Vehicle Detection, Tracking, and Counting using YOLOv8 and SORT

Muna Alsaber

This assignment aims to detect and track vehicles (cars and trucks) in a given video using computer vision techniques. The main objectives are to accurately detect vehicles, track their movements, and count the number of vehicles entering and exiting each side of the road.

To achieve this, the i used these components:

YOLOv8 (You Only Look Once, Version 8): A state-of-the-art, real-time object detection model that identifies vehicles in each frame of the video.
SORT (Simple Online and Realtime Tracking): A lightweight tracking algorithm that associates detected objects across frames to maintain a unique identifier for each vehicle.

In ths assignment I process a video, detect vehicles, track their movements, and count them based on their trajectory relative to predefined lines representing entry and exit points on the road.

Implementation

The assignment is implemented in Python, using the following libraries and frameworks:

- `cv2`: OpenCV for video processing and drawing functionalities.
- `numpy`: For array manipulations.
- `ultralytics`: For loading and utilizing the YOLOv8 model.
- `sort`: An open-source implementation of the SORT tracking algorithm from the repository

  https://github.com/abewley/sort

Approach

1. **Video Loading**: The video is loaded using OpenCV.
2. **Detection**: YOLOv8 detects vehicles in each frame.
3. **Tracking**: SORT tracks the detected vehicles across frames.
4. **Counting**: Vehicles are counted based on their movements across predefined lines representing inbound and outbound lanes.

1. Import Libraries

```
import cv2
import numpy as np
from ultralytics import YOLO
from sort.sort import Sort
```

cv2 (OpenCV): I Used for it video capture and frame manipulation, drawing bounding boxes and lines, and displaying the output.
numpy: Utilized for numerical operations and handling arrays.
ultralytics: Contains the YOLOv8 model for object detection.
sort: The SORT tracker implementation for tracking detected objects across video frames.

2. Load YOLOv8 Model

```
model = YOLO('yolov8n.pt')
```

I Loaded+ the pre-trained YOLOv8 model from a specified file (yolov8n.pt). This model is used to detect objects (vehicles) in each frame of the video.

3. Initialize SORT Tracker

```
sort_tracker = Sort(max_age=5, min_hits=3)
```

Initialize the SORT tracker with parameters is (by experimental):
max_age=5: which is the maximum number of frames an object can be missing before it is deleted.
min_hits=3: which is the minimum number of frames an object must be detected before it is considered a valid track.

4. Load Video

```
cap = cv2.VideoCapture('56310-479197605_small.mp4')
```

Open the video file using OpenCV. This will be used to read frames one by one for processing.

5. Define Regions for Counting Vehicles

```
line_in = [(480, 300), (1000, 300)]  # Inbound/green Lane (right side of
the video)
line_out = [(0, 300), (480, 300)]   # Outbound/red lane (left side of
the video)
```

line_in: Represents the line for counting vehicles entering the inbound lane.
line_out: Represents the line for counting vehicles exiting the outbound lane.

6. Initialize Counters and Data Structures

```
count_in = 0
count_out = 0
last_positions = {}
counted_in = set()
counted_out = set()
```

count_in: Counter for vehicles entering the inbound lane.
count_out: Counter for vehicles exiting the outbound lane.
last_positions: Dictionary to store the last known position (center point) of each tracked vehicle.
counted_in: Set to keep track of vehicle IDs that have already been counted as entering.
counted_out: Set to keep track of vehicle IDs that have already been counted as exiting.

7. Process Video Frames

```
while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break
```

cap.isOpened(): Check if the video file is opened successfully.
cap.read(): Read the next frame from the video. If no frame is returned (ret is False), break the loop.

8. Perform Detection Using YOLOv8

```
results = model(frame)
detections = []
for result in results:
    for box in result.boxes:
        x1, y1, x2, y2 = map(int, box.xyxy[0].tolist())
        label = model.names[int(box.cls)]
        score = box.conf.item()
        if label in ['car', 'truck'] and score > 0.4:
            detections.append(([x1, y1, x2, y2], score, label))
            center = (int((x1 + x2) / 2), int((y1 + y2) / 2))
            cv2.circle(frame, center, 5, (255, 255, 0), -1)
            cv2.rectangle(frame, (x1, y1), (x2, y2), (255, 0, 0), 2)
            cv2.putText(frame, f'{label}', (x1+50, y1 - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 0, 0), 2)
```

model(frame): Use the YOLOv8 model to detect objects in the current frame.
detections: List to store filtered detections (only cars and trucks with confidence scores above 0.4).

Draw bounding boxes and labels for detected objects on the frame.

9. Convert Detections to Numpy Array to match update function needs in sort

```
detections_array = []
for detection in detections:
    bbox, score, label = detection
    x1, y1, x2, y2 = bbox
    detections_array.append([x1, y1, x2, y2, score])
detections_np = np.array(detections_array)
```

Convert the list of detections to a numpy array for input to the SORT tracker.

10. Update SORT Tracker

```
tracked_objects = sort_tracker.update(detections_np)
```

sort_tracker.update(detections_np): Update the SORT tracker with the current frame's detections and get the updated list of tracked objects.

