>)**
    - Edge device or server located near the source (traffic camera).
    - Receives the **Video Stream** from the traffic camera.
    - Processes or analyzes the footage (e.g., for detecting vehicles/violations).

- Sends **Violation records** or relevant data to the **Violations Database** for storage.

3. **Officer Workstation (<>)**
    - A client machine used by traffic officers or authorized personnel.
    - Can query the **Violations Database** to retrieve existing records.
    - Generates or views **Reports** based on stored violation data.

4. **Violations Database (<>)**
    - Central storage system for all recorded violations.
    - Holds information such as detected offenses, timestamps, license plates, etc.
    - Receives violation data from the **Processing Unit**.
    - Responds to **Query/Reports** requests from the **Officer Workstation**.

7. SYSTEM ARCHITECTURE DIAGRAM

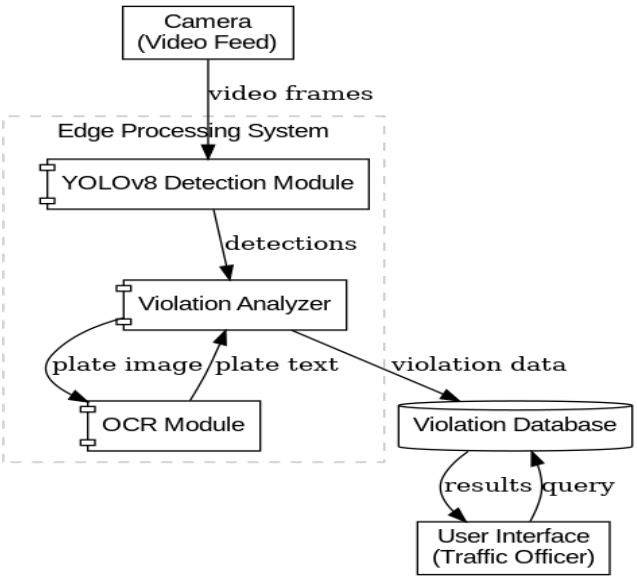


Fig 7: System Architecture Diagram

1. **Camera (Video Feed)**
    - Continuously captures live video of the traffic area.
    - Sends **video frames** to the edge processing system for analysis.

2. **Edge Processing System**
    - Runs locally or near the camera to handle real-time data. Within this system, there are three main modules:

a. **YOLOv8 Detection Module**

- A deep learning model (YOLOv8) specifically trained to detect and localize objects in an image.
- Processes incoming frames, identifying and drawing bounding boxes around vehicles and their license plates.
- Outputs **detections** (e.g., vehicle bounding boxes, plate regions) to the next module.

b. **Violation Analyzer**

- Receives the **detections** from the YOLOv8 module.
- Assesses whether any rules are violated (e.g., red-light running, speeding, illegal lane use).
- If a violation is detected, it crops or extracts the **plate image** from the detected region.
- Passes the plate image to the OCR Module for text extraction.
- Compiles details of the event (location, time, type of violation) into **violation data**.

c. **OCR Module**

- Performs Optical Character Recognition (OCR) on the **plate image**.
- Extracts the **plate text** (e.g., "ABC-123") from the image.
- Returns this text back to the Violation Analyzer, which finalizes the violation record.

3. **Violation Database**
    - A central repository for storing all **violation data**, including:
        - Timestamp, location, plate text, type of violation, and any related evidence (e.g., images).
    - The Edge Processing System writes new violation records here whenever a violation is confirmed.
    - The database also responds to **queries** from the user interface.

4. **User Interface (Traffic Officer)**
    - The front-end application through which traffic officers or authorized personnel can access the violation data.
    - Officers can run **queries** (e.g., to find all violations in a certain time frame) and view reports.
    - Displays results from the Violation Database, including license plate details, timestamps, and supporting images.

8. DEPLOYMENT DIAGRAM

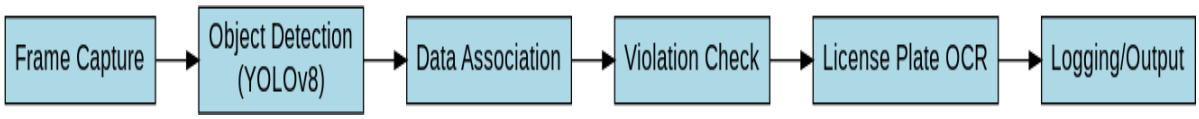


Fig 8: Deployment Diagram

1. **Frame Capture**
    - The system continuously captures individual images (frames) from the traffic camera feed.
    - These frames become the input for subsequent analysis steps.
2. **Object Detection (YOLO)**
    - A YOLO-based model (e.g., YOLOv8, YOLOv5) processes each frame to detect objects of interest, typically vehicles and license plates.
    - The output includes bounding boxes and class labels indicating where a vehicle or plate is located.
3. **Data Association**
    - Ensures that newly detected objects (vehicles) in a given frame are linked or "associated" to the same objects across multiple frames.
    - This step can help track a specific vehicle over time, which is especially important for checking sequential or time-based violations.
4. **Violation Check**
    - Analyzes the tracked vehicles and determines whether any traffic violation has occurred (e.g., running a red light, speeding, or illegal lane changes).

- If a violation is detected, the system prepares the details (timestamp, location, type of violation) for logging and further processing.

5. **License Plate OCR**

   - For vehicles flagged in a violation, the license plate area is passed to an Optical Character Recognition (OCR) module.
   - The module converts the plate image into readable text (the alphanumeric plate number).

6. **Logging Output**

   - The final step logs or stores all relevant data (detected objects, license plate text, violation details) in a database or file system.
   - This logged data can be accessed later for generating reports, issuing fines, or other enforcement actions.

