

Traffic Violation Detection and Automated Ticketing Using Computer Vision

Submitted in partial fulfillment of the requirements for the award of
Bachelor of Engineering degree in Computer Science and Engineering

By

AMIYA KUMAR BARIK (40110080)



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SCHOOL OF COMPUTING

SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

Category-1 University by UGC

Accredited with Grade “A++” by NAAC

JEPPIAAR NAGAR, RAJIV GANDHI SALAI, CHENNAI – 600119

APRIL - 2024



SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

Category - I University by UGC

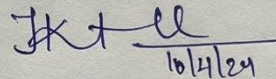
Accredited "A++" by NAAC | Approved by AICTE

www.sathyabama.ac.in

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of **AMIYA KUMAR BARIK (40110080)** who carried out the Project entitled "**TRAFFIC VIOLATION DETECTION AND AUTOMATED TICKETING USING COMPUTER VISION**" under my supervision from November 2023 to April 2024.

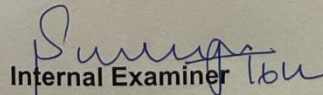

16/4/24

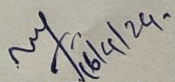
Internal Guide

Ms. Karthika J, M.Tech., (Ph.D.)


HEAD OF THE DEPARTMENT
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Jepplaar Nagar, Rajiv Gandhi Salai,
Dr. L. LAKSHMANAN, M.E., Ph.D., 600 119.

Submitted for Viva voce Examination held on 16-04-2024


Internal Examiner
(Dr. SARANYA S)


16/4/24
External Examiner

DECLARATION

I, **AMIYA KUMAR BARIK (40110080)**, hereby declare that the Project Phase-2 Report entitled **TRAFFIC VIOLATION DETECTION AND AUTOMATED TICKETING USING COMPUTER VISION** done by me under the guidance of **Ms. Karthika J, M.Tech., (Ph.D.)** is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering degree in **Computer Science and Engineering**.

DATE: 10/4/24

Amiya Kumar

PLACE: Chennai

SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to **Board of Management of Sathyabama Institute of Science and Technology** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T. Sasikala M.E., Ph. D, Dean**, School of Computing, **Dr. L. Lakshmanan M.E., Ph.D.**, Head of the Department of Computer Science and Engineering for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Ms. Karthika J, M.Tech., (Ph.D)**, for her valuable guidance, suggestions and constant encouragement paved way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the **Department of Computer Science and Engineering** who were helpful in many ways for the completion of the project.

ABSTRACT

India's excessive population and still rising, poor management of traffic signals and rider mentality make traffic violation monitoring and control, a big challenge. A total of 72 thousand personnel of traffic police are there in all over India, while total number of vehicles currently on roads are approximately 200 million. It is clear that physical traffic police monitoring is insufficient to simultaneously monitor such high traffic volumes and trace offenses. Furthermore, many existing solutions lack the flexibility to recognize, analyze, and track the large variety of vehicle types, license formats, ever-changing traffic patterns, and street layouts. Many cities still have outdated control systems that simply cannot be scaled up efficiently to handle the volume and variety of traffic. This has caused numerous infringers to go undiscovered. The offenders, in turn, bring to more serious accidents on the road that endanger both their own life and the lives of others. As a result, we need a method to monitor the traffic continuously (in real-time) and take action against the violated vehicle owner in a fast and efficient way. If the traffic laws are not enforced or are perceived as not being enforced it is likely they will not be followed properly and will have negative influencing behaviour. Thus, video-based artificial intelligence systems are becoming a vital tool for governments to automate traffic control, detect violations, and even dispense tickets.

TABLE OF CONTENTS

CHAPTER No.	TITLE	PAGE No.
	Abstract	v
	List of Figures	viii
	List of abbreviations	ix
1	INTRODUCTION	1-2
2	LITERATURE SURVEY	3-4
2.1	Inferences from Literature Survey	3
2.2	Open Problems in Existing System	4
3	REQUIREMENTS ANALYSIS	5-11
3.1	Description of Programming Language and Software	5
3.2	Feasibility Studies/Risk Analysis of The Project	10
3.3	Software Requirements Specification Document	10
4	DESCRIPTION OF PROPOSED SYSTEM	12-14
4.1	Selected Methodology	13
4.1.1	Object Detection and Tracking	13
4.1.2	Violation Detection	13
4.1.3	License/Number Plate Recognition	13
4.2	System Architecture of Proposed System	14
5	IMPLEMENTATION DETAILS	15-28
5.1	Module 1: Object Detection and Tracking	15
5.2	Module 2: Violation Detection	18
5.3	Module 3: License/Number Plate Recognition	20
6	RESULTS AND DISCUSSION	29-31
6.1	Results and Discussion	29

7	CONCLUSIONS	32
7.1	Conclusions and Future works	32
	REFERENCES	33-34
	APPENDIX	35-52
	A1 Screenshots	35
	A2 Source Code	36
	A3 Research Paper	45
	A4 Presentation Certificate	52

LIST OF FIGURES

FIGURE NO.	FIGURE NAME	PAGE NO.
4.1	Flow chart diagram	12
4.2	System Architecture	14
6.1	ROC curve	29
6.2	Results of Violation and Number Plate Detection On Video Feed	30
6.3	Result of Number Plate Detection on Test Dataset	31

LIST OF ABBREVIATIONS

ABBREVIATION	EXPANSION
AI	Artificial Intelligence
CCTV	Closed Circuit Television
OCR	Optical Character Recognition
ANPR	Automatic Number Plate Recognition
CNN	Convolutional Neural Network
YOLO	You Only Look Once
RFID	Radio Frequency Identification
OPENCV	Open-Source Computer Vision Library
DEEPSORT	Deep Simple Online and Realtime

CHAPTER 1

INTRODUCTION

In India most of the traffic junctions are still manually monitored if not completely not monitored, which lead to large traffic congestion and human errors. More than onethird (37%) of those killed in road accidents in 2019 were two-wheeler riders, as per the Ministry of Road Transport and Highways' report. Further, not wearing helmets resulted in 44,666 deaths (30,148 drivers and 14,518 million passengers) that is 29.82% of total road accident fatalities during 2019. These are just small amount of possible violations that cause to an accident, others are over speeding, crossing the red light, using phone while driving etc. Thus, statistics have shown that most of the traffic related incident have occurred due to violation of traffic rules and regulations. Manual monitoring requires human labour and is very tedious task. Hence by using Computer Vision technology, we can overcome this problem. By computer vision technology not only we can monitor traffic, we can also detect any possible traffic violation and take action by issuing ticket/challan. Automatic ticketing system offer a solution to the problem of manual traffic ticketing. By eliminating errors and bias, these systems can improve accuracy and efficiency, leading to safer and more efficient roads.

Accidents on the road are a major problem in today's world. Every year, several countries invest millions of dollars to reduce traffic accidents. To keep road users from being badly wounded or killed, many road safety systems and measures are in place. Nevertheless, countries face a significant number of vehicle accidents every day. The majority of these incidents are caused by persons who are unwilling to respect traffic laws and a lack of supervision. According to research, improvements to road infrastructure, particularly design standards that include the safety of all road users, are crucial to keeping roads safe. 112 countries have national design standards for the management of speed. But infrastructure alone cannot make the roads safer without monitoring and adherence to traffic rules. The problem is particularly acute in countries like India where people tend to follow rules only when traffic police are watching. If there was a system that could detect traffic

rules violations using CCTV cameras, it would definitely contribute to road safety. The idea presented in the project can be used to monitor traffic rules violations using road side camera. People often tend to ignore rules, and regulations whenever they feel like there won't be any consequences. Many road accidents are caused in India every year by sheer ignorance. It not only compromises the safety of the driver but also the safety of other people on the road. Especially in the crowded areas, traffic jams during rush hour have become a norm. Undoubtedly, there is poor road infrastructure in these areas, however it is not the sole cause of the problem. Lack of road discipline has a significant role in it, a simple lane violation has caused huge traffic jams in the past. Police are finding traffic management increasingly complex as the number of vehicles increases every year. Government has prioritized installation of CCTV cameras for security, and surveillance in recent years. Implementation of computer vision based automatic traffic violation systems utilizing the footage from already installed cameras will increase their utility as well as help in betterment of overall road safety.

The scope of this project encompasses the design, development, and evaluation of a comprehensive traffic violation detection system. Key components of the system will include image acquisition devices such as cameras or sensors, image processing algorithms for real-time analysis, machine learning models for violation detection, and backend infrastructure for ticket generation and enforcement. The system will be designed to operate in diverse traffic environments, including urban streets, highways, and intersections, while accounting for varying lighting conditions, weather effects, and vehicle types. Additionally, the study will assess the legal and ethical implications of automated ticketing systems, as well as their potential impact on traffic management, public safety, and regulatory compliance. Through empirical testing and validation, the project aims to demonstrate the efficacy and feasibility of employing computer vision technology for enhancing traffic enforcement efforts and fostering safer roadways.

CHAPTER 2

LITERATURE SURVEY

2.1 INFERENCES FROM LITERATURE SURVEY

1. **Xialing Wang, Li-Min Meng, Biaobiao Zhang (2013). A video-based traffic violation detection system. [1]**

This journal proposes an improved background-updating algorithm by using wavelet transform on dynamic background, and then track moving vehicles by feature-based tracking method. A complete traffic violation detection system is realized in C++ with OpenCV.

2. **Krishna, Madhav Poddar, Gridhar M K, Amit Suresh Prabhu, Umadevi V (2016). Automated traffic monitoring system using computer vision. [2]**

They used the concept of multiple reference lines for identifying vehicles. Using the total duration of frames the vehicle took to cross the reference lines we determine the speed and hence the possibility of a traffic violation.

3. **Katanyoo Klubsuwan, Wittaya Koodtalang (2013). Traffic violation detection using multiple trajectories evaluation of vehicle. [3]**

This system they proposed can detect red-light and lane-change violation using trajectories behavior of multiple vehicles using mean square displacement (MSD) to detect both violations.

4. **Aaron Christian, Rhen Anjerome Bedruz (2015). Machine vision for traffic violation detection system through genetic algorithm. [4]**

This paper presents a machine vision algorithm to detect traffic violations specifically swerving and blocking the pedestrian lane. The proposed solution consists of background difference method and focuses on the genetic algorithm of the system to detect these violations.

5. **S. F. Wong, H. C. Mak, C. H. ku, W. I. Ho (2017). Developing advance traffic violation detection system with RFID technology for smart city. [5]**

This study aims to develop a system to instantly detect traffic conditions by using computer vision with RFID technology for obtaining vehicles data. The system detects the presence of vehicle and also calculates the detected objects. The method of background subtraction on motion objects detection is proposed.

2.2 OPEN PROBLEMS IN EXISTING SYSTEM

- Most of the algorithm used in projects is background subtraction method, which is used to detect objects, but this algorithm cannot deal with sudden change in lighting and illumination.
- Many projects are done on specific type of traffic violation rather not whole/complete, such as only for over-speeding, crossing red-light, wearing helmet etc.
- After detection of violation there are very less number of projects that actually try to issue e-challans or tickets by recognizing the license plate.
- Most of the projects concentrates on the detection from pre-processed images but not live feed which requires real-time processing and preparation of frames and real-time predictions.

CHAPTER 3

REQUIREMENTS ANALYSIS

3.1 DESCRIPTION OF PROGRAMMING LANGUAGE AND SOFTWARE

- **PYTHON**

Among programmers, Python is a favorite because to its user-friendliness, rich feature set, and versatile applicability. Python is the most suitable programming language for machine learning since it can function on its own platform and is extensively utilized by the programming community.

Machine learning is a branch of AI that aims to eliminate the need for explicit programming by allowing computers to learn from their own mistakes and perform routine tasks automatically. However, artificial intelligence encompasses a broader definition of "machine learning," which is the method through which computers are trained to recognize visual and auditory cues, understand spoken language, translate between languages, and ultimately make significant decisions on their own. The desire for intelligent solutions to real-world problems has necessitated the need to develop AI further in order to automate tasks that are arduous to program without AI.

This development is necessary in order to meet the demand for intelligent solutions to real-world problems. Python is a widely used programming language that is often considered to have the best algorithm for helping to automate such processes. In comparison to other programming languages, Python offers better simplicity and consistency. In addition, the existence of an active Python community makes it simple for programmers to talk about ongoing projects and offer suggestions on how to improve the functionality of their programs.

- **VS Code**

Visual Studio Code, commonly known as VS Code, stands as a versatile and widely embraced source-code editor developed by Microsoft. Its appeal lies in its crossplatform compatibility, supporting Windows, macOS, and Linux, and its lightweight nature, ensuring swift startup and efficient resource usage. What sets

VS Code apart is its extensibility; a robust marketplace of community-contributed extensions allows developers to tailor the editor to their specific needs. Moreover, it offers broad language support, and the incorporation of extensions expands this support to encompass even more programming languages. The inclusion of an integrated terminal streamlines command-line operations, script execution, and server management. Additionally, VS Code provides IntelliSense, a feature that enhances coding efficiency by offering context-based code completions and suggestions. It also includes built-in Git support, facilitating version control, collaboration, and code management. As a result, Visual Studio Code has become the code editor of choice for a diverse array of developers, ranging from web and software developers to data scientists and artificial intelligence engineers, due to its adaptability and userfriendly interface.

