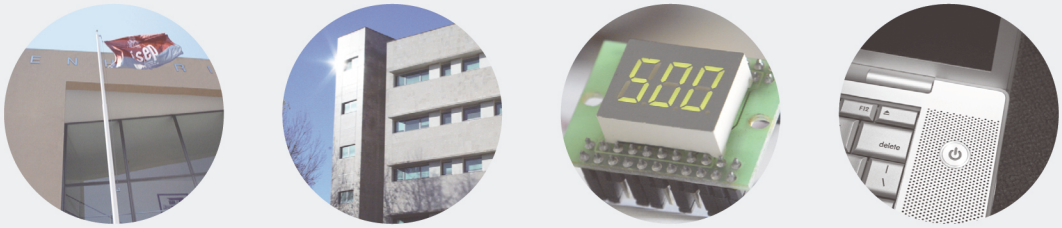
INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES



# Deep Learning em Testes Automatizados para a Indústria Automóvel

**BEATRIZ CUNHA BRANCO**
setembro de 2023

POLITÉCNICO
DO PORTO

INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES

# Deep Learning in Automated Tests for the Automotive Industry

**BEATRIZ CUNHA BRANCO**
Setembro de 2023

POLITÉCNICO
DO PORTO

Politécnico do Porto
Instituto Superior de Engenharia do Porto

# Deep Learning in Automated Tests for the Automotive Industry

**Beatriz Cunha Branco**

Master in Electrical and Computer Engineering
Specialization Area of Automation and Systems


Departamento de Engenharia Eletrotécnica
Instituto Superior de Engenharia do Porto

September 23, 2023

*This dissertation partially satisfies the requirements of the Thesis/Dissertation course of the program Master in Electrical and Computer Engineering, Specialization Area of Automation and Systems.*

**Candidate:** Beatriz Cunha Branco, No. 1180973, 1180973@isep.ipp.pt

**Scientific Guidance:** Ramiro Barbosa, rsb@isep.ipp.pt

**Company:** Capgemini Engineering

**Advisor:** Raquel Ribeiro and João Oliveira,
raquel.ribeiro@capgemini.com//joao.lemosoliveira@capgemini.com


Departamento de Engenharia Eletrotécnica
Instituto Superior de Engenharia do Porto
Rua Dr. António Bernardino de Almeida, 431, 4200–072 Porto

September 23, 2023

# Acknowledgements

This dissertation marks the end of my academic life and even though that is a bittersweet feeling I am proud of what I achieved until now that, without a doubt, would not be possible without everyone that had some kind of impact on my life. And so, I want to express my sincere gratitude to those who stood by me not only during my academic journey but also during my entire life.

First, I would like to give my sincere thank you to Capgemini Engineering, the company that believed in me since the day I finished my Bachelor's degree. During these more than two years as an employee, I learned, grew and met wonderful people, especially Raquel Ribeiro, the one who has been guiding me since day one. Thank you for all the shared knowledge and moments that we spent together, learning together and laughing together. I would also like to give my thanks to the Verification and Validation department, emphasizing my manager João, for all the opportunities that he gave me and made possible for me.

Then, I would like to thank my second home for the past five years, for all the friends that it gave me and the professionals that I encountered. To these friends for life, thank you for believing in and helping me, you made my experience in university even better! To the great professionals of the university, I would like to give my special acknowledgement to *Prof. Dr.* Ramiro Barbosa, not only for the support during the thesis, clarification and availability but also for all the knowledge transferred now and during the subjects lectured where I had the opportunity of being a student. But, most importantly, thank you for believing in me and in my work.

To my parents, words are not enough. Thank you mom and dad for giving me the opportunity to study what I wanted and liked and for being my biggest supporters in everything. For sharing with me all my conquers and for being the reason why I can achieve them in the first place. Thank you for giving me a good and safe environment that allowed me to grow happy and become who I am now, I will never be able to thank you enough or do as much as you do for me everyday.

Finally, to my boyfriend João, who has been by my side for more than four years, pulling me up when I am down, giving me the motivation when I had none and believing in me and that I would be capable when even I did not. Thank you for being my best friend, for putting your things in second to help me and for listening to me. I can say with all certainty that I would not be capable of doing this without

you.

# Abstract

Artificial Intelligence (AI) usage has increased over the years, whether in speech recognition, predictive analytics and mainly image recognition, being useful for industries such as the medical industry, for medical diagnosis, the automotive industry, for autonomous driving, traffic signals detection and also for an advanced driver assistance systems that can help prevent accidents.

The automotive industry is one of the most popular industries in our society, either by the convenience that it gives to people on their travels, the designs of the vehicles themselves and nowadays, mostly the technology that it offers. And, with this increase in technology the rise of technical failures also occurs, which makes one step of the production of any kind of vehicle more important than ever, testing, which caused the companies to start investing in automated tests in order to decrease both their costs in a long term and human effort.

There are a lot of techniques when it comes to automated testing, but with the increase in the popularity of Deep Learning (DL) and Machine Learning (ML), automated software testing using this branch of AI has started to gain popularity as well. Therefore, the project documented in this report has the goal of using a DL network that can help to perform automated software tests in Android Automotive infotainment together with Python and its dedicated frameworks, which can then increase the quality of testing but also decrease the human effort. Besides this, reports with the results of the executions and the suggestion of tests' names based on the network output will be implemented.

**Keywords**: Artificial Intelligence, Machine Learning, Deep Learning, Automotive Sector, Automated Software Testing, Android Automotive, Python.

# Resumo

O uso de Inteligência Artificial (IA) tem vindo a aumentar ao longo dos anos, quer seja no reconhecimento de voz, análise preditiva e principalmente, no reconhecimento de imagens, sendo este último útil em indústrias tais como a médica, para diagnósticos e também a automóvel, para a condução autónoma, deteção de sinais de trânsito e também para sistemas de assistência em viagem que pode prevenir acidentes.

A indústria automóvel é uma das mais populares na nossa sociedade, quer pela conveniência que oferece às pessoas nas suas viagens, pelo *design* dos próprios veículos e atualmente, principalmente pela tecnologia que oferece. Assim, com o aumento da tecnologia o crescimento de falhas técnicas também acontece, o que faz com que um passo da produção, a testagem, seja mais importante do que nunca, o que fez com que as empresas começassem a investir em testes automatizados de forma a diminuir tanto os custos a longo prazo, mas também o esforço humano.

Existem bastantes técnicas quando se trata da automatização de testes, no entanto, com o aumento da popularidade de *Deep Learning* (DL) e *Machine Learning* (ML), os testes de *software* automatizados utilizando este ramo da inteligência artificial também começaram a ganhar popularidade. Dessa forma, o projeto documentado no presente relatório tem como objetivo o uso de uma rede de DL que, juntamente com a utilização de Python e respetivas *frameworks*, seja capaz de auxiliar à execução de testes de *software* automatizados num *infotainment* Android Automotive, podendo levar assim à melhoria da qualidade dos testes executados e diminuição do esforço humano. Para além disso, relatórios com os resultados das execuções e a sugestão de nomes de testes baseada no resultado da rede irão ser implementados.

**Palavras-Chave**: Inteligência Artificial, *Machine Learning*, *Deep Learning*, Setor Automóvel, Testes de *Software* Automatizados, Android Automotive, Python.

