ABSTRACT

- Efficient traffic management has become a critical challenge in cities due to rapid urbanization and the growing number of vehicles. Traditional traffic signals, often based on fixed timings, struggle to adapt to actual traffic conditions, leading to inefficient vehicle flow, longer wait times, and congestion, especially during peak hours.

- This project introduces a practical solution using computer vision tools like **YOLO and Deep SORT** to optimize traffic flow at intersections. By analyzing live video feeds from strategically placed cameras, the system continuously assesses traffic density, vehicle types, and movement patterns, adjusting signal timings based on real-time data. This adaptive approach reduces congestion and waiting times by responding to current traffic conditions.

- Vehicle tracking provides further insights, allowing unique monitoring of vehicles across frames, which can support long-term traffic pattern analysis.

- An **admin dashboard** allows remote control of signals, enabling traffic managers to monitor and adjust signals in real-time. This feature provides essential insights, empowering city officials to make more effective traffic management decisions.

- The key benefits of this system include reduced travel time, lower fuel consumption, and improved road safety. This approach is designed to be scalable across different urban environments, offering a reliable solution to modern traffic complexities and helping cities move toward smoother traffic flow.

PROBLEM STATEMENT

Traffic management has increasingly become complex and hence an effective solution is needed to automate vehicle tracking and the management of vehicle flow in signals. CCTV footage is analyzed to compute the optimal timing & dynamically update the signal based on traffic density.

