Robotic Vision with Deep Learning: Implementing Real-Time Object Detection and Tracking for Mobile Robots Using Convolutional Neural Networks

Oke Iyanuoluwa Enoch

00659936

Supervisor: Dr. Maziar Nezhad

Moderator: Prof. Wei Yao

Programme: MScs. Robotics and Automation

School of CSE, University of Salford, Manchester M5 4WT, UK

i.e.oke@edu.salford.ac.uk

20th April 2025

University of Salford

School of Computing Science and Engineering

Student Name: Oke Iyanuoluwa Enoch

Course Code: SG-H690-M0063-56850-25

Project Title: Robotic Vision with Deep Learning: Implementing Real-Time Object Detection and Tracking for Mobile Robots Using Convolutional Neural Networks

I certify that this report is my own work. I have properly acknowledged all material that has been used from other sources, references, etc.

Student Signature: *Oke Iyanuoluwa Enoch*

Date: ....

Official stamp:

Submission date (to be entered by relevant School Office staff):
....

Abstract

The integration of deep learning in robotic vision has revolutionized real-time object detection and tracking, enabling mobile robots to perceive and interact with dynamic environments effectively. This dissertation explores the implementation of convolutional neural networks (CNNs), specifically YOLO (You Only Look Once) and SSD (Single Shot MultiBox Detector), for object detection and tracking in autonomous robotic systems. The research focuses on developing a vision-based perception framework using the CARLA simulator, which provides a high-fidelity urban environment for evaluating the performance of deep learning models in real-world traffic scenarios.

The methodology involves training and fine-tuning pre-trained CNN models using PyTorch, leveraging large-scale datasets to enhance object detection accuracy. The system is tested on various environmental conditions, including different lighting, occlusions, and varying object densities, to assess robustness and real-time performance. Metrics such as detection accuracy, inference speed, and tracking efficiency are analysed to evaluate the effectiveness of the proposed system.

The results demonstrate that deep learning-based object detection significantly improves robotic vision capabilities, allowing mobile robots to detect and track multiple objects with high precision in real-time. The findings contribute to enhancing robotic vision capabilities, particularly for mobile robots operating in complex and dynamic environments. This research lays the foundation for applications in autonomous navigation, intelligent traffic monitoring, and human-robot interaction. Future work will focus on optimizing computational efficiency by exploring techniques such as model pruning, quantization, and deployment on edge computing platforms to achieve real-time performance on resource constrained robotic systems.

Acknowledgements

I wish to express my profound gratitude to my supervisor, Dr. Maziar Nezhad, for his invaluable guidance, encouragement, and unwavering support throughout this project. His expertise and insightful feedback have been instrumental in shaping this work.

I am also deeply grateful to Prof. Wei Yao, who served as the moderator, for his valuable input and support.

I extend my sincere appreciation to the University of Salford, particularly the School of Computing, Science, and Engineering, for providing an excellent academic environment and the necessary resources to complete this dissertation.

My heartfelt gratitude goes to my parents for their unwavering love and support, which have been the foundation of my journey. Their belief in my abilities has been a constant source of motivation.

I would also like to thank my friends and colleagues for their encouragement, insightful discussions, and much-needed moments of respite. Their support has made this journey all the more enriching.

Finally, I acknowledge all those who, directly or indirectly, have contributed to my academic and personal growth. Your impact on this work and my life is deeply appreciated.
Thank you all.

Table of Content

List of Figures

List of Tables

Chapter 1: Introduction

1.1. Background

The integration of deep learning into robotic vision has significantly enhanced the perception and decision-making capabilities of autonomous systems. Object detection and tracking are critical for mobile robots, enabling them to understand their surroundings, avoid obstacles, and navigate safely in dynamic environments. Traditional computer vision methods, while effective in controlled conditions, often struggle with robustness, real-time performance, and generalization across different environments.

Recent advancements in convolutional neural networks (CNNs), particularly architectures like YOLO (You Only Look Once) and SSD (Single Shot MultiBox Detector), have demonstrated state-of-the-art performance in real-time object detection. These models offer a balance between speed and accuracy, making them well-suited for robotic applications. However, deploying such computationally intensive models on real-world mobile robots poses challenges in terms of inference speed, hardware constraints, and energy efficiency.

This research investigates the implementation of deep learning-based object detection and tracking for robotic vision using the CARLA simulator, a high-fidelity autonomous driving simulation platform. By leveraging CARLA's rich sensor suite and realistic urban environments, this work aims to assess the feasibility of integrating CNN-based perception models into mobile robotic systems.

1.2. Motivation

The demand for intelligent robotic systems capable of operating autonomously in complex, unstructured environments is rapidly increasing. Applications such as autonomous vehicles, robotic surveillance, industrial automation, and assistive robotics rely heavily on real-time perception for safe and efficient operation. Traditional rule-based vision algorithms lack adaptability and scalability, making deep learning-based approaches a promising alternative.

However, despite significant progress, real-world deployment of deep learning models in robotic systems is hindered by computational overhead, latency, and the need for robust adaptation to varying environmental conditions. This research is motivated by the need to bridge this gap by evaluating and optimizing CNN-based object detection models for real-time robotic vision applications.

1.3. Problem Statement

The primary challenge in robotic vision is achieving real-time, high-accuracy object detection and tracking in mobile robotics remains a challenging problem due to the following factors:

- Computational Complexity: Deep learning models require substantial computational resources, which may not be available on embedded robotic platforms.
- Real-Time Constraints: Many object detection models achieve high accuracy but fail to meet real-time processing requirements, which is crucial for autonomous robots.
- Robustness to Environmental Variability: Mobile robots operate in diverse environments with changing lighting conditions, occlusions, and sensor noise. Ensuring model generalization is critical.

This dissertation focuses on addressing these challenges by implementing and evaluating CNN-based object detection and tracking models in the CARLA simulator. The research will explore trade-offs between accuracy and inference speed, assess the impact of environmental conditions on model performance, and investigate optimization strategies for real-time deployment.

1.4. Research Objectives

The key objectives of this research are:

1. To implement and evaluate deep learning-based object detection models (YOLO and SSD) for robotic vision.
2. To compare and justify why YOLO outperforms SSD in terms of accuracy, inference speed, and robustness.
3. To implement and compare multi-object tracking models (ByteTrack vs. StrongSORT) for real-time tracking in robotic vision.
4. To examine and validate the factors that contribute to StrongSORT's enhanced performance over ByteTrack by evaluating critical metrics, such as its higher MOTA and MOTP scores for tracking accuracy, improved identity preservation indicated by considerably fewer false negatives, and the associated tradeoffs in real-time processing efficiency, to highlight its overall benefits in multi-object tracking systems.
5. To integrate and evaluate the combined object detection and tracking system (YOLO + StrongSORT or ByteTrack) within the CARLA simulator.
6. To analyse the performance of different models based on key metrics such as accuracy, inference speed (FPS), and robustness under various conditions.
7. To explore how CNN-based models contribute to vehicle control in CARLA, assessing their role in autonomous decision-making.
8. To assess the feasibility of deploying the trained models in real-world robotic vision applications, including potential optimizations for embedded systems.

1.5. Scope of the Thesis

This research focuses on the design, implementation, and evaluation of deep learning models for robotic vision using a simulation-based approach. The study is limited to:

- Implementing CNN-based object detection models (YOLO, SSD) in the CARLA simulator.
- Assessing model performance based on accuracy, precision, recall, F1-score, and inference speed.
- Investigating real-time constraints and computational efficiency for embedded deployment.
- Exploring potential real-world applications such as autonomous navigation, smart transportation, and robotic surveillance.

1.6. Dissertation Overview

This dissertation is structured as follows:

- Chapter 1: Introduction: Introduces the research topic, motivation, problem statement, objectives, and scope.
- Chapter 2: Literature Review: Analyses prior research on robotic vision, deep learning-based object detection, and perception systems, identifying research gaps.
- Chapter 3: Problem Modelling and Methods: Details the theoretical foundations, algorithm selection, and experimental setup for implementing deep learning models in robotic vision.
- Chapter 4: Development (Simulation, Algorithms, Systems): Covers the implementation of object detection models (YOLO, SSD) in the CARLA simulator, data collection, and model training processes.
- Chapter 5: Experiments: Describes the testing procedures, dataset preparation, simulation configurations, and benchmarking of the object detection models.
- Chapter 6: Results Analysis: Presents the evaluation metrics (accuracy, precision, recall, F1-score, FPS, computational efficiency) and discusses the performance of different models.
- Chapter 7: Conclusions and Future Work: Summarizes key findings, highlights limitations, and suggests future research directions such as real-world deployment and optimization strategies for embedded systems.

1.7. Data Sources and Analysis Techniques

Data Sources

The dataset used in this research is sourced from two primary sources:

- The dataset will be collected from CARLA simulator using camera sensors to generate annotated image data.
- Using the dataset of Berkeley DeepDrive 100K (BDD100k) which is the largest driving video dataset with 100K videos and 10 tasks which is used to train SSD and YOLO.

Data Preparation

- **Preprocessing Techniques**: Image normalization, data augmentation, and bounding box annotations to improve model generalization.
- **Feature Extraction**: CNN-based feature learning for object detection.

Analysis Techniques

- **Deep Learning Models**: Implementation of YOLO and SSD for object detection and tracking.
- **Evaluation Metrics**:
    - Accuracy, precision, recall, and F1-score for detection performance.
    - Inference speed (FPS) to measure real-time capability.
    - Robustness testing in different environmental conditions.

System Integration and Testing

- **Simulation Environments**: Models will be tested in the CARLA simulator, replicating real-world traffic and urban environments.
- **Real-World Testing**: Future work may explore deployment on embedded robotic platforms for real-time applications.

Chapter 2: Literature Review

The field of robotic vision has witnessed significant advancements in recent years, primarily due to the integration of artificial intelligence (AI) and deep learning techniques. Object detection and tracking are fundamental capabilities that enhance robotic perception, enabling autonomous navigation, surveillance, and interaction with dynamic environments. Various models and methodologies have been developed to improve accuracy, speed, and robustness in different robotic applications. This literature review categorizes recent contributions based on key aspects, including objectives, methodologies, sensor technologies, applications, challenges, future directions, and models used.

2.1 Object Detection in Robotic Vision

Object detection plays a critical role in robotic applications requiring real-time decision-making. Various deep learning-based approaches have been explored to enhance detection accuracy and efficiency in different robotic domains:

- (Foa, 2019) evaluates state-of-the-art object detection methods and selects YOLOv3 as the optimal framework for real-time implementation in mobile robots. The study compares Faster R-CNN, SSD, and YOLOv3, ultimately choosing YOLOv3 due to its balance between accuracy and computational efficiency. Implemented in Python, the system integrates 3D object localization using Kinect sensors, ensuring high-speed and precise tracking. The application focuses on autonomous mobile robots for industrial automation and service robotics. The study highlights challenges such as real-time processing constraints and sensor limitations. Future research will optimize computational efficiency and explore the integration of multi-sensor fusion for improved object tracking.
- (Dewi et al., 2024) propose a CNN-based vision system for an Autonomous UV (A-UV) disinfection robot. The system is designed to detect and classify obstacles while ensuring collision-free navigation in hospital environments. The CNN model achieves 97% training accuracy and 99% validation accuracy, demonstrating high reliability in obstacle detection. The study highlights applications in healthcare, sanitation, and industrial automation. Key challenges include real-time processing limitations and the need for robust navigation algorithms. Future work involves integrating sensor fusion techniques to improve detection performance in diverse environments.
- (Redmon et al., 2016) introduce the YOLO framework, which predicts bounding boxes and class probabilities simultaneously using a single deep neural network. The study demonstrates that YOLO is significantly faster than Faster R-CNN, achieving 45 FPS on a Titan X GPU, while maintaining high detection accuracy. However, YOLO struggles with small object detection and localization errors, requiring improvements in spatial resolution and anchor box design. Future

research aims to enhance YOLO's generalization capability and real-time efficiency on embedded systems.

- (Wei Liu et al., 2016) present SSD (Single Shot MultiBox Detector), a deep learning model designed for fast and accurate object detection. Unlike region-based methods, SSD directly predicts bounding boxes using multi-scale feature maps, significantly improving speed and accuracy. The model outperforms Faster R-CNN in real-time scenarios, achieving 74.3% mAP at 59 FPS with a 300×300 input size. SSD's advantage lies in its efficient feature fusion mechanism, enabling robust detection across different object sizes. Key challenges include handling small objects and reducing computational complexity for embedded applications. Future improvements focus on better anchor box selection and lightweight model adaptations.
- (Shafiee et al., 2017) propose Fast YOLO, an optimized version of YOLOv2, reducing parameters by 2.8× while maintaining high detection accuracy. The method leverages evolutionary deep intelligence to improve network efficiency and introduces motion-adaptive inference for video applications. Fast YOLO achieves 18 FPS on a Nvidia Jetson TX1, making it viable for real-time mobile and embedded robotics applications. Challenges include balancing accuracy with computational constraints. Future improvements involve better network pruning techniques and adaptive inference models.
- (Wahab et al., 2022) develop a real-time object detection system utilizing SSD and OpenCV, achieving 97% accuracy in detecting and classifying objects in real-world environments. The study evaluates COCO, PASCAL VOC, and Kitti datasets, demonstrating SSD's superior speed and precision. Applications include autonomous navigation, surveillance, and industrial robotics. The primary challenge is real-time inference on limited hardware, requiring optimized model deployment strategies.
- (Santos et al., 2023) propose an autonomous mobile inspection robot that utilizes YOLOv7 for object detection and feedback control to navigate within an industrial setting. The robot relies on Jetson Xavier NX, Raspberry Pi 4, a camera, and LiDAR for object detection and path planning. Their approach significantly reduces the need for human intervention in factory inspections by leveraging real-time AI processing on the edge. The study highlights computational challenges, such as CPU and GPU usage, and suggests optimizing sensor fusion techniques to improve real-time efficiency. Future work includes integrating reinforcement learning and Markov Chains for enhanced autonomy.
- (Poppinga & Laue, 2019) introduce JET-Net, a CNN-based real-time object detection model for NAO V5 robots in robotic football. JET-Net enhances detection efficiency by eliminating region proposal steps, reducing inference time to 9.0ms while maintaining high accuracy. The model integrates Simulation Transfer Learning (STL) for distance estimation and outperforms traditional

detectors in recall and detection range. Future improvements focus on XNOR-Net optimizations and orientation estimation for broader robotic applications.

