October 2022 José Miguel Fernandes Madeira Pinto **Automatic Driving: 2D Detection and Tracking using Artificial Intelligence Techniques**

José Miguel Fernandes Madeira Pinto

Automatic Driving: 2D Detection and Tracking using Artificial Intelligence Techniques

October 2022

Universidade do Minho
Escola de Engenharia

José Miguel Fernandes Madeira Pinto

Automatic Driving: 2D Detection and Tracking using Artificial Intelligence Techniques

Master Dissertation
Integrated Master in Informatics Engineering

Dissertation supervised by
Victor Alves
Helena López

October 2022

DECLARATION

Name: José Miguel Fernandes Madeira Pinto

Dissertation Title: Artificial Intelligence Applied to Object Detection and Tracking

Advisors: Helena Fernández López, Victor Manuel Rodrigues Alves

Conclusion Year: 2022

Master Designation: Mestrado Integrado em Engenharia Informática

Universidade do Minho, 31 / 10 / 2022

Signature: José Pinto

ACKNOWLEDGEMENTS

The realization of this master's thesis counted on important support and incentives, without which it would not have become a reality. I will be eternally grateful for all the learning and support I had throughout my academic journey.

To the University of Minho, professors and collaborators who have always provided me with the necessary means to grow, in knowledge and as a person. To my supervisors Victor Alves and Helena López, for their guidance, total support, availability, for the knowledge they conveyed, opinions and criticisms, total collaboration in solving the problems that have arisen during the course of this work and for all the words of incentive. To my friends and colleagues, I met along my academic path, who were by my side with their companionship, strength, and support. My cousin Márcia and Ricardo for their friendship, strength and help on this journey. My godmother, uncle, and cousins, for their friendship and for all the support and advice on this challenging journey. My grandmother for her friendship, love, and support. To my brother, who has always been by my side with his friendship, patience, companionship, strength, and help. Finally, being aware that none of this would have been possible alone, I would like to thank my parents, for being models of courage, for their unconditional support, encouragement, friendship and help in overcoming the obstacles that have arisen along this journey.

I dedicate this work to all of them!

Statement of Integrity

I hereby declare having conducted this academic work with integrity. I confirm that I have not used plagiarism or any form of undue use of information or falsification of results along the process leading to its elaboration.

I further declare that I have fully acknowledged the Code of Ethical Conduct of the University of Minho.

Universidade do Minho, 31 / 10 / 2022

Signature: José Pinto

ABSTRACT

Road accidents are estimated to be the cause of millions of deaths and tens of millions of injuries every year. For this reason, any measure that reduces accidents' probability or severity will save lives.
Speeding, driving under the influence of psychotropic substances and distraction are leading causes of road accidents. Causes that can be classified as human since they all come from driver errors. Autonomous driving is a potential solution to this problem as it can reduce road accidents by removing human error from the task of driving.
This dissertation aims to study Artificial Intelligence techniques and Edge Computing networks to explore solutions for autonomous driving. To this end, Artificial Intelligence models for detecting and tracking objects based on Machine Learning and Computer Vision, and Edge Computing networks for vehicles were explored.
The YOLOv5 model was studied for object detection, in which different training parameters and data pre-processing techniques were applied. For object tracking, the StrongSORT model was chosen, for which its performance was evaluated for different combinations of its components. Finally, the Simu5G simulation tool was studied in order to simulate an edge computing network, and the viability of this type of network to aid autonomous driving was analysed.

Keywords: Autonomous Driving, Artificial Intelligence, Machine Learning, Computer Vision, Edge Computing.

Resumo

É estimado que os acidentes rodoviários sejam a causa de milhões de mortes e dezenas de milhões de lesões todos os anos. Por esta razão, qualquer medida que diminua a probabilidade de acidentes ou que diminua a sua gravidade acabará por salvar vidas.

Excesso de velocidade, condução sob influência de substâncias psicotrópicas e distração no ato da condução são algumas das principais causas de acidentes rodoviários. Causas essas que podem ser classificadas como humanas visto que são oriundas de um erro do condutor.

A condução autónoma surge como solução para este problema. Esta tem o potencial de diminuir acidentes rodoviários removendo o erro humano da tarefa da condução.

Esta dissertação teve como objetivo o estudo de técnicas Inteligência Artificial e redes Computação de Borda de forma a explorar soluções para a condução autónoma. Para tal foram estuados modelos Inteligência Artificial de deteção e rastreamento de objetos com base nas áreas de Aprendizagem Máquina e Visão por Computador e redes de Computação de Borda para veículos.

Para a deteção de objetos foi estudado o modelo YOLOv5, no qual diferentes combinações de parâmetros de treino e técnicas de pré-processamento de dados foram aplicadas. Para o rastreamento de objetos foi escolhido o modelo StrongSORT, para o qual foi avaliada a sua performance para diferentes combinações das suas componentes. Por fim, foi estudada a ferramenta de simulação Simu5G, de forma a simular uma rede de computação de borda, e foi feita uma análise sobre a viabilidade deste tipo de redes no auxílio à condução autónoma.

Palavras-Chave: Condução Autónoma, Inteligência Artificial, Aprendizagem de Máquina, Visão por Computador, Computação de Borda.

.

Table of Contents

LIST OF FIGURES

List of Tables

List of Abbreviations and acronyms

A

AP	Average Precision
AI	Artificial Intelligence
ANN	Artificial Neural Network

B

BoT	Bag of Tricks

C

CNN	Convolutional Neural Network
CV	Computer Vision

D

DL	Deep Learning
DSR	Design Science Research

E

EC	Edge computing
ECC	Enhanced Correlation Coefficient Maximization
EMA	Exponential Moving Average

F

FPS	Frame Per Second

G

GA	Genetic algorithm

H

HOG	Histogram of Oriented Gradients

L

LIDAR	Light Detection and Ranging

M

MEC	Multi-access Edge Computing
ML	Machine Learning
MOT	Multiple Object Tracking
MOTA	Multiple Object Tracking Accuracy
MOTP	Multiple Object Tracking Precision
MS-COCO	Microsoft COCO
mAP	Mean Average Precision

O

OSNet	Omni-Scale network

R

RL	Reinforcement Learning
RPN	Region Proposal Network
re-ID	re-Identification

S

SL	Supervised Learning
SOT	Single Object Tracking
SSD	Singe Shot Multi-Box Detector

U

UE	User Equipment
UL	Unsupervised Learning

Y

YOLO	You Only Look Once

1 Introduction

1.1 CONTEXT AND MOTIVATION

Autonomous driving promises to revolutionize the transportation industry. Automating driving requires vehicles to understand the surrounding space, and to have this capability, they must do various vital tasks. Two of these essential tasks are object detection and object tracking. These two tasks turn the vehicle capable of not only detecting objects around it and classifying them into various classes and subclasses but also making it capable of tracking the movements of the objects in different instances of time [1], [2]. These tasks are crucial for detecting immediate danger, predicting the future behaviour of objects around the vehicle, and calculating the best paths.

Self-driving vehicles have six levels that define the degree of driving automation. Level 0 is no automation, and Level 5 is full automation. Nowadays, the highest level of autonomy achieved is Level 4, which means that the vehicle can perform all driving tasks under specific circumstances and by being limited to a specific geographical area [3].

The guarantees needed for a system to have autonomy higher than Level 4 are difficult to achieve with the technology used nowadays, and the reason for this difficulty is that autonomous driving is a critical service with needs for fast and precise outputs. The lack of precision or speed in object detection and tracking can lead to catastrophic consequences. For this reason, when building an architecture, developers need to balance the speed and precision in a way that both have acceptable values for real-world use. The struggle is that more precise models usually need more computational power, meaning their execution time is longer [4].

Furthermore, in the autonomous driving industry, there is an ongoing debate about using Light Detection and Ranging (LIDAR) sensors that help make a 3D render of the space around the vehicle. Some are beating on using these sensors alongside camera systems for retrieving all input information for autonomous driving. Others defend that fully automated driving with only camera systems can be achieved [5].

At first glance, LIDAR sensors with cameras may seem the way to go since they make a 3D render of the space around the vehicle, which is not only new information but also gives the system redundancy that can help with camera malfunctions. However, LIDAR has its cons. For example, it is affected by variations in temperature, poor signal-to-noise ratio, and the fact that it is an expensive piece of equipment [5].

Knowing that autonomous driving requires precise outputs that need considerable computational power leads us to limitations in the hardware inside a vehicle. When building a system architecture for autonomous driving software, developers need to consider the hardware's limitations inside the vehicle. With such capable cloud services available, developers may wonder what would happen if they could run the entire computation workload of autonomous driving on a cloud-based system. It would drop computational times and give them room for making more computationally hungry but precise architectures. Unfortunately, this is not a viable option for an autonomous driving service that needs to work around the globe. The latency of communication with datacenters potentially far from the end-user does not reach the required speed required for autonomous driving [6].

Edge computing is a mid-point between cloud-based and on-premises computing. It provides a more computation-capable machine than the end-device hardware and lower latency than cloud-based computation [6]. Low latency real-time applications are the use cases where Edge Computing shines, which turns it into a potentially good solution for the object detection and tracking tasks of autonomous driving.

The processing of data collected by embedded sensors in the vehicle may require that this vehicle be endowed with high computational capacity. On the other hand, taking advantage of edge computing resources can improve the algorithms' accuracy, reduce the algorithms' execution time, and cut down costs in terms of the necessary resources. This dissertation aims to contribute to the emerging use of edge computing to advance autonomous driving and experiment with Artificial Intelligence (AI) techniques using real-world data from embedded sensors.

1.2 OBJECTIVES

This work aims to develop an edge application for object detection and tracking, in addition to the implementation and testing of a simulated edge computing network.

These tasks will be achieved in the following steps: the first step is to acquire and analyse object detection and tracking data for autonomous driving; the second and third steps are the training and testing of AI models in 2D Object Detection and 2D Object Tracking, respectively; the final step involves the development of an edge computing network that will make it possible to achieve the objective of this work.

1.3 INVESTIGATION METHODOLOGY

Rigorous research is required to gather knowledge, so the appropriate methods, tools, and strategies are applied to develop the best possible solution. The research strategy selected for this dissertation was Design Science Research (DSR). DSR is a scientific methodology of problem-solving that involves creating new knowledge through the design of innovative artefacts, analysis of use and performance of these artefacts to improve information systems [7].

The DSR methodology was applied into the fallowing stages [7]:

1. Problem identification and motivation – The research problem identified was using edge computing to execute object detection and tracking algorithms and return results to the appropriate users for autonomous driving. The motivations behind the interest in this problem were the emerging and promising application of edge computing and autonomous driving solutions.
2. Definition of the solution's objectives - The objectives traced for the research problem resolution were the research, selection, and training of object detection and tracking model and the implementation of a simulated edge network.
3. Design and development – Object detection and tracking models training and choosing an environment for MEC app testing. This stage of the methodology can be found in the sections 3.3.3, 4.3 and 5.2.
4. Demonstration – Test the edge application developed in the simulation tool identified in the previous phase. This stage can be found in section 5.2.
5. Evaluation – The developed AI models were tested and evaluated by the appropriate metrics. In addition, the simulated edge network was also scrutinised by evaluating the response times given from the edge server to the end-users, representing the autonomous cars. These analyses can be found in sections 3.4, 4.4 and 5.3.
6. Communication – The problem and the utility of the created artefacts for autonomous driving and the application of edge computing in future solutions is discussed in this document.

1.4 STRUCTURE OF THE DISSERTATION

This dissertation is divided into six chapters. Chapter 1 comprises the contextualisation and motivation of the problem, the dissertation objectives, investigation methodology and structure. Chapter 2 introduces the concepts and technologies used in this dissertation. Chapter 3 refers to the development of the object detection model, whereas chapter 4 refers to the development of the tracking model. Chapter 5 is referent to the simulated edge computing network development. Finally, chapter 6 is composed of a summary of the work, the conclusions drawn and future work.

2 Technologies and Concepts

2.1 AUTONOMOUS DRIVING

Autonomous driving refers to a vehicle or transport system capable of transporting people or goods without human intervention. An autonomous vehicle can go anywhere a traditional vehicle can and do everything that an experienced human driver does.

Autonomous vehicles create and maintain a map of their surroundings based on various sensors in different parts of the vehicle. Video cameras, radar, LIDAR sensors and ultrasonic sensors are some of the tools used by this type of vehicle to collect information. The software then processes the sensory input, plots a path, and sends instructions to the vehicle's actuators, which control acceleration, braking, and steering.

The concept of a self-driving vehicle is used interchangeably with an autonomous vehicle, although it was a different meaning. A self-driving vehicle can drive itself in some situations, but unlike an autonomous vehicle, a human passenger must be present and ready to take control.

The international society of automotive engineers defined the standard of the various development levels up to fully autonomous vehicles. As can be seen in Table 1, adapted from [3], there are currently six levels for autonomous driving, ranging from Level 0 (no automation) up to Level 5 (full vehicle autonomy) [3].

Table 1 - Levels of driving automation.

0	1	2	3	4	5
The driver performs all driving tasks, even when enhanced by active safety systems.	The driving automation system of either the lateral or the longitudinal vehicle motion control subtask, but the driver performs the remaining driving tasks.	Driving automation system of both the lateral and longitudinal vehicle motion control subtasks of the driving, but the driver supervises all the tasks and can take control at any time.	Automated driving system of the entire driving task with the expectation that the user is ready to take control.	Automated driving system of the entire driving task and fallback without any expectation that a user will need to intervene under specific conditions.	Automated driving system of the entire driving task and fallback without any expectation that a user will need to intervene under all conditions.

The upsides of an autonomous vehicle are numerous. It promises convenience and quality-of-life improvements like offering more freedom to people, reducing traffic, lowering transportation costs, freeing up city space and reducing CO2 emissions. As a result, there is much motivation to push for autonomous technology.

However, it is not just a question of automated vehicles being able to set themselves in motion. Safety will continue to be a top priority in the future, and passenger comfort will become even more critical. Autonomy faces many challenges that need to be overcome. The challenges range from the technological and legislative to the environmental and philosophical, and all of them must be tackled to achieve true autonomy.

2.2 ARTIFICIAL INTELLIGENCE

AI refers to computing systems capable of performing activities by mimicking human behaviour and performing tasks by learning and problem-solving [7].

Within AI, there is Machine Learning (ML), as described in Figure 3, which is an approach to achieving AI, where a computer is trained using algorithms, analytics, and a large amount of data to build predictive models without relying on rules-based programming [7]. ML algorithms can be divided into three main categories, as Figure 1 illustrates: Supervised Learning (SL), Unsupervised Learning (UL) and Reinforcement Learning (RL) [8], [9].

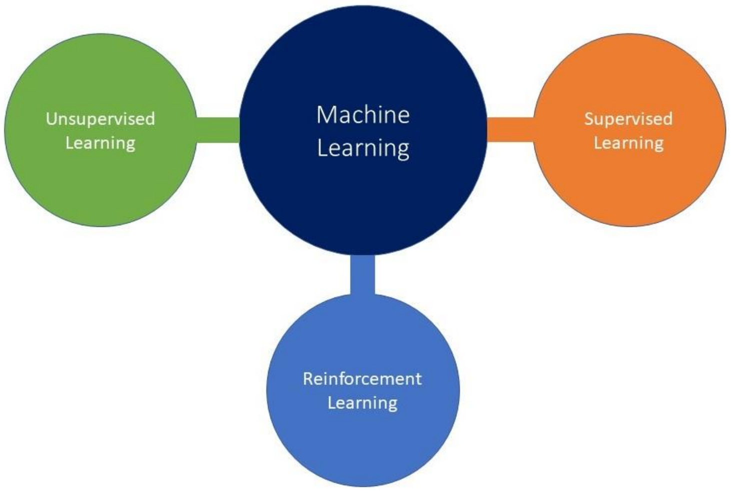


Figure 1 - Sub-fields of Machine Learning.

SL uses categorized training data, where the input and the desired output are already known. With this method, systems can predict outputs for unseen data based on past data. In the case of UL, training data is not labelled and has no known result. Instead, algorithms deduce patterns from unlabelled data on their own. Finally, RL tries to model agents in an environment that rewards the agent based on successful actions concerning completing the task goal [8], [9].

One method to achieve ML is using Artificial Neural Networks (ANN). ANN are generalizations of mathematical models of biological nervous systems. ANN aims to capture non-linear patterns in data by adding layers of parameters to the model. The architecture of an ANN can be found in Figure 2. An ANN is composed of artificial neurons, the basic neural network processing elements. An ANN is composed of artificial neurons, the basic neural network processing elements. An artificial neuron is, at its core, a variable that holds a number. However, it is best represented as a function that takes the weights, values that represent an edge between neurons of consecutive layers, and the bias that is an offset that ensures that the output is not only affected by the weights. This type of function is called an activation function, and its result dictates if a neuron is or is not activated. Neurons are organized in layers within an ANN, and the activated neurons of one layer dictate the activations in the following layer. At a basic level, a neural network comprises four main components: inputs, weights, a bias, and an output. The basic workflow of an ANN is that input nodes receive information, which is expressed numerically and activates a group of neurons. This group of neurons represents a specific pattern that will cause a specific pattern of activations in the following layer, making the information flow from node to node until it arrives at the output layer that presents the model's predicted value [7], [10].

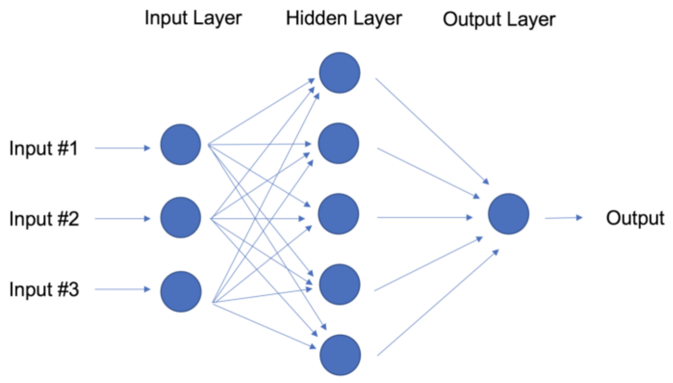


Figure 2 - Artificial Neural Network.

A loss function calculates the difference between the predicted and actual values as training errors. The loss function enables the network to learn by adjusting the weights and biases per the chosen learning algorithm. The learning is done by minimizing the loss function by calculating the loss function gradient that specifies how the weights and biases values should vary.

Deep Learning (DL) is a subfield of ML, as represented in Figure 3, which uses ANNs with multiple hidden layers.

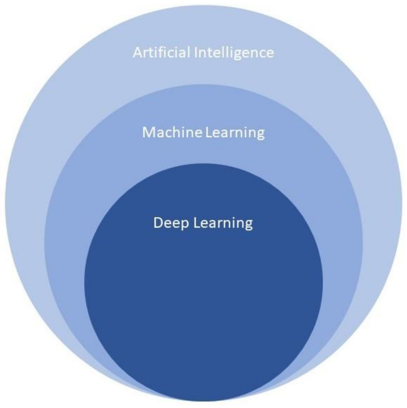


Figure 3 - How do Artificial Intelligence, Machine Learning and Deep Learning compare.

These multiple layers are organized in cascade and form a hierarchical feature representation. The layers closer to the input layer learn simple features, while layers closer to the output layer learn more complex features derived from the features of the previous layers. This type of architecture is known as Deep Neural Network and is represented in Figure 4 [7], [10], [11].

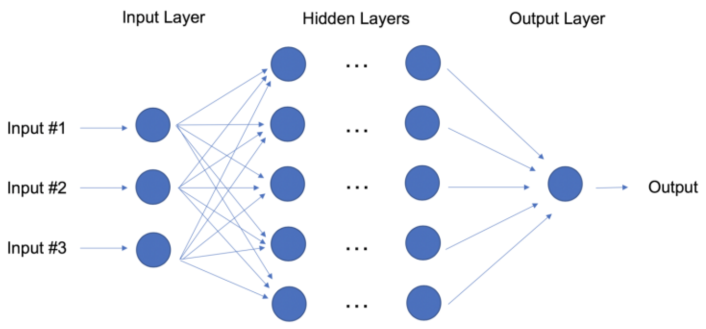


Figure 4 - Deep Neural Network.

