KENYATTA UNIVERSITY
SCHOOL OF PURE AND APPLIED SCIENCES

ADAPTIVE PEDESTRIAN DETECTION SYSTEM BASED ON DEEP LEARNING

Aquino Nyapara Joctum BSc (Computer Science)

J562/20198 /2021

Sign……..……..Date…27/03/2025……..

Department of Computing and Information Science

This research project submitted in partial fulfilment of the requirements for the award of the degree of Master of Science in Computer Science in the School of Pure and Applied Sciences of Kenyatta University

Supervisor

Dr John Kandiri (PhD - IS, Msc. IS)
Senior Lecturer, Department of Computing and Information Science (CIS) & IS Researcher
Kenyatta University

Signature…….. Date…….

27/03/2025

Declaration

I declare that this project is my original work and has not been presented in any other university/institution for consideration of any certification. This research project has been complemented by referenced sources duly acknowledged. Where text, graphics, pictures, figures or tables have been borrowed from other sources, including the internet, these are specifically accredited, and references cited using current APA system and in accordance with antiplagiarism regulations

Signature_____ **Date:** _____

Name: Aquino Nyapara Joctum

Registration number: J562/20198 /2021

Department: Computing and Information Science

Supervisors' declaration*:*

This project has been submitted for appraisal with our/my approval as University Supervisor(s).

Signature _____Date_____

Name: Dr. J. Kandiri

Senior Lecturer

Department of Computing and Information Science

Kenyatta University

Abstract

The existing pedestrian detection algorithms have the potential to improve road safety on a regional level, however their effectiveness in dynamic rural and urban environments remains unexploited. With this potential capability, their efficacy remains uncertain due to infrastructure and operational limitations. Nationally, integration into Kenya's transport system is still in its infancy, with challenges in policy, infrastructure, and technological readiness limiting real-world deployment. The problem lies in the inability of current systems to provide accurate and timely detection, particularly in complex road topologies such as Type-S roads with sharp curves and frequent occlusions. To address this, this research proposes a YOLO-APD network to enhance detection accuracy and achieve real-time processing. A cost-effective RGB camera in the CARLA simulator was used to generate a custom dataset reflecting diverse traffic scenarios. Enhancements to the YOLOv8 baseline include a novel SimSPPF module for improved feature extraction and speed, a modified detection head with a gather-and-distribute mechanism, and C3Ghost modules for balancing efficiency and accuracy. The model was evaluated through ablation experiments, algorithm comparisons, and robustness tests. Results show YOLO-APD achieved a mean Average Precision (mAP) of 97.8%, with pedestrian detection exceeding 99.5%, outperforming state-of-the-art models. The model demonstrated robust performance with a 94% F1 score, validating its generalization ability in challenging environments. By enhancing detection accuracy and efficiency in Type-S roads, YOLO-APD presents a viable solution for improving autonomous navigation in complex traffic environments.

Table of Contents

LIST OF FIGURES

LIST OF EQUATIONS

TABLE OF ABBREVIATIONS

Abbreviation	Definition
RGB	Red, Green Blue
ADAS	Advanced Driver Assistance System
DSR	Design Science Research
AV	Autonomous Vehicle
DL	Deep Learning
IR	Infra-Red
CNN	Convolutional Neural Network
LIDAR	light detection and ranging
C4D	Cinema 4Dimension
DAVE	DARPA Autonomous Vehicle
FPS	Frames Per Second
CSPN	Convolutional spatial propagation network
TP	True Positive
FN	False Negative
FP	False Positive
YOLO-APD	YOLO Adaptive Pedestrian Detection

CHAPTER ONE

INTRODUCTION AND BACKGROUND STUDY

1.1. Introduction

Advanced Driver Assistance System (ADAS) is a rapidly growing topic eliciting interest in both the intelligent automotive sector and academia. ADAS seeks to increase global safety and responsiveness in autonomous vehicles (AV) with the main aim of reducing accidents that claim millions of lives (Hassaballah et al., 2021). With increasing global acceptance of AVs, the requisite for robust perception systems has become even more crucial in decreasing accidents which claim millions of lives each year. For AV to be safe and intelligent, they are expected to navigate traffic with pedestrian in proximity and make critical pedestrian decision based on the complexity of the traffic scene. While advances in computer vision and deep learning have contributed significantly to improved object detection capabilities, pedestrian detection in difficult road settings and difficult weather remains a serious challenge (Hassaballah et al., 2021).

In developing regions, road infrastructure poses immense challenge for pedestrian detection. Many roads are unstructured, others are characterized by erratic terrains and pedestrian movements is highly unpredictable for instance in many African countries, roads typically experience mixed traffic conditions with pedestrians sharing space with automobiles making pedestrian detection even more complex as highlighted by Ziryawulawo et al., (2023) in their work which focuses on a lane departure warning system for a local motor company in Uganda. These factors exacerbate the difficulties faced by AV perception systems highlighting the need for solutions that consider real world driving environments beyond structured road networks.

Light Detection and Ranging (LiDAR), Radio Detection and Ranging (RADAR), RGB cameras, infrared (IR) cameras , and depth sensors are some of the sensor technologies used by state-of -art pedestrian detection systems. In the context of predictable road topologies, deep

learning-based detection algorithms have demonstrated remarkable performance. Their utility is however, reduced in dynamic and uncertain environments such as roadways with steep curves and uneven terrain. Large scale deployment is not feasible with current solutions since they frequently presume clear fields of view and depend on very costly sensor configurations. Moreover, unfavourable weather conditions, sensor occlusion and unpredictable pedestrian behaviour are examples of real-world problems that increase complexity and impairment of object detection performance.

A critical issue in autonomous navigation is detecting pedestrians and other hazards on complex road geometries like type S roads, which are characterized by rapid curves and limited visibility as shown in the Figure 1. Obstructed field of view and the dynamic motion of the vehicle make it difficult for timely pedestrian detection, in contrast to linear roads where the field of view is comparatively unhindered. Since this type of road geometry is not explicitly considered by conventional object detection algorithms used in autonomous driving, it poses difficulty for AVs to anticipate hazards in real-time due to missed detections leading to potential accidents.

This project addresses these issues by investigating a novel pedestrian detection approach that

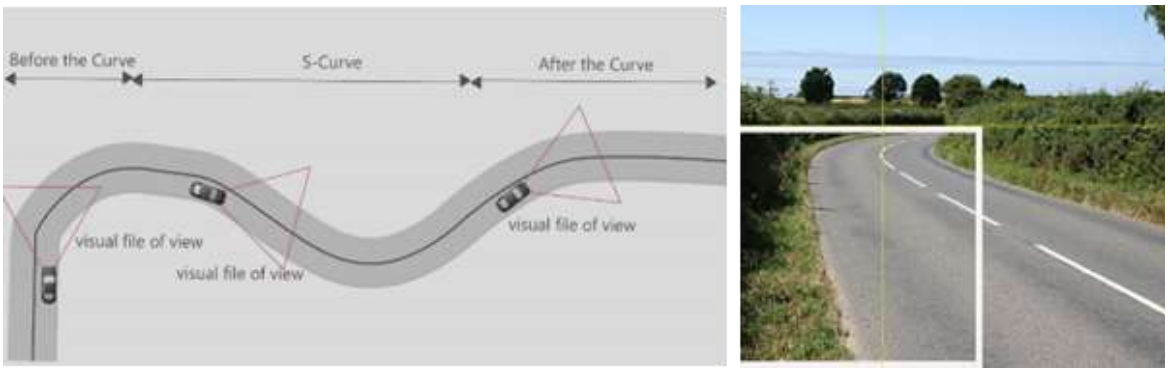

Figure 1 Type S Road and difficulty of detecting hazard by AV

is optimized for roads with complex geometries such as type S road. The study explores a cost-effective yet scalable deep learning model that uses steering wheel angle data to improve awareness and reduce delay in pedestrian detection. The objective is to create a prototype that

enhances detection accuracy and adaptability while preserving real-time performance in dynamic traffic environments.

1.2. Background information

According to the Society of Automotive Engineers (SAE) , five levels of autonomy can be used to describe self driving vehicles as demonstrated by Lowe & Kuru, 2024) . These levels can vary from level 0 (being a nonautomated vehicle) to level 5 (being a fully autonomous vehicle)

Today's Autonomous Vehicles (AV) use cutting edge hardware and software technologies to complete their driving as well as object detection. This has largely been attributed to the recent emergence, success and state of art results in Deep learning (DL) in computer vision applications as highlighted by which has contributed immensely to the popularity of these self-driven vehicles as it enables them plan, manoeuvre their environment with ease and predictability of satisfiable degree (Gawade & Suresh Kumar, 2021; Khanum et al., 2020; Kuutti et al., 2021) . Worth noting is that this wide acceptability emanates from affordability, high accuracy and ability of deep learning models to self-optimise its behaviour from data and adapt to new scenarios as contended by Qiu et al., (2021), superseding prior machine learning based methods (Lu, 2022), which depended on individual, iterative parameter finetuning while trying to maintain performance in all foreseeable scenarios.

Despite these advances, achieving collision free movement in an interactive traffic remains a persisting challenge in autonomous driving. For the autonomous vehicle to detect the object Infront of it as a hazard and make critical split-second decision, detection needs to be computed quickly. such efficiency is only possible with real- time and accurate interpretation of its surrounding without human intervention, based on through input from sensory data (Wu et al., 2023).

In autonomous vehicles, sensor technology is used to measure the surrounding of a vehicle for estimating the state of a system. The sensors used for this purpose are Light Detection And Ranging (LIDAR), Radio Detection And Ranging (RADAR), Colour camera (RGB), infrared cameras (IR or thermal cameras), depth sensors, wireless sensor network. a deep learning algorithm is integrated with these sensor technologies to enable the perception unit in the autonomous vehicles. In their work (Akyol et al., 2020), argues that real-time pedestrian detection implementation remains a challenge due to high sensor cost, a sentiment widely shared by (*The Future of Autonomous Vehicles (AV) | McKinsey*, n.d.) who notes in their report from 2023 the challenges of commercializing more advanced autonomous driving systems due to steep up-front costs, besides complex algorithms, limited hardware, unpredictable human behaviour, erratic weather and roads of varying geometry in different regions in the world. Some roads exhibit rough and undulated terrains without defined lanes, others are smooth and characterised by lane markings while others may exhibit rapid curves and turns.

The pedestrian detection environment this work was concerned with is complex road geometry known a type S road characterised by rapid curves and turns as shown in Figure 1. Since road geometry is linked to the dynamics of the vehicle; in roads of linear geometry where the field of view is clear, pedestrian detection is straight forward. However, in road with rapid curves, when the driver first negotiates the first curve and subsequent one appears there is limited field of view and detecting in coming hazard in opposite lane becomes tricky.

Indeed, autonomous vehicle control often has a strong link to perception (Basit et al., 2023), since all driving decisions, planning depend on surrounding environment understanding of the autonomous vehicle (AV).

This research developed an efficient and adaptive pedestrian detection system by leveraging affordable RGB sensors and an optimised deep learning model. Simulation provided cost

effective and practical means of testing effectivity of the pedestrian detection model in autonomous driving application prior to any future physical deployment. Simulation of physical systems have similar requirements in terms of object detection. A simulation of an experiment that is too costly or impractical to build can be of significant aid in the experimental process. Autonomous driving applications such as automobile test crash and collision are potential examples of where a simulation of the event is useful before physical deployment. The simulation environment used in this research replicated real-world conditions to evaluate fiscal viability and efficacy of the proposed pedestrian detection algorithm. The result of this study has broad applicability in enhancing simulation-based testing for autonomous driving applications and improving real-world pedestrian detection in autonomous driving applications with negligible variation.

This research aimed to develop a pedestrian detection system by applying deep learning detection techniques to increase pedestrian detection efficacy through simulation, using affordable sensors. Simulation of physical systems have similar requirements in terms of object detection. A simulation of an experiment that is too costly or impractical to build can be of significant aid in the experimental process. Autonomous driving applications such as automobile test crash and collision are potential examples of where a simulation of the event is useful before physical deployment. This project presents simulations of physical systems as representative of a large class of simulated environment-based application. As a result, any techniques developed to enhance their performance are widely applicable with negligible variation.

1.3. Statement of the Problem

Although the last decade has witnessed substantial growth in sensing, perception and localization for automated driving (Cao et al., 2023; Wu et al., 2023; D. Zhu et al., 2021), how to improve pedestrian detection performance in complex situations remains a persistent

challenge. This is the problem that this work seeks to address. Despite the success of the previous works in this field (Naeem et al., 2020), it appears that exploiting vehicle dynamics for effective pedestrian detection in complex road scenes using affordable sensors remains a dissertation in this area. More work needs to be done in exploring and developing effective, yet feasible sensor based (F. Bu et al., 2020; Jhong et al., 2022; Sahba et al., 2020; Urmila & Megalingam, 2020) detection mechanism if the dream of an all-autonomous driving is to be made a reality. Current models struggle with occlusion and small pedestrian detection. Latency effects become more pronounced when the collision cannot be avoided immediately after detection. Knowing how and when to execute optimal decision on a traffic scene with varied road geometry characteristics such as unpredictable curvature, slope, superelevation and the random nature of human behaviour remains an open challenge for an intelligent vehicle.

Prior works in this domain did not extensively explore using dynamic convolution techniques, affordable sensors with steering wheel dynamics to detect pedestrians on complex road settings such as type S roads. Addressing this gap is critical for advancing pedestrian detection systems that could operate effectively in real-world autonomous driving conditions. If the dream of achieving a fully autonomous vehicle deployment is to become a reality, it is essential to design a system that is adaptable and able to respond in real-time. The system must also leverage perception input from a scalable, cost-effective sensor mechanism without compromising speed and accuracy. This research aimed to realize the above goal by exploring novel deep earning techniques that enhanced real-time adaptable pedestrian detection in autonomous driving on road with complex geometry.

1.4. Significance of the Study

One intended outcome of this study, on a theoretical level, is advancing knowledge in the domain of pedestrian detection in autonomous driving. On practical level, this work contributes to solving the problem that may be faced by autonomous vehicle in challenging road scenes

such as those with complex infrastructure such as type S roads. Finally, this study demonstrates a novel use of cost effective, scalable sensor technology with detection algorithm in autonomous driving.

1.5. Objectives

1.5.1. General Objective

The general objective of this research is to reduce latency and improve perception adaptability in autonomous vehicle in type S roads without compromising on accuracy.

1.5.2. Specific Objective

- To investigate which algorithms can effectively contribute to solving real-time detection of pedestrians with high accuracy and precision.
- To investigate methods for limiting an object detection algorithm to only detect objects in critical region of an autonomous vehicle
- To design an adaptable pedestrian detection algorithm that minimizes latency and improves perception in autonomous vehicle in challenging road scenes like type S road.
- To evaluate the algorithm against state of art pedestrian detection and compare its performance.

1.5.3. Research Questions (Optional)

- How can deep learning algorithms improve pedestrian detection speed without compromising on accuracy?
- How can the object detection algorithms be optimised to focus only on critical regions within autonomous vehicles field of view?

- How can an adaptable pedestrian detection algorithm be designed to dynamically adjust to varying road conditions and what adaptive strategies would be effective in such scenarios.
- How does the developed pedestrian detection algorithm compare to state-of-the-art algorithms in terms of accuracy, speed and robustness?

1.6. Justification

With the spread of Autonomous vehicles in the current century, accurate and timely pedestrian detection is of critical priority in ensuring safety and reducing road accidents which cost approximately 1.3 million lives annually (Ranjan & Senthamilarasu, 2020; Razzaghi et al., 2020). Understanding the complexity of given road geometry and making subsequent split-second decisions that affect operations of the Autonomous Vehicle on the road depending on the context, remains a complex and challenging task of future Autonomous Vehicle given the highly dynamic environment and the unpredictable nature of human behaviour (Markkula et al., 2020). Dependence on conventional convolutional neural network operations limit the effectivity of the model leading to overall poor outcome for some complex input scenes (Chen et al., 2020). A case in point that highlights the need for adaptive real time detection is in 2018 after pedestrian was fatally struck by Uber's autonomous test vehicle (Ranjan & Senthamilarasu, 2020). Authorities revealed in their review that the vehicle was not able to recognize the victim as a jaywalker. This scenario and possibly many unforeseen cases calls for a detection mechanism that can flexibly adapt to the changing traffic scenario with minimal latency. To tackle this challenge this research adopts an adaptive technique for understanding the scene complexity and position of the pedestrian in relation to the region Infront of the steering wheel, which determines a lot of decisions affecting the autonomous vehicle. Existing solutions are expensive and not scalable. For instance, the available detection algorithms are memory intensive and have to train several models this is not feasible in critical situations

owing to limited representation. it is the belief of this work that a more compressive attack on this problem has a good chance to help improve upon the existing approaches.

1.7. Scope of the Study

The study focuses on improving pedestrian detection in autonomous vehicles navigating complex road geometry, particularly type S roads. The study confines in the exploration of integration of deep learning techniques, dynamic convolution methods and cost-effective scalable sensor to enhance perception adaptability in complex dynamic traffic environments. Adaptability in this context refers to the algorithms ability to maintain accurate detection performance under varying light conditions, occlusion levels and road curvatures. The prototype developed is based on YOLOv8 base model, a widely used architecture for object detection. The adaptive pedestrian detection algorithm works perfectly in providing descriptive analysis and prediction.

1.8. Limitations of the study

While this study strives to enhance pedestrian detection in complex scenarios, certain limitations must be acknowledged. The primary sensor configuration is affordable RGB camera with less emphasis on multi-sensor fusion considerations. This study does not delve into physical implementation of the prototype. Further, while vehicle dynamics are integrated, the focus is largely on steering wheel angle data neglecting other parameters like acceleration and braking. Furthermore, this work does not delve into details of lane detection which produce steering wheel angle. Finally the study is also specific to pedestrian detection task , without branching into other related areas of autonomous perception.

1.9. Assumptions of the study

This project claims simulations of physical systems as representative of a large class of simulated environment-based application. As a result, any techniques developed to enhance their performance are widely applicable with negligible variation.

CHAPTER TWO

LITERATURE REVIEW

2.1. Introduction

In this section a comprehensive review of what has been done in pedestrian detection in autonomous driving is presented. Pedestrian detection is examined as a subproblem of object detection with a keen exploration of deep learning techniques that have shaped this domain. Further adaptation to changing environments is examined as an extension of attention mechanisms. The goal here is to critically analyse existing literature, identify key advancements and persistent challenges, and importantly, identify the research gaps with an aim to further shape the dimension of this study.

2.2. Theoretical Framework

This study is theoretically grounded on three basic pillars: Computer Vision (CV) theory, Adaptive Systems (AS) theory, and Deep Learning (DL) theory. These interrelated theories offer a strong foundation for examining how autonomous systems interpret visual information, comprehend intricate data, and reach well-informed conclusions (Basit et al., 2023). Their choice is validated by its direct importance to pedestrian detection, allowing machines to identify and understand visual data (computer Vision), learn from large datasets (Deep learning), and prioritize focus on the most pertinent features in real-time to changing environments.

2.3. Computer Vision (CV) Theory

Vision is a concept that enables the biological vision to observe and recognize its surroundings. Zhang (2024) in the opening of his book “3D Computer Vision”, presents a detailed argument on complexity of the visual process, starting from the process of image acquisition which often involves experiencing one’s surrounding in order to have a concrete photorealistic comprehension of our environment image as shown in Figure 2 below.

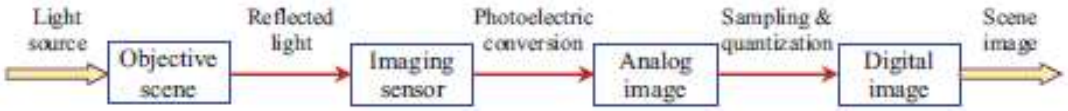

Figure 2 Concept of Vision

Further, the author demonstrates a universe of subjects such as chemistry, understanding of geometry, psychology, optics and other fields being intertwined in the complete understanding of the idea of vision.

The human vision is highly adaptable, capable of integrating multiple sensory channels such as colour, visual acuity and relies on complex neural networks and biological processes to achieve its functionality. Attempting to mimic this complexity into machine conceivable form is the subject of Computer vision (CV) (Szeliski, 2022). The fundamental idea of computer vision is for machines to realize near human vision functionalities, that is, the sensation, perception and interpretation of the objective word(Y.-J. Zhang, 2024).

Key ideas in this theory that apply to pedestrian detection include motion tracking(Methods to track an object's movement over time), image processing (approaches for enhancing and manipulating raw images), object recognition (algorithms that categorize and locate objects within a scene), and feature extraction(strategies for identifying salient visual characteristics).Computer Vision plays an essential role intelligent vehicles to accurately determine, monitor, and react to pedestrians in changing and unexpected situations.

2.4. Adaptive Systems (AS) Theory

Adaptation refers to adjustment in response to some anticipated change in a given environment. This term is largely borrowed from biological systems that change their structure or behaviour to fit in new ecological settings, in computing world, adaptive systems are systems that can constantly learn and accumulate experience in the process modify their behaviour, structure, or functionality in response to changes in their environment.

These systems leverage on recurrence-based analysis to handle complex, changing time series data, like self-reported AV states. According to Hasselman and Bosman (2020), recurrence methods such as change profiles (CP) quantify nonlinear dynamics across all time scales without assuming ergodicity or stationarity, in contrast to standard statistical models like TV augmented realities. This techniques allows adaptive systems to identify shifts from stable to chaotic behaviour while streaming data for real-time applications. AI integration occurs through agent-based modelling (ABM), wherein AI agents utilize recurrence metrics to dynamically adjust decision-making thresholds (Listopad, 2023). For example, in the event of daylight or terrain changes, the working algorithm would adjust to ensure real-time detection efficiency of the AV.

The principles of adaptive systems are manifested by autonomous vehicles in their constant modification of actions according to changes in their surroundings This flexibility is essential for real-time navigation, as the vehicles need to analyse large volumes of sensor information from cameras, LiDAR, radar, and GPS to anticipate pedestrian movements, identify obstacles, and enhance route planning. In this work adaptation is implemented through attention mechanism to ensure real-time feedback that enables the autonomous vehicles to dynamically react to unforeseen situations such as sudden obstacles, or variations in weather and road conditions. For example, sensor fusion methods integrate information from various sensors to improve precision perception and decrease latency, facilitating prompt decision-making.

2.5. Deep Learning (DL) Theory

Advancements in deep learning (DL) concepts has contributed immensely to evolution of Computer vision with numerous state of art results in autonomous vehicle planning, perception and control (Gawade & Suresh Kumar, 2021; Khanum et al., 2020; Kuutti et al., 2021; Tarchoun et al., 2020).

Convolutional Neural Networks (CNN) and more recently Vision transformers (Vit) are outstanding forms of deep learning models that have improved the field of computer vision and expanded the limits of object detection accuracy and preciseness in autonomous driving. Latest emerging concepts of attention mechanism, which aim to focus on particular area of interest has seen wide acceptance into deep learning domain, further improving the quality of computer vision related solutions (Hou et al., 2021; Y. Liu et al., 2021).

2.6. CARLA Simulator

CARLA is one of the most popular open-source simulation platforms for autonomous driving research, with an extensive customization capability using Python and C++ APIs. With its ability to incorporate dynamic weather conditions, a variety of traffic behaviours, and pedestrian interactions, it allows researchers to model intricate urban environments, dynamic weather conditions, and diverse pedestrian behaviour (T. Bu et al., 2021). CARLA supports multi-sensor integration, including LiDAR, RGB/depth cameras, GPS, and other related sensors whose mountings can be adjusted to precisely simulate real-world perception scenarios. To provide complete flexibility, certain sensor placements must be manually adjusted because they depend on predetermined connection points.

The Carla scenario runner engine enables the creation of complex traffic scenarios, such as lane change, crossroads, and roads of different topologies, such as type S roads. Although this

offers structured scenario execution, unexpected agent behavior frequently necessitates additional scripting7. Further, the simulator is compatible with ROS through the carla-ros-bridge, which enables external control of autonomous agents and real-time data streaming. However, effective ROS message handling and network latency are prerequisites for real-time performance (Kaljavesi et al., 2024).

CARLA excels in autonomous driving research compared to other simulators because of its scenario control capabilities and urban-centric setting. Despite providing beautiful landscapes, alternative simulators such as AirSim is primarily intended for procedural world generation simulations of airborne and ground vehicles; traffic dynamics are not integrated into its functionality.

Figure 3 Simulation of Urban and rural Countryside roads in Carla Simulator

On the other other hand, Gazebo, a general-purpose robotics simulator, can simulate high-fidelity physics for robotic applications, however it necessitates a lot of setup for situations involving autonomous driving (Zagirov et al., 2024). SUMO, another close competitor, lacks the sensor integration and in-depth perception modeling that CARLA offers, but it excels at macroscopic traffic flow analysis(Srivastava & Kumar, 2020).

CARLA excels in autonomous driving research compared to other simulators because of its scenario control capabilities and urban-centric setting. Despite providing beautiful landscapes, alternative simulators such as AirSim is primarily intended for procedural world generation simulations of airborne and ground vehicles; traffic dynamics are not integrated into its functionality. On the other otherhand, Gazebo, a general-purpose robotics simulator, can simulate high-fidelity physics for robotic applications, however it necessitates a lot of setup for situations involving autonomous driving (Zagirov et al., 2024). SUMO, another close competitor, lacks the sensor integration and in-depth perception modeling that CARLA offers, but it excels at macroscopic traffic flow analysis(Srivastava & Kumar, 2020).

CARLA is supported by an active open-source community that is constantly improving its capabilities thanks to the efforts of hundreds of developers. Recent developments include the creation of standardized benchmarking tools through the CARLA leaderboard and the co-simulation of real-world sensor data with CARLA, which allows for more realistic training of perception models. Because of its photorealism, sensor integrity, and variety of scenarios, CARLA is the go-to option for both industrial autonomous car development and scholarly study.

2.7. Autonomous Driving datasets

While prominent autonomous driving datasets such as Cityscapes, KITTI, and nuScenes (M. Liu et al., 2024) have been extensively used in research, each has limitations: Cityscapes focuses on semantic segmentation mainly in German cities and thus lacks diversity in road and weather conditions. The KITTI dataset, on the other hand, provides high-quality object recognition annotations but primarily relies on axis-aligned bounding boxes, which may not sufficiently capture occlusions or complex pedestrian movements (Janai et al., 2020). Finally, nuScenes offers multi-modal data, including nighttime scenes, but exhibits class imbalance and is limited to semi-structured environments with constrained pedestrian behaviours (Caesar et al., 2020).