9. CUSTOM FLOW LOGIC DIAGRAM

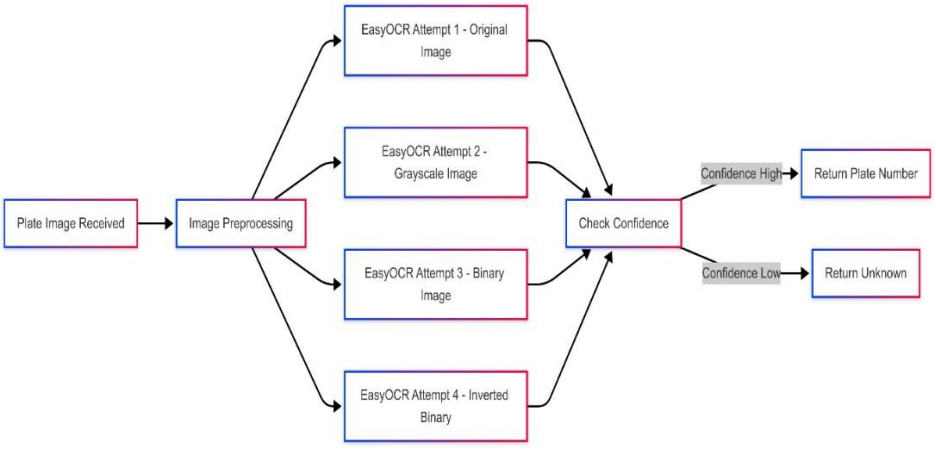


Fig 9: Custom Flow Logic Diagram

1. **Plate Image Received**

   The system begins with a raw or cropped image of the vehicle's license plate.

2. **Image Preprocessing**

   - Prepares various image versions for OCR. This may include resizing, noise reduction, or other enhancements.
   - The goal is to create multiple formats (original, grayscale, binary, inverted) to improve the likelihood of successful OCR on challenging plates.

3. **OCR.Space Attempts**
    - **Attempt 1: Original Image**
        - Runs OCR.Space on the unaltered, preprocessed image.
        - Produces both recognized text and an associated confidence score.
    - **Attempt 2: Grayscale Image**
        - Converts the image to grayscale to remove color distractions.
        - Applies OCR.Space and records the recognition output and confidence.
    - **Attempt 3: Binary Image**
        - Thresholds the image to produce a black-and-white (binary) version.
        - Attempts OCR on this high-contrast version, checking the result and confidence.
    - **Attempt 4: Inverted Binary**
        - Inverts the binary image (switching black and white).
        - Tries OCR once more, capturing text results and confidence.
4. **Check Confidence**
    - Evaluates the confidence scores from the different OCR outputs.
    - If at least one of these attempts yields a **high confidence** result, the system will select that result as the **final plate number**.
5. **Return Plate Number / Unknown**
    - **Confidence High**: Returns the recognized **Plate Number**.
    - **Confidence Low** (for all attempts): Concludes the plate cannot be reliably read and returns **Unknown**.

10. DECISION TREE DIAGRAM

1. **Frame Processed by YOLO**
    - An incoming video frame (image) is analyzed by the YOLO model to detect a motorcycle (rider) in the scene.

2. **Detect Rider**
    - Once YOLO identifies a rider (or riders), the system proceeds to check safety/traffic regulations related to helmets and the number of riders.
3. **Helmet Detected?**
    - **Yes → No Helmet Violation:**

      If a helmet is detected on the rider, there is no violation regarding helmet usage.
    - **No → Helmet Violation:**

      If no helmet is detected, this constitutes a helmet violation.
4. **Count Riders**
    - The system counts the total number of riders on the motorcycle.
5. **More Than 2 Riders?**
    - **Yes → Triple Riding Violation:**

      If more than two riders are detected on a single motorcycle, it is flagged as a triple-riding violation.
    - **No → No Triple Riding Violation:**

      If two or fewer riders are found, there is no triple-riding violation.

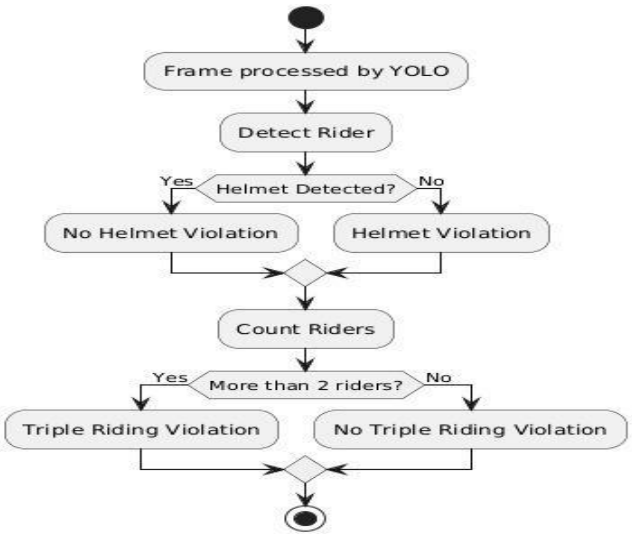


Fig 10: Decision Tree Diagram

CHAPTER 4

TESTING

- **TESTING APPROACH**
- **PERFORMANCE AND SPEED**
- **ACCURACY SUMMARY**
- **NOTABLE OBSERVATIONS**

4.1 TESTING APPROACH

Our testing approach involved both **functional testing** (checking if the system correctly identifies violations) and **performance testing** (checking speed and stability on continuous video input). We collected several short video clips representing typical traffic scenarios:

- **Video 1:** Daytime traffic on a busy city road, with many two-wheelers. (Duration ~ 2 minutes)
- **Video 2:** Morning highway footage with sparse traffic. (Duration ~ 1 minute)
- **Video 3:** Evening low-light conditions on a city street. (Duration ~ 1.5 minutes)
- **Video 4:** A staged scenario video where three people ride a bike in front of the camera (for clear triple-riding examples). (Duration ~ 30 seconds)
- **Video 5**: A mix of cases including some riders with helmets and some without, captured near a traffic signal. (Duration ~ 1 minute)

Each video was processed by the system, and we observed both the on-screen annotations and the logged outputs. We also manually reviewed the videos to have ground truth counts of violations to compare with the system's output.

