



## **Studienarbeit**

# **YOLO-Based Real-Time Object Detection for Autonomous Driving in CARLA**

by

**Suraj Bhardwaj**

University of Siegen

**Examiner 1: Prof. Dr.-Ing. Roman Obermaisser**

Chair for Embedded Systems Department

University of Siegen

**Examiner 2: Dongchen Li M.Sc.**

Doctoral Researcher at Embedded Systems Department

University of Siegen

A Studienarbeit Report Submitted to the Faculty of Electrical Engineering and Computer Science  
In Partial Fulfillment of the Requirements for the Degree of Master of Science (M.Sc.)

**International Graduate Studies in Mechatronics**

---

Universität Siegen

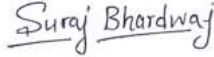
North Rhine-Westphalia, Germany

Date of Submission: January 21, 2025

## Declaration

I hereby confirm that this Studienarbeit entitled “YOLO-Based Real-Time Object Detection for Autonomous Driving in CARLA” is entirely my own work. Where I have consulted the work of others, this is always clearly stated. All statements taken literally from other writings or referred to by analogy are marked, and the source is always given. This paper has not yet been submitted to another examination office in the same or similar form.

Place, Date: **Siegen, January 21, 2025**

Student's Signature: 

## Abstract

Developing robust autonomous driving systems requires diverse datasets to train and evaluate object detection algorithms. However, real-world data collection is costly, time-intensive, and limited in capturing rare or hazardous scenarios. Simulation platforms like Car Learning to Act ([CARLA](#)) address these challenges by enabling controlled synthetic dataset generation and experimentation.

This research project utilized the [CARLA](#) simulator to develop a real-time object detection simulation for autonomous driving. A simulated Tesla Model 3, equipped with eight cameras, 12 ultrasonic sensors, and a radar, was used to generate multi-modal sensor data. The project adopted a two-stage, semi-automated annotation process to label the data. In the first stage, the GroundedSAM foundation model generated labels based on an established ontology. The second stage involved manual refinement to ensure accuracy and compliance with the annotation format required for training You Only Look Once ([YOLO](#)) models. Using this approach, the project generated a synthetic dataset containing 6,400 annotated images, capturing diverse driving scenarios, including vehicles, traffic lights, pedestrians, and traffic signs.

The dataset was used to train and evaluate state-of-the-art [YOLO](#) models, including *YOLOv8*, *YOLOv9*, *YOLOv10*, and *YOLO11*, along with their size-based variants. Among the 12 models tested, *YOLOv8m* and *YOLO11m* demonstrated the best performance, prioritizing higher localization precision and accurate classification of classes. These models also attained the highest mAP@0.5:0.95 metric. Integrating *YOLO11m* into the [CARLA](#) simulator enabled real-time object detection within the simulation environment.

The study highlighted limitations in the dataset, particularly its size and class diversity, underscoring the need for optimization techniques such as hyperparameter tuning and improvements in balancing recall and precision to reduce missed detections. Additionally, the research established a robust pipeline for collecting multi-modal sensor data, paving the way for future advancements in object detection and sensor fusion algorithms. This work contributes to progress in multi-modal synthetic datasets, object detection methodologies, model generalization, and lays the foundation for sensor fusion research in autonomous driving.

**Keywords:** CARLA simulator, synthetic datasets, object detection, YOLO models, autonomous driving, sensor fusion, real-time inference, Tesla Model 3

## **Acknowledgements**

I am deeply grateful to my advisors, Prof. Dr.-Ing. Roman Obermaisser and Dongchen Li, M.Sc., for their invaluable guidance, enduring patience, and profound expertise throughout my research journey. Their insightful feedback and thoughtful recommendations were instrumental in shaping this Studienarbeit.

I would also like to acknowledge the pioneering authors and researchers in my field whose foundational work has inspired and guided my research. Their significant contributions have illuminated my path and set a high standard for my own work.

Finally, my heartfelt gratitude goes to my family—my parents, Shri Sohan Singh and Smt. Jai Dei, and my niece, Shivanshi. Their unwavering support and understanding throughout the demands of my academic pursuits have been a constant source of strength and motivation. Their love and encouragement have been the pillars of my perseverance and inspiration.

# Contents

<b>Declaration</b>	<b>i</b>
<b>Abstract</b>	<b>ii</b>
<b>Acknowledgements</b>	<b>iii</b>
<b>List of Tables</b>	<b>vii</b>
<b>List of Figures</b>	<b>viii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Context and Motivation . . . . .	1
1.2 Problem Statement . . . . .	2
1.2.1 Objectives . . . . .	2
1.3 Outline . . . . .	3
<b>2 Background</b>	<b>4</b>
2.1 Overview of Perception in Autonomous Driving . . . . .	4
2.2 Key Components of Perception . . . . .	5
2.3 CARLA Simulator . . . . .	6
2.4 Tesla Model 3 . . . . .	7
2.5 Traditional Object Detection Techniques . . . . .	10
2.6 Deep Learning-Based Object Detection Methods . . . . .	11
2.6.1 CNN-based Methods . . . . .	11
2.6.2 Transformer-based Methods . . . . .	13
<b>3 Related Work</b>	<b>15</b>
3.1 Selecting YOLO: A Rationale . . . . .	15
3.2 YOLO: You Only Look Once . . . . .	16
3.3 Evolution of YOLO Models . . . . .	17

3.4	Foundation Models in AI	27
<b>4</b>	<b>Methodology</b>	<b>29</b>
4.1	Overview of Methodology	29
4.2	Simulation Setup	31
4.2.1	Simulation Environment Implementation	31
4.2.2	Sensor Suite Configuration	33
4.2.3	Weather and Lighting Conditions	35
4.3	Data Acquisition	35
4.3.1	Data Collection Pipeline	37
4.4	Dataset Creation	39
4.4.1	Data Reorganisation	40
4.4.2	Labeling Process	41
4.5	Model Training	45
4.6	Real-Time Object Detection	47
<b>5</b>	<b>Experimental Setup and Evaluation Metrics</b>	<b>49</b>
5.1	Overview	49
5.2	Simulation Environment Configuration	49
5.3	Dataset Organization and Labeling	50
5.4	EDA and Model Training Environment	50
5.5	Computational Resources	50
5.6	Evaluation Metrics	51
5.6.1	Precision, Recall, and F1 Score	53
5.6.2	Metric Curves	54
5.6.3	Mean Average Precision (mAP)	54
<b>6</b>	<b>Results and Evaluation</b>	<b>57</b>
6.1	Data Acquisition and Preparation	57
6.2	Exploratory Data Analysis (EDA)	58
6.2.1	Dataset Overview	58
6.2.2	Dataset Label Analysis	59
6.2.3	Label Distribution and Null Images	60
6.2.4	Class Co-occurrence Analysis	61
6.2.5	Insights and Implications	62
6.3	Quantitative Results	62
6.3.1	Overview of Training and Testing	65
6.3.2	Training Results	66

---

6.3.3	Validation Results . . . . .	67
6.3.4	Testing Results . . . . .	70
6.3.5	Real-Time Performance . . . . .	71
6.4	Additional Evaluation Results . . . . .	72
6.5	Global Performance Conclusion . . . . .	76
6.5.1	YOLO11m: Overfitting vs Underfitting . . . . .	77
6.6	Qualitative Evaluation of Real-Time Object Detection . . . . .	78
<b>7</b>	<b>Discussion</b>	<b>84</b>
7.1	Challenges and Limitations . . . . .	84
7.2	Implications . . . . .	85
7.3	Summary . . . . .	85
<b>8</b>	<b>Conclusion and Future Work</b>	<b>86</b>
8.1	Conclusion . . . . .	86
8.2	Future Work . . . . .	87
	<b>Bibliography</b>	<b>88</b>
<b>A</b>	<b>Supplementary Material</b>	<b>104</b>
A.1	Qualitative Evaluation: YOLO11m . . . . .	104
A.2	Confusion Matrix in Scenario A . . . . .	104
A.3	Trade-Off Between Detection Accuracy and Inference Speed . . . . .	105

## List of Tables

3.1	Summary of Advancements in YOLO Models . . . . .	28
6.1	Data Collection Summary . . . . .	58
6.2	Dataset Overview . . . . .	58
6.3	Training Results for YOLO Models . . . . .	68
6.4	Validation Results for YOLO Models . . . . .	70
6.5	Test Results for YOLO Models . . . . .	70
6.6	Performance Metrics for YOLO Models . . . . .	73
6.7	Test Results for YOLO Models in Scenario A . . . . .	76
6.8	Test Results for YOLO Models in Scenario B . . . . .	76
6.9	Best Performing Models in Scenario A . . . . .	77
6.10	Best Performing Models in Scenario B . . . . .	77

## List of Figures

2.1	Tesla Model 3 sensor suite . . . . .	7
2.2	Tesla Model 3 perception overview . . . . .	8
2.3	Deep Learning-Based Object Detection Methods . . . . .	12
3.1	YOLO Architecture . . . . .	16
3.2	Modeling detection task as a regression task . . . . .	16
3.3	Workflow of the YOLOv1 object detection system . . . . .	17
3.4	Timeline of YOLO Models . . . . .	18
3.5	Darknet-19 framework . . . . .	19
3.6	Darknet-53 in YOLOv3 . . . . .	20
3.7	Architecture of YOLOv4 . . . . .	21
3.8	Architecture of YOLOv8 . . . . .	24
3.9	Dual assignments in YOLOv10 . . . . .	25
3.10	Key architectural modules in YOLO11 . . . . .	26
4.1	Five stages involved in this project . . . . .	30
4.2	Component Diagram of the System . . . . .	32
4.3	Class Diagram of the CARLA Simulation System . . . . .	32
4.4	Sequence Diagram for Sensor Setup . . . . .	33
4.5	Activity Diagram for Data Acquisition . . . . .	36
4.6	Sequence Diagram for Sensor Data Acquisition . . . . .	37
4.7	Example of reorganized dataset images . . . . .	40
4.8	Two-Stage Dataset Labeling Process . . . . .	41
4.9	Visualization of generated labels . . . . .	43
4.10	Visualization of Generated Labels in Roboflow . . . . .	44
4.11	Conversion of Generated Labels . . . . .	44
4.12	Converted Bounding Box . . . . .	45
4.13	Complete annotated image in Roboflow . . . . .	45

4.14 Model Training . . . . .	46
4.15 Sequence Diagram for YOLO Inference . . . . .	47
4.16 Activity Diagram for YOLO Object Detection Inference . . . . .	48
5.1 IoU Illustration . . . . .	52
6.1 Visualization of camera sensor metadata file. . . . .	58
6.2 Visualization of radar sensor metadata file. . . . .	59
6.3 Visualization of ultrasonic sensor metadata file. . . . .	60
6.4 Class distribution and object center heatmap for the train dataset . . . . .	61
6.5 Label Correlogram for the train dataset . . . . .	62
6.6 Visual representation of label distributions . . . . .	63
6.7 Class co-occurrence matrix plots . . . . .	64
6.8 Visualization of Training and Validation curves for YOLO11m model . . . . .	68
6.9 Metric curves for the YOLO11m model on validation set . . . . .	69
6.10 Confusion Matrix for YOLOv8m on Test set . . . . .	72
6.11 Real-Time Performance: Latency vs. mAP(50-95) Analysis . . . . .	74
6.12 Metric curves for the YOLO11m model on test set in Scenario A . . . . .	75
6.13 Box Loss plot for YOLO11m . . . . .	78
6.14 Classification Loss plot for YOLO11m . . . . .	79
6.15 Distribution Focal Loss plot for YOLO11m . . . . .	80
6.16 Real-time object detection in Town07 . . . . .	81
6.17 Real-time object detection in Town05 . . . . .	81
6.18 Real-time object detection in Town03 . . . . .	82
6.19 Real-time object detection limitations . . . . .	82
6.20 Generalization limitations of YOLO11m . . . . .	83
A.1 Training batch 0 visualization during training . . . . .	105
A.2 Ground truth labels for validation batch 0 . . . . .	106
A.3 Model predictions for validation batch 0 . . . . .	107
A.4 Ground truth labels for test dataset batch 1 . . . . .	108
A.5 Model predictions for test dataset batch 1 . . . . .	109
A.6 Confusion Matrix for YOLO11m on Test set . . . . .	110
A.7 Trade-Off Between Detection Accuracy and Inference Speed in Scenario A . . . . .	110
A.8 Trade-Off Between Detection Accuracy and Inference Speed in Scenario B . . . . .	111

## Acronyms

**AI** Artificial Intelligence

**AMP** Automatic Mixed Precision

**YOLO** You Only Look Once

**CARLA** Car Learning to Act

**IoU** Intersection over Union

**CIoU** Complete Intersection over Union

**CSP** Cross-Stage-Partial-connections

**CBN** Cross-Iteration Batch Normalization

**SAM** Spatial Attention Module

**SLAM** Simultaneous Localization and Mapping

**FPN** Feature Pyramid Network

**SPP** Spatial Pyramid Pooling

**PANet** Path Aggregation Network

**EDA** Exploratory Data Analysis

# Chapter 1

## Introduction

Recent technological advancements have enabled the development of reliable self-driving cars, with sensors playing a crucial role in environmental perception. This research focuses on synthetic data collection for sensor-based perception and simulation-driven real-time object detection in autonomous driving applications.

Section 1.1 presents the necessary background information and highlights the significance of object detection in autonomous driving. Section 1.2 defines the problem statement and outlines the objectives of this research project. Lastly, Section 1.3 provides a detailed overview of the project's structure.

### 1.1 Context and Motivation

The advancement of autonomous driving systems hinges on the capability to detect objects accurately and efficiently in real time. Object detection, a fundamental task in computer vision, involves identifying and localizing objects in images or video frames [1, 2]. This process is critical for enabling autonomous vehicles to perceive their environment, make informed decisions, and navigate safely. Applications of object detection extend beyond autonomous driving to domains such as surveillance and robotics [1, 2, 3]. In the context of autonomous driving, it underpins the ability to identify other vehicles, pedestrians, traffic signals, and road signs, which is essential for avoiding collisions and responding to dynamic road conditions. The precision and speed of object detection algorithms are directly tied to the safety and reliability of autonomous systems [1, 2].

A major challenge in developing autonomous driving systems is the collection of annotated real-world datasets required for training and testing deep learning models for object detection. Acquiring these datasets is expensive and time-intensive [4]. Autonomous driving simulators, such as [CARLA](#), offer a scalable and cost-effective alternative by generating synthetic datasets [5]. These simulators enable the creation of diverse datasets with detailed annotations, covering a range of scenarios, including rare and hazardous situations. Additionally, simulation environments mitigate the risks associated with real-world testing, providing researchers with controlled conditions for experimentation [5, 6].

For example, the SHIFT dataset leverages the [CARLA](#) simulator to capture dynamic environments, producing over 4,800 multi-view sensor sequences with comprehensive annotations for 2.5 million images [7]. Similarly, the CARLA-Loc Dataset integrates multiple sensors with precise calibration and synchronization, providing valuable data for evaluating Simultaneous Localization and Mapping ([SLAM](#)) algorithms [8]. These datasets underscore the utility of [CARLA](#) in advancing autonomous driving technologies by enabling synthetic data generation and serving as a robust testbed for evaluating autonomous driving systems.

## 1.2 Problem Statement

Developing robust autonomous driving systems depends on extensive and diverse datasets to train and evaluate object detection algorithms. However, real-world data collection is expensive, time-consuming, and limited in its ability to capture rare or hazardous scenarios. Additionally, testing underdeveloped systems in real-world environments can pose significant safety risks [6]. Simulation platforms, such as [CARLA](#), address these challenges by providing a controlled environment for generating synthetic datasets and conducting experiments [9]. Despite these advantages, there is limited research on evaluating state-of-the-art YOLO object detection models using synthetic data generated from complex simulation setups [10]. This gap necessitates a systematic approach to assess the performance, generalizability, and robustness of such models on synthetic data.

### 1.2.1 Objectives

This research project aims to achieve the following objectives:

- **Simulation Setup:** Develop a detailed simulation of a Tesla Model 3 equipped with 8 cameras, 12 ultrasonic sensors, and 1 radar. Establish a pipeline for collecting and storing sensor data from this configuration.
- **Dataset Creation:** Generate a synthetic image dataset from the recorded camera data captured by the Tesla Model 3's camera sensors.
- **YOLO Training:** Train [YOLO](#) object detection models YOLOv8, YOLOv9, YOLOv10, and YOLO11 on the custom synthetic dataset and compare their performance.
- **Real-Time Integration:** Integrate one of the best-performing [YOLO](#) models based on highest localization precision and classification accuracy into the [CARLA](#) simulator to enable real-time object detection within the simulation environment.

### 1.3 Outline

The structure of the rest of the Studienarbeit is as follows:

- **Chapter 2: Background:** This chapter introduces perception systems in autonomous driving, with a particular focus on the CARLA Simulator and its customization capabilities, including the sensor parameters used in simulating a Tesla Model 3. It also reviews the evolution of object detection methods, from traditional approaches to modern deep learning-based techniques, providing the necessary foundation for understanding the project.
- **Chapter 3: Related Work:** This chapter examines the development of the YOLO series, tracing its evolution from YOLOv1 to YOLO11. Additionally, it explores foundational models like Grounded-SAM and their applications in automated labeling, offering insights into state-of-the-art object detection techniques.
- **Chapter 4: Methodology:** This chapter details the methodology for simulating the Tesla Model 3 in CARLA, including sensor data collection, the creation of a customized image dataset, and the execution of real-time inference using a trained YOLO model. Unified Modeling Language (UML) diagrams are used to visually represent the workflow and interactions within the simulation framework, enhancing clarity and understanding.
- **Chapter 5: Experimental Setup and Evaluation Metrics:** This chapter provides a detailed overview of the technical setup, including the software, computational resources, and configurations required for replicating the experiments. It also introduces the evaluation metrics used to assess the YOLO models' performance, ensuring a clear and reproducible methodology.
- **Chapter 6: Results and Evaluation:** This chapter presents the empirical findings, starting with an Exploratory Data Analysis (EDA) of the dataset to highlight its structure and challenges. It then discusses the evaluation metrics, hyperparameter configurations, and experimental outcomes, combining quantitative results with qualitative visualizations to assess model performance.
- **Chapter 7: Discussion:** This chapter critically examines the challenges, limitations, and broader implications of the research. It situates the experimental results within the context of autonomous driving and object detection, providing actionable insights for addressing limitations and advancing the field.
- **Chapter 8: Conclusion and Future Work:** This final chapter summarizes the key findings, reflecting on their alignment with the project's objectives. It also identifies potential directions for future research, paving the way for continued progress in autonomous driving technologies.

## Chapter 2

### Background

Building on the previous chapter, which outlined the objectives of this project, this chapter begins with an overview to the perception systems in autonomous driving, the CARLA Simulator and the Tesla Model 3 autonomous vehicle. The discussion then shifts to the evolution of object detection methods, tracing the progression from traditional methods to modern deep learning-based approaches.

#### 2.1 Overview of Perception in Autonomous Driving

Recent advances in machine learning and deep learning have significantly improved the performance of image classifiers, with Convolutional Neural Networks (CNNs) being the most widely used classifiers and Vision Transformers (ViTs) representing the current state-of-the-art [11, 12, 13, 14, 15]. The success of these methods relies heavily on high-quality annotated datasets, which are essential for supervised learning algorithms [16]. In the context of self driving cars, we can categorise perception systems into two high-level categories: internal and external perception systems.

**Internal Perception Systems:** Internal perception involves developing computer vision-based detection systems for driver monitoring or driver assistance, facilitating human-machine interaction. This includes systems that track driver attentiveness, monitor fatigue levels, detects distracted drivers and provide real-time alerts or assistance to ensure driver safety through safe driving [17, 11, 16, 18, 19]. Advances in deep learning models, such as CNNs and ViTs, have enabled more accurate and efficient detection of driver behaviors and conditions, crucial for enhancing human-machine interaction in semi-autonomous vehicles [20, 19, 21].

**External Perception Systems:** External perception pertains to the vehicle’s awareness of its surroundings. Autonomous vehicles (AVs) equipped with sensors such as LiDAR, cameras, radar, and ultrasonic sensors must detect and interpret objects in their environment with the help of advance algorithms to ensure

safe driving <sup>1</sup>. These sensors provide comprehensive data for object detection, object localization, Simultaneous Localization and Mapping (SLAM), and sensor fusion, enabling the vehicle to navigate complex environments safely [22, 23].

## 2.2 Key Components of Perception

Following are the key components in various perception systems:

- **Camera:** Camera-based perception systems capture visual data crucial for tasks such as object detection, lane keeping, and traffic sign recognition. These systems provide essential visual context and enable the vehicle to make informed decisions based on road conditions and traffic signs. [24, 25, 26, 27, 28] are some of the examples of camera based object detection methods which falls in external perception category.
- **LiDAR (Light Detection and Ranging):** LiDAR sensors emit laser pulses to create detailed 3D maps of the vehicle's surroundings. These sensors are highly accurate in measuring distances and detecting objects, making them indispensable for precise localization and obstacle detection. Recent advancements in deep learning have significantly enhanced 3D object detection capabilities using LiDAR. Techniques such as 3D object detection have shown promising results in improving the vehicle's environmental understanding. With ongoing developments, methods like PointRCNN, PointPillars, and others have demonstrated substantial progress in leveraging LiDAR data for autonomous driving [29, 30, 31, 32, 33, 34, 35].
- **Radar:** Radar systems excel in detecting objects and measuring their speed, especially under adverse weather conditions where cameras and LiDAR may struggle. They are highly effective for long-range detection, making them essential for adaptive cruise control and collision avoidance systems. Recent research has validated the effectiveness of radar in 3D object detection, with various studies demonstrating its capability in enhancing situational awareness and safety in autonomous driving [36, 37, 38, 39, 40, 41].
- **Ultrasonic Sensors:** Ultrasonic sensors are primarily utilized for near-field perception in autonomous driving, particularly for short-range detection tasks such as parking assistance. These sensors are essential for detecting nearby objects and providing precise distance measurements in close proximity to the vehicle. They are a low-cost, durable, and robust technology suitable for near-range detection even in harsh weather conditions. Despite their advantages, ultrasonic sensors have received limited attention in the perception literature [42, 43]. Recent advancements include the first end-to-end multi-

---

<sup>1</sup><https://solution1.com.tw/the-importance-of-environment-perception-technology-in-autonomous-driving/>  
- Accessed on January 20, 2025. This article highlights the critical role of environmental perception technology in enabling safe and efficient autonomous driving.

modal fusion model designed for efficient obstacle perception in a bird's-eye-view (BEV) perspective, integrating fisheye cameras and ultrasonic sensors [44].

- **Sensor Fusion:** Sensor fusion combines data from multiple sensors (e.g., LiDAR, radar, cameras) to create a more comprehensive understanding of the environment. This approach leverages the strengths of each sensor type to improve detection accuracy and robustness. For instance, LiDAR provides precise distance measurements, radar offers reliable velocity data, and cameras deliver rich contextual information. By integrating these diverse data sources, fusion techniques enhance the overall perception capabilities of autonomous vehicles. This integration and its benefits are supported by several research studies [45, 46, 47, 48, 49].
- **V2X (Vehicle-to-Everything) Communication:** Vehicle-to-Everything (V2X) communication encompasses Vehicle-to-Vehicle (V2V) and Vehicle-to-Infrastructure (V2I) communications, allowing vehicles to share information with each other and with roadside infrastructure. This collaborative perception helps expand the sensory range and improve situational awareness, thus enhancing safety and efficiency in autonomous driving. The advantages and implementations of V2X communication have been explored in numerous research articles [22, 50].

### 2.3 CARLA Simulator

CARLA [5] is an open-source simulator designed to advance autonomous driving research by providing a realistic yet controlled environment for developing, training, and validating autonomous driving systems. It excels at generating synthetic datasets with precise annotations for objects such as traffic lights, signs, vehicles, and pedestrians, significantly reducing the time and costs associated with manual data collection and annotation.

CARLA uses the Unreal Engine to simulate different environmental conditions, including adverse weather, which is critical for testing object detection algorithms under various scenarios [5]. It supports realistic sensor simulations for cameras, LiDAR, radar, and Vehicle-to-everything (V2X) communication, making it an ideal platform for developing and validating perception algorithms before real-world deployment [51]. The simulator allows for the creation of large and diverse datasets, including rare and hazardous scenarios like pedestrians with unusual body poses [52]. Such scenarios are essential for evaluating the robustness of Simultaneous Localization and Mapping (SLAM) algorithms under dynamic conditions [53, 8].

Additionally, tools like CarFree enable the automatic generation of large-scale datasets with precise annotations, further reducing the manual effort and cost involved in data collection [54]. These features make CARLA a crucial resource for advancing object detection technologies in autonomous driving, accelerating the development cycle, and ensuring that object detection models are robust, accurate, and ready for deployment in complex urban environments [5, 54].

## 2.4 Tesla Model 3

The Tesla Model 3 features an advanced sensor suite that includes 21 sensors: 8 RGB cameras, 1 radar sensor, and 12 ultrasonic sensors as shown in figure 2.1. These sensors are strategically configured using the available settings in CARLA on a simulated Tesla Model 3 to enable comprehensive environmental perception, supporting accurate object detection and interaction under autonomous driving conditions [5]. Since the precise configuration and setup details of the Tesla Model 3 sensor suite are not publicly disclosed, this project relies on informed assumptions and available data to replicate the sensor setup as closely as possible.

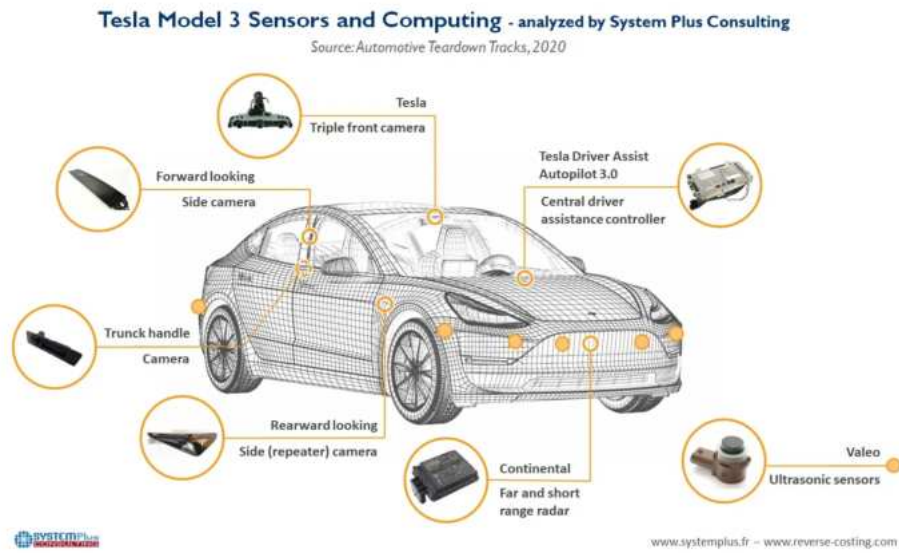


Figure 2.1: Tesla Model 3 equipped with various sensors <sup>2</sup>.

### Key Sensor Parameters

This subsection discusses the key parameters for each sensor type integrated into the Tesla Model 3 within the CARLA Simulator. It provides detailed descriptions of the configurable parameters available in CARLA, highlighting their impact on the scope and accuracy of the data collection process [5, 56].

**Camera Parameters:** Cameras are configured with the following parameters:

- **Field of View (FOV):** Represents the angular extent of the observable environment captured by the camera. A narrow FOV (e.g., 35°) focuses on distant objects, enabling detailed long-range observa-

<sup>2</sup><https://www.eetimes.com/a-tesla-model-3-tear-down-after-a-hardware-retrofit/> - Accessed on January 20, 2025. This article provides an image of Tesla Model 3 equipped with a sensor suite generated by System Plus Consulting.

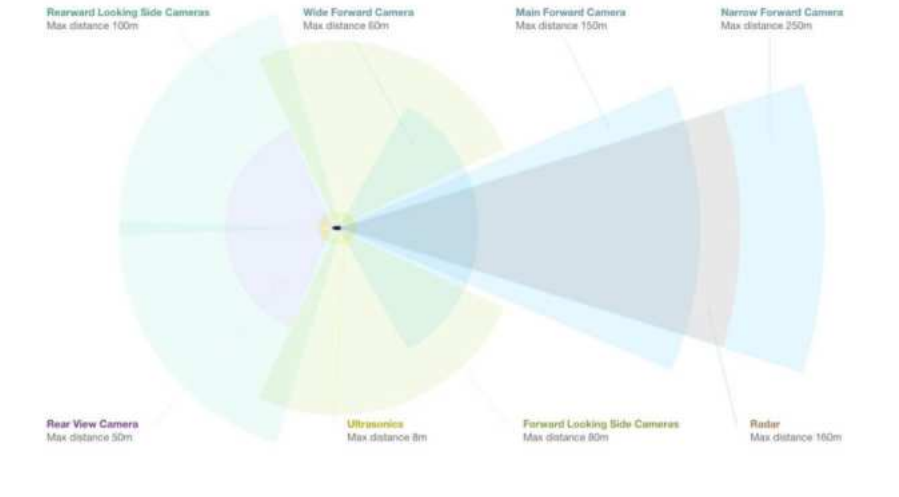


Figure 2.2: Tesla Model 3 sensors range and perception overview [55].

tions, whereas a wide FOV (e.g.,  $120^\circ$ ) captures more peripheral information, crucial for detecting nearby objects or vehicles [56].

- **Position:** Defines the camera's location relative to the vehicle's coordinate frame  $(x, y, z)$ . Strategic placement ensures unobstructed views, minimizes occlusions, and maximizes environmental coverage [56].
- **Rotation:** Specifies the camera's orientation using pitch (tilt), yaw (horizontal angle), and roll (rotation about the axis). These parameters determine the camera's focus direction and coverage angle [56].
- **Resolution:** Refers to the dimensions of the captured image, measured in pixels (e.g.,  $1280 \times 960$ ). Higher resolution ensures finer detail, essential for object detection, though it demands greater computational resources [56].
- **Sensor Tick:** Defines the interval (in seconds) at which the camera captures frames. For this simulation, a tick rate of 0.033 seconds (corresponding to 30 FPS) ensures smooth and temporally consistent image sequences [56].

**Radar Parameters:** Radar sensors provide critical distance and velocity measurements, particularly for long-range object detection. Their parameters include:

- **Horizontal FOV:** Specifies the angular detection range across the horizontal plane (e.g.,  $35^\circ$ ). This parameter determines the radar's lateral scanning capability [56].

- **Vertical FOV:** Represents the angular detection range in the vertical plane (e.g.,  $5^\circ$ ). A narrower vertical FOV ensures focused sensing in a specific elevation range [56].
- **Range:** Defines the maximum distance (in meters) within which objects can be detected (e.g., 160 meters). This parameter is vital for highway scenarios and long-range obstacle detection [56].
- **Sensor Tick:** The time interval (in seconds) between consecutive data captures (e.g., 0.033 seconds). This ensures temporal consistency in object tracking [56].
- **Points Per Second:** Represents the number of data points the radar can generate per second (e.g., 1500 points/sec). This defines the density and precision of the radar's output data [56].
- **Position:** Specifies the mounting location of the radar sensor relative to the vehicle's coordinate frame  $(x, y, z)$ . Proper positioning ensures unobstructed forward detection [56].
- **Rotation:** Describes the sensor's orientation in pitch, yaw, and roll, enabling precise alignment with the desired detection area [56].

**Ultrasonic Sensor Parameters:** Ultrasonic sensors are designed for short-range obstacle detection, especially useful during low-speed maneuvers. Their parameters are as follows:

- **Horizontal FOV:** Defines the angular sensing range in the horizontal plane (e.g.,  $60^\circ$ ). This broad coverage is ideal for detecting nearby objects such as curbs or parked vehicles [56].
- **Vertical FOV:** Specifies the angular sensing range in the vertical plane (e.g.,  $5^\circ$ ). This limited vertical coverage is optimized for detecting obstacles near ground level [56].
- **Range:** Refers to the maximum sensing distance of the sensor (e.g., 5 meters). Ultrasonic sensors are calibrated for close-proximity applications such as parking assistance [56].
- **Sensor Tick:** Defines the interval between consecutive data captures (e.g., 0.033 seconds). This ensures real-time responsiveness during maneuvering [56].
- **Position:** Describes the location of the sensor relative to the vehicle's coordinate frame  $(x, y, z)$ . Ultrasonic sensors are symmetrically placed along the vehicle's front and rear to provide complete coverage [56].
- **Rotation:** Indicates the orientation of the sensor using pitch, yaw, and roll angles. For example, a forward-facing ultrasonic sensor may have a slight downward pitch to focus on low-lying obstacles [56].

## 2.5 Traditional Object Detection Techniques

The journey of object detection in computer vision began in the 1960s [57], marked by the pioneering work of Roberts who emphasized the need for matching two-dimensional image features to three-dimensional object representations [58]. This era laid the foundational work for what would later evolve into more complex systems. By the early 1970s, the field had seen significant advancements, such as the development of fully automated assembly machines, which utilized image processing for tasks like semiconductor device assembly [59]. The term “machine vision” originated in Japan at Hitachi Labs [57]. This concept was defined with a focus on practical applications, laying the groundwork for its pivotal role in industrial automation [60]. In traditional computer vision, object detection encompasses several tasks: starting with detection (identifying the presence of an item), followed by localization (determining the precise location of the item), recognition (identifying and locating all items in an image), and finally, understanding (recognizing items and their roles within a context) [61, 62]. These tasks build upon each other to form a comprehensive vision system capable of interpreting complex scenes [57].

**Viola-Jones Detector:** Introduced in 2001 by Paul Viola and Michael Jones [63], the Viola-Jones detector was a pivotal development in real-time human face detection. This method, which leverages integral images, Haar-like features, and a cascading classifier system, significantly advanced the efficiency and accuracy of object detection. However, it has limitations such as struggling with non-frontal faces or varying lighting conditions. The integral image component is crucial as it allows for the quick calculation of features across the image, enabling constant-time computation. The AdaBoost machine learning technique enhances detection accuracy by focusing on predictive features, although it may still miss complex facial features under challenging conditions. Moreover, the sliding window approach employed in the detection process might not efficiently detect faces under rapid motion or varied poses [63, 64, 65].

**Histogram of Oriented Gradients (HOG):** Developed in 2005 by Navneet Dalal and Bill Triggs [66], the HOG descriptor enhances object detection, particularly for pedestrian detection, by providing a detailed representation of object shapes. The process starts with calculating the image gradients to highlight edges and contours, but it may struggle in cluttered backgrounds. Spatial and orientation binning captures dominant gradient orientations but can falter with non-rigid or oddly shaped objects. The block normalization step ensures consistency of features across different lighting conditions, although it may not be effective under extreme variations. The descriptor formation creates a robust feature descriptor that encapsulates essential shape and edge information but lacks the flexibility needed for complex object detection scenarios [66, 67].

**Deformable Part-based Model (DPM):** Proposed in 2008 by Pedro Felzenszwalb and colleagues [68], DPM offers flexible object detection but is computationally demanding. It includes a root filter and multiple part filters, allowing for robust detection but at the cost of increased computational complexity. The training

process is extensive and can be a barrier for real-time applications. The detection phase, which utilizes a feature pyramid and dynamic programming, achieves high accuracy but may not perform well in resource-constrained environments. DPM is adept at managing occlusions and variations in object pose, making it versatile yet slower compared to newer, more efficient algorithms [68, 69, 67].

**Comparative Analysis:** These traditional methods each contributed uniquely to the field. Viola-Jones was revolutionary for real-time applications but limited by rigid detection scenarios. HOG provided robust feature descriptions ideal for pedestrian detection but was less effective in complex, dynamic environments. DPM excelled in accuracy and adaptability but at the expense of computational efficiency. The limitations of these methods spurred the development of deep learning techniques, which address many of their constraints through enhanced learning capabilities and greater adaptability [57, 69, 64, 65].

## 2.6 Deep Learning-Based Object Detection Methods

Deep learning has revolutionized the field of object detection by introducing sophisticated hierarchical architectures that surpass traditional feature-engineering approaches. As detailed in figure 2.3, these techniques are categorized into CNN-based [70] and transformer-based [71] approaches. They automate the extraction of semantic features, significantly improving detection capabilities in various applications [72].

### 2.6.1 CNN-based Methods

CNN-based object detection is primarily categorized into two principal types: two-stage object detection and one-stage object detection. The two-stage models primarily utilize regions of interest (RoI) to build candidate bounding boxes, subsequently extracting features from these boxes to identify objects. Consequently, all models inside the Region Proposal Based Framework are classified as two-stage detectors. In comparison, the one-stage methods do not use RoI mechanism to locate objects, both steps are combined in just one stage, regression and classification based models like YOLO, SSD, RetinaNet as well as end to end detection based models like DETR model series falls under one stage detection category [72].