2.3 COMPUTER VISION

Computer Vision (CV) is a broad scientific field that deals with the ability of computers to gain high-level understanding through digital images and videos. Recent advances in computing power, memory, and consumption of computers, as well as the abundance of available data, have led to the introduction of DL in CV, as illustrated in Figure 5. Although there are still domains within CV where the use of traditional algorithms produces better results, excelling for its simplicity, for the most part, CV has directed its

research to the application of DL models. DL methods mostly improve prediction performance using big data and plentiful computing resources and have pushed the boundaries of what was possible [12].
CV's most common real-world applications include image classification, image generation, object detection, semantic segmentation, and object tracking. These applications can be seen in technology like self-driving cars, natural language processing, visual recognition, image and speech recognition, virtual assistants, chatbots, fraud detection, medical image analysis, and others [13], [14].

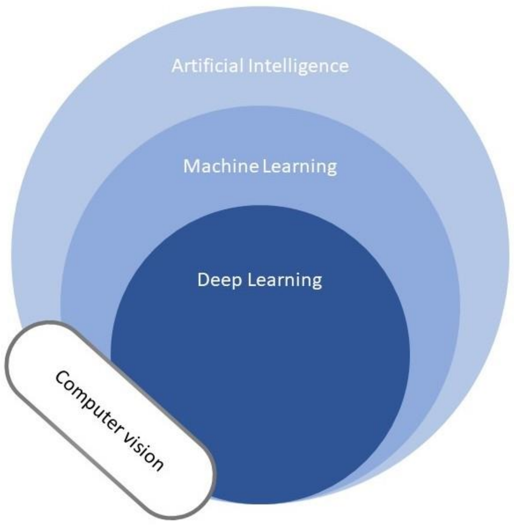


Figure 5 - Artificial Intelligence and Computer Vision.

2.4 EDGE COMPUTING

Cloud computing is also a computing paradigm that offers on-demand services to the end-users through a pool of computing resources. Edge computing (EC) is a distributed computing paradigm that brings the service and utilities of cloud computing closer to the end-user. The main difference between EC and cloud computing lies in the location of the servers. Cloud computing has a centralized model that can lead to high distances between the cloud server and the end-user, meaning high latency. On the other hand, EC uses a distributed model that offers the advantage of lower distances and, therefore, lower latencies. This approach also has drawbacks since EC is not as scalable as cloud computing and has hardware with limited capabilities [15]. The unique characteristics of EC are described in Table 2 [15]. EC has three computing models to resolve the cloud computing issues. These models and their descriptions can be found in Table 3 [15].

Table 2 - Edge Computing característics.

Dense geographical distribution	Numerous computing platforms in the edge networks.
Mobility support	The host identity from and the location identity are decupled.
Location awareness	Users can access services from the edge server closest to their physical location.
Proximity	The availability of computation resources and services in the proximity of the users.
Low latency	Low latency derives from the proximity of computation resources and services.
Context-awareness	Context information of the mobile device in EC can be used to take offloading decisions and access the edge services.
Heterogeneity	Existence of different infrastructures, platforms, architectures, computing, and communication technologies used by end devices, edge servers and networks.

Table 3 - Edge computing models.

Mobile Edge	Mobile users can utilize the computing services from the base station, off-loading processing, application services and storage to the edge servers.
Fog	Enables the applications to run directly at the network edge through billions of smart connected devices.
Cloudlets	Uses the computer resources available in the local network.

To avoid resource competition and improve resource management, virtualization technology has been applied to EC. Applications are set up as a virtual machine or container with all the necessary dependencies according to the application requirements and configurations [16].

Use cases like industrial automation, augmented reality, location services, and vehicle-to-vehicle communications, require fast processing and quick response time. However, end users usually run these applications on their resource-constrained devices while the core service and processing are performed on cloud servers, which results in high latency. EC complements cloud computing by enhancing the end user service for delay-sensitive applications [15].

3 2D OBJECT DETECTION

3.1 LITERATURE REVIEW

Object detection is the task of detecting instances of objects from a particular class in a digital image, and it is a fundamental task in CV [1].

Early object detection algorithms are known as traditional object detectors, and most of them were built on handcrafted features [1]. Some of the early object detection methods include the Viola-Jones detectors [17], Histogram of Oriented Gradients (HOG) detector [18] and Deformable Part-based model [19].

In recent years, with the development of deep learning and the rebirth of Convolutional Neural Networks (CNN's), which resulted from access to more computational power and large amounts of data, tremendous progress has been achieved, which brought considerably more attention to the technology [1], [20]. Krizhevsky, et al. [21] were the first to outperform traditional methods with CNN's. Since then, Object Detection has entered a period known as deep learning-based detection [1].

Figure 6 illustrates the evolution of object detection through its existence, referencing the most important milestones.

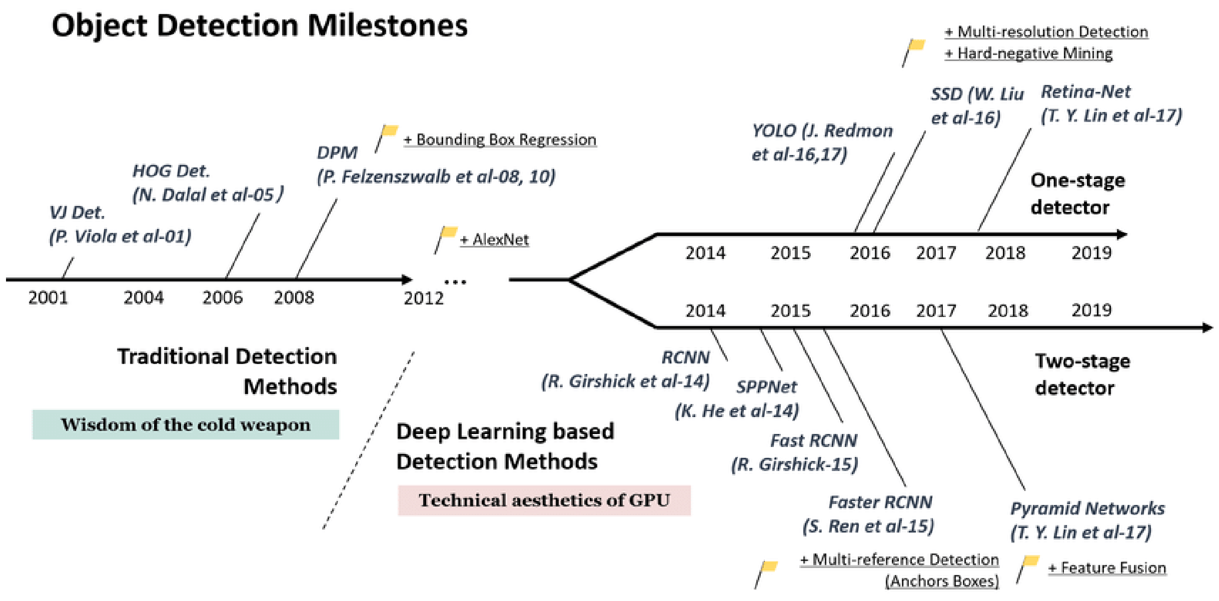


Figure 6 - Road map of object detection. Retrieved from [1].

Nowadays, there are mainly two types of state-of-the-art object detectors. There are two-stage detectors, such as Faster RCNN and Mask R-CNN, that use Region Proposal Networks (RPNs) [4], as shown in Figure 7 [23]. RPNs are fully convolutional networks used to generate regions of interest and then send the region proposals down the pipeline for object classification and bounding-box regression, taking an image as input and outputting a set of rectangular object proposals with an object score [4], [24]. Such models usually reach the highest accuracy rates but are typically slower [4]. There are also single-stage detectors, such as You Only Look Once (YOLO) [25] and Singe Shot Multi-Box Detector (SSD) [26]. This type of object detector takes an input image and calculate the class probabilities and bounding boxes coordinates, treating Object Detection as a regression problem. Such models usually reach lower accuracy rates but are faster than two-stage object detectors [4].

An ordinary modern object detector model, as illustrated by Figure 7, is composed of: an Input layer; a Backbone that is a deep CNN that acts as a feature generator network by taking an image as input and generating a feature map [2]; a Neck that is a set of layers between the Backbone and Head used to collect feature maps from different stages and a Head, that works in a different way depending if the model is a one-stage or two-stage detector [23]. In the case of one-stage detectors, the Head takes in the features from the Neck and performs both object localization and bounding-box classification simultaneously to complete the detection process. On the other hand, two-stage detectors use a sparse prediction with an RPN, which, as said before, generates regions of interest for object localization, meaning that the Head only performs the bounding box classification [23], [27].

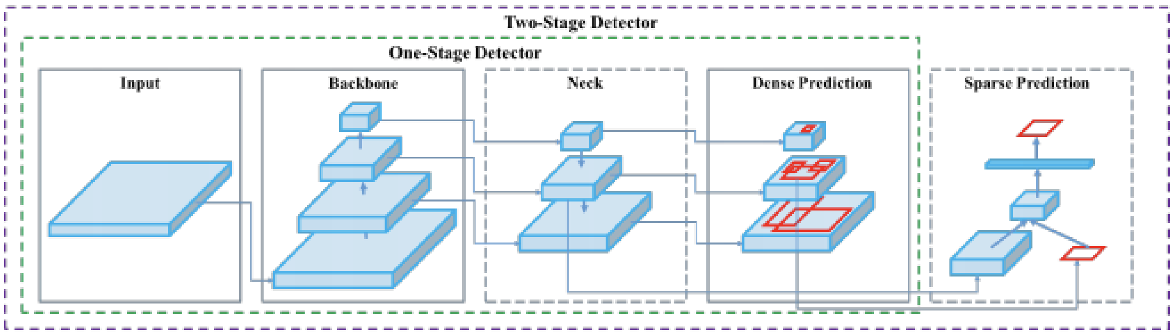


Figure 7 - Object Detector. Retrieved from [23].

Currently, the most frequently used metric for evaluating the effectiveness of an object detector is the Average Precision (AP) [1].

AP is defined as the average detection accuracy across different recalls and is generally assessed in a category-specific manner. An object is successfully detected if the Intersection over Union (IoU) between the predicted box and the ground truth box is greater than a predefined threshold, otherwise will be identified as missed [1], [20].

To compare performance over multiple object categories, the AP averaged over all object categories is usually used as the final metric of performance [1], [20].

So that progress in Object Detection can be achieved is not only necessary to have a universally used metric, but also large datasets with labelled data with the minimum possible bias [28], so that fair comparisons between object detectors can be made. Several well-known datasets and benchmarks have been released in recent years, including Microsoft COCO (MS-COCO) [29].

MS-COCO is a large-scale and most challenging detection dataset available today that has become the standard for the object detection community with a considerable number of images, instances, and categories. Furthermore, the MS-COCO dataset sets itself apart because of the object labelling with per-instance segmentation to aid in precise localization and the fact that it contains a large amount of small and densely located objects. These features make the object distribution in MS-COCO closer to the real world [1], [29].

However, it is not the best solution to train detection models for autonomous driving exclusively in non-autonomous driving datasets. The researchers in autonomous driving suffer from data inadequacy for real-world driving scenarios. The major problems in acquiring and exploiting data for autonomous driving drift from scarcity and lack of diversity in data resources and the fact that scenes must be collected by a driving car on the roads in compliance with local regulations [30].

Image-based datasets like MS-COCO [29] obtain training data directly from websites, and the annotation pipeline is relatively simple. On the other hand, most autonomous driving datasets collect data on the roads with multiple sensors mounted on a vehicle, and the obtained images are further annotated for perception tasks, including Object Detection and Tracking. The KITTI dataset [31] was the first considerable-size autonomous driving dataset released. The nuScenes dataset [32] and the Waymo Open dataset [33] are currently the most widely used autonomous driving datasets, the Waymo Open dataset being the biggest of the two [30].

Although MS-COCO was not used for benchmarking in section 3.3, comparisons of the accuracy and speed of different models in this dataset support the selection of models and their pre-training.

The reason for using the MS-COCO dataset to aid in the selection of models is that, as has said previously, it is a very challenging dataset with a large amount of high-quality data. It also has the same object categories found in Waymo's Perception Open Dataset [29], [33]. Furthermore, as has also been said, MS-COCO is highly benchmarked, meaning much more comparisons between models on this dataset can be found. These benchmarks make it easier to withdraw conclusions even though results might not be totally translatable from one dataset to another because of differences in the cases represented in the data, the input image resolutions and boundary box encoding [20].

Relatively to the pre-training, this strategy is supported by the Waymo Open Dataset Challenge - 2D Detection, described in section 3.2.1. It permitted models to be pre-trained on the MS-COCO dataset [33]. Advantage exploited by some of the teams with the best results in the competition, and Waymo's base model, which pre-trained the model weights to save computational resources and combat labelled data scarcity and then use transfer learning by fine-tuning the network with Waymo's dataset.[33]–[35].

For Object Detection in real-life applications, there is no clear way of comparing different object detectors. Therefore, a balance of speed and accuracy must be reached. The case of object detection for autonomous driving is no different. It requires both high accuracy and real-time inference speed to accurately perceive the environment and take the right control decisions to ensure safety [36].

Most autonomous driving Object Detection approaches rely on LIDAR sensors that give accurate depth information [37]. Although highly precise and reliable, LIDAR sensors are expensive, have a short perception range, and have sparse information [37], [38].

Recently, strategies using only camera systems have been introduced as an alternative [37]. Where the depth of information can be predicted by semantic properties in scenes and object size [38], compared with using LIDAR sensors using only camera systems is a low-cost approach that achieves a comparable depth accuracy. Although, there is still a notable performance gap, especially for faraway objects. Also, camera systems have the potential ability to provide a greater range of perception as the perception range depends on the focal length and the baseline [37], [38].

Even though camera systems object detection and depth estimation have suffered many improvements, stereo images are still inherently 2D, and it is unclear if they can ever match the accuracy and reliability of a LIDAR sensor. Moreover, this trade-off between affordability and safety creates an ethical dilemma [37].

One fact is common to all autonomous vehicle solutions. Even though they may have different combinations of perception sensors, image-based object detection is almost irreplaceable. This is because camera systems are inexpensive, and image data is much more abundant, easier to collect and annotate. Furthermore, recent progress in deep learning shows the tendency that with more and more data, more powerful neural networks can always be designed [36].

Nowadays, there are many different object detectors. As already been said, benchmarks on well-known object detection datasets help to make comparations between different models that then can be used to make an informed decision when choosing a model for a specific use case.

Figure 8, and Table 4, also adapted from [23], show the results of different object detectors on the MS-COCO dataset for different thresholds. YOLOv4 is a significant update to the YOLO range and stands out as having high Frames Per Second (FPS). This means that the object detector processes the input data and generates the detections in a short amount of time. YOLOv4 also has accurate results with a high AP compared to other architectures used in the state-of-the-art, proving the viability of one-stage detectors [23] [39].

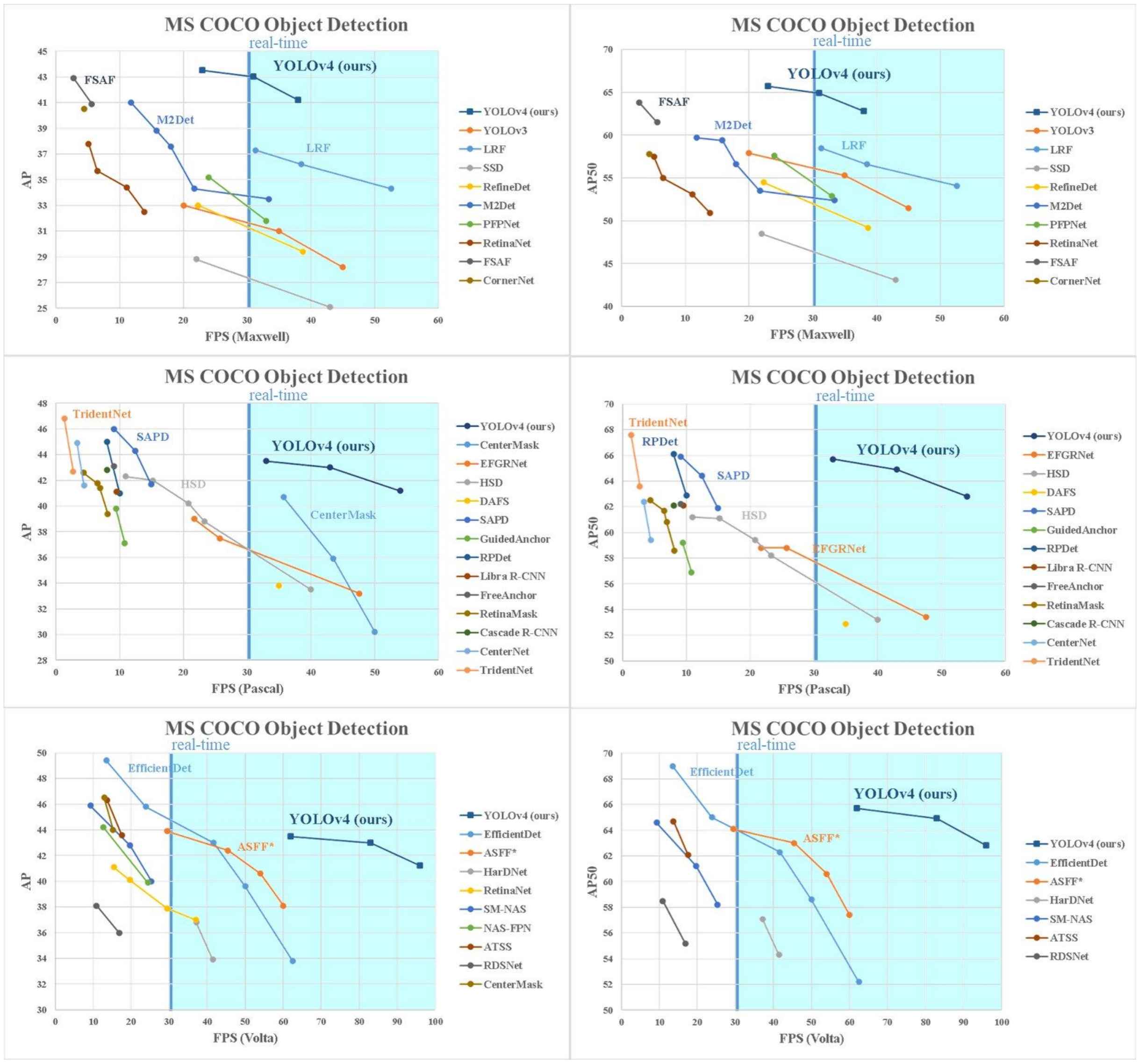


Figure 8 - Object detectors speed and accuracy comparison. Retrieved from [23].

Table 4 - Speed and accuracy of different object detectors on the MS-COCO dataset for different thresholds.

	Size	FPS	AP 30	AP 50	AP 75
YOLOv4	512	31(M)	43.0%	64.9%	46.5%
SSD	300	43(M)	25.1%	43.1%	25.8%
Faster R-CNN	-	9.4(P)	39.8%	59.2%	43.5%
Cascade R-CNN	-	8(P)	42.8%	62.1%	46.3%

After the release of YOLOv4, YOLOv5 one-stage algorithm was proposed by Glenn Jocher [40]. As the name indicates, YOLOv5 is the fifth iteration of YOLO, despite existing more than five implementations of the model, and the five iterations not being all from the same author. Over time YOLO received some improvements that led to faster and more precise predictions. YOLOv5 takes these improvements and builds on them to make a more robust object detector than its predecessors (YOLO, YOLOv2 and YOLOv3). Despite not being an incrementation of YOLOv4. YOLOv5 is architecturally similar to YOLOv4. YOLOv5 claims to have comparable results to YOLOv4 in the MS-COCO dataset with a faster training speed [41], [42]. Although, the authenticity of the performance cannot be guaranteed as there was no official paper from the model authors by the time the model was studied.

The release of YOLOv5 includes five different models' sizes: YOLOv5s, YOLOv5m, YOLOv5l, YOLOv5x.

As can be seen in Figure 9, YOLOv5 has an outstanding performance in object detection, especially the 140FPS reasoning speed of the YOLO V5 model [39].