- **YOLOv8**

YOLOv8 is a state-of-the-art object detection model developed by Ultralytics. It is the latest version of the YOLO family of models, which are known for their speed and accuracy. It is built on the PyTorch framework and uses a variety of deep learning techniques to achieve its performance.

YOLOv8 is an anchor-free model, which means that it does not rely on anchor boxes to predict the location of objects in an image. Instead, YOLOv8 directly predicts the center of an object and its size. This makes it faster and more accurate than previous YOLO models.

It also introduces a number of other improvements over previous YOLO models, including:

- A new backbone network that is more efficient and accurate.
- A new neck network that is better at extracting features from images.
- A new head network that is better at predicting the location and class of objects.

YOLOv8 is a powerful and versatile object detection model that can be used for a variety of tasks, including:

- Real-time object detection in video streams.

- Object detection in images and videos for offline processing.
- Instance segmentation, which is the task of segmenting individual objects in an image or video.

YOLOv8 is still under development, but it has the potential to be a valuable tool for a variety of applications, including:

- Self-driving cars.
- Security and surveillance systems.
- Medical imaging.
- Robotics.

Here are some of the key features of YOLOv8:

- **Speed:** it is one of the fastest object detection models available. It can detect objects in real time on a variety of hardware platforms.
- **Accuracy:** it is also one of the most accurate object detection models available. It has achieved state-of-the-art results on a variety of benchmarks.
- **Versatility:** it can be used for a variety of tasks, including real-time object detection, object detection in images and videos for offline processing, and instance segmentation.
- **Open source:** it is an open-source model, which means that it is free to use and modify.

Overall, YOLOv8 is a powerful and versatile object detection model that has the potential to be used in a variety of applications.

- **Cross Stage Partial Network**

YOLO is a deep network; it uses residual and dense blocks in order to enable the flow of information to the deepest layers and to overcome the vanishing gradient problem. However, one of the perks of using dense and residual blocks is the problem of redundant gradients. CSPNet helps tackling this problem by truncating the gradient flow. According to the authors of [21]: CSP network preserves the

advantage of DenseNet's feature reuse characteristics 9 and helps reduce the excessive amount of redundant gradient information by truncating the gradient flow.

- **OpenCV**

OpenCV is an open-source computer vision and machine learning library. It was originally developed by Intel in 1999 and has since become one of the most popular computer vision libraries in the world. It is written in C++ and has bindings for Python, Java, and other languages. It is available for free under the BSD license.

OpenCV provides a wide range of functions for computer vision tasks, including:

- Image processing: it provides functions for a variety of image processing tasks, such as filtering, resizing, and converting between different color spaces.
- Feature detection and extraction: OpenCV provides functions for detecting and extracting features from images, such as corners, edges, and keypoints.
- Object detection and tracking: it provides functions for detecting and tracking objects in images and videos.
- Machine learning: it provides functions for a variety of machine learning tasks, such as classification, regression, and clustering.

OpenCV is used in a wide range of applications, including:

- Robotics: OpenCV is used in robots to help them perceive their environment and navigate.
- Self-driving cars: OpenCV is used in self-driving cars to detect and track other vehicles, pedestrians, and road signs.
- Security and surveillance systems: OpenCV is used in security and surveillance systems to detect and track intruders and other suspicious activity.
- Medical imaging: OpenCV is used in medical imaging to detect and analyze tumors, lesions, and other abnormalities.

- Augmented reality and virtual reality: OpenCV is used in augmented reality and virtual reality to track the position and orientation of the user's head and hands.

OpenCV is a powerful and versatile computer vision library that can be used for a wide range of applications. It is a valuable tool for researchers, developers, and hobbyists alike.

- **TesseractOCR**

TesseractOCR is an open-source optical character recognition (OCR) engine developed by Google. Originally created by Hewlett-Packard in the 1980s, it was later taken over by Google in 2006 and released as an open-source project in 2009. It is designed to recognize text within images, making it a powerful tool for extracting text from scanned documents, images, and even screenshots. It supports multiple languages and can handle a variety of font types and sizes. It operates on a command-line interface and can be integrated into various applications and platforms through its APIs.

The engine utilizes advanced machine learning techniques, including neural networks, to accurately identify and extract text from images. It continues to be actively developed and improved by contributors from around the world, making it one of the most popular and widely used OCR engines available.

- API request: Tesseract OCR can only be accessed via API integration. Once that connection between your solution and Tesseract is established, you can send API requests from your solution to the Tesseract OCR engine.
- Input Image: With an API request, you can send in your input image for text extraction.
- Image pre-processing: Before data extraction, the image pre-processing features of the Tesseract OCR engine kick in. This step exists to ensure that the image quality is as high as possible to achieve accurate data extraction results. Often OpenCV is coupled with Tesseract to increase the image quality before data extraction.

- **Data Extraction:** Together with trained data sets and Leptonica or OpenCV, the Tesseract OCR engine processes the input image and extracts the data.
- **Text conversion:** As the data (text) has been extracted from the input, it can now be converted into a desired format that Tesseract supports, including PDF, plain text, HTML, TSV, and XML.
- **API response back:** Once the output is ready, your solution will get an API response back with the finalized output.

•

3.2 FEASIBILITY STUDIES/RISK ANALYSIS OF THE PROJECT

- **Accuracy:** Computer vision algorithms are not perfect. They can be fooled by environmental conditions, such as poor lighting or weather, or by objects that resemble vehicles or traffic signs. This can lead to false positives (identifying a traffic violation where none exists) or false negatives (missing a traffic violation).
- **Data privacy:** data collected from the traffic surveillance camera need to send the data through network for processing with can lead to data vulnerability.
- **Security:** traffic violation detection systems could be hacked, which could allow attackers to disable the system or issue false tickets.
- **Regular updates and training:** the system needs to be regularly updated and trained in order to efficiently detect possible traffic violations.
- **Affordability:** other than installing traffic surveillance cameras, any other hardware component (such as sensors) will make the system more expensive.

3.3 SOFTWARE REQUIREMENTS SPECIFICATION DOCUMENT

- OS: windows 10
- IDE: visual studio
- Coding language: python

Standards

Compliance with existing traffic regulations and laws: The system must adhere to established traffic laws and regulations to ensure legal validity and enforceability of automated tickets. Data privacy and security standards: Adherence to data protection laws and security standards to safeguard sensitive information collected during violation detection and ticketing processes. Performance standards: Meeting predefined performance metrics such as accuracy, speed, and reliability to ensure effective enforcement and public trust.

Constraints

Environmental factors: Variability in lighting conditions, weather effects, and environmental clutter may impact the performance of computer vision algorithms, requiring robust adaptation mechanisms. Computational resources: Limited computational resources, especially in onboard systems for mobile applications, may impose constraints on the complexity and speed of image processing algorithms. Hardware limitations: Dependence on hardware components such as cameras, sensors, and processors, which may have inherent limitations in terms of resolution, field of view, and processing power.

Trade-offs

Accuracy vs. Speed: There may be a trade-off between the accuracy of violation detection and the speed of processing, particularly in real-time applications where rapid decision-making is essential. Complexity vs. Efficiency: Balancing the complexity of computer vision algorithms with computational efficiency to achieve optimal performance within resource constraints. False Positives vs. False

Negatives: Striking a balance between minimizing false positives (incorrectly identifying violations) and false negatives (missing genuine violations) to optimize the overall effectiveness and fairness of enforcement actions.

CHAPTER 4 DESCRIPTION OF PROPOSED SYSTEM

The proposed methodology is to make a system, which can detect traffic violations, with minimum human interaction, cost, with very high accuracy and ticketing system. In order to get high accuracy with speed we are using YOLOv8, which out-performs all its previous versions.

It is nothing but an algorithm that utilizes a single convolutional network for object detection, which will be used to detect vehicle and number plate too. Then by using deepSORT we can track those objects (vehicles) in order to remove duplicate counts. We strategically use the same YOLOv8 for violation detection and to retrieve license plate to prevent additional computation of pure image processing or feature extraction, which save lot of computational resources. Finally, after getting the number of the violator vehicle's license plate, we can store it and further process it for verification and issuing e-challan/tickets.

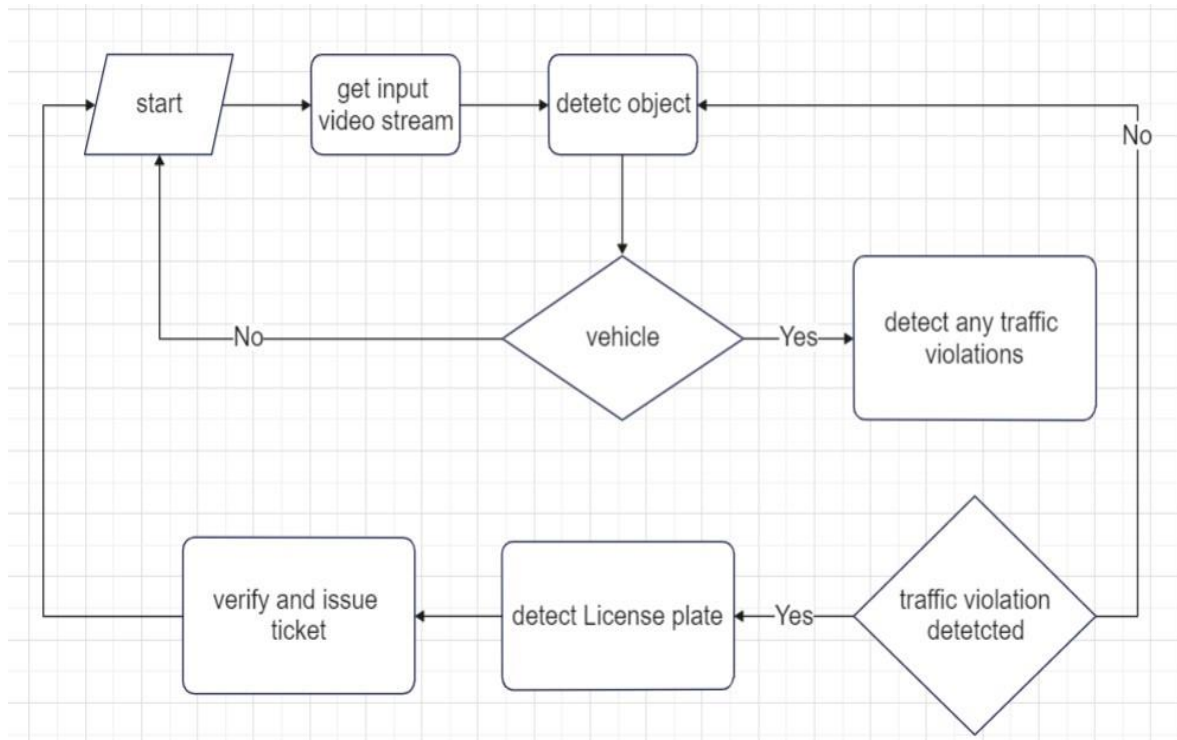


Fig 4.1: Flow chart diagram

4.1 SELECTED METHODOLOGY OR PROCESS MODEL

4.1.1 OBJECT DETECTION AND TRACKING:

Frames are taken as an input (from the video) for object detection which is performed by YOLOv8 algorithm. In our case the objects are vehicles. (pre-trained). The object is then tracked using deepSORT algorithm to prevent repetitive counting. It also ensures to capture the vehicles in multiple frames.

YOLOv8 divides the input image into a grid and predicts bounding boxes and class probabilities for each grid cell in a single pass. This approach is highly efficient, allowing for fast detection of multiple objects in real-time. Unlike traditional two-stage detection methods, it directly predicts bounding boxes and class probabilities without requiring a separate region proposal step. This results in faster inference times and improved efficiency. It achieves high accuracy in object detection tasks by leveraging deep neural networks and training on large-scale datasets. It can effectively detect objects of various sizes and classes with remarkable precision.

4.1.2 VIOLATION DETECTION:

The dataset obtained from various online sources, (especially Kaggle) is used to train our violation detection module. It consists of roughly 6000 labelled images for training and 1200 images for testing. Dataset consist of traffic violation such as not wearing helmet, crossing red light, over speeding etc. the algorithm detects objects in images through a single neural network pass, dividing the image into a grid and predicting bounding boxes, class probabilities, and object scores for each grid cell. Non-maximum suppression and confidence thresholding are then applied to filter and refine the detections, resulting in a list of bounding boxes with associated class labels and confidence scores for the detected objects. This approach enables realtime object detection with high accuracy and efficiency.

4.1.3 LICENSE/NUMBER PLATE RECOGNITION:

After detecting a violated vehicle, its license plate is recognized by ANPR (automatic number plate recognition). It is done by tesseractOCR which extracts the alphanumeric text in the license plate for further actions. Human interaction is still required to validate the traffic violation because the algorithm is not 100% accurate.