- (Aslan et al., 2022) introduce four new lightweight convolutional neural network (CNN) models designed for object recognition in humanoid robots. These models focus on reducing computational complexity while maintaining high accuracy. The proposed models were evaluated using MNIST and CIFAR-10 datasets, demonstrating superior performance in training time and parameter efficiency compared to VGG-16 and ResNet-20. Additionally, the models were deployed on a Robotis-Op3 humanoid robot for real-world object classification. Future improvements involve enhancing network generalization to different environments and integrating multi-modal perception to improve object detection in cluttered scenes.
- (Zhou et al., 2022) propose a deep-learning-driven 3D object detection system for robotic manipulators using point cloud data from 3D cameras. The system utilizes PV-RCNN for robust object detection, overcoming challenges posed by occlusions, dynamic lighting, and background variations. The robot autonomously localizes, picks, and manipulates objects in a plug-in charging station and name-tag production line. The study suggests enhanced 3D SLAM integration for improved real-world performance.
- (Singh, 2023) present a computer-vision-based object detection framework optimized for indoor service robots, incorporating deep learning techniques such as YOLO and Faster R-CNN. The system integrates RGB cameras and depth sensors, allowing the robot to detect and classify objects with improved spatial awareness and navigation accuracy. The study demonstrates that YOLOv5 significantly outperforms Faster R-CNN in speed, making it suitable for real-time applications in indoor service scenarios like smart homes, retail, and healthcare assistance. Future research focuses on enhancing real-time object segmentation and reducing computational overhead on embedded robotic platforms.
- (Keča et al., 2023) explore deep learning approaches for detecting silver balls in the Robo-Cup-Junior Rescue Line challenge. The study compares Hough Transform (HT), convolutional neural networks (CNNs), and semantic segmentation (U-Net) for real-time ball detection on a Raspberry Pi-based educational robot. Experimental results show that RESNET50 achieved the highest accuracy, while MOBILENET_V3_LARGE_320 provided the fastest performance. However, HT was the most computationally efficient but lacked precision in complex environments. The study highlights challenges related to real-time processing on low-power hardware and suggests future research should integrate multi-sensor fusion and optimized deep learning architectures for enhanced performance.
- (Martinson & Yalla, 2016) introduce a hybrid detection approach that integrates a convolutional neural network (CNN) with a geometric feature-based layered pre-filter. The layered classifier acts

as a pre-filter, reducing the number of objects needing CNN classification, thus optimizing computational efficiency. The system, tested on Toyota Human Support Robot (HSR) and Toyota Hand-Over Robot (THOR), enhances precision-recall metrics while operating on low-end GPUs like Tegra K1. The study suggests future research should focus on refining multi-modal fusion techniques for greater occlusion robustness and speed improvements.

- (Bhardwaj et al., 2023) provides a comprehensive comparison of traditional vs. deep learning-based object detection. They analyse YOLO, SSD, and R-CNN architectures, emphasizing trade-offs between speed and accuracy. The study discusses the integration of AI robots in autonomous systems, highlighting the challenges in occlusion handling and real-time processing. Future research should focus on enhancing AI models for resource-limited hardware.
- (Gupta & Majumdar, 2019) compare YOLOv2 and Mask R-CNN for human tracking on a skid-steer mobile robot (SSMR). The Linear Quadratic Gaussian (LQG) controller is employed for velocity control, allowing smooth trajectory adaptation. Results indicate that YOLOv2 achieves real-time tracking (22-30 FPS), while Mask R-CNN provides higher accuracy at the cost of computational efficiency. Future research focuses on refining hybrid tracking approaches combining CNNs and Kalman filters.

2.2 Object Tracking in Autonomous Systems

Tracking and following moving objects in dynamic environments is a crucial task for autonomous robots. Several studies have focused on enhancing real-time tracking efficiency, addressing occlusions, motion blur, and environmental variations.

- (Chung & Duy, 2023) propose an approach that combines YOLOv8 for object detection with Strong SORT for tracking, improving object identification and tracking in robotic applications. The methodology involves training the YOLOv8 model to detect objects and utilizing Strong SORT to assign unique identifiers, ensuring seamless tracking even when objects are temporarily occluded. The authors implemented their model on a two-wheeled differential mobile robot and tested its ability to follow individuals dynamically. Their findings indicate the algorithm performs well in handling occlusions and rapid movements, offering significant potential for real-world person-following applications in healthcare, security, and service robotics. Future work includes optimizing computational efficiency and integrating multi-sensor data fusion for improved performance.
- (Rohan et al., 2019) explore the use of Convolutional Neural Networks (CNNs) for object detection and tracking on the Parrot AR Drone 2. The proposed system integrates CNN-based detection with a real-time tracking algorithm optimized for drones. The study highlights the performance of SSD (Single Shot Detector), which provides an accuracy of 98% for object

detection and 96.5% tracking efficiency. The tracking algorithm, which controls the drone's movement based on PID controllers, enables stable and continuous tracking. Applications include surveillance, search and rescue, and autonomous navigation. Future challenges involve improving real-time performance on embedded hardware and enhancing drone adaptability in varying environmental conditions.

- (Chen et al., 2023) introduce a stereo vision-based CNN tracker that allows a mobile robot to follow a human target in real-time, even in the presence of occlusions, illumination changes, and pose variations. The CNN is trained online and operates at 20 FPS, processing both RGB images and stereo depth data. This method allows the robot to predict the person's trajectory when temporarily out of sight, enhancing tracking reliability. The study introduces a new stereo dataset for person-following tasks and evaluates the approach against other real-time tracking algorithms, showing superior performance in real-world environments.
- (Juel et al., 2020) propose a tracking-by-detection framework that projects RGB-D sensor data onto a robot's map frame, allowing it to track human and object trajectories in real-time. The framework integrates YOLO, Faster R-CNN, and Mask R-CNN for object detection and uses DeepSORT for tracking. It operates on embedded systems such as the Jetson AGX Xavier, achieving a tracking speed of 2.6–13.3 Hz depending on camera configurations. Key applications include autonomous hospital robots, industrial automation, and mobile surveillance. Future research aims to improve occlusion handling, tracking stability in crowded environments, and computational efficiency.
- (Guerrero-Higueras et al., 2019) introduce PeTra, a CNN-based people tracking tool using 2D LIDAR scans. PeTra outperforms conventional Leg Detector (LD) in accuracy and robustness, achieving superior results in indoor security and navigation applications. The system was validated on a ROS-based mobile robot (Orbi-One) in indoor mock-up apartments, demonstrating higher recall and tracking efficiency. Future work aims at expanding detection to 3D LIDAR and integrating depth sensors for enhanced multi-person tracking.
- (Zhang et al., 2023) compare three deep learning-based tracking models: YOLOv5, Faster R-CNN, and SSD. They introduce four occlusion datasets and evaluate F1-score, precision, and recall. Results show that SSD achieves the highest overall tracking performance, outperforming YOLOv5 by 3.39× and Faster R-CNN by 13.34×. The study recommends hybrid models integrating RNNs for long-term tracking stability in industrial and security applications.
- (Rajasri et al., 2023) investigate YOLO-based tracking algorithms, where moving objects are detected and continuously tracked using adaptive detection cycles. The study highlights the balance between accuracy and computing cost, which is a major challenge in real-time detection. Proposed enhancements include adaptive frame rate regulation to optimize computational load

while maintaining tracking accuracy. The system has demonstrated effectiveness in applications like blind navigation assistance, railway safety, and autonomous driving.

- (Esfahlani et al., 2022) present SROBO, a ground robot integrating Deep-CNNs and Real-Time Graph-Based SLAM (RTAB-Map) for real-time navigation. The robot employs an RGB-D MYNTEYE camera, 2D LiDAR, and an IMU, combining Deep-CNN with SLAM to improve object detection accuracy by 35% compared to conventional SLAM. Key challenges include computational efficiency and sensor fusion, with future work exploring reinforcement learning for adaptive mapping.
- (Rajasri et al., 2023) investigate YOLO-based tracking algorithms, where moving objects are detected and continuously tracked using adaptive detection cycles. The study highlights the balance between accuracy and computing cost, which is a major challenge in real-time detection. Proposed enhancements include adaptive frame rate regulation to optimize computational load while maintaining tracking accuracy. The system has demonstrated effectiveness in applications like blind navigation assistance, railway safety, and autonomous driving.
- (Liu et al., 2017) propose a CNN-based vision model for end-to-end learning, where a robot directly converts camera input into steering commands. The model, trained using a human-operated robot in a structured environment, enables the system to navigate safely by distinguishing between forward, left, and right turns. The study also explores alternative methods, such as monocular vision-based obstacle detection and inverse perspective mapping (IPM), which aim to reduce computational power consumption. Future improvements include refining multi-modal sensor fusion to handle complex and unstructured terrains.

2.3 Why YOLO & StrongSORT?

YOLO (You Only Look Once) has revolutionized real-time object detection due to its single-pass detection framework. StrongSORT enhances tracking by providing robust association mechanisms:

- (Cherubin et al., 2024) implement YOLO on Raspberry Pi 4B, evaluating its mAP detection accuracy and CPU consumption during inference. The robot, operating with ROS Noetic, is equipped with LiDAR, Kinect, and odometry sensors. The study reveals that YOLOv3 offers the best balance between accuracy and computational efficiency on embedded platforms. The research highlights challenges in real-time processing and proposes edge AI optimizations for enhanced performance.
- (Abdul-Khalil et al., 2023) present a comprehensive review of AI-driven object detection techniques for AMRs, emphasizing sensor fusion and deep learning approaches. The study categorizes detection techniques into single-stage (YOLO, SSD, RetinaNet) and two-stage (Faster R-CNN, Cascade R-CNN, Mask R-CNN) methods. The authors analyse sensor fusion strategies

integrating LiDAR, radar, and vision-based sensors, concluding that multi-sensor systems enhance detection accuracy in unpredictable environments. Key challenges include computational efficiency, real-time processing, and environmental adaptability. The review suggests future research should focus on lightweight neural networks and edge AI to optimize AMRs for real-world deployment.

2.4 Object Detection in Embedded and Edge AI Systems

Deploying deep learning models on embedded systems presents challenges in computation and power efficiency. Several studies address these concerns:

- (Jiang et al., 2022) propose S-SSD (ShuffleNet-SSD), a lightweight deep learning model that enhances SSD (Single Shot Detector) for efficient object detection in indoor environments. The model replaces VGG-16 with ShuffleNet, significantly reducing computation while maintaining detection accuracy. Experimental results indicate that S-SSD surpasses Tiny-YOLO in terms of both detection speed and accuracy. Future research will focus on integrating low-power embedded platforms to further optimize inference times.

- (Martinez-Alpiste et al., 2022) design a lightweight object recognition framework optimized for Android-based mobile devices, integrating OpenCV, TensorFlow Lite, and Qualcomm Snapdragon SDK. The study evaluates YOLOv3, SSD, and Tiny-YOLO to compare efficiency, showing that TensorFlow Lite provides the best speed-accuracy trade-off. The architecture significantly reduces RAM usage and power consumption, making it ideal for portable, real-time applications. Future improvements aim at edge AI integration and optimizing CNN inference on mobile GPUs.

2.5 Simulation and Datasets in Autonomous Robotics

Simulation and datasets are essential components in training and evaluating AI models for autonomous robotics:

- (Dosovitskiy et al., 2017) introduce CARLA, an open-source driving simulator built on Unreal Engine 4, designed to support the development of autonomous driving systems. CARLA provides high-fidelity urban environments, realistic physics, and customizable sensor suites, enabling researchers to test models in diverse conditions. The simulator supports three driving approaches: a modular perception-based pipeline, an end-to-end model trained via imitation learning, and a reinforcement learning-based policy. CARLA also facilitates the evaluation of multi-agent interactions, road hazards, and weather variability. Future work focuses on enhancing dynamic pedestrian behaviour and improving real-world transferability of trained models.

- (Yu et al., 2020) present BDD100K, the largest driving video dataset, consisting of 100K video clips collected across diverse geographic locations, weather conditions, and times of the day. The dataset includes annotations for ten tasks, covering image classification, object detection, lane detection, drivable area segmentation, and multi-object tracking. The study also establishes benchmarks for heterogeneous multitask learning, where models must perform tasks with different output structures. The experiments highlight domain adaptation challenges and emphasize the need for specialized training techniques to handle real-world variability in driving conditions. Future research aims to improve cross-task learning efficiency and enhance dataset diversity for better generalization.

2.5.1 Limitations of Simulation-Based Approaches

Simulation tools like CARLA are valuable for training autonomous systems but face sim-to-real transfer challenges due to differences from real-world conditions. Factors like lighting, weather, and sensor noise are often oversimplified, affecting model generalization. This domain adaptation issue can reduce detection and tracking accuracy in real-world deployments. To address this, domain randomization and hybrid training with real data improve robustness. Future work should enhance CARLA's realism by integrating diverse environments, behaviors, and sensor imperfections for better real-world applicability.

2.6 Future Directions in Robotic Vision

Despite advancements, challenges such as real-time processing constraints, occlusion handling, and multi-sensor fusion persist:

- Improving YOLO’s small object detection and spatial resolution (Redmon et al., 2016).
- Enhancing computational efficiency in embedded applications (Cherubin et al., 2024).
- Integrating reinforcement learning for adaptive tracking (Santos et al., 2023).
- Refining multi-modal sensor fusion techniques (Guerrero-Higueras et al., 2019).

Literature Review Summary

The literature review highlights significant contributions in robotic vision, particularly in object detection and tracking. Studies demonstrate the effectiveness of deep learning models like YOLO, SSD, and Faster R-CNN in various robotic applications. Recent advancements focus on optimizing these models for embedded systems, enhancing real-time tracking, and integrating multi-sensor fusion for improved accuracy. Future research aims to address computational constraints, improve detection robustness, and develop lightweight models for resource-constrained platforms. The integration of AI-driven vision systems in autonomous robots is expected to play a crucial role in healthcare, surveillance, industrial automation, and human-robot interaction.

Chapter 3: Problem Modelling

3.1 Overview of CARLA Simulator

3.1.1 Introduction to CARLA

CARLA is an open-source simulation platform designed for research on autonomous driving. CARLA is meticulously designed to facilitate the creation, training, and validation of autonomous urban driving systems, offering a fully customized simulation environment. Alongside open-source code and protocols, it provides an extensive array of open digital assets, encompassing urban designs, structures, and vehicles, available for unrestricted usage in research and development. The platform accommodates the adaptable configuration of sensor suites, encompassing RGB cameras, LiDAR, radar, and depth sensors, together with modifiable ambient parameters such as illumination, meteorological conditions, and traffic dynamics.