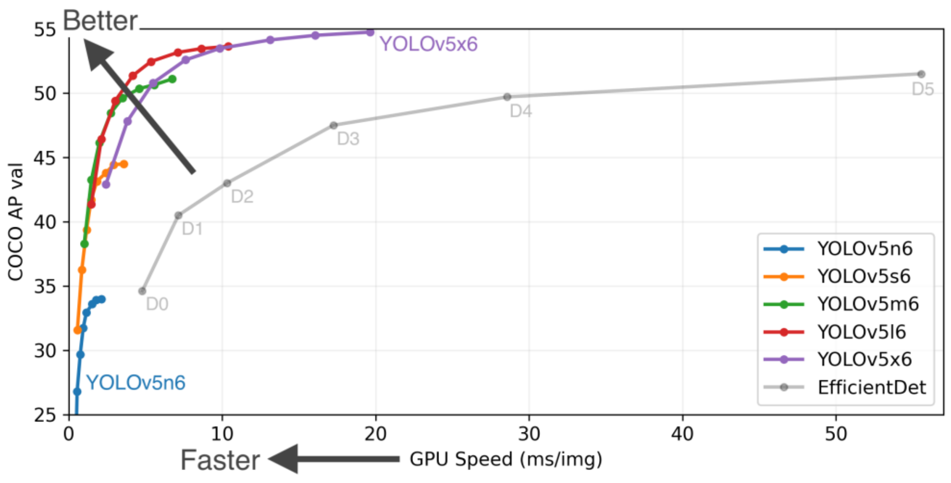


Figure 9 - YOLOv5 and EfficientDet models speed and accuracy comparison. Retrieved from [40].

YOLOv5 was the architecture chosen to train and test on Waymo Perception Open Dataset. YOLO is a great choice because it is not a traditional image classifier that was repurposed as an object detector. YOLO looks at an image only once but in a clever way. It divides an image into a grid where each cell is responsible for predicting up to five bounding boxes, where one bounding box describes one rectangle that encloses an object. YOLO also returns a confidence score relative to a bounding box containing an object, the score is given depending on whether there is a promising shape, and it also returns a separate confidence score of the object class. This methodology keeps the most promising bounding boxes and makes all the predictions simultaneously, which results in a fast and precise model [25]. Also, the fact that YOLOv5 implementation has been done in PyTorch [43] makes it easier to understand, train and deploy.

As further explained in sections 3.3.2 and 3.2.2, throughout the year 2020, took place Challenge 3 of the Waymo Open Dataset Challenges, Waymo Open Dataset Challenge – 2D Detection. Where Waymo Open Perception Dataset was used to benchmark object detectors for autonomous driving, and many researchers contributed with their solution. The dataset in question will be used in section 3.3, and the baseline and best models of the competition were object of study.

For the challenge, a baseline model Faster R-CNN was released, achieving the results of Table 5 [24]. The model was pre-trained on the MS-COCO Dataset [29] before being fine-tuned on Waymo Perception Dataset [33].

Table 5 – Baseline model benchmarks on Waymo's dataset separated by label difficulty. LEVEL 2 labels are cumulative and include LEVEL1 labels, and the more difficult detections.

	Vehicles	Pedestrians
LEVEL 1	63.7	55.8
LEVEL 2	53.3	52.7

An interesting solution found by the second-place team of Waymo Open Dataset Challenge – 2D Detection seems to be a great strategy to explore. The implemented solution produces detections on state-of-the-art two-stage and one-stage detectors and then fuses them to improve results [35]. The two-stage object detector used was a Cascade R-CNN [44] which is skilled in precisely localizing object instances. In contrast, the one-stage detector used was CenterNet [45], which may be better suited for detecting small objects and objects in crowded scenes. The authors of the second-place model [35] also pre-trained a base model detector in MS-COCO. Then to further improve results, expert models based on the base

model were fine-tuned for specific situations. For example, night-time, daytime, pedestrian, and cyclist experts. The results of the expert models were then merged into one group of detections.
The mean average precision (mAP) of the two best-classified teams in Waymo's challenge for the different types of labels can be found in Table 6 [28].

Table 6 – Leader board of the Waymo Open Dataset Challenge – 2D Detection.

	mAP/Level1	mAP/Level2
1°Place	79.4	74.4
2°Place	75.6	70.3

3.2 MATERIALS

Object Detection is one of the essential prerequisites to autonomous navigation, as it allows the car controller to account for obstacles when considering future trajectories. Therefore, the object detection model must be reliable and accurate. However, object detection in autonomous driving presents unique challenges that include the need for detection results in a timeframe under one second with accurate results [46]. Moreover, object detectors have to accomplish this by operating strictly on the input image and hardware memory limitations, which turn infeasible to store and run detectors with many parameters, especially with large input image volumes [46].

3.2.1 WAYMO 2D DETECTION CHALLENGE

The Waymo Open Dataset used in the competition provides high-quality data collected by multiple LIDAR and camera sensors in real self-driving scenarios. As said in section 3.1 this is the largest object detection dataset for autonomous driving, for this reason it was the selected dataset to work with for the experiments of sections 3.3.3 and 3.4.
The Waymo 2D Detection challenge consists of, given a camera image, producing a set of 2D boxes for the objects in the scene [47]. The 2D Detection Challenge restricts the input data to camera images. The classes included in the challenge (vehicle, pedestrian, and cyclist) are annotated with tight-fitting 2D bounding boxes based on the camera images [35]. For metrics purposes, each one of these classes has an IoU overlap threshold. Thresholds are as follows: Vehicle 0.7, Pedestrian 0.5, Cyclist 0.5 [35].
The leader board ranking for this challenge is done using mAP among the challenge classes [35].
The Average Precision (AP) formula is the flowing:

$$AP = \int p(r)\, dr,$$

Equation 1 - Average Precision

where $p(r)$ is the precision-recall curve that plots the precision against recall for different confident thresholds, and precision is the ratio between the true positives, this being correct predictions and the total number of predictions. Recall, on the other hand, is the ratio between true positives and the total number of existing bounding boxes.

3.2.2 OBJECT DETECTION DATASET

Waymo Open Perception Dataset is a large-scale multimodal camera-LIDAR dataset for autonomous driving research. This dataset is significantly larger, higher quality, and geographically diverse, both in terms of complete area coverage and in the distribution of that coverage across geographies, compared to any existing similar dataset [34] as suggested by Table 7, adapted from [33].

The dataset consists of 1150 scenes. Each scene contains about 200 frames for each of the five high-resolution cameras with resolutions of 1280×1920 and 886×1920 and well-synchronized and calibrated LIDAR and camera data captured across a range of urban and suburban geographies. The disposition of the camera system can be found in Figure 10 [33]. Overall, the dataset contains about 1.15M images and 9.9M 2D bounding boxes for vehicles, pedestrians, and cyclists [33], [34].

Table 7 - Comparation between Waymo Open Dataset and other popular autonomous driving datasets.

	KITTI	NuScenes	Waymo
Scenes	22	1000	1150
Hours	1.5	5.5	6.4
2D Boxes	80K	-	9.9M
Lidars	1	1	5
Cameras	4	6	5
Visited Area (km^2)	-	5	76

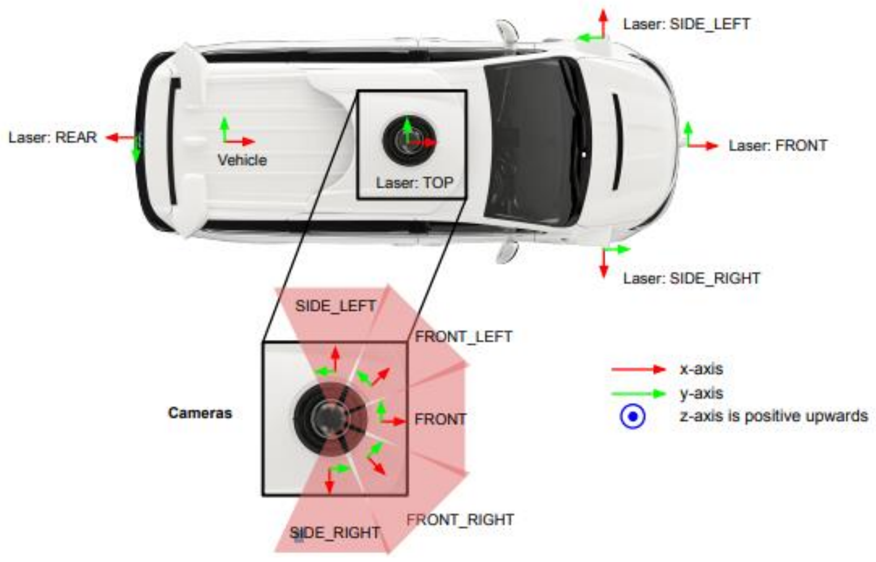


Figure 10 - Sensor layout and coordinate systems. Retrieved from [33].

The dataset data was recorded across various conditions, in multiple cities, with extensive geographic coverage. As a result, the dataset has scenes from suburban and urban areas at different times of the day. Table 8 and Table 9 [33] show the scenes distribution present in the dataset in different cities and by the time of the day.

Table 8 - Scene counts for Phenix (PHX), Mountain View (MTV), and San Francisco (SF).

	PHX	MTV	SF
Train	286	103	409
Validation	93	21	88

Table 9 -Scene counts for different times of the day.

	Day	Night	Dawn
Train	646	79	73
Validation	160	23	19

Vehicles, pedestrians, and cyclists in all camera images were manually annotated with tightly fitting bounding boxes. The bounding boxes' labels are encoded as (cx, cy, l, w) with a unique tracking ID, where cx and cy represent the center pixel of the box, l represents the length of the box along the horizontal (x) axis in the image frame, and w represents the width of the box along the vertical (y) axis in the image frame [33].

The Image frame is a 2D coordinate system defined for each camera image, where +x is along the image width, with the column index starting from the left, and +y is along the image height, with the row index starting from the top. The origin is the top-left corner [33].

3.3 METHODS

The following sections describe the employed approaches to pre-process data, model training and testing of the YOLOv5 object detector model. Due to the large volume of data and high computational needs, a container was created and deployed to a computational capable machine with a Xeon 12 CPU, NVIDIA Quadro P6000 GPU and SATA disk with a total of 1.8 terabytes of available space for storage. The created container has Ubuntu16 as the operating system and Anaconda [48] 4.12.0 installed with a base environment with multiple packages. However, this machine had to be shared with multiple users.

3.3.1 DATA ANALYSES

The first step of the dataset analysis consisted of following the tutorial made available by Waymo [49]. For that to happen, some alterations to the initial script were made driven by incompatibility errors within the library's versions used in the Jupiter notebook [50] made available by Waymo.

The extraction of the Waymo dataset from the current public repositories [51], [52] proceeded, where data is made available in two different ways: individual *tfrecord* files store a sequence of binary records [53] and *tar* [54] files that compress multiple *tfrecord* files. The data was downloaded as individual *tfrecord* files to facilitate data extraction and manipulation. In these files are stored all data captured in a short period of time by a Waymo autonomous driving system.

First, only one of the available *tfrecord* files was downloaded to elaborate all scripts needed for data processing and train a first test object detection model. This file of size 888 MB had a total of 995 images and labels.

For training object detection models with a larger amount of data, it was necessary to resort to Google Cloud APIs to download multiple *tfrecord* files to the container where the model will train. In order to do that, it was necessary to do environment configurations since credentials are needed to access Google cloud storage. Therefore, the steps described in [55] were followed to obtain the credentials.

With access to the Google API a subset of Wayno's training dataset was downloaded, amounting to 360 *tfrecord* file. This data amounted to 217G, which translated to 86219 images. This amount of data was chosen driven by the available space in the container, considering that it is a shared environment.

In all collected data, there are 79690 images with at least on vehicle and a total of 800835 vehicle instances, 52623 images with at least one pedestrian and 420719 pedestrian instances, and 7509 images with at least one cyclist and 11074 cyclist instances. The classes distribution in the collected data can be seen in Figure 11 and Figure 12.

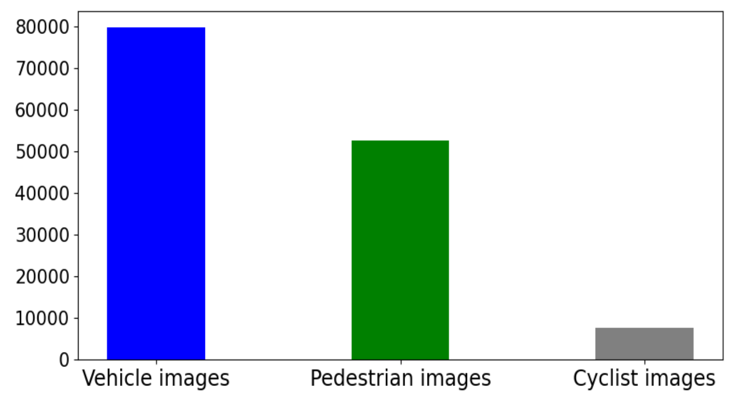


Figure 11 - Images of each class in the collected data.

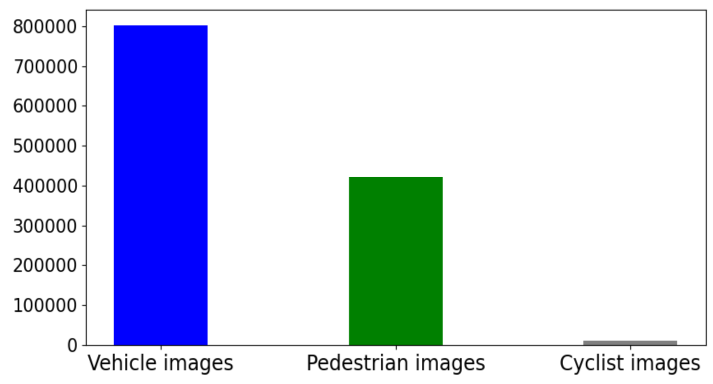


Figure 12 - Instances of each class in the collected data.

The selected object detection model YOLOv5 has some considerations about the ideal dataset conditions to achieve the best results. It recommends more than 1500 images and more than 10000 instances per class, criteria met by the collected data. YOLOv5 also recommends images representative of the deployed environment, with label consistency and accuracy, features that the Waymo dataset ensures [40].

The disparity verified between the existing classes comes from the fact that there is already this disparity in the original Waymo dataset. Moreover, it is an understandable disparity since the data was recorded in an actual driving environment where there is more likely to find cars, people, and cyclists by that order.

In Waymo's public repository, the files *dataset.proto* and *label.proto* are available, where the structure of the data stored in the *.tfrecord* files can be found [52]. The *tfrecord* files are composed of a list of *Frame* structures where each one has the information collected on an instance of time. To access the images collected by the vehicle cameras the *Images* property of the *Frame* structure was accessed where it was possible to not only extract the image but also know which camera it was taken from. Each element of the *Images* property of a frame was a *Camera_Labels* property where the labels referent to that image have a Box structure where the coordinates and dimensions of the bounding box are stored. Recapitulating, and as synthesized in Figure 13 - TFRecord data structureFigure 13, for each *tfrecord* file exists a list of Frame structures where each of these structures has images and labels referent to an instance of time.

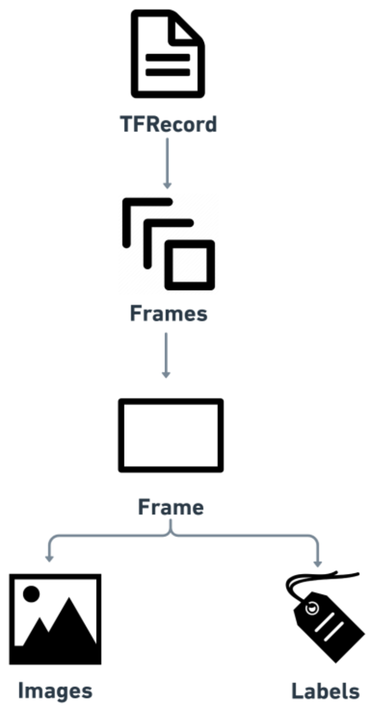


Figure 13 - TFRecord data structure.

The data analysis's final step consisted of trying to draw the bounding boxes in the images. The bounding box coordinates follow the logic of the inverted grid, as mentioned in section 3.2.2. With that information, the bounding boxes were drawn with the help of the OpenCV library [56] by calculating the top left and bottom right nodes of the bounding box that were found with the following formula:

$$Top\ left\ node = (center_x - \text{length} * 0.5, center_y - \text{width} * 0.5)$$

Equation 2 - Top left node equation.

$$Bottom\ right\ node = (center_x + length * 0.5, center_y + width * 0.5)$$

Equation 3 - Bottom right node equation.

Where the center is the coordinates of the center point of the bounding box, and the length and width are his dimensions.

3.3.2 DATA PROCESSING

This section describes approaches to pre-processing the images and labels of the Waymo dataset.
In order to develop algorithms to pre-process data and train a test object detection model, a small part of the downloaded data was used. This data is approximately 17G and allowed for faster development and correction of the pre-processing algorithms and the object detection test model.
YOLOv5 model follows a similar coordinate system as Waymo Open Dataset labels, as illustrated in Figure 14. In this coordinate system, the x-axis extends across the image width, with the column index starting from the left, and y-axis extends along the image height, with row index starting from the top, that in the case of Waymo dataset is referred as length, and the origin is the top-left corner. Therefore, it was not necessary to do any transformations to the dataset labels other than using the length coordinates as height coordinates.

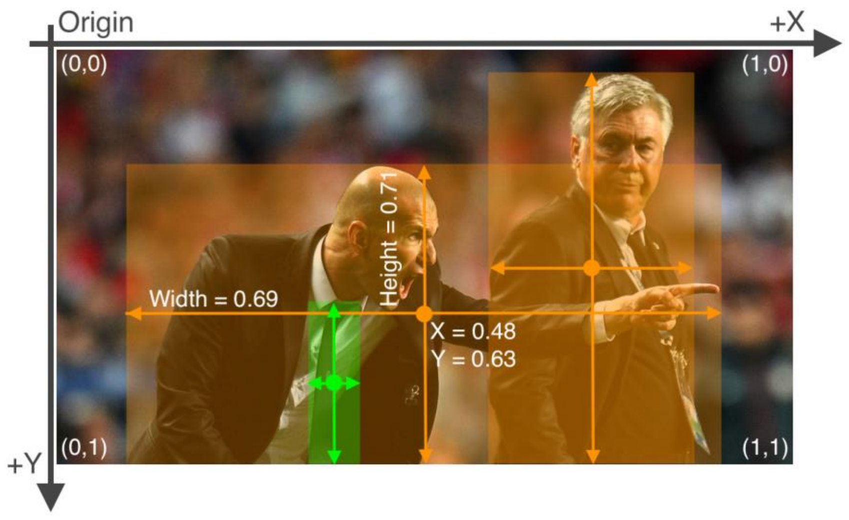
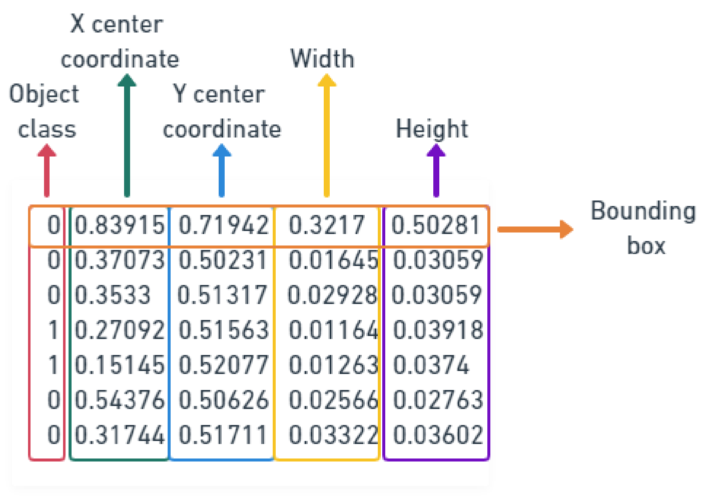


Figure 14 - YOLOv5 coordinate system. Retrieved from [57].

YOLOv5 expects the input image to have its annotations included on a text file. This text file with the format described in Figure 15 expects bounding box coordinates and dimensions to be normalized between 0 and 1 and the class numbers to be zero-indexed.



0	0.83915	0.71942	0.3217	0.50281
0	0.37073	0.50231	0.01645	0.03059
0	0.3533	0.51317	0.02928	0.03059
1	0.27092	0.51563	0.01164	0.03918
1	0.15145	0.52077	0.01263	0.0374
0	0.54376	0.50626	0.02566	0.02763
0	0.31744	0.51711	0.03322	0.03602

Figure 15 – YOLOv5 object detection label file format.