# Contents

# List of Figures

# List of Tables

# Listings

# List of Acronyms

| | |
|---|---|
| **adb** | Android Debug Bridge |
| **AI** | Artificial Intelligence |
| **AP** | Average Precision |
| **CAN** | Controller Area Network |
| **CNN** | Convolutional Neural Network |
| **CPU** | Central Processing Unit |
| **CSP** | Cross Stage Partial |
| **DL** | Deep Learning |
| **ECU** | Electronic Control Unit |
| **FN** | False Negative |
| **FP** | False Positive |
| **FPN** | Feature Pyramid Network |
| **GPU** | Graphics Processing Unit |
| **HiL** | Hardware-in-the-Loop |
| **HMI** | Human Machine Interface |
| **IA** | Inteligência Artificial |
| **IoU** | Intersection over Union |
| **ISEP** | *Instituto Superior de Engenharia do Porto* |
| **LSTM** | Long Short-Term Memory |
| **mAP** | Mean Average Precision |
| **MBTCC** | Model-Based Test Case Creation |
| **MiL** | Model-in-the-Loop |

| | |
|---|---|
| **ML** | Machine Learning |
| **OEMs** | Original Equipment Manufacturers |
| **PANet** | Path Aggregation Network |
| **PiL** | Processor-in-the-Loop |
| **R&D** | Research and Development |
| **RNN** | Recurrent Neural Network |
| **RoI** | Region of Interest |
| **RPN** | Region Proposal Network |
| **RvNN** | Recursive Neural Network |
| **SDLC** | Software Development Life Cycle |
| **SiL** | Software-in-the-Loop |
| **SiLU** | Sigmoid Linear Unit |
| **SMArDT** | Specification Method for Requirements, Design and Test |
| **SPP** | Spatial Pyramid Pooling |
| **SSD** | Single Shot Detector |
| **TN** | True Negative |
| **TP** | True Positive |
| **UDP** | User Datagram Protocol |
| **V&V** | Verification and Validation |
| **VPN** | Virtual Private Network |
| **YOLO** | You Only Look Once |

# Chapter 1

# Introduction

In this Chapter, the Master's Thesis developed in the second year of the Master Degree in Electrical and Computer Engineering in the field of Automation and Systems, of the Electrical Engineering Department of *Instituto Superior de Engenharia do Porto* (ISEP) is presented.

The project had its focus on using a Deep Learning (DL) network capable of improving the execution of automated tests in an Android Automotive Infotainment, with the goal of being used in the Verification and Validation (V&V) Department of Capgemini Engineering.

## 1.1 Context

Over the years the vehicle, whether it's a car, bus or even a motorcycle, has become more and more a part of our everyday life. With the technology growth the vehicle as we knew before needed to improve as well to respond to people's needs, leading to complex engineering projects to innovate in the automotive industry, especially in the electronics of the car and its embedded software.

Nowadays the major part of vehicle innovations are in the entertainment field, thus Original Equipment Manufacturers (OEMs), such as BMW, had the need to strengthen the way they test their products by improving their techniques and hiring new companies to help them to be able to test all the software included in a vehicle. However, most companies still perform the major part of the tests manually, which takes a lot of time and as a consequence it's expensive.

In a study, that had the purpose of showing the current open issues of automotive software testing [1], the authors mention the biggest open issues for automotive software testing, emphasizing the challenges of Human Machine Interface (HMI) testing, which is in constant development and growth since it needs to answer to all consumer desires, such as having the latest audio and video capabilities in the vehicle and where one of the most important requirements is that the experience provided to the user is faultless. Since the infotainment system HMI has the concept of screens and can include multiple screens, as well as menu and background changes, these screens and screen changes need to be tested and this multiple options testing is difficult to verify manually. Another challenge pointed out by the authors was the fact that the HMI is an embedded system that communicates with different Electronic Control Units (ECU) via underlying applications and therefore the dynamic menu's behaviour is dependent on these applications.

Together with all of these challenges, companies still need to face several HMI error types, such as unpredictable menus, screen content and design errors that only increases the facts of why manual testing is time-consuming and not effective for the automotive industry. Finally, the authors in this same article mention different approaches to deal with all of the mentioned problems, one being the capture and replay alternative approach where, while the tests are executed manually, they are recorded so that later they can be replayed, this being the starting idea of the company for this project.

Capgemini Engineering [2], the promoter of this work, is a part of the Capgemini Group, a world leader in developing partnerships with companies, such as OEMs, to transform and enable them to manage their businesses by harnessing the power of technology. Capgemini Engineering is a world leader in Engineering and Research and Development (R&D) services that counts more than 52000 Engineers and Scientists in more than 30 countries in sectors such as aeronautics, railways, communications, life sciences, pharmaceutical, and mainly automotive.

The proponent continues to innovate in ways of performing automated tests for the HMI systems in a vehicle, and therefore acknowledged the need for a solution capable of decreasing even more the use of manual testing using one of the most utilized techniques today: Deep Learning (DL).

## 1.2 Definition of the Problem

As the automotive industry approximately represents 80% of the projects currently existing in the company, this has a great impact on it and especially on the V&V department, since it allocates a large part of its resources to this area. Thus, the question of how to innovate in the verification and validation procedures of this large

sector is a constant, and the present project emerged as one of the answers to this question.

The automation of tests has great prominence in all projects in this sector; however, it is still time-consuming and difficult. Firstly because, the automated tests have to be previously developed by someone manually, and techniques like the four-eye principle, where two people must verify the work before it is delivered, need to be used, both in code and in verification, in order to prevent errors. Also, because each functionality to be tested has hundreds of tests associated, and by manually developing automated test cases and then executing them, the forgetting of the execution of some tests can easily happen. In this way, this report aims to explain how it will be possible to solve these problems and the advantages that this approach presents.

### 1.2.1 Objectives

In order to be able to solve the presented problem, the following objectives are proposed:

- Development of a tool able to stream the display under test and also record each icon pressed, being this the input of the neural network;
- Usage of a DL algorithm to predict the position of the different icons in the display under test as well as the type of test to be executed;
- Development of an algorithm so that, the automated tests are created based on a recording of a manual test, in order to save time and make the creation of automated tests easier;
- Organization of reports capable of showing the result of the executions.

It should be noted that the main purpose of this project is to assist the Engineers in the department with the interface testing for Android Automotive Infotainment and not replace their work, as manual tests need to be performed by someone as well as reports' analysis to confirm that the testing is performed properly.

### 1.2.2 Expected Results

After completing the project, it is expected that the system will be trained in a way that it's able to recognize the icons of an Android Automotive display, associate each icon to a certain coordinate, and then be able to execute the test automatically and therefore be able to perform most of the interface test on an Android Automotive Infotainment on its own. It should be able to give test name suggestions to the user based on the output of the network.

Finally, it should be possible for the user to read the test reports and verify and validate whether a feature was correctly tested.

## 1.3 Plan of Work

The plan of work, visible in Figure 1.1, is composed of the main tasks of the project with the respective workload associated with each task. Also, it is necessary to take into account that some of the main tasks were dependent on smaller ones.

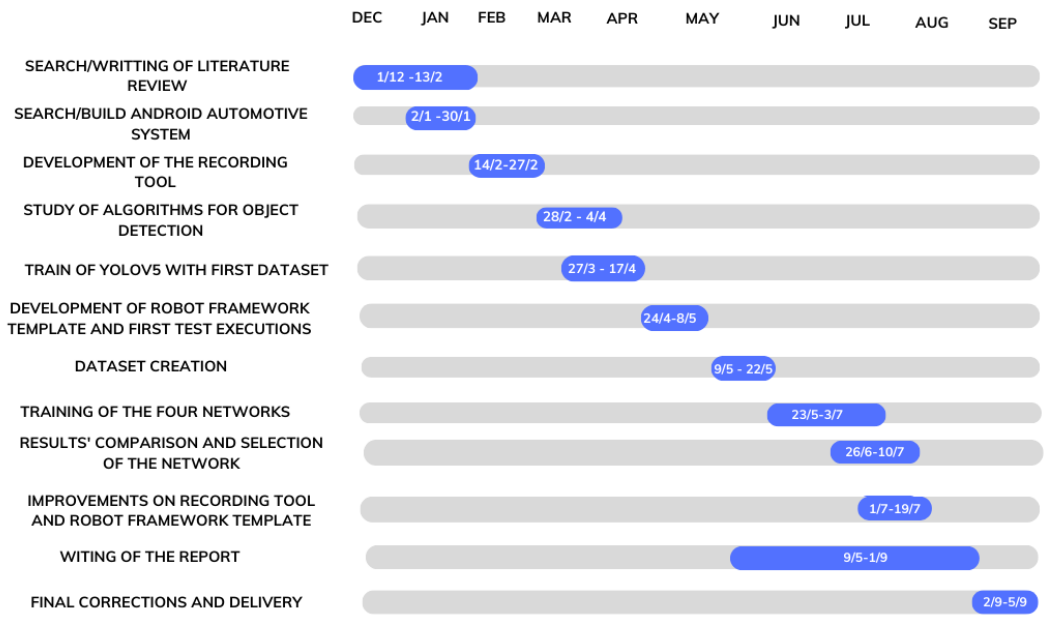



Figure 1.1: Plan of Work.

