COMPUTER VISION EECS-312

PROJECT TITLE: Vehicle Detection and Speed Tracking

Name: Om Govind Jha

Roll No: 22227 Dept: EECS

ABSTRACT: This project presents a computer vision-based system for vehicle detection, tracking, and speed estimation from a video feed. The system utilizes perspective transformation techniques to handle depth perception, ensuring accurate distance estimation and improved vehicle tracking. After the user selects a region of interest (ROI) within the video, a YOLO-based vehicle detection model identifies vehicles, and their movements are tracked across frames using the Byte Tracker. The speed of each detected vehicle is then calculated based on the known distance of the region and the vehicles' positions across consecutive frames. The project aims to extract valuable insights for traffic monitoring, like vehicle type detection, tracking, and speed limits.

DATASET:

The dataset for training the vehicle detection model is in link 1 and the other two are sample videos which is used as input to get the desired results.

We have chosen images from the dataset where there is no Gray scaling applied, and the splitting of data is as follows:

Test: 13 images (~9%) Train-102 images(~71%) Validation-28 images(~20%)

1. https://www.kaggle.com/datasets/pkdarabi/vehicle-detection-image-dataset

2. sample_video_1

3. sample_video_2

Algorithm/Functions Used

1. **YOLO:** You Only Look Once, is a family of state-of-the-art object detection algorithms that are designed for real-time detection with high accuracy and efficiency. It is widely used in computer vision tasks for detecting objects in images and videos. It is a single neural network model that processes the entire image at once. It has a grid-based prediction. Each grid cell is responsible for predicting:
   - Bounding boxes for objects whose center falls within the cell.
   - Confidence scores for these bounding boxes (indicating the likelihood that the box contains an object and the accuracy of its location).
   - Class probabilities for each object category.

   Usage in Code: We have used the YOLOv8 version, and the parameters are given as:

   (epochs=50, batch=16, imgsz=1280)

   - Epoch: An epoch refers to one complete pass through the entire training dataset by the model. So, here 50 such passes are happening.
   - Batch Size: Number of images processed together in a single training step.
   - Image Size: The resolution to which images are resized before being fed into the model. Larger images provide better accuracy but increase computation time.

2. **Perspective Transformation:** A geometric operation that maps points in a 2D plane from one perspective (source plane) to another (target plane). This is particularly useful in computer vision tasks where we want to "flatten" a view or adjust the perspective of an image to align with a desired orientation.

   Usage in Code:

   - Four corner points of the region of interest (ROI) in the original perspective are given by the “SOURCE” matrix, and the desired corresponding four corners are given by the “TARGET” matrix.
   - OpenCV's “cv2.getPerspectiveTransform” function is used to compute a transformation matrix.A 3x3 perspective transformation matrix (transformation matrix) that maps points from the source plane to the target plane is obtained.
   - A “transform_points” function is defined, which applies the transformation matrix on a given set of points from the source plane to get the corresponding points on the target plane.
   - So, for each of the four points in the source plane, we get two equations to solve. So, in total we have eight equations and the transformation matrix also has eight variables (as for the 3,3 element, we take 1 for reference and ensure that the scaling can be done properly)

3. **Byte Track:** It is an object-tracking algorithm that effectively combines detection results from an object detector (e.g., YOLO) and matches these results to maintain track identities across frames.
   Working and usage in code

- At every frame of the video, YOLO provides the detected bounding boxes, confidence scores, and class IDs (e.g., "Car" or "Truck"). Byte Track maintains a list of active tracks; each track corresponds to a single detected object, storing information like the bounding box location in the previous frame(s), the unique ID assigned to the object (Tracker ID), and information like its trajectory and state ("visible" or "occluded").
- track_activation_threshold=CONFIDENCE_THRESHOLD: Sets a confidence threshold for detections to start a new track or update an existing one
- For each new frame it calculates a "similarity score" between current detections and existing tracks using Bounding box overlap by comparing the positions of bounding boxes from the current frame with those from the previous frame.
- If a detection cannot be matched to any existing track, ByteTrack starts a new track with a unique ID.
- If an active track doesn’t receive any matching detections for a certain number of frames, ByteTrack terminates it.
- YOLO's detections are refined before passing them to Byte Track:
  1. Confidence Filter: Keeps detections above the confidence threshold(0.3)
  2. Zone Filter: Keeps only detections within the region of interest (polygon zone used using the source matrix)
  3. Non-Max Suppression (NMS): Removes overlapping bounding boxes, keeping only the most confident one(using IOU threshold to 0.5)

