Vehicle Classification

**Introduction**:

This code is designed to classify and track vehicles in a video using the **YOLOv8 (You Only Look Once)** model for object detection and a custom tracking algorithm. It focuses on identifying vehicles, counting them as they cross predefined lines, and determining the direction of movement (either "up" or "down").

Traffic management has become increasingly important as urbanization and the number of vehicles on the road continue to rise. Congestion, accidents, and traffic violations are growing concerns that necessitate advanced solutions for monitoring and controlling traffic flow. Traditional traffic surveillance systems, such as manual vehicle counting or fixed sensors, have several limitations, including low accuracy, high labor costs, and inability to operate efficiently in real time. With the advent of deep learning and computer vision technologies, there has been a significant leap forward in the automation of traffic management systems.

This project focuses on developing an automated **vehicle detection, tracking, and counting system** using the **YOLOv8** (You Only Look Once) model, one of the most advanced and efficient object detection algorithms. YOLOv8 is renowned for its real-time performance and high accuracy, making it an ideal choice for traffic monitoring applications. In this project, YOLOv8 detects vehicles like cars, trucks, and buses from video footage, assigns bounding boxes to these vehicles, and tracks their movement across frames.

The primary objective of this project is to develop an **accurate, efficient, and real-time solution** for vehicle detection, tracking, and counting. The project aims to be lightweight enough to operate in real-time scenarios, making it suitable for deployment in smart traffic management systems, where timely data is crucial for effective decision-making. By providing real-time insights into traffic flow, this system could significantly enhance traffic control, congestion management, and urban planning efforts.

**Objective of the Code:**

The code aims to detect and classify vehicles in a traffic video using the YOLOv8 object detection model. It tracks each vehicle's movement across frames by assigning unique IDs to them. The primary objective is to monitor the direction of vehicle movement, whether upward or downward, by analyzing when vehicles cross predefined lines. It provides real-time visual feedback by drawing lines, circles, and IDs on the video frames. The code also counts vehicles crossing the lines to offer insights into traffic flow. It aims to assist traffic monitoring applications like traffic density estimation and congestion analysis. The ultimate goal is to gather useful data for traffic management and road planning purposes.

### Drawbacks of the Code:

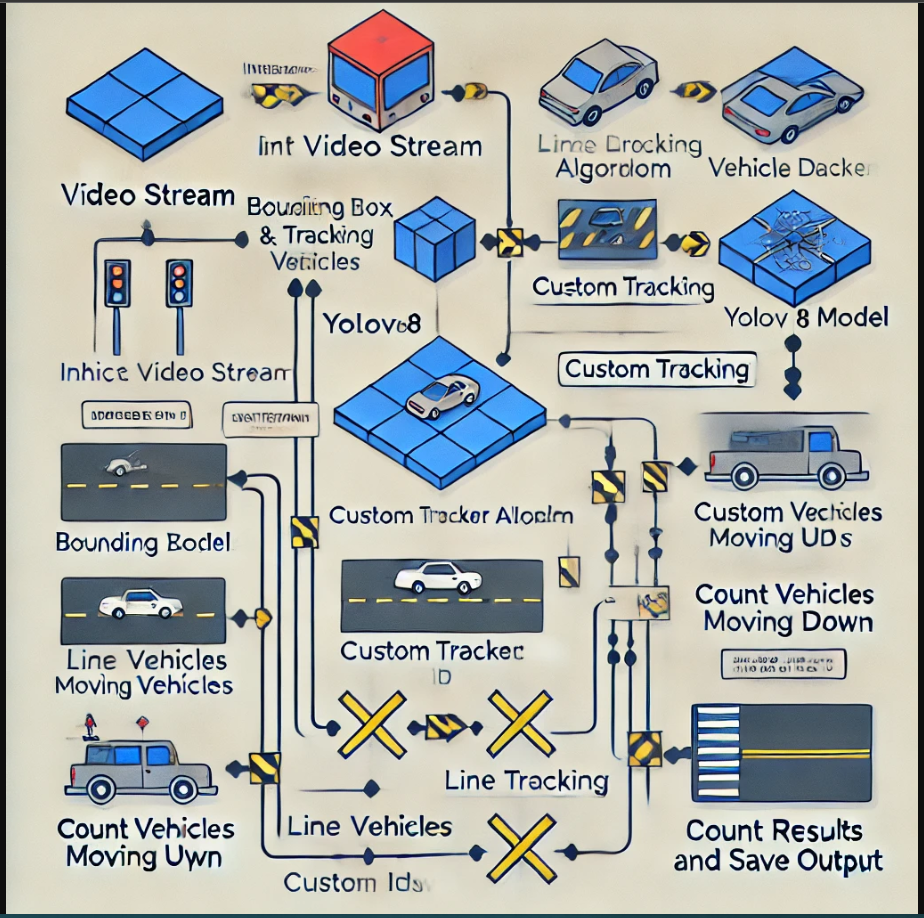
The tracking system does not re-identify vehicles if they temporarily leave the frame, which can lead to double counting. The simplistic distance-based tracking can fail in situations where vehicles overlap or are close to each other. It mainly focuses on car detection, limiting its usefulness for analyzing multiple vehicle types. Complex scenarios like vehicle occlusion or lighting changes are not well-handled, reducing accuracy in real-world applications. The use of static lines for vehicle counting doesn't adapt to varying road conditions or camera angles. The code relies on hardcoded parameters for line placement and threshold values, making it less flexible for different traffic videos. Finally, it lacks optimization for handling large-scale traffic footage and may suffer from slow performance on high-resolution videos.

