Highway Traffic Analysis for Vehicle Detection and Tracking

Objectives This project aims to develop a traffic monitoring system that detects and tracks vehicles using deep learning techniques. The primary objectives include:

- Implementing an object detection model to identify different types of vehicles (cars, trucks, buses, and motorbikes).
- Tracking vehicles using a robust tracking algorithm to count them accurately.
- Analyzing traffic flow by detecting and counting vehicles crossing a predefined limit.

2. Methodology

Data and Model

- The project utilizes the YOLOv8 object detection model for vehicle detection. The model is pre-trained and fine-tuned to optimize accuracy for traffic scenarios.
- A **confidence threshold of 0.3** is used to filter out low-confidence detections. The YOLO model detects objects, and relevant vehicles are filtered based on class labels and confidence scores.
- The video feed is processed frame-by-frame.
- A **mask is applied** to focus on the region of interest , eliminating unnecessary background noise.
- The YOLO model detects objects, filtering relevant vehicles based on class labels and confidence scores.
- The **SORT** algorithm tracks detected objects across frames: After detecting the bounding boxes around the vehicles, the SORT algorithm is used to track the vehicles across different frames. SORT can identify the position of vehicles in the current frame.
- Assigning Unique Identifiers to Vehicles: A unique identifier is assigned to each vehicle when detected by the SORT algorithm. This identifier helps to track the vehicle across successive frames, distinguishing it from other vehicles in the scene.
- Interaction with a Predefined Line: A virtual line (across a specific area in the video) is defined to count vehicles that cross it.

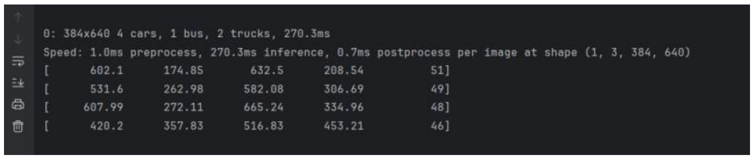
Implementation Tools

Programming Language: Python
Libraries**:** OpenCV, cvzone, Ultralytics YOLO framework, filterpy, scikit-image, and numpy.
Hardware Considerations: The system is tested on a standard PC setup with potential for optimization on embedded devices.

3. Results and Evaluation

The model successfully detected various types of vehicles, including cars, buses, and trucks, as evident from the detected bounding boxes. The results demonstrated that the model accurately identified the correct vehicle categories.

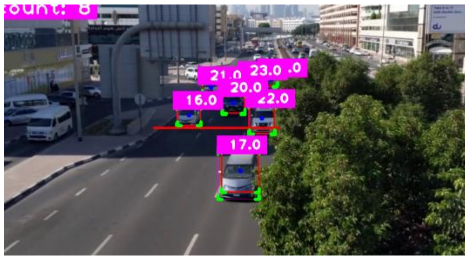
The model showed processing capability, with an average inference time per frame ranging from 270ms to 282ms, along with an additional (0.6ms-0.7ms) post-processing time for filtering results, drawing bounding boxes, and applying additional operations.

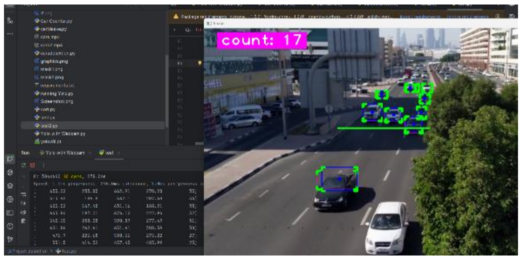


```
0: 384x640 4 cars, 1 bus, 2 trucks, 270.3ms
Speed: 1.0ms preprocess, 270.3ms inference, 0.7ms postprocess per image at shape (1, 3, 384, 640)
[      602.1      174.85       632.5      208.54          51]
[      531.6      262.98      582.08      306.69          49]
[     607.99      272.11      665.24      334.96          48]
[      420.2      357.83      516.83      453.21          46]
```

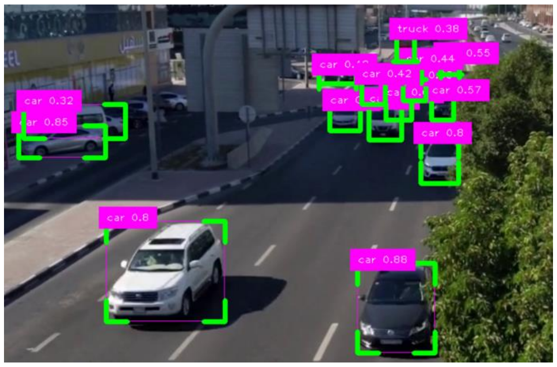
The SORT algorithm was effective in tracking vehicles across frames, assigning unique IDs to each vehicle, allowing seamless tracking throughout the video sequence.

The system successfully counted vehicles based on their crossing of a predefined line, ensuring accurate counting and preventing duplicate detections.





The confidence score for the detected vehicles was good, with some values exceeding 0.80, indicating good model performance. The Intersection over Union (IoU) between the detected boxes and the ground truth boxes was calculated, and results exceeding 0.69 were considered acceptable, demonstrating good object localization accuracy.



4. Challenges and Possible Improvements

Challenges

- Detecting vehicles when partially occluded: The model struggles to recognize vehicles when overlapping.
- Duplicate vehicle counting when crossing the predefined line: Some vehicles are counted multiple times, causing statistical errors. This was resolved by verifying the vehicle's unique ID to prevent duplicate counts.
- Matching mask size with the video: When using cv2.bitwise_, the mask size must match the video size to ensure correct pixel-wise comparison.
- **Possible Improvements:**
- Enhancing performance using GPU instead of CPU: Running the code on a GPU can speed up video processing and reduce inference time.
- Integrating with embedded systems: Deploying the model on devices like Raspberry Pi or NVIDIA Jetson can improve efficiency and lower power consumption.
- Using advanced tracking algorithms: Implementing Deep SORT instead of SORT can improve accuracy and reduce tracking loss.
- Improving model performance in low-light conditions: Training on datasets with poor lighting and occlusions can enhance detection accuracy