PROPOSED WORK

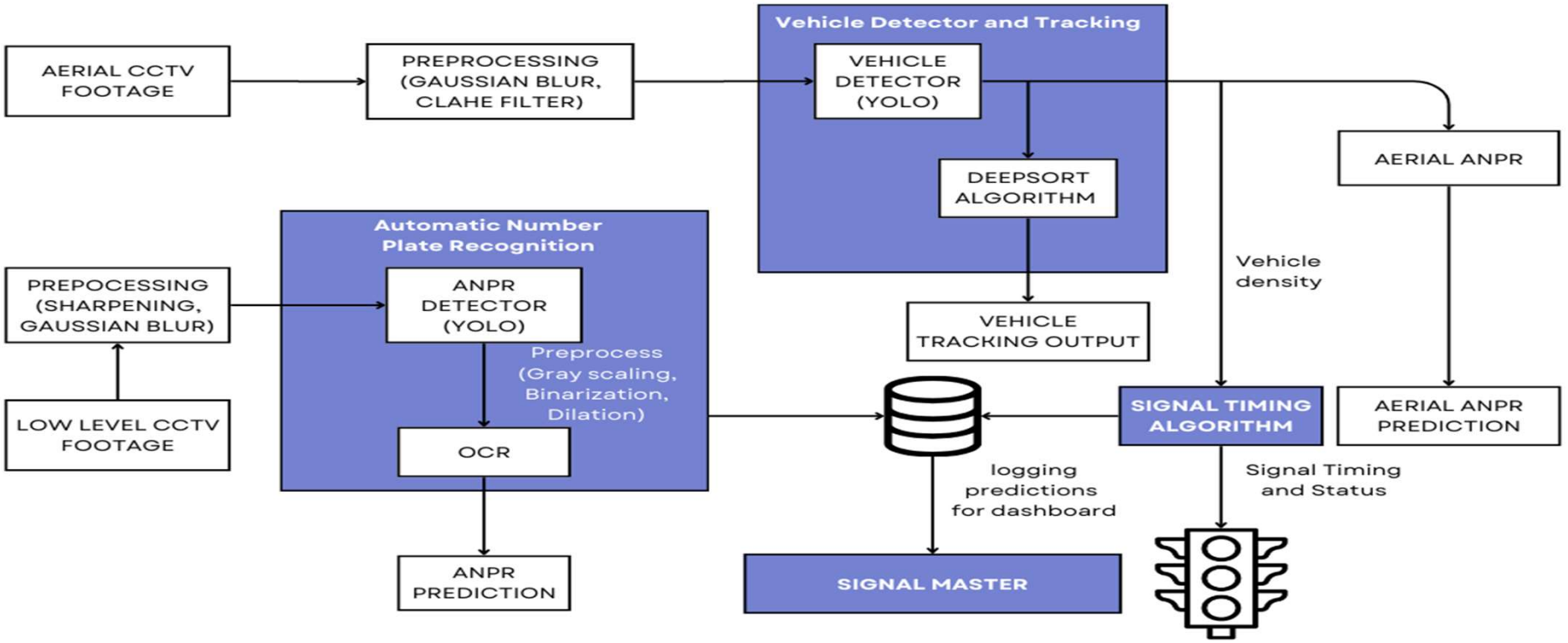

Fig.1. Architecture Diagram

HARDWARE & SOFTWARE REQUIREMENTS

- **Name of the Processor** : Intel i5 or AMD Ryzen 5 3500U
- **RAM** : 8 GB
- **Operating System** : MS Windows 10 or 11 / MAC OS / LINUX
- **Python Version** : 3.7 or above
- **Development Environment** : VS Code, Google Colab, Jupyter Notebook
- **Dependencies** : OpenCV, Matplotlib, NumPy, scikit-learn, Pytorch, Tensorflow, Flask, React JS

MODULES & ITS DESCRIPTIONS

MODULES:

1. **Preprocessing:** YOLO training images are augmented with preprocessing techniques such as exposure, hue adjustments, flipping, and skewing. Test images undergo sharpening, binarization, and advanced preprocessing like adaptive thresholding and Gaussian blur.
2. **Vehicle Detector**
3. **Vehicle Tracking**
4. **Automatic Number Plate Recognition**
5. **Signal Timing Algorithm**
6. **Dashboard and Visualisation**

2.VEHICLE DETECTOR

- The **YOLOv8** (You Only Look Once) model is a state-of-the-art deep learning architecture designed for real-time object detection. In this particular application, the YOLO model has been custom trained on a specialized dataset composed of traffic images.

- The training process involves feeding the model a large collection of annotated traffic images, where each vehicle type is labeled accordingly. The labels used here are car, Tri wheel, bus, truck, and bike.The model predicts the vehicle and its class with a bounding box.

- The vehicle detector model attain an **accuracy of 91% on training and 88%** on testing, The model achieves the mAP50 of 88% and mAP50-90 of 72% which consider the Intersection Over Union of the bounding boxes predicted by the vehicle detector model.

- The Dataset used to train the vehicle detector model was custom built dataset which consist of **24000 train images and 2000 test images** which was equally distributed across different class

3.VEHICLE TRACKING

- The DeepSORT (Simple Online and Realtime Tracking with a Deep Association Metric) algorithm is an advanced tracking system used to follow the movement of detected objects across video frames.
- **Detection and Initialization:** After the vehicles are detected using YOLO object detection model in the previous step. Each detected vehicle is represented by a bounding box and is assigned a unique identifier.
- **Centroid Calculation:** For each detected vehicle, DeepSORT calculates the centroid, which is the geometric center of the bounding box. This centroid represents the vehicle's position in the frame.
- **Feature Extraction:** DeepSORT extracts appearance features of each detected vehicle using a deep learning-based feature extractor. This helps in distinguishing vehicles from each other based on their visual appearance, which is crucial when they come close or overlap.