## 1.4 Organization of the Dissertation

In Chapter 1 an introduction of the project is made, giving a brief context of the topic and highlighting the objectives as well as the expected results.

In Chapter 2 a theoretical presentation of important topics for the project will be presented, as well as the analysis of both machine and deep learning in the automotive sector. Finally, a comparison between infotainments in Linux and Android will be presented, and the reason why Android Automotive infotainments are starting to be more used by manufacturers.

Then, in Chapter 3 the previously developed work is analysed and a context of the starting point is made. Still in this chapter, the work to be developed is presented in detail.

In Chapter 4 the neural networks used in the project will be explained, as well as some important concepts related to DL that are needed to take into account while training the network.

Chapter 5 will present the different phases of the project, including the implementation for each stage, tests done and final results.

Finally, in Chapter 6 final considerations of the project will be made, highlighting the difficulties upon its implementation and the achieved goals. Also, future work on the project will be presented.

# Chapter 2

# Literature Review

In this Chapter, concepts inherent to the project such as Machine Learning (ML), Deep Learning (DL) and automated software testing will be presented and deeply analyzed. Thus, the influence that these concepts and techniques have in the automotive industry as well as the impact that they have on the presented project will be mentioned.

## 2.1 Automated Software Testing

Software programs have become a major part of our lives, impacting millions of people everyday, therefore, no matter what is the software or the process chosen to test it, testing is always a phase of high importance to guarantee that the final product works as expected and, most importantly, to ensure its reliability.

Software testing consumes 30 to 60 percent of all life-cycle costs, depending on product criticality and complexity [3], and it is defined as being the process of evaluating a software program with the purpose of finding faults or errors [4], since software errors (commonly called bugs) have a big negative impact in the final product and, if not discovered early in the development phase, the cost to detect and fix them increases with time. For instance, a study conducted by [5], in about 675 companies, to evaluate software quality in 2012, shows that software with more bugs, due to lack of testing, has the lowest cost per defect, however, since the quality of the product is low, the price of fixing the defects can be seven times higher when

comparing to a high-quality software, leading to big losses. Being the study conducted eleven years ago, when the amount of software daily was significantly lower, in case the techniques stayed stalled, the amount of money lost for the companies would be unaffordable.

Software testing is also a crucial phase in the Software Development Life Cycle (SDLC) [6] which consists of a process of building and maintaining software systems and typically includes various phases from preliminary requirements' analysis to post-development like software testing and evaluation. In the modern SDLC, there are two different methodologies: the traditional development methodology, like the Waterfall, and the Agile methodology.

The traditional SDLC methodologies are usually used in large-scale projects and are based on sequential steps, which means that in order to pass to the next step, the previous one needs to be completely done, causing the need to have stable requirements at the beginning of the project. Normally, there are four characteristic phases, visible in Figure 2.1.

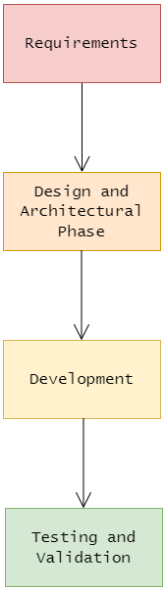



Figure 2.1: Traditional SDLC Phases.

The SDLC phases work as follows:

- Requirements - crucial to determine the time needed to implement all the other phases, as it defines what needs to be done to fulfil the final goal;
- Design and Architectural Phase - presentation of a roadmap and potential issues that the project might face;
- Development - production of the code until a certain goal is reached;

- Testing and Validation - evaluation of the developed software to verify if it runs as intended and if it meets the customer requirements.

The Agile SDLC is based on the idea of incremental and iterative development, in which the phases are revisited when needed, being customer feedback a great part of the improvement during the cycle. This type of methodology is divided into small steps, usually called iterations, possible to see in Figure 2.2, being the major factors of Agile SDLC methodology the early customer involvement, iterative development, self-organizing teams, and finally the adaptation to change.

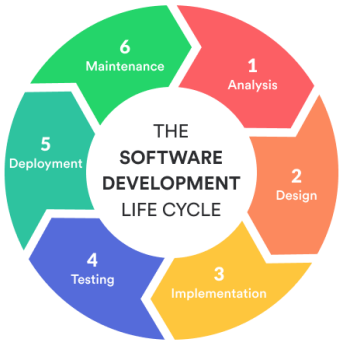



Figure 2.2: Agile SDLC Phases [7].

Software testing has the goal of making sure that software performs for its intended purpose, to achieve and preserve its quality, to verify if it is fit for use, but not to demonstrate that the system is free from errors, in fact, a system with no errors means that either no testing or poor testing was done [4]. Software testing can be manual or automated. In the case of manual testing, no knowledge of any tool is required, but on the other hand, a lot of human effort is necessary, whereas automated testing, which consists of the process of using software separate from the software under test to control the execution of tests and the comparison of actual outcomes with the expected outcomes [8], knowledge of automation tools and programming skills are needed, however, it saves time and effort of the tester.

It is possible to split software testing [4, 9] into two different types:

- Functional Testing - goal of verifying each functionality of the software and if each function works as per the requirements;
- Non-Functional Testing - goal of testing the quality and performance of the software.

In each one of these categories, it is possible to enhance different test levels, as visible in Figure 2.3.

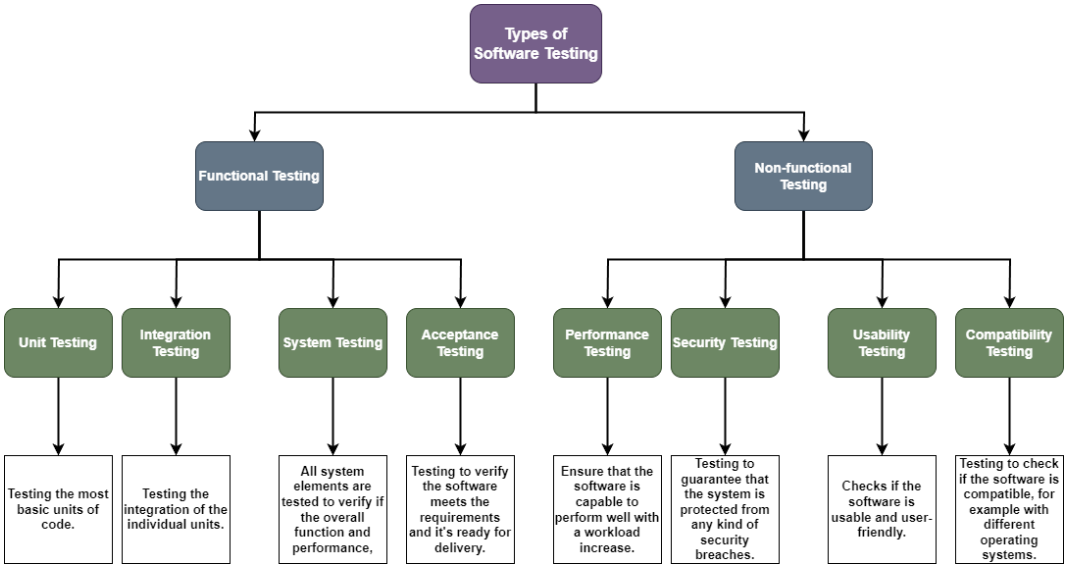



Figure 2.3: Software Testing Techniques.