3.1.2 Real-World Environment Simulation in CARLA

CARLA simulates real-world environments by accurately modelling road networks, traffic behaviour, pedestrian movement, and dynamic environmental factors. The simulation engine, built on Unreal Engine, ensures high-fidelity graphics and physics-based interactions, making it a powerful tool for studying perception, planning, and control in autonomous driving. The simulator follows the ASAM OpenDRIVE standard to define highway and urban environments, allowing for realistic and diverse testing scenarios. The image below illustrates CARLA's high-fidelity simulation environment, highlighting meticulously modeled road networks, dynamic traffic behavior, and realistic pedestrian movement.

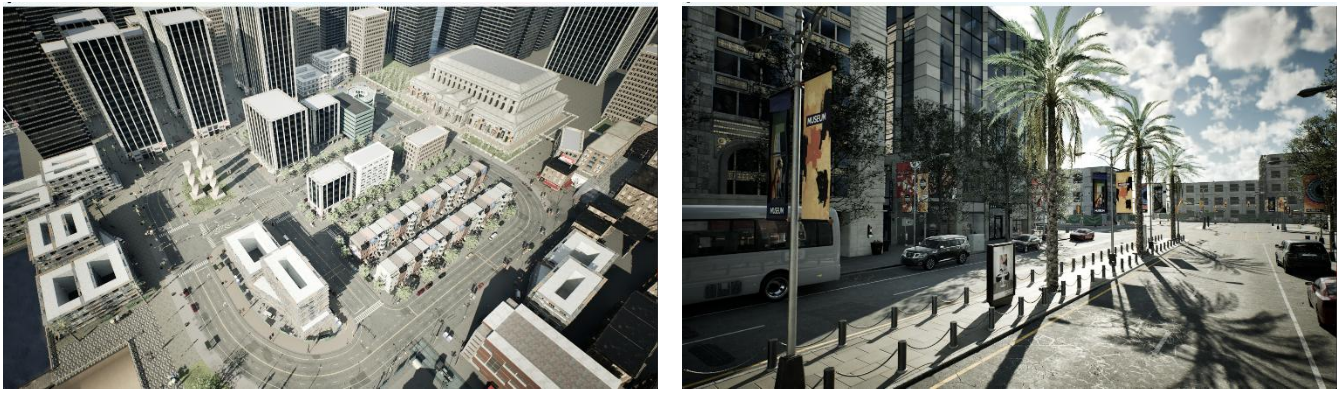


Fig 3.1: A Detailed View of CARLA's Simulated World

Alternatives to CARLA

There are several open-source simulators for autonomous vehicles and racing, but few offer capabilities on par with or beyond CARLA. Here are some I considered: AirSim, TORCS, Autoware

Additionally, two simpler options: CoppeliaSim, Udacity Simulator, Donkeycar Simulator
Ultimately, I chose to work with CARLA because I wanted a UE4-based simulator and found its community easier to engage with.

3.1.3 CARLA's Role in Deep Learning Applications

CARLA plays a crucial role in deep learning applications by generating and processing high-resolution camera footage for perception tasks. The simulator provides multiple viewpoints from onboard cameras, capturing essential data for object detection, lane segmentation, and tracking. This footage is processed using deep learning frameworks such as TensorFlow and PyTorch, where convolutional neural networks (CNNs) analyse the images for real-time decision-making. The CARLA API, implemented in Python and C++, enables seamless data extraction and real-time streaming of sensor data, supporting the training and validation of deep learning models.

3.2 Camera Footage Processing for Deep Learning

3.2.1 Camera Initialization and Attachment

In CARLA, the multi-modal sensor suite is initialized and attached to a vehicle using the 'spawn_actor' function, with precise positioning along the X, Y, and Z axes. This setup allows the simulation to capture real-time footage from several camera types such as:

- RGB Camera: Captures colour images of the environment.
- Semantic Segmentation Camera: Differentiates and labels various objects and surfaces.
- Depth Camera: Provides distance information for scene understanding.
- Instance Segmentation Camera: Distinguishes individual objects within the scene.

These camera outputs, combined with LiDAR data, form a robust perception system essential for deep learning applications in autonomous driving. Below is an image of the sensor window, displaying the multi-modal camera feeds alongside the LiDAR output:

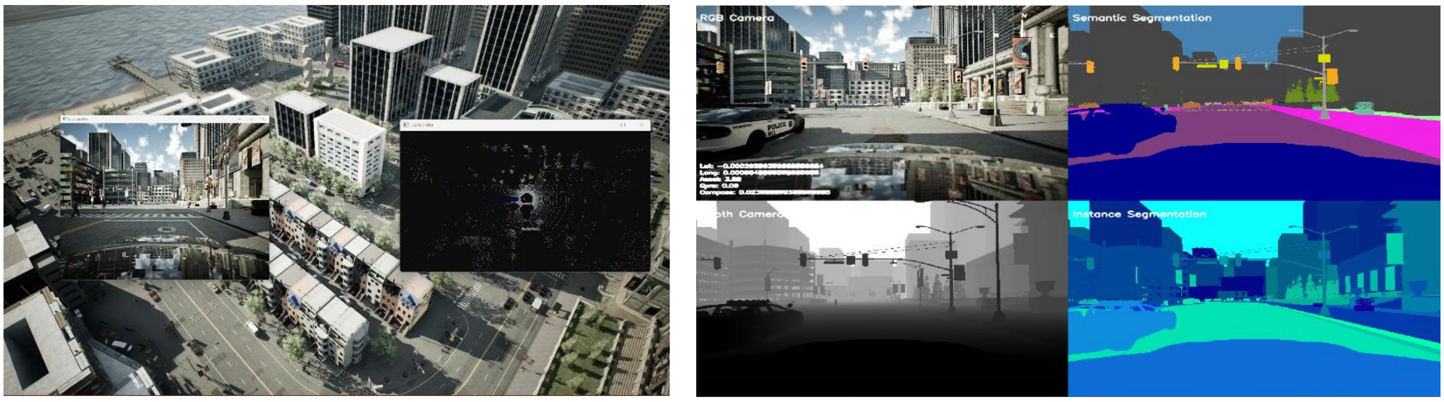


Figure 3.2: Camera and LiDAR Sensor Output Window

3.2.2 Image Processing and Integration with Deep Learning

The captured camera footage is processed using OpenCV, a widely used computer vision library, to handle image transformations, preprocessing, and real-time visualization. The frames are typically retrieved as NumPy arrays, enabling their direct integration into deep learning pipelines. These images are then fed into deep learning models, often using frameworks such as TensorFlow or PyTorch, for tasks such as object detection, semantic segmentation, and scene understanding.

3.2.3 Data Streaming and Augmentation for Training

By leveraging CARLA's API, camera data can be streamed in real-time, stored for offline training, or augmented with additional annotations such as bounding boxes and semantic labels. This ensures that the deep learning models trained on CARLA-generated data can generalize effectively to real-world driving scenarios.

3.3 Object Detection Methodology

3.3.1 Convolutional neural networks

A Convolutional Neural Network (CNN) is a deep learning framework intended for object identification tasks, including picture categorization, detection, and segmentation. Convolutional Neural Networks (CNNs) are modeled after the stratified architecture of the human visual cortex and specialize in the analysis of visual data.

3.3.1.1 CNN Architecture

A CNN consists of three main types of layers:

1. Input Layer
   - Receives raw data as input.
   - The number of neurons corresponds to the total number of features (e.g., pixels in an image).
2. Convolutional Layer
   - Applies filters (kernels) to extract local features from the input.
   - The operation is often defined as:

$$Z = W * X + b$$

   where, W represents learnable weights, X is the input, and b is the bias.
3. Activation Function
   - Introduces non-linearity into the model, with common choices being ReLU, sigmoid, or tanh.
   - This step enables the network to learn more complex patterns.
4. Pooling Layer
   - Reduces the spatial dimensions of the feature maps while retaining important features.

- Prevalent approaches including max pooling and average pooling.

5. Fully Connected Layers
   - Flatten the feature maps and map them to the final output predictions.
   - The output layer converts these features into class probabilities using activation functions such as softmax (for multi-class classification) or sigmoid (for binary classification).

A typical CNN architecture diagram (Fig 3) visually represents this layered process, highlighting how input data flows through convolutional, activation, pooling, and fully connected layers to produce the final classification or prediction output.

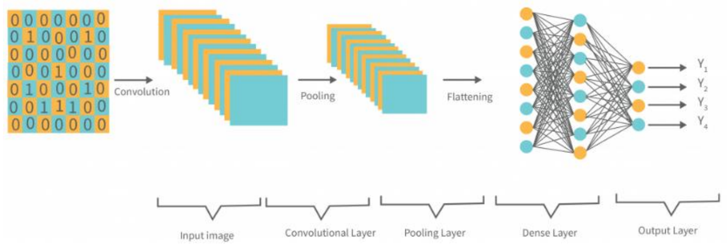


Fig 3.3: Diagram of CNN Architecture

3.3.1.2 Training Process

1. Feedforward Propagation
   - Data passes through each layer, generating an output.
2. Loss Calculation
   - An error function (e.g., cross-entropy, mean squared error) evaluates the model's performance.
3. Backpropagation & Optimization
   - The network adjusts weights using gradient descent to minimize the loss.
   - This iterative process improves model accuracy over multiple training epochs.

CNNs leverage hierarchical feature extraction, making them highly efficient for visual recognition tasks.

3.3.2 Dataset Preparation for Object Detection

3.3.2.1 What is Dataset?

A dataset in machine learning and artificial intelligence is a structured collection of data used for training, validating, and testing algorithms. Proper dataset management ensures that models learn accurate patterns and generalize well.

3.3.2.2 Dataset Splitting

A dataset is typically divided into three subsets:

1. Training Set: Used to train the model by learning patterns and relationships.
2. Validation Set: Helps fine-tune model parameters and prevent overfitting.
3. Testing Set: Evaluates final model performance with unseen data.

Separating datasets ensures unbiased model evaluation and prevents overfitting.

3.3.2.3 Steps involved in building a Dataset

1. **Data Collection**:

This is the first step in preparing a dataset for machine learning is selecting appropriate data sources. Sources of data collections are: Open-source datasets, Internet sources, Synthetic data generators

2. **Data Preprocessing:**

This stage is the fundamental principle in data science is to always assume a dataset is flawed unless it has been thoroughly validated. Preprocessing involves:

- Handling missing data
- Removing duplicates and inconsistencies
- Normalizing or standardizing values
- Feature engineering and selection

3. **Data Annotation:**

This stage involves data that must be annotated to make it interpretable for machine learning models. Annotation involves tasks such as:

- Labeling images for computer vision tasks.
- Tagging text for natural language processing.
- Assigning categories to structured data.

4. **Conversion from Annotation to Machine Learning Format (YOLO/SSD):**

Object detection models require properly formatted annotations to train effectively. Raw annotation data, often in formats like Pascal VOC (XML) or COCO (JSON), must be converted into the correct format based on the model being used.

Two of the most popular object detection architectures, YOLO (You Only Look Once) and SSD (Single Shot MultiBox Detector), require specific annotation formats for optimal performance.

3.3.3 Introduction to YOLO and SSD

3.3.3.1 Object Detection Models

Object detection aims to identify and locate objects in images. Two widely used deep learning-based object detection models are:

- YOLO (You Only Look Once): Efficient real-time detection using a single forward pass.
- SSD (Single Shot MultiBox Detector): Detects objects at multiple scales using multi-feature maps.

Both models balance accuracy and speed for real-time applications.

3.3.4 YOLO (You Only Look Once)

3.3.4.1 Overview

YOLO is a real-time object detection system introduced in 2015 by Joseph Redmon et al. In contrast to conventional object identification techniques, YOLO approaches detection as a unified regression issue, concurrently predicting bounding boxes and class probabilities.

3.3.4.2 Why is YOLO Popular?

YOLO gained prominence due to its:

- High Speed: Processes images in real-time with minimal latency.
- Accurate Detection: Achieves high precision while maintaining fast inference.
- Strong Generalization: Effectively detects objects in unseen environments.
- Open-Source Availability: Widely accessible and continuously improved by the research community.

These features make YOLO one of the most efficient and widely used object detection models in real-world applications.

3.3.4.3 Training and Fine-Tuning YOLO Models

Training a YOLO model involves feeding it labelled data and optimizing its parameters to accurately detect objects. The process includes:

- Dataset Preparation: Images with annotations in YOLO format.
- Model Selection: Pre-trained YOLO variants (YOLOv4, YOLOv5, YOLOv8) or training from scratch.
- Hyperparameter Tuning: Adjusting learning rate, batch size, anchor boxes, etc.
- Loss Function Optimization: Minimizing localization, confidence, and classification loss.
- Augmentation & Regularization: Preventing overfitting using data augmentation techniques.

Fine-Tuning a YOLO Model

Fine-tuning involves transfer learning, where a pre-trained YOLO model (trained on large datasets like COCO) is adapted to a specific task. This includes:

- Freezing early layers to retain feature extraction while training the final layers on new data.

- Adjusting the number of classes in the model's architecture.
- Training with a lower learning rate to prevent catastrophic forgetting.
- Fine-tuning significantly reduces training time while improving performance on domain-specific datasets.

3.3.5 SSD (Single Shot MultiBox Detector)

3.3.5.1 Overview

SSD is an object detection algorithm introduced in 2016 by Wei Liu et al. It achieves over 74% mean Average Precision (mAP) at 59 FPS on datasets like Pascal VOC and COCO.

3.3.5.2 Understanding the SSD Name

The name Single Shot MultiBox Detector reflects the key features of the architecture:

- Single Shot: Performs localization and classification in a single forward pass.
- MultiBox: Uses a bounding box regression technique introduced by Szegedy et al.
- Detector: Simultaneously detects and classifies objects.

3.3.5.3 Training and Fine-Tuning SSD Models

Training SSD follows a similar process to YOLO:

- Dataset Preparation: Images with annotations in Pascal VOC or COCO format.
- Model Selection: Pre-trained SSD models or custom training.
- Multi-Scale Feature Maps: Improves detection of small and large objects.
- Optimization Techniques: Uses default anchor boxes and prior boxes for better localization. SSD offers an excellent trade-off between speed and accuracy, making it a popular choice for real-time object detection applications.

In this work, we adapt the pre-trained SSD300 model to operate on 512×512 input images. This adaptation involves resizing the input images, reconfiguring the anchor generator to better suit the higher resolution, and modifying the detection head to support the increased number of anchors and classes. The resulting architecture, which we refer to as an SSD512-like model, leverages the robust feature extraction of SSD300 while providing improved performance on higher-resolution images.

Fine-tuning SSD models involves:

- Freezing early layers to retain general features.
- Adjusting anchor box sizes to match object scales.
- Using a lower learning rate to prevent performance degradation.

SSD provides an excellent trade-off between speed and accuracy, making it widely used in real-time applications.