To address these dataset limitations, CARLA's flexible simulation environment allows researchers to create and test diverse scenarios with complex pedestrian behaviours, dynamic weather, and challenging road geometries like S-shaped roads. By leveraging CARLA's Python API, complex synthetic environments were created to simulate real-world challenges, such as pedestrians occluded by vehicles, nighttime visibility issues, and unpredictable pedestrian movements.

Carla's real-time rendering capabilities enabled iterative validation of detection accuracy and computational efficiency, while its vehicle telemetry interface facilitated synchronized testing with contextual telemetry data (e.g., speed, acceleration, steering angle). The choice of CARLA 0.9.15 was motivated by its compatibility with autonomous driving benchmarks and its efficiency in handling large-scale dataset generation without excessive computational overhead (Niranjan et al., 2021).

By utilizing CARLA's sensor setup, including RGB cameras and LiDAR, labeled datasets were generated in a controlled environment, addressing the challenges of real-world pedestrian data

collection in complex scenarios. This study's integration of efficient deep learning strategies, contextual vehicle telemetry, and scalable simulation-based evaluation contributes to the advancement of robust, real-time, and cost-effective pedestrian detection systems for future autonomous vehicles. Key performance metrics—including mean average precision (mAP), inference speed, and false positive rates—were analysed to assess YOLO-APD's detection robustness. These evaluations confirmed YOLO-APD's computational efficiency, solidifying its role as a robust solution for real-time autonomous driving.

2.8. Convolutional Neural Networks (CNN)

One of the most substantial advancements in deep learning applications is the creation of Convolutional Neural Networks (CNNs), which have transformed the tasks of image processing into something spectacular. They have done this by performing image classification, object detection, and segmentation at levels of accuracy that are unparalleled in the history of computing. Convolutional neural networks (CNNs) have nuanced implementations that require precise technical characterization. In fact, CNNs perform out cross-correlation rather than convolution, strictly speaking. Key distinctions include no kernel flipping during the sliding operation and a computation based on dot products (i.e., element-wise multiplication followed by summation) rather than matrix multiplication. These characteristics maintain spatial relationships and reduce computational complexity.

CNNs are architecturally defined as convolutional layers with sliding window cross correlation to detect local features spatially by means of learned filters, pooling layer (sampling by max/average) to obtain spatial down sampling (translation invariance), and ReLU activation after convolution for enhancement of nonlinear features, and finally fully connected layer for

extraction of global patterns for the final classification. Nowadays, CNN architectures usually make use of stride control, padding strategies such as zero padding, channel management with 3D tensors, feature progression from edge detection to full objects, etc.

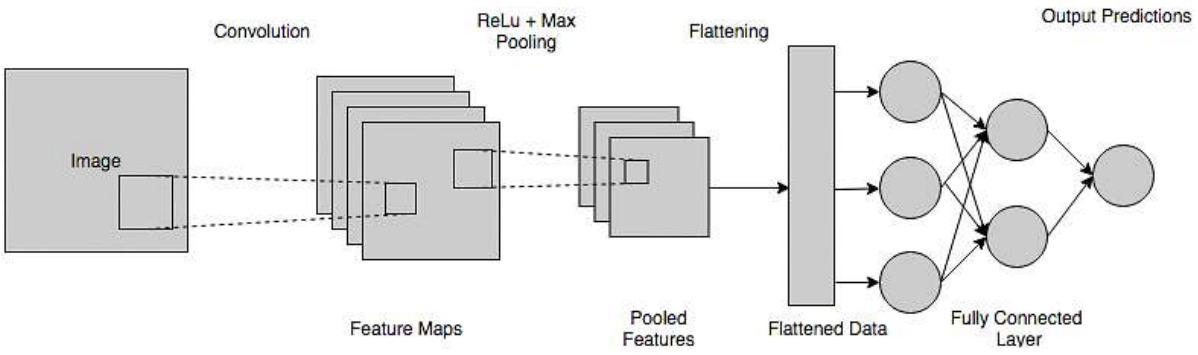

Figure 4 Convolutional Neural Networks Architecture

where $O^{(l)}$ represents activation map, $I^{(l-1)}$ is input from previous layer, $K^{(l)}$is learnable kernel and $b^{(l)}$ represents the bias term.

$$O_{i,j}^{l} = \sum_{m} \sum_{n} I_{(i+m),(j+n)}^{l-1} \cdot K_{(m,n)}^{l} + b^{l}$$

Equation 1: Convolution operation formula

Convolutional neural networks (CNNs) are extremely powerful features extraction tools from input data based on filters that can generate feature maps. Backpropagation, gradient descent, and weight update help these filters to be optimized through CNNs and learn the patterns necessary for tasks like image segmentation, object detection and classification.

Pooling or subsampling layers follow convolutions reduction aspect of dimensionality, lowering computational cost and increase robustness to add noise, rotation and positional changes on extracted features. Max pooling selects the maximum value in each region,

averaging will average. Finally, these techniques achieve feature reliability for the following processing.

Gating mechanisms are played by activation layers, which choose which signals activate neurons in the subsequent layer. Apart from activation, they provide non linearity, which helps in further capturing the complex patterns. Commonly used activation functions are ReLU (Rectified Linear Unit) and Swish which are applied after the convolutional layers. Specific applications have benefitted from a more recent function that sets negative values to zero: Swish.

The last layer of a CNN is usually fully connected where every neuron from the previous layer is linked to all neurons in the following layer, flattening the feature maps to a matrix. Finally, in classification tasks, a SoftMax layer is followed that converts outputs into the probability that each class received. This gives the probabilities an interpretable meaning as confidence scores, and it is ensured that the sum of the probabilities is one by using the SoftMax function.

An important design parameter in CNNs is the receptive field, that is, the input region that a neuron processes. Moreover, how much local and global information is captured by the model is influenced. These layers are arranged as CNN architecture, which can be customized based on the complexity, variance in data and application requirements. As such, CNNs are highly adaptable across multiple domains. For instance, in video sequences, they can predict position of objects through analyzing consecutive frames over time to determine pedestrian positions accurately. The versatility of CNN leads to the fact that they are fundamental to modern deep learning systems from image recognition and object detection to speech processing, to natural language processing. Object detection: Among all the problems related to Computer Vision, Object Detection is the most basic one. As deep learning techniques evolved, especially convolutional neural networks (CNNs) enable auto learning of hierarchical features, this field

has evolved greatly, and recently even with transformer-based architecture. Detected using the Faster R-CNN framework on resource constrained environment is the best approach because Resource less constraint environment has a very high necessity of a more robust system that include Real time. Traditional methods still use other methods like Haar cascades and Histogram of Oriented Gradients (HOG). YOLO solves the object detection problem as a direct regression problem, for which bounding boxes and class probabilities are predicted in a single pass, leading to considerable performance speed ups over real time.

2.9. Object detection.

Object detection is a fundamental task in computer vision that involves identifying and localizing objects within images or videos. Deep learning advancements, particularly CNNs, have propelled this field forward by enabling automatic hierarchical feature learning. More recently, transformer-based architecture has further enhanced detection performance. While traditional methods such as Haar cascades and Histogram of Oriented Gradients (HOG) remain relevant in resource-constrained environments, more robust frameworks like Faster R-CNN, YOLO (You Only Look Once), and SSD (Single Shot MultiBox Detector) now dominate. Notably, YOLO formulates object detection as a direct regression problem, predicting bounding boxes and class probabilities in a single pass, significantly improving real-time performance.

Modern object detection models balance accuracy and efficiency, making them suitable for applications such as autonomous vehicles, surveillance, medical imaging, robotics, and video analysis. Large-scale datasets like COCO and Pascal VOC support their development, offering diverse and challenging training scenarios.

Recent advancements focus on improving efficiency, robustness in challenging environments, and real-world adaptability. This includes optimizing architectures for speed and accuracy and

integrating transformer-based models such as DETR (Detection Transformer), which streamline detection by eliminating region proposals. Emerging techniques like zero-shot object detection leverage vision-language models such as CLIP, enabling recognition based on text descriptions without extensive labeled training data. Consequently, object detection remains a critical AI and computer vision research area, with ongoing innovations expanding its applications, such as pedestrian detection in autonomous driving.

State-of-the-art object detection architectures like YOLO and SSD significantly enhance speed and precision. Studies (e.g., Ali et al., 2022; Meng et al., 2022; Tian et al., 2022) demonstrate that these detectors employ region proposal algorithms, feeding selected image regions (as pixels or precomputed neural features) into multiway classifiers. Object detectors are broadly categorized into single-stage (YOLO, SSD) and two-stage detectors (Faster R-CNN, Mask R-CNN) as shown below.

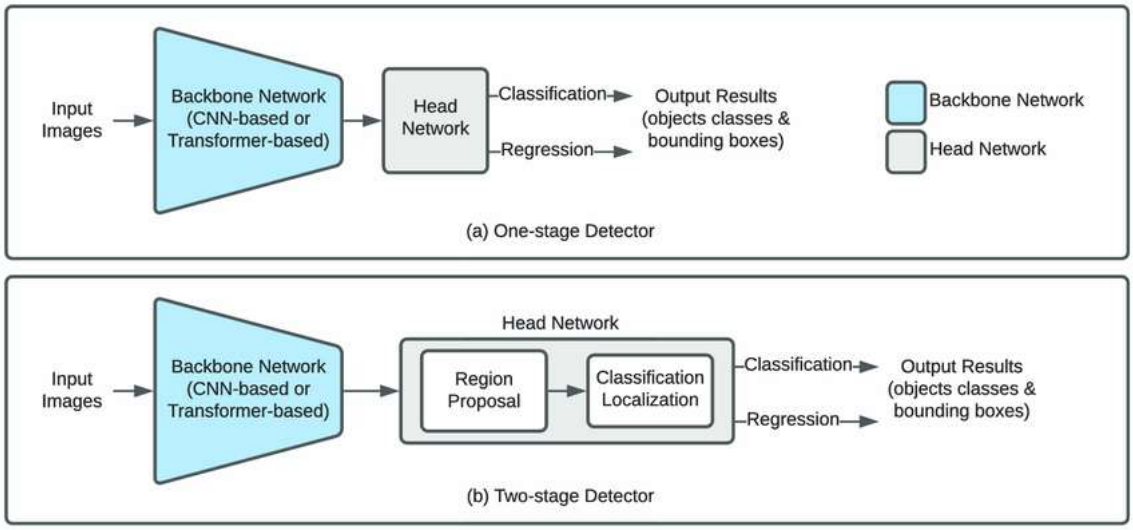

Figure 5 One Stage detector and Two Stage detector (image Adapted from Research Gate)

According to Y. Li et al., (2021) selection between these types depends on speed versus accuracy trade-offs. Single-stage detectors prioritize speed, making them suitable for real-time applications, while two-stage detectors emphasize precision, essential in fields like medical imaging, whereas minute details impact outcomes.

2.10. The YOLO (You only Look Once) algorithm and its evolution

YOLO is a state-of-the-art model widely recognized as a family of detectors known for its efficiency and speed (Terven & Cordova-Esparza, 2023). YOLO divides the input image into a grid of cells and predicts a fixed number of bounding boxes alongside a confidence score for every cell. It can also predict the lesson probabilities for every bounding box and merge them with the confidence scores to provide final detection results as shown in the figure 5 below. While YOLO is renowned for its compelling speed and execution on large and medium-sized objects it has some limitations, including low resolution, poor localization and inability to effectively detect smaller objects (Varghese & Sambath, 2024).

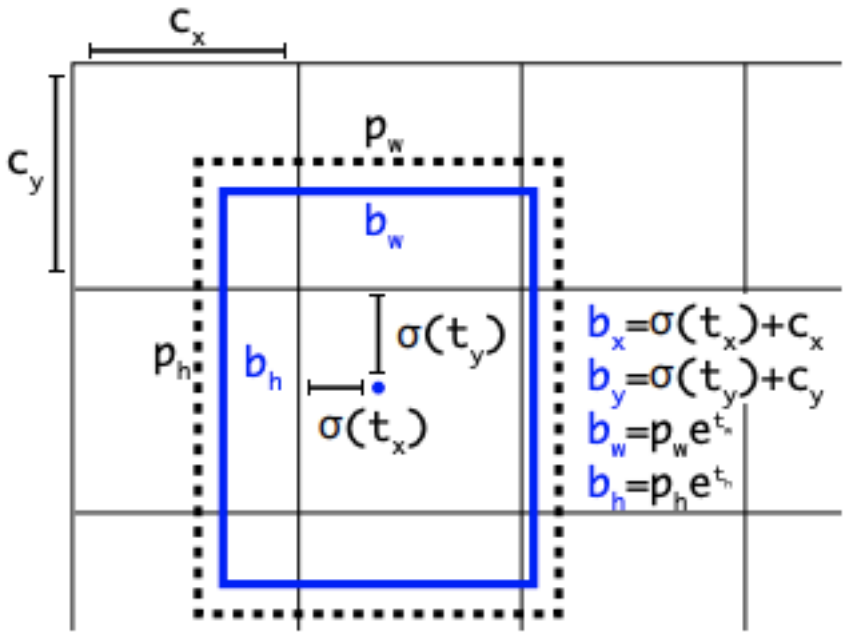

Figure 6 Bounding box adapted from roboflow as cited by *(Khanam & Hussain, 2024)*

The first member of the YOLO family was introduced in 2016 by Joseph Redmond as YOLOv1. YOLOv1 was grid-based and had trouble with not only detecting small objects but also detecting objects that were densely packed. This reason essentially led to the creation of

the next model in the series, YOLOv2 (or YOLO9000). YOLOv2 was a big step forward compared to its predecessor, not just in terms of design philosophy but also in accuracy, and it employed something called anchor boxes, which are used in object detection by majority of the state of art detectors(Terven & Cordova-Esparza, 2023) .The limitations of YOLOv2 gave birth to YOLOv3 which like its predecessor suffered from challenge with smaller objects due to its grid based approach, in which each grid cell predicted a limited number of bounding boxes. YOLOv4 (Bochkovskiy et al., 2020), was sooner developed to circumvent these challenges, bringing speed and better accuracy compared to its predecessors, however it too degraded under varying environmental conditions besides persistent problem of detecting small objects in complex or crowded place, these issues let to researchers coming up with YOLOv5.This new entrant was optimized for efficiency using CSPDarknet53, SPP, and self-adversarial training to balance speed and accuracy; moreover, YOLOv5 was developed independently by Ultralytics, with a switch to PyTorch, adding Mosaic augmentation, and providing multiple model sizes for flexibility (Khanam & Hussain, 2024). Later iterations such as YOLOv6 and YOLOv7 focused on increasing the accuracy of instance segmentation and making the models lighter and faster to run on edge devices such as mobile phones.

2.11. Related Works

Deep learning has proven to be a revolutionary force in enhancing pedestrian detection in intelligent vehicles in the last decade. Two approaches can be used to investigate pedestrian detection in intelligent vehicles using deep learning techniques: non-line of sight (NLOS) detection techniques and line of sight (LOS) set of detectors. Line of sight approaches can be used in scenarios when the autonomous vehicle can identify pedestrians inside its range of vision. The identification of pedestrians outside the AV's field of vision is accommodated by NLOS systems in which unexpected traffic conditions, such circumstances are typical (Verstraete & Muhammad, 2024). In light of the ever-evolving needs of users and race towards

social acceptability of the Avs, the research community has started looking on NLOS detection techniques more recently.

Several works have explored pedestrian detection in autonomous driving using deep learning techniques over the decade using RGB images, thermal images and multiple sensor images. Majority of these publications have laid more emphasis on improving the overall structure of the detection component for reliable pedestrian detection as demonstrated by Xu et al., (2020). In the last decade, RGB based deep learning pedestrian detection techniques have been popular among AV's for example in their work Pranav & Manikandan, (2020), proposed a real time pedestrian detection model for autonomous vehicle built on CNN architecture without inclusion of standard feature extraction head for object detection. They claimed accuracy levels between 82.73% which they argued was in tandem with existing Realtime detection architectures. Tarchoun et al., (2020) proposed a method focused on Faster R-CNN utilizing transfer learning based on various pre-trained architectures to efficiently detect pedestrians. However, the problem of poor detection in complex scenes persisted, posing additional challenge to detection accuracy this motivated further research for a possible solution.

Although research works that utilize RGB images for pedestrian detection have been at the fore front they experience accuracy challenges in poor visibility such as in night-time scenes and Bright sun glare scenarios posing a major challenge in autonomous driving(Uřičář et al., 2020) prompting interest in exploitation of other sensual signals such as infrared images(Huo et al., 2022; W. Li, 2021; Song et al., 2020) for pedestrian detection. A notable advantage of infrared based methods is the ability to easily differentiate objects based on level of heats they emit, making pedestrians easily differentiable from buildings. In their work (Manda et al., 2020) Proposed a pedestrian detection model using thermal images using raised cosine distribution based on image segmentation method. They collected pedestrian images in different atmospheric conditions and subjected their model to assess its performance. They claimed

evaluated results through quantitative parameter sensitivity methods and promising outcomes compared with other state of art methods.(Song et al., 2020) proposed a three-tier end to end infrared detection model utilizing CSP (Centre and Scale Prediction) network, they designated full-time, daytime and night, sub-detection components utilizing a standard ResNet-50 backbone pretrained on ImageNet dataset. They claimed the results indicated the daytime component recording the highest optimal outcome.

However, a notable misgiving of thermal images is lack of color and texture and overall poor image quality than RBG images besides unpredictability in circumstances where objects emit considerable heat leading to confusion with pedestrians. To overcome these challenges researchers further explored use of different sensor images(Abbasi et al., 2023; Jhong et al., 2022; Sahba et al., 2020; Zhao et al., 2020) fused together to improve pedestrian detection. Several studies indicate model performance and optimal fusion exploration, can be improved significantly through utilization of multispectral fusion (Chan et al., 2023). For example, (Zhao et al., 2020) proposed a novel sensor fusion object detection model utilizing information from lidar sensors which detects in three-dimension and vision cameras to obtain a robust object detection. They used lidar to obtain an accurate object region proposal then mapped the results onto image space from which region of interest were selected and fed into convolutional neural network. They claimed in their findings a reduced extraction time of 66.79ms which met Realtime demand for autonomous vehicles.

Although Conventional CNN-based algorithms have formed the basis of pedestrian and vehicle detection in autonomous driving in the last decade, research has shown that they suffer from limited representation capabilities leading to degraded performance in complex scenes resulting from use of same kernel weight for all inputs. To mitigate the above challenges, Chen Y et al (2020) proposed dynamic convolution which calculated the different weights per kernel based on input image thereby enabling better feature representation. However, the problem it

increased network complexity leading to large number of parameters. To solve these problems Li et al. (C. Li et al., 2022) in their work, suggested a new improvement to dynamic convolution architecture known as omni-dimensional, which learned attention in much flexible manner leading to overall efficient network performance than its predecessors. the whole concept of dynamic convolution manipulates convolutional kernels by applying soft attention and adjusting the kernel parameters based on varied scene input. this means that, using prior knowledge of samples, the dynamic convolutional neural networks can improve performance of a model through much better feature representation.

Considering these challenges, several scholars explored the potential inclusion of attention mechanism (Huang et al., 2022; Lee & Cho, 2022; W. Li et al., 2020; Xue et al., 2023) in pedestrian detection models to mitigate the challenges of high missed detection rate in scene with varied complexity. Some of these works reported promising results with improved detection rates, for example, Shen et al., (2022) presents a modest pedestrian detection model using an improved YOLO (You Look Only Once) architecture known as eca-Yolov5. In their work, the authors replaced the original bottleneck CSP module with a transformer block and incorporated self-designed, efficient coordinate attention to enhance the model's feature detection field. The results were encouraging in comparison to the existing state of art. In this study, however we will focus on pedestrian detection based on vehicle dynamics, dynamic convolution, attention mechanism with yolov8 architecture as a base.

2.12. Gaps in Knowledge

While the advances in pedestrian detection summarized above are notable, a review of the literature reveals several key remaining knowledge gaps that this study aims to bridge. Firstly, while deep learning has become ubiquitous, many existing studies remain largely limited to conventional, foundational neural network architectures. These architectures, while effective, may be constrained in representational capacity when faced with the challenges of real-world

driving environments, particularly in handling occlusions, dynamic lighting conditions, and varying pedestrian appearances. In particular, the ability of these models to effectively capture and utilize contextual information remains an open question.

Secondly, although the use of context has been increasingly recognized, there is a lack of specific research integrating vehicle dynamics information (e. g. steering wheel angle, vehicle speed, vehicle acceleration) into pedestrian detection systems. The inclusion of vehicle dynamics information could provide useful context about the vehicle's intended path and potential interactions with pedestrians which could improve the perception accuracy and reduce false positives. This is particularly true in complex and ambiguous road geometries like type S roads where driver intent and vehicle manoeuvring are crucial for understanding pedestrian risk.

Many of the existing high-performing approaches rely on computationally demanding and expensive sensor technologies (e. g., LiDAR and multi-sensor fusion) that increase robustness, but can reduce the likelihood of large-scale deployment in consumer autonomous vehicles due to their cost and computational demands. Thus, scalable and cost-efficient pedestrian detection models that maintain good accuracy and real-time performance using more accessible and affordable sensor suites (e. g., monocular cameras or cost-effective stereo setups) remains an important research direction for democratizing autonomous driving technology.

This project contributes to ongoing research in pedestrian detection by directly addressing these gaps, exploring the potential of incorporating dynamic convolution and attention mechanisms within a deep learning framework to increase representational power and improve detection performance in road environments. In addition, this work will explore the use of vehicle dynamics data as a complementary input to the pedestrian detection model to improve contextual understanding and accuracy, particularly in challenging scenarios. Finally, by

focusing on efficient model design and potentially leveraging more accessible sensor data, this work contributes towards the development of robust, adaptable, efficient and importantly, more scalable pedestrian detection solutions for wider autonomous driving applications.

CONCEPTUAL MODEL

This study leveraged YOLOv8 as the main deep learning technology while utilizing an affordable RGB stereo camera as the source of perceptual input. YOLOv8 represents one of the latest advancements in the You Only Look Once (YOLO) object detection family, as reported by (Yaseen, 2024a) , in his exploration of internal features of the next generation object detector. Its development was inspired by the evolution of the family from YOLOv1, YOLOv2...YOLOv7, With v8 riding on improvements tailored especially for small object detection which had been an issue for a very long time, YOLOv8 addresses problems that had been identified in yolov7. Though instance segmentation was first presented by YOLOv7 it was improved upon in YOLOv8, which has an anchor-free structure(Varghese & Sambath, 2024). YOLOv8 does not use any predefined bounding box format. Instance segmentation is when the model is able to not only detect an object but also segment the object instance from the background. For example, if there are two people in your image, then segmenting would mean defining the boundary of each person such that the two segmentations do not overlap.

The architecture of YOLOV8

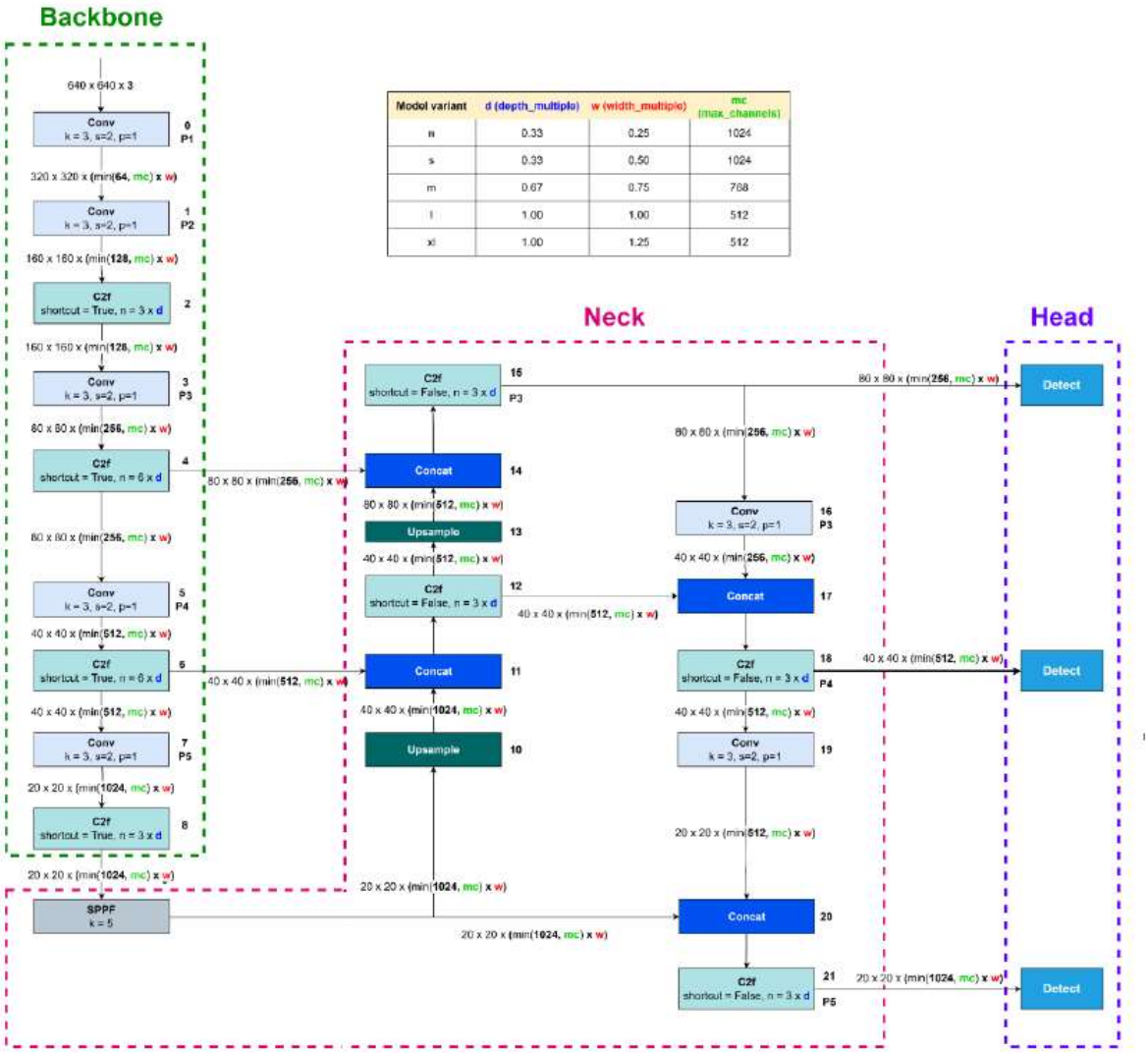

Figure 7 YOLOv8 Architecture

Represented as a state-of-the-art object detection framework, YOLOv8 is optimized for real-time applications and is built around a modular architecture that balances detection accuracy and computational efficiency. What is most noteworthy about YOLOv8, compared to earlier models, is that it introduces a tight, lightweight backbone, which is a convolutional neural network (CNN) on which the entire architecture is built (Terven & Cordova-Esparza, 2023). This new backbone was inspired by CSPNet. Intuitively, this is a faster new way of doing the same thing, and, because of that, there's less inference time and also just basically less of a

computational burden incurred in terms of Floating Point Operations(FLOPs). The neck module contains the Cross-Stage Partial Fusion, or C2f, module, which is an optimized operating system for feature reuse. This mechanism enhances gradient flow and improves the efficiency of the entire YOLOv8 object-detection system. Moreover, YOLOv8 contains an improved version of the feature pyramid-based aggregation system . This enhancement allows the system to perform better when detecting properly scaled objects, particularly small ones.