**Region Proposal Based Framework:** The Region-based Convolutional Neural Network (R-CNN) [73] is an innovative technique in object detection that integrates region proposal methods with deep learning to achieve accurate object localization and classification. It operates by first generating potential object regions using a method like Selective Search [74], then classifying these regions using a convolutional neural network (CNN) that has been fine-tuned for the task. R-CNN Object Detection system takes an input image, extracts bottom-up region proposals, computes features for each proposal using a large convolutional neural network (CNN), and then classifies each region using class-specific linear SVMs [75, 76]. R-CNN Object Detection Process can be divided into following four parts:

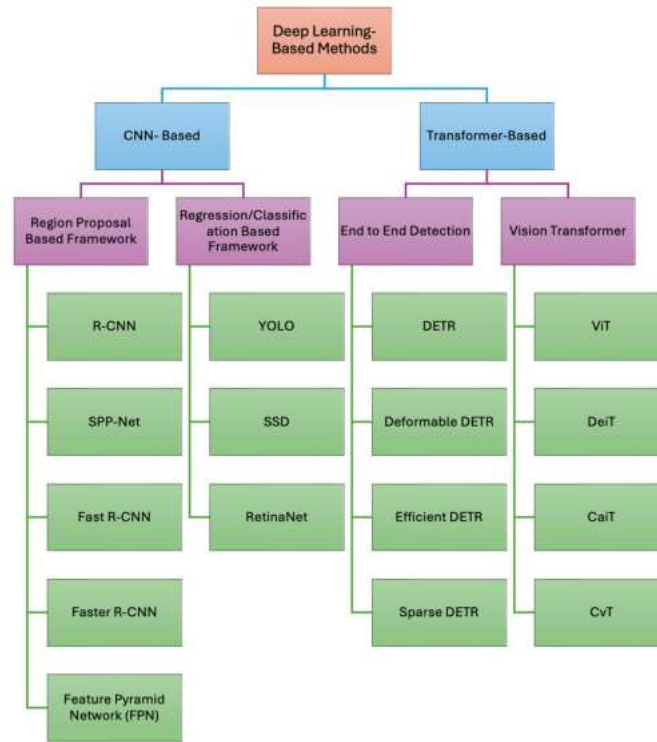


Figure 2.3: Hierarchical classification of deep learning-based object detection methods as adapted from [72]. This figure showcases the division into CNN-based and transformer-based methodologies, illustrating their distinct frameworks and evolutionary trajectories in object detection technology.

- **Region Proposal:** R-CNN begins by using Selective Search to generate a set of region proposals, which are potential areas in the image that may contain objects. This step is crucial for narrowing down the areas that need to be analyzed further [73, 77, 72].
- **Feature Extraction and Classification:** Each proposed region is then passed through a CNN that has been pre-trained on a large dataset. The CNN extracts features from these regions and classifies them into object categories [73, 77, 72].
- **Bounding Box Regression:** To improve localization accuracy, R-CNN applies bounding box regression to refine the coordinates of the detected objects [73, 77, 72].
- **Non-Maximum Suppression:** Finally, non-maximum suppression is used to eliminate overlapping bounding boxes, ensuring that only the most accurate detections are retained [73, 77, 72].

The R-CNN family represents a significant evolution in object detection, beginning with the original R-CNN model, which introduced the concept of using region proposals for object detection. This was further

enhanced by SPP-Net [78], which added a spatial pyramid pooling layer to handle inputs of varying sizes more efficiently. Fast R-CNN [79] improved upon this by integrating the ROI (regions of interest) pooling layer, which significantly sped up the process by sharing computations across the entire image, eliminating the need for repetitive feature extraction for each region proposal. Faster R-CNN [80] took this a step further by introducing the Region Proposal Network (RPN), which made the generation of region proposals nearly cost-free by sharing features with the detection network. Lastly, the Feature Pyramid Network (FPN) [81] built on these foundations by creating a multi-scale feature pyramid that improves performance across different object scales, effectively handling the detection of objects at varying resolutions by maintaining a balance between resolution and semantic strength at different levels of the network. Together, these models form a robust framework for addressing a wide range of challenges in object detection.

**Regression/Classification Based Framework:** The Regression/Classification Based Framework for object detection simplifies the detection process by eliminating the need for explicit region proposal generation, enabling faster and more efficient detection models [72]. YOLO (You Only Look Once) [82] is a groundbreaking model that processes the entire image at once to predict bounding boxes and class probabilities, significantly enhancing detection speed and making it suitable for real-time applications. SSD (Single Shot Detector) [83] advances this approach by utilizing multiple feature maps at different resolutions to improve detection accuracy across a variety of object sizes, integrating a set of predefined anchor boxes for various aspect ratios and scales. RetinaNet [84] addresses the challenge of class imbalance with the introduction of focal loss, which focuses training on hard-to-classify examples to improve overall accuracy. Collectively, these models represent significant advancements in the ability to detect objects directly from full images, efficiently and accurately, without the complex multi-stage pipeline characteristic of earlier models.

### 2.6.2 Transformer-based Methods

The advancement of end-to-end detection frameworks using transformer [71] architecture has revolutionized object detection by simplifying the detection process and enhancing the models' capabilities [72]. A seminal example, DETR (Detection Transformer) [85], treats object detection as a direct set prediction problem, removing the need for complex procedures such as non-maximum suppression and anchor boxes, while employing a straightforward encoder-decoder structure. Building on DETR's foundations, Deformable DETR [86] introduces deformable attention mechanisms that focus on a small set of key sampling points to improve model efficiency and performance on small objects. This modification not only enables faster convergence—up to ten times quicker than its predecessor—but also enhances overall detection performance. However, it introduces new challenges such as a significant increase in the number of encoder tokens and a heightened computational load due to the encoder attention mechanisms [87]. To address these limitations, further innovations have been proposed. For instance, Sparse DETR optimizes computational efficiency by selectively updating only those tokens that are likely to be referenced by the decoder, thus significantly reducing the computational burden while maintaining robust detection capabilities [87]. Moreover, Efficient

DETR [88] innovatively leverages a dense prior to initialize object containers, thereby narrowing the performance gap associated with different numbers of decoder layers. This series of advancements underscores the transformative impact of transformer-based methods in the realm of object detection.

The Vision Transformer (ViT) [14] further extends transformers to vision tasks by processing images as sequences of fixed-sized patches, enabling global context capture without convolutional layers. DeiT (Data-efficient Image Transformer) [89] optimizes ViT for practical applications by using techniques like knowledge distillation [90] to enhance training efficiency with less data [72]. CaiT (Class-attention in Image Transformers) [91] refines this approach by focusing more on class tokens through specialized class-attention layers, enhancing model accuracy and training stability. Lastly, CvT (Convolutional vision Transformer) [92] integrates convolutions into the transformer architecture, balancing the extraction of local and global features to better accommodate diverse visual tasks. Together, these models underscore the transformative impact of integrating transformers into object detection, setting new standards for efficiency and accuracy.

## Chapter 3

### Related Work

Building on the previous chapter, which covered the classification of perception systems essential to autonomous driving, introduced the CARLA Simulator and the Tesla Model 3 autonomous vehicle, and discussed the evolution of object detection methods, this chapter will delve deeper into the evolutionary trajectory of the **YOLO** series, tracing its development from YOLOv1 through to YOLO11. Additionally, this chapter provides details about the foundation models such as GroundedSAM and their applications in automated labeling.

#### 3.1 Selecting YOLO: A Rationale

Variations in image resolutions and aspect ratios pose substantial challenges in object detection, particularly when the target objects vary significantly in size. This issue is compounded by class imbalance, where some classes are underrepresented in the training dataset, leading to biased model predictions. These challenges are documented in the literature [93]. Furthermore, the computational demands of object detection architectures—which require significant power, memory, and processing time—are considerable, posing a challenge for their deployment in resource-constrained environments [94, 95]. While two-stage detectors are known for their high accuracy, their computational intensity often renders them impractical for real-time applications.

In contrast, single-stage detectors, which streamline the detection process by eliminating the need for a separate region proposal generation, provide a more efficient alternative. These models are characterized by their faster processing times and reduced computational overhead, making them ideal for use in environments where resources are limited. Among single-stage detectors, **YOLO** stands out due to its simplicity and efficiency, offering robust accuracy and the capability for real-time performance, which are crucial for the deployment in dynamic environments like autonomous driving [96].

### 3.2 YOLO: You Only Look Once

The YOLO stands for You Only Look Once. The YOLO model offered a transformative approach to object detection by conceptualizing it as a regression problem. It employed a highly efficient architecture comprising 24 convolutional layers followed by 2 fully connected layers as shown in the figure 3.1. The convolutional layers, initially pretrained on the ImageNet [97] classification task at a resolution of  $224 \times 224$ , are then adapted to double the resolution for more effective detection. The YOLO architecture seg-

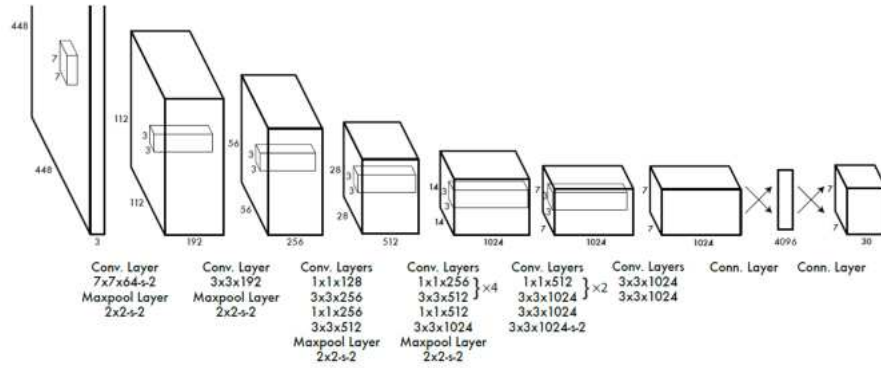


Figure 3.1: YOLO Architecture. This figure depicts the neural network architecture of the YOLO model. Adapted from [82]

ments the image into an  $S \times S$  grid, where each grid cell predicts  $B$  bounding boxes and their confidence levels, along with  $C$  class probabilities. This prediction format encapsulates the bounding boxes' dimensions  $(x, y, w, h)$  and the confidence score, which estimates the Intersection Over Union (IOU) between the predicted box and the ground truth. Additionally, each grid cell calculates  $C$  conditional class probabilities, factoring in only the presence of an object as depicted in the figure 3.2.

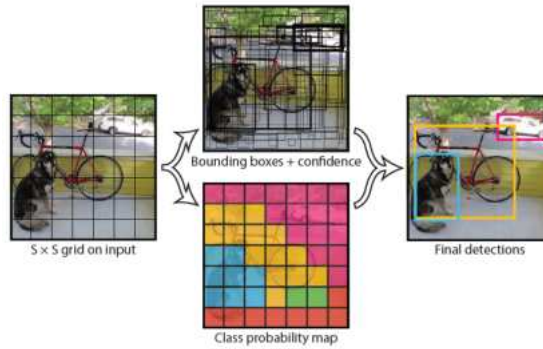


Figure 3.2: Modeling detection task as a regression task. Adapted from [82]

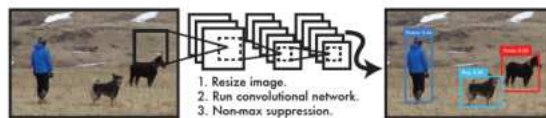


Figure 3.3: Workflow of the **YOLOv1** object detection system. Initially, the input image is resized to a resolution of  $448 \times 448$  pixels. Subsequently, the image is processed through a single convolutional neural network to identify potential objects. The final stage involves applying a threshold to the detected objects based on the model's confidence scores to ascertain the final detections [82].

**Capabilities and Limitations:** Despite its advanced capabilities and rapid processing speed—45 frames per second for the standard model and 155 frames per second for the lighter version named Fast YOLO—the YOLO framework suffered from notable limitations [82]. The YOLO framework imposes strict spatial constraints as each grid cell can only predict two boxes and assign one class label per box. This limitation hampers its ability to detect multiple nearby objects, particularly small ones that appear in clusters, like flocks of birds. The model's reliance on data to predict bounding boxes also restricts its effectiveness with objects that exhibit unusual aspect ratios or configurations, as it predominantly learns from common object appearances and may use coarser features due to multiple downsampling layers. Moreover, the model's loss function, designed to approximate overall detection performance, does not differentiate between errors in large versus small bounding boxes, leading to disproportionately high impacts from minor inaccuracies in smaller boxes. The primary source of errors in YOLO typically stems from these incorrect localizations [82].

### 3.3 Evolution of YOLO Models

The YOLO (You Only Look Once) series, pioneered by Redmon et al. [82], revolutionized object detection by introducing a single-stage detector capable of processing images in real time with significant accuracy. Figure 3.4 illustrates the chronological development of various YOLO models, tracing their evolution from the initial launch in 2015 through to the latest iteration in 2024. This visual representation provides a clear overview of the progressive enhancements and expansions in the YOLO architecture over nearly a decade.

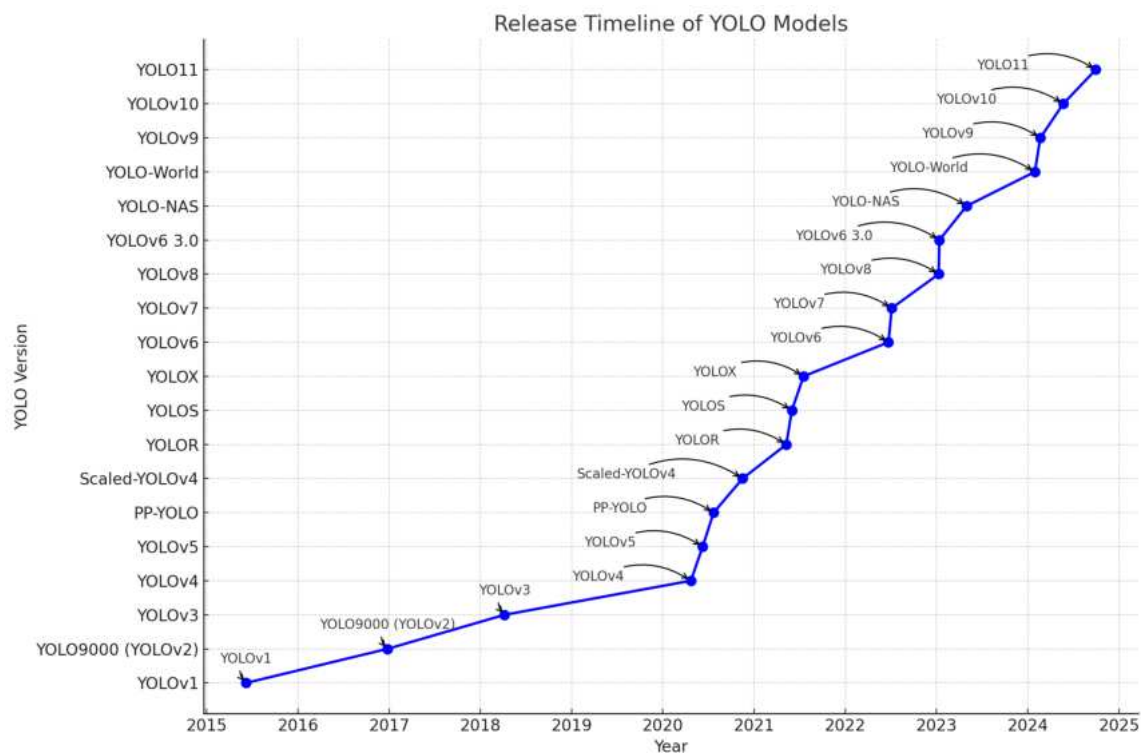


Figure 3.4: Timeline of YOLO Models. This figure is inspired from [98].

**YOLOv2:** Introduced in 2016 by Redmon and Farhadi [99], the **YOLO9000** paper (also known as YOLOv2) implemented several crucial improvements to the YOLOv1 framework to enhance recall, localization accuracy, and classification efficacy. Incorporating batch normalization across all convolutional layers, YOLOv2 saw a significant boost in model convergence and performance, leading to a 2% increase in mean Average Precision (mAP) and obviating the need for dropout. Pre-training the network at a higher resolution of 448x448 allows it to better adapt to high-resolution inputs, improving mAP by nearly 4% [99].

Type	Filters	Size/Stride	Output
Convolutional	32	$3 \times 3$	$224 \times 224$
Maxpool		$2 \times 2/2$	$112 \times 112$
Convolutional	64	$3 \times 3$	$112 \times 112$
Maxpool		$2 \times 2/2$	$56 \times 56$
Convolutional	128	$3 \times 3$	$56 \times 56$
Convolutional	64	$1 \times 1$	$56 \times 56$
Convolutional	128	$3 \times 3$	$56 \times 56$
Maxpool		$2 \times 2/2$	$28 \times 28$
Convolutional	256	$3 \times 3$	$28 \times 28$
Convolutional	128	$1 \times 1$	$28 \times 28$
Convolutional	256	$3 \times 3$	$28 \times 28$
Maxpool		$2 \times 2/2$	$14 \times 14$
Convolutional	512	$3 \times 3$	$14 \times 14$
Convolutional	256	$1 \times 1$	$14 \times 14$
Convolutional	512	$3 \times 3$	$14 \times 14$
Convolutional	256	$1 \times 1$	$14 \times 14$
Convolutional	512	$3 \times 3$	$14 \times 14$
Maxpool		$2 \times 2/2$	$7 \times 7$
Convolutional	1024	$3 \times 3$	$7 \times 7$
Convolutional	512	$1 \times 1$	$7 \times 7$
Convolutional	1024	$3 \times 3$	$7 \times 7$
Convolutional	512	$1 \times 1$	$7 \times 7$
Convolutional	1024	$3 \times 3$	$7 \times 7$
Convolutional	1000	$1 \times 1$	$7 \times 7$
Avgpool		Global	1000
Softmax			

Figure 3.5: The Darknet-19 architecture used in YOLOv2. It features 19 convolutional layers and 5 max-pooling layers, optimizing for efficient feature extraction and down-sampling. This streamlined design incorporates  $1 \times 1$  convolutions to enhance feature integration, significantly improving detection capabilities. [99].

Further enhancements include the use of convolutional anchor boxes, which refine the model's resolution and accuracy in localization, and dimension clusters that optimize anchor shapes through k-means clustering based on Intersection Over Union (IOU) scores. The introduction of direct location prediction enhances training stability by constraining bounding box coordinates to a  $[0,1]$  range, resulting in a 5% increase in mAP. Additionally, a passthrough layer enhances detection sensitivity for smaller objects, contributing a marginal performance increase of 1%. Finally, multi-scale training dynamically adjusts input sizes, boosting the model's adaptability to various resolutions and balancing speed with accuracy, thereby maintaining exceptional performance on benchmarks while still achieving real-time processing speeds [99].

**YOLOv3:** Building upon its predecessors, YOLOv3, introduced by Redmon and Farhadi [100] in 2018, markedly enhances object detection capabilities by incorporating multi-scale detection and an objectness score for bounding boxes, particularly improving the detection of smaller objects [98]. Utilizing the Darknet-53 backbone, as depicted in Figure 3.6, this version enhances feature representation and mitigates the vanishing gradient problem. Moreover, YOLOv3 employs a similar concept to Feature Pyramid Networks (FPN) [81] for detecting objects at three different scales and diverges from previous softmax approaches by using logistic regression for objectness scores and independent logistic classifiers for each class. These

advancements make YOLOv3 particularly effective in real-time object detection scenarios [100].

	Type	Filters	Size	Output
	Convolutional	32	$3 \times 3$	$256 \times 256$
	Convolutional	64	$3 \times 3 / 2$	$128 \times 128$
1x	Convolutional	32	$1 \times 1$	$128 \times 128$
	Convolutional	64	$3 \times 3$	
	Residual			
	Residual			
	Convolutional	128	$3 \times 3 / 2$	$64 \times 64$
2x	Convolutional	64	$1 \times 1$	$64 \times 64$
	Convolutional	128	$3 \times 3$	
	Residual			
	Residual			
	Convolutional	256	$3 \times 3 / 2$	$32 \times 32$
8x	Convolutional	128	$1 \times 1$	$32 \times 32$
	Convolutional	256	$3 \times 3$	
	Residual			
	Residual			
	Convolutional	512	$3 \times 3 / 2$	$16 \times 16$
8x	Convolutional	256	$1 \times 1$	$16 \times 16$
	Convolutional	512	$3 \times 3$	
	Residual			
	Residual			
	Convolutional	1024	$3 \times 3 / 2$	$8 \times 8$
4x	Convolutional	512	$1 \times 1$	$8 \times 8$
	Convolutional	1024	$3 \times 3$	
	Residual			
	Residual			
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figure 3.6: The Darknet-53 architecture, employed as the feature extractor in YOLOv3, supersedes the Darknet-19 used in earlier iteration. This model comprises 53 convolutional layers, including alternating  $3 \times 3$  and  $1 \times 1$  convolutions, enhanced by residual connections inspired by ResNet. These connections mitigate the vanishing gradient problem, facilitating the learning of complex feature representations in deep networks, thus enhancing detection performance. [100, 101].

**YOLOv4 and YOLOv5-Advances in Architecture:** Introduced in April 2020 by Bochkovskiy et al. [102], YOLOv4 represents a significant advancement in the architecture of object detection models, enhancing both detection accuracy and robustness while maintaining real-time performance [102]. This model integrates key architectural components that define modern object detectors: a pre-trained backbone, a neck consisting of various feature aggregation paths, and heads for predicting object classes and bounding boxes. Specifically, YOLOv4 incorporates the CSPDarknet53 [95] backbone, a SPP module [103], and a modified PANet [104] path-aggregation neck, complemented by mechanisms like Cross-Iteration Batch Normalization (CBN) [105] and Spatial Attention Module (SAM) [106]. These features combined with the use of multiple anchor points for single ground truth detection collectively improve the selection ratio of positive samples and boundary detection accuracy, underscored by the adoption of Complete Intersection over Union (CIoU) loss for enhanced localization [102]. Figure 3.7 illustrates the components of the YOLOv4 object detector, depicting the integration of backbone, neck, and heads in a structured manner.

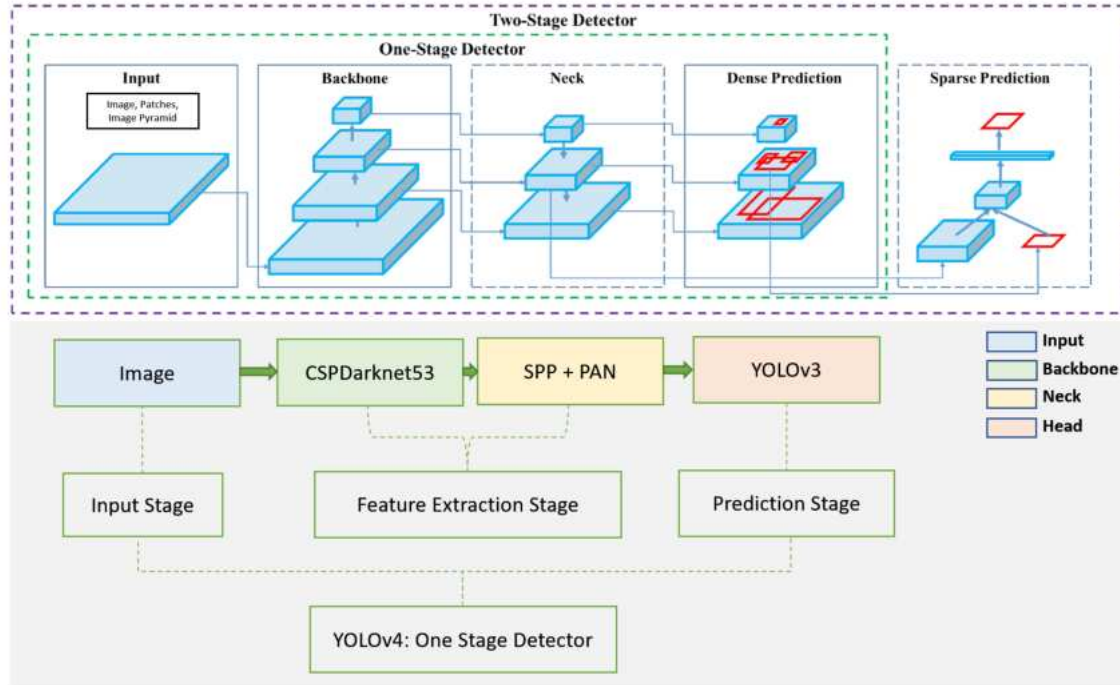


Figure 3.7: Block diagram of object detectors focusing on YOLOv4, illustrating the architectural divisions into input stage, feature extraction stage (backbone and neck), and prediction stage (heads). The diagram also abstracts the structure of object detectors into one-stage and two-stage categories, highlighting the differences in how these models process inputs to produce dense and sparse predictions, respectively. Adapted from [102].

Shortly after the release of YOLOv4, YOLOv5 was launched in 2020 by Glenn Jocher and managed by Ultralytics [107, 96]. Unlike its predecessor, YOLOv5 departs from the Darknet framework, utilizing PyTorch to facilitate accessibility and extend usability, particularly beneficial due to PyTorch’s user-friendly interface. The architecture of YOLOv5 is built upon a Cross-Stage-Partial-connections (CSP) [95] based CSPNet backbone derived from ResNet [101], enhanced by Spatial Pyramid Pooling (SPP) blocks and a Path Aggregation Network (PANet) module for more effective feature extraction and aggregation across multiple scales. Its prediction mechanism employs anchor-based strategies with a loss function that combines Binary Cross-Entropy and CIoU, aiming to optimize class detection, objectness, and localization performance. Unlike YOLOv4, which uses predefined anchor boxes, YOLOv5 can auto-learn anchor sizes directly from the training data during the training process. This leads to better adaptation to specific datasets and potentially improves detection performance. [107, 108, 96].

**PP-YOLO:** Building upon the advancements of its predecessors, PP-YOLO was introduced by Long et al. [109] in August 2020, utilizing the foundational YOLOv3 framework to develop an object detector that achieves a commendable balance between effectiveness and efficiency, suitable for real-world applications.

Operating within the PaddlePaddle ecosystem developed by Baidu, PP-YOLO surpasses YOLOv4 in performance by incorporating a series of refined techniques derived from subsequent research [110, 102, 111, 112, 113, 114, 78, 115].

A significant enhancement in PP-YOLO is the replacement of the DarkNet-53 backbone used in YOLOv3 with the ResNet50-vd-dcn, which includes deformable convolutional layers (DCNs). The adoption of the ResNet50-vd-dcn backbone not only optimizes the balance between performance and computational efficiency but also enhances the multi-scale detection capabilities of the model. Additional improvements in PP-YOLO include the use of DropBlock regularization [110] within the Feature Pyramid Network (FPN) neck, enhancing model generalization by preventing co-adaptation of feature detectors. Matrix NMS [113] is employed for more efficient and accurate non-maximum suppression, while CoordConv layers [114] provide the network with spatial context, aiding in improved localization. Furthermore, the model integrates IoU Loss [111] and IoU-aware detection [112], which refine prediction accuracy aligned with the mean Average Precision (mAP) metric. Additionally, PP-YOLO employs Grid Sensitive techniques [102] to enhance the handling of anchor box boundaries, ensuring that the boxes are more accurately aligned with the detected objects. It also utilizes Spatial Pyramid Pooling (SPP) [78], a method that aggregates and processes spatial features from various scales, thus bolstering the model's robustness in detecting objects across different resolutions. The incorporation of a distilled ResNet50-vd model as a pre-trained backbone [115] leverages enhanced transfer learning capabilities, making PP-YOLO a potent tool for object detection tasks [109].

**Advancing YOLO- Diverse Architectural Innovations:** Following the trajectory set by earlier models, the YOLO architecture continued to evolve with Scaled-YOLOv4. **Scaled-YOLOv4**, introduced by Wang et al. [116], scales the CSPDarknet53 to boost performance across various network sizes, benefiting from training techniques used in YOLOv5 for enhanced performance even in scaled-down versions [116]. **YOLOv5**, emerging in the subsequent period, integrates multi-task learning by blending explicit and implicit knowledge to enhance shared representations, which aids in reducing the parameter count while ensuring competitive performance across tasks like object detection and image captioning [117]. **YOLOS** redefines the application of Transformers to vision, adopting a minimalistic approach to the Vision Transformer architecture to deliver significant results with minimal reliance on conventional convolutional networks [118]. **YOLOv6**, surpassing previous YOLO iterations in 2021, adopts an anchor-free detection method, integrating a decoupled head and SimOTA for advanced label assignment, achieving state-of-the-art results [119]. Lastly, **PP-YOLOv2**, released by Baidu in April 2021, advances PP-YOLO with enhancements like the Mish activation function and a Path Aggregation Network, optimizing performance while maintaining high inference speeds [120, 98].

**YOLOv6:** YOLOv6, introduced in 2022 by Li et al. [121], is designed for industrial applications with a hardware-conscious architecture. Key components include the EfficientRep backbone, Rep-PAN neck, and an Efficient Decoupled Head, all optimized for accuracy and efficiency. Smaller networks use Rep-

Block [122] as the backbone, while larger networks employ the CSPStackRep block [121]. Task Alignment Learning (TAL) [123] is adopted for label assignment, and RepOptimizer [124] ensures quantization-friendly weights. Quantization-Aware Training (QAT) with channel-wise distillation [125] and graph optimization further enhance performance. YOLOv6 employs VariFocal Loss [126] for classification and SIOU [127] or GIoU [128] for regression, achieving substantial performance gains.

**YOLOv7:** That same year, YOLOv7, introduced by Wang et al. [129], set a new benchmark in real-time object detection by optimizing memory utilization and gradient propagation through its Extended-ELAN (E-ELAN) architecture. E-ELAN enhances learning capability without disrupting gradient pathways, utilizing expand, shuffle, and merge cardinality operations. YOLOv7 introduces trainable bag-of-freebies, including batch normalization integration, implicit knowledge from YOLOR [117], and an Exponential Moving Average (EMA) model for inference optimization. Key innovations include planned model re-parameterization to improve gradient propagation and a coarse-to-fine dynamic label assignment strategy for multi-output layers. Additionally, YOLOv7 proposes extended and compound scaling methods to effectively utilize parameters and computation, achieving approximately 40% reduction in parameters and 50% reduction in computation compared to previous state-of-the-art detectors, while delivering faster inference speeds and higher accuracy [129].

**YOLOv8:** YOLOv8 [130], released in 2023 by Ultralytics, introduces significant architectural advancements to enhance performance and efficiency in real-time object detection as depicted in figure 3.8. Key components of its architecture include the C2f module, which features two parallel gradient flow branches to improve gradient information flow and overall model robustness [130, 131]. The model also incorporates the Spatial Pyramid Pooling Fusion (SPPF) layer, a feature shared with YOLOv5, to extract contextual information from images at varying scales, enhancing generalization capabilities [107]. The backbone, CSP-Darknet53, facilitates feature extraction, while the neck adopts Path Aggregation Network (PAN) principles, with modifications to the Feature Pyramid Network (FPN), such as replacing the C3 module with the C2f module and removing convolutional structures during upsampling [130, 132, 131]. Additionally, YOLOv8 removes the objectness branch and employs a decoupled head in the prediction stage, separating classification and detection heads to optimize outputs. YOLOv8 supports a wide range of tasks, including object detection, instance segmentation, pose estimation, and classification, with pre-trained models available to suit various application requirements [130].

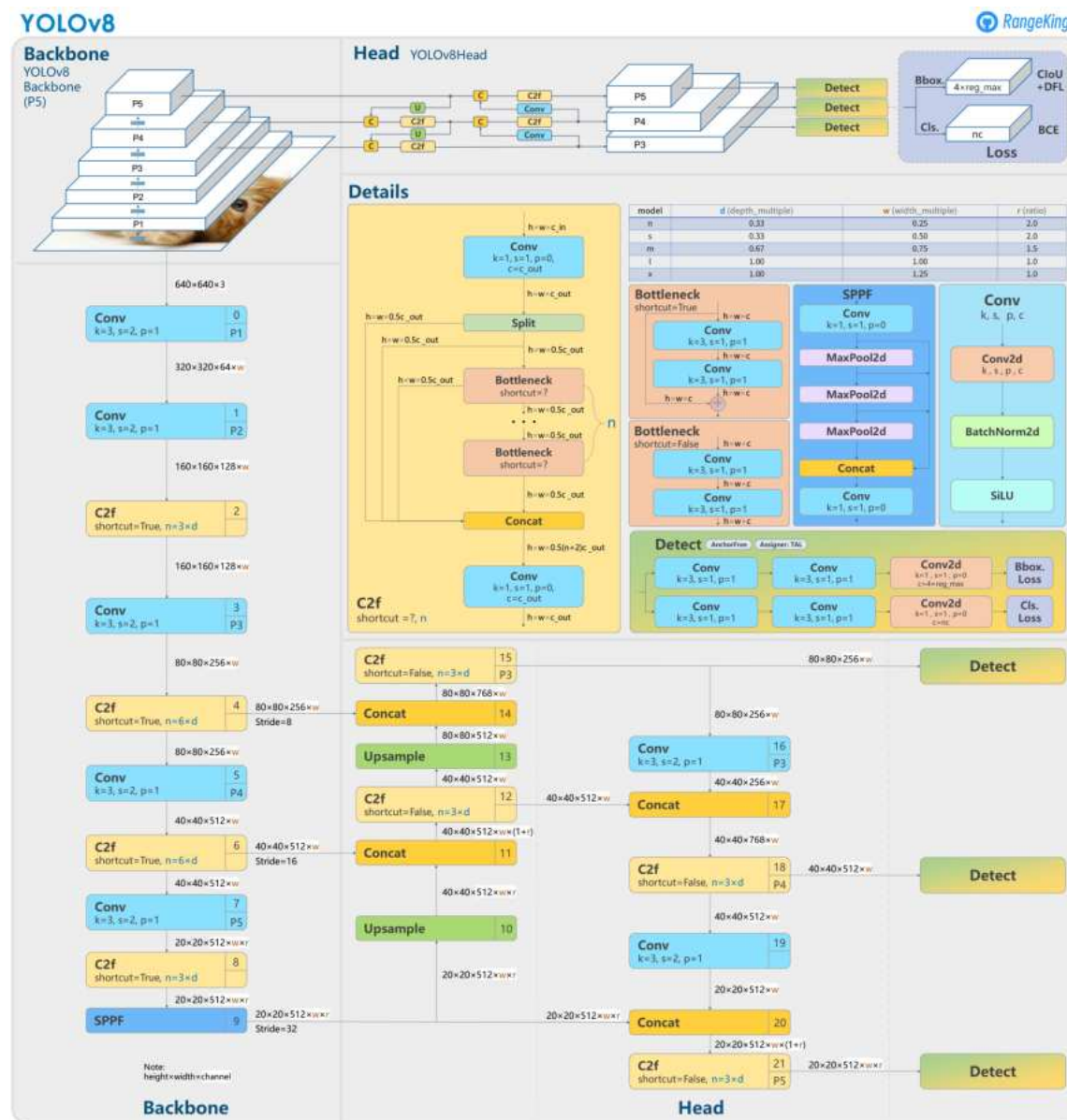


Figure 3.8: Detailed architecture diagram of YOLOv8, illustrating the model's backbone, feature extraction modules, and detection heads. The diagram showcases key components such as the CSPDarknet backbone, SPPF layer, C2f modules, and multi-scale detection modules [130, 132]. This detailed visualization of the YOLOv8 architecture was created by GitHub user RangeKing [133].

**YOLOv9:** The YOLOv9 [134] architecture introduced in 2024, represents a significant leap in object detection by introducing two major innovations: Programmable Gradient Information (PGI) and the Generalized Efficient Layer Aggregation Network (GELAN). PGI tackles challenges such as information bot-

tlenecks and unreliable gradient updates in deep neural networks through the use of auxiliary reversible branches. These branches ensure that gradients remain accurate and effective, even in lightweight and shallow networks. GELAN, an evolution of ELAN [135], integrates gradient path planning from CSPNet [95] and ELAN to balance lightweight design, inference speed, and accuracy. This novel architecture extends ELAN’s capabilities by supporting a wide range of computational blocks, making it highly adaptable to various inference devices. By combining PGI and GELAN, YOLOv9 achieves state-of-the-art performance, surpassing previous methods in both parameter efficiency and accuracy, while offering substantial improvements for lightweight models [134].

**YOLOv10:** Wang et al. [136] introduced YOLOv10 in 2024, targeting the challenges of accuracy and efficiency in real-time object detection. A defining feature of YOLOv10 is its NMS-free framework, which eliminates the reliance on non-maximum suppression during post-processing, significantly reducing inference latency while maintaining robust performance. The model incorporates a consistent dual assignment strategy, as illustrated in Figure 3.9. During training, a one-to-many head enriches supervision by generating multiple positive samples for each instance, while a one-to-one head ensures precise label assignment. These heads are jointly optimized, allowing the backbone and neck to leverage the complementary strengths of both branches. For inference, the one-to-many head is discarded, and the one-to-one head exclusively generates predictions, streamlining the process without additional computational overhead.

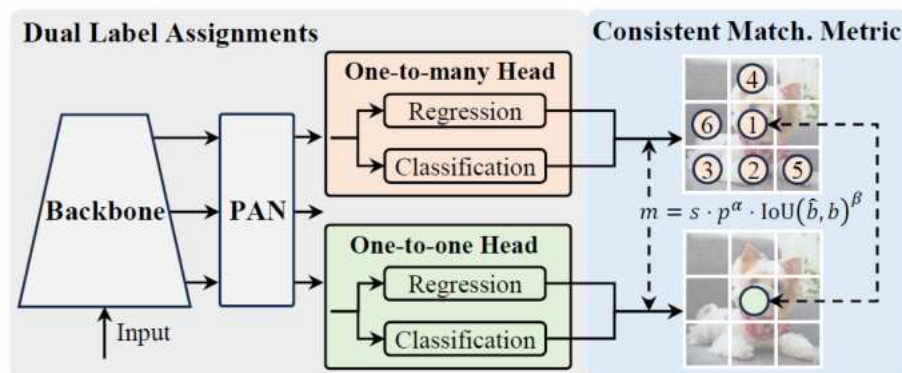


Figure 3.9: Illustration of YOLOv10’s dual label assignment strategy and consistent matching metric. The one-to-many head provides enriched supervision during training, while the one-to-one head ensures precise predictions during inference. The consistent matching metric harmonizes the optimization of both heads by balancing classification scores, spatial priors, and IoU, enabling efficient and accurate real-time object detection [136].