3.4 Object Tracking Methodology

3.4.1 Introduction to Object Tracking

Object tracking is a key feature of computer vision that includes recognizing and following objects throughout a sequence of frames in a movie. In the field of autonomous driving, object tracking enables vehicles to monitor the positions and movements of various entities, such as other vehicles, pedestrians, and obstacles, thereby ensuring safe and efficient navigation.

3.4.1.1 Overview of Object Tracking in Autonomous Driving

In autonomous vehicles, object tracking systems continuously analyse data from sensors like cameras, LiDAR, and radar to detect and monitor objects in the vehicle's vicinity. These systems employ algorithms that estimate or predict the positions and trajectories of moving objects, even in the presence of occlusions or dynamic changes in the environment

3.4.1.2 Importance of Tracking in Real-Time Perception Systems

The success of real-time perception systems in autonomous vehicles rely heavily on accurate object tracking to make instantaneous decisions. The ability to track objects in real-time ensures that the vehicle can respond promptly to sudden changes, such as a car braking abruptly or a pedestrian stepping onto the road. This responsiveness is vital for maintaining safety and comfort.

3.4.2 Integration of ByteTrack and StrongSORT with YOLO Outputs

The integration of advanced tracking algorithms such as ByteTrack and StrongSORT with YOLO (You Only Look Once) object detection models significantly improves the accuracy and reliability of multi-object tracking systems. This integration utilizes the efficient object detection capabilities of YOLO to deliver accurate inputs for tracking algorithms, facilitating real-time performance in applications like autonomous driving and surveillance.

- **ByteTrack**

ByteTrack is an algorithm that facilitates the association of detected objects across video frames through a tracking-by-detection approach. This approach employs a combination of high-confidence and low-confidence detections, facilitating a more thorough tracking process, particularly in situations where objects might be partially obscured or display differing detection scores.

Tracking-by-detection approach.

In the tracking-by-detection paradigm, object tracking is performed by first detecting objects in each frame and then associating these detections across frames to form trajectories. ByteTrack enhances this approach by considering every detection box, thereby improving the association process and reducing missed detections.

Strengths in associating object identities.

ByteTrack excels in maintaining consistent object identities over time, even in challenging conditions such as crowded scenes or rapid object movements. By leveraging a comprehensive set of detections, it reduces identity switches and enhances the overall robustness of the tracking system.

- **StrongSORT**

StrongSORT builds upon the DeepSORT algorithm by integrating advanced appearance descriptors and motion models. It employs deep learning-based re-identification (Re-ID) networks to extract robust feature embeddings, facilitating more accurate association of detections across frames.

Deep association metric for tracking.

The algorithm utilizes a deep association metric that combines appearance features and motion information. This metric enhances the discriminative power of the tracker, enabling it to distinguish between similar-looking objects and maintain accurate trajectories.

Role of re-identification (Re-ID) networks in object tracking.

ReID (Re-Identification) weight is a word used in connection to object tracking algorithms. It indicates the weight assigned to the ReID (Re-Identification) loss term in the tracking algorithm's objective function. The Re-ID networks play a crucial role in StrongSORT by providing robust appearance features that help in reidentifying objects across non-consecutive frames or different camera views. This capability is essential for handling occlusions and long-term tracking scenarios.

- **Integration with YOLO**

YOLO models are renowned for their real-time object detection capabilities, producing bounding boxes and class probabilities for objects within each frame. These detections serve as inputs for tracking algorithms like ByteTrack and StrongSORT, which then associate the detections across frames to generate object trajectories.

Workflow of integrating YOLO with ByteTrack and StrongSORT.

- Detection: Each video frame is passed through a YOLO model, which quickly identifies objects and produces bounding boxes along with associated confidence scores.

- Filtering: The raw detections from YOLO are filtered to remove any low-confidence results. This helps to eliminate false positives and ensure only reliable detections are forwarded for tracking.
- Association: The filtered detections are then passed to the tracking algorithm.
  - ByteTrack: Uses a tracking-by-detection strategy to associate detections across consecutive frames, ensuring that object identities are maintained even when detections vary in confidence.
  - StrongSORT: Leverages both motion cues and deep appearance features (via Re-ID networks) to robustly match objects between frames.
- Trajectory Management: After association, the tracker manages object trajectories over time. This involves:
  - Updating existing tracks: Continuously refining the trajectory of each tracked object.
  - Handling new entries and exits: Initiating new tracks for entering objects and removing tracks for objects that leave the frame.

3.5 Performance Evaluation: ByteTrack vs. StrongSORT

3.5.1 Metrics for Tracking Evaluation

Multiple Object Tracking Accuracy (MOTA)

MOTA analyzes total tracking performance by considering false positives, false negatives, and identity switches, offering a complete evaluation of a tracker's accuracy in preserving proper object trajectories. A MOTA score of 1 indicates excellent system accuracy, while a score around zero or below signifies poor accuracy. MOTA can be calculated using:

$$\text{MOTA} = 1 - \frac{\sum_t (FN_t + FP_t + IDS_t)}{\sum_t GT_t}$$

where:

- FP_t is the number of false positives in frame t.
- FN_t is the number of false negatives (missed detections) in frame t.
- IDS_t is the number of identity switches in frame t.
- GT_t is the number of ground truth objects in frame t.

Multiple Object Tracking Precision (MOTP)

MOTP assesses the precision of the tracker by measuring the alignment between predicted object positions and ground truth positions. In this metric, a lower MOTP indicates more precise localization, if the value approaches 0, the system's accuracy is outstanding, whereas a value near 1 reflects inadequate system accuracy. MOTP can be calculated using:

$$\text{MOTP} = \frac{\sum_{i,t} d_t^i}{\sum_t c_t}$$

where:

- c_t is the number of matches (correct associations) in frame t.
- $d_{i,\,t}$ is the distance (or overlap error) between the ground truth and the corresponding predicted bounding box for the i^{th} match in frame t.

ID switches and tracking robustness.

ID switches occur when a tracker mistakenly assigns a new identity to a previously tracked object. Minimizing ID switches is crucial for maintaining the robustness of the tracking system, as frequent switches can lead to fragmented trajectories and unreliable tracking data.

Chapter 4 presents a comparative evaluation highlighting the distinct trade-offs between the two trackers when used with YOLO detections. While StrongSORT delivers superior tracking accuracy and precision, ByteTrack stands out for its consistency by exhibiting considerably fewer ID switches and false positives. Overall, integrating either ByteTrack or StrongSORT with YOLO significantly enhances multi-object tracking systems, ensuring both accurate and robust real-time tracking performance.

3.6 CNN for Vehicle Control

Introduction to Deep Learning for Autonomous Vehicle Control

Deep learning has significantly transformed the landscape of autonomous vehicle control, facilitating the development of end-to-end driving systems that acquire knowledge directly from data. In contrast to conventional rule-based methods that depend on the explicit programming of driving regulations, deep learning techniques, especially Convolutional Neural Networks (CNNs), possess the capability to autonomously identify pertinent features from sensory data to inform driving decisions. This allows for more adaptable and robust vehicle control in complex environments.

Traditional rule-based systems depend on predetermined rules and models, making them less adaptable in unanticipated settings. In contrast, deep learning models may learn from enormous amounts of driving data, identifying complex patterns and behaviours that rule-based systems can ignore. This leads to enhanced performance in activities like as lane holding, obstacle avoidance, and decision-making under ambiguity.

3.6.1 CNN Architecture for Vehicle Control

Input Data: Sensor and Camera Feeds

CNNs process visual inputs from sensors and cameras to perceive the driving environment. In simulations like CARLA, these inputs include images capturing road conditions, traffic signs, and other vehicles. The CNN learns to interpret these inputs to understand the vehicle's surroundings.

Feature Extraction for Decision-Making

Through convolutional layers, CNNs extract features such as lane markings, road edges, and obstacles. These features are crucial for making informed driving decisions. For instance, identifying lane boundaries helps in maintaining lane discipline, while recognizing obstacles is essential for collision avoidance.

Output Layer: Steering, Acceleration, and Braking Commands

The CNN's output layer translates the extracted features into control commands, including steering angles, acceleration, and braking. This end-to-end approach allows the vehicle to autonomously navigate by continuously adjusting its actions based on real-time sensory input.

3.6.2 Training CNN for Autonomous Vehicle Control in CARLA

Dataset Collection

Training a CNN requires a robust dataset that captures a wide range of driving scenarios. In the CARLA simulator, data is collected by recording vehicle movements under various conditions. During manual driving sessions in the environment, key control inputs such as steering movements, throttle activations, and braking actions are meticulously recorded. Consequently, the dataset includes video frames, timestamps, image filenames, and the corresponding records of steering, throttle, and brake applications, providing the comprehensive examples needed for effective model training.

Model Training and Optimization

In this phase, the CNN can be trained using supervised learning. The process begins by loading labelled data, images paired with corresponding control signals (steering, throttle, and brake) from a CSV file generated in the CARLA simulator. Custom preprocessing functions handle tasks such as image resizing, normalization, and conversion to the appropriate colour space.
The pre-processed data is organized into training and validation sets using TensorFlow's 'tf.data.Dataset' API, which efficiently batches, shuffles, and prefetches the data for optimized performance. The CNN architecture, based on the NVIDIA PilotNet model, employs a series of convolutional layers to extract features from the input images, followed by dense layers that output continuous control commands.
The model is compiled with the Adam optimizer and employs a mean squared error (MSE) loss function, which measures the discrepancy between the predicted and actual control actions. Backpropagation is then used to iteratively minimize this loss, thereby refining the model's

predictions. The training process is further enhanced with a TQDM callback, providing real-time progress updates.

While this implementation utilizes supervised learning, reinforcement learning remains a viable alternative, where the model would learn optimal behaviours through trial and error by receiving rewards for desirable actions.

Applications for CNN-Based Decision-Making in Real-World Driving

- Autonomous Lane Keeping and Path Following: CNNs enable vehicles to maintain lanes and follow paths by accurately interpreting road features.
- Traffic Light and Pedestrian Recognition: Deep learning models can detect traffic signals and pedestrians, enhancing safety and compliance with traffic laws.
- Adaptive Cruise Control Using Deep Learning: CNNs facilitate adaptive cruise control by understanding the speed and distance of preceding vehicles, allowing for smooth and responsive speed adjustments.

Incorporating CNNs into vehicle control systems offers significant advancements over traditional methods, providing more flexible and efficient autonomous driving capabilities.

Chapter 4: Development (Implementation & System Design)

Data Collection from CARLA

The CARLA simulation environment provides a rich, realistic urban setting for autonomous driving research. In this section, we detail the steps and methodologies used to extract and preprocess data focusing on video footage extraction for object detection and tracking, as well as the preprocessing pipeline for models like YOLO and StrongSORT.

4.1.1 Extracting Video Footage from CARLA

Overview of CARLA's Simulation Environment

- Realistic Urban Scenarios: CARLA offers diverse and dynamic urban settings that include varied weather conditions, traffic patterns, pedestrian behaviours, and multiple sensor configurations. This realism is crucial for developing robust detection and tracking models.
- Flexible Sensor Setup: The platform supports multiple types of sensors (cameras, LiDAR, etc.) which can be configured to capture high-fidelity data from different perspectives.

Recording Video Footage for Object Detection and Tracking

- Dynamic Scene Capture: Within the simulation, cameras are deployed to record continuous video footage capturing the movement of vehicles, pedestrians, and other dynamic objects. This footage forms the primary dataset for training object detection and tracking systems.
- Synchronization with Simulation Events: Video recordings are often synchronized with simulation events (e.g., traffic light changes or pedestrian crossings) to ensure that the dataset captures a wide range of real-world scenarios.

4.1.2 Preprocessing the Data for Object detection

In the context of modern object detection tasks, preprocessing the dataset is a critical step to ensure that the data fed into deep learning models is of high quality and properly formatted. In this work, we use the Bdd100k dataset, a diverse collection of urban driving scenes as the foundation for training object detection models based on frameworks such as YOLO and SSD. This section details the systematic approach adopted for data cleaning, formatting, and annotation.

4.1.2.1 Data Cleaning and Formatting

Data cleaning and formatting form the initial phase in our preprocessing pipeline. The following outlines the following procedures:

- Frame Extraction and Consistency: The Bdd100k dataset comprises sequences of images captured in real-world driving conditions. Our preprocessing script begins by verifying the integrity of each frame extracted from the videos. Frames are checked for issues such as blurriness, incorrect aspect ratios, or incomplete scene captures. Automated quality control measures help in discarding corrupt frames or those that fail to meet our preset quality thresholds.

- Image Resizing and Normalization: Given that the YOLO and SSD frameworks require inputs of standardized dimensions, every image is resized to the target resolution (e.g., 416×416 for YOLO). This step ensures that the spatial dimensions are consistent across the dataset. Further, pixel intensity values are normalized, scaling the raw image data typically into the range [0, 1]. Normalization reduces the effects of lighting variations and accelerates convergence during training.
- Data Augmentation: The augmentation techniques such as random cropping, flipping, and rotation can be applied to increase dataset variability. This optional step is implemented in the code to further improve the model's robustness against real-world variability.

4.1.2.2 Annotation for Object Detection

The effectiveness of object detection algorithms is highly dependent on the precision and quality of the annotations used during training. In this study, a dual-faceted annotation strategy was employed, encompassing both the detailed labelling of bounding boxes and the systematic partitioning of the dataset.

Bounding Box Labelling in YOLO and SSD Format

For the annotation process, each image within the Bdd100k dataset was meticulously labeled with bounding boxes that clearly delineate objects of interest including bike, bus, car, motor, person, rider, traffic light, traffic sign, train, and truck. Two distinct formats were utilized to accommodate different object detection frameworks:

- YOLO Format: In this format, each annotation is recorded in a text file with every line representing an individual object. The annotations include the object's class label and the normalized bounding box coordinates (center_x, center_y, width, height). The normalization is performed relative to the image dimensions, which is critical for YOLO's grid-based prediction mechanism.
- SSD Format: For SSD, annotations are typically structured in XML or JSON formats. These files not only capture the coordinates of the bounding boxes but also include additional metadata such as object class information and, where applicable, confidence scores. This format is designed to meet the specific requirements of SSD implementations.

Preparing Dataset Splits (Training, Validation, Testing)

In addition to precise annotation, the dataset was systematically divided into three key subsets to support robust model training and evaluation:

- Training Set: The majority of the data was allocated to the training set, ensuring that the models were exposed to a broad and representative range of scenarios depicted in the dataset. This balanced approach aimed to capture the diversity inherent in the dataset.
- Validation Set: A separate subset was designated for validation purposes. This set played a crucial role in tuning model hyperparameters and implementing early stopping criteria, thereby helping to prevent overfitting during the training process.
- Testing Set: The testing set was reserved exclusively for the unbiased assessment of the trained models. This subset provided a critical benchmark to evaluate the generalization capabilities of the models on data that was not encountered during training.