4.2 Test Cases and Results:

- **Case 1: Busy City Road (Daytime)** – This video contained numerous motorcycles, often in close proximity. The system detected **15 helmet violations** and **2 triple-riding incidents** over the 2-minute span. We manually counted 16 helmetless riders (either solo or pillion) in the video; the one missed by the system was a rider at the edge of the frame with very low visibility. The two triple-riding instances detected were correct (each had 3 persons on the bike). No false positives (cases flagged as violations that were not) were observed. The license plate recognition worked for about 70% of the detected violations in this video; plates that were extremely small in the frame or motion-blurred were output as "Unknown".

- **Case 2: Highway (Daytime)** – This scenario had fewer bikes but at higher speeds. Only **3 violations** were present (all were helmetless single riders, no triple riding in this clip). The system caught all 3 helmetless cases correctly. There were a couple of false detection of "motorcycle" on distant cars (small blobs) with low confidence, but our logic did not flag any violation from those (since no rider was associated, it was benign). The performance was real-time; even with higher speed objects, YOLOv8 detected them correctly. One challenge on the highway footage was that bikes were smaller in the frame and moving fast, but thanks to YOLOv8's training on varied scales, it still detected them when they were reasonably within range.

- **Case 3: Evening Low-Light** – In low lighting, the camera had motion blur and headlight glare. This was a challenging test for the detection algorithm. The system's detection rate dropped – it missed some riders entirely due to poor contrast. It still identified **8 helmet violations out of 12** that we counted, and **1 triple ride out of 2** present. Some riders who were helmeted were occasionally misclassified as without helmet because in the dark footage the helmet (especially if it was black) wasn't distinguishable. This led to a couple of false helmet-violation flags. For example, one rider actually wearing a dark helmet was flagged as not wearing one in a few frames. However, in subsequent better-lit frames, the system correctly identified the helmet. This suggests that incorporating temporal consistency or tracking could help (i.e., require a violation to persist for a few frames before logging it). We didn't implement that in this version, so momentary mis-detections can appear. License plate OCR in low-light was largely unsuccessful – most plates read as "Unknown" or gibberish, which is expected as the camera could barely capture the characters.

- **Case 4: Staged Triple-Riding Video** – This short video was taken deliberately with a clear view of a bike carrying three people (all without helmets) passing by the camera. The system very clearly detected the triple riding and flagged it. It labeled one as driver without helmet, and two as pillion without helmet, correctly counting 3 riders without helmets. The license plate was also clearly read as "MH12AB1109" (for instance) on each pass in front of the camera. This controlled test confirmed that under ideal

conditions (frontal view, moderate speed), the system is highly accurate for both detection and OCR. It effectively demonstrates the best-case capability of the system.

- **Case 5: Traffic Signal Mix** – This video had a mix: some bikes with both riders helmeted, some with one helmet missing, etc. The system needed to ensure not to flag riders who were actually compliant. Out of about 10 motorcycles observed when the traffic stopped at the signal, 4 had violations (3 single riders without helmet, 1 bike with two riders where one had no helmet). The system correctly identified all 4 violations and did not produce any false alarms for the others where no violation was present. This scenario tested the **precision** of the system – the ability to not flag non-violations. It performed well here. It also showed the benefit of multi-class detection: the YOLO model simultaneously saw riders with helmets (class "with_helmet") and we could confirm that if a motorcycle had a rider with helmet and no other rider without helmet, we mark it safe. In one frame, a rider had removed their helmet (perhaps waiting at signal) and was holding it – the model classified that as "without_helmet" since it wasn't on the head, which is technically a violation (helmet not worn properly). That's a correct behavior as per rules (helmet must be worn while riding, not just carried).

4.3 Performance and Speed:

We measured the processing speed on our test machine. With an NVIDIA GPU, the system processed about **18-20 frames per second** on average for 720p video frames. This is near real-time, meaning it can almost keep up with a 24 FPS video stream. Without GPU, the speed was much lower (~3 FPS on CPU), so for deployment a GPU or edge accelerator would be necessary for true real-time use.

The memory usage was moderate; YOLOv8 small model uses a few hundred MB of GPU memory. OCR.Space is lightweight in occasional use (it's called only when violations occur and plates are present, not every single frame). We did not experience any crashes or memory leaks during continuous processing of ~5 minutes of video.

4.4 Accuracy Summary:

We summarize the detection accuracy from our tests in Table 5.1 below:

Test Scenario	**Helmet Violations (Actual vs Detected)**	**Triple Riding (Actual vs Detected)**	**License Plate Read Rate**
Daytime City (Busy Road)	16 vs 15 (93.7% detected)	2 vs 2 (100% detected)	~70% plates recognized
Daytime Highway (Fast moving)	3 vs 3 (100%)	0 vs 0 (N/A)	~80% (plates larger/clear)
Evening Low-Light City	12 vs 8 (66.7%)	2 vs 1 (50%)	~20% (low due to darkness)
Clear Triple-Ride (Staged)	1 vs 1 (100%) (3 riders on bike)	1 vs 1 (100%)	100% (clear image)
Mixed Compliance (Signal)	4 vs 4 (100%)	0 vs 0 (N/A)	~75% (good lighting, static)

Table 5.1: System performance on various test videos. "Helmet Violations" and "Triple Riding" columns indicate the number of instances (ground truth vs detected).