The detection head that is free of anchors predicts bounding boxes dynamically using the center points and offsets, which are more to the liking for the localization task. They optimize Generalized Intersection over Union (GIoU) and Complete IoU (CIoU) losses that make the bounding box predictions much better. Recognizing that the center point is not the best image region to anchor the spatial attention mechanism, the authors integrate a spatial attention mechanism that looks at other, more relevant parts of the image. Indeed, they also do some clever things with regard to the training strategy. To meet the requirements of the diverse deployment requirements, YOLOv8 has various variants such as YOLOv8n (nano), YOLOv8s (small), YOLOv8m (medium), YOLOv8l (large), and YOLOv8x (extra-large). These ensure adaptability from edge devices to high-performance computing in various environments. YOLOv8, with these enhancements over its predecessor, solidifies itself as the go-to tool for high-speed, high-accuracy real-time object detection in autonomous systems, surveillance, and robotics.

Limitations of Yolov8

Although YOLOv8 is very strong in real-time object detection, its weaknesses can be alleviated by slight refinements for optimal results in specific use cases. The first limitation lies within localization in cluttered environment where accuracy is impaired by overlapped objects and tiny detections. Increasing the accuracy of detection can be done through improvements of adaptive tiling methods and transformer-based attention modules (Transformers, e.g., Swin

Transformer). Even though it has incorporated instance segmentation properties, YOLOv8s performs less well than models such as Mask R-CNN. Augmenting segmentation features with deformable DETR (X. Zhu et al., 2020) similar style masks, or adding a lightweight mask branch as in YOLACT, to increase per pixel precision for use in critical application such as critical pedestrian detection . further its performance is degraded in clutter and poor illumination when performing detection in low light and in complicated backgrounds. Finally, the detection robustness can be improved by including incorporating self-attention mechanisms such as SimAM (Yang et al., 2021), domain adaptation through using CycleGAN, and augmented augmentations, to name a few. The spatiotemporal attention mechanisms or the improved post-processing using DeepSORT or Kalman filters can significantly increase the tracking accuracy. YOLOv8's default built-in object recognition limit of about 300 objects per image may also hinder the application of the technique to applications that need high-density item counting, such as traffic monitoring and agriculture. It can be alleviated using Soft-NMS/Adaptive-NMS to reduce false positive detections in a high-density scene, the use of dilated convolutions, and Cascade R-CNN approaches. Another viable approach is to change its backbone to MobileNetV3 or GhostNet (Han et al., 2019), to lower the processing cost while preserving accuracy. By carefully improving its architecture it to specific use cases, YOLOv8 could be made versatile for the autonomous vehicles, surveillance, and industrial automation. This will ensure improved accuracy, speed, and universality in real application.

CHAPTER THREE

RESEARCH METHODOLOGY

3.1. Introduction

This section begins with a brief introduction on the on the detail and justification for the choice of research methodology used, alongside the principles, rules and procedures adopted in this work. This involved designing, developing and implementing an adaptive pedestrian detection algorithm for autonomous vehicle on complex road scenes.

3.2. Research Design

In view of the research objectives, this work adopted elements of design science research methodology (DSR). The choice for this approach was guided based on DSR's widely acknowledged application in most Information System (IS) workflows as demonstrated by Agung Premananda et al. (2022) and van der Merwe et al. (2020). Design Science Research Methodology is a problem-solving oriented paradigm that advocates for an iterative build and evaluation cycle, which means that if an artifact won't satisfy a requirement, then the process can be restarted, or the appropriate design element can be changed to satisfy the requirement.

DSR process structured approach emphasises six steps: problem identification and motivation, objectives formulation for a solution, artefact design and development, demonstration, evaluation and communication of the findings. Further, four entry points are also advocated for in this process: problem-centred initiation, objective cantered initiation, design and development-centred initiation, and context initiation as shown in Figure 4, to ensure reproducibility, validity and reliability in achieving the study objectives. Each DSR activity is briefly discussed in the following section.

3.2.1. Problem Identification and Objectives

In this phase, the existing challenges of pedestrian detection in autonomous driving were studied in detail. The problem statement underscored the limitations of the existing detection models especially the difficulties in detecting small pedestrians, occlusion and delays in dynamic situations. This study was motivated by the need to improve pedestrian detection using low-cost but effective sensors while leveraging on vehicle steering dynamics for advancement of autonomous vehicles. Further the highlighted issues were addressed by establishing clear, quantitative objectives as shown in chapter one which guided the next phase.

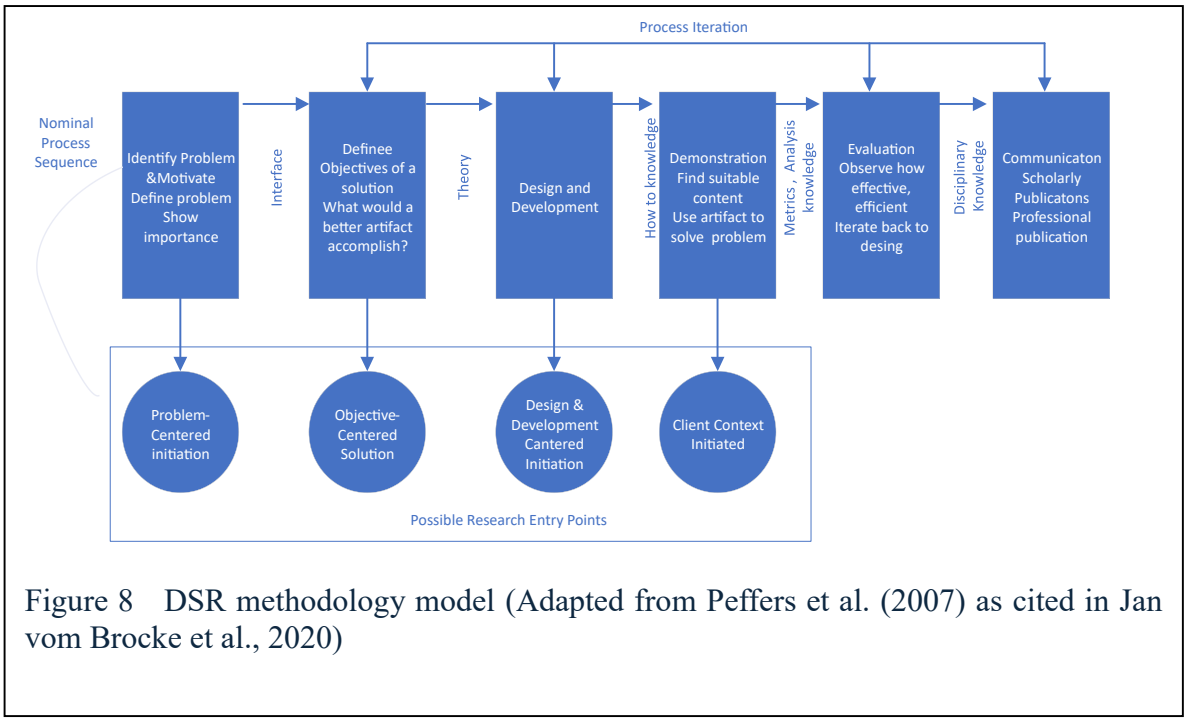

Figure 8 DSR methodology model (Adapted from Peffers et al. (2007) as cited in Jan vom Brocke et al., 2020)

3.2.2. Artifact design and Development

The primary artefact was identified as an adaptive pedestrian detection algorithm and YOLOv8 was chosen as a baseline architecture because of its real-time detection capabilities, community support and proven reliability. To support the modifications required for real-time ROI

adjustments, the Pytorch framework (Stevens et al., 2020), was used because of its rich ecosystem, straightforward support for CUDA which supports complex mathematical operations such as training complex deep learning models and large community support. This involved developing techniques that enabled the algorithm to dynamically modify the Region of interest (ROI) based on the steering wheel angle, improving detection efficiency in real-time scenarios. In order to effectively address the issues identified in the problem, the design focused on incorporating affordable sensors and ensuring that the system could be adapted to different road geometry and accommodate for unpredictable human behaviour.

3.2.3. Demonstration and Evaluation

The performance of the proposed algorithm was tested in a proof-of-concept environment using the CARLA simulator (version 0. 9. 15). This allowed the simulation of different urban scenarios and weather conditions that simulated the complexity of type S roads. By showing the behaviour of the algorithm in these conditions and verifying the focus region as the vehicle steering changed, the research validated the practical implementation of the proposed solution and showed its potential effectiveness in real world applications. A broad evaluation of the algorithm's performance was conducted and comparative study carried out against state-of-the-art pedestrian detection model to understand the improvements achieved this then informed the next phase which involved publishing the findings in a peer review platform.

3.3. Research Tools and Instruments

This research utilizes a variety of specialized instruments to effectively accomplish the research objectives. Every tool was thoughtfully chosen according to how well it suited the tasks required for dataset creation, model building, and evaluation.

3.3.1. CARLA Simulation Engine

To create high-fidelity pedestrian detection datasets, CARLA Simulator was selected as the ideal platform, since it is based on Unreal Engine 4 (UE4) runtime with a capability to replicate a wide range of road conditions, weather patterns, and lighting as shown in , settings besides being able to run on client and server mode, thanks to its robust rendering capabilities. Carla and UE4 leverage the power of C++ for its server architecture. With this configuration, a realistic and thorough dataset that is suited to this particular study was reproduced. The Python script used to initiate the simulator was run in Visual Studio Code (VS Code) and controlled by Windows PowerShell. The workflow's smooth integration with other tools was made possible with this configuration. It is established by various studies that the CARLA simulator is useful in autonomous driving research, especially in pedestrian recognition and behaviour modelling, as demonstrated by (Martín Serrano et al., 2024).

Figure 9 Carla Diverse Environment Simulation

3.3.2. Cloud Computing and Training Environment

Kaggle's cloud-based interactive computing platform was used for effective model training owing to its user-friendly approach, scalability, and open-source access to its powerful NVIDIA Tesla P100 GPU (with 16GB RAM). Its Jupyter-based notebooks were used to facilitate the creation of the training scripts and repeatable training and smooth integration with our datasets. During training, the notebooks enabled debugging, hyperparameter adjustment, and real-time result display.

3.3.3. Development Environment

Visual Studio Code (VS Code) played a crucial role in creating and debugging code associated with CARLA simulation scripts. Its vast extension library facilitated effective code management. Its Windows PowerShell extension streamlined the workflow by managing the dependencies and enabling server-side in Unreal Engine.

3.3.4. Version Control

GitHub is employed for version control, collaboration, and tracking model improvements. This ensures that all changes to the codebase are documented and accessible, facilitating collaboration among team members while maintaining a transparent development process.

GitHub was used for forking the YOLOv8 base model, recording the model improvement, and version control. This guaranteed that all modifications to the source code were available and documented.

These tools were then integrated in a cohesive workflow as summarized in the table below

Table 1:Tools integration

Tool	**Integration**
Dataset Generator	Carla Simulator powered by Unreal Engine
Model development	Pytorch model developed locally but trained in Kaggle Cloud

Training Execution	All training scripts were written and executed within Kaggle Notebook to leverage its interactive environment and GPU resources
Code Management	VS Code and PowerShell manage code development and dependencies for CARLA-related tasks.
Collaboration	GitHub ensures version control thought the projects lifecycle

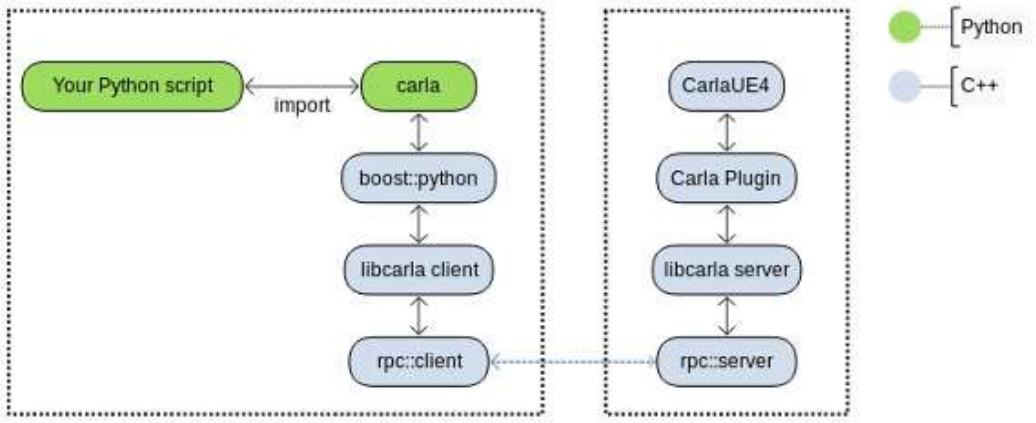

Figure 10 How Carla integrates with Python

3.4. Improved Algorithm: Yolo-APD

Aiming at special needs of pedestrian recognition task, this study carried out an in-depth optimization of the YOLOV8 model, innovatively integrating several advanced modules and proposed a pedestrian detection model named YOLO-APD, and its architecture is shown in figure 4. First replacing traditional convolution with CBM module which utilizes Mish activation instead of SiLu activation function. Replacing the head of YoloV8 with much optimized modules that utilize gather and distribute mechanism (Wang et al., 2023) , to improve the overall detection accuracy.

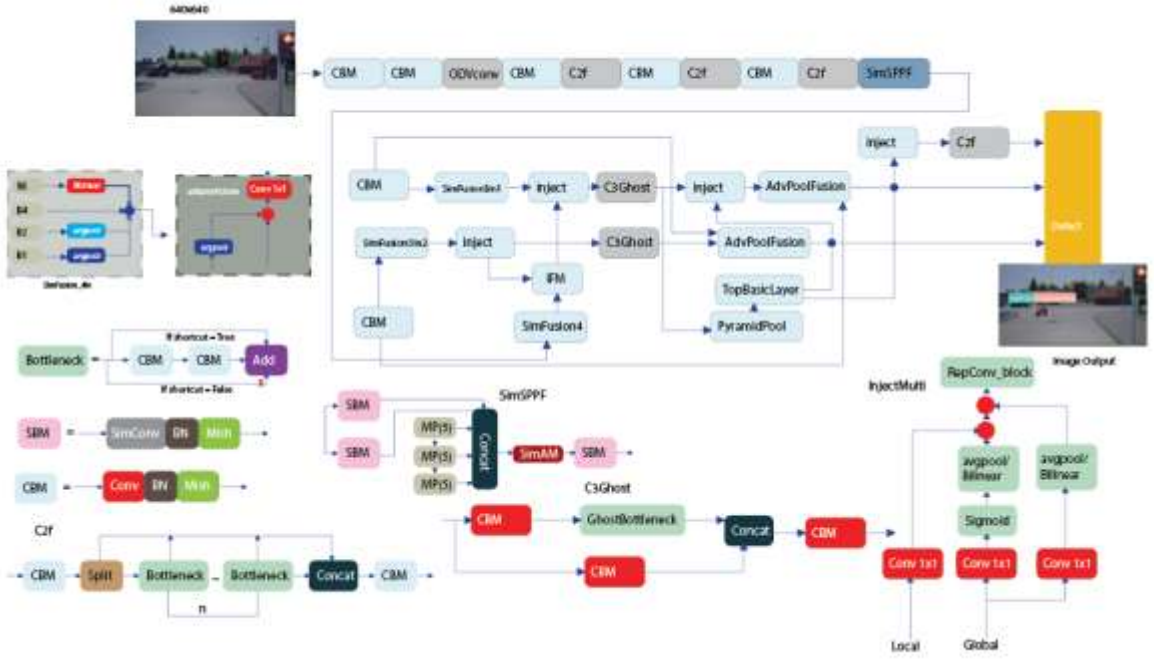

Figure 12 The structure of YOLO-APD

```
Overriding model.yaml nc=80 with nc=10
WARNING ⚠️ no model scale passed. Assuming scale='m'.

                 from  n    params  module                                       arguments
  0                -1  1      1392  ultralytics.nn.modules.conv.Conv             [3, 48, 3, 2]
  1                -1  1     41664  ultralytics.nn.modules.conv.Conv             [48, 96, 3, 2]
  2                -1  2    111360  ultralytics.nn.modules.block.C2f             [96, 96, 2, True]
  3                -1  1    166272  ultralytics.nn.modules.conv.Conv             [96, 192, 3, 2]
  4                -1  4    813312  ultralytics.nn.modules.block.C2f             [192, 192, 4, True]
  5                -1  1    664320  ultralytics.nn.modules.conv.Conv             [192, 384, 3, 2]
  6                -1  4   3248640  ultralytics.nn.modules.block.C2f             [384, 384, 4, True]
  7                -1  1   1991808  ultralytics.nn.modules.conv.Conv             [384, 576, 3, 2]
  8                -1  2   3985920  ultralytics.nn.modules.block.C2f             [576, 576, 2, True]
  9                -1  1    831168  ultralytics.nn.modules.block.SimSPPF        [576, 576, 5]
 10       [2, 4, 6, 9]  1         0  ultralytics.nn.modules.conv.SimFusion_4in    []
 11                -1  1    407616  ultralytics.nn.modules.conv.IFM              [1248, [64, 32]]
 12                 9  1    221952  ultralytics.nn.modules.conv.Conv             [576, 384, 1, 1]
 13         [4, 6, -1]  1    517632  ultralytics.nn.modules.conv.SimFusion_3in    [[192, 384, 384], 384]
 14           [-1, 11]  1    265216  ultralytics.nn.modules.conv.InjectionMultiSum_Auto_pool[384, 512, [64, 32], 0]
 15                -1  2    390816  ultralytics.nn.modules.block.C3Ghost         [512, 384, 2, False]
 16                 6  1     74112  ultralytics.nn.modules.conv.Conv             [384, 192, 1, 1]
 17         [2, 4, -1]  1    129792  ultralytics.nn.modules.conv.SimFusion_3in    [[96, 192, 192], 192]
 18           [-1, 11]  1     67072  ultralytics.nn.modules.conv.InjectionMultiSum_Auto_pool[192, 256, [64, 32], 1]
 19                -1  2    100176  ultralytics.nn.modules.block.C3Ghost         [256, 192, 2, False]
 20        [19, 15, 9]  1    406208  ultralytics.nn.modules.conv.PyramidPoolAgg   [1152, 352, 2]
 21                -1  1   2222528  ultralytics.nn.modules.conv.TopBasicLayer    [352, [64, 128]]
 22           [19, 16]  1         0  ultralytics.nn.modules.conv.AdvPoolFusion    []
 23           [-1, 21]  1    132608  ultralytics.nn.modules.conv.InjectionMultiSum_Auto_pool[384, 256, [64, 128], 0]
 24                -1  2    100176  ultralytics.nn.modules.block.C3Ghost         [256, 192, 2, False]
 25           [-1, 12]  1         0  ultralytics.nn.modules.conv.AdvPoolFusion    []
 26           [-1, 21]  1    429056  ultralytics.nn.modules.conv.InjectionMultiSum_Auto_pool[576, 512, [64, 128], 1]
 27                -1  2   1821696  ultralytics.nn.modules.block.C2f             [512, 384, 2, False]
 28       [19, 24, 27]  1   2896750  ultralytics.nn.modules.head.Detect           [10, [192, 192, 384]]
```

Figure 11 Layer wise architecture breakdown of YOLOAPD

3.5. Contributions

This work introduces below novel improvements to the architecture of baseline YOLOv8 to enhance its performance in adaptively detecting objects in a complex scene such as type S roads.

3.5.1. Attention Mechanism: SimAM

In this project an attention module which is parameter free, known as SimAM attention module (Yang et al., 2021), was used to enhance feature extraction for the proposed model. This module is inspired by studies of the mammalian nervous system that discovered neurons of higher elevation domineering effects on peripheral neurons. Thus, this module manipulates feature maps in CNN layers through generation of 3D attention weights(Koohpayegani & Pirsiavash, 2022). Different attention is given to the various areas of interest, depending on the significance of the information carried by neurons in those regions. This causes the network to prioritize important characteristics within the target zones while improving the target's feature representation. Unlike typical methods, the module generates 3D weights without the need for extra subnetworks. This is enabled by directly calculating the weights using a particular energy function. The 3D weights for the present neuron can be inferred from the analytical solution specified by the function. All neurons of the same dimension are treated equally by the current attention module, which is worth mentioning. While taking into account the significance of both channel and spatial dimensions, this module assigns unique weight to each neurons, this is consistent with the properties of human attention. Additionally, the SimAM mechanism was introduced into the backbone to verify its generalization in different tasks.

Finding neurons containing more significant information in visual neuroscience involves figuring out if target neuron is linearly separable from nearby neurons. Neurons carrying varying quantities of information fire in distinct ways. The formulation of the energy function

can be minimized to achieve linear separability. Initially, the energy function is explained following the addition of the regularization term as follows:

$$e_{t(w_1,b_t,y,x_i)}=\frac{1}{M-1}\sum_{i=1}^{M-1}(-1-(w_t x_i+ b_t))^2+(1-(w_t t+ b_t))^2+\lambda w_t^2$$

Equation 2:Energy Function in SimAM attention

where t and x_i are the target and surrounding neurons of the input feature $X=R^{C\times H\times W}$

in a channel; $M=H\times W$ indicates how many neurons are present in that channel; and i denotes the exact loci of the spatial location. w_t and b_t are the weight and bias transforms (Yu et al., 2023). Further, the specific forms of the analytical solutions of w_t and b_t are defined using Equations x and x.

$$b_t = -\frac{1}{2}(t-\mu_t)w_t$$

$$w_t = -\frac{2(t-\mu_t)}{(t-\mu_t)^2 + 2\sigma_t^2 + 2\lambda}$$

Equation 3: Analytical solutions of w_t and b_t

Where $\mu_t = \frac{1}{M-1}\sum_{i=1}^{M-1} x_i$ and $\sigma_t^2 = \frac{1}{M-1}\sum_{i=1}^{M-1}(x_i + \mu_t)^2$ represent the mean and variance of each neuron except *t*, respectively. It then follows that , the minimum energy of the system can be calculated using the below formula:

$$e_t^* = \frac{4(\hat{\sigma}^2 + \lambda)}{(\mathrm{t} - \hat{\mu})^2 + 2\hat{\sigma}^2 + 2\lambda}$$

Equation 4:The minimum Energy of neural in SimAM

Here, $\hat{\mu}=\frac{1}{M-1}\sum_{i=1}^{M-1} x_i$ and $\hat{\sigma}^2 = \frac{1}{M-1}\sum_{i=1}^{M-1}(x_i + \mu_t)^2$. Equation x demonstrates that the difference between the target neuron and other neurons is more noticeable at lower energies,

$$\tilde{x} = sigmoid\left(\frac{1}{E}\right)\Theta x$$

Equation 5:Sigmoid function in SimAM

therefore the importance of the target neuron may shown using $\frac{1}{e_t^*}$. At this point, E is used to define $\frac{1}{e_t^*}$ across all spatial dimensions and channels, followed by application of the sigmoid to $\frac{1}{E}$ as 3D weight and multiply it by the original input feature.

3.5.2. C3Ghost Module

C3Ghost leverages the strength of both C3 and Ghost modules, aimed at reducing computational load while maintaining effective feature extraction capabilities. In order to

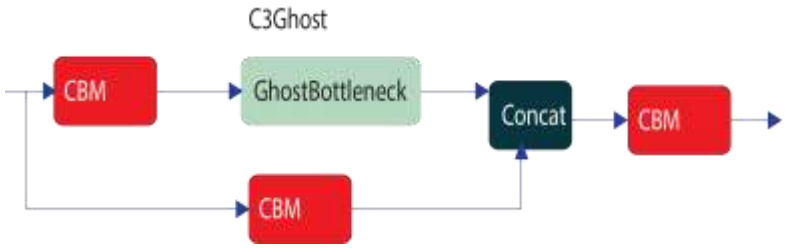

Figure 13. C3Ghost Module

enrichen deeper networks while maintaining balance between performance and speed we change the activation function to Mish. Code for C3Ghost is …

3.5.3. SimSPPF Module

Since YOLOv8 simplifies the number of CBS basic convolution modules in SPPF, as shown in Fig. 4(a), it causes the loss of model detection precision. To maximize the detection precision of the model with a small increase in the number of parameters and obtain a very "cost-effective" module, this article proposes a novel SimSPPF module, whose structure and pseudocode are shown in Figure 15 and its pseudocode in Table 1.

The major difference between our novel SimSPPF and the baseline SPPF lies in the activation function used within the module besides addition of SimAM attention mechanism. While SPPF employs the SiLU activation function, our SimSPPF incorporates the Mish activation function. Using SimSPPF yielded better detection accuracy compared to SPPF as evidenced in the ablation section.

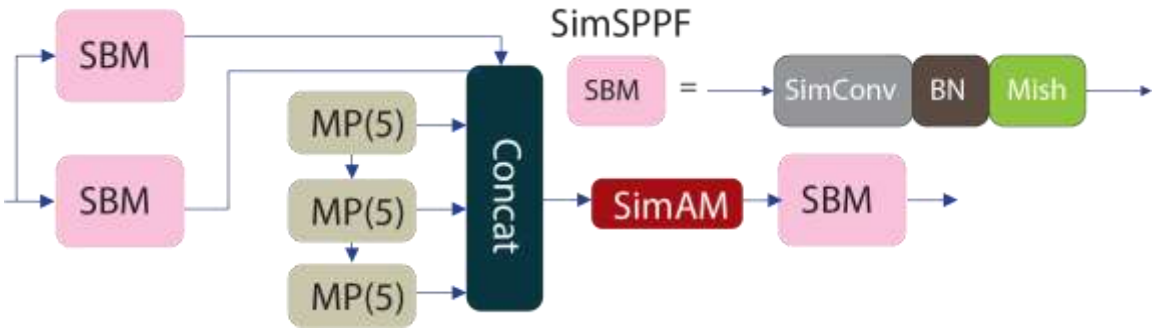

Figure 14. SimSPPF

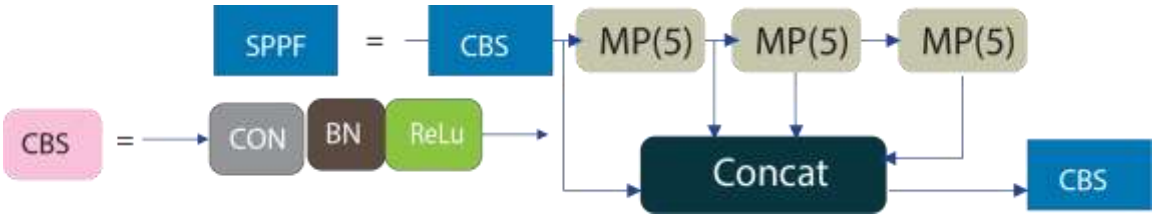

Figure 15. SPPF

3.5.4. Mish Activation Function

Mish is a smooth, continuous, self-regularized, non-monotonic activation function(Misra, 2019):

$$f(x) = xtahh\big(softplus(x)\big) = xtah(\ln(1 + e^{x}))$$

Equation 6:Mish Activation function

Mish is bounded below at around 0.31 and unbounded above this feature eliminates by design the preconditions necessary for the Dyring ReLu phenomenon. Being unbounded above, Mish avoids saturation, which generally causes training to slow down. Unlike ReLu, Mish is continuously differentiable, a property that is preferable because it avoids singularities (Dasgupta et al., 2021). In this paper on the proposed YOLO-APD we replace Relu activation function in all Convolutional blocks with Mish with an aim to improve the detection robustness of the model .