Great care was taken to ensure that the statistical distribution of object classes and scene types was preserved across all splits. This was achieved through a stratified sampling methodology, which minimized potential biases and contributed to more reliable performance assessments.

4.2 Object Detection: YOLO vs SSD

This section provides a comprehensive comparison between YOLO and SSD for object detection, with a focus on their inference performance and experimental outcomes using the CARLA dataset. The discussion begins with an in-depth analysis of each model's implementation on the CARLA dataset, followed by a detailed performance evaluation based on various metrics.

4.2.1 Object Detection: YOLO Performance

This section presents an analysis of the YOLO model's performance on the BDD100k dataset, focusing on both qualitative and quantitative metrics.

YOLO Performance Analysis

- **Confusion Matrix:** The confusion matrix provides a class-by-class breakdown of correct and incorrect predictions for YOLO. High diagonal values indicate strong performance in detecting the respective classes, while off-diagonal entries reveal instances of misclassification. Classes such as car and truck show a high number of true positives, reflecting YOLO's reliability in detecting larger vehicles. Meanwhile, classes with fewer instances like train display a relatively lower count, suggesting that performance could be further improved with more targeted data or advanced fine-tuning.

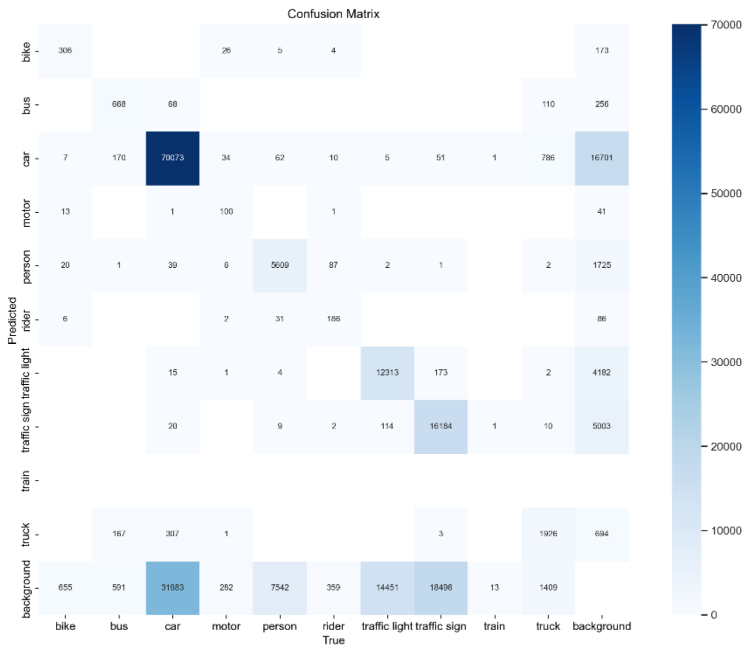


Fig 4.1 Confusion Matrix

- **Normalized Confusion Matrix**

  The normalized confusion matrix offers a clearer view of prediction accuracy relative to each class. It highlights the proportion of correct predictions out of all predictions made for a particular class. Here, YOLO demonstrates strong recall for classes with abundant training examples (e.g., *car*), while classes with fewer samples, such as *bike* or *train*, exhibit lower normalized values.

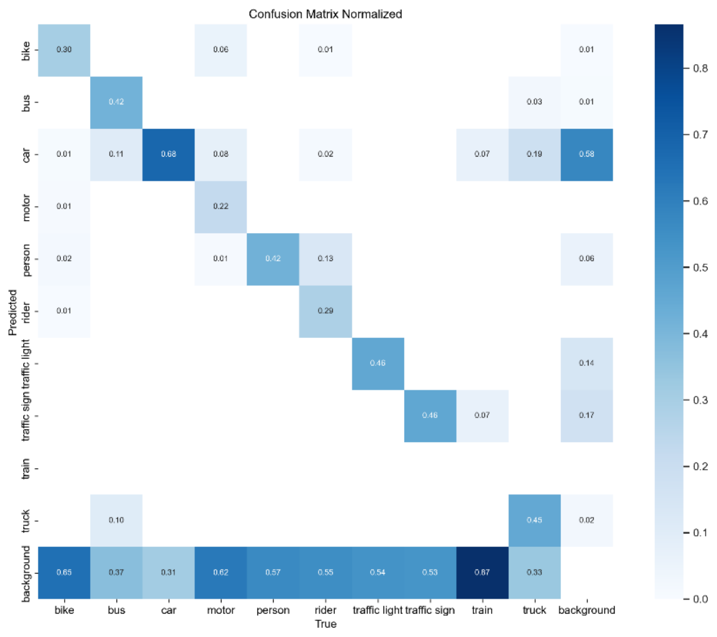


Fig 4.2 Normalized Confusion Matrix

- **F1-Confidence Curve**

  The F1-Confidence curve depicts how the F1 score (the harmonic mean of precision and recall) increases with different confidence thresholds. YOLO gets a high F1 score of 0.47 at a confidence threshold of 0.207 for all classes, demonstrating a decent balance between precision and recall. Class-specific curves reveal that objects such as *bus* and *car* can achieve higher F1 scores due to more consistent detections and lower rates of false positives.

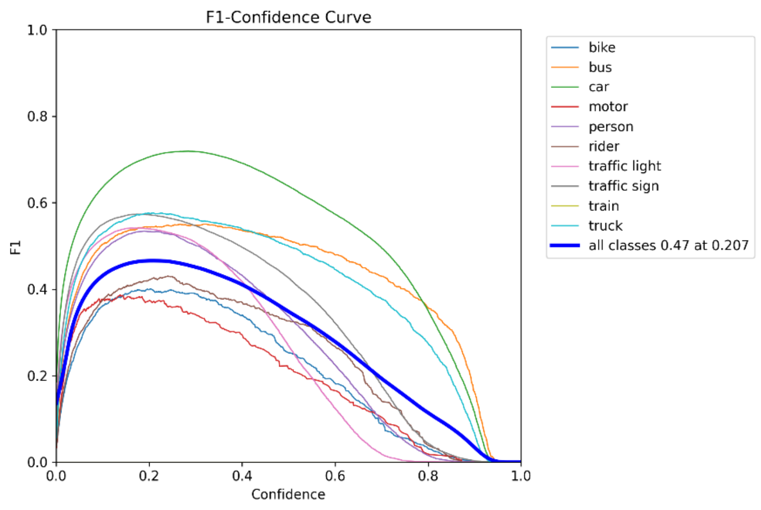


Fig 4.3 F1-Confidence Curve

- **Precision-Confidence Curve**

  The precision-confidence curve shows that YOLO maintains relatively high precision across a broad range of confidence thresholds, especially for frequent classes like *car* and *bus*. The curve for smaller or less frequent objects (e.g., *bike*, *motor*) starts to dip more sharply as the confidence threshold decreases, highlighting the challenges of detecting smaller or rarer classes with absolute precision.

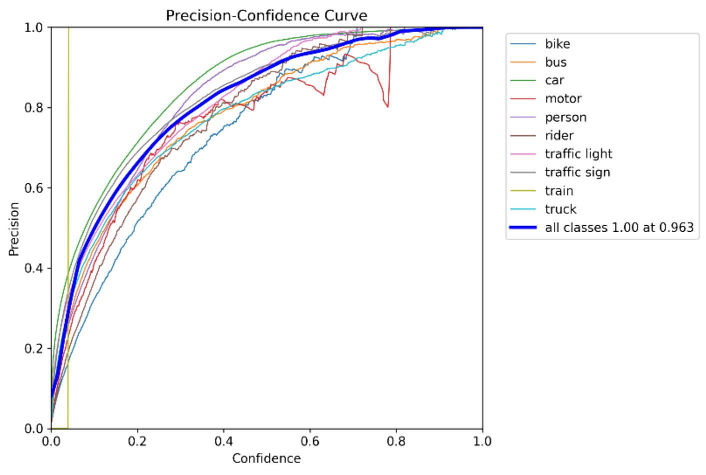


Fig 4.4: Precision-Confidence Curve

- **Precision-Recall (PR) Curve**

  The PR curve offers further insight into the trade-off between precision and recall across varying thresholds. YOLO achieves a mean average precision (mAP) of 0.444 at an IoU threshold of 0.5, which, while respectable, indicates room for improvement. Class specific PR curves highlight that *car* and *bus* maintain higher precision across a wide range of recall values, whereas *motor* and *bike* drop off more sharply.

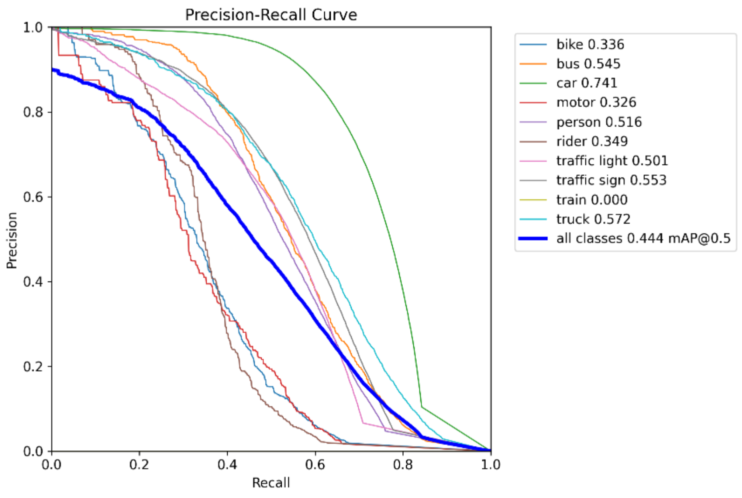


Fig 4.5: Precision-Recall (PR) Curve

- **Recall-Confidence Curve**

  The recall-confidence curve depicts how recall changes with varying confidence thresholds. As the confidence threshold decreases, YOLO detects more objects overall, thus increasing recall but potentially introducing more false positives. Notably, YOLO attains a maximum recall of 0.65 across all classes at a confidence threshold of 0.0 (i.e., detecting nearly every instance but with many low-confidence detections).

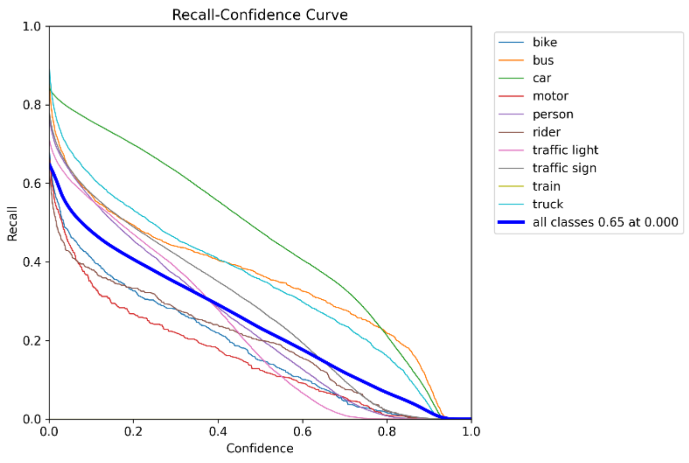


Fig 4.6: Recall-Confidence Curve

- **Training and Validation Losses, Metrics**

  Figure 4.7 displays the evolution of training and validation losses, along with key metrics such as precision and recall over multiple epochs. The smooth curves suggest stable convergence, with the model steadily improving as training progresses. The final mAP values at 0.5 and 0.5–0.95 IoU thresholds reflect the model's capacity to detect objects of various scales.

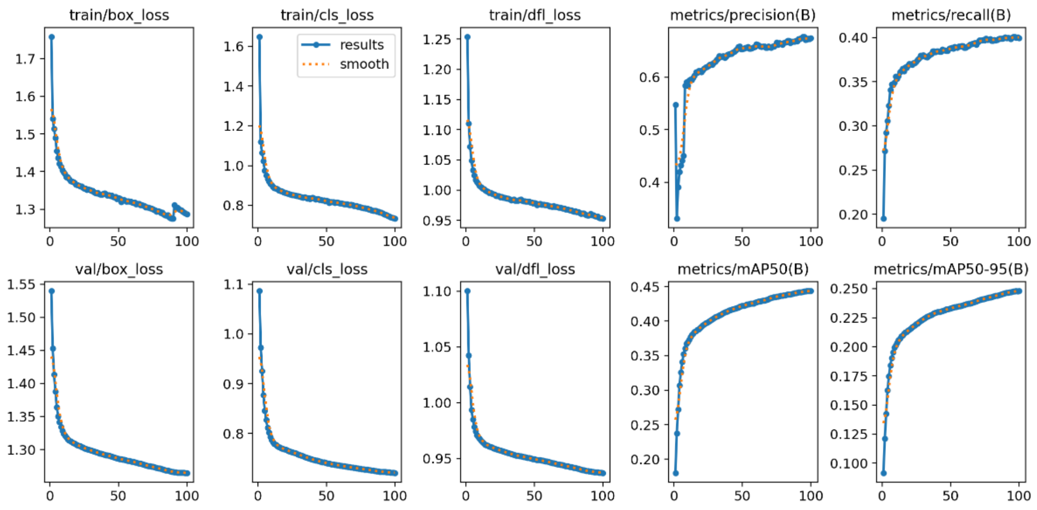


Fig 4.7: Training and Validation Losses, Metrics

4.2.2 Preliminary Observations

1. **Robustness in Common Classes:**

   YOLO excels in detecting objects like *car* and *bus*, likely due to the abundance of these classes in the BDD100k dataset. High precision and F1 scores for these categories confirm YOLO's reliability in typical urban driving scenes.

2. **Challenges in Rare or Small Objects:**

   Performance for classes such as train and bike is relatively lower, as evidenced by confusion matrix entries and class-specific curves. This suggests a need for either class-specific data augmentation or specialized strategies (e.g., focal loss, higher-resolution input) to improve detection.

Balanced Precision and Recall: The F1 and precision-recall curves indicate a reasonable trade-off between precision and recall for most classes. YOLO's real-time capabilities, combined with its balanced detection metrics, make it well-suited for applications requiring both speed and accuracy.

4.2.2 Object Detection: SSD Performance

This section provides a comprehensive analysis of the SSD model's performance on the BDD100k dataset. The evaluation focuses on quantitative metrics specifically, the distributions of precision, recall, and mAP50 and incorporates qualitative insights derived from the results.