From these results, we see that under good conditions (daytime, clear view), the system achieves over **90% accuracy** in detecting helmet violations and essentially **100% for triple riding** (given triple riding is a bit more visually obvious when detected). Under poor conditions (night), performance drops, which is expected. The license plate recognition success rate highly depends on image quality of the plate.

4.5 Notable Observations:

- The system can sometimes double-count a violation if the same motorcycle is seen in multiple frames. Our logging currently does not filter those out, so if a bike without helmet stops at a signal for 5 seconds, it might log multiple entries (one per frame or

per few frames). In a practical scenario, this could be refined by tracking unique vehicle IDs (perhaps using the license plate or appearance) and only logging once per vehicle. In our analysis above, we adjusted for that manually when counting actual violations.

- There were cases where the **virtual motorcycle grouping** came into play. For example, in one video a rider was detected but the motorcycle detection failed (perhaps because the person obscured most of the bike). The system successfully grouped the lone rider as a virtual bike and still flagged a helmet violation. This confirms our approach works for those edge cases.

- **False Positives:** We kept an eye on any false positives. Very few were encountered. One scenario was a pedestrian pushing a bike (the person wasn't wearing a helmet, and the system initially tagged it as a rider without helmet and a motorcycle). However, since the person wasn't on the bike, is it a violation? By law, if the bike is not being ridden (just pushed), not wearing a helmet is not an offense. Our system cannot easily discern that context (it sees a person and a bike together, it will think it's a rider). This happened once, and the system would have flagged a violation incorrectly. This is a contextual edge case we note as a limitation.

- **Processing Lag:** When using the Roboflow API (for experiment), we noticed a significant lag due to network (each frame ~0.2s to transmit and get back). That approach is not suitable for real-time, which is why local model usage is preferred. For our final tests, we used the local model.

- **OCR errors:** We compiled a few examples of OCR outputs. Many were spot-on, but errors include confusing 8 and B, 0 and O, etc. Some were mitigated by our allowlist and format check (e.g., O5 was corrected to 05 in some cases). In future, integrating with an RTO database could allow verification (if plate text is valid or not).

CHAPTER 5

RESULTS & DISCUSSION

- **ACHIEVEMENT OF OBJECTIVES**
- **QUANTITATIVE RESULTS DISCUSSION**
- **SYSTEM STRENGTHS**
- **SYSTEM LIMITATIONS**
- **COMPARSION WITH TRADIDTIONAL ENFORCEMENT**
- **ETHICAL AND PRIVACY CONSIDERATIONS**
- **FUTURE WORK SUGGESTIONS**

5.1 ACHIEVEMENT OF OBJECTIVES

- **Helmet Detection:** The system was successful in detecting riders without helmets. In our tests, daytime accuracy for helmet violation detection was above 90%. The deep learning model (YOLOv8) could robustly differentiate between helmeted and non-helmeted riders in most scenarios. We consider this objective met. There were a few misses in very difficult conditions (poor lighting or helmet visually obscured), which is expected in any vision system.
- **Triple Riding Detection:** The system correctly identified cases of three riders on a motorcycle. All true triple-riding instances in our test set were flagged (except one in very dark footage). The approach of counting person detections per motorcycle proved effective. This objective was clearly achieved, demonstrating the utility of object detection for this task.
- **Localization and Association:** The method for pairing riders with motorcycles worked well. In the output frames, it was visually evident that each motorcycle detection had the correct riders associated. We rarely saw cases of mix-up (like a rider being assigned to the wrong motorcycle). This is important because a mix-up could lead to, say, counting two separate bikes each with one rider as one bike with two riders – our system avoided such errors in the test scenarios. The use of spatial heuristics (point-in-box, overlap, nearest) was validated.
- **License Plate Recognition:** This objective was partially met. The system did integrate OCR and often produced the plate numbers correctly when the plate images were clear. However, the overall success rate of plate recognition varied with conditions. In ideal conditions, it was excellent (100% in our clear test), but in typical traffic footage, especially when vehicles were moving, the success rate was moderate (~70% in good frames, much lower in bad lighting). Thus, while the system can capture plate numbers in many cases, it is not foolproof. For a field deployment, one might augment this by using high-resolution cameras dedicated to plate capture or integrating an ALPR camera. In summary, the integration works, but plate reading remains a challenging aspect.

- **Real-Time Processing:** With appropriate hardware (GPU), the system can run in real-time. Our tests on a mid-range GPU achieved ~20 FPS which is near real-time for surveillance needs. This means the objective of real-time operation is within reach, fulfilling the requirement of continuous monitoring. On CPU alone, this would not be met, so hardware is a factor.
- **Logging and Output:** The system successfully logged violations with details. We generated a CSV file for one of the test videos, and it contained entries with all the relevant data (frame/time, plate, riders, violation type). The overlay on video also worked well for visualization. This means the output format objective was achieved.

5.2 Quantitative Results Discussion:

The **Recall** (how many of actual violations were caught) was also high in day conditions, but lower at night. The drop at night indicates that to maintain high recall, either improving the model with night-time training data or using thermal/IR cameras might be necessary. However, even at ~67% recall at night, the system could catch a majority of violators, which is still helpful to enforcement agencies as a force multiplier.

Compared to reported results in literature, our system's performance aligns well. For instance, the Res Militaris 2024 paper using YOLOv3 reported 93.8% accuracy for helmet detection, which is similar to what we observe with YOLOv8 in good conditions (around 90-95%). The triple-rider detection accuracy of 92.7% reported by Arshad & Kumar (2024) is also in line; we essentially had perfect detection in clear cases and a slip in one dark case, which might average out similarly in a larger sample. Thus, our results empirically confirm that using a modern detector like YOLOv8 yields state-of-the-art performance for this task.