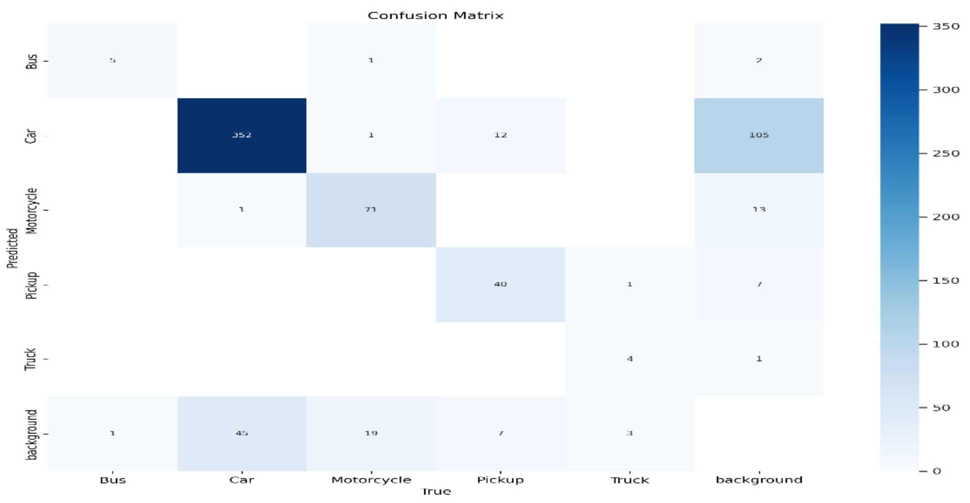
4. **Tracking and Speed Calculation:**
   points =
   detections.get_anchors_coordinates(anchor=sv.Position.BOTTOM_CENTER)

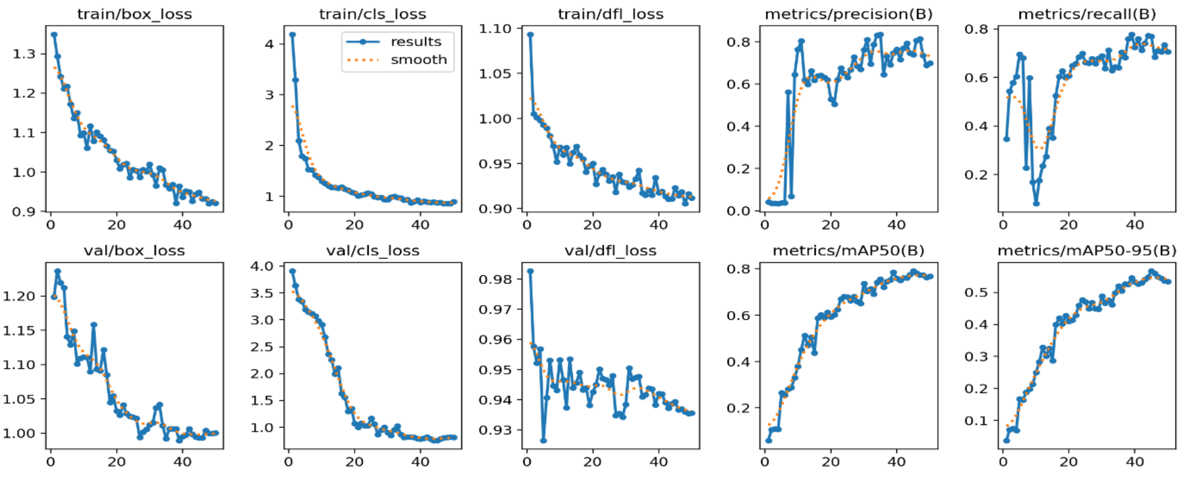
The anchor points(bottom-center of bounding boxes taken here) are selected and transformed to the target plane for speed calculation. The y-coordinates of the object's position (anchor point) in the transformed space are stored frame-by-frame. The speed is calculated based on how far the object has moved in recent frames. An overspeed bar is set to > 110 km/hr.

5. **Some Libraries used:**
    1. **OpenCV**: for image and video processing (perspective transformations, frame manipulation).
    2. **Supervision**: Video processing, annotations, and object tracking.
    3. **Collections**: Data structures for managing object positions and states. Functions used:
        - **defaultdict** : used to store object states (coordinates per tracker) in a dictionary-like structure, with a default value of an empty deque (queue).
        - **deque** : a double-ended queue that stores the coordinates of detected objects over time, allowing easy addition and removal of positions while keeping track of their movement.
    4. **TQDM**: Progress bar for loops.