- **Kalman Filter:** To predict and update the position of each vehicle in subsequent frames, DeepSORT uses the Kalman filter. The Kalman filter is a mathematical algorithm that estimates the state of a system over time using a series of measurements.

4.AUTOMATIC NUMBER PLATE RECOGNITION

- License plate is detected using **YOLOv8**, which gives the bounding box coordinates that outline the detected number plate within the image.
- After the localisation of license plates, Optical Character Recognition is done on that localised image to extract text information from the number plate using **EasyOCR**.
- Before applying OCR, the isolated number plate image may undergo preprocessing to enhance its quality. This can include resizing, **grayscale conversion, binarization (converting to black and white), and noise reduction.**
- OCR algorithms analyze the preprocessed image of the number plate to detect and recognize individual characters.
- license plate detector model was trained on a custom dataset which consist of 15000 images for training and 1000 images were used for validation purpose

5.SIGNAL TIMING ALGORITHM

- A dynamic optimization algorithm for signal operation.
- Traffic signal is operated with the help of the density information collected in the previous steps, which helps in determining the appropriate green timing for the current signal based on density.
- The signals are updated efficiently by calculating green timing depending on the current traffic density.
- Because of this real-time response to any situation the traffic management is efficiently achieved.
- ***greenTime = math.ceil(((noOfCars * carTime) + (noOfThreeWheeler * threeWheelerTime) + (noOfBuses * busTime) + (noOfTrucks * truckTime) + (noOftwoWheeler * twoWheelerTime)) / noOfLanes)***

6.DASHBOARD & VISUALISATION

- The dashboard application is a sophisticated web solution designed with a React JS frontend and a Flask backend, incorporating the machine learning models.
- A separate database is also maintained in order to store the generated traffic footage for advanced analytics later.
- The dashboard provides the functionality for live monitoring of every traffic signal in the city. The **Control panel** can modify any signal timing on the current chosen location. Operations like Stop , Resume and update signal, Vehicle tracking , ANPR, Aerial ANPR are available from the control panel.
- Police officials and traffic signal managers can monitor and control the traffic signal and view analysis on traffic data through the analytics page of the control panel.