3.5.5. Gather and Distribute Module

In this project, Yolov8's neck is substituted with few components of Gold-Yolo(C. Wang et al., 2023b) mainly the gather. In this project we introduce injection-multi and Sim4modules to optimise the model by incorporating an innovative gather-and-distribute mechanism, particularly beneficial for improving the detection of anomalies within complex scenarios.in YOLO series FPN(W. Wang et al., 2021) is used which comprises multiple branches for multi-scale feature fusion propagating bits of representation from lower domains to higher levels, this transfer mechanism can result in a significant loss of information during calculation.

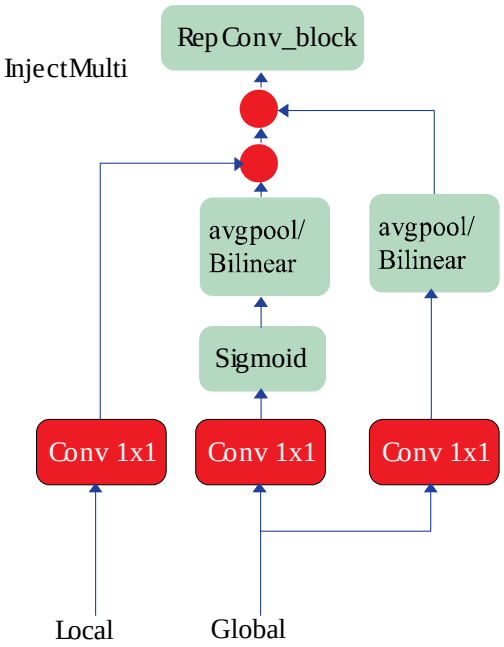

Figure 16: Gather and Distribute Module

Only information chosen by intermediate layers can be exchanged during information interactions between layers; unselected information is rejected during transmission. As a result, information at one level can only effectively support neighbouring levels, which reduces the support given to other global layers. Consequently, the information fusion's overall efficacy might be constrained.

However, to enhance pedestrian identification in challenging scenarios, such as S-shaped roads, we developed an optimized variant that was named YOLO Adaptive Pedestrian Detector (YOLO-APD). YOLO-APD integrates efficient coordinate attention and omni-dimensional

convolution into its gather-and-distribute module, significantly improving detection capabilities while maintaining real-time performance. Unlike conventional YOLO architectures, which may struggle with occlusions, fluctuating lighting, and complex pedestrian behaviour, YOLO-APD leverages these enhancements to improve robustness.

Studies have demonstrated that YOLO-based models achieve real-time inference with high precision, making them well-suited for autonomous driving applications (R & S, 2021; Xiao et al., 2021). A key advantage of YOLO architectures is their computational efficiency over two-stage detectors like Faster R-CNN while still achieving competitive accuracy (Niranjan et al., 2021) . Recent advancements, such as dynamic convolution and attention mechanisms, further refine feature representation and adaptability in complex environments (Chen et al., 2020; Guo et al., 2021). The superiority of our algorithm over existing models lies in its ability to adapt to diverse pedestrian appearances, varying road topologies, and dynamic environmental conditions. Unlike the baseline YOLOv8 model, YOLO-APD integrates contextual enhancements to refine detection accuracy in intricate urban settings, where precise identification is critical. Additionally, this study explored how pedestrian recognition can be improved by incorporating vehicle telemetry data, such as steering angle. Our study used the steering wheel angle to predict the region ahead where potential hazards may emerge. The captured image is divided into an upper half and a lower half, and based on the vehicle's steering angle, the system determines the area that is not directly in front of the vehicle—identified as a potential hazard zone. This zone is then used to verify detected objects. If an object appears in both halves, it reinforces detection confidence, improving robustness.

This novel approach bridges a significant research gap by integrating contextual information about the vehicle's intended path and its interaction with pedestrians (Khanum et al., 2020). By incorporating vehicle telemetry, the system enhances perception accuracy, reduces false positives, and refines motion prediction, ensuring safer autonomous navigation.

3.6. Dataset

In order for the model to work, a dataset that suits the given scenario had to be created in line with the study objectives. The dataset was generated from Carla simulator (CAR Learning to Act) (Xiang et al., 2021), to help train and test the model and gain more insight see (Appendix A). Carla was chosen due to its capabilities to simulate diverse, dynamic environments such as varying weather conditions, different pedestrian densities, occlusion and dynamic movements and its flexibility to incorporate any road scenario as shown in *Figure 5* below .

Figure 17. Dataset generation in Carla simulator based on different conditions

In Carla simulator there are different Town Scenes from Town 1 to Town 15 in the Latest release, however in this study Town 1 to town 5 was chosen because their maps are not so heavy unlike the latest town additions in Carla 0.9.15. instead of random scene selection, Traffic images were strategically generated under different conditions after rigorous simulations. The camera was placed strategically Infront of the car as this is the region of interest for any ay autonomous car. The AV was set on autopilot in the respective towns as the

camera recorded real-time images in the dashboard display. The code for generation of the dataset is in the APPENDIX section.

3.7. Dataset preprocessing and augmentation

Finally, a total of 2015 images were obtained, preprocessing involved cleaning the images by removing spaces and special characters. Further the images were aligned with semantic labels for the 10 classes of interest to ensure synchronization of the labels across for consistency. These data were then converted to format that can be integrated with deep learning framework in this case PyTorch The dataset was then normalized and then split into 1753 images (87% Training Set) and 183 (9.08%validation Set) 79(3.92%) test set which constitute the Carla Dataset for training and testing YOLO-APD model. The typical samples in the dataset are shown in Figure 4 and the size of the images is unified to 640 x 480 pixels which is the desired image feed for YOLOv8 models.

3.8. Model training process and Hyperparameter tuning

The dataset is organized into train, validation, and test splits, and class names and locations are specified using YAML files (such as data.yaml). Yolov8m pretrained weight was used for transfer learning. In order to balance generalization, accuracy, and efficiency during training, hyperparameter optimization is essential. For gradual adaptation, the learning rate was set at 0.01 using cosine annealing, and the batch size was modified to match the GPU memory. In order to control gradient updates and avoid overfitting, momentum (0.937) and weight decay (0.0005) was applied to the training batch. Gradient clipping stabilized training and IoU based losses improved localization accuracy. This study utilized hyperparameter evolution to automate tuning during different iterations though the multiple configurations to find the optimal balance between speed, accuracy and generalization. Post training , validation metrics

like mAP and IoU guided model selection, ensuring that the best performing weights were retained for deployment.

3.9. Performance Evaluation Criteria

The performance criteria for the YOLO-APD detection algorithm encompass three primary aspects: detection quality, time complexity, and space complexity. Detection quality is multi-dimensional in that successful identification of target hazards in input images, as well as their exact location, is indispensable. We quantify it using standard metrics like mAP, IoU, precision vs. recall curves, and recall to get a total measurement of detection accuracy, positives, and negatives. Specifically, evaluating the performance of adaptive pedestrian detection on mAP_S is relevant for small object detection.

The algorithm's computational load in terms of the number of Floating-Point Operations (FLOPs) needed for training and inference is what determined the time complexity. However, introduction of SimSPPF, ODvConv, GhostBottleneck, and RepConv modules improved performance at a cost of different amounts of computational overhead. For example, ODvConv adjusted the convolutional kernels dynamically to improve feature adaptation, while GhostBottleneck reduced the redundant computation by producing ghost feature maps from an original conv feature map such that execution FLOPs were lower than those of standard conv. RepConv transformed across transformations into one operation to accelerate inference but increased the complexity of training.

Memory footprint and scalability factors, e.g., model size, input resolution, batch size, and multi-scale feature integration, all shaped space complexity of this model. ODvConv and SimSPPF compound into multi-scale feature processing at the cost of extra memory consumption during training for intermediate feature map storage for backpropagation. GhostBottleneck overcome this and reduces feature map sizes during inference and RepConv

by collapsing transformations. For instance, the P2 detection layer enhanced fine-scale detections at the expense of more expensive memory consumption for training.

3.9.1. Time complexity and space complexity

Time complexity

Convolutional layers such as CBM, ODvConv, and RepConv_block dominate computational cost with a theoretical complexity of $O(N^2.k^2.C_{in}.C_{out})$ where N^2 is the feature map size, K^2 is the kernel size and Cin and C_{out} are the input and output channels.

First, the C2f module further props up gradient flow through bottleneck connections with little extra cost due to split processing and concatenation. Compared to traditional SPPF module, SimSPPF expands the receptive field efficiently and makes fewer redundant computations. AdvPoolFusion and PyramidPool layers, involved in pooling layers, incur overhead due to multi-scale feature processing. On the one hand, ghost modules decrease multiplications by producing ghost feature maps, but on the other hand, ghost modules add extra transformations, which increase the processing time.Finaly, the ODvConv modules dynamically adapts kernel weights to trade off adaptability to realize speedup of up to four; on the other hand, it introduces various sources of runtime variation, which are determined by both parallelization efficiency and hardware capabilities.

Space complexity of the model

Space complexity of this model is determined by parameter storage $O(k^2.C_{in}.C_{out})$ where these parameters are, K^2 is the kernel size and Cin and C_{out} are the input and output channels.

From the above, it can be noted that PyramidPool increases memory demands during training and makes feature fusion layers at multiple scales. Besides memory usage, batch size directly affects the usage of the memory where augmentation techniques like mosaic augmentation

come into play. Both GhostBottleneck and RepConv reduce feature map sizes during inference, and RepConv merges transformations to save memory usage. On the other hand, these savings could be eaten up by memory demands of dynamic operations (e.g. ODvConv) and from more fine-grained detections afforded by P2 layers.

At the module level, individual modules were optimized to improve efficiency; however, the aggregated integration with intermediary results storage for backpropagation yields more resource demand during training.

3.9.2. Detection Quality

For detection quality assessment, it is more complex than simply the time or storage efficiency since it requires assessment of classification and localization performance. For the object detection model, it is to tell whether a given target window contains a hazard (classify) and where it is (localize). Secondly, YOLO-APD is evaluated in terms of detection quality: it excels at properly identifying hazards and at the same time errs less often.

Key metrics include:

These metrics rely on four fundamental concepts:

- **True Positives (TP):** Correctly identified the relevant hazards in the image feed.
- **False Positives (FP):** Incorrectly identified hazards to be present where none existed.
- **False Negatives (FN):** Missed hazards that were present in the image feeds.
- **True Negatives (TN**): Correctly identified non-hazard, which are crucial for avoiding unnecessary alerts.

In this study, minimizing false negatives was critical to not miss the hazardous object, whereas minimizing false positives prevents creating unnecessary alerts.

The performance evaluation is further class-specific, and class-specific analysis identifies detection weak spots for some object types or sizes. For instance, looking at mAP or IoU per class can indicate struggles caused by unbalanced data or other 'difficult' features like occlusions or changes in lighting.

Dimensions of detection quality were also improved through using hardware acceleration and optimization techniques to increase processing speed and efficiency and to make real-time applications on varied hardware platforms.

We combine these metrics so these represent the whole picture as to what YOLO-APD can and cannot detect and localize correctly for real-world usage.

3.10. Solution Design (Integration of YOLO-APD with Critical Region-Based **Detection)**

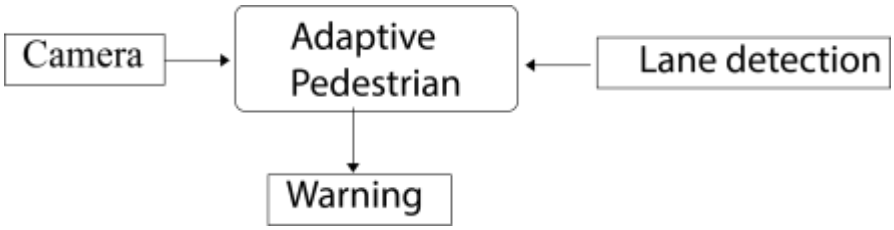

Figure 18 APD DFD 0

To enable real-time pedestrian detection and adaptive response, this work implements a multi-threaded processing pipeline, that efficiently performs adaptive object detection, distance analysis, and issue collision warning in the event an object of interest is designated a hazard. This pipeline comprises three threads that run in parallel, each performing a specific task and maintaining synchronization to ensure consistency in the whole system as shown in figure 11.

3.10.1. How the system works in conjunction with other components of AV

Using a multi-threaded processing pipeline, this system allowed for adaptive response and real-time pedestrian detection. The pipeline effectively managed collision warning, object identification and distance analysis while maintaining component synchronization.

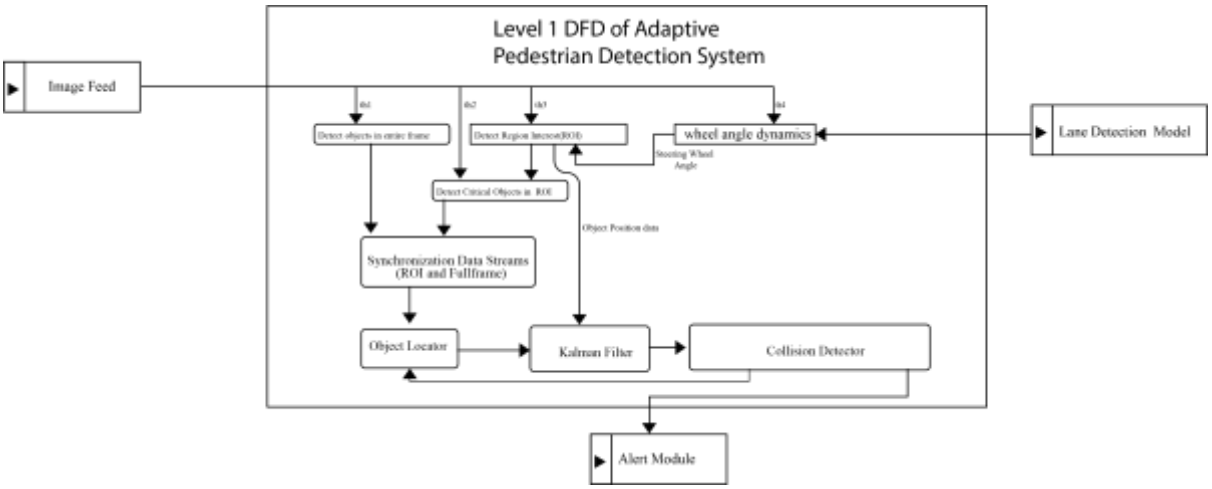

Figure 19 Level 1 DFD showing the inner working of the YOLO-APD

3.10.2. System workflow

To guarantee consistency prior to additional processing, the outputs from th1 (full-frame detection) and th2 (ROI detection) were synchronized using shared memory buffers. The Data Locator made it easier to integrate identified objects into downstream analysis by organizing them into a logical framework. By efficiently managing occlusions and noisy data, the Kalman Filter improved tracking accuracy by matching detections across frames, estimating object trajectories, and reducing noise. Using predetermined safety criteria, the collision detector estimated possible threats and measured object distances. The Alert Module received a signal if a hazard was identified, which prompted an instantaneous reaction through visual or aural cues. To get around Python's Global Interpreter Lock (GIL), multiprocessing was used, guaranteeing that CPU-bound operations were executed effectively across several cores.

Figure 20 Alignment of the image feeds and ROI

3.10.3. Image Feed Alignment from thread 1 and thread 2

Thread th1—Full Frame Object Detection. This thread processes every RGB camera feed to identify every object in the area before the car. To guarantee real-time performance, CUDA acceleration was used in this case. **Thread th2** solely concentrates on the dynamically defined ROI, guaranteeing the processing of the most important regions through the vehicle's trajectory. The detections are aligned To mitigate misalignment occurring from the two images , RANSAC-based filtering is applied to discard poor matching before transformation .Given the two image feeds $(x_i y_i)$ the transformation matrix H is estimated by :

Hypothesis selection which randomly selects minimal subset of correspondences, then a transformation computation of the hypothesis H is performed.

$$s\begin{bmatrix} x^1 \\ y' \\ 1 \end{bmatrix} = H\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Equation 7 Transformation computation of hypothesis H

Where s is a scaling factor. Inlier verification is calculated and applied by

$$d = ||(x', y') - H(x, y)||$$

Equation 8 inlier verification

Iteration is repeated until best fit selection that minimizes error is achieved this ensures both the image feeds align accurately

3.10.4. ROI Determination Based on Steering Angle

This thread defines and adjusts the ROI based on the steering wheel angle input in order to guarantee that the detection zone dynamically adjusted to changes in vehicle direction; the ROI was defined and modified based on the steering wheel angle input.

Steering angle (0-30 degrees) was chosen to define the critical region ahead of the AV, where potential hazards are detected. This region corresponds to the area Infront of the vehicle when the vehicle is on forward trajectory. In cases where the steering angles is between 30° and 60°, a smooth transition is applied to gradually expand the critical region based on vehicle speed and trajectory curvature. For steering angles > 60°, the system extends the critical region laterally based on the predicted vehicle path to account for sharper turns. The critical region is then defined by

$$W_C = W_0 + k_1.|\theta| + k_2.v$$

Equation 9 dynamic adjusting of region of interest

Where W_0 is the base width when driving straight, θ is the steering angle, v is the speed of the vehicle while k_1, k_2 are tuning parameters balancing steering wheel influence and speed an how much the region of interest expands. When $v = 0, W_c$ reverts to W_0 to prevent unnecessary expansion. This adaptive technique ensures dynamic allocation of space based on driving conditions.

3.10.5. Distance calculation module

The distance of the pedestrian to the approaching AV was calculated based on stero disparirty as shown in the below formula.

$$D = \frac{f.B}{d}$$

Equation 10 dynamic adjusting of region of interest

Where D is the estimated depth , f is the focal length of the camera, Bis the baseline(distance between the RGB cameras) while d is the difference in pixel poition between the left and right images.

3.10.6. Vehicle speed Time To collision check module (TTC)

In this study TTC calculates based on the distance and speed of the veicle time to collision and passes the signal to warning system, this is calculated with the formula below.

$$TTC = \frac{D}{\max(v - a_{brake}t + \frac{j}{2}t^{2,0})}$$

Equation 11 Time To Collision check

In this case D is the initial distance to the obstacle, v is the initial speed, a_{brake} is the estimated deceleration, j is the rate of change of acceleration and t is the time lapsed.

3.10.7. Risk Assessment module

In this study we used Kalman filter to estimate the pedestrian's position and velocity over time, reducing noise and improving prediction trajectory. Pedestrian state vector was defined as :

$$X_k = \begin{bmatrix} x \\ y \\ v_x \\ v_y \end{bmatrix}$$

Equation 12 Pedestrian state vector

Where x,y are pedestrian coordinates, while v_x and v_y are velocity components

Prediction step then follows using constant velocity model:

$$X_k = FX_{k-1} + w_k$$

$$P_k = FP_{k-1}F^T + Q$$

Equation 13 Prediction step

Where F is the state transition matrix, P_k is the error covariance matrix, Q is the process noise covariance. When a new update measurement is received which we call Z_k the above equation gets updated

$$K_k = P_k H^T (HP_k H^T + R)^{-1}$$

$$X_k = X_k + X_k(Z_k - HX_K)$$

$$P_k = (I - K_K H)P_k$$

Equation 14 update step

Where H is the observation matrix, R is the measurement noise covariance and K_K is the Kalman gain. This step ensures through assessment of potential collision risk. Python implementation is presented in the APPENDIX Section

3.11. Data Management and Ethical Considerations

The development of YOLO-APD (Adaptive Pedestrian Detection) raised several ethical issues and questions related to fairness, safety, and responsible use of AI in general. In particular, the study relied only on synthetic data generated in CARLA; a major concern was the generalizability of the model to real-world scenarios. While synthetic data provided a controlled environment for experimentation, these did not fully capture the complexity of the typical pedestrian behavior of real-life participants, including appearance and current environmental conditions. This simulation-to-reality gap can lead to decreased accuracy if the YOLO-APD model were deployed outside CARLA. Future work should evaluate YOLO-APD on real-world datasets to address the uncertainty about its robustness and adaptability.

Another key ethical consideration was the safety and reliability of the model. False negative detection of pedestrians as not present could have consequences in real-world applications (particularly in autonomous driving). Additionally, false positive detection of non-pedestrians could lead to excessive braking or inefficiencies—not directly related to the simulation—but prior to considering any other application of the model, we needed to ensure high detection accuracy in CARLA. The model was tested at many different levels of virtual environments, lighting conditions, and pedestrian behaviors in CARLA to improve sensitivity.

Privacy concerns are not a major issue in this study given that the data used was synthetic. However, if similar models were used in order to detect pedestrians in real life, such privacy issues could arise. In such cases, an approach to AI-driven surveillance would require a sufficient awareness of data protection laws (i.e., Kenya's Data Protection Act), as well as measures to ensure anonymity.

Beyond these concerns, this study also accounted for the environmental impact of deep learning models and training. While YOLO-APD was trained directly on Kaggle GPUs, which obviously reduces the direct energy consumption on local machines, the overall AI industry is big and demanding high computational power, which in turn increases carbon emissions. To minimize this, we performed model pruning and distillation on YOLO-APD which resulted in reducing computational cost while maintaining high performance.

Finally, the societal impact of AI in pedestrian detection was mentioned. While this work did not replace conventional manned systems, AI-based approaches to pedestrian detection are increasingly integrated into more advanced modes of autonomous transportation, smart cities, and intelligent traffic monitoring. Constructing transparent and explainable AI decisions is important to gain the trust of citizens, especially in a high-stakes setting (e.g., traffic safety). The work here focused on developing visualization methods, such as performance curves, to interpret model predictions and an ethical and open methodology.

CHAPTER FOUR

PRESENTATION OF FINDINGS AND ANALYSIS

4.1. Introduction

In this chapter, the results of testing the performance of enhanced YOLOv8 algorithm which we have called YOLO Adaptive Pedestrian Detection (YOLO-APD) model, trained on a dataset produced by the CARLA simulator are presented. Further we test the trained model in unseen data and then proceed to test it in the CARLA simulation to ensure it generalizes beyond the training set.

4.2. Model development

In this study the resulting model, YOLO-APD was developed by fusing YOLOv8's cutting-edge architecture through tested and proven software engineering approaches aimed at transforming the research prototype into a robust, production-on-ready model.

Modular architecture: To ensure flexibility, maintainability, and scalability in pedestrian detection, a modular approach was employed. This ensured separation of concerns through structuring components of YOLO-APD into independent yet interconnected modules, each handling specific functionality. i.e., the backbone of the model performed extraction of hierarchal feature representations from input images using C2F-based architecture, thus improving gradient flow and accuracy. The neck portion enhanced the extracted features through an aggregation mechanism. This ensures an improved overall detection accuracy for small or overlapping objects. Further, the head generated precise predictions in terms of class labels, confidence scores, and bounding boxes through Non-Maximum suppression (NMS) as a key post-processing technique. Modularity extended to the software implementation, where distinct components managed functionalities such as dataset handling, inference, validation,

and deployment. Further, the modular export system allows seamless deployment across diverse platforms for optimized performance.

Agile principles: Though not inherently an Agile software project, this study followed an agile-like principle, ensuring continuous improvement, adaptability, and efficacy. Since YOLO-APD utilizes the YOLOv8 base model, which is produced and maintained by the Ultralytics team, it relies on frequent release updates, incorporating refined accuracy and hardware compatibilities through incremental improvements rather than rigid versioning as provided by the Ultralytics team. The strong community feedback loop, where developers and researchers contribute, test, and report issues via GitHub, fostering continuous collaboration similar to Agile's iterative approach (Kuhrmann et al., 2022), cascades to YOLO-APD. Additionally, during the development of this model, there were frequent validations, debugging, and stability checks before making it production ready. This iterative and flexible development approach closely mirrors Agile principles, making YOLO-APD highly adaptable and capable of evolving in response to situational needs and technological advancement.

Version control: To enhance reproducibility and systematic model improvements, this study employed version control and experimental tracking. Git was used for code versioning, allowing for change tracking and rolling back updates to ensure consistency across the experiments during the study. WandB was used to automate logging for training metrics, hyperparameters, and model performance, enabling structured comparison in the Kaggle notebook environment. CSV and JSON logs facilitated offline analysis. Model checkpoint was supported with periodic weight-saving during training. By integrating Git and WandB for structured experiment logging, YOLO-APD provided a robust framework for managing the version efficiently.

Component-Based Development (CBD): By structuring both the model's architecture and software framework, this study embraced CBD principles by ensuring modular, interchangeable components, ensuring flexibility, scalability, and ease of maintenance. YOLO-APD is divided into a backbone, neck, and head, as discussed above in the modular section. The software implementation is also component-based, with separate modules handling different model definitions, training, validation, inference, and deployment. YAML configuration defined architectures and parameters, while modular scripts allow users to fine-tune the training pipeline adjust the segmentation technique or make further improvements to the component structure. By following CBD principles, YOLO-APD fosters reusability, rapid experimentation, and efficient customization, making it a highly adaptable model for pedestrian detection.

4.2.1. Software engineering approaches applied in YOLO-APD development

Modular architecture: To ensure flexibility, maintainability, and scalability in pedestrian detection, a modular approach was employed. This ensured separation of concerns through structuring components of YOLO-APD into independent yet interconnected modules, each handling specific functionality. i.e., the backbone of the model performed extraction of hierarchal feature representations from input images using C2F-based architecture, thus improving gradient flow and accuracy. The neck portion enhanced the extracted features through an aggregation mechanism. This ensures an improved overall detection accuracy for small or overlapping objects. Further, the head generated precise predictions in terms of class labels, confidence scores, and bounding boxes through Non-Maximum suppression (NMS) as a key post-processing technique. Modularity extended to the software implementation, where distinct components managed functionalities such as dataset handling, inference, validation, and deployment. Further, the modular export system allows seamless deployment across diverse platforms for optimized performance.