In order to convert the data to the specified format, the first step was to extract the data stored in the *tfRecord* to JPG and text files that store the image and bounding box coordinates. Some label parameters were filtered, and only the parameters relative to the width, length, and bounding box center coordinates were kept. The coordinates and dimensions were rescaled between 0 and 1 by the image shape, and class names were renamed to integers.

The next step of the data pre-processing is relative to the different shapes of images in the dataset. There are images with the following resolutions: 1280x1920 pixels or 886x1920 pixels. The different sizes translate to different resizes when training the YOLOv5 model, and the smaller size of some of the images obliges training the model with a smaller image size, which can negatively impact the model accuracy. In order to compare models trained on different image sizes, two baseline object detection models were trained. One with the normal image data from the Waymo dataset with a smaller image size, and the other model trained with data that suffered padding to the bottom side of the images with the resolution of 886×1920 so that the 1280×1920 becomes the only existing resolution.

The results of these two baseline models were compared and used to choose what training data to use to fine-tune the final object detection model.

The logic behind the pre-processing script that transformed the data to the correct format can be summarized in the steps of **Algorithm 1**.

Algorithm 1 Pseudocode for pre-processing Waymo's Perception dataset data for YOLOv5 training.

```
    Input: padding ← Boolean that indicates if the images need padding
    Output: folder with all extracted images and labels
1   filesNames ← pre-defined directory gets searched for all tfRecord files names
2   path ← created folder for storing the images and labels that will be extracted
3   foreach(fileName : filesNames) //Read all tfRecord files
4       dataset ← Extract dataset from tfRecord file
5       foreach(data : dataset) //Iterate through the dataset data
6           frame ← Parse the attribute Frame from data
7           foreach(image : frame.images) //Iterate through the frame images
8               img ← Read image data from memory cache and convert to image format
9               rgb_img ← Convert an image for RGB color space
10              If(padding)
11                 Color ← 255
12                 width, height ← rgb_img.size
13                 padding_size ←1280 - height
14                 new_height ← height + padding_size
15                 rgb_img ← Image.new(rgb_img.mode, (width, new_height), color)
16              Save rgb_img  on path + “/Images” directory
17          end
18          foreach(image : frame.images) //Iterate through the frame images
19              img ← Convert to image format
20              width, height ← img.size
21              foreach(camera_labels:frame.camera_labels)//Iterate through camera_labels
22                  If(camera_labels.name == image.name)
23                     labels_file ← create a file for the image frame labels in the path +
24                     foreach(label: camera_labels.labels)//Iterate through the labels
25                         center_x_norm, center_y_norm, width_norm, height_norm ←
                           Calculate the normalized coordinates of the bounding box
27                         label_type ← Rename the label name from 4 to 2 or subtract one
                           to the name of the label in the other cases
28                         labels_file ← write the bounding box information to a line of
                           txt file
29                     end
30                     labels_file ← Close txt file
31              end
32          end
33      end
34  end
```

During the development of the pre-processing script, some issues regarding Waymo's dataset, the CV2 [56] and Pillow [58] Python libraries' different definitions of an image height, length, and width, were observed. These differences originated normalization errors confirmed by drawing bounding boxes stored in the text files in the respective images. This problem has been overcome by figuring out the different meanings for the same designations between the dataset and the used libraries.

A script to remove all images with no labels and a script to divide the available data into three different datasets for training, testing and validation, with the distribution described in Table 10, were also developed.

Table 10 - Extracted data datasets distribution.

	Total	Train	Test	Validation
Images	86215	68972	8622	8621

3.3.3 MODEL TRAINING

A Conda [48] environment was created to train YOLOv5 models. This environment had Python installed with a version equal or superior to 3.7.0, the package PyTorch [43] installed with a version equal or superior to 1.7, and the rest of the dependencies of the YOLOv5 were also installed by running a script file made available in the YOLO's repository.

YOLOv5 has four different models that differ in size, inference time and precision. The model selected for training was YOLOv5x, the model with the most parameters and the most accurate.

The YOLO model training can be configured with the number of epochs, batch size, image size, and a group of 30 hyperparameters that can also be tuned. YOLO authors have some considerations relative to the training settings. They recommend starting training with 300 epochs and, depending on if it overfits or not, increasing or decreasing the number of epochs. The largest batch size that the hardware allows should be used as small batch sizes produce poor batch normalization statistics and a large batch size produces faster training and better resource utilization. Regarding image size, the dataset MS-COCO where the pre-trained weights come from trains at a native resolution of 640 pixels, so it seems like a good resolution for a base model. Although a higher image size can help detect small objects, on the other hand, they may not benefit the overall accuracy of the model. The best inference results seem to be obtained at the same image size as the training was run at, meaning testing and detecting should be in the same resolution as the training should be used [40]. With this in mind, knowing that Waymo's dataset does not have all images with the same size, and in order to use the maximum size of the images possible, the smaller images suffered transformations, so a universal could be reached. This transformation existed in the form of padding, referred to in 3.3.2, and allowed for a universal image size of 1280 x 1920. Padding was the best approach since merely resizing the images would distort the objects.

Finally, regarding the hyperparameters, YOLOv5 has about 30 hyperparameters used for various training settings. There are three different configurations of the hyperparameters that the YOLOv5 authors recommend using. The default group of hyperparameters was used in the base model, and different configurations were explored while fine-tuning the base models [40].

The first step for the YOLO model training, and to confirm if the environment was correctly configured and if the data was correctly labelled a first model was trained for testing purposes, allowing for corrections to the pre-processing scripts, finding incompatibilities between YOLOv5 and the used hardware and it also served as a test of the capabilities of the available hardware.

The YOLOv5 model is prepared for running in a single GPU or multiple GPUs and allows for multiple data loader workers being used in training. As the machine available for this dissertation only had one GPU, the other option to improve training time was to use multiple data loader workers. Nonetheless, the shared memory limit was repeatedly an issue. Since the limit was often trespassed and the training interrupted. This problem is specific to the hardware where the training occurred, and the only definitive solution was to run with 0 data loader workers, which resulted in long training sessions.

Two baseline YOLOv5X models, with pre-trained weights trained in 300 epochs on the MS-COCO dataset [29], which characteristics can be seen in Table 11, were trained in Waymo's dataset in order to determine the optimal image size for the used dataset. Table 12 shows the YOLOv5x model summary, and Table 13Table 13 shows the baseline models training settings. The required training time of these models proved to be a problem that limited the training to 30 epochs, although the minimum recommended number of training epochs by the YOLOv5 team is 300. However, the fact that pre-trained weights were used helped with a faster conversion to an optimal solution.

As described in Table 13, the larger image size implicated a lower batch size value since a run time error relative to CUDA [59] running out of memory caused by GPU limitations kept showing for higher batch size values.

The first conclusion withdrawn from training was that using an image size of 640 pixels led to 40453 object labels occupying less than 3 pixels in size, a problem that does not occur with an image size of 1280 pixels.

Table 11 - Pre-trained weights for object detection.

	Image size (pixels)	mAP Val 0.5:0.95	mAP Val 0.5	Paaramters (M)	FLOPs @640 (B)
YOLOv5x	640	50.7	68.9	86.7	205.7
YOLOv5x6	1280	55.0	72.7	140.7	209.8

Table 12 - YOLOv5 model summary

	Layers	Parameters	Gradients	FLOPs
YOLOv5	567	86231272	86231272	204.7 G

Table 13 - Baseline models training settings

	Epochs	Image Size	Batch Size	Hyperparameters
Baseline	30	640x640	32	Default
Baseline with padding	30	1280x1280	8	Default

With the analyse of the o baseline models, exposed in section 3.4, it was possible to conclude the best image size and batch size combination. The only thing missing now was to find the optimal set of the 30 available YOLOv5 hyperparameters that minimize the loss function [40], [60].

The Genetic algorithm (GA) is the recommended technique by the YOLOv5 team for hyperparameter optimization [40]. Traditional hyperparameter optimisation methods, like grid searches, can become unviable due to the high dimensional search space, unknown correlations among the dimensions, and the expensive nature of evaluating the fitness at each point [40]. The GA is inspired by the process of natural selection relies on biologically inspired operators such as mutation, crossover, and selection [61]. The GA also presents a problem, as it is generally computationally expensive and time-consuming, as the base scenario is trained hundreds of times. In the case of YOLOv5, a minimum of 300 generations of evolution are recommended for best results [40]. The available computational resources could not match these compactional requirements, so GA was not an option for the experiments of this dissertation.

YOLOv5 ensures that images are never presented in the same manner. This diversity is achieved by applying augmentations in the training loader, a component designed to load train dataset images and labels, to present a new and unique augmented Mosaic, the combination of the original image plus three random images, each time an image is loaded for training. Increasing augmentation hyperparameters will generally reduce and delay overfitting, allowing for more extended training and higher final mean AP [40]. YOLOv5 developers recommend training with default hyperparameters first, which is why the baseline models were trained on the default parameters [40]. As previously mentioned, there are three sets of hyperparameters that the YOLOv5 team recommends using. These sets are the settings explored during the model's fine-tuning. There is a set for low, medium, and high augmentation. According to the developers of YOLOv5, the set of hyperparameters with low augmentation is recommended for nano or small models, medium augmentation for medium models and high augmentation for larger models [40]. Since the YOLOv5 model used was the YOLOv5X, hyperparameters with high augmentation may delay overfitting resulting in improvements relative to the baseline models.

Table 14 shows what hyperparameters change between the default and the high augmentation hyperparameter sets. The definition of these hyperparameters is listed below [40].

- Cls: Classification loss gains.
- Obj: Objectness loss gains.
- Scale: It is used to resize an image to match with grid size or optimise results.
- Mixup: Dictates the probability of the image suffering the mixup augmentation. Figure 16 shows an example of mixup data augmentation.
- Copy-Paste: Dictates the probability of image suffering segment copy-paste augmentation. Figure 17 shows an example of copy-paste data augmentation.

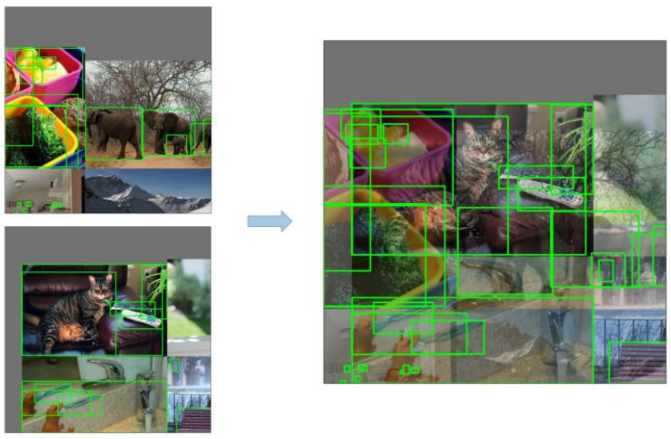


Figure 16 - Image mixup. Retrieved from [40].

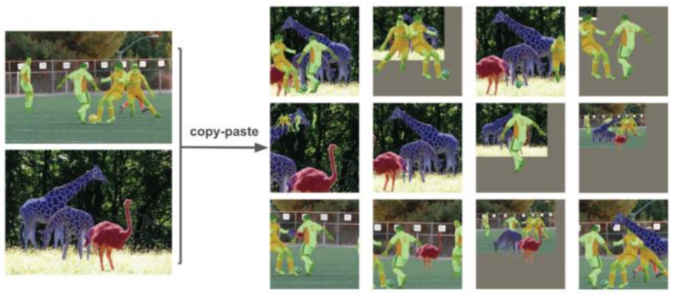


Figure 17 - Image copy-paste. Retrieved from [40].

Table 14 - Differences between default and high augmentation hyperparameters.

Parameter	Default	High augmentation
Cls	0.5	0.3
Obj	1.0	0.7
Scale	0.5	0.9
Mixup	0.0	0.1
Copy-Paste	0.0	0.1

A model with the weights of the baseline model with padding, the baseline model with the best results as explained in section 3.4, and with the high augmentation hyperparameters was trained with the same image size, batch size and number of epochs of the baseline model with padding of Table 13, and also with the same model summary of Table 12. Results and comparisons to the baseline models can be found in section 3.4.

3.4 RESULTS & DISCUSSION

The Weights and Biases [62] framework was used for model training, analysis of results and version management of the different experiments. The following results were extracted with the help of this framework.

The first experiment that took place was the comparison of the baseline models, the No Padding model with bigger batch size but smaller image size, and the Padding model with smaller batch size but bigger image size, introduced in section 3.3.3 and whose characteristics are summarized in Table 13. The models were differentiated as having been trained with data whose images suffered padding transformation or not. In image Figure 18, it is possible to see the results of this experiment. The graphic suggests that the Padding model started with a slightly lower mAP of 47.4 compared with the other model that started with a mAP of 50.6. These differences can be explained by the fact that the different models used different pre-trained weights, as explained in section 3.3.3 whose characteristics are summarized in Table 11. However, this disadvantage was quickly recovered along the training process since the Padding model finished the 30 epochs with a mAP 86.2 and the No Padding model finished with a mAP of 77.6.

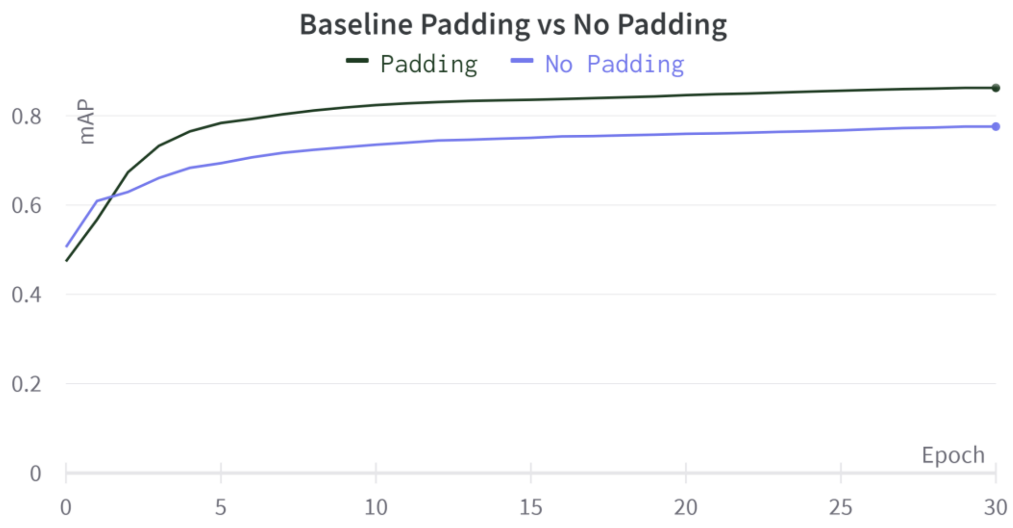


Figure 18 - Baseline models mean AP results.

In ideal conditions, the training would continue for more epochs until a point the model had clearly reached a plateau, or it started overfitting. However, as the graphic of Figure 18 suggests, both models were still slightly improving between epochs. Furthermore, looking at the training and validation losses shown in Figure 19, Figure 20 and Figure 21 is possible to see that as is the case of the mAP they are also still slightly improving between epochs.

The graphics of Figure 19, Figure 20 and Figure 21 represent the bounding box regression loss (box_loss), classification loss (cls_loss) and objectness loss (obj_loss), that is, the confidence of object presence, which are the three components of the loss in YOLOv5. Looking at the loss graphics slopes is possible affirm that the models are close to a scenario where they run out of learning capacity reaching a plateau, or at least are close to a local minimum. The overfitting scenario was not taken as a possibility as the validation and training losses seem to be stagnating at a similar passe, and for this type of scenario the training loss would continue to improve while the validation loss would stagnate or even increase.

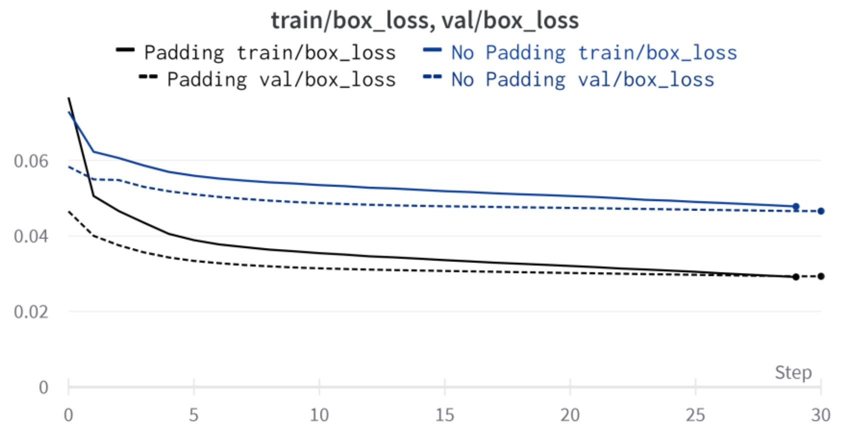


Figure 19 - Baseline models training and validation bounding box regression losses.

.

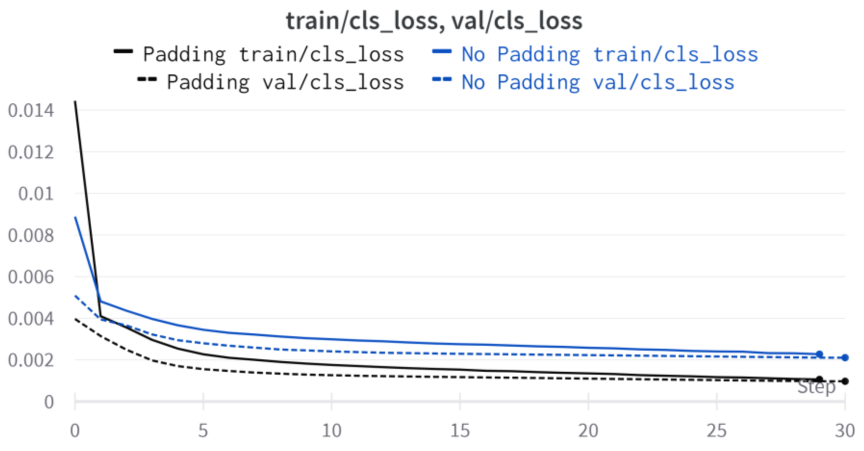


Figure 20 - Baseline models training and validation classification losses.

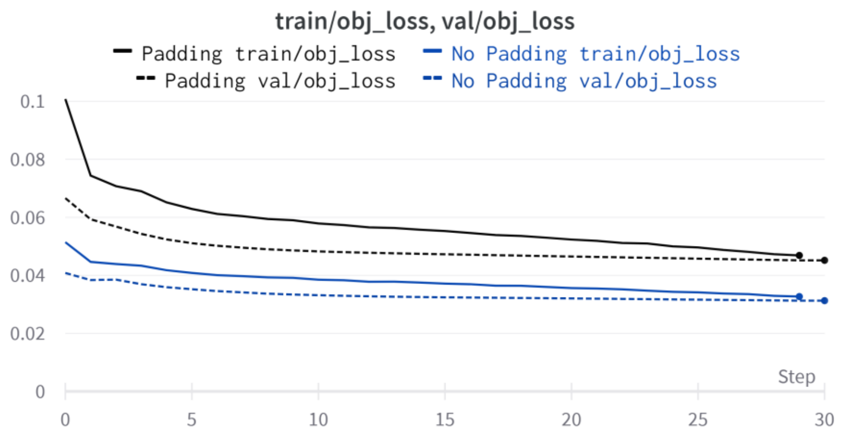


Figure 21 - Baseline models training and validation objectness losses.

There are some interesting observations to withdraw from the loss and mAP graphics. First, the rapid conversion of the models in Figure 18 can be explained, as already mentioned in section 3.3.3, by the pre-trained weights on the MS-COCO dataset. Secondly, Figure 19, Figure 20 and Figure 21 show an interesting scenario where the loss is higher when the model is being trained than when the model is being evaluated. This is not the most recurrent scenario since the model performs worse in the data it is trained on than with the validation data. This phenomenon is explained by the YOLOv5 team as the result of augmentation during training and is absence during validation. The augmentation technique presents a unique mosaic composed of the original image and three random images.