In order to automate all the mentioned test levels, there is a need to use automation tools so that automated software testing can be possible. An automation tool is defined as software that helps to test the actual software in focus, and the correct automation tool along with the correct testing technique allows the software to meet nowadays' quality standards. Some of the most used tools are described in Table 2.1.

Table 2.1: Most used automation tools.

| **Tool** | **Usage** |
|---|---|
| JUnit. | Used for Java programming language. |
| Selenium. | Used for testing web applications and compatible with multiple languages. |
| Unified Functional Testing. | Most popular commercial automation tool for functional testing. |
| Robot Framework. | Python-based framework keyword driven for acceptance testing. |

As mentioned, automated software testing has great advantages [4] when compared with manual testing, such as the fact that it allows companies to save time and money by making the testing process more efficient because, for instance, an automation script is repeatable, and the fact that it improves the accuracy and enables the quick finding of bugs. However, because not all test cases can be easily automated, especially when requiring knowledge from a specific domain [10], this method still has some disadvantages [4] such as the fact that the knowledge of the tool used for automation is needed, the cost of buying and maintaining the tools is expensive

and choosing the correct testing technique as well as the correct automation tool requires a great effort.

A summary of the advantages and disadvantages of automated software testing can be seen in Table 2.2.

Table 2.2: Advantages and Disadvantages of Automated Software Testing.

| **Advantages** | **Disadvantages** |
|---|---|
| Allows companies to save time and money. | Not all test cases can be easily automated. |
| Makes the testing process more efficient. | Knowledge of the automation tool is needed. |
| Automation script is repeatable. | Buying and maintaining the tools is expensive. |
| Improves accuracy and enables quick finding of bugs. | Choosing the correct automation tool and technique requires a great effort. |

### 2.1.1 Automated Software Testing in the Automotive Industry

The automotive industry [11, 12, 13, 14] is usually linked to electrical and mechanical components that together create the vehicle. However, in recent years, this industry has become a combination of software, electrical and mechanical engineering, where the software present in each car can reach up to 100 million lines of code. So, with this complexity level and with human safety requirements increasing, testing is more crucial than ever to successfully engineer reliable automotive software.

The development of automotive systems [12] requires that the software and hardware components are designed simultaneously, which results in a need for an iterative process and temporary software releases until the final release is ready to be implemented and put to the market. In every one of these releases, testing is necessary to ensure that requirement and design inconsistencies and implementation faults can be detected as soon as possible since, the more advanced the testing phase of the product is, the more expensive it is to repair the discovered faults. This means that the same tests need to be repeated multiple times during the entire development cycle, which makes test automation essential since the number of iterations and the workload needed if performing manual tests would be unpractical and expensive, making the quality assurance standards practically unreachable. Therefore, automated testing's most relevant advantage in this industry [11] is improved testing accuracy, since unlike a human the computer is not influenced by the routine of performing repetitive tasks and also by the fact that testing in this industry sometimes requires test scenarios with specific and precise sequences of actions that are time-sensitive that cannot be performed manually. Furthermore, test automation

simplifies the coordination between car manufacturers and suppliers during development since every ECU sample that is delivered by the supplier must fulfil as a minimum the acceptance test already defined before being integrated into a car [12].

In this sector like in many others model-based testing is used which consists of a testing technique [15] where the run time behaviour of the software under test is checked against predictions made by a model, that corresponds to the description of a system's behaviour, such as input sequences, actions and flow of data from input to output. Using other words, the model-based testing technique describes how a system behaves when responding to an action. By using this approach, it is possible to test the functionality in the model before the software is implemented and integrated into the final ECU.

However, automotive model-based development has specific characteristics that require the use of dedicated model-based testing approaches. For starters, it uses a V-Model as a development process, which is inspired by the Waterfall model [16] (Figure 2.4) where the whole process is divided into separated phases being the output of one phase the input of the other one, meaning that it is a sequential model.

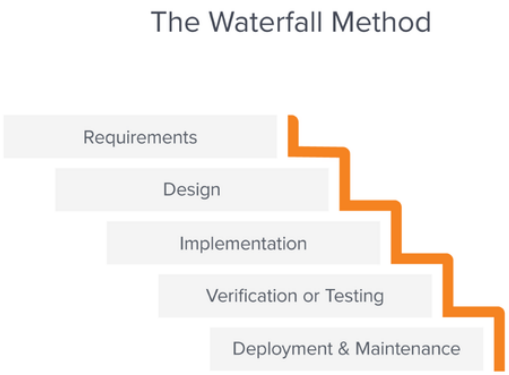



Figure 2.4: Waterfall model [16].

Then, some other points need to be taken into account, such as:

1. The functionality of the system should remain constant and independent of the integration level;

2. Relevant test cases should be constant through the integration and implementation;

3. The model-based development used in the automotive industry should support the portability of reusable test cases between the different platforms, in order to save time, money and reduce the effort.

Figure 2.5 represents the V-Model used in the Automotive Industry.

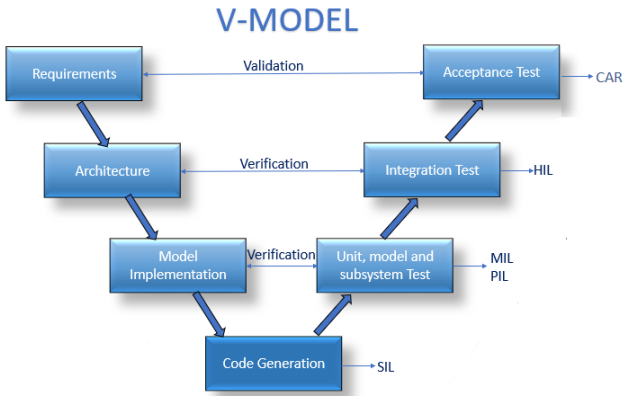



Figure 2.5: Automotive V-Model.

In addition to all the particularities of the model-based development in this sector, it is also possible to distinguish different integration levels that allow the testing to be completed in a vehicle, such as [12, 17]:

- Model-in-the-Loop (MiL) - used to test the controller logic on the simulated model and consists of developing a model of the hardware in a simulation environment to capture the most important features of the hardware system;
- Software-in-the-Loop (SiL) - the embedded software is tested in the simulated model created previously. It is used to verify if the control logic implemented can be converted into code and if the hardware can be implemented. In this phase, the input-output should practically match the input-output of the MiL phase, and if not, the model in the first level should suffer any necessary changes and the SiL testing should be redone;
- Processor-in-the-Loop (PiL) - consists of the integration of embedded processors in the model to run a closed-loop simulation with the simulated model. By doing tests on PiL faults caused by the target compiler (embedded processor) or by the processor architecture can be detected, and if this is the case the two previous steps need to be reworked once again. This is also the last integration level that allows debugging during tests with low costs, which makes this the phase where the higher effort is spent;
- Hardware-in-the-Loop (HiL) - in this level, the software runs on the final ECU, however, the surrounding environment remains simulated and the communication between the two, the real ECU and the simulated environment, is done via digital and analogue electrical connectors using protocols like Controller Area Network (CAN) and User Datagram Protocol (UDP) and it has the goal of revealing defects in the low-level service of the ECU;

- Car - the last level is the car itself, after going through all stages the final ECU runs in the real car, which can either be a sample or a car from the production line.