**Literature Survey for Vehicle Classification and Tracking Code:**

|  |  |  |  |
| --- | --- | --- | --- |
| Author(s) | Methodology/Approach | Findings/Results | Limitations |
| Redmon et al. (2016) | Proposed **YOLO** (You Only Look Once), a real-time object detection system that processes images in a single pass. | Demonstrated faster detection compared to traditional models with good accuracy on multiple objects, including vehicles. | Struggles with detecting small objects and overlapping objects in complex scenes. |
| Bewley et al. (2016) | Introduced the **Simple Online and Realtime Tracking (SORT)** algorithm, using Kalman filters and the Hungarian algorithm for tracking objects in real-time. | Achieved high-speed object tracking in real-time applications with low computational requirements. | Ineffective in cases of heavy occlusion and object re-identification challenges. |
| Bochkovskiy et al. (2020) | Presented **YOLOv4**, an enhanced version of YOLO with improved detection accuracy and speed for object detection tasks, particularly in dense environments. | Improved detection for smaller objects and complex scenes with higher accuracy and robustness. | Requires extensive computational resources for real-time high-resolution videos. |
| Chen et al. (2020) | Proposed a vehicle tracking system combining **YOLO with DeepSORT** for enhanced object detection and tracking in real-world traffic scenarios. | Achieved better accuracy in detecting and tracking vehicles even under occlusion conditions. | DeepSORT increases complexity and may slow down real-time performance in large-scale setups. |

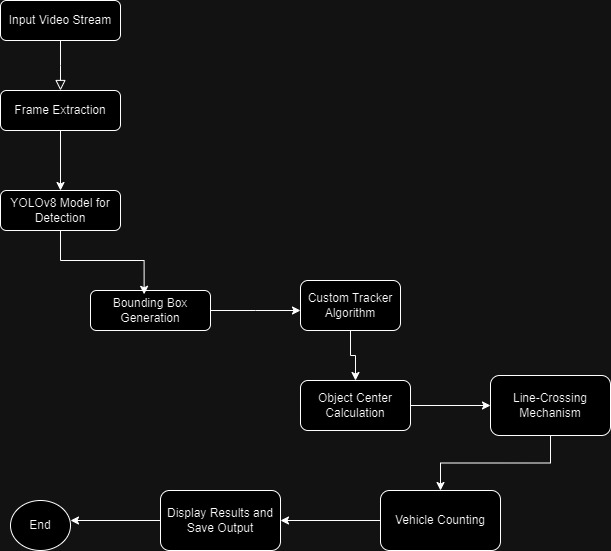
This survey highlights the evolution of object detection and tracking methods, focusing on real-time applications in vehicle detection and classification. While YOLO-based systems provide speed and accuracy, tracking algorithms like SORT and DeepSORT address the challenges of tracking multiple objects but introduce trade-offs in computational complexity.