4.2 ARCHITECTURE / OVERALL DESIGN OF PROPOSED SYSTEM

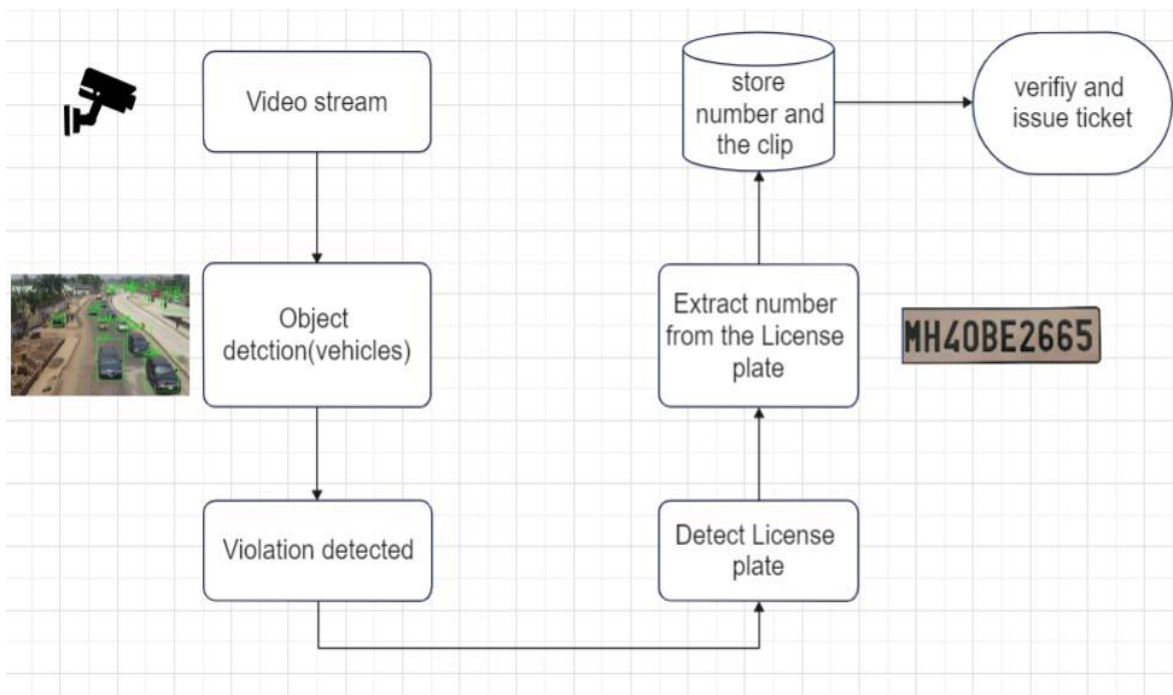


Fig 4.2: System Architecture diagram

CHAPTER 5 IMPLEMENTATION DETAILS

5.1 MODULE 1: OBJECT DETECTION AND TRACKING

- **Object Detection**

Before the image is fed to the YOLOv8 CNN model in our project the data is preprocessed. The general practices for preprocessing were practiced and for robustness data augmentation was implemented too, but before we dive into data preprocessing and its augmentation we need to talk about the methods and hurdles we faced during data acquisition.

- **Data Acquisition**

Data had to be collected for training dataset and to collect data for creating a dataset for YOLOv8 to detect vehicles and license plates, we used a camera to capture video and we used a tripod to ensure stability and accuracy in capturing the images. We positioned the camera at various angles and heights to capture different views of the vehicles. We ensured that each image contained at least one vehicle, and the

license plate was visible, which was especially challenging due to the high volume of traffic on the roads. When using a camera to collect data, it is important to be aware of the artifacts that might be introduced to the images and data. For instance, the depth of field blur might cause certain parts of the image to be out of focus which might make it hard to see the characters of the license plate. The motion blur might result from the movement of the vehicle or camera and could also lead to image distortions. These artifacts can significantly affect the dataset's quality and accuracy, making it challenging to detect the vehicles and license plates. In order to address the impact of these unwanted visual effects on the quality of our dataset, we took measures to appropriately tweak camera settings. We made sure that depth-of-field was appropriate by adjusting aperture size and avoided motion blur through control over shutter speed. Furthermore, lighting conditions were closely monitored in order to ensure optimal illumination so as not obscure any license plate details captured in images taken for this project's purposes. By carefully following such procedures during data collection processes overall resulting images obtained had minimal defects thus allowing YOLOv8 algorithms capabilities accurate detection of objects like vehicles and their associated label information with reliability being paramount throughout each step. To stay consistent, we took the video in the same resolution and same fps throughout the process. The video was then parsed by a python script we wrote that would take the video and take images from it after every 6 frames and store the frames that will be used to create a labeled dataset as well as for test dataset.

The images collected thus far need to be labeled properly in order to turn it into useful data to create a dataset for our training model. In order to label the images, we used an open-source tool called "makesense.ai" which is free to use under APLv3 license. It does not store any of our images as it runs locally and supports multiple output formats like YOLO, VOC XML, VGG JSON, CSV. More importantly it allows us to use a custom model that we trained on the 1st batch of 1000 images to label the next batch of images which makes our manually labeling job a bit easier. We used a lightweight YOLOv8 model to help us label the images we acquired.

- **Data Preprocessing**

Regardless of all the measures we took while acquiring the data some unwanted artifacts are introduced in the image which hinders the accuracy of the model. To minimize its effects the images 1st goes through histogram equalization which is a commonly used technique for image preprocessing to adjust the image's pixel intensities, enhance the contrast of the image and increase its visual features. The process of histogram equalization involves computing a histogram of the image's intensity values and then redistributing these values to create a more uniform distribution. The result is an image with enhanced contrast, where the dark and light areas of the image are more distinguishable. This technique is especially useful for images that have a limited range of pixel intensities, resulting in low contrast and making it challenging for YOLOv8 to detect objects accurately

- **Data Augmentation**

To expand the amount of data that is at disposal when training a model in machine learning, it's common to apply a technique known as data augmentation. The basic idea behind this strategy involves deriving fresh sets of training samples by subjecting original datasets to diverse transformations. Such manipulations can include completely random rotations and flips or modifications involving crops with different brightness levels, contrast variations or even changes in saturation levels too. This method offers an opportunity for models' exposure towards more complex dynamics present amongst input information thus enhancing their capacity for generalization during tests with unseen data inputs. The implementation of data augmentation techniques can prevent overfitting, whereby the model becomes prone to retaining information in the training dataset rather than acquiring generalizable patterns. Generating supplementary samples for training purposes not only decreases the likelihood of overfitting but also bolsters performance efficiency. Moreover, by using data augmentation practices that balance class distribution within a given dataset leads to adequate exposure of examples from all classes before modeling commences. We followed generally followed practices for augmentation of data. The data augmentation techniques we used for our dataset are:

1. **Rotation:** We introduced rotation to the original images anywhere between – 15 degree to +15 degree.

2. Shear: We introduced shearing to the original images anywhere between ± 15 degree horizontal, ± 15 degree vertical.
3. Grayscale: 20% of the images we used for the dataset we applied grayscale filters.
4. Hue: The Hue value of the image was adjusted anywhere between ± 25 degree.
5. Saturation: The saturation value of the images was changed to be anywhere between $\pm 25\%$
6. Brightness: The brightness value of the images was changed to be anywhere between $\pm 25\%$.
7. Exposure: The exposure value of the images was changed to be anywhere between $\pm 25\%$.
8. Blur: Up to 0.75px blur was introduced to images
9. Noise: Up to 3% of pixel noise was introduced to images

After the dataset is ready, we split the dataset into training dataset, validation dataset and finally test dataset. We used YOLOv8 model from [22] to train on our custom dataset.

5.2 MODULE 2: VIOLATION DETECTION

• Confusion matrix

A confusion matrix is a table that is often used to evaluate the performance of a classification model. It provides a detailed breakdown of how many instances were correctly or incorrectly classified by the model, allowing us to measure the model's overall accuracy. A confusion matrix is typically composed of four different metrics:

1. True Positives (TP): This refers to the number of positive instances that were correctly classified by the model.
2. False Positives (FP): This refers to the number of negative instances that were incorrectly classified as positive by the model.
3. False Negatives (FN): This refers to the number of positive instances that were incorrectly classified as negative by the model.
4. True Negatives (TN): This refers to the number of negative instances that were correctly classified by the model.

By combining these metrics, we can create a table that summarizes the performance of the classification model. The table is organized into a grid, with the actual class labels on one axis and the predicted class labels on the other.

Using this table, we can calculate a number of different metrics that are useful for evaluating the performance of a classification model.

1. Accuracy: This refers to the overall percentage of instances that were correctly classified by the model. It is calculated as $(TP + TN) / (TP + TN + FP + FN)$.
2. Precision: This refers to the percentage of instances that were classified as positive by the model that were actually positive. It is calculated as $TP / (TP + FP)$.
3. Recall: This refers to the percentage of actual positive instances that were correctly identified by the model. It is calculated as $TP / (TP + FN)$.
4. F1-Score: This is a weighted average of precision and recall, with a higher score indicating better performance. It is calculated as $2 * ((precision * recall) / (precision + recall))$.

Overall, the confusion matrix is a powerful tool for evaluating the performance of a classification model. By breaking down the model's predictions into a detailed table, we can gain valuable insights into its strengths and weaknesses, and identify areas for improvement.

- **mAP**

mAP(mean Average Precision), is an evaluation metric in object detection and image classification tasks. It measures the accuracy of an algorithm in identifying objects within an image and assigning a level of confidence to the identified objects. The value of mAP is calculated by taking the average of the precision values at different recall levels. Here, the recall level represents the percentage of objects detected correctly out of all the objects that are actually present in the image. mAP at 0.5 is a specific threshold used in object detection tasks. It refers to the minimum level of intersection over union (IoU) required between the predicted bounding box and the ground truth bounding box for the detection to be considered correct. In other words, if the predicted bounding box overlaps with the ground truth bounding box by at least 50%, then the detection is considered correct and contributes to the mAP at 0.5 score. This threshold of 0.5 is commonly used in object detection tasks as it strikes a balance between precision and recall. Our model's performance upon

referring to the mAP curve at 0.5 threshold shows that the average precision is more than 0.85 at approximately 460 epochs.

- **DeepSORT Tracking**

Tracking is the process of predicting an object's position over the course of a video with the use of both spatial and temporal features. It is a two-step procedure:

- A detection module for target localization: It is responsible for locating the object in the frame using an object detector, in our case YOLOv8.
- A motion predictor: It is responsible for making motion predictions based on the object's past motion.

DeepSORT is a computer vision tracking algorithm used to track objects while giving each one a unique ID. It is an addition to the SORT algorithm. It incorporates deep learning into the SORT algorithm by adding an appearance descriptor to reduce identity switches. This makes tracking more efficient.

The SORT and Deep-SORT are the two well-known Kalman Filter - based tracking methods. The Deep-SORT is an extended version of the SORT algorithm as it uses DL based association metrics on the Data association module. Both the SORT and Deep-SORT have 3 main modules: KF Estimation, Data Association and Track Management.

- **Violation Detection**

A line is draw on the video frame, if any vehicle crosses that line when the light is either yellow, or red, it is annotated with red bounding box. The python pseudo code of line crossing implementation is given below:

1. tracker list = { }
2. For each frame in the video:
3. Draw (a, b) to (c, d)
4. detections = All 'bounding-boxes' and associated 'tracker ids' from DeepSORT
5. For bounding-box, tracker id in detections:
 - a. x1, y1, x2, y2 = bounding-box
 - b. anchors = [(x1, y1), (x1, y2), (x2, y1), (x2, y2)]
 - c. triggers = []

- d. for (x, y) in anchors: $v1 = (c - a), (d - b)$ $v2 = (x - a), (y - b)$ cross product = $v1 \times v2 < 0$ state = True if cross product else False Add 'state' to 'triggers'
- e. If all 'triggers' are not same: continue
- f. tracker state = triggers [0]
- g. If 'tracker id' is not in 'tracker list': Add 'tracker id' and 'tracker state' to 'tracker list': continue
- h. If 'tracker id' has same 'tracker state' in 'tracker list': continue
- i. Add 'tracker id' and 'tracker state' to 'tracker list'

5.3 MODULE 3: LICENSE/NUMBER PLATE RECOGNITION

- **Plate Localization**

Localization refers to identifying the location of an object in the image. In any ANPR system, plate localization is the first stage. There are many approaches to this essential task, before getting ready for plate recognition. Basic image processing techniques can be used in ANPR when done under regulated lighting settings with predictable license plate types. Some authors have summarized various methods of license plate localization; histogram, morphological processing, texture, edge detection, and transformation are the bases for the number plate detection techniques.

Many used morphological operation for license plate localization. The algorithm used morphological operations on the pre-processed, edge images of the vehicles.

