Vehicle Detection, Tracking and Speed Estimation

NEGIN HEZARJARIBI

Computer Vision Project (DLBAIPCV01)

Studies: Bachelor in Applied Artificial Intelligence

IU INTERNATIONAL UNIVERSITY OF APPLIED SCIENCES (IU) | 14 FEBRUARY 2025

Table of Contents

List of Figures and Tables ..... *III*

Introduction **1**

1. Faster R-CNN **1**

1.1 Faster R-CNN Architecture 1

1.2 Advantages and Applications of Faster R-CNN 3

2. Single Shot Multi-Box Detector (SSD) **3**

2.1 Architecture and Key Features 3

2.2 Advantages and Limitations 4

3. The You Only Look Once (YOLO) **4**

4. Comparative Analysis of YOLOv8, Faster R-CNN, and SSD: Selecting the Optimal Model for Our System **5**

4.1 Comparison of YOLOv8, Faster R-CNN, and SSD 5

4.2 Model Selection for Our System 6

5. Development of a Vehicle Detection, Tracking, and Speed Monitoring System Using YOLOv8 ..... **6**

5.1 Video Input and Preprocessing 7

5.2 Vehicle Detection using YOLOv8 8

5.3 Vehicle Tracking with DeepSORT 9

5.4 Speed Estimation 9

5.5 Finalizing Output 10

5.6 Results and Video Demonstration..... 11

5.7 Discussion and Challenges 11

Conclusion.....12

Tools and Libraries Used 13

References.....14

List of Figures and Tables

FIGURE 1:FASTER R-CNN ARCHITECTURE. 2

Introduction

Road traffic accidents are a significant global concern, causing approximately 1.3 million deaths annually (World Health Organization, 2022). Speeding is one of the main factors contributing to traffic-related fatalities, making speed monitoring essential for road safety.One approach to preventing such accidents is the implementation of a vehicle speed tracking system that detects speeding vehicles and issues alerts when they exceed the speed limit.

This report presents a project aimed at developing a vehicle speed tracking system that detects vehicles on a highway, estimates their speed, and issues alerts for speed violations. Various algorithms can be used for detecting and tracking vehicles to build such a system. However, their accuracy and reliability may vary. Therefore, we first address three leading object detection models—Faster R-CNN, Single Shot MultiBox Detector (SSD), and YOLO (You Only Look Once)—and their features and limitations.

After a thorough evaluation, we selected YOLOv8 because it offers a notable balance between speed and accuracy, and DeepSORT for maintaining consistent tracking across frames, even under challenging conditions such as occlusions or varying lighting. Additionally, we explain how vehicle speed is estimated and how the system issues alerts for speed violations. At the end of this report, we provide a link to a video demonstrating the results of our project, and we discuss the challenges in this project.

1. Faster R-CNN

Faster R-CNN is a significant advancement in object detection, introducing a Region Proposal Network (RPN) to improve efficiency in generating region proposals within a single deep learning framework (Aboyomi & Daniel, 2023). Its architecture consists of several interconnected modules designed to improve both accuracy and speed in object detection tasks.

1.1 Faster R-CNN Architecture

Backbone Network: Faster R-CNN starts with a deep convolutional neural network (CNN) such as ResNet or VGGNet, which is pretrained on large-scale datasets like ImageNet. This backbone acts as a feature extractor, transforming input images into feature maps with more abstract representations (Aboyomi & Daniel, 2023).

Region Proposal Network (RPN): A significant advancement of Faster R-CNN is the integration of RPN, which operates on the feature maps generated by the backbone network. The RPN is in charge

of producing region proposals, which are potential bounding boxes containing objects. It slides a small network over the feature maps to predict objectness scores and bounding box coordinates (Aboyomi & Daniel, 2023).

Region of Interest (RoI) Pooling and RoI Align: After generating region proposals, Faster R-CNN uses RoI Pooling or RoI Align to extract fixed-size feature maps from the feature maps produced by the backbone network. This guarantees that each region of interest (RoI) is represented by a uniform-sized feature map, regardless of its scale or aspect ratio (Programmathically, 2022).