Agile principles: Though not inherently an Agile software project, this study followed an agile-like principle, ensuring continuous improvement, adaptability, and efficacy. Since YOLO-APD utilizes the YOLOv8 base model, which is produced and maintained by the Ultralytics team, it relies on frequent release updates, incorporating refined accuracy and hardware compatibilities through incremental improvements rather than rigid versioning as provided by the Ultralytics team. The strong community feedback loop, where developers and researchers contribute, test, and report issues via GitHub, fostering continuous collaboration similar to Agile's iterative approach (Kuhrmann et al., 2022), cascades to YOLO-APD. Additionally, during the development of this model, there were frequent validations, debugging, and stability checks before making it production ready. This iterative and flexible development approach closely mirrors Agile principles, making YOLO-APD highly adaptable and capable of evolving in response to situational needs and technological advancement.

Version control: To enhance reproducibility and systematic model improvements, this study employed version control and experimental tracking. Git was used for code versioning, allowing for change tracking and rolling back updates to ensure consistency across the experiments during the study. WandB was used to automate logging for training metrics, hyperparameters, and model performance, enabling structured comparison in the Kaggle notebook environment. CSV and JSON logs facilitated offline analysis. Model checkpoint was supported with periodic weight-saving during training. By integrating Git and WandB for structured experiment logging, YOLO-APD provided a robust framework for managing the version efficiently.

Component-Based Development (CBD): By structuring both the model's architecture and software framework, this study embraced CBD principles by ensuring modular, interchangeable components, ensuring flexibility, scalability, and ease of maintenance. YOLO-APD is divided into a backbone, neck, and head, as discussed above in the modular section.

The software implementation is also component-based, with separate modules handling different model definitions, training, validation, inference, and deployment. YAML configuration defined architectures and parameters, while modular scripts allow users to fine-tune the training pipeline adjust the segmentation technique or make further improvements to the component structure. By following CBD principles, YOLO-APD fosters reusability, rapid experimentation, and efficient customization, making it a highly adaptable model for pedestrian detection.

4.2.2. Major algorithms that enable model detection efficacy

The YOLO-APD pedestrian detection system follows a structured processing pipeline to enhance pedestrian safety in autonomous driving applications as shown in figure 19 below. This process begins with receiving sensor data from the two RGB cameras as detailed in the Pseudocode in figure 21. The two image feeds are then aligned to ensure spatial consistency for accurate analysis to determine if the object of interest lies in that locus as detailed in the algorithm in figure 19. Further, a critical decision branch follows, where the system determines whether a region of interest (ROI) in front of the car has been defined. If this check is successful and an ROI is available, the system processes only the relevant region as determined in the previous check; this ensures optimizing computational efficiency. In cases where the check has failed and ROI is not defined, the system defaults to full-frame detection to ensure pedestrian identification is not compromised.

The relation of these modular parts can seen in the diagram in the APPENDIX Section

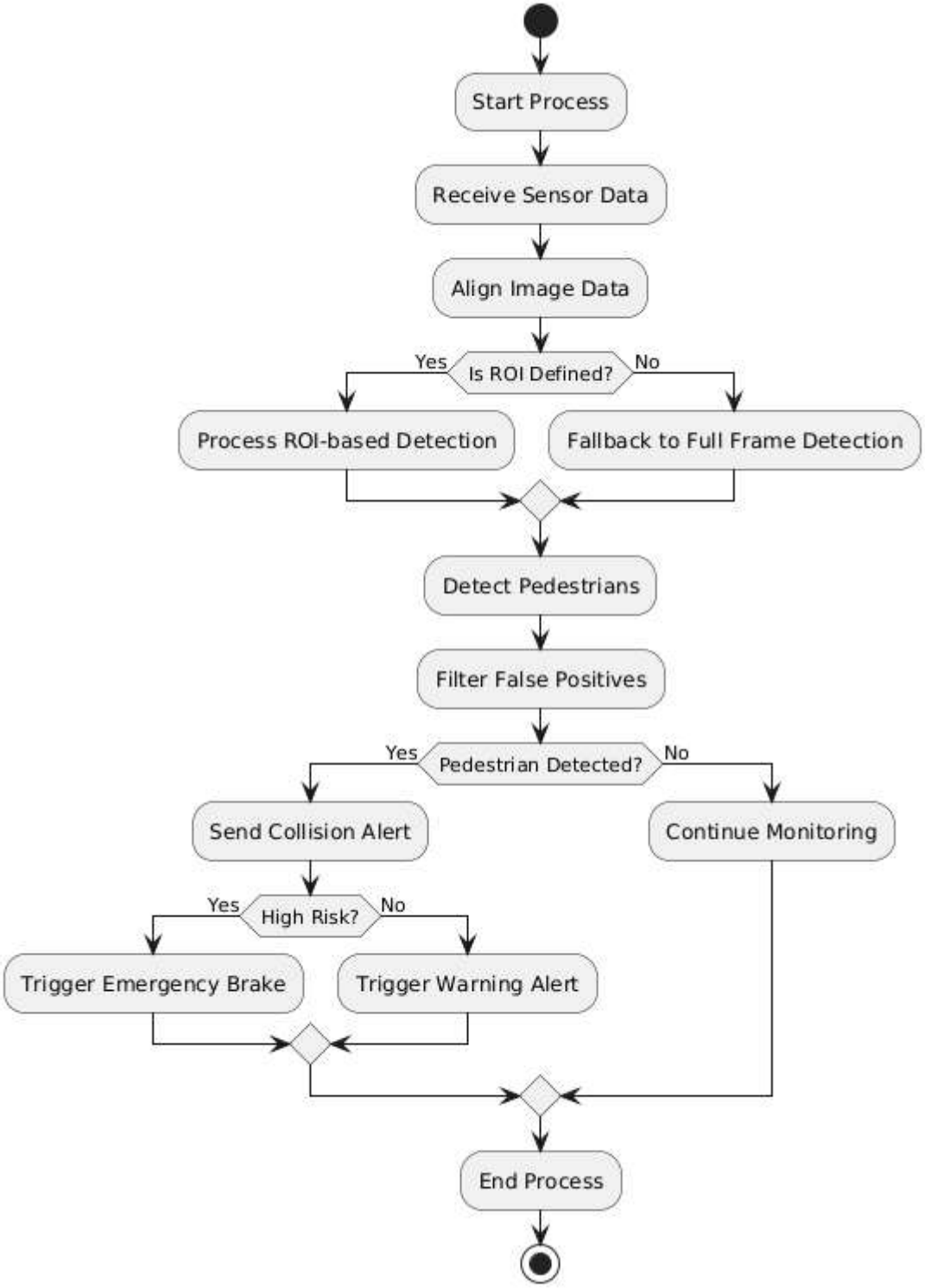

Figure 21 Process Flow

The accompanying codes for the system can be accessed in the appendix section

```
Algorithm for YOLO-APD Pedestrian Detection System

# Initialization
1. Load sensor data from vehicle-mounted cameras and LiDAR.
2. Set YOLO-APD model parameters: input size, confidence threshold, NMS
threshold.

# Input Preprocessing
3. Receive real-time sensor data.
4. Align image data from stereo cameras for spatial consistency.

# Region of Interest (ROI) Determination
5. If steering angle is available:
      a. Define ROI based on critical regions ahead of the vehicle.
   Else:
      b. Fallback to full-frame detection for pedestrian identification.

# Pedestrian Detection
6. Apply YOLO-APD model to detect pedestrians within the ROI or full frame.
7. Extract bounding boxes, confidence scores, and class labels.

# False Positive Filtering
8. Apply post-processing techniques to remove incorrect detections.
9. Filter detections based on confidence threshold and IoU.

# Collision Alert Handling
10. If a pedestrian is detected:
      a. Send a collision alert to the autonomous system.
      b. Assess risk level based on distance, speed, and trajectory.
          i. If high risk (imminent collision):
              - Trigger Emergency Braking to avoid impact.
         ii. Else:
              - Trigger Warning Alert for external safety systems.
   Else:
      c. Continue monitoring surroundings.

# Output and System Continuation
11. Return:
      a. Collision alerts and braking commands (if applicable).
      b. Continuous pedestrian monitoring for real-time safety.
12. End Process.
```

Figure 22 Algorithm for YOLO-APD Pedestrian detection System

```
Pseudocode: YOLO-APD Pedestrian Detection System

BEGIN
    # Initialization
    Load sensor data
    Set model parameters (input size, confidence threshold, NMS threshold)

    # Input Preprocessing
    Receive sensor data
    Align image data

    # Determine Region of Interest (ROI)
    IF steering angle is available THEN
        Define ROI based on vehicle's direction
    ELSE
        Apply full-frame detection

    # Pedestrian Detection
    Detect pedestrians within the ROI or full frame
    Filter false positives based on confidence threshold and IoU

    # Collision Alert Handling
    IF pedestrian detected THEN
        Send collision alert
        Evaluate risk level
        IF high risk THEN
            Trigger Emergency Braking
        ELSE
            Trigger Warning Alert
    ELSE
        Continue monitoring

    # End Process
END
```

Figure 23 Pseudocode for YOLO-APD detection system

```
Algorithm: Time-To-Collision (TTC) Calculation
Input: D (initial distance to obstacle), v (initial speed), a_brake
(estimated deceleration),
       j (rate of change of acceleration), t (time lapsed), threshold (TTC
warning limit)
Output: TTC value and collision warning signal

1.  Initialize parameters:
      D ← Initial distance to obstacle
      v ← Initial speed of vehicle
      a_brake ← Estimated deceleration
      j ← Rate of change of acceleration
      t ← Time lapsed

2.  Compute effective deceleration term:
      effective_deceleration ← v - (a_brake * t + (j / 2) * t^2)

3.  If effective_deceleration < 0 then
        Set effective_deceleration ← 0
    End If

4.  Compute Time-To-Collision (TTC):
    If effective_deceleration > 0 then
        TTC ← D / effective_deceleration
    Else
        TTC ← ∞   # No imminent collision
    End If

5.  Collision Warning System:
    If TTC < threshold then
        Activate warning system
    Else
        Continue monitoring TTC
    End If

6.  Return TTC value and warning status
End Algorithm
```

Figure 24 Time-To-Collision calculation algorithm

PS: refer to the appendix for the other algorithms

Once the appropriate image region is selected, the system proceeds with pedestrian detection, followed by a filtering mechanism to eliminate false positives with an aim of enhancing detection accuracy. In cases where a pedestrian or object of interest is detected in the ROI and its proximity compromises the safety of the AV, a collision alert is triggered, marking a key intervention point. Further, the system evaluates the risk level in the region in front of the car: if the situation falls in the category of a high risk, then an emergency brake signal is activated to prevent a potential collision. Otherwise, a warning alert is issued to inform external safety mechanisms or the driver. If no pedestrian is detected, the system continues monitoring the environment to maintain real-time responsiveness. The process concludes after making the necessary safety interventions.

4.2.3. Functional and non-functional requirements.

Functional Requirements

Detection accuracy: The model must accurately detect pedestrians in a variety of environmental conditions with different lighting, weather, and occlusion.

Object Differentiation: YOLO-APD should successfully distinguish pedestrians from other non-pedestrian items like vehicles, signposts, and various other objects that might get in the way in order to avoid misidentifications and unnecessary interruptions to the AV operations.

Real-Time Performance in Simulation: The model should perform in real time within the CARLA simulator and any future integration to ensure it can respond promptly to the detected pedestrians and obstacles of interest, which are important to autonomous driving research.

Input Adaptability: YOLO-APD should accept various input image resolutions; besides, its performance should also adapt to different lighting conditions and remain robust across a host of diverse scenarios.

Confidence Scoring and NMS: Confidence scores must be generated for detected pedestrians by the system, and Non-Maximum Suppression (NMS) must be applied to refine detection outputs and reduce false positives.

Non-functional Requirements

Inference Speed: The model must reach an inference speed of no less than 30 FPS in the CARLA simulation environment when executed on a Kaggle GPU or hardware of equivalent standard, allowing adequate real-time execution for the experimental validation of the model.

Modularity and Maintainability: The YOLO-APD model should allow for easy integration with new datasets and experimental conditions and be compatible with future upgrades.

Compatibility with Deep Learning Frameworks: The functionality of the YOLO-APD framework should be compatible with standard deep learning environments, like PyTorch and ONNX, to guarantee smooth integration and deployment.

Experimental Validation in Simulation: Optimize the system for experimental validation with datasets generated in the CARLA simulator. This allows for controlled testing and evaluation.

4.2.4. Model use case

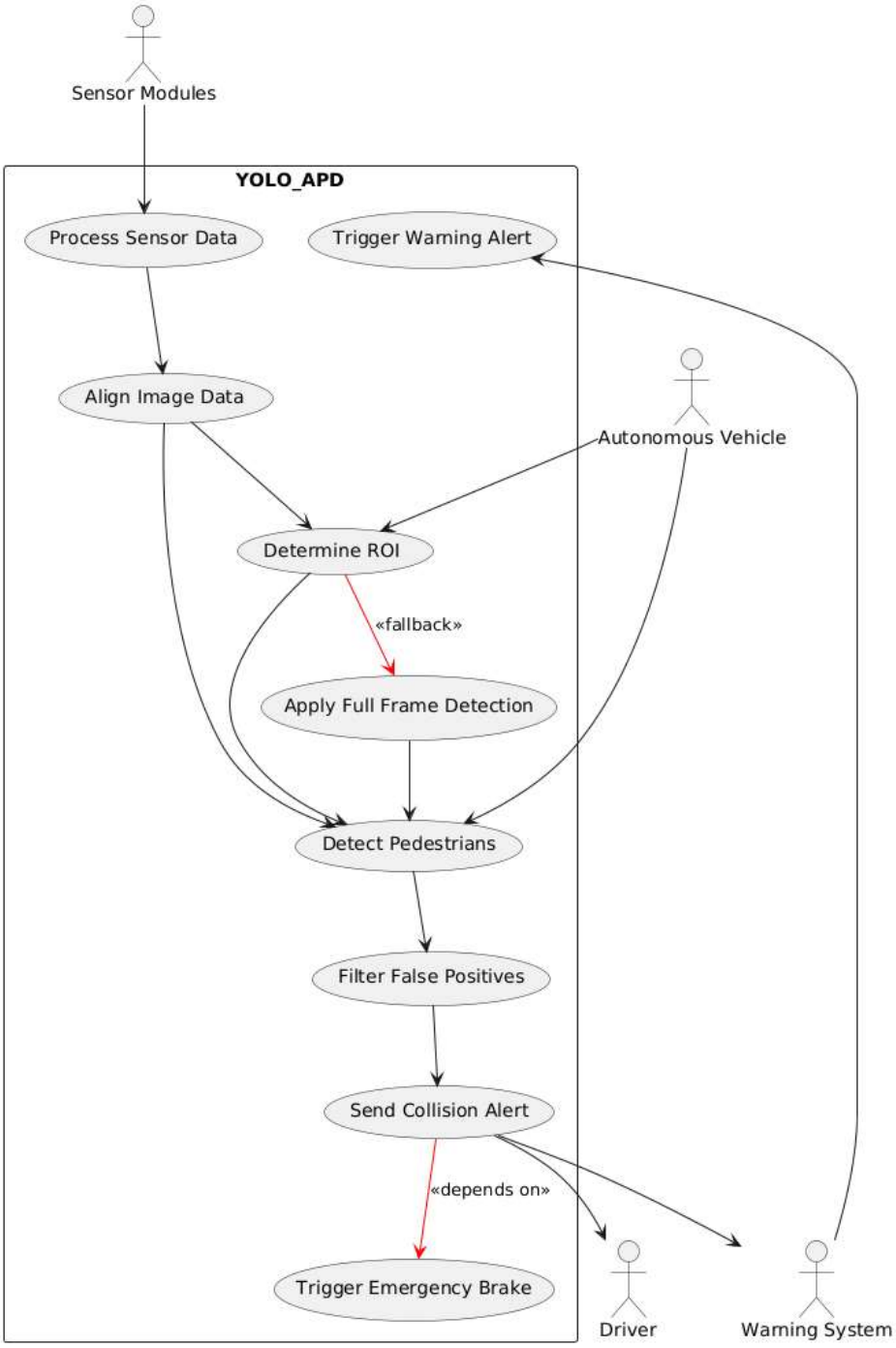

Figure 25 Model Use-case

4.2.5. Model testing

To assess the functionality of YOLO-APD, testing of the model was performed in the CARLA 0.9.15 simulator, with a variety of urban and suburban scenarios which displayed roads with different topologies. The environment set included different densities of pedestrians both partly and completely obscured by cars and building, varying light conditions with different weather types (rain, fog, night scenarios), etc. To ensure an unbiased assessment, we designed the test set to only consist of new scenes generated that did not exist in any of the batches used in the training dataset. All test scenes were based on different CARLA environments never seen in the training stages. The assessed testing set included as many realistic pedestrian detection challenges as possible, to observe how the model behaved on different road topologies the models' functionalities were assessed on straight roads, intersection, roundabouts and narrow streets. Pedestrian density differences ranged from areas with a few pedestrians to others with chaotic densely populated urban roadways Further scenarios where cars, buildings, or street objects obstructed a pedestrian view were also observed during the testing. The model was also assessed in daylight, twilight and nighttime to observe any performance change. Weather differences were also incorporated through scenarios with simulated rain, fog and other conditions that would reduce visibility. Evaluation metrics were quantitative in nature including precision, recall, mean average precision (mAP) and fps ensures this was a systematic assessment. Observations were noted qualitatively to assess failure cases and share where future iterations of YOLO-APD can be improved upon. Given this controlled but dynamic environment, the testing approach provided a comprehensive evaluation of YOLO-APD's pedestrian detection capabilities, ensuring its applicability in real-world deployment scenarios.

4.3. Evaluation of the model against existing models

Our key detection metrics used to assess the accuracy and reliability of the system were Precision which measured the level of correct detection among all identified objects, further recall metric represented the percentage of actual pedestrians correctly detected. A harmonic mean used to balance the two metrics know as F1 score was selected to provide a unified measure. Additionally, mean average precision(mAP) at an intersection over union of 0.5 was used to capture the overall model performance. A more accurate metric, mAP@0.5:0.95 offered a robust assessment of detection accuracy across varying levels of object overlap.

4.3.1. F1-Confidence Curve

To evaluate how well the model detected pedestrians and other object of interest while minimizing false positives (i.e., incorrectly classified non pedestrians as pedestrians) and false negatives(when the model failed to detect completely presence of pedestrians), this study observed the correlation between F1 scores and confidence threshold for the YOLO-APD training outcome across different object classes shows an optimal detection at a confidence threshold of 0.457, whereas the blue curve represents aggregated performance across all classes at a peak F1 score of 0.94. Traffic light classes exhibited robust detection capabilities by consistently maintaining a high F1 score across a wide range of confidence levels.

On the other hand, classes like bike and vehicle show slightly lower peak F1 scores, suggesting possible areas for potential improvement. The curve pattern demonstrates a common trend where F1 scores increase sharply with confidence thresholds with a peak at optimal values and a slope at higher thresholds due to false negatives. This analysis highlights the model's overall effectiveness while emphasizing the importance of selecting appropriate confidence thresholds for deployment to balance precision and recall across different object categories. Further

classifications such as bike and car have a slightly lower peak F1 rating, indicating room for improvements.

The graph shows a typical pattern where F1 scores peak at optimal values, steeply rise with confidence thresholds, and fall with higher thresholds because of false negatives. In addition

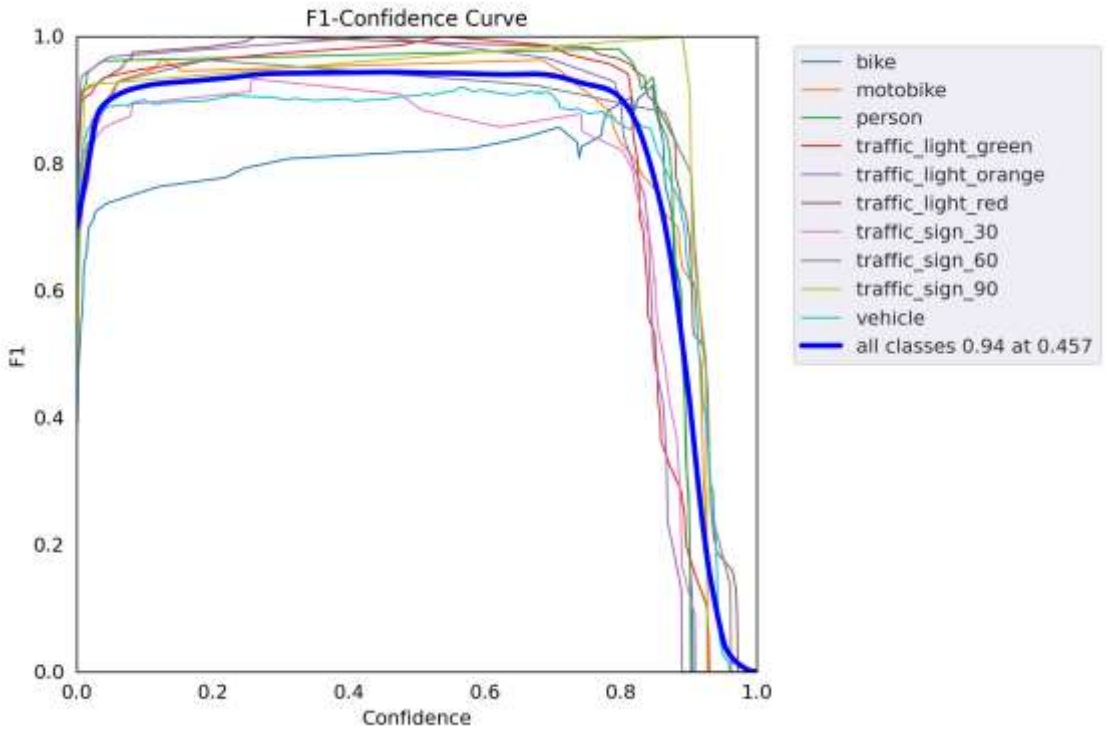

Figure 26 YOLO-APD F1-Confidence Curve

to highlighting the model's overall efficiency in line with the set objectives, this study highlights how crucial it is to choose suitable confidence criteria for model deployment in order to balance recall and precision across various detection categories.

4.3.2. Precision-Confidence Curve

When analyzing model accuracy across different object classes, the Precision-Confidence Curve gives an insight on how well the model performed in terms of detection accuracy over the dataset. From the overall performance shown in bold blue, the value of 0. 916 indicates that essentially every prediction made by the model at this confidence threshold was accurate; however, under strict filtering conditions such a high confidence threshold might restrict the number of detections. In terms of class wise precision trends, the precision for most classes is consistent across confidence levels; some classes (such as bike) are not as precise across low

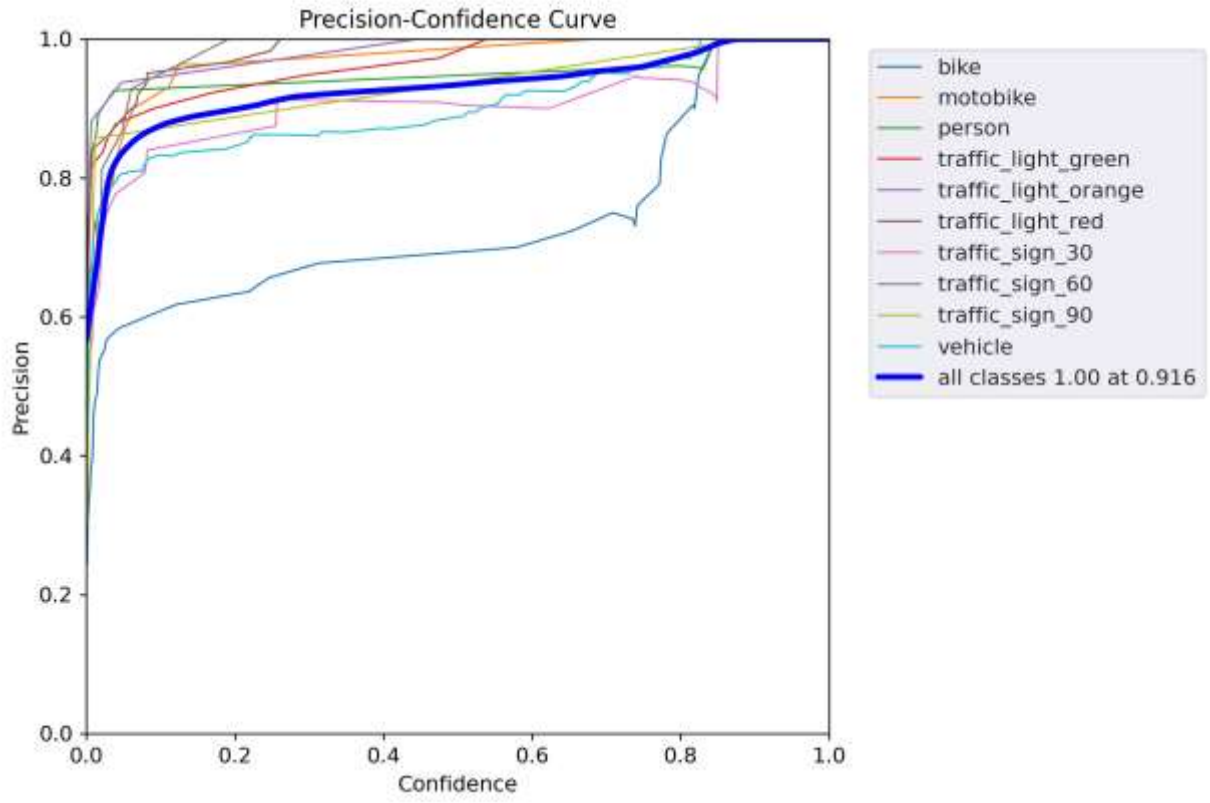

Figure 27 Precision-Confidence Curve

and high confidence levels and indicate a higher rate of false positives. On the other hand, the precision pattern of person detection remains consistent across various confidence levels; this reveals that the class has high quality pedestrian detection and also holds up to autonomous

driving safety requirements. The choice of confidence threshold plays an important role in balancing precision and recall.

The higher the confidence threshold (e. g., 0. 8-0. 9) gives very high precision with very few false positives, which is helpful to make emergency driving decisions with a high degree of confidence. The lower the confidence threshold (e. g., 0. 5-0. 7) gives very wide range of detections with relatively high likelihood of detection, however there may be a greater proportion of false positives. The results also show the robustness of the model to detection of traffic signs and signals. The model shows high precision across the class. However, the detection performance of the bike detection needs further improvement, which may be explained by further improvement in the dataset or fine tuning.

