Get Fined

Sundarakrishnan N
R.V. College of Engineering
sundarakn.cs22@rvce.edu.in

Sohan Varier
R.V. College of Engineering
sohanvarier.cs22@rvce.edu.in

Tarun Bhupathi
R.V. College of Engineering
tarunbhupathi.cs22@rvce.edu.in

Manaswini Simhadri Kavali
R.V. College of Engineering
manaswinisk.cs22@rvce.edu.in

I. Introduction to Problem Statement

A. Our Understanding

The problem statement aims to develop a vehicle re-identification model for estimating the number of vehicles that reappear in another camera. This has to be done on sequential traffic camera clips, and class-wise. The model's performance in this task would provide insights into the source and destination of vehicles moving across these regions of Bangalore.

II. Methodology

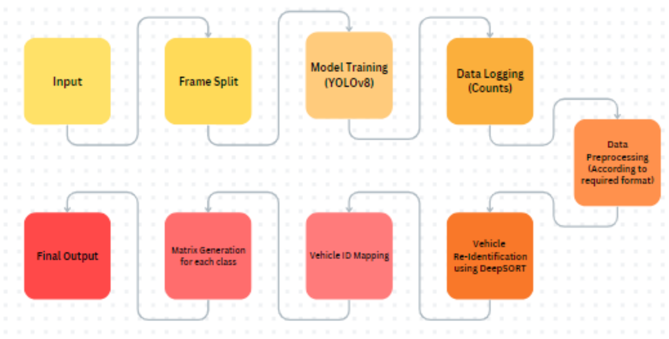

Input
Frame Split
Model Training (YOLOv8)
Data Logging (Counts)
Data Preprocessing (According to required format)
Vehicle Re-Identification using DeepSORT
Vehicle ID Mapping
Matrix Generation for each class
Final Output

Fig. 1: Methodology Flowchart

A. Model Architecture

YOLOv8 achieves state-of-the-art accuracy on various object detection benchmarks, boasts impressive inference speeds suitable for real-time applications such as autonomous vehicles and robotics, is lightweight and efficient requiring fewer computational resources making it ideal for deployment on edge devices, and is open-source with strong community support that fosters continuous development and improvement. Because of these attributes, YOLOv8 has been widely used for vehicle counting, given its effectiveness in real-time scenarios. The YOLOv8 Medium model, in particular, was chosen to balance performance and accuracy, making it an excellent fit for applications that demand both of these. DeepSORT (Simple Online and Realtime Tracking) is a popular object tracking algorithm that extends the original SORT algorithm by incorporating deep learning-based appearance features for robust multi-object tracking. It enhances the basic Kalman filter framework used in SORT by adding an embedder network to generate high-dimensional feature vectors from object detections. These appearance features help distinguish objects that are spatially close or occluded over time, reducing identity switches in the tracking process. The Kalman filter predicts the future position of each tracked object, while the Hungarian algorithm is used to associate new detections with existing tracks based on both motion and appearance information. This combination makes DeepSORT more robust in challenging environments with occlusions, crowded scenes, and objects with similar motion patterns.

III. Approach

This section deals with the detailed approach towards the problem statement.