Implementation Screenshots

PRE-PROCESSING :

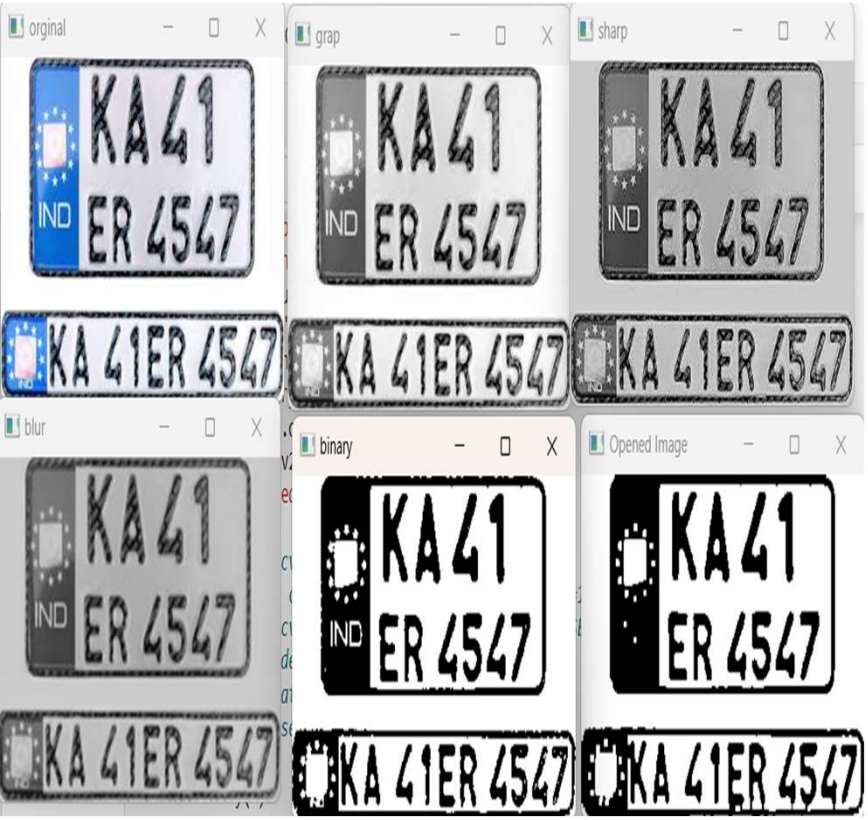

Fig.2. Preprocessing for ANPR

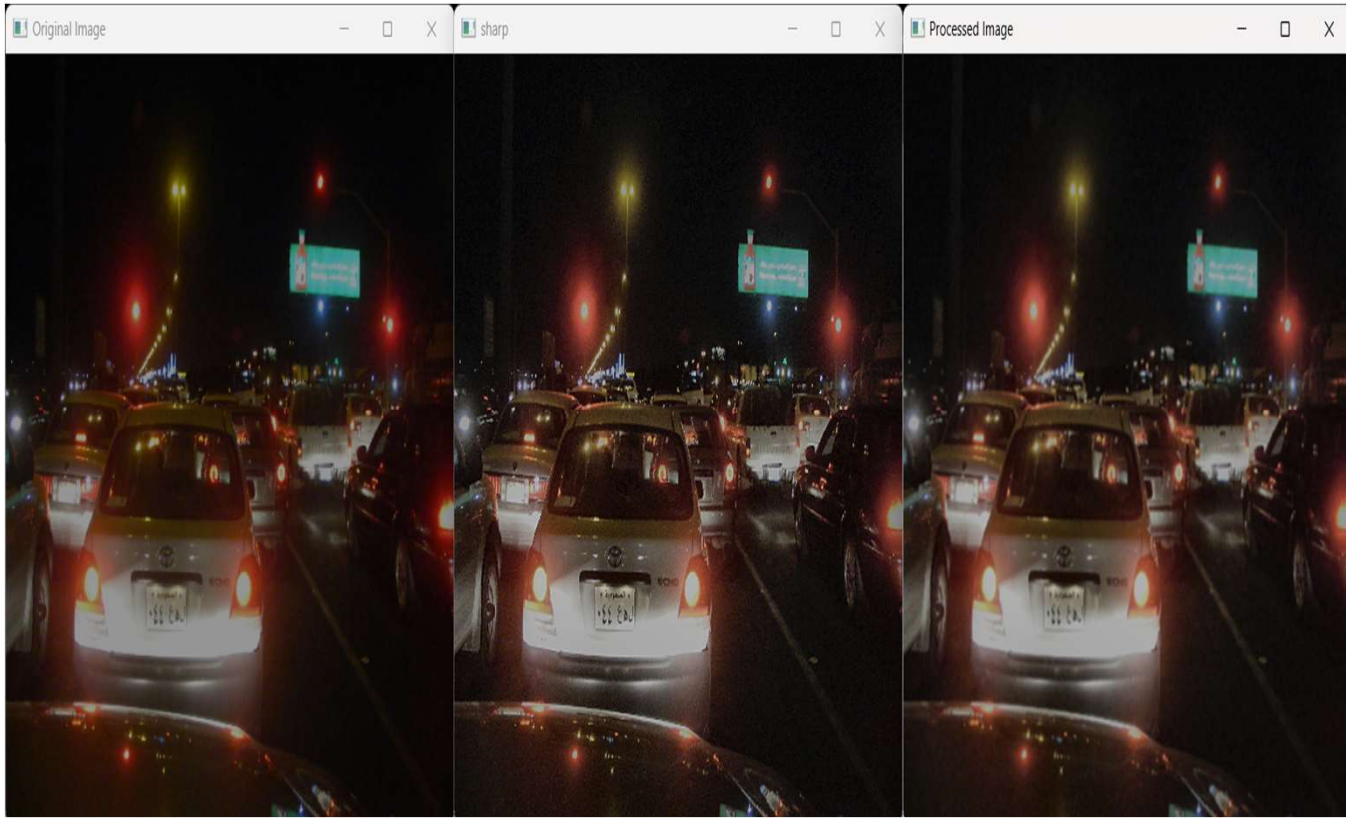

Fig.3. Preprocessing for low lighting and noisy environment