As already mentioned, the Padding baseline model was the best of the two trained ones. The weights of this model were used to train a second model with high augmentation hyperparameters, as explained in section 3.3.3. This experiment aimed to determine if the high augmentation hyperparameters could delay the overfitting and consequent stagnation of the mAP. The results of the experiment can be seen in Figure 22. As the graphic indicates, there was a considerable loss in the mAP results in the first epochs of the experiment. This drop can be explained by the fact that the baseline model weights were trained with a different set of hyperparameters. Therefore, the differences in the hyperparameters, namely the augmentation increase, most likely impacted the results. After the initial adaptation of the model, the mAP results improved epoch by epoch, eventually surpassing the mAP achieved with the baseline models with a final result of 86.6. However, it took many epochs to reach the mAP of the Padding baseline model, a model which was still slowly improving at the end of the 30'th epoch and whose heights served as the base for the high augmentation experience.

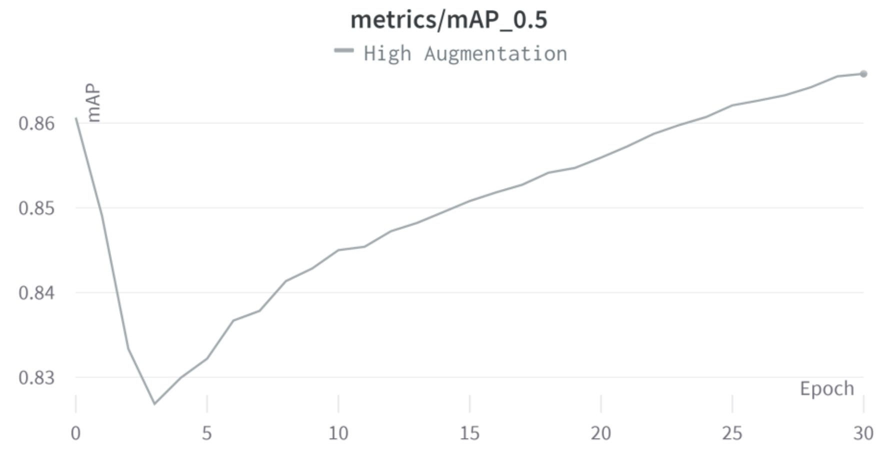


Figure 22 - Model with high augmentation hyperparameters.

Table 15 shows the final mAP for a 0.5 threshold of the High Augmentation model for the different classes in the dataset. The class with the best results is the vehicle, followed by the pedestrian and the worst performing class was the cyclist. Looking at Figure 11 and Figure 12, these results reflect the distribution of the classes in the dataset.

Table 15 - mAP results for the different classes.

Class	Intances	mAP
Vehicle	81515	91.6
Pedestrian	42092	87.6
Cyclist	1190	80.6

Figure 23 shows the confusion matrix of the High Augmentation model. In the matrix is possible to see that 89% of the Vehicle class entities were correctly labelled, and 11% were background false negatives (FN), meaning that they were not detected. For the Pedestrian class, 85% of the pedestrians were correctly labelled, and 15% were background FN. Finally, the Cyclist class was correctly predicted 73% of the time, 2% of the time was mistaken by a vehicle, 5% by a pedestrian, and 20% of the time was a background FN. As for background false positives (FP), which are objects belonging to the background of an image mistaken by an object class, 56% of the false positives were labelled as vehicles, 43% as pedestrians and 1% as cyclists.

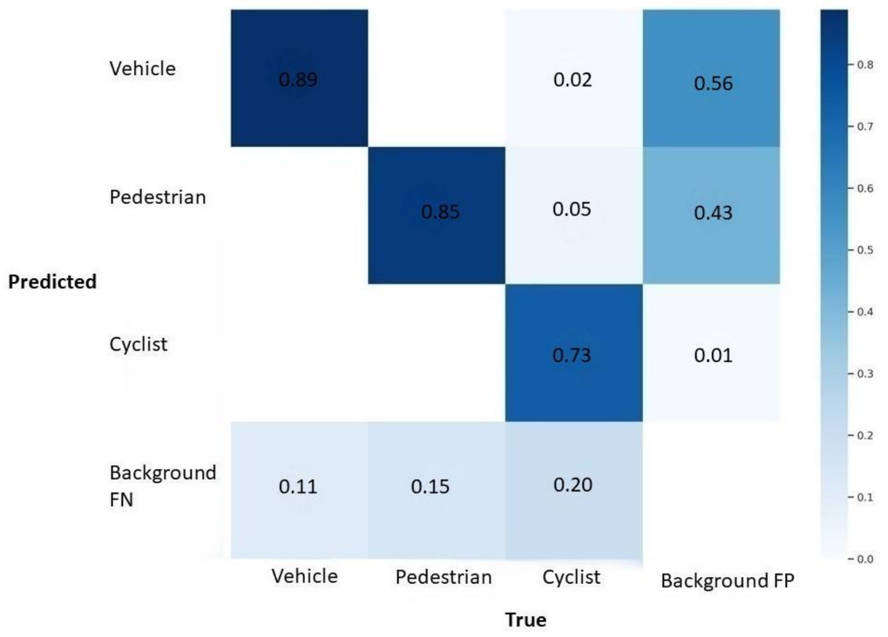


Figure 23 - High Augmentation model confusion matrix.

Figure 24, Figure 25 and Figure 26 show graphics with the training and validation losses of the High Augmentation model and the Padding baseline model. These graphics show that all three losses of the High Augmentation model increased in the first epochs, likely due to the adaptation to the new hyperparameters. However, after this period, the validation and training losses decreased again. Comparing the losses of the Padding baseline model to the High Augmentation model is clear to see that the classification and objectness losses decreased relative to the baseline model. This can be explained by the lower values of the hyperparameters referent to the classification and objectness losses in the high augmentation model, as explained in Table 14. On the other hand, the bounding box regression loss, whose hyperparameters did not suffer any alterations, finished with a validation loss similar to the baseline model and a training loss even slightly worst due to the higher training augmentation. Looking at the slopes of the training and validation losses, the High augmentation model finished the 30'th epochs in a state further away from a possible plateau, local minimum, or overfitting scenario than the baseline models, even though this model started with the weights of the Padding baseline model that was already close to a stage of plateau or local minimum.

These results suggest that the alteration to hyperparameters caused the model to improve, although slightly, and did indeed allow for a more extended training session delaying the stagnation of the model training. With hindsight, minor improvements in the High Augmentation model were expected as, looking at the losses of the Padding baseline model, it had not yet reached the total learning capacity. For that reason, increasing augmentation, which translates into more training data, was not yet necessary at the stage of training the Padding baseline model was at the end on the 30'th epoch.

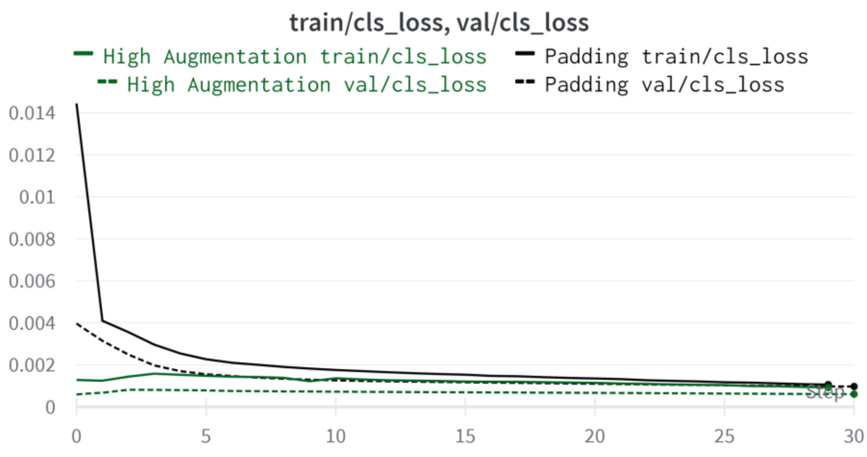


Figure 24 - High augmentation and baseline model training and validation classification losses.

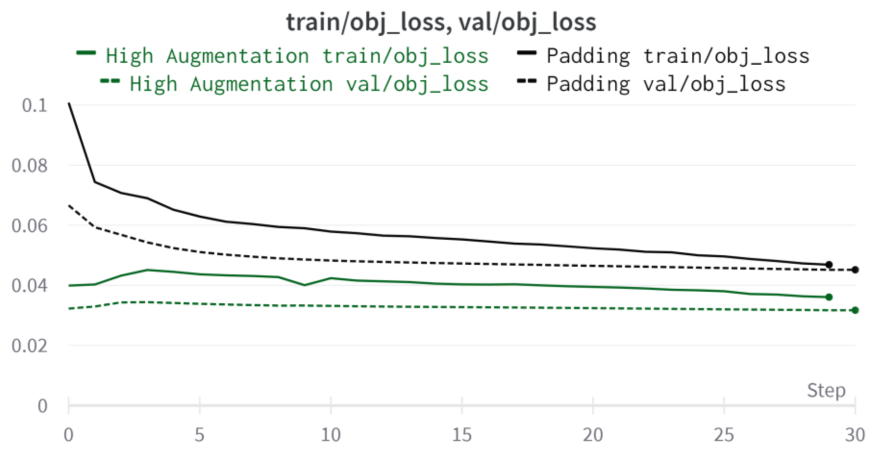


Figure 25 - High augmentation and baseline model training and validation objectness losses.

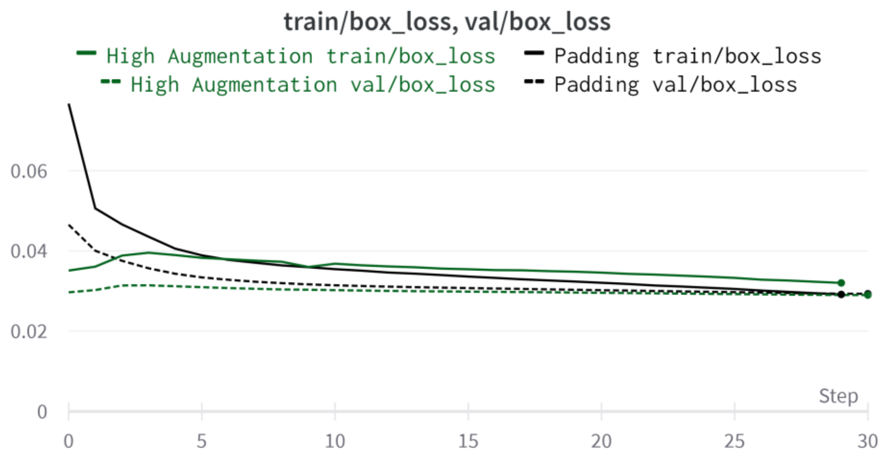


Figure 26 - High augmentation and baseline model training and validation bounding box regression losses.

Even though it would be interesting to compare the obtained results to the Waymo's detection challenge leader board with would not be a fair comparison as the experiments of this dissertation were extracted from a small subset of the Waymo's dataset. Also, the leader board of the competition divides the mAP results into easier labels and harder-to-detect labels, which they name Level 1 and Level 2 labels, and the evaluation process uses different thresholds for the classes of the dataset, opposite to the experiments done that used a 0.5 threshold for every class [33]. For these reasons and the limited time and computational resources available for this dissertation, there is not much information that can be extracted from comparisons.

Compared to the results of the first and second place of the competition, whose results can be seen in Table 6, with mAPs between 70% and 80% for the different types of labels, it is possible to infer that the results obtained in these experiments were good. However, it is essential to emphasize that these results would not be maintained with the data on which the challenge models were tested. However, it is still a solid solution that validates the potential of the YOLOv5 model. This validation is further emphasized by the YOLOv5 team, which also tested in Waymo's dataset, obtaining a mAP of 70.3 for Level1 labels and 64.1 for Level 2 labels.

4 2D Object Tracking

4.1 LITERATURE REVIEW

Object Tracking is the task of given an initial set of object detections, assigning them a unique ID, and estimating their trajectory [2]. Unlike Object Detection algorithms, whose output is a collection of bounding boxes, Object Tracking algorithms also associate a target ID to each box in order to distinguish intra-class objects [63]. The reason for using Object Tracking instead of Object Detection is the need for a correlation of tracked features between sequential video images. Otherwise, any occlusion will lead to losing track of the targeted object. Occlusion occurs when multiple objects come so close that they either partially or entirely appear to be merged [64].

Just like Object Detection, Object Tracking also benefited from the rapid increase in processing power and availability of large amounts of quality data over the last few decades, which led to advancements in the field. As a result of these advancements, Object Tracking has been widely applied in various fields, such as healthcare, robot vision, security, anomaly detection and autonomous driving [65].

Object Tracking can be done with either of the following approaches: Single Object Tracking (SOT), and Multiple Object Tracking (MOT).

SOT involves only a single object being tracked in a video or a set of frames. It creates bounding boxes that are given to the tracker based on the first frame of the input image [63]. Similarly, MOT is when various objects from one or more categories are being tracked simultaneously within the same video or set of frames [33], [63].

MOT is of great importance in autonomous driving, where it is used to detect and predict the behaviour of pedestrians or other vehicles. Hence, the algorithms are often benchmarked on autonomous driving datasets like KITTI [31] tracking test.

To the detriment of SOT for autonomous driving, the preference for MOT comes down to the main difficulty in tracking multiple targets simultaneously. Interactions between objects result in multiple occlusions that reveal that SOT models usually struggle to distinguish between similar-looking intra-class objects. Because of that, simply applying SOT models directly to multiple targets leads to poor results, often incurring target drift and numerous ID switch errors [63].

There are also two ways of classifying a tracking algorithm depending on if they have access to future frames when identifying an object in a particular frame. Batch-tracking algorithms use information from future frames when deducing the identity of an object. Online tracking algorithms only use present and prior frames to produce bounding boxes regarding the identity of an object in a particular frame. Real-time problems requiring the tracking of objects do not have access to future video frames, which is why online tracking methods are the only viable option for these situations [33], [66].

In this work, only MOT Online object trackers are considered, as they are the only viable solutions for autonomous driving.

Object Tracking methods generally consist of two main components. A motion model that describes and predicts the states of an object over time and an observation model that verifies predictions in each frame depicting the appearance information of the tracked object [67].

Traditionally Object Tracking followed interest points through space and time by developing strong hand-crafted features [67], [68]. Some milestones of this period include [69]–[76]. However, these techniques were prone to noise and easily lost track during occlusion, which did not make them usable for real-life applications [67]. For this reason, traditional approaches were abandoned, and with the rapid development of DL networks, computing power and available data, the performance of object trackers greatly increased [2])

DL methods are progressively used in most top-performing MOT algorithms. The strength of deep neural networks resides in their ability to learn rich representations and extract complex and abstract features from their input. CNNs currently constitute state-of-the-art in spatial pattern extraction [63].

Nowadays, the best-performing MOT methods usually follow the tracking-by-detection paradigm. Which first detects objects in each frame and then associates them over time [77], [78].

This model type can even be separated into two categories based on whether they use one or two models to detect and track objects. The main advantage of having separate models for detection and tracking is the possibility of developing the most suitable model for each task separately without compromising. As a result, these approaches have achieved the best performance on public datasets. However, they are usually slow as the runtime is the sum of the two models [77], [78].

With the quick maturity of multi-task learning in deep learning, single-model detection and tracking have attracted more research attention. These models usually achieve faster runtime. However, the accuracy is usually lower than the two-step models [77].

Object Tracking has many types of errors. A metric should then be able to judge the tracker's precision in determining exact object location and reflect its ability to track object configurations through time consistently [79].

A group of metrics has been established as a standard to provide a common experimental setup where algorithms can be reasonably tested and compared. Multiple Object Tracking Precision (MOTP) and Multiple Object Tracking Accuracy (MOTA) are novel metrics that intuitively express a tracker's characteristics and can be used in general performance evaluations [79].

The MOTA metric accounts for all object configuration errors. Namely false positives, mises and mismatches over all frames. MOTA is given by the sum of the number of misses (m_t), false positives (f_{p_t}), and mismatches $(mmme_t)$ averaged by the number of ground truth boxes [63], [79].

$$MOTA = 1 - \frac{\sum_t (m_t + f_{p_t} + mmme_t)}{\sum_t g_t}$$

Equation 4 - Multiple Object Tracking Accuracy.

MOTP, on the other hand, is the metric that shows the ability of the tracker to estimate precise object positions. It is given by the total position error for matched hypothesis pairs $(d_{i,t})$ over all frames averaged by the total number of matches made between ground truth and the detection output (c_t) [63], [79].

$$MOTP = \frac{\sum_{i,t} d_{i,t}}{\sum_t c_t}$$

Equation 5 - Multiple Object Tracking Precision.

These two metrics complement each other, so they are usually used together when classifying an object tracker.

As well as in Object Detection, standardized benchmarks help progress in Object Tracking models' performance. In order to choose what models to explore in this dissertation, an analysis of public datasets benchmarks was made. More specifically, MOTChallenge and Waymo Open Dataset – 2D Tracking challenge results were studied. MOTChallenge is the most used benchmark for MOT. It provides a large pedestrian dataset and uses MOTA as a primary score metric, but many other metrics are used [80]. In addition, Waymo Open Perception Dataset was also used for comparing models since the Waymo Open Dataset – 2D Tracking challenge has a public leader board where models are classified with MOTA and MOTP metrics [33].

Waymo released a base model for the Waymo Open Dataset – 2D Tracking competition with the method Tracktor [81] based on a Faster R-CNN object detector pre-trained on the MS-COCO dataset [29] and then fine-tuned on Waymo Perception Dataset. When tracking vehicles, the resulting model achieved a MOTA of 34.8 at LEVEL 1 labels and 28.3 at LEVEL 2 labels. The first place of the challenge developed an efficient and pragmatic tracking-by-detection framework known as HorizonMOT. This framework is a one-stage joint detection and tracking model that uses a single network developed on top of CenterNet

[45] that focuses on frame-to-frame prediction and association [34]. Typically, an association between tracks and detections are done through the Hungarian algorithm that handles the association as an assignment problem. Contrary to the norm, HorizonMOT has a three-stage data association scheme to improve tracking performance. 2D tracking on HorizonMOT also relies on re-Identification (re-ID) features extracted by a small independent network to complement bounding box overlap-based association metrics that help to handle long-term occlusion or objects with large displacement [34]. HorizonMOT used pre-trained weights on MS-COCO, and daytime, night-time, pedestrian, and cyclist expert models were fine-tuned to handle the imbalanced training data problem. The outputs of all models were then merged by the naive non-maxima suppression (NMS) algorithm [34]. HorizonMOT obtained a MOTA/L1 and MOTA/L2 of 51.01 and 45.13, respectively and MOTP/L1 and MOTP/2 of 14.18 and 14.27, respectably in the Waymo's dataset [8] [9] [34]. An adaption of this solution would be a great case study, especially using the NMS algorithm to merge expert models' results.

Other architectures besides those used in the Waymo challenge were explored. FairMOT, for instance, is a great model for future experiments as it claims to outperform state-of-the-art models on public datasets at 30 frames per second. FairMOT is a one-shot tracker that only uses one model for detecting and tracking objects. Although this type of tracker usually has lower accuracies, this is not the case with FairMOT, which was benchmarked in MOT15, MOT16, MOT17 and MOT20, beating all other state-of-the-art methods in the MOTA metric [77]. On the other end, DeepSORT is a two-stage tracker and extension of the SORT algorithm that is one of the most widely used object-tracking architectures [82]. DeepSORT model implementations present respectable benchmarks on public datasets [80], [82]. Nonetheless, what makes this model interesting is the possibility of pairing the YOLOv5 model developed in section 3.3 as a base detection model to a tracking algorithm, a combination that shows promising results [83]. This paring is especially interesting as it would save time and computation resources in the training process. By exploring this possibility, a variation of DeepSORT stood out as a good model choice to train and benchmark in sections 4.3 and 4.4, as there is an implementation that facilitates the use of YOLOv5 models for detection [84]. This implementation feeds the detections generated by YOLOv5 to StrongSORT [85], which combines motion and appearance information based on a re-ID CNN to track objects [84].

StrongSORT revisits the classic DeepSORT tracker, updating the outdated techniques that cause it to underperform compared with state-of-the-art methods [85]. DeepSORT, in short, is a framework composed of two branches: the appearance branch and the motion branch. The appearance branch applies the deep appearance descriptor to the detections in each frame to extract their appearance features. The motion branch uses the Kalman filter algorithm to predict the objects' positions in the current frame. Then, the spatio-temporal dissimilarity between objects being tracked and detections are calculated and used to filter unlikely associations [85].