Characteristic features such as license plate width and height, character height and spacing are considered for defining structural elements for morphological operations. Traditional approach of plate localization using morphological operations, and four-point contour detection is difficult to generalize given the various types of license plate being used in India right now. Law clearly defines the size, color, character size and spaces between lines and character, but people have taken liberty to use number plate that fits their vehicle. Introduction of emboss number plate has certainly brought some standardization, but its adoption rate has been very slow. Given the pretext, the task of plate localization was done by training YOLOv8 on a custom dataset, both images and video, taken at various location and from multiple angles. YOLOv8 model trained on our custom dataset

identifies motorcycle, car, bus, truck, and license plates. The object detection step draws a bounding box around the detected license plate above a set confidence threshold. The plate candidate detected by the object detector is called the region of interest (ROI).

- **Processing ROI**

Pre-processing the acquired image is the first stage in any image processing system. Pre-processing involves some actions on the image to improve the area of interest, and as a result, these procedures are wholly dependent on context. In our situation, the area of interest is the vehicle's license plate, so we put the following processes into place to transform the image into one that can be processed further and improve the effectiveness of the segmentation algorithm. Pre-processing of ROI includes, region grow, HSV Color Space Conversion, color masking, and perspective transform.

Region Grow: The object detector may not always perfectly bound the license plate in every image or frame; therefore, the detected bounding box is increased by ten pixels if the image size is not exceeded.

HSV Color Space Conversion: Colors are described in terms of Hue, Saturation, and Value according to the Hue, Saturation, and Value (HSV) color space model. When color description is important, the HSV color model is frequently chosen over the RGB model. Similar to how the human eye perceives color, the HSV model defines color. Whereas RGB depicts color as a mixture of primary colors, HSV characterizes color using more relatable comparisons like color, vibrancy, and brightness. Color definitions are contained within a hexcone and the coordinate system is cylindrical. The range of the hue value H is 0° to 360° . The saturation S ranges from 0 to 1 and indicates the degree of strength or purity.

The brightness, which similarly varies from 0 to 1, is represented by value V.

Color Masking: The image in HSV color space is now ready to be masked by using appropriate color mask. license plate can come in multiple colors as discussed earlier depending on the ownership of the vehicle. Our interest of vehicle is private vehicles, which have red color plate with white characters. Thus, we masked the red color from the image. In HSV model, the Hue value from around 0° - 10° and 350° - 360° can be approximated as the red color. Using

these range, we masked the red color regions from the images or frames [15].

4. Perspective Transform: The color masked image is sent for external contour detection, the contours are arranged by area, and the largest contour by area is approximated by a minimum rectangle. This is where the characters in the plate are located. Due to variation in camera position, and angle, the plate could be oriented in any direction. If such plate is sent for segmentation, the algorithm won't perform well. The four coordinate points of the minimum rectangle found earlier is arranged in order: top-left, top-right, bottom-right, and bottom-left. A blank image whose width is the maximum distance between bottom-right and bottom-left x-coordinate or the top-right and top-left x-coordinate, and height is the maximum distance between the top-right and bottom right y-coordinate or top-left and bottom-left y-coordinate is created.

Now we have the size of new image, a set of destinations points are obtained in the same order: top-left, top-right, bottom-right, and bottom-left. To obtain a "birds eye view" or top-down view of the plate, a perspective transform matrix is calculated using OpenCV's `getPerspectiveTransform ()` method, and this matrix is used to compute the perspective transform.

- **Character Segmentation**

The vehicles on the road are detected, and those vehicles violating the traffic rules are flagged, but it makes no sense if we cannot identify the offender. The license plate number or simply called vehicle number in common language, is unique to ever vehicle on the road. It conveys some additional information about the vehicle, such as vehicle type, and ownership. The detected license plate must be processed prior feeding it to a CNN trained on license plate characters for recognition. Character segmentation means extracting the characters from a given license plate.

Contrast stretching transformation is used to improve character areas, and then Laplacian Transformation is used to detect edges. With the aid of the regiongrowing algorithm, the locations of the potential regions are discovered. Using predetermined parameters, the process of "region growing" groups pixels

or smaller regions into bigger ones. The method was tested many images with a success rate of 97.2%.

- **Pre-processing**

The object detector gives a region of interest, we process the region of interest and get the region of the plate where characters are present. The plate's perspective must be top-down view to ensure proper segmentation of characters. In the pre-processing step, we want to reveal the characters clearly from the background of the plate. Before extracting characters, raw plate image is fine-grained. It is the stage when unnecessary marks, noise, and blobs are removed. Dirt and dust on the plate, old discoloured plates, and distorted characters cause problem in segmentation. The primary pre-processing steps are: histogram equalization (Clip Limited Adaptive Histogram Equalization), grayscale, median filter, morphological operation, and binarization.

- **RGB to Grayscale Conversion**

By adding the R, G, and B weighted sums, a 24-bit RGB image is transformed into an 8-bit grayscale image. The grayscale signal used to display images on monochrome televisions is chosen using the same weights as the NTSC color space. The grayscale picture for the RGB image $f(x, y)$ is provided by the formula:

$$g(x, y) = 0.2989 * f_R + 0.5870 * f_G + 0.1140 * f_B$$

Where, f_R , f_G , and f_B are the red, green, and blue components of the RGB image $f(x, y)$, respectively.

- **Histogram Equalization**

A histogram is a representation of frequency distribution of pixel intensity. This method usually increases the global contrast of many images, especially when the image is represented by a narrow range of intensity values. Global Histogram Equalization (GHE) is very simple and fast, but its contrast enhancement power is low.

- **Median Filter**

One of the crucial steps in image preparation for segmentation is noise removal. Filtering is used to remove noisy image pixels. The noise removal method utilized in this case is non-linear median filtering. An efficient way to reduce noise is to use a median filter, which can do so without obliterating sharp edges [30]. A pixel is replaced with a median filter with the neighbourhood median value.

The median filtered image for the digital image $f(x, y)$ is derived as, $g(x, y) = \text{median}\{f(i, j) \mid (i, j) \in w\}$

Where, w is the neighbourhood in the image centred on position (x, y) .

- **Binarization**

The central task of character segmentation is distinguishing the characters (foreground) from the plate (background). The segmented image $g(x, y)$ for the grayscale image $f(x, y)$ is produced using the image binarization procedure described below.

$$g(x, y) = 1 \text{ if } f(x, y) \geq T \quad 0 \text{ if } f(x, y) < T$$

where, T is the threshold value and it can be obtained using the Otsu's threshold selection technique for grayscale image segmentation [31]. The automatic threshold selection method developed by Otsu for grey level picture binarization is nonparametric and unsupervised. An optimal threshold is selected by the discriminant criteria i.e., by maximizing the interclass variance between white and black pixels.

- **Morphological Operation**

Morphological transformations are some simple operations based on the image shape. It often takes place on binary images. It requires two inputs: our original image as the first, and the structural element or kernel as the second, which determines the type of operation. Erosion and dilation are two fundamental morphological operations. Then, its alternative forms, such as Opening, Closing, Gradient, etc., also come into play.

Erosion: The fundamental concept of erosion is similar to soil erosion, except it only removes the boundaries of foreground objects (always try to keep foreground in white). The kernel traverses the picture (as in 2D convolution). A

pixel in the original image—whether it is 1 or 0—will only be treated as 1 if every pixel under the kernel is 1, otherwise it is eroded (made to zero).

Thus, depending on the size of the kernel, all pixels close to the boundary are eliminated. So, the foreground object's thickness or size, or just the white region itself, reduces in the image. It can be used to eliminate faint white noises, separate two related items, like the characters in our case, and more.

Dilation: It is directly opposed to erosion. A pixel element in this case is a "1" if at least one pixel under the kernel is "1". Therefore, either the size of the foreground object or the white area in the image grows. Typically, erosion is followed by dilation in situations like noise abatement. Because erosion reduces the size of our item while simultaneously removing white noise. So, we dilate it.

They won't return because the noise is gone, but our object area grows. It can also be used to repair disconnected pieces of a character.

- **Cleaning**

The pre-processed plate is not yet ready for segmentation, there might be plate boundary, dirt, dust, and noise left in the plate. These unwanted objects will drastically hamper the performance of segmentation algorithm. We clean them by finding contours all contours and filtering the unwanted objects based on area, aspect ratio, width, height, solidity and extent. A mask is created for all those objects which pass these tests, and 'bitwise-and' of original image is taken with the mask. The threshold for above mentioned tests is determined based on experiment, and such a value is chosen that performs well for all types of plates.

-

Validation

Binarized license plate after pre-processing went through cleaning, and both horizontal and vertical projection profile analysis. There is a chance that some characters might get missed out while cleaning if it is touching the boundary, if characters are joined together (usually observed in old discoloured plates), and two or more characters might be grouped into one when threshold value fails in vertical projection profile. To take care of these problems, we run a series of validation steps to make sure the segmented region is indeed a character. The first test is character size test. If the integer ratio of segmented character width and the median character width is greater than one, the region is divided into as many segments each of median width. It is then followed by spacing test. It finds the gap between the segmented characters. The gap at the start and end of the plate row is usually large compared to the gap between the characters. The ratio of start and end gap is not greater than 1.5, none of the gaps between character is greater than the median character width. Besides, if a character is less than 1.5 times the previous character, it is considered a part of previous character. Finally, to ensure that each segmented region contains only one character, connect component analysis is performed and masked by area, height, width, and aspect ratio set to a preset threshold. For all the tests, threshold values are determined by experiment.

- **Segmented Characters**

Based on the number of horizontal, and vertical projection peaks, we determine the type of license plate. If the plate passes tests in all rows, characters are segmented from the license plate image at the specified bounding box of each character, and same for recognition.

Character Recognition

- **Neural Networks:** A neural network is a type of machine learning which models itself after the human brain. This creates an artificial neural network that via an algorithm allows the computer to learn by incorporating new data. Image is a 2dimensional data, containing spatial information of pixel intensity. Before

- feeding the ANN, we need to extract the features from the input image. Feature extraction is a crucial step in image classification tasks.
Feature Extraction: A convolution operation with the kernel is performed to extract the features. This gives rise to a Convolution Neural Network. Convolution operation is the fundamental building blocks of a Convolution Neural Network. CNNs are state of the art algorithms for computer vision problems like object detection, localization, recognition and so on. Filters/kernels in the CNN layers are used to extract the features from the images. This difficult task of assigning the filters parameters is learned in a supervised manner. We emphasize that all weights in all layers of a convolutional network are learned through training. Moreover, the network learns to extract its own features automatically.
- **Convolutional Layer:** When designing a CNN, each convolutional layer within neural network should have the following attribute: Input is a tensor with shape (number of images) x (images width) x (images height) x (images depth) can convolutional kernels whose width and height are hyper-parameters, and whose depth must be equal to that of the image. In our case, image width = image height = 64, image depth = 1, number images in single step of feed forward are 32.
- **Padding layer:** Every time we apply convolution, image size shrinks. Pixels in the corner are used very less than pixels in the middle for convolution. Pixels holding the shape/structure of a character in the boundary won't contribute much during convolutional. So, padding of 0 values is done before feeding to the network. Convolution operation with $(n \times n)$ image with $(f \times f)$ gives $(n - f + 1) \times (n - f + 1)$ output. This shrinkage of output drastically reduces the size of output in the deep layers.
- **Pooling layer:** ConvNet uses pooling to reduce the size of representation to speed the computation. This layer has no parameters to learn. It is based on the

- intuition that most of the features in the kernel size are held by pixels with the largest value (max pooling).

Activation Function: Activation function allows us to transform the given input into a required output that has a certain range. Activation function introduces non-linearity, so as to make the network learn complex patterns.

- **ReLU Layer:** The rectified linear activation function or ReLU for short is a piecewise linear function that will output the input directly if it is positive, otherwise, it will output zero. The ReLU function has been found to be very good for networks with many layers because it can prevent vanishing gradients when training deep networks and fast computation.

$$f(x) = \max(0, x)$$

- **Sigmoid Activation:** Sigmoid function is a mathematical function having a characteristic “S”-shaped curve or sigmoid curve. The Sigmoid function transforms a real value into one that can be interpreted as a probability, such as between 0 and 1, or -1 and 1.
- **Fully Connected Layers:** Features extracted from the Convolution Neural Network are flattened to form a 1-dimensional feature set. This 1- dimensional feature is fed to Fully connected layers, with input from earlier convolution layers (filter height x filter width x number of channels).
- **Optimization:** Optimization is the iterative process of minimizing the model error. It concerns minimizing the loss. A loss function is often referred to as the objective function for the optimization problem. An optimum solution for the objective function is the global minimum value of the loss function for that model. Optimization algorithms: Gradient Descent, Mini-batch gradient descent, RMS prop, Adams Optimization algorithm.

.

CHAPTER 6 RESULTS AND DISCUSSION

6.1 RESULTS AND DISCUSSION

Violation detection and tracking were performed using the YOLO-v8 + DeepSORT algorithm, which was trained on images with data augmentation consisting of various kind of violation classes. This module is evaluated with test images too. The results containing correct prediction and wrong prediction for each type of violation are presented in below figure. It is noted that the mean average precision (mAP) for violations detection is 98.09% for all classes. Below figure shows ROC curve across all the classes of detection.