Customers more and more request new cars with more functionalities at similar costs, which requires companies to do an expense reduction yearly, therefore, reducing the cost and time of software development when increasing the complexity of the systems is the main goal of the companies in the automotive sector. Because of that, several surveys were conducted over the years to understand which test methods were used in the industry and how to improve the testing.

The authors in [18] created a questionnaire answered by 68 participants which included people working on OEMs, suppliers, engineering service providers, universities and specialized departments and partners focused on testing and requirements engineering. The survey was conducted in 2014 and it included 24 open questions with the main goal of understanding the use of testing methods and tools in the respective departments but, it also had the purpose of comprehending the use of test automation within the automotive industry. Around ten test automation tools were referred by the inquiries, such as: Polyspace tool from Mathworks [19], which is a static code analysis tool to prove the absence of errors in the code, CANoe [20] which is a tool for development, test and analysis of individual ECUs and entire ECU networks and Jenkins [18] that consists in an open source build server, being Polyspace the most referred one.

By analysing the obtained results, the authors were able to conclude that automated test execution was heavily in use by the participants, but automated test case generation was not.

In 2018 a study was conducted at BMW [13] where the authors had the purpose of creating a model-based improvement of the testing activities, based on the answers from the test engineers working there. Until then, BMW used a four-layer specification method, Specification Method for Requirements, Design and Test (SMArDT), developed by the company, which describes a semiformal specification for requirement, design, and testing according to an ISO norm. For this study, 196 professional test engineers and test managers of BMW answered 27 questions, and based on the answers, the authors were able to adapt the first two layers of the SMArDT method, to enable the increasing of the development quality by extending it with automated generation of tests, named Model-Based Test Case Creation (MBTCC), being possible to see, in Figure 2.6, the differences between the original process for test case creation (on the left) and the newly developed method (on the right).

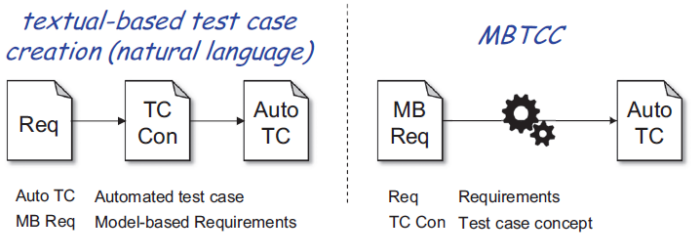



Figure 2.6: Original vs new test case creation [13].

By presenting the new model, the authors were able to conclude that independently of the experience level, professional background and focus of testing, more than one-third of the participants expect to benefit from MBTCC. Also, by analysing the survey the authors got the information that a good portion of the test engineers used old test cases, personal experience and error description as a base for the test case creation, which made it possible for them to conclude that new requirements only have small changes based on the old ones, making the MBTCC model developed even more useful, since it could directly create new test cases instead of adjusting old ones.

The automotive industry suffered big changes over the years, and the companies had to adapt to the technological growth. Manual testing was enough to guarantee software quality in the beginning but with the increase complexity and usage of vehicles, it was no longer a sustainable approach for the companies and therefore automated software testing was adopted. Also, the initial techniques had to be improved to achieve more test coverage and efficiency while also some methods based on machine learning and DL were created to assist in testing regards.

## 2.2 Machine Learning and Deep Learning

Artificial Intelligence (AI) includes any technique that allows computers to imitate human behaviour and reproduce or exceed human decision-making to solve certain tasks with minimum or no human intervention. Therefore ML is a field of AI [21] that can be described as the ability of systems to learn from training data for a specific task in order to solve the associated problem, without the machine being directly programmed and many other concepts used nowadays are based on it, such as DL which unlike ML uses artificial neural networks.

In ML the performance of the program improves with experience, so this field of AI aims to automate the tasks of analytical model building in order to perform rational activities like object detection, natural language processing or speech recognition, by applying algorithms that repetitively learn from the problem-specific training data, allowing computers to find hidden perspectives and patterns without

being explicitly programmed. This technique shows good results when it comes to tasks related to high dimensional data such as classification, since learning from previous computations and extracting regularities from big databases can help to make reliable and repeatable decisions leading to good results mostly in the mentioned areas.

DL consists of a subset of ML [22] that is inspired by the way information is processed and patterns are found in the human brain. It uses a large amount of data to map the given input to a specific label, in other words, deep neural networks are fed with raw input data and are capable of discovering the correct representation for it. This subfield of AI is designed using numerous layers of algorithms (artificial neural networks) and each of them provides a different interpretation of the data given to it.

Figure 2.7 represents a graphical distinguishment between ML and DL where the most visible difference settles on the feature extraction since in the first case it is manually done whereas in the second case, it is automatically done by the network.

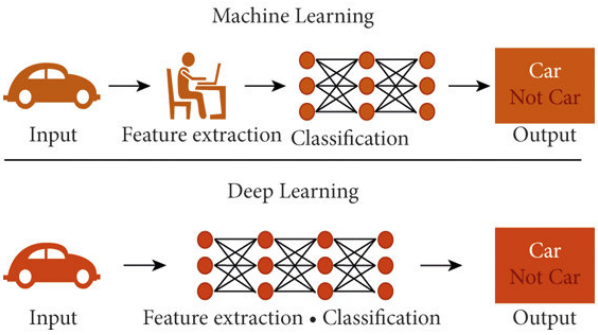



Figure 2.7: ML vs DL [23].

When it comes to DL networks it is possible to split them into different types [22], being these the Recursive Neural Network (RvNN), Recurrent Neural Network (RNN) and Convolutional Neural Network (CNN).

RvNN is capable of doing predictions in a hierarchical structure and classifying the outputs utilizing vectors. The architecture in this type of network is generated for processing objects with specific structures such as graphs and trees, where this approach generates a representation with a fixed width from a variable-size data structure. The training of the network is done by auto-association and it allows the generation of the pattern of the input layer in the output one and because of that, since it permits the reduction of the network depth, these networks are widely used in natural language processing. In Figure 2.8 it is possible to see the transformation from a tree or graph structure into the network itself.

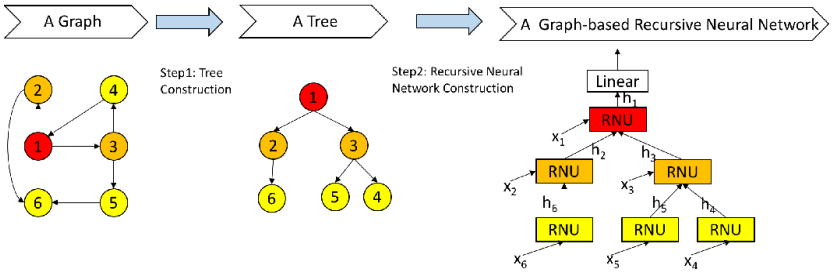



Figure 2.8: Recursive Neural Network [24].

When mentioning RNN, Figure 2.9, [22] differentiates from the conventional networks since the data used is sequential, making this network useful in applications such as speech recognition since it allows the comprehension of a sentence by knowing the position of each specific word. However, this method makes the network accuracy decay over time, since with the entrance of new inputs, the network stops thinking about the old ones.

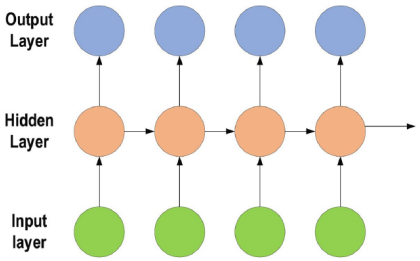



Figure 2.9: Recurrent Neural Network [22].