A. Data Processing

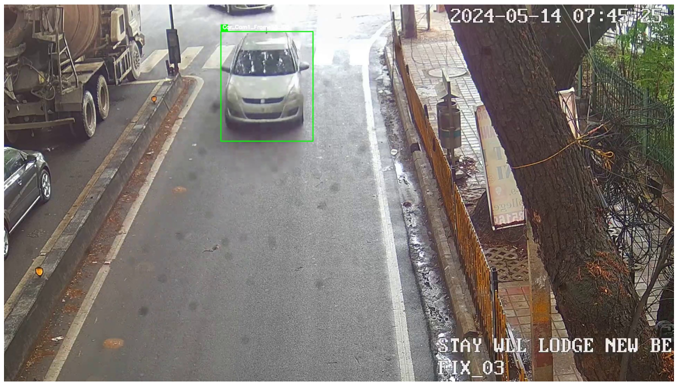
Selected videos were chosen from the huge dataset provided and these videos were split into frames and around 1 frame every two seconds was saved to be annotated. Camera junctions were strategically chosen at different times to ensure variety of data across all cameras and times. The first task was to create a basic YOLOv8 model for seven vehicle classes (Car,Bus,Two-wheeler,Three-wheeler, LCV and Truck). Given that annotation is a laborious and time consuming process, Auto Annotation was used. An initial model was created which was further used to annotate more images. Then it was manually reviewed and errors were fixed. Further 12 such iterations were carried with model improvement at every step and annotation accuracy was greater. Transfer Learning was used to improve accuracy despite a smaller dataset. At the end of all iterations a dataset containing 8000 train images and 800 test and validation images was used. This dataset comprises junctions strategically chosen based on model weaknesses at every iteration, with higher focus on locations where the model needed improvement. Data augmentation was used to increase images even for classes like Bicycle that had severe shortage. Overall the entire pipeline was to create auto annotate with current model,review and add data and retrain the model. Metrics like confusion matrix was analyzed while adding more data. Further metrics and training details are discussed in results.

B. Identification of Vehicles

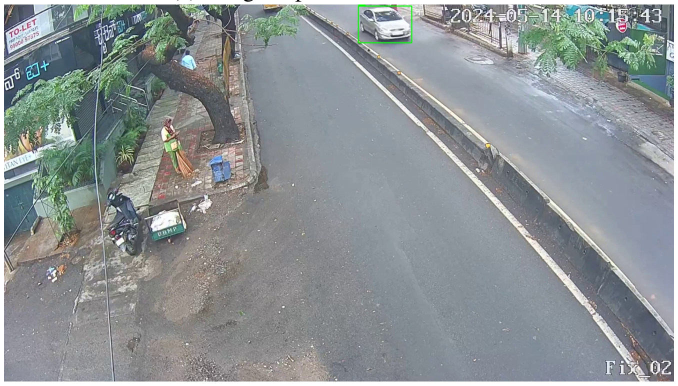
The previously trained YOLOv8 model was used to predict the classes and corresponding bounding boxes of the vehicles in the frame. This data is passed onto a re-identification model, called DeepSort-realtime. DeepSort consists of various pre-trained re-identification models such as torchreid, mobilenet, CLIP, that can be applied on the YOLO predictions. Here, torchreid was used as it gave the best tentative results. DeepSort assigns a unique ID to every new vehicle it comes across, which can be used to check for re-appearances in multiple videos. DeepSort by default does not return the classes of the detected vehicles, so a small change in the source code was made to return the unique ID along with its corresponding vehicle class.

C. Data Processing and Output

A set of unique IDs and their corresponding classes are stored for each video. After processing of all the videos, repeated vehicles that occur in at least two of the videos are noted, and stored in a map format, with vehicle classes as the keys and an NxN matrix as the values. The matrix is of order N, which is the number of videos that is to be tested on. In a row-column pair, the row represents the camera in which the initial detection of a vehicle took place, and the column represents the camera where the same vehicle was re-identified. For example, if 2 videos were passed in, as Cam1, Cam2 and Cam3, any vehicle that is identified in Cam1 and Cam2, will be counted in the (1,2) cell of the matrix. Thus, the diagonal elements of the matrix will be zero throughout, because re-identification in the same camera is not considered.



(a) Image captured in first camera



(b) Image of same car captured in second camera

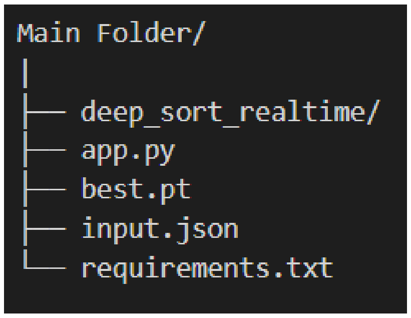
Fig. 2: Re-identification

IV. Explanation

This section covers each file in detail, explaining the work done in the modular code.

A. Directory Structure

The directory structure is organized as follows:



```
Main Folder/
|
├── deep_sort_realtime/
├── app.py
├── best.pt
├── input.json
└── requirements.txt
```

Fig. 3: Directory Structure

B. File Overview and Functionality

C. app.py

1) Loading video paths: The script begins by loading a JSON file that contains paths to the video files. The user specifies both the input JSON file and the output directory via command-line arguments. This ensures the program knows where to find the video data and where to store results.