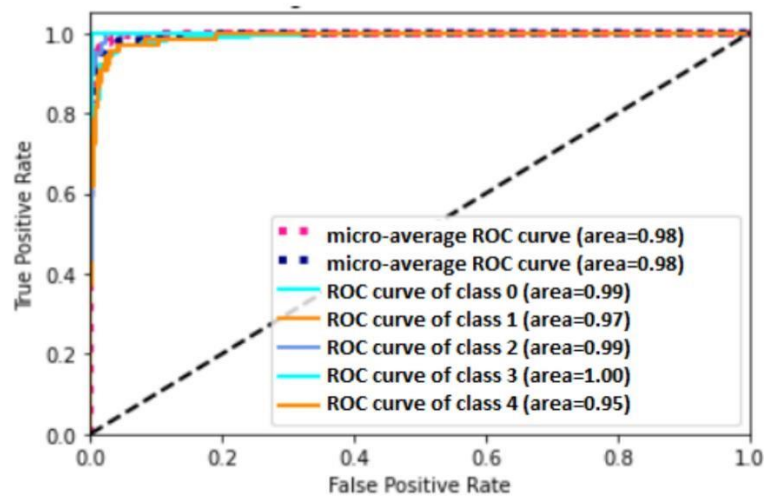


Fig. 6.1: ROC curve

The results for the video feed are presented in. It is noted that 83 out of 93 violations were accurately captured resulting in an accuracy of 90.32%. In addition, it should be noted that the proposed system gave no false positives which meant wrong violations was tagged. On analysing the false negatives, it can be noted that most of the non-detections are mainly due to usage of caps or fancy head gear which the model tags as helmets and hence doesn't capture. Also, most violations of usage of mobiles go undetected as the angle of capture is not able to provide a label of high accuracy. Further, 79 out of 83 number plates was accurately captured giving the accuracy of the Number plate detection module to be 94.04%. This is very much in

line with other State-of-the-Art Models. This is because the same benchmark dataset is used for training and most of literature's detector models work on similar deep learning networks. Finally, 79 out of 93 violations was captured making the system 84.94% accurate which is very high when considering there is no human intervention until this stage and 0% false positive triggers.

Video	Frames per second	Actual Violations	Detected Violations	Violators-Number plate detected	Error in violation detection %	Error in number plate detection %
Video 1	24	5	5	5	0	0.00
Video 2	24	13	11	10	15.38	9.09
Video 3	24	22	20	18	9.09	10.00
Video 4	36	1	1	1	0	0.00
Video 5	34	3	3	3	0	0.00
Video 6	44	13	11	11	15.38	0.00
Video 7	43	14	14	13	0	7.14
Video 8	56	22	18	13	13.64	11.11
Total		93	83	79		

Fig. 6.2: RESULTS OF VIOLATION AND NUMBER PLATE DETECTION ON VIDEO FEED

The 2058 predicted True violations images from the violation detection module are passed to the Number plate detection module. Not all the violators' vehicle number plates are visible or readable, mainly due to camera angles and occlusions. The results of the Number plate detection module are given below figure. It can be seen that out of 1853 readable number plates, 1842 was correctly detected and recognized which gives an accuracy of 99.41%. Thus, showing the effectiveness of using an object detector like Yolo-v4 for the detection of number plates. The performance is very similar to literature performance as its a pure single class detection and processing.

	Number samples	Number plates visible/readable	Correct Prediction	Success Rate (%)
Detection	2058	1853	1849	0.9978
Segmentation	1849	1845	1843	0.9989
Recognition	1843	1843	1842	0.9995

Fig. 6.3 RESULT OF NUMBER PLATE DETECTION ON TEST DATASET

The system demonstrated high accuracy in detecting various traffic violations, including speeding, red light violations, illegal parking, and improper lane usage. By leveraging advanced computer vision algorithms and machine learning models, the system achieved robust performance in real-world scenarios, accurately identifying and documenting violations with minimal false positives. Real-time Detection is one of the notable advantages of the system is its ability to perform real-time violation detection, enabling immediate enforcement actions to be taken. This capability significantly enhances the responsiveness of traffic enforcement agencies, allowing them to promptly address violations and mitigate potential safety risks on the road. The system offers potential cost savings for traffic enforcement agencies. The elimination of manual monitoring and ticketing tasks reduces the need for human resources and operational expenses, leading to more efficient resource allocation and budget management.

CHAPTER 7 CONCLUSIONS

7.1 CONCLUSION AND FUTURE WORK

In this paper, we present an end-to-end system to automate two-wheeler violation detection to have zero or Minimum human intervention for ticketing. Custom model training requires a large amount of labelled data. The task involves creating, gathering, and labelling many proprietary data for the task, which is very time consuming and human resource intensive. In addition, there are issues of good quality images, as most CCTV footages are of low resolution making most data useless. Another challenge observed: inconsistent number plates with fancy fonts, local language number plates, and at times no number plates present in vehicles. These are a few of the challenges in addition to the large computational resource required for efficiently running the system.

For future work, the system can be improved by including other types of road vehicles and detecting corresponding vehicle-related traffic violations to make the system, more robust and complete. Also, architectures can be built to handle multiple camera views and to tackle more violation types. The OCR for license plate detection can be further trained to capture number plates having non-traditional fonts and designs. Additionally, newer object detection models can be experimented with to benchmark models for accuracy and speed.

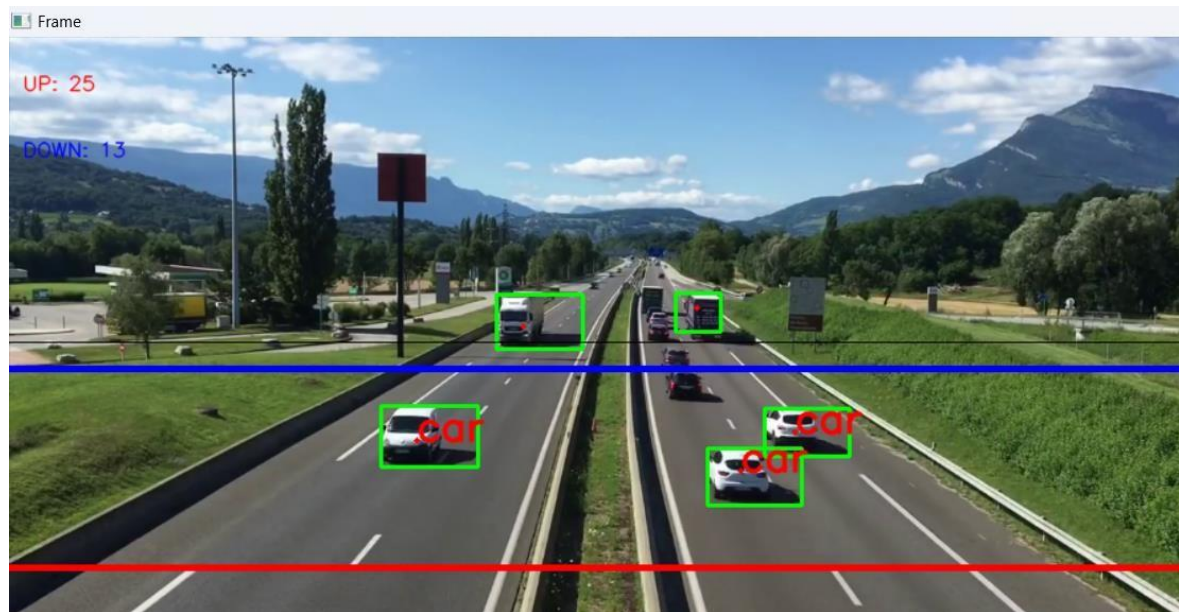
REFERENCES

- [1] Xialing Wang, Li-Min Meng, Biaobiao Zhang (2013). A video-based traffic violation detection system.
- [2] Krishna, Madhav Poddar, Gridhar M K, Amit Suresh Prabhu, Umadevi V (2016). Automated traffic monitoring system using computer vision.
- [3] Katanyoo Klubsuwan, Wittaya Koodtalang (2013). Traffic violation detection using multiple trajectories evaluation of vehicle.
- [4] Aaron Christian, Rhen Anjerome Bedruz (2015). Machine vision for traffic violation detection system through genetic algorithm.
- [5] S. F. Wong, H. C. Mak, C. H. ku, W. I. Ho (2017). Developing advance traffic violation detection system with RFID technology for smart city.
- [6] A. Bochkovskiy, C.-Y. Wang, and H.-Y. Mark Liao, "YOLOv4: Optimal speed and accuracy of object detection," 2020, arXiv:2004.10934.
- [7] R. Smith, "An overview of the Tesseract OCR engine," in Proc. 9th Int.Conf. Document Anal. Recognit. (ICDAR), Sep. 2007, pp. 629–633, doi:10.1109/ICDAR.2007.4376991.
- [8] K. Dahiya, D. Singh, and C. K. Mohan, "Automatic detection of bike riders without helmet using surveillance videos in real-time," in Proc. Int. Joint Conf. Neural Netw. (IJCNN), Jul. 2016, pp. 3046–3051.
- [9] X. Wang, L.-M. Meng, B. Zhang, J. Lu, and K.-L. Du, "A video-based traffic violation detection system," in Proc. Int. Conf. Mech. Sci., Electr. Eng. Comput. (MEC), Dec. 2013, pp. 1191–1194
- [10] N. C. Mallela, R. Volety, R. P. Srinivasa, and R. K. Nadesh, "Detection of the triple riding and speed violation on two-wheelers using deep learning algorithms," *Multimedia Tools Appl.*, vol. 80, no. 6, pp. 8175–8187, Mar. 2021.
- [11] Vandal and Y. Adu-Gyamfi, "Object detection and tracking algorithms for vehicle counting: A comparative analysis," *J. Big Data Anal. Transp.*, vol. 2, pp. 251–261, Nov. 2020.
- [12] S. Kumari, D. K. Gupta, and R. M. Singh, "A novel methodology for vehicle number plate recognition using artificial neural network," in Proc. 3rd Int. Symp. Comput. Vis. Internet, Sep. 2016, pp. 110–114

- [13] J. Singh and B. Bhushan, "Real time Indian license plate detection using deep neural networks and optical character recognition using LSTM tesseract," in Proc. Int. Conf. Comput., Commun., Intell. Syst. (ICCCIS), Oct. 2019, pp. 347–352.
- [14] A. Tonge, S. Chandak, R. Khiste, U. Khan, and L. A. Bewoor, "Traffic rules violation detection using deep learning," in Proc. 4th Int. Conf. Electron., Commun. Aerosp. Technol. (ICECA), Nov. 2020, pp. 1250–1257.
- [15] N. Wojke, A. Bewley, and D. Paulus, "Simple online and realtime tracking with a deep association metric," in Proc. IEEE Int. Conf. Image Process. (ICIP), Sep. 2017, pp. 3645–3649, doi: 10.1109/ICIP.2017.8296962.
- [16] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, "Simple online and realtime tracking," in Proc. IEEE Int. Conf. Image Process. (ICIP), Sep. 2016, pp. 3464–3468, doi: 10.1109/ICIP.2016.7533003.
- [17] Bui-Minh T, Ghita O, Whelan PF, Hoang T, Truong VQ(2012) "Two occluding traffic signs" in Control, Automation and Information on, pp.120-125: IEEE.
- [18] De La Escalera A, Armingol JM, Pastor JM, Rodriguez FJ (2004) Visual sign information extraction and identification by deformable models for intelligent vehicles: IEEE Trans Intell Transp Syst 5(2):57-68.
- [19] Elotmani S, El Hitmy M(2014) "A light traffic signs recognition system," in multimedia Computing and system (ICMCS), 2014 international conference on, pp. 459-464: IEEE.
- [20] Jeon S, Hong B (2016) Monte Carlo simulation-based traffic speed forecasting using historical big data. Futur Gener Comput Syst 65:182-195.