StrongSort upgrades the appearance feature extractor in the appearance branch, replacing the simple CNN with Bag of Tricks (BoT) [86] and an improved ResNet50-TP [87], making it capable of extracting more discriminative features of the instances being tracked and performing association. The feature bank was also replaced with the feature updating strategy Exponential Moving Average (EMA), which enhances the matching quality and reduces the time consumption. In addition, the Enhanced Correlation Coefficient Maximization (ECC) was adopted for camera motion compensation in the motion branch [88]. NSA Kalman algorithm [89] was also used as a detriment of the vanilla algorithm since it proposes a formula to calculate the noise covariance adaptively. Other alterations came from solving the assignment problem with appearance and motion information and replacing the matching cascade with vanilla global linear assignment [85]. Figure 27 summarizes the framework differences between DeepSORT and StrongSORT [85].

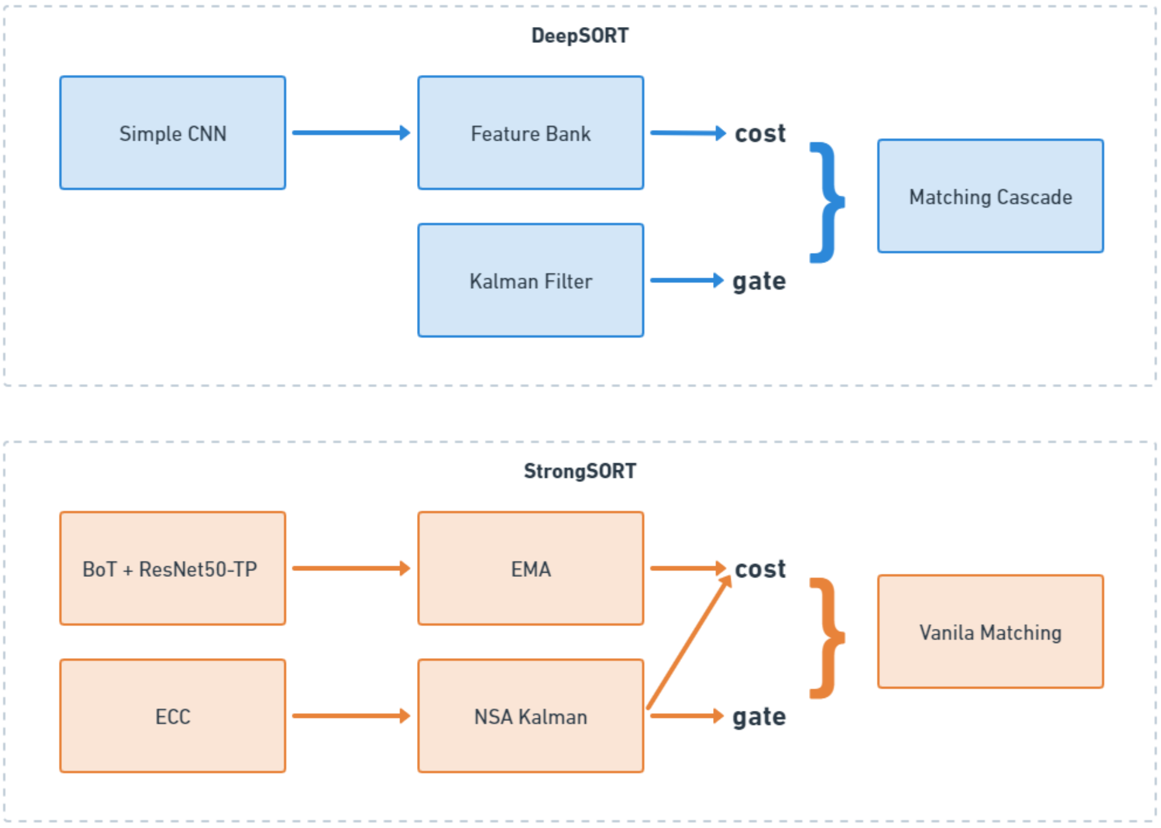


Figure 27 - Framework comparison between DeepSORT and StrongSORT.

The authors of StrongSORT also proposed two algorithms, AFLink and GSI, that can be plugged into various trackers. Their introduction to StrongSORT resulted in StrongSORT++, which presents even better results in MOT17 and MOT20. However, StrongSORT++ will not be used in sections 4.3 and 4.4 as there was not a stable version of StrongSORT++ with YOLOv5 when the experiments of section 4.3 were conducted . In addition, StrongSORT++ post-processing steps can impact tracking speeds for real-time tracking necessary for the experiments and results of sections 5.2 and 5.3 [85].

The re-ID CNN, named OSNet [90], used in the StrongSORT implementation with YOLOv5 replaces the ResNet50-TP model, as illustrated in Figure 28 [84]. This alteration raises a problem for the experiments of 4.3 and 4.4 since the OSNet model, responsible for associating images of the same object taken on different occasions or cameras, is only trained for people re-ID, which means the model will not be equally suitable for the re-ID of other classes [90].

.

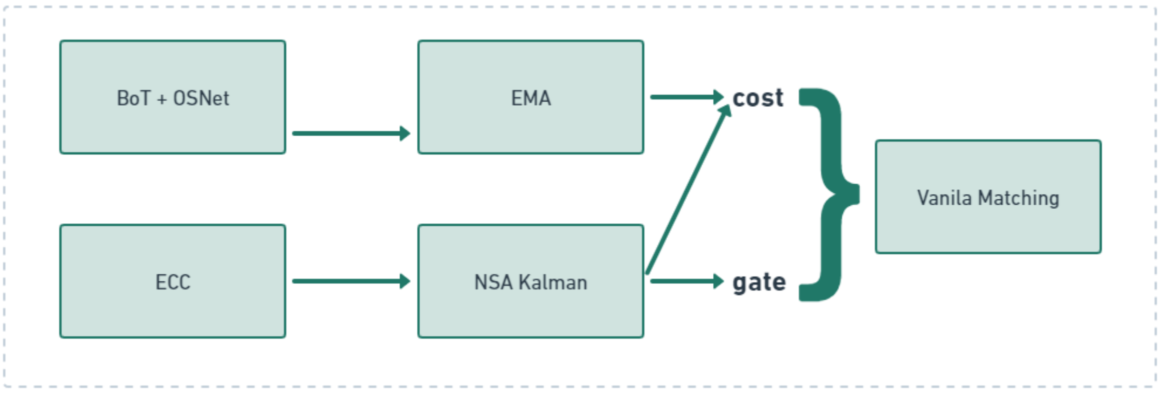


Figure 28 - StrongSORT with YOLOv5 implementation framework.

There are few available results of the used StrongSORT with YOLOv5 implementation on public datasets. The only available results that allow for any correlation between this implementation and other models are the MOT16 evaluation performed on the train split. Important to clarify that this is not an issue, as the training dataset is never used for training. The results were also obtained with a limited detector YOLOv5m which negatively impacts results [84]. Figure 15 shows the results of the models referred in this section on the MOT16 [84].

Table 16 -Trackers evaluation on MOT16. FPS column refers to frames per second.

	MOTA	FPS
DeepSORT	78.0	13.5
StrongSORT	78.3	7.5
FairMOT	73.7	25.9
StrongSORT + YOLOv5 + OSNet	60.1	10.2

4.2 MATERIALS

Similarly, to Object Detection, Object Tracking is also one of the essential prerequisites to autonomous navigation, as it allows the prediction of trajectories and future behaviour. Therefore, the object tracker model also must be reliable and accurate. Furthermore, object tracking, as for Object Detection, requires tracking results in a timeframe under one second with accurate results, operating strictly on the input image.

4.2.1 WAYMO 2D TRACKING CHALLENGE

The Waymo 2D Tracking challenge consists of given a camera image, producing a set of 2D boxes and the correspondence between boxes across frames [33]. The dataset used for this challenge was Waymo Perception Open Dataset. The input data was restricted to camera images, and a causal system was enforced where for a frame, only sensor data up to that point could be used. The classes included in the challenge (vehicle, pedestrian, and cyclist) are annotated with tight-fitting 2D bounding boxes based on the camera images [91]. This challenge used as the primary metric MOTA, and the second metric is MOTP [33]. As already has been mentioned in 4.1, some of the solutions that resulted from this challenge were studied.

4.2.2 DATASET

For the case of the chosen object tracker, the development and testing process turned out to be more complex than the one in section 3.3, as the used tracker is composed of independent components. The solution to this problem was to use different datasets for training and testing the different components of the tracker.

Waymo's dataset described in section 3.2.2 was used for training and testing the tracker's object detector component, YOLOv5. As said in 3.2.2, Waymo's dataset is one of the largest high-quality datasets for autonomous driving. Therefore, this dataset was a logical choice as all the work put into training in section 3.3.3 could be translated to the base detector for the object trackers being developed in 4.3, saving computational time and resources [33].

The Omni-Scale network (OSNet) deep-learning re-ID CNN is used in StrongSORT with YOLOv5 implementation, as explained in 4.1. The Torchreid framework [92] is used to implement this component, which provides a list of image and video datasets compatible for training. Torchreid builds each dataset on top of base classes, which provide basic functions for sampling, reading and pre-processing images [92]. A list of trained re-ID models was made available by the authors of Torchreid. The models' mAP results were compared in three different datasets, Market-1501 [93], DukeMTMC-reID [94] and MSMT17 [95], and the results were considered when choosing the best model for the trackers described in section 4.3.

Market-1501 is a high-quality dataset for person re-ID. This dataset contains 1,501 identities collected by six cameras with 32,668 annotated bounding boxes, plus a distractor set of over 500K irrelevant images and 3,368 query images. This dataset employed a Deformable Part Model detector to draw bounding boxes. All the data was collected in an open environment where at least two cameras captured images of each annotated identity, and each identity had multiple images under the same camera [93].

DukeMTMC-reID is a dataset for person re-ID with more than 2 million frames taken from 1080p, 60fps video recorded by eight static cameras observing more than 2,700 identities over 85 minutes. All identity has a single-frame bounding box, and ground-plane world coordinates across all cameras in which it appears [94], [96].

MSMT17 person re-ID dataset is composed of 180 hours of video taken by fifteen cameras, twelve outdoor and three indoor cameras. The videos cover a long period of time with complex scenes and backgrounds and weather and lighting variations. The dataset contains 4,101 identities and 126,441 bounding boxes drawn with the help of a more reliable bounding box detector Faster RCNN, to the detriment of outdated detectors like Deformable Part Model [95].

Finally, the MOT16 dataset [97] was selected as the dataset to evaluate the developed object trackers. The main reasons for this choice are that it allows for comparisons of results of other trackers that can be seen in 4.1. Furthermore, there is already compatibility in testing the used StrongSORT with YOLOv5 implementation in this dataset, allowing the choice of what classes from the MSCOCO dataset should be tested, which is important to match the capabilities of the object detector used by the tracker [84]. In addition, this dataset has an image size of 1280, coinciding with Waymo's dataset's image size after the padding pre-processing method explained in section 3.3.3 [29]. MOT16 is composed of fourteen sequences that differ between them according to if it is a moving or static camera, according to the viewpoint, and can differ in weather or illumination conditions. Table 17 shows the dataset annotation types and their distribution within the dataset [97].

Table 17 - MOT16 types of annotations.

Pedestrian	292733
Person on vehicle	2430
Car	8818
Bicycle	26046
Motorbike	2550
Non-motorized vehicle	11
Static person	27966
Distractor	8238
Occluder on the ground	73319
Occluder full	33473
Reflection	948
Total	476532

4.3 METHODS

This section describes the approaches to develop and test a StrongSORT object tracker model.
The container described in the section 3.3 was also used for development and testing of the object tracker.
The data extracted from Waymo's dataset described in sections 3.3.1 and 3.3.2 was indirectly used in the development of the object tracker in the sense that the object detector trained with this data would be the base detector for the object tracker. For this reason, all data analyses and pre-processing done previously continues to be relevant in this section and to the results of 4.4. One of the things to consider relative to Waymo's dataset is that it is has data of three different classes vehicles, pedestrians, and cyclists. This information is relevant because these are the classes that will be detected and then tracked by the object tracker.

The used implementation of StrongSORT with YOLOv5 uses OSNet [90] model with the help of the Torchreid framework [92], wich is trained exclusively on person re-ID data. For this reason, it was necessary to resort to all person re-ID datasets as the likes of Market-1501, DukeMTMC-reID and MSMT17, although it is not the ideal type of data considering the classes that the object detector was trained to detect.

MOT16 [97] was chosen as the dataset to evaluate the tracker. This dataset is relevant because there are public results of the models referred to in section 4.1 and results from the tracker implementation used, which allows for meaningful analysis and conclusion of the obtained results. A second reason for the use of this dataset for evaluation is, as already mentioned in section 4.2.2, that the used StrongSORT with YOLOv5 implementation allows choosing what classes from MSCOCO dataset [29] should be tracked. This feature helps match the objects tracked to the capabilities of the object detector used by the tracker. In this case, the object detector can detect pedestrians, vehicles, and cyclists.

On the other hand, MOT16 has all annotation types present in Table 17. MOT16 has more classes than Waymo's datasets, and another issue is that the Waymo dataset and MOT16 seem to have different definitions to represent the same objects. For example, a Car and Non-motorized vehicle in the MOT16 dataset would be simply labelled as a vehicle in the Waymo dataset. Other problems with the differences in labelling are the bicycle in MOT16 and cyclist in Waymo, or person on a vehicle on MOT16 and Pedestrian in Waymo dataset, that there is no clear answer if a person on a vehicle will be classified as a pedestrian and vehicle or simply a vehicle. For this reason, it makes sense to evaluate the StrongSORT implementation with not only YOLOv5 weights trained in Waymo's dataset but also with provided weights of the YOLOv5 team in MS-COCO dataset since these are the recommended YOLOv5 weights of the authors of the StrongSORT with YOLOv5 implementation [84].

A CONDA [48] environment was created where all the package requirements necessary for executing the StrongSORT with YOLO5 implementation were installed. During the development of the object tracker, the repository of the used implementation underwent significant changes while it was being studied. The tracker passed from DeepSORT to StrongSORT, which implicated a new environment configuration. During this transition, some problems arise in the form of conflict errors between packages that lead to the creation of a new environment for the new repository version.

After the environment set-up, a script that uses several images of the Waymo dataset to generate a video from the images in temporal order was created. The results of the StrongSort tracker, with the best detection model of YOLOv5 from section 3.4, on the generated video, can be seen in section 4.4.

Though the results were satisfactory, it was not a clear way of classifying the developed trackers. For that, capabilities within the used repository were used to classify and compare trackers on the MOT16 dataset, which results can be seen in section 4.4.

One thing considered before the model evaluation was the list of pre-trained models made available by the authors of the Torchreid framework [92]. The authors of the framework performed different experiments on the different OSNet models.

OSNet achieves state-of-the-art performance, outperforming re-ID models with a far larger number of parameters, in datasets like Market-1501 [93], DukeMTMC-reID [94] and MSMT17 [95]. Table 18 summarizes the OSNet model characteristics and mean AP results on the different datasets [98].

Table 18 - OSNet model characeristics.

	Param (10^6)	GFLOPs	Market-1501	DukeMTMC-reID	MSMT17
OSNet	2.2	0.98	86.7	76.6	55.1

In order to improve domain generalisation across datasets and better cope with cross-dataset discrepancies, the Torchreid team incorporated instance normalisation into OSNet. This variant of OSNet is called OSNet-AIN. OSNet-AIN has a remarkable generalisation ability, having good performances on unseen target domains and maintaining strong source domain performance, requiring neither the target domain data nor per-domain training [98].

The fact that the experiments of section 4.4 aim to evaluate the tracker capability of different classes not only classes referent to people. For this reason, an OSNet-AIN model trained by the Torchreid team on MSMT17 [95] dataset and tested on Market-1501 [93] and DukeMTMC-reID [94] datasets seems to be the most capable available model for tracking objects it was not trained for. Therefore, this was the chosen model to incorporate into the StrongSORT framework and perform the experiments of section 3.4.

4.4 RESULTS & DISCUSSION

Figure 29 and Figure 30 are examples of the tracking results obtained by a StrongSORT tracker in images from the same scene extracted from the Waymo's Perception dataset, with a YOLOv5 detector trained in this dataset and a OSNET AIN trained in MSMt17 dataset. In the bounding boxes present in the figures it is possible to see the tracking id assigned to the object, the class, and the confidence in the detection. As can be seen in the figures, at first glance, the developed tracker with the YOLOv5 Padding baseline model seems to be attempting to track all objects detected, even those that are not people.

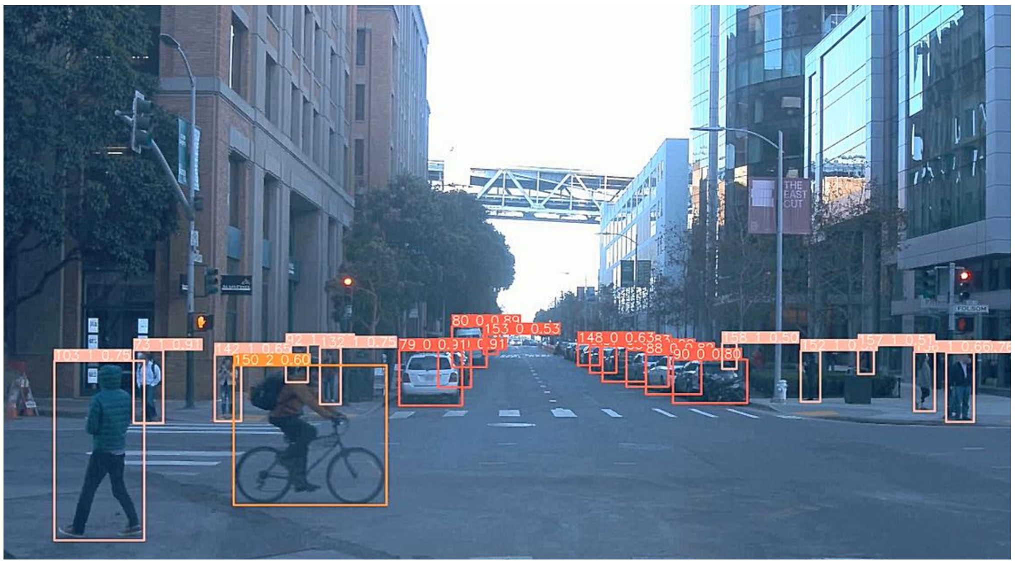


Figure 29 - Tracking results example 1.

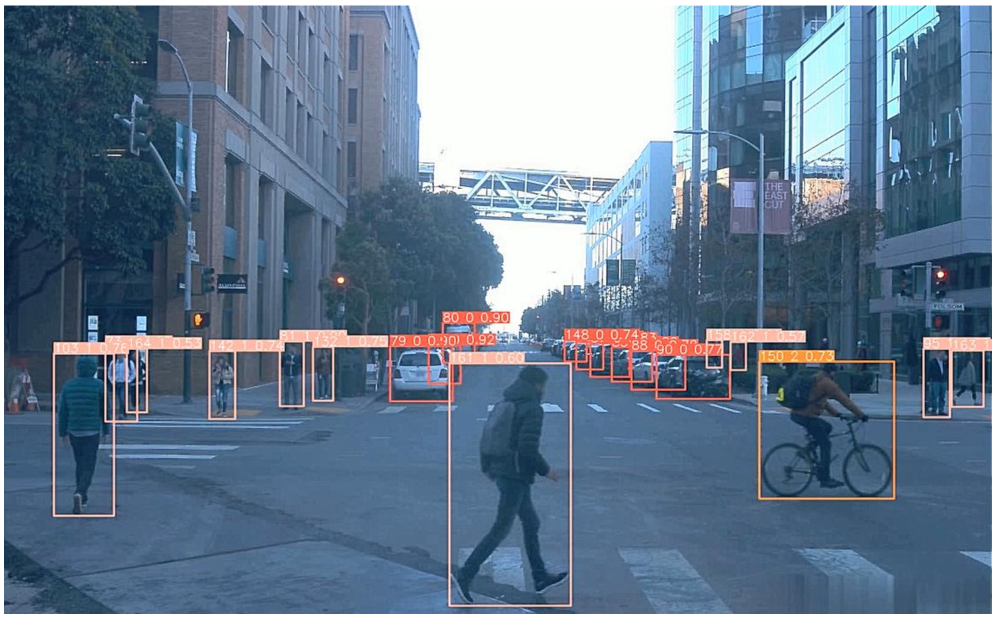


Figure 30 - Tracking results example 2.