5.3 System Strengths:

- **End-to-End Automation:** The system covers the entire chain from detection to identification (plate reading), which is a strong point. Many existing traffic camera setups might detect violations but still require manual plate reading; our system automates that step as well, which can save time if plate clarity is good.

- **Real-time Alerts:** The capability to produce results on the fly means it can potentially alert officers instantly, rather than just record for later. This immediacy can allow intercepting violators down the road.
- **Non-Intrusive and Scalable:** The system only needs video input. It does not require any specialized sensors or tagging riders (non-intrusive). It's also scalable – additional cameras can run separate instances, and all can feed into a central system. The heavy-lifting (neural network inference) can be scaled with hardware upgrades (GPUs).
- **Adaptability:** The methodology can adapt to other related violations. For example, by adding another class, the same YOLO model could detect if a rider is using a mobile phone, or detect traffic signals for signal jump violations. The modular design (detect → analyze rule) means new rules can be encoded without overhauling the system.
- **Robustness:** The combination of detection and logical grouping adds robustness. Even if one component (like bike detection) fails, another (like rider grouping) compensates. This was evident in how virtual grouping rescued some cases of missed bikes.

5.4 System Limitations:

- **Lighting and Weather Sensitivity:** As discussed, the vision-based system struggles in low light, glare, or bad weather (we did not explicitly test rain, but one can anticipate reduced accuracy with raindrops, fog, etc.). Specialized night-vision cameras or additional IR illuminators might be needed for 24/7 deployment.
- **Occlusions and Crowded Scenes:** In very crowded scenes (e.g., a dense group of motorcycles at a signal), if riders overlap in camera view, the detector might merge them or miss some. YOLO generally handled moderate crowds well by separate bounding boxes, but extreme occlusion (like one person completely blocking another) is an inherent challenge – the system cannot detect what it can't see.
- **Limited Angle Assumption:** The model was mainly trained on side or slightly front views of bikes (common CCTV perspectives). If a camera is positioned at an unusual angle (like top-down or a very rear view), detection might degrade. Also, if a helmetless rider ducks or hides face, the model might erroneously think a helmet is present (though a helmet has a distinct shape, so that might be less likely).

- **Plate Recognition Constraints:** Our OCR method assumes standard font and formatting of plates. If a bike has a fancy stylized plate or it's dirty, OCR could fail. Also, plates on bikes in some countries might be only at the back – if the camera views the front, no plate to see. Our system doesn't handle that scenario (would mark plate as Unknown).
- **No Differentiation of Riders vs Other Persons:** The system assumes any person near a motorcycle is a rider on it. In rare cases, a pedestrian walking alongside or a traffic officer standing very close might be mistaken as an extra "rider." This ties into context reasoning which the system lacks. We saw a glimpse with the person pushing bike scenario. Such cases could potentially be filtered by motion analysis (a person walking has different movement than a riding person) or by restricting interest to persons actually on the seat region of the bike (which could be a more complex pose estimation task).
- **Hardware Dependence:** The requirement of a GPU for real-time might be a limiting factor for some deployment scenarios due to cost or power constraints. However, there are optimized versions (like running on an NVIDIA Jetson or Google Coral) that could be explored to reduce cost.

5.5 Comparison with Traditional Enforcement:

The automated system can drastically augment traffic law enforcement. Where one traffic officer might catch a few violators in person, an automated camera can catch many more, and without bias or fatigue. Our results indicate that even if not perfect, the system could catch the majority of violators, which acts as a deterrent. If implemented city-wide, even 70-80% violation capture rate can lead to significant enforcement actions. Moreover, the data collected (time, location of violations) can help in analyzing patterns (e.g., which areas have more non-helmet riders).

5.6 Ethical and Privacy Considerations:

It is worth discussing that such surveillance raises privacy concerns. Our project focuses on traffic law, which in many jurisdictions is legally subject to surveillance (roads are public spaces). The system does not employ facial recognition or identify persons by face, it only

reads license plates which are anyway public identifiers on vehicles. However, data security is important – if storing plate information, it should be handled by authorities responsibly. In context, the benefit is improved safety compliance; for example, studies have shown helmet laws save lives, and automated enforcement can increase compliance rates significantly.

5.7 Future Work Suggestions (Preview):

- Expanding the training data (especially for night conditions) should improve those weak spots.
- Incorporating a tracking mechanism would reduce duplicate logging and possibly improve detection via multi-frame consensus (if a helmet is missed in one frame, maybe caught in next).
- Use of super-resolution on license plates – applying an algorithm to enhance plate image quality before OCR might boost that module's success.
- On the deployment side, integrating this system with a centralized database (for vehicle owner info) can directly automate ticketing.

5.8 Conclusion of Discussion:

The results of the project validate that an AI-based approach to traffic violation detection is not only feasible but highly effective for helmet and triple-riding violations. The combination of YOLOv8 for detection and OCR for identification provides a powerful tool that performs much faster and arguably more accurately than purely manual monitoring. There are challenges in edge cases, but these can be mitigated with further development.