A consistent matching metric, depicted in Figure 3.9, harmonizes the optimization of the one-to-one and one-to-many heads by quantitatively evaluating concordance between predictions and ground truths. This metric balances the classification score, spatial prior, and Intersection over Union (IoU) through tunable hyperparameters, ensuring alignment between the two branches. Complementing this innovative training strategy are architectural refinements such as lightweight classification heads, spatial-channel decoupled

downsampling, and rank-guided block designs, all of which enhance computational efficiency. Additionally, YOLOv10 leverages large-kernel convolutions for broader receptive fields and partial self-attention for improved feature representation, achieving superior detection accuracy with minimal computational cost. These advancements established YOLOv10 as a state-of-the-art real-time object detection system of its time, offering remarkable accuracy, scalability, and efficiency across various model scales [136].

**YOLO11:** Concluding the evolution of the YOLO series, **YOLO11** [137] represents the pinnacle of real-time object detection technology, encapsulating years of progressive innovations. Building upon the foundation laid by its predecessors, YOLO11 introduces the *Cross-Stage Partial with Kernel Size 2 (C3k2)* block, a more efficient alternative to the *C2f* block from YOLOv8 (refer to Figure 3.10). This architectural enhancement significantly boosts computational efficiency and processing speed. Furthermore, the integration of advanced modules like *Spatial Pyramid Pooling - Fast (SPPF)* and *Convolutional Block with Parallel Spatial Attention (C2PSA)* elevates feature extraction capabilities and detection precision to new heights, making YOLO11 a transformative advancement in real-time object detection [138, 137].

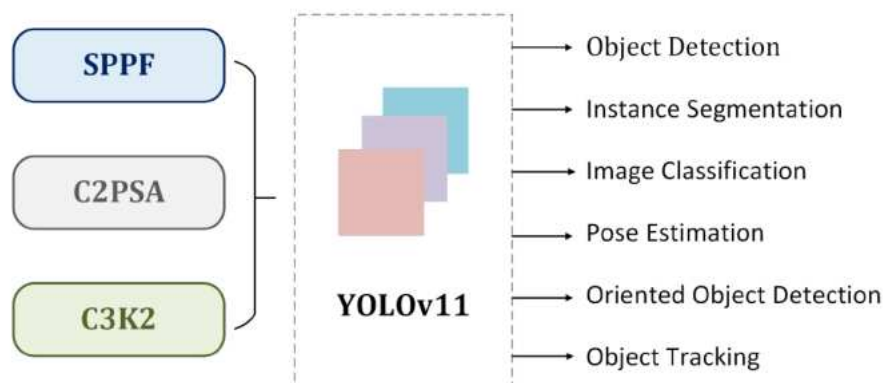


Figure 3.10: Key architectural modules- SPPF, C2PSA, and C3k2 in YOLO11. Figure adapted from [138].

Designed with versatility in mind, YOLO11 seamlessly adapts across diverse deployment environments, ranging from resource-constrained edge devices to cloud platforms and GPU-accelerated systems. It extends its support to a broad spectrum of computer vision tasks, including object detection, instance segmentation, image classification, pose estimation, oriented object detection, and even object tracking [139]. As the latest milestone in the series, it sets a new standard for real-time computer vision, reinforcing YOLO's legacy as a transformative force in the field.

Table 3.1 summarizes nearly a decade of advancements in YOLO models, spanning from 2015 to 2024, along with their respective sources for further reading. For a more comprehensive analysis, readers can refer to the provided citations for detailed insights into each YOLO model. In essence, each iteration of the YOLO series has significantly advanced real-time object detection by improving speed, accuracy, and adaptability

across diverse computing environments, reinforcing YOLO’s pivotal role in computer vision.

### 3.4 Foundation Models in AI

The term “foundation model” was introduced by researchers at the Stanford Center for Research on Foundation Models and the Stanford Institute for Human-Centered Artificial Intelligence (HAI) in their 2021 paper titled “*On the Opportunities and Risks of Foundation Models*”. This seminal work defined foundation models as large-scale machine learning systems trained on extensive and diverse datasets, often leveraging self-supervised or semi-supervised learning techniques. These models are designed to generalize across various downstream tasks and can be fine-tuned for specific applications, making them a cornerstone of modern Artificial Intelligence (AI) [143].

A notable example of a foundational model in computer vision is **CLIP (Contrastive Language–Image Pretraining)** [144], developed by OpenAI. CLIP aligns visual and textual representations, enabling tasks like image classification and object recognition in a zero-shot fashion. This means that the model can perform tasks on data it has never explicitly seen during training. In the domain of object detection, **YOLO-NAS** [141], developed by Deci AI, exemplifies another foundational model. YOLO-NAS combines Neural Architecture Search (NAS) techniques with pre-training on large datasets and a quantization-aware design, achieving superior efficiency and accuracy in real-world applications [141].

**Zero-shot segmentation** refers to a model’s ability to segment objects in an image or video without requiring prior training on the specific object or domain [145]. A key example of this is the **Segment Anything Model (SAM)** [146], developed by Meta AI in 2023. SAM enables users to segment objects using prompts such as points, bounding boxes, or text, and demonstrates exceptional generalization across diverse datasets and tasks [146]. Similarly, **Grounding DINO** [147], developed by IDEA Research, specializes in **zero-shot object detection**, allowing the model to detect and localize arbitrary objects in images based on textual prompts without task-specific training [148]. This flexibility makes it highly effective for open-set object detection scenarios.

By combining the segmentation capabilities of SAM with the object detection strengths of Grounding DINO, **GroundedSAM** [149] emerges as a powerful composite model. GroundedSAM leverages textual prompts for object detection and refines results using pixel-perfect segmentation masks. This innovative integration showcases how foundational models can be assembled to create versatile tools for specific applications. One of the most significant applications of GroundedSAM is in dataset labeling. It can automate the generation of precise bounding boxes and segmentation masks, streamlining the creation of annotated datasets. This capability accelerates the development of high-quality datasets for fields such as autonomous vehicles, and industrial inspection, reducing manual effort and transforming workflows [150, 151, 149].

Table 3.1: Summary of Advancements in YOLO Models

Model	Year	Advancements	Supported Tasks and Modes
YOLOv1 [82]	2015	One-shot object detector	Object Detection, Basic Classification
YOLOv2 [99]	2016	Darknet-19 Framework, Batch Normalization, High Resolution Classifier, Convolutional With Anchor Boxes, Dimensional clustering, Direct location prediction, Multi-scale training	Object Detection, Improved Classification
YOLOv3 [100]	2018	Objectness score for each bounding box, Logistic classifiers for each class, Darknet-53 backbone, Detection at Multiple Scales	Object Detection, Multi-scale Detection
YOLOv4 [102]	2020	Weighted-Residual Connections (WRC), Cross-Stage Partial connections (CSP), DropBlock Regularization, Mosaic Data Augmentation, CSPDarknet-53 backbone, Spatial Pyramid Pooling (SPP) and Path Aggregation Network (PAN), Self-Adversarial Training (SAT)	Object Detection, Basic Object Tracking
YOLOv5 [107]	2020	AutoLearning of Anchor Boxes, PANet, Training Enhancements	Object Detection, Basic Instance Segmentation (via custom modifications)
PP-YOLO [109]	2020	Larger Batch Size, EMA, DropBlock regularization, IoU Loss, IoU Aware, Grid Sensitive, Matrix NMS, SPP, Distilled ResNet50-vd as a better Pretrained Model	Object Detection
Scaled-YOLOv4 [116]	2020	Network scaling approach to modify the depth, width, resolution and the structure of the network.	Object Detection
YOLOR [117]	2021	How to construct a unified network?, Integration of implicit and explicit knowledge	Object Detection
YOLOS [118]	2021	2D Object detection in a pure sequence-to-sequence manner	Object Detection
YOLOX [119]	2021	Anchor-free detection, Decoupled head, Label assignment strategy: SimOTA	Object Detection, Classification
YOLOv6 [121]	2022	Self-Distillation Strategy, Reformed the quantization scheme for detection, RepOptimizer, Channel-wise distillation, Label Assignment using Task Alignment Learning (TAL)	Object Detection, Instance Segmentation
YOLOv7 [129]	2022	E-ELAN reparameterization, Extended and Compound Scaling, Dynamic Label Assignment	Object Detection, Instance Segmentation, Pose Estimation-(via custom modifications)
YOLOv8 [130]	2023	C2f module, Spatial Pyramid Pooling Fusion (SPPF) layer, Anchor-free detections, Decoupled head in the prediction stage	Object Detection, Instance Segmentation, Pose/Keypoints, Oriented Detection, Classification
YOLOv6 3.0 [140]	2023	Bidirectional Concatenation (BiC) Module, Anchor-Aided Training (AAT) Strategy, New self-distillation strategy	Object Detection
YOLO-NAS [141]	2023	Neural architecture search to optimize detection tasks	Object Detection
YOLO-World [142]	2024	Zero-shot detection using text prompts, prompt then detect	Object Detection
YOLOv9 [134]	2024	Programmable Gradient Information (PGI), Generalized Efficient Layer Aggregation Network (GELAN)	Object Detection, Instance Segmentation
YOLOv10 [136]	2024	NMS-Free Training, Dual label assignments, Holistic model design, Large-kernel convolutions and Partial self-attention modules	Object Detection
YOLO11 [137]	2024	Advanced architectural optimizations for real-time detection	Object Detection, Instance Segmentation, Pose/Keypoints, Oriented Detection, Classification

## Chapter 4

### Methodology

Building on the extensive review of the object detection literature outlined in previous chapters, this chapter delineates a detailed methodology for simulating the Tesla Model 3 within the CARLA simulator environment. It describes in detail the procedures involved in sensor data collection, the creation of a customized image dataset for object detection, and the execution of real-time inference using a trained YOLO model. This exploration provides a thorough examination of assumptions, methodologies, and technical implementations, offering a complete overview of the approach undertaken. To improve clarity and understanding of the system's design and workflow, various Unified Modeling Language (UML) diagrams are integrated throughout this chapter to visually articulate the complex processes and interactions within the simulation framework.

#### 4.1 Overview of Methodology

This section systematically outlines the methodology of the project, introducing the five key stages involved to ensure comprehensive understanding and clarity. The stages are visually represented in Figure 4.1, which illustrates the sequential steps of the project workflow. The stages include: Stage 1: Simulation Setup, where the simulation environment for the Tesla Model 3 is configured; Stage 2: Data Acquisition, involving the collection of sensor data; Stage 3: Dataset Creation, where data is curated into a structured image dataset for object detection; Stage 4: Model Training, during which the YOLO models are trained and evaluated; and Stage 5: Real-Time Object Detection, where the best performing trained YOLO model is applied within the CARLA simulator to detect objects in real-time. This structured breakdown provides a clear roadmap of the project's process flow from setup to implementation. Let us now proceed to examine each of these stages in detail, starting with the simulation setup and culminating in real-time object detection, to gain a comprehensive understanding of the entire project lifecycle.

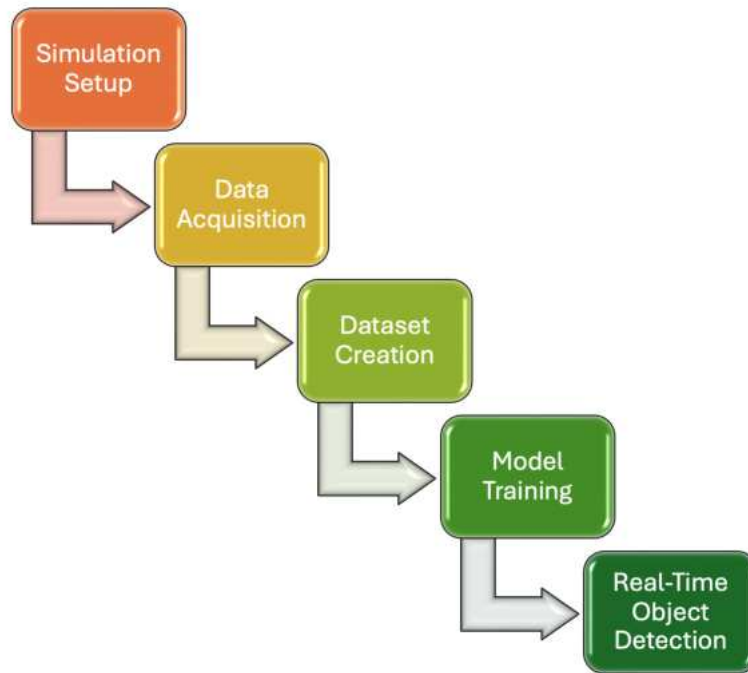


Figure 4.1: Five stages involved in this project.

**Simulation Setup:** This initial phase focuses on establishing a realistic simulation environment that closely mimics real-world conditions. It involves configuring the CARLA simulation environment to incorporate a detailed vehicle model—specifically the Tesla Model 3—complete with a comprehensive suite of sensors. The setup is further refined by simulating dynamic environmental conditions and diverse weather scenarios to test the robustness of the object detection algorithms under varying operational contexts.

**Data Acquisition:** The second phase is dedicated to the systematic collection of data necessary for training and validating the object detection models. This involves deploying a data collection pipeline that utilizes the sensors configured in the simulation setup to gather synthetic data from the simulated environment. The data collected spans various scenarios, including different times of day and weather conditions, to ensure a rich dataset that supports robust model training.

**Dataset Creation:** Following data acquisition, the raw data is processed to create a structured dataset suitable for training object detection models. This dataset is initially annotated automatically with an AI-based labeling process, followed by manual corrections to ensure high accuracy and relevance of the data labels. This curated dataset is specifically tailored for training object detection models, categorized into classes such as vehicles, pedestrians, traffic lights, and signs.

**Model Training:** The fourth phase involves training multiple object detection models, particularly different configurations and versions of the YOLO model, using the curated dataset. This phase focuses on optimizing the models to achieve high accuracy and efficiency, with the best-performing model selected based on the metric mean Average Precision (mAP50-95).

**Real-Time Object Detection:** The final phase implements the best-performing YOLO model for inference in a real-time object detection setup within the CARLA environment. This setup tests the model's effectiveness in detecting objects or classes in real-time as the vehicle navigates through the simulated environment. The outcomes are visually documented using the OpenCV library to capture and display the detection results in action.

## 4.2 Simulation Setup

The Tesla Model 3 is selected for its compatibility with CARLA's physics engine. Initialized using CARLA's blueprint library, the vehicle is randomly spawned within the *Town06* map which can be changed to switch between a complex urban and rural setting. It operates under autopilot, adhering to traffic rules and dynamically interacting with its environment. CARLA's Traffic Manager oversees the vehicle's behavior, controlling parameters like target speed (set at 80 km/h), lane changes, and responses to traffic signals. This setup not only simulates realistic driving behaviors but also integrates dynamic elements such as NPC vehicles and pedestrians. These entities, managed by CARLA's Traffic Manager and pedestrian AI, engage with the ego vehicle in real-time, creating realistic traffic scenarios and enhancing the simulation's complexity. Environmental variations, like weather changes, are introduced to further test the system's robustness under diverse conditions.

### 4.2.1 Simulation Environment Implementation

Implemented in Python, this section manages data collection and initiates real-time object detection. It interacts with the *CarlaWorld* to manage simulation parameters and with the *YOLOInference* to process detection tasks, orchestrating the simulation workflow effectively. The **Component Diagram** (Figure 4.2) illustrates the system's architecture, divided into three main sections: Client, CARLA Server, and Configurations.

- **Client Section:** Manages the primary operations including data collection and real-time object detection.
- **CARLA Server Section:** Details the components that interact directly with the CARLA simulation environment, ensuring seamless integration and operation.
- **Configurations Section:** Houses configuration files that provide essential parameters for accurate sensor settings and customized operations, thereby ensuring the simulation's fidelity and adaptability.

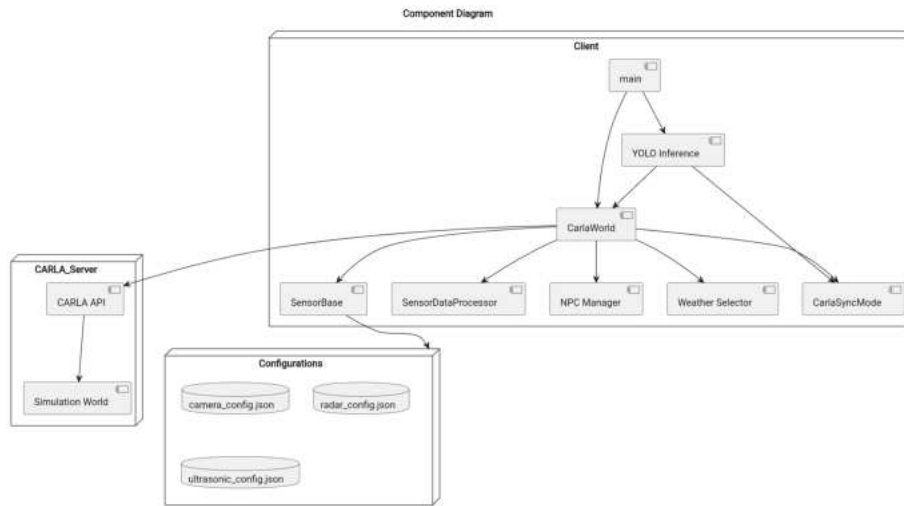


Figure 4.2: Component Diagram of the System

The CARLA simulation environment is set up using the `CarlaWorld` class, which serves as the central controller. This class manages the CARLA client, loads maps, and integrates sensors and NPCs into the simulation. As shown in the **Class Diagram** (Figure 4.3), `CarlaWorld` coordinates with `SensorBase` and its subclasses (`CameraSensor`, `RadarSensor`, `UltrasonicSensor`) to deploy sensors, which are configured using JSON files and attached to the vehicle.

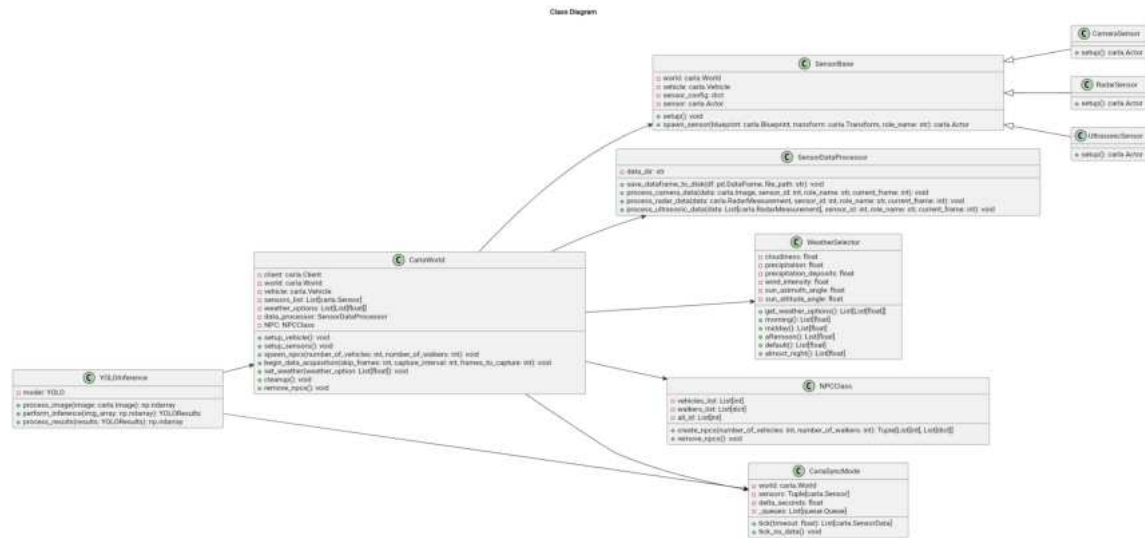


Figure 4.3: Class Diagram of the CARLA Simulation System

The **Sequence Diagram for Sensor Setup** (Figure 4.4) outlines the workflow for configuring and at-

taching sensors. The `setup_sensors` method reads configurations, instantiates sensors, and integrates them into the CARLA world. Errors during this process are logged to ensure robust execution.

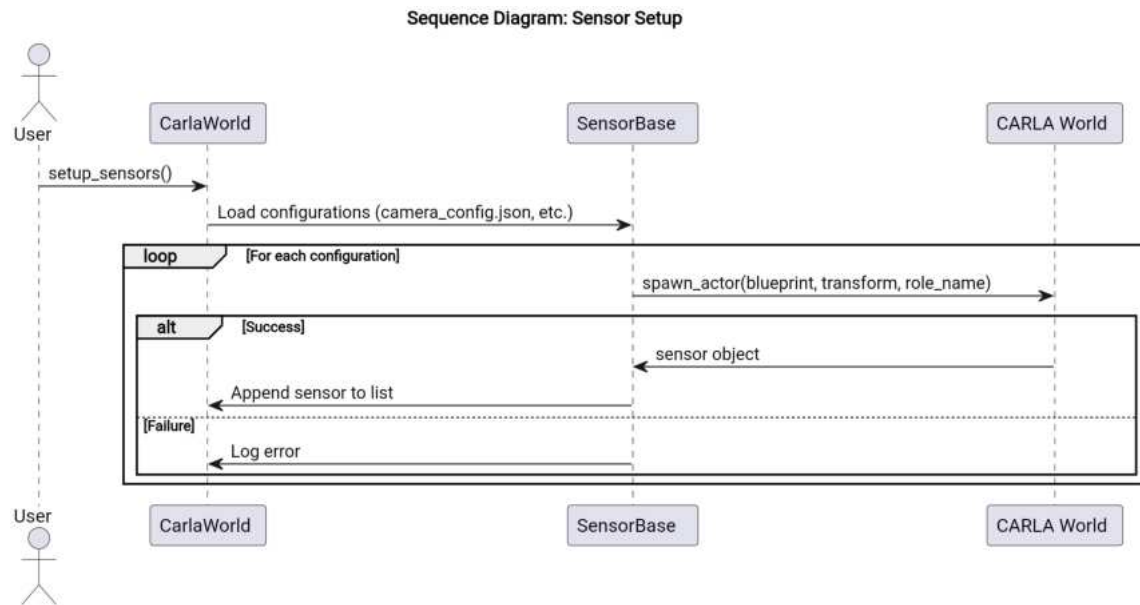


Figure 4.4: Sequence Diagram for Sensor Setup

### 4.2.2 Sensor Suite Configuration

The Tesla Model 3 in this simulation is equipped with a sophisticated array of sensors, modeled to reflect a potential real-world autonomous Tesla Model 3 vehicle setup. This suite comprises 21 sensors: 8 RGB cameras, 1 radar, and 12 ultrasonic sensors, detailed as follows:

- RGB Cameras:** Positioned to provide 360-degree coverage, each camera operates at a resolution of 1280x960 pixels and a synchronized frame rate of 30 FPS, ensuring high-quality, consistent imagery. Specific orientations and fields of view are tailored to distinct operational roles, enhancing environmental awareness and object detection capabilities.
- Radar Sensor:** Mounted on the front, this sensor excels in long-range detection with a 35-degree horizontal field of view and a range of up to 160 meters, contributing to the vehicle's forward sensing capabilities.
- Ultrasonic Sensors:** These are symmetrically distributed around the vehicle, each with a 5-meter range and a 60-degree field of view, crucial for short-range obstacle detection during parking and low-speed maneuvers.

Configuration files for these sensors (`configs/camera_config.json`, `configs/radar_config.json`, `configs/ultrasonic_config.json`) outline the parameters and positioning, ensuring reproducibility and accuracy in simulations.

*Note: The configurations and parameters described are based on assumed values to emulate a realistic autonomous vehicle setup within our simulated environment for testing and development purposes.*

Below, a comprehensive breakdown of the assumed configurations for each type of sensor is provided, ensuring reproducibility and a clear understanding of their functional role in the simulation:

### Camera Suite

The 8 RGB cameras are strategically configured to provide 360-degree coverage:

- **Front Narrow Camera:** Positioned at [2.0, 0.0, 1.5], FOV: 35°, oriented directly forward. Captures high-detail images for long-range detection.
- **Front Wide Camera:** Positioned at [2.0, 0.2, 1.5], FOV: 50°. Provides broader coverage of the road ahead.
- **Front Fisheye Camera:** Positioned at [2.0, -0.2, 1.5], FOV: 120°. Captures peripheral data for lateral object detection.
- **Side Cameras (Left and Right):** Positioned at [1.5, -0.9, 1.2] (left) and [1.5, 0.9, 1.2] (right) with a -45° and 45° yaw, respectively. FOV: 100°. Monitors side lanes for overtaking or merging vehicles.
- **Rear Cameras (Left and Right):** Positioned at [-1.5, -0.9, 1.2] (left) and [-1.5, 0.9, 1.2] (right) with 175° and -175° yaw, respectively. FOV: 100°. Covers blind spots in the rear-side areas.
- **Rear Fisheye Camera:** Positioned at [-2.0, 0.0, 1.5], FOV: 150°, oriented backward. Provides a panoramic view of the rear environment.

### Radar Sensor

A front-facing radar sensor is configured with:

- **Horizontal FOV:** 35°, **Vertical FOV:** 5°
- **Range:** 160 meters, **Points Per Second:** 1500
- **Position:** [1.0, 0.0, 0.2], **Rotation:** [0°, 0°, 0°]
- **Role:** Detects long-range objects, providing velocity and depth measurements for collision avoidance.

### Ultrasonic Sensors

Twelve ultrasonic sensors are distributed evenly around the vehicle, with:

- **Range:** 5 meters, **Horizontal FOV:** 60°, **Vertical FOV:** 5°
- **Six Front Sensors:** Positioned symmetrically across the front to detect objects in the vehicle's immediate path.
- **Six Rear Sensors:** Distributed along the rear to assist in reversing and close-proximity obstacle detection.

#### 4.2.3 Weather and Lighting Conditions

Weather and lighting variations introduce realism into the simulation. The defined conditions include:

- **Morning:** Cloudiness: 20%, Precipitation: 90%, Sun Altitude: 30°—emulates a rainy, low-light morning.
- **Midday:** Cloudiness: 30%, Precipitation: 0%, Sun Altitude: 80°—bright, clear conditions for maximum visibility.
- **Afternoon:** Cloudiness: 50%, Precipitation: 0%, Sun Altitude: -40°—diffused lighting for overcast conditions.
- **Almost Night:** Cloudiness: 30%, Precipitation: 30%, Sun Altitude: -60°—dim lighting to simulate dusk.

These conditions test the robustness of object detection under varying levels of visibility and environmental noise [5].

### 4.3 Data Acquisition

The data acquisition phase is essential for collecting the synthetic data needed to train and validate object detection models. This process, managed by the `CarlaSyncMode` class, ensures that all sensors operate in sync with the simulation timeline, covering various scenarios such as different times of day and weather conditions to compile a diverse and robust dataset. Data acquisition begins with loading sensor configurations from JSON files and adjusting the simulation environment's weather conditions for realism. Non-player characters (NPCs) are introduced to enrich the scenarios. The process continuously loops as long as the acquisition status is active. Each iteration advances the CARLA world, collects data, and processes it through the `SensorDataProcessor` class.

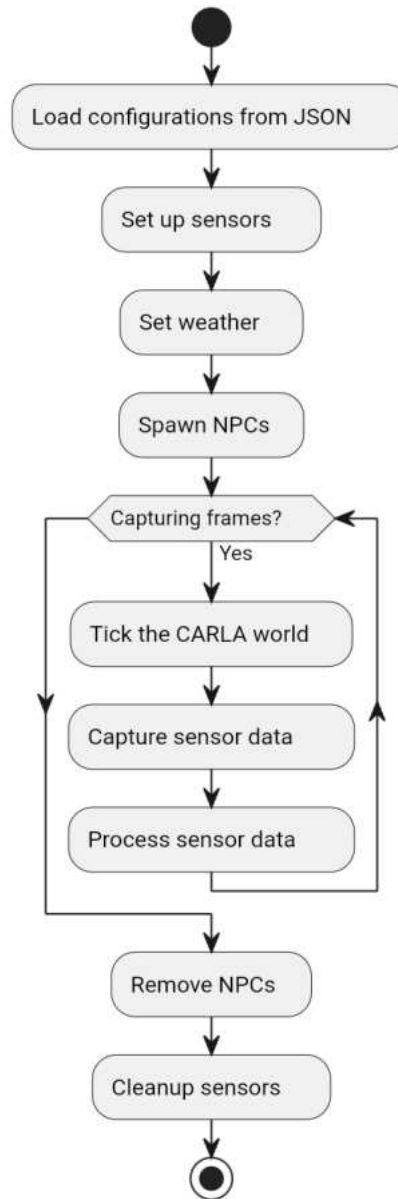


Figure 4.5: Activity Diagram for Data Acquisition

To detail the operational flow, the **Sequence Diagram for Sensor Data Acquisition** (Figure 4.6) shows the activation of `CarlaSyncMode` by the `begin_data_acquisition` function, maintaining synchronization throughout the data collection process. Data collection commands are dispatched to the sensors mounted on the Tesla Model 3. Collected data is then buffered in a queue, and depending on the sensor type, directed to

specific processing functions: `process_camera_data()` for camera sensors, `process_radar_data()` for radar sensors, and `process_ultrasonic_data()` for ultrasonic sensors. Once processed, the data is saved to disk. The procedure concludes with a comprehensive cleanup of all sensors, preparing the system for the next data acquisition cycle.

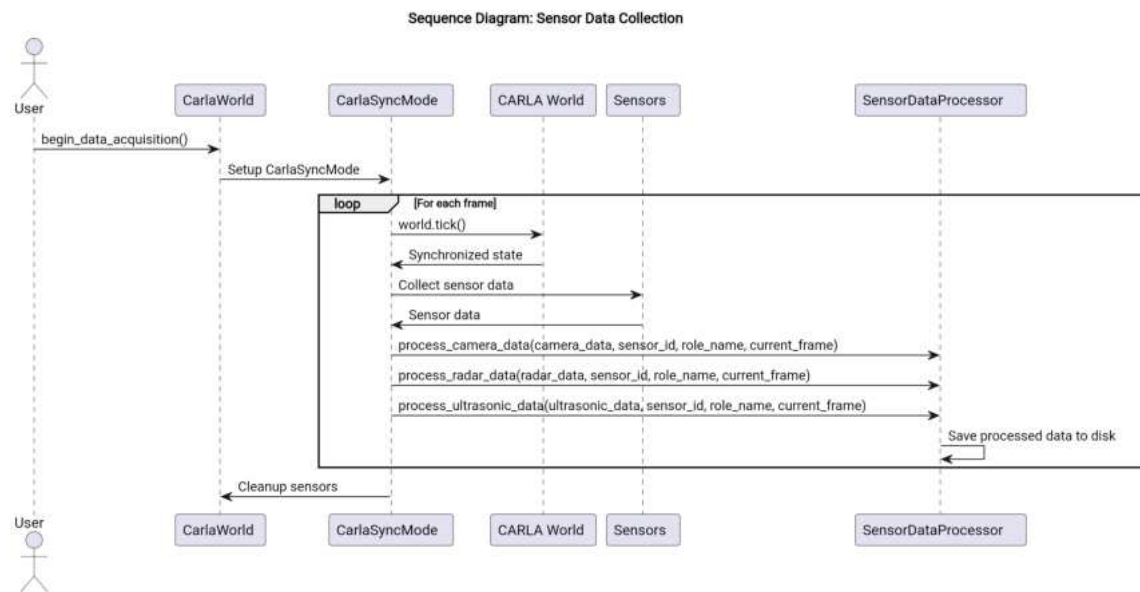


Figure 4.6: Sequence Diagram for Sensor Data Acquisition

### 4.3.1 Data Collection Pipeline

The data collection pipeline is an integral part of the simulation framework, designed to generate synthetic datasets that are temporally and spatially aligned. This pipeline integrates seamlessly with the CARLA simulator to facilitate synchronized data acquisition, metadata logging, and structured data storage, crucial for training and evaluating object detection models in autonomous driving contexts. The pipeline captures data from a simulated Tesla Model 3 equipped with an array of sensors, including RGB cameras, radar, and ultrasonic sensors. These sensors are strategically positioned to provide comprehensive 360-degree environmental coverage. The data from all sensors is synchronized to ensure alignment with the same simulation frame, set at a frame rate of 20 FPS for optimal balance between temporal resolution and computational demand. This synchronization is managed by the `CarlaSyncMode` class and configured in the `set_synchronous_mode.py` file.

**Functionalities of the Data Collection Pipeline:** The data collection pipeline provides several key functionalities:

- **Synchronous Data Acquisition:** The pipeline uses the *CarlaSyncMode* class to enable CARLA's synchronous mode, ensuring that the simulation advances in fixed time steps and all sensors capture data corresponding to the same simulation frame. The `delta_seconds` parameter is set to achieve a frame rate of 20 FPS during data collection, ensuring adequate temporal resolution for most machine learning applications.
- **Sensor Data Processing:** Raw data from sensors is processed in real-time using the *SensorDataProcessor* class. The pipeline handles data from:
  - **Cameras:** Captures RGB images, which are converted and stored in `.png` format.
  - **Radar:** Logs object measurements such as velocity, altitude, azimuth, and depth in `.csv` format.
  - **Ultrasonic Sensors:** Records short-range obstacle detections in `.csv` format.
- **Data Storage and Organization:** The pipeline organizes the data into a hierarchical directory structure, categorizing it by sensor type and sensor role (e.g., `camera_front_wide`, `radar_front`). Metadata associated with each sensor reading, such as timestamp, location, and orientation, is logged in accompanying CSV files. This facilitates easy retrieval and analysis of data.
- **Dynamic Environment Simulation:** NPC vehicles and pedestrians are dynamically introduced into the simulation using the *NPCClass*. Their behaviors and positions are randomized to create diverse scenarios.
- **Scalability and Automation:** The pipeline is designed to handle large-scale data collection tasks, automating the entire process from simulation initialization to data storage.

**Synthetic Data Collection:** The pipeline collects synthetic data by running the simulation with the Tesla Model 3 equipped with the following sensors:

- **RGB Cameras:** Eight cameras placed around the vehicle provide 360-degree coverage. Images are captured at 20 FPS and stored as `.png` files, with corresponding metadata logged in CSV files.
- **Radar:** A single front-facing radar captures object measurements at 20 FPS. Data points include attributes such as velocity and depth, which are stored in a structured CSV file.
- **Ultrasonic Sensors:** Twelve sensors around the vehicle detect short-range obstacles. Measurements are captured at 20 FPS and logged in CSV files.

The data collection process begins after an initial stabilization phase, where frames are skipped to ensure that all sensors are properly initialized and synchronized. For each simulation step, the pipeline captures and processes sensor data, which is then saved to disk in real-time.

**Raw Data Storage:** The collected data is stored in a hierarchical directory structure for efficient management and retrieval. The directory is organized as follows:

```
data/
├── camera_front_wide/
│   ├── images/
│   │   ├── 00001.png
│   │   └── 00002.png
│   └── camera_data.csv
├── radar_front/
│   └── radar_data.csv
├── ultrasonic_left/
│   └── ultrasonic_data.csv
```

Each directory contains:

- **Raw Data:** Sensor outputs such as images or measurements.
- **Metadata:** Associated information such as frame ID, timestamp, sensor position, and orientation.

#### Advantages of the Pipeline:

- **Temporal and Spatial Consistency:** Synchronous mode ensures all sensor data aligns perfectly in time, which is critical for sensor fusion tasks.
- **Diverse Dataset Generation:** Randomized NPC behavior and weather variations create a rich and varied dataset suitable for training robust object detection models.
- **Scalable and Modular Design:** The pipeline can easily be extended to include additional sensors or modified for different simulation scenarios.

## 4.4 Dataset Creation

Following data acquisition, raw camera data from the CARLA simulator is processed into a structured dataset optimized for training object detection models. Initially, an AI-based labeling system automatically annotates the data, which is then manually refined to ensure the precision and relevance of the labels. This dual-stage annotation readies the dataset for categorizing images into classes: vehicles, pedestrians, traffic lights, and signs.

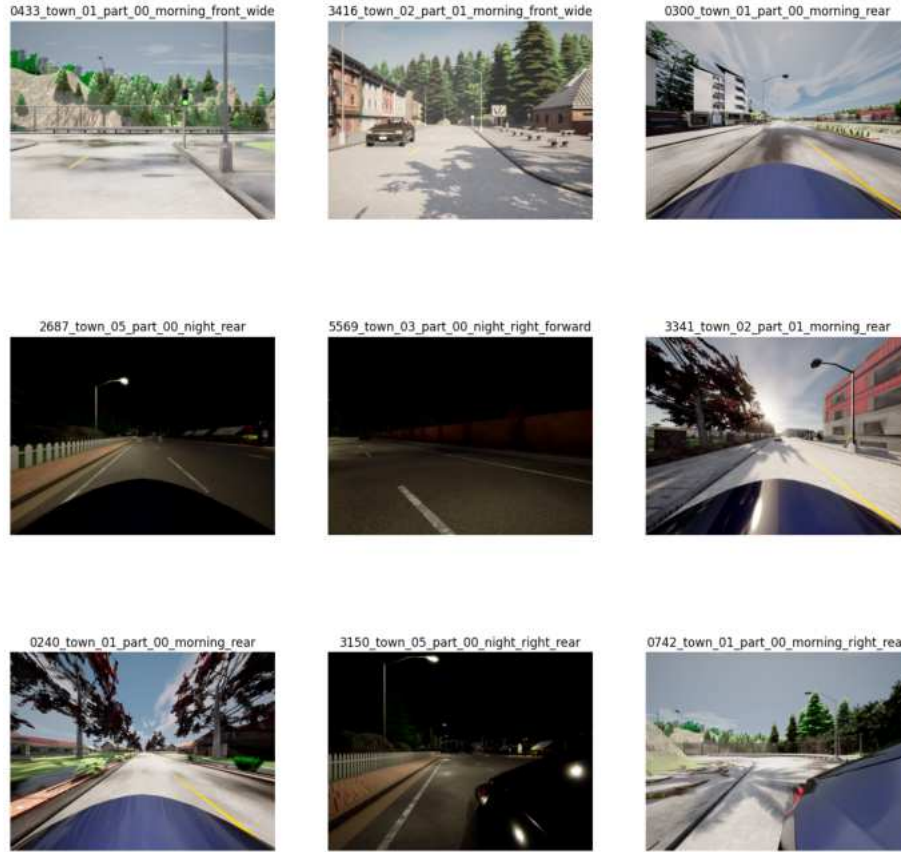


Figure 4.7: Example of reorganised images with new filename convention.