11. Process Tracked Objects

```
for track in tracked_objects:
    x1, y1, x2, y2, track_id = track
    track_id = int(track_id)
    center = (int((x1 + x2) / 2), int((y1 + y2) / 2))
    cv2.rectangle(frame, (int(x1), int(y1)), (int(x2), int(y2)), (0, 0,
255), 2)
    cv2.circle(frame, center, 5, (125, 155, 0), -1)
    cv2.putText(frame, f'ID: {track_id}', (int(x1), int(y1) - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)
```

track: Each tracked object contains bounding box coordinates and a unique track ID.
Draw bounding boxes, centers, and track IDs on the frame for visualization.

12. Check for Line Crossings and Update Counters

```
if track_id not in last_positions:
    last_positions[track_id] = center
if track_id not in counted_in and last_positions[track_id][1] >
line_in[0][1] and center[1] <= line_in[0][1]:
    count_in += 1
    counted_in.add(track_id)
    print(f'Vehicle {track_id} entered (Inbound)')
if track_id not in counted_out and last_positions[track_id][1] <
line_out[0][1] and center[1] >= line_out[0][1]:
    count_out += 1
    counted_out.add(track_id)
    print(f'Vehicle {track_id} exited (Outbound)')
last_positions[track_id] = center
```

last_positions: Store the last known center point of each tracked vehicle.
count_in: Increment if a vehicle crosses the inbound line and has not been counted yet.
count_out: Increment if a vehicle crosses the outbound line and has not been counted yet.
Update last positions for each track ID.

13. Draw Lines and Display Counts

```
cv2.line(frame, line_in[0], line_in[1], (0, 255, 0), 2)
cv2.line(frame, line_out[0], line_out[1], (0, 0, 255), 2)
cv2.putText(frame, f'In: {count_in}', (50, 50),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
cv2.putText(frame, f'Out: {count_out}', (50, 100),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)
cv2.imshow('YOLOv8 + SORT', frame)
if cv2.waitKey(0) & 0xFF == ord('q'):
    break
```

Draw inbound and outbound lines on the frame.
Display the current counts of inbound and outbound vehicles on the frame.
Show the processed frame in a window and wait for the 'q' key to quit the loop.

14. Release Video and Destroy Windowspython

```
cap.release()
cv2.destroyAllWindows()
```

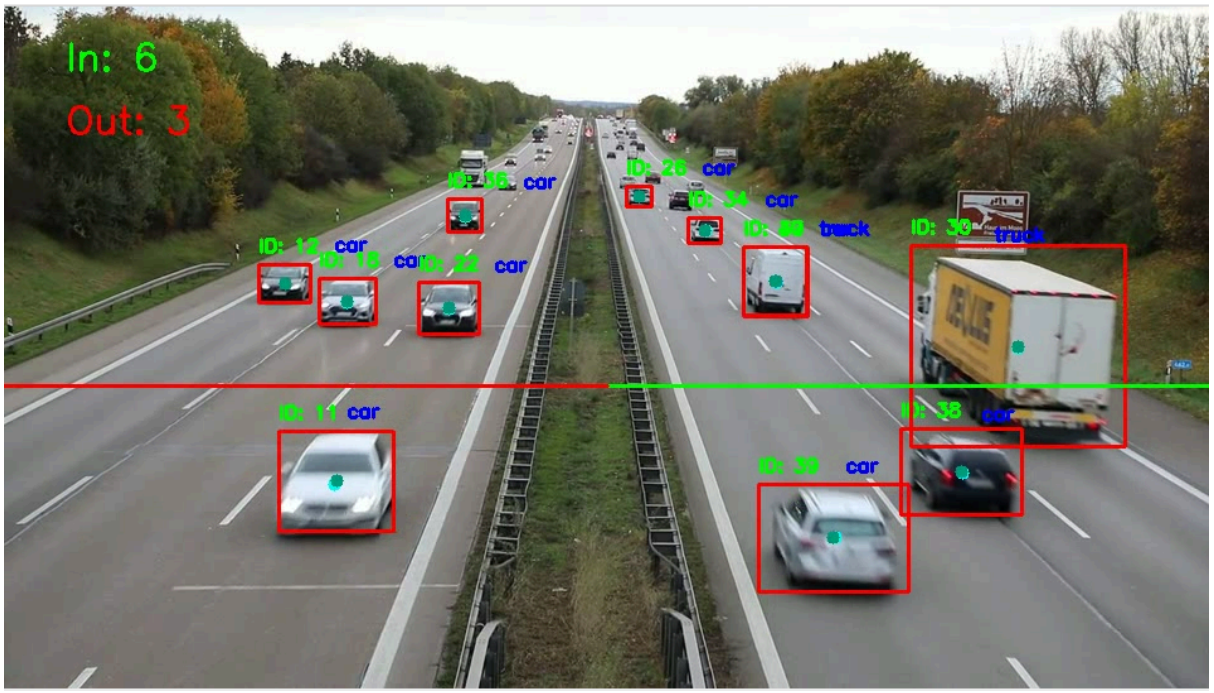
Release the video capture object.
Close all OpenCV windows.

Results

The output video is saved as output.mp4

```
    out.write(frame)
```

https://drive.google.com/file/d/1aX8AU2NcbDin_sFIMAU7SF2ZxpRmZvSi/view?usp=sharing



OBJ