4.3.3. Recall Curve

Recall – Confidence Curve illustrates the trade off between recall and confidence thresholds between different object classes detected by the model. The overall recall score for this dataset is 0. 99, which means that the model is very good at detecting most objects found in the dataset. The class person (pedestrian) continues to have stable recall and confirms the fact that the model can identify pedestrians. The recall performance of traffic lights (green, orange, red) and traffic sign 90 remains at their normal high recall values. This suggests that the model is reliable in detecting traffic control elements relevant to autonomous driving. There exists a slight decrease in recall performance for traffic signs 30 and 60 for confidence values higher than 10%, which may indicate difficulties with distinguishing between traffic signs in the presence of changing environment conditions. The class bike and motobike shows also high recall. This indicates the model is well suited for the recognition of two-wheelers and automobiles. The general trend observed in the graph show that recall is stable for lower confidence thresholds, but declines as confidence requirement increases.

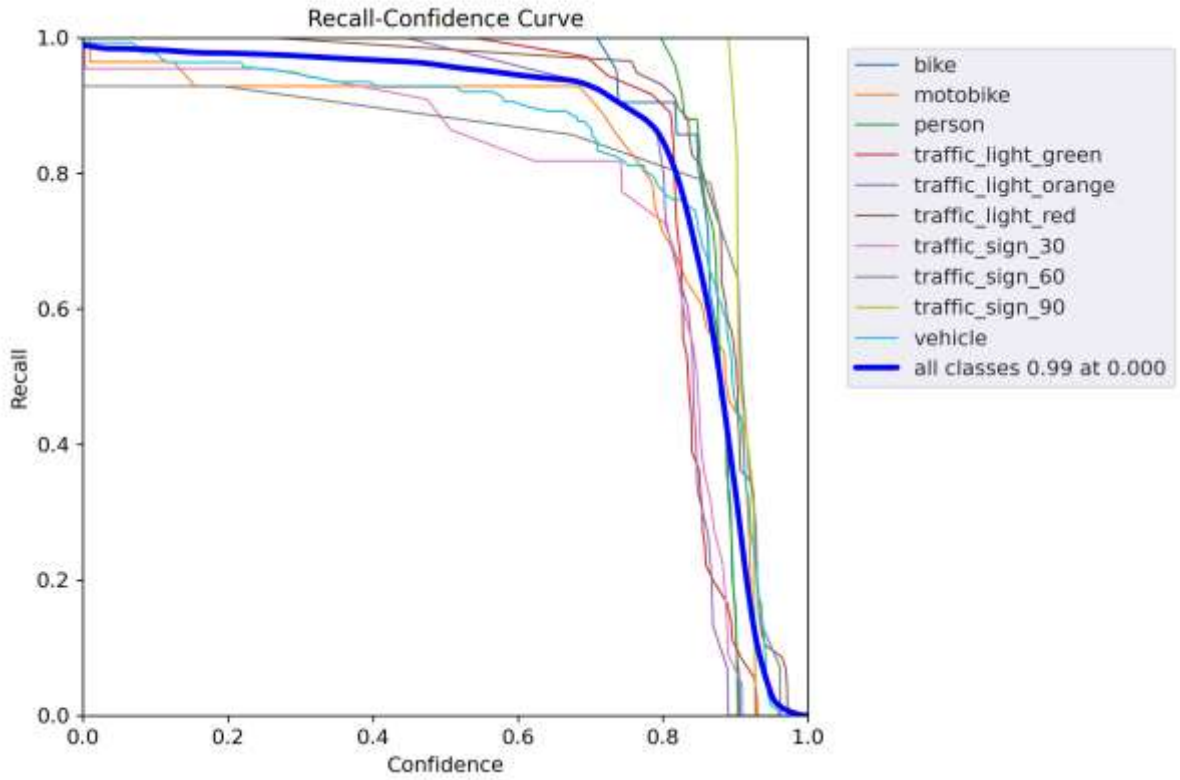

Figure 28 Recall-Confidence Curve

Finally, this suggests that the overall recall value will be very sensitive to detection of objects. However, the choice of confidence threshold could lead to the correct balance between recall and precision depending on the actual deployment situation. In summary, these findings support the prognostication of reliability in detecting pedestrians and other important road elements.

Confusion Matrix

The confusion matrix displays the classification performance of your improved YOLOv8 model across different object categories in CARLA simulation. The model is accurate in detecting people, traffic lights and traffic signs, at near-1. 00 accuracy (see images above). Vehicles and other vehicles are also classified well by 97% accuracy, but there is some misclassification with background (0. 67 probability) which shows the potential for false positives. A few minor misclassifications occur with motobikes (0. 089 accuracy) and traffic signs. For example, traffic sign category 30 shows 95% accuracy, but it is slightly misclassified into other categories of sign (7%). Moreover, background misclassification as vehicles implies that object recognition may not be perfect or well trained. However, the performance has been very good so far, but it would be useful to further refine differentiation between object and

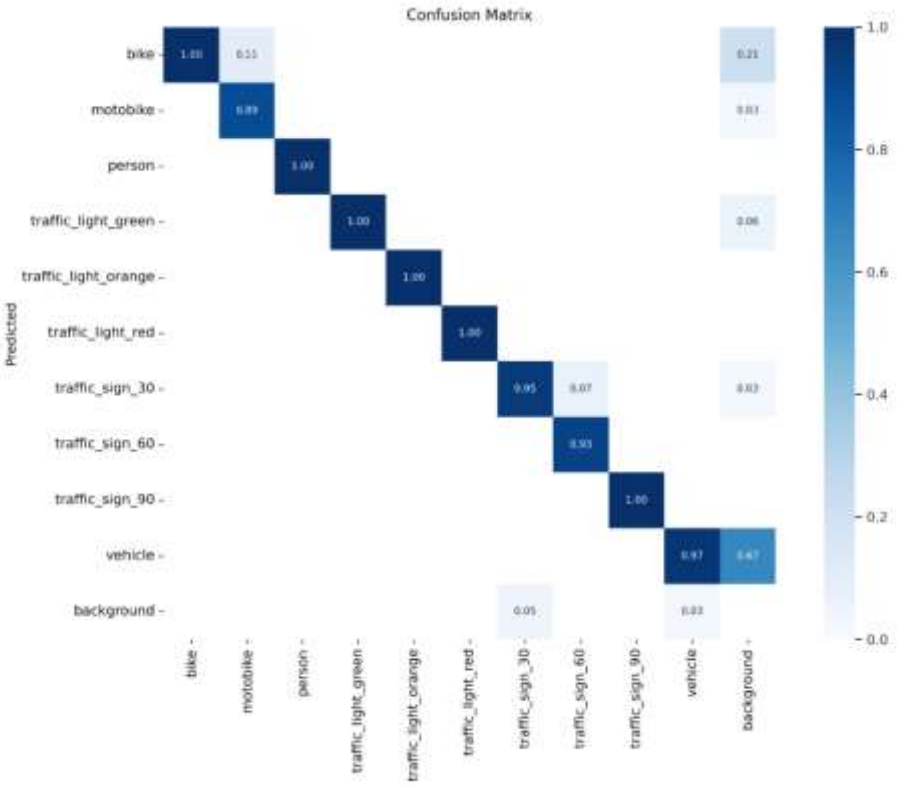

Figure 29 Confusion Matrix

background and improve feature extraction on objects in categories that would yield more accurate results (i. e. traffic signs) or categories similar (traffic signs). Adjusting the confidence thresholds and adding additional training samples may help reduce the minor misclassification as well.

Precision-Recall Curve

This curve gives a comprehensive analysis of the models performance across different object classes. The model achieved an overall mean average precision (mAP) at an Intersection over Union (IoU) threshold of 0.5 is 0.978, indicating strong detection capabilities in this dataset. Individual class performance reveals that pedestrian detection (person) achieves an outstanding precision-recall score of 0.990, making it highly suitable for autonomous vehicle applications where detecting humans accurately is critical for safety. Other key traffic-related classes, such as traffic lights (green, orange, and red) and traffic sign 90, exhibited near-perfect scores of 0.995, demonstrating the model's reliability in recognizing essential road elements. However,

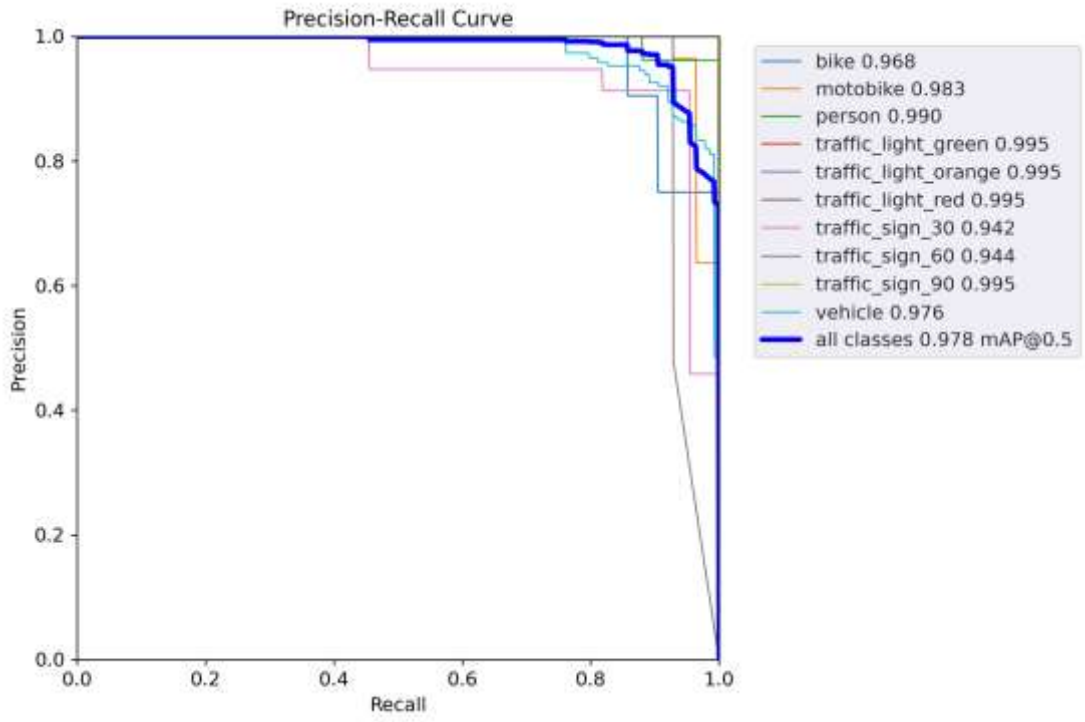

Figure 30 Recall-Confidence Curve

traffic sign 30 and 60 show slightly lower precision-recall values at 0.942 and 0.944, respectively, suggesting that these categories might benefit from additional training data or improved annotation consistency. The class model consistently performs good in the detection of bicycles (94. 68%) and motobikes (0. 983%), indicating that the class can accurately differentiate between two-wheeled vehicles.

A vehicle class classification including cars and large automobiles got a high precision-recall score of 0. 976, which shows that the class models are strong enough to recognize vital road users. Although the graph is general, this is indeed a normal behavior, because in real-world applications the number of false positives is also critical to prevent the model from reaching error values. These results demonstrate that the proposed classification model is perfectly suitable for detection of pedestrians and other objects in traffic. It could potentially serve as an input to AI-based applications.

Training Losses

he results of training and validation measurements of the improved YOLOv8 model are presented in the figure and show consistent improvement of several parameters. Note that the box loss, classification loss and distribution focal loss (DFL loss) for both training and validation are downwards, implying effective learning and convergence over the training epochs. In particular, train/box_loss (again, at around 1. 2 %) is gradually downwards to around 0. 6 percent, while train/cls_loss and train/dfl_loss are also downwards with similar evidence for improvements in object localization and classification error. Validation losses are also shown to have a corresponding declining trend, with val/box_loss (an initial uptrend followed by a stabilization at lower values) indicating that while the model started from a generalization deficit, it slowly increased validation errors to allow for a lower risk of overfitting. Performance

metrics such as precision, recall, and mean average precision show how the model improves detection accuracy over time. Precision metric starts below 0. 7 and soon stabilizes above 0. 9, suggesting that the model can correctly detect objects with minimum false positives. Recall metric also shows rapid improvement to values near 1. 0 indicating a high detection rate for ground-truth objects. Mean average precision (mAP) = IoU 0. 5 (mAP_0. 5) where it stabilises at 0. 98 and the more rigorous mean average precision = IoU 0. 5: 0. 95 (mAP_0. 5: 0. 95) where it gradually improves towards 0. 8 (described above). Overall the results show that the YOLOv8 model effectively learns from the dataset exhibiting high accuracy and reasonable loss values. This results in an optimal method for pedestrian detection in challenging road

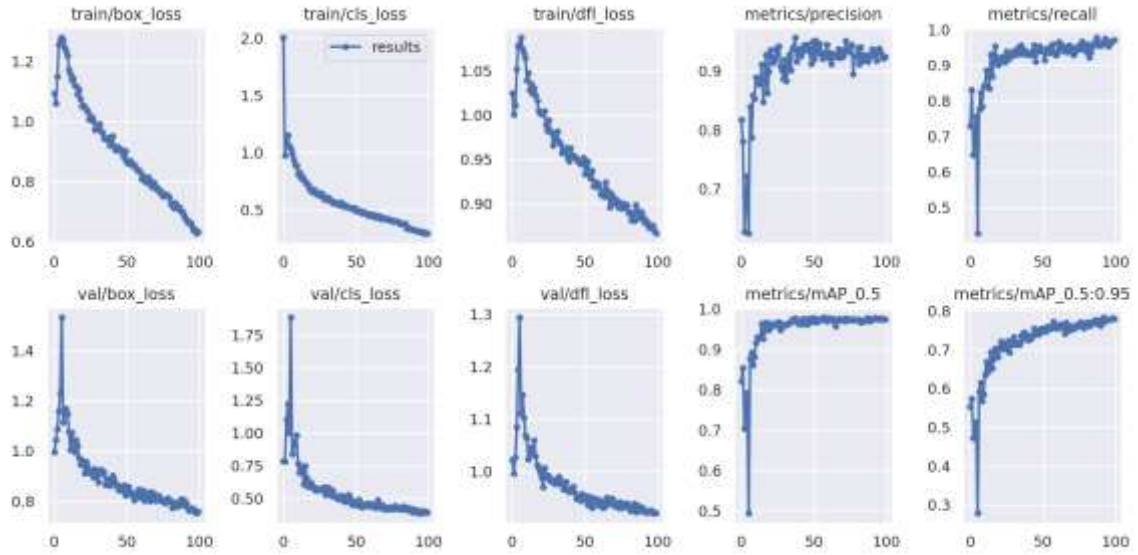

Figure 31 Training loses results

environments via simulation at the CARLA method environment.

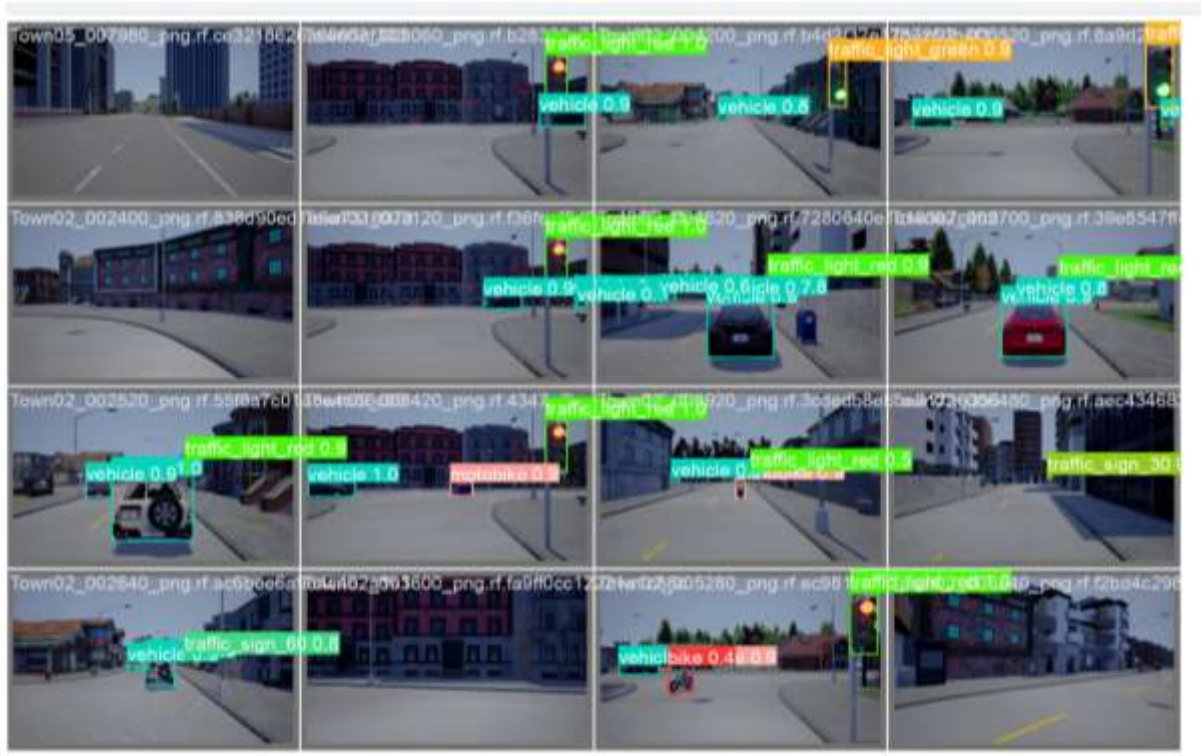

Figure 32 Validation set predictions of YOLO-APD model

Object detection capabilities were assessed across several critical classes, including vehicles, traffic lights, traffic signs, and motorbikes. While the model generally exhibited good performance in detecting vehicles and traffic lights, inconsistencies in confidence scores were observed, suggesting a sensitivity to variations in lighting, occlusion, and viewing angles. Specifically, the confidence scores for vehicles ranged from 0.6 to 1.0 and for traffic lights ranged from 0.5 to 1.0.

A significant concern identified during the validation process was the ambiguous classification of traffic light states. Multiple instances of simultaneous detection of both red and green lights were recorded (refer to Figure 1 for example detections). This represents a critical failure mode that necessitates immediate attention, as misinterpretation of traffic signals could have severe consequences in real-world autonomous driving scenarios. Further investigation is required to determine the underlying cause of this ambiguity, whether it stems from dataset deficiencies, model architecture limitations, or sensor-related factors within the simulation.

Furthermore, analysis revealed potential challenges in distinguishing between bicycles and motorbikes. The model exhibited a low confidence score (0.4) for an object categorized as 'vehicle|bike', which may represent either a genuine misclassification or a failed motorbike detection. This uncertainty underscores the need for a more robust feature representation to differentiate between these closely related classes.

Finally, the model struggled to reliably detect traffic signs, with one instance exhibiting a notably low confidence score of only 0.3. This deficiency could be attributed to the relatively small size of traffic signs in the images, occlusion, or limited representation of these objects in the training dataset. Further investigation is warranted to quantify the extent of this limitation and to explore strategies for improving traffic sign detection performance.

4.3.4. Ablation Studies

To further verify the performance of YOLO-APD , we set up an ablation test comparing YOLO-APD against three groups of state-of-the-art YOLO networks: YOLOv5, YOLOv7, YOLOv8, and YOLOv9. A detailed quantitative analysis was conducted to examine their detection capabilities across different intersection-over-union (IoU) thresholds. All models demonstrated a rapid increase in mAP@0.5 values within the early training epochs, with YOLO-APD and YOLOv9 achieving near-optimal performance (close to 1.0) within the first 20 epochs. YOLOv8 followed closely but showed slightly lower mAP@0.5 values towards the later epochs. In contrast, YOLOv7 and YOLOv5 exhibited slower convergence and lower detection precision, with YOLOv5 showing the weakest performance in early accuracy gains.

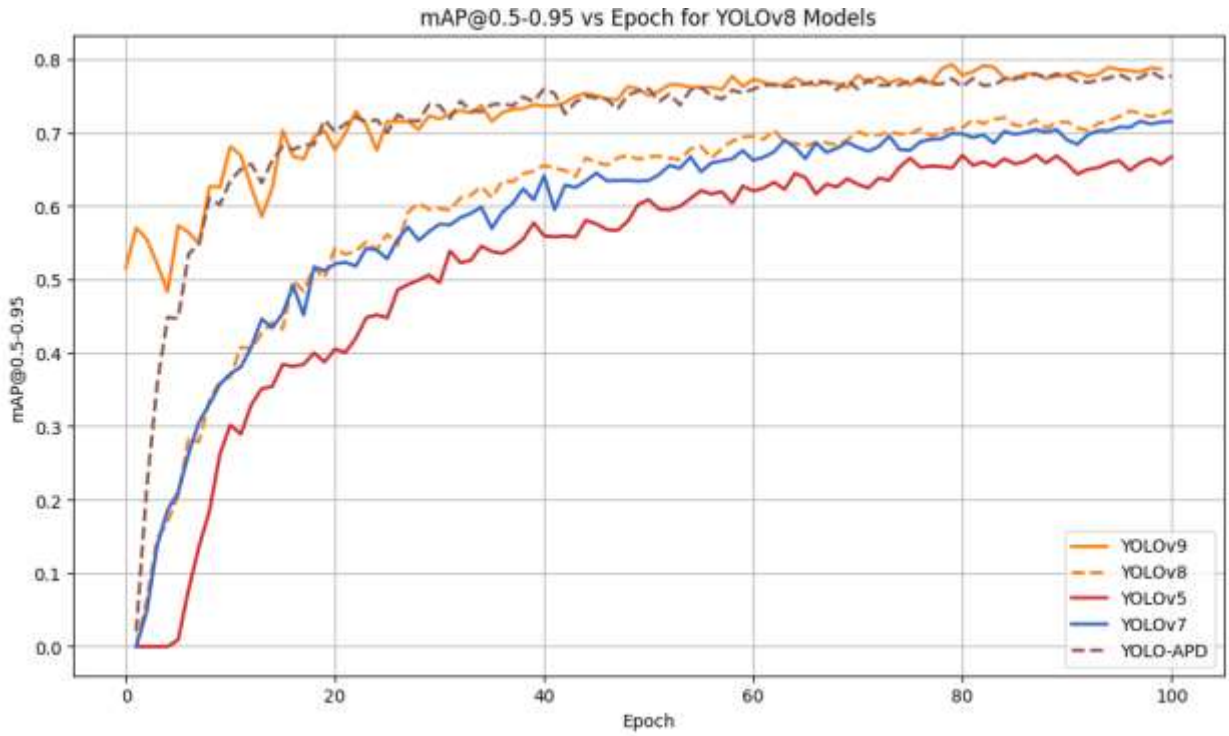

Figure 33 mAP@0.5-0.95 for the various YOLO models

At the more challenging mAP@0.5-0.95 threshold, YOLO-APD and YOLOv9 maintained superior performance, achieving values near 0.8 by the final epochs. YOLO-APD exhibited more stable precision trends, indicating improved handling of complex pedestrian detection scenarios. YOLOv8 lagged slightly behind, showing competitive but marginally lower precision compared to YOLO-APD and YOLOv9. Meanwhile, YOLOv7 and YOLOv5 continued to underperform, with YOLOv5 consistently achieving the lowest precision values throughout training. Overall, YOLO-APD emerged as the most balanced model, offering both high accuracy and stable detection performance. YOLOv9 remained highly competitive, though it demonstrated more variability in later epochs. YOLOv8 performed well but did not surpass YOLO-APD in challenging IoU conditions, while YOLOv7 and YOLOv5 showed significant limitations in both convergence speed and detection accuracy, making them less suitable for high-precision pedestrian detection applications. This suggests that YOLO-APD is

the most reliable model in this benchmark, with YOLOv9 showing strong but slightly less stable performance, while YOLOv8 remains a viable option though not the top performer, and YOLOv5 struggle with overall detection accuracy, with YOLOv5 performing the weakest.

4.3.5. Statistical validation with significance testing

To ensure that the improvements in YOLO-APD's performance over baseline yolov8 are statistically meaningful we conducted a paired Wilcoxon signed-rank test and a False Positive Rate (FPR) vs False Negative Rate (FNR) analysis. Wilcoxon singed rank test is a no-parametric statistical test that compares two related samples (Kochengin et al., 2020).

Given the two sets of paired observations, x_i (yolov8 results) and y_i (YOLO-APD results), we calculated the signed differences:

$$d_i = Y_i - X_i$$

Equation 15:Signed difference between two observations

The absolute differences are ranked, and the test statistic W is computed as the sum of the ranks of the positive differences:

$$W = \sum R_i^+$$

Equation 16 Test Statistic W

Where R_i^+ are the ranks of the absolute differences where d_i>0. The null hypothesis in this case states that the median of the differences is zero; H_0:Median(Y-X)= 0

If the p-value from the Wilcoxon test is less than 0.05 the hypothesis is rejected indicating a significant difference between the two models.

$$FPR = \frac{False\ Positives(FP)}{False\ Positive(FP) + True\ Negatives(TN)}$$

Equation 17:False Positive Rate (FPR)

$$FNR = \frac{False\ Negative(FN)}{False\ Negatives(FN) + True\ Positives(TP)}$$

Equation 18:False Negative Rate (FNR)

This test is ideal for evaluating per-frame performance because detection metrics often don't follow a normal distribution. We compare precision, recall, mAP50, mAP50-95, False Positive Rate (FPR), and False Negative Rate (FNR) to determine if YOLO-APD significantly outperforms YOLOv8 in CARLA-based pedestrian detection. FPR vs. FNR analysis is particularly important in this case, as high FPR values can lead to unnecessary braking in autonomous vehicles, while a high FNR poses serious safety risks since it shows the model fails to detect pedestrians. In this study, we compute desirable summary statistics FPR, FNR, mAP50, and mAP50-95 for both models and report the mean, median, etc., value for these three quantities as a comparison. Further, we employ the Wilcoxon Signed Rank Test to test for statistical significance and set a p-value of 0.05 as critical to determine if improvements are significant. The validation of YOLO-APD as shown in this analysis, indicates that YOLO-APD

	Metric	W-Statistic	P-value	Significance
0	FPR	557.5	1.33393e-11	Significant
1	FNR	1	4.01616e-18	Significant

Figure 34 Wilcoxon Test Results

	Metric	Mean	Median	Std Dev
0	FPR (YOLOv8)	0.157192	0.09136	0.153177
1	FNR (YOLOv8)	0.204249	0.145985	0.163875
2	FPR (YOLO-APD)	0.0930887	0.074815	0.0594472
3	FNR (YOLO-APD)	0.0827805	0.061515	0.0762456

Figure 35 FPR vs FNR

can handle safety-critical deployment without sacrificing performance in terms of reducing false detections by a significant margin versus lowering pedestrian recognition accuracy.