#### 4.4.1 Data Reorganisation

Image data captured from eight cameras on a simulated Tesla Model 3 undergo a comprehensive reorganization process. Initially stored across various subdirectories according to camera position and type, images are consolidated into a unified directory structure under a new naming convention. This convention embeds essential metadata within each filename, such as sequence number, town, simulation segment, time of day, and camera type, for example, `0001_town_01_part_00_night_front_fisheye.png`. This approach not only simplifies data management but also significantly enhances the dataset's utility for object detection model training. A total of 6400 images have been systematically processed and organized, making the dataset robust and ready for detailed annotation. Figure 4.7 illustrates nine random images arranged in a 3 x 3 grid, showcasing the updated filenames according to the new naming convention. This visual representation highlights the systematic and structured nature of the dataset post-reorganization.

### 4.4.2 Labeling Process

Following the reorganization of the image data, the dataset undergoes a two-stage labeling process crucial for training YOLO models for object detection. This process combines automated and manual techniques to ensure high accuracy and relevance of the data labels for four designated classes: vehicles, pedestrians, traffic lights, and signs.

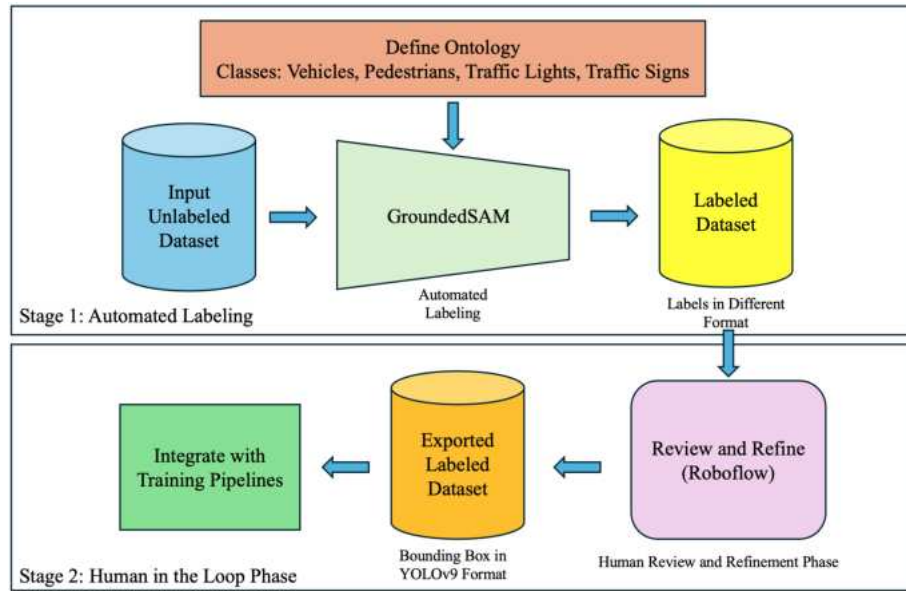


Figure 4.8: Two-Stage Dataset Labeling Process. This figure details each step in the dataset labeling workflow. In the first stage, it shows the preparation of unlabeled input data, definition of class ontologies, configuration of the GroundedSAM model according to these ontologies, and the automated generation of preliminary labels through advanced object detection and segmentation. The second stage illustrates the manual review and refinement process using the Roboflow framework, where labels generated by GroundedSAM are meticulously adjusted to ensure they conform to the precise bounding box format necessary for model specific training for example YOLOv9 format required for training YOLOv9 models for 2D object detection task.

As depicted in Figure 4.8, the labeling process begins with automated label generation followed by manual review. Below is a detailed explanation of each step in the dataset labeling process:

- **Define Ontology:** The first step involves defining the ontology or categories of objects to label. GroundedSAM utilizes these textual descriptions to guide its detection process. It is essential to specify the types of objects (e.g., vehicles, pedestrians, traffic lights) and their attributes (e.g., color, action) to ensure comprehensive detection [151].
- **Automated Label Generation:** GroundedSAM automates the labeling of the 6400 images. This model integrates advanced detection and segmentation capabilities to handle diverse visual scenes. It processes the unlabeled images from a unified directory, generating initial annotations that detail

object boundaries and class identifications. These preliminary labels are stored in an intermediate format for further refinement.

- **Manual Review and Refinement:** Following the automated annotation process, each label is meticulously reviewed manually to ensure precision and improve the delineation of object boundaries. This review phase leverages the capabilities of the Roboflow framework to visualize the segmented contours generated by GroundedSAM, as illustrated in Figure 4.9 and Figure 4.10. Subsequent adjustments align the annotations with the specific format requirements of YOLOv9, highlighted in Figure 4.13, with a particular focus on the accuracy of bounding boxes:
  - **Conversion to Bounding Boxes:** The annotations are transformed from segmented contours into bounding boxes. Each object is encapsulated by coordinates, formatted as `class_id`, `center_x`, `center_y`, `width`, and `height`, to accurately represent their spatial presence.
- **Output Preparation:** The refined annotations are formatted to YOLOv9 specifications and the dataset is segmented into training, validation, and test sets. This structured dataset is then ready for integration into the training pipeline and is compatible with all YOLO models trained in this project.

Figure 4.9 shows the visualization of the automatically generated labels from the GroundedSAM model, providing an initial assessment of object segmentation.

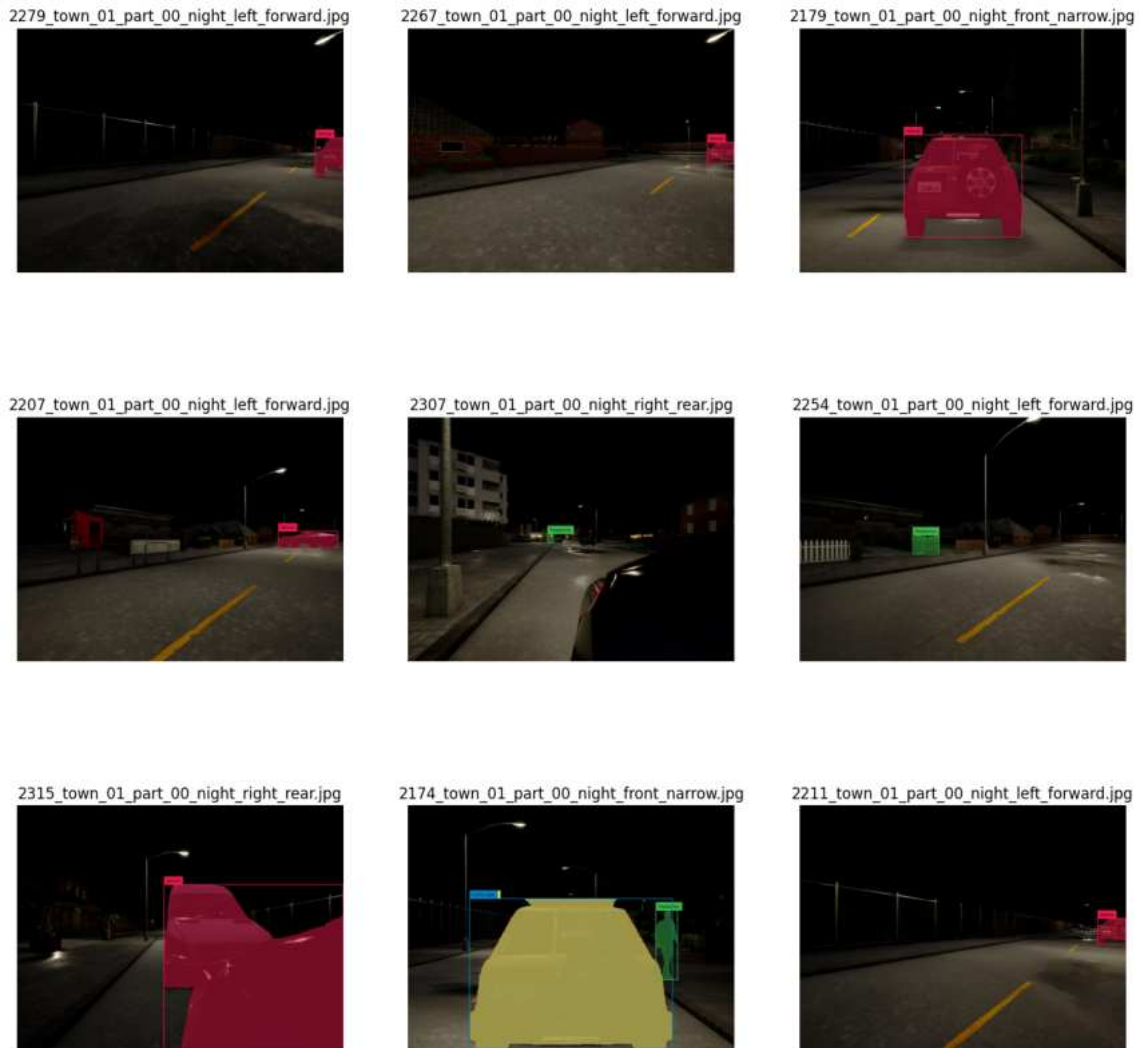


Figure 4.9: Visualization of automatically generated labels using supervision library [152].

Figures 4.10 to 4.13 illustrate the manual review and refinement process. Initially, the generated labels, as shown in Figure 4.10, undergo adjustments using the “Convert to Box” functionality of the Roboflow framework, depicted in Figure 4.11. These modifications ensure the labels conform to the specific format required by YOLOv9, as detailed in Figures 4.11 and 4.13.

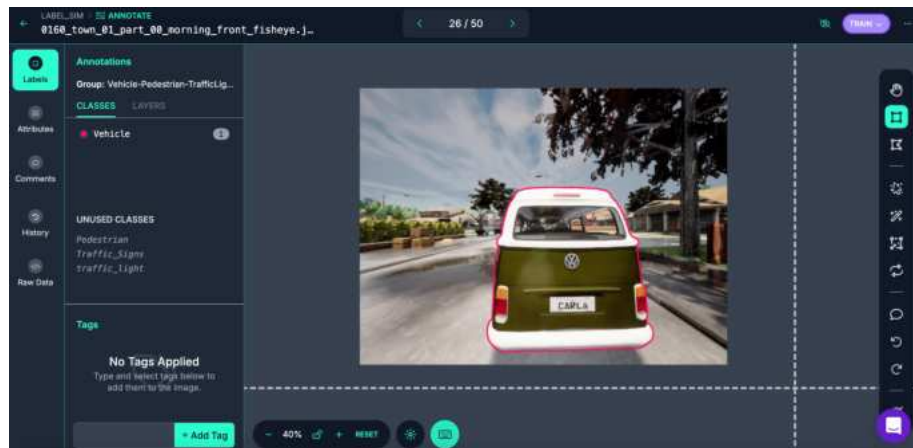


Figure 4.10: Visualization of Generated Labels using Roboflow.

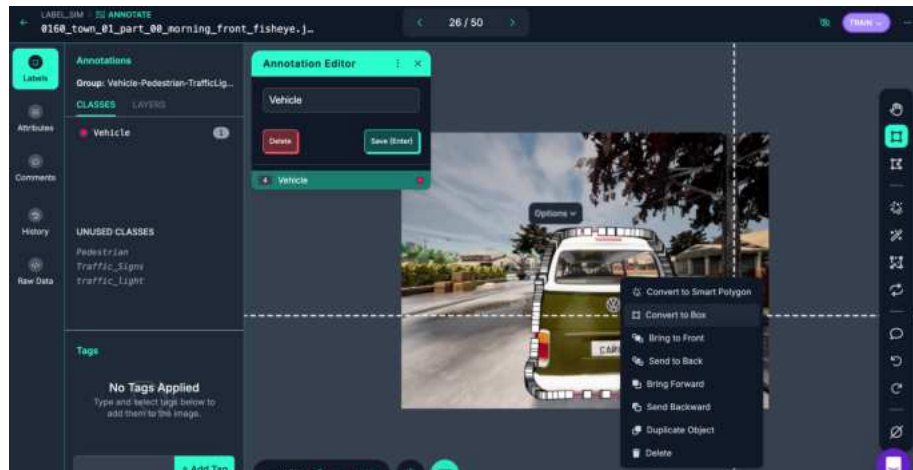


Figure 4.11: Conversion of Generated Labels using Roboflow functionality.

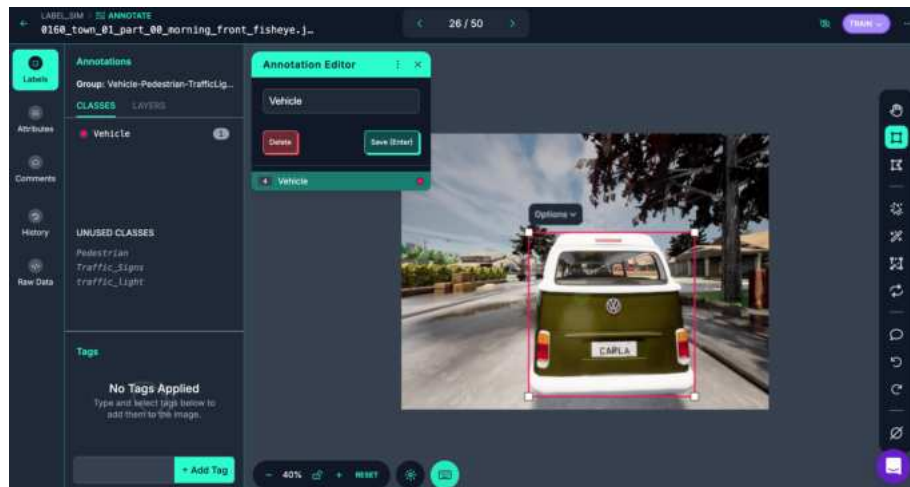


Figure 4.12: Converted Bounding Box using Roboflow.

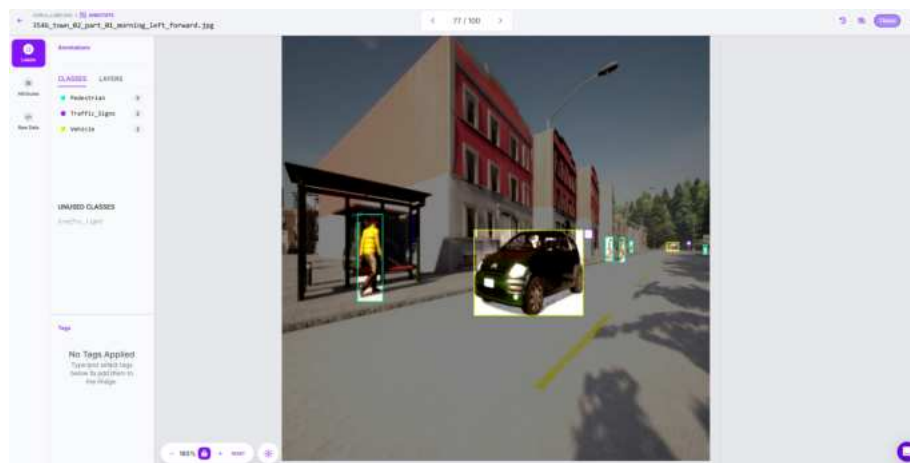


Figure 4.13: Annotated image in Review and Refinement phase using Roboflow framework.

## 4.5 Model Training

The fourth phase involves training multiple configurations and versions of the YOLO model series on the curated dataset to optimize accuracy and efficiency. The performance of the models is evaluated based on two mean Average Precision (mAP) based metrics, with the best-performing model selected based on both high localization precision and classification accuracy.

Figure 4.14 illustrates the methodology used for training the YOLO models. The custom dataset, comprising four classes, is processed using a distributed sampler and fed into the modified YOLO models. The

model architecture includes a frozen pretrained backbone and a reconfigured output layer to accommodate four classes, aligning with the dataset. The output consists of five key components: bounding boxes, segmentation masks, keypoints pose, class probabilities, and oriented boxes [137]. While these outputs can address diverse tasks such as object detection, instance segmentation, pose estimation, and oriented object detection, this project focuses exclusively on 2D object detection. For this purpose, bounding boxes provide spatial localization of objects, while class probabilities assign specific labels, enabling simultaneous object localization and classification.

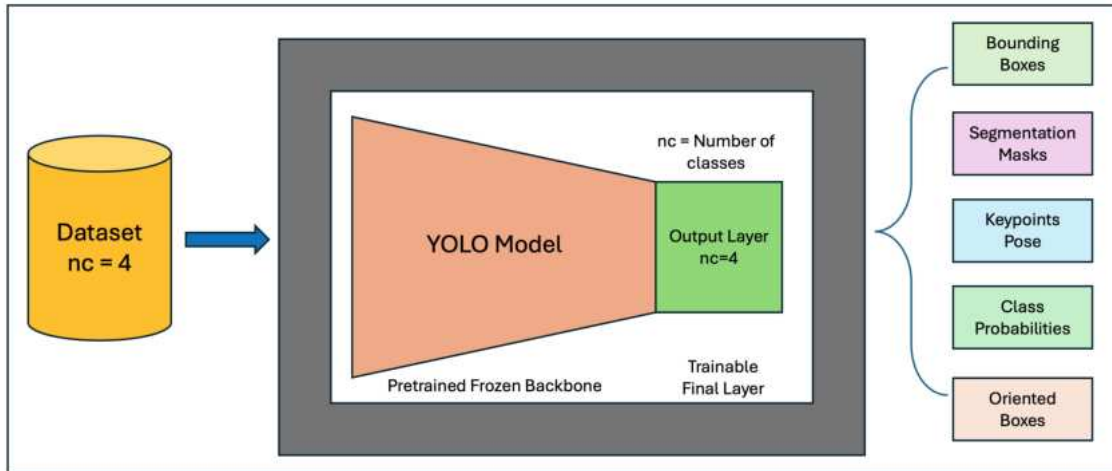


Figure 4.14: Model training using the Ultralytics framework with a focus on YOLO11 model.

The training process was conducted using the Ultralytics framework on a custom dataset split into 70% training (4,485 images), 20% validation (1,274 images), and 10% testing (641 images). The models, ranging from YOLOv8n to YOLO11m, utilized pretrained weights specific to each version, such as “yolo11m.pt,” pretrained on the COCO dataset [153] with 80 classes. The output layer was modified to match the four-class custom dataset, and fine-tuning was performed over 100 epochs. Training employed 4 NVIDIA GPUs (48 GB each) with a distributed setup handled by Distributed Data Parallel (DDP) [154] and a total batch size of 128 (32 per GPU). The Ultralytics framework [155] automatically managed hyperparameters, including optimizer settings and learning rate scheduling, ensuring optimal configurations. Automatic Mixed Precision (AMP) [156] was utilized to enhance computational efficiency.

## 4.6 Real-Time Object Detection

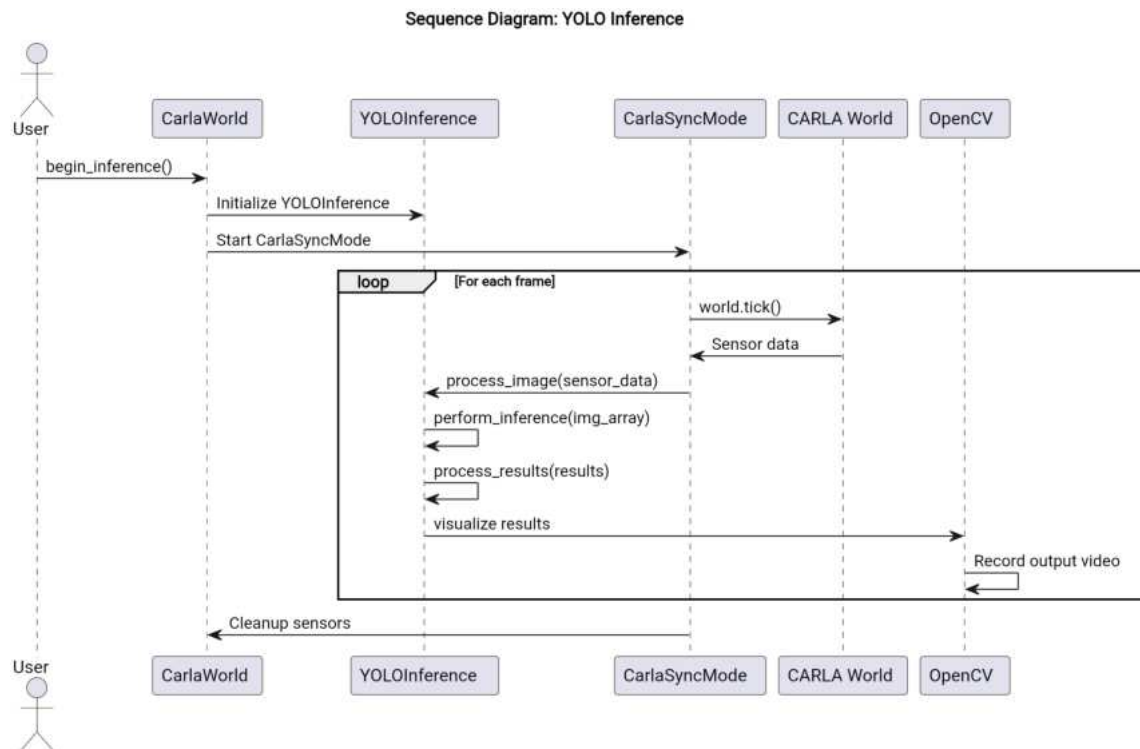


Figure 4.15: Sequence Diagram for YOLO Inference

The final phase involves deploying the best-performing YOLO model for real-time object detection within the CARLA simulation environment. The workflow for inference is depicted in the **Sequence Diagram for YOLO Inference** (Figure 4.15) and the **Activity Diagram for YOLO Object Detection Inference** (Figure 4.16). The system captures synchronized image data from eight RGB cameras mounted on the Tesla Model 3, with images initially recorded at a resolution of 1280x960 pixels. These images are resized to a resolution of 640x640 pixels to match the YOLO model's input requirements. Using the YOLOInference class, the pre-trained YOLO model processes the images in real time, detecting objects and outputting bounding boxes, class labels, and confidence scores.

The results are visualized by overlaying bounding boxes and labels on the images, which are then organized into a 2x4 grid layout for a comprehensive view of the vehicle's surroundings. The processed frames are recorded as an MP4 video using OpenCV's [157] VideoWriter, allowing for post-analysis and performance evaluation. The system operates at a frame rate of 30 FPS, leveraging the *CarlaSyncMode* class to synchronize sensor data with simulation frames and maintain temporal consistency during inference.

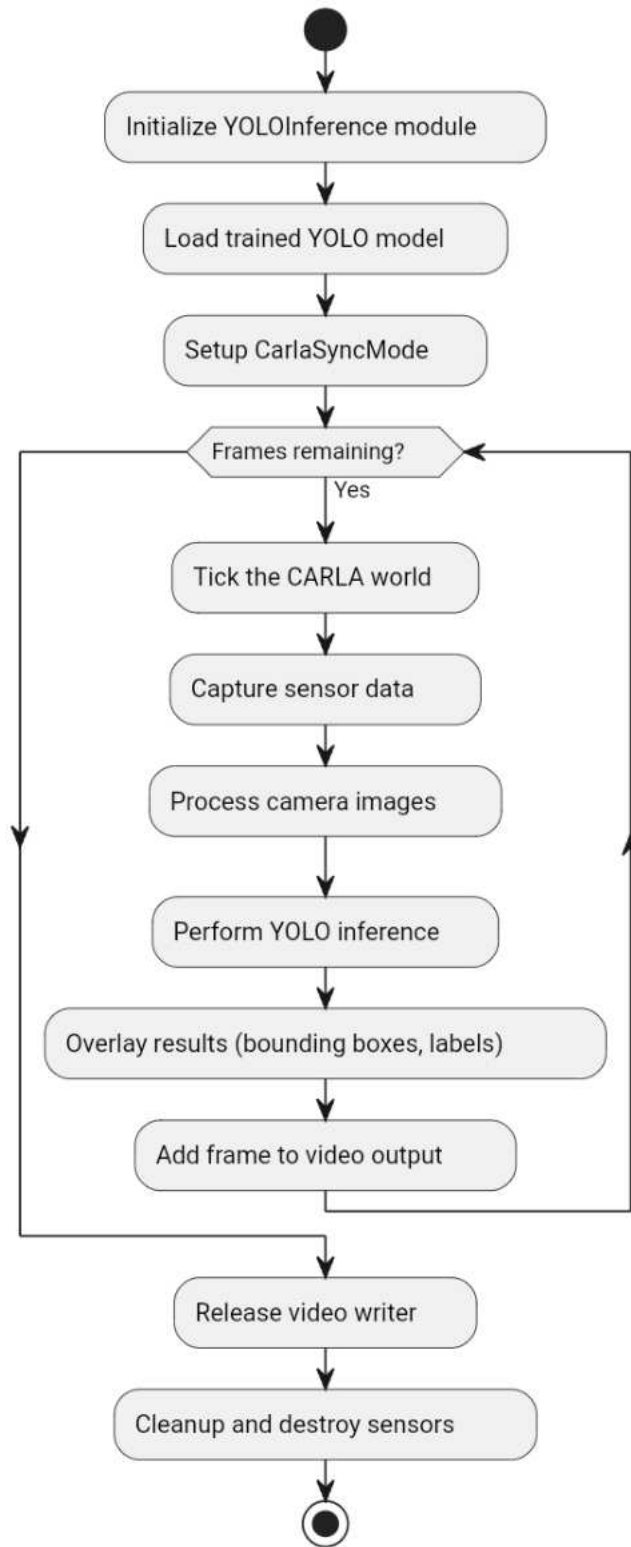


Figure 4.16: Activity Diagram for YOLO Object Detection Inference

## Chapter 5

### Experimental Setup and Evaluation Metrics

This chapter provides a comprehensive overview of the technical framework and resources required to replicate the experiments conducted in this project. It meticulously details the software libraries, packages, operating systems, and computational resources that are essential across different phases of the project, which include simulation setup, dataset creation, exploratory data analysis, and model training. The chapter is organized to facilitate easy understanding of the technical setup and execution steps. It concludes with a discussion of the various evaluation metrics used to analyze the experimental results, ensuring a clear path for both replication and assessment of the findings.

#### 5.1 Overview

The project's workflow is methodologically divided into five stages: setting up the simulation environment, data acquisition, dataset creation, model training, and real-time object detection. This chapter further categorizes the implementation into three parts for clarity: the simulation part (comprising the simulation setup, data collection, and real-time inference), the dataset creation phase (involving data reorganization and labeling), and the Exploratory Data Analysis (EDA) and model training phase.

#### 5.2 Simulation Environment Configuration

The simulation tasks, including data collection and real-time inference, were executed using CARLA version 0.9.13 on an Ubuntu 20.04.6 LTS system. The hardware setup included a single NVIDIA GPU with 8 GB of memory, sufficient for running simulations and YOLO model inferences. Python 3.10 was chosen for its extensive support for deep learning libraries. Key software components used were:

- OpenCV [157] (v4.9.0) for video processing and visualization.
- NumPy (v1.26.2) for numerical operations.
- Pandas (v2.1.4) for data handling.

- Matplotlib (v3.10.0) for visualization.
- PyTorch [158] (v2.2.0) and Torchvision (v0.17.0) for deep learning, with CUDA acceleration (v11.8).
- Ultralytics [155] (v8.3.55) for YOLO model operations.

*Note:* Version compatibility testing is recommended if alternative package versions are used.

### 5.3 Dataset Organization and Labeling

The dataset was reorganized using Python 3.10, with automated labeling performed on a Google Colab NVIDIA L4 GPU instance. The labeling process utilized the Autodistill package [150] and Roboflow framework [159] to automate the annotation of image data, which was then refined manually in Roboflow framework. Key steps included:

1. Mounting Google Drive to access and unzip organised dataset images.
2. Utilizing the Supervision library to verify image loading.
3. Applying GroundedSAM using autodistill-grounded\_sam for initial automated labeling, followed by manual refinement in Roboflow.

This process ensured accurate labels for the training dataset, critical for the subsequent [EDA](#) and model training phase.

### 5.4 EDA and Model Training Environment

A dedicated Python 3.9.19 Conda environment, named `yolov_env`, was created to isolate dependencies and facilitate reproducible research. This environment contained libraries essential for [EDA](#) and model training, including:

- `torch-2.4.1` with `cuda-12.1`, `ultralytics-8.3.13` for model operations.
- `pandas`, `matplotlib`, `seaborn` for data visualization.
- `PyYAML`, `opencv-python`, `numpy`, `scipy`, `torchvision`, `roboflow`, `requests`, `urllib3` for data handling and network operations.

### 5.5 Computational Resources

The model training and [EDA](#) were performed on a high-performance computing server equipped with:

- Two Intel Xeon Gold 5218R CPUs (80 logical cores) and 503 GB of RAM.

- Four NVIDIA RTX A6000 GPUs, each with 48 GB of memory.

This configuration enabled efficient parallel processing and distributed model training, significantly reducing computational time and ensuring robust performance metrics.

## 5.6 Evaluation Metrics

This section outlines the evaluation metrics utilized in assessing the performance of the object detection task. Among the commonly used metrics, Average Precision (AP) and Mean Average Precision (mAP) are the standard for measuring the accuracy of object detection models [160, 161]. In addition to AP, metrics such as Precision, Recall, and F1 Score are employed to provide a comprehensive evaluation of model performance [162]. These metrics rely on the classification of detections into the following categories:

- **True Positives (TP):** Predictions where the detected bounding box sufficiently overlaps with a ground-truth bounding box (based on an Intersection over Union (IoU) threshold) and both belong to the same class [161].
- **False Positives (FP):** Predictions where the detected bounding box does not overlap sufficiently with any ground-truth bounding box, or it overlaps but is assigned to the wrong class [161].
- **False Negatives (FN):** Ground-truth objects that were not detected by the model, either due to missing bounding boxes or incorrect classifications [161].
- **True Negatives (TN):** Predictions where the absence of an object is correctly identified. However, TNs are typically not relevant in object detection, as the focus is on detecting objects [160, 161].

The classification of detections into these categories is based on the concept of Intersection over Union (IoU), which quantifies the overlap between the predicted and ground-truth bounding boxes. IoU serves as the foundation for determining whether a detection is correct or incorrect. By comparing the IoU value to a predefined threshold, detections are classified as follows:

- If the IoU value is greater than or equal to the threshold, the detection is considered correct.
- If the IoU value is below the threshold, the detection is classified as incorrect.

These metrics, in combination with IoU-based classification, form the basis for evaluating object detection models, providing insights into their accuracy, robustness, and generalization capabilities [160].

**Intersection over Union (IoU):** The Intersection over Union (IoU) is a widely used metric for evaluating object detection models. IoU measures the overlap between the predicted or detected bounding box and the ground-truth bounding box, relative to their union. Object detectors not only identify bounding boxes but also classify each one. Therefore, only ground-truth and detected boxes that belong to the same class can be

compared using the Intersection over Union (IoU) metric. This metric is critical for assessing the accuracy of the spatial localization of detected objects. Mathematically, the IoU is defined as:

$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}} = \frac{\text{area}(B_{gt} \cap B_d)}{\text{area}(B_{gt} \cup B_d)}, \quad (5.1)$$

where  $B_d$  represents the detected or predicted bounding box and  $B_{gt}$  denotes the ground-truth bounding box [160]. The numerator calculates the area of overlap between the two bounding boxes, while the denominator computes the total area covered by both boxes combined. A perfect match between the predicted and ground-truth bounding boxes corresponds to  $\text{IoU} = 1$ , indicating complete overlap. Conversely, if the boxes do not intersect,  $\text{IoU} = 0$ . Typically, an IoU threshold is set to determine whether a detection is considered a true positive. For example, a threshold of 0.5 means that the predicted bounding box must overlap at least 50% of the ground-truth bounding box to be classified as correct [161]. Figure 5.1 illustrates the calculation

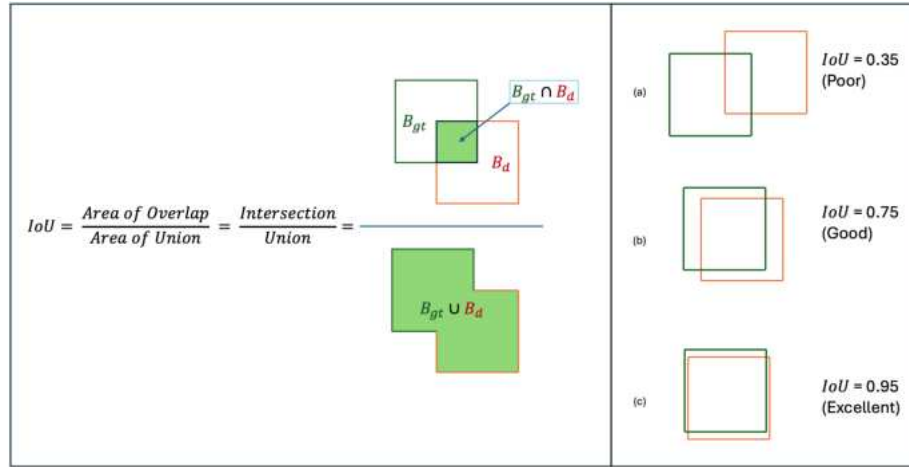


Figure 5.1: Illustration of Intersection over Union (IoU). The overlap between the predicted or detected bounding box (orange) and the ground-truth bounding box (green) is divided by their union to compute the IoU. This figure illustrates three scenarios based on the Intersection over Union (IoU) scores. An IoU score close to zero - case (a), signifies poor detection accuracy. A score around 0.5 or slightly higher - case (b), is indicative of good detection. An IoU score near one, such as 0.95 - case (c), denotes excellent detection performance by the object detector.

of IoU, where the orange and green rectangles represent the predicted and ground-truth bounding boxes, respectively. IoU is based on the Jaccard Index, a coefficient of similarity for two sets of data [163]. This metric plays a pivotal role in object detection by providing a quantitative measure of the agreement between predicted and actual object locations. As stated in Padilla et al. [161], IoU thresholds can be adjusted to make the evaluation more or less strict, with thresholds closer to 1 requiring highly precise predictions. In this project, IoU serves as the foundation for multiple evaluation metrics, such as precision, recall, and mean average precision, which are discussed in subsequent subsections.

### 5.6.1 Precision, Recall, and F1 Score

Precision, Recall, and F1 Score are fundamental metrics for evaluating machine learning models. These metrics provide insights into models ability to correctly identify or classify objects and minimize errors. These metrics are used particularly in contexts where data may be imbalanced or where the cost of false positives and false negatives is significant [164].

**Precision:** Precision measures the proportion of correctly predicted objects (true positives) out of all predicted positive objects (true positives and false positives). It quantifies the model's ability to avoid false positives.

Mathematically, Precision is defined as:

$$\text{Precision (Pr)} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Positives (FP)}}. \quad (5.2)$$

A high precision value indicates that the model produces fewer false positives, focusing on the relevance of its predictions [165, 164, 161]. However, precision alone does not account for missed detections (false negatives), which motivates the need for recall.

**Recall:** Recall measures the proportion of correctly predicted objects (true positives) out of all actual objects in the ground truth (true positives and false negatives). It reflects the model's ability to detect all relevant instances.

Mathematically, Recall is defined as:

$$\text{Recall (Rc)} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Negatives (FN)}}. \quad (5.3)$$

A high recall value indicates that the model successfully detects most of the ground-truth objects, minimizing missed detections. However, it does not penalize false positives, which is why it is often considered alongside precision [165, 164, 161].

**F1 Score:** The F1 Score combines Precision and Recall into a single metric by calculating their harmonic mean. It provides a balanced measure of a model's accuracy, particularly in scenarios where precision and recall exhibit a trade-off.

Mathematically, the F1 Score is defined as:

$$\text{F1 Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}. \quad (5.4)$$

The F1 Score ranges from 0 to 1, with higher values indicating better performance. It is especially useful in imbalanced datasets, where a model might achieve high precision or recall independently but fail

to balance both effectively [165, 164].

### 5.6.2 Metric Curves

Metric curves are pivotal for evaluating YOLO’s performance across varying thresholds <sup>1</sup>. The Precision-Confidence Curve and Recall-Confidence Curve depict how precision and recall, respectively, change as confidence thresholds are adjusted, offering insights into the trade-offs between false positives and false negatives. The F1-Confidence Curve illustrates the threshold where precision and recall are optimally balanced, marking the point of peak overall performance. The Precision-Recall Curve provides a comprehensive view of the interplay between precision and recall, particularly valuable for understanding performance in imbalanced datasets. Complementing these curves, the Confusion Matrix and its Normalized version detail true positives, false positives, true negatives, and false negatives, allowing class-specific performance analysis and comparison across datasets [160, 155, 162].