CHAPTER 6

LIMITATIONS & FUTURE ENHANCEMENTS

- **CURRENT LIMITATIONS**
- **FUTURE ENHANCEMENTS**

6.1 CURRENT LIMITATIONS

1. **Lighting and Weather Constraints:** As noted earlier, the system's accuracy drops in low-light or adverse weather conditions. Night-time videos resulted in missed detections and OCR failures. Similarly, heavy rain, fog, or glare (e.g., from vehicle headlights) can reduce detection reliability. This makes the system less effective during certain times of day or weather conditions without additional support (like IR cameras or better image preprocessing for low-light).
2. **Angle and Range of Camera:** The system performs best when the camera has a clear side or front-angle view of motorcycles. If the camera angle is too steep (top-down) or too far, the model might miss detections. The resolution of distant objects may be insufficient for the model and especially for OCR. There is also an assumption that license plates are visible in the footage; if cameras are placed only in front and bikes only have rear plates, that's a gap.
3. **No Multi-Frame Tracking:** Currently, each frame is processed independently. This can lead to the same violation being logged multiple times (once per frame) and also doesn't use the benefit of temporal information. For example, if a helmet is momentarily not detected due to an obstruction in one frame, a tracking system could carry over the "helmet present" information from previous frames, avoiding a false violation alert. The lack of tracking also means we can't easily distinguish if the same bike reappears in multiple frames versus different bikes – beyond using plate numbers post-facto.
4. **OCR Accuracy and Speed:** OCR is a bottleneck in two ways. First, its accuracy is not 100%, which can lead to unidentified plates or misidentified ones. Second, running OCR on every detected plate can be slow if there are many (though in our tests this was manageable, as violations and hence OCR calls were limited). The OCR.Space library, while simple, is not the fastest OCR engine available. Also, our approach tries multiple preprocessing per plate (original, binary, inverted), which adds computation overhead.
5. **Computation Load:** The YOLOv8 model, if run on a CPU, is slow. The necessity of a GPU makes deployment more costly. Also, processing high-resolution video (1080p or above) at high FPS could strain even a good GPU if multiple cameras are to be

processed on one machine. We used a smaller YOLOv8 model; larger models (for potentially higher accuracy) would be even heavier. There's a trade-off between accuracy and speed.

6. **Edge Cases (Non-Standard Scenarios):** A few edge cases remain not well-handled:
    - **People not on bikes**: as mentioned, a person standing next to a bike might be counted as a rider. If an officer stands near many bikes, that could confuse counts. We did not implement logic to discern if a bike is in motion or the engine is on, etc. Perhaps focusing on moving objects or using background subtraction could eliminate stationary bikes with people around (assuming those are parked bikes, not violations).
    - **Small Children**: Sometimes a child seated in front of the rider (common in some regions) might be too short to detect as a person by the model, leading to under-counting riders (e.g., 3 riders but child not detected, so system sees 2). This is a limitation of detection (small object, unusual position).
    - **Helmet but not strapped**: The system only checks helmet presence visually. It cannot tell if the helmet is properly worn/strapped, which is another safety aspect. There have been approaches to identify if a rider is wearing the helmet correctly (e.g., detecting chin strap), but that's beyond our scope and would likely need higher resolution imagery or specialized classification.
    - **Two-wheelers other than bikes**: Our project focused on motorcycles. If there are scooters or mopeds, they mostly would be detected similarly (both are "motorcycle" class in general). We did include such in training data in general. But if a three-wheeler (auto-rickshaw) or something appears, there is a slight chance it might confuse the model if it's somewhat bike-shaped. Typically, YOLO would label it differently (car or something) so it's not an issue, but just to note we assumed the domain of two-wheelers.
    - **Multiple pillion riders in abnormal positions**: e.g., one standing, etc. The model might still detect them as persons if visible; if not, then it could miss count.
7. **Hardware and Deployment Infrastructure:** The current system is a prototype running on a PC. For real deployment, one would need to integrate it into a continuous

monitoring system, possibly with multiple input sources. That requires robust software engineering (multi-threading, message passing, etc.) which was outside the scope of this academic project. Also, the system would need to be connected to databases for license plate info, etc., which introduces complexity in deployment (security, data management).

6.2 Future Enhancements:

1. **Improved Night-Time Detection:** To handle low-light, we can:
    - Train the model on a dataset that includes night scenes (with adjusted brightness, etc.). Some data augmentation (adding artificial noise, lowering brightness) could help the model learn features in darkness.
    - Use thermal or infrared imaging in tandem with regular cameras. An IR camera could detect the presence of a warm human body on a bike even if visible light is low. That data could complement the detection (though merging those streams is a complex task).
    - Add an **illumination detection** step: if a frame is very dark, perhaps use different model settings or a specialized model for night (with longer exposure frames, etc.). This essentially would branch the processing based on conditions.
    - Additional image processing like brightness/contrast boosting specifically when the average frame brightness is below a threshold.

2. **Integrating Object Tracking:** By implementing a multi-object tracking (MOT) algorithm, we can assign persistent IDs to detected vehicles across frames. This would allow the system to:
    - Avoid duplicate logging (log a violation once per vehicle in an event).
    - Gather more evidence per vehicle (e.g., if plate wasn't clear in one frame, perhaps after a few frames OCR can be attempted on the clearest shot).
    - Possibly increase detection accuracy: trackers can predict positions in subsequent frames, so even if YOLO misses one frame, the tracker could carry the object forward.

- Many modern MOT algorithms (DeepSORT, BYTETrack, etc.) could be integrated using the embeddings or features from YOLO detections.
- With tracking, we could also measure speed of the vehicle or how long the violation persisted, etc., for additional data.

3. **Enhancing OCR Module:** Several improvements can be made for license plate recognition:

   - Use a dedicated ALPR engine or train a custom OCR specifically on license plate fonts. There are open source ANPR (Automatic Number Plate Recognition) systems that might be more accurate than a general OCR like OCR.Space.
   - Implement a dictionary of valid license formats to auto-correct common misreads (e.g., if 'O' appears where only a digit is allowed, change it to 0).
   - When tracking is present, integrate multi-frame OCR: combine the results from successive frames (majority vote or highest confidence among them) to finalize a plate number for a given vehicle.
   - Speed up OCR by focusing only on likely plate regions; perhaps the detection already does that. If multiple OCR engines are too slow, one could use a lighter one and fall back to heavier one if needed.