Implementation Screenshots

ORGINAL IMAGE

PRE-PROCESSED IMAGE

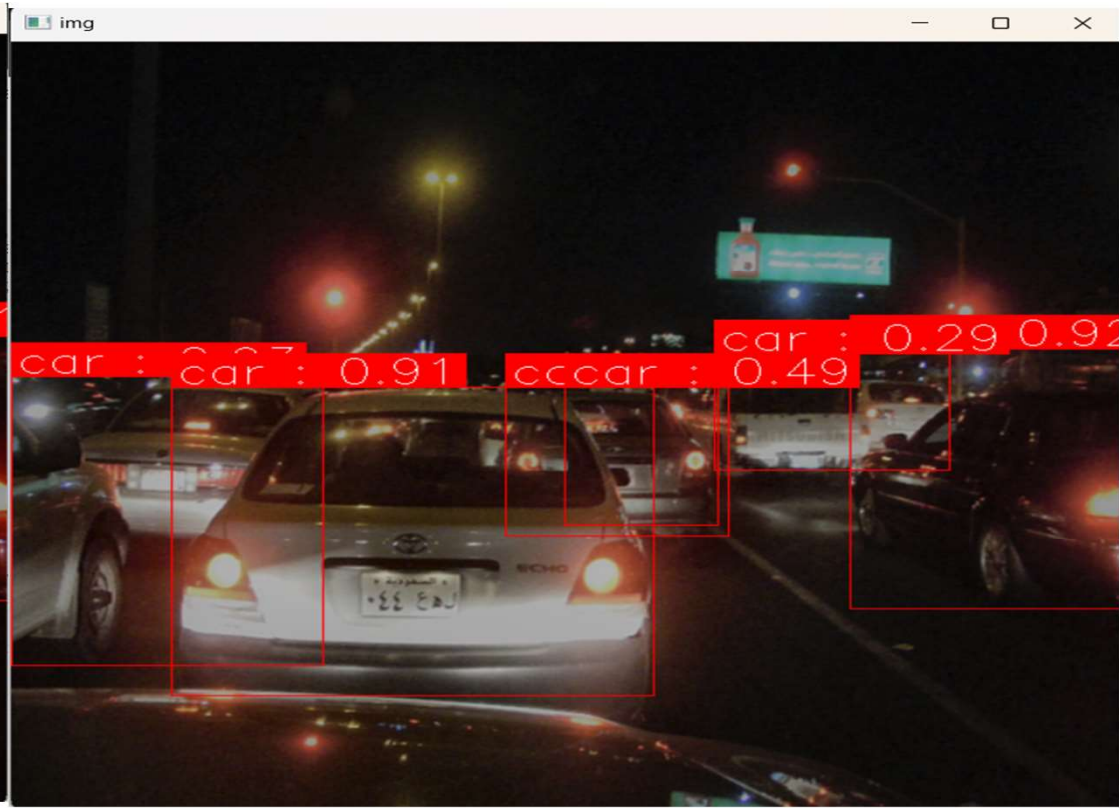

Fig.4. Comparison between detection on original and preprocessed images

Implementation Screenshots