Detection Head: The extracted RoI features are fed into a detection head consisting of fully connected layers and softmax classifiers. This stage predicts class probabilities and refines the bounding box coordinates for each region proposal, improving the accuracy of object localization and classification (Aboyomi & Daniel, 2023).

Loss Function: During training, Faster R-CNN optimizes its parameters using a multi-task loss function. This function includes both classification loss (e.g., Softmax Cross-Entropy) and localization loss (e.g., Smooth L1 Loss) to penalize incorrect predictions and encourage accurate object localization (Aboyomi & Daniel, 2023).

Post-Processing: After making predictions, non-maximum suppression (NMS) is applied to filter out redundant bounding boxes and retain only the most confident detections for each object class (ResearchGate, 2020).

Its architecture, illustrated in Figure 1 (Aboyomi & Daniel, 2023), demonstrates the step-by-step process of Faster R-CNN, from input image processing to final object classification.

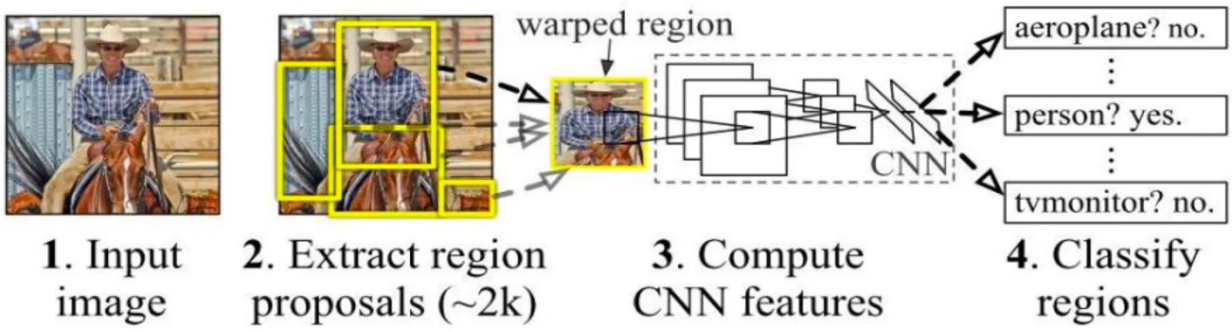

Figure 1:Faster R-CNN Architecture.

1.2 Advantages and Applications of Faster R-CNN

Thanks to its modular architecture and the integration of RPN for efficient region proposal generation, Faster R-CNN has become a cornerstone in object detection. It offers high accuracy while maintaining relatively fast inference speeds, making it suitable for a wide range of applications such as autonomous driving, aerial image analysis, and medical imaging (Aboyomi & Daniel, 2023). Since precise object localization and classification are crucial in many industries, Faster R-CNN is an ideal choice for tasks requiring high accuracy.

2. Single Shot Multi-Box Detector (SSD)

Another significant model in object detection is the Single Shot Multi-Box Detector (SSD), designed to improve detection efficiency by eliminating the need for region proposals while still achieving real-time performance. Unlike the R-CNN family, which follows a two-stage detection process, SSD performs object detection in a single pass, significantly reducing computational complexity while maintaining competitive accuracy (Aboyomi & Daniel, 2023).

2.1 Architecture and Key Features

Base Network: SSD employs a convolutional neural network (CNN) such as VGG16 as its backbone for feature extraction. Unlike Faster R-CNN, which relies on region proposal networks, SSD directly predicts object classes and bounding boxes from feature maps (Liu et al., 2016).

Multi-Scale Feature Maps: SSD utilizes multiple feature maps at different scales to improve the detection of objects of varying sizes. This multi-scale detection approach enhances accuracy, particularly for small objects, which are often missed by single-resolution detectors (Programmathically, 2022).