1. Distribution of Precision, Recall, and mAP50

An analysis of per-image performance across 10,000 validation images reveals the following average metrics for the SSD model:

- Precision: Mean ≈ 0.068
- Recall: Mean ≈ 0.771
- mAP50: Mean ≈ 0.053

These values indicate that, while the SSD model can capture a substantial portion of the ground-truth objects (as evidenced by high recall), it struggles to maintain high precision or achieve robust average precision. The histograms in Figures 4.8, 4.9, and 4.11 further substantiate these findings:

- Precision**:** Most images exhibit precision values below 0.10, suggesting that the model generates a significant number of false positives.

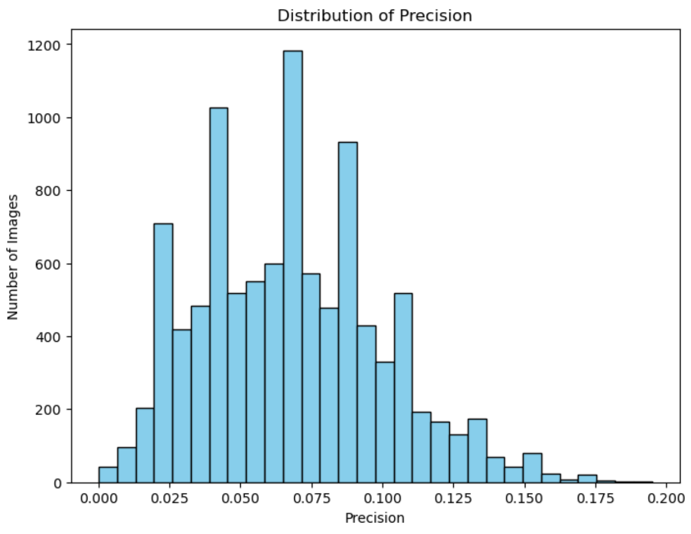


Fig 4.8: Precision Histogram

- Recall: The majority of images peak in the 0.75–1.0 range, indicating that many ground-truth objects are detected. However, a subset of images shows recall values exceeding 1.0, potentially due to overlapping predictions or the handling of partial matches.

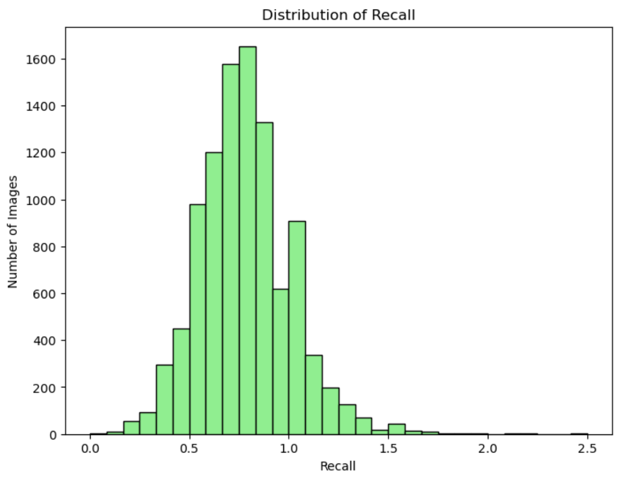


Fig 4.9: Recall Histogram

- mAP50: The clustering of mAP50 values predominantly below 0.10 highlights deficiencies in localization accuracy and confidence scoring, despite the model's ability to detect numerous objects.

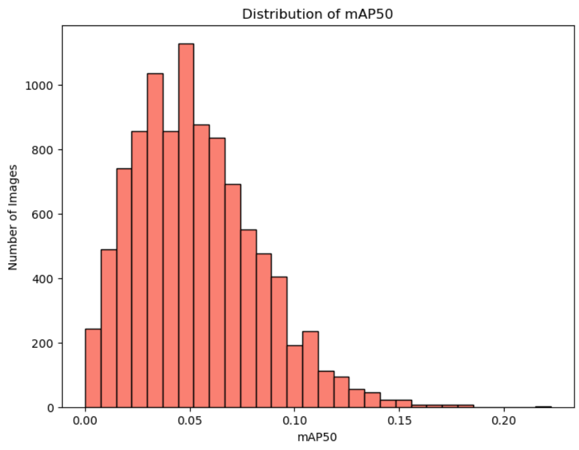


Fig 4.10: mAP50 Histogram

2. Precision vs. Recall Scatter Plot

Figure 4.11 shows a scatter plot of precision versus recall for individual images, offering deeper insight into the trade-offs inherent in the SSD model's predictions:

- **High Recall, Low Precision:** A substantial cluster of points in the upper-left quadrant indicates that, although the model detects most objects (high recall), it simultaneously produces numerous false positives (low precision).
- **Occasional Balanced Cases:** A minority of images display more moderate performance (e.g., precision around 0.1 and recall between 0.5 and 0.8), suggesting that SSD can achieve a balanced trade-off under certain conditions.
- **Outliers:** Some observations exhibit recall greater than 1.0 alongside low precision, pointing to potential anomalies in dataset annotations or the evaluation methodology.

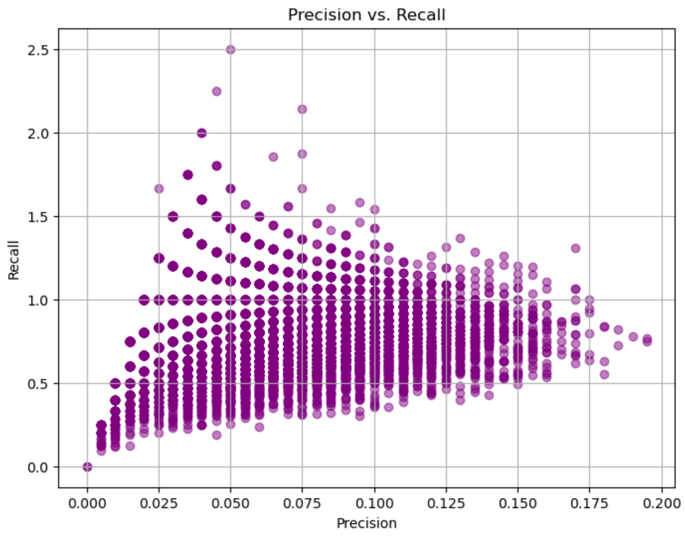


Fig 4.11: Precision vs. Recall Scatter Plot

4.2.2.3 Inference and Performance Evaluation

Comparison with YOLO Observations

When contrasted with the performance of YOLO which demonstrated a more balanced trade-off between precision and recall (mean mAP ~0.444 at an IoU threshold of 0.5) the following distinctions emerge for SSD:

- **Lower Average Precision:** The mean precision of approximately 0.068 is significantly lower than YOLO’s, indicating that SSD is more susceptible to false positives under the current configuration.
- **Comparable or Higher Recall:** With a mean recall of about 0.771, SSD generates a larger volume of bounding boxes, thereby detecting more ground-truth objects but at the cost of precision.

4.2.3.2 Experimental Results on CARLA Dataset

Experimental results on the CARLA dataset reveal how each model behaves under real-time conditions. The analysis covers:

- Speed, Accuracy, and Robustness: YOLO consistently outperformed SSD in terms of processing speed and exhibited superior adaptability to dynamic scenarios. The results also highlight how each model’s tracker performed across various driving conditions.
- Real-Time Performance: The experiments underscore the importance of real-time processing in autonomous driving applications, with YOLO providing a clear advantage in this regard.

Justification of YOLO as the Superior Model

Based on the experimental outcomes, YOLO emerged as the superior model for the given application. The key justifications include:

- Real-Time Processing Advantage: YOLO’s higher FPS ensures that it can efficiently process frames in real-time, a critical factor in dynamic environments.
- Adaptability: Its ability to adapt to rapidly changing driving scenarios makes it more suitable for autonomous systems.

Limitations of SSD

While SSD shows promise in detection accuracy, it is constrained by:

- Higher Computational Cost: SSD requires more computational resources, which can hinder its deployment in real-time applications.
- Challenges in Small Object Detection: SSD's performance tends to decline when detecting smaller objects, further limiting its applicability in complex driving scenarios.

In summary, while both models have their merits, the comprehensive evaluation clearly indicates that YOLO offers a more balanced solution for object detection in autonomous driving scenarios, combining superior speed with robust accuracy.

4.3 Object Tracking: ByteTrack vs. StrongSORT

ByteTrack and StrongSORT are two widely used tracking algorithms that improve multi-object tracking by associating detections across consecutive frames.

4.3.1 Implementation of ByteTrack

4.3.1.1 Overview of ByteTrack

ByteTrack is a tracking-by-detection algorithm that associates object detections across video frames to maintain consistent identities over time. Unlike conventional object trackers, ByteTrack improves upon tracking precision by handling both high-confidence and low-confidence detections, leading to enhanced association and reduced identity switches.

How ByteTrack Tracks Objects Using Bounding Box Association

ByteTrack operates by:

- Detection: It receives bounding box outputs from an object detection model (e.g., YOLO).
- High-confidence filtering: The detections are filtered based on a confidence threshold. High-confidence detections are used for initial tracking.
- Low-confidence association: Instead of discarding low-confidence detections, ByteTrack attempts to associate them with existing tracked objects.
- IoU Matching: The tracker employs Intersection over Union (IoU) matching to measure spatial overlap and track object movement across frames.
- Kalman Filter for Motion Prediction: It predicts the next location of objects using a Kalman Filter, refining object trajectories for smoother tracking.

4.3.1.2 Training ByteTrack with YOLO Outputs

ByteTrack is integrated with YOLO-based detections to enhance tracking consistency. The process follows these steps:

- **Object Detection with YOLO**
  - The YOLO model detects objects in individual frames and outputs bounding boxes with confidence scores.
  - The detections are stored as (x1, y1, x2, y2, confidence, class_id), representing the bounding box coordinates, confidence score, and object class.

- **Data Preprocessing**
  - The raw frame is resized using letterbox resizing, ensuring aspect ratio preservation before feeding it into YOLO.
  - The code normalizes the bounding box coordinates to match the new image dimensions.
- **Passing Detections to ByteTrack**
  - The YOLO detections are processed and sent to ByteTrack.
  - Where detections contain the bounding box coordinates, class IDs, and confidence scores.
- **Tracking and Identity Assignment**
  - The tracker assigns unique IDs to objects and maintains them across frames.
  - The IoU matching method ensures that objects retain their identities even when occlusions occur.
  - The Kalman Filter refines predictions, preventing abrupt object disappearance due to detection inconsistencies.

The ByteTrack tracker, once trained on YOLO outputs, successfully maintains object identities across multiple frames.

4.3.1.3 Performance in CARLA Simulation

The ByteTrack tracker was tested within CARLA, to evaluate its real-time tracking capabilities. The evaluation focused on three key aspects:

- Accuracy in Maintaining Object Identities:
  - ByteTrack showed a strong ability to reduce ID switches, ensuring that the same object retained its identity across multiple frames.
  - Objects moving at high speeds (such as cars) were tracked consistently with minimal fragmentation.
  - Occlusion handling was improved due to ByteTrack's low-confidence detection integration, reducing missed detections.
- Computation Efficiency and FPS Performance:
  - ByteTrack was able to process tracking updates at real-time speeds.
  - The frame processing time was significantly lower compared to StrongSORT, leading to a higher FPS (frames per second), making it suitable for real-time applications.
- Handling of Occlusions and Fast Motion:
  - ByteTrack demonstrated resilience to occlusions, as it used previous frame detections to re-identify objects that momentarily disappeared.
  - The algorithm was effective in tracking fast-moving vehicles in CARLA, making it a viable choice for autonomous navigation and robotic vision applications.

4.3.2 Implementation of StrongSORT

StrongSORT is an advanced object tracking algorithm that extends the capabilities of DeepSORT by integrating a robust appearance-based tracking system using a Re-Identification (Re-ID) network. This approach enhances object identity preservation, making it particularly useful in complex tracking scenarios.

4.3.2.1 Overview of StrongSORT

StrongSORT is an enhancement of DeepSORT, a tracking-by-detection algorithm that improves object identity association by leveraging appearance features in addition to motion cues.

How StrongSORT Works:

- YOLO-Based Object Detection:
  - StrongSORT receives bounding box detections from a YOLO model.
  - Each detection consists of (x1, y1, x2, y2, confidence, class_id).
- Motion-Based Tracking (DeepSORT Component):
  - It uses Kalman Filtering to predict object locations between frames.
  - The Hungarian Algorithm matches detected objects with previous frame identities.
- Re-Identification (Re-ID) Network for Robust Identity Matching:
  - Unlike traditional motion-based tracking, StrongSORT introduces appearance-based re-identification.
  - A deep Re-ID model extracts feature embeddings from detected objects to improve tracking consistency.
  - Even if an object disappears temporarily due to occlusion, the Re-ID network helps re-associate it accurately.
- IoU-Based and Deep Feature Matching:
  - StrongSORT combines IoU-based association with deep feature extraction, making it more robust against identity switches.

4.3.2.2 Training StrongSORT with YOLO Outputs

StrongSORT requires YOLO detections to function as an input. The following steps describe the training and tracking process.

- Object Detection with YOLO:
  - The YOLO model is used to detect objects in each frame.
  - The detections include bounding boxes, class labels, and confidence scores.
  - The model is loaded and set to evaluation mode
- Feature Extraction using Re-ID Model:
  - The Re-ID model extracts feature embeddings for detected objects
  - This helps in distinguishing visually similar objects and re-identifying lost objects after occlusions.

- Detection Processing and Tracking:
  - The YOLO detections are passed to StrongSORT for tracking
  - Detections contain bounding box coordinates, class IDs, and confidence scores.
  - The tracker associates new detections with existing tracked objects using both motion and appearance-based features.
- Handling Object Disappearance:
  - If an object disappears for a few frames due to occlusion, the Re-ID model recovers its identity when it reappears.
  - This is particularly beneficial in crowded or high-occlusion environments.

4.3.2.3 Performance in CARLA Simulation

The StrongSORT tracker was tested within CARLA, the following performance aspects were evaluated:

1. Identity Preservation and ID Switches
   - StrongSORT achieved higher identity preservation compared to ByteTrack.
   - However, it exhibited more ID switches, particularly in fast-moving objects.
   - The deep appearance matching provided robust tracking for similar-looking objects.
2. Accuracy in Complex Environments
   - StrongSORT was more accurate in distinguishing objects compared to ByteTrack.
   - It worked well in scenarios with overlapping objects, such as pedestrians crossing roads or cars parked close together.
3. Computational Cost and FPS Performance
   - Due to the Re-ID model, StrongSORT requires more computational power than ByteTrack.
   - This led to lower FPS:
     - ByteTrack FPS: ~6.3 FPS
     - StrongSORT FPS: ~3.8 FPS
   - The tracking update time was significantly higher in StrongSORT due to deep feature extraction overhead.
4. Occlusion Handling
   - StrongSORT outperformed ByteTrack when handling occluded objects.
   - It could re-identify objects after disappearing for multiple frames.