**System Architecture Diagram:**



* The architecture consists of three key modules: YOLOv8 for object detection, the tracking system for object ID assignment, and the counting logic for line-crossing analysis.

**Flowchart:**



**Explanation:**

* The system leverages YOLOv8 for real-time vehicle detection and a custom tracking algorithm to follow vehicles throughout the video sequence. Vehicles are identified by their bounding boxes and tracked by monitoring their center points, ensuring efficient vehicle ID assignment. Two horizontal lines, a red line (top) and a blue line (bottom), are drawn across the frame. The system counts vehicles as they cross these lines, allowing it to differentiate between upward and downward traffic.

**Implementation (Complete Code):**

**pip install ultralytics**

**import ultralytics**

**ultralytics.\_\_version\_\_**

**pip install tracker**

**import cv2**

**import pandas as pd**

**from ultralytics import YOLO**

**from tracker import\***

**model=YOLO('yolov8s.pt')**

**!apt-get install**

**class\_list = ['person', 'bicycle', 'car', 'motorcycle', 'airplane', 'bus', 'train', 'truck', 'boat', 'traffic light', 'fire hydrant', 'stop sign', 'parking meter', 'bench', 'bird', 'cat', 'dog', 'horse', 'sheep', 'cow', 'elephant', 'bear', 'zebra', 'giraffe', 'backpack', 'umbrella', 'handbag', 'tie', 'suitcase', 'frisbee', 'skis', 'snowboard', 'sports ball', 'kite', 'baseball bat', 'baseball glove', 'skateboard', 'surfboard', 'tennis racket', 'bottle', 'wine glass', 'cup', 'fork', 'knife', 'spoon', 'bowl', 'banana', 'apple', 'sandwich', 'orange', 'broccoli', 'carrot', 'hot dog', 'pizza', 'donut', 'cake', 'chair', 'couch', 'potted plant', 'bed', 'dining table', 'toilet', 'tv', 'laptop', 'mouse', 'remote', 'keyboard', 'cell phone', 'microwave', 'oven', 'toaster', 'sink', 'refrigerator', 'book', 'clock', 'vase', 'scissors', 'teddy bear', 'hair drier', 'toothbrush']**