Default (Anchor) Boxes: SSD assigns multiple default bounding boxes with different aspect ratios to each feature map cell. This approach increases detection flexibility and accuracy by refining object locations using bounding box regression (GeeksforGeeks, 2024).

Single Shot Detection: Unlike Faster R-CNN, which involves separate steps for region proposals and classification, SSD processes the entire image in a single pass. This enhances its speed, achieving up

to 59 FPS with SSD300 on the VOC-2012 dataset while maintaining a mean Average Precision (mAP) of 74.3% (Aboyomi & Daniel, 2023).

Loss Function: SSD employs a combined loss function with:

Localization Loss (Lloc): Smooth L1 loss to minimize bounding box prediction errors.

Confidence Loss (Lconf): A softmax-based classification loss to determine object categories (Liu et al., 2016).

2.2 Advantages and Limitations

The main advantage of SSD is its ability to balance speed and accuracy, making it a strong alternative to Faster R-CNN. SSD's real-time detection capability makes it suitable for applications like autonomous driving, surveillance, and robotics. However, SSD suffers from higher classification errors compared to R-CNN-based models, particularly for small and overlapping objects (GeeksforGeeks, 2024). Still, SSD is widely used in modern computer vision tasks because it is fast and can handle different scales well.

3. The You Only Look Once (YOLO)

The You Only Look Once (YOLO) algorithm revolutionized object detection by adopting a single-shot detection approach. Unlike previous methods that relied on region proposals and multiple iterations, YOLO applies a single convolutional neural network (CNN) to the entire image, dividing it into a grid and predicting bounding boxes and class probabilities simultaneously. This architectural design significantly enhances detection speed, making YOLO one of the fastest object detection algorithms available. For instance, the original YOLO model ran at 155 frames per second (fps) with a mean average precision (mAP) of 52.7% on the VOC-2007 dataset, while an improved version achieved 63.4% mAP at 45 fps (ResearchGate, 2020).

Although YOLO is very fast, it has some drawbacks. One major issue is lower accuracy in detecting small objects because its grid-based method has trouble capturing fine details. Also, compared to two-stage detectors like Faster R-CNN, YOLO has lower localization accuracy. To improve this, later versions like YOLOv2 and YOLOv3 added features such as anchor boxes and better feature extraction (Programmathically, 2022).

YOLOv8 represents the latest advancement in the YOLO series, combining high accuracy with real-time processing capabilities. It has been optimized for both precision and speed, achieving a mean average precision (mAP@50) of 0.62 while maintaining low latency. The efficiency of YOLOv8 makes it particularly suitable for applications that require real-time object detection, such as autonomous driving, surveillance, and robotics (Keylabs, 2024).

4. Comparative Analysis of YOLOv8, Faster R-CNN, and SSD: Selecting the Optimal Model for Our System

In selecting the most suitable object detection model for our project, which focuses on detecting, tracking, and estimating vehicle speeds, we compared three state-of-the-art models: **Faster R-CNN, YOLOv8, and SSD**. Each of these models has its own strengths and weaknesses in accuracy, speed, and efficiency. This section compares them to explain our choice of model.

4.1 Comparison of YOLOv8, Faster R-CNN, and SSD

Faster R-CNN is widely recognized for its high accuracy due to its two-stage detection process. However, this accuracy comes at the cost of slower inference times, making it less suitable for real-time applications (Keylabs, 2024).

YOLOv8 follows a single-shot detection approach, enabling real-time performance while maintaining a high level of accuracy. Compared to Faster R-CNN, YOLOv8 significantly reduces inference time while achieving competitive precision, making it ideal for real-time tasks like vehicle speed estimation (Keylabs, 2024).

SSD (Single Shot MultiBox Detector) balances accuracy and speed by detecting objects in a single pass. However, SSD tends to perform worse than YOLOv8 in detecting small objects, which may impact vehicle tracking reliability (ResearchGate, 2020).