4.3.3 Performance Comparison: ByteTrack vs. StrongSORT

Object tracking performance is evaluated using key metrics such as Multiple Object Tracking Accuracy (MOTA), Multiple Object Tracking Precision (MOTP), and robustness in handling occlusions and object re-identification. The following sections compare ByteTrack and StrongSORT based on their tracking performance and computational efficiency.

4.3.3.1 Tracking Accuracy Metrics

The table below presents a comparative analysis of ByteTrack and StrongSORT based on multiple evaluation parameters:

Metric	**ByteTrack**	**StrongSORT**
Total Frames Processed	1100	1100
MOTA (Accuracy)	0.572	0.754
MOTP (Precision)	0.002	0.005
ID Switches	50	144
False Positives (FP)	2	365
False Negatives (FN)	3902	1766

Key Observations:

- MOTA (Accuracy): StrongSORT achieves higher tracking accuracy (75.4% vs. 57.2%), indicating that it successfully detects and tracks more objects.
- MOTP (Precision): StrongSORT also records a higher localization precision, despite both trackers having relatively low absolute values.
- ID Switches: ByteTrack demonstrates better identity consistency, with fewer ID switches (50 vs. 144), ensuring that objects maintain their identities over multiple frames.
- False Positives & False Negatives: ByteTrack produces fewer false positives (2 vs. 365), but it also misses more objects (high FN: 3902 vs. 1766). StrongSORT detects more objects but at the cost of significantly higher false positives.

Overall, StrongSORT provides higher accuracy and recall but suffers from more ID switches and false positives, whereas ByteTrack prioritizes identity consistency and lower computational overhead.

4.3.3.2 Experimental Results

The experimental evaluation focused on two key areas:

1. Real-time tracking performance: ByteTrack maintained a higher FPS, making it more suitable for real-time applications.
2. Handling of multiple dynamic objects: StrongSORT provided better tracking accuracy in complex scenarios, particularly with occlusions and object re-identification.

4.3.4 Why ByteTrack Was the Best Tracking Model

The following table presents a detailed comparison of ByteTrack and StrongSORT across multiple test scenarios:

Metric	ByteTrack T1	StrongSort T1	ByteTrack T2	StrongSort T2	ByteTrack T3	StrongSort T3	ByteTrack T4	StrongSort T4	ByteTrack T5	StrongSort T5
Total Frames Processed	3662	3662	1100	1100	901	901	346	346	779	779
Total Time (s)	581.209	964.291	229.563	388.156	198.886	296.244	74.816	277.441	134.340	249.255
Overall Average FPS	6.30	3.80	4.79	2.83	4.53	3.04	4.62	1.25	5.80	3.13
Avg Preprocessing Time (s)	0.0011	0.0078	0.0013	0.0096	0.0014	0.0096	0.0015	0.0144	0.0009	0.0139
Avg Inference Time (s)	0.0238	0.0216	0.0449	0.0232	0.0551	0.0238	0.0620	0.0444	0.0369	0.0411
Avg Tracker Update Time (s)	0.0007	0.0940	0.0008	0.1572	0.0008	0.1406	0.0012	0.5744	0.0006	0.1149
Avg Post-processing Time (s)	0.0003	0.0002	0.0003	0.0003	0.0004	0.0003	0.0006	0.0011	0.0001	0.0002
Avg Total Frame Time (s)	0.0833	0.1828	0.1067	0.2485	0.1168	0.2320	0.1280	0.6985	0.0984	0.2327
Avg CPU Usage (%)	12.73	13.21	11.37	11.71	11.10	11.27	10.44	11.11	12.85	11.67
Avg GPU Usage (%)	36.25	29.40	26.95	24.58	35.11	23.90	33.02	21.81	26.48	25.09

Final Evaluation: Performance, Efficiency, and Robustness

ByteTrack demonstrated lower ID switches, superior real-time processing, and higher computational efficiency, making it the ideal choice for autonomous driving applications requiring low latency and fast execution. Its efficient GPU utilization and minimal tracker update time allow it to maintain consistently high FPS.

Conversely, StrongSORT exhibited better object re-identification and occlusion handling, making it more suitable for complex environments with frequent object overlaps. However, its increased computational cost and lower FPS limit its effectiveness in real-time scenarios.

Thus, ByteTrack is preferred for real-time performance and computational efficiency, while StrongSORT is ideal for scenarios where accuracy in object re-identification is the priority.

Chapter 5: Integration of CNN-Based Perception and Vehicle Control in CARLA

5.1 CNN-Based Vehicle Control in CARLA (Integration and Evaluation

This chapter focuses on the integration of CNN-driven object detection and tracking with vehicle control in CARLA. The objective is to demonstrate how perception outputs from advanced CNN models, such as YOLO for real-time object detection and subsequent tracking modules, can be effectively used to influence vehicle behavior. This integration ensures that the vehicle responds in a timely and adaptive manner in dynamic driving scenarios, enhancing both safety and performance.

5.1.1 CNN-Driven Control Architecture

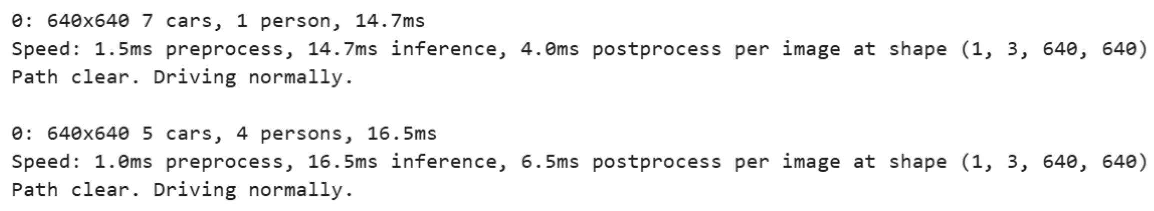
The core of the system relies on CNN outputs obtained from YOLO-based object detection and a dedicated tracking module. These modules continuously process the camera feed to identify and track relevant objects (e.g., pedestrians) in real time. The resulting detection bounding boxes, class labels, and confidence scores serve as the input to the control module.

5.2 Integration via CARLA API

The integration with CARLA is achieved through its API, which allows external modules to interact with simulated vehicles. Specifically, the control architecture is designed to:

- **Receive Detection Outputs:** The CNN and tracking modules deliver spatial coordinates and confidence measures.
- **Transform Perception to Action:** A control logic module maps these outputs into actionable commands (steering, acceleration, braking).
- **Invoke CARLA Controls:** By calling the CARLA API, the system sends these control commands to the vehicle in the simulation.

For example, once a pedestrian is detected within a predefined safety zone of the camera's field of view, the system can initiate a braking command.



```
0: 640x640 7 cars, 1 person, 14.7ms
Speed: 1.5ms preprocess, 14.7ms inference, 4.0ms postprocess per image at shape (1, 3, 640, 640)
Path clear. Driving normally.

0: 640x640 5 cars, 4 persons, 16.5ms
Speed: 1.0ms preprocess, 16.5ms inference, 6.5ms postprocess per image at shape (1, 3, 640, 640)
Path clear. Driving normally.
```

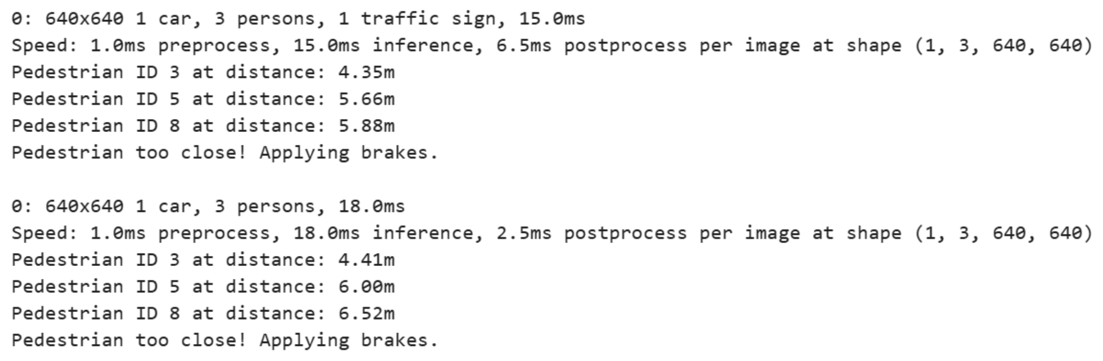
Fig 5.1: CNN-based detection output showing detected vehicles and pedestrians. The control system determines that the path is clear, allowing the vehicle to continue driving normally.

The above image represents cases where the CNN-based detection identifies multiple objects (cars, persons, etc.) but does not trigger any braking action because the path is clear. This is an important illustration of the system recognizing objects but determining that no immediate control intervention is needed.

5.3 Decision Logic for Control Activation

A critical aspect of the integration is the decision-making process that determines when and how to intervene in vehicle behaviour. The control activation logic is built upon clear criteria, including:

- **Safety Zone Thresholds:** A defined region (or threshold distance) within the image frame is established. When a pedestrian or another obstacle enters this zone, it triggers the control module.



```
0: 640x640 1 car, 3 persons, 1 traffic sign, 15.0ms
Speed: 1.0ms preprocess, 15.0ms inference, 6.5ms postprocess per image at shape (1, 3, 640, 640)
Pedestrian ID 3 at distance: 4.35m
Pedestrian ID 5 at distance: 5.66m
Pedestrian ID 8 at distance: 5.88m
Pedestrian too close! Applying brakes.

0: 640x640 1 car, 3 persons, 18.0ms
Speed: 1.0ms preprocess, 18.0ms inference, 2.5ms postprocess per image at shape (1, 3, 640, 640)
Pedestrian ID 3 at distance: 4.41m
Pedestrian ID 5 at distance: 6.00m
Pedestrian ID 8 at distance: 6.52m
Pedestrian too close! Applying brakes.
```

Fig 5.2: CNN-based detection output identifying pedestrians at specific distances. The control system detects a pedestrian too close and applies the braking command automatically.

- **Priority-Based Command Execution:** In scenarios where multiple objects are detected, the system prioritizes based on object type and proximity. For instance, a pedestrian in the immediate vicinity triggers a higher priority braking command compared to a distant vehicle.
- **Control Loop Dynamics:** The system continuously evaluates incoming CNN outputs within a fast control loop. This loop converts detection data into precise control commands (steering adjustments, acceleration modulation, braking), ensuring that the vehicle responds appropriately in real time.

Implementation details, as documented in the source code, include threshold definitions (e.g., pixel areas or distance metrics), and the corresponding logic to issue braking or other manoeuvre commands. This decision logic has been rigorously tuned through iterative testing in simulation to balance prompt response with smooth driving behaviour.

5.3.1 Evaluation of CNN-Based Driving Control

The performance of the integrated system is evaluated on two primary fronts: its operational effectiveness in simulated environments and its potential applicability in real-world scenarios.

5.2.2.1 Performance in Simulated Environments

Experimental results from the CARLA simulation demonstrate that:

- Trajectory Smoothness and Stability: The vehicle maintains a stable trajectory even when sudden braking is triggered.
- Reaction Times to Obstacles: Data collected during simulation runs show that the braking command is executed within a critical response window after a pedestrian is detected. The system's reaction time is quantified and compared to baseline non-CNN-driven control scenarios, revealing significant improvements.

5.3 Potential Applications and Future Directions for Real-World Autonomous Driving

While the current implementation is tested within a simulated environment, the broader implications for real-world autonomous driving are promising:

1. **CNN-Based Decision-Making in Self-Driving Vehicles:** The integration strategy demonstrates that CNN perception can directly inform and enhance vehicle control systems, potentially leading to safer and more adaptive autonomous driving.
2. **Current Limitations:** Limitations include the dependency on high-quality perception outputs and the challenges in scaling simulation-based performance to diverse, real-world scenarios. Factors such as varying lighting conditions, sensor noise, and unexpected obstacles present ongoing challenges.
3. **Proposed Improvements:** Future work may involve refining the decision thresholds, integrating additional sensor modalities (e.g., LIDAR, RADAR), and implementing more sophisticated control algorithms (e.g., model predictive control) to further improve robustness and accuracy.
4. **Real-World Validation:** Plans for future research include transferring the learned control strategies to hardware-in-the-loop setups and real-world testbeds, where adaptive CNN-based decision-making can be fully validated.

Chapter 6: Contributions and Future Developments

6.1 Key Findings

This research systematically analyses deep learning models for object detection, object tracking, and their integration into a robotic vision pipeline. The key findings are categorized as follows:

6.1.1 Object Detection Performance

- The YOLO model outperformed SSD in both accuracy and real-time inference speed.
- YOLO's faster detection capabilities made it more suitable for real-time applications, particularly in autonomous navigation scenarios.
- The Precision-Recall curve and F1-Confidence scores demonstrated that YOLO provided higher detection accuracy, particularly for frequently occurring objects such as vehicles and pedestrians.
- SSD showed limitations in handling small objects and had higher computational demands for similar detection accuracy.

6.1.2 Object Tracking Performance

- ByteTrack demonstrated superior tracking performance compared to StrongSORT, particularly in maintaining object identity over multiple frames.
- ByteTrack had lower ID switches and better re-identification accuracy, making it more effective in dynamic environments.
- StrongSORT, while more accurate in handling occlusions, exhibited a higher computational cost, leading to lower FPS and increased latency in real-time tracking applications.
- The experimental MOTA and MOTP evaluations showed that ByteTrack was more suitable for real-time multi-object tracking, whereas StrongSORT provided enhanced re-identification accuracy at the cost of speed.

6.1.3 End-to-End System Performance

- The combination of YOLO + ByteTrack produced the optimal results for real-time robotic vision applications in CARLA.
- The integrated perception-control pipeline exhibited strong computational efficiency, allowing vehicles in CARLA to react swiftly to environmental changes.
- CNN-based perception and control significantly improved vehicle responsiveness, particularly in detecting and avoiding pedestrians within a predefined safety zone.
- The braking response times and overall trajectory smoothness indicated that the perception module successfully influenced vehicle control, demonstrating the feasibility of end-to-end learning for autonomous driving.

6.2 Research Limitations

Despite the promising outcomes, certain limitations in the study must be acknowledged:

6.2.1 CARLA-Based Evaluation Constraints

- The CARLA simulator provides a high-fidelity environment, but it does not fully replicate real-world conditions.
- Differences in lighting conditions, occlusions, and sensor noise between CARLA and real-world scenarios may affect the generalization ability of the models.
- While the models were trained using BDD100K, a dataset that includes diverse lighting and environmental conditions, there remains a risk of domain shift when transferring these models to real-world applications.