2) Setting up directories: The script then sets up the necessary directories to store images of detected vehicles, classified into the following vehicle classes:

- Car
- Bus
- Truck
- Three-Wheeler
- Two-Wheeler
- LCV (Light Commercial Vehicle)
- Bicycle

In addition, a separate directory is created to store matrices that track repeated vehicle detections across multiple videos. This organizational step ensures that outputs are saved in appropriate locations.

3) Loading YOLO model: A YOLOv8 model, trained and saved as `best.pt`, is loaded by the script. This model is used for detecting and classifying vehicles in each video frame, assigning each detection to one of the pre-defined vehicle classes.

4) Initializing DeepSort tracker: The script initializes a DeepSort tracker to handle vehicle tracking across frames. The tracker assigns unique IDs to each detected vehicle and updates these IDs as vehicles are detected in consecutive frames. This tracking helps in ensuring that vehicles are not counted multiple times.

5) Processing each video: For each video in the list, the script performs the following steps:

1) Frames from the video are read sequentially.
2) YOLO detects vehicles in each frame, generating bounding boxes and classifying them into vehicle classes.
3) The DeepSort tracker updates the vehicle IDs based on YOLO's detections.
4) The tracker ensures that each vehicle is uniquely tracked across frames. The vehicle IDs and their respective classes are stored in a set, `ids_list`, which is maintained separately for each video.

6) Drawing bounding boxes and saving snapshots: The script draws bounding boxes around each detected vehicle and displays its unique track ID on the video frames. If a vehicle is detected for the first time in a frame, a snapshot is captured and saved in the corresponding vehicle class folder. This helps in visualizing and documenting detected vehicles.

7) Measuring performance: The script measures the time taken to process each frame and calculates the frames per second (FPS) for performance evaluation. These metrics provide insight into the efficiency of the video processing pipeline.

8) Identifying repeated vehicles: After processing all videos, the script compares the tracked vehicle IDs across the videos to identify vehicles that appear in more than one video. The repeated detections are recorded in a `repeats` dictionary, which maps the detected vehicle IDs to the videos in which they were found.

9) Building final matrices: For each vehicle class, the script creates a matrix that tracks the number of times vehicles from that class were detected across different video pairs. These matrices are saved as JSON files in the `Matrices` folder, one matrix for each vehicle class, facilitating easy cross-video comparison.

10) Saving results: The script ensures that all outputs, including matrices and vehicle snapshots, are saved in the specified output directory. The results include the detection matrices for each vehicle class as well as any relevant performance data.

V. RESULTS

The final model was trained for 350 epochs with features like dropout, early stopping and label smoothing to account for bounding box inaccuracies. The YOLOv8 model achieved a precision of 0.909 and recall of 0.875 while classes like cars and buses achieved a precision of 0.95. The overall pipeline achieved an error in recall of 66.9% in the leaderboard.

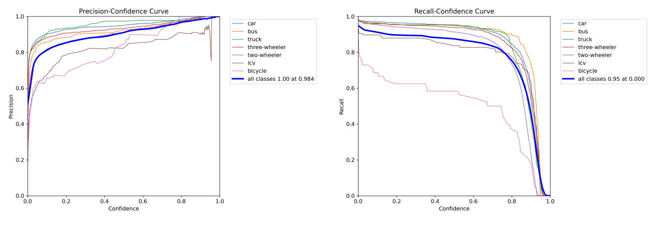


Fig. 4: P curve (left) and R curve (right)