APPENDIX

A.1 SCREENSHOTS





A.2 SOURCE CODE

```

Vehicle_detection_main.py 2 X
C:\Users\amirya\Desktop> 4th year project > project > Module 1 > Vehicle_detection_main.py > ...
1 import cv2
2 import numpy as np
3 import time
4 import vehicles
5
6 cap=cv2.VideoCapture("video.mp4")
7 fgbg=cv2.createBackgroundSubtractorMOG2(detectShadows=False,history=200,varThreshold = 90)
8 kernelOp = np.ones((3,3),np.uint8)
9 kernelOp2 = np.ones((5,5),np.uint8)
10 kernelCl = np.ones((11,11),np.uint8)
11 font = cv2.FONT_HERSHEY_SIMPLEX
12 cars = []
13 max_p_age = 5
14 pid = 1
15 cnt_up=0
16 cnt_down=0
17
18 print("Car counting and classification")
19
20 line_up=400
21 line_down=250
22
23 up_limit=230
24 down_limit=int(4.5*(500/5))
25
26 while(cap.isOpened()):
27     ret,frame=cap.read()
28     frame=cv2.resize(frame,(900,500))
29     for i in cars:
30         i.age_one()
31     fgbg.apply(frame)
32
33     if ret==True:
34         ret,imbin=cv2.threshold(fgbg,200,255,cv2.THRESH_BINARY)
35         mask = cv2.morphologyEx(imbin, cv2.MORPH_OPEN, kernelOp)
36         mask = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernelCl)
37

```

```

Vehicle_detection_main.py 2 X
C:\Users\amiya\Desktop> 4th year project > project > Module 1 > Vehicle_detection_main.py > ...
38
39 (countours0,hierarchy)=cv2.findContours(mask,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_NONE)
40 for cnt in countours0:
41     area=cv2.contourArea(cnt)
42     print(area)
43     if area>300:
44
45         m=cv2.moments(cnt)
46         cx=int(m['m10']/m['m00'])
47         cy=int(m['m01']/m['m00'])
48         x,y,w,h=cv2.boundingRect(cnt)
49
50
51 new=True
52 if cy in range(up_limit,down_limit):
53     for i in cars:
54         if abs(x - i.getX()) <= w and abs(y - i.getY()) <= h:
55             new = False
56             i.updateCoords(cx, cy)
57
58             if i.going_UP(line_down,line_up)==True:
59                 cnt_up+=1
60
61             elif i.going_DOWN(line_down,line_up)==True:
62                 cnt_down+=1
63
64             break
65         if i.getState()=='1':
66             if i.getDir()=='down' and i.getY()>down_limit:
67                 i.setDone()
68             elif i.getDir()=='up' and i.getY()<up_limit:
69                 i.setDone()
70         if i.timedOut():
71             index=cars.index(i)
72             cars.pop(index)
73             del i

```

```

Vehicle_detection_main.py 2 X
C:\Users\amiya\Desktop> 4th year project > project > Module 1 > Vehicle_detection_main.py > ...
75
76     if new==True:
77         p=vehicles.car(pid,cx,cy,max_p_age)
78         cars.append(p)
79         pid+=1
80         cv2.circle(frame, (cx, cy), 2, (0, 0, 255), -1)
81
82
83     img=cv2.rectangle(frame,(x,y),(x+w,y+h),(0,255,0),2)
84
85
86
87
88
89
90
91 for i in cars:
92     cv2.putText(frame, str(i.getId()), (i.getX(), i.getY()), font, 0.3, (255,255,0), 1, cv2.LINE_AA)
93     if line_down+20<= i.getY() <= line_up-20:
94         a = (h + (.74*w)- 100)
95
96         if a >= 0:
97             cv2.putText(frame, "Truck", (i.getX(), i.getY()), font, 1, (0,0,255), 2, cv2.LINE_AA)
98         else:
99             cv2.putText(frame, "car", (i.getX(), i.getY()), font, 1, (0,0,255), 2, cv2.LINE_AA)
100
101 str_up='UP: '+str(cnt_up)
102 str_down='DOWN: '+str(cnt_down)
103 frame=cv2.line(frame,(0,line_up),(900,line_up),(0,0,255),3,8)
104 frame=cv2.line(frame,(0,up_limit),(900,up_limit),(0,0,0),1,8)
105
106 frame=cv2.line(frame,(0,down_limit),(900,down_limit),(255,255,0),1,8)
107 frame = cv2.line(frame, (0, line_down), (900, line_down), (255, 0,0), 3, 8)
108
109
110

```

```

Vehicle_detection_main.py 2 X
C:\Users\amiya\Desktop> 4th year project > project > Module 1 > Vehicle_detection_main.py > ...
109
110
111
112 cv2.putText(frame, str_up, (10, 40), font, 0.5, (0, 0, 255), 1, cv2.LINE_AA)
113 cv2.putText(frame, str_down, (10, 90), font, 0.5, (255, 0, 0), 1, cv2.LINE_AA)
114 cv2.imshow('Frame',frame)
115
116 if cv2.waitKey(1)&&0xff==ord('q'):
117     break
118
119 else:
120     break
121
122 cap.release()
123 cv2.destroyAllWindows()
124

```

```
myMorpho.py 2 X
C:\Users\amiya\Desktop> 4th year project > project > Module 1 > myMorpho.py > ...
1 import numpy as np
2 import cv2
3
4
5
6 def mymorpho(image):
7
8     k1 = np.ones((3,3),np.uint8)
9     erosion = cv2.erode(image,k1,iterations = 1)
10
11     k1 = np.ones((5,5),np.uint8)
12     dilate= cv2.dilate(erosion,k1,iterations = 2)
13
14     k1 = np.ones((5,5),np.uint8)
15     dilate1= cv2.dilate(dilate,k1,iterations = 1)
16
17     k1 = np.ones((13,13),np.uint8)
18     erosion1= cv2.erode(dilate1,k1,iterations = 1)
19     return erosion1
20
21
22
23
24
```

```
vehicles.py X
C:\Users\amiya\Desktop> 4th year project > project > Module 1 > vehicles.py > ...
1 from random import randint
2 import time
3
4 class Car:
5     tracks=[]
6     def __init__(self,i,xi,yi,max_age):
7         self.i=i
8         self.x=xi
9         self.y=yi
10        self.tracks=[]
11        self.done=False
12        self.state='0'
13        self.age=0
14        self.max_age=max_age
15        self.dir=None
16
17    def getTracks(self):
18        return self.tracks
19
20    def getId(self):
21        return self.i
22
23    def getState(self):
24        return self.state
25
26    def getDir(self):
27        return self.dir
28
29    def getX(self):
30        return self.x
31
32    def getY(self):
33        return self.y
34
35    def updateCoords(self, xn, yn):
36        self.age = 0
37        self.tracks.append([self.x, self.y])
38
```

```
vehicles.py X
C:\Users\amiya\Desktop> 4th year project > project > Module 1 > vehicles.py > ...
4 class Car:
35    def updateCoords(self, xn, yn):
36        self.age = 0
37        self.tracks.append([self.x, self.y])
38        self.x = xn
39        self.y = yn
40
41    def setDone(self):
42        self.done = True
43
44    def timedOut(self):
45        return self.done
46
47    def going_UP(self, mid_start, mid_end):
48        if len(self.tracks)>=2:
49            if self.state=='0':
50                if self.tracks[-1][1]<mid_end and self.tracks[-2][1]>=mid_end:
51                    state='1'
52                    self.dir='up'
53                    return True
54                else:
55                    return False
56            else:
57                return False
58        else:
59            return False
60
61    def going_DOWN(self, mid_start, mid_end):
62        if len(self.tracks)>=2:
63            if self.state=='0':
64                if self.tracks[-1][1]>=mid_start and self.tracks[-2][1]<=mid_start:
65                    state='1'
66                    self.dir='down'
67                    return True
68                else:
69                    return False

```

```

vehicles.py X
C:\Users\amiya\Desktop> 4th year project > project > Module 1 > vehicles.py > ...
4 class Car:
60
61     def going_DOWN(self,mid_start,mid_end):
62         if len(self.tracks)>2:
63             if self.state=='0':
64                 if self.tracks[-1][1]>mid_start and self.tracks[-2][1]<mid_start:
65                     start='1'
66                     self.dir='down'
67                     return True
68             else:
69                 return False
70         else:
71             return False
72     else:
73         return False
74
75     def age_one(self):
76         self.age+=1
77         if self.age>self.max_age:
78             self.done=True
79         return True
80
81
82 class MultiCar:
83     def __init__(self,cars,xi,yi):
84         self.cars=cars
85         self.x=xi
86         self.y=yi
87         self.tracks=[]
88         self.done=False
89

```

```

speed_checkpy 2 X
C:\Users\amiya\Desktop> 4th year project > project > Module 2 > speed_checkpy > ...
1 import cv2
2 import dlib
3 import time
4 import threading
5 import math
6
7 carCascade = cv2.CascadeClassifier('myhaar.xml')
8 video = cv2.VideoCapture('cars.mpg')
9
10 WIDTH = 1280
11 HEIGHT = 720
12
13
14 def estimateSpeed(location1, location2):
15     d_pixels = math.sqrt(math.pow(location2[0] - location1[0], 2) + math.pow(location2[1] - location1[1], 2))
16     ppm = 8.8
17     d_meters = d_pixels / ppm
18     fps = 18
19     speed = d_meters * fps * 3.6
20     return speed
21
22
23 def trackMultipleObjects():
24     rectangleColor = (0, 255, 0)
25     frameCounter = 0
26     currentCarID = 0
27     fps = 0
28
29     carTracker = {}
30     carNumbers = {}
31     carLocation1 = {}
32     carLocation2 = {}
33     speed = [None] * 1000
34
35     out = cv2.VideoWriter('outpy.avi',cv2.VideoWriter_fourcc('M','J','P','G'), 10, (WIDTH,HEIGHT))
36

```

```

speed_checkpy 2 X
C:\Users\amiya\Desktop> 4th year project > project > Module 2 > speed_checkpy > ...
23 def trackMultipleObjects():
37
38     while True:
39         start_time = time.time()
40         rc, image = video.read()
41         if type(image) == type(None):
42             break
43
44         image = cv2.resize(image, (WIDTH, HEIGHT))
45         resultImage = image.copy()
46
47         frameCounter = frameCounter + 1
48
49         carIDToDelete = []
50
51         for carID in carTracker.keys():
52             trackingQuality = carTracker[carID].update(image)
53
54             if trackingQuality < 7:
55                 carIDToDelete.append(carID)
56
57         for carID in carIDToDelete:
58             print ('Removing carID ' + str(carID) + ' from list of trackers.')
59             print ('Removing carID ' + str(carID) + ' previous location.')
60             print ('Removing carID ' + str(carID) + ' current location.')
61             carTracker.pop(carID, None)
62             carLocation1.pop(carID, None)
63             carLocation2.pop(carID, None)
64
65         if not (frameCounter % 10):
66             gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
67             cars = carCascade.detectMultiScale(gray, 1.1, 13, 18, (24, 24))
68
69             for (x, y, _w, _h) in cars:
70                 x = int(x)
71                 y = int(y)
72                 w = int(w)

```

```

speed_check.py 2 X
C:\Users\amiya\Desktop> 4th year project > project > Module 2 > speed_check.py > trackMultipleObjects
23 def trackMultipleObjects():
69     for (x, y, w, h) in cars:
70         x = int(x)
71         y = int(y)
72         w = int(w)
73         h = int(h)
74
75         x_bar = x + 0.5 * w
76         y_bar = y + 0.5 * h
77
78         matchCarID = None
79
80         for carID in carTracker.keys():
81             trackedPosition = carTracker[carID].get_position()
82
83             t_x = int(trackedPosition.left())
84             t_y = int(trackedPosition.top())
85             t_w = int(trackedPosition.width())
86             t_h = int(trackedPosition.height())
87
88             t_x_bar = t_x + 0.5 * t_w
89             t_y_bar = t_y + 0.5 * t_h
90
91             if ((t_x <= x_bar <= (t_x + t_w)) and (t_y <= y_bar <= (t_y + t_h)) and (x <= t_x_bar <= (x + w)) and (y <= t_y_bar <= (y + h))):
92                 matchCarID = carID
93
94         if matchCarID is None:
95             print('creating new tracker ' + str(currentCarID))
96
97             tracker = dlib.correlation_tracker()
98             tracker.start_track(image, dlib.rectangle(x, y, x + w, y + h))
99
100             carTracker[currentCarID] = tracker
101             carLocation1[currentCarID] = [x, y, w, h]
102
103

```

```

speed_check.py 2 X
C:\Users\amiya\Desktop> 4th year project > project > Module 2 > speed_check.py > trackMultipleObjects
23 def trackMultipleObjects():
91     if ((t_x <= x_bar <= (t_x + t_w)) and (t_y <= y_bar <= (t_y + t_h)) and (x <= t_x_bar <= (x + w)) and (y <= t_y_bar <= (y + h))):
92         matchCarID = carID
93
94     if matchCarID is None:
95         print('creating new tracker ' + str(currentCarID))
96
97         tracker = dlib.correlation_tracker()
98         tracker.start_track(image, dlib.rectangle(x, y, x + w, y + h))
99
100         carTracker[currentCarID] = tracker
101         carLocation1[currentCarID] = [x, y, w, h]
102
103         currentCarID = currentCarID + 1
104
105
106
107     for carID in carTracker.keys():
108         trackedPosition = carTracker[carID].get_position()
109
110         t_x = int(trackedPosition.left())
111         t_y = int(trackedPosition.top())
112         t_w = int(trackedPosition.width())
113         t_h = int(trackedPosition.height())
114
115         cv2.rectangle(resultImage, (t_x, t_y), (t_x + t_w, t_y + t_h), rectanglecolor, 4)
116
117         carLocation2[carID] = [t_x, t_y, t_w, t_h]
118
119     end_time = time.time()
120
121     if not (end_time == start_time):
122         fps = 1.0/(end_time - start_time)
123