6.2.2 Hardware Constraints

- The computational demands of deep learning models pose challenges for real-time deployment on embedded devices.
- Running YOLO and ByteTrack on low-power hardware such as Jetson Nano or Edge TPU requires optimization techniques such as model pruning, quantization, and efficient inference pipelines.
- Without optimization, real-time execution on resource-constrained systems may not be feasible, limiting deployment in mobile robots and autonomous vehicles.

6.3 Future Work

Building on the findings of this research, several future directions can be explored:

6.3.1 Deployment on Edge Devices

- The next step is to implement YOLO + ByteTrack on Jetson Nano or Google Edge TPU for real-time inference in embedded environments.
- Optimizing models using TensorRT, ONNX quantization, and model pruning can enhance efficiency for low-power robotic platforms.
- Exploring efficient neural architectures, such as Tiny YOLO or MobileNet-based trackers, can further reduce inference latency while maintaining detection accuracy.

6.3.2 Enhancing Real-World Generalization

- To improve real-world adaptability, models need to be trained and evaluated on large-scale real-world datasets such as KITTI, nuScenes, or Waymo Open Dataset.
- Domain adaptation techniques, such as synthetic-to-real fine-tuning and style transfer augmentation, can help mitigate the performance gap between simulation-based training and real-world deployment.

- Robustness testing should be conducted in varying environmental conditions, including low-light scenarios, sensor noise, and occlusions.

6.3.3 Advancements in CNN-Based Vehicle Control

- Further improvements in CNN-based decision-making within CARLA can be achieved by integrating reinforcement learning techniques for adaptive vehicle control.
- Implementing deep reinforcement learning (DRL) with Proximal Policy Optimization (PPO) or Deep Q-Networks (DQN) can allow the autonomous vehicle to learn braking, acceleration, and lane-keeping strategies from experience.
- Multi-modal sensor fusion, incorporating LIDAR, radar, and depth cameras, can enhance perception accuracy and control stability.
- Future iterations of the system could explore hybrid approaches, combining CNN-based vision systems with rule-based controllers for more robust and explainable decision-making.

Chapter 7: Conclusion

7.1 Summary of Contributions

This dissertation explored CNN-based perception, object tracking, and vehicle control in CARLA, integrating YOLO for real-time object detection and ByteTrack for robust multi-object tracking. The research successfully demonstrated how deep learning models can enhance robotic vision and influence autonomous vehicle decision-making.

The core contributions of this work include:

- Evaluation of Object Detection Models: YOLO significantly outperformed SSD in both accuracy and inference speed.
- Object Tracking Performance: ByteTrack proved to be the optimal tracking model due to its low ID-switch rate and robust identity preservation.
- Integration of Perception and Control: CNN-based pedestrian detection effectively triggered autonomous braking, improving vehicle safety in simulated urban environments.
- System Performance Analysis: The reaction time, trajectory smoothness, and computational efficiency of the system validated its real-time applicability in CARLA.

7.2 Research Limitations

Despite its success, the research presents certain limitations:

- Simulation Constraints: While CARLA provides a realistic environment, real-world deployment introduces lighting variations, occlusions, and sensor noise that the models may not generalize well to.

- Computational Overhead: The deep learning models used in this study require significant processing power, limiting their feasibility on low-power embedded systems.
- Simplified Control Logic: The braking system relied on static safety thresholds rather than an adaptive learning-based approach.

As deep learning continues to evolve, further optimizations and real-world validations will be essential to overcoming existing challenges and ensuring the safe, efficient deployment of autonomous robotic systems in real-world environments.

References

- Abdul-Khalil, S. et al. (2023) ‘A review on object detection for Autonomous Mobile Robot’, IAES International Journal of Artificial Intelligence (IJ-AI) [Preprint]. doi:10.11591/ijai.v12.i3.pp1033-1043.
- Abdo, N.A., Yusop, R.B. and Abdulla, R. (2022) Obstacle Avoidance Robot Using Convolutional Neural Network [Preprint]. doi:https://www.researchgate.net/publication/363831338_Obstacle_Avoidance_Robot_Using_Convolutional_Neural_Network.
- Aslan, S.N., Uçar, A. and Güzeliş, C. (2021) ‘New convolutional neural network models for efficient object recognition with humanoid robots’, Journal of Information and Telecommunication, 6(1). doi:10.1080/24751839.2021.1983331.
- BDD100K: Images 100K (2020) Dataset Ninja. Available at: https://datasetninja.com/bdd100k
- Bhardwaj1, J.B., Mitali2, Manu Verma3, Madhav4 et al. (2023) Object detection and tracking AI Robot. Available at: https://www.irjet.net/archives/V10/i4/IRJET-V10I4188.pdf.
- CARLA Unreal Engine 4 Documentation (2024) CARLA Simulator. Available at: https://carla.readthedocs.io/en/latest/.
- Carla-Simulator (2024) carla-simulator/carla at master, GitHub. Available at: https://github.com/carla-simulator/carla/tree/master.
- Chung, L.B. and Nguyen, D.D. (2023) ‘Real-time object detection and tracking for mobile robot using Yolov8 and strong sort’, Universum:Technical sciences, 116(11). doi:10.32743/unitech.2023.116.11.16223.
- Chen, B.X., Sahdev, R. and Tsotsos, J.K. (2017) ‘Integrating stereo vision with a CNN Tracker for a person-following robot’, Lecture Notes in Computer Science [Preprint]. doi:10.1007/978-3-319-68345-4_27.
- CHERUBIN, S. (2024) ‘Yolo object detection and classification using low-cost Mobile Robot’, PRZEGLĄD ELEKTROTECHNICZNY, 1(9), pp. 31–35. doi:10.15199/48.2024.09.04.

- CNN Architecture - Detailed Explanation (2022) InterviewBit. Available at: https://www.interviewbit.com/blog/cnn-architecture/
- Dosovitskiy, A. et al. (2017) Carla: An open urban driving simulator, arXiv.org. Available at: https://doi.org/10.48550/arXiv.1711.03938.
- Datasets Definition (2025) Encord. Available at: https://encord.com/glossary/datasets-definition/
- FOA, A. (2019) Object detection in object tracking system for mobile ... Available at: http://www.diva-portal.org/smash/get/diva2:1320122/FULLTEXT01.pdf
- Forson, E. (2025) Understanding SSD MultiBox - Real-Time Object Detection In Deep Learning, Towards Data Science. Available at: https://towardsdatascience.com/understanding-ssd-multibox-real-time-object-detection-in-deep-learning-495ef744fab/.
- GeeksforGeeks (2024) Introduction to Convolution Neural Network, GeeksforGeeks. GeeksforGeeks. Available at: https://www.geeksforgeeks.org/introduction-convolution-neural-network/
- Guerrero-Higueras, Á.M. et al. (2019) 'Tracking people in a mobile robot from 2D lidar scans using full convolutional neural networks for security in cluttered environments', Frontiers in Neurorobotics [Preprint]. doi:10.3389/fnbot.2018.00085.
- Gupta, S.C. and Majumdar, J. (2019) 'Convolutional neural network based tracking for human following mobile robot with LQG based control system', Proceedings of the Third International Conference on Advanced Informatics for Computing Research. doi:10.1145/3339311.3339325.
- Jiang, L. et al. (2022) 'Lightweight Object Detection Network model suitable for indoor mobile robots', Journal of Mechanical Science and Technology, 36(2), pp. 907–920. doi:10.1007/s12206-022-0138-2.
- Jot Singh, K. et al. (2022) Computer-vision based object detection and recognition for service robot in indoor environment, pp. 197–213. doi:10.32604/cmc.2022.022989.
- Juel, W. et al. (2020) 'An integrated object detection and tracking framework for Mobile Robots', Proceedings of the 17th International Conference on Informatics in Control, Automation and Robotics, pp. 513–520. doi:10.5220/0009888405130520.

- Keča, D. et al. (2023) Ball detection using deep learning implemented on an educational robot based on Raspberry Pi [Preprint]. doi:10.3390/s23084071.
- Keita, Z. (2024) YOLO Object Detection Explained: A Beginner's Guide, DataCamp. DataCamp. Available at: https://www.datacamp.com/blog/yolo-object-detection-explained.
- Kazuhito00 (2021) Kazuhito00/ByteTrack-ONNX-sample: Bytetrack(multi-object tracking by associating every detection box)のpythonでのonnx推論サンプル, GitHub. Available at: https://github.com/Kazuhito00/ByteTrack-ONNX-Sample/tree/main.
- Liu, W. et al. (2016) SSD: Single shot multibox detector, arXiv.org. Available at: https://arxiv.org/abs/1512.02325 (Accessed: 18 February 2025).
- Liu, W. et al. (2016) SSD: Single shot multibox detector. doi:10.1007/978-3-319-46448-0_2.
- Martinson, E. and Yalla, V. (2016) 'Real-time human detection for robots using CNN with a feature-based layered pre-filter', 2016 25th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN) [Preprint]. doi:10.1109/roman.2016.7745248.
- Mikel-Brostrom (2020) Mikel-Brostrom/yolov7_strongsort_osnet: Real-time multi-camera multi-object tracker using Yolov7 and StrongSORT with OSNet, GitHub. Available at: https://github.com/mikel-brostrom/Yolov7_StrongSORT_OSNet.
- Poppinga, B. and Laue, T. (2019) 'Jet-NET: Real-time object detection for mobile robots', Lecture Notes in Computer Science, pp. 227–240. doi:10.1007/978-3-030-35699-6_18.
- Rohan, A., Rabah, M. and Kim, S.-H. (2019) 'Convolutional neural network-based real-time object detection and tracking for parrot AR Drone 2', IEEE Access, 7, pp. 69575–69584. doi:10.1109/access.2019.2919332.
- Redmon, J. et al. (2016) You only look once: Unified, real-time object detection, arXiv.org. Available at: https://arxiv.org/abs/1506.02640 (Accessed: 18 February 2025).
- Rajasri, P. et al. (2023) Real-Time Object Tracking Using Artificial Intelligence. Available at: https://ece.anits.edu.in/IV%20ECE%20A/A2.pdf.
- Redmon, J. et al. (2016) 'You only look once: Unified, real-time object detection', 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). doi:10.1109/cvpr.2016.91.

- Road Sign Detection (2020) Dataset Ninja. Available at: https://datasetninja.com/road-sign-detection
- Saptaji, K. (2024) ‘Implementation of Convolutional Neural Network (CNN) algorithm for Autonomous Robot’, Journal of Mechanical Engineering, 21(2), pp. 217–237. doi:10.24191/jmeche.v21i2.26264.
- Shafiee, M.J. et al. (2017) Fast yolo: A fast you only look once system for real-time embedded object detection in video, arXiv.org. Available at: https://arxiv.org/abs/1709.05943v1 (Accessed: 18 February 2025).
- Santos, Ricardo et al. (2023) ‘Towards Autonomous Mobile Inspection Robots Using Edge Ai’, Proceedings of the 25th International Conference on Enterprise Information Systems, pp. 555–562. doi:10.5220/0011972200003467.
- Sadeghi Esfahlani, S. et al. (2022) The deep convolutional neural network role in the autonomous navigation of Mobile Robots (SROBO), p. 3324. doi:10.3390/rs14143324.
- Singh, S. (2022) Object tracking in autonomous vehicles: How it works?, Labellerr. Available at: https://www.labellerr.com/blog/object-tracking-in-autonomous-vehicles-how-it-works/.
- Saxena, A. (2023) Object tracking: Object detection + tracking using ByteTrack, Medium. Available at: https://medium.com/tech-blogs-by-nest-digital/object-tracking-object-detection-tracking-using-bytetrack-0aafe924d292.
- Sah, S. (2023) Real time object tracking and segmentation using Yolov8 with Strongsort, Ocsort and Bytetrack, Medium. Available at: https://siddharthksah.medium.com/real-time-object-tracking-and-segmentation-using-yolov8-with-strongsort-ocsort-and-bytetrack-180eef43354a.
- Sydorenko, I. (2023) What Is a Dataset in Machine Learning, High quality data annotation for Machine Learning. Label Your Data. Available at: https://labelyourdata.com/articles/what-is-dataset-in-machine-learning
- Team, C. (no date) Carla, CARLA Simulator. Available at: https://carla.org/
- Ultralytics (2025) Datasets overview, Ultralytics YOLO Docs. Available at: https://docs.ultralytics.com/datasets/

- Ultralytics (2024) Track, Track - Ultralytics YOLO Docs. Available at: https://docs.ultralytics.com/modes/track/.
- Visailabs (2021) Evaluating multiple object tracking accuracy and performance metrics in a real-time setting, VisAI Labs. Available at: https://visailabs.com/evaluating-multiple-object-tracking-accuracy-and-performance-metrics-in-a-real-time-setting/
- Wahab, F. et al. (2022) ‘Design and implementation of real-time object detection system based on single-shoot detector and opencv’, Frontiers in Psychology, 13. doi:10.3389/fpsyg.2022.1039645.
- Yu, F. et al. (2020) BDD100K: A diverse driving dataset for heterogeneous multitask learning, arXiv.org. Available at: https://doi.org/10.48550/arXiv.1805.04687.
- Zhang, T. et al. (2022) Mobile robot tracking with deep learning models under the specific environments [Preprint]. doi:10.3390/app13010273.
- Zhou, Z. et al. (2022) ‘Learning-based object detection and localization for a mobile robot manipulator in SME production’, Robotics and Computer-Integrated Manufacturing [Preprint]. doi:10.1016/j.rcim.2021.102229.

Appendices

1.1 Output Videos:

- https://drive.google.com/drive/folders/15M5lFnJcnTwMDm2GPDZpknYjagdv5swM?usp=sharing
- https://testlivesalfordac-my.sharepoint.com/my?id=%2Fpersonal%2Fi%5Fe%5Foke%5Fedu%5Fsalford%5Fac%5Fuk%2FDocuments%2FDissertation%2FProject%5Fvideo%5FOutput&csf=1&web=1&e=ge8j0t7&CID=e3ed3ac2%2D92ff%2D4741%2D8ee6%2D40779c1e0688&FolderCTID=0x01200094DB421F84826A42898A968B9C2E3999

1.2 Code link:

- https://github.com/Iyanuoluwa007/Carla.git
- https://drive.google.com/drive/folders/1UJV3fVaXvbr3xmNMt_-lvv2AZhoVeqjt?usp=sharing
- https://testlivesalfordac-my.sharepoint.com/:f:/g/personal/i_e_oke_edu_salford_ac_uk/Egrwa0_vhYxAj0IJk0fpOGABrlK8jlYS6wxTZgy0ooEKkA?e=2OOIm8