As explained in section 4.2.2, the developed trackers were evaluated using the MOT16 dataset. The results of the trackers evaluation were obtained in the hardware described in section 3.3, and since the OSNET feature extractor used is only trained for recognising features of people, only the Pedestrian class was evaluated.

The first evaluated trackers used the recommended combination of YOLO weights by the authors of the used repository. The YOLO weights were trained in the CrowdHuman dataset [99], with data on human instances. Two different OSNET models were combined with this YOLO detector: the OSNET weights trained in the DukeMTMC-reID, and OSNET AIN trained in the MSMt17 dataset. The evaluation of these trackers for the Pedestrian class with a threshold of 0.5 can be seen in Table 19.

Table 19 - Object trackers with YOLOv5 detector trained on CrowdHuman.

	MOTA	MOTP
OSNET	60.10	79.53
OSNET AIN	59.91	79.38

It was expected that the OSNET AIN model would improve the MOTA and MOTP metrics as it is a more capable model than the OSnet in generalizing feature extraction. Also, it was trained in MSM17, a larger dataset that presents a larger variety of data, as explained in section 4.2.2. However, as seen in Table 19, this was not the case. Instead, the slight drop in performance can be a result of the fact that a significant portion of MOT16 data comes from a stationary camera on a pedestrian street, at night, with an elevated viewpoint scenario which is closer to the type of data present in DukeMTMC-reID dataset.

Finally, the baseline YOLOv5 model trained with data that suffered padding described in sections 3.3.3 and 3.4 was also evaluated with the two OSNET models already mentioned in this section. The results of these combinations can be seen in Table 20.

Table 20 - Object trackers with YOLOv5 detector trained in Waymo detector.

	MOTA	MOTP
OSNET	42.55	80.20
OSNET AIN	42.48	80.26

Table 20 suggests that the MOTA metric is much lower for the trackers using the YOLO model trained in Waymo’s dataset. This phenomenon is probably caused by the fact that the data the detector was trained for was not people but pedestrians. The MOT16 dataset has data in different scenarios where people's instances are not pedestrian. These differences hurt the capacity of the model to detect and, by repercussion, track a large number of instances. Therefore, MOTA, the metric that evaluates the number of errors that occur during tracking, is expected to decrease using the YOLO model trained in Waymo's dataset. On the other hand, the MOTP metric that evaluates the tracking precision had a slight improvement relative to the models that use the default YOLO model. One explanation for this result is that a lower number of instances are being correctly tracked, and the ones being tracked are likely the easier ones to localise.

Comparing the obtained results to Table 16, the evaluated object trackers have lower MOTA scores than other state-of-the-art trackers. However, these results must be scrutinised as the used detectors were trained on different datasets, which limits the obtainable results.

5 MULTI-ACCESS EDGE COMPUTING (MEC)

5.1 Literature Review

Multi-access Edge Computing (MEC) provides computing at the network's edge by placing MEC hosts nodes as close as possible to the end user [100]. Standardized APIs allow applications and content providers to take advantage of computing capabilities at the edge of the network and to receive information about the status of the network and its users [16], [100]. The main differentiators of a MEC environment are extreme user proximity, ultra-low latency, high bandwidth, real-time access to radio network and context information, and location awareness. On top of a secure environment, it offers to provide and consume services [16]. Figure 31 shows a simplified view of the environment offered to MEC applications (MEC app). A Client app running on a User Equipment (UE) may request computing services from a MEC app running on a MEC host, usually deployed on the edge of the network. On the other hand, this MEC app may also require services provided by the MEC platform (represented by the red boxes "MEC services"), which may be discovered via a standardized Restful API, Mp1.

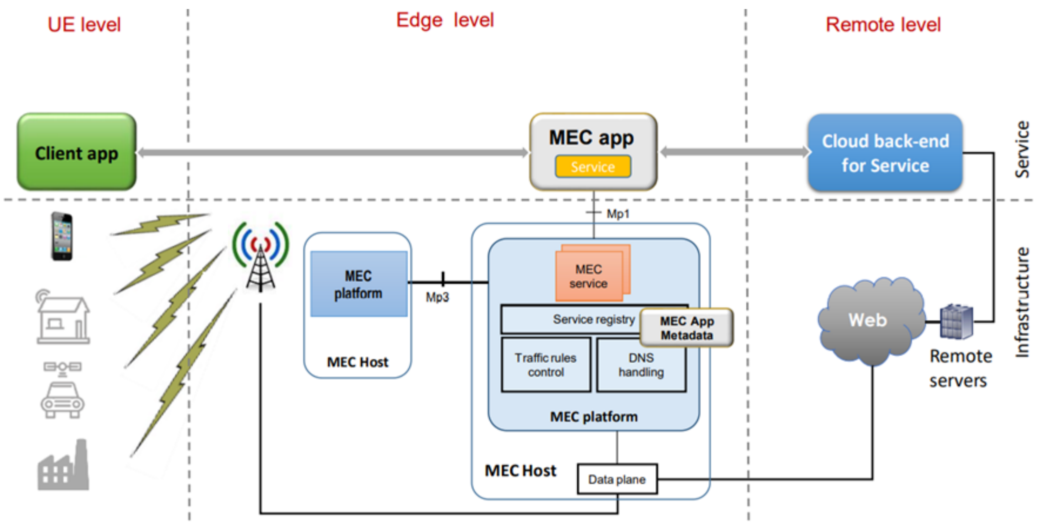


Figure 31 - ETSI MEC environment. Retrieved from [16]

MEC is compatible with any application with the advantage that if the application is designed to run on MEC, it can give additional information on expected latency, throughput and available MEC services. This means edge applications can benefit from low latency and high throughput in a controllable way [16].

MEC applications must, however, meet certain criteria. They run on a virtualized environment, such as a virtual machine or container. MEC applications must support a DNS-based solution, and a domain name is needed so that a client application can gain access in order to redirect traffic. If a service is required to be available regardless of whether a local MEC system is available, then a back-end service hosted in the cloud will be required as a fall-back. User context transfer data is also a sensitive matter as it may not be allowed to transfer data from one jurisdiction to another for legal reasons. Lastly, these applications need to be aware of resource requirements (like CPU, storage, and network resources), latency tolerance and if it provides or requires one or more services [16].

5.2 SIMPLIFIED ARCHITECTURE

The goal of this section is to develop an edge network. Figure 32 illustrates how an edge server may provide services to connected vehicles. The vehicles send images to the MEC host to detect and track detected objects. Next, the applications will process the received images by running the AI models and will return the results to the vehicles.

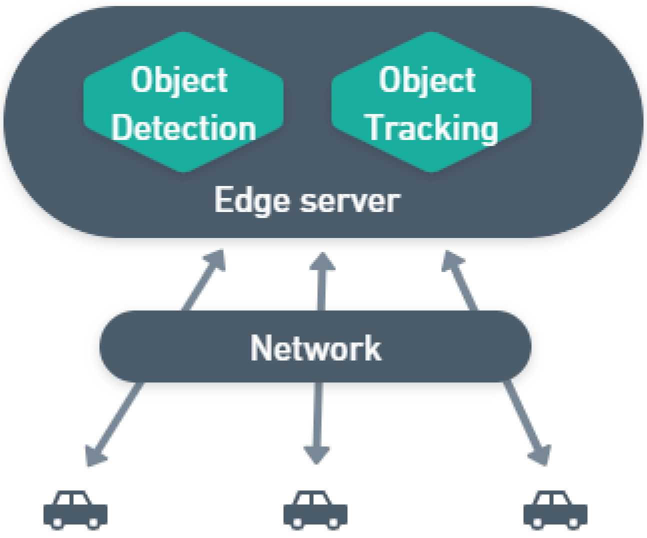


Figure 32 - Edge Computing application architecture

Due to the inaccessibility of an actual edge network to experiment on, a simulated environment was considered. Simu5G [101] emerged as a solution to this problem, as it allows for network simulations where 4G and 5G coexist and benchmark solutions. Furthermore, Simu5G includes a model of ETSI MEC that allows for rapid prototyping of MEC applications in credible and controllable 5G mobile network scenarios. In addition, Simu5G provides MEC services able to produce real information from mobile networks and offer them through ETSI compliant APIs.

Simu5G can simulate communications in vehicular networks with the help of two simulation tools: Veins [102] and SUMO [103]. Simu5G also requires the installation of OMNeT++ [104] simulation framework and INET-Framework [105].

Due to time constraints, it was not possible to successfully integrate all these tools and it was decided to employ the simplified scenario that can be seen in Figure 33. This scenario contains an external application that emulates the behaviour of an application inside the User equipment (UE) that sends an image through the network. This application is connected to the simulator. Inside the simulator, a gNB entity communicates with the UE and MEC system using the New Radio protocol stack, allowing packets to flow between the UE and the MEC system [106]. The MEC app is also running as an external application to the simulated Simu5G environment, as is the case for the UE app, communicating with the simulated environment with the help of an interface, and is responsible for executing AI models to detect and track incoming images from the UEs.

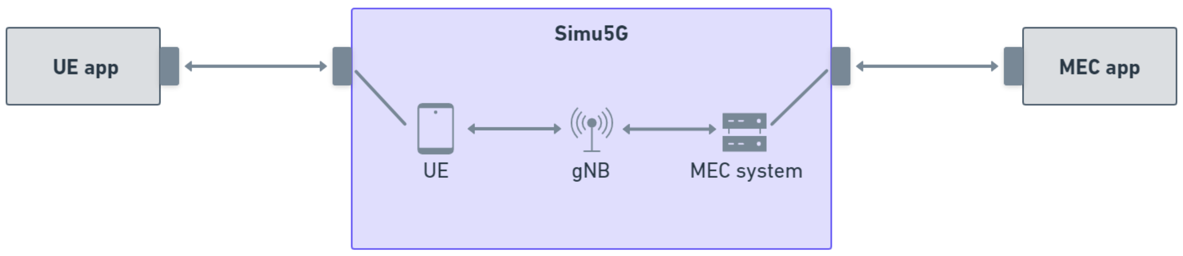


Figure 33 - Simulated Edge Computing network environment.

5.3 RESULTS

The real value of edge AI computing comes from collecting and processing data and making predictions and decisions in real-time.

The average execution time of the detection and tracking models was calculated in the hardware described in section 3.3, the results can be seen in Table 21 , where it is also possible to see what each execution time translates to the distance travelled by a vehicle moving at 100km/h. As can be deduced by the distances travelled, the execution times of the detection and tracking models turned out to be higher than what is required for real-time detection and tracking for autonomous driving. However, these results can be mitigated by more powerful hardware and custom accelerator AI chips for DL applications that allow for faster execution times relative to traditional chip architectures.

Table 21 - Detection and tracking average execution times.

	Execution time (ms)	Distance traveled by a vehicle with a 100 km/h velocity (m)
YOLOv5	20.6	0.57
StrongSORT	58.0	1.61
YOLOV5 + StrongSORT	78.6	2.18

The EC network simulation environment was developed in a virtual machine with low computation power. For this reason, the execution times of the detection and tracking models in the simulation environment are exponentially higher. The fact that these limitations exist in terms of computational power means that it is impossible to draw valid conclusions from the execution times of the models.

Low latency for automotive safety is a must. However, the response time for sending an image with resolutions of 1280×1920 pixels, the resolution used for the developed detectors and trackers of sections 3.3.3 and 4.3, through the simulated network with a distance of only 50m between the UE and the MEC system and executing the AI models were abnormally high, surpassing 12 seconds. This phenomenon is possibly a result of packets being sent too fast, resulting in too much traffic into the test bed shown in Figure 33 where Simu5G was used in emulation mode, and making it unable to run in real time. Real-time emulation is only possible when the complexity of the simulation is such that simulation time can run in parallel with wall-clock time [101]. One possible solution to this problem in a future iteration of this experiment is not using external applications and simulating the whole environment.

However, it was still possible to evaluate, in an isolated manner, the time it took to uplink, transfer the image from the UE to the MEC system, and downlink, transfer the image with the detection from the MEC system to the UE. The uplink and downlink times were similar and amounted to 38.64 ms. Looking at the example used previously in this section for AI models execution time, for a vehicle moving at 100 km/h, this time amounts to a total of 1.07m travelled which is too big of a distance waiting for information in an autonomous driving scenario.

Although further experiments have to be conducted to test the viability of sending frames to be processed in a MEC system, evaluating the benefits and flaws of the integration of MEC in autonomous driving is still possible.

MEC-based Vehicle to Infrastructure (V2I) Data Offloading faces many challenges. For example, the high mobility of vehicles makes handover management complex. In addition, edge server resources need to be managed so that the vehicle is constantly connected to a server that can guarantee the quality of service required by the client application. Finally, the fact that different vehicle users can access edge servers makes it imperative to have robust protection mechanisms for security and privacy [108]. For these reasons, relying on the edge server application for detection and tracking of results increases potential points of failure, making it more dangerous than running these models locally.

However, an edge server running this type of application can still have a meaningful impact on road safety. An edge server can analyse and process data in real-time and alert vehicles of possible danger. Edge servers can also collect information like vehicle location and speed and collect weather and road conditions, allowing the server to possibly make decisions to control the traffic flow and provide vehicles with optimal routes [108].

6 CONCLUSION

This work aimed to investigate Artificial Intelligence techniques to detect and track objects and test them in an MEC environment. This study is divided into three components: object detection, object tracking and evaluation of the feasibility of processing data in MEC servers. Several architectures, hyperparameters, pre-processing techniques, and evaluating metrics were studied for object detection and tracking. In addition, the MEC fundamentals and architectures and simulation tools for these environments were also studied.

For Object detection, the YOLOv5X model was used, which was trained in the Waymo Perception dataset with labelled data of vehicles, pedestrians, and cyclists. One of the most challenging parts of this work was analysing and pre-processing Waymo's dataset data. The main difficulties came from the fact that the dataset proved too large for the available hardware, and the way the data is stored in public repositories forced a considerable number of data transformation and filtering operations, so the final dataset was in the correct format for training a YOLOv5 model. Another issue encountered was the hardware limitations and the fact that the computation resources were being shared with other colleagues, which limited the amount of training available to the developed models.

Looking at the results, the strategy of increasing the image size throw padding and lowering the batch size proved to be a winning one as the baseline model trained on a lower image size achieved a mAP on the used subset of Waymo's dataset of 77.6, and the model with a larger image size and data that suffered padding achieved a mAP on the same subset of data of 86.2. Also, the increase of the augmentation hyperparameters caused a slight improvement to the best mAP of the baseline models with a mAP of 86.6. In addiction, by analysis of the training loss and validation loss, the alterations in hyperparameters resulted in a decrease in the objectness losses and classification losses and seem to have delayed the improvement stagnation of the model during training. As explained in section 3.4, although there is not a fair comparison between the developed models and the results of the leader board of the Waymo 2D Detection Challenge, these experiments showed promising results on the subset of data used.

For Object tracking, StrongSORT was used. The StrongSORT model is composed of an object detector and a CNN used for feature recognition. StrongSORT uses the YOLOv5 model for object detection and the OSNET model for feature recognition. Some combinations of feature recognition CNNs and object detectors trained in different datasets were combined and evaluated with StrongSORT in the MOT16 dataset. Though, the classes being tracked were restricted to tracking people, given that the only publicly available OSNET feature extractor models are exclusively trained in people re-ID datasets.

Although it was not an evaluation, the results of tracking images from the Waymo's Perception dataset showed that the trackers with a YOLOv5 detector trained in this dataset, even though they have a feature extractor only trained for people, still attempted to track the other classes present in the dataset.
The evaluation of the developed trackers was done in MOT16, and the results on the MOTA metric were lower than other state-of-the-art architectures on the same dataset. Some possible explanations and comparisons between the evaluated trackers are given in section 4.4, but the main reason for these results comes down to the fact that both OSNET and YOLO models used did not train in the MOT16 dataset, which negatively impacts the results.
Relative to MEC, a simulated environment was created for experimentation purposes. Although further experiments have to be conducted, our preliminary results show that, if a vehicle is dependent on detections and tracking results calculated by an edge server to make decisions, the EC turns to a liability instead of an advantage. Despite this, implementing this type of system can still help, for instance, improve road safety if it is used to improve information dissemination between the vehicles by informing possible road situations like accidents, road obstructions and traffic state.
Although this dissertation helped summarize progress and withdraw interesting conclusions in the object detection, object tracking and EC fields, there are still many exciting paths to develop this work further.
One thing to explore is the hyperparameter evolution available in YOLOv5. This technique is a method of hyperparameter optimization using a Genetic Algorithm (GA) that has the potential to improve further the results of the object detectors, which was not explored due to available limited computing power.
During the development of this work, multiple different implementations of YOLO were released at a rapid pace. Some of the most interesting implementations are YOLOX, PP-YOLOE, YOLO6 and YOLO7. All of these show promising results on the MS-COCO dataset and are worth exploring and comparing to YOLOv5 in a future iteration of this work.
In the Waymo 2D detection challenge, the solution of the second-place team was studied. This team used both one-stage and two-stage detectors and had expert models for different classes and times of the day. This team used this variety of models to fuse detections to improve results. A similar strategy can be pursued in a future iteration of this work.
Relatively to the tracking part of this work, in a future iteration, the implementation of a StrongSORT tracker with a CNN responsible for feature extraction not exclusively trained in people data would lead to better results for autonomous driving. Ideally, the CNN would be trained in the same classes as the object tracker used, and the evaluation process would be conducted in the same dataset where the object detector and feature extractor were trained. Another case of study would be to conduct experiments on

the FairMOT model so comparisons between one-stage and two-stage trackers relative to precision and execution time could be withdrawn.

Finally, regarding the MEC part of this study, more experiments could be conducted to test the limits of the network and the MEC system capabilities by congesting the network with multiple users and requests to access the existing MEC applications. Moreover, another case study is to use GatcomSUMO and OpenStreetMap to develop experiments on real-world scenarios. These real-world scenarios could help determine how extensive and capable a MEC system would need to be to serve vehicular applications such as the ones studied. Ultimately the same experiments could be made on a real edge network to evaluate if this work's findings hold up in a real-work scenario.

Autonomous driving has the potential to save millions of lives and advance our society by turning road driving safer and more fluid. Artificial intelligence will be essential to this future by allowing a vehicle to have the road awareness of a human and a machine's reaction times and precision. Edge computing can play an active role in this feature by offering powerful hardware at a lower latency which is essential to a critical application like the ones needed for autonomous driving.

REFERENCES

[1] Z. Zou, Z. Shi, Y. Guo, J. Ye, and S. Member, “Object Detection in 20 Years: A Survey,” May 2019.

[2] S. K. Pal, A. Pramanik, J. Maiti, and P. Mitra, “Deep learning in multi-object detection and tracking: state of the art”, doi: 10.1007/s10489-021-02293-7.

[3] “J3016_202104: Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles - SAE International,” 2021.

[4] P. Soviany and R. Ionescu, “Optimizing the Trade-off between Single-Stage and Two-Stage Deep Object Detectors using Image Difficulty Prediction”.

[5] “LIDAR vs. Camera — Which Is The Best for Self-Driving Cars? | by Vincent Tabora | 0xMachina | Medium.” https://medium.com/0xmachina/lidar-vs-camera-which-is-the-best-for-self-driving-cars-9335b684f8d (accessed Oct. 20, 2021).

[6] “Autonomous Vehicles Drive AI Advances for Edge Computing - 3D InCites.” https://www.3dincites.com/2021/07/autonomous-vehicles-drive-ai-advances-for-edge-computing/ (accessed Oct. 20, 2021).

[7] “AI vs. Machine Learning vs. Deep Learning vs. Neural Networks: What’s the Difference? | IBM.” https://www.ibm.com/cloud/blog/ai-vs-machine-learning-vs-deep-learning-vs-neural-networks (accessed Aug. 18, 2022).

[8] B. Mahesh, “Machine Learning Algorithms-A Review,” 2019, doi: 10.21275/ART20203995.

[9] S. Kaur and S. Jindal, “A Survey on Machine Learning Algorithms,” pp. 6–14, 2016.

[10] A. Abraham, “Artificial Neural Networks,” *Handbook of measuring system design*, Jul. 15, 2005.

[11] P. P. Shinde and S. Shah, “A Review of Machine Learning and Deep Learning Applications,” *Proceedings - 2018 4th International Conference on Computing, Communication Control and Automation, ICCUBEA 2018*, Jul. 2018, doi: 10.1109/ICCUBEA.2018.8697857.

[12] N. O’ Mahony *et al.*, “Deep Learning vs. Traditional Computer Vision”.

[13] D. Shen, G. Wu, and H.-I. Suk, “Deep Learning in Medical Image Analysis,” 2017, doi: 10.1146/annurev-bioeng-071516.

[14] S. Grigorescu, B. Trasnea, T. Cocias, and G. Macesanu, “A Survey of Deep Learning Techniques for Autonomous Driving,” 2020.

[15] W. Z. Khan, E. Ahmed, S. Hakak, I. Yaqoob, and A. Ahmed, “Edge computing: A survey,” *Future Generation Computer Systems*, vol. 97, pp. 219–235, Aug. 2019, doi: 10.1016/J.FUTURE.2019.02.050.

[16] D. Sabella *et al.*, “Developing Software for Multi-Access Edge Computing,” 2019.

[17] P. Viola and M. Jones, "Fast and Robust Classification using Asymmetric AdaBoost and a Detector Cascade," vol. 14, 2001.

[18] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," *Proceedings - 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2005*, vol. I, pp. 886–893, 2005, doi: 10.1109/CVPR.2005.177.

[19] P. Felzenszwalb, D. McAllester, and D. Ramanan, "A discriminatively trained, multiscale, deformable part model," *26th IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, 2008, doi: 10.1109/CVPR.2008.4587597.

[20] C. B. Murthy, M. F. Hashmi, N. D. Bokde, and Z. W. Geem, "Investigations of Object Detection in Images/Videos Using Various Deep Learning Techniques and Embedded Platforms—A Comprehensive Review," *Applied Sciences 2020, Vol. 10, Page 3280*, vol. 10, no. 9, p. 3280, May 2020, doi: 10.3390/APP10093280.

[21] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks".

[22] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition; Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition," *IEEE Trans Pattern Anal Mach Intell*, vol. 37, 2015, doi: 10.1109/TPAMI.2015.2389824.

[23] C.-Y. Wang, A. Bochkovskiy, and H.-Y. Liao, "YOLOv4: Optimal Speed and Accuracy of Object Detection," Apr. 2020.

[24] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," *IEEE Trans Pattern Anal Mach Intell*, vol. 39, 2017, doi: 10.1109/TPAMI.2016.2577031.

[25] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection".

[26] W. Liu *et al.*, *SSD: Single Shot MultiBox Detector*, vol. 9905 LNCS. Springer, Cham, 2016. doi: 10.1007/978-3-319-46448-0_2.

[27] D. Thuan, "Evolution of YOLO Algorithm and YOLOV5: The state-of-the-art object detection algorithm," 2021.

[28] R. Simhambhatla, K. Okiah, S. Kuchkula, and R. Slater, "Self-Driving Cars: Evaluation of Deep Learning Techniques for Object Detection in Different Driving Conditions," *SMU Data Science Review*, vol. 2, no. 1, May 2019.

[29] T. Y. Lin *et al.*, “Microsoft COCO: Common Objects in Context,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 8693 LNCS, no. PART 5, pp. 740–755, May 2014, doi: 10.1007/978-3-319-10602-1_48.

[30] J. Mao *et al.*, “One Million Scenes for Autonomous Driving: ONCE Dataset,” Jun. 2021.

[31] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, “Vision meets robotics: The KITTI dataset:,” *http://dx.doi.org/10.1177/0278364913491297*, vol. 32, no. 11, pp. 1231–1237, Aug. 2013, doi: 10.1177/0278364913491297.

[32] H. Caesar *et al.*, “nuScenes: A Multimodal Dataset for Autonomous Driving.” pp. 11621–11631, 2020.

[33] P. Sun *et al.*, “Scalability in Perception for Autonomous Driving: Waymo Open Dataset”.

[34] Y. Wang *et al.*, “1st Place Solutions for Waymo Open Dataset Challenges – 2D and 3D Tracking,” Jun. 2020.

[35] S. Chen *et al.*, “2nd Place Solution for Waymo Open Dataset Challenge – 2D Object Detection,” Jun. 2020.

[36] B. Wu, F. Iandola, P. H. Jin, and K. Keutzer, “SqueezeDet: Unified, Small, Low Power Fully Convolutional Neural Networks for Real-Time Object Detection for Autonomous Driving”.

[37] Y. You *et al.*, “Pseudo-LiDAR++: Accurate Depth for 3D Object Detection in Autonomous Driving,” Jun. 2019.

[38] P. Li, X. Chen, and S. Shen, “Stereo R-CNN Based 3D Object Detection for Autonomous Driving.” pp. 7644–7652, 2019.

[39] T. Wei, Q. Lei, H. Zhong, and Y. Cao, “Apply and Optimize 2D Object Detection in Assembling Components,” *2021 International Conference on Electronic Information Engineering and Computer Science, EIECS 2021*, pp. 763–768, Sep. 2021, doi: 10.1109/EIECS53707.2021.9588066.

[40] G. Jocher *et al.*, “ultralytics/yolov5: v6.0 - YOLOv5n ‘Nano’ models, Roboflow integration, TensorFlow export, OpenCV DNN support.” Oct. 12, 2021. doi: 10.5281/ZENODO.5563715.

[41] M. A. Duran-Vega, M. Gonzalez-Mendoza, L. Chang, and C. D. Suarez-Ramirez, “TYolov5: A Temporal Yolov5 Detector Based on Quasi-Recurrent Neural Networks for Real-Time Handgun Detection in Video,” Nov. 2021.

[42] S. Jin and L. Sun, “Application of Enhanced Feature Fusion Applied to YOLOv5 for Ship Detection,” pp. 7242–7246, Dec. 2021, doi: 10.1109/CCDC52312.2021.9602100.

[43] A. Paszke *et al.*, “PyTorch: An Imperative Style, High-Performance Deep Learning Library,” 2019.

[44] Z. Cai and N. Vasconcelos, “Cascade R-CNN: Delving Into High Quality Object Detection.” pp. 6154–6162, 2018.

[45] X. Zhou, D. Wang, and P. Krähenbühl, “Objects as Points,” Apr. 2019.

[46] G. Lewis, “Object Detection for Autonomous Vehicles”.

[47] “2D Detection – Waymo.” https://waymo.com/open/challenges/2020/2d-detection/# (accessed Dec. 02, 2021).

[48] “Anaconda Software Distribution.” Anaconda Inc., 2020.

[49] “Waymo Open Dataset Tutorial.ipynb - Colaboratory.” https://colab.research.google.com/github/waymo-research/waymo-open-dataset/blob/r1.0/tutorial/tutorial.ipynb (accessed Jan. 22, 2022).

[50] “GitHub - jupyter/jupyter: Jupyter metapackage for installation, docs and chat.” https://github.com/jupyter/jupyter (accessed Jul. 21, 2022).

[51] “Download – Waymo.” https://waymo.com/intl/en_us/dataset-download-terms/ (accessed May 10, 2022).

[52] “waymo-research/waymo-open-dataset: Waymo Open Dataset.” https://github.com/waymo-research/waymo-open-dataset (accessed May 09, 2022).

[53] “TFRecord e tf.train.Example | TensorFlow Core.” https://www.tensorflow.org/tutorials/load_data/tfrecord (accessed May 10, 2022).

[54] “Tar - GNU Project - Free Software Foundation.” https://www.gnu.org/software/tar/ (accessed Jul. 21, 2022).

[55] “gsutil Error: serviceexception 401 anonymous caller does not have storage objects list - cpming.” https://blog.cpming.top/p/gsutil-serviceexception-401-anonymous-caller (accessed May 06, 2022).

[56] G. Bradski, “The OpenCV Library.” 2000.

[57] “How to Train YOLOv5 On a Custom Dataset.” https://blog.roboflow.com/how-to-train-yolov5-on-a-custom-dataset/ (accessed Jan. 22, 2022).

[58] P. Umesh, “Image Processing in Python,” *CSI Communications*, vol. 23, 2012.

[59] J. Nickolls, I. Buck, M. Garland, and K. Skadron, “Scalable Parallel Programming with CUDA,” *Queue*, vol. 6, no. 2, pp. 40–53, Mar. 2008, doi: 10.1145/1365490.1365500.

[60] M. Claesen and B. de Moor, “Hyperparameter Search in Machine Learning,” Feb. 2015, doi: 10.48550/arxiv.1502.02127.

[61] M. Mitchell, *An introduction to genetic algorithms*. 1996.

[62] L. Biewald, “Experiment Tracking with Weights and Biases,” 2020. https://www.wandb.com/ (accessed Jul. 25, 2022).

[63] G. Ciaparrone, F. Luque Sánchez, S. Tabik, L. Troiano, R. Tagliaferri, and F. Herrera, “Deep learning in video multi-object tracking: A survey,” *Neurocomputing*, vol. 381, pp. 61–88, Mar. 2020, doi: 10.1016/J.NEUCOM.2019.11.023.

[64] Y. Z. Cheong and W. J. Chew, “The Application of Image Processing to Solve Occlusion Issue in Object Tracking”, doi: 10.1051/matecconf/201815203001.

[65] A. Dutta *et al.*, “Vision Tracking: A Survey of the State-of-the-Art,” *SN Computer Science 2020 1:1*, vol. 1, no. 1, pp. 1–19, Jan. 2020, doi: 10.1007/S42979-019-0059-Z.

[66] Y. Xiang, A. Alahi, and S. Savarese, “Learning to Track: Online Multi-Object Tracking by Decision Making”.

[67] P. Li, D. Wang, L. Wang, and H. Lu, “Deep visual tracking: Review and experimental comparison,” *Pattern Recognit*, vol. 76, pp. 323–338, Apr. 2018, doi: 10.1016/J.PATCOG.2017.11.007.

[68] X. Zhou, V. Koltun, and P. Krähenbühl, “Tracking Objects as Points”.

[69] D. A. Ross *et al.*, “Incremental Learning for Robust Visual Tracking,” *International Journal of Computer Vision 2007 77:1*, vol. 77, no. 1, pp. 125–141, Aug. 2007, doi: 10.1007/S11263-007-0075-7.

[70] B. Babenko, M. H. Yang, and S. Belongie, “Robust object tracking with online multiple instance learning,” *IEEE Trans Pattern Anal Mach Intell*, vol. 33, no. 8, pp. 1619–1632, 2011, doi: 10.1109/TPAMI.2010.226.

[71] Z. Kalal, K. Mikolajczyk, and J. Matas, “Tracking-learning-detection,” *IEEE Trans Pattern Anal Mach Intell*, vol. 34, no. 7, pp. 1409–1422, 2012, doi: 10.1109/TPAMI.2011.239.

[72] C. Bao, Y. Wu, H. Ling, and H. Ji, “Real time robust L1 tracker using accelerated proximal gradient approach,” *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 1830–1837, 2012, doi: 10.1109/CVPR.2012.6247881.

[73] Y. Wu, J. Lim, and M. H. Yang, “Online object tracking: A benchmark,” *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 2411–2418, 2013, doi: 10.1109/CVPR.2013.312.

[74] W. Zhong, H. Lu, and M. H. Yang, “Robust object tracking via sparse collaborative appearance model,” *IEEE Transactions on Image Processing*, vol. 23, no. 5, pp. 2356–2368, 2014, doi: 10.1109/TIP.2014.2313227.

[75] S. Hare *et al.*, “Struck: Structured Output Tracking with Kernels,” *IEEE Trans Pattern Anal Mach Intell*, vol. 38, no. 10, pp. 2096–2109, Oct. 2016, doi: 10.1109/TPAMI.2015.2509974.

[76] X. Jia, H. Lu, and M. H. Yang, “Visual Tracking via Coarse and Fine Structural Local Sparse Appearance Models,” *IEEE Transactions on Image Processing*, vol. 25, no. 10, pp. 4555–4564, Oct. 2016, doi: 10.1109/TIP.2016.2592701.

[77] Y. Zhang, C. Wang, X. Wang, W. Zeng, and W. Liu, “FairMOT: On the Fairness of Detection and Re-identification in Multiple Object Tracking,” *Int J Comput Vis*, vol. 129, no. 11, pp. 3069–3087, Nov. 2021, doi: 10.1007/S11263-021-01513-4/FIGURES/5.

[78] Z. Wang, L. Zheng, Y. Liu, Y. Li, and S. Wang, “Towards Real-Time Multi-Object Tracking,” *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 12356 LNCS, pp. 107–122, Sep. 2019, doi: 10.1007/978-3-030-58621-8_7.

[79] K. Bernardin, A. Elbs, and R. Stiefelhagen, “Multiple Object Tracking Performance Metrics and Evaluation in a Smart Room Environment”.

[80] P. Dendorfer *et al.*, “MOTChallenge: A Benchmark for Single-Camera Multiple Target Tracking,” *Int J Comput Vis*, vol. 129, no. 4, pp. 845–881, Apr. 2021, doi: 10.1007/S11263-020-01393-0/FIGURES/13.

[81] C. Kim, F. Li, and J. M. Rehg, “Multi-object Tracking with Neural Gating Using Bilinear LSTM.” pp. 200–215, 2018.

[82] N. Wojke, A. Bewley, and D. Paulus, “Simple Online and Realtime Tracking with a Deep Association Metric,” *Proceedings - International Conference on Image Processing, ICIP*, vol. 2017-September, pp. 3645–3649, Mar. 2017, doi: 10.1109/ICIP.2017.8296962.

[83] C. Labit-Bonis, J. Thomas, and F. Lerasle, “Visual and automatic bus passenger counting based on a deep tracking-by-detection system,” Oct. 2021.

[84] M. Broström, “Real-time multi-camera multi-object tracker using YOLOv5 and StrongSORT with OSNet.” 2022. Accessed: Sep. 13, 2022. [Online]. Available: https://github.com/mikel-brostrom/Yolov5_StrongSORT_OSNet

[85] Y. Du, Y. Song, B. Yang, and Y. Zhao, “StrongSORT: Make DeepSORT Great Again,” Feb. 2022, doi: 10.48550/arxiv.2202.13514.

[86] H. Luo *et al.*, “A Strong Baseline and Batch Normalization Neck for Deep Person Re-identification,” *IEEE Trans Multimedia*, vol. 22, no. 10, pp. 2597–2609, Jun. 2019, doi: 10.1109/tmm.2019.2958756.

[87] J. Gao and R. Nevatia, "Revisiting Temporal Modeling for Video-based Person ReID," May 2018, doi: 10.48550/arxiv.1805.02104.

[88] G. Evangelidis and E. Psarakis, "Parametric Image Alignment Using Enhanced Correlation Coefficient Maximization," *IEEE Trans Pattern Anal Mach Intell*, vol. 30, no. 10, pp. 1858–1865, 2008, Accessed: Sep. 14, 2022. [Online]. Available: https://hal.inria.fr/hal-00864385

[89] Y. Du, J. Wan, Y. Zhao, B. Zhang, Z. Tong, and J. Dong, "GIAOTracker: A comprehensive framework for MCMOT with global information and optimizing strategies in VisDrone 2021," Feb. 2022, doi: 10.48550/arxiv.2202.11983.

[90] K. Zhou, Y. Yang, A. Cavallaro, and T. Xiang, "Omni-Scale Feature Learning for Person Re-Identification," *Proceedings of the IEEE International Conference on Computer Vision*, vol. 2019-October, pp. 3701–3711, May 2019, doi: 10.48550/arxiv.1905.00953.

[91] S. Chen *et al.*, "2nd Place Solution for Waymo Open Dataset Challenge-2D Object Detection".

[92] K. Zhou and T. Xiang, "Torchreid: A Library for Deep Learning Person Re-Identification in Pytorch," Oct. 2019, doi: 10.48550/arxiv.1910.10093.

[93] L. Zheng, L. Shen, L. Tian, S. Wang, J. Wang, and Q. Tian, "Scalable Person Re-identification: A Benchmark," *Proceedings of the IEEE International Conference on Computer Vision*, vol. 2015 International ..., pp. 1116–1124, Feb. 2015, Accessed: Sep. 16, 2022. [Online]. Available: http://www.liangzheng.com.cn.

[94] E. Ristani, F. Solera, R. Zou, R. Cucchiara, and C. Tomasi, "Performance Measures and a Data Set for Multi-Target, Multi-Camera Tracking," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9914 LNCS, pp. 17–35, Sep. 2016, doi: 10.48550/arxiv.1609.01775.

[95] L. Wei, S. Zhang, W. Gao, and Q. Tian, "Person Transfer GAN to Bridge Domain Gap for Person Re-Identification," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 79–88, Nov. 2017, doi: 10.48550/arxiv.1711.08565.

[96] Z. Zheng, L. Zheng, and Y. Yang, "Unlabeled Samples Generated by GAN Improve the Person Re-identification Baseline in vitro," *Proceedings of the IEEE International Conference on Computer Vision*, vol. 2017-October, pp. 3774–3782, Jan. 2017, doi: 10.48550/arxiv.1701.07717.

[97] A. Milan, L. Leal-Taixé, T. Taixé, I. Reid, S. Roth, and K. Schindler, "MOT16: A Benchmark for Multi-Object Tracking," Mar. 2016, doi: 10.48550/arxiv.1603.00831.

[98] K. Zhou, Y. Yang, A. Cavallaro, and T. Xiang, “Learning Generalisable Omni-Scale Representations for Person Re-Identification,” *IEEE Trans Pattern Anal Mach Intell*, vol. 44, no. 9, pp. 5056–5069, Oct. 2019, doi: 10.48550/arxiv.1910.06827.

[99] S. Shao *et al.*, “CrowdHuman: A Benchmark for Detecting Human in a Crowd,” Apr. 2018, doi: 10.48550/arxiv.1805.00123.

[100] A. Noferi, G. Nardini, G. Stea, and A. Virdis, “Deployment and configuration of MEC apps with Simu5G,” Sep. 2021, doi: 10.48550/arxiv.2109.12048.

[101] G. Nardini, D. Sabella, G. Stea, P. Thakkar, and A. Virdis, “SiMu5G–An OMNeT++ library for end-to-end performance evaluation of 5G networks,” *IEEE Access*, vol. 8, pp. 181176–181191, 2020, doi: 10.1109/ACCESS.2020.3028550.

[102] C. Sommer, R. German, and F. Dressler, “Bidirectionally coupled network and road simulation for improved IVC analysis,” *IEEE Trans Mob Comput*, vol. 10, no. 1, pp. 3–15, Jan. 2011, doi: 10.1109/TMC.2010.133.

[103] P. A. Lopez *et al.*, “Microscopic Traffic Simulation using SUMO,” *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*, vol. 2018-November, pp. 2575–2582, Dec. 2018, doi: 10.1109/ITSC.2018.8569938.

[104] A. Varga, “OMNeT++,” *Modeling and Tools for Network Simulation*, pp. 35–59, 2010, doi: 10.1007/978-3-642-12331-3_3/COVER.

[105] “INET Framework - INET Framework.” Accessed: Oct. 06, 2022. [Online]. Available: https://inet.omnetpp.org/

[106] A. Noferi, G. Nardini, G. Stea, and A. Virdis, “Rapid prototyping and performance evaluation of MEC-based applications,” Mar. 2022, doi: 10.48550/arxiv.2203.13511.

[107] G. Nardini, G. Stea, A. Virdis, D. Sabella, and P. Thakkar, “Using Simu5G as a realtime network emulator to test MEC apps in an end-to-end 5G testbed,” *IEEE International Symposium on Personal, Indoor and Mobile Radio Communications, PIMRC*, vol. 2020-August, Aug. 2020, doi: 10.1109/PIMRC48278.2020.9217177.

[108] L. Liu, C. Chen, Q. Pei, S. Maharjan, and Y. Zhang, “Vehicular Edge Computing and Networking: A Survey,” *Mobile Networks and Applications 2020 26:3*, vol. 26, no. 3, pp. 1145–1168, Jul. 2020, doi: 10.1007/S11036-020-01624-1.