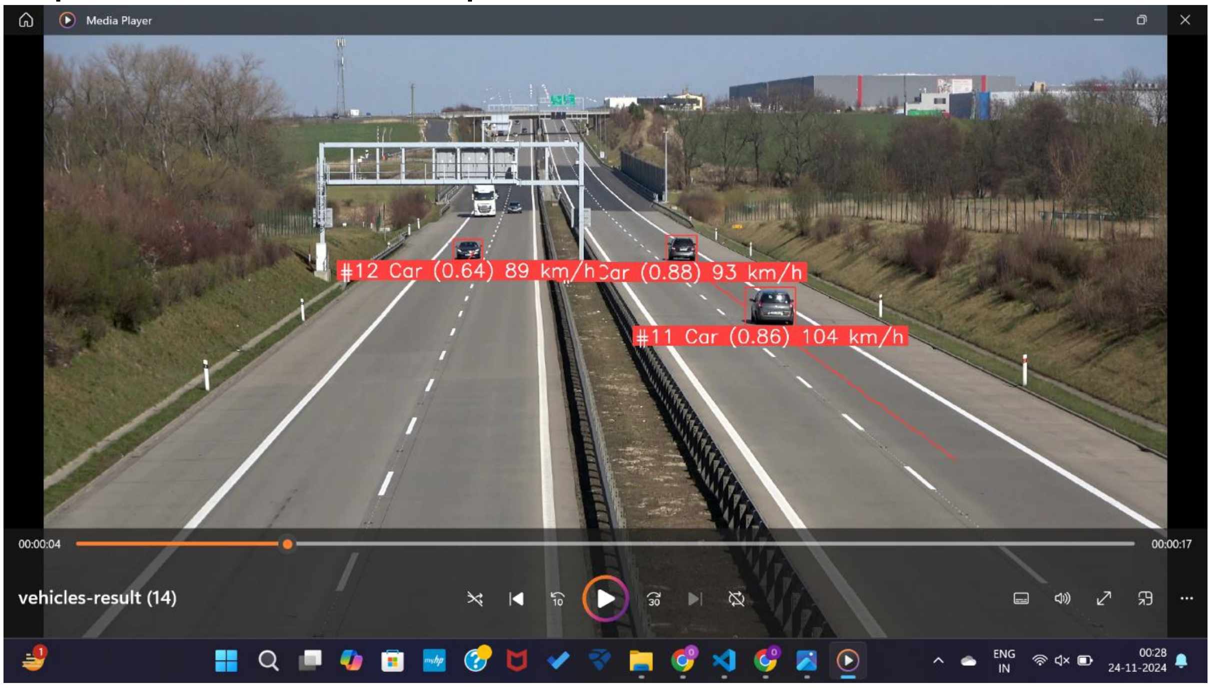
RESULTS

1. YOLO MODEL RESULTS:

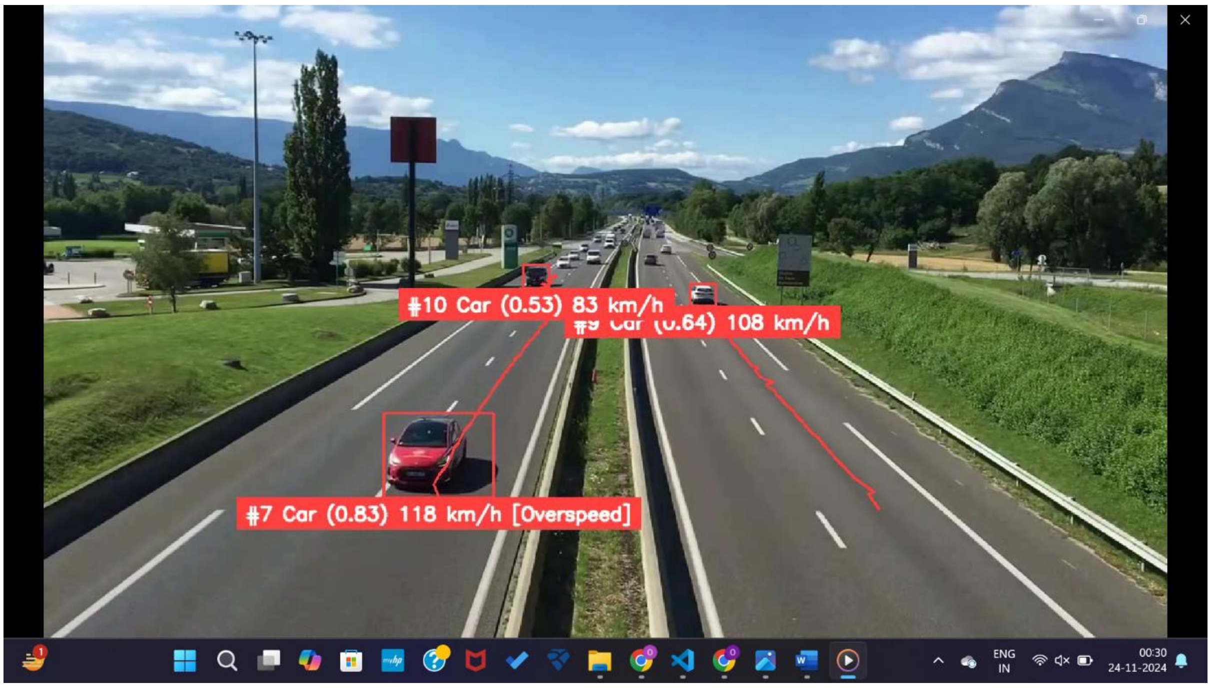




2. **Output Video screenshot for sample video 1**



3. **Output Video screenshot for sample video 2**


#10 Car (0.53) 83 km/h
#9 Car (0.64) 108 km/h
#7 Car (0.83) 118 km/h [Overspeed]
ENG IN
00:30
24-11-2024