### 5.6.3 Mean Average Precision (mAP)

#### Average Precision (AP)

Average Precision (AP) is a key evaluation metric in object detection that quantifies the trade-off between precision and recall across varying confidence thresholds. It is computed as the area under the Precision-Recall (P-R) curve and provides a single scalar value summarizing a model’s performance in detecting objects accurately. This metric accounts for both the accuracy of detections (precision) and the model’s ability to find all objects (recall) [162].

**Precision and Recall Formulation:** In object detection, the output consists of bounding boxes, class labels, and confidence scores. These confidence scores are compared against a threshold  $\tau$  to classify detections as positive or negative. Predictions with confidence scores greater than or equal to  $\tau$  are considered positive detections, while those below  $\tau$  are negative.

Assume a dataset with  $G$  ground-truth objects,  $N$  detections, and  $S$  correct detections ( $S \leq G$ ). Let  $n$  represent the index of detections,  $k$  denote confidence thresholds in decreasing order, and  $\tau(k)$  represent the  $k$ -th confidence threshold [161]. Precision and recall are formally defined as:

$$\text{Precision (Pr)}(\tau) = \frac{\sum_{n=1}^S TP_n(\tau)}{\sum_{n=1}^S TP_n(\tau) + \sum_{n=1}^{N-S} FP_n(\tau)} \quad (5.5)$$

$$\text{Recall (Rc)}(\tau) = \frac{\sum_{n=1}^S TP_n(\tau)}{\sum_{n=1}^S TP_n(\tau) + \sum_{n=1}^{G-S} FN_n(\tau)} \quad (5.6)$$

Here:

---

<sup>1</sup>For more details on YOLO performance metrics, refer to the discussion at <https://github.com/ultralytics/ultralytics/issues/7307>.

- $TP_n(\tau)$ : True positives for detection  $n$  at threshold  $\tau$ .
- $FP_n(\tau)$ : False positives for detection  $n$  at threshold  $\tau$ .
- $FN_n(\tau)$ : False negatives for detection  $n$  at threshold  $\tau$ .

Precision measures the proportion of correct detections among all predicted positives, while recall measures the proportion of detected ground-truth objects [161].

**Precision-Recall Curve and Challenges:** The Precision-Recall curve is constructed by plotting precision against recall for varying confidence thresholds  $\tau(k)$ . Ideally, a good object detector maintains high precision as recall increases, resulting in a large area under the curve (AUC). However, in practice, the P-R curve is often non-monotonic, exhibiting a zigzag pattern due to fluctuations in precision at different thresholds. This non-monotonic behavior complicates accurate AUC computation.

**Interpolated Precision for Monotonicity:** To address the non-monotonic behavior, precision is interpolated to ensure monotonicity. The interpolated precision  $Pr_{\text{interp}}(R)$  at recall  $R$  is defined as:

$$Pr_{\text{interp}}(R) = \max_{k | Rc(\tau(k)) \geq R} \{Pr(\tau(k))\}, \quad (5.7)$$

where:

- $\tau(k)$ : Confidence threshold for the  $k$ -th detection, ordered such that  $\tau(k) > \tau(k+1)$ .
- $Rc(\tau(k))$ : Recall value corresponding to threshold  $\tau(k)$ , calculated using Equation 5.6.

This ensures that the resulting precision-recall curve is monotonic, enabling accurate area computation [161].

**Computation of Average Precision:** AP is computed as the area under the interpolated precision-recall curve, sampled at discrete recall values  $R_r(k)$  [161]. Using a Riemann sum, AP is defined as:

$$AP = \sum_{k=0}^K (R_r(k) - R_r(k+1)) Pr_{\text{interp}}(R_r(k)), \quad (5.8)$$

where:

- $R_r(k)$ : Reference recall value at the  $k$ -th position.
- $Pr_{\text{interp}}(R_r(k))$ : Interpolated precision at  $R_r(k)$ .

**Interpolation Methods:** There are two common approaches for computing AP:

- **N-Point Interpolation:** The reference recall values  $R_r(k)$  are equally spaced in the interval  $[0, 1]$ . For example, the COCO challenge uses  $N = 101$  points [166].
- **All-Point Interpolation:** All recall values corresponding to unique confidence thresholds are used. This method, adopted by the Pascal VOC challenge [167], ensures maximum precision.

### Mean Average Precision (mAP)

Mean Average Precision (mAP) extends AP by computing the average of AP values across multiple object classes. It is particularly useful for datasets with multiple object categories, as it provides an aggregated measure of the model's performance [160].

Mathematically, mAP is expressed as:

$$\text{mAP} = \frac{1}{C} \sum_{i=1}^C \text{AP}_i, \quad (5.9)$$

where  $C$  is the total number of classes, and  $\text{AP}_i$  is the Average Precision for the  $i$ -th class [160, 161].

**mAP Across IoU Thresholds:** The mAP metric is often computed across a range of IoU thresholds to provide a more comprehensive evaluation. For example:

- **mAP@[0.5]:** This metric averages AP values computed at 0.5 IoU threshold value.
- **mAP@[0.5:0.95]:** This metric averages AP values computed at IoU thresholds ranging from 0.5 to 0.95, with a step size of 0.05 [161].

**Fitness score:** In YOLOv5, the default fitness function/score is defined as a weighted combination of mAP50 and mAP50-95 metrics. The fitness score in YOLO models is a single value that represents the overall performance of the model, combining multiple evaluation metrics. It is used to assess and compare different model iterations during training and optimization [107].

Mathematically,

$$\text{Fitness Score} = 0.1 * (\text{mAP50}) + 0.9 * (\text{mAP50} - 95) \quad (5.10)$$

where, 0.1 and 0.9 are weights specific to each metric used in equation 5.10 respectively <sup>2</sup>.

---

<sup>2</sup><https://github.com/ultralytics/yolov5/issues/2303>

## Chapter 6

### Results and Evaluation

In Chapters 4 and 5, the methodologies and experimental setup for this research project were outlined. This chapter builds on that foundation by presenting the empirical results and corresponding analyses. It begins with a detailed Exploratory Data Analysis (EDA) of the dataset, offering insights into its structure, class distributions, and challenges relevant to the object detection task. Following this, the evaluation metrics and hyperparameter configurations employed in the experiments are discussed. The chapter concludes with a comprehensive review of the experimental results, incorporating both quantitative metrics and qualitative visualizations to assess the model’s performance and identify areas for potential improvement.

#### 6.1 Data Acquisition and Preparation

The data was generated using a simulated Tesla Model 3 equipped with a comprehensive sensor suite comprising 8 cameras, 12 ultrasonic sensors, and 1 radar. The data collection process was conducted in the CARLA simulator across six different maps (*Town01* to *Town06*), each capturing a diverse range of scenarios. The dataset includes environmental variations such as different times of day (morning, midday, and night) and varying numbers of non-player characters (NPCs) like vehicles and pedestrians.

Table 6.1 summarizes the details of the data collection process, including the number of frames captured per camera, the environment’s conditions, and the total number of images collected. The total data comprises 6,400 images captured over 9 simulation runs. Simulations collecting 50 frames per camera required an average of 10 minutes per run, while those capturing 100 frames per camera took approximately 23 minutes per run. The metadata for the captured data, including sensor IDs and configurations, is stored in CSV files. Figures 6.1, 6.2, and 6.3 illustrate examples of metadata for camera, radar, and ultrasonic sensors, respectively.

Map	Time of Day	NPCs (Vehicles, Walkers)	Frames/Camera	Total Cameras	Total Images
Town01	Night	29, 34	100	8	800
Town01	Morning	43, 32	100	8	800
Town02	Morning	40, 43	50	8	400
Town02	Midday	31, 42	50	8	400
Town03	Night	28, 34	100	8	800
Town04	Night	31, 32	100	8	800
Town05	Night	37, 32	100	8	800
Town05	Morning	40, 43	100	8	800
Town06	Midday	40, 55	100	8	800

Table 6.1: Summary of data collection in the CARLA simulator across different maps, environmental conditions, and NPC configurations.

camera_data										
sensor_id	role_name	captured_frame	frame_timestamp	image_path	width	height	fov	location_x	location_y	location_z
200	front_camera	0	167.25249033000000	sensor_data_brown_01_camera_front_camera1707.png	1280	960	120.0	81.36903333333333	217.832418832432	1.0177320333333333
200	front_camera	1	170.27249033000000	sensor_data_brown_01_camera_front_camera1708.png	1280	960	120.0	81.36903333333333	217.832418832432	1.0177320333333333
200	front_camera	2	173.29249033000000	sensor_data_brown_01_camera_front_camera1709.png	1280	960	120.0	81.36903333333333	217.832418832432	1.0177320333333333
200	front_camera	3	176.31249033000000	sensor_data_brown_01_camera_front_camera1710.png	1280	960	120.0	81.36903333333333	217.832418832432	1.0177320333333333
200	front_camera	4	179.33249033000000	sensor_data_brown_01_camera_front_camera1711.png	1280	960	120.0	81.36903333333333	217.832418832432	1.0177320333333333
200	front_camera	5	182.35249033000000	sensor_data_brown_01_camera_front_camera1712.png	1280	960	120.0	81.36903333333333	217.832418832432	1.0177320333333333
200	front_camera	6	185.37249033000000	sensor_data_brown_01_camera_front_camera1713.png	1280	960	120.0	81.36903333333333	217.832418832432	1.0177320333333333
200	front_camera	7	188.39249033000000	sensor_data_brown_01_camera_front_camera1714.png	1280	960	120.0	81.36903333333333	217.832418832432	1.0177320333333333
200	front_camera	8	191.41249033000000	sensor_data_brown_01_camera_front_camera1715.png	1280	960	120.0	81.36903333333333	217.832418832432	1.0177320333333333
200	front_camera	9	194.43249033000000	sensor_data_brown_01_camera_front_camera1716.png	1280	960	120.0	81.36903333333333	217.832418832432	1.0177320333333333
200	front_camera	10	197.45249033000000	sensor_data_brown_01_camera_front_camera1717.png	1280	960	120.0	81.36903333333333	217.832418832432	1.0177320333333333
200	front_camera	11	200.47249033000000	sensor_data_brown_01_camera_front_camera1718.png	1280	960	120.0	81.36903333333333	217.832418832432	1.0177320333333333
200	front_camera	12	203.49249033000000	sensor_data_brown_01_camera_front_camera1719.png	1280	960	120.0	81.36903333333333	217.832418832432	1.0177320333333333
200	front_camera	13	206.51249033000000	sensor_data_brown_01_camera_front_camera1720.png	1280	960	120.0	81.36903333333333	217.832418832432	1.0177320333333333
200	front_camera	14	209.53249033000000	sensor_data_brown_01_camera_front_camera1721.png	1280	960	120.0	81.36903333333333	217.832418832432	1.0177320333333333
200	front_camera	15	212.55249033000000	sensor_data_brown_01_camera_front_camera1722.png	1280	960	120.0	81.36903333333333	217.832418832432	1.0177320333333333
200	front_camera	16	215.57249033000000	sensor_data_brown_01_camera_front_camera1723.png	1280	960	120.0	81.36903333333333	217.832418832432	1.0177320333333333
200	front_camera	17	218.59249033000000	sensor_data_brown_01_camera_front_camera1724.png	1280	960	120.0	81.36903333333333	217.832418832432	1.0177320333333333
200	front_camera	18	221.61249033000000	sensor_data_brown_01_camera_front_camera1725.png	1280	960	120.0	81.36903333333333	217.832418832432	1.0177320333333333
200	front_camera	19	224.63249033000000	sensor_data_brown_01_camera_front_camera1726.png	1280	960	120.0	81.36903333333333	217.832418832432	1.0177320333333333
200	front_camera	20	227.65249033000000	sensor_data_brown_01_camera_front_camera1727.png	1280	960	120.0	81.36903333333333	217.832418832432	1.0177320333333333
200	front_camera	21	230.67249033000000	sensor_data_brown_01_camera_front_camera1728.png	1280	960	120.0	81.36903333333333	217.832418832432	1.0177320333333333
200	front_camera	22	233.69249033000000	sensor_data_brown_01_camera_front_camera1729.png	1280	960	120.0	81.36903333333333	217.832418832432	1.0177320333333333
200	front_camera	23	236.71249033000000	sensor_data_brown_01_camera_front_camera1730.png	1280	960	120.0	81.36903333333333	217.832418832432	1.0177320333333333
200	front_camera	24	239.73249033000000	sensor_data_brown_01_camera_front_camera1731.png	1280	960	120.0	81.36903333333333	217.832418832432	1.0177320333333333

Figure 6.1: Visualization of the camera metadata file, which organizes camera data for further processing steps.

## 6.2 Exploratory Data Analysis (EDA)

This section presents the exploratory data analysis conducted on the custom dataset curated for training the YOLO models. The dataset is split into three subsets: Train-(70%), Validation-(20%), and Test-(10%). Each subset contains images labeled with four primary classes: *Vehicles*, *Traffic Lights*, *Pedestrians*, and *Traffic Signs*. The analysis highlights label distributions, null image percentages, and class co-occurrence patterns across these splits.

### 6.2.1 Dataset Overview

Table 6.2 summarizes the distribution of images and labels across the dataset splits.

Split	Total Images	Unlabeled Images	Vehicle	Traffic Light	Pedestrian	Traffic Signs
Train	4,485	430 (9.58%)	10,031	13,071	2,227	1,459
Validation	1,274	124 (9.73%)	2,820	3,523	600	434
Test	641	53 (8.26%)	1,377	1,762	332	230

Table 6.2: Distribution of images and labels across Train, Validation, and Test dataset splits.

sensor_id	role_name	captured_frame	velocity	altitude	azimuth	depth
206	front_radar	0	-0.0	-0.0062058172188699245	0.033872995525598526	19.578081130981445
206	front_radar	0	-0.0	-0.021314790472388268	0.256797730922699	9.46497917175293
206	front_radar	0	-3.9448998734314955e-08	0.02429826557636261	-0.100291408598423	0.3331710398197174
206	front_radar	0	-0.0	-0.02906625345349312	0.012709475122392178	6.941241264343262
206	front_radar	0	-4.749589521679809e-08	0.02908201515674591	-0.18346811830997467	0.2877475619316101
206	front_radar	0	-0.0	-0.02714630402624607	0.004670689348131418	7.432023525238037
206	front_radar	0	0.0	0.0020380255300551653	0.03047478012740612	34.01275634765625
206	front_radar	0	-6.496510707165726e-08	0.039420146495103836	-0.09464540332555771	0.22123120725154877
206	front_radar	0	0.0	0.0016340280417352915	0.005097811575978994	115.85770416259766
206	front_radar	0	-0.0	-0.023187197744846344	-0.006137609947472811	8.700682640075684
206	front_radar	0	-7.034369087222103e-09	0.00501371081918478	0.14544084668159485	0.9424766302108765
206	front_radar	0	0.0	0.0031064630020409822	-0.2539791166782379	94.65534973144531
206	front_radar	0	-0.0	-0.009612152352929115	-0.09091591835021973	13.854597091674805
206	front_radar	0	-1.21470025149506e-08	0.008121145889163017	-0.09353453665971756	0.7263563275337219
206	front_radar	0	-5.1042817972302146e-08	0.031110621988773346	0.09566483646631241	0.27130791544914246
206	front_radar	0	-7.992931649880575e-09	0.005648151505738497	-0.044032882899045944	0.8848145604133606
206	front_radar	0	-6.58643770634626e-08	0.03989305719733238	0.10117986053228378	0.21894925832748413
206	front_radar	0	-5.969620797685593e-09	0.004460234194993973	-0.08803833276033401	0.9908952116966248

Figure 6.2: Visualization of the radar metadata file, which organizes radar data for further processing steps.

### 6.2.2 Dataset Label Analysis

This subsection presents a detailed analysis of the training dataset using two visualizations: the class distribution and label distribution heatmaps (Figure 6.4) and the label correlogram (Figure 6.5). These visualizations provide insights into the distribution and relationships of object annotations across different classes and attributes.

The class distribution bar chart in the top left of Figure 6.4 highlights a significant imbalance across the four object classes. The *traffic\_light* class has the highest prevalence, with over 12,000 instances-(precisely 13,071 instances, refer Table 6.2), followed by the *vehicle* class. In contrast, the *Pedestrian* and *Traffic\_Signs* classes are underrepresented, which may lead to biased predictions favoring the dominant classes. The object center heatmap, shown in the top right of Figure 6.4, indicates that object centers are primarily concentrated near the center of the images, with normalized coordinates clustered around (0.5, 0.5). The 2D histogram of object center coordinates in the bottom left of Figure 6.4 reinforces this pattern, highlighting a central bias in the spatial distribution of objects. Finally, the scatter plot of width versus height, located in the bottom right of Figure 6.4, shows that the majority of objects have small bounding boxes, with normalized dimensions predominantly below 0.2, while larger objects are less frequent.

The label correlogram in Figure 6.5 provides a detailed overview of the pairwise relationships between label attributes (*x*, *y*, *width*, and *height*) and their individual distributions. The diagonal histograms in Figure 6.5 illustrate the distribution of individual attributes, with *x* and *y* showing a relatively uniform spread across the image but with slight peaks near the center. The histograms for *width* and *height* are heavily skewed toward smaller values, indicating the dominance of small objects in the dataset. The scatter plots below the diagonal show the pairwise relationships. The *x* versus *y* scatter plot confirms the central cluster-

sensor_id	role_name	captured_frame	depth
208	front_ultrasonic_2	0	1.1830233335494995
208	front_ultrasonic_2	0	1.147486925125122
208	front_ultrasonic_2	0	1.022645354270935
208	front_ultrasonic_2	0	1.048068642616272
208	front_ultrasonic_2	0	1.1983758211135864
208	front_ultrasonic_2	0	1.090689778327942
208	front_ultrasonic_2	0	1.1546086072921753
208	front_ultrasonic_2	0	1.3177372217178345
208	front_ultrasonic_2	0	1.159629464149475
208	front_ultrasonic_2	0	1.1408143043518066
208	front_ultrasonic_2	0	1.1691155433654785
208	front_ultrasonic_2	0	1.1090335845947266
208	front_ultrasonic_2	0	0.2563400864601135
208	front_ultrasonic_2	0	1.3352532386779785
208	front_ultrasonic_2	0	1.3102996349334717
208	front_ultrasonic_2	0	0.3707808554172516
208	front_ultrasonic_2	0	1.1191096305847168

Figure 6.3: Visualization of the ultrasonic sensor metadata file, which organizes ultrasonic data for further processing steps.

ing of object centers, while the *width* versus *height* scatter plot reveals a triangular relationship, with smaller dimensions dominating the dataset.

These observations highlight key characteristics of the dataset. The imbalance in class frequencies suggests the need for strategies such as data augmentation, oversampling, or class weighting to mitigate potential biases in model training. The central clustering of object centers, as observed in the heatmap and scatter plots, may limit the ability of the model to generalize to edge-located objects, indicating the importance of diversifying object placement during dataset preparation. The prevalence of small bounding boxes suggests that anchor box configurations and data augmentation techniques should prioritize the detection of objects across a range of scales.

### 6.2.3 Label Distribution and Null Images

The dataset exhibits significant class imbalance as discussed in previous subsection, with *Traffic Light* being the most frequent class and *Traffic Signs* the least frequent across all splits. Additionally, a proportion of images in each split are unlabeled, as summarized in Table 6.2. These null images aim to simulate real-world conditions where some images may not contain any detectable objects, enhancing the model's generalization. Figures 6.6a, 6.6c, 6.6e illustrates the label distributions across all images in all splits of the dataset, while figures 6.6b, 6.6d, 6.6f, 6.7d shows the count and percentage of unlabeled images across splits of the dataset.

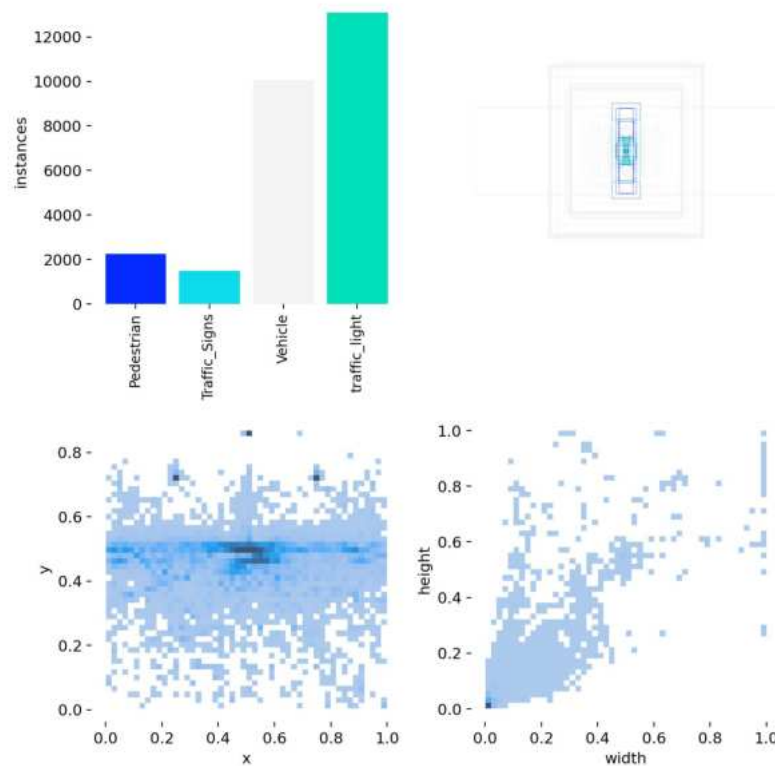


Figure 6.4: Class distribution, object center heatmap, and object scale distributions for the training dataset.

#### 6.2.4 Class Co-occurrence Analysis

The *Class Co-occurrence Matrix* provides insights into how frequently different classes appear together within the same images. Each matrix cell  $(i, j)$  represents the number of images containing both class  $i$  and class  $j$ , while the diagonal elements represent the frequency of each class appearing individually [168]. Figures 6.7a, 6.7b, and 6.7c display the co-occurrence matrices for the Train, Validation, and Test splits, respectively.

Key insights from the co-occurrence analysis:

- **High Co-occurrence:** *Vehicles* and *Traffic Lights* exhibit the highest co-occurrence across all splits, reflecting realistic traffic environments.
- **Moderate Co-occurrence:** *Pedestrians* often co-occur with *Vehicles* and *Traffic Lights*, particularly in urban crosswalk settings.
- **Low Co-occurrence:** *Traffic Signs* have the least co-occurrence with *Pedestrians*, indicating fewer pedestrian-centric scenes containing road signage.

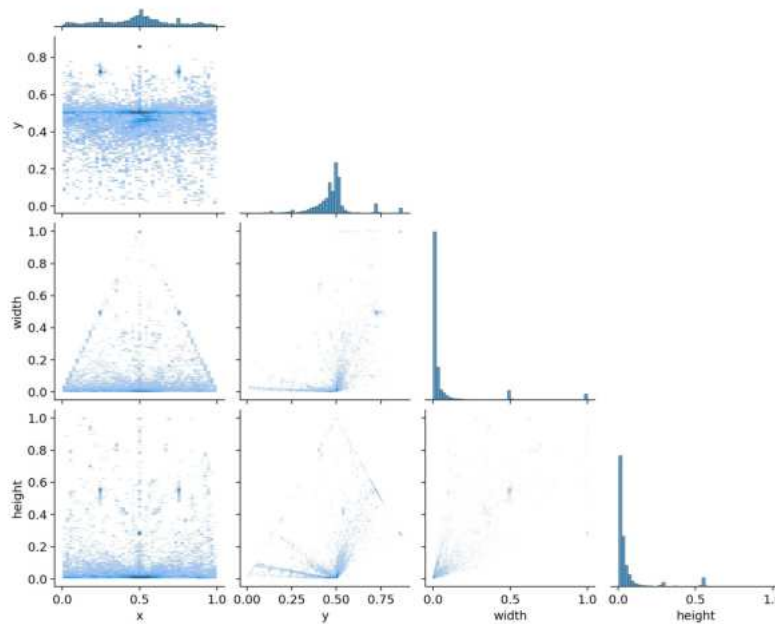


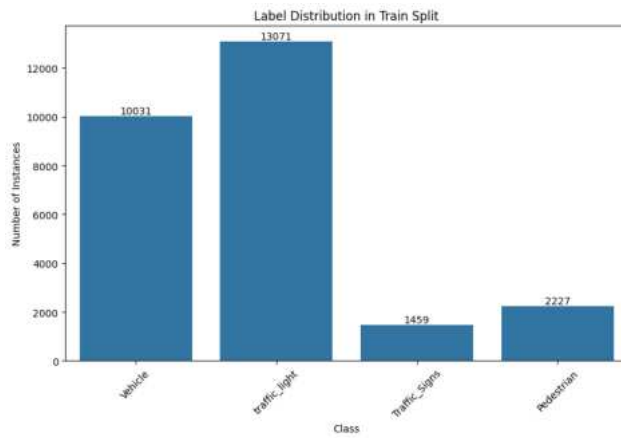
Figure 6.5: Label correlogram showing pairwise relationships and distributions of label attributes.

### 6.2.5 Insights and Implications

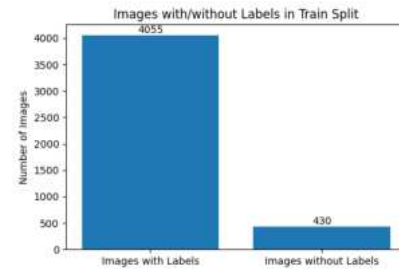
- **Class Imbalance:** The dataset is imbalanced, with *Traffic Lights* overrepresented and *Traffic Signs* underrepresented. This imbalance may lead to biased predictions and requires mitigation strategies like data augmentation or class weighting.
- **Null Images:** The inclusion of null images aligns with real-world scenarios, potentially improving model robustness.
- **Class Relationships:** Strong co-occurrence patterns between *Vehicles* and *Traffic Lights* emphasize the need for context-aware models.

## 6.3 Quantitative Results

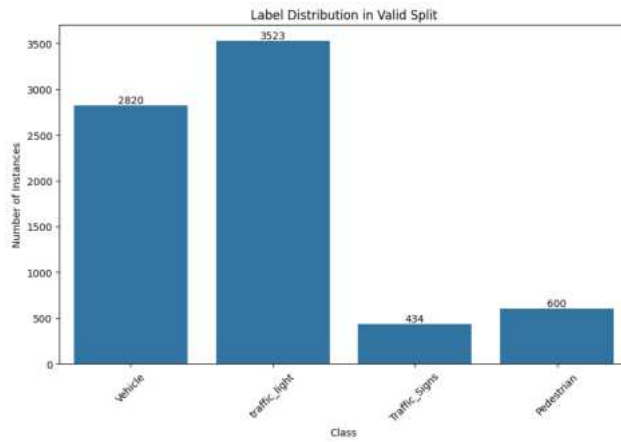
This section presents the results of 12 experiments conducted using the Ultralytics package on the curated custom dataset for the 2D object detection task. The experiments aimed to evaluate the performance of 12



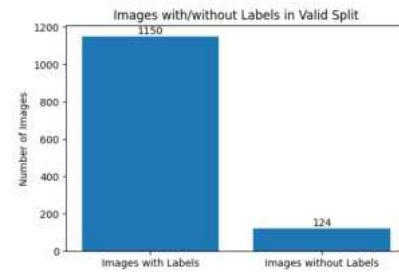
(a) Label distribution in Train split



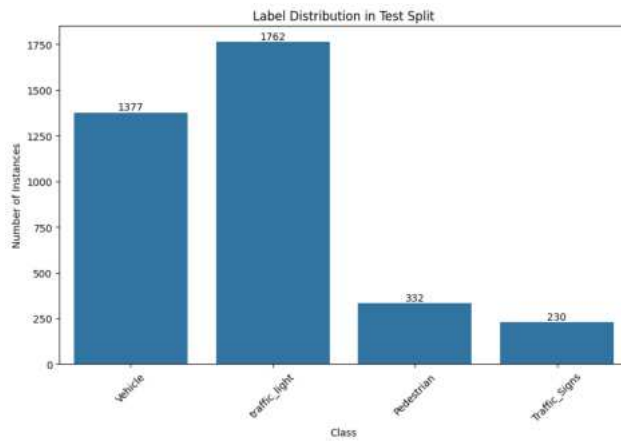
(b) Images with and without labels in Train split



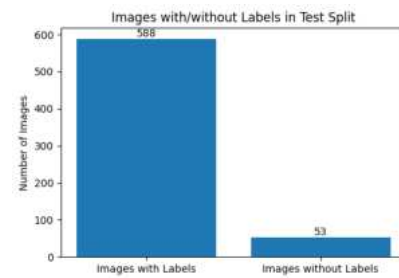
(c) Label distribution in Validation split



(d) Images with and without labels in Validation split



(e) Label distribution in Test split



(f) Images with and without labels in Test split

Figure 6.6: Visual representation of label distributions and the presence of unlabeled images across dataset splits.

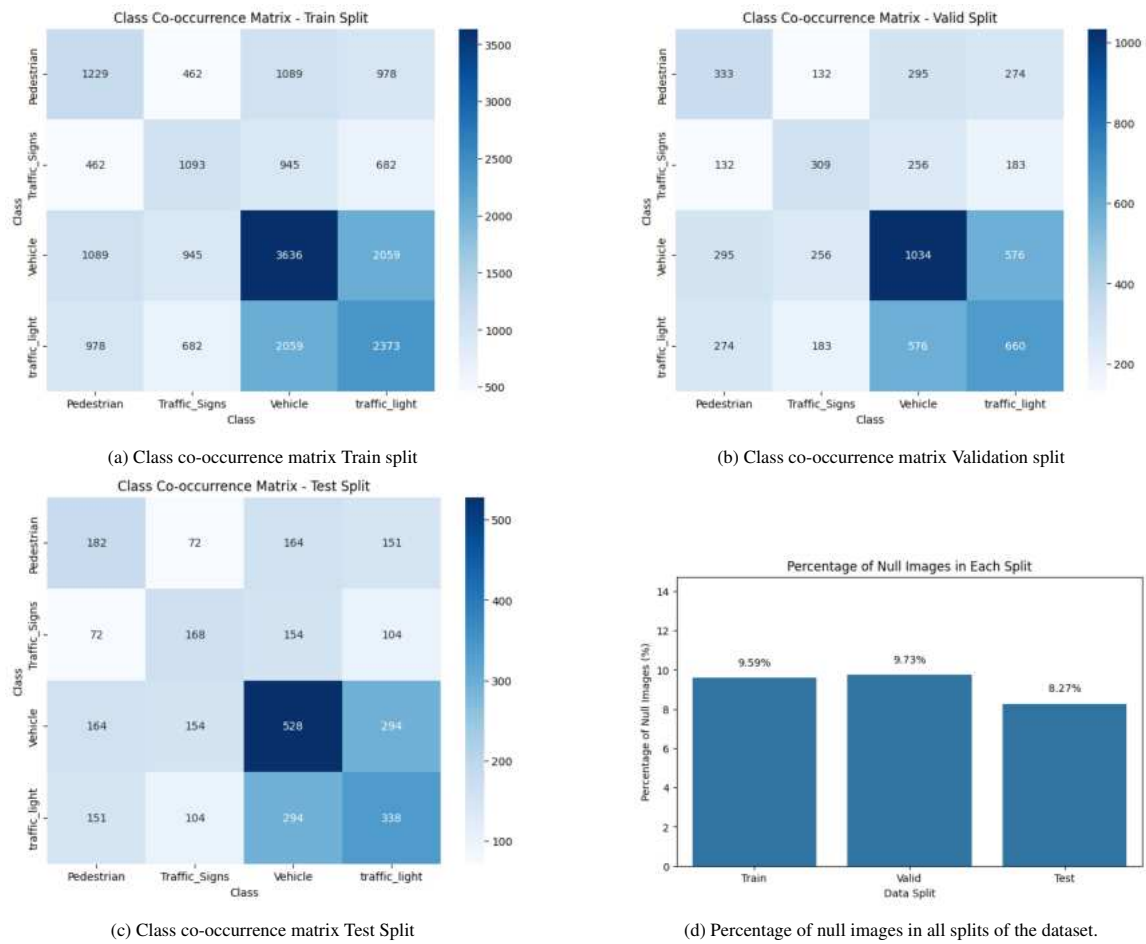


Figure 6.7: Class co-occurrence matrices for Train, Validation, and Test splits and percentage of unlabeled images across splits.

YOLO model configurations across training, validation, and testing phases and to find the best performing model.

### 6.3.1 Overview of Training and Testing

The models were trained for 100 epochs using distributed computing across four NVIDIA RTX A6000 GPUs, each equipped with 48 GB of memory, as described in Chapter 5, section 5.5. Each GPU handled an effective batch size of 32, resulting in a total batch size of 128 for the experiments. During testing, a single NVIDIA RTX A6000 GPU was utilized. The training process leveraged the Ultralytics YOLO framework with the following hyperparameters:

- **Optimizer:** AdamW [169], with a learning rate of 0.000714, momentum of 0.9, and weight decay of 0.001.
- **Batch Size:** 128 across all GPUs combined.
- **Image Size:** All images were resized to  $640 \times 640$  pixels.
- **Early Stopping:** Patience was set to 10 epochs to prevent overfitting and save computation.
- **Pretrained Weights:** The models were initialized with pretrained weights, to accelerate convergence and improve performance.
- **Automatic Mixed Precision (AMP) [156]:** Enabled to optimize memory usage and speed during training.

Validation was performed during training to monitor performance, while testing was conducted separately to evaluate the generalizability of the models. The following commands demonstrate the pseudocode for training and testing using the Ultralytics package, incorporating key parameters that define the dataset configuration, computational setup, and evaluation thresholds.

- **Training Command:**

```
model.train(data="data.yaml", epochs=100, batch=128, imgsz=640,  
            patience=10, optimizer="AdamW", device=[0,1,2,3], amp=True)
```

- **Testing Command:**

```
model = YOLO("/last.pt")  
model.val(data="data.yaml", batch=128, imgsz=640, conf=0.25,  
          iou=0.6, device=0, split="test")
```

The ‘data’ parameter specifies a YAML configuration file that contains dataset-specific settings, such as the paths to training and validation data, class labels, and the total number of classes. For distributed training, the ‘device’ parameter is assigned as a list (e.g., ‘[0, 1, 2, 3]’), indicating that the training is performed in distributed mode across multiple GPUs. This setup configures the ‘CUDA\_VISIBLE\_DEVICES’ environment variable to make GPUs with IDs 0, 1, 2, and 3 available for parallel computation across four GPUs, enhancing training efficiency.

To optimize memory usage and accelerate training, [AMP](#) [156] is enabled by setting the ‘amp’ parameter to ‘True’. This allows a combination of single-precision and half-precision computations during training, reducing memory requirements while maintaining model accuracy. During testing, several key parameters are used to ensure robust evaluation. The ‘conf’ parameter sets the minimum confidence threshold for detections, where detections with confidence scores below the threshold (e.g., ‘conf=0.25’) are discarded. The ‘iou’ parameter defines the Intersection Over Union (IoU) threshold for Non-Maximum Suppression (NMS) [170], which helps reduce duplicate detections by suppressing overlapping bounding boxes with IoU values greater than set threshold value, for example 0.6. The ‘split’ parameter specifies the dataset split (e.g., ‘test’) to be used during testing, and inference is performed on a single GPU, defined by setting ‘device=0’. This configuration ensures efficient resource utilization during training and precise evaluation during testing. The use of distributed computing and advanced optimization techniques, such as [AMP](#), aligns well with the requirements of 2-Dimensional (2D) object detection tasks, providing a balance between computational efficiency and model performance. The performance of each model was evaluated using key metrics, including Precision, Recall, mAP(50), and mAP(50-95). For validation and testing phases, inference speed per image (measured in milliseconds) was also recorded to assess computational efficiency and suitability for real-time applications. These metrics allow a comprehensive comparison of model performance, highlighting the trade-off between accuracy and speed.

Tables 6.3, 6.4, and 6.7 summarize the training, validation, and testing results, respectively, for all 12 YOLO model configurations. Each of the three tables lists the model names and their corresponding performance results. The class type ‘all’ encompasses all four classes from the custom dataset: Vehicle, Pedestrian, Traffic Light, and Traffic Signs. The numeric values in the model names indicate the generation of the YOLO model; for example, ‘8’ represents the eighth variant, while ‘11’ denotes the 11th and the latest generation at the time this report was composed. The naming convention for models categorizes them based on their size and complexity. The categories include: ‘n’ for nano models, ‘t’ for tiny models, ‘s’ for small models, and ‘m’ for medium size models. In the specific case of YOLOv9, the model lineup includes a tiny model variant, denoted as ‘t’, instead of a nano model, which is not offered for this version. Notably, the nano and tiny models are similar in terms of their specifications and capabilities.

### 6.3.2 Training Results

Table 6.3 shows the performance metrics achieved during the training phase, where YOLOv8m, YOLOv9m and YOLO11m are top-performing models across various benchmarks. Models ending in “m” (*medium*)

generally outperform their smaller counterparts, such as “s” (*small*) or “n” (*nano*), due to their more complex architectures. Among the smallest models, YOLOv8n demonstrates superior performance compared to its counterparts. Among the medium size models, YOLOv9m exhibits higher precision (0.927 vs. 0.914), slightly lower recall (0.694 vs. 0.712), approximately equal performance metric mAP50 (0.83 vs. 0.837) and second best mAP50-95 (0.653 vs. 0.66) when compared to YOLOv8m. These results indicate that YOLOv8m has an edge over YOLOv9m and YOLO11m in training metrics.