To provide a clearer comparison, we summarize key performance metrics in the table below:

Model	mAP (Mean Average Precision)	Inference Speed (ms)	Strengths	Weaknesses
Faster R-CNN	0.41 (Keylabs, 2024)	54 ms (Keylabs, 2024)	High accuracy	Slow inference time
YOLOv8	0.62 (Keylabs, 2024)	1.3 ms (Keylabs, 2024)	Fast inference, high accuracy	Slightly lower localization accuracy than Faster R-CNN
SSD	0.50 (ResearchGate, 2020)	~10ms (ResearchGate, 2020)	Balance of speed and accuracy	Struggles with small object detection

4.2 Model Selection for Our System

Since our project requires real-time vehicle detection and speed estimation, YOLOv8 is the best choice. While Faster R-CNN provides superior accuracy, its high computational cost makes it impractical for real-time applications. SSD, although faster than Faster R-CNN, sacrifices some accuracy, particularly in detecting smaller objects.

Considering both accuracy and speed, YOLOv8 achieves the best balance, ensuring that vehicles can be accurately detected and tracked with minimal delay. The model's real-time processing capability allows timely responses to speed violations, making it an ideal solution for our system (Keylabs, 2024).

5. Development of a Vehicle Detection, Tracking, and Speed Monitoring System Using YOLOv8

The development of an efficient vehicle detection, tracking, and speed monitoring system is crucial for various applications, including traffic management, law enforcement, and autonomous driving. In this study, we implemented a system using YOLOv8 for object detection and DeepSORT for multi-object tracking. The goal was to accurately identify and track vehicles in real-time while estimating their speeds based on motion analysis.

The system consists of three main components: vehicle detection using YOLOv8, vehicle tracking using DeepSORT, and speed estimation based on frame-to-frame displacement. The implementation follows a structured pipeline as described below.

5.1 Video Input and Preprocessing

The input to the system is a video file containing road traffic scenes. The video used for testing the system was obtained from Pixabay (Pixabay, 2016).

The video is read frame by frame using OpenCV (cv2), ensuring compatibility with various video formats. The system first checks whether the video file exists before proceeding to avoid runtime errors.

```
import os

import cv2

import math

# Video path and directory setup

video_path = "highway_video.mp4"

os.makedirs("results", exist_ok=True)

if not os.path.exists(video_path):

    raise FileNotFoundError(f"Video not found at {video_path}")

# Load video file

cap = cv2.VideoCapture(video_path)

fps = int(cap.get(cv2.CAP_PROP_FPS))

width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))

height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

fourcc = cv2.VideoWriter_fourcc(*'mp4v')

out = cv2.VideoWriter("results/output.mp4", fourcc, fps, (width, height))
```

5.2 Vehicle Detection using YOLOv8

To detect vehicles in each frame, we use the YOLOv8 (You Only Look Once, version 8) model. A pre-trained YOLOv8n model is loaded to identify objects with their respective class labels and confidence scores. The model processes each frame and extracts bounding boxes for detected vehicles.

```
from ultralytics import YOLO

# Load YOLOv8 model

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

Once the model is loaded, vehicle detection is performed on every frame.

```
# Process video frame by frame

while cap.isOpened():

    ret, frame = cap.read()

    if not ret or frame is None:

        break

    # Perform YOLO detection

    results = model(frame, stream=True)

    detections = []

    for result in results:

        for box in result.boxes.cpu().numpy():

            class_id = int(box.cls[0])

            conf = float(box.conf[0])

            if result.names[class_id] == 'car' and conf > 0.5:

                x1, y1, x2, y2 = map(int, box.xyxy[0])

                detections.append(((x1, y1, x2 - x1, y2 - y1), conf, 'car'))
```

5.3 Vehicle Tracking with DeepSORT

After detecting vehicles, the next step is tracking them across multiple frames. DeepSORT (Deep Simple Online and Realtime Tracker) is used for this purpose. The tracker assigns a unique ID to each detected vehicle and maintains its trajectory across frames.