The most employed network is the CNN [22], since when compared to the previously mentioned ones, it is capable of identifying relevant features without any human supervision, making it extensively applied in fields such as computer vision, speech processing, face recognition and also automotive visual applications such as object, vehicle and road-marking detection. The structure of the CNN can be seen in Figure 2.10 and was inspired by human neurons and cat brains and like in these cases, CNN share weights and local connections mostly for two reasons: to completely use the input-data structure and to use a small amount of parameters which simplifies the training process and speeds up the network.

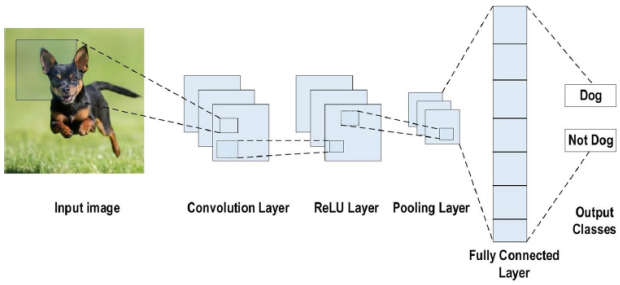



Figure 2.10: Convolutional Neural Network [22].

The first convolution layer represents the most significant component in the architecture and it consists of multiple convolutional filters (kernel), where the input image is convolved with these filters to generate the output features map. This layer brings several concepts, including these:

- Kernel - described by a grid of discrete numbers where each value, that is randomly assigned at the beginning of the training process, is denominated as the kernel weight. Every weight is adjusted at each epoch, making it possible for the kernel to learn or extract significant features;
- Convolutional Operation - being the input of the network a multi-channelled image, this operation consists of the process of sliding the image both vertically and horizontally by the kernel. With this process, which is repeated until no further sliding is possible, the dot product between the input image and the kernel, that represents the feature map of the output, is determined;
- Sparse Connectivity - describes the small availability of weights between two adjacent layers, which makes the number of required weights or connection small making the memory needed for storing the weights small;
- Weight Sharing - one of the biggest advantages of these types of networks is the no allocation of weights between any two neurons of consecutive layers since the entire weights operate as a single group which, as mentioned, decreases the required training time.

The ReLU layer represents the most common function used in CNN and it converts the whole values of the input into positive numbers requiring low computational load. The third layer has the main task of sub-sampling the feature maps, which means that the main activity of the pooling layer is to reduce the existing feature map creating smaller ones, in vector form, and simultaneously maintain the majority of the dominant information in every step. The final of the network is given by the output of the fully connected layer that has as an input the vector created in the pooling layer. Inside the last layer, each neuron is connected to all neurons of the previous one, making it possible to see the described behaviour in Figure 2.11.

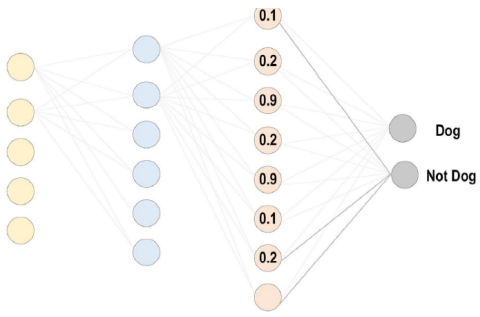



Figure 2.11: Fully Connected Layer [22].

The system, no matter whether using DL or ML, can learn using different learning techniques, including these supervised, semi-supervised, unsupervised learning and reinforcement learning.

The first one, supervised learning [25], learns from labelled training data, which means that the data already is tagged with the correct answers, to help predict the outcomes for unpredicted data. Since this type of learning uses datasets that contain inputs and the correct outputs, the model learns faster, being this and the fact that in this type of learning the possibility to collect or produce a data output from the previous experience, the main advantages for its usage, however, it also has some disadvantages including the fact that there is the chance of having irrelevant input features in the training data that can lead into inaccurate results.

Contrary to the previously described learning method, the unsupervised learning technique [26] does not need any labelled data as it allows the model to work alone and discover patterns and information previously undetected, and therefore it allows the users to perform more complex tasks such as clustering and anomaly detection, however when comparing to the previous method, the accuracy of the model decreases since the input data is not labelled by people in advance.

Since both supervised and unsupervised learning come with great advantages, but also with some disadvantages a new learning method started being used, named semi-supervised learning [27], where like the name suggests it combines the two previous techniques and uses a small amount of labelled data and a large quantity of unlabelled data which provides the benefits of both approaches without needing to find a large amount of labelled data.

The final technique, reinforcement learning [28], does not use the same learning approach as the first three, in this case, the method is concerned with how software agents should take actions in an environment, being the agents the entities which perform actions in an environment in order to gain some reward. The environment is given by a scenario that an agent needs to face and lastly, a reward corresponds to an immediate return that is given to an agent when the task is performed.

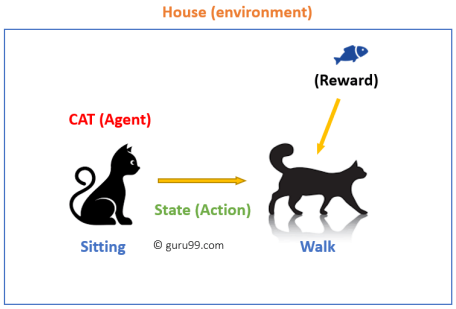



Figure 2.12: Practical Case of Reinforcement Learning [28].

In the case of Figure 2.12, where the cat is the agent and the house is the

environment, the goal action is to make the cat walk. The cat will be stimulated to walk, however, it does not know which is the correct action to make, but once the cat walks, a reward will be given. Therefore, the cat will learn from positive experience that every time it gets the stimulus to walk and performs the action it will get a reward. The same flow occurs in reinforcement learning applications such as robotics for industrial automation, business strategy planning and data processing being some of its main characteristics the fact that there is sequential decision-making where the agent's actions determine the subsequent data it receives. The main advantage of using this type of learning when compared to the previous ones is the fact that since it automatically adapts to new environments there is no need to retrain the algorithm, however, one of its biggest drawbacks is the fact that the parameters affect the speed of learning.

Table 2.3 summarizes all the main characteristics of the learning methods described in this section.

Table 2.3: Characteristics of ML and DL learning types.

| **Supervised** | **Semi-Supervised** | **Unsupervised** | **Reinforcement** |
|---|---|---|---|
| All data is labelled. | Most of the data is unlabelled, however there is labelled data. | All data is unlabelled. | Works on interacting with the environment. |
| Does not require high computer power. | Computing complexity depends on the amount of labelled and unlabelled data. | Computationally complex. | Computing heavy and time-consuming. |
| Highly accurate and trustworthy method. | Less accurate than supervised learning, accuracy depends on the amount of labelled data. | Less accurate and trustworthy. | Accuracy cannot be compared with previous methods. |
| Classifying big data can be a major issue. | Cannot be used in complex problems. | It is not possible to get precise information regarding data sorting. | Too much reinforcement learning may cause an overload and weaken the results. |
| Used in fraud detection, for instance. | Used in customer clustering, for example. | Used, among others, in medical predictions. | Used in AI games, for instance. |

### 2.2.1 Deep Learning and Machine Learning in the Automotive Industry

AI fields such as ML and DL have much potential in the automotive industry due to the increasing complexity and useful functionalities that have been implemented in new vehicles [29, 30]. Because of that, AI has been widely used both inside and outside the vehicle, as visible in Figure 2.13.