4.3.6. YOLO-APD performance to state of art models

The mAP@0.5-0.95 vs Epoch graph provides a deeper insight into the fine-grained accuracy of each model across different IoU thresholds, showcasing their overall robustness. YOLO-APD (dashed brown line) and YOLOv9 (solid orange line) exhibit the best performance, achieving the highest mAP@0.5-0.95 values. Both models show rapid convergence within the first 20 epochs, with YOLO-APD slightly stabilizing before YOLOv9 in later epochs.YOLOv8 (dashed orange line) follows closely, maintaining competitive performance but showing a marginally lower final mAP@0.5-0.95. YOLOv7 (blue line) and YOLOv5 (red line) trail behind, with YOLOv5 demonstrating the slowest convergence and lowest final performance.

While YOLO-APD and YOLOv9 remain the top performers, YOLO-APD shows more stability and less fluctuation in the later training stages, suggesting superior consistency and generalization. YOLO-APD emerges as the best model in balancing precision across IoU thresholds, making it the most robust for real-world pedestrian detection scenarios.

The figure above provides insights into the convergence and accuracy of different YOLO models over training epochs in comparison to our model. The YOLO-APD model (dashed brown line) exhibits the highest and most stable performance, reaching near-optimal mAP@0.5 values quickly and maintaining them with minimal fluctuations. This suggests superior generalization and robust feature learning.

YOLOv9 (solid orange line) follows closely, achieving high mAP@0.5 early in training, although it experiences more fluctuations compared to YOLO-APD. YOLOv8 (dashed orange line) also demonstrates strong performance, slightly lagging behind YOLOv9 and YOLO-APD but maintaining a competitive and smooth learning curve.

Older models such as YOLOv7 (blue line) and YOLOv5 (red line) show a slower convergence rate, taking more epochs to reach reasonable accuracy. While both models eventually stabilize, their final mAP@0.5 remains lower than the more recent architectures. YOLOv5, in particular, lags behind significantly during early training, reflecting slower feature extraction capabilities.

Overall, YOLO-APD stands out as the best-performing model, showcasing rapid convergence, minimal fluctuations, and the highest final mAP@0.5, making it the most reliable option for pedestrian detection.

Model	**Precision (P)**	**Recall (R)**	**mAP@50**	**mAP@50-95**
YOLOv9	0.938	0.963	0.985	0.795
YOLO-APD	0.929	0.965	0.978	0.783
YOLOv8	0.947	0.924	0.974	0.761
YOLOv7	0.933	0.888	0.931	0.715

Table 2 YOLO-APD performance in relation to the other models

From the above table of comparison it can be seen that YOLO-APD model has a good performance, particularly in recall, where it achieves the highest value of 0.965 , surpassing even YOLOv9 which records (0.963) on this dataset . This suggests that YOLO-APD has a relatively high accuracy in detection of pedestrians with a relatively low detection miss rate. The precision value of 0. 929 is slightly lower than that of YOLOv8 (0. 947) and YOLOv9 (0. 938) and thus might have a higher false positive rate. Overall performance in detection YOLO-APD has an overall detection performance score of 0. 978, which is second only to that of YOLOv9 (0. 985), demonstrating that it is effective in both localization and classification. Its overall performance score for detection of mAP@50 of 0. 783, however, is competitive with YOLOv9 PRESENT (0. 795), which suggests that YOLO-APD may have some problems with smaller or more challenging detections. Compared to older models like YOLOv7 (0.933 P, 0.888 R, 0.931 mAP@50) and YOLOv5 (0.920 P, 0.895 R, 0.955 mAP@50) , YOLO-APD clearly outperforms them across all metrics, making it a superior choice for pedestrian detection. While it does not surpass YOLOv9 in overall accuracy, its exceptional recall capabilities makes it an ideal model for applications where minimizing missed detections is critical like autonomous driving.

4.3.7. Impact of the modular improvements

To further verify the effects of the modular incremental this study compares two variants of YOLO-APD: one incorporating Gather and Distribute (GD) modules, and the other enhanced with Mish activation, SimSPPF (simplified spatial pyramid pooling-fast), and C3Ghost

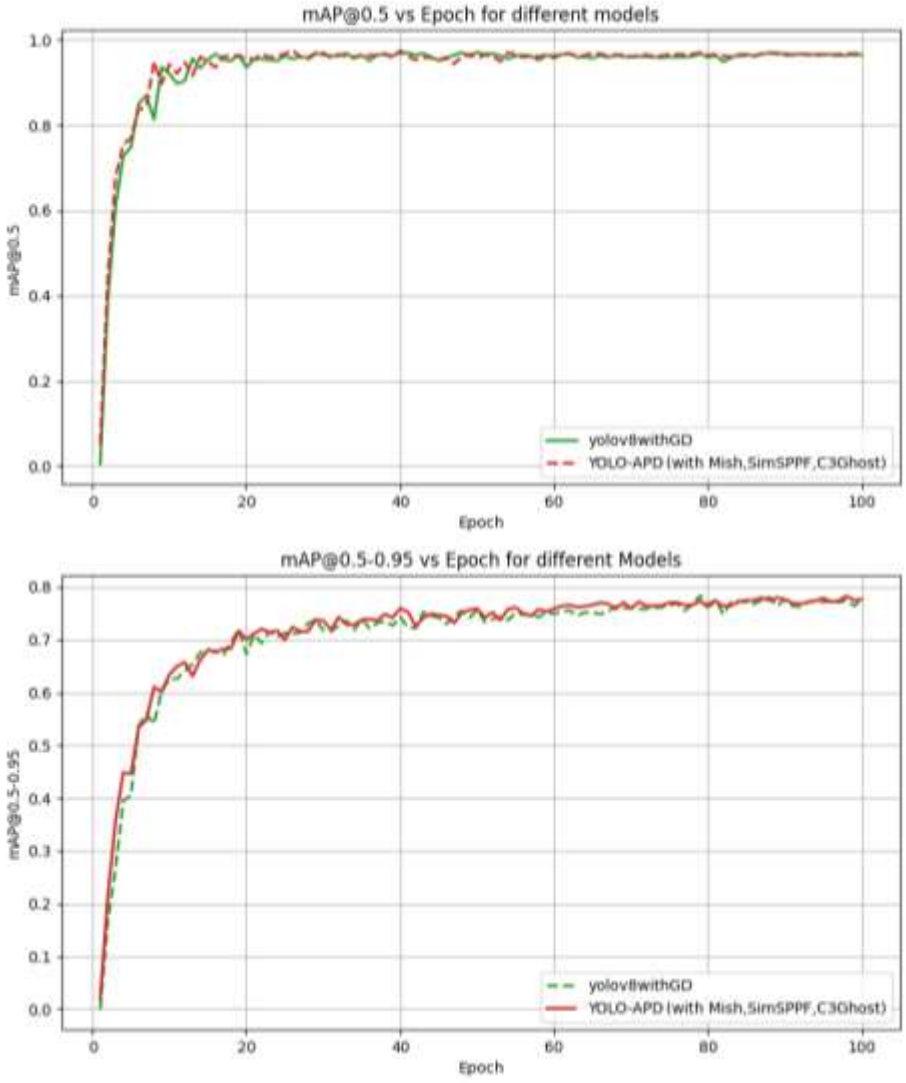

Figure 36 Comparison with incremental modular improvements

modules. The performance of these configurations was evaluated over 100 epochs in terms of mAP@0.5 and mAP@0.5-0.95. Both models demonstrated strong detection capabilities, reaching high mAP@0.5 values (close to 1.0) within 20 epochs, maintaining stable and

consistent performance throughout training. The GD-enhanced YOLOv8 and the Mish-SimSPPF-C3Ghost variant followed nearly identical trends, indicating that while the latter incorporates additional modules, their impact on performance at this IoU threshold remains minimal. The mAP@0.5-0.95 metric, which presents a more challenging evaluation, showed both models achieving values close to 0.8 by the end of training, with minor fluctuations. The Mish-SimSPPF-C3Ghost variant exhibited a slight advantage during early training epochs, suggesting a potential benefit in handling varying IoU thresholds, though the overall difference between the two remained marginal. Across both evaluations, the choice of additional modules did not significantly alter detection accuracy, indicating that GD and Mish-SimSPPF-C3Ghost configurations offer comparable performance. While the Mish-SimSPPF-C3Ghost setup may provide slight advantages in more complex detection scenarios, these improvements remain relatively minor, and both configurations exhibit strong and reliable precision throughout the training process.

4.4. Summary of the chapter

In sumaary, our assessment of YOLO-APD showed substantial improvements as compared to YOLOv8, particularly in the case of pedestrian detection in difficult conditions, which included occlusions, low-light environments, and urban settings with motion. The improved accuracy with statistical significance is an indication that YOLO-APD decreased false negatives while maintaining a similar false positive rate. YOLO-APD also demonstrated real-time performance on low-power embedded hardware, which may lead to deployment feasibility for an autonomous driving application. Through utilizing steering-aware detection combined with dual-feed validation, YOLO-APD prioritized the safety of pedestrians through detection zones in conjunction with the trajectory of the vehicle. Ultimately, these results suggested that YOLO-APD embodied a reliable and efficient object detection framework for seamless

autonomous navigation and offered a framework to address next steps toward optimization and on-road implementation.

CHAPTER FIVE

CONCLUSION AND RECOMMENDATIONS

5.1. Introduction

The aim of this study was to examine the viability of the adaptive pedestrian detection model for autonomous vehicle. Based on the results obtained it can be concluded that this model can be used as a viable alternative to the existing techniques that utilize expensive sensor technologies.

To improve real-time pedestrian detection in front of Autonomous Vehicle in roads with challenging settings this study leveraged on deep learning (DL) techniques, based on YOLOv8 to develop as solution to this persisting problem. The name of the model was named YOLO-APD. It utilizes a number of improvements such as incorporation of the Mish activation function, novel SimSPPF and SimAM methods, C3Ghost methods and gather and distribute algorithms to enhance the performance significantly. YOLO-APD showed an impressive improvement on the baseline model (mAP50 = 97. 8%) on this dataset, faster convergence time, and stable performance under various conditions. The next chapters provide results from the work according to research objectives with key findings, broader implications, study limitations, and recommendations for future work.

5.2. Summary of the conclusions

Objective 1: To investigate which algorithms can effectively contribute to solving real-time detection of pedestrians with high accuracy and precision.

This study succeeded in identifying YOLO family of detectors as a suitable starting point from which it developed YOLO-APD model that has effective balance between accuracy and computational accuracy. By exploiting techniques like as pruning and knowledge distillation, the model was able to be trained more rapidly and it obtained stable performance while still maintaining high accuracy. YOLO-APD emerges as an alternative to core pedestrian detection

models especially for real time autonomous driving applications in uncertain traffic conditions like type s road.

Objective 2: To investigate methods for limiting an object detection algorithm to only detect objects in critical region of an autonomous vehicle

The study explored techniques to limit detection to relevant sections of an image feed Infront of autonomous vehicle. Making use of the vehicle steering dynamics and identifying a particular area where a lot of action happens Infront of the vehicle substantially enhanced detection efficiency, since computing resources were utilized on pertinent detection components with a focus on detection on real-time a hazard before AV.The ability of YOLO-APD to prioritize critical visual data suggests potential for further improvements in task specific optimization.

Objective 3: To design an adaptable pedestrian detection algorithm that minimizes latency and improves perception in autonomous vehicles in challenging road scenes like type S road.

The study successfully developed a low-latency, high-precision pedestrian detection model optimized for resource-limited environments using affordable yet efficient and scalable RGB stereo camera and smart deep learning techniques. YOLO-APD maintains consistent detection accuracy across different scenarios, including complex road structures such as Type-S roads. The findings indicate that the model is well-suited for rugged autonomous driving applications, balancing computational demand with detection reliability.

Objective 4: To Evaluate the algorithm against state of art pedestrian detection and compare its performance.

Comparative analysis revealed that YOLO-APD outperformed existing YOLO-based pedestrian detectors ie YOLOv5, YOLOv7 including a slight completive edge to YOLOv9,

particularly in terms of early convergence and stable detection performance. The study highlights YOLO-APD's robustness, making it a strong candidate for real-world pedestrian detection deployment.

Several key insights emerged from this research:

- Computational Optimization: The use of pruning and knowledge distillation significantly reduced model complexity while maintaining detection accuracy, making YOLO-APD more efficient for real-time applications.
- Stable Training and Convergence: Unlike some traditional models that require extensive training cycles, YOLO-APD exhibits fast and stable convergence, allowing for quicker deployment.
- Adaptability to Different Scenarios: YOLO-APD demonstrated strong performance in diverse autonomous driving environments, including challenging road conditions.

Nighttime and Weather Robustness: Unlike some pedestrian detection models that struggle with low-light conditions or adverse weather, YOLO-APD maintained consistent performance, thanks to its attention mechanism, suggesting its resilience to environmental variations

5.3. Limitations

Despite the success of the study, several limitations were encountered.

Although pruning and distillation helped improve efficiency of the model, there was a need for a powerful computing platform in order to further optimize the model for real world experience on latest Simulation platform for Unreal Engine is expensive.

There are several components of this that pose a potential study point yet given the time limit they remained unexplored.

While the model performed well on the Carla dataset, further testing on real-world balanced datasets is required to evaluate its performance beyond Carla Simulation environment.

5.4. Recommendations for future Work.

To address the limitations and expand on the findings of this study, future research should focus on

1. Real-World Testing: Expanding evaluation beyond CARLA by testing YOLO-APD on real-world pedestrian datasets to verify its robustness under natural traffic conditions.
2. Further Computational Optimization: Implementing quantization techniques alongside pruning and distillation to further reduce the model size while maintaining accuracy for possible deployment on edge-devices.
3. Integration with Multi-modal perception systems ie fusion with LiDAR, radar inputs to enhance its detection capabilities further.
4. Integration with Artificial Intelligent Agents to automate the process of Adaptability of this model.

REFERENCES

Abbasi, R., Bashir, A. K., Alyamani, H. J., Amin, F., Doh, J., & Chen, J. (2023). Lidar Point Cloud Compression, Processing and Learning for Autonomous Driving. *IEEE Transactions on Intelligent Transportation Systems*, *24*(1), 962–979. https://doi.org/10.1109/TITS.2022.3167957

Agung Premananda, I. G., Tjahyanto, A., & Mukhlason, A. (2022). Design Science Research Methodology and Its Application to Developing a New Timetabling Algorithm. *Proceedings - 2022 IEEE International Conference on Cybernetics and Computational Intelligence, CyberneticsCom 2022*, 433–438. https://doi.org/10.1109/CYBERNETICSCOM55287.2022.9865661

Akyol, G., Kantarcı, A., Çelik, A. E., & Cihan Ak, A. (2020). Deep Learning Based, Real-Time Object Detection for Autonomous Driving. *2020 28th Signal Processing and Communications Applications Conference (SIU)*, 1–4. https://doi.org/10.1109/SIU49456.2020.9302500

Ali, K., Moetesum, M., Siddiqi, I., & Mahmood, N. (2022). Marine Object Detection using Transformers. *Proceedings of 2022 19th International Bhurban Conference on Applied Sciences and Technology, IBCAST 2022*, 951–957. https://doi.org/10.1109/IBCAST54850.2022.9990099

Basit, A., Ejaz, M. U., Ayaz, Q., & Malik, F. M. (2023). Real-time object detection and 3D scene perception in self-driving cars. *2023 3rd International Conference on Artificial Intelligence (ICAI)*, 109–115. https://doi.org/10.1109/ICAI58407.2023.10136623

Bochkovskiy, A., Wang, C.-Y., & Liao, H.-Y. M. (2020). *YOLOv4: Optimal Speed and Accuracy of Object Detection*. http://arxiv.org/abs/2004.10934

Bu, F., Le, T., Du, X., Vasudevan, R., & Johnson-Roberson, M. (2020). Pedestrian Planar LiDAR Pose (PPLP) Network for Oriented Pedestrian Detection Based on Planar LiDAR and Monocular Images. *IEEE Robotics and Automation Letters*, *5*(2), 1626–1633. https://doi.org/10.1109/LRA.2019.2962358

Bu, T., Zhang, X., Mertz, C., & Dolan, J. M. (2021). CARLA Simulated Data for Rare Road Object Detection. *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*, *2021-September*, 2794–2801. https://doi.org/10.1109/ITSC48978.2021.9564932

Caesar, H., Bankiti, V., Lang, A. H., Vora, S., Liong, V. E., Xu, Q., Krishnan, A., Pan, Y., Baldan, G., Beijbom, O., & Company, A. (2020). *nuScenes: A multimodal dataset for autonomous driving*.

Cao, Y., Wang, Y., & Fan, H. (2023). Improved YOLOv5s Network for Traffic Object Detection with Complex Road Scenes. *2023 IEEE 13th International Conference on CYBER Technology in Automation, Control, and Intelligent Systems (CYBER)*, 362–366. https://doi.org/10.1109/CYBER59472.2023.10256488

Chan, H. T., Tsai, P. T., & Hsia, C. H. (2023). Multispectral Pedestrian Detection Via Two-Stream YOLO With Complementarity Fusion For Autonomous Driving. *2023 IEEE 3rd International Conference on Electronic Communications, Internet of Things and Big Data, ICEIB 2023*, 313–316. https://doi.org/10.1109/ICEIB57887.2023.10170473

Chen, Y., Dai, X., Liu, M., Chen, D., Yuan, L., & Liu, Z. (2020). Dynamic convolution: Attention over convolution kernels. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 11027–11036. https://doi.org/10.1109/CVPR42600.2020.01104

Dasgupta, R., Chowdhury, Y. S., & Nanda, S. (2021). Performance Comparison of Benchmark Activation Function ReLU, Swish and Mish for Facial Mask Detection Using Convolutional Neural Network. In A. Sheth, A. Sinhal, A. Shrivastava, & A. K. Pandey (Eds.), *Intelligent Systems* (pp. 355–367). Springer Singapore.

Gawade, P., & Suresh Kumar, P. (2021). In-Vehicle Speech Command Operated Driver Assist System for Vehicle Actuators Control using Deep Learning Techniques. *2021 5th International Conference on Computer, Communication, and Signal Processing, ICCCSP 2021*, 262–267. https://doi.org/10.1109/ICCCSP52374.2021.9465511

Guo, M.-H., Xu, T.-X., Liu, J.-J., Liu, Z.-N., Jiang, P.-T., Mu, T.-J., Zhang, S.-H., Martin, R. R., Cheng, M.-M., & Hu, S.-M. (2021). Attention Mechanisms in Computer Vision: A Survey. *Computational Visual Media*, *8*(3), 331–368. https://doi.org/10.1007/s41095-022-0271-y

Han, K., Wang, Y., Tian, Q., Guo, J., Xu, C., & Xu, C. (2019). GhostNet: More Features from Cheap Operations. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1577–1586. https://doi.org/10.1109/CVPR42600.2020.00165

Hassaballah, M., Kenk, M. A., Muhammad, K., & Minaee, S. (2021). Vehicle Detection and Tracking in Adverse Weather Using a Deep Learning Framework. *IEEE Transactions on Intelligent Transportation Systems*, *22*(7), 4230–4242. https://doi.org/10.1109/TITS.2020.3014013

Hou, Q., Zhou, D., & Feng, J. (2021). Coordinate attention for efficient mobile network design. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 13708–13717. https://doi.org/10.1109/CVPR46437.2021.01350

Huang, X. X., Yin, Z. Y., & Fan, C. (2022). Towards Better Pedestrian Detection Using Multi-Scale CSPN and Dual Attention. *Proceedings - 2022 11th International Conference of Information and Communication Technology, ICTech 2022*, 451–456. https://doi.org/10.1109/ICTECH55460.2022.00096

Huo, L., Wang, Y., Liu, T., & Fan, Y. (2022). Overview of pedestrian detection based on infrared image. *2022 41st Chinese Control Conference (CCC)*, 6357–6362. https://doi.org/10.23919/CCC55666.2022.9902509

Janai, J., Güney, F., Behl, A., & Geiger, A. (2020). Computer Vision for Autonomous Vehicles: Problems, Datasets and State of the Art. *Foundations and Trends® in Computer Graphics and Vision*, *12*(1–3), 1–308. https://doi.org/10.1561/0600000079

Jhong, S. Y., Wang, Y. Q., Cheng, W. J., Hwang, H. W., & Chen, Y. Y. (2022). LiDAR-Based Pedestrian Detection Using Multiple Features and Dimensionality Reduction Scheme. *ICSSE 2022 - 2022 International Conference on System Science and Engineering*, 7–10. https://doi.org/10.1109/ICSSE55923.2022.9948235

Kaljavesi, G., Kerbl, T., Betz, T., Mitkovskii, K., & Diermeyer, F. (2024). *CARLA-Autoware-Bridge: Facilitating Autonomous Driving Research with a Unified Framework for Simulation and Module Development*. http://arxiv.org/abs/2402.11239

Khanam, R., & Hussain, M. (2024). *What is YOLOv5: A deep look into the internal features of the popular object detector*. http://arxiv.org/abs/2407.20892

Khanum, A., Lee, C. Y., & Yang, C. S. (2020). End-to-end deep learning model for steering angle control of autonomous vehicles. *Proceedings - 2020 International Symposium on Computer, Consumer and Control, IS3C 2020*, 189–192. https://doi.org/10.1109/IS3C50286.2020.00056

Kochengin, A., Shikhin, V., & Pavliuk, G. (2020). Improving performance of wilcoxon criterion under solving problems with typical singularities in measurements. *Proceedings - 2020 International Conference on Industrial Engineering, Applications and Manufacturing, ICIEAM 2020*. https://doi.org/10.1109/ICIEAM48468.2020.9112003

Koohpayegani, S. A., & Pirsiavash, H. (2022). *SimA: Simple Softmax-free Attention for Vision Transformers*. https://arxiv.org/abs/2206.08898v1

Kuhrmann, M., Tell, P., Hebig, R., Klünder, J., Schneider, K., Münch, J., Linssen, O., Pfahl, D., Scott, E., Felderer, M., Prause, C. R., MacDonell, S. G., Nakatumba-Nabende, J., Raffo, D., Beecham, S., Tüzün, E., López, G., Paez, N., Fontdevila, D., … Richardson, I. (2022). What Makes Agile Software Development Agile? *IEEE Transactions on Software Engineering*, *48*(9), 3523–3539. https://doi.org/10.1109/TSE.2021.3099532

Kuutti, S., Bowden, R., Jin, Y., Barber, P., & Fallah, S. (2021). A Survey of Deep Learning Applications to Autonomous Vehicle Control. *IEEE Transactions on Intelligent Transportation Systems*, *22*(2), 712–733. https://doi.org/10.1109/TITS.2019.2962338

Lee, G., & Cho, J. (2022). STDP-Net: Improved Pedestrian Attribute Recognition Using Swin Transformer and Semantic Self-Attention. *IEEE Access*, *10*, 82656–82667. https://doi.org/10.1109/ACCESS.2022.3196650

Li, C., Zhou, A., & Yao, A. (2022). Omni-Dimensional Dynamic Convolution. *ICLR 2022 - 10th International Conference on Learning Representations*. https://arxiv.org/abs/2209.07947v1

Li, W. (2021). Infrared Image Pedestrian Detection via YOLO-V3. *2021 IEEE 5th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*, *5*, 1052–1055. https://doi.org/10.1109/IAEAC50856.2021.9390896

Li, W., Liu, K., Zhang, L., & Cheng, F. (2020). Object detection based on an adaptive attention mechanism. *Scientific Reports 2020 10:1*, *10*(1), 1–13. https://doi.org/10.1038/s41598-020-67529-x

Li, Y., Zhang, H., & Zhang, Y. (2021). *Rethinking Training from Scratch for Object Detection*. https://arxiv.org/abs/2106.03112v1

Listopad, S. (2023). Architecture of Reflexive-active Systems of Artificial Heterogeneous Intelligent Agents. *Proceedings - 2023 5th International Conference on Control Systems, Mathematical Modeling, Automation and Energy Efficiency, SUMMA 2023*, 274–278. https://doi.org/10.1109/SUMMA60232.2023.10349610

Liu, M., Yurtsever, E., Fossaert, J., Zhou, X., Zimmer, W., Cui, Y., Zagar, B. L., & Knoll, A. C. (2024). *A Survey on Autonomous Driving Datasets: Statistics, Annotation Quality, and a Future Outlook*. https://arxiv.org/abs/2401.01454v2

Liu, Y., Shao, Z., Teng, Y., & Hoffmann, N. (2021). *NAM: Normalization-based Attention Module*. https://arxiv.org/abs/2111.12419v1

Lowe, J., & Kuru, K. (2024). Development of Machine Intelligence for Self-Driving Vehicles Through Video Capturing. *MESA 2024 - 20th International Conference on Mechatronic, Embedded Systems and Applications, Proceedings*. https://doi.org/10.1109/MESA61532.2024.10704876

Lu, J. (2022). *Autonomous Vision of Driverless car in Machine Learning*.

Manda, M. P., Park, C., Oh, B., Hyun, D., & Kim, H. S. (2020). Pedestrian Detection in Infrared Thermal Images Based on Raised Cosine Distribution. *Proceedings - International SoC Design Conference, ISOCC 2020*, 278–279. https://doi.org/10.1109/ISOCC50952.2020.9332804

Markkula, G., Madigan, R., Nathanael, D., Portouli, E., Lee, Y. M., Dietrich, A., Billington, J., Schieben, A., & Merat, N. (2020). Defining interactions: a conceptual framework for understanding interactive behaviour in human and automated road traffic. *Https://Doi.Org/10.1080/1463922X.2020.1736686*, *21*(6), 728–752. https://doi.org/10.1080/1463922X.2020.1736686

Meng, L., Zhou, J., Ma, J., & Wang, Z. (2022). MPDFF: Multi-source Pedestrian detection based on Feature Fusion. *IGARSS 2022 - 2022 IEEE International Geoscience and Remote Sensing Symposium*, 7906–7909. https://doi.org/10.1109/IGARSS46834.2022.9884864

Naeem, M. A., Jia, X., Saleem, M. A., Akbar, W., Hussain, A., Nazir, S., & Ahmad, K. M. (2020). Vehicle to Everything (V2X) Communication Protocol by Using Vehicular AD-HOC Network. *2020 17th International Computer Conference on Wavelet Active Media*

Technology and Information Processing, ICCWAMTIP 2020, 384–388. https://doi.org/10.1109/ICCWAMTIP51612.2020.9317386

Niranjan, D. R., VinayKarthik, B. C., & Mohana. (2021). Performance Analysis of SSD and Faster RCNN Multi-class Object Detection Model for Autonomous Driving Vehicle Research Using CARLA Simulator. *2021 Fourth International Conference on Electrical, Computer and Communication Technologies (ICECCT)*, 1–6. https://doi.org/10.1109/ICECCT52121.2021.9616712

Pranav, K. B., & Manikandan, J. (2020). Design and Evaluation of a Real-time Pedestrian Detection System for Autonomous Vehicles. *2020 Zooming Innovation in Consumer Technologies Conference, ZINC 2020*, 155–159. https://doi.org/10.1109/ZINC50678.2020.9161768

Qiu, J., Liu, J., & Shen, Y. (2021). Computer Vision Technology Based on Deep Learning. *Proceedings of 2021 IEEE 2nd International Conference on Information Technology, Big Data and Artificial Intelligence, ICIBA 2021*, 1126–1130. https://doi.org/10.1109/ICIBA52610.2021.9687873

R, K., & S, N. (2021). Pothole and Object Detection for an Autonomous Vehicle Using YOLO. *2021 5th International Conference on Intelligent Computing and Control Systems (ICICCS)*, 1585–1589. https://doi.org/10.1109/ICICCS51141.2021.9432186

Ranjan, S., & Senthamilarasu, S. (2020). *Applied Deep Learning and Computer Vision for Self-Driving Cars*. Packt Publishing.