**import math**

**class Tracker:**

**def \_\_init\_\_(self):**

**# Store the center positions of the objects**

**self.center\_points = {}**

**# Keep the count of the IDs**

**# each time a new object id detected, the count will increase by one**

**self.id\_count = 0**

**def update(self, objects\_rect):**

**# Objects boxes and ids**

**objects\_bbs\_ids = []**

**# Get center point of new object**

**for rect in objects\_rect:**

**x, y, w, h = rect**

**cx = (x + x + w) // 2**

**cy = (y + y + h) // 2**

**# Find out if that object was detected already**

**same\_object\_detected = False**

**for id, pt in self.center\_points.items():**

**dist = math.hypot(cx - pt[0], cy - pt[1])**

**if dist < 35:**

**self.center\_points[id] = (cx, cy)**

**# print(self.center\_points)**

**objects\_bbs\_ids.append([x, y, w, h, id])**

**same\_object\_detected = True**

**break**

**# New object is detected we assign the ID to that object**

**if same\_object\_detected is False:**

**self.center\_points[self.id\_count] = (cx, cy)**

**objects\_bbs\_ids.append([x, y, w, h, self.id\_count])**

**self.id\_count += 1**

**# Clean the dictionary by center points to remove IDS not used anymore**

**new\_center\_points = {}**

**for obj\_bb\_id in objects\_bbs\_ids:**

**\_, \_, \_, \_, object\_id = obj\_bb\_id**

**center = self.center\_points[object\_id]**

**new\_center\_points[object\_id] = center**

**# Update dictionary with IDs not used removed**

**self.center\_points = new\_center\_points.copy()**

**return objects\_bbs\_ids**

**tracker=Tracker()**

**count=0**

**cap=cv2.VideoCapture('/content/4K Video of Highway Traffic!.mp4')**

**down={}**

**up={}**

**counter\_down=[]**

**counter\_up=[]**

**from google.colab.patches import cv2\_imshow**

**while True:**

**ret,frame = cap.read()**

**if not ret:**

**break**

**count += 1**

**frame=cv2.resize(frame,(1020,500))**

**results=model.predict(frame)**

**# print(results)**

**a=results[0].boxes.data**

**a = a.detach().cpu().numpy() # added this line**

**px=pd.DataFrame(a).astype("float")**

**#print(px)**

**list=[]**

**for index,row in px.iterrows():**

**#print(row)**

**x1=int(row[0])**

**y1=int(row[1])**

**x2=int(row[2])**

**y2=int(row[3])**

**d=int(row[5])**

**c=class\_list[d]**

**if 'car' in c:**

**list.append([x1,y1,x2,y2])**

**#print(c)**

**bbox\_id=tracker.update(list)**

**#print(bbox\_id)**

**for bbox in bbox\_id:**

**x3,y3,x4,y4,id=bbox**

**cx=int(x3+x4)//2**

**cy=int(y3+y4)//2**

**red\_line\_y=198**

**blue\_line\_y=268**

**offset = 7**

**''' both lines combined condition . First condition is for red line'''**

**## condition for counting the cars which are entering from red line and exiting from blue line**

**if red\_line\_y < (cy + offset) and red\_line\_y > (cy - offset):**

**down[id]=cy**

**if id in down:**

**if blue\_line\_y < (cy + offset) and blue\_line\_y > (cy - offset):**

**cv2.circle(frame,(cx,cy),4,(0,0,255),-1)**

**cv2.putText(frame,str(id),(cx,cy),cv2.FONT\_HERSHEY\_COMPLEX,0.8,(0,255,255),2)**

**#counter+=1**

**counter\_down.append(id) # get a list of the cars and buses which are entering the line red and exiting the line blue**

**# condition for cars entering from blue line**

**if blue\_line\_y < (cy + offset) and blue\_line\_y > (cy - offset):**

**up[id]=cy**

**if id in up:**

**if red\_line\_y < (cy + offset) and red\_line\_y > (cy - offset):**

**cv2.circle(frame,(cx,cy),4,(0,0,255),-1)**

**cv2.putText(frame,str(id),(cx,cy),cv2.FONT\_HERSHEY\_COMPLEX,0.8,(0,255,255),2)**

**#counter+=1**

**counter\_up.append(id) # get a list of the cars which are entering the line 1 and exiting the line 2**

**text\_color = (255,255,255) # white color for text**

**red\_color = (0, 0, 255) # (B, G, R)**

**blue\_color = (255, 0, 0) # (B, G, R)**

**green\_color = (0, 255, 0) # (B, G, R)**

**cv2.line(frame,(172,198),(774,198),red\_color,3) # starting cordinates and end of line cordinates**

**cv2.putText(frame,('red line'),(172,198),cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, text\_color, 1, cv2.LINE\_AA)**

**cv2.line(frame,(8,268),(927,268),blue\_color,3) # seconde line**

**cv2.putText(frame,('blue line'),(8,268),cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, text\_color, 1, cv2.LINE\_AA)**

**downwards = (len(counter\_down))**

**cv2.putText(frame,('going down - ')+ str(downwards),(60,40),cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, green\_color, 1, cv2.LINE\_AA)**

**upwards = (len(counter\_up))**

**cv2.putText(frame,('going up - ')+ str(upwards),(60,60),cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, text\_color, 1, cv2.LINE\_AA)**