Vehicle Detection

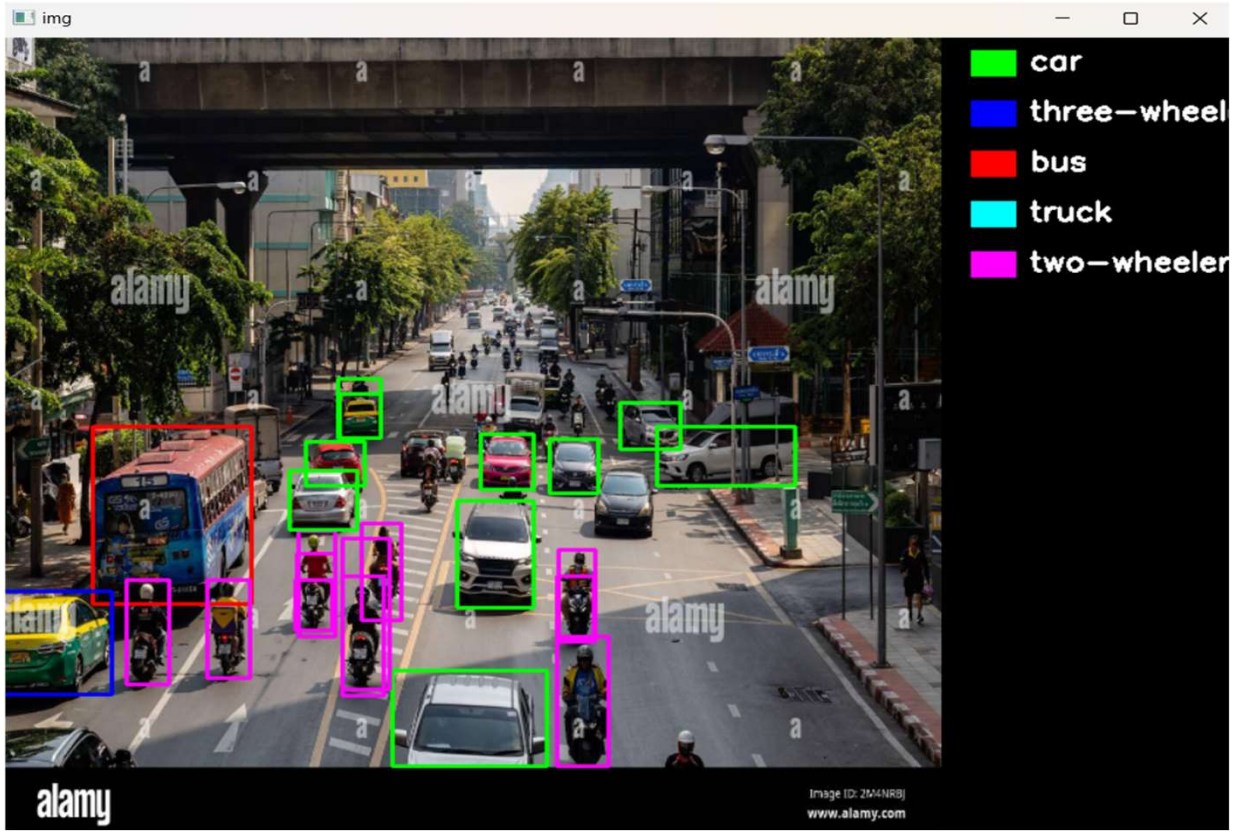

Fig.5.Vehicle detector model detections

ANPR

Fig.6. ANPR Prediction

Implementation Screenshots

DeepSort Tracking:-

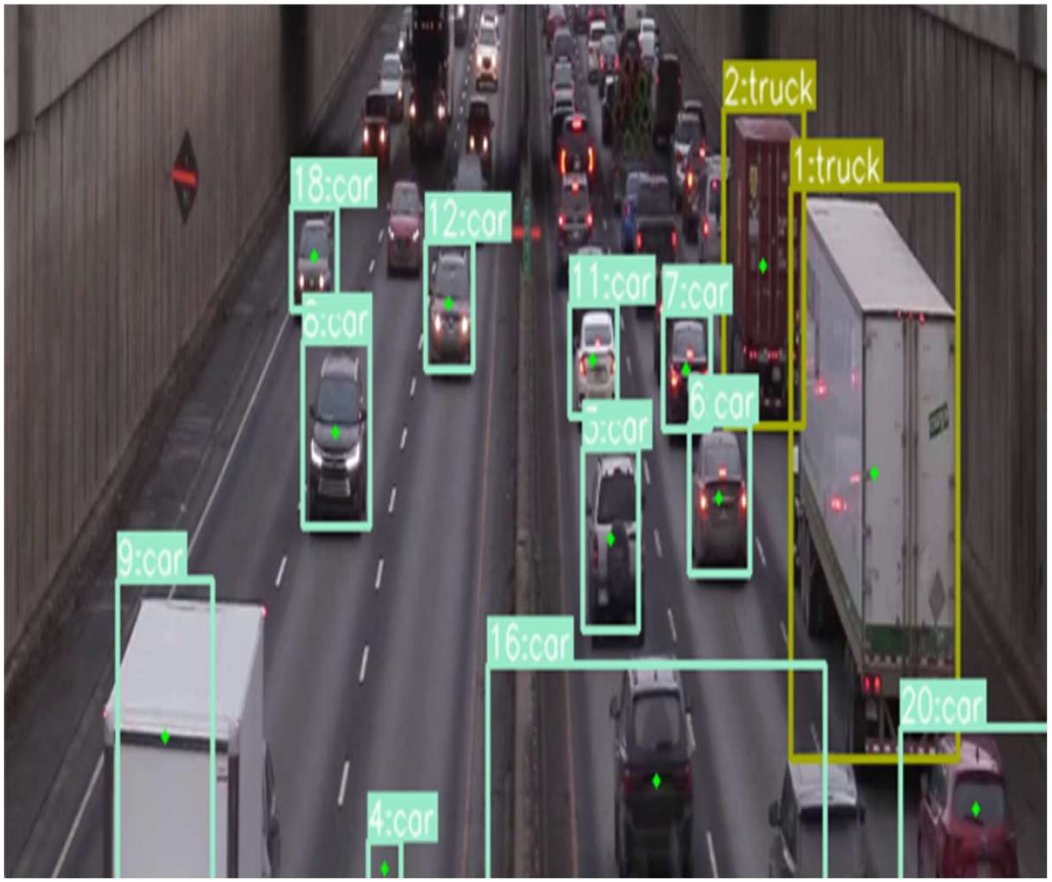