Razzaghi, A., Soori, H., Abadi, A., & Khosravi, A. (2020). World Health Organization's estimates of death related to road traffic crashes and their discrepancy with other countries' national report. *Journal of Injury and Violence Research*, *12*(3 Suppl 1), 39. https://doi.org/10.5249/JIVR.V12I3.1425

Sahba, R., Sahba, A., & Sahba, F. (2020). Using a Combination of LiDAR, RADAR, and Image Data for 3D Object Detection in Autonomous Vehicles. *2020 11th IEEE Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, 427–431. https://doi.org/10.1109/IEMCON51383.2020.9284930

Shen, G., Li, X., & Wei, Y. (2022). Improved algorithm for pedestrian detection of lane line based on YOLOv5s model. *IEEE Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*, *2022-October*, 405–409. https://doi.org/10.1109/IAEAC54830.2022.9930101

Song, Y., Li, M., Qiu, X., Du, W., & Feng, J. (2020). Full-time infrared feature pedestrian detection based on csp network. *Proceedings - 2020 International Conference on Intelligent Transportation, Big Data and Smart City, ICITBS 2020*, 516–518. https://doi.org/10.1109/ICITBS49701.2020.00112

Srivastava, R., & Kumar, V. (2020). Accident avoidance simulation using SUMO. *Proceedings of the 2020 9th International Conference on System Modeling and Advancement in*

Research Trends, SMART 2020, 179–183. https://doi.org/10.1109/SMART50582.2020.9337079

Stevens, E., Antiga, L. P. G., & Viehmann, T. (2020). *Deep Learning with PyTorch*. Manning. http://ieeexplore.ieee.org/document/10280272

Szeliski, R. (2022). Computer vision : algorithms and applications . In *Computer vision : algorithms and applications* (Second edition.). Springer.

Tarchoun, B., Jegham, I., Khalifa, A. Ben, Alouani, I., & Mahjoub, M. A. (2020). Deep CNN-based Pedestrian Detection for Intelligent Infrastructure. *2020 5th International Conference on Advanced Technologies for Signal and Image Processing (ATSIP)*, 1–6. https://doi.org/10.1109/ATSIP49331.2020.9231712

Terven, J., & Cordova-Esparza, D. (2023). *A Comprehensive Review of YOLO: From YOLOv1 to YOLOv8 and Beyond*. http://arxiv.org/abs/2304.00501

The future of autonomous vehicles (AV) | McKinsey. (n.d.). Retrieved March 10, 2025, from https://www.mckinsey.com/industries/automotive-and-assembly/our-insights/autonomous-drivings-future-convenient-and-connected

Tian, L., Chin, C.-T., Cheng, C.-H., & Huang, R.-P. (2022). Night Pedestrian Detection Using Thermal Image Feature Extraction Enhanced YOLOv3 (TIFEEY). *2022 International Conference on Image Processing, Computer Vision and Machine Learning (ICICML)*, 80–89. https://doi.org/10.1109/ICICML57342.2022.10009888

Uřičář, M., Rashed, H., Ranga, A., Dahal, A., & Yogamani, S. (2020). VisibilityNet: Camera visibility detection and image restoration for autonomous driving. *Electronic Imaging*, *32*(16), 79–1–79–1. https://doi.org/10.2352/ISSN.2470-1173.2020.16.AVM-079

Urmila, O., & Megalingam, R. K. (2020). Processing of LiDAR for Traffic Scene Perception of Autonomous Vehicles. *Proceedings of the 2020 IEEE International Conference on Communication and Signal Processing, ICCSP 2020*, 298–301. https://doi.org/10.1109/ICCSP48568.2020.9182175

van der Merwe, A., Gerber, A., & Smuts, H. (2020). Guidelines for conducting design science research in information systems. *Communications in Computer and Information Science*, *1136 CCIS*, 163–178. https://doi.org/10.1007/978-3-030-35629-3_11/COVER

Varghese, R., & Sambath, M. (2024). YOLOv8: A Novel Object Detection Algorithm with Enhanced Performance and Robustness. *2024 International Conference on Advances in Data Engineering and Intelligent Computing Systems, ADICS 2024*. https://doi.org/10.1109/ADICS58448.2024.10533619

Verstraete, T., & Muhammad, N. (2024). Pedestrian Collision Avoidance in Autonomous Vehicles: A Review. *Computers 2024, Vol. 13, Page 78*, *13*(3), 78. https://doi.org/10.3390/COMPUTERS13030078

Wang, C., He, W., Nie, Y., Guo, J., Liu, C., Han, K., & Wang, Y. (2023). *Gold-YOLO: Efficient Object Detector via Gather-and-Distribute Mechanism*. https://arxiv.org/abs/2309.11331v5

Wu, C., Zhao, P., Zhang, T., Zhang, W., Meng, F., & Guo, R. (2023). A Small Target Detection Algorithm of Complex Traffic Road Based on SDC-YOLO. *2023 IEEE 6th International Conference on Electronic Information and Communication Technology, ICEICT 2023*, 1266–1269. https://doi.org/10.1109/ICEICT57916.2023.10245967

Xiang, Y., Wang, S., Su, T., Li, J., Mao, S. S., & Geimer, M. (2021). KIT Bus: A Shuttle Model for CARLA Simulator. *IEACon 2021 - 2021 IEEE Industrial Electronics and Applications Conference*, 7–12. https://doi.org/10.1109/IEACON51066.2021.9654633

Xiao, B., Guo, J., & He, Z. (2021). Real-Time Object Detection Algorithm of Autonomous Vehicles Based on Improved YOLOv5s. *2021 5th CAA International Conference on Vehicular Control and Intelligence (CVCI)*, 1–6. https://doi.org/10.1109/CVCI54083.2021.9661149

Xu, C., Wang, G., Yan, S., Yu, J., Zhang, B., Dai, S., Li, Y., Xu, L., & Hwang, F. J. (2020). Fast Vehicle and Pedestrian Detection Using Improved Mask R-CNN. *Mathematical Problems in Engineering*, *2020*. https://doi.org/10.1155/2020/5761414

Xue, P., Chen, H., Li, Y., & Li, J. (2023). Multi-scale pedestrian detection with global–local attention and multi-scale receptive field context. *IET Computer Vision*, *17*(1), 13–25. https://doi.org/10.1049/CVI2.12125

Yang, L., Zhang, R.-Y., Li, L., & Xie, X. (2021). SimAM: A Simple, Parameter-Free Attention Module for Convolutional Neural Networks. In M. Meila & T. Zhang (Eds.), *Proceedings of the 38th International Conference on Machine Learning* (Vol. 139, pp. 11863–11874). PMLR. https://proceedings.mlr.press/v139/yang21o.html

Yaseen, M. (2024). *What is YOLOv8: An In-Depth Exploration of the Internal Features of the Next-Generation Object Detector*. https://arxiv.org/abs/2408.15857v1

Yu, J., Wang, H., Zhou, S., & Li, S. (2023). Infrared Target Detection Based on Interval Sampling Weighting and 3D Attention Head in Complex Scenario. *Applied Sciences 2024, Vol. 14, Page 249*, *14*(1), 249. https://doi.org/10.3390/APP14010249

Zagirov, A., Chebotareva, E., & Osokin, S. (2024). Human-Robot Collision Analysis in Gazebo Environment. *Proceedings of the 2024 8th International Conference on Information, Control, and Communication Technologies, ICCT 2024*. https://doi.org/10.1109/ICCT62929.2024.10874932

Zhang, Y.-J. (2024). 3D Computer Vision. In *3D Computer Vision*. Springer Nature Singapore. https://doi.org/10.1007/978-981-19-7603-2

Zhao, X., Sun, P., Xu, Z., Min, H., & Yu, H. (2020). Fusion of 3D LIDAR and Camera Data for Object Detection in Autonomous Vehicle Applications. *IEEE Sensors Journal*, *20*(9), 4901–4913. https://doi.org/10.1109/JSEN.2020.2966034

Zhu, D., Xu, G., Zhou, J., Di, E., & Li, M. (2021). Object Detection in Complex Road Scenarios: Improved YOLOv4-Tiny Algorithm. *2021 2nd Information Communication Technologies Conference, ICTC 2021*, 75–80. https://doi.org/10.1109/ICTC51749.2021.9441643

Zhu, X., Su, W., Lu, L., Li, B., Wang, X., & Dai, J. (2020). Deformable DETR: Deformable Transformers for End-to-End Object Detection. *ICLR 2021 - 9th International Conference on Learning Representations*. https://arxiv.org/abs/2010.04159v4

Ziryawulawo, A., Mduma, N., Lyimo, M., Mbarebaki, A., Madanda, R., & Sam, A. (2023). An Integrated Deep Learning-based Lane Departure Warning and Blind Spot Detection System: A Case Study for the Kayoola Buses. *1st International Conference on the Advancements of Artificial Intelligence in African Context, AAIAC 2023*. https://doi.org/10.1109/AAIAC60008.2023.10465276

APPENDIX A: Python Code snippet for Region of Interest(ROI)

```
import cv2

def ROI(image_path, roi_coordinates):
    """
    Extracts the region of interest (ROI) from the image and resizes it.

    :param image_path: Path to the input image.
    :param roi_coordinates: Tuple (x, y, width, height) defining the
ROI.
    :return: The processed image (640x640).
    """
    # Read the image
    image = cv2.imread(image_path)
    if image is None:
        raise FileNotFoundError(f"Image not found at {image_path}")

    # Extract the ROI
    x, y, w, h = roi_coordinates
    roi = image[y:y+h, x:x+w]

    # Resize the ROI to 640x640
    resized_roi = cv2.resize(roi, (640, 640),
interpolation=cv2.INTER_AREA)

    return resized_roi

# Example usage
input_image_path = "input_image.jpg"  # Replace with your image path
roi_coordinates = (100, 200, 800, 600)  # Replace with ROI from lane
detection model

try:
    # Process the image
    processed_image = ROI(input_image_path, roi_coordinates)

    # Save the processed image
    output_image_path = "processed_image.jpg"
    cv2.imwrite(output_image_path, processed_image)
    print(f"Processed image saved at {output_image_path}")
except FileNotFoundError as e:
    print(e)
```

Figure 37: Python Code for Region of Interest (R.O.I)

NB: other codes attached in softcopy

APPENDIX B: Code for object motion using Kalman filter

```
import numpy as np

class KalmanFilter:
    def __init__(self, dt=1.0, process_noise=1.0, measurement_noise=1.0):
        self.dt = dt  # Time step

        # State vector [x, y, vx, vy]^T
        self.x = np.zeros((4, 1))

        # State transition matrix (F)
        self.F = np.array([[1, 0, dt, 0],
                           [0, 1, 0, dt],
                           [0, 0, 1, 0],
                           [0, 0, 0, 1]])

        # Process noise covariance (Q)
        self.Q = process_noise * np.array([[dt**4/4, 0, dt**3/2, 0],
                                           [0, dt**4/4, 0, dt**3/2],
                                           [dt**3/2, 0, dt**2, 0],
                                           [0, dt**3/2, 0, dt**2]])

        # Observation matrix (H) - We measure only position (x, y)
        self.H = np.array([[1, 0, 0, 0],
                           [0, 1, 0, 0]])

        # Measurement noise covariance (R)
        self.R = measurement_noise * np.eye(2)

        # Initial state covariance matrix (P)
        self.P = np.eye(4)

    def predict(self):
        """Prediction step: Estimates the next state."""
        self.x = np.dot(self.F, self.x)  # State prediction
        self.P = np.dot(np.dot(self.F, self.P), self.F.T) + self.Q  #
Covariance update

    def update(self, z):
        """Update step: Incorporates a new measurement (z)."""
        z = np.reshape(z, (2, 1))  # Ensure measurement is a column vector

        # Kalman Gain (K)
        S = np.dot(self.H, np.dot(self.P, self.H.T)) + self.R
        K = np.dot(np.dot(self.P, self.H.T), np.linalg.inv(S))

        # State update
        y = z - np.dot(self.H, self.x)  # Measurement residual
        self.x += np.dot(K, y)
```

APPENDIX C Pseudocode of SimSPPF

```
Pseudocode for SimSPPF Module
Input: Feature maps of Pedestrian or road images
Output: Processed feature maps after SimSPPF operations

1. Define SimConv module:
    a. function Conv2d(cin, cout, k, s, g, bias=False)
    b. function BatchNorm2d(cout, eps, momentum)
    c. function Mish(α, inplace=True)
    d. return Mish(BatchNorm2d(Conv2d(x)))

2. Define SimSPPF module:
    a. Set parameters:
        - kernel_size = 5, stride (s) = 1, groups (g) = 4, padding (p) = 2
        - α = 0.1, eps = 0.01

    b. Initialize convolutional layers:
        - cv1 = SimConv(c1, c, 1, 1, g)
        - cv2 = SimConv(4 * c, c, 3, 1, g)
        - cv3 = SimConv(c, c, 3, 1, g)
        - cv4 = SimConv(c, c, 3, 1, g)

    c. Define MaxPool2d function:
        - function MaxPool2d(kernel_size, stride, padding)

3. Forward operation:
    a. x1 = cv3(cv2(cv1(input)))
    b. y1 = MaxPool2d(x1)
    c. y2 = MaxPool2d(y1)
    d. y2 = cv2(cv4(concatenate((x1, y1, y2, MaxPool2d(y2)), dim=1)))

4. Return processed feature map y2
End
```

Figure 38 Pseudocode of SimSPPF

NB: other codes included in soft copy for brevity

APPENDIX D: Code for initializing Carla Server and Client simultaneously

```
import carla
import numpy as np
import cv2
import pygame
from pygame.locals import *
import random
import time
import torch
from ultralytics import YOLO

# Connect to CARLA
client = carla.Client("localhost", 2000)
client.set_timeout(10.0)
world = client.get_world()
blueprint_library = world.get_blueprint_library()

# Traffic Manager
traffic_manager = client.get_trafficmanager()
traffic_manager.set_global_distance_to_leading_vehicle(2.5)
```

Figure 39: Connecting Client to Carla Simulator

```
# Function to spawn vehicle with retries
def spawn_vehicle_with_retry(world, blueprint_library, max_attempts=20):
    spawn_points = world.get_map().get_spawn_points()
    vehicle_bp = blueprint_library.filter("vehicle.tesla.model3")[0]

    for attempt in range(max_attempts):
        vehicle_spawn_point = random.choice(spawn_points)
        vehicle = world.try_spawn_actor(vehicle_bp, vehicle_spawn_point)
        if vehicle is not None:
            print(f"Vehicle spawned at {vehicle_spawn_point.location}")
            return vehicle
    raise RuntimeError("Failed to spawn vehicle.")


# Spawn the vehicle
vehicle = spawn_vehicle_with_retry(world, blueprint_library)

# Enable autopilot
vehicle.set_autopilot(True, traffic_manager.get_port())
```

Figure 40 Spawning a vehicle and setting it into Autopilot

```python
import carla
import numpy as np
import cv2
import pygame
from pygame.locals import *
import random
import torch
from ultralytics import YOLO
import queue

# Connect to CARLA server
client = carla.Client("localhost", 2000)
client.set_timeout(6.0)

# Create player vehicle
world = client.get_world()
blueprint_library = world.get_blueprint_library()
vehicle_bp = blueprint_library.filter("vehicle.tesla.model3")[0]

spawn_point = random.choice(world.get_map().get_spawn_points())
vehicle = world.spawn_actor(vehicle_bp, spawn_point)

# Enable autopilot
vehicle.set_autopilot(True)

# Get the world spectator
spectator = world.get_spectator()

# Create camera
camera_bp = blueprint_library.find("sensor.camera.rgb")
camera_transform = carla.Transform(carla.Location(x=1.5, z=2.4))
camera = world.spawn_actor(camera_bp, camera_transform, attach_to=vehicle)

# Load YOLOv8 model
model = YOLO("best.pt")  # Load the YOLOAPD model

# Create a Pygame window
pygame.init()
screen_width = 640
screen_height = 480
display = pygame.display.set_mode((screen_width, screen_height))
clock = pygame.time.Clock()


# Camera image capture function
def process_image(image):
    img_array = np.array(image.raw_data)
    img_rgb = img_array.reshape((image.height, image.width, 4))[:, :, :3]
```

```
import random
import time
import numpy as np
import math
import cv2
import gym
from gym import spaces
import carla
import sys
sys.path.append('C:/CARLA_0.9.13/PythonAPI/carla') # tweak to where you put
carla
from agents.navigation.global_route_planner import GlobalRoutePlanner

from tensorflow.keras.models import load_model

SECONDS_PER_EPISODE = 25

N_CHANNELS = 3
HEIGHT = 240
WIDTH = 320

HEIGHT_REQUIRED_PORTION = 0.5 #bottom share, e.g. 0.1 is take lowest 10% of
rows
WIDTH_REQUIRED_PORTION = 0.9

FIXED_DELTA_SECONDS = 0.2

SHOW_PREVIEW = True

class CarEnv(gym.Env):
    SHOW_CAM = SHOW_PREVIEW
    STEER_AMT = 1.0
    im_width = WIDTH
    im_height = HEIGHT
    front_camera = None
    CAMERA_POS_Z = 1.3
    CAMERA_POS_X = 1.4
    PREFERRED_SPEED = 20 # what it says
    SPEED_THRESHOLD = 2 #defines when we get close to desired speed so we drop
the
```

Figure 41 code for positioning the camera Infront of the car

APPENDIX E FLOWCHART OF THE SYSTEM

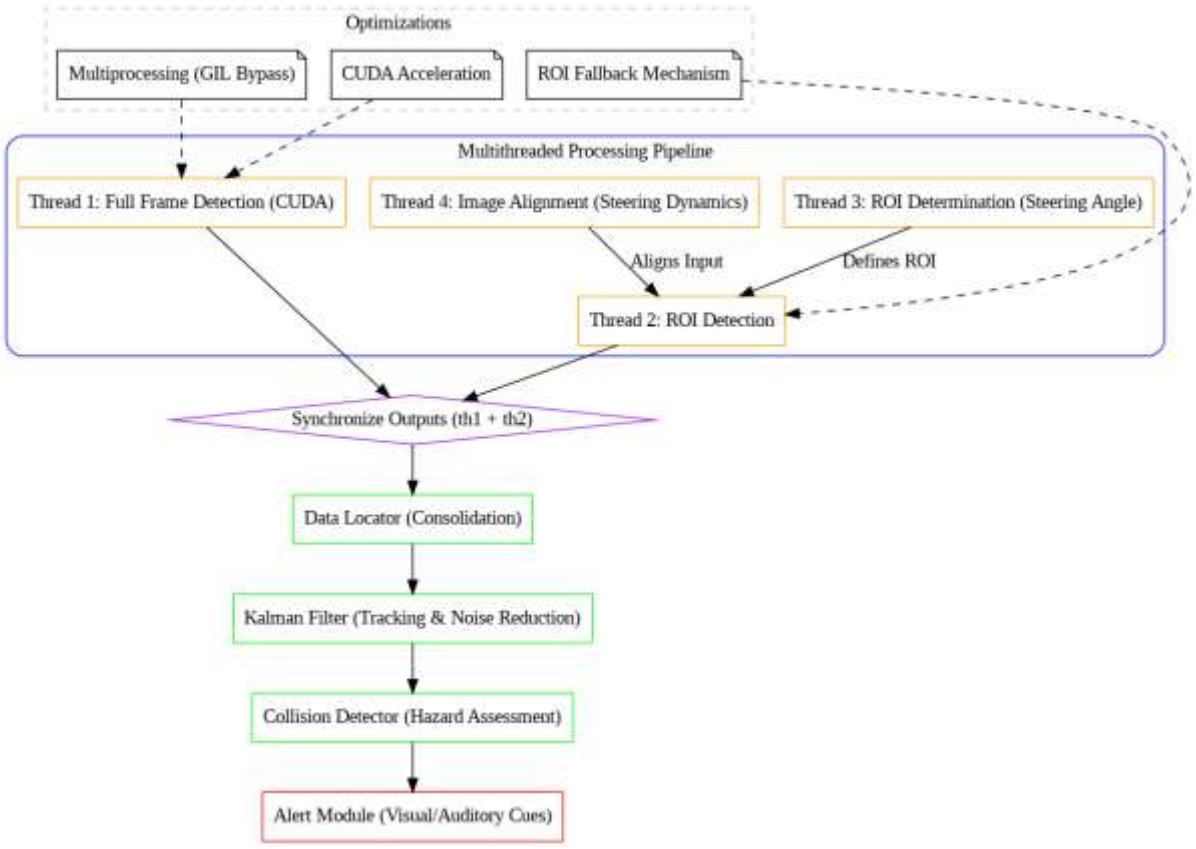

Figure 42 how the model components relate

APPENDIX F Python Code for Distance estimation

```
import numpy as np

class KalmanFilter:
    def __init__(self):
        # State vector [x, y, vx, vy]
        self.X_k = np.zeros((4, 1))

        # State transition matrix (Constant velocity model)
        self.dt = 1  # Time step (adjustable)
        self.F = np.array([[1, 0, self.dt, 0],
                           [0, 1, 0, self.dt],
                           [0, 0, 1, 0],
                           [0, 0, 0, 1]])

        # Observation matrix (we observe only x, y positions)
        self.H = np.array([[1, 0, 0, 0],
                           [0, 1, 0, 0]])

        # Process noise covariance
        self.Q = np.eye(4) * 0.01

        # Measurement noise covariance
        self.R = np.eye(2) * 0.1

        # Initial error covariance
        self.P_k = np.eye(4)

    def predict(self):
        # Predict state
        self.X_k = np.dot(self.F, self.X_k)

        # Predict error covariance
        self.P_k = np.dot(np.dot(self.F, self.P_k), self.F.T) + self.Q

    def update(self, Z_k):
        # Compute Kalman Gain
        S = np.dot(self.H, np.dot(self.P_k, self.H.T)) + self.R
        K_k = np.dot(np.dot(self.P_k, self.H.T), np.linalg.inv(S))

        # Update state estimate
        self.X_k = self.X_k + np.dot(K_k, (Z_k - np.dot(self.H, self.X_k)))

        # Update error covariance
        self.P_k = np.dot((np.eye(4) - np.dot(K_k, self.H)), self.P_k)
```

APPENDIX F PSEUDOCODE FOR YOLOAPD DETECTION

```
BEGIN
    # Initialization
    Load YOLO-APD model with pre-trained weights
    Set model parameters (input size, confidence threshold, NMS threshold)

    # Input Preprocessing and Feature Extraction
    FOR each input image DO
        Resize image to (640, 640)
        Normalize pixel values (0-255 → 0-1)
        Convert image to tensor

        # Feature Extraction using Gather Module
        Extract hierarchical features using Backbone (CSPDarknet + C2f)
        Enhance multi-scale features using Neck (FPN + PAN)

        # Distribute features for different levels of detection
        IF high-resolution features needed THEN
            Process through **small-object detection branch**
        ELSE
            Process through **standard detection branch**
        END IF

        # Predict Bounding Boxes, Class Scores, and Objectness
        Generate multi-scale feature maps
        Assign predictions to **adaptive anchor-free heads**

        # Segmentation (If enabled)
        IF segmentation is enabled THEN
            Generate segmentation masks
        END IF

        # Post-processing (Inference)
        Filter detections based on confidence threshold
        Apply Non-Maximum Suppression (NMS)
        Scale bounding box coordinates back to original image size
    END FOR

    # Loss Calculation (During Training)
    IF training is enabled THEN
        Compute classification loss using VariFocal Loss
        Compute bounding box regression loss using CIoU Loss
        Compute label assignment loss using Distribution Focal Loss
        Optimize gather-and-distribute mechanism for detection refinement
    END IF

    # Output
    RETURN:
        List of detected objects (bounding boxes, class labels, confidence
scores)
        Segmentation masks (if applicable)
        Class labels (if applicable)
END
```

APPENDIX G ALGORITHM FOR YOLO-APD PEDESTRIAN DETECTION PART

```
Algorithm: YOLO-APD
Input: Image I
Output: List of detected objects O with bounding boxes, class labels, and
confidence scores

1.  Initialize YOLO-GOLD model with pre-trained weights
2.  Set model parameters: input size (640x640), confidence threshold
(0.25), NMS threshold (0.45)

3.  FOR each input image I DO
    4.   Resize I to (640, 640)
    5.   Normalize pixel values to range [0,1]
    6.   Convert I to tensor

    7.   Extract hierarchical features using Backbone (CSPDarknet + C2f)
    8.   Enhance multi-scale features using Neck (FPN + PAN)

    9.   IF high-resolution features are needed THEN
           10.   Process I through the small-object detection branch
         ELSE
           11.   Process I through the standard detection branch
         ENDIF

    12.  Generate multi-scale feature maps
    13.  Assign predictions to adaptive anchor-free heads

    14.  IF segmentation is enabled THEN
           15.   Generate segmentation masks
         ENDIF

    16.  Apply confidence threshold filtering
    17.  Apply Non-Maximum Suppression (NMS)
    18.  Scale bounding box coordinates back to the original image size
END FOR

19.  IF training is enabled THEN
        20.  Compute classification loss using VariFocal Loss
        21.  Compute bounding box regression loss using CIoU Loss
        22.  Compute label assignment loss using Distribution Focal Loss
        23.  Optimize gather-and-distribute mechanism for detection
refinement
    ENDIF

24.  RETURN O: List of detected objects with bounding boxes, class labels,
and confidence scores
```