**cv2\_imshow(frame)**

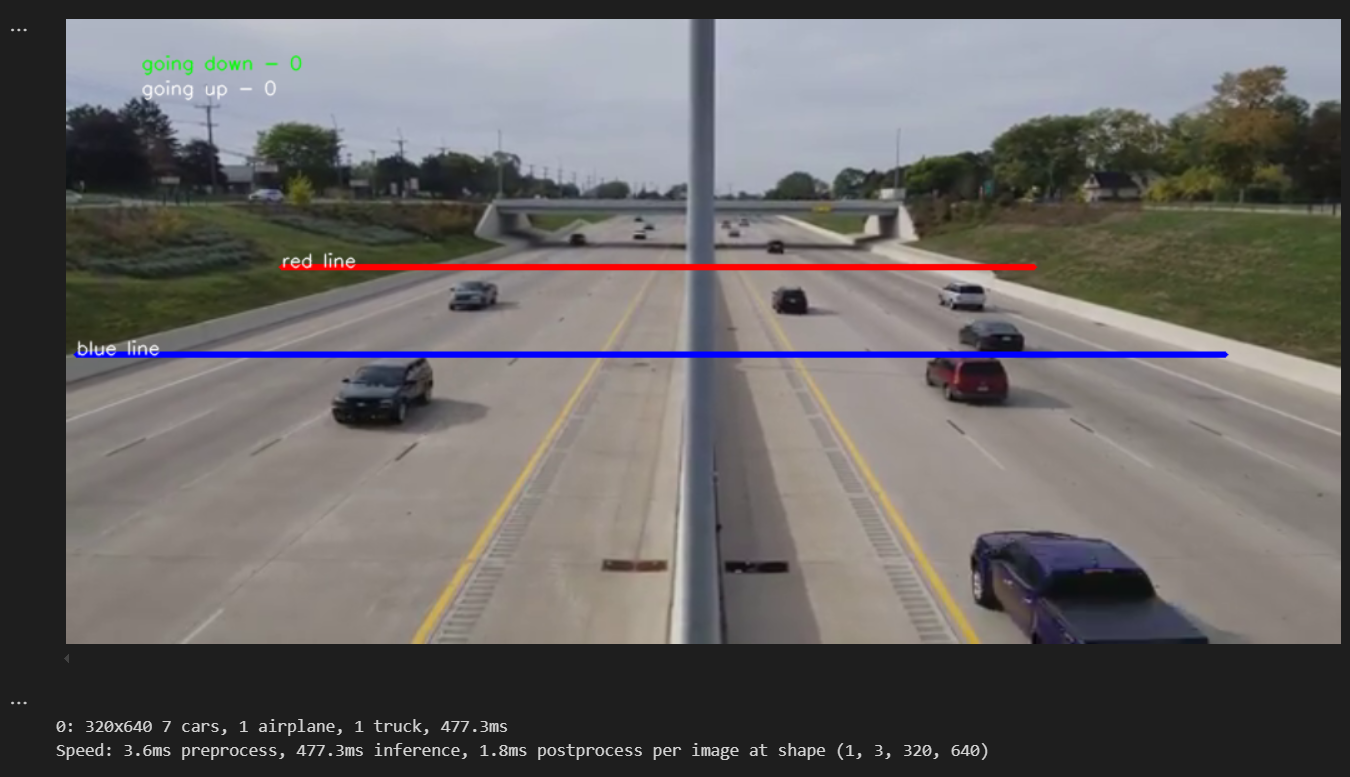
**if cv2.waitKey(1)&0xFF==27:**

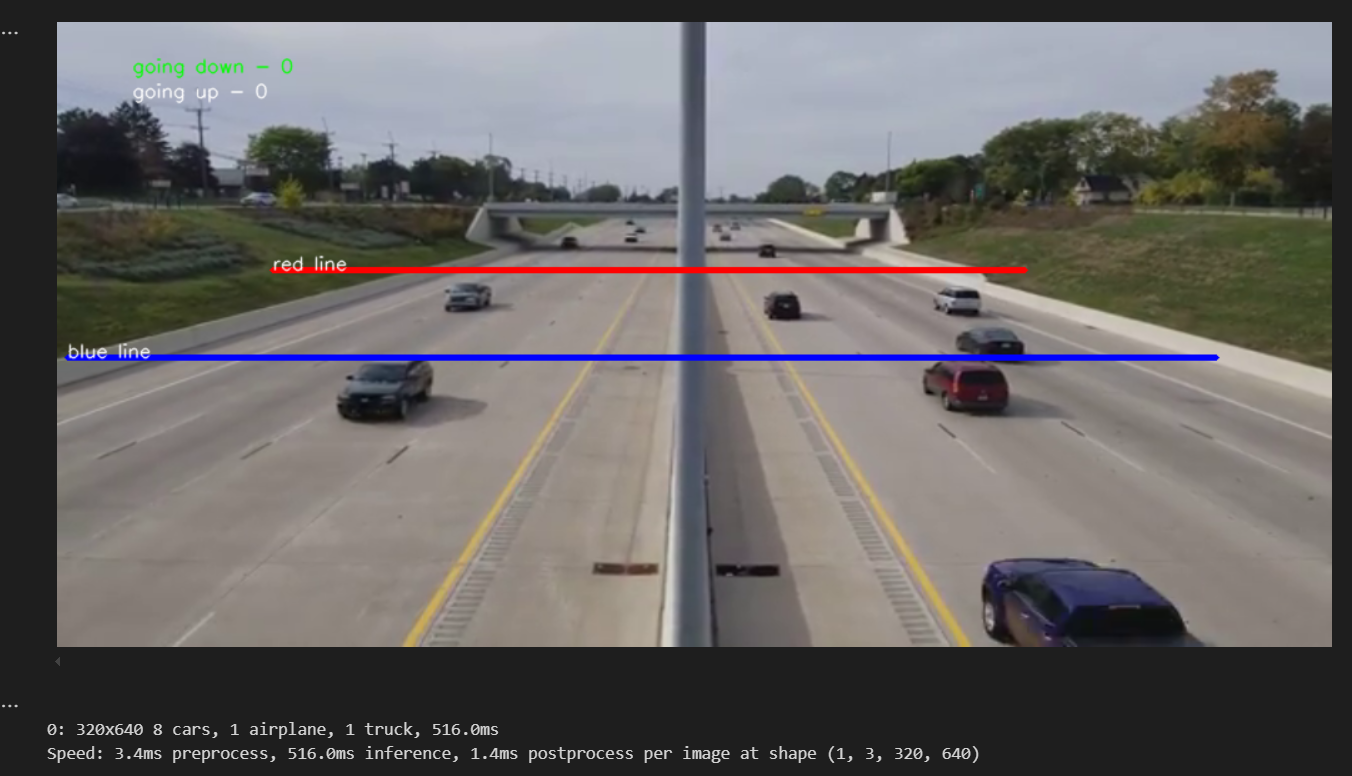
**break**

**cap.release()**

**cv2.destroyAllWindows()**

**Results and Discussion:**

****

****

This image is the output of a vehicle detection and tracking system using the YOLOv8 model, applied to a highway scene. Two key lines, a red line and a blue line, are drawn across the road to track vehicles moving in different directions. The red line, closer to the top of the image, monitors vehicles moving downward, while the blue line, positioned lower, tracks vehicles moving upward. In this frame, the system has detected multiple vehicles, including cars and a truck. However, no vehicles have crossed either the red or blue line yet, as shown by the counts at the top left: "going down - 0" and "going up - 0." YOLOv8 is actively identifying objects, and details of the detection (e.g., 8 cars, 1 airplane, 1 truck) are displayed below the frame. Additionally, the system provides information about the speed of different stages of processing: 3.4ms for preprocessing, 516.0ms for inference, and 1.4ms for postprocessing. This output demonstrates the system's ability to track vehicles and count their movement as they cross predefined reference lines.

**Conclusion:**

In this project, we developed a real-time vehicle detection, tracking, and counting system using YOLOv8. The system effectively tracks vehicles as they cross predefined lines and counts their movement in different directions. By combining a powerful object detection model with a simple tracking algorithm, we achieved real-time performance suitable for traffic monitoring applications. While the system works well in straightforward traffic scenarios, future improvements can be made to handle occlusions, overlapping vehicles, and varying traffic conditions. Overall, this system provides a lightweight and efficient solution for automated traffic surveillance.

### ****References****:

1. Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition.
2. Bewley, A., Ge, Z., Ott, L., Ramos, F., & Upcroft, B. (2016). Simple Online and Realtime Tracking. Proceedings of the IEEE International Conference on Image Processing.
3. Bochkovskiy, A., Wang, C., & Liao, H. M. (2020). YOLOv4: Optimal Speed and Accuracy of Object Detection. arXiv preprint arXiv:2004.10934.
4. Chen, L., Ai, H., Zhuang, Z., & Shang, C. (2020). Real-time Vehicle Detection and Tracking Using YOLO Combined with DeepSORT. Journal of Advanced Transportation.