Similarly, YOLOv8s outperforms YOLO11s in precision (0.907 vs. 0.903) and mAP50-95 (0.629 vs. 0.627), although YOLO11s achieves slightly higher recall (0.687 vs. 0.678) and mAP50 (0.822 vs. 0.817). Overall, YOLOv8s presents a better balance of performance metrics. Furthermore, YOLOv9m surpasses YOLOv9s in precision (0.927 vs. 0.901), recall (0.694 vs. 0.665), mAP50 (0.83 vs. 0.809), and mAP50-95 (0.653 vs. 0.617), establishing itself as the better-performing model. On the other hand, YOLOv10n underperforms in comparison to YOLOv8n and YOLOv11n in nearly all metrics. For example, YOLOv8n exhibits higher precision (0.899 vs. 0.833), recall (0.625 vs. 0.603), and mAP50-95 (0.583 vs. 0.555). These findings confirm that among the smallest models, YOLOv8n achieves the best performance.

Figure 6.8 provides a comprehensive view of the training and validation loss curves for the YOLO11m model, along with validation metrics including precision, recall, mAP(50), and mAP(50-95). The figure demonstrates a clear convergence of the loss curves, with both training and validation losses decreasing steadily over the 100 epochs. Concurrently, the validation metrics show an increasing trend, indicating improved model performance and effective learning. These trends emphasize the optimization and convergence of YOLO11m during training. The three losses—Box, DFL, and Classification—are integral to calculating the total loss in YOLO11m. The box loss corresponds to the model’s ability to accurately locate objects within their bounding boxes. The DFL (Distribution Focal Loss) is designed to address class imbalance during object detection, ensuring that underrepresented classes are effectively learned. Meanwhile, the classification loss ensures that detected objects are correctly classified into their respective categories. Together, these losses guide the model towards accurate localization, classification, and balance across all classes in the dataset [130, 137, 171, 172, 155].

While these results highlight strong training performance, they do not fully account for the model’s generalization ability, which necessitates further validation and testing evaluations.

### 6.3.3 Validation Results

Table 6.4 provides the validation results for all 12 YOLO models. The best-performing models in terms of mAP(50-95) are YOLO11m (0.529), YOLOv8m (0.525), and YOLOv9m (0.521). These models also show competitive mAP(50) values, confirming their robustness during validation.

At an IoU threshold of 0.6 and a confidence threshold of 0.25, the following curves depicted in figure 6.9 were obtained on the validation set. The precision-confidence curve (6.9a) plots precision (y-axis) against confidence (x-axis). The legend entry “all classes 1.00 at 1.00” indicates perfect precision (100%) at the highest confidence threshold of 1.00, which is uncommon and may suggest overfitting. In the recall-

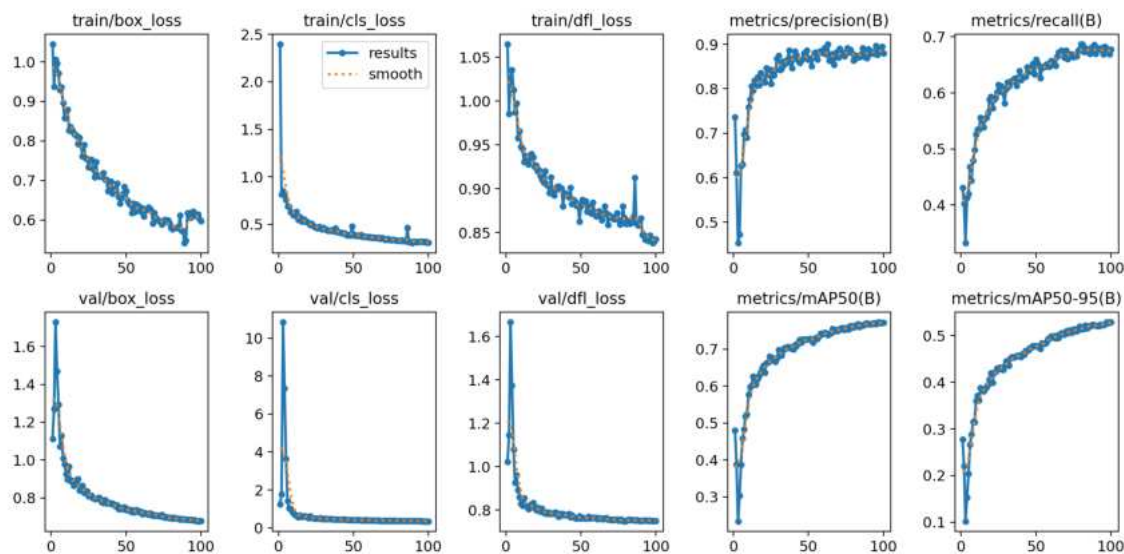


Figure 6.8: Training and Validation Loss Curves for YOLO11m on our Custom Dataset, Alongside Validation Metrics (Precision, Recall, mAP@50, mAP@50-95). The loss curves exhibit a decreasing and converging trend, while the metrics demonstrate an increasing trend with convergence, indicating model optimization and optimal performance by the 100th epoch.

Model Name	Epoch	Precision $\uparrow$	Recall $\uparrow$	mAP50 $\uparrow$	mAP50-95 $\uparrow$	Fitness Score $\uparrow$
YOLOv8n	100	0.899	0.625	0.786	0.583	0.6033
YOLOv8s	97	0.907	0.678	0.817	0.629	0.6478
YOLOv8m	100	<b>0.914</b>	<b>0.712</b>	<b>0.837</b>	<b>0.660</b>	<b>0.6777</b>
YOLOv9t	90	0.880	0.602	0.768	0.567	0.5871
YOLOv9s	100	0.901	0.665	0.809	0.617	0.6362
YOLOv9m	98	<b>0.927</b>	0.694	0.830	0.653	<b>0.6707</b>
YOLOv10n	99	0.833	0.603	0.752	0.555	0.5747
YOLOv10s	99	0.885	0.672	0.808	0.618	0.6370
YOLOv10m	99	0.893	0.684	0.816	0.638	0.6558
YOLO11n	98	0.884	0.617	0.777	0.573	0.5934
YOLO11s	100	0.903	0.687	0.822	0.627	0.6465
YOLO11m	99	<b>0.913</b>	<b>0.711</b>	<b>0.836</b>	0.652	<b>0.6704</b>

Table 6.3: Model specific training results at IoU of 0.6 and confidence of 0.25 for all classes on the train split of custom dataset with 4485 training images, showing Precision, Recall, mAP(50), mAP(50-95) and Fitness Score for model comparison.

confidence curve (6.9b), “all classes 0.79 at 0.000” shows a recall of 79% at the lowest confidence threshold, reflecting the model’s ability to capture most true positives but with a potential increase in false positives.

The precision-recall curve (6.9c) reveals a mean average precision (mAP) of 77.3% across all four classes at an IoU threshold of 0.5. In the F1-confidence curve (6.9d), the peak F1 score of 76% is achieved at a confidence threshold of 0.317, indicating the optimal balance between precision and recall. This optimal confidence threshold of 0.317 highlights the importance of testing the YOLO11m model at this value, as discussed in the next section.

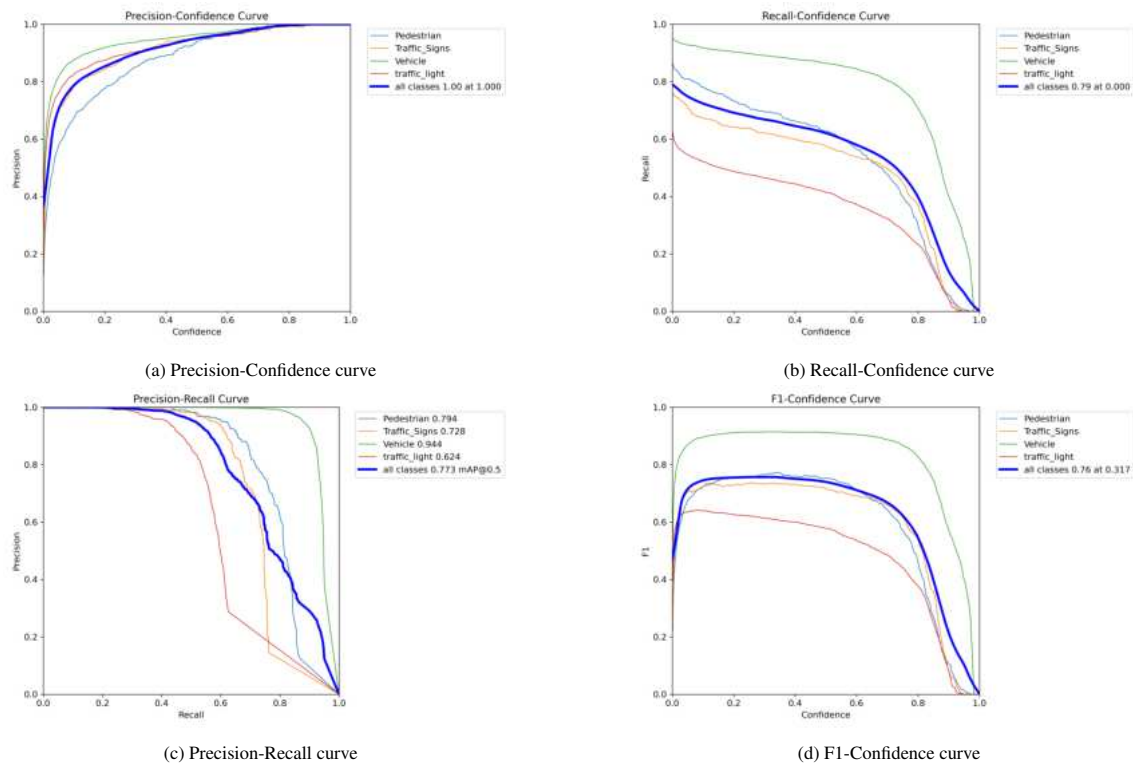


Figure 6.9: Performance evaluation of the YOLO11m model on the validation dataset, depicted through Precision-Confidence, Recall-Confidence, Precision-Recall, and F1-Confidence curves.

Model	Class	Epoch	Dataset Split	Precision $\uparrow$	Recall $\uparrow$	mAP(50) $\uparrow$	mAP(50-95) $\uparrow$
YOLOv8n	all	100	Validation	0.867	0.602	0.681	0.442
YOLOv8s	all	97	Validation	0.870	0.653	0.739	0.496
YOLOv8m	all	100	Validation	0.892	0.672	<b>0.771</b>	<b>0.525</b>
YOLOv9t	all	90	Validation	0.859	0.590	0.667	0.434
YOLOv9s	all	100	Validation	0.872	0.643	0.726	0.486
YOLOv9m	all	98	Validation	0.853	0.685	<b>0.764</b>	<b>0.521</b>
YOLOv10n	all	99	Validation	0.829	0.583	0.661	0.433
YOLOv10s	all	99	Validation	0.867	0.650	0.737	0.494
YOLOv10m	all	99	Validation	0.858	0.665	0.757	0.518
YOLO11n	all	98	Validation	0.870	0.597	0.678	0.438
YOLO11s	all	100	Validation	0.884	0.650	0.737	0.499
YOLO11m	all	99	Validation	0.895	0.670	<b>0.773</b>	<b>0.529</b>

Table 6.4: Validation results for YOLO models at IoU=0.6 and confidence=0.25, showing Precision, Recall, mAP(50), and mAP(50-95). The  $\uparrow$  symbol indicates that a higher value is preferable. The epoch number corresponds to the best model.

### 6.3.4 Testing Results

An IoU threshold of 0.6 and a fixed confidence score of 0.25 were used during training and validation. For further evaluation, optimal confidence values were derived from F1-Confidence curves on the validation set using inference on the best model checkpoint in each case. These values were applied during testing at an IoU threshold of 0.6 to ensure consistent comparisons across models. Table 6.5 summarizes these results.

Model	Epoch	Class	Dataset Split	IoU	Confidence	Precision	Recall	mAP(50)	mAP(50-95)
YOLOv8n	100	all	Test	0.6	0.247	0.856	0.595	0.753	0.534
YOLOv8s	97	all	Test	0.6	0.241	0.850	0.636	0.776	0.563
YOLOv8m	100	all	Test	0.6	0.315	<b>0.906</b>	0.652	0.798	<b>0.589</b>
YOLOv9t	90	all	Test	0.6	0.290	0.867	0.550	0.733	0.527
YOLOv9s	100	all	Test	0.6	0.266	0.886	0.616	0.775	0.560
YOLOv9m	98	all	Test	0.6	0.156	0.853	0.674	<b>0.799</b>	0.580
YOLOv10n	99	all	Test	0.6	0.264	0.812	0.562	0.719	0.518
YOLOv10s	99	all	Test	0.6	0.233	0.857	0.613	0.765	0.558
YOLOv10m	99	all	Test	0.6	0.320	0.879	0.618	0.773	0.575
YOLO11n	98	all	Test	0.6	0.280	0.867	0.572	0.745	0.532
YOLO11s	100	all	Test	0.6	0.304	0.899	0.627	0.785	0.576
YOLO11m	99	all	Test	0.6	0.317	0.901	<b>0.653</b>	<b>0.799</b>	<b>0.588</b>

Table 6.5: Comparison of YOLO models based on Precision, Recall, mAP(50), and mAP(50-95) at an IoU threshold of 0.6, optimal confidence scores and best model checkpoint for inference. The evaluation was performed on the test dataset consisting of 641 images.

The analysis of Table 6.5 provides valuable insights into the performance of the YOLO models under evaluation, with the mAP(50-95) metric serving as the primary indicator for precise classification and localization. Among the models, **YOLOv8m** stands out as the best-performing model, achieving the highest mAP(50-95) of 0.589, demonstrating its consistent ability to predict accurate bounding boxes across various IoU thresholds. This performance is further supported by its strong precision (0.906) and recall (0.652). For a balanced model, **YOLO11m** is a close contender, with an mAP(50-95) of 0.588, complemented by high precision (0.901) and slightly better recall (0.653). The negligible 0.17% difference in mAP(50-95) between YOLO11m and YOLOv8m suggests that YOLO11m offers equivalent overall performance, making it partic-

ularly suitable for applications prioritizing higher recall, which translates to fewer false negatives and more robust detection of positive instances. In contrast, the weakest model based on the mAP(50-95) metric is **YOLOv10n**, with a score of 0.518. This lower score aligns with its relatively weaker precision (0.812) and recall (0.562), highlighting its limited effectiveness in tasks requiring accurate detection and localization. In summary, YOLOv8m excels as the top-performing model, YOLO11m provides a well-balanced alternative with strong recall, and YOLOv10n demonstrates the lowest performance among the evaluated models.

Additionally, two supplementary testing scenarios were evaluated on the last model checkpoint from the 100th epoch of each YOLO model: (A) IoU: 0.6 with a fixed confidence of 0.25, and (B) IoU: 0.5 with optimal confidence values from the F1-Confidence curve. Scenario (A) provides a consistent baseline, while Scenario (B) reflects each model's balanced performance by leveraging optimal confidence thresholds. Detailed results for these scenarios are presented in the section 6.4.

**Confusion Matrix:** The confusion matrix shown in figure 6.10 illustrates the performance of the YOLOv8m model on the test dataset across four classes: Pedestrian, Traffic\_Signs, Vehicle, and Traffic\_Light. Among these, the Vehicle class achieved the highest accuracy with 1227 true positives, highlighting the model's strong detection capabilities for this category. However, misclassifications are notable, particularly with Traffic\_Light (979), Vehicle (150), Pedestrian (99), and Traffic\_Signs (91) being frequently confused with the Background, reflecting challenges in distinguishing these objects from non-object regions. While Traffic\_Lights showed good detection performance with 783 true positives, they were also significantly misclassified as Background (979). These results indicate that, despite strong detection performance of YOLOv8m model for Vehicles, improvements are needed in detecting smaller or less distinct classes such as Traffic\_Lights, Pedestrians, and Traffic\_Signs, as well as in minimizing confusion with the Background.

### 6.3.5 Real-Time Performance

The analysis of the YOLO models reveals key insights into their performance for real-time applications, with a focus on the tradeoff between mAP(50-95) accuracy, model size, and speed. Among the models, **YOLOv8m** and **YOLO11m** are nearly identical in terms of mAP(50-95), scoring 0.589 and 0.588 respectively, demonstrating superior classification and localization capabilities. However, YOLOv8m has a slightly higher speed (8.3 ms) compared to YOLO11m (8.5 ms), which translates to marginally better real-time performance, though the difference is negligible. These two models represent the most accurate choices, but their higher latency makes them less suitable for extremely time-sensitive applications.

The **best-performing model** in terms of mAP(50-95) is YOLOv8m, reflecting its ability to deliver precise results, albeit at the cost of higher processing time. Conversely, the **least-performing model** is YOLOv10n, with an mAP(50-95) of 0.518, despite being the fastest model with a speed of 4.3 ms. This highlights the significant tradeoff between speed and accuracy, as smaller models like YOLOv10n offer rapid processing at the expense of detection reliability.

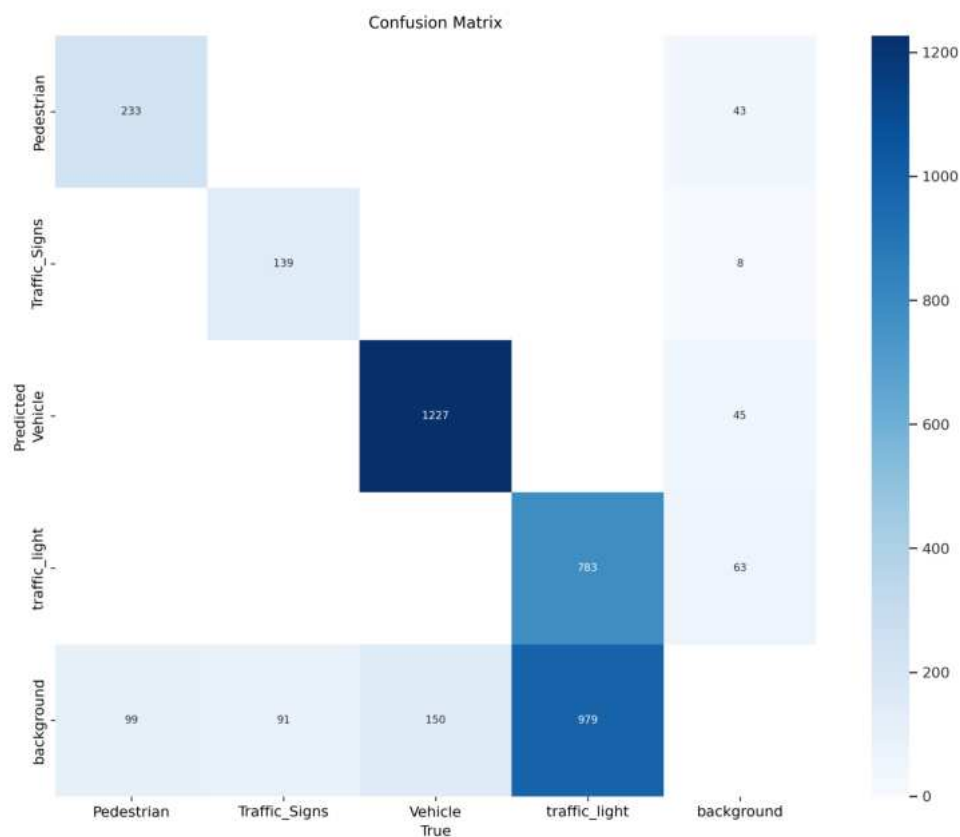


Figure 6.10: Confusion matrix for the YOLOv8m model on test set, highlighting its classification accuracy across all object classes.

Overall, the tradeoff between mAP(50-95) and speed is a crucial consideration in model selection. Larger models, such as YOLOv8m and YOLO11m, achieve high accuracy but come with increased latency, whereas smaller models like YOLOv10n prioritize speed but compromise on accuracy. For applications requiring a balance, models like YOLO11s provide a practical middle ground with competitive mAP(50-95) and reasonable speed.

## 6.4 Additional Evaluation Results

The additional testing results are categorized into two parts based on the [IoU](#) and confidence score values. Table [6.7](#) summarizes the results for scenario (A), while Table [6.8](#) presents scenario (B), which utilizes the more commonly used IoU threshold of 0.5. This scenario highlights the models' generalization capabilities under optimal conditions.

Model	Preprocess (ms/image)↓	Inference (ms/image)↓	Postprocess (ms/image)↓	Speed (ms/image)↓	Frames (↑)
YOLOv8n	1.9	<b>1.3</b>	2.6	5.8	172
YOLOv8s	1.9	2.5	1.8	6.2	161
YOLOv8m	1.9	4.8	1.6	8.3	120
YOLOv9t	1.9	1.7	1.9	5.5	182
YOLOv9s	1.9	3.0	1.6	6.5	154
YOLOv9m	1.9	5.7	1.3	8.9	112
YOLOv10n	1.9	1.7	0.7	<b>4.3</b>	<b>233</b>
YOLOv10s	2.0	2.7	<b>0.3</b>	5.0	200
YOLOv10m	1.9	5.3	<b>0.3</b>	7.5	133
YOLO11n	1.9	1.9	3.1	6.9	145
YOLO11s	1.9	2.5	2.0	6.4	156
YOLO11m	1.9	5.1	1.5	8.5	118

Table 6.6: Performance metrics for various YOLO models including pre-processing time, inference time, post-processing time, speed, and frames per second. Metrics marked with ↓ indicate that lower values are better, while those marked with ↑ indicate that higher values are better.

**Scenario A (IoU: 0.6, Confidence: 0.25, Checkpoint: Last):** The testing results, summarized in Table 6.7, further validate the generalization capabilities of the models. YOLO11m and YOLOv8m both achieve the highest mAP(50) of 0.802, demonstrating strong detection performance on the unseen test dataset. However, YOLOv8m outperforms YOLO11m in terms of inference speed (4.8 ms vs. 5.2 ms) and offers equal mAP(50-95) (0.588 vs. 0.588), showing its suitability for real-time applications. The testing results also highlight the trade-off between inference speed and detection accuracy. For instance, YOLOv8n achieves fast inference (1.1 ms) but sacrifices detection accuracy, with a lower mAP(50) of 0.753. In contrast, YOLO11m maintains high accuracy while achieving reasonable inference speed, offering a balanced solution for real-time object detection tasks. At an IoU threshold of 0.6 and a confidence threshold of 0.25, the test set results are depicted in Figure 6.12. The precision-confidence curve (Figure 6.12a) indicates perfect precision (100%) at a confidence level of 0.911, suggesting potential overfitting. The recall-confidence curve (Figure 6.12b) shows a recall of 67% at the lowest confidence threshold, capturing a majority of true positives but accompanied by higher false positives.

The precision-recall curve (Figure 6.12c) reports a mean average precision (mAP) of 80.2% across four classes at an IoU threshold of 0.5. Additionally, the F1-confidence curve (Figure 6.12d) identifies 0.227 as the optimal confidence threshold, where the peak F1 score of 75% is achieved, effectively balancing precision and recall.

**Scenario B (IoU: 0.5, Confidence: Optimal Confidence scores, Checkpoint: Last):** Table 6.8 compares the performance of various YOLO models evaluated under Scenario B. The Confidence column indicates the optimal thresholds derived from the F1-Confidence curve during validation, ensuring the best balance of precision and recall. YOLO11m demonstrates superior overall performance with the highest mAP@0.5:0.95 (0.590) followed by YOLOv8m (0.588), highlighting their ability to balance precision and recall while excelling in accurate detection and localization across varying IoU thresholds. This metric ensures robust

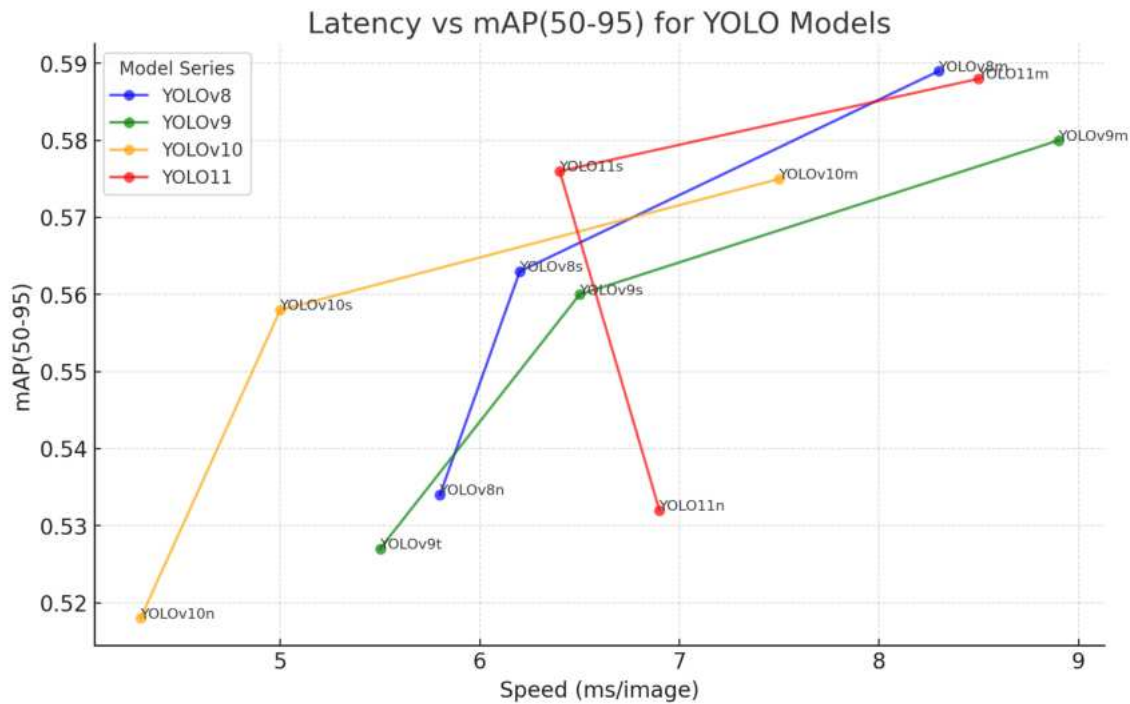


Figure 6.11: Latency vs. mAP(50-95) plot showcasing the real-time performance evaluation of YOLO models. Models closer to the top-left corner exhibit higher accuracy and faster overall speeds, making them ideal candidates for real-time deployments. Overall speed in this case means sum of preprocessing time, inference time and postprocessing time in ms per image.

performance not just at lenient thresholds like  $\text{IoU}=0.5$  but also at more stringent levels like  $\text{IoU}=0.95$ , making YOLO11m and YOLOv8m particularly effective for applications requiring both detection accuracy and precise localization.

While YOLOv9m excels in Recall (0.680) and  $\text{mAP}@0.5$  (0.802), it does not provide the same level of comprehensive performance across all  $\text{IoU}$  thresholds as YOLO11m and YOLOv8m. Additionally, although YOLOv8n and YOLOv10n are the fastest models with inference speeds of 1.6ms per image, their lower  $\text{mAP}@0.5:0.95$  (0.534 and 0.520, respectively) indicates a trade-off in accuracy and localization capability.

For real-time applications where accurate object detection and localization are critical, YOLO11m and YOLOv8m strikes the right balance. With an inference speed of 5.1ms and 5.0 ms per image respectively, these models meet the requirements for real-time performance without sacrificing detection precision or localization accuracy, as evidenced by their high  $\text{mAP}@0.5:0.95$  score. This makes YOLO11m and YOLOv8m the optimal choice for scenarios that demand accurate real-time detection rather than prioritizing speed above all else.

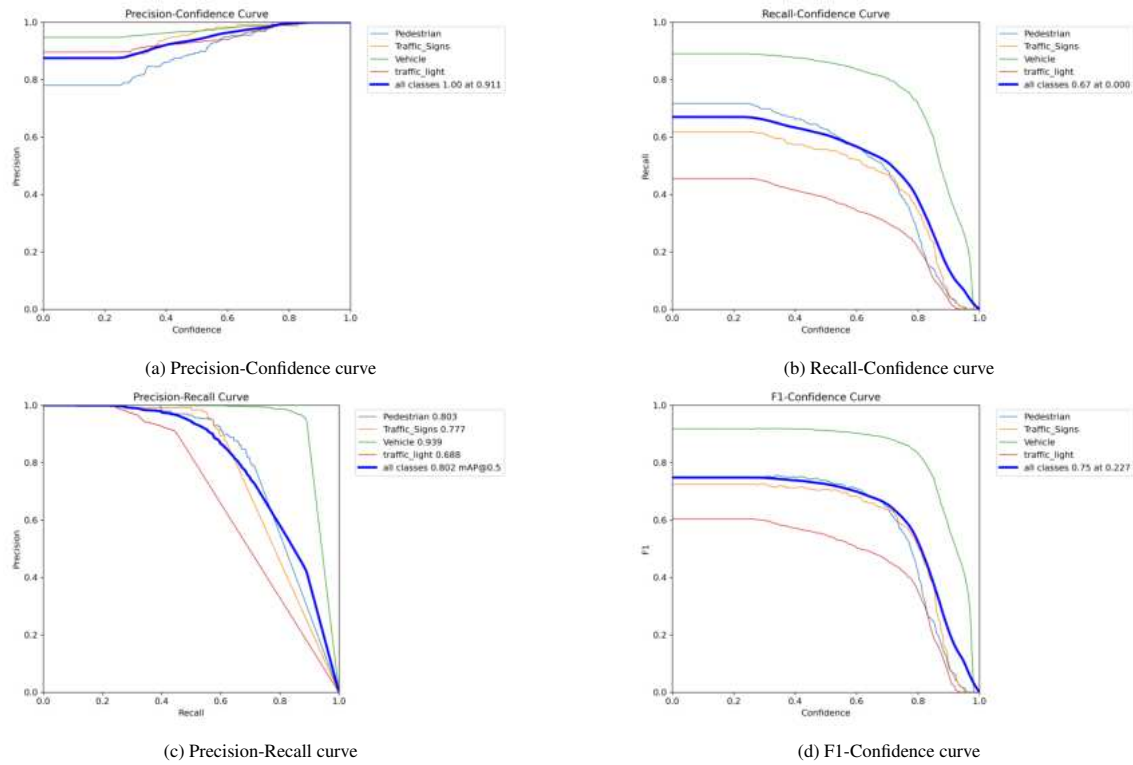


Figure 6.12: Performance evaluation of the YOLO11m model on the test dataset in Scenario A, depicted through Precision-Confidence, Recall-Confidence, Precision-Recall, and F1 Score-Confidence curves.

Model	Class	Images	Dataset Split	Precision	Recall	mAP(50)	mAP(50-95)	Inference Speed (ms/image)↓
YOLOv8n	all	641	Test	0.857	0.595	0.753	0.534	1.1
YOLOv8s	all	641	Test	0.868	0.630	0.776	0.565	2.2
YOLOv8m	all	641	Test	0.886	<b>0.670</b>	<b>0.802</b>	<b>0.588</b>	4.8
YOLOv9t	all	641	Test	0.860	0.572	0.743	0.531	1.5
YOLOv9s	all	641	Test	0.875	0.620	0.775	0.557	3.1
YOLOv9m	all	641	Test	<b>0.896</b>	0.649	0.795	0.586	5.9
YOLOv10n	all	641	Test	0.804	0.572	0.722	0.517	2.2
YOLOv10s	all	641	Test	0.837	0.625	0.765	0.559	3.8
YOLOv10m	all	641	Test	0.852	0.644	0.779	0.572	6.1
YOLO11n	all	641	Test	0.840	0.593	0.748	0.527	1.5
YOLO11s	all	641	Test	0.873	0.643	0.787	0.575	2.6
YOLO11m	all	641	Test	0.875	<b>0.670</b>	<b>0.802</b>	<b>0.588</b>	5.2

Table 6.7: Test results for YOLO models, showing Precision, Recall, mAP(50), mAP(50-95), and Inference Speed (ms/image) in Scenario A.

Model	Confidence	Class	Precision	Recall	mAP(50)	mAP(50-95)	Inference Speed (ms/image)↓
YOLOv8n	0.247	all	0.862	0.595	0.754	0.534	1.8
YOLOv8s	0.241	all	0.861	0.632	0.777	0.565	2.4
YOLOv8m	0.315	all	<b>0.907</b>	0.652	0.798	0.589	5.0
YOLOv9t	0.290	all	0.888	0.560	0.744	0.535	1.9
YOLOv9s	0.266	all	0.889	0.615	0.776	0.560	3.0
YOLOv9m	0.156	all	0.859	<b>0.680</b>	<b>0.802</b>	0.582	5.6
YOLOv10n	0.264	all	0.816	0.565	0.723	0.520	<b>1.6</b>
YOLOv10s	0.233	all	0.829	0.627	0.764	0.558	2.9
YOLOv10m	0.320	all	0.876	0.623	0.775	0.576	5.1
YOLO11n	0.280	all	0.865	0.580	0.749	0.531	<b>1.6</b>
YOLO11s	0.304	all	0.901	0.626	0.785	0.576	2.5
YOLO11m	0.317	all	0.896	0.655	0.799	<b>0.590</b>	5.1

Table 6.8: Comparison of YOLO models tested in Scenario B, highlighting Precision, Recall, mAP(50), mAP(50-95), and Inference Speed (ms/image) at an IoU threshold of 0.5 with optimal confidence scores. The evaluation was performed on the test dataset consisting of 641 images.

## 6.5 Global Performance Conclusion

**Scenario A:** As shown in Table 6.9, the **YOLO11m** model achieves the highest mAP@0.5 of **0.802** and mAP@0.5:0.95 of **0.588**, on par with YOLOv8m, while maintaining a slightly higher inference speed (5.2ms compared to 4.8ms for YOLOv8m). This highlights YOLO11m’s capability to balance accuracy with computational efficiency.

**Scenario B:** As presented in Table 6.10, the **YOLO11m** model achieves the highest mAP@0.5:0.95 of **0.590** and the second-highest mAP@0.5 of 0.799. Despite these high detection accuracies, YOLO11m maintains a lower inference speed (5.1ms compared to 5.6ms for YOLOv9m), showcasing its strength in scenarios that demand real-time performance.

Across both Scenario A and Scenario B, YOLO11m and YOLOv8m consistently exhibit their ability to balance detection accuracy and inference speed. These models achieve the highest mAP@0.5:0.95 scores in

both scenarios, underlining their effectiveness in precise localization and comprehensive detection. While certain models excel in specific metrics, such as inference speed or mAP@0.5, YOLO11m and YOLOv8m provide an optimal solution for applications requiring accurate real-time detection without sacrificing overall performance.

These findings align with the evaluation in Subsection 6.3.4. Furthermore, Figure 6.11 clearly illustrates the trade-off between speed and performance (mAP@0.5:0.95) for each YOLO series. The YOLOv10 series emerges as the fastest in every size-based model variant, whereas YOLO11m demonstrates the highest performance among nano and small variants. Notably, YOLO11m performs comparably to the highest-performing model, YOLOv8m. The YOLO11 series offers high-performing model variants with robust mAP@0.5:0.95 scores while maintaining reasonable inference speeds.

Given that ultra-high overall speed is not a critical requirement for this project, the YOLO11 and YOLOv8 models strike an ideal balance between detection accuracy and performance efficiency. Conversely, in scenarios prioritizing compact model size and faster operational capacity—such as deployment on edge devices—YOLOv10n stands out as the preferred choice. Its smaller footprint and expedited processing capabilities, as evidenced by its positioning in the plot, make it an excellent option for edge-based applications.

Model	Dataset Split	mAP(50)	mAP(50-95)	Inference Speed (ms/image)
YOLOv8m	Test	<b>0.802</b>	<b>0.588</b>	<b>4.8</b>
YOLOv9m	Test	0.795	0.586	5.9
YOLOv10m	Test	0.779	0.572	6.1
YOLO11m	Test	<b>0.802</b>	<b>0.588</b>	5.2

Table 6.9: Performance summary of top-performing YOLO models on the test dataset in Scenario A.

Model	Dataset Split	mAP(50)	mAP(50-95)	Inference Speed (ms/image)
YOLOv8m	Test	0.798	0.589	5.0
YOLOv9m	Test	<b>0.802</b>	0.582	5.6
YOLOv11s	Test	0.785	0.576	<b>2.5</b>
YOLO11m	Test	0.799	<b>0.590</b>	5.1

Table 6.10: Performance summary of top-performing YOLO models on the test dataset in Scenario B.

### 6.5.1 YOLO11m: Overfitting vs Underfitting

The training and validation loss curves for the YOLO11m model, depicted in Figures 6.13, 6.14 and 6.15 provide valuable insights into the model's learning dynamics and potential issues related to overfitting or underfitting. In Figure 6.13, the box loss [172] curves demonstrate that the training loss remains slightly lower than the validation loss throughout the training process. This behavior suggests minor overfitting, as the model performs slightly better on the training data than on the validation set. However, the close alignment between the two curves indicates good generalization, with only modest scope for further improvement.

Figure 6.14 presents the classification loss (cls loss). Here, the training and validation loss curves converge after epoch 20, reflecting balanced learning and the absence of notable overfitting or underfitting. This suggests that the model is effectively optimizing the classification component of the loss. In contrast, Figure 6.15 highlights an atypical pattern in the distribution focal loss (dfl loss) [171]. The training loss consistently exceeds the validation loss, indicating underfitting and suggesting a suboptimal optimization strategy for this component. This disparity points to potential weaknesses in the model’s ability to effectively minimize the dfl loss during training.

Overall, these observations underscore areas for potential improvement, such as mitigating overfitting in the box loss and refining the optimization strategy for the dfl loss to further enhance the model’s overall performance.