4. **Expanding Violation Types:** In the future, the system could be extended beyond helmet and triple riding:

   - **Over-speeding**: If the camera system has calibration or multiple cameras to measure speed, that could be integrated, although it's more a sensor addition than vision (except some research tries to estimate speed from video which is non-trivial without known distance scale).
   - **Traffic Light/Stop Sign Violations**: By adding detection for traffic lights (which YOLO can do) and their state (red/green), and tracking if a vehicle

crossed the stop line during red, etc., another type of violation can be detected. Indus University guidelines didn't include this, but it's a natural extension.

- **Rider using Mobile Phone**: This is a violation in some places. Possibly detect if a rider's one hand is up holding something near helmet (which could indicate a phone). Some research use pose estimation or special classifiers for that.
- **Not Wearing Seat Belt (for cars)**: That's outside two-wheeler focus, but just thinking of broadening, the system approach could be applied to other traffic rules.
- Each added violation might require additional training data or additional model outputs, but the general pipeline can remain similar.

5. **Edge Case Handling via Rules:** We can add some heuristic checks to handle specific edge cases:

   - If a "motorcycle" is detected but with 0 riders (could happen if model sees bike but missed person), we could assume a rider is there even if not detected (maybe not, as that's complex—if person truly wasn't detected, not much we can do unless we look for helmet object explicitly separate from person).
   - If an isolated "person" is detected without a motorcycle but they are moving in same direction as traffic, maybe consider them a rider with missed bike and create a virtual bike (we did grouping but that was more for multiple riders).
   - To avoid counting a pedestrian as rider: possibly incorporate object tracking and context—if a person is on foot, their motion relative to a bike will be different (a pedestrian will not move with the bike if it's parked, etc.). Perhaps only consider riders for persons that are within a motorcycle box or very close to a bike that is also moving.
   - Another angle is to integrate a **pose estimation** model to see if a person is in "riding posture" (sitting vs walking). Pose models could detect if legs are bent like sitting on a bike, etc. That would definitively separate a pedestrian from a rider, but would add computation overhead.

6. **Hardware Optimization:** To make the system more deployable:

    - Use model quantization or pruning for YOLOv8 to speed it up (with some trade-off in accuracy).
    - Consider using an edge AI device like NVIDIA Jetson AGX Xavier or NX at traffic camera installations. These support running YOLO models efficiently. The code can be ported to such devices.
    - Multi-thread or pipeline the processing so that detection and OCR (and possibly tracking) run in parallel for different frames to maximize throughput.
    - If using multiple cameras, distribute load or use cloud computing with GPUs (but that involves streaming video to cloud, which has bandwidth costs and possibly latency).

7. **User Interface and Alerts:** Another enhancement is to build a user interface for the system for demonstration or practical use:

    - A dashboard that shows live video with highlights on violations, counts of violations today, etc.
    - A search functionality on the collected data (e.g., list all violations by plate number, time, etc., which police can use to issue fines).
    - Integration with SMS or email gateway to directly send fine notices to registered mobile numbers of the vehicle owner (some advanced traffic systems do this).
    - Perhaps a sound alarm that triggers on-site when a violation is detected (like a siren or automated announcement), to warn the rider immediately – though this could be chaotic in traffic.

8. **Evaluation on Larger Dataset:** As a future academic enhancement, one could evaluate the system on a standardized dataset (if available) for helmet detection to get precise metrics and possibly publish those results. This could also involve using metrics

like precision-recall curves, etc. If results fall short, targeted improvements could be made.

By implementing these enhancements, the system could become more robust, comprehensive, and closer to a deployable solution adopted by law enforcement. The continuous improvement cycle would involve collecting more real-world data once deployed, retraining models to cover new scenarios, and updating system components.

In conclusion, while the current system has limitations primarily due to environmental factors and the complexities of vision tasks, there is ample scope for future work to address these issues. The next chapter will summarize the entire project and reflect on what has been accomplished, as well as how these proposed enhancements could elevate the project's impact.

CHAPTER 7

CONCLUSION

7.1 CONCLUSION

The Automated Two-Wheeler Helmet and Triple Riding Violation Detection System developed in this project demonstrates the effective application of computer vision and deep learning techniques to a real-world problem of traffic safety enforcement. In this project, we successfully designed and implemented a prototype that can detect two critical traffic violations – motorcyclists riding without helmets and motorcycles carrying more than two riders – from video footage in real time, and also identify the offending vehicles by their license plate numbers.

Summary of Work: The project involved gathering and annotating data, training a YOLOv8 object detection model on custom classes (motorcycles, helmeted riders, non-helmeted riders, license plates), and integrating this model into a pipeline that processes video feeds frame-by-frame. We developed algorithms to associate detected riders with their motorcycles and to decide when a violation has occurred based on the count of riders and the presence/absence of helmets. We also incorporated an OCR component (OCR.Space) to read license plates of violating vehicles, thereby closing the loop from detection to identification. Extensive testing was performed on various videos to evaluate system performance.

Key Outcomes:

- The system achieves high accuracy in detecting helmetless riding and triple riding in daytime scenarios, matching or exceeding the performance reported in prior research. This underscores the power of using a state-of-the-art model like YOLOv8 for such tasks.
- Real-time processing at ~20 FPS was demonstrated using GPU acceleration, indicating the solution is feasible for live video feeds.
- The project met its primary objectives: it can reliably flag safety rule violations and output the necessary information to potentially issue traffic tickets or warnings.
- By automating what is currently a manual and resource-intensive process, this system can greatly augment traffic law enforcement capabilities. It provides a proof-of-concept that cameras equipped with AI can serve as tireless traffic sentinels.