```

```

speed_check.py 2 X
C:\Users\amiya\Desktop> 4th year project > project > Module 2 > speed_check.py > trackMultipleObjects
23 def trackMultipleObjects():
118     end_time = time.time()
119
120     if not (end_time == start_time):
121         fps = 1.0/(end_time - start_time)
122
123
124
125     for i in carLocation1.keys():
126         if frameCounter % 1 == 0:
127             [x1, y1, w1, h1] = carLocation1[i]
128             [x2, y2, w2, h2] = carLocation2[i]
129
130             carLocation1[i] = [x2, y2, w2, h2]
131
132             if [x1, y1, w1, h1] != [x2, y2, w2, h2]:
133                 if (speed[i] == None or speed[i] == 0) and y1 >= 275 and y1 <= 285:
134                     speed[i] = estimateSpeed([x1, y1, w1, h1], [x2, y2, w2, h2])
135
136                 if speed[i] != None and y1 >= 180:
137                     cv2.putText(resultImage, str(int(speed[i])) + " km/hr", (int(x1 + w1/2), int(y1-5)), cv2.FONT_HERSHEY_SIMPLEX, 0.75, (255, 255, 255), 2)
138
139
140     cv2.imshow('result', resultImage)
141
142     if cv2.waitKey(33) == 27:
143         break
144
145     cv2.destroyAllWindows()
146
147
148 if __name__ == '__main__':
149     trackMultipleObjects()
150

```

```

license_plate_detection.py 5 X
C:\Users\amiya\Desktop> 4th year project > project > Module 3 > license_plate_detection.py > ...
1 import cv2
2 import pytesseract
3 pytesseract.pytesseract_cmd=r"C:\Program Files\Tesseract-OCR\tesseract.exe"
4 import numpy as np
5 from matplotlib import pyplot as plt
6 import os
7
8 files = [
9     {'name': 'car_1.jpg', 'color': 'white'},
10    {'name': 'car_2.jpg', 'color': 'special', 'option':{
11        'type': 'color',
12        'lower_white': np.array([0,0,200], dtype=np.uint8),
13        'upper_white': np.array([255,30,255], dtype=np.uint8),
14        'no_bitwise': False
15    }},
16    {'name': 'car_3.jpg', 'color': 'yellow'},
17    {'name': 'car_4.jpg', 'color': 'white_bg'},
18    {'name': 'car_5.jpg', 'color': 'white'},
19    {'name': 'car_6.jpg', 'color': 'white'},
20    {'name': 'car_7.jpg', 'color': 'white'},
21    {'name': 'car_8.jpg', 'color': 'yellow'},
22    {'name': 'car_9.jpg', 'color': 'white'},
23    {'name': 'car_10.jpg', 'color': 'white'},
24    {'name': 'car_11.jpg', 'color': 'white'},
25    {'name': 'car_12.jpg', 'color': 'yellow'},
26    {'name': 'car_13.jpg', 'color': 'yellow'},
27    {'name': 'car_14.jpg', 'color': 'yellow'}
28 ]
29
30 def img_process(ori_img, color, option):
31     image = cv2.cvtColor(ori_img, cv2.COLOR_BGR2GRAY)
32     mask = cv2.threshold(image, thresh=200, maxval=255, type=cv2.THRESH_BINARY+cv2.THRESH_OTSU)
33     if option is not None and option['no_bitwise']:
34         img_mask = image
35     else:
36         img_mask = cv2.bitwise_and(image, mask)

```

```

license_plate_detection.py 5 X
C:\Users\amiya\Desktop> 4th year project > project > Module 3 > license_plate_detection.py > ...
30 def img_process(ori_img, color, option):
31
32     if color in ('white', 'yellow'):
33         hsv = cv2.cvtColor(ori_img, cv2.COLOR_BGR2HSV)
34         h, s, v1 = cv2.split(hsv)
35         if color == 'white':
36             lower_white = np.array([0,0,150], dtype=np.uint8)
37             upper_white = np.array([255,40,255], dtype=np.uint8)
38         elif color == 'yellow':
39             lower_white = np.array([20, 100, 100], dtype=np.uint8)
40             upper_white = np.array([30, 255, 255], dtype=np.uint8)
41         res_mask = cv2.inRange(hsv, lower_white, upper_white)
42         res_img = cv2.bitwise_and(v1, image, mask=res_mask)
43     else:
44         if color == 'special' and option is not None and option['type'] == 'color':
45             print(":::::Special - color")
46             hsv = cv2.cvtColor(ori_img, cv2.COLOR_BGR2HSV)
47             h, s, v1 = cv2.split(hsv)
48             upper_white = option['upper_white']
49             lower_white = option['lower_white']
50             res_mask = cv2.inRange(hsv, lower_white, upper_white)
51             res_img = cv2.bitwise_and(v1, image, mask=res_mask)
52         else:
53             res_img = img_mask
54     return res_img
55
56 def contour(res_img, result, color):
57     contours, _ = cv2.findContours(
58         res_img,
59         cv2.RETR_TREE,
60         cv2.CHAIN_APPROX_SIMPLE
61     )
62
63     contours = sorted(contours, key = cv2.contourArea, reverse = True)[:5]
64
65     NumberPlateCnt = None
66     found = False

```

```

license_plate_detection.py 5 X
C:\Users\amiya\Desktop> 4th year project > project > Module 3 > license_plate_detection.py > ...
63 def contour(res_img, result, color):
73     lt, rb = [10000, 10000], [0, 0]
74
75     if color == 'white_bg':
76         for c in contours:
77             peri = cv2.arcLength(c, True)
78             approx = cv2.approxPolyDP(c, 0.06 * peri, True)
79             if len(approx) == 4:
80                 found = True
81                 NumberPlateCnt = approx
82                 break
83     if found:
84         cv2.drawContours(result, [NumberPlateCnt], -1, (255, 0, 255), 2)
85
86         for point in NumberPlateCnt:
87             cur_cx, cur_cy = point[0][0], point[0][1]
88             if cur_cx < lt[0]: lt[0] = cur_cx
89             if cur_cx > rb[0]: rb[0] = cur_cx
90             if cur_cy < lt[1]: lt[1] = cur_cy
91             if cur_cy > rb[1]: rb[1] = cur_cy
92
93         cv2.circle(result, (lt[0], lt[1]), 2, (150, 200, 255), 2)
94         cv2.circle(result, (rb[0], rb[1]), 2, (150, 200, 255), 2)
95
96         crop = res_img[lt[1]:rb[1], lt[0]:rb[0]]
97     else:
98         crop = res_img.copy()
99     elif len(contours) > 0:
100         hull = cv2.convexHull(contours[0])
101         approx2 = cv2.approxPolyDP(hull, 0.01 * cv2.arcLength(hull, True), True)
102         cv2.drawContours(result, [approx2], -1, (255, 0, 255), 2, linestyle=8)
103
104         for point in approx2:
105             cur_cx, cur_cy = point[0][0], point[0][1]
106             if cur_cx < lt[0]: lt[0] = cur_cx
107             if cur_cx > rb[0]: rb[0] = cur_cx
108             if cur_cy < lt[1]: lt[1] = cur_cy

```

```

license_plate_detection.py 5 X
C:\Users\amiya\Desktop> 4th year project > project > Module 3 > license_plate_detection.py > ...
109         if cur_cy > rb[1]: rb[1] = cur_cy
110
111         cv2.circle(result, (lt[0], lt[1]), 2, (150, 200, 255), 2)
112         cv2.circle(result, (rb[0], rb[1]), 2, (150, 200, 255), 2)
113
114         crop = res_img[lt[1]:rb[1], lt[0]:rb[0]]
115     else:
116         crop = res_img.copy()
117     return crop
118
119 def show_plt(crop_img, img_name, directory, flag):
120
121     plt.subplot(235), plt.imshow(crop_img, cmap='gray')
122     plt.title('Number Plate Cropped'), plt.xticks([]), plt.yticks([])
123     plt.suptitle(img_name)
124
125     if flag:
126         plt.show()
127
128     if not os.path.exists(directory):
129         os.makedirs(directory)
130
131     cv2.imwrite("{} / {}".format(directory, img_name), crop_img)
132
133     return pytesseract.image_to_string(crop_img, config='--psm 11')
134
135 def main(flag):
136     source_directory = "input"
137     license_number = {}
138     for file in files:
139         print("License Plates - {}".format(file['name']))
140         ori_img = cv2.imread("{} / {}".format(source_directory, file['name']))
141         option = None
142         if file['color'] == 'special':
143             option = file['option']
144         img_name = file['name']
145         result = ori_img
146         res_img = img_process(ori_img.copy(), file['color'], option)
147         crop_img = contour(res_img, result, file['color'])
148         number_text = show_plt(crop_img, img_name, "output", flag)
149         license_number[img_name] = number_text
150
151     print(license_number)
152
153 if __name__ == '__main__':
154     flag = True
155     main(flag)
156

```

```

license_plate_detection.py 5 X
C:\Users\amiya\Desktop> 4th year project > project > Module 3 > license_plate_detection.py > ...
153     return pytesseract.image_to_string(crop_img, config='--psm 11')
154
155 def main(flag):
156     source_directory = "input"
157     license_number = {}
158     for file in files:
159         print("License Plates - {}".format(file['name']))
160         ori_img = cv2.imread("{} / {}".format(source_directory, file['name']))
161         option = None
162         if file['color'] == 'special':
163             option = file['option']
164         img_name = file['name']
165         result = ori_img
166         res_img = img_process(ori_img.copy(), file['color'], option)
167         crop_img = contour(res_img, result, file['color'])
168         number_text = show_plt(crop_img, img_name, "output", flag)
169         license_number[img_name] = number_text
170
171     print(license_number)
172
173 if __name__ == '__main__':
174     flag = True
175     main(flag)
176

```

```
JS server.js index.html X # styles.css
public > index.html > ...
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>E-Challan Generator</title>
7   <link rel="stylesheet" href="styles.css">
8 </head>
9 <body>
10   <div class="container">
11     <h1>E-Challan Generator</h1>
12     <input type="text" id="numberPlateInput" placeholder="Enter Number Plate">
13     <button onclick="generateChallan()">Generate E-Challan</button>
14     <div id="result"></div>
15   </div>
16   <script src="script.js"></script>
17 </body>
18 </html>
```

```
JS server.js JS script.js # styles.css X
public > # styles.css > ...
48 }
49
50 /* Style for result area */
51 #result {
52   margin-top: 20px;
53   font-weight: bold;
54 }
55
56 /* Style for the footer */
57 footer {
58   background-color: #007bff; /* Blue footer */
59   color: white;
60   text-align: center;
61   padding: 10px 0;
62   position: fixed;
63   bottom: 0;
64   left: 0;
65   width: 100%;
66 }
67
```

```
JS server.js JS script.js X # styles.css
public > JS script.js > generateChallan
1 function generateChallan() {
2   const numberPlate = document.getElementById("numberPlateInput").value;
3   fetch("/generate-challan", {
4     method: "POST",
5     headers: {
6       "Content-Type": "application/json"
7     },
8     body: JSON.stringify({ numberPlate: numberPlate })
9   })
10  .then(response => response.json())
11  .then(data => {
12    document.getElementById("result").innerText = data.message;
13  })
14  .catch(error => {
15    console.error('Error:', error);
16  });
17 }
18
```

```
JS server.js X Go Back (Alt+LeftArrow)
JS server.js > ...
1 const express = require('express');
2 const sqlite3 = require('sqlite3').verbose();
3 const path = require('path');
4
5 const app = express();
6 const port = 3000;
7
8 const db = new sqlite3.Database('e_challan.db');
9
10 db.serialize(() => {
11   db.run('CREATE TABLE IF NOT EXISTS e_challans (
12     id INTEGER PRIMARY KEY AUTOINCREMENT,
13     number_plate TEXT,
14     violation TEXT,
15     fine_amount INTEGER
16   )');
17 });
18
19 app.use(express.json());
20 app.use(express.static(path.join(__dirname, 'public')));
21
22 app.post('/generate-challan', (req, res) => {
23   const numberPlate = req.body.numberPlate;
24   const violation = "Speeding";
25   const fineAmount = 1000;
26
27   db.run('INSERT INTO e_challans (number_plate, violation, fine_amount) VALUES (?, ?, ?)', [numberPlate, violation, fineAmount], function(err) {
28     if (err) {
29       console.error(err.message);
30       res.status(500).json({ message: 'Error generating e-challan.' });
31       return;
32     }
33     res.json({ message: `E-Challan generated for ${numberPlate}. Violation: ${violation}. Fine Amount: ${fineAmount}` });
34   });
35 });
36
37 app.get('/', (req, res) => {
```

A.3 RESEARCH PAPER

TRAFFIC VIOLATION DETECTION AND AUTOMATED TICKETING USING COMPUTER VISION

Amiya Kumar Barik
Department of Computer Science
Sathyabama Institute of Science and Technology

Mrs.Karthika J
Assistant Professor
Department of CSE
Sathyabama Institute of Science and Technology

ABSTRACT

India's population is massive and still rising, due to which there arises the problem of bad management of traffic signals and un supervised roads. A total of 72 thousand personnel of traffic police are there in all over India. A total 2 crore vehicles are currently on roads. It is clear that monitoring traffic physically is not sufficient to manage and monitor such high traffic volumes and trace violation. In addition, a lot of the current solutions aren't flexible enough to identify, analyse, and monitor the wide range of vehicle types, dynamic traffic patterns, and road layouts. The control systems in many cities are still antiquated and cannot be efficiently scaled up to variety and randomness of traffic. This has led to the identification of multiple infringers. As a result, the offenders cause more severe traffic accidents that put other people's lives as well as their own at risk. Therefore, we require a mechanism to swiftly and effectively take action against the owner of the violated vehicle and monitor the traffic continually (in real-time). Governments are increasingly utilizing computer vision based systems to monitor traffic, identify infractions, and issue citations, as traffic laws may not be enforced or perceived as non-existent, potentially affecting behaviour negatively.