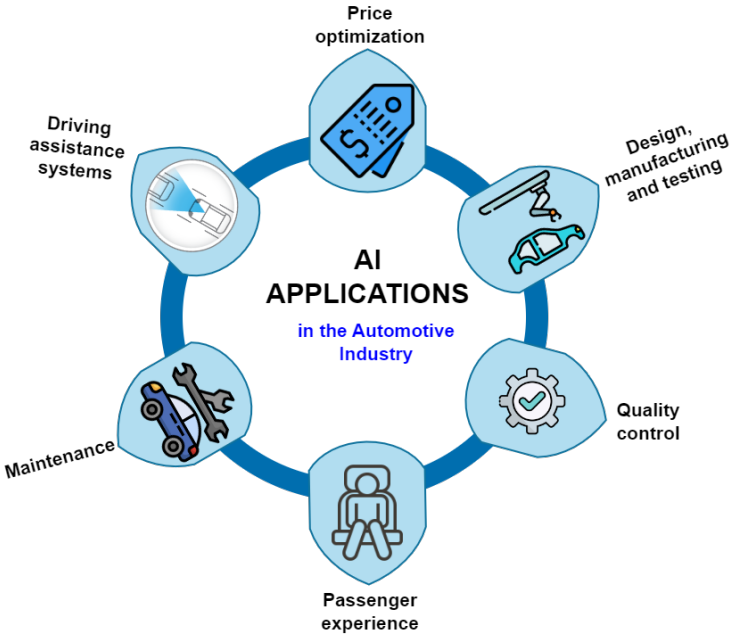



Figure 2.13: AI applications in the automotive industry.

ML faces some big challenges in the automotive industry: the need for storage and processing of large volumes of data as well as the necessity of dealing with unstructured data from sensors on the vehicle or machines in the manufacturing process. Because of that, the usage of DL instead of ML started to be required since it, in some cases overcomes the human work, and mostly because it allows the possibility to overpower the challenges that ML faces. Therefore, the use of DL started to be implemented primarily for infotainment and cluster testing but also for improving the model-based testing.

Mostly CNNs are used in the industry, which can be very powerful in diverse domains such as visual inspection in manufacturing, autonomous driving, robots and smart machines [29].

In 2019, a master's thesis was developed [31] to improve the efficiency of automated testing by analysing log files that contain detailed information about each executed automated test case, like the executed functions and information about the result of the execution (pass or fail). The project developed by the author had its goal on reducing the overall cost of testing in the company, by making it possible

to find bugs earlier in production, which would decrease both the time of testing and consequently the costs of testing.

To make it possible, the author used a DL approach capable of predicting the test cases that would fail, based on prior experience, and once those were fixed they were proposed to run which would increase the proportion of failed test cases to passed test cases during real-time execution. For achieving the main goal that the author set, the objective was to develop suitable DL methods to try to classify test scripts as pass/fail based on their input and hence enable auto-generated test scripts with a low probability of resulting in a failed execution.

The model adopted by the developer can be seen in Figure 2.14.

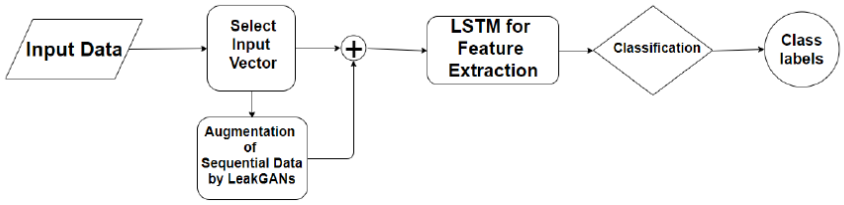



Figure 2.14: Model used for the developed detection system [31].

The dataset – input data – was composed by 1810 log files, of which 1158 were used for training purposes, 290 for validation and 362 for testing. The selection of the input vector was made using the regular expression method, which consists of a special sequence of characters that define a search pattern used to match the string pattern and extract information from text, in the specific case, the log-file. Then, more data was fed into the network, being the main network used the Long Short-Term Memory (LSTM) network for sequential prediction, and finally, the classification was made.

Out of the 362 log files used for testing, in 175 cases the result was "fail" and in the rest (187 cases) the result was "pass". However, the main goal of the author was to predict more failed test executions and, to achieve that it was needed to define a threshold that would compare the output from the neural network with the defined value and in case the output was bigger than the given threshold, the class label would be "pass". After several experiences, the author was able to find a threshold value of 0.862 that had an accuracy of 97.7% in the failed test execution prediction, allowing the company where the work was developed to save at least 50% of the needed to do the test automatization and still discover the majority of test issues.

When it comes to infotainment systems, DL is widely used to improve the driver and passenger experience in the vehicle, as well as improve security. In [32] the authors pointed up the main factors that cause the passengers to not have a pleasant experience in taxi travel, being these:

1. The fact that the music is selected by the driver which sometimes does not match the passengers' emotions;
2. Driver's distraction;
3. Forgetting of personal items after leaving the taxi;
4. Misbehavior of the driver both physically and vocally.

Being the main purpose of the technological increase in vehicles to give an amusing experience to the driver, when it comes to public transportation, like taxis, the experience needs to be good both for the driver but especially for the passenger, hence the authors proposed a DL based smart infotainment system capable of solving the four main issues mentioned. For that, four different datasets already available were used, one for each issue. The system's architecture can be seen in Figure 2.15.

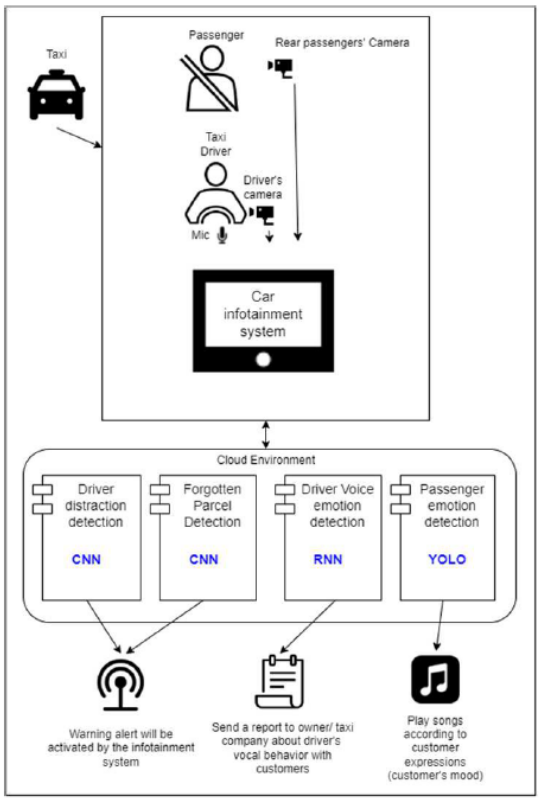



Figure 2.15: System overview [32].

The first block represents the devices: two cameras, a microphone and an Android infotainment and the second block corresponds to the backend cloud server which connects the models to the infotainment system, acting as the brain of the system.

To solve the first pointed issue the authors used a CNN in a supervised learning method, where the output would consist of seven different classes, such as happy, sad or anger, being the validation accuracy of around 62%. The workflow can be seen in Figure 2.16.

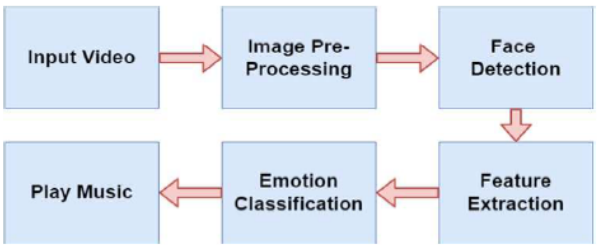



Figure 2.16: Structure of the facial emotion detection [32].

The second topic would follow a similar approach when compared to the previous one, however instead of having an emotion classification with the output being a music playing in the infotainment, there would be a classification of the distraction of the driver with the output being an alert to the driver. In this particular case, the authors were able to achieve almost 90% accuracy in the trained model.

For the third problem, an already trained algorithm was used - YOLO, being able to distinguish 80 different classes. Finally, the driver vocal behaviour detection was built based on Figure 2.17, where the system can distinguish 8 different voice tones such as calm, happy and surprised.