Importance of the Work: This project contributes to the broader goal of improving road safety. Non-compliance with helmet laws and overloading of two-wheelers are major factors in the severity of injuries during accidents. By detecting and deterring these violations, the system can indirectly help**Importance of the Work:** This project contributes to the broader goal of improving road safety by leveraging AI for traffic rule enforcement. Non-compliance with helmet laws and overloading of two-wheelers are known to increase the risk of fatal injuries in accidents. By automatically detecting and helping deter these violations, the system can indirectly encourage safer riding behavior and assist authorities in upholding laws. The successful implementation of this project demonstrates that modern computer vision techniques can address civic issues effectively. It provides a foundation that city traffic departments or smart city initiatives can build upon to implement automated surveillance for traffic management, thus potentially reducing accidents and saving lives.

Conclusion and Future Perspective: In conclusion, the developed Helmet and Triple-Riding Violation Detection system achieved its objectives of identifying critical traffic violations and capturing vehicle details. The use of deep learning (YOLOv8) and OCR proved to be a potent combination for analyzing complex visual data and extracting actionable information in real time. The results validate that such an AI-driven system can serve as a force multiplier for traffic enforcement, operating continuously and impartially.

However, there remains room for enhancement. Implementing improvements such as advanced night-time vision, multi-frame tracking, and expansion to additional violations would further increase the system’s robustness and utility. With further development and larger-scale deployment, this system could become an integral part of intelligent transportation systems. It showcases how technology can be applied to solve real-life problems and paves the way for smarter and safer cities.

Overall, this project has been a valuable learning experience in applying theoretical knowledge to a practical challenge. It highlights the interdisciplinary nature of engineering projects, involving aspects of computer science (AI, programming), electronics (camera hardware), and civil engineering (traffic management). The results also open opportunities for future work.

Future work

1. **Real-Time Edge Inference**

   Port and optimize the trained YOLOv8 model (quantization, pruning) for deployment on low-power edge devices (e.g. Raspberry Pi with Coral TPU) to eliminate round-trip latency and enable truly live roadside enforcement.

2. **Multi-Camera Fusion**

   Synchronize feeds from multiple overlapping cameras to handle occlusions (e.g. a rider hidden in one view) and improve detection continuity by combining detections across viewpoints.

3. **Robust Low-Light & Adverse Weather Handling**

   Integrate image enhancement modules (denoising, dehazing, night-vision transforms) upstream of detection so the system works reliably at night, in rain, or fog.

4. **Expanded Violation Set**

   Add detectors for:

   - Mobile-phone usage by riders (using hand-pose classification)
   - Overspeeding (via radar or video-based speed estimation)
   - Signal-jumping at intersections (by combining object detection with traffic-light state recognition).

5. **User-Configurable Rules Engine**

   Build a GUI for non-technical users to define custom violation rules and penalty schedules (e.g. different fines for repeat offenses, time-of-day adjustments).

RESEARCH

1. **Federated Learning for Privacy-Preserving Detection**

   Explore federated training across multiple agencies' video streams so models improve collectively without sharing raw footage, protecting citizen privacy.

2. **Self-Supervised Adaptation to New Environments**

   Investigate domain-adaptation methods that allow the detector to fine-tune itself on unlabeled footage from a new city or camera without manual annotations.

3. **Explainable AI for Traffic Enforcement**

   Develop techniques to generate human-readable "reason codes" (e.g. saliency maps) showing exactly why a detection was flagged as a violation, to increase trust and contestability.

4. **Multi-Modal Violation Detection**

   Fuse audio (e.g. horn-blast detection), vehicle engine sound classification, and video to catch infractions like illegal honking or engine modifications.

5. **Longitudinal Behavioural Analysis**

   Use the violation logs to model driver behavior over time, identifying chronic offenders or hotspots—an area ripe for research at the intersection of computer vision and transportation policy.

REFERENCES

1. **Arshad, M. & Kumar, P. (2024). "TR-TRVD: Triple Riders** – Traffic Rule Violation Detection using YOLOv8-BBIA for Intelligent Transportation System." Preprint, Nov 2024.
2. **Pradeep Kumar, G. M., Deeksha, N., Chandana, G. A., Krupa, T. S., & Krishnakanth Kumar (2022).** "Detection of Bike Riders without Helmet and Triple Riding using YOLOv3 model." International Journal of Scientific Research in Engineering and Management (IJSREM), 6(6), 1–5.
3. **Solawetz, J. & Piserchia, F. (2024).** "What is YOLOv8? A Complete Guide." Roboflow Blog, Oct 23, 2024.
4. **Suma, S., Anoop, G. L., & Mithun, B. N. (2024).** "Helmet Detection Based on Improved YOLO v8." Journal of Emerging Trends and Novel Research (JETNR), 2(3), 22–28.
5. **Yenugonda, M. & Maddi, H. L. (2023).** "Advanced Computer Vision-Based Surveillance System for Helmet Detection and Triple Rider Identification on Motorcycles using Machine Learning." International Journal of Scientific Research in Science and Technology (IJSRST), 10(5), 163–168.
6. **Zhang, P., Li, J., & Chen, R. (2024).** "Enhanced YOLOv8 for Real-Time Vehicle License Plate Detection and Recognition." IEEE Access.
7. **Li, X. & Wang, Y. (2023).** "Deep Learning-Based Multi-Infraction Detection in Urban Traffic Surveillance." Transportation Research Part C: Emerging Technologies, 142, 103771.
8. **Gupta, S. & Patel, R. (2022).** "Real-Time Motorcycle Helmet Detection Using Deep Learning." Proceedings of the International Conference on Artificial Intelligence and Robotics.
9. **Silva, J. & Oliveira, L. (2021).** "Automatic Number Plate Recognition for Intelligent Transportation Systems." Journal of Transportation Technologies, 11(2), 45–59.
10. **Chen, Y., Huang, Z., & Zhou, L. (2023).** "Federated Learning for Privacy-Preserving Traffic Surveillance." Computers & Security, 120, 102873.