Fig.7. Vehicle tracking using Deep SORT

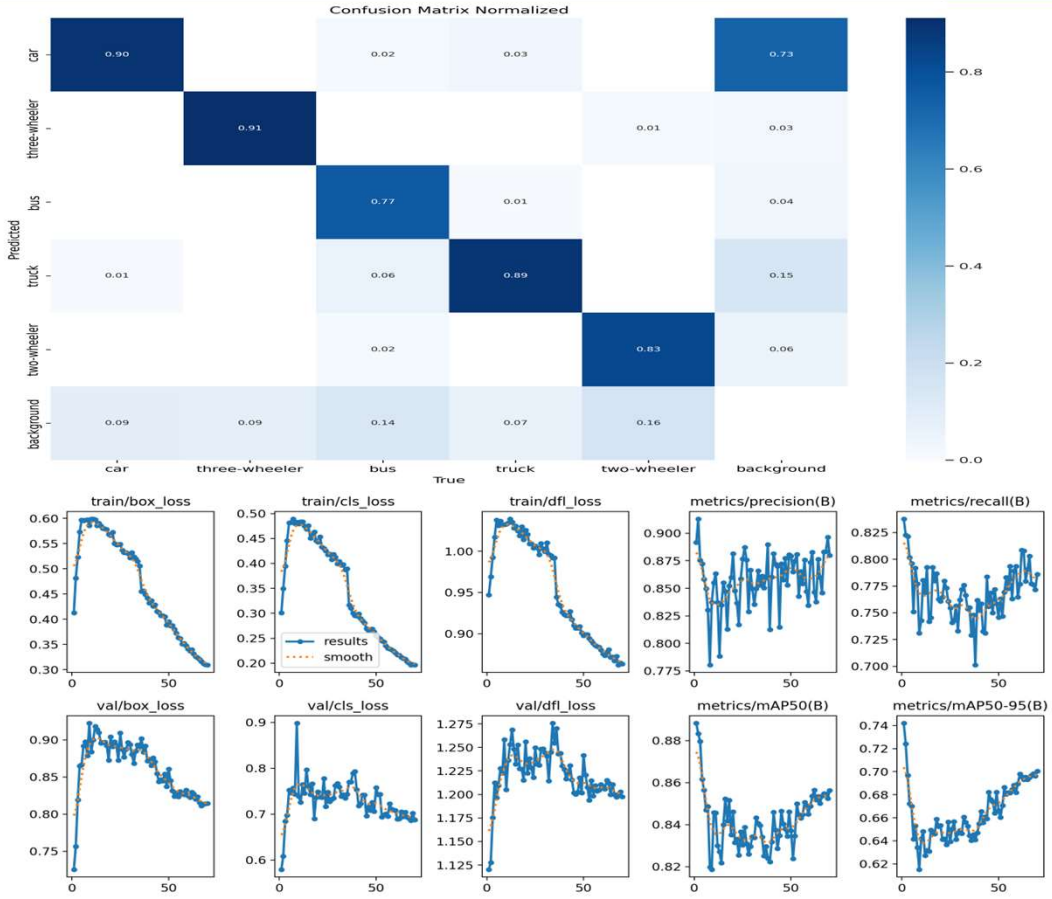

Fig.8.Vehicle detector model metrics

Implementation Screenshots

SignalMaster:-

Fig.9.Signal Master (Remote dashboard)