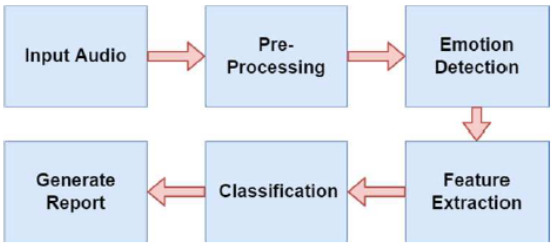



Figure 2.17: Structure of the voice emotion detection model [32].

Altogether, with the proposed system that they were capable of implementing, the authors accomplished the main goal of the research - minimising the distrust in the taxi industry by giving the passenger a better experience.

### 2.2.2 Used Frameworks

Python has become the most common open-source language for the construction of neural networks [33] for a diverse number of reasons like the fact that it is an easy-to-learn programming language, it has a standard C interpreter that allows it to be used

for low-level libraries and mainly due to its big community that makes the finding of the problems' solution easier. Although it has a significant disadvantage - it is low performance, it can be conquered by writing demanding parts of the software using C, for example. Many Python libraries are used to develop and execute AI algorithms, being the most used:

- TensorFlow [34] - an open-source library mainly used for deep ML, where the computational core is written in C++ and the interface part is implemented in Python. It is a library built on the principle of dataflow which means that the program is organized in computational blocks connected to each other in the shape of a graph - computational graph. With this library the data is processed by passing from one block to the other and it is possible to use parallel calculations;

- PyTorch [33] - previously developed in C, with the increase of Python's popularity the library was rewritten in C++ and Python. PyTorch similarly to the previous library was developed based on the dataflow concept, however, when compared to TensorFlow - which has a static computational graph - in PyTorch the graph is dynamic which means that it is possible to add, remove and modify nodes as needed while the model is running;

- Keras [35] - it provides a high-level programming interface for building the neural networks and it is built on top of TensorFlow. This library, entirely written in Python, is based on three principles: easy to use, modularity and extensibility, where the second one allows the possibility of describing the neural layers, optimizers and activator functions separately and then combining everything into a single model;

- Theano [33] - corresponds to an optimized Python interface since it translates the functions to C++ and compiles them to run on the Central Processing Unit (CPU) and once the optimization and compilation are done the functions become regular Python functions but with higher performance. Theano library is normally used by Keras library, as a backend for building the networks.

In order to do a review and analysis of the performance of the mentioned libraries, a study was conducted [33] in which the authors chose a problem of handwritten digit recognition being the training file composed of 60 000 copies and the test file by 10 000 copies. The used neural network, Figure 2.18, is constituted by an input layer, two hidden layers and an output layer in which SoftMax, an activation function that transforms the inputs that come from the previous layer into a vector of probabilities, is used.

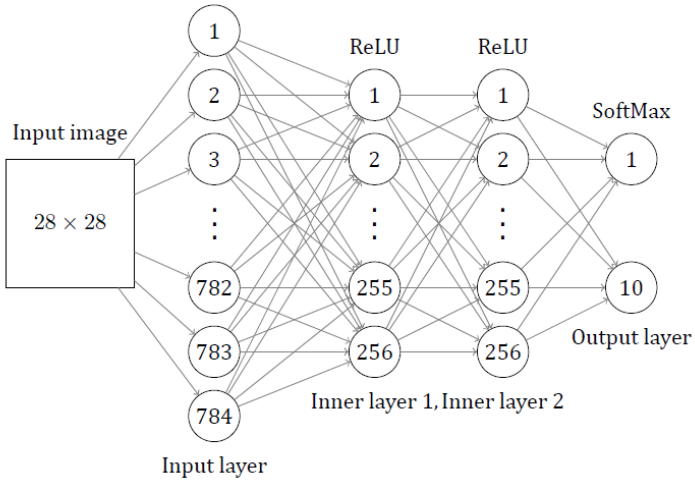



Figure 2.18: Used neural network [33].

The model was built and trained with TensorFlow, PyTorch, Keras and Theano libraries where the training time and accuracy were measured during 50 epochs, being the obtained results visible in Table 2.4.

Table 2.4: Obtained results [33].

| **Library** | **Accuracy** (%) | **Time (sec)** |
|---|---|---|
| Keras | 98.56 | 113.80 |
| TensorFlow | 89.07 | 63.48 |
| Theano | 97.38 | 257.29 |
| PyTorch | 98.07 | 1492.29 |

The authors were able to conclude that Keras and Theano were the libraries with better accuracy results, however, the learning time was significantly lower with the Keras library. Tensorflow library was the only one where the accuracy did not exceed 90%, which when compared to the other three is low, and PyTorch library presented the worst results in terms of time, where it was possible to see that even though the learning time increases with the number of epochs, this particular library presents better accuracy results with many epochs.

## 2.3 Automotive In-Vehicle Infotainment

The term infotainment [36, 37] is a mix of the words "information" and "entertainment" and refers to the touchscreen that is in the middle of the vehicle. Besides providing information about the vehicle or offer entertainment such as playing a personal Spotify playlist, it also allows the driver and front passenger to control many functions of the vehicle like the climatization. The infotainment system plays

a major role in defining customers' expectations, whereas nowadays the more functionalities and attractive it is, the better the probability of the customer buying the car. Thus the biggest challenge of producers is to have an operating system both easy to use and appealing to the customer.

The operating system of the infotainment was usually Linux based and developed by the OEMs which led to several drawbacks and unsatisfaction of the customers [38] mainly because since the systems usually do not allow software updates, they would become outdated soon. Also, these were complex operating systems which made it very difficult for the producers to put the systems to work smoothly and intuitively. Therefore, two open-source operating systems were developed: Automotive Grade Linux and Android Automotive.

The first one [37, 39] was launched in 2012 by the Linux Foundation and until now was the most used operating system and its main goal was to act as a reference framework that can be supported and used by OEMs to create their commercial products by integrating their innovations, by other words, by using this operating system the brands can create a different layout, different design, distinct ways to control the vehicle functionalities even though the base operating system is the same, which has the big advantage of developing a personalized infotainment system based on the users that brands have. The Automotive Grade Linux provides a Unified Code Base that includes the operating system, middleware components and application framework and provides from 70 to 80% of the starting point of the production of the infotainment system, instrument cluster or telematics, making it possible to see in Figure 2.19 an example of the infotainment's climatization menu design.


Figure 2.19: Example of Automotive Grade Linux climatization menu [40].

Android Automotive [38, 41], usually mistaken for Android Auto (that mirrors the user's smartphone to the infotainment via a USB cable) is an Android operating system built for vehicles launched by Google in 2017 that offers three main functionalities:

- Openness - which makes it more efficient by providing basic automotive infotainment features in a free and open source codebase;
- Customization - enables developers to differentiate their product as they wish;
- Scale - that is achieved through Android's common framework, language and APIs, allowing the reuse of development expertise and completed software from Android Developers worldwide.

One big differentiation from the previously described operating system is the fact that the Android Automotive-based platform acts as an independent device, connected to the user's Google account, which means, a smartphone is not required to get the user's contacts and music playlists, for example. And like a smartphone, an Android Automotive operating system allows both the possibility of software updates, which gives the user the sensation that the car has not become outdated but also the option to use some of the most popular applications like Google Maps.

An example of the Android Automotive climatization menu can be seen in Figure 2.20, in which it is possible to observe that it follows the typical Android design that the users are accustomed to.


Figure 2.20: Example of Android Automotive climatization menu [42].

Table 2.5 summarizes all the major characteristics of the two operating systems, in which it is possible to conclude that Android Automotive, even though more recent than Automotive Grade Linux has more advantages and because of that it has started to be used as the only operating system of multiple brands like Volvo, making the growth rate for Android Automotive deployment [43] the biggest until

2027 where it is estimