



Universidade do Minho
Escola de Engenharia

Rafael Meleiro Marques

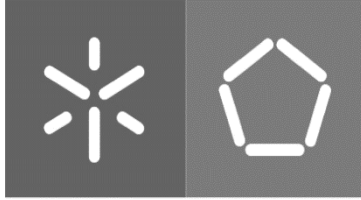
**Analysis of Traffic Signs and Traffic Lights
for Autonomously Driven Vehicles**



Universidade do Minho
Escola de Engenharia

Rafael Meleiro Marques

**Analysis of Traffic Signs and Traffic Lights
for Autonomously Driven Vehicles**



Universidade do Minho
Escola de Engenharia

Rafael Meleiro Marques

Analysis of Traffic Signs and Traffic Lights for Autonomously Driven Vehicles

Dissertação de Mestrado
Mestrado Integrado em Engenharia
Eletrónica Industrial e Computadores

Trabalho efetuado sob a orientação do
Professor Doutor Fernando Ribeiro

DIREITOS DE AUTOR E CONDIÇÕES DE UTILIZAÇÃO DO TRABALHO POR TERCEIROS

Este é um trabalho académico que pode ser utilizado por terceiros desde que respeitadas as regras e boas práticas internacionalmente aceites, no que concerne aos direitos de autor e direitos conexos.

Assim, o presente trabalho pode ser utilizado nos termos previstos na licença abaixo indicada.

Caso o utilizador necessite de permissão para poder fazer um uso do trabalho em condições não previstas no licenciamento indicado, deverá contactar o autor, através do RepositóriUM da Universidade do Minho.

Licença concedida aos utilizadores deste trabalho



Atribuição-Compartilha Igual
CC BY-SA

<https://creativecommons.org/licenses/by-sa/4.0/>

Acknowledgements

Throughout my academic journey, several people contributed with essential support that culminated in the completion of this dissertation.

A very special thanks to my parents Manuel Marques and Maria Marques for all the constant support, love and belief in my capabilities. Without their constant support, this journey wouldn't be achievable. They encouraged me throughout my worst moments and were always there when I needed them.

A big thanks to my supervisor, Professor Fernando Ribeiro, for the excellent opportunity, the constant guidance and the availability demonstrated through this dissertation. To Tiago Ribeiro who guided me in all the stages of this dissertation in the correct direction. To the *Laboratório de Automação e Robótica* members for the great work environment and the shared knowledge.

To my close friends André Gonçalves, Bruno Sousa and Rafael Araújo, whom I was able to share unforgettable memories throughout this journey and in their own way supported and encouraged me to reach this goal. Also to my classmates, who helped me all these years.

A special thanks to Mariana Sarmiento for the constant support and continuous encouragement that were essential for the development of this dissertation.

I am thankful for those who directly or indirectly had contributed to this dissertation.

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.

Resumo

Os *Advanced Driver Assistance Systems* (ADAS) estão relacionados a vários sistemas nos veículos que se destinam a melhorar a segurança do tráfego rodoviário, ajudando os condutores a terem melhor consciência da estrada e dos seus perigos inerentes, bem como de outros motoristas nas proximidades. A deteção e o reconhecimento de sinais de trânsito são parte integrante do ADAS. Os sinais de trânsito fornecem informações sobre as regras de trânsito, condições das estradas, direções de rotas e auxiliam os motoristas para uma condução segura. Isto garante que o limite de velocidade atual e outros sinais de trânsito sejam exibidos para o motorista continuamente. O projeto de reconhecimento de sinais de trânsito tem sido um problema desafiador por muitos anos e, portanto, tornou-se um tópico de pesquisa importante e ativo na área de sistemas de transporte inteligentes.

Esta tecnologia está a ser desenvolvida por uma variedade de fornecedores automóveis. Esta pode usar técnicas de processamento de imagem para detetar sinais de trânsito.

Uma abordagem do problema de deteção e reconhecimento de sinais/semáforos usando *Supervised Learning* é apresentada nesta dissertação para dois cenários diferentes. Nesta dissertação, para cada objetivo, são apresentadas duas abordagens diferentes de *Supervised Learning*, bem como um estudo estendido dos hiperparâmetros. Para os dois primeiros objetivos, foram desenvolvidas as abordagens para um robô produzido pela equipa de Condução Autónoma do Laboratório de Automação e Robótica. O robô deve detetar e classificar corretamente o sinal/semáforo de trânsito apresentado. O terceiro objetivo é para a via pública e a abordagem desenvolvida deve detetar e classificar corretamente o sinal/semáforo de trânsito apresentado no conjunto de dados restritos.

Palavras-chave: *Supervised Learning*, *RoboCup*, *YOLOV3*, Deteção de sinais de trânsito, Robô móvel autónomo, Robótica, Robô simulado.

Abstract

Advanced Driver Assistance Systems (ADAS) relate to various in-vehicle systems that are intended to improve road traffic safety by supporting and improve drivers awareness of the road and its dangers as well as other drivers in the vicinity. Traffic sign detection and recognition is part of ADAS. Traffic signs give knowledge about the traffic rules, road conditions, route directions and assist drivers for safe driving. This ensures that the current speed limit and other road signs are displayed to the driver on an ongoing basis. The design of traffic sign recognition has been a challenging problem for many years and therefore became an important and active research topic in the area of intelligent transport systems.

This technology is being developed by a variety of automotive suppliers. Typically it uses classical image processing techniques to detect traffic signs.

An approach to the problem of Traffic Sign/Light detection and recognition using Supervised Learning is presented in this dissertation for two different scenarios. Two different Supervised Learning approaches are presented for each objective as well as an extended hyperparameter study. For the first two objectives, the approaches were developed for a robot produced by the Autonomous Driving team from the *Laboratório de Automação e Robótica* from University of Minho. The robot must correctly detect and classify the presented Traffic Sign/Light. The third objective is for the public road and the developed approach must correctly detect and classify the presented Traffic Sign/Light in the restrained dataset.

Keywords: Supervised Learning, Traffic Sign Detection, Autonomous Mobile Robot, Robotics, Simulated Robot, RoboCup, YOLOV3

Table of Contents

Resumo	vi
Abstract	vii
Table of Contents	viii
List of Figures	xi
List of Tables	xvii
List of Listings	xviii
Acronyms	xviii
1 Introduction	1
1.1 Motivation	2
1.2 Objectives	3
1.3 Dissertation Structure	4
2 Literature Review	5
2.1 Artificial Intelligence	6
2.2 Machine Learning	6
2.3 Deep Learning	7
2.4 Convolutional Neural Networks	10
2.4.1 Convolutional Layer	11
2.4.2 Pooling Layer	13
2.4.3 Fully-connected Layer	14
2.5 Theoretical foundations	14
2.5.1 Supervised and unsupervised learning	14

2.5.2	Classification and regression	15
2.5.3	Intersection over Union	15
2.5.4	Average Precision	16
2.5.5	Convolutional neural network architectures	17
2.6	Context	18
2.6.1	Tesla	19
2.6.2	General Motors	20
2.7	RoboCup Portuguese Open	21
2.7.1	Autonomous Driving Competition	21
2.7.2	Indicating Signals	22
2.7.3	Vertical Traffic Signs	23
2.7.4	Vertical Traffic Signs Detection Challenge	24
2.8	State of the Art	24
2.8.1	You Only Look Once (YOLO)	24
2.8.2	YOLO9000: Better, Faster, Stronger	27
2.8.3	YOLOv3: An Incremental Improvement	30
2.8.4	Traffic Sign Recognition for Autonomous Driving Robot	32
3	System Specification	37
3.1	Model Framework	37
3.2	Autonomous Driving Competition of the RoboCup Portuguese Open in simulation	39
3.2.1	Acquisition	49
3.2.2	Training	52
3.2.3	Testing	60
3.3	Autonomous Driving Competition of the RoboCup Portuguese Open	64
3.3.1	Acquisition	67
3.3.2	Training	74
3.3.3	Testing	74
3.4	Public road	75
3.4.1	Acquisition	78
3.4.2	Train Labelling Network	82

3.4.3	Acquisition Using The Labelling Network	82
3.4.4	Train Final Network	83
3.4.5	Testing	84
4	Tests	85
4.1	Frame Rate	85
4.2	Hyperparameters	86
4.2.1	YoloV3	87
4.2.2	YoloV3_tiny	97
4.2.3	Conclusion	105
5	Analysis of Results	107
5.1	Final networks	107
5.1.1	Autonomous Driving Competition of the RoboCup Portuguese Open in simulation	108
5.1.2	Autonomous Driving Competition of the RoboCup Portuguese Open .	110
5.1.3	Public Road	112
5.2	Comparison between networks per objective	114
5.2.1	Autonomous Driving Competition of the RoboCup Portuguese Open in simulation	114
5.2.2	Autonomous Driving Competition of the RoboCup Portuguese Open .	118
5.2.3	Public Road	121
5.3	Results Discussion	125
6	Conclusions and Future Work	127
6.1	Future Work	128
Appendix A	Videos for each objective	130
A.1	Autonomous Driving Competition of the RoboCup Portuguese Open in simulation	130
A.2	Autonomous Driving Competition of the RoboCup Portuguese Open	130
A.3	Public road	130
References		136

List of Figures

1.1	Autonomous Driving Test Track (left) and An example of autonomous driving vehicles developed by <i>Laboratório de Automação e Robótica</i> (right)	3
2.1	Artificial intelligence, Machine Learning, and Deep Learning	5
2.2	Difference between Machine Learning and Classical programming	7
2.3	Human Brain (figure from [1] displayed with permission from Bard Ermentrout and David Terman)	8
2.4	Artificial Neuron (figure from [2] displayed with permission from Çagri Kaymak)	8
2.5	Difference between Simple Neural Network (left) and Deep Neural Network (right)	8
2.6	Neuron in Neural Network	9
2.7	Simple CNN architecture (figure from [3] displayed with permission from Keiron O'Shea)	11
2.8	Application of a kernel in a matrix (figure from [4] displayed with permission from Chris Thomas)	11
2.9	Example Input to the convolution layer and Kernel (figure from [5] displayed with permission from Arden Dertat)	12
2.10	Convolution operation (figure from [5] displayed with permission from Arden Dertat)	12
2.11	Convolution operation (figure from [5] displayed with permission from Arden Dertat)	13
2.12	MaxPooling example (figure from [6] displayed with permission from Shadman Sakib)	14
2.13	Fully-Connected Layers (figure from [6] displayed with permission from Shadman Sakib)	14
2.14	Area of Overlap and Area of Union	16
2.15	LesNet architecture (figure from [7] displayed with permission from Yann LeCun)	17

2.16	Residual learning: a building block	18
2.17	Track overview (figure from [8] displayed with permission from Manuel Silva)	21
2.18	Signaling panels (figure from [8] displayed with permission from Manuel Silva)	22
2.19	Signaling panel drawings (figure from [8] displayed with permission from Manuel Silva)	22
2.20	Traffic sign placed on the side of the track (figure from [8] displayed with permission from Manuel Silva)	23
2.21	The twelve traffic signs for the competition (figure from [8] displayed with permission from Manuel Silva)	24
2.22	The YOLO model (figure from [9] displayed with permission from Ali Farhadi)	25
2.23	The architecture (figure from [9] displayed with permission from Ali Farhadi)	26
2.24	Combining datasets using WordTree hierarchy (figure from [10] displayed with permission from Ali Farhadi)	29
2.25	Bounding boxes with dimension priors and location prediction (figure from [10] displayed with permission from Ali Farhadi)	31
2.26	Structure of Tiny YOLOv3	32
2.27	Examples of signs to recognize (figure from [11] displayed with permission from Vitor Filipe)	33
2.28	a) Frame acquired with a visible traffic sign. b) Regions detected after thresholding. c) Detected external contour (figure from [11] displayed with permission from Vitor Filipe)	34
2.29	a) Sub-image (grayscale) with the sign extracted from the RGB frame. b) Binary image after threshold operation. c) Result of pictogram extraction (figure from [11] displayed with permission from Vitor Filipe)	34
2.30	a) External contour obtained in the detection stage. b) Internal symbol obtained in pictogram extraction stage. c) Binary pattern obtained after logical OR operation (figure from [11] displayed with permission from Vitor Filipe)	35
3.1	Model Framework	38
3.2	Image from one of the Datasets and its correspondent text file with the bounding box coordinates, dimensions and label	38
3.3	Example of the testing phase	39

3.4	The six types of Traffic Signs built in Solidworks	42
3.5	The six types of Traffic Signs in CoppeliaSim	43
3.6	All Traffic Signs in CoppeliaSim	43
3.7	Example of a Traffic Light in CoppeliaSim	44
3.8	Comparison between the real robot (left) and its 3D model (right)	45
3.9	The joints that allow the movement of the car	45
3.10	Camera in the car	46
3.11	ROS Master	47
3.12	Graphical representation of how both of the ROS nodes communicate	47
3.13	Subscriber creation in CoppeliaSim	48
3.14	Publisher creation in CoppeliaSim	48
3.15	Comparison between the YOLOv3 (left) and YOLOv3_tiny (right) structure	56
3.16	Images from two different frames from the test	62
3.17	Start of the test	63
3.18	Speed detection	63
3.19	End of the test after the STOP detection	64
3.20	Constructed Signs	65
3.21	Constructed Track	65
3.22	Graph with the distance from the T S/L to the camera	66
3.23	First Example	67
3.24	Second Example	68
3.25	Third Example	68
3.26	Fourth Example	69
3.27	Template	71
3.28	Input	71
3.29	All the detections with confidence scores over 40%	72
3.30	Detection with the highest confidence score	73
3.31	Information returned using the Template Matching function	73
3.32	Steps taken to accomplish this objective	75
3.33	Zone recorded	79
3.34	Rainy day image	81

3.35	Bright day image	81
3.36	Night image	81
3.37	Example of Acquisition using The Labelling Network	83
4.1	YOLOV3 performance with different input dimensions	85
4.2	YOLO_tiny performance with different input dimensions	86
4.3	Influence of max_batches on mAP	87
4.4	Influence of max_batches on Loss	88
4.5	Influence of learning_rate on mAP	89
4.6	Influence of learning_rate on Loss	90
4.7	Influence of momentum on mAP	91
4.8	Influence of momentum on Loss	91
4.9	Influence of Burn_in on mAP	92
4.10	Influence of Burn_in on Loss	93
4.11	Influence of decay on mAP	94
4.12	Influence of decay on Loss	95
4.13	Influence of width and height on mAP	96
4.14	Influence of width and height on Loss	96
4.15	Influence of max_batches on mAP	98
4.16	Influence of max_batches on Loss	98
4.17	Influence of learning_rate on mAP	99
4.18	Influence of learning_rate on Loss	100
4.19	Influence of momentum on mAP	101
4.20	Influence of momentum on Loss	101
4.21	Influence of Burn_in on mAP	102
4.22	Influence of Burn_in on Loss	102
4.23	Influence of decay on mAP	103
4.24	Influence of decay on Loss	104
4.25	Influence of width and height on mAP	104
4.26	Influence of width and height on Loss	105

5.1	YOLOV3 mAP for the Autonomous Driving Competition of the RoboCup Portuguese Open in simulation	108
5.2	YOLOV3 loss for the Autonomous Driving Competition of the RoboCup Portuguese Open in simulation	108
5.3	YOLOV3_tiny mAP for the Autonomous Driving Competition of the RoboCup Portuguese Open in simulation	109
5.4	YOLOV3_tiny loss for the Autonomous Driving Competition of the RoboCup Portuguese Open in simulation	109
5.5	YOLOV3 mAP for the Autonomous Driving Competition of the RoboCup Portuguese Open	110
5.6	YOLOV3 loss for the Autonomous Driving Competition of the RoboCup Portuguese Open	110
5.7	YOLOV3_tiny mAP for the Autonomous Driving Competition of the RoboCup Portuguese Open	111
5.8	YOLOV3_tiny loss for the Autonomous Driving Competition of the RoboCup Portuguese Open in simulation	111
5.9	YOLOV3 mAP for the Public Road	112
5.10	YOLOV3 loss for the Public Road	112
5.11	YOLOV3_tiny mAP for the Public Road	113
5.12	YOLOV3_tiny loss for the Public Road	113
5.13	Two frames with high confidence detection from the YOLOV3 network	115
5.14	Two frames with high confidence detection from the YOLOV3_tiny network	115
5.15	Frame where the robot reacted to the Left Obligation sign	116
5.16	Frame where the robot reacted to the Left Obligation sign	117
5.17	Frame where the robot reacted to the 30km/h sign	117
5.18	Frame where the robot reacted to the STOP sign	118
5.19	Detection from the YOLOV3 (left) and YOLOV3_tiny (right)	119
5.20	Detection distance from the YOLOV3 (left) and YOLOV3_tiny (right)	119
5.21	Bounding Boxes from the YOLOV3 (left) and YOLOV3_tiny (right)	120
5.22	Detection with less stability from the YOLOV3 (left) and YOLOV3_tiny (right)	120
5.23	Detection with extreme distortion from the YOLOV3 (left) and YOLOV3_tiny (right)	121

5.24	Detection with ideal conditions from the YOLOV3 (left) and YOLOV3_tiny (right)	122
5.25	Detection with rain from the YOLOV3 (left) and YOLOV3_tiny (right)	122
5.26	Detection of a sign not facing the camera from the YOLOV3 (left) and YOLOV3_tiny (right)	123
5.27	Traffic Light detection from the YOLOV3 (left) and YOLOV3_tiny (right)	123
5.28	Detection of an unknown sign from the YOLOV3 (left) and YOLOV3_tiny (right)	124
5.29	Results of the YOLOV3 network in poor conditions	124
5.30	Results of the YOLOV3_tiny network in poor conditions	125

List of Tables

2.1	Activation Function	10
2.2	Correspondence between each of the functions and the information (table adapted from [8] with permission from Manuel Silva)	23
2.3	Performace of Real-Time Detectors	26
2.4	The path from YOLO to YOLOv2 (table adapted from [12] with permission from Ali Farhadi)	28
2.5	Darknet-19 (table adapted from [12] with permission from Ali Farhadi)	29
2.6	Darknet-53 (figure from [10] displayed with permission from Ali Farhadi)	30
3.1	The Traffic Signs/Lights in the Dataset	41
3.2	The Traffic Signs/Lights in the Dataset	42
3.3	Number of images per Traffic Sign/Light	50
3.4	Number of images per Traffic Sign/Light	70
3.5	The Traffic Signs/Lights in the Road Dataset	78
3.6	Number of images per Traffic Sign/Light	80
4.1	The hyperparameters and the corresponding tested values	87
4.2	Time each train lasted	88
4.3	Time each train lasted	93
4.4	Time each train lasted	97
4.5	The hyperparameters and the corresponding tested values	97
4.6	Time each train lasted	99
4.7	Time each train lasted	103
4.8	Time each train lasted	105
4.9	Optimized hyperparameters for the YOLOV3 network	106
4.10	Optimized hyperparameters for the YOLOV3_tiny network	106

Acronyms

ADAS - Advanced Driver Assistance Systems

AI - Artificial Intelligence

API - Application Programming Interface

ANN - Artificial Neural Networks

CNN - Convolutional Neural Network

CUDA - Compute Unified Device Architecture

DNN - Deep Neural Networks

FP - False Positives

FN - False Negatives

FPS - Frames Per Second

GPU - Graphics Processing Unit

IDE - Integrated Development Environment

IoU - Intersection over Union

mAP - Mean Average Precision

NN - Neural Network

OpenCV - Open Source Computer Vision Library

RGB - Red Green Blue

ROI - Region Of Interest

ROS - Robot Operating System

TP - True Positives

TSR - Traffic Sign Recognition

TS/L - Traffic Signs/Lights

URDF - Unified Robot Description Format

YOLO - You Only Look Once

Chapter 1

Introduction

The world is under continuous development and with this, vehicles have become an essential means of transportation for people every day. This advancement in society has produced many traffic safety predicaments, like traffic congestion and regular road accidents, which are mainly caused by subjective reasons related to the driver. These are mostly a result of negligence, inappropriate driving manoeuvre and disregarding traffic signs [13]. The previously mentioned human factors can be evaded using self-driving technology which can support or control the driving operation. This can reduce the occurrence of accidents [14]. Traffic sign detection and recognition is essential in the evolution of intelligent vehicles.

This dissertation presents the development of traffic sign detection and recognition using Machine Learning (ML).

ML provides systems with the ability to automatically learn and improve from experience without being explicitly programmed. It examines the development and research of algorithms that can learn from and make predictions on data. These algorithms operate by building a model from inputs in order to make data-driven predictions rather than following strict guidelines [15].

ML is used to provide knowledge and understanding of the environment around the vehicle. This mainly involves the use of camera-based systems to detect and classify objects but can also be based on LiDAR or radar. The main concern around this technology is the possibility of a misclassification of an object that can be caused by just a few pixels of difference in an image produced by a camera system. This can lead to an incorrect action from the car causing a road accident. Using enhanced and more generalized training of the ML models can lead to an improvement in the network's accuracy to correctly detect the object. Giving the system more diverse inputs on key parameters also improves performance [16].

1.1 Motivation

According to [17], the future of the automotive industry consists of connected and autonomous vehicles which will outnumber vehicles controlled by humans. This technological progress will lead to advancements in Advanced Driver Assistance Systems (ADAS). Adjacent to ADAS is Traffic Sign Recognition (TSR), a system that discovers and classifies traffic signs captured by a camera positioned in the vehicle. With this system, self-driving cars have a better perception of the road and its surroundings. Unlike most systems that are contained in ADAS, TSR only requires a camera and the corresponding ML software. The software associated with TSR demands cutting edge hardware to perform properly. One option is NVIDIA's General Purpose Graphics Processing Unit (GPGPU), the DRIVE PX which is an Artificial Intelligence (AI) car computer that empowers vehicle manufacturers to stimulate the production of automated and autonomous vehicles [18]. It can facilitate real-time heavy machine learning software to run in the vehicle, supporting more accurate TSR systems.

TSR systems face two major technical challenges [19]:

- The broad disparity in illumination: two elements can contribute to this disparity, the time of day in which the incidence of sunlight will change and be reduced by nighttime, and the weather in which it will fluctuate depending on the presented conditions;
- Low image resolution when the vehicle is at high speeds: the frames light from the camera and the other vehicles captures at high speeds are more blurred and the resolution affected.

To overcome these challenges pre-processing methods are proposed.

The TSR system is usually divided into two phases, the first one has the goal of discovering the Region of Interest (ROI) where the traffic sign is located in the input frame, the second one has the aim of classifying the sign in the ROI. The first phase of the TSR system can consist of colour or shape detection but have limited accuracy and can be affected by illumination. Another option is using Haar-like features [20]. For the second phase methods Convolutional Neural Network (CNN) can be used.

1.2 Objectives

For autonomous driving vehicles, like the ones developed by *Laboratório de Automação e Robótica* to be able to move in a highly dynamic and regulated environment such as vehicle traffic, it is necessary to develop a strategy that complies with the Traffic Signaling Regulation. For this, it is necessary to develop a identification and classification system for traffic signs (for vertical and light signs). The developed algorithm will be based on Supervised Learning and Deep Learning (DL) strategies, where it has to search in non-standardized images, the occurrence of one or several signals and their respective classification. In the first instance, the algorithm will be tested in two environments:

1. the RoboCup Portuguese Open – Autonomous Driving competition;
2. an autonomous driving environment on real road and vehicle traffic.

The first environment refers to the Autonomous Driving test of the RoboCup Portuguese Open, represented in figure 1.1. This event is characterized by creating a reduced and more controlled version of a two-lane road, crosswalks, obstacles, traffic lights, traffic signals, tunnels, parking spaces and construction sites with temporary signage. The robot has to be completely autonomous and respect all traffic rules in a more controlled version of public roads. In this track, the vertical traffic signs are the same as those regulated in the highway code, while the traffic light has some variations, as it also indicates the direction that the vehicle has to go. Before testing in the event a simulation must be developed that reproduces this competition. Participation in this event will serve as a benchmark of the developed algorithm as well as its application in a real environment.



Figure 1.1: Autonomous Driving Test Track (left) and An example of autonomous driving vehicles developed by *Laboratório de Automação e Robótica* (right)

The second test environment consists of driving a real vehicle on a public road, with cameras mounted on the vehicle running the algorithm in real-time identifying and classifying vertical and light signals.

1.3 Dissertation Structure

To better understand and describe the essential steps and features to the progression of the project, this dissertation is divided into five chapters. The first one is the Introduction chapter, where the motivation and objectives are described, followed by the Literature Review, where a study of Machine Learning was performed to understand its specificities and a state of the art is present and analysed, in which the used architecture is deepened. The third chapter, System Specification is the fundamental chapter of this dissertation since all the used methodologies to solve the objectives are explained. In this chapter all the used software, environments and procedures are disclosed. The following chapter, Tests, firstly presents a study to uncover the most optimized hyperparameters for the chosen networks. The next chapter, Results, describes the results for each network of each objective. Finally, in the Conclusions chapter, the core outcomes are mentioned as well as further work.

Chapter 2

Literature Review

This chapter provides essential context about Artificial Intelligence (Section 2.1) and its subfields Machine Learning (Section 2.2) and Deep Learning (Section 2.3), as shown in figure 2.1. The concepts regarding the structure and layers of Convolutional Neural Networks are deepened (Section 2.4) and theoretical concepts will be presented (Section 2.5). A context is presented concerning Traffic Sign Recognition, as well as the levels of driving automation and examples of how recognition is carried out in car companies currently (Section 2.6). One of the objectives of the project is to participate in the RoboCup Portuguese Open Autonomous Driving Competition so its rules and information will be presented in Section 2.7. In Section 2.8, the You Only Look Once (YOLO) model is explained and an implementation of Traffic Sign Recognition for Autonomous Driving Robot is presented.

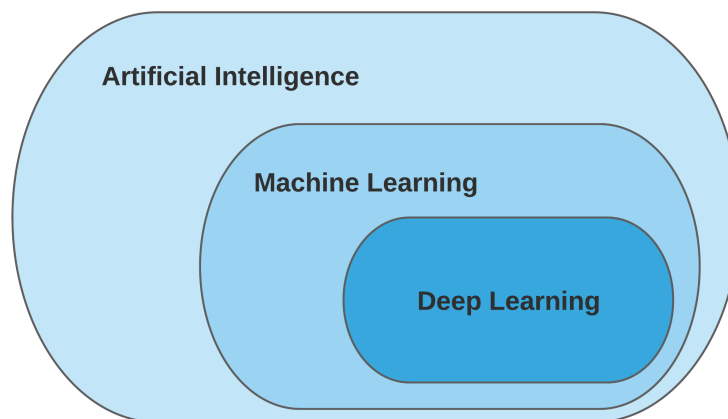


Figure 2.1: Artificial intelligence, Machine Learning, and Deep Learning

2.1 Artificial Intelligence

Artificial Intelligence (AI) is the development of computer systems that are able to perform tasks that would require human intelligence.

Defined by John McCarthy in 1956 as "science engineering of making intelligent machines", AI is a branch in computer science that studies and designs intelligent agents that recognise its context and takes actions which maximize its chances of success [21].

AI should possess the following traits [22]:

- Should be able to predict and adapt: AI uses multiple algorithms from huge amounts of data to discover patterns;
- To make decisions on its own: It can augment human intelligence, deliver insights and enhance productivity;
- It is continuous learning: in order to construct analytical models AI uses algorithms. From countless rounds of trial and error AI finds out how to perform tasks;
- It is forward-looking: AI allows people to reevaluate how data is analyzed and information integrated, and then use these insights to make more reliable judgments;
- It is capable of motion and perception.

2.2 Machine Learning

In 1959, Arthur Samuel defined Machine Learning (ML) as a "Field of study that gives computers the ability to learn without being explicitly programmed" [23]. A concise definition of the field would be the effort to automate intellectual tasks normally performed by humans [24].

ML is a form of AI that enables a system to learn from data rather than through specific programming, it emerged from the study of pattern recognition and computational learning theory. As seen in figure 2.2, this form of AI is trained rather than explicitly programmed.

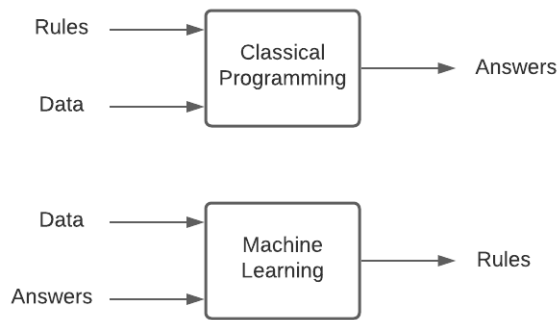


Figure 2.2: Difference between Machine Learning and Classical programming

Using a diversity of algorithms, ML iteratively learns from data and predicts outcomes. The output produced when a ML algorithm is trained with data is the model. In order to make a prediction, the user can provide data to the model and receive the prediction as an output [25].

ML has strong ties to mathematical optimization, which deliver methods, theory and application domains to the field. It is employed in a range of computing tasks where designing and programming explicit algorithms are infeasible [26].

2.3 Deep Learning

Deep learning (DL) is a subfield of ML and a recent way of learning representations from data. This subfield emphasizes learning through successive layers of gradually more significant representations. A representation is a different way of looking or encoding data [24]. These successive layers form, as a whole, what are called neural networks, which are the basic structure of DL [27].

Neural networks were designed to simulate the behaviour of the human brain in problem-solving, either for its structure or for its method of processing information [28]. The human brain is composed of a network of neurons that process information. AI professionals try to replicate the structure and the function of the brain with the intent to produce similar cognitive abilities for AI. It is a simple mathematical model of the brain which is used to process nonlinear relationships between inputs and outputs in parallel, like a human brain does every instance [27]. The comparison can be observed in figures 2.3 and 2.4.

Within the neural networks, there are some variants. Deep Neural Networks (DNN), used in DL, are an evolution of Artificial Neural Networks (ANN), which appeared in their first version in

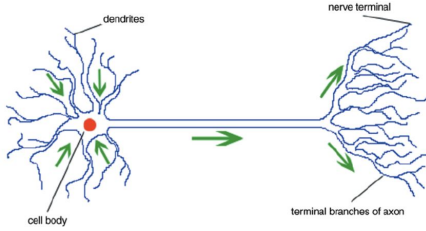


Figure 2.3: Human Brain (figure from [1] displayed with permission from Bard Ermentrout and David Terman)

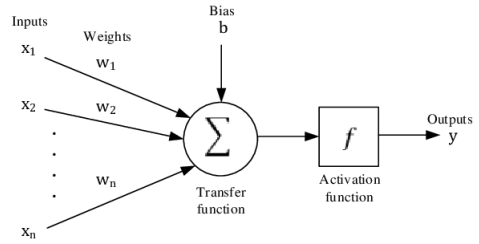


Figure 2.4: Artificial Neuron (figure from [2] displayed with permission from Çağrı Kaymak)

1948 by Donald Hebb [29]. The difference between them is the number of hidden layers in each one. (Figure 2.5)

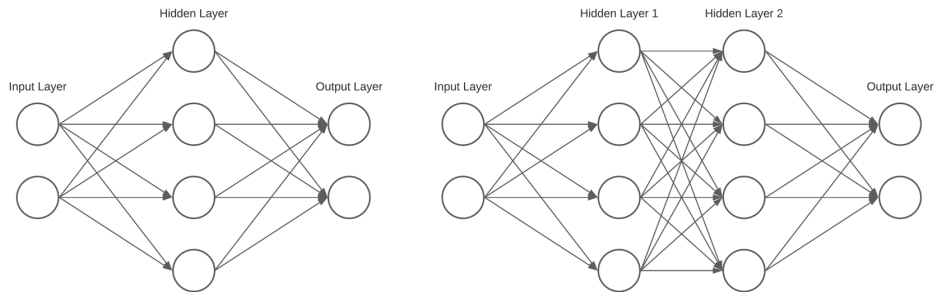


Figure 2.5: Difference between Simple Neural Network (left) and Deep Neural Network (right)

The basic processing unit of a neural network is the neuron. A group of neurons forms what is called a layer and a set of layers forms the neural network [28].

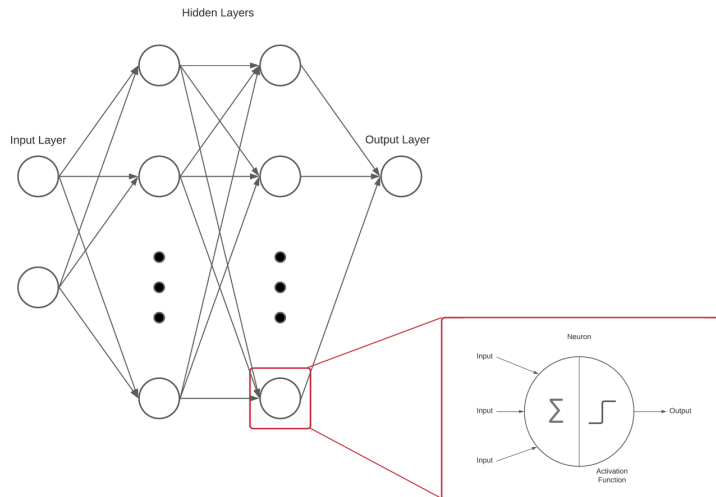


Figure 2.6: Neuron in Neural Network

The sum of the input variables of the neuron is processed by a non-linear function, also called the activation function, which will give rise to the output variable of the neuron [30]. In table 2.1, some known activation functions are represented.

Name	Equation	Plot
Identity	$f(x) = x$	
Logistic	$f(x) = \frac{1}{1+e^{-x}}$	
Binary Step	$\begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	

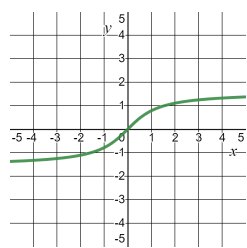
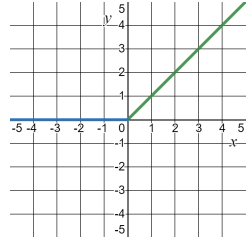
Name	Equation	Plot
ArcTan	$f(x) = \tan^{-1}(x)$	
Rectified Linear Unit (ReLU)	$\begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	

Table 2.1: Activation Function

The constants between different neurons in the adjacent layers are called weights, w , which represent the strength of a connection signal. As the learning progresses, the value of the weights varies, and in this sense, the weights affect the level of influence that a change in an input variable can have on the output of the neuron [30].

2.4 Convolutional Neural Networks

Convolutional Neural Networks (CNN) which are used predominantly for images, assign weights and biases to various objects in the image and distinguishes one from the other [31] and were introduced in 1998 by Yann LeCun [7].

When compared to other classification algorithms CNN require less preprocessing. The CNN is a supervised method and consists of an input and an output layer, as well as multiple hidden layers. In this method, the layers within the CNN are comprised of neurons organised into three dimensions: height, width and depth [3]. These layers are generally divided into three types: convolutional layers, pooling layers and fully-connected layers. In figure 2.7, an example of a simple CNN is represented.

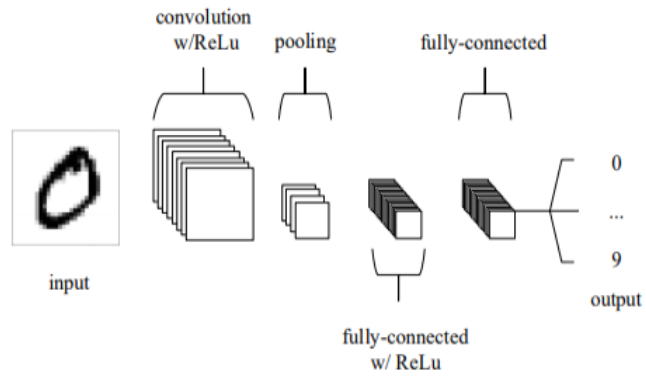


Figure 2.7: Simple CNN architecture (figure from [3] displayed with permission from Keiron O'Shea)

2.4.1 Convolutional Layer

The convolutional layer, as the name implies, is the layer where convolutional operations are carried out in the image fed to the network. To perform the operation, the convolutional layer uses a learnable kernel that runs through the entire image, which is seen as a matrix [4].

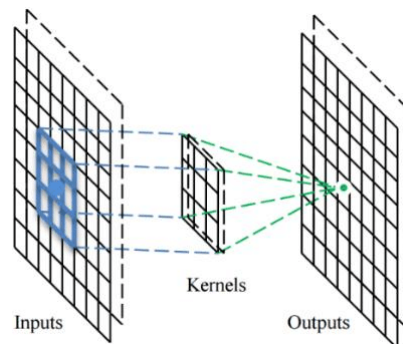


Figure 2.8: Application of a kernel in a matrix (figure from [4] displayed with permission from Chris Thomas)

The kernel is an element composed of a matrix of weights that, as it travels through the matrix, multiplies its values with the values of the matrix. During the training process, the network is able to learn the kernels that react best when they see a specific feature in a given spatial position in the input [4].

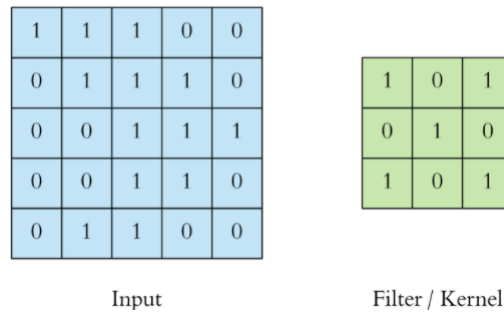


Figure 2.9: Example Input to the convolution layer and Kernel (figure from [5] displayed with permission from Arden Dertat)

To perform the analysis on the entire image, the kernel starts the process in the upper left corner of the matrix and with a certain value of stride, which represents the value of cells that the kernel moves, analyzes the image across the entire width. Then, the kernel moves again to the left side of the matrix and downwards with the same stride value and repeats the same process, as seen in figure 2.8. When the analysis in the whole matrix is finished, it results in a matrix called feature map [6].

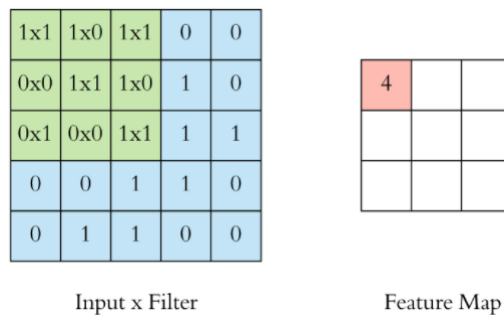


Figure 2.10: Convolution operation (figure from [5] displayed with permission from Arden Dertat)

In examples in which the input is an image composed by Red Green Blue (RGB) colour spectrum, the kernel has the same depth, that is, the same number of channels as the input matrix. Each kernel channel does its multiplication process in its respective input matrix channel, then the results are added to the bias that results in a value that will fill a cell in the output matrix, that is, in the feature map. Thus, at the end of the process, it results in a compressed depth feature map [6].

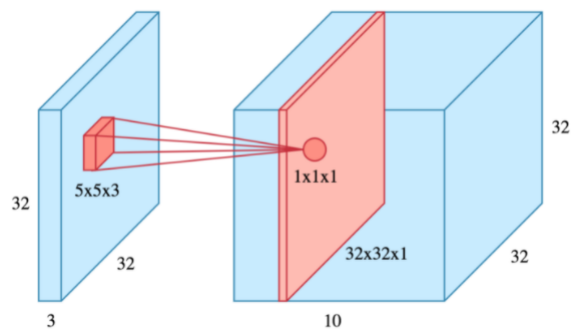


Figure 2.11: Convolution operation (figure from [5] displayed with permission from Arden Dertat)

2.4.2 Pooling Layer

The pooling layer is responsible for reducing the dimensions of the feature maps, thereby reducing computational costs and the complexity of the model. Thus allowing to better control the phenomenon of overfitting [6].

This layer also uses filters, similar to the convolutional layer, but instead of performing the weighted sum, it performs arithmetic functions in all previous activation maps [32].

The most common methods of the pooling layer are maximum pooling (Figure 2.12), which looks for the maximum value of all pixels within the pool, and average pooling, in which the average value is calculated [33].

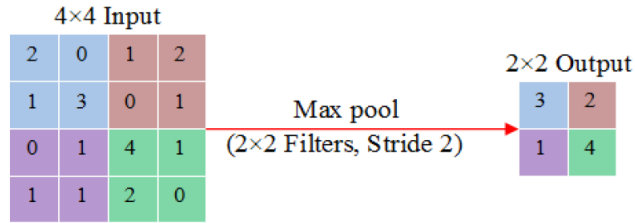


Figure 2.12: MaxPooling example (figure from [6] displayed with permission from Shadman Sakib)

2.4.3 Fully-connected Layer

In a basic CNN model, the characteristics generated by the last convolution layer represent a portion of the input image since its receptive field does not integrate the entire spatial dimension of the image.

Each node in a fully-connected layer is directly connected to every node in both the previous and in the next layer [6] as shown in figure 2.13.

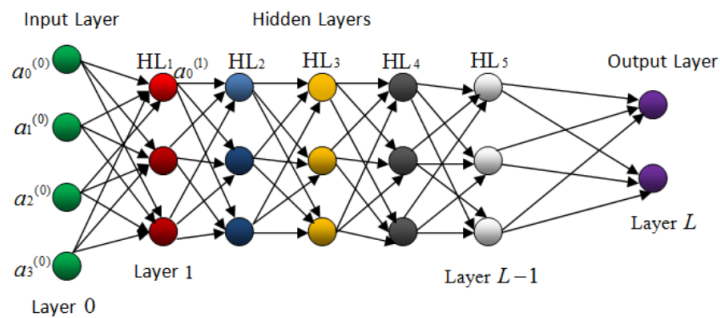


Figure 2.13: Fully-Connected Layers (figure from [6] displayed with permission from Shadman Sakib)

2.5 Theoretical foundations

2.5.1 Supervised and unsupervised learning

ML is a vast field with a complex subfield taxonomy. This algorithms can be supervised and unsupervised learning based on the way the data is fed to.

Supervised learning is the most common type of learning. It is based on training data with correct classification attached to approximate the mapping function [34]. Supervised learning problems can be further grouped into classification or regression problems.

Unlike the previous type of learning, Unsupervised Learning has the purpose to model the underlying structure or distribution in the data in order to learn more about the data. It is applied in problems where it is either impossible or unrealistic for a human to propose patterns in the data [35].

2.5.2 Classification and regression

Two major prediction problems which are usually dealt with in ML are Classification and Regression.

Classification is the process of determining a model that distributes data into various classes. Data is designated under distinct labels conforming to the parameter given as an input. This process deals with problems where the data can be sorted into binary or multiple discrete labels [36].

Regression is the process of determining a model that detects the data into continuous real values and can also recognize the distribution movement depending on the historical data [36].

2.5.3 Intersection over Union

Used to measure the accuracy of an object detector, Intersection over Union (IoU) is an evaluation metric. If an algorithm produces predicted bounding boxes as output, IoU can judge it [37].

Concerning the evaluation using IoU in object detection, two inputs are required:

- The predicted bounding boxes of the model;
- The ground-truth bounding boxes (where the object in the image is specified)

$$IoU = \frac{Area_of_Overlap}{Area_of_Union} \quad (2.1)$$

In equation 2.1, the numerator is the area of overlap connecting both inputs and the denominator is the area of union among both inputs, both represented in Figure 2.14. The IoU is a ratio that symbolizes how much the predicted box is confined by the ground truth [32].

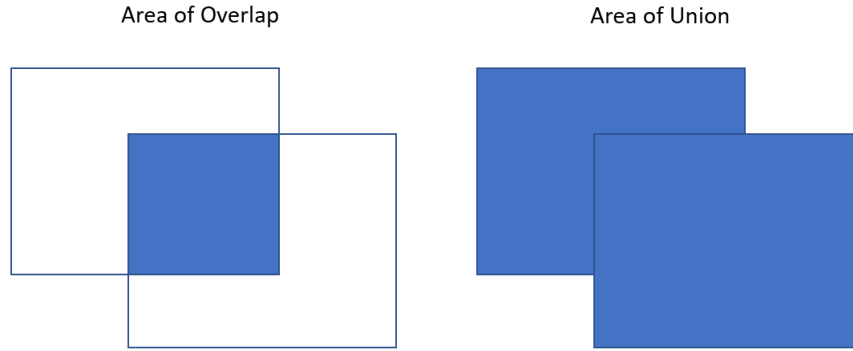


Figure 2.14: Area of Overlap and Area of Union

2.5.4 Average Precision

Mean Average Precision (mAP) is an object detection evaluation metric used mainly in computer vision. It assesses object detection systems, such as YOLO [9] [12] [10] and Faster R-CNN [38]. mAP gives developers a view of the way the model is performing versus other models on the same test dataset or if improvements were made when advancements were implemented.

Precision quantifies how accurate is the prediction and is represented by equation 2.2.

$$Precision = \frac{TP}{TP + FP} \quad (2.2)$$

TP = True Positives (Predicted as positive and was correct)

FP = False Positives (Predicted as positive but was incorrect)

Recall quantifies the correctly detected positives in the input and is represented by equation 2.3.

$$Recall = \frac{TP}{TP + FN} \quad (2.3)$$

FN = False Negatives (Failed to predict an object that was there)

Precision recall curve allows the visualization of the performance of the model when the confidence threshold of the predictions is decreased. If the model is in a situation where avoiding false positives is more important than dodging false negatives, it can set its confidence threshold higher to encourage the model to only deliver high precision predictions at the risk of lowering its amount of coverage (recall)

2.5.5 Convolutional neural network architectures

2.5.5.1 LesNET-5

Developed by Yann LeCun in 1998, the LesNET-5 [7] is one of the most simple architectures. It was significant for the deep learning for image recognition because it was the first to state that image features are distributed across the image and the best way to extract them was with the employment of convolutions [32].

As demonstrated in figure 2.15, it is composed of:

- Input layer: where the image is provided;
- Two convolutional layers: which are the result of the convolution applied at the previous matrix;
- Subsampling: which results on the Pooling Layer;
- Three fully-connected layers: which allow each neuron to receive data from all the neurons in the previous layers;
- Output layer: where the classification is conferred.

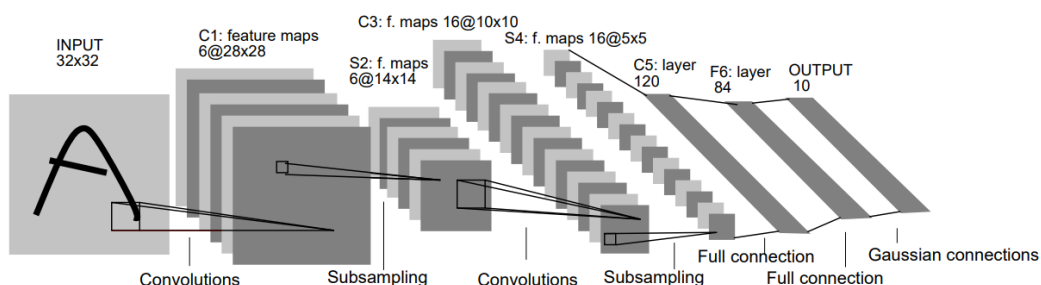


Figure 2.15: LesNet architecture (figure from [7] displayed with permission from Yann LeCun)

2.5.5.2 ResNet

The ResNet consists of a deep net with 152 layers of depth and acquainted the idea of residual learning to deep neural networks.

In the ResNet, each layer of depth is called residual block [39]. (figure 2.16).

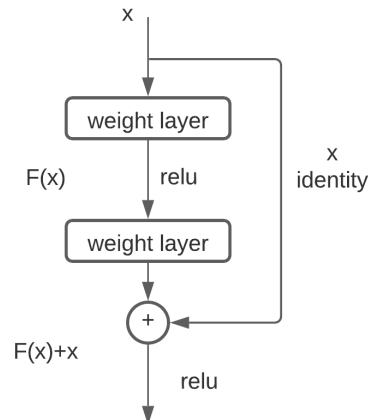


Figure 2.16: Residual learning: a building block

The residual block fundamentally employs convolutional filters to the input x , and the result of those filters is added to the original input. Differentiating it from other architectures, instead of computing the whole transformation of the input, that is, x to $F(x)$, this residual block allows to only compute the variation of the input [39]. This architecture makes backpropagation an easier process [32].

The Resnet architecture won the ImageNet Large Scale Video Recognition Competition (ILSVRC) in 2015. The ILSVRC is an image classification contest, where teams develop neural networks, to classify a set of images in 1000 classes.

2.6 Context

Technological advances offer the opportunity to change transportation with the introduction of autonomous vehicles. These features may fluctuate from supporting the driver like adaptive cruise control or crash warning systems to autonomous driving.

The Society of Automotive Engineers specifies 6 levels of driving automation ranging from 0 (fully manual) to 5 (fully autonomous) in the context of motor vehicles [40]:

- Level 0 - No Driving Automation: The driver provides the dynamic driving tasks. Today, most vehicles on the road are level 0.
- Level 1 - Driver Assistance: To assist the driver the vehicle features a single automated system like adaptive cruise control that keeps the car at a safe distance behind the next car.
- Level 2 - Partial Driving Automation: Systems with Advanced Driver Assistance Systems or ADAS [41] can avoid accidents that are caused by human error. The vehicle can control accelerating/decelerating and steering but the driver can take command of the car at any time. Teslas Autopilot and General Motors Super Cruise are in this level.
- Level 3 – Conditional Driving Automation: The vehicle has environmental detection capabilities and can make knowledgeable decisions but requires the driver to remain alert and ready to take control if the system is incapable to execute the task.
- Level 4 - High Driving Automation: These cars don't require human interaction but the driver can still manually override. They can only operate in limited areas.
- Level 5 - Full Driving Automation: In this level, the cars are fully autonomous. They don't have steering wheels or acceleration/braking pedals and can go anywhere that an experienced human driver can go.

In the following sub chapters 2.6.1 and 2.6.2, examples of how recognition is carried out in car companies currently like Tesla and General Motors.

2.6.1 Tesla

Led by Elon Musk, Tesla is one of the early pioneers and leaders within the self-driving cars market. Every Tesla car made since October 2016 is equipped with the necessary sensor suite for full self-driving, each of these cars also support our autonomous driving development [42]

Tesla's traffic lights and signs detection is possible by the combination of two modules [43]:

- Radars - located in the front bumpers, they can detect cars and objects from a substantial distance and all around;
- Cameras - which have an average wide angled camera, are located in the front or on the roof of the car. They can detect various objects such as cars, cyclists, pedestrians and road markings.

With the combination of these modules, Mobileye, the manufacturer of Tesla's processor, was able to deploy the first Digital Neural Network on the road in Tesla's vehicles. The DNN was trained with cars in the surroundings until it was able to detect them with consistent accuracy and consequently create a 3D model of the same. It is responsible for [43]:

- Free Space Pixel Labeling - recognition of the area on-camera with no obstructions in which the car is allowed to perform;
- Holistic Path Planning - an attribute that tells the car where to drive with little visual hints;
- General Object Detection and Sign Detection - the software can recognize over 250 traffic signs in more than 50 countries which include turn signs, speed limits and traffic lights. It also detects debris and other undesirable situations such as potholes on the road.

2.6.2 General Motors

Traffic Sign Recognition is a General Motors active safety technology that aids the driver.

The software identifies speed limit and many other sorts of signs and displays them on the vehicle's instrument panel. It also detects LED changing road signs. The current system can recognize signs at a distance of 60 meters. The camera mounted used for the Traffic Sign Recognition, the Opel Eye, is mounted in the front of the vehicle [44].

The Opel Eye camera allows the vehicle to detect the road, vehicles, pedestrians and traffic signs. Continually monitoring the scene around the car this technology processes distinct amounts of data providing the driver with warnings and information [44].

2.7 RoboCup Portuguese Open

The RoboCup Portuguese Open promotes Science and Technology among young people, teachers and researchers, through competitions for autonomous robots. Robotic competitions are designed to promote an innovative, entrepreneurial spirit in children and young people through active teaching methods, as well as the acquisition of transversal skills. The 2022 edition is in Santa Maria da Feira from April 27 to May 1 [45].

2.7.1 Autonomous Driving Competition

In this challenge, the robot must be a completely autonomous vehicle in which all decisions must be taken by the systems included in it. Competitors can not include communication between the robot and other external devices. The track is installed within an area of 6.95 x 16.7 m. This has the format of a traffic road. In figure 2.17, a representative view of this route is shown. The track floor is dark and infrared absorbing with lines painted in white [8].

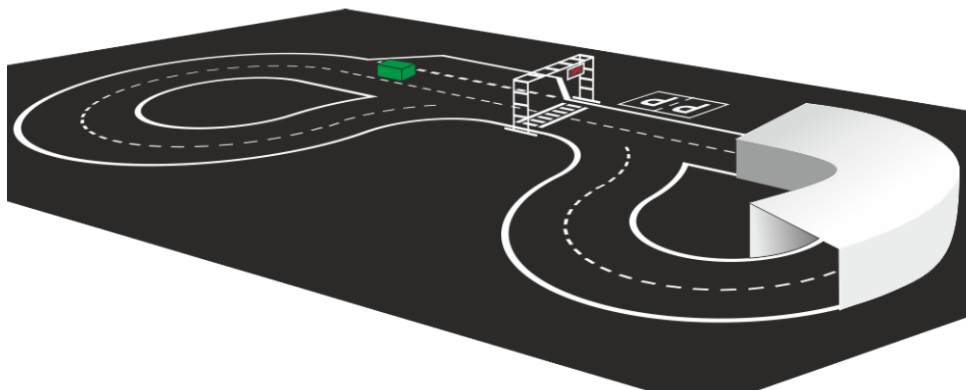


Figure 2.17: Track overview (figure from [8] displayed with permission from Manuel Silva)

Two Thin Film Transistor (TFT) panels are mounted right above the zebra crossing, one in each direction of the track. These panels will show indicating signals to the competing vehicles. The base of the panels are 87 cm above the ground as seen in figure 2.18 [8].

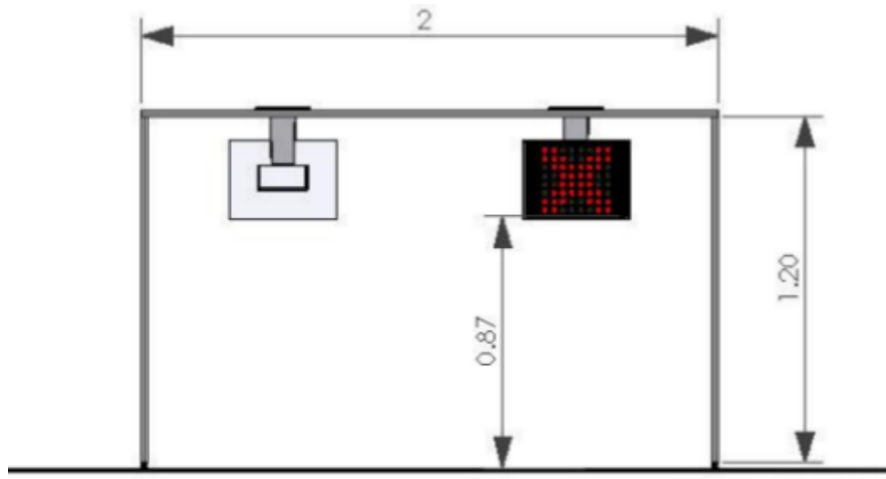


Figure 2.18: Signaling panels (figure from [8] displayed with permission from Manuel Silva)

2.7.2 Indicating Signals

Six possible indicating signals presented on the TFT are shown in figure 2.19. The symbols are displayed over a black background and bounded by a square box [8].

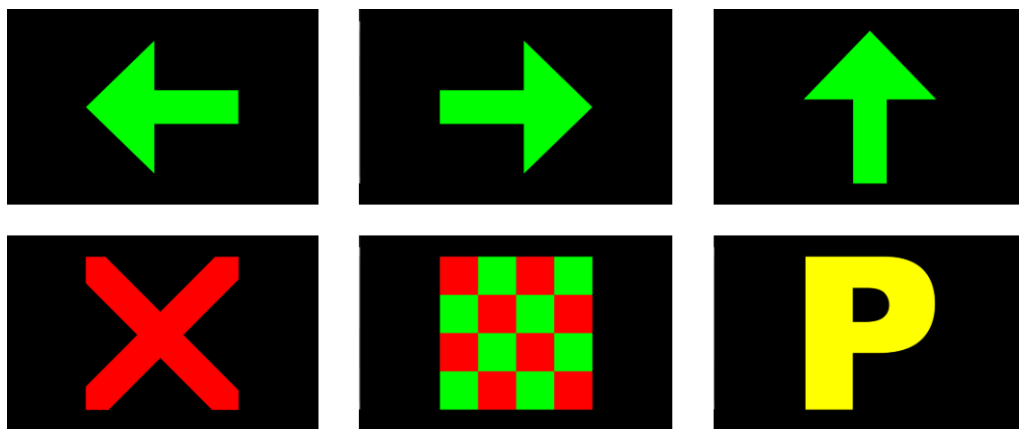


Figure 2.19: Signaling panel drawings (figure from [8] displayed with permission from Manuel Silva)

The purpose of the indicating signals is to conduct the robots trials, giving them orders to give some randomness. The correspondence between each of the functions and the information presented in the indicating panels is represented in the following table [8]:

Function	Action	Sign
1	Follow to the left	A left pointing green horizontal arrow
2	Follow to the right	A right pointing green horizontal arrow
3	Follow straight ahead	A green colored vertical arrow
4	Stop	A red colored "X"
5	End of trial	Red and green checkers flag
6	Follow to parking area	A yellow colored "P"

Table 2.2: Correspondence between each of the functions and the information (table adapted from [8] with permission from Manuel Silva)

2.7.3 Vertical Traffic Signs

Twelve different traffic signs available in the competition are represented in figure 2.21. The vertical signs are standing on a post at 87cm and 10cm away from the sideline as shown in figure 2.20. The signs are placed on the right side of the track [8].



Figure 2.20: Traffic sign placed on the side of the track (figure from [8] displayed with permission from Manuel Silva)

The twelve signals are three different types [8]:

- Four warning signs (triangular shape);
- Four mandatory signs (round shape);
- Four information or services signs (square shape).



Figure 2.21: The twelve traffic signs for the competition (figure from [8] displayed with permission from Manuel Silva)

2.7.4 Vertical Traffic Signs Detection Challenge

The competition is organized in a set of four rounds. Every round has a vertical signs detection challenge. In this challenge, six vertical traffic signs (two per type) are placed along the track, and the competition aims to detect and identify the vertical signs while driving along the track. The signalling panel is only used to trigger the departure of the robot. For each sign type correctly identified, 25 points are assigned and an additional 25 point are added if the number of the sign is also perceived [8]. The goal is to do two laps in the shortest time possible.

2.8 State of the Art

2.8.1 You Only Look Once (YOLO)

In 2016 J. Redmon [9] introduced the YOLO model which was faster than the faster R-CNN [32]. The performance values can be visualized in the following Table 2.3.

These results were achieved because it performs the whole process of detection with only one pass, that is, a single pass on the whole network predicts at the same time the bounding boxes and the respective class probabilities. This model can “see” all the input image, consequently, it promotes a better distinction between the objects and the background, improving the precision of the network. As shown in figure 2.22 the YOLO system divides each image into a grid of $S \times S$ squares. In each square, it is predicted a set of bounding boxes and a confidence score [9].

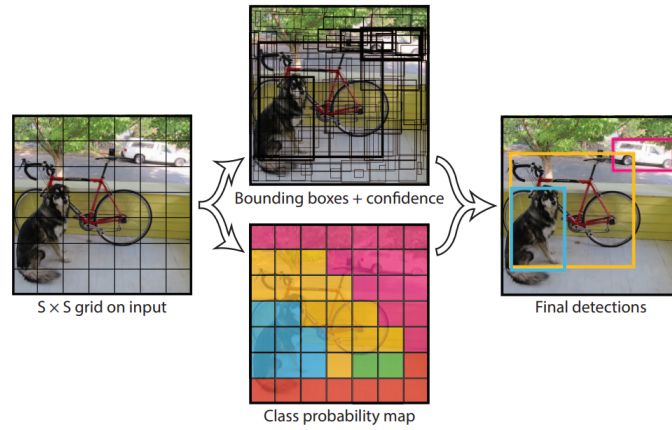


Figure 2.22: The YOLO model (figure from [9] displayed with permission from Ali Farhadi)

These confidence scores reflect how confident the model is that the box contains an object and also how accurate it thinks the box is that it predicts [9]. It can be described by equation 2.4.

$$Pr(object) * IOU_{pred}^{truth} \quad (2.4)$$

The confidence score should be the IoU between the predicted box and the ground truth, but only if the object exists on that cell. Subsequently, each grid cell predicts the class probabilities for each class [32]. The probability that a certain bounding box contains a specific type of object is represented on the following equation 2.5.

$$Pr(Class_i|Object) * Pr(Object) * IOU_{pred}^{truth} = Pr(Class_i) * IOU_{pred}^{truth} \quad (2.5)$$

Where the confidence score for each bounding box and the class prediction are combined into one final score as shown [9].

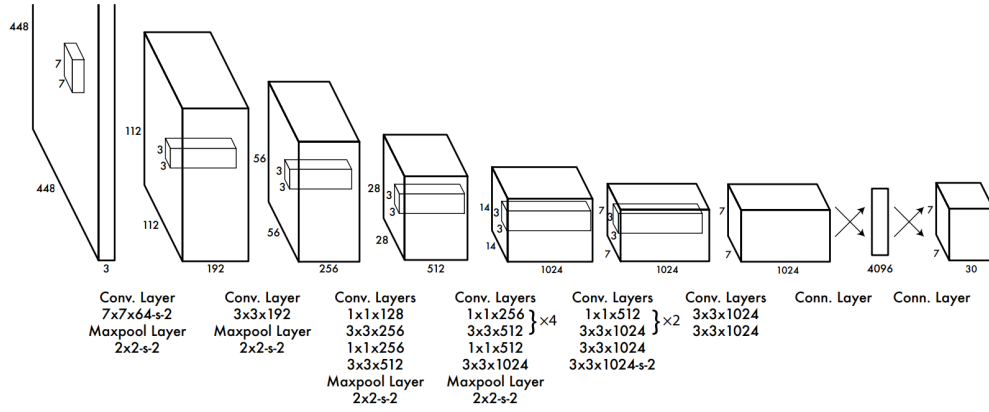


Figure 2.23: The architecture (figure from [9] displayed with permission from Ali Farhadi)

Inspired by the GoogLeNet model for image classification this network architecture has 24 convolutional layers succeeded by 2 fully connected layers. It uses 1×1 convolutional layers followed by 3×3 convolutional layers and is represented in figure 2.23. The last layer is a $7 \times 7 \times 30$ matrix [9] and can be defined by equation 2.6

$$S * S * (B * 5 + C) \quad (2.6)$$

where S represents the size of each grid, B is the number of bounding boxes for each cell and C is the number of classes.

The network can process images in real-time at 45 FPS. With better results than other real-time detectors, Fast YOLO can reach 155 FPS [46].

Real-Time Detectors	mAP	FPS
Fast YOLO	52.7	155
YOLO	63.4	45

Table 2.3: Performance of Real-Time Detectors

Furthermore, YOLO produces fewer false positives in the background, which makes cooperation with Fast R-CNN become possible. An improved version, YOLOv2, was later proposed, which adopts several impressive strategies, such as Batch normalization, anchor boxes, dimension cluster and multi-scale training [46].

2.8.2 YOLO9000: Better, Faster, Stronger

Six months after the release of the first version of the YOLO model [9] (section 2.8.1), an improved version, based on the YOLO detection system, was produced, YOLOv2 . It introduced a training algorithm that trains object detection on detection and classification data [12].

YOLO [9] had an array of flaws such as errors in localization and had relatively low recall when compared with other region proposal-based methods. YOLOv2 had the main target of improving these weaknesses while preserving classification accuracy [12].

2.8.2.1 Better

With the aim of making YOLOv2 a more accurate detector but still fast, the network was simplified and the representation was then easier to learn. Several ideas were proposed to achieve this ultimate target [12]:

- Batch Normalization (BN): leads to developments in convergence, regularization of the model and removes dropout from the model preventing overfitting. Adding BN on all convolutional layers got a 2% mAP improvement;
- High-Resolution Classifier: unlike Yolo, which trains the classifier network with the resolution of 224x224 and 448x448 in detection, YOLOv2 used 448x448 for 10 epochs and then fine-tuned the network on detection. This granted the networks time to settle its filters to work strongly on high-resolution inputs. This produced a 4% increase in mAP;
- Convolutional with Anchor Boxes: Yolo predicted the bounding boxes employing fully connected layers on top of a convolutional feature extractor. It was intended to predict bounding boxes using hand-picked prior, as used in Faster R-CNN [38]. This region proposal network (RPN) predicted offsets and confidences for anchor boxes which simplified the problem and facilitated the learning of the network. This idea diminished accuracy and mAP but enhanced recall;
- Dimension Clusters: K-means clustering on the training set bounding boxes was utilised to find good priors. At 5 priors the centroids performed better which shows that using k-means to form bounding boxes made the task easier to learn;

- Direct location predictions: Yolo had a model instability mainly during early iterations that were caused by predicting the (x,y) for the boxes. Predicting the location coordinates relative to the location of the grid cell made the network easier to learn and more stable. Using dimension clusters along with directly predicting the center of the bounding boxes improved YOLO by 5%;
- Fine-Grained Features: Adding a passthrough layer that brings features from an earlier layer turned the 26x26x512 feature map into a 13x13x2048 feature map giving a 1% performance improvement;
- Multi-Scale Training: To make YOLOv2 more robust than its previous version, which used a 448x448 input resolution, every 10 batches the network adopts a new image dimension size that is a multiple of 32. The smallest choice is 320x320 and the most extensive 608x608.

A review of the results of the proposed ideas can be found in table 2.4.

	YOLO								YOLOv2
batch norm?		✓	✓	✓	✓	✓	✓	✓	✓
hi-res classifier?			✓	✓	✓	✓	✓	✓	✓
convolutional?				✓	✓	✓	✓	✓	✓
anchor boxes?				✓	✓				
new network?					✓	✓	✓	✓	✓
dimension priors?						✓	✓	✓	✓
location prediction?						✓	✓	✓	✓
passthrough?							✓	✓	✓
multi-scale?								✓	✓
hi-res detector?									✓
VOC2007 mAP	63.4	65.8	69.5	69.2	69.6	74.4	75.4	76.8	78.6

Table 2.4: The path from YOLO to YOLOv2 (table adapted from [12] with permission from Ali Farhadi)

2.8.2.2 Faster

To make YOLOv2 a faster detector, a new classification model was proposed, Darknet-19. As demonstrated in table 2.5, this model has 19 convolutional layers and 5 maxpooling layers [12].

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

Table 2.5: Darknet-19 (table adapted from [12] with permission from Ali Farhadi)

2.8.2.3 Stronger

This method uses labelled images to learn the location and the classification of the objects. In order to improve classification, hierarchical classification is introduced. Using Wordnet [47], a language dataset that arranges concepts and relates them, the network associates labels with their type. For example, "Norfolk terrier" and "Yorkshire terrier" are both hyponyms of "terrier", which is a type of "dog", this creates a structured tree. Figure 2.24 demonstrates the application of a WordTree hierarchy in comparison to ImageNet and COCO label structures [12].

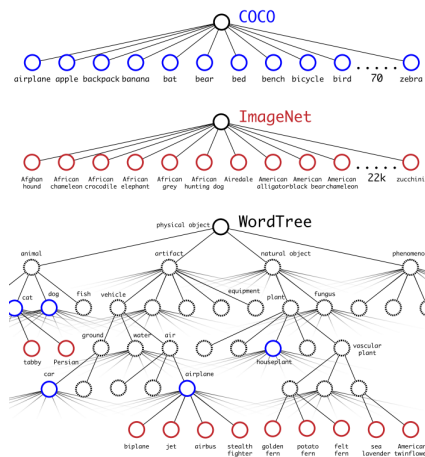


Figure 2.24: Combining datasets using WordTree hierarchy (figure from [10] displayed with permission from Ali Farhadi)

2.8.2.4 Yolo9000

Using the previous explained YOLOv2 architecture and combining datasets using WordTree, YOLO9000 was created using 9418 classes. YOLO9000 gets 19.7 mAP overall [12].

2.8.3 YOLOv3: An Incremental Improvement

In 2018 the third iteration of YOLO, YOLOv3 was published [10]. This version is more accurate, features multi-scale detection and a stronger feature extractor network. This iteration can be divided into two elements: Feature Extractor and Detector. Both of these elements work with multiple scales. When an input is presented to the network, it goes through the Feature Extractor first and with the obtained features, the Detector gets the bounding boxes and predicts the correspondent class.

2.8.3.1 Darknet-53

A new network was presented to perform feature extraction, Darknet-53. This is an upgrade from the previous YOLOv2 version, Darknet-19. This network uses successive 3x3 and 1x1 convolutional layers, has shortcut connections and has 53 convolutional layers [10]. The network is represented in the following Table 2.6.

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

Table 2.6: Darknet-53 (figure from [10] displayed with permission from Ali Farhadi)

2.8.3.2 Bounding Box Prediction

The system uses dimension clusters as anchor boxes to predict bounding boxes [10]. Four coordinates are predicted for each bounding box: t_x, t_y, t_w, t_h . If an offset is detected from the top left corner of the image (c_x, c_y) and the bounding box has width and height p_w, p_h , then the prediction corresponds to, as shown in Figure 2.25:

$$b_x = \sigma(t_x) + c_x$$

$$b_y = \sigma(t_y) + c_y$$

$$b_w = p_w e^{t_w}$$

$$b_h = p_h e^{t_h}$$

The width and height of the box are predicted as offsets for cluster centroids. The centre coordinates are predicted using a sigmoid function.

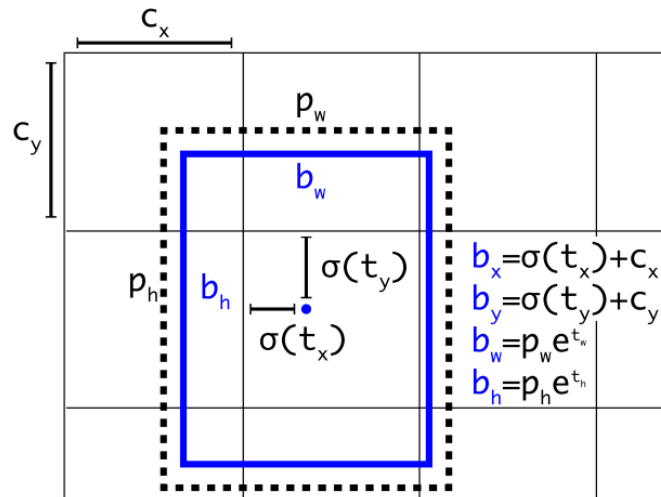


Figure 2.25: Bounding boxes with dimension priors and location prediction (figure from [10] displayed with permission from Ali Farhadi)

Using logistic regression the network predicts an objectness score for each bounding box [10].

2.8.3.3 Tiny Yolov3

YOLOv3 is one of the fastest deep learning-based object detectors but its computational demand for embedded devices such as the Raspberry Pi, for examples, is exceeding. In order to make

implementations in such embedded devices, the YOLO creators introduced Tiny YOLOv3, a variation of the YOLO architecture [48]. This architecture allows a network to be approximately 442% faster than YOLO.

This network is comprised of 7 convolutional layers, 6 maxpool layers for image feature extraction and 2 scales of detection layers as demonstrated in Figure 2.26 [49].

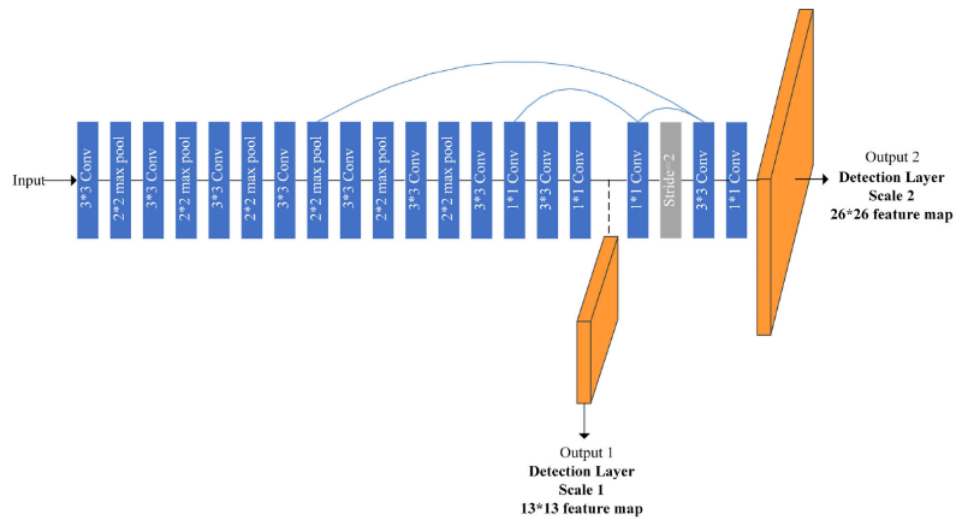


Figure 2.26: Structure of Tiny YOLOv3

Since Tiny YOLOv3 is a smaller version, this means that it is unfortunately even less accurate.

2.8.4 Traffic Sign Recognition for Autonomous Driving Robot

This project proposes a Traffic Sign Recognition software produced for a robot or a car. The aim of this project was the participation in the Autonomous Driving Competition in the Portuguese Festival of Robotics. The robot acquires the images for the detection and classification of traffic signs and traffic lights with a camera attached to its chassis. The system should be qualified to recognize nine different traffic signs that appear on the side of the track and five traffic lights displayed by a LED panel above the departure area [11]. Some examples are shown in figure 2.27



Figure 2.27: Examples of signs to recognize (figure from [11] displayed with permission from Vitor Filipe)

The software was divided into three stages [11]:

- Detection - using colour segmentation the image regions with signs are identified;
- Pictogram extraction - in order to identify the type of signal the system obtains the pictogram;
- Classification - a binary pattern is presented to a feed-forward neural network to classify the sign.

2.8.4.1 Traffic Signs Algorithm

- Detection

Using a digital camera the system acquires frames of the surrounding environment where the traffic signs and traffic lights are present. The frames collected are in the RGB colour space and are converted to HSV colour space in order to make the detection more simplified and robust to diverse illumination condition. The detection is achieved by thresholding the image in the HSV colour space. By thresholding the image, a binary image with several potential regions for sign location is achieved. This location has blue or red as the dominant colour. Every potential region is examined for a minimum and maximum value (region size filtering). Only roughly square regions are further evaluated as signs [11]. In this stage, the pictogram might be missing due to its colour (Figure 2.28).

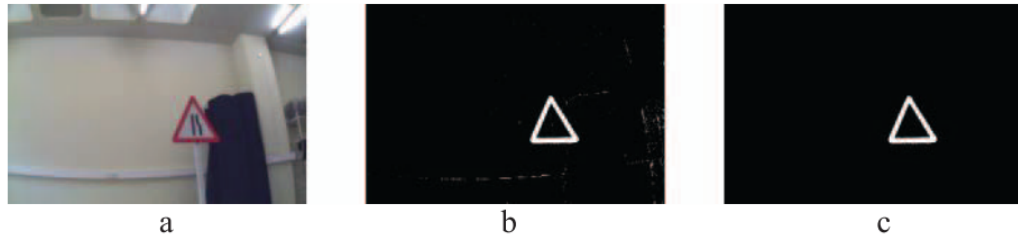


Figure 2.28: a) Frame acquired with a visible traffic sign. b) Regions detected after thresholding. c) Detected external contour (figure from [11] displayed with permission from Vitor Filipe)

- Pictogram Extraction

Once the detection is performed, the area of the sign in the image is extracted, scaled and converted to grayscale. Two thresholdings are employed to segment the interior of the pictogram, one for dark pictograms and one for white pictograms. A region size filter is employed to eliminate the small noise regions of the resulting binary image [11]. (Figure 2.29)

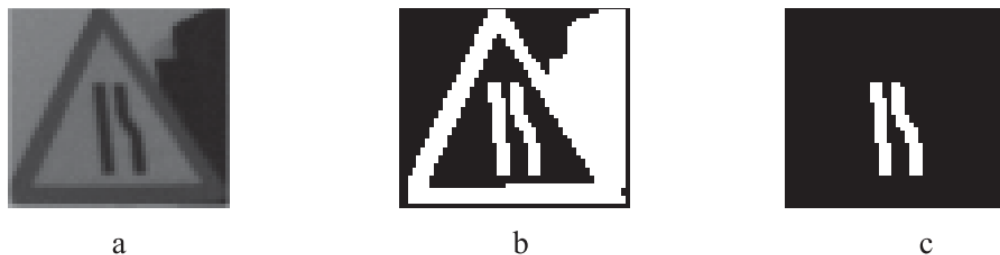


Figure 2.29: a) Sub-image (grayscale) with the sign extracted from the RGB frame. b) Binary image after threshold operation. c) Result of pictogram extraction (figure from [11] displayed with permission from Vitor Filipe)

Applying a logical OR operation combines the external contour and the pictogram of the traffic sign and produces a 50x50 pixels binary pattern matrix [11]. (Figure 2.30)

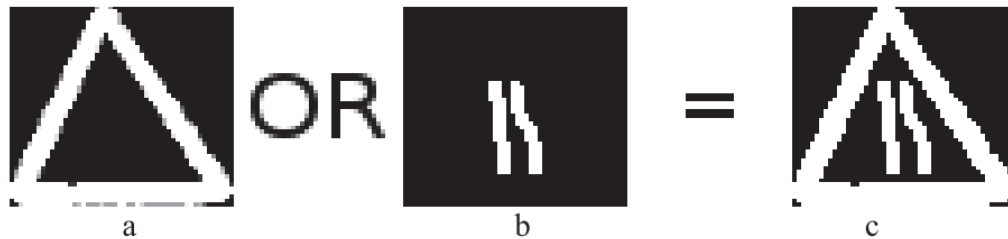


Figure 2.30: a) External contour obtained in the detection stage. b) Internal symbol obtained in pictogram extraction stage. c) Binary pattern obtained after logical OR operation (figure from [11] displayed with permission from Vitor Filipe)

- Classification

The classifier is a feed-forward Neural Network (NN) that distinguishes the sign. The NN has 3 layers [11]:

- An input layer with 2500 neurons, one for each 50*50 pixels in the input;
- A hidden layer with 20 neurons;
- An output layer with 9 possible outputs, one for each sign trained.

Using a sigmoid activation function, the NN was trained in supervised mode by the backpropagation learning algorithm. To train the NN a dataset with 3150 real traffic sign images was used. In order to make the NN more robust, these images had different levels of perspective distortion and imprecision in the pictogram [11].

2.8.4.2 Traffic Lights Algorithm

The proposed traffic lights recognition algorithm is identical to the traffic sign algorithm however some steps are not needed because it is less complex. For the detection, thresholding is used to obtain colour segmentation to detect red, yellow and green regions. A conversion to HSV colour space is employed. This stage suppresses the pictogram extraction stage because it locates and obtains the binary pattern matrix. The NN for traffic lights is less complex than the traffic sign because there are fewer outputs and the patterns are simpler. It has the same 2500 neurons, one for each 50*50 pixels in the input but the hidden layer only has 15 neurons with a logistic regression activation function. The output layer is made up of 5 outputs [11].

2.8.4.3 Results

To assess the algorithms, tests were performed under the same conditions as the competition. In most signs, 100% precision was obtained in both algorithms. The traffic lights obtained over 96% recall and the traffic signs obtained between 52% and 88.2% recall. Most failures are in detection and pictogram extraction [11].

Chapter 3

System Specification

The software were implemented on Ubuntu 20.04 operating system on an ASUS Vivobook Pro N580VD with an Intel Core i7 7th Gen 7700HQ CPU and an Nvidia GeForce GTX 1050.

Utilizing Python as the main programming language, the chosen Integrated Development Environment (IDE) was Pycharm Community 2021.1. The library that was essentially used was OpenCV. OpenCV (Open Source Computer Vision Library) is an open-source computer vision and machine learning software library and was used in multiple image processing tasks through the software development.

Google Colab was used in every train that was performed in order to cut down the time that each train took. Colab provides free access to powerful GPUs (Graphics Processing Unit) that empowers users to write and execute arbitrary python code through the browser and is particularly well suited for machine learning.

3.1 Model Framework

Two software were developed for analysis of traffic signs and traffic lights for each one of the following objectives:

- Autonomous Driving Competition of the RoboCup Portuguese Open in simulation
- Autonomous Driving Competition of the RoboCup Portuguese Open
- Public road

In order to develop the algorithms to solve each of the three objectives, it was necessary to divide the problem into three sequential parts:

- Acquisition
- Training
- Testing

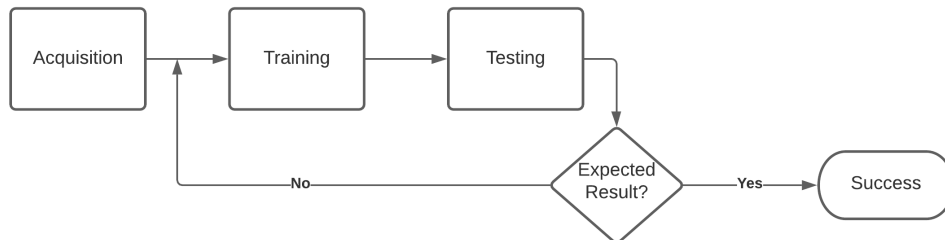


Figure 3.1: Model Framework

In the acquisition part, different frames were acquired and labelled in order to train the network, and one example is shown in figure 3.2.



Figure 3.2: Image from one of the Datasets and its correspondent text file with the bounding box coordinates, dimensions and label

In the previous image, a bounding box is around the STOP sign and in the associated text file the sign location and label, which is explained in the following sections.

After gathering a sufficient amount of labelled images for the network, a train is performed.



Figure 3.3: Example of the testing phase

In the test part, the frames from the camera are sequentially obtained and passed to the network that makes predictions of the bounding boxes and the corresponding label and confidence, as seen in figure 3.3. As observed in figure 3.1 if the testing phase is not successful the train has to be repeated with different parameters to ensure the expected results.













3.2 Autonomous Driving Competition of the RoboCup Portuguese Open in simulation

The first objective of the project is the development of a detection and classification software for the Autonomous Driving Competition of the RoboCup Portuguese Open in simulation.

The simulation environment chosen to simulate the entire competition environment was CoppeliaSim Edu, Ubuntu 20.04 version. The software was chosen because it is the one used by the Minho autonomous driving team for the development of the project. CoppeliaSim is a robot simulator based on a distributed control architecture in which each object/model can be individually controlled. This software is used for fast algorithm development, factory automation simulations, fast prototyping and verification and robotics-related education.

In this section, the three phases are described in order to accomplish the objective.

As referenced in section 2.7 the autonomous driving competition of the RoboCup Portuguese Open consists of correctly identifying 6 traffic lights, represented in figure 2.19 and 12 traffic signs, represented in the figure 2.21. In addition to these 18 signs/lights, 12 new traffic signs were chosen to upgrade the variety of the signs. These were specially chosen because they give instructions of movement such as whether it is to stop, turn to a direction or increase/decrease speed. In total the dataset contains 24 traffic signs and 6 traffic lights which are embodied in table 3.2.

ID	Traffic Sign/- Light	Image	ID	Traffic Sign/- Light	Image
0	Left Direction		1	Roundabout	
2	Lights		3	Public Transport	
4	60km/h		5	Hospital	
6	Car Park		7	Crosswalk	
8	Animals		9	Road Depression	
10	Narrow Passage		11	Dangers	










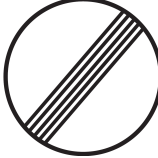


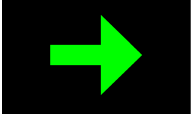

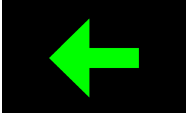

ID	Traffic Sign/- Light	Image	ID	Traffic Sign/- Light	Image
12	Yield		13	STOP	
14	U-Turn		15	Overtaking	
16	Prohibited Direction		17	30km/h	
18	40km/h		19	Parking	
20	End Overtaking		21	End Prohibitions	
22	Right Obligation		23	Left Obligation	
24	Traffic Light Right		25	Traffic Light Front	
26	Traffic Light Left		27	Traffic Light Park	

Table 3.1: The Traffic Signs/Lights in the Dataset


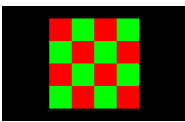
ID	Traffic Sign/- Light	Image	ID	Traffic Sign/- Light	Image
28	Traffic Light Stop		29	Traffic Light Fin- ish	

Table 3.2: The Traffic Signs/Lights in the Dataset

Following the RoboCup rules, models of the traffic signs were created using Solidworks. Solidworks is used for planning, visual idealization, modelling, feasibility assessment, prototyping, and project management. The software is then also used for the design and building of mechanical, electrical, and software elements. This software was chosen to construct the 3D models because of its easy access to all tools and also because it is user friendly.

The traffic signs were created standing on an 87cm post and the sign limited by a bounding square of 305x305mm. The base of the sign is 305mmx200mm.

The 24 traffic signs can be split into 6 shapes (square, circular, triangular, hexagonal, inverted triangle and rectangle). For each shape, a model of the sign was created. In figure 3.4 the 3D models, which are according to the *Festival Nacional de Robótica* guidelines, are represented.

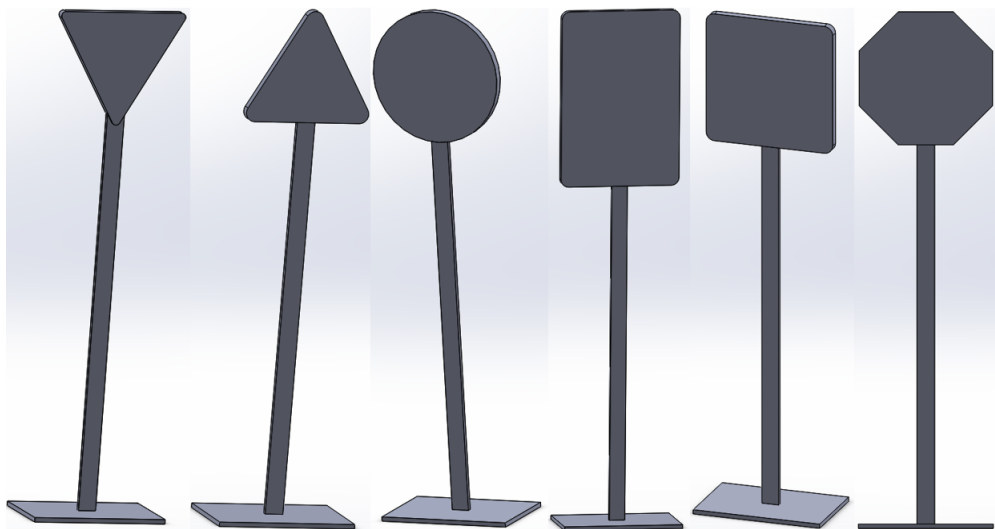


Figure 3.4: The six types of Traffic Signs built in Solidworks

In order to import the models to CoppeliaSim, they were converted into URDF (Unified Robot Description Format) files. This type of file allows the user to use, in the simulator, any kind of 3D model created in SolidWorks. In figure 3.5 the 6 imported models are represented.

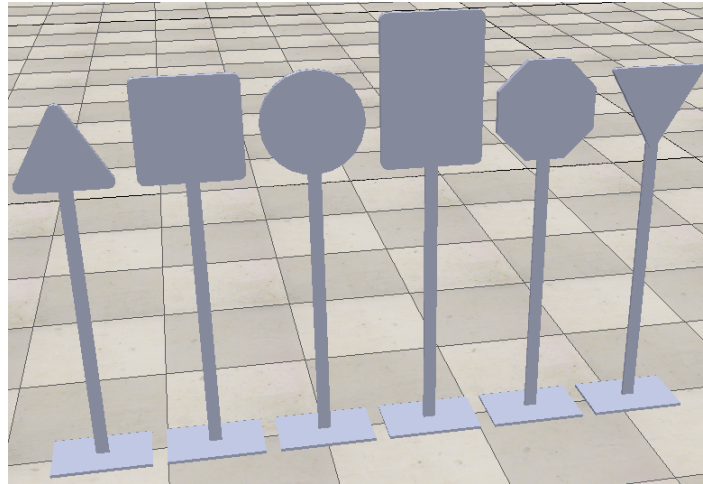


Figure 3.5: The six types of Traffic Signs in CoppeliaSim

To distinguish each sign the pattern has to be inserted in the shape. The pattern cannot be added to the shape because it would apply it to the entire shape and the pattern should only be in the upper part of the sign. A Solidworks file is created using only the traffic sign without the post and base. After importing this file and placing it on top of the sign, the traffic sign has the pattern in the correct position. In the following figure, every traffic sign that was built with this method can be visualized.

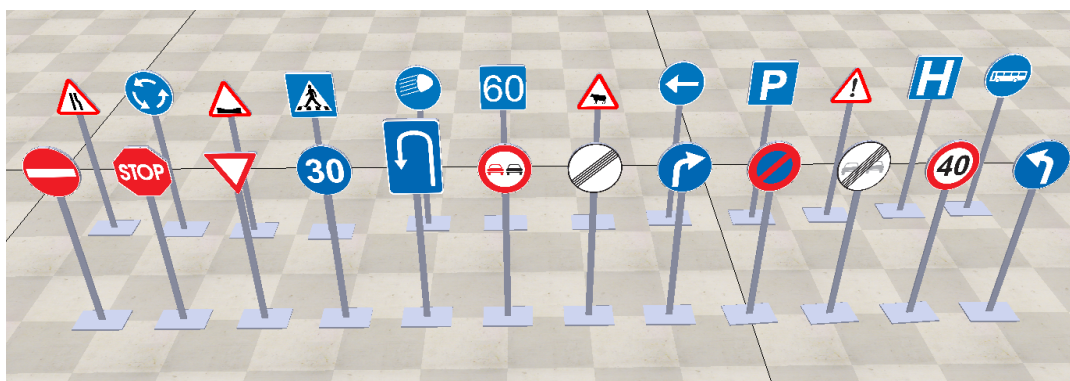


Figure 3.6: All Traffic Signs in CoppeliaSim

The virtual traffic light was already previously developed by another member from *Laboratório de Automação e Robótica*. It is a representation of the traffic light referenced in section 2.7.1, and is shown in figure 2.18. It consists of a structure with the size of the traffic sign from the competition. This shape can change its pattern to match the corresponding desired traffic light. This is controlled by the computer keyboard in which the light is chosen by its letter:

- S: Stop
- P: Park
- F: Finish
- Left arrow: left
- Right arrow: right
- Front arrow: front

In the following figure 3.7, the front arrow light is represented.

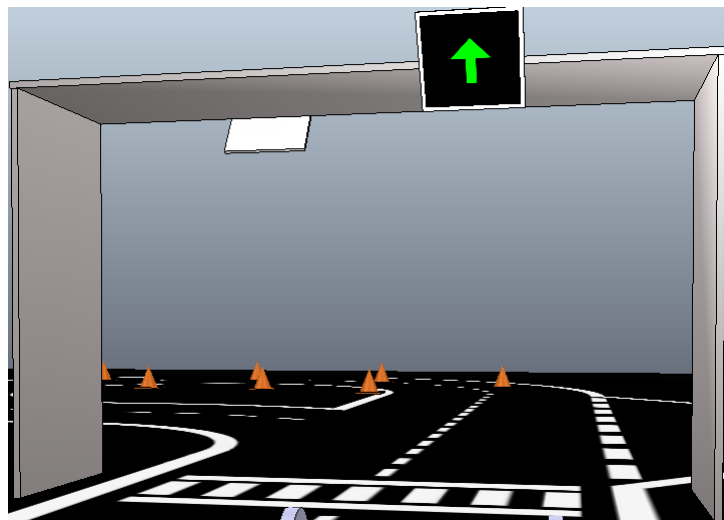


Figure 3.7: Example of a Traffic Light in CoppeliaSim

To make the simulation more realistic a model of the robot was built with the aim of testing its reaction to the detection of each sign. Based on the real robot from the *Laboratório de Automação*

e Robótica and using SolidWorks, a 3D model of the chassis was built. In figure 3.8, a comparison between the real robot and the 3D model can be seen.

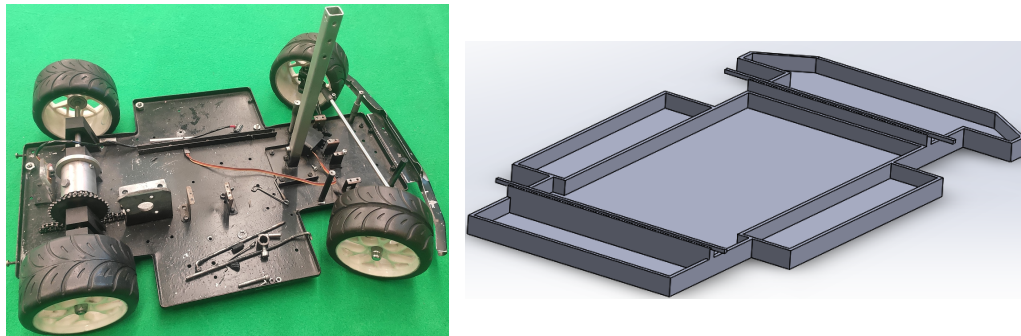


Figure 3.8: Comparison between the real robot (left) and its 3D model (right)

Afterwards, the model was imported as a URDF file to CoppeliaSim. Also according to the real robot, four wheels with the same dimensions were added. To simulate the motors from the rear wheels and to bind the base to the wheels, revolute joints were placed in each wheel. Revolute joints are a CoppeliaSim object that allows the user to emulate a motor by rotating the object that is attached to it. In the rear wheels, the joints were used to move the car forward or backwards, depending on the direction that the joints were rotating. The front wheels have joints in which the main purpose is to control the car steering. Depending on the direction the joints were rotating, the robot can control the steering and the angle that it's heading. In figure 3.9, the addition of the wheels and the joints can be visualized. The dimensions of the joints were increased to capture this figure, usually, they are not visible.

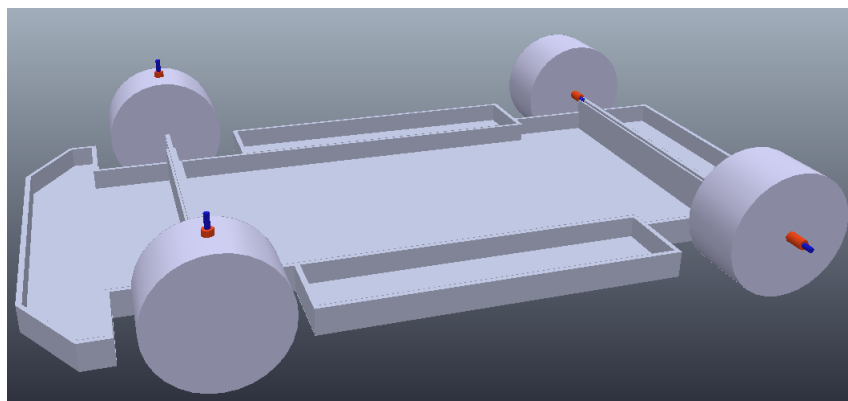


Figure 3.9: The joints that allow the movement of the car

A CoppeliaSim camera was added to the car in the same position as in the real robot to detect and categorize each traffic sign/light that is in the presence of the robot. This camera grabs frames that are processed in the network to detect which traffic sign/light is present.

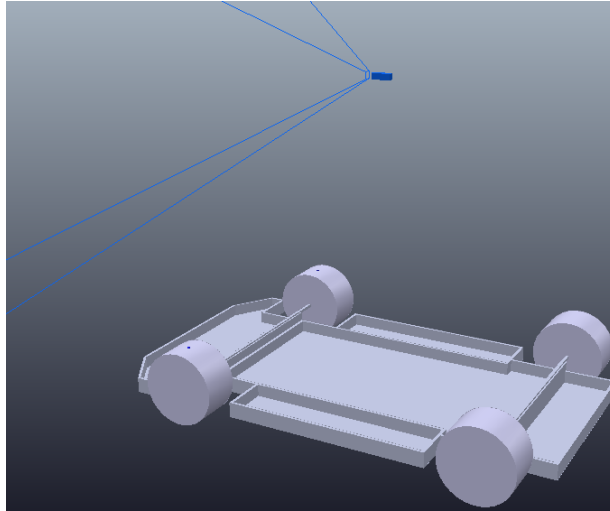


Figure 3.10: Camera in the car

The frames that the car camera records are later processed in a python script, which is explained later, that outputs the signs/lights that are in the frame.

To make a connection between CoppeliaSim, where the car and the signs/lights are simulated, and Pycharm, where the frames are processed, ROS was used.

ROS (Robot Operating System) is a framework for writing robot software. It has multiple tools, libraries, and conventions that simplify the task of creating complex and robust robot behaviour across multiple robotic platforms.

ROS can have multiple processes running in parallel, each one is called a node. Every node should be responsible for one task. Nodes communicate with each other using messages passing via logical channels called topics. Each node can send or get data from another node using the publish/subscribe model. If data needs to be transferred from one node to another, one node publishes it into a topic and another node subscribes to the topic to receive the data.

The ROS Master provides naming and registration services to the rest of the nodes in the ROS system. It traces publishers and subscribers to topics as well as services. The role of the Master is to enable individual ROS nodes to locate one another. Once these nodes have located each other they can communicate among them peer-to-peer.

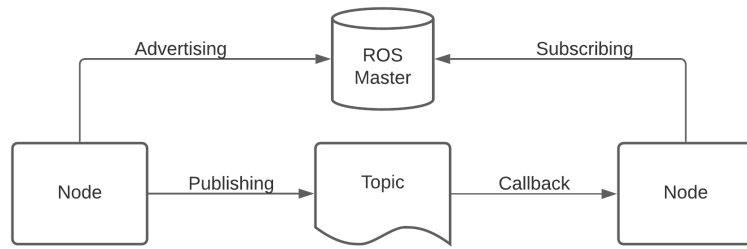


Figure 3.11: ROS Master

All the supervised learning algorithms software were developed on a Python script external to CoppeliaSim.

The system that controls the robot depending on the identified traffic signs/lights, consisted of two ROS nodes, one for the simulator, named `/sim_ros_interface` and another for the control script, named `/talker`.

In figure 3.12, the implemented ROS `rqt_graph` is presented illustrating how both of the ROS nodes communicate.

`/image` (image) - Frame from the car's camera

`/chatter` (array) - Label from the traffic sign/light that is detected in the image. If no sign/light is detected, this array is empty.



Figure 3.12: Graphical representation of how both of the ROS nodes communicate

To create a node in the CoppeliaSim environment to communicate with PyCharm, Non threaded associated child scripts are used to customise the simulation. This method consists of a Lua script. Lua is powerful, fast and designed to be a lightweight embeddable scripting language. It is used for all sorts of applications, including image processing. It is completely compatible with CoppeliaSim. This method allows users to control every element of the simulation. Two embedded scripts were developed, one regarding the image from the camera and the other one regarding the control of the car.

The following part of the script is the creation of the subscriber in CoppeliaSim. This subscriber will receive the labels of the detected signs and will subscribe to the "chatter" node created in Pycharm. This creation is specified in function `sysCall_init()` which is the function that runs at the start of the simulation. The `subscriber_callback(msg)` is the function in which CoppeliaSim receives the label of the detected sign and changes the robot speed and direction according to the detection.

```
function subscriber_callback(msg)
function sysCall_init()
-- The child script initialization
objectHandle=sim.getObjectAssociatedWithScript(sim.handle_self)
objectName=sim.getObjectHandle(objectHandle)
rosInterfacePresent=simROS

-- Prepare the float32 publisher and subscriber (we subscribe to the topic we advertise):
if rosInterfacePresent then
subscriber=simROS.subscribe("/chatter", "std_msgs/String", 'subscriber_callback')
end

-- Joint/ Back Motor:
motorR = sim.getObjectHandle("VelocityJointR")
motorL = sim.getObjectHandle("VelocityJointL")

-- Joint/ Front Motor:
motorFrontR = sim.getObjectHandle("MotorionJointR")
motorFrontL = sim.getObjectHandle("MotorionJointL")
```

Figure 3.13: Subscriber creation in CoppeliaSim

In the other script, the publisher that sends the frame from the robot camera is created. In the `sysCall_init()` this initialization is performed.

```
function sysCall_init()
-- Get some handles:
activeVisionSensor=sim.getObjectHandle("Vision_sensor")

-- Enable an image publisher:
if simROS then
sim.addLog(sim.verbosity_scriptinfos,"ROS interface was found.")
pub=simROS.advertise("/Image", 'sensor_msgs/Image')
simROS.publisherTreatUInt8ArrayAsString(pub) -- treat uint8 arrays as strings (much faster)
else
sim.addLog(sim.verbosity_scripterrors,"ROS interface was not found. cannot run.")
end
end
```

Figure 3.14: Publisher creation in CoppeliaSim

Using this system the frames from the car's camera are sent to Pycharm, which processes them and if there is any traffic sign/light present, a command is returned to CoppeliaSim so that the car can react to the sign/light.

3.2.1 Acquisition

The acquisition phase of the first objective has the goal of creating a dataset with images from all the traffic signs and traffic lights in order to train the network. A great number of images with signs/lights in different positions, distances to the camera make the network more robust.

The first step to create the dataset, after creating all the signs and lights, is to record videos. This method was chosen because it allows the capture of the signs/lights in different situations that will be later explained. Multiple videos were created diversifying the following traits:

- Background: changing the colour, inserting CoppeliaSim objects such as robots, tables, chairs and other objects;
- Number of signs/lights: videos with one or more;
- The angle of the camera: ranging from a top, bottom or side view;
- Distance to the camera: recorded from near or far away.

After this step was finished, all the videos were combined creating a longer video. Just like the individual videos, the final video had 60 fps (frames per second).

Using Python as the main language and Pycharm Community 2021.1 as the IDE, a script was developed in order to convert the video frames into images. Instead of converting every frame, only one in every six frames was converted. This decision was made to prevent having extremely similar images. Since the video camera is always in motion, converting a frame every 6 frames allows the user to have mostly different images.

The final video has 4 minutes and 47 seconds and using the previously mentioned script, 2873 images were created. In the following table 3.3, the number of images in which the signs/lights appear is presented.

Traffic Sign/Light	Number of images
Left Direction	397
Roundabout	241
Lights	406
Public Transport	323
60km/h	322
Hospital	303
Car Park	391
Crosswalk	290
Animals	366
Road Depression	287
Narrow Passage	243
Dangers	334
Yield	323
STOP	345
U-Turn	324
Overtaking	362
Prohibited Direction	350
30km/h	370
40km/h	351
Parking	381
End Overtaking	358
End Prohibitions	354
Right Obligation	361
Left Obligation	330
Traffic Light Right	149
Traffic Light Front	73
Traffic Light Left	41
Traffic Light Park	83
Traffic Light Stop	69
Traffic Light Finish	66

Table 3.3: Number of images per Traffic Sign/Light

The inputs for the YOLOv3 algorithm are images, each one with a correspondent text file. This text file has the following format for each object in an image:

```
Label_ID X_CENTER_NORM Y_CENTER_NORM WIDTH_NORM HEIGHT_NORM
```

where,

```
Label_ID = Correspondent label
X_CENTER_NORM = X_CENTER/IMAGE_WIDTH
Y_CENTER_NORM = Y_CENTER/IMAGE_HEIGHT
```

$WIDTH_NORM = WIDTH_OF_BOUNDING_BOX/IMAGE_WIDTH$

$HEIGHT_NORM = HEIGHT_OF_BOUNDING_BOX/IMAGE_HEIGHT$

- The Label_ID is a number from 0 to (number of classes - 1) and expresses the class of the sign/light that is in the bounding box;
- X_CENTER_NORM: the X coordinate in percentage from 0 to 1 of the centre of the bounding box relative to the image width;
- Y_CENTER_NORM: the Y coordinate in percentage from 0 to 1 of the centre of the bounding box relative to the image height;
- WIDTH_NORM: width in percentage from 0 to 1 of the bounding box relative to the image width;
- HEIGHT_NORM: height in percentage from 0 to 1 of the bounding box relative to the image height.

If an image has more than one sign/light, the correspondent text file will have each sign/light in each line.

Unlike the acquisition phase from the following objectives, in order to label every image, Labellmg software was chosen. Labellmg is a graphical image annotation tool and allows users to manually save annotations as XML files in PASCAL VOC format or text in YOLO format. Every object from every image was manually labelled. This methodology is not the most time and work efficient because it could be made using a python script as will be explained in the following objectives. It was also used because the project was at an early stage when the images were gradually being labelled to also learn more about the network described in section 3.2.2.

With the goal of evaluating the network the dataset must be divided into two datasets:

- Training - the dataset that is used to train the model (weights and biases in the case of a Neural Network). The model sees and learns from this data.
- Validation - the sample of data used to provide an unbiased evaluation of a model fit on the training dataset while tuning model hyperparameters. The evaluation becomes more biased as skill on the validation dataset is incorporated into the model configuration.

In this objective, the chosen ratio was validation 20% and training 80%.

To achieve this ratio a Python script was developed to divide the dataset. In the script, one in every five images was sent to the validation dataset and the remaining were sent to the training dataset. Every image that was sent to every dataset was sent with its corresponding text file.

3.2.2 Training

Following the acquisition phase, where the Dataset was created, the data is ready to be entered into the network to train itself. For the training phase, two networks were used to develop two possible solutions for the objective, YOLOV3 and YOLOV3_tiny. These networks were chosen due to their high FPS and accuracy. As mentioned in section 2.8.3 where both of the networks were described, YOLOV3_tiny is a variation of the YOLOV3 architecture that was built for devices that have limited computational power. In this chapter, the steps to train the dataset in each of the networks are explained and in the following chapter 5 a conclusion of which is the best solution for the objective is presented.

While training, a neural network takes in inputs, which are then processed in hidden layers using weights that are adjusted to find patterns in order to make better predictions. These operations are essentially matrix multiplications and can be trained faster in DL models by running all operations at the same time instead of one after the other. This can be achieved by using a GPU to train the model. The more powerful the GPU is, the faster the training will be. With the aim of lowering the training time and avoiding having the network training in the computer for multiple days at full power, Google Colab Pro was chosen to perform every train. Colab enables training in state of the art GPU, mainly the Nvidia Tesla P100.

YOLOV3 and YOLOV3_tiny have similar training steps which are explained. All the commands here presented were executed in Colab.

The first step to train the network, after the dataset is completed, is to clone the Darknet repository. Darknet is an open source neural network framework that allows users to use its two optional dependencies: OpenCV if users want a wider variety of supported image types or CUDA if they want GPU computation. CUDA (Compute Unified Device Architecture) is a parallel computing platform and application programming interface (API) model developed by Nvidia. It allows users to use a CUDA-enabled graphics processing unit (GPU) for general purpose processing which accelerates the processes.

This framework features, among other options, YOLOv3 and YOLOV3_tiny which are used in this phase.

Darknet is cloned with the following command:

```
!git clone https://github.com/AlexeyAB/darknet
```

The following step is configuring the Makefile in order to enable OpenCV, CUDA and GPU. OpenCV is enabled because it will be used after the train to process the input images to predict if any traffic signs/lights are present. CUDA and GPU are enabled to take full advantage of Nvidia's CUDA enabled GPU training. Initially, the directory is changed to darknet. This is obtained with the following commands:

```
%cd darknet
!sed -i 's/OPENCV=0/OPENCV=1/' Makefile
!sed -i 's/GPU=0/GPU=1/' Makefile
!sed -i 's/CUDNN=0/CUDNN=1/' Makefile
```

After the Makefile has every desired Darknet setup option, the repository is compiled to be used with all these new options. This step is made by running the following command:

```
!make
```

After Darknet is compiled, every available functionality can be used. In this step, the first difference between YOLOv3 and YOLOV3_tiny appears. Darknet provides a config file for each YOLO option. For YOLOv3, the yolov3.cfg is used and for the tiny version, yolov3-tiny.cfg is used. This step is the most important because every hyperparameter for the network is established. These hyperparameters are in both config files. The most general hyperparameters are:

- batch;
- subdivisions;
- width;
- height;
- channels;

- classes;
- max_batches.

The batch parameter sets the batch size used during training. The train involves iteratively refreshing the weights of the network based on the detection/location errors made on the training dataset. The training dataset contains a great number of images and instead of using all these images at once to update the weights, a small division of images is used in one iteration, this is called the batch size. For example, if the batch size is 64 then 64 images are employed in one iteration to update the weights.

Even with small batch sizes, the computational power required can be too demanding for a GPU with low memory. Subdivision is used in Darknet to allow users to process a fraction of the batch size at one time on the GPU. The GPU will process $\frac{\text{batch}}{\text{subdivision}}$ images each time but the iteration would be complete only after all the batch images are processed.

The width and height parameters define the image dimensions in which the images will be resized. The maximum value is 608x608 and with these dimensions, the results should improve but it takes longer to train. The channels parameter indicates the number of channels in the input, every image will be converted to this number of channels during Training and Detection. In every train performed the network had 3 channels because RGB images were used.

Classes represent the number of different classes contained in the dataset. For example, if the dataset has 10 traffic lights and 2 traffic signs, the classes parameter should be 12.

Finally, it needs to be specified how many iterations should the training last. For multi-class object detectors, the max_batches number is higher, the train requires to run for more batches. For this class object detector, it is advisable to run the training for at least (average number of images per class)*(number of classes).

The following are hyperparameters for optimization:

- momentum;
- decay;
- learning_rate;
- policy;

- steps;
- scales;
- burn_in.

Since the weights are refreshed in each iteration using a smaller group of images instead of the entire dataset, they can fluctuate a lot. The momentum parameter is used to prevent this large weight fluctuation.

Overfitting is common in neural networks due to the great number of weights contained. This occurs when a function is too closely aligned to a limited set of data. As a result, the model is useful in reference only to its training dataset, and not to any other datasets, such as the testing one. This happens when the model learns too much about the training data. In this case, the model proves to be suitable only for the training data, as if the model had only memorized the training data and was not able to generalize to other data never seen before. To prevent overfitting, the decay parameter is used to penalize large values for weights.

Learning rate controls how quickly the model is adjusted to the problem. Smaller learning rates require more training iterations given the smaller changes made to the weights in each update, whereas larger learning rates result in rapid changes and require fewer training iterations.

At the start of the train, the learning rate needs to be high because it is the point that it has less information about the data. In the rest of the train, the weights must have a smaller learning rate due to the great amount of data. At this point, the learning rate needs to be decreased over time. To specify this decrease in the config file, the learning rate decrease policy must be defined into steps. With this policy, the learning rate will decrease every number of iterations, defined by the parameter steps, with a scale defined by the parameter scales. For example, if `learning_rate=0.001`, `policy=steps`, `steps=3800` and `scales=.1`, the learning rate will decrease by $\text{current_learning_rate} * 0.1$ ($\text{current_learning_rate} * \text{scales}$) every 3800 (steps) iterations. It has been found that the training speed tends to increase if there is a lower learning rate for a short period at the very beginning. The `burn_in` parameter can control this.

To make the maximum use of the dataset the config file has 4 data augmentation parameters:

- angle;
- saturation;

- exposure;
- hue.

Data augmentation is a technique to increase the diversity of the training set by applying transformations such as image rotation, orientation, location, scale and brightness. Usually, a dataset of images is held in a limited set of conditions but the target application may exist in a variety of conditions. Data augmentation can make convolutional neural network robustly classifying objects even if they are placed in different conditions.

The configuration file enables the user to randomly rotate the given image by \pm angle with the angle parameter. To modify the colours of the inputs the parameters saturation, exposure and hue can be adjusted to improve the network robustness.

The config file doesn't have only all the network hyperparameters that were previously explained. It also contains the network structure such as all its layers and here is the difference between the yolov3.cfg and yolov3-tiny.cfg. This structure can also be remodelled. The difference between the two structure can be visualized in figure 3.15

Type	Filters	Size	Output	Layer	Type	Filters	Size/Stride	Input	Output
Convolutional	32	3 × 3	256 × 256	0	Convolutional	16	3 × 3/1	416 × 416 × 3	416 × 416 × 16
Convolutional	64	3 × 3 / 2	128 × 128	1	Maxpool		2 × 2/2	416 × 416 × 16	208 × 208 × 16
Convolutional	32	1 × 1		2	Convolutional	32	3 × 3/1	208 × 208 × 16	208 × 208 × 32
Convolutional	64	3 × 3		3	Maxpool		2 × 2/2	208 × 208 × 32	104 × 104 × 32
Residual			128 × 128	4	Convolutional	64	3 × 3/1	104 × 104 × 32	104 × 104 × 64
Convolutional	128	3 × 3 / 2	64 × 64	5	Maxpool		2 × 2/2	104 × 104 × 64	52 × 52 × 64
Convolutional	64	1 × 1		6	Convolutional	128	3 × 3/1	52 × 52 × 64	52 × 52 × 128
Convolutional	128	3 × 3		7	Maxpool		2 × 2/2	52 × 52 × 128	26 × 26 × 128
Residual			64 × 64	8	Convolutional	256	3 × 3/1	26 × 26 × 128	26 × 26 × 256
Convolutional	256	3 × 3 / 2	32 × 32	9	Maxpool		2 × 2/2	26 × 26 × 256	13 × 13 × 256
Convolutional	128	1 × 1		10	Convolutional	512	3 × 3/1	13 × 13 × 256	13 × 13 × 512
Convolutional	256	3 × 3		11	Maxpool		2 × 2/1	13 × 13 × 512	13 × 13 × 512
Residual			32 × 32	12	Convolutional	1024	3 × 3/1	13 × 13 × 512	13 × 13 × 1024
Convolutional	512	3 × 3 / 2	16 × 16	13	Convolutional	256	1 × 1/1	13 × 13 × 1024	13 × 13 × 256
Convolutional	256	1 × 1		14	Convolutional	512	3 × 3/1	13 × 13 × 256	13 × 13 × 512
Convolutional	512	3 × 3		15	Convolutional	255	1 × 1/1	13 × 13 × 512	13 × 13 × 255
Residual			16 × 16	16	YOLO				
Convolutional	1024	3 × 3 / 2	8 × 8	17	Route 13				
Convolutional	512	1 × 1		18	Convolutional	128	1 × 1/1	13 × 13 × 256	13 × 13 × 128
Convolutional	1024	3 × 3		19	Up-sampling		2 × 2/1	13 × 13 × 128	26 × 26 × 128
Residual			8 × 8	20	Route 19 8				
Avgpool		Global		21	Convolutional	256	3 × 3/1	13 × 13 × 384	13 × 13 × 256
Connected		1000		22	Convolutional	255	1 × 1/1	13 × 13 × 256	13 × 13 × 256
Softmax				23	YOLO				

Figure 3.15: Comparison between the YOLOv3 (left) and YOLOv3_tiny (right) structure

The next step is the creation of the obj.names file. This file contains, respectively, the name of each of the traffic sign/light. The names are the labels for each object.

Another file needs to be created, the obj.data. This file contains information for the training in the following format:

```
classes = <number of classes>
train = <directory of the training dataset>
valid = <directory of the validation dataset>
names = <directory of the obj.names file>
backup = <directory in which the network will create a backup>
```

The Darknet framework concedes users the ability to make a backup after the train starts in case the training process is stopped to continue the train with the last saved weights. This backup stores the last saved weights in the provided directory. For both of the chosen YOLOv3 versions, a copy is made in Google Drive of the config file and the obj.names file. These files are required to restart the training.

The following step consists of uploading and allocating the dataset to the directories defined in the obj.data file.

CNN's like YOLO, require loads of images to train. These images must have lots of variations to make the network robust enough (variations such as lighting conditions, viewing angles, distance from the camera, etc.).

A popular option, which is used in the YOLO framework, is to use transfer learning where a pre-trained model is used, this allows the user to train a new model starting from that point. Transfer learning is the improvement of learning in a new task through the transfer of knowledge from a related task that has already been learned. The intention is that the first few layers have already learnt what kind of features to extract and what to expect from the images in the custom dataset. In the YOLO framework, these pre-trained weights can be downloaded (with the corresponding config file) to detect and classify with the COCO dataset that has 80 classes. In this objective, the pre-trained weights are employed, using transfer learning, to train the previously referenced, in section 3.2.1, custom dataset. The pre-trained weights for the YOLOv3 and YOLOV3_tiny are different. For the YOLOv3 the command used is:

```
!wget https://pjreddie.com/media/files/darknet53.conv.74
```

And for the YOLOV3_tiny was:

```
!wget https://pjreddie.com/media/files/yolov3-tiny.weights
```

Finally, after all the previously explained steps are fulfilled, the train of the network can be initiated with the following command template:

```
!.<directory of darknet> detector train <directory of obj.data>
  <directory of config file> <directory pre-trained weights>
  -dont_show
```

An example of this command can be:

```
!./darknet detector train data/obj.data cfg/yolov3_training.cfg
  yolov3-tiny.weights -dont_show
```

Following this command, the network will be loaded and the train started. For each batch, an output will be presented. The following is an example of this output:

```
11292: 0.161011, 0.172331 avg loss, 0.001000 rate, 3.760432 seconds,
  722688 images, 62.986576 hours left
```

Where,

11292: Current number of the iteration of the current training;

0.161011: Overall Loss;

0.172331 avg loss: Average Loss;

0.001000 rate: Current learning rate;

3.760432 seconds: Total time spent on the current batch training;

722688 images: Total number of pictures that have been trained so far;

62.986576 hours left: Estimated training time remaining.

Using the validation dataset the network will be tested and in each test the mAP will be calculated. An example of the output for every test is:

```
class_id = 0, name = Left Direction, ap = 100.00% (TP = 4, FP = 1)
class_id = 1, name = Roundabout, ap = 98.48% (TP = 11, FP = 1)
class_id = 2, name = Lights, ap = 100.00% (TP = 49, FP = 0)
(...)
class_id = 29, name = Traffic Light Finish, ap = 100.00% (TP = 31,
  FP = 0)
```

```
for conf_thresh = 0.25, precision = 0.93, recall = 0.98
for conf_thresh = 0.25, TP = 1909, FP = 138, FN = 43, average IoU =
    72.21 %

mean average precision (mAP@0.50) = 0.965636, or 96.56 %
```

Where,

class_id = Correspondent label number

name = Correspondent label

ap = Average Precision

TP = True Positives

FP = False Positives

FN = False Negatives

conf_thresh = Confidence Threshold

mAP@0.50 = Mean Average Precision with a 0.5 IoU thresholds

The weights are saved every 100 batches. Since a backup directory was provided in the obj.data file, the weights are saved in this directory in the yolov3_training_last.weights file. Every test performed will calculate an mAP. Every time the mAP reaches the highest value until that point, the weights are saved in the yolov3_training_best.weights in the backup directory. At the end of the train, the weights are saved in the yolov3_training_final.weights file in the backup directory.

As previously explained, the Darknet framework concedes users the ability to recurrently make a backup after the train starts in case the training process is stopped. If this happens the user can proceed with the train with the following command template using the last saved weights:

```
!<directory of darknet> detector train <directory of obj.data>
    <directory of config file> <directory of the last saved weights in
    the backup> -dont_show
```

An example of this command can be:

```
!./darknet detector train data/obj.data cfg/yolov3_training.cfg
    /mydrive/v3_tiny_comp/yolov3_training_last.weights -dont_show
```

3.2.3 Testing

The test part is where, after the network is trained, the frames from the camera are being successively processed by the network which provides the location in the frame of the bounding boxes from the identified traffic sign/light and the confidence for each one.

To perform the test of the classification and location of the traffic signs/lights in the input frames, a Python script was developed using Pycharm Community 2021.1 as the IDE.

3.2.3.1 Testing Script

The first step of this process is loading the network. This step uses the OpenCV library by loading the best weights and the config files that are created in the train, as was explained in the previous section.

The next step is a cycle that gets the input frame, which can be from a camera, video or image, and sends it to the network. After the frame goes through the network, the resulting detected traffic lights/signs are obtained. With the information from each detected sign/light, and using OpenCV, the bounding boxes are drawn in the image and over them a text with the label and confidence of the detection. When using the network in an application a fixed confidence threshold was introduced to filter the detections to only the confident ones. This limits the detected bounding boxes to avoid miss detection. Depending on the confidence score of the detection, the colour of the bounding box varies in order to make the confidence more intuitive for the user.

One feature that was introduced in the network is an FPS counter that allows the user to verify the performance of the network. This counter is also displayed in the output image to make it user friendly when enabled.

The fps counter corresponds to the amount of frames that are processed in 1 second. This value can be determined by calculating the time that takes the frame to be processed. To perform this operation using the Time library, the time at the moment that the frame is obtained (t_1) is saved and also the time after the image is processed (t_2). The difference between these two values, t_1 and t_2 , provides the time that the script takes to process the frame. It is measured with the following formula, in milliseconds:

$$time_of_process = t_2 - t_1 \quad (3.1)$$

With the value of the time that the script takes to process the frame, the fps, that corresponds to the frequency, can be calculated with the following formula:

$$fps = \frac{1000(ms)}{time_of_process(ms)} \quad (3.2)$$

In the previous formula, the 1000 value corresponds to the conversion from milliseconds to seconds.

This script is used in the three objectives described in section 3.1. Its application changes in each one with the three corresponding files that are loaded: the weights and config file for the network and the labels file. For each objective three types of frame inputs are accepted: image, video and camera. This script will be referenced in the following two objectives.

3.2.3.2 Testing Framework

The network was tested twice:

- First Test

The first test had the purpose of evaluating how the network correctly detected the TS/L (traffic signs/lights). Videos were made in different scenarios from the dataset ones, referred to in section 3.2.1, so that it was possible to know how the network would behave in different conditions and to examine its robustness. This test did not involve the car or the ROS communication, only the TS/L (traffic signs/lights) detection and classification. Using the testing script, previously explained in section 3.2.3.1, and using the resulting weights, config and labels files from the train, the video frames were processed and the output analysed in order to visualize if the objects were correctly identified.

In figure 3.16, the output of the testing script is presented in two different frames from the videos.

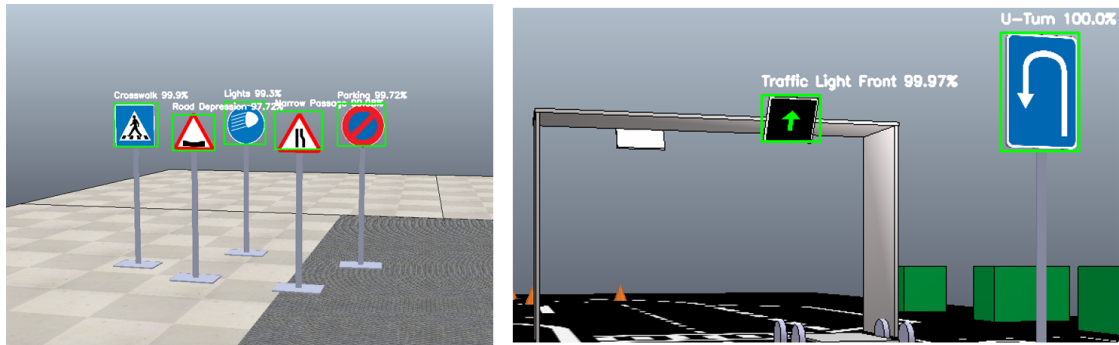


Figure 3.16: Images from two different frames from the test

- Second Test

The second test is a more realistic simulation of the competition, in which the network detects the possible TS/L that would be on the track. The main goal is the detection of the TS/L and with this information, the car will act depending on the traffic sign.

Not all TS/L were chosen to assign an action because their sign/light meaning does not involve movement. The S/L and their correspondent actions are placed in the following list:

- STOP, Yield and Prohibited Direction: stop the motion
- Traffic Light Front: start of forwarding motion
- 30km/h, 40km/h and 60km/h: change speed to traffic signal value
- Left Obligation: turn 45 degrees left
- Right Obligation: turn 45 degrees right
- U-Turn: turn 180 degrees

In figures 3.17, 3.18 and 3.19, the output of PyCharm and CoppeliaSim are presented. All the communications in the software are carried out using ROS as previously explained.

On each imagesleft side can be seen the processed frame that is sent from CoppeliaSim to PyCharm and processed using the script referenced in section 3.2.3.1. This script detects and classifies the present sign in the frame. The frame label is returned to CoppeliaSim so that the robot can perform the corresponding action.

On the figures right side there is CoppeliaSim simulator in which it is possible to view the robot movement. At the bottom of this simulator, the commands are presented in which the speed is presented in real time.

Initially, the robot waits for the detection of the front traffic light to start its forward motion, as seen in figure 3.17.

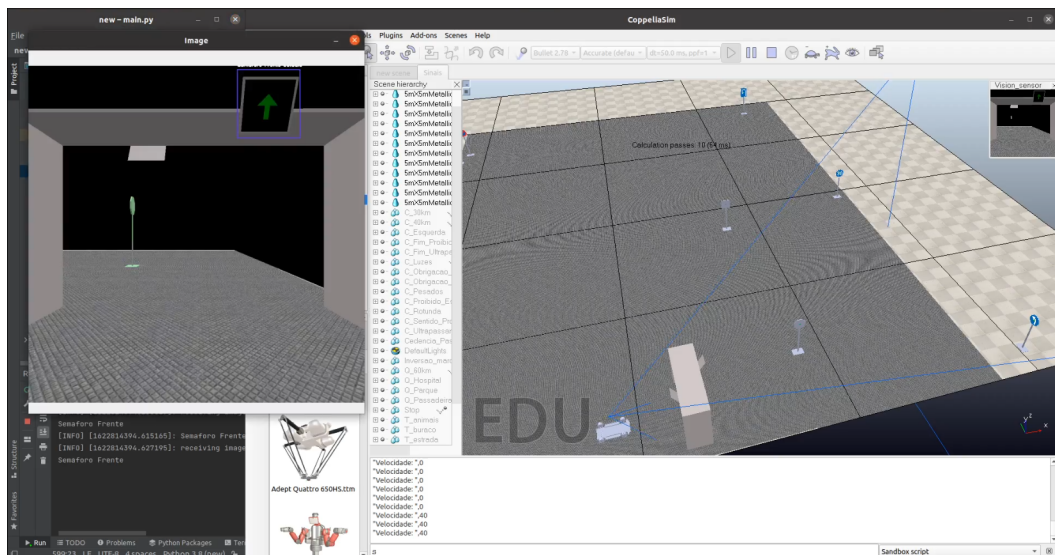


Figure 3.17: Start of the test

From this point, the robot moves forward until it detects another signal, according to which it reacts. When it detects a speed signal the robot changes its speed depending on the sign value.

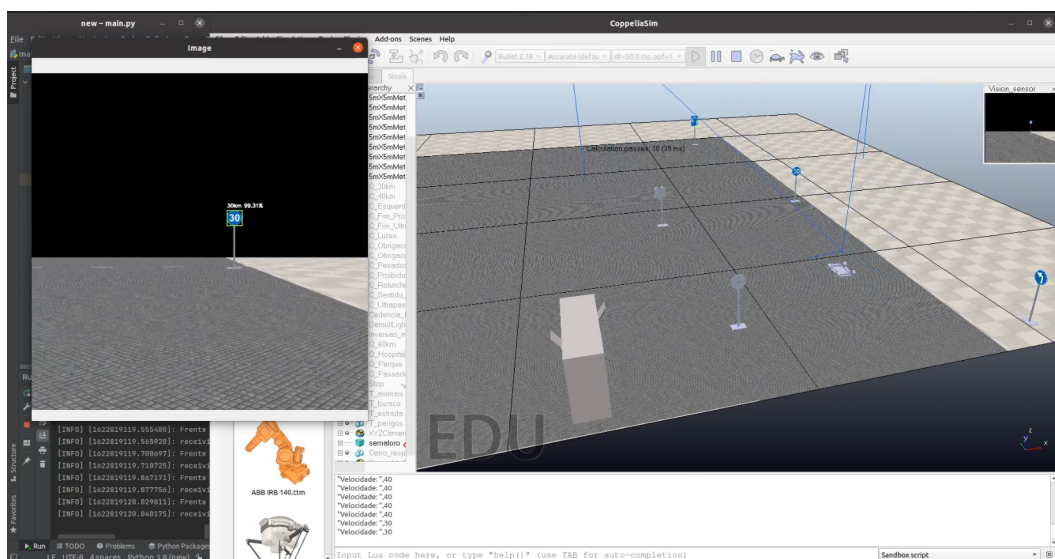


Figure 3.18: Speed detection

The robot continues to react to the detected S/L until it reaches a sign with a stop action.

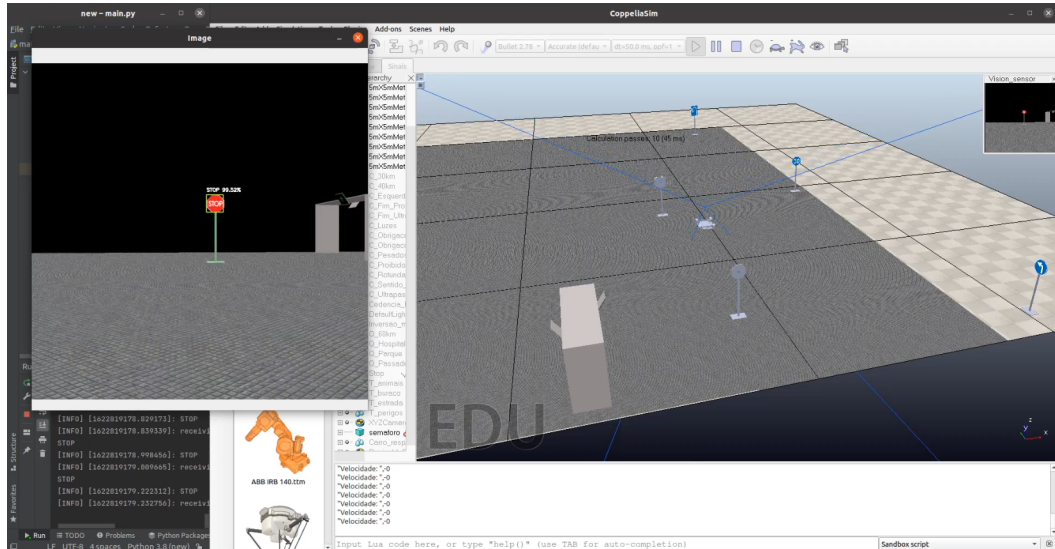


Figure 3.19: End of the test after the STOP detection

3.3 Autonomous Driving Competition of the RoboCup Portuguese Open

The second objective of the project is the development of a detection and classification algorithm for the Autonomous Driving Competition of the RoboCup Portuguese Open. Unlike the objective described in section 3.2 this objective is not in simulation but in a real competition context. In this section, the three phases are described in order to accomplish the objective. As in section 3.2, for the autonomous driving competition of the RoboCup Portuguese Open, the network has the goal of correctly identifying 24 traffic signs and 6 traffic lights which are listed in table 3.2.

Following the competition guidelines and to obtain the images for the dataset, identical TS/L were constructed to mimic the environment that would be in the competition.

Using images from *Autoridade Nacional de Segurança Rodoviária*, which has the mission of planning and coordination at a national level to support the Government's policy in the field

3.3.0.1 Distance to the Traffic Sign/Light software

During the competition, the robot needs to know how far away are the TS/L so that with this information it knows the ideal moment to react to the sign. Given this problem, a Python script was developed that, starting from the Bounding Boxes area detected in the frame, could determine the distance to the TS/L.

The first step was to create a relationship between the Bounding Box area and the distance to the camera. For this, a large number of values of the Bounding Box area resulting from the neural network were registered while increasing the distance between the signal and the camera.

After obtaining this data, it was entered into Microsoft Excel and from this, a regression was performed to determine the relationship between the Bounding Box area and the distance to the camera. The equation resulting from this regression was: $39757.381 x^{-0.597}$

$$Distance = 39757.381 * Area^{-0.597} \quad (3.3)$$

where,

Distance = Distance from the TS/L to the camera

Area = Area of the Bounding Box

In figure 3.22, it is possible to verify in black the line of the determined equation and in orange the points used to perform the regression.

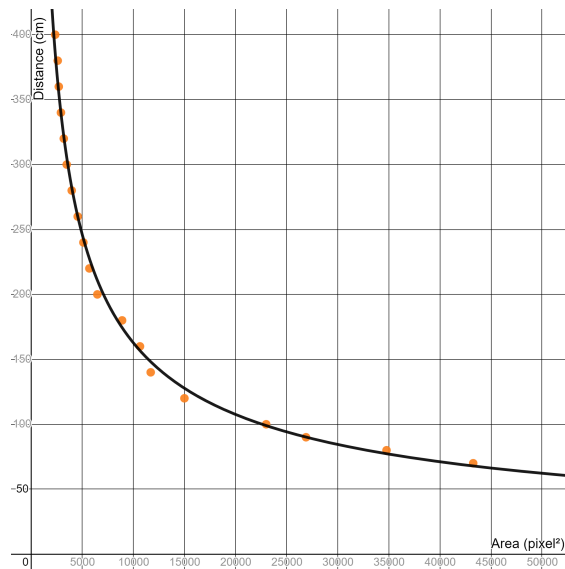


Figure 3.22: Graph with the distance from the T S/L to the camera

Using the Bounding Box area proved to be the best method compared to other tested variables such as Bounding Box height and width individually.

3.3.1 Acquisition

The acquisition phase of the second objective (section 3.2.1) has the goal of creating a dataset with images from all the traffic signs and traffic lights in order to train the networks. Using the same methods referenced in the acquisition phase from the previous objective, videos of the TS/L were performed with the same goal.

Although the previous objective is in simulation, in the second objective the diversity of traits is approximately the same:

- Background: Using different scenarios in the *Laboratório de Automação e Robótica* facilities, varying the colour, textures and objects such as robots, tables and chairs;
- Number of signs/lights: videos with one or multiple;
- The angle of the camera: ranging from a top, bottom or side view;
- Distance to the camera: recorded from near or far away.

In the following figures, the variety of previously explained traits is demonstrated for the U-Turn sign in the images from the dataset.

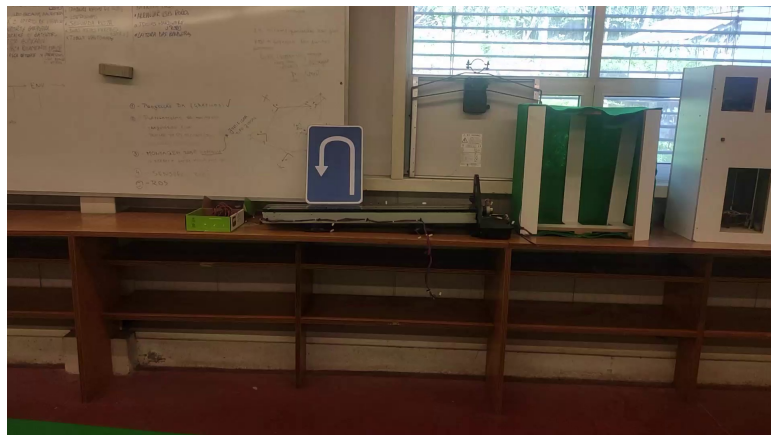


Figure 3.23: First Example

In figure 3.23 the image only has the U-Turn sign, it is at an average distance to the sign and the brightness is lower than normal.



Figure 3.24: Second Example

Unlike figure 3.23, in figure 3.24 the image has more than one traffic sign, has a shorter distance and is brighter.

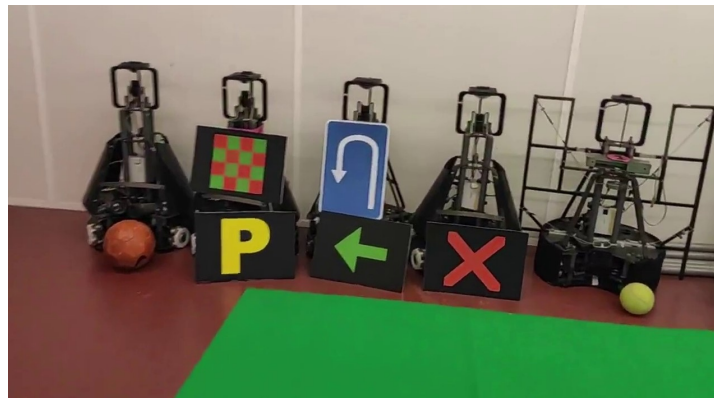


Figure 3.25: Third Example

The main difference between figure 3.24 and 3.25 is that figure 3.25 has more objects in the background and is blurrier.



Figure 3.26: Fourth Example

In figure 3.26 there is a greater amount of signs and these are further away. This figure has the most objects in the background.

The Xiaomi Mi 9 SE smartphone was used to record the videos with 1080 resolution and with 30 fps. The smartphone was used due to its camera stabilization.

The smartphone was used to obtain the videos because it would emulate the conditions in which the network would be tested. This method was also chosen because not all frames would be in focus and this would help the final network to be more robust in case the input has the same conditions. If the network is trained with some blurrier images it will perform better in the testing phase.

Using Python as the main language and Pycharm Community 2021.1 as the IDE, the same script described in section 3.2.1 was used to convert the video frames into images. A different ratio of 1 in every 3 frames was used due to the videos having half of the FPS. This allowed the user to have always different images. With this frame rate, the script generated 5 images every second instead of the 10 obtained with a 60 fps input.

The final video has 9 minutes and 54 seconds and using the script as in the previous objective, 5949 images were created. In table 3.4, the number of images in which the signs/ lights appear is presented.

Traffic Sign/Light	Number of images
Left Direction	362
Roundabout	372
Lights	440
Public Transport	436
60km/h	459
Hospital	440
Car Park	433
Crosswalk	435
Animals	395
Road Depression	390
Narrow Passage	371
Dangers	361
Yield	420
STOP	387
U-Turn	415
Overtaking	383
Prohibited Direction	367
30km/h	450
40km/h	377
Parking	386
End Overtaking	454
End Prohibitions	354
Right Obligation	418
Left Obligation	399
Traffic Light Right	281
Traffic Light Front	424
Traffic Light Left	461
Traffic Light Park	262
Traffic Light Stop	363
Traffic Light Finish	378

Table 3.4: Number of images per Traffic Sign/Light

Regarding the associated text file to the images, the same format referred to in section 3.2.1 is used since the network used in this objective will be the same as the last one.

Unlike the acquisition phase in the previous objective in which the labels were manually performed, in this one, most of the labels were deployed using a developed Python script using the Template Matching function from OpenCV. Template Matching is a method for searching and finding the location of a template image in a larger image, it slides the template image over the input image returns all the detections with the corresponding confidence. For each TS/L, a template was generated. To improve detection, this template must have a black background with

black colour. This background change was performed using the Windows Paint 3D tool where the sign was selected and the remaining background painted black. This process of painting the background black proved to be the best and black the best colour from the ones tested since identified the objects most frequently. In figure 3.27, as an example, the template for the Public Transport sign is presented.



Figure 3.27: Template

With this template, the Template Matching function can be applied to the generated images and the output is an array with the locations of the detections and the corresponding confidences. To explain this process an input image is presented in figure 3.28.



Figure 3.28: Input

The Template Matching function is performed using the following template:

```
<output> = cv2.matchTemplate(<input image>,<template image>,<method>)
```

where the <method> variable corresponds to the method used for searching and finding the location of the template in the input image. The method that had the best results from all the 6 methods was cv2.TM_CCOEFF_NORMED since identified the objects most frequently.

The array returns multiple detections in the input image and in order to prevent this, the final detection was the one that had the highest confidence score.



Figure 3.29: All the detections with confidence scores over 40%

In figure 3.29, all the detections with confidence scores over 40% are represented. In this figure, all the detections are overlapping and the other round and blue sign is also detected. The final detection with the highest confidence score is represented in the following figure and correctly identifies the sign. One of the disadvantages found using this method is that the bounding box does not get placed exactly, leaving some gaps. For example, in this figure the bounding box leaves a small gap on the right side of the sign.



Figure 3.30: Detection with the highest confidence score

The Template Matching function returns, for each detection, two points to define the bounding box (the top left and the bottom right) and the corresponding confidence. In figure 3.31, these two points are represented in the previous example.

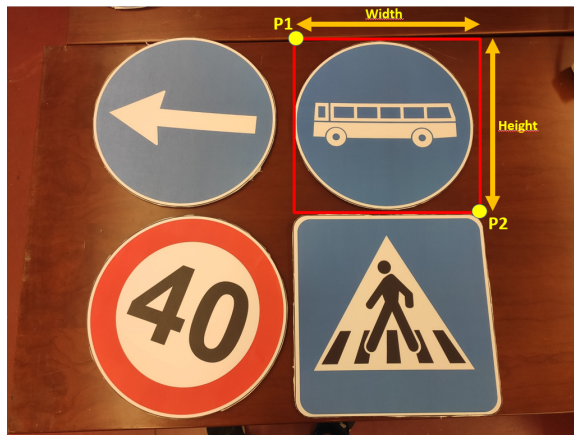


Figure 3.31: Information returned using the Template Matching function

The YOLO network requires a different format, as was explained in section 3.2.1. To perform the conversion the following equations were used:

$$x = \frac{x_{P1} + x_{P2}}{2} \quad (3.4)$$

$$x_{YOLO} = \frac{x}{width} \quad (3.5)$$

$$y = \frac{y_{P1} + y_{P2}}{2} \quad (3.6)$$

$$y_{YOLO} = \frac{y}{height} \quad (3.7)$$

$$width_{YOLO} = \frac{x_{P2} - x_{P1}}{width} \quad (3.8)$$

$$height_{YOLO} = \frac{y_{P2} - y_{P1}}{height} \quad (3.9)$$

Using this method, all the images that were not distant were labelled and this process was automatic. This method had extremely poor performance when deployed in images where the TS/L were distant. For the images in which the TS/L were distant, for example, in figure 3.26 all the TS/L were manually inserted using Labellmg.

Using these methods all the images were labelled and the dataset was achieved for the train.

3.3.2 Training

Since the training phase of the second objective consists of the development of two networks (YOLOV3 and YOLOV3_tiny), just like it was for the first objective, all the steps remain the same as explained in section 3.2.2. The only difference is the dataset, described in section 3.3.1, in which the train is performed.

3.3.3 Testing

The main purpose of this objective was participation in the Autonomous Driving Competition of the RoboCup Portuguese Open which would be the ultimate test for the developed networks but due to the coronavirus pandemic, the competition did not take place. To check the performance of the developed networks the testing script described in section 3.2.3.1 was used. Using an A4Tech PK-910H Webcam which provided 1080p (1920x1080 pixels) resolution as an input, videos were created and the script was tested.

3.4 Public road

The third objective of the project is the development of a detection and classification algorithm for Road TS/L. In this section, all the phases are described to accomplish this objective. These do not follow the same steps as the other two objectives. The steps taken to accomplish this objective are explained in this section and are represented in figure 3.32.

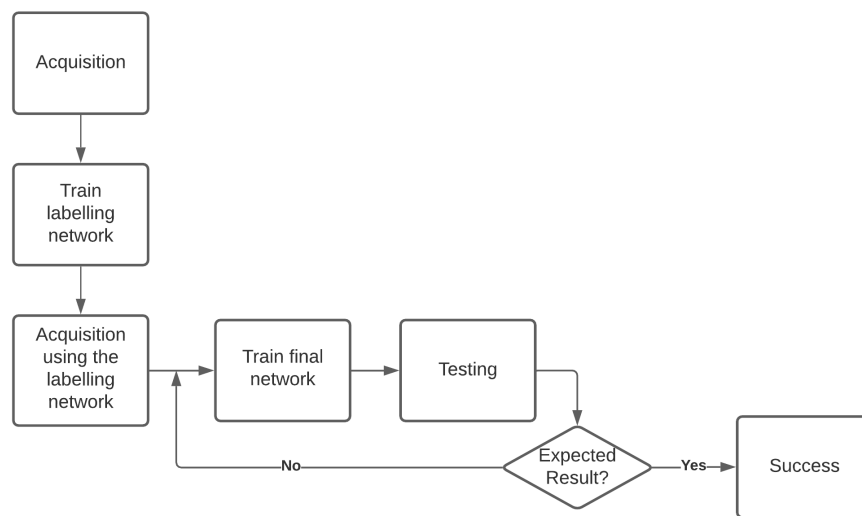




















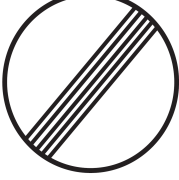









Figure 3.32: Steps taken to accomplish this objective

As in previous objectives, a dataset needed to be created with a determined number of TS/L. These TS/L were chosen if they were the most commonly detected while recording in two different cities: Braga and Guimarães. These TS/L are represented in table 3.5.

ID	Traffic Sign/- Light	Image	ID	Traffic Sign/- Light	Image
0	10km/h		1	30km/h	
2	40km/h		3	50km/h	
4	60km/h		5	70km/h	
6	80km/h		7	90km/h	
8	100km/h		9	120km/h	
10	30km/h Obliga- tion		11	40km/h Obliga- tion	
12	Prohibited Direc- tion		13	Obligation Arrow	

ID	Traffic Sign/- Light	Image	ID	Traffic Sign/- Light	Image
14	Prohibition Turn		15	Roundabout	
16	Lights		17	Overtaking	
18	Parking		19	STOP and Park	
20	End Prohibitions		21	Danger Cross-walk	
22	Traffic Conges- tion		23	Danger Round- about	
24	Junction		25	Curve	
26	Narrow Passage		27	Road Hump	









ID	Traffic Sign/- Light	Image	ID	Traffic Sign/- Light	Image
28	Crosswalk		29	Car Park	
30	Tunnel		31	STOP	
32	Traffic light		33	Yield	
34	Directional Sign		35	50km/h Obliga- tion	

Table 3.5: The Traffic Signs/Lights in the Road Dataset

3.4.1 Acquisition

With the goal of acquiring images for the Road Dataset, the methodology chosen was similar to the one used in the previous objective (section 3.3.1). The main goal of the acquisition phase was obtaining multiple images from the TS/L from different scenarios. To make the network more robust the TS/L needed to be in different locations in which the following characteristics would change:

- Background: the TS/L are placed in different environments and subsequently all of them have different backgrounds;
- Angle: the TS/L can have different angles;
- Distance: the TS/L can be at different distances;

- Brightness: over the course of the day, the incidence of brightness will vary on TS/L.

Videos were recorded to accomplish the acquisition of the images with the previously referred characteristics. These videos were recorded from the front passenger seat in the car as demonstrated in figure 3.33. In this figure, the blue shape corresponds to the zone that was being recorded.



Figure 3.33: Zone recorded

The videos were recorded on a Xiaomi Mi 9 SE smartphone. This smartphone was chosen due to the camera stabilization to reduce the blur of videos.

These videos were recorded on multiple days and at different hours. The videos also were recorded during nighttime to ensure the network performs all day. The paths also vary in order to find the least common TS/L in the dataset and to have different characteristics in the images.

The videos were joined and a final video was created. The final video has 15 minutes and 30 seconds and using the script from the other objectives, 4648 images were created. In the following table 3.6, the number of images in which the T S/L appear is presented.

Traffic Sign/Light	Number of images
10km/h	6
30km/h	58
40km/h	248
50km/h	476
60km/h	155
70km/h	286

80km/h	67
90km/h	135
100km/h	38
120km/h	110
30km/h Obligation	38
40km/h Obligation	67
Prohibited Direction	401
Obligation Arrow	1057
Prohibition Turn	414
Roundabout	652
Lights	91
Overtaking	132
Parking	51
STOP and Park	80
End Prohibitions	58
Danger Crosswalk	299
Traffic Congestion	15
Danger Roundabout	182
Junction	281
Curve	186
Narrow Passage	52
Road Hump	140
Crosswalk	1044
Car Park	113
Tunnel	41
STOP	367
Traffic light	700
Yield	1162
Directional Sign	490
50km/h Obligation	153

Table 3.6: Number of images per Traffic Sign/Light

In table 3.6 the values from each TS/L fluctuate a lot which is representative of their quantity on the roads where the videos were recorded.

In the following images examples from the dataset are presented with different characteristics.



Figure 3.34: Rainy day image

In figure 3.34 the image has a low level of brightness due to the rain and there are multiple traffic signs at different distances.



Figure 3.35: Bright day image

Figure 3.35 is an example of an image in which the traffic signs are at a greater distance and the image is bright.



Figure 3.36: Night image

Figure 3.36 is an example in which the images have the lowest brightness because they were recorded at night.

After the images were generated, the corresponding labels needed to be produced. The method of using the Template Matching function from the OpenCV library, as described in section 3.3.1, was not used to generate images because the distance of the images to the TS/L was too high for the function to properly work.

The Labelling software, as in section 3.2.1 was used to label every image. This method was not the most time-efficient because the process took several days but it was the one that guaranteed the most accuracy from the ones that were tested.

3.4.2 Train Labelling Network

Due to the immense time needed to label images with the methodology used in section 3.4.1, a network was created with the already labelled images as input. This network will label more images to make the final network more complete and robust.

To create this network for labelling, a YOLOV3 network was created just like it was for the first objective, all the steps remain the same as explained in section 3.2.2. The only difference is the dataset described in section 3.4.1, in which the train is performed. For this network, only the YOLOV3 network was created, and not the YOLOV3_tiny, because the accuracy from YOLOV3 was required to label the images.

3.4.3 Acquisition Using The Labelling Network

The number of videos used to train the previously described network is not enough because the network is not as accurate as it should be. This network is used to speed up the labelling process for the final network.

After acquiring more videos in the same conditions referenced in section 3.4.1, the videos were joined, which resulted in a 12 minutes and 23 seconds video. Using the script explained in the previous objectives, 3686 images were created. These images were processed in the network that was developed in section 3.4.2 and the labels for each sign that the network outputs are saved with the YOLO format in a text file with the same name of the image.

Since this network is not the final one, the accuracy is not the desired one and every label needed to be checked. Figure 3.37 is an example of this process, the network correctly detects 3 traffic lights and 2 traffic signs but 1 traffic light is not detected. This is why every image needs to be checked. Even though all the labels were checked and the Bounding Boxes adjusted, this method proved to be faster than doing the labels manually. With this method, all the labels for the final dataset were performed.



Figure 3.37: Example of Acquisition using The Labelling Network

3.4.4 Train Final Network

With the final dataset ready for the Public Road network the train was performed. As in previous sections two networks were developed (YOLOV3 and YOLOV3_tiny). All the steps remain the same as explained in section 3.2.2. The only difference is the dataset, described in section 3.4.3, in which the train is performed.

3.4.5 Testing

The main objective is the correct detection and classification of the traffic signs that are being captured by a camera from a moving car. To replicate this process and has was performed in the acquisition phases, images were recorded on a Xiaomi Mi 9 SE smartphone and then these frames were processed in the testing script, explained in section 3.2.3.1, to evaluate how accurate the network is.

Chapter 4

Tests

This chapter introduces the numerous tests performed on the two networks (YOLOV3 and YOLOV3_tiny) that were carried out to determine the best parameters that represent the best results in terms of performance and accuracy.

4.1 Frame Rate

The first tests that were performed consisted of testing the performance with different input dimensions. The tests were performed on both networks (YOLOV3 and YOLOV3_tiny) using the testing script referenced in section 3.2.3.1 and the value of the FPS was monitored.

As was mentioned in section 2.8.2.1, the YOLO architecture can perform with multiple scales which can adopt an image dimension size that is multiple of 32 to make it more robust. The smallest choice is 320x320 and the most extensive 608x608.

Figure 4.1 corresponds to the performance results for the YOLOV3 network and figure 4.2 corresponds to the performance results for the YOLO_tiny network.

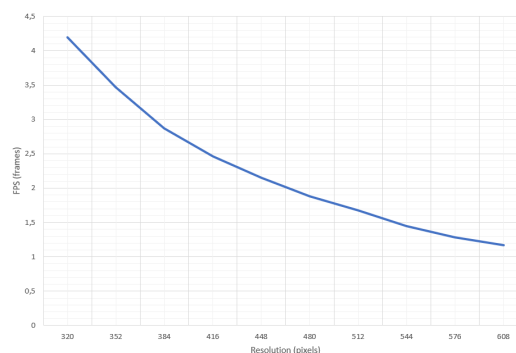


Figure 4.1: YOLOV3 performance with different input dimensions

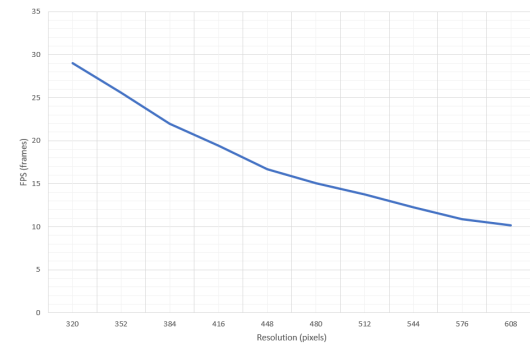


Figure 4.2: YOLO_tiny performance with different input dimensions

In both of the networks, the frame rate decreases when the input size increases. With a higher resolution, the networks have to process more pixels and this results in a higher processing time which lowers the FPS.

As was explained in section 2.8.3.3, the YOLO_tiny network ensures a higher frame rate compared to YOLOV3. With the lowest input value (width = 320 and height = 320), the YOLO_tiny network has 29,03 fps in comparison to the 4,19 fps obtained in YOLOV3. On the other hand, accuracy is where YOLO_tiny has lower values but this was tested in the following sections.

Following the values provided by the YOLOV3 creators, the resolution used in the following tests are width = 416 and height = 416. This way there is a balance between the accuracy and the FPS.

4.2 Hyperparameters

For each of the three objectives explained in section 3, two networks were developed (YOLOV3 and YOLOV3_tiny). In each training phase the hyperparameters needed to be optimized to obtain the best performance.

The YOLO architecture provides initial values for these hyperparameters and these are tested, in this section, in comparison to higher or lower values.

In the following sections, the output of three tests with different values from the corresponding hyperparameter is going to be compared. The output values compared are the mAP and the Loss

because they provide information on how the test performed. To facilitate the visual comparison between the tested values, a graph was generated for the mAP and another for the loss. This allows visual analysis of how the network performed over the train.

4.2.1 YoloV3

In this section, the values for the hyperparameters in Table 4.1 are tested in the YOLOV3 network and the results are presented.

Hyperparameter	Test 1	Test 2	Test 3
Max Batches	9500	19000	28500
Learning Rate	0.01	0.001	0.0001
Momentum	0.2	0.45	0.9
Burn in	500	1000	1500
Decay	0.005	0.0005	0.00005
Width x height	320x320	416x416	544x544

Table 4.1: The hyperparameters and the corresponding tested values

4.2.1.1 Max Batches

The first hyperparameter tested was max_batches, which defines the number of iterations that the train lasts. This architecture requires a higher number of iterations due to its multi-class detection and usually, the hyperparameter is set by (average number of images per class)*(number of classes). The two generated graphs are granted in figure 4.3 and 4.4.

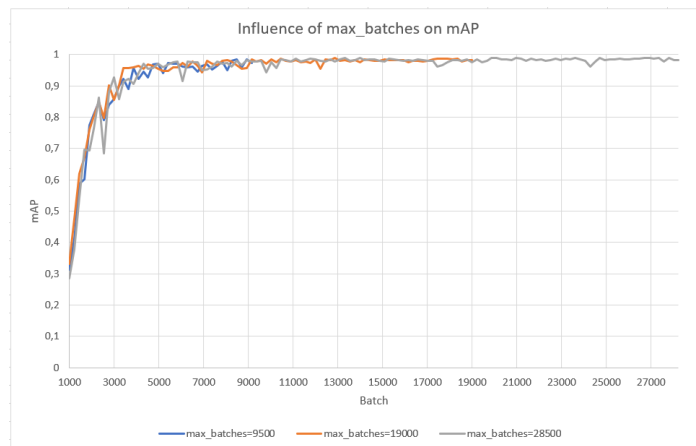


Figure 4.3: Influence of max_batches on mAP

In figure 4.3, the outputs of the variation of the mAP are displayed. Each line corresponds to a train performed with a different value of maximum iterations. Through the train, the best mAP result is disputed between the `max_batches=19000` line and `max_batches=28500` line. The `max_batches=19000` line reaches its best mAP value at batch 12880, `mAP=0,98804` or 98,804%, while the `max_batches=28500` one achieves a slightly higher value in batch 13980, `mAP=0,99023` or 99,023%. In the rest of the train, neither test achieves better values.

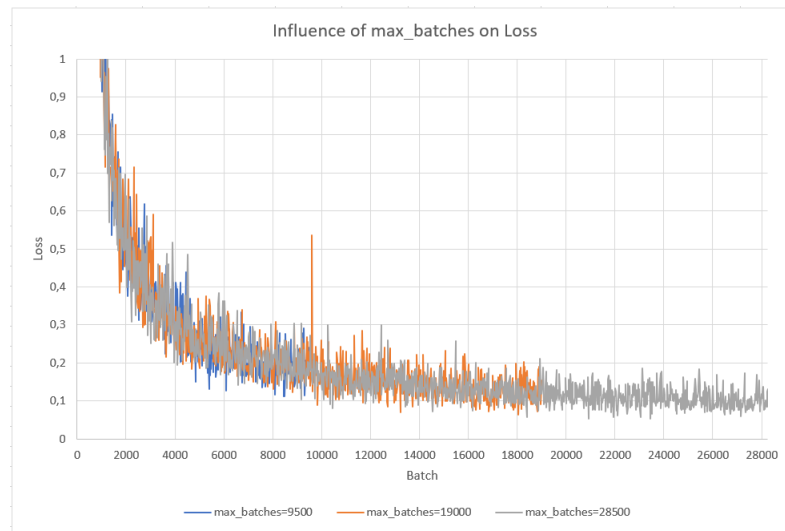


Figure 4.4: Influence of `max_batches` on Loss

Analysing figure 4.4, the output variation of the loss is represented. Each line corresponds to a train corresponding to the one presented in figure 4.3. It is possible to verify that the `max_batches=9500` line did not obtain enough iterations to acquire the lower values reached by the other two trains. The `max_batches=19000` and `max_batches=28500` lines had similar values during the train but after the `max_batches=19000` one finished, the `max_batches=28500` one maintained its value.

Max_batches value	Time the train lasted
9500	8 hours 47 minutes
19000	18 hours 6 minutes
28500	28 hours 19 minutes

Table 4.2: Time each train lasted

With these two tests and the data in table 4.2, it is possible to conclude that the `max_batches=9500` line did not achieve the values obtained by the other lines. It is also noticeable that the extra computational power used for the extra iterations in the `max_batches=28500` line did not result in sufficient enhancements in comparison to the `max_batches=19000` one.

4.2.1.2 Learning Rate

Learning rate is a hyper-parameter used to control the rate at which an algorithm updates the weights. The two generated graphs are presented in figure 4.5 and 4.6.

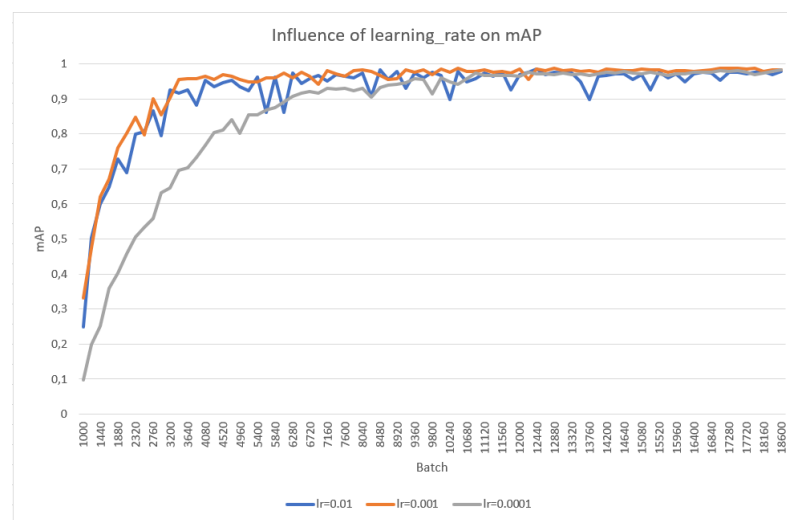


Figure 4.5: Influence of learning_rate on mAP

Analyzing figure 4.5, it is possible to verify that the `lr=0.001` output is the one that reaches the maximum mAP of 0.98804, or 98.804%, and is the value that throughout the train has regularly the highest mAP. The `lr=0.0001` line takes more iterations to reach approximate values comparatively to the other lines but never overcomes these. The `lr=0.01` line is the one in which the values float more which is a result of the high learning rate value.

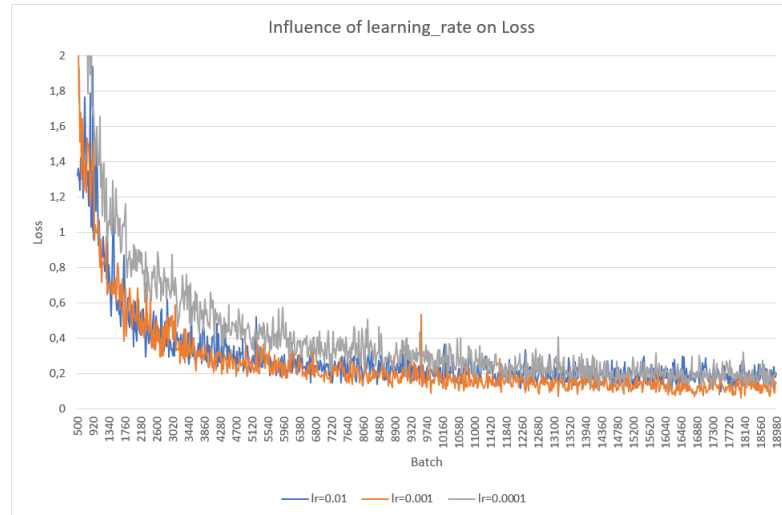


Figure 4.6: Influence of learning_rate on Loss

Examining figure 4.8, the same conclusion is found regarding the best learning rate value. Through the train, the lr=0.001 output has regularly the lowest loss.

Consequently, the effects of the learning rate hyperparameter are presented. Higher values of learning rate lead to a high fluctuation in mAP and low values lead to a slow rise in this metric. It can be concluded that the best value for the learning rate hyperparameter is 0.001.

4.2.1.3 Momentum

The YOLOV3 architecture requires a high number of iterations due to its multi-class detection and in each iteration, the network learns and changes its weights. This network also uses, in each iteration, a smaller group of images instead of the entire dataset. This leads to a higher fluctuation of weights. The momentum hyperparameter is employed to prevent this large weight fluctuation. The two generated graphs are reproduced in figure 4.7 and 4.8.

More tests were performed using momentum values over 0.9 but the train crashed because the loss result was extremely high which lead to a loss variable overflow.

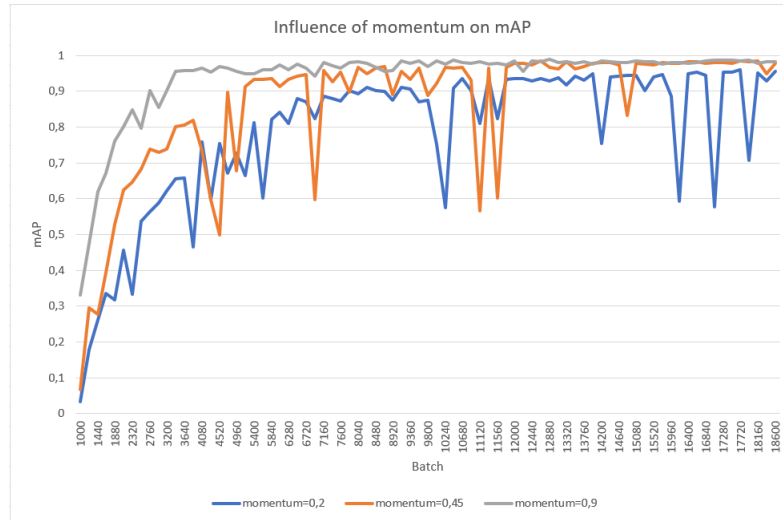


Figure 4.7: Influence of momentum on mAP

Interpreting figure 4.7, it is possible to conclude that the best value for the momentum hyperparameter is 0.9. Through the train, it mostly has the highest mAP value, mAP=0.98804 or 98.804% in batch 12880, and this metric does not fluctuate as much as the other two trains. The line in which momentum=0.2 does not reach the best mAP values and has a high fluctuation. The line with momentum=0.45 has the same fluctuation but has higher mAP outcomes.

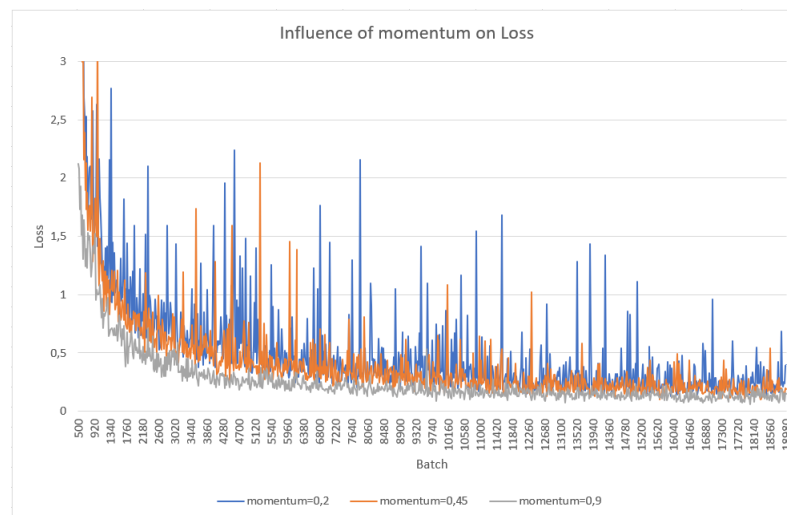


Figure 4.8: Influence of momentum on Loss

In the loss outputs, figure 4.8, one can reach the same conclusion. The best value for momentum is 0.9 because it constantly has the lowest loss and the values do not fluctuate as much as the other tests. Comparing the other two tests, the momentum=0.4 line has the highest loss and both have a high fluctuation.

With these two tests, it is possible to conclude that the best value for the momentum hyperparameter is 0.9. These two figures also demonstrated the effect that the prevention of large weight fluctuations have on the network. Low momentum values lead to the worst results.

4.2.1.4 Burn in

It has been found that the training speed tends to increase if there is a lower learning rate for a short period at the very beginning. The burn_in parameter can control this. The value attributed to this hyperparameter will settle the iteration in which the network has the learning rate value attributed by the tests performed in section 4.2.1.2, in the previous iterations the learning rate was extremely reduced. The two generated graphs are represented in figure 4.9 and 4.10.

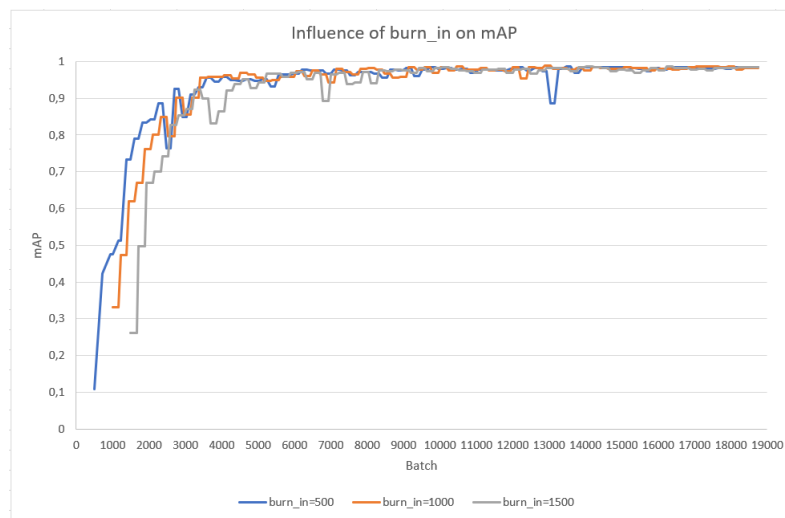


Figure 4.9: Influence of Burn_in on mAP

Unlike the other tested hyperparameters that influence the mAP and the loss, the burn_in parameter mostly affects the training speed. Analysing figure 4.9, it is possible to check that during the trains, the three tested values converge to the same mAP although there are some fluctuations until this mAP is reached. The maximum mAP value, mAP=0.988043 or 98.8043%,

is reached in iteration 12880 in the burn_in=1000 line. Although one of the lines had the best result, as the training progressed, the three lines tended to the same value, which leads to the conclusion that this hyperparameter does not affect mAP.

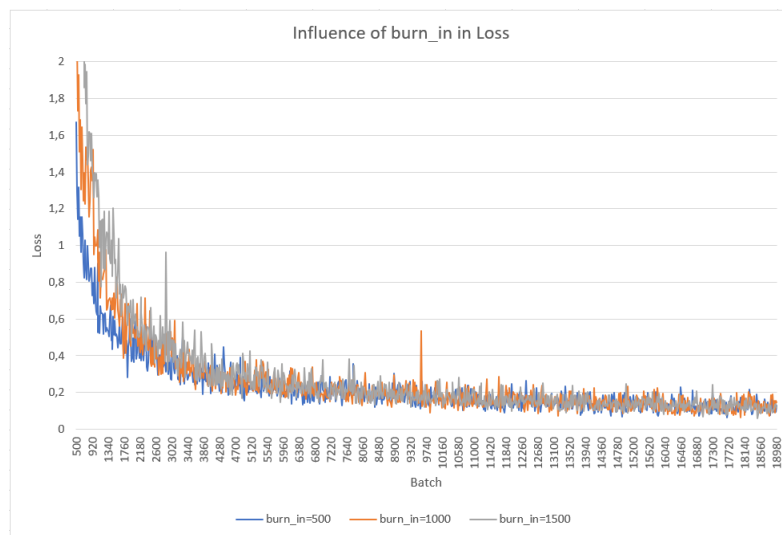


Figure 4.10: Influence of Burn_in on Loss

The same conclusion can be made regarding the loss since the three values, despite the fluctuations and the beginning of the train, tend towards the same value during training, analyzing figure 4.10.

Burn in value	Time the train lasted
500	17 hours 33 minutes
1000	16 hours 9 minutes
1500	16 hours 3 minutes

Table 4.3: Time each train lasted

Unlike previously analyzed metrics, mAP and loss, training times are shown in table 4.3. It is possible to establish that the lower the burn_in value, the longer the training time. This leads to the conclusion that burn_in affects training time and not the metrics already referenced.

These tests were performed in Google Colab Pro to save time using the provided GPUs. The computational power available can fluctuate throughout the tests and this can lead to a slightly different training time for two equal trains.

4.2.1.5 Decay

To prevent overfitting in situations where the model learns too much about the training data, the decay hyperparameter is used to penalize large values for weights. In these circumstances, the model is useful in reference only to its training dataset, and not to any other datasets, such as the testing one. In this case, the model proves to be fitting only for the training data, as if the model had only memorized the training data and was not able to generalize to other data never seen before. In figure 4.11 and 4.12 the two generated graphs are presented.

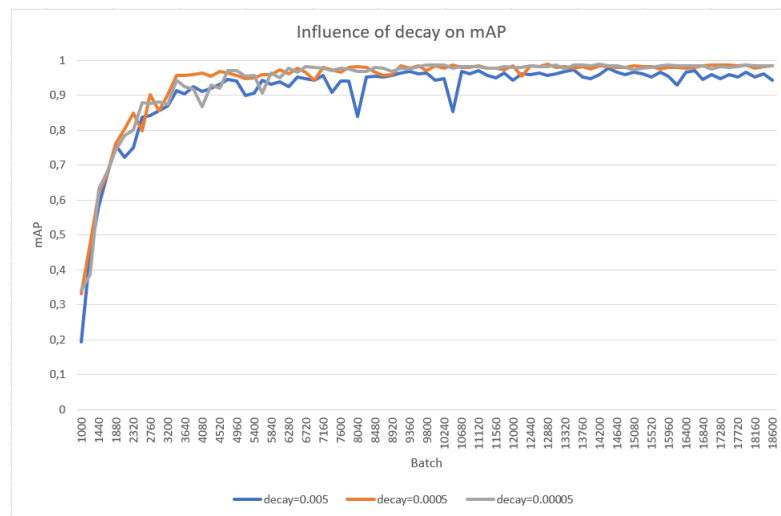


Figure 4.11: Influence of decay on mAP

The main conclusion that can be reached by analysing figure 4.11 is that the decay=0.005 line is the worst one of the ones tested because it constantly has inferior mAP values. There is not a great difference between the other two values for decay since they constantly have approximately the same values throughout the train. The highest mAP value, mAP=0.98857 or 98.857, was achieved in iteration 14200 for the decay=0.00005 value.

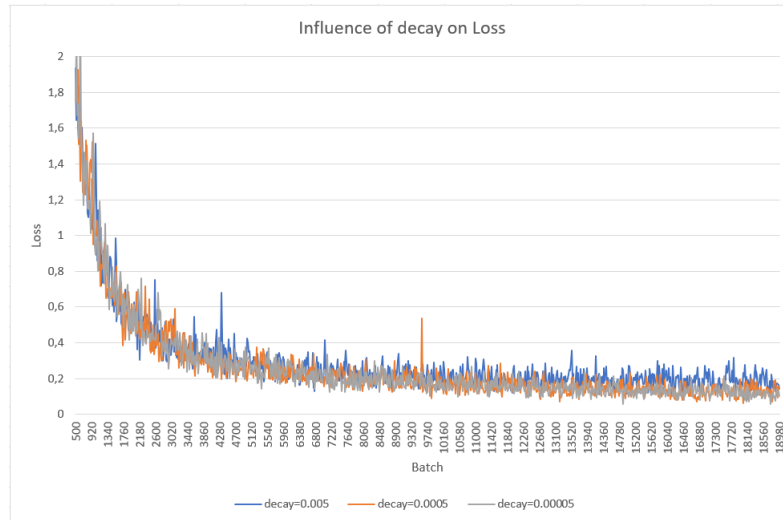


Figure 4.12: Influence of decay on Loss

Interpreting figure 4.12, the same conclusion can be reached regarding the decay values. The decay=0.005 line has the highest loss values in comparison to the other two. The other two values have similar values throughout the train.

It can be concluded that 0.0005 and 0.00005 values are the most optimized from the ones tested for the decay hyperparameter.

4.2.1.6 Width and Height

The width and height parameters define the image dimensions in which the images will be resized in order to be used in the training process. The maximum value is 608x608 and with these dimensions, the results should improve but it takes longer to train. In figure 4.13 and 4.14 the two generated graphs are presented and table 4.4 presents the training times.

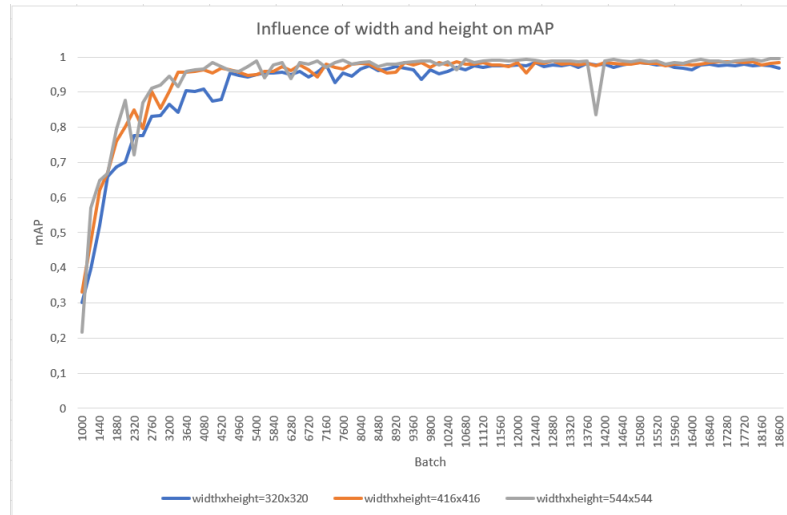


Figure 4.13: Influence of width and height on mAP

Performing an examination on figure 4.13, the highest mAP value, $mAP = 0.99514$ or 99.514%, is achieved in iteration 18380 when the width and height have a value of 544. The other two lines throughout the train also reach near mAP values. The line with width and height=416 reaches $mAP = 0.98804$ or 98.804% and the line with width and height=320 reaches $mAP = 0.98392$ or 98.392%.

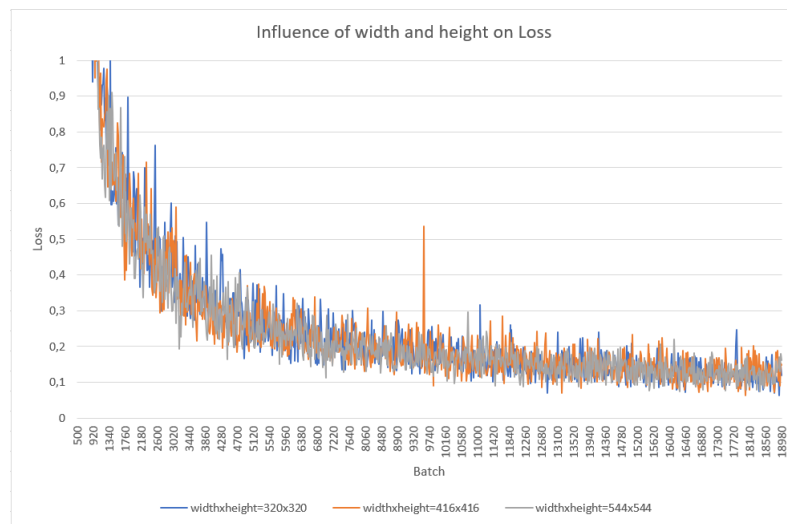


Figure 4.14: Influence of width and height on Loss

Width and Height value	Time the train lasted
320	8 hours 56 minutes
416	18 hours 6 minutes
544	28 hours 13 minutes

Table 4.4: Time each train lasted

Analysing figure 4.14, the resulting lines for the three values for width and height are similar. The three lines evolve at the same rate and have a similar fluctuation.

Since the resolution of the image influences the number of pixels each image has the higher the input resolution, the longer the train time for each iteration. This is proven in table 4.4 where train time is higher for higher resolutions.

4.2.2 YoloV3_tiny

In this section, the values for the hyperparameters in Table 4.5 are tested in the YOLOV3_tiny network and the results are presented.

Hyperparameter	Test 1	Test 2	Test 3
Max Batches	19000	50000	72000
Learning Rate	0.01	0.001	0.0001
Momentum	0.2	0.45	0.9
Burn in	500	1000	1500
Decay	0.005	0.0005	0.00005
Width x height	320x320	416x416	544x544

Table 4.5: The hyperparameters and the corresponding tested values

4.2.2.1 Max Batches

The max_batches hyperparameter was tested with the most optimized value tested in section 4.2.1.1, max_batches=1900, but analysing the mAP results the conclusion was reached that the network could increase its accuracy with higher iterations. The following values tested were max_batches=50000 and max_batches=72000 and the two generated graphs are granted in figure 4.15 and 4.16.

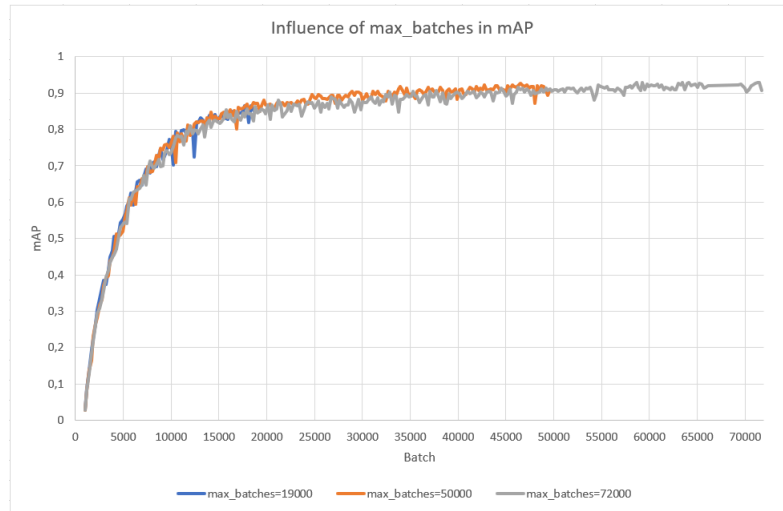


Figure 4.15: Influence of max_batches on mAP

The max_batches=19000 line is the one whose max mAP value, mAP=0.86536 or 86.536%, is the lowest in comparison to the other two networks. With more iterations, max_batches=50000, the network is able to reach higher mAP values and strikes an mAP of 0.92611 or 92.611%. Increasing, even more, the number of iterations, max_batches=72000, the network only increases its mAP slightly to 0.92895 or 92.895%.

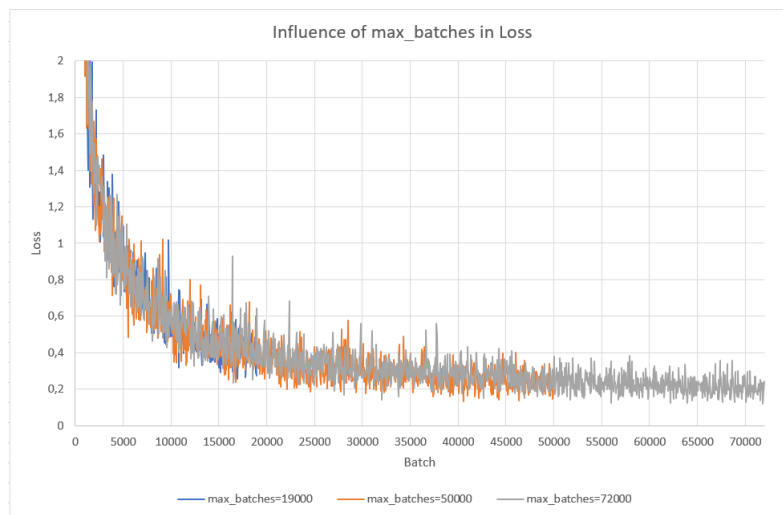


Figure 4.16: Influence of max_batches on Loss

Regarding the loss, the same conclusion can be reached concerning the max_batches=19000 line where it does not have enough iterations to reach lower values. In comparison to the

max_batches=50000 line, the extra iterations from the max_batches=72000 line does not offer an improvement in terms of loss.

Max_batches value	Time the train lasted
19000	4 hours 24 minutes
50000	14 hours 7 minutes
72000	17 hours 51 minutes

Table 4.6: Time each train lasted

As concluded in section 4.2.1.1, the train takes longer if it has more iterations. This outcome can be visualised by interpreting table 4.6.

In conclusion, the max_batches=19000 line does not have enough iterations to reach the values that the other two lines obtain. The extra computational power for the max_batches=72000 line in comparison to the max_batches=50000 line does not result in significant upgrades. Consequently, the best result and the one that is going to be used in the following sections for the YOLO_tiny network is max_batches=50000.

4.2.2.2 Learning Rate

The same values used in section 4.2.1.2 were used to test the optimal learning_rate value for the YOLOV3_tiny network. The output graphs are represented in figure 4.17 and 4.18.

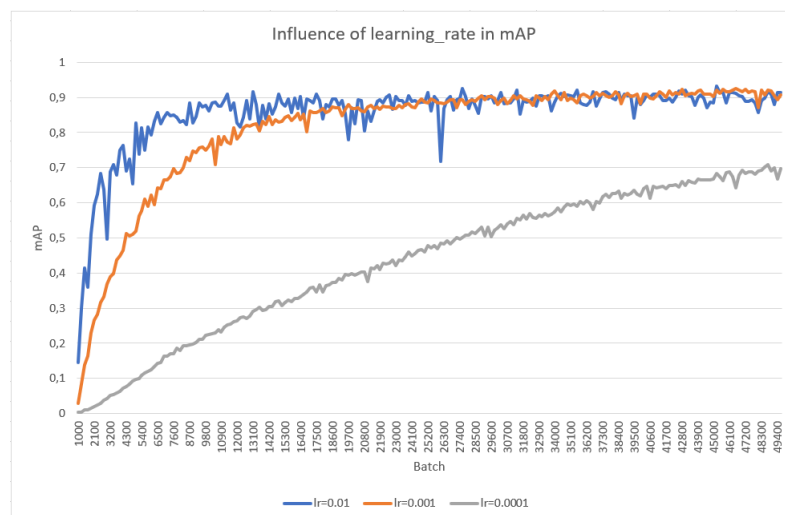


Figure 4.17: Influence of learning_rate on mAP

Interpreting figure 4.17, the effect that this hyperparameter has is visible. With a lower learning_rate value, as conceived in the lr=0.0001 line, the mAP does not reach the desired value. With a higher value, as shown in the lr=0.01 line, the curve has more fluctuation and reaches early its maximum value. This early maximum value can lead to overfitting. The line that reaches the highest value is the learning_rate=0.001 with mAP=0.92611 or 92.611%.

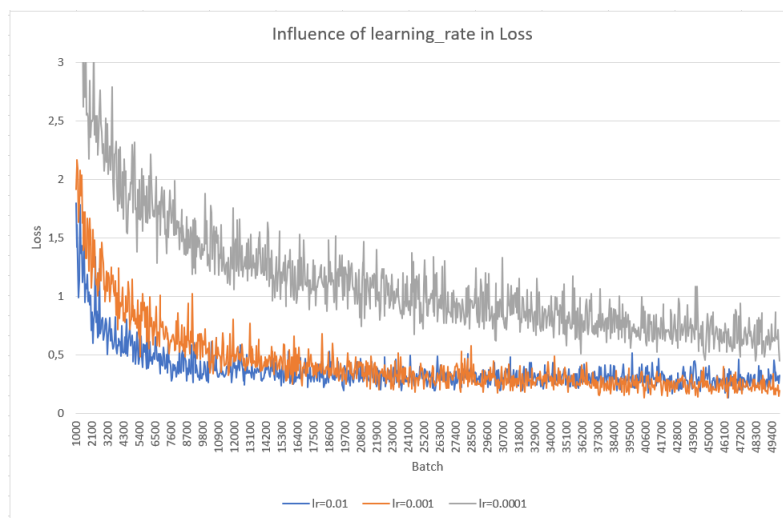


Figure 4.18: Influence of learning_rate on Loss

The loss of the lr=0.0001 line is the worst result obtained stabilizing with a value that doubles the achieved in the other two tests. The best result was accomplished with the lr=0.001 line that despite having a higher loss value for half of the train, obtained an inferior and more stable loss value at the second half of the train, compared to the lr=0.01 line.

Analyzing the results obtained in figure 4.17 and 4.18 it was concluded that the best value for the learning_rate hyperparameter is lr=0.001.

4.2.2.3 Momentum

Regarding the momentum hyperparameter, the values tested are the same as the ones used in section 4.2.1.3. Also, as stated in the previously mentioned section, values above 0.9 led to a train crash because the loss value was too high, leading to a variable overflow. The output graphs are reproduced in figure 4.19 and 4.20.

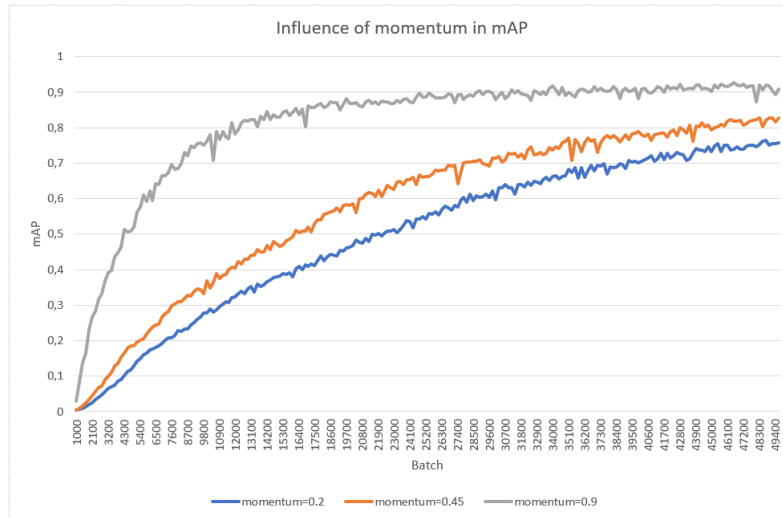


Figure 4.19: Influence of momentum on mAP

The differences between the three output lines allow an easy conclusion regarding which value is the most optimized for the hyperparameter tested. Over the three trains, the momentum=0.9 line consistently contains the highest mAP values by a solid margin.

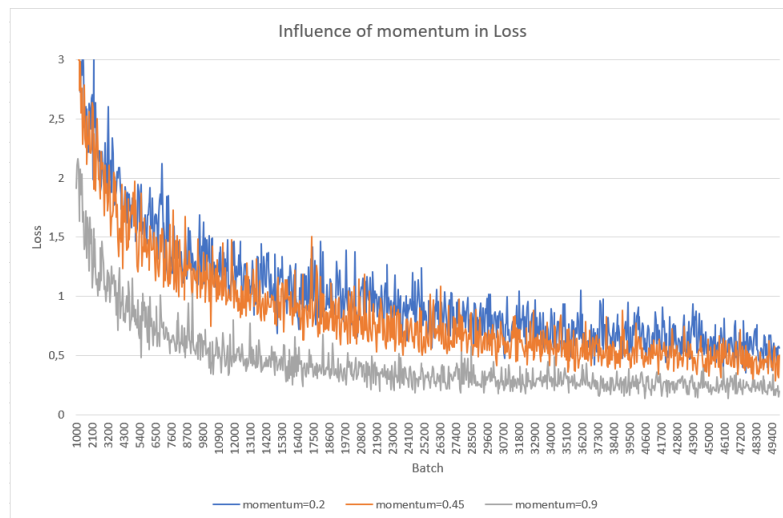


Figure 4.20: Influence of momentum on Loss

Analysing figure 4.20, the same conclusion can easily be made regarding the loss in which the momentum=0.9 reaches a lower value by a great margin.

Concluding, the best value for the momentum is 0.9.

4.2.2.4 Burn in

To test the optimal value for the burn_in hyperparameter, the values used in section REF are used for the YOLOV3_tiny network. The results are displayed in figure 4.21 and 4.22.

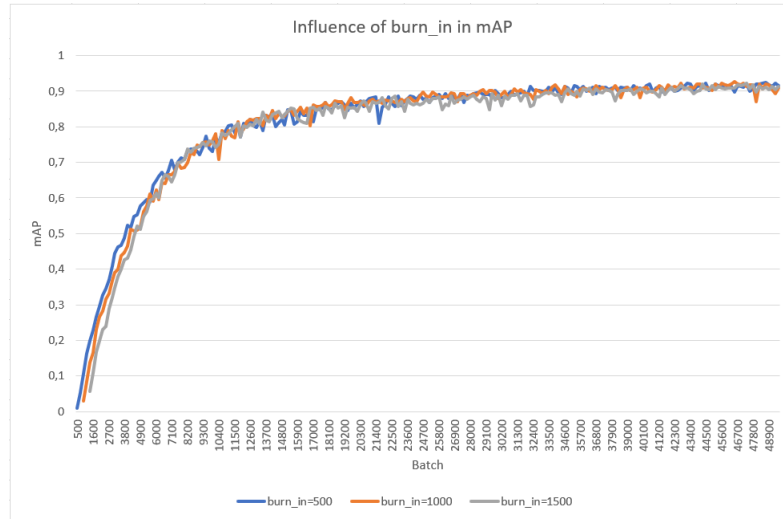


Figure 4.21: Influence of Burn_in on mAP

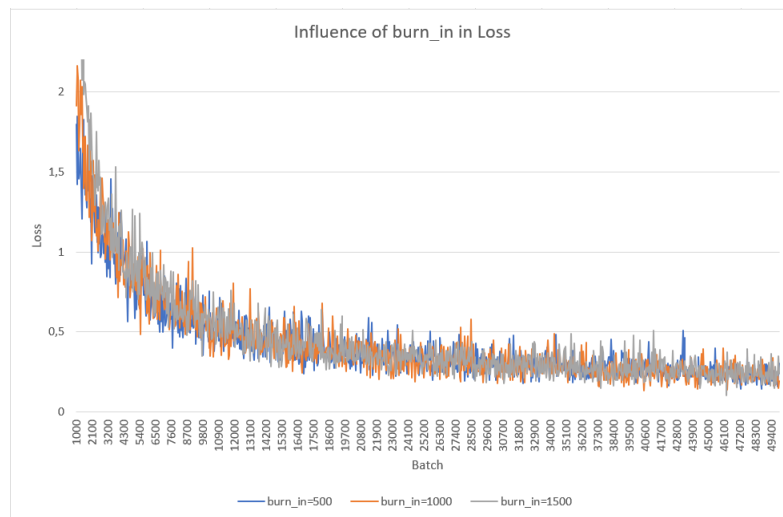


Figure 4.22: Influence of Burn_in on Loss

Analysing the results shown in figure 4.21 and 4.22, it is visible that in both graphs the lines tend to the same values and have similar fluctuations. This is caused by the burn_in variable changing the train time and not its performance. The highest value was recorded by the burn_in=1000 line in iteration 46480.

Burn in value	Time the train lasted
500	13 hours 33 minutes
1000	12 hours 27 minutes
1500	11 hours 22 minutes

Table 4.7: Time each train lasted

In table 4.7 it is visible the effect that the burn_in variable has in the time each train lasted.

4.2.2.5 Decay

The same values used in section 4.2.1.5 were used to test the optimal decay value for the YOLOV3_tiny network. The output graphs are represented in the following figures.

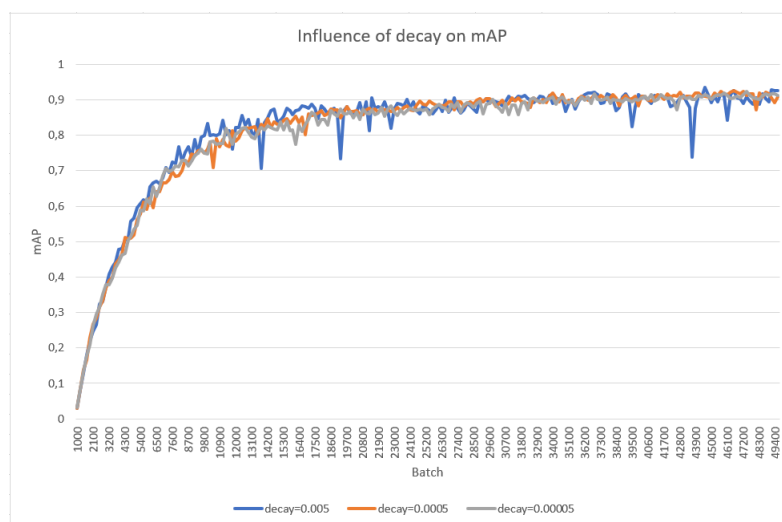


Figure 4.23: Influence of decay on mAP

Analysing both figures it is possible to conclude that the outputs are similar for the three tested values. In the mAP figure, all the lines tend to the same value. The line that has less fluctuation is the decay=0.0005 line and reached mAP=0.9261 or 92.61%.

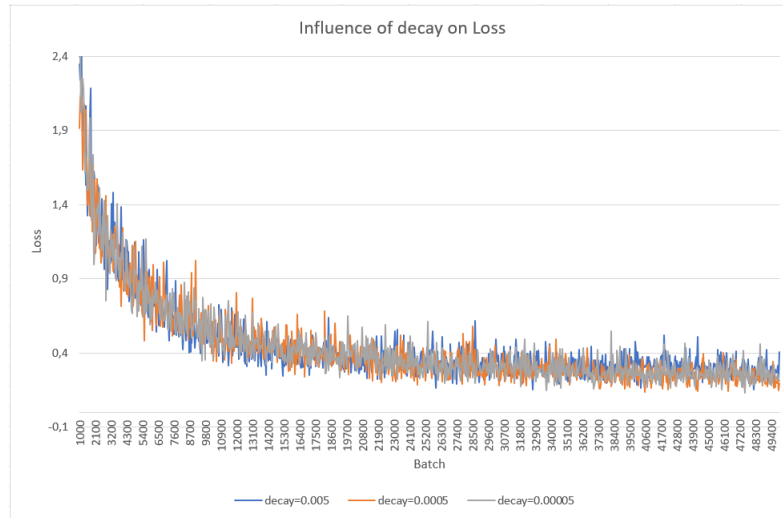


Figure 4.24: Influence of decay on Loss

The same conclusion is shown in the loss figure in which all the lines tend to the same value. The fluctuation is lower in the decay=0.0005 line.

4.2.2.6 Width and Height

Regarding the width and height hyperparameter, the values tested are the same as the ones used in section 4.2.1.6. The output lines are presented in the following images.

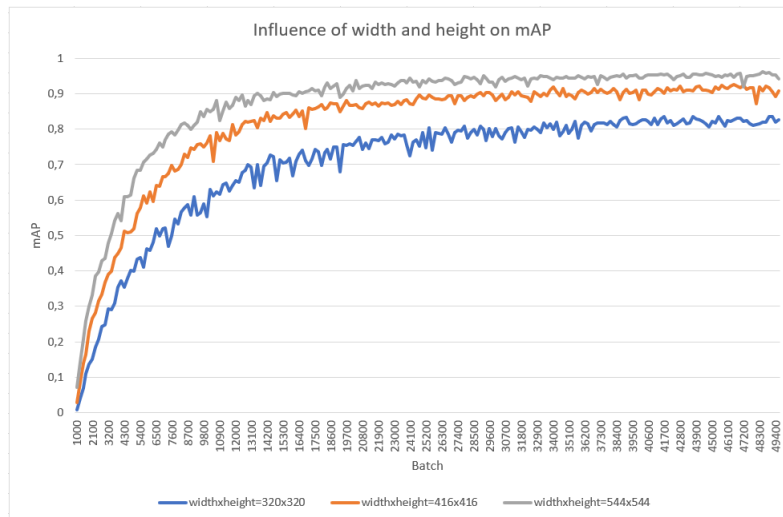


Figure 4.25: Influence of width and height on mAP

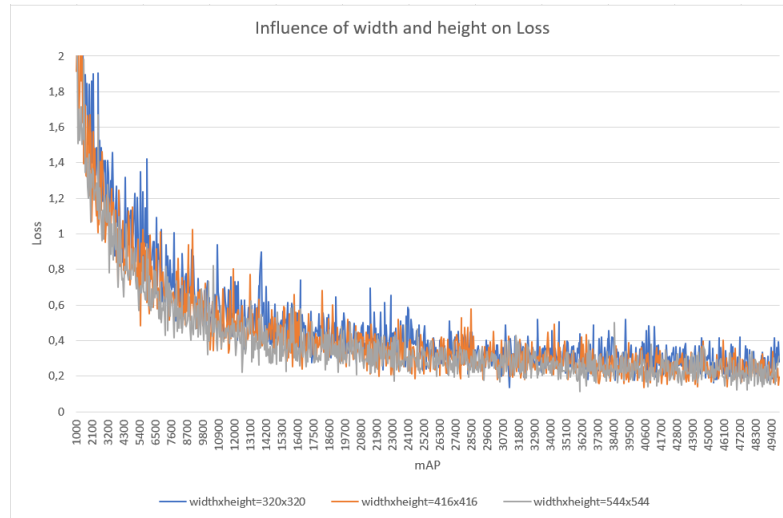


Figure 4.26: Influence of width and height on Loss

Interpreting both figures, the most optimized value is easily concluded. The resolution of 544x544 has significantly better mAP results reaching mAP=0.96109 or 96.109%. This is an increase of 3.498% compared to the 416x416 resolution.

Regarding the loss figure, the 544x544 resolution constantly reaches the lowest value throughout the tests.

Width and Height value	Time the train lasted
320	11 hours 42 minutes
416	14 hours 7 minutes
544	20 hours 4 minutes

Table 4.8: Time each train lasted

Analysing table 4.8, the conclusion that higher resolutions lead to higher training times is verified. The extra computational power required for the 544x544 resolution in comparison to the other resolutions results in significant improvements.

4.2.3 Conclusion

After performing tests on both networks for the chosen hyperparameters the most optimized values are presented in the following tables.

These tests proved that the values for the hyperparameters provided by the YOLO architecture are mostly the most optimized ones. In the table 4.9, only the max_batches has a different value

Hyperparameter	Optimized value
max_batches	19000
learning_rate	0.001
momentum	0.9
burn_in	1000
decay	0.0005
widthxheight	416x416

Table 4.9: Optimized hyperparameters for the YOLOV3 network

Hyperparameter	Optimized value
max_batches	50000
learning_rate	0.001
momentum	0.9
burn_in	1000
decay	0.0005
widthxheight	544x544

Table 4.10: Optimized hyperparameters for the YOLOV3_tiny network

due to the number of classes. In table 4.10, the max_batches was changed due to the number of classes and the width and height resolution was increased to enhance accuracy.

These values were used to train the final networks that are presented in the following chapter.

Chapter 5

Analysis of Results

In chapter 4, tests were carried out with different values to reach the most optimized values of the hyperparameters.

In this chapter, initially, the training of the final networks is presented using their most optimized hyperparameters. Subsequently, the results of the two developed networks per objective will be compared using videos.

5.1 Final networks

For each of the objectives presented in section 3.1, and using the corresponding dataset and the optimized hyperparameters, two networks were created (YOLOV3 and YOLOV3_tiny).

5.1.1 Autonomous Driving Competition of the RoboCup Portuguese Open in simulation

5.1.1.1 YOLOV3

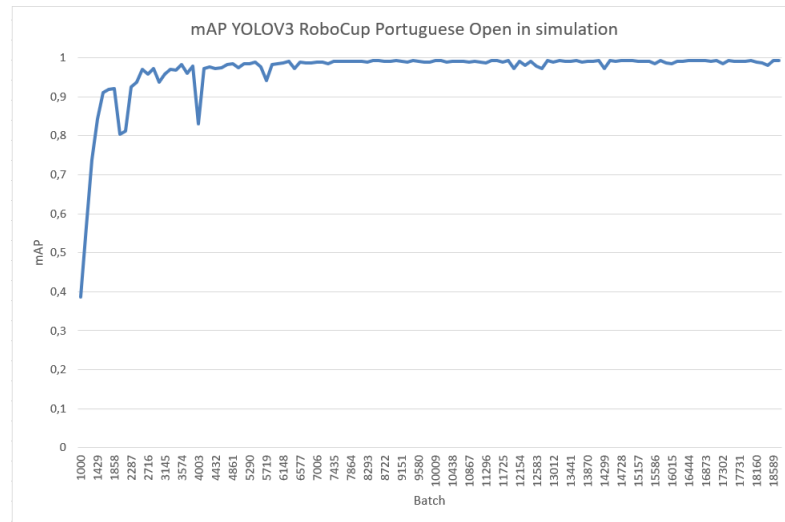


Figure 5.1: YOLOV3 mAP for the Autonomous Driving Competition of the RoboCup Portuguese Open in simulation

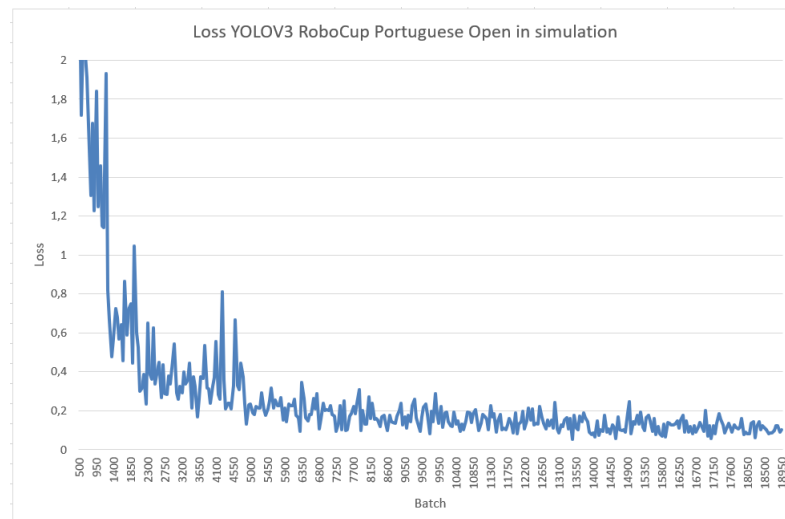


Figure 5.2: YOLOV3 loss for the Autonomous Driving Competition of the RoboCup Portuguese Open in simulation

The best mAP, mAP=0.9932 or 99.32%, was achieved at iteration 12869.

5.1.1.2 YOLOV3_tiny

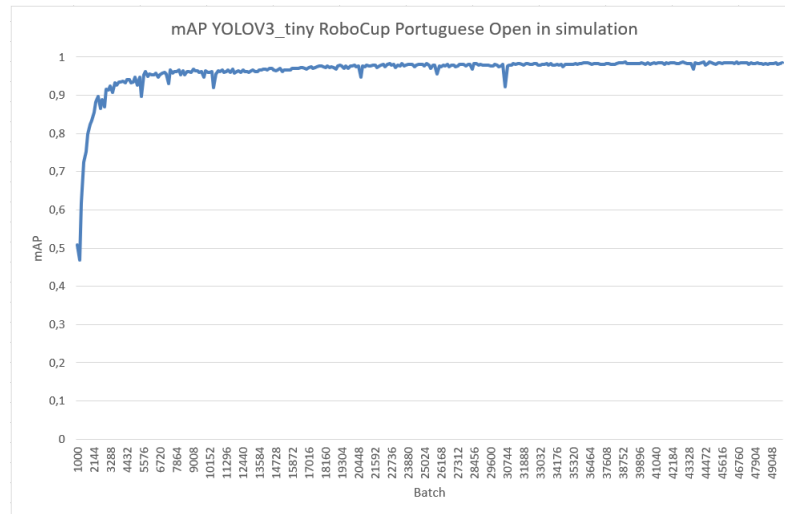


Figure 5.3: YOLOV3_tiny mAP for the Autonomous Driving Competition of the RoboCup Portuguese Open in simulation

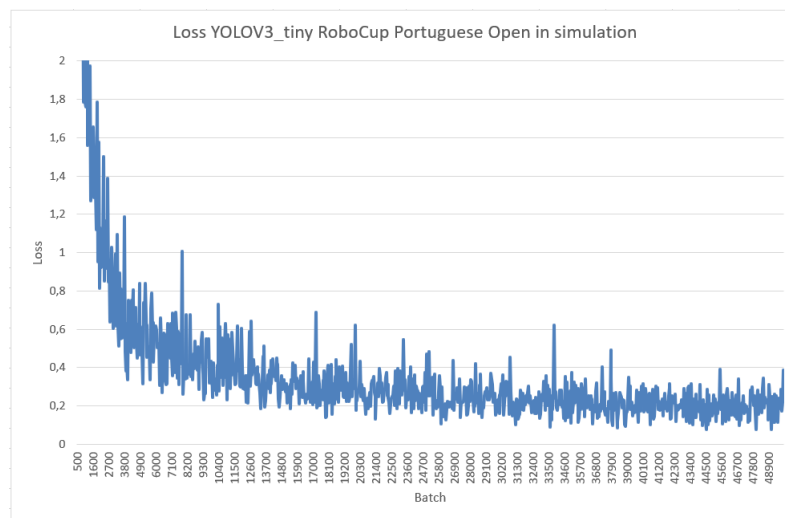


Figure 5.4: YOLOV3_tiny loss for the Autonomous Driving Competition of the RoboCup Portuguese Open in simulation

The best mAP, mAP=0.98791 or 98.791%, was achieved at iteration 46617.

5.1.2 Autonomous Driving Competition of the RoboCup Portuguese Open

5.1.2.1 YOLOV3

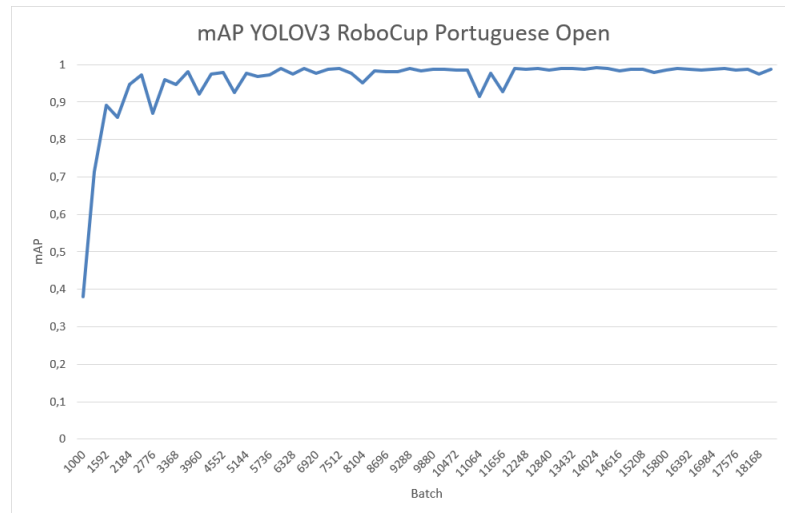


Figure 5.5: YOLOV3 mAP for the Autonomous Driving Competition of the RoboCup Portuguese Open

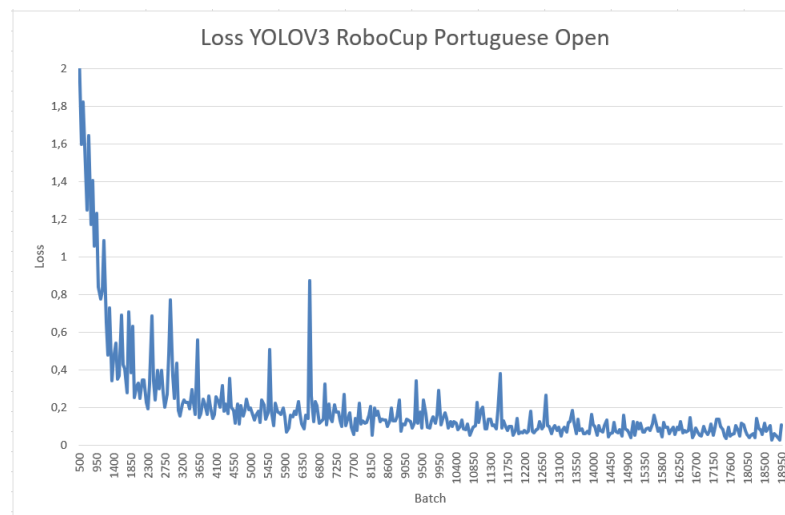


Figure 5.6: YOLOV3 loss for the Autonomous Driving Competition of the RoboCup Portuguese Open

The best mAP, mAP=0.99086 or 99.086%, was achieved at iteration 14024.

5.1.2.2 YOLOV3_tiny

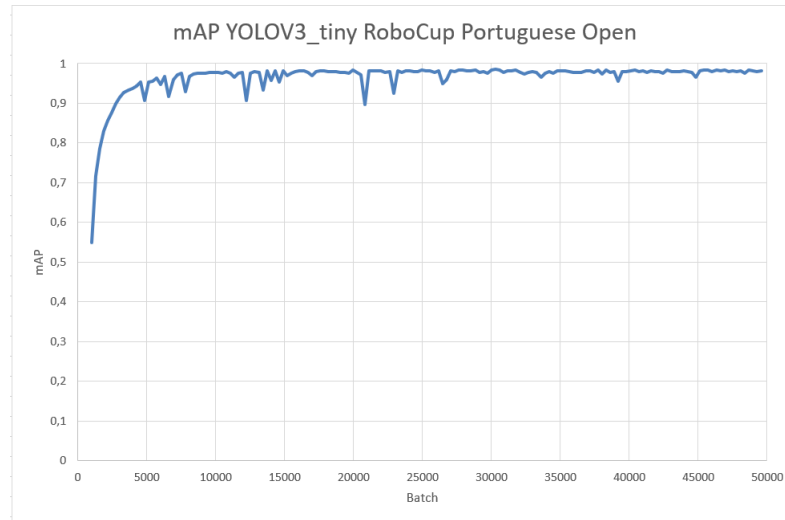


Figure 5.7: YOLOV3_tiny mAP for the Autonomous Driving Competition of the RoboCup Portuguese Open

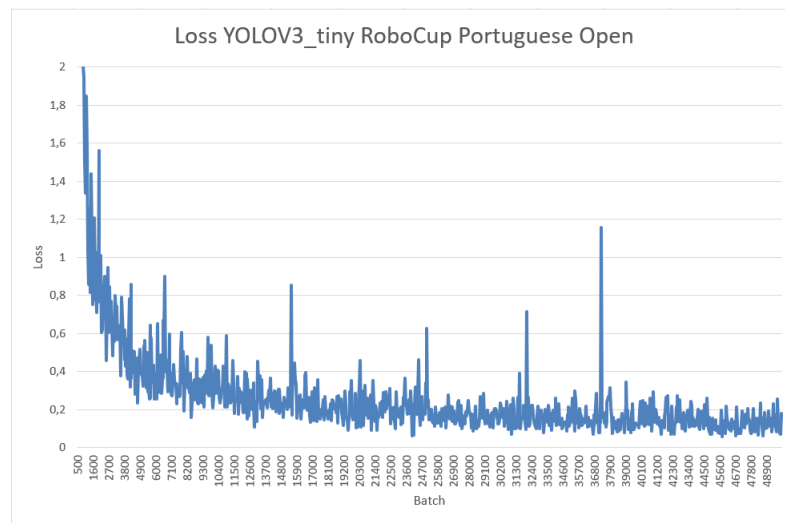


Figure 5.8: YOLOV3_tiny loss for the Autonomous Driving Competition of the RoboCup Portuguese Open in simulation

The best mAP, $mAP=0.98479$ or 98.479%, was achieved at iteration 30600.

5.1.3 Public Road

5.1.3.1 YOLOV3

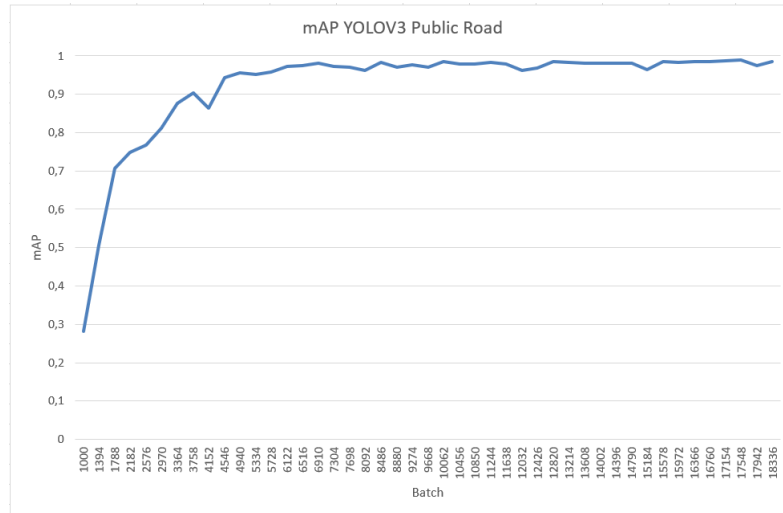


Figure 5.9: YOLOV3 mAP for the Public Road

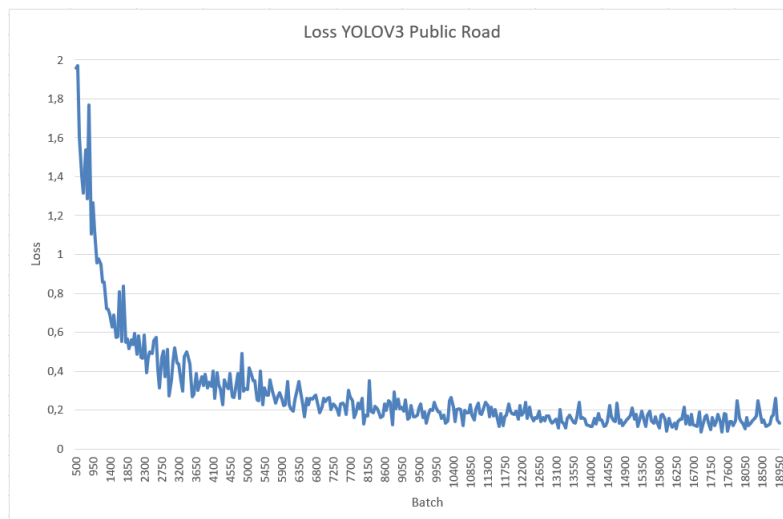


Figure 5.10: YOLOV3 loss for the Public Road

The best mAP, mAP= 0.98914 or 98.914%, was achieved at iteration 17548.

5.1.3.2 YOLOV3_tiny

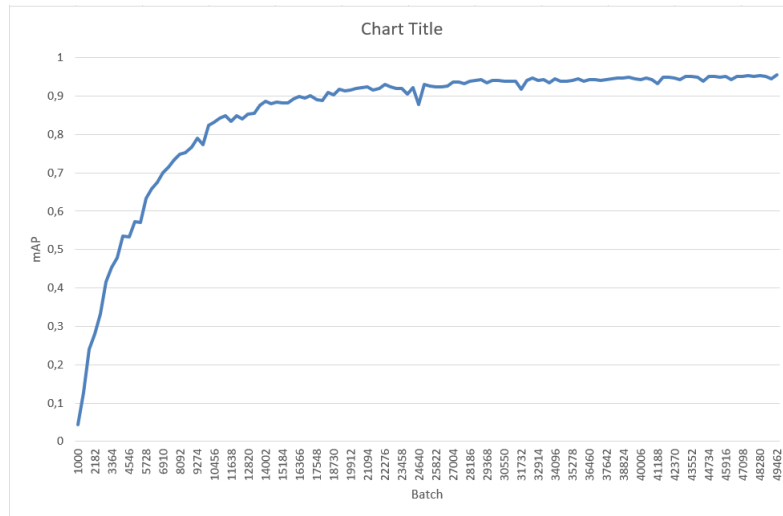


Figure 5.11: YOLOV3_tiny mAP for the Public Road

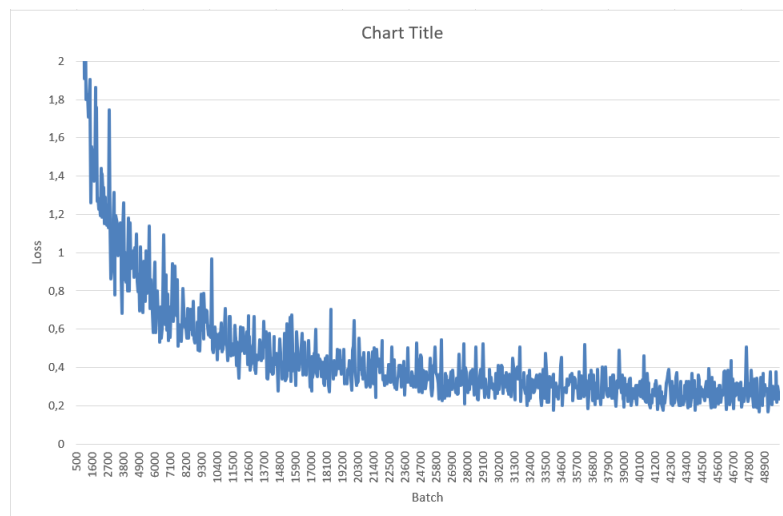


Figure 5.12: YOLOV3_tiny loss for the Public Road

The best mAP, mAP=0.95584 or 95.584%, was achieved at iteration 49462.

5.2 Comparison between networks per objective

In this section, the output of the developed YOLOV3 and YOLOV3_tiny networks is compared in each objective.

The outputs consists of the videos that were clarified in the corresponding objectives from section 3. These videos were created and later processed by the corresponding networks.

5.2.1 Autonomous Driving Competition of the RoboCup Portuguese Open in simulation

To test the simulation environment for the RoboCup Portuguese Open, the two tests referenced in section 3.2.3 will be used. These tests allow network performance visualization and a more detailed comparison between the two developed networks.

5.2.1.1 First Test

For this test, the signs were placed in two parallel lines to verify their correct classification and detection. The minimum confidence for which signals were detected was 80%.

Comparing both networks in the first test located in appendix A.1, it is possible to verify that the computational power required for the YOLOV3 network is superior to what the computer offers. This means that only a fraction of the frames is processed. Regarding classification, the fraction of frames that are processed unveils that the signs are all classified and detected with confidence over 95%. In Figure 5.13, two examples of this high confidence in detection are demonstrated.

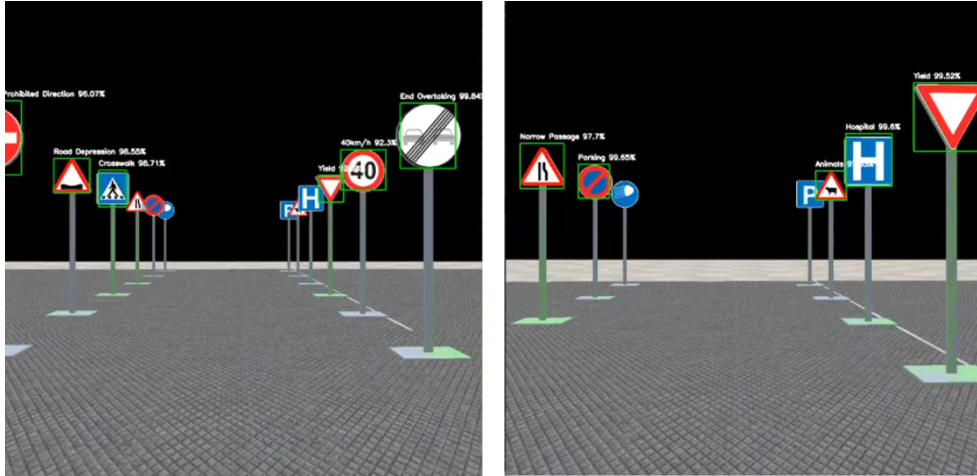


Figure 5.13: Two frames with high confidence detection from the YOLOV3 network

On the other hand, the YOLOV3 tiny network manages to process the frames in such a way that the robot is constantly identifying and classifying the signs. Compared to the other network, YOLOV3_tiny does not have such high detection confidence or stable Bounding Boxes but its ability to process frames overcomes these limitations. In Figure 5.14, this detection is displayed.

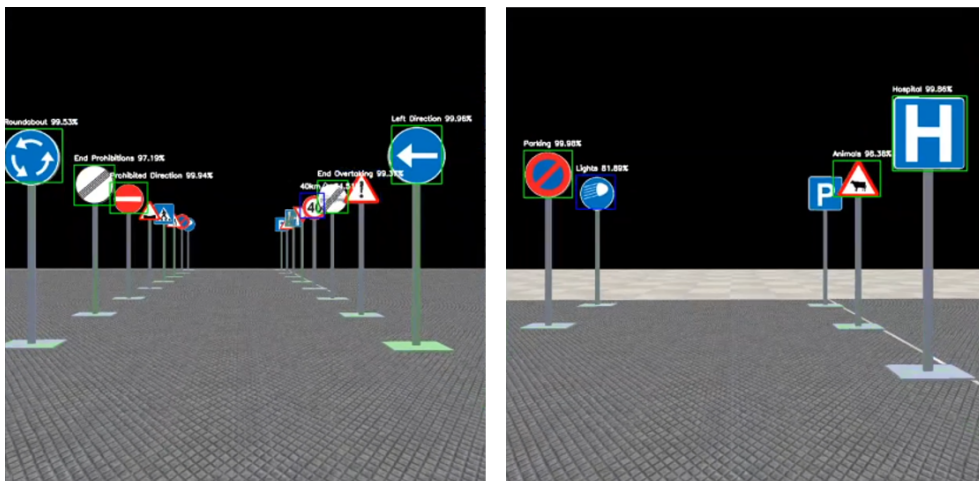


Figure 5.14: Two frames with high confidence detection from the YOLOV3_tiny network

5.2.1.2 Second Test

For the second test, the signs were placed so that the robot would detect them sequentially while performing the respective movements. The robot reacted to the signs when confidence was over 98%.

Comparing the outputs of the two networks in appendix A.1 from the second test, it is possible to verify the difference between the two networks. As concluded in the previous test, the output for the YOLOV3 network cannot process all the frames so that the robot reacts to the TS/L promptly to perform the corresponding movements. This lack of performance implied that the robot only reacted to the Left Obligation sign when it was already extremely close to it. This meant that the robot could not turn in time to move to the other signs. Regarding classification, the network was able to correctly classify with confidence above 99% the signs it obtained in the frames. The moment the robot reacted to the Left Obligation sign is presented in Figure 5.15.

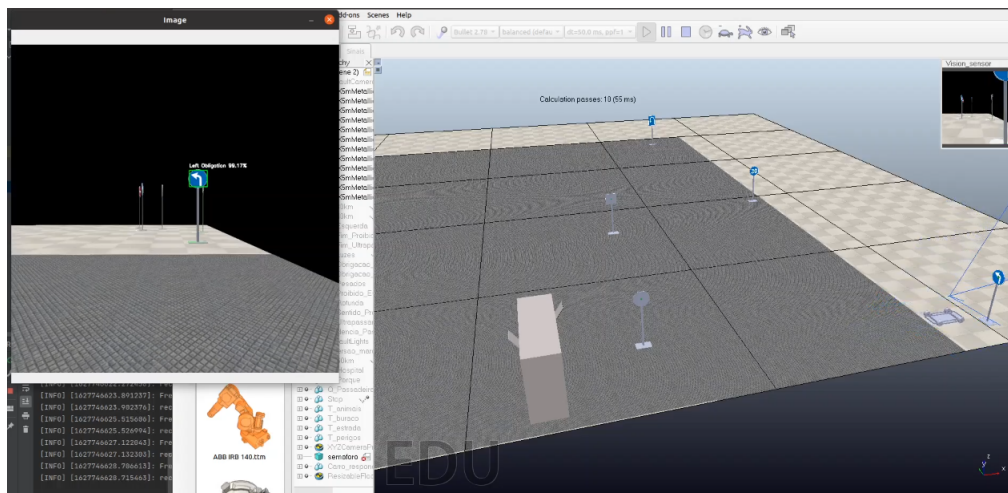


Figure 5.15: Frame where the robot reacted to the Left Obligation sign

Unlike the YOLOV3 network, the YOLOV3_tiny network was able to timely react to all traffic signs. In Figure 5.16, it is possible to visualize the frame in which the robot reacted to the Left Obligation sign. It reacted earlier than the YOLOV3 network. This difference can be visualized by comparing the position of the robot in figure 5.15 and 5.16.

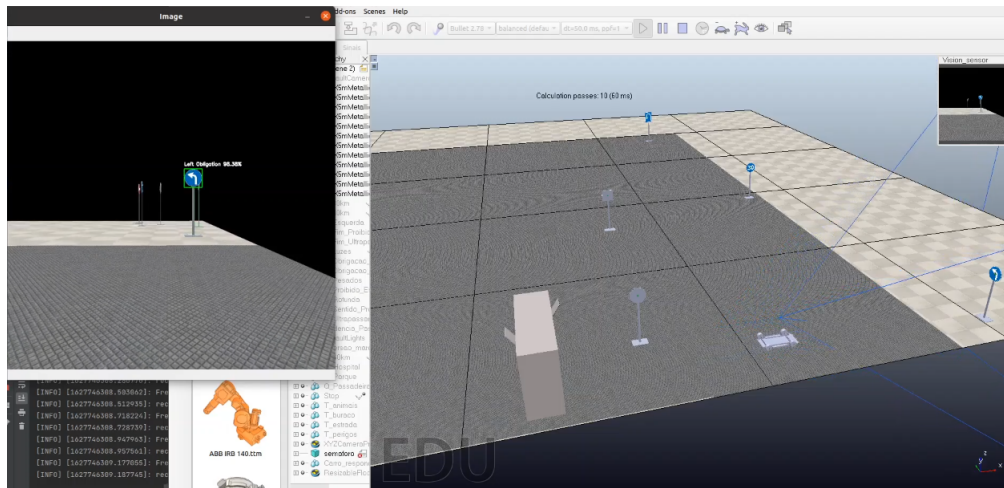


Figure 5.16: Frame where the robot reacted to the Left Obligation sign

As explained in section 3.2.3.2 the robot has the ability to change its speed if it detects one of the three available speed signs. This change of speed can be visualized in the videos, in the robot or the CoppeliaSim command line.

In figure 5.17, the robot correctly identifies the 30km/h sign and changes its velocity to achieve the velocity designated by the sign.

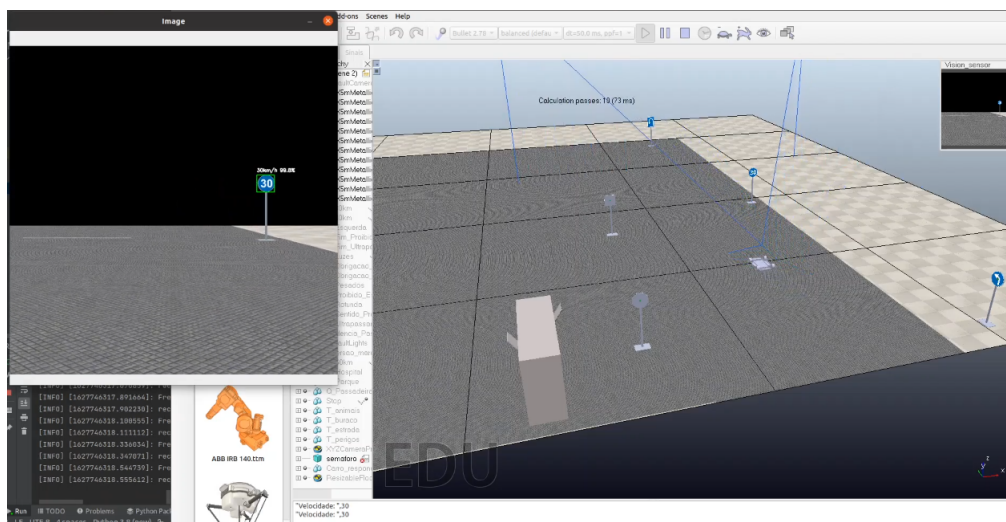


Figure 5.17: Frame where the robot reacted to the 30km/h sign

In this test, the YOLOV3_tiny network enabled the robot to go all the way until it stopped at the STOP sign, frame displayed in figure 5.18. Before it achieved the previously mentioned sign, the robot also accurately classified the U-Turn sign and performed its corresponding manoeuvre.

Along its path, the robot was able to detect all the signs in time and this led to being able to perform the corresponding movements to reach the end of the path.

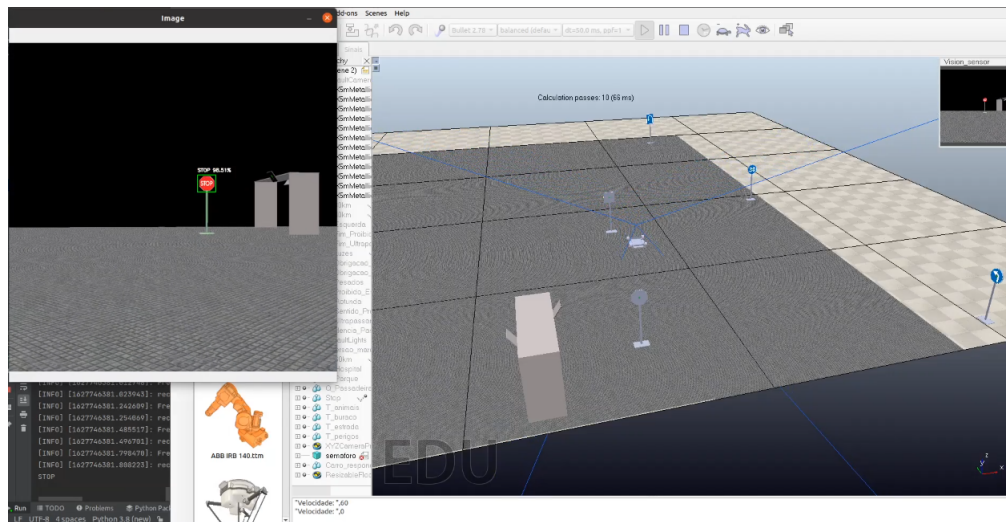


Figure 5.18: Frame where the robot reacted to the STOP sign

5.2.2 Autonomous Driving Competition of the RoboCup Portuguese Open

To test the environment that was built to emulate the Autonomous Driving Competition of the RoboCup Portuguese Open, since the competition did not take place due to the pandemic, two videos were recorded. Both videos demonstrate the robot driving along the track and observing the signs/lights that were randomly placed along. The only difference between these is that the first video was recorded with as much stability as possible and the second was recorded on the robot prototype while it was remotely controlled manually, this led to a lower stabilization. In appendix A.2, these videos are presented sequentially for each of the networks. In these videos, the signs are only detected when confidence is over 80%. In these videos, it is not possible to control the processing time of each frame as they were recorded before and processed afterwards. The YOLOV3 network managed to process an average of 2 fps while the YOLOV3_tiny 17 FPS.

If applied in real-time, the YOLOV3 network would have problems reacting to signs in a timely manner whereas the YOLOV3_tiny would do so with a greater margin.

Comparing both videos it is possible to verify the correct functioning of both networks. The results of both networks are very similar and for YOLOV3 the detection percentages are slightly higher, as can be seen in figure 5.19.



Figure 5.19: Detection from the YOLOV3 (left) and YOLOV3_tiny (right)

Regarding the distance of detections, the results are also slightly higher on the YOLOV3 detecting at 4 meters, half the length of the track. In figure 5.20, the detection at the middle of the track is demonstrated. In this figure, it is possible to verify that, at the presented distance, the YOLOV3 network can detect four of the five signals shown while the tiny version can only detect two. The Narrow Passage sign is the only one that neither of the two networks could classify.



Figure 5.20: Detection distance from the YOLOV3 (left) and YOLOV3_tiny (right)

Bounding Boxes have superior accuracy and are more stable on the YOLOV3 network. The comparison between the Bounding Boxes is shown in figure 5.21. Analysing this figure, the Boxes

are slightly more accurate in the YOLOV3 network but in the Narrow Passage sign, this network places the Bounding Box in a position that leaves a part of the sign out of the Box.



Figure 5.21: Bounding Boxes from the YOLOV3 (left) and YOLOV3_tiny (right)

Regarding videos with less stability, both networks managed to detect the signs that appeared on the track but slightly better precision and stability than the YOLOV3 network. This difference in performance is demonstrated in figure 5.22.

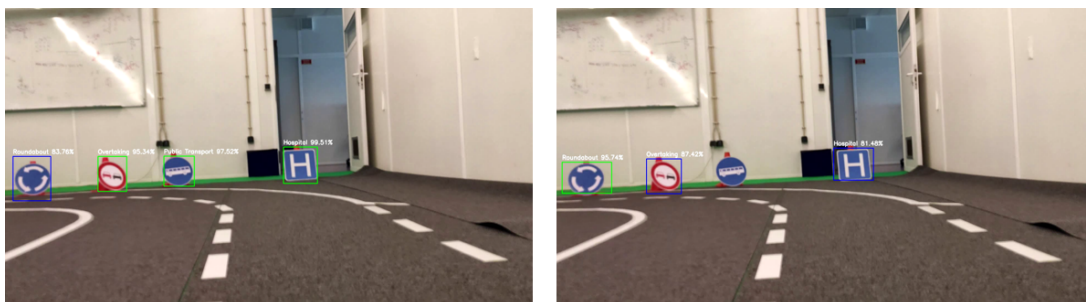


Figure 5.22: Detection with less stability from the YOLOV3 (left) and YOLOV3_tiny (right)

In figure 5.23, a frame with severe distortion is shown to demonstrate how both networks perform in the worst conditions and with a significant distance.

From the six possible signs, the YOLOV3 network correctly identifies two and incorrectly identifies the Narrow Passage sign as the Dangers one. The tiny version presents the worst results as it cannot classify any of the signs presented.

As the robot approaches the signs, even with significant distortion, both networks correctly identify all signs.



Figure 5.23: Detection with extreme distortion from the YOLOV3 (left) and YOLOV3_tiny (right)

5.2.3 Public Road

The Public Road is the most complex of the three objectives. The other two were more constrained and did not have so many background variations that can lead to difficulties in detection and classification.

To test both networks multiple videos were recorded showing traffic signs/lights to verify their correct behaviour. These videos were recorded in multiple places in northern Portugal. Different characteristics appear in the videos, for example, luminosity, weather conditions, time of day and luminosity incidence. These different characteristics allow the visualization of how the networks behave in possible scenarios on public roads. In these videos, the signs only detected when confidence is over 80%.

Comparing both videos in Appendix A.3 it is possible to verify the behaviour of both networks and also the differences between them. In the first moments of the video, the frames have conditions that can be considered ideal as it is sunny and the road has good lighting.

In figure 5.24, multiple signs appear at different distances and the differences between the networks are quite visible. The YOLOV3 network can detect all the signs presented with accurate Bounding Boxes. The YOLOV3_tiny network can only detect about half of the signs and the Bounding Boxes fluctuate a lot in position and they cut some signs, for example, the Bounding Box from the Obligation Arrow sign on the left of the figure does not contain the detected traffic sign.



Figure 5.24: Detection with ideal conditions from the YOLOV3 (left) and YOLOV3_tiny (right)

In figure 5.25 a different scenario is shown, or it is cloudy, darker and the frame is blurrier. The YOLOV3 network is able to correctly classify every presented Traffic Sign with confidence over 95%, even though some signs are not directly facing the camera. The YOLOV3_tiny is not able to deploy the same results. It can only detect the Roundabout sign with 81.69% leaving three signs undetected.

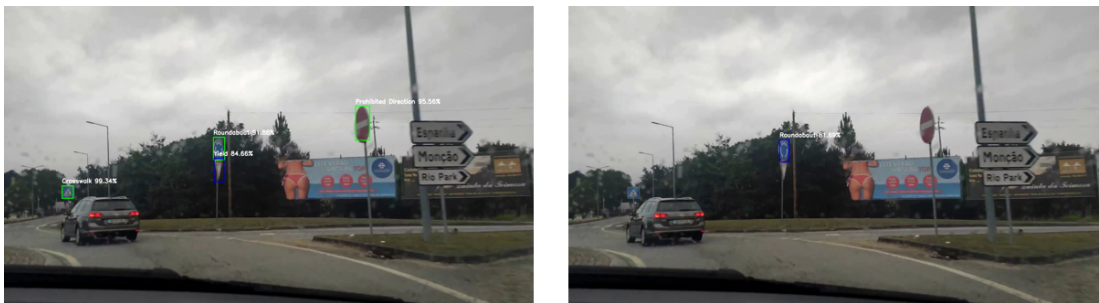


Figure 5.25: Detection with rain from the YOLOV3 (left) and YOLOV3_tiny (right)

In ideal conditions, the frame presented in figure 5.26, presents the detection of three traffic signs, provided that the Prohibited Direction Sign is not facing the camera. Unlike the YOLOV3, the tiny version is unable to detect the previously mentioned sign. The Bounding Boxes from the detected signs are similar in both networks.



Figure 5.26: Detection of a sign not facing the camera from the YOLOV3 (left) and YOLOV3_tiny (right)

The detection of traffic lights is also tested in rainy conditions. In figure 5.27, the comparison of the two networks is shown. The YOLOV3 correctly identifies the traffic light and the color from the two top lights but incorrectly merges two lights into one at the bottom. The tiny version also detects the two traffic signs at the top of the frame and does not detect the two traffic lights at the bottom. In this version, the color on the top light is correctly detected. The YOLOV3 has higher confidence for the detections.



Figure 5.27: Traffic Light detection from the YOLOV3 (left) and YOLOV3_tiny (right)

In the frame presented in figure 5.28, a Traffic Sign that is not in the Public Road dataset is displayed. The YOLOV3 network detects all the Direction signs and does not detect the unknown sign but in the following frames detects it as the Overtaking sign. The YOLOV3_tiny can only detect one Direction sign and does not detect the unknown sign but in the following frames detects it as the End Prohibitions sign.



Figure 5.28: Detection of an unknown sign from the YOLOV3 (left) and YOLOV3_tiny (right)

In the second third of the videos, a video was inserted in order to demonstrate that the network did not detect anything under normal conditions where no signs were presented. Both networks perform correctly and did not detect false positives.

In the last quarter of the video, a video was inserted with a continuous recording in poor conditions, as it was raining and the brightness conditions were reduced. Some frames of the output of the YOLOV3 network, in the previously mentioned conditions, are displayed in figure 5.29. The network correctly identifies the signs in three of the four frames. In the frame with the 50km/h signs, the network can only correctly identify one of these and does not identify the Danger Crosswalk sign.



Figure 5.29: Results of the YOLOV3 network in poor conditions

Some frames of the output of the YOLOV3_tiny network, using the same input, are presented in figure 5.30. The network correctly identifies the Direction Signs and the Obligation Arrow but does not recognize the 50km/h, Danger Crosswalk or the Junction signs.



Figure 5.30: Results of the YOLOV3_tiny network in poor conditions

5.3 Results Discussion

In this chapter, the results for the three dissertation objectives are described in section 3.1. For each objective, a comparison is made between the YOLOV3 and the YOLOV3_tiny networks.

For the Autonomous Driving Competition of the RoboCup Portuguese Open in simulation, the results presented proved that the YOLOV3_tiny network is the most suitable for this environment. The YOLOV3 network exhibited a slightly higher detection and classification confidence but the required computational power to run the simulation in real-time was higher than the computational provided by the used computer. The tiny version was able to correctly detect the TS/L in a timely manner and this provided enough time for the robot to perform the corresponding manoeuvres.

For the Autonomous Driving Competition of the RoboCup Portuguese Open, the same conclusion as the previous objective was reached. The device controlling the robot would not have

enough computational power to provide the required YOLOV3 network power. This would lead the robot to only detect the TS/L when it has already passed it or perform the corresponding manoeuvres belatedly. The tiny version would allow timely detection and the robot would be able to perform the manoeuvres correctly.

The Public Road is the objective where the YOLOV3 network is the most suitable. The detection in this environment must be the most accurate since its application is destined for autonomous cars. Cars that have TSR software have enough computational power to allow a low processing time that allows the network to be able to identify the signs without delays. From the provided results, the YOLOV3_tiny does not have enough accuracy to ensure a correct and safe detection.

Chapter 6

Conclusions and Future Work

The work presented in this dissertation describes the development of neural networks capable of detecting and classifying traffic signs and traffic lights in three different scenarios.

Initially, some introductory concepts are presented regarding fundamentals from Artificial Intelligence to Convolutional Neural Networks that leads to a better understanding of the networks mentioned. A context of how two automotive companies perform Traffic Sign Recognition is presented to better understand how this technology is performed in an industrial environment. Follows an analysis surrounding the Autonomous Driving Competition from the RoboCup Portuguese Open to acquaint how the robot must perform concerning the Vertical Traffic Signs Detection Challenge. A detailed explanation about the architecture used to develop the final networks is shown, which includes its evolution, starting in the first version until the third one. Finally, an application of a TSR for the same competition performed by another team is exhibited using different methods for the same purpose.

The second part of this dissertation focus is on the used methodologies to develop the final networks. For each of the presented objectives, an explanation of the adopted steps is presented. In the first objective, initially, the steps for the creation of all the parts for the simulation are presented. Then, an explanation regarding how these parts were used to develop the dataset is made. The following step had the greatest importance because it was used in all the objectives. It consisted of making a detailed explanation of how the training plan in the YOLO architecture was performed, this included every command line and clarification of the hyperparameters and network structure. Next, the testing phase of each objective was presented to judge if any improvements were required for the networks. The detailed explanation of the YOLO architecture throughout the dissertation and all the adjacent steps to create all networks can allow further projects of Laboratório de Automação e Robótica to use this work.

The third part of this project consisted of testing the hyperparameters for each of the networks. With this study, for the YOLOV3 and the YOLOV3_tiny networks, the optimized values for the hyperparameters were discovered. Testing three values per hyperparameter, the mAP and the loss were used to select which one was the most optimized for each network.

Using the most optimized values for each hyperparameter, concerning each objective, two final networks were developed (YOLOV3 and YOLOV3_tiny), in the fourth part of the dissertation. The mAP and loss graphs for each of the six networks were displayed. Videos representing the results of these networks were created using the corresponding input videos explained in the third part, which allow a study of how these performed.

Regarding the first objective, the YOLOV3_tiny network guarantees results that allow the robot to detect and classify the Traffic Signs/Lights in time to perform the necessary manoeuvres along its path. Despite having a higher accuracy, the processing time of each frame on the V3 network does not allow the robot to work correctly.

In the second objective, the processing time led to the choice to commit to the YOLOV3_tiny network since the computational power of the onboard computer for the competition is usually limited and with the YOLOV3 network, it would not be able to react to the Traffic Signs/Lights in time.

Unlike the networks chosen for the previous objectives, for Public Road, the YOLOV3 network provides the most accurate results and usually, cars have onboard computers that provide enough computational power. Despite correct detection in most signs, the results displayed classification and detection errors when faced with Traffic Lights and with Signs not contained in the dataset.

6.1 Future Work

The work carried out in this dissertation is available to be used by the Autonomous Driving team of Laboratório de Automação e Robótica at the RoboCup Portuguese Open when it occurs.

The future of this project aims to enhance the accuracy of the detection and classification for the three objectives. This can be performed by introducing more images in different scenarios into the dataset and adding new signs making the networks more complete. In the Public Road example, the use of a camera from the car or one that enables a higher stabilization will most

likely improve the results. The application of these datasets for the YOLOV4 [50] and YOLOV5 networks can also be analyzed.

Appendix A

Videos for each objective

A.1 Autonomous Driving Competition of the RoboCup Portuguese Open in simulation

<https://youtu.be/oaBd6Ub-o7E>

A.2 Autonomous Driving Competition of the RoboCup Portuguese Open

<https://youtu.be/T2USKNakM9w>

A.3 Public road

<https://youtu.be/zzIkw8sunny4>

References

- [1] A. Friedman, "Introduction to neurons," pp. 1–20, jan 2005.
- [2] Ayşegül Uçar, "The-structure-of-an-artificial-neuron.png." [Online]. Available: [https://www.researchgate.net/profile/Ayseguel\[_\]Ucar/publication/327260166/figure/fig3/AS:664442090049538@1535426747843/The-structure-of-an-artificial-neuron.png](https://www.researchgate.net/profile/Ayseguel[_]Ucar/publication/327260166/figure/fig3/AS:664442090049538@1535426747843/The-structure-of-an-artificial-neuron.png)
- [3] K. O'Shea and R. Nash, "An Introduction to Convolutional Neural Networks," nov 2015. [Online]. Available: <http://arxiv.org/abs/1511.08458>
- [4] Thomas Christopher, "An introduction to Convolutional Neural Networks | by Christopher Thomas BSc Hons. MIAP | Towards Data Science," 2019. [Online]. Available: <https://towardsdatascience.com/an-introduction-to-convolutional-neural-networks-eb0b60b58fd7>
- [5] D. Arden, "Applied Deep Learning - Part 4: Convolutional Neural Networks | by Arden Dertat | Towards Data Science," 2019. [Online]. Available: <https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2>
- [6] S. Sakib, Ahmed, A. Jawad, J. Kabir, and H. Ahmed, "An Overview of Convolutional Neural Network: Its Architecture and Applications," *ResearchGate*, no. November, nov 2018. [Online]. Available: www.preprints.orghttps://www.researchgate.net/publication/329220700
- [7] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2323, 1998.
- [8] SPR, "Robótica 2019 - Rules for Autonomous Driving," Tech. Rep.
- [9] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2016-Decem. IEEE Computer Society, jun 2016, pp. 779–788. [Online]. Available: <http://arxiv.org/abs/1506.02640>

- [10] J. Redmon and A. Farhadi, "YOLOv3: An incremental improvement," 2018. [Online]. Available: <https://pjreddie.com/yolo/>.
- [11] T. Moura, A. Valente, A. Sousa, and V. Filipe, "Traffic sign recognition for autonomous driving robot," in *2014 IEEE International Conference on Autonomous Robot Systems and Competitions, ICARSC 2014*, 2014, pp. 303–308.
- [12] J. Redmon and A. Farhadi, "YOLO9000: Better, faster, stronger," in *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, vol. 2017-Janua, 2017, pp. 6517–6525. [Online]. Available: <http://pjreddie.com/yolo9000/>
- [13] J. Cao, C. Song, S. Peng, F. Xiao, and S. Song, "Improved traffic sign detection and recognition algorithm for intelligent vehicles," *Sensors (Switzerland)*, vol. 19, no. 18, sep 2019.
- [14] J. Yang and J. F. Coughlin, "In-vehicle technology for self-driving cars: Advantages and challenges for aging drivers," *International Journal of Automotive Technology*, vol. 15, no. 2, pp. 333–340, 2014.
- [15] P. Dönmez, *Introduction to Machine Learning The Wikipedia Guide*, 2013, vol. 19, no. 2.
- [16] A. Osman, "The Role of Machine Learning in Autonomous Vehicles | Electronic Design," 2020. [Online]. Available: <https://www.electronicdesign.com/markets/automotive/article/21147200/nxp-semiconductors-the-role-of-machine-learning-in-autonomous-vehicles>
- [17] S. Chapman and N. Agashe, "Traffic Signs in the Evolving World of Autonomous Vehicles," Tech. Rep., 2019.
- [18] Nvidia, "Autonomous Car Development Platform from NVIDIA DRIVE PX2." [Online]. Available: https://www.nvidia.com/content/nvidiaGDC/sg/en_SG/self-driving-cars/drive-px/
<http://www.nvidia.com/object/drive-px.html>
- [19] K. Lim, Y. Hong, Y. Choi, and H. Byun, "Real-time traffic sign recognition based on a general purpose GPU and deep-learning," *PLoS ONE*, vol. 12, no. 3, mar 2017. [Online]. Available: [/pmc/articles/PMC5338798/](https://pubmed.ncbi.nlm.nih.gov/pmc/articles/PMC5338798/)
[https://pubmed.ncbi.nlm.nih.gov/pmc/articles/PMC5338798/](https://pubmed.ncbi.nlm.nih.gov/pmc/articles/PMC5338798/?report=abstract)

- [20] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 1, pp. 11–18, 2001.
- [21] M. Y. Zhou and W. F. Lawless, "An Overview of Artificial Intelligence in Education," in *Encyclopedia of Information Science and Technology, Third Edition*, 2014, pp. 2445–2452. [Online]. Available: https://www.researchgate.net/publication/236346414_AN_OVERVIEW_OF_ARTIFICIAL_INTELLIGENCE
- [22] Z. Mohammed, "Artificial Intelligence Definition, Ethics and Standards," Tech. Rep. [Online]. Available: https://www.researchgate.net/publication/332548325_Artificial_Intelligence_Definition_Ethics_and_Standards
- [23] P. W. and Simon, "Too Big to Ignore : The Business Case for Big Data," *Journal of Chemical Information and Modeling*, vol. 53, no. 9, pp. 1689–1699, 2013. [Online]. Available: <https://www.goodreads.com/book/show/17134962-too-big-to-ignore>
- [24] F. Chollet, *Deep Learning with Python*.
- [25] D. K. Judith Hurwitz, *Machine Learning, IBM Limited Edition*, 2018, vol. 35, no. 5. [Online]. Available: <http://www.wiley.com/go/permissions>.
- [26] O. Pentakalos, "Introduction to machine learning," in *CMG IMPACT 2019*, vol. 19, no. 2, 2019, pp. 285–288.
- [27] P. Boucher, "How artificial intelligence works," *STOA / Panel for the Future of Science and Technology*, vol. 3, no. March, p. 10, 2019. [Online]. Available: <https://www.globalme.net/blog/the-present-future-of-speech-recognition>
- [28] A. Zayegh and N. Al Bassam, "Neural Network Principles and Applications," in *Digital Systems*. IntechOpen, nov 2018.
- [29] R. E. Brown, "Donald O. Hebb and the Organization of Behavior: 17 years in the writing," apr 2020.

- [30] S. Sagar, "Activation Functions in Neural Networks | by SAGAR SHARMA | Towards Data Science," 2017. [Online]. Available: <https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6>
- [31] A. Mathew, P. Amudha, and S. Sivakumari, "Deep learning techniques: an overview," in *Advances in Intelligent Systems and Computing*, vol. 1141. Springer, 2021, pp. 599–608.
- [32] T. Pinto, "Object detection with artificial vision and neural networks for service robots," Tech. Rep., 2018. [Online]. Available: <http://repositorium.sdum.uminho.pt/handle/1822/62251><http://repositorium.sdum.uminho.pt/http://repositorium.sdum.uminho.pt/handle/1822/62251><http://repositorium.sdum.uminho.pt/>
- [33] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a convolutional neural network," in *Proceedings of 2017 International Conference on Engineering and Technology, ICET 2017*, vol. 2018-Janua. Institute of Electrical and Electronics Engineers Inc., mar 2018, pp. 1–6.
- [34] R. Sathya and A. Abraham, "Comparison of Supervised and Unsupervised Learning Algorithms for Pattern Classification," *International Journal of Advanced Research in Artificial Intelligence*, vol. 2, no. 2, 2013.
- [35] S. Devin, "Supervised vs. Unsupervised Learning | by Devin Sony | Towards Data Science," 2018. [Online]. Available: <https://towardsdatascience.com/supervised-vs-unsupervised-learning-14f68e32ea8d>
- [36] A. Bisht, "ML | Classification vs Regression - GeeksforGeeks." [Online]. Available: <https://www.geeksforgeeks.org/ml-classification-vs-regression/><https://www.geeksforgeeks.org/ml-classification-vs-regression/><https://www.geeksforgeeks.org/ml-classification-vs-regression/?ref=rp>
- [37] H. Rezatofighi, N. Tsoi, J. Gwak, A. Sadeghian, I. Reid, and S. Savarese, "Generalized intersection over union: A metric and a loss for bounding box regression," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2019-June, 2019, pp. 658–666.

- [38] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1137–1149, 2017. [Online]. Available: <http://image-net.org/challenges/LSVRC/2015/results>
- [39] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2016-Decem. IEEE Computer Society, dec 2016, pp. 770–778. [Online]. Available: <http://image-net.org/challenges/LSVRC/2015/>
- [40] SAE International, "Surface Vehicle," *SAE International*, vol. 4970, no. 724, pp. 1–5, 2018.
- [41] T. Denton and T. Denton, "Advanced driver assistance systems (ADAS)," in *Automated Driving and Driver Assistance Systems*, 2019, pp. 21–33. [Online]. Available: www.maximintegrated.com/ADAS
- [42] Tesla Inc., "2019-Tesla-Impact-Report," p. 59, 2020. [Online]. Available: https://www.tesla.com/ns_videos/2019-tesla-impact-report.pdf
- [43] S. Ingle and M. Phute, "Tesla Autopilot : Semi Autonomous Driving, an Uptick for Future Autonomy," *International Research Journal of Engineering and Technology*, vol. 3, no. 9, pp. 369–372, 2016. [Online]. Available: www.irjet.net
- [44] O. Eye, O. Eye, E. Ncap, O. Eye, O. Eye, and I. O. Eye, "Opel Eye," vol. 1, no. 2, pp. 40–41, 2010.
- [45] SPR, "Festival Nacional de Robótica," 2019. [Online]. Available: https://www.festivalnacionalrobotica.pt/?lang=enhttp://www.sprobotica.pt/index.php?option=com_content&view=article&id=108&Itemid=62
- [46] Z. Q. Zhao, P. Zheng, S. T. Xu, and X. Wu, "Object Detection with Deep Learning: A Review," pp. 3212–3232, nov 2019.
- [47] G. A. Miller, R. Beckwith, C. Fellbaum, D. Gross, and K. J. Miller, "Introduction to wordnet: An on-line lexical database," *International Journal of Lexicography*, vol. 3, no. 4, pp. 235–244, dec 1990.

- [48] I. Khokhlov, E. Davydenko, I. Osokin, I. Ryakin, A. Babaev, V. Litvinenko, and R. Gorbachev, "Tiny-YOLO object detection supplemented with geometrical data," *IEEE Vehicular Technology Conference*, vol. 2020-May, aug 2020. [Online]. Available: <http://arxiv.org/abs/2008.02170><http://dx.doi.org/10.1109/VTC2020-Spring48590.2020.9128749>
- [49] W. Fang, L. Wang, and P. Ren, "Tinier-YOLO: A Real-Time Object Detection Method for Constrained Environments," *IEEE Access*, vol. 8, pp. 1935–1944, 2020.
- [50] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "YOLOv4: Optimal Speed and Accuracy of Object Detection," apr 2020. [Online]. Available: <https://arxiv.org/abs/2004.10934v1><http://arxiv.org/abs/2004.10934>