Implementation Screenshots

Control Panel:-

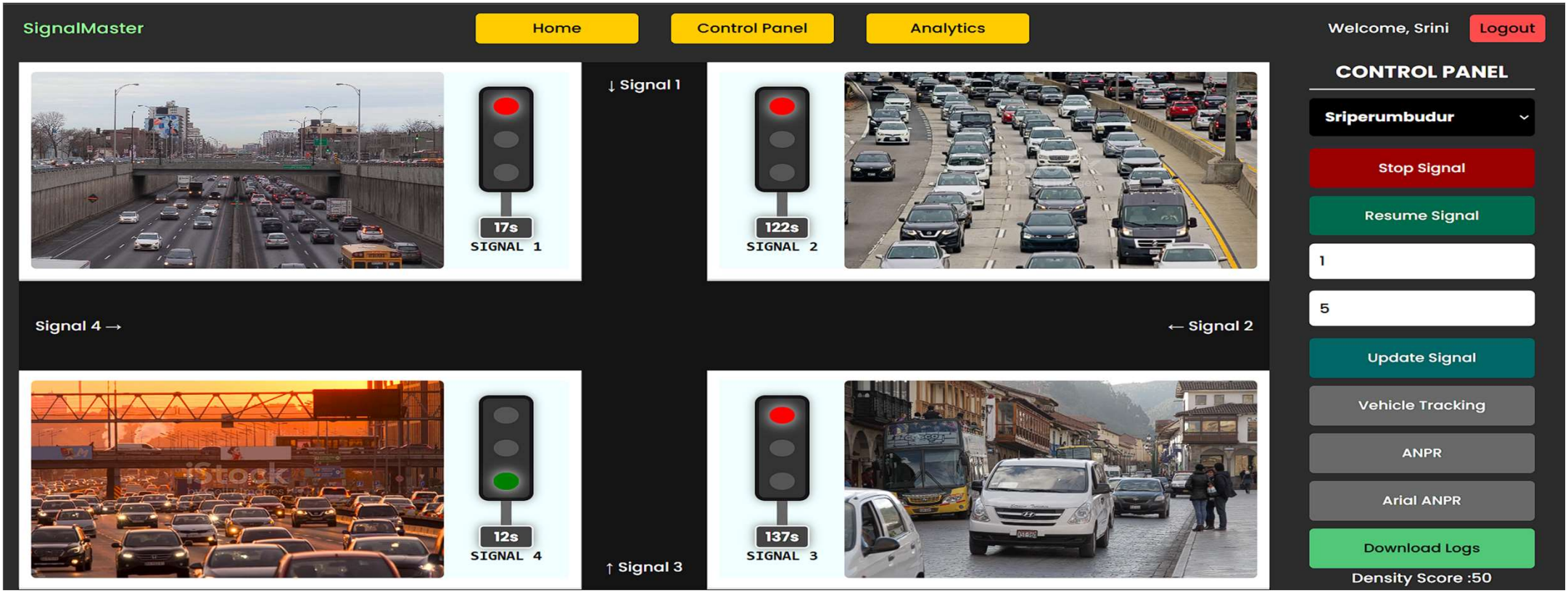

Fig.10.Signal Master(Control Panel)

Implementation Screenshots

Analytics dashboard:-

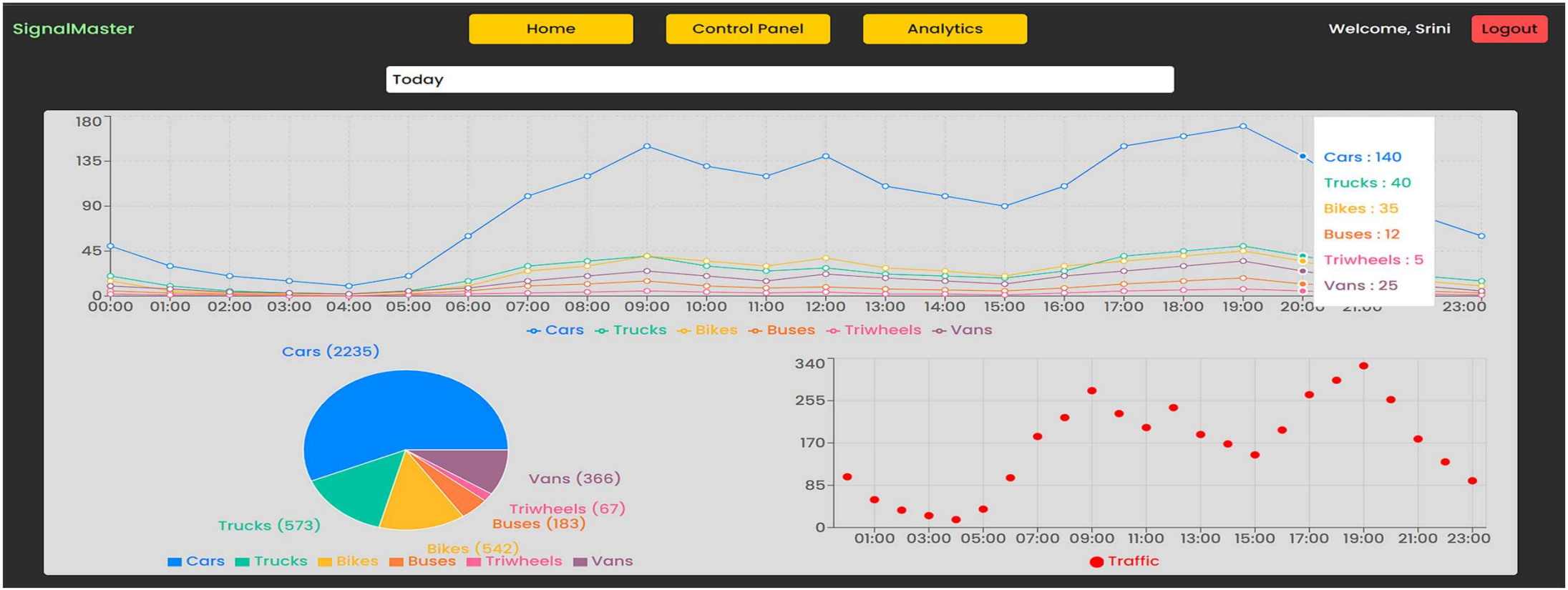

Fig.11.Signal Master (Analytics Page)