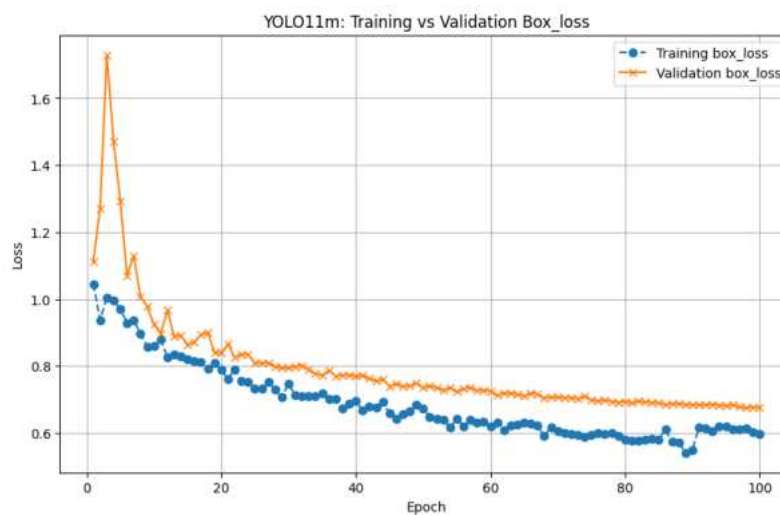


Figure 6.13: Box Loss Curves for YOLO11m Across 100 Epochs, Illustrating Training and Validation Loss Trends.

## 6.6 Qualitative Evaluation of Real-Time Object Detection

This section evaluates the performance of the YOLO11m model for real-time 2D object detection when deployed in both previously unseen and seen CARLA simulation maps/environments. The qualitative analysis is conducted using visualizations of real-time detection outputs, highlighting the model’s generalization capability, advantages, and limitations. The YOLO11m model, trained on a custom dataset for 100 epochs, uses the best and last checkpoints from the 99th epoch and 100th epoch for inference, leveraging the `YOLOInference` class for deployment. The methodology for this phase is detailed in Section 4.6. Between YOLOv8m and YOLO11m, either model can be selected for deployment due to their high performance. However, for this project, the YOLO11m model has been chosen as it belongs to the YOLO11 series—comprising YOLO11m, YOLO11s, and YOLO11n—which demonstrates superior performance when

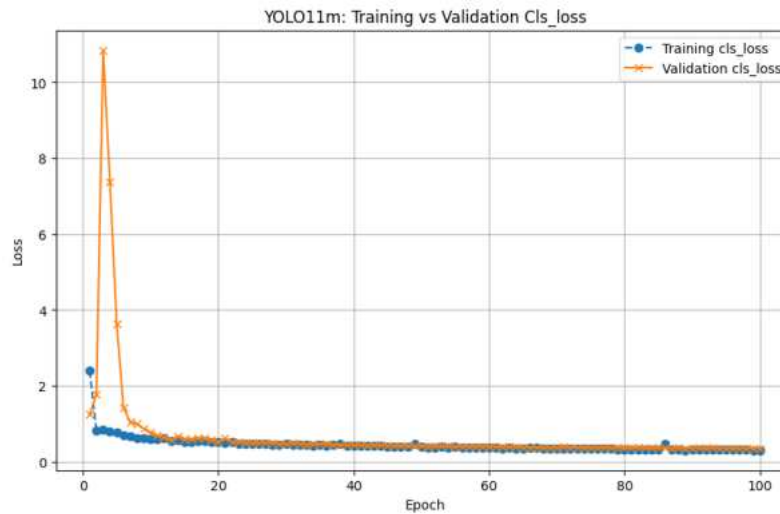


Figure 6.14: Classification Loss Curves for YOLO11m Across 100 epochs, Illustrating Training and Validation Loss Trends.

evaluated globally against other model series in figure 6.11.

#### Advantages:

- The generalization capability of the YOLO11m model is demonstrated in figure 6.16, which shows a grid of 2x4 images corresponding to 8 cameras mounted on a Tesla Model 3 in CARLA's Town07 map. Importantly, this map is entirely unseen during training, providing a robust test for generalization. The results indicate that the model performs well in detecting objects within a new environment, with differing background and object locations, under default weather conditions resembling evening or almost night.
- Similarly, figure 6.17 illustrates the model's performance in Town05 during nighttime with moderate traffic. Despite the challenges posed by low-light conditions, the model successfully detects objects illuminated by urban lighting, showcasing its potential for deployment in realistic nighttime urban lighting based driving scenarios.
- In moderate-traffic, daytime conditions, figure 6.18 highlights the model's ability to generalize effectively, even in environments with significant occlusions. The model demonstrates robust object detection, accurately identifying vehicles, pedestrians, traffic lights, and traffic signs amidst challenging scenarios.

**Limitations:** While the YOLO11m model exhibits strong generalization capabilities, some limitations are observed during inference.

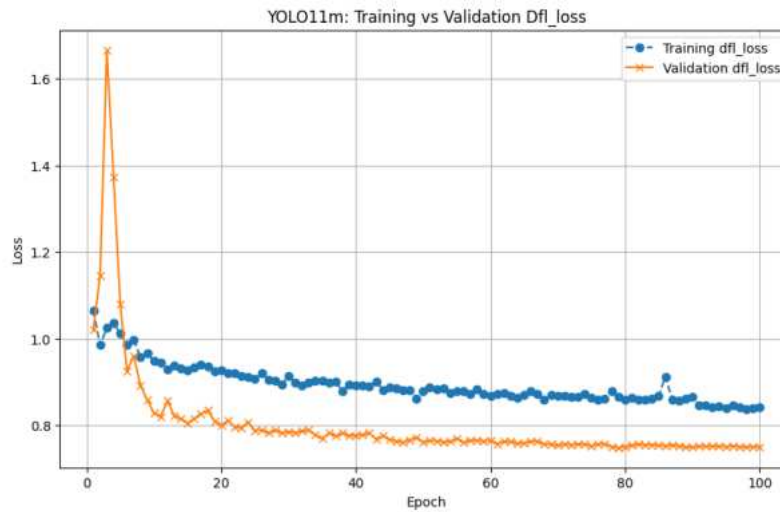


Figure 6.15: Distribution Focal Loss curves for YOLO11m across 100 epochs, Illustrating Training and Validation Loss Trends.

- Figure 6.19 highlights duplicate detections, where the model predicts the same vehicle twice: once with 64% confidence for the full body and again with 36% confidence for a partial front view. Additionally, the model fails to detect a distant traffic sign in the same frame, indicating challenges with long-range detections using RGB images and the need for further improvement in recall.
- In complete darkness, the model's bias toward the vehicle class becomes evident. Figures( 6.20a,6.20b) shows the model incorrectly labeling a background as a vehicle. This limitation suggests inefficiencies in using RGB-only image data for nighttime detection. Alternatives such as multi-spectral cameras, far-infrared systems, or near-infrared imaging could address this issue. This observation is further supported by the confusion matrix results, which show that the background class is often misclassified as other classes (refer to Figure A.6).
- Moreover, Figure 6.20c illustrates a scenario where the model predicts the entire camera frame as a vehicle with 74% confidence, underscoring the need for improved dataset curation and hyperparameter tuning.



Figure 6.16: Real-time object detection using YOLO11m in Town07 of CARLA simulation under default weather conditions. The results demonstrate the model's generalization to a completely unseen environment.

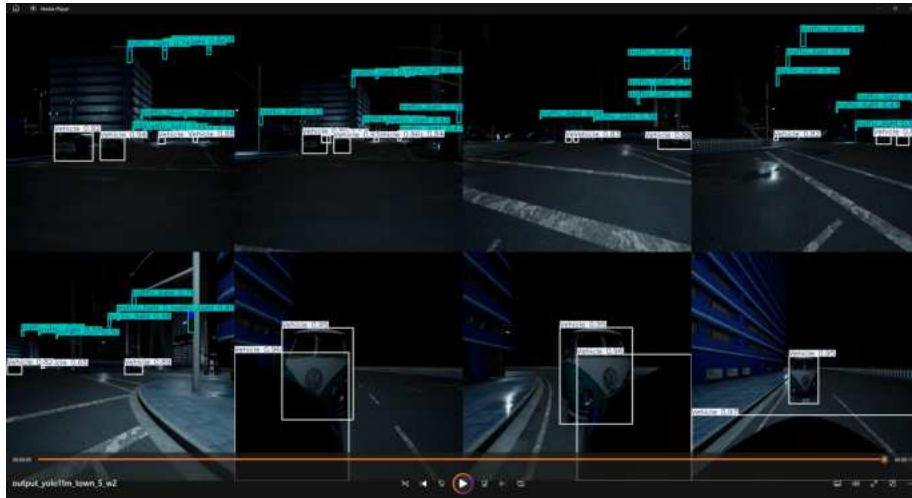


Figure 6.17: Real-time object detection using YOLO11m in Town05 of CARLA simulation during nighttime. The model demonstrates effective performance under moderate urban visibility or lighting conditions during night.

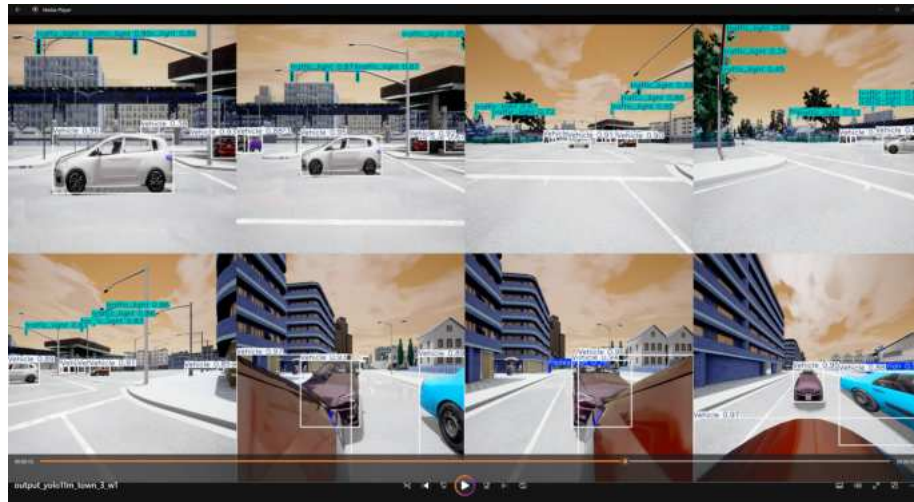
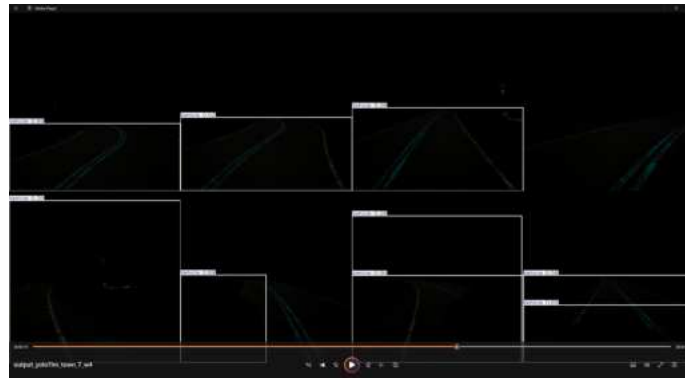


Figure 6.18: Real-time object detection using YOLO11m in Town03 of CARLA simulation during daytime. The model performs well in a moderate-traffic scenario with significant occlusions.



Figure 6.19: Real-time object detection using YOLO11m in CARLA simulation. The frame highlights duplicate detections and missed long-range traffic signs.



(a) Model bias toward the vehicle class in nighttime scenarios.



(b) Model incorrectly predicts a road as a vehicle.



(c) Model predicts the entire frame as a vehicle.

Figure 6.20: Generalization limitations of YOLO11m model during nighttime scenarios. These results highlight issues with model bias, long-range detection, hyperparameter tuning and dataset quality.

## Chapter 7

### Discussion

Building on the previous chapters, this chapter critically examines the key challenges, limitations, and implications of the conducted research. By analyzing the experimental results and situating them within the broader context of autonomous driving and object detection, it provides actionable insights for addressing identified issues and advancing the field. The discussion is organized into two main sections: challenges and limitations, and implications.

#### 7.1 Challenges and Limitations

The research faced several challenges and limitations that constrained the scope and outcomes of the study. First, the dataset used was relatively small, which limited the extent of the experiments and potentially affected the generalizability of the results. Additionally, the dataset included only four labeled classes, which were insufficient to represent the complexity of real-world environments. The manual review step in the labeling process was also time-intensive, making it difficult to scale the data preparation phase.

Experimentally, baseline experiments were conducted without employing advanced techniques such as data augmentation, regularization, or hyperparameter tuning. This lack of optimization highlighted areas where the model's performance could be further improved. Duplicate detections were frequently observed, where the same object was predicted multiple times with varying confidence levels, while missed detections, particularly for long-range objects like traffic signs, exposed challenges in recall and multi-scale detection.

Environmental factors also posed significant challenges. For instance, the reliance on RGB-only data in low-light conditions led to frequent misclassifications, such as backgrounds being labeled as vehicles. Some scenarios also resulted in false positives, where entire frames were incorrectly predicted as single objects, underscoring the need for better dataset curation and model refinement.

Finally, generalization and scalability presented key limitations. While the model generalized well to unseen environments, it was not tested under extreme weather conditions such as heavy rain or snow. Additionally, extending the framework to include tasks such as Sensor fusion or 3D object detection requires significant effort in sensor alignment and dataset expansion. These challenges reflect the broader issues

associated with deploying object detection systems in real-world scenarios.

## 7.2 Implications

Despite these challenges, the findings of this research have important implications for the field of autonomous driving and object detection. The study highlights the critical need for larger, more diverse datasets that include additional classes and environmental conditions to ensure better real-world representation. Fully Automated labeling methods could also play a pivotal role in reducing the time and cost associated with data preparation.

From a technical perspective, improving the detection process is essential. Enhanced loss functions and further advancements in model architecture or post-processing techniques could mitigate the issue of duplicate and missed detections. Moreover, integrating multi-modal imaging, such as far-infrared or near-infrared images, could significantly improve detection performance under challenging conditions like low light.

This research also opens up several avenues for future exploration. Techniques such as data augmentation, advanced regularization, and hyperparameter tuning offer promising opportunities for enhancing model performance. Investigating model architectures optimized for specific challenges, including occlusions, dynamic backgrounds, and diverse object sizes, is another crucial direction for future work.

The custom simulation framework demonstrates significant potential for advancing autonomous driving research through real-world applications. Tailored to simulate a Tesla Model 3 equipped with 21 sensors, it provides a robust platform for testing 2D object detection algorithms and can be extended to support tasks such as sensor fusion, 3D object detection, and panoptic segmentation. Its modular, scalable design enables seamless integration of additional sensors and adaptation to diverse simulation scenarios, making it an indispensable tool for a wide range of research objectives.

## 7.3 Summary

This chapter has discussed the key challenges and limitations encountered during the research, including dataset constraints, experimental inefficiencies, and environmental factors. It has also highlighted the broader implications of the study, emphasizing its potential to advance object detection systems in autonomous driving. These insights lay the foundation for addressing limitations and exploring future research directions.

## Chapter 8

### Conclusion and Future Work

This final chapter summarizes the key findings presented in Chapter 6, reflecting on their alignment with the objectives outlined in Section 1.2.1. It also outlines potential directions for future research, offering insights to guide further advancements in the field.

#### 8.1 Conclusion

The objectives of this research project were successfully achieved. A comprehensive simulation setup of a Tesla Model 3, equipped with 8 cameras, 12 ultrasonic sensors, and 1 radar, was developed to facilitate sensor data collection. This setup enabled the creation of a synthetic dataset consisting of 6,400 annotated images, capturing vehicles, traffic lights, pedestrians, and traffic signs. The dataset, distributed across training, validation, and test splits, represented diverse driving scenarios and served as the foundation for training state-of-the-art YOLO models, including *YOLOv8*, *YOLOv9*, *YOLOv10*, and *YOLO11*, as well as their size-based versions. Among the 12 models evaluated, *YOLOv8m* and *YOLO11m* emerged as the top-performing models, achieving the highest mAP@0.5:0.95—a rigorous and comprehensive metric assessing both detection accuracy and precise localization. On a global comparison at the series level, the YOLO11 series demonstrated superior overall performance. Additionally, the *YOLO11m* model was successfully integrated into the CARLA Simulator, facilitating accurate and real-time object detection within the simulated environment.

Despite these accomplishments, the study faced limitations that point to opportunities for future work. Dataset constraints, such as limited size and class diversity, posed challenges to model generalization. Additionally, the absence of advanced techniques like data augmentation and hyperparameter tuning hindered further optimization. Nonetheless, this research provides valuable insights into the baseline performance and generalization capabilities of YOLO models on the custom dataset and establishes a solid foundation for future advancements in autonomous driving simulations.

## 8.2 Future Work

Building on the contributions of this research project, several promising directions for future work are identified. **Expanding the synthetic dataset** to encompass a broader range of environmental conditions, including adverse weather and diverse lighting scenarios, is essential for enhancing model robustness and generalizability. The integration of real-world data and the application of domain adaptation techniques offer valuable opportunities to bridge the gap between simulation and real-world performance.

Future research should also prioritize **advanced optimization strategies**, such as self-supervised learning, automated architecture search, and innovative data augmentation techniques, to further improve model efficiency and accuracy. Incorporating **multi-modal data**—including radar, ultrasonic sensor data, RGB images, and near-infrared imagery—could provide significant advantages in addressing challenges posed by extreme weather and lighting conditions. Developing robust sensor fusion algorithms for precise temporal and spatial alignment of multi-modal data will be crucial for achieving optimal performance.

**Extending the framework** to include additional tasks, such as instance segmentation and 3D object detection, represents another exciting avenue for future exploration. These extensions will enhance the versatility and applicability of the simulation framework, supporting more complex autonomous driving scenarios.

Finally, transitioning the proposed framework from simulation to real-world deployment is a critical next step. This transition will require addressing challenges related to real-time performance, hardware constraints, and edge deployment, ensuring the system's feasibility for practical applications.

In summary, this research project has made significant contributions to the development of simulation-based object detection systems for autonomous driving. While the findings establish a strong foundation, the proposed directions for future work underscore the exciting opportunities to further advance the field, paving the way for safer, more reliable, and efficient autonomous driving technologies.

## Bibliography

- [1] Zhengxia Zou, Zhenwei Shi, Yuhong Guo, and Jieping Ye. Object detection in 20 years: A survey. *CoRR*, abs/1905.05055, 2019. URL <http://arxiv.org/abs/1905.05055>.
- [2] Jaskirat Kaur and Williamjeet Singh. Tools, techniques, datasets and application areas for object detection in an image: a review. *Multimedia Tools and Applications*, 81(27):38297–38351, 2022.
- [3] Syed Sahil Abbas Zaidi, Mohammad Samar Ansari, Asra Aslam, Nadia Kanwal, Mamoon Naveed Asghar, and Brian Lee. A survey of modern deep learning based object detection models. *CoRR*, abs/2104.11892, 2021. URL <https://arxiv.org/abs/2104.11892>.
- [4] Zvonimir Benčević, Ratko Grbić, Borna Jelić, and Mario Vranješ. Tool for automatic labeling of objects in images obtained from carla autonomous driving simulator. In *2023 Zooming Innovation in Consumer Technologies Conference (ZINC)*, pages 119–124. IEEE, 2023.
- [5] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017.
- [6] Supriya Sarker, Brent Maples, and Weizi Li. A comprehensive review on traffic datasets and simulators for autonomous vehicles, 2024. URL <https://arxiv.org/abs/2412.14207>.
- [7] Tao Sun, Mattia Segu, Janis Postels, Yuxuan Wang, Luc Van Gool, Bernt Schiele, Federico Tombari, and Fisher Yu. Shift: a synthetic driving dataset for continuous multi-task domain adaptation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 21371–21382, 2022.
- [8] Yuhang Han, Zhengtao Liu, Shuo Sun, Dongen Li, Jiawei Sun, Chengran Yuan, and Marcelo H. Ang Jr. Carla-loc: Synthetic slam dataset with full-stack sensor setup in challenging weather and dynamic environments, 2024. URL <https://arxiv.org/abs/2309.08909>.
- [9] Liang Liang, Haihua Ma, Le Zhao, Xiaopeng Xie, Chengxin Hua, Miao Zhang, and Yonghui Zhang. Vehicle detection algorithms for autonomous driving: a review. *Sensors*, 24(10):3088, 2024.

- [10] Martin Georg Ljungqvist, Otto Nordander, Markus Skans, Arvid Mildner, Tony Liu, and Pierre Nugues. Object detector differences when using synthetic and real training data. *SN computer science*, 4(3):302, 2023.
- [11] Ofonime Dominic Okon and Li Meng. Detecting distracted driving with deep learning. In *Interactive Collaborative Robotics: Second International Conference, ICR 2017, Hatfield, UK, September 12-16, 2017, Proceedings 2*, pages 170–179. Springer, 2017.
- [12] Penghua Li, Yifeng Yang, Radu Grosu, Guodong Wang, Rui Li, Yuehong Wu, and Zeng Huang. Driver distraction detection using octave-like convolutional neural network. *IEEE Transactions on Intelligent Transportation Systems*, 23(7):8823–8833, 2022. doi: 10.1109/TITS.2021.3086411.
- [13] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.
- [14] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *ArXiv*, abs/2010.11929, 2020. URL <https://api.semanticscholar.org/CorpusID:225039882>.
- [15] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9650–9660, 2021.
- [16] Bing Li, Jie Chen, Zhixiang Huang, Haitao Wang, Jianming Lv, Jingmin Xi, Jun Zhang, and Zhongcheng Wu. A new unsupervised deep learning algorithm for fine-grained detection of driver distraction. *IEEE Transactions on Intelligent Transportation Systems*, 23(10):19272–19284, 2022.
- [17] Dohun Kim, Hyukjin Park, Tonghyun Kim, Wonjong Kim, and Joonki Paik. Real-time driver monitoring system with facial landmark-based eye closure detection and head pose recognition. *Scientific reports*, 13(1):18264, 2023.
- [18] B Janet, U Srinivasulu Reddy, et al. Real time detection of driver distraction using cnn. In *2020 Third International Conference on Smart Systems and Inventive Technology (ICSSIT)*, pages 185–191. IEEE, 2020.
- [19] Yingzhi Zhang, Taiguo Li, Chao Li, and Xinghong Zhou. A novel driver distraction behavior detection method based on self-supervised learning with masked image modeling. *IEEE Internet of Things Journal*, 2023.
- [20] R Archana and PS Eliahim Jeevaraj. Deep learning models for digital image processing: a review. *Artificial Intelligence Review*, 57(1):11, 2024.

- [21] Chen Huang, Xiaochen Wang, Jiannong Cao, Shihui Wang, and Yan Zhang. Hcf: A hybrid cnn framework for behavior detection of distracted drivers. *IEEE Access*, 8:109335–109349, 2020. doi: 10.1109/ACCESS.2020.3001159.
- [22] Tao Huang, Jianan Liu, Xi Zhou, Dinh C. Nguyen, Mostafa Rahimi Azghadi, Yuxuan Xia, Qing-Long Han, and Sumei Sun. V2x cooperative perception for autonomous driving: Recent advances and challenges, 2024. URL <https://arxiv.org/abs/2310.03525>.
- [23] Scott Drew Pendleton, Hans Andersen, Xinxin Du, Xiaotong Shen, Malika Meghjani, You Hong Eng, Daniela Rus, and Marcelo H. Ang. Perception, planning, control, and coordination for autonomous vehicles. *Machines*, 5(1), 2017. ISSN 2075-1702. doi: 10.3390/machines5010006. URL <https://www.mdpi.com/2075-1702/5/1/6>.
- [24] Yan Wang, Wei-Lun Chao, Divyansh Garg, Bharath Hariharan, Mark Campbell, and Kilian Q Weinberger. Pseudo-lidar from visual depth estimation: Bridging the gap in 3d object detection for autonomous driving. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 8445–8453, 2019.
- [25] Jonah Philion and Sanja Fidler. Lift, splat, shoot: Encoding images from arbitrary camera rigs by implicitly unprojecting to 3d. In *Computer Vision—ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XIV 16*, pages 194–210. Springer, 2020.
- [26] Zhiqi Li, Wenhai Wang, Hongyang Li, Enze Xie, Chonghao Sima, Tong Lu, Yu Qiao, and Jifeng Dai. Bevformer: Learning bird’s-eye-view representation from multi-camera images via spatiotemporal transformers. In *European conference on computer vision*, pages 1–18. Springer, 2022.
- [27] Yongzhi Su, Yan Di, Guangyao Zhai, Fabian Manhardt, Jason Rambach, Benjamin Busam, Didier Stricker, and Federico Tombari. Opa-3d: Occlusion-aware pixel-wise aggregation for monocular 3d object detection. *IEEE Robotics and Automation Letters*, 8(3):1327–1334, 2023.
- [28] Haisong Liu, Yao Teng, Tao Lu, Haiguang Wang, and Limin Wang. Sparsebev: High-performance sparse 3d object detection from multi-camera videos. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 18580–18590, 2023.
- [29] Shaoshuai Shi, Xiaogang Wang, and Hongsheng Li. Pointtrcnn: 3d object proposal generation and detection from point cloud. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 770–779, 2019.
- [30] Alex H Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. Pointpillars: Fast encoders for object detection from point clouds. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12697–12705, 2019.

- [31] Tianwei Yin, Xingyi Zhou, and Philipp Krahenbuhl. Center-based 3d object detection and tracking. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11784–11793, 2021.
- [32] Li Wang, Ziyang Song, Xinyu Zhang, Chenfei Wang, Guoxin Zhang, Lei Zhu, Jun Li, and Huaping Liu. Sat-gcn: Self-attention graph convolutional network-based 3d object detection for autonomous driving. *Knowledge-Based Systems*, 259:110080, 2023.
- [33] Chenhao He, Ruihuang Li, Shuai Li, and Lei Zhang. Voxel set transformer: A set-to-set approach to 3d object detection from point clouds. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 8417–8427, 2022.
- [34] Tao Lu, Xiang Ding, Haisong Liu, Gangshan Wu, and Limin Wang. Link: Linear kernel for lidar-based 3d perception. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1105–1115, 2023.
- [35] Hai Wu, Chenglu Wen, Shaoshuai Shi, Xin Li, and Cheng Wang. Virtual sparse convolution for multimodal 3d object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 21653–21662, 2023.
- [36] Weiyi Xiong, Jianan Liu, Yuxuan Xia, Tao Huang, Bing Zhu, and Wei Xiang. Contrastive learning for automotive mmwave radar detection points based instance segmentation. In *2022 IEEE 25th International Conference on Intelligent Transportation Systems (ITSC)*, pages 1255–1261. IEEE, 2022.
- [37] Jianan Liu, Weiyi Xiong, Liping Bai, Yuxuan Xia, Tao Huang, Wanli Ouyang, and Bing Zhu. Deep instance segmentation with automotive radar detection points. *IEEE Transactions on Intelligent Vehicles*, 8(1):84–94, 2022.
- [38] Andras Palffy, Jiaao Dong, Julian FP Kooij, and Darius M Gavrila. Cnn based road user detection using the 3d radar cube. *IEEE Robotics and Automation Letters*, 5(2):1263–1270, 2020.
- [39] Lianqing Zheng, Zhixiong Ma, Xichan Zhu, Bin Tan, Sen Li, Kai Long, Weiqi Sun, Sihan Chen, Lu Zhang, Mengyue Wan, et al. Tj4dradset: A 4d radar dataset for autonomous driving. In *2022 IEEE 25th International Conference on Intelligent Transportation Systems (ITSC)*, pages 493–498. IEEE, 2022.
- [40] Bin Tan, Zhixiong Ma, Xichan Zhu, Sen Li, Lianqing Zheng, Sihan Chen, Libo Huang, and Jie Bai. 3-d object detection for multiframe 4-d automotive millimeter-wave radar point cloud. *IEEE Sensors Journal*, 23(11):11125–11138, 2022.
- [41] Jianan Liu, Qiuchi Zhao, Weiyi Xiong, Tao Huang, Qing-Long Han, and Bing Zhu. Smurf: Spatial multi-representation fusion for 3d object detection with 4d imaging radar. *IEEE Transactions on Intelligent Vehicles*, 2023.

- [42] Leena Chandrashekar. Characteristics of sonar sensors for short range measurement. *Int. J. Eng. Res. Technol.(IJERT)*, 3(11):340–344, 2014.
- [43] Tommaso Nesti, Santhosh Boddana, and Burhaneddin Yaman. Ultra-sonic sensor based object detection for autonomous vehicles. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pages 210–218, June 2023.
- [44] Arindam Das, Sudarshan Paul, Niko Scholz, Akhilesh Kumar Malviya, Ganesh Sistu, Ujjwal Bhattacharya, and Ciarán Eising. Fisheye camera and ultrasonic sensor fusion for near-field obstacle perception in bird’s-eye-view, 2024. URL <https://arxiv.org/abs/2402.00637>.
- [45] Xuanyao Chen, Tianyuan Zhang, Yue Wang, Yilun Wang, and Hang Zhao. Futr3d: A unified sensor fusion framework for 3d detection. In *proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 172–181, 2023.
- [46] Zizhang Wu, Guilian Chen, Yuanzhu Gan, Lei Wang, and Jian Pu. Mvfusion: Multi-view 3d object detection with semantic-aligned radar and camera fusion. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2766–2773. IEEE, 2023.
- [47] Youngseok Kim, Juyeb Shin, Sanmin Kim, In-Jae Lee, Jun Won Choi, and Dongsuk Kum. Crn: Camera radar net for accurate, robust, efficient 3d perception. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 17615–17626, 2023.
- [48] Lianqing Zheng, Sen Li, Bin Tan, Long Yang, Sihan Chen, Libo Huang, Jie Bai, Xichan Zhu, and Zhixiong Ma. Rcfusion: Fusing 4-d radar and camera with bird’s-eye view features for 3-d object detection. *IEEE Transactions on Instrumentation and Measurement*, 72:1–14, 2023.
- [49] Yanlong Yang, Jianan Liu, Tao Huang, Qing-Long Han, Gang Ma, and Bing Zhu. Ralibev: Radar and lidar bev fusion learning for anchor box free object detection system. *arXiv preprint arXiv:2211.06108*, 2022.
- [50] Rudolf Krecht, Tamás Budai, Ernő Horváth, Ákos Kovács, Nobert Markó, and Miklós Unger. Network optimization aspects of autonomous vehicles: Challenges and future directions. *IEEE Network*, 37(4):282–288, 2023.
- [51] Mustafa Ridvan Cantas and Levent Guvenc. Customized co-simulation environment for autonomous driving algorithm development and evaluation. *arXiv preprint arXiv:2306.00223*, 2023.
- [52] Lucas Fonseca Alexandre de Oliveira, Johannes Bernhard, Lars Schories, Martin Meywerk, and Eric Sax. Enhancing carla traffic simulation with pedestrian animation for testing perception functions in automated driving. In *2023 IEEE 26th International Conference on Intelligent Transportation Systems (ITSC)*, pages 3859–3864. IEEE, 2023.

- [53] Hugh Durrant-Whyte and Tim Bailey. Simultaneous localization and mapping: part i. *IEEE robotics & automation magazine*, 13(2):99–110, 2006.
- [54] Jaesung Jang, Hyeongyu Lee, and Jong-Chan Kim. Carfree: Hassle-free object detection dataset generation using carla autonomous driving simulator. *Applied Sciences*, 12(1), 2022. ISSN 2076-3417. doi: 10.3390/app12010281. URL <https://www.mdpi.com/2076-3417/12/1/281>.
- [55] Teslamag. Radar: Unklare angaben zu sensoren in teslas autopilot-system, 2021. URL <https://teslamag.de/news/radar-unklare-angaben-sensoren-teslas-autopilot-system-web-37248>.
- [56] CARLA Simulator. Sensors reference, 2025. URL [https://carla.readthedocs.io/en/latest/ref\\_sensors/](https://carla.readthedocs.io/en/latest/ref_sensors/). Accessed on January 21, 2025.
- [57] Alexander Andreopoulos and John K Tsotsos. 50 years of object recognition: Directions forward. *Computer vision and image understanding*, 117(8):827–891, 2013.
- [58] Peter Elias and Lawrence G Roberts. *Machine perception of three-dimensional solids*. PhD thesis, Massachusetts Institute of Technology, 1963.
- [59] Seiji Kashioka, Masakazu Ejiri, and Yuzaburo Sakamoto. A transistor wire-bonding system utilizing multiple local pattern matching techniques. *IEEE Transactions on Systems, Man, and Cybernetics*, (8):562–570, 1976.
- [60] Masakazu Ejiri. Machine vision in early days: Japan’s pioneering contributions. In *Computer Vision—ACCV 2007: 8th Asian Conference on Computer Vision, Tokyo, Japan, November 18-22, 2007, Proceedings, Part I* 8, pages 35–53. Springer, 2007.
- [61] David Marr. *Vision: A computational investigation into the human representation and processing of visual information*. MIT press, 2010.
- [62] Alexander Andreopoulos, Stephan Hasler, Heiko Wersing, Herbert Janssen, John K Tsotsos, and Edgar Korner. Active 3d object localization using a humanoid robot. *IEEE Transactions on Robotics*, 27(1):47–64, 2010.
- [63] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE computer society conference on computer vision and pattern recognition. CVPR 2001*, volume 1, pages I–I. Ieee, 2001.
- [64] Paul Viola and Michael J Jones. Robust real-time face detection. *International journal of computer vision*, 57:137–154, 2004.

- [65] R. Lienhart and J. Maydt. An extended set of haar-like features for rapid object detection. In *Proceedings. International Conference on Image Processing*, volume 1, pages I–I, 2002. doi: 10.1109/ICIP.2002.1038171.
- [66] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, volume 1, pages 886–893. Ieee, 2005.
- [67] Piotr Dollar, Christian Wojek, Bernt Schiele, and Pietro Perona. Pedestrian detection: An evaluation of the state of the art. *IEEE transactions on pattern analysis and machine intelligence*, 34(4):743–761, 2011.
- [68] Pedro Felzenszwalb, David McAllester, and Deva Ramanan. A discriminatively trained, multiscale, deformable part model. In *2008 IEEE conference on computer vision and pattern recognition*, pages 1–8. Ieee, 2008.
- [69] Pedro F Felzenszwalb, Ross B Girshick, David McAllester, and Deva Ramanan. Object detection with discriminatively trained part-based models. *IEEE transactions on pattern analysis and machine intelligence*, 32(9):1627–1645, 2009.
- [70] Yann LeCun, Bernhard Boser, John Denker, Donnie Henderson, Richard Howard, Wayne Hubbard, and Lawrence Jackel. Handwritten digit recognition with a back-propagation network. *Advances in neural information processing systems*, 2, 1989.
- [71] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023. URL <https://arxiv.org/abs/1706.03762>.
- [72] Yibo Sun, Zhe Sun, and Weitong Chen. The evolution of object detection methods. *Engineering Applications of Artificial Intelligence*, 133:108458, 2024.
- [73] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation, 2014. URL <https://arxiv.org/abs/1311.2524>.
- [74] Jasper RR Uijlings, Koen EA Van De Sande, Theo Gevers, and Arnold WM Smeulders. Selective search for object recognition. *International journal of computer vision*, 104:154–171, 2013.
- [75] Bernhard E Boser, Isabelle M Guyon, and Vladimir N Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152, 1992.

- [76] Jair Cervantes, Farid Garcia-Lamont, Lisbeth Rodríguez-Mazahua, and Asdrubal Lopez. A comprehensive survey on support vector machine classification: Applications, challenges and trends. *Neurocomputing*, 408:189–215, 2020.
- [77] K Raja, Satyam Pandey, Shivangi Rani Verma, Saket Sharma, and A Senthilselvi. An intelligent approach to object detection using r-cnn. In *2024 Second International Conference on Advances in Information Technology (ICAIT)*, volume 1, pages 1–6. IEEE, 2024.
- [78] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. *Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition*, page 346–361. Springer International Publishing, 2014. ISBN 9783319105789. doi: 10.1007/978-3-319-10578-9\_23. URL [http://dx.doi.org/10.1007/978-3-319-10578-9\\_23](http://dx.doi.org/10.1007/978-3-319-10578-9_23).
- [79] Ross Girshick. Fast r-cnn, 2015. URL <https://arxiv.org/abs/1504.08083>.
- [80] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks, 2016. URL <https://arxiv.org/abs/1506.01497>.
- [81] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection, 2017. URL <https://arxiv.org/abs/1612.03144>.
- [82] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection, 2016. URL <https://arxiv.org/abs/1506.02640>.
- [83] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. *SSD: Single Shot MultiBox Detector*, page 21–37. Springer International Publishing, 2016. ISBN 9783319464480. doi: 10.1007/978-3-319-46448-0\_2. URL [http://dx.doi.org/10.1007/978-3-319-46448-0\\_2](http://dx.doi.org/10.1007/978-3-319-46448-0_2).
- [84] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection, 2018. URL <https://arxiv.org/abs/1708.02002>.
- [85] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers, 2020. URL <https://arxiv.org/abs/2005.12872>.
- [86] Xizhou Zhu, Weijie Su, Lewei Lu, Bin Li, Xiaogang Wang, and Jifeng Dai. Deformable detr: Deformable transformers for end-to-end object detection, 2021. URL <https://arxiv.org/abs/2010.04159>.
- [87] Byungseok Roh, JaeWoong Shin, Wuhyun Shin, and Saehoon Kim. Sparse detr: Efficient end-to-end object detection with learnable sparsity, 2022. URL <https://arxiv.org/abs/2111.14330>.