```
# Ensure detections exist before passing to DeepSORT

if detections:

    tracks = tracker.update_tracks(detections, frame=frame)

    for track in tracks:

        if not track.is_confirmed():

            continue



        track_id = track.track_id

        x1, y1, x2, y2 = map(int, track.to_ltrb())

        center_x, center_y = (x1 + x2) // 2, (y1 + y2) // 2
```

5.4 Speed Estimation

To estimate vehicle speed, the system calculates the displacement of a tracked vehicle between consecutive frames. Given the frame rate of the video, the speed is computed in meters per second and converted to kilometers per hour.

```
import math

pixel_to_meter = 0.05  # Conversion factor (must be calibrated based on real-world
dimensions)

speed_limit_kmh = 30  # Speed threshold in km/h
```

```
car_positions = {}
```

For each tracked vehicle, the displacement is measured between consecutive frames to compute its speed.

```
        # Speed estimation
        if track_id in car_positions:
            prev_x, prev_y = car_positions[track_id]
            distance_pixels = math.sqrt((center_x - prev_x)**2 + (center_y - 
prev_y)**2)
            distance_meters = distance_pixels * pixel_to_meter
            time_seconds = 1 / fps
            speed_kmh = (distance_meters / time_seconds) * 3.6
        else:
            speed_kmh = 0  # Initial value for new vehicles
```

Vehicles exceeding the speed limit are marked in red, while others are displayed in green.

```
        # Assign color based on speed threshold
        color = (0, 255, 0) if speed_kmh <= speed_limit_kmh else (0, 0, 255)
        cv2.rectangle(frame, (x1, y1), (x2, y2), color, 2)
        cv2.putText(frame, f"ID: {track_id} Speed: {speed_kmh:.2f} km/h", (x1, y1 
- 10),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2)
        # Store position for next frame
        car_positions[track_id] = (center_x, center_y)
```

5.5 Finalizing Output

The processed video frames are saved as an output file to visualize detection, tracking, and speed monitoring.