INTRODUCTION

In India, the majority of traffic intersections are still either entirely unmonitored or only partially so, which causes significant traffic jams and human error. Research made by Ministry of Road Transport and Highways showed that, two-wheeler riders made up more than one-third (37%) of those who died in traffic accidents in 2019. In addition, the failure to wear a helmet caused 44,666 deaths in 2019 (30,140 drivers and 14,510 million passengers), or 29.80% of all road accident deaths. These are just small number of possible violations that cause to an accident, others are over speeding,

crossing the red light, using phone while driving etc. Thus, statistics have shown that most of the traffic related incident have occurred due to violation of traffic rules and regulations. Manual monitoring requires human labour and is very tedious task. Hence by using Computer Vision technology, we can overcome this problem. By computer vision technology not only we can monitor traffic, we can also detect any possible traffic violation and take action by issuing ticket/challan. Automatic ticketing system offer a solution to the problem of manual traffic ticketing. By eliminating errors and bias, these systems can improve accuracy and efficiency, leading to safer and more efficient roads.

OBJECTIVES

The usage of computer vision and machine learning will provide us to monitor roads 24/7 without the need for human interaction. Which will greatly help in monitoring remote area or monitoring in off-duty hours. Detection of traffic violation such as (crossing red light, not wearing helmet, over speeding, etc.) can be now done with such accuracy and without much error. Thus, freeing up traffic police officers for more complex tasks.

RELATED WORKS

Xialing Wang, Li-Min Meng, Biaobiao Zhang (2013). A video-based traffic violation detection system[1]. The journal presents an algorithm called background updating algorithm, and features-based tracking for traffic violation detection, utilizing OpenCV in C++.

Katanyoo Klubsuan, Wittaya Koodtalang (2013). Traffic violation detection using multiple trajectories evaluation of vehicle [3]. The proposed system uses mean square displacement (MSD) to detect red-light and lane-change violations by analyzing the trajectories of multiple vehicles.

Aaron Christian, Rhen Anjerome Bedruz (2015). Machine vision for traffic violation detection system through genetic algorithm[4]. The paper introduces a algorithm for detecting traffic violations, specifically pedestrian-lane blocking, using a BD method and a genetic algorithm for detection.

S. F. Wong, H. C. Mak, C. H. ku, W. I. Ho (2017). Developing advance traffic violation detection system with RFID technology for smart city [5]. The study proposes a system that uses computer vision and RFID technology to detect traffic conditions instantly, calculates detected objects, and proposes a method for background subtraction in motion object detection.

OPEN PROBLEMS IN EXISTING SYSTEM

The background subtraction approach is the most often used algorithm in projects; however, it is not suitable for handling abrupt changes in illumination or lighting.

Many programs focus solely on certain types of traffic infractions, such as speeding, running red lights, wearing helmets, etc., rather than being entire or comprehensive.

Very few initiatives really attempt to issue e-challans or citations by identifying the license plate when an infraction is detected.

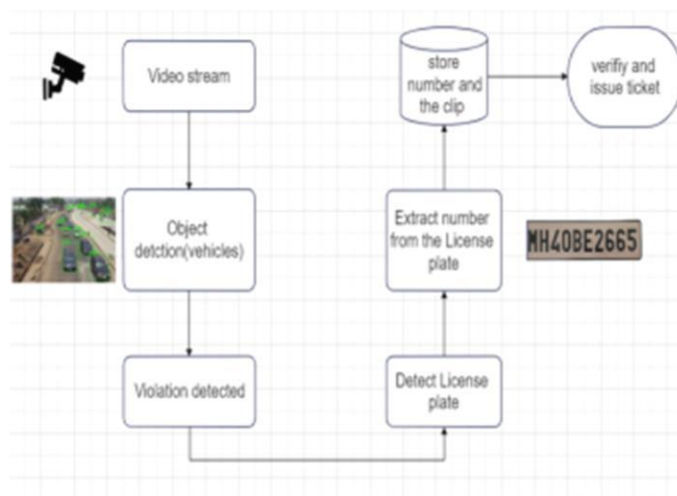
While live feeds necessitate real-time processing, frame preparation, and real-time prediction, the majority of studies focus on detection from pre-processed images.

PROPOSED SYSTEM

The proposed methodology is to make a system, which can detect traffic violations, with minimum human interaction, cost, with very high accuracy and ticketing system. In order to get high accuracy with speed we are using YOLOv8, which out-performs all its previous versions.

Then by using deepSORT we can track those objects (vehicles) in order to remove duplicate counts. We strategically use the same YOLOv8 for violation detection and to retrieve license plate to prevent unwanted computation of image processing, which save lot of computational resources.

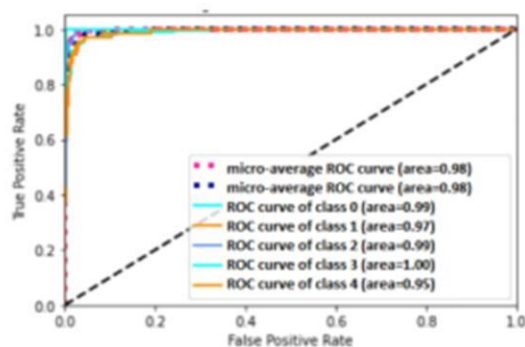
Finally, after getting the number of the violator vehicle's license plate, we can store it and further process it for verification and issuing e-challan/tickets.



OBJECT DETECTION AND TRACKING

Frames are taken as a input (from the video) for object detection which is performed by YOLO algorithm.

In our case the objects are vehicles. (pre-trained). The object is then tracked using deepSORT algorithm to prevent repetitive counting. It also ensures to capture the vehicles in multiple frames.



VIOLATION DETECTION

The dataset obtained from various online sources, (especially Kaggle) is used to train our violation detection module.

It consist of roughly 6000 labelled images for training and 1200 images for testing. Dataset consist of traffic violation such as not wearing helmet, crossing red light, over speeding etc.

Video	Frames per second	Actual Violations	Detected Violations	Violators-Number plate detected	Error in violation detection %	Error in number plate detection %
Video 1	24	5	5	5	0	0.00
Video 2	24	13	11	10	15.38	9.09
Video 3	24	22	20	18	9.09	10.00
Video 4	36	1	1	1	0	0.00
Video 5	34	3	3	3	0	0.00
Video 6	44	13	11	11	15.38	0.00
Video 7	43	14	14	13	0	7.14
Video 8	56	22	18	13	13.64	11.11
Total		93	83	79		

	Accuracy samples	Precision visible/readable	Recall Prediction	Time per detection Rate (%)
Aniruddha et al.[9]	0.8955	0.8883	0.9208	1.08 sec
Mallela et al.[13]	0.9174	0.9139	0.9415	.22 sec
Proposed Methodology	0.9874	0.9886	0.9890	0.09 sec

LICENSE/NUMBER PLATE RECOGNITION

After detecting a violated vehicle, its license plate is recognized by ANPR (automatic number plate recognition).

It is done by tesseractOCR which extracts the alphanumeric text in the license plate for further actions.

Human interaction is still required to validate the traffic violation because the algorithm is not 100% accurate.

	Number samples	Number plates visible/readable	Correct Prediction	Success Rate (%)
Detection	2058	1853	1849	0.9978
Segmentation	1849	1845	1843	0.9989
Recognition	1843	1843	1842	0.9995

	Total Violations	Violations detected	Total number plates detected	Success Rate (%)	Total Time
Mallela et al.[13]	23	14	6	0.2609	345 seconds
+ Singh et al.[18]					
Proposed Methodology	23	21	17	0.7391	198 seconds

	Number samples	Number plates visible/readable	Correct Prediction	Success Rate (%)
Detection	2058	1853	1849	0.9978
Segmentation	1849	1845	1843	0.9989
Recognition	1843	1843	1842	0.9995

CONCLUSION

This study presents an end-to-end automated traffic violation detection system that requires little to no human intervention at all during the ticketing process. Much labelled data is needed for custom model training. The work is labour, time-intensive and requires a large amount of data to be created, cleaned, and labelled. Furthermore, there are problems with high-quality photos because most CCTV footage has low resolution, which renders much of the data meaningless. Another issue seen was the irregular use of fancy fonts on number plates, local language plates, and sometimes the licence plate is not even present. These are some of the problem we faced besides the high need to computational power to run the system effectively.

REFERENCES

- [1] Xialing Wang, Li-Min Meng, Biaobiao Zhang (2013). A video-based traffic violation detection system.
- [2] Krishna, Madhav Poddar, Gridhar M K, Amit Suresh Prabhu, Umadevi V (2016). Automated traffic monitoring system using computer vision.
- [3] Katanyoo Klubsuwan, Wittaya Koodtalang (2013). Traffic violation detection using multiple trajectories evaluation of vehicle.
- [4] Aaron Christian, Rhen Anjerome Bedruz (2015). Machine vision for traffic violation detection system through genetic algorithm.
- [5] S. F. Wong, H. C. Mak, C. H. ku, W. I. Ho (2017). Developing advance traffic violation detection system with RFID technology for smart city.
- [6] A. Bochkovskiy, C.-Y. Wang, and H.-Y. Mark Liao, "YOLOv4: Optimal speed and accuracy of object detection," 2020, arXiv:2004.10934.
- [7] R. Smith, "An overview of the Tesseract OCR engine," in Proc. 9th Int.Conf. Document Anal. Recognit. (ICDAR), Sep. 2007, pp. 629–633, doi:10.1109/ICDAR.2007.4376991.

- [8] K. Dahiya, D. Singh, and C. K. Mohan, "Automatic detection of bike riders without helmet using surveillance videos in real-time," in Proc. Int. Joint Conf. Neural Netw. (IJCNN), Jul. 2016, pp. 3046–3051.
- [9] X. Wang, L.-M. Meng, B. Zhang, J. Lu, and K.-L. Du, "A video-based traffic violation detection system," in Proc. Int. Conf. Mech. Sci., Electr. Eng. Comput. (MEC), Dec. 2013, pp. 1191–1194.
- [10] N. C. Mallela, R. Volety, R. P. Srinivasa, and R. K. Nadesh, "Detection of the triple riding and speed violation on two-wheelers using deep learning algorithms," *Multimedia Tools Appl.*, vol. 80, no. 6, pp. 8175–8187, Mar. 2021.
- [11] Vandal and Y. Adu-Gyamfi, "Object detection and tracking algorithms for vehicle counting: A comparative analysis," *J. Big Data Anal. Transp.*, vol. 2, pp. 251–261, Nov. 2020.
- [12] S. Kumari, D. K. Gupta, and R. M. Singh, "A novel methodology for vehicle number plate recognition using artificial neural network," in Proc. 3rd Int. Symp. Comput. Vis. Internet, Sep. 2016, pp. 110–114.
- [13] J. Singh and B. Bhushan, "Real time Indian license plate detection using deep neural networks and optical character recognition using LSTM tesseract," in Proc. Int. Conf. Comput., Commun., Intell. Syst. (ICCCIS), Oct. 2019, pp. 347–352.
- [14] A. Tonge, S. Chandak, R. Khiste, U. Khan, and L. A. Bewoor, "Traffic rules violation detection using deep learning," in Proc. 4th Int. Conf. Electron., Commun. Aerosp. Technol. (ICECA), Nov. 2020, pp. 1250–1257.
- [15] N. Wojke, A. Bewley, and D. Paulus, "Simple online and realtime tracking with a deep association metric," in Proc. IEEE Int. Conf. Image Process. (ICIP), Sep. 2017, pp. 3645–3649, doi: 10.1109/ICIP.2017.8296962.
- [16] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, "Simple online and realtime tracking," in Proc. IEEE Int. Conf. Image Process. (ICIP), Sep. 2016, pp. 3464–3468, doi: 10.1109/ICIP.2016.7533003.
- [17] Bui-Minh T, Ghita O, Whelan PF, Hoang T, Truong VQ(2012) "Two occluding traffic signs" in Control, Automation and Information on, pp.120-125: IEEE.
- [18] De La Escalera A, Armingol JM, Pastor JM, Rodriguez FJ (2004) Visual sign information extraction and identification by deformable models for intelligent vehicles: IEEE Trans Intell Transp Syst 5(2):57-68.
- [19] Elotmani S, El Hitmy M(2014) "A light traffic signs recognition system," in multimedia Computing and system (ICMCS), 2014 international conference on, pp. 459-464: IEEE.
- [20] Jeon S, Hong B (2016) Monte Carlo simulation-based traffic speed forecasting using historical big data. *Futur Gener Comput Syst* 65:182-195.

A.4 PRESENTATION CERTIFICATE