- [88] Zhuyu Yao, Jiangbo Ai, Boxun Li, and Chi Zhang. Efficient detr: Improving end-to-end object detector with dense prior, 2021. URL <https://arxiv.org/abs/2104.01318>.
- [89] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers distillation through attention, 2021. URL <https://arxiv.org/abs/2012.12877>.
- [90] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network, 2015. URL <https://arxiv.org/abs/1503.02531>.
- [91] Hugo Touvron, Matthieu Cord, Alexandre Sablayrolles, Gabriel Synnaeve, and Hervé Jégou. Going deeper with image transformers, 2021. URL <https://arxiv.org/abs/2103.17239>.
- [92] Haiping Wu, Bin Xiao, Noel Codella, Mengchen Liu, Xiyang Dai, Lu Yuan, and Lei Zhang. Cvt: Introducing convolutions to vision transformers, 2021. URL <https://arxiv.org/abs/2103.15808>.
- [93] Shivang Agarwal, Jean Ogier Du Terrail, and Frédéric Jurie. Recent advances in object detection in the age of deep convolutional neural networks. *arXiv preprint arXiv:1809.03193*, 2018.
- [94] Li Liu, Wanli Ouyang, Xiaogang Wang, Paul Fieguth, Jie Chen, Xinwang Liu, and Matti Pietikäinen. Deep learning for generic object detection: A survey. *International journal of computer vision*, 128: 261–318, 2020.
- [95] Chien-Yao Wang, Hong-Yuan Mark Liao, Yueh-Hua Wu, Ping-Yang Chen, Jun-Wei Hsieh, and I-Hau Yeh. Cspnet: A new backbone that can enhance learning capability of cnn. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, pages 390–391, 2020.
- [96] Mujadded Al Rabbani Alif and Muhammad Hussain. Yolov1 to yolov10: A comprehensive review of yolo variants and their application in the agricultural domain. *arXiv preprint arXiv:2406.10139*, 2024.
- [97] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.
- [98] Joseph Nelson. What is yolo? the ultimate guide [2024]. Roboflow Blog, Jul 17 2024. URL <https://blog.roboflow.com/guide-to-yolo-models/>.
- [99] Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger, 2016. URL <https://arxiv.org/abs/1612.08242>.
- [100] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement, 2018. URL <https://arxiv.org/abs/1804.02767>.

- [101] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL <http://arxiv.org/abs/1512.03385>.
- [102] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection, 2020. URL <https://arxiv.org/abs/2004.10934>.
- [103] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *CoRR*, abs/1406.4729, 2014. URL <http://arxiv.org/abs/1406.4729>.
- [104] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia. Path aggregation network for instance segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8759–8768, 2018.
- [105] Zhuliang Yao, Yue Cao, Shuxin Zheng, Gao Huang, and Stephen Lin. Cross-iteration batch normalization, 2021. URL <https://arxiv.org/abs/2002.05712>.
- [106] Sanghyun Woo, Jongchan Park, Joon-Young Lee, and In So Kweon. CBAM: convolutional block attention module. *CoRR*, abs/1807.06521, 2018. URL <http://arxiv.org/abs/1807.06521>.
- [107] Glenn Jocher. Ultralytics yolov5, 2020. URL <https://github.com/ultralytics/yolov5>.
- [108] Jacob Solawetz. What is yolov5? a guide for beginners. *Roboflow Blog*, June 2020. URL <https://blog.roboflow.com/yolov5-improvements-and-evaluation/>.
- [109] Xiang Long, Kaipeng Deng, Guanzhong Wang, Yang Zhang, Qingqing Dang, Yuan Gao, Hui Shen, Jianguo Ren, Shumin Han, Errui Ding, and Shilei Wen. PP-YOLO: an effective and efficient implementation of object detector. *CoRR*, abs/2007.12099, 2020. URL <https://arxiv.org/abs/2007.12099>.
- [110] Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V Le. Dropblock: A regularization method for convolutional networks. *Advances in neural information processing systems*, 31, 2018.
- [111] Jiahui Yu, Yuning Jiang, Zhangyang Wang, Zhimin Cao, and Thomas Huang. Unitbox: An advanced object detection network. In *Proceedings of the 24th ACM international conference on Multimedia*, pages 516–520, 2016.
- [112] Shengkai Wu, Xiaoping Li, and Xinggang Wang. Iou-aware single-stage object detector for accurate localization. *Image and Vision Computing*, 97:103911, 2020.
- [113] Xinlong Wang, Rufeng Zhang, Tao Kong, Lei Li, and Chunhua Shen. Solov2: Dynamic and fast instance segmentation. *Advances in Neural information processing systems*, 33:17721–17732, 2020.

- [114] Rosanne Liu, Joel Lehman, Piero Molino, Felipe Petroski Such, Eric Frank, Alex Sergeev, and Jason Yosinski. An intriguing failing of convolutional neural networks and the coordconv solution. *Advances in neural information processing systems*, 31, 2018.
- [115] PaddlePaddle Authors. Model distillation, 2025. URL [https://github.com/PaddlePaddle/PaddleClas/blob/release/static/docs/en/advanced\\_tutorials/distillation/distillation\\_en.md](https://github.com/PaddlePaddle/PaddleClas/blob/release/static/docs/en/advanced_tutorials/distillation/distillation_en.md). Accessed on January 17, 2025.
- [116] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. Scaled-yolov4: Scaling cross stage partial network. *CoRR*, abs/2011.08036, 2020. URL <https://arxiv.org/abs/2011.08036>.
- [117] Hung-Shuo Chang, Chien-Yao Wang, Richard Robert Wang, Gene Chou, and Hong-Yuan Mark Liao. Yolor-based multi-task learning, 2023. URL <https://arxiv.org/abs/2309.16921>.
- [118] Yuxin Fang, Bencheng Liao, Xinggang Wang, Jiemin Fang, Jiyang Qi, Rui Wu, Jianwei Niu, and Wenyu Liu. You only look at one sequence: Rethinking transformer in vision through object detection. *CoRR*, abs/2106.00666, 2021. URL <https://arxiv.org/abs/2106.00666>.
- [119] Zheng Ge, Songtao Liu, Feng Wang, Zeming Li, and Jian Sun. YOLOX: exceeding YOLO series in 2021. *CoRR*, abs/2107.08430, 2021. URL <https://arxiv.org/abs/2107.08430>.
- [120] Xin Huang, Xinxin Wang, Wenyu Lv, Xiaying Bai, Xiang Long, Kaipeng Deng, Qingqing Dang, Shumin Han, Qiwen Liu, Xiaoguang Hu, Dianhai Yu, Yanjun Ma, and Osamu Yoshie. Pp-yolov2: A practical object detector, 2021. URL <https://arxiv.org/abs/2104.10419>.
- [121] Chuyi Li, Lulu Li, Hongliang Jiang, Kaiheng Weng, Yifei Geng, Liang Li, Zaidan Ke, Qingyuan Li, Meng Cheng, Weiqiang Nie, Yiduo Li, Bo Zhang, Yufei Liang, Linyuan Zhou, Xiaoming Xu, Xiangxiang Chu, Xiaoming Wei, and Xiaolin Wei. Yolov6: A single-stage object detection framework for industrial applications, 2022. URL <https://arxiv.org/abs/2209.02976>.
- [122] Xiaohan Ding, Xiangyu Zhang, Ningning Ma, Jungong Han, Guiguang Ding, and Jian Sun. Repvgg: Making vgg-style convnets great again. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 13733–13742, 2021.
- [123] Chengjian Feng, Yujie Zhong, Yu Gao, Matthew R Scott, and Weilin Huang. Toood: Task-aligned one-stage object detection. In *2021 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 3490–3499. IEEE Computer Society, 2021.
- [124] Xiaohan Ding, Honghao Chen, Xiangyu Zhang, Kaiqi Huang, Jungong Han, and Guiguang Ding. Re-parameterizing your optimizers rather than architectures. *arXiv preprint arXiv:2205.15242*, 2022.

- [125] Changyong Shu, Yifan Liu, Jianfei Gao, Zheng Yan, and Chunhua Shen. Channel-wise knowledge distillation for dense prediction. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5311–5320, 2021.
- [126] Haoyang Zhang, Ying Wang, Feras Dayoub, and Niko Sunderhauf. Varifocalnet: An iou-aware dense object detector. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 8514–8523, 2021.
- [127] Zhora Gevorgyan. Siou loss: More powerful learning for bounding box regression. *arXiv preprint arXiv:2205.12740*, 2022.
- [128] Hamid Rezatofighi, Nathan Tsoi, JunYoung Gwak, Amir Sadeghian, Ian Reid, and Silvio Savarese. Generalized intersection over union: A metric and a loss for bounding box regression. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 658–666, 2019.
- [129] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors, 2022. URL <https://arxiv.org/abs/2207.02696>.
- [130] Glenn Jocher, Ayush Chaurasia, and Jing Qiu. Ultralytics yolov8, 2023. URL <https://github.com/ultralytics/ultralytics>.
- [131] Ruihan Bai, Feng Shen, Mingkan Wang, et al. Improving detection capabilities of yolov8-n for small objects in remote sensing imagery: Towards better precision with simplified model complexity. *Research Square*, June 2023. doi: 10.21203/rs.3.rs-3085871/v1. Preprint, Version 1.
- [132] RangeKing. Brief summary of yolov8 model structure, January 2023. URL <https://github.com/ultralytics/ultralytics/issues/189>. GitHub issue.
- [133] RangeKing. Github repositories, 2025. URL <https://github.com/RangeKing?tab=repositories>. Accessed on January 18, 2025.
- [134] Chien-Yao Wang, I-Hau Yeh, and Hong-Yuan Mark Liao. YOLOv9: Learning what you want to learn using programmable gradient information, 2024. URL <https://arxiv.org/abs/2402.13616>.
- [135] Chien-Yao Wang, Hong-Yuan Mark Liao, and I-Hau Yeh. Designing network design strategies through gradient path analysis, 2022. URL <https://arxiv.org/abs/2211.04800>.
- [136] Ao Wang, Hui Chen, Lihao Liu, Kai Chen, Zijia Lin, Jungong Han, and Guiguang Ding. YOLOv10: Real-time end-to-end object detection, 2024. URL <https://arxiv.org/abs/2405.14458>.
- [137] Glenn Jocher and Jing Qiu. Ultralytics yolo11, 2024. URL <https://github.com/ultralytics/ultralytics>.

- [138] Rahima Khanam and Muhammad Hussain. Yolov11: An overview of the key architectural enhancements. *arXiv preprint arXiv:2410.17725*, 2024.
- [139] Ultralytics. Ultralytics yolo11, 2025. URL <https://docs.ultralytics.com/models/yolo11/>. Accessed on January 19, 2025.
- [140] Chuyi Li, Lulu Li, Yifei Geng, Hongliang Jiang, Meng Cheng, Bo Zhang, Zaidan Ke, Xiaoming Xu, and Xiangxiang Chu. Yolov6 v3.0: A full-scale reloading, 2023. URL <https://arxiv.org/abs/2301.05586>.
- [141] Shay Aharon, Louis-Dupont, Ofri Masad, Kate Yurkova, Lotem Fridman, Lkdci, Eugene Khvedchenya, Ran Rubin, Natan Bagrov, Borys Tymchenko, Tomer Keren, Alexander Zhilko, and Eran-Dei. Super-gradients, 2021. URL <https://zenodo.org/record/7789328>.
- [142] Tianheng Cheng, Lin Song, Yixiao Ge, Wenyu Liu, Xinggang Wang, and Ying Shan. Yolo-world: Real-time open-vocabulary object detection, 2024. URL <https://arxiv.org/abs/2401.17270>.
- [143] Rishi Bommasani, Drew A. Hudson, Ehsan Adeli, Russ B. Altman, Simran Arora, Sydney von Arx, Michael S. Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, Erik Brynjolfsson, Shyamal Buch, Dallas Card, Rodrigo Castellon, Niladri S. Chatterji, Annie S. Chen, Kathleen Creel, Jared Quincy Davis, Dorottya Demszky, Chris Donahue, Moussa Doumbouya, Esin Durmus, Stefano Ermon, John Etchemendy, Kawin Ethayarajh, Li Fei-Fei, Chelsea Finn, Trevor Gale, Lauren E. Gillespie, Karan Goel, Noah D. Goodman, Shelby Grossman, Neel Guha, Tatsunori Hashimoto, Peter Henderson, John Hewitt, Daniel E. Ho, Jenny Hong, Kyle Hsu, Jing Huang, Thomas Icard, Saahil Jain, Dan Jurafsky, Pratyusha Kalluri, Siddharth Karamcheti, Geoff Keeling, Fereshte Khani, Omar Khattab, Pang Wei Koh, Mark S. Krass, Ranjay Krishna, Rohith Kuditipudi, and et al. On the opportunities and risks of foundation models. *CoRR*, abs/2108.07258, 2021. URL <https://arxiv.org/abs/2108.07258>.
- [144] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. *CoRR*, abs/2103.00020, 2021. URL <https://arxiv.org/abs/2103.00020>.
- [145] Maxime Bucher, Tuan-Hung Vu, Matthieu Cord, and Patrick Pérez. Zero-shot semantic segmentation. *CoRR*, abs/1906.00817, 2019. URL <http://arxiv.org/abs/1906.00817>.
- [146] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C. Berg, Wan-Yen Lo, Piotr Dollár, and Ross Girshick. Segment anything, 2023.

- [147] Shilong Liu, Zhaoyang Zeng, Tianhe Ren, Feng Li, Hao Zhang, Jie Yang, Qing Jiang, Chunyuan Li, Jianwei Yang, Hang Su, Jun Zhu, and Lei Zhang. Grounding dino: Marrying dino with grounded pre-training for open-set object detection, 2024. URL <https://arxiv.org/abs/2303.05499>.
- [148] Ankan Bansal, Karan Sikka, Gaurav Sharma, Rama Chellappa, and Ajay Divakaran. Zero-shot object detection. *CoRR*, abs/1804.04340, 2018. URL <http://arxiv.org/abs/1804.04340>.
- [149] Grounded-SAM Contributors. Grounded-Segment-Anything, April 2023. URL <https://github.com/IDEA-Research/Grounded-Segment-Anything>.
- [150] Autodistill. Autodistill: Groundedsam base model, 2025. URL <https://github.com/autodistill/autodistill-grounded-sam>.
- [151] Tianhe Ren, Shilong Liu, Ailing Zeng, Jing Lin, Kunchang Li, He Cao, Jiayu Chen, Xinyu Huang, Yukang Chen, Feng Yan, Zhaoyang Zeng, Hao Zhang, Feng Li, Jie Yang, Hongyang Li, Qing Jiang, and Lei Zhang. Grounded sam: Assembling open-world models for diverse visual tasks, 2024. URL <https://arxiv.org/abs/2401.14159>.
- [152] Roboflow. Supervision. URL <https://github.com/roboflow/supervision>.
- [153] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *Computer Vision—ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, pages 740–755. Springer, 2014.
- [154] Shen Li, Yanli Zhao, Rohan Varma, Omkar Salpekar, Pieter Noordhuis, Teng Li, Adam Paszke, Jeff Smith, Brian Vaughan, Pritam Damania, and Soumith Chintala. Pytorch distributed: Experiences on accelerating data parallel training, 2020. URL <https://arxiv.org/abs/2006.15704>.
- [155] Glenn Jocher, Jing Qiu, and Ayush Chaurasia. Ultralytics YOLO, January 2023. URL <https://github.com/ultralytics/ultralytics>.
- [156] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory F. Diamos, Erich Elsen, David García, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. Mixed precision training. *CoRR*, abs/1710.03740, 2017. URL <http://arxiv.org/abs/1710.03740>.
- [157] G Bradski. The opencv library. *Dr. Dobb’s Journal of Software Tools*, 2000.
- [158] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

- [159] B Dwyer, J Nelson, J Solawetz, et al. Roboflow (version 1.0)[software]. URL: <https://roboflow.com>. *computer vision*, 2022.
- [160] Rafael Padilla, Sergio L. Netto, and Eduardo A. B. da Silva. A survey on performance metrics for object-detection algorithms. *2020 International Conference on Systems, Signals and Image Processing (IWSSIP)*, pages 237–242, 2020. URL <https://api.semanticscholar.org/CorpusID:220734135>.
- [161] Rafael Padilla, Wesley L Passos, Thadeu LB Dias, Sergio L Netto, and Eduardo AB Da Silva. A comparative analysis of object detection metrics with a companion open-source toolkit. *Electronics*, 10(3):279, 2021.
- [162] Ultralytics. Yolo performance metrics, 2025. URL <https://docs.ultralytics.com/guides/yolo-performance-metrics/>. Accessed on January 20, 2025.
- [163] Paul Jaccard. Étude comparative de la distribution florale dans une portion des alpes et des jura. *Bull Soc Vaudoise Sci Nat*, 37:547–579, 1901.
- [164] Slamet Riyanto, Sukaesih Sitanggang Imas, Taufik Djatna, and Tika Dewi Atikah. Comparative analysis using various performance metrics in imbalanced data for multi-class text classification. *International Journal of Advanced Computer Science and Applications*, 14(6), 2023.
- [165] Marina Sokolova and Guy Lapalme. A systematic analysis of performance measures for classification tasks. *Information processing & management*, 45(4):427–437, 2009.
- [166] COCO Consortium. Coco detection challenge (bounding box). <https://competitions.codalab.org/competitions/20794>, 2019. Accessed: January 06, 2025.
- [167] Mark Everingham, SM Ali Eslami, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes challenge: A retrospective. *International journal of computer vision*, 111:98–136, 2015.
- [168] Loris Nanni, Sheryl Brahnam, Stefano Ghidoni, Emanuele Menegatti, and Tonya Barrier. Different approaches for extracting information from the co-occurrence matrix. *PloS one*, 8(12):e83554, 2013.
- [169] Ilya Loshchilov and Frank Hutter. Fixing weight decay regularization in adam. *CoRR*, abs/1711.05101, 2017. URL <http://arxiv.org/abs/1711.05101>.
- [170] Jan Hendrik Hosang, Rodrigo Benenson, and Bernt Schiele. Learning non-maximum suppression. *CoRR*, abs/1705.02950, 2017. URL <http://arxiv.org/abs/1705.02950>.
- [171] Xiang Li, Chengqi Lv, Wenhai Wang, Gang Li, Lingfeng Yang, and Jian Yang. Generalized focal loss: Towards efficient representation learning for dense object detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(3):3139–3153, 2023. doi: 10.1109/TPAMI.2022.3180392.

- 
- [172] Pierre Le Jeune and Anissa Mokraoui. Rethinking intersection over union for small object detection in few-shot regime. *arXiv preprint arXiv:2307.09562*, 2023.

## Appendix A

### Supplementary Material

#### A.1 Qualitative Evaluation: YOLO11m

This section presents the qualitative evaluation of the YOLO11m model during the testing phase, using visualizations of test batches. The visualizations compare the ground truth labels and the predicted labels for selected batches from the validation and test dataset, offering insights into the model’s generalization capability on unseen data. This serves as the qualitative evaluation step to assess the model’s reliability for deployment in real-time object detection within the CARLA simulation environment.

To begin, we examine the visualization of a training batch used during the training phase of the YOLO11m model, depicted in Figure A.1. This figure highlights how the different object classes—Pedestrians, Traffic Signs, Vehicles, and Traffic Light—are numerically labeled as 0, 1, 2, and 3, respectively. This labeling schema is consistent throughout the dataset and is essential for training the model to differentiate between the four classes effectively.

Next, the visualization of ground truth labels for validation batch 0 is shown in Figure A.2, followed by the corresponding model predictions for the same batch in Figure A.3. Moving to the testing phase, Figure A.4 illustrates the ground truth labels for test dataset batch 1, while Figure A.5 shows the predictions made by the YOLO11m model for the same batch. These visualizations assess the model’s generalization capability after being fully trained for 100 epochs. The results demonstrate how effectively the YOLO11m model generalizes to unseen data.

Collectively, these visualizations highlight the robustness and reliability of the YOLO11m model for 2D object detection in real-world applications.

#### A.2 Confusion Matrix in Scenario A

The confusion matrix A.6 illustrates the performance of the YOLO11m model on the test dataset in scenario (A) across four classes: Pedestrian, Traffic\_Signs, Vehicle, and Traffic\_Light. Among these, the Vehicle class achieved the highest accuracy with 1233 true positives, highlighting the model’s strong detection capabili-

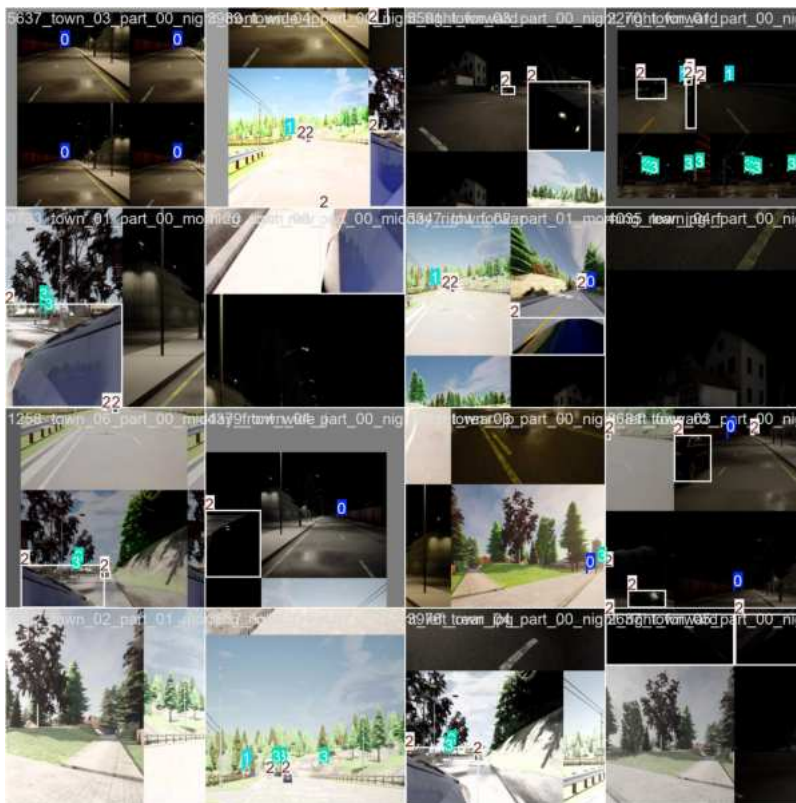


Figure A.1: Visualization of the training batch 0 during YOLO11m model training. The figure shows how the dataset numerically encodes classes (0: Pedestrian, 1: Traffic Signs, 2: Vehicle, 3: Traffic Light).

ties for this category. However, misclassifications are notable, particularly with Traffic\_Light (940), Vehicle (144), Pedestrian (91), and Traffic\_Signs (84) being frequently confused with the Background, reflecting challenges in distinguishing these objects from non-object regions. While Traffic\_Lights showed good detection performance with 822 true positives, they were also significantly misclassified as Background (940). These results indicate that, despite strong performance for Vehicles, improvements are needed in detecting smaller or less distinct classes such as Traffic\_Lights, Pedestrians, and Traffic\_Signs, as well as in minimizing confusion with the Background.

### A.3 Trade-Off Between Detection Accuracy and Inference Speed

**Scenario A:** Figure A.7 illustrates the trade-off between inference speed and mAP(50) for all YOLO models in scenario A. In the comparative evaluation depicted in the plot, models such as YOLOv8n, YOLOv9t, and YOLO11n distinguish themselves by their moderate detection accuracy and rapid inference times, making them particularly suitable for real-time applications where immediacy is essential. Notably, YOLOv11s

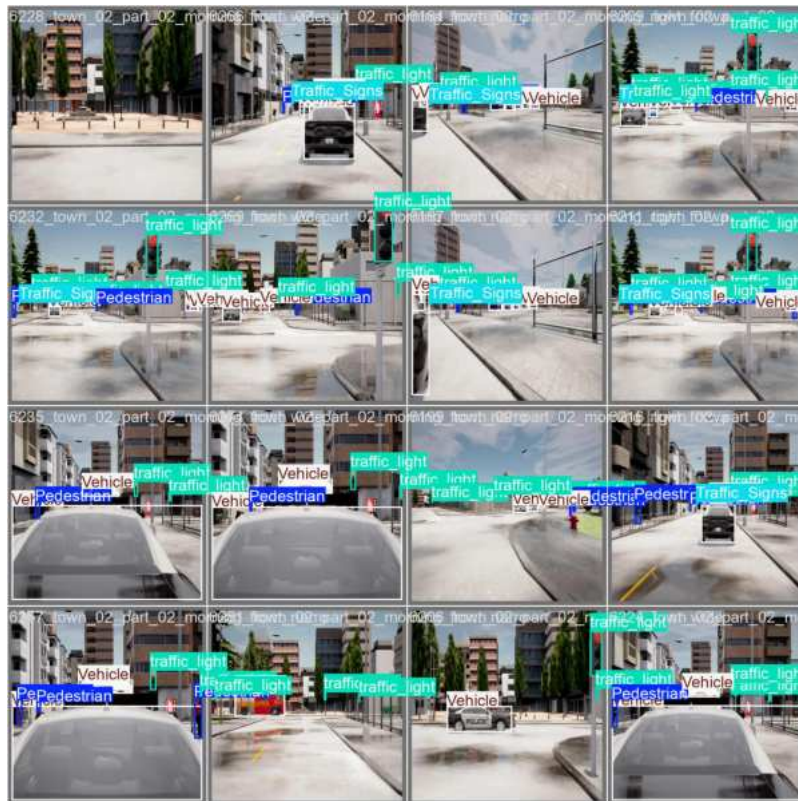


Figure A.2: Ground truth labels visualization for validation batch 0 during YOLO11m model training. This figure depicts the expected annotations for objects in the dataset.

also exhibits a commendable balance, achieving one of the highest detection accuracies or mAP(50) score with only a modest increase in inference time, as illustrated. However, YOLOv8n and YOLO11n, with their respective inference speeds of 1.1 ms and 1.5 ms, do compromise on accuracy compared to their more balanced counterparts, YOLOv8m and YOLO11m. This trade-off is clearly visible in the plot, where YOLOv8m and YOLO11m demonstrate an optimal balance, offering robust mAP(50) scores while maintaining reasonable inference speeds. Given that ultra-high inference speed is not a critical requirement for this project, these models provide an ideal blend of detection accuracy and performance efficiency. Conversely, in scenarios that prioritize compact model size and faster operational capacity—such as deployment on edge devices—YOLOv8n is shown to be the preferred choice due to its smaller footprint and expedited processing capabilities, as indicated in the plot’s positioning of the models.

**Scenario B:** The figure A.8 highlights the trade-off between model accuracy and inference speed across various YOLO models in scenario B. Models such as YOLOv8m, YOLOv9m, and YOLO11m achieve high mAP(50) and mAP(50-95) scores, indicating strong object detection performance, but come at the cost

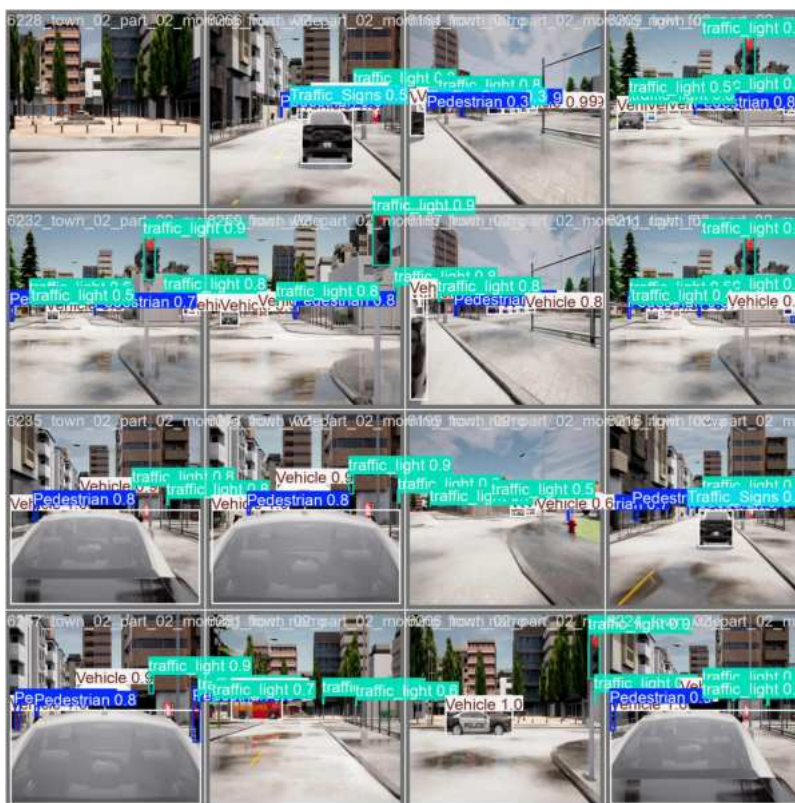


Figure A.3: Predicted labels visualization for validation batch 0 during YOLO11m model inference on the validation dataset. This figure shows how the model identifies objects based on the ground truth.

of slower inference speeds (5.0–5.6 ms per image). Conversely, models like YOLO11n, YOLOv10n and YOLOv8n exhibit the fastest inference speeds (1.6–1.8 ms per image), making them ideal for deployment on edge devices where size, and inference speed are of paramount importance, but with lower mAP scores compared to their larger counterparts.

For applications requiring a balance between accuracy and speed, YOLO11m and YOLOv8m offers compelling options, delivering strong accuracy metrics (highest mAP(50-95)) while maintaining a reasonable inference speed (5.1 ms per image).

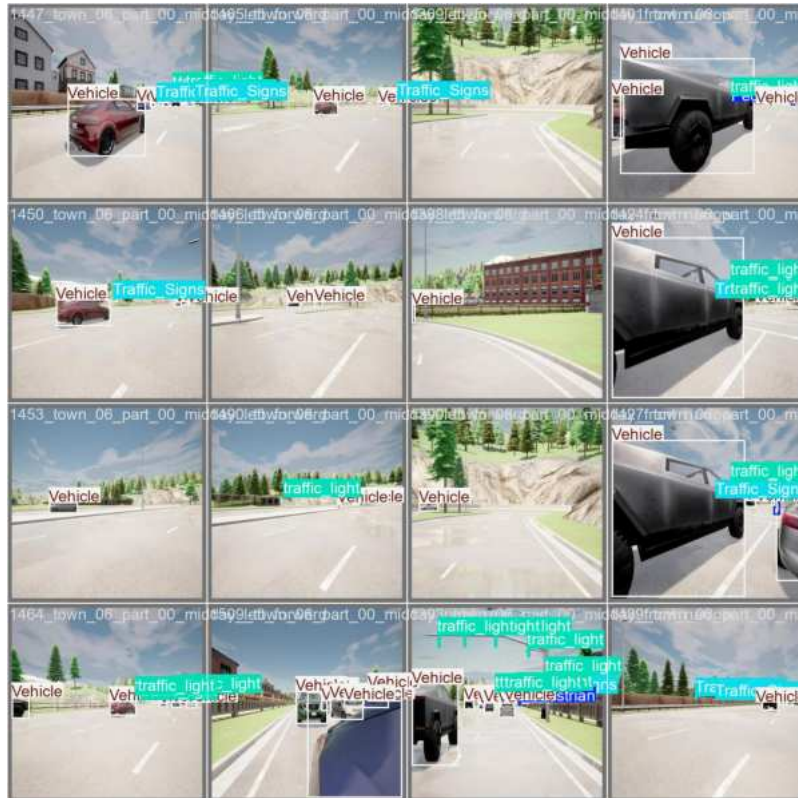


Figure A.4: Ground truth labels visualization for test dataset batch 1 during YOLO11m model inference. This figure illustrates the manually annotated labels for the test images.



Figure A.5: Predicted labels visualization for test dataset batch 1 during YOLO11m model inference. This figure shows how the model predicts objects in the test dataset.

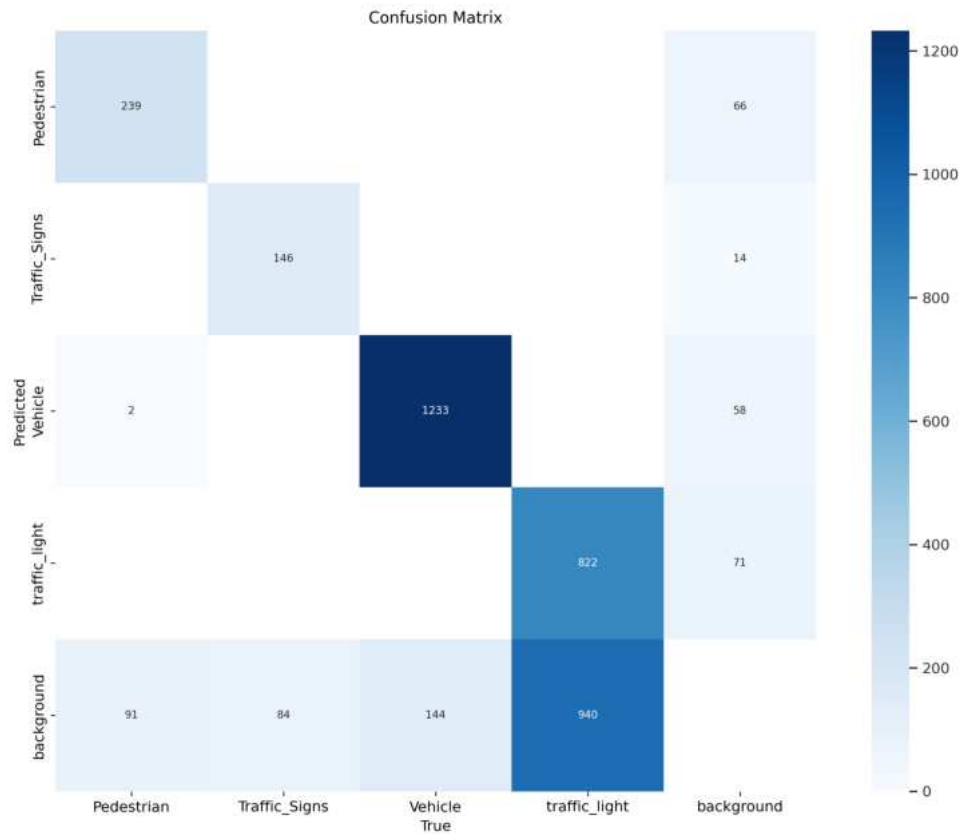


Figure A.6: Confusion matrix for the YOLO11m model on test set, highlighting its classification accuracy across all object classes.

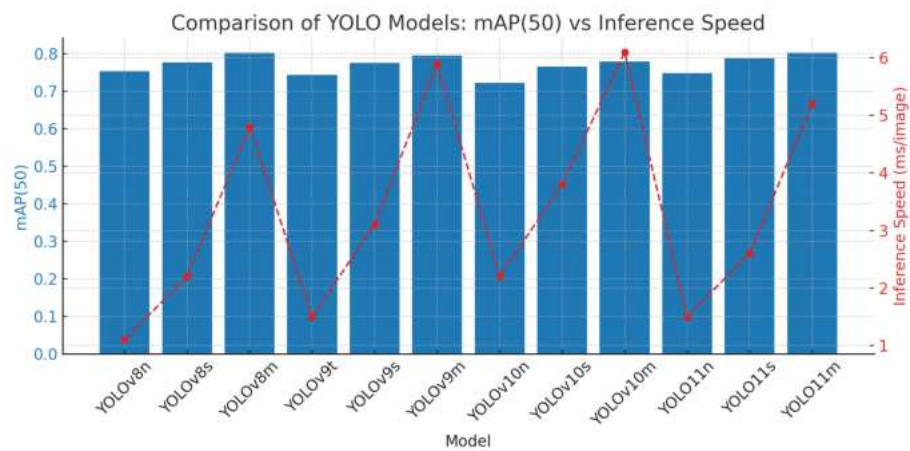


Figure A.7: Trade-off between mAP(50) and inference speed for all YOLO models based on test dataset in Scenario A. The bar chart represents the mAP(50) values, and the line graph shows the inference speeds for each model.

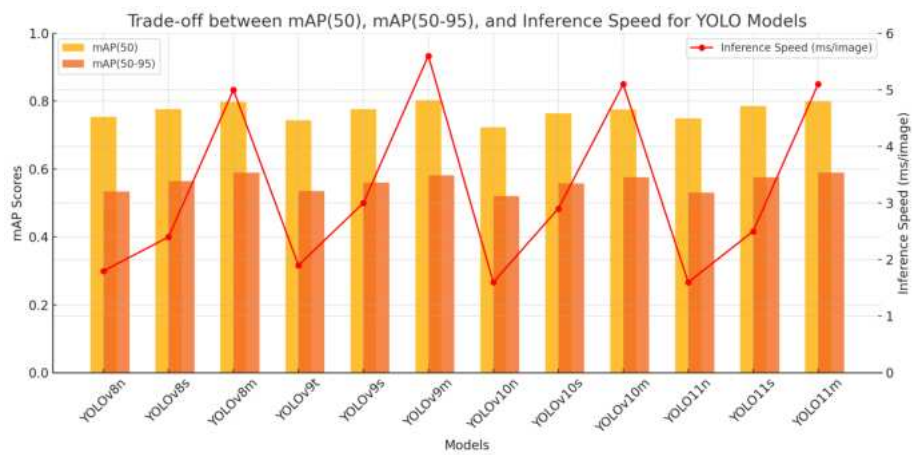


Figure A.8: The plot above illustrates the trade-off between mAP(50), mAP(50-95), and inference speed for various YOLO models tested on the test dataset in scenario B. The bar chart displays mAP(50) and mAP(50-95) on the left y-axis, while the line plot shows inference speed in milliseconds per image on the right y-axis. This visualization highlights the performance and efficiency of each model, aiding in the selection of a model that balances accuracy and speed based on specific application needs.