```
    # Save processed frame
    out.write(frame)
cap.release()
out.release()
cv2.destroyAllWindows()
print("Processing complete. Video saved in results/output.mp4")
```

5.6 Results and Video Demonstration

The system successfully detects, tracks, and monitors vehicle speeds in real time, changing the bounding box from green to red to visually alert speeding vehicles when their speed exceeds the predefined threshold of 30 km/h. The final implementation is demonstrated in the video linked below.

[https://youtu.be/D9xluA0tCR4?si=FPgfp9i3av6CXEVx]

To improve accuracy, several adjustments were made:

- The pixel-to-meter scale was set to 0.0461 based on real-world measurements.
- The YOLO detection confidence threshold was increased to 0.6 to reduce false detections.
- DeepSORT parameters were fine-tuned to max_age = 15, n_init = 5, max_cosine_distance = 0.25 for better tracking stability.

The final implementation, including these improvements, is demonstrated in the video linked below:

[https://youtu.be/QpQxzGtMqsk?si=hjId_GMAUS7RwVRF]

And you can see the code in the github link below:

[https://github.com/NeginHz/Vehicle-Detection-Tracking-and-Speed-Estimation]

5.7 Discussion and Challenges

While the vehicle detection and speed estimation system performs well, several challenges still exist:

Lighting Conditions : Changes in lighting and shadows can affect the accuracy of YOLOv8 in detecting vehicles, especially in low-light or high-glare situations.

Vehicles Close to Each Other : When multiple vehicles are moving close together, the model may struggle to correctly separate them, leading to tracking errors.

Perspective and Size Variations : Vehicles appear smaller when they are farther from the camera, which can impact the accuracy of speed estimation and object tracking.

Tracking Fast-Moving Vehicles : At high speeds, the model may experience delays in tracking, causing occasional misalignment or missed detections.

Addressing these challenges requires further improvements, such as better calibration for speed estimation, advanced tracking algorithms, and additional training data to enhance detection accuracy in complex environments.

Conclusion

In this report, we first introduced three popular object detection models: Faster R-CNN, SSD, and YOLO. After comparing them based on past projects and their key features, we chose YOLOv8 as the best fit for our needs.

We then built a system that detects vehicles in real time, tracks them with DeepSORT, and measures their speed. If a vehicle goes over the speed limit, its bounding box turns from green to red as a warning. At the end of the report, there is a demo video showing how the system works.We also discussed some challenges related to using the YOLOv8 model in this project, such as lighting variations, difficulty in distinguishing closely moving and tracking inconsistencies with fast-moving objects.

Tools and Libraries Used

The implementation of the vehicle detection, tracking, and speed monitoring system utilized several key tools and libraries:

- Python 3.10: The primary programming language used for development (Van Rossum, 1995).
- OpenCV 4.8.0: Used for video processing, including frame extraction and object annotation (Bradski, 2000).
- YOLOv8: A state-of-the-art deep learning model for real-time object detection (Jocher et al., 2023).
- DeepSORT: A tracking algorithm integrating appearance-based association and motion-based Kalman filtering (Wojke et al., 2017).
- NumPy 1.23.5: Used for numerical computations and array manipulations (Harris et al., 2020).
- Matplotlib 3.7.1: Used for visualizing results and performance metrics (Hunter, 2007).

References

1. Bradski, G. (2000). The OpenCV library. *Dr. Dobb's Journal of Software Tools*.
2. Dakari Aboyomi, D., & Daniel, C. (2023). A comparative analysis of modern object detection algorithms: YOLO vs. SSD vs. Faster R-CNN. *ITEJ, 8(2).* https://doi.org/10.24235/itej.v8i2.123
3. GeeksforGeeks. (2024). How Single Shot Detector (SSD) works? Retrieved from https://www.geeksforgeeks.org/how-single-shot-detector-ssd-works/
4. Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., … & Oliphant, T. E. (2020). Array programming with NumPy. *Nature, 585*(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2
5. Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science & Engineering, 9*(3), 90–95. https://doi.org/10.1109/MCSE.2007.55
6. Jocher, G., Chaurasia, A., Qiu, J., & Stoken, A. (2023). YOLO by Ultralytics. Retrieved from https://github.com/ultralytics/ultralytics
7. Keylabs. (2024). *YOLOv8 vs. Faster R-CNN: A comparative analysis*. https://keylabs.ai/blog/yolov8-vs-faster-r-cnn-a-comparative-analysis/
8. Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C. Y., & Berg, A. C. (2016). SSD: Single shot multibox detector. In *Proceedings of the European Conference on Computer Vision (ECCV)* (pp. 21–37).
9. Oliphant, T. (2006). *A guide to NumPy*. Trelgol Publishing.
10. Pixabay.(2016). Roads, motorways, highway [Video]. Pixabay. https://pixabay.com/videos/roads-motorways-highway-1952/
11. Programmathically. (2022). *Deep learning architectures for object detection: YOLO vs. SSD vs. RCNN*. Retrieved from https://programmathically.com/deep-learning-architectures-for-object-detection-yolo-vs-ssd-vs-rcnn/
12. ResearchGate. (2020). Investigations of object detection in images/videos using various deep learning techniques and embedded platforms—A comprehensive review. *Applied Sciences, 10*(93280). https://doi.org/10.3390/app10093280
13. Ultralytics. (2023). *YOLOv8* documentation. Retrieved from https://docs.ultralytics.com/
14. Van Rossum, G. (1995). Python tutorial. *Centrum voor Wiskunde en Informatica (CWI)*.

15. Wojke, N., Bewley, A., & Paulus, D. (2017). Simple online and real-time tracker with a deep association metric. *2017 IEEE International Conference on Image Processing (ICIP)*, 3645–3649. https://doi.org/10.1109/ICIP.2017.8296962
16. World Health Organization. (2022). Road traffic injuries.Retrieved from https://www.who.int/news-room/fact-sheets/detail/road-traffic-injuries