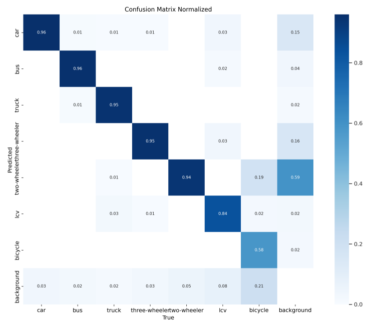


Fig. 5: Confusion matrix

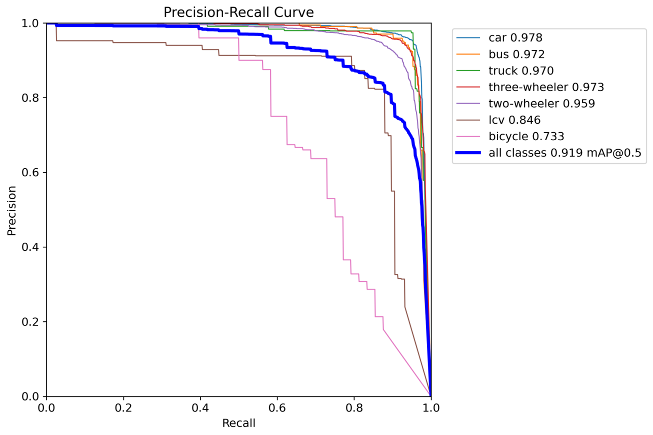


Fig. 6: PR curve

VI. CONCLUSION

In this project, we successfully implemented a robust system for real-time vehicle detection and tracking using YOLOv8 and DeepSORT with vehicle reidentification. The combination of YOLOv8's efficient object detection and DeepSORT's appearance-based tracking, enhanced by the Torchreid embedder, enabled accurate vehicle tracking even in challenging scenarios such as occlusions, abrupt motion changes, and reentrances into the camera's field of view.

The modifications to DeepSORT allowed us to extend the tracking capabilities by capturing appearance features and

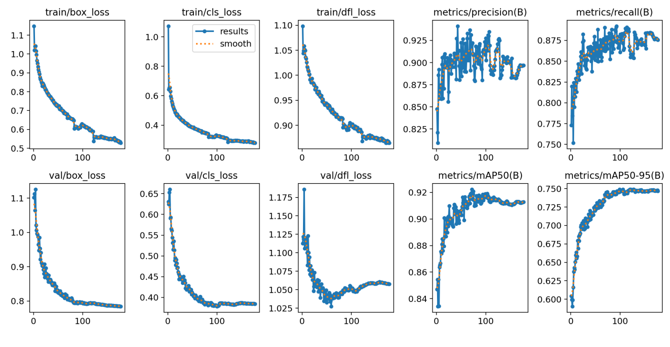


Fig. 7: Results

storing snapshots, ensuring that vehicles could be reidentified after temporarily leaving the scene. By utilizing the cosine similarity metric and setting an appropriate threshold, our system was able to match vehicles based on appearance, providing a more reliable tracking mechanism.

Overall, the integration of vehicle detection, tracking, and reidentification created a comprehensive system capable of handling real-world traffic monitoring challenges. This approach can be further extended to larger traffic networks, providing valuable insights for smart city traffic management and vehicle behavior analysis.

Overall, this system provides a robust framework for real-time traffic monitoring and forecasting, which can be adapted to different environments and expanded with additional features or more sophisticated models. The approach taken in this project serves as a strong foundation for further research and development in traffic analysis and prediction.

A. *Shortcomings of our model*

- False reidentification of similar looking vehicles is a major issue especially because the model doesnt depend on liscence plate.

References

[1] Wojke, Nicolai, Alex Bewley, and Dietrich Paulus. ”Simple online and realtime tracking with a deep association metric.” 2017 IEEE international conference on image processing (ICIP). IEEE, 2017.

[2] @miscSuperbTUM, author = Mingzhe Hu, title = real-time ReID Tracking, year = 2023, publisher = GitHub, journal = GitHub repository, howpublished = https://github.com/SuperbTUM/real-time-ReID-tracking

[3] @articletorchreid, title=Torchreid: A Library for Deep Learning Person Re-Identification in Pytorch, author=Zhou, Kaiyang and Xiang, Tao, journal=arXiv preprint arXiv:1910.10093, year=2019

[4] YOLOv8 - Jocher, G., Chaurasia, A., & Qiu, J. (2023). Ultralytics YOLO (Version 8.0.0) [Computer software]. https://github.com/ultralytics/ultralytics

[5] Tzutalin. LabelImg. Git code (2015). https://github.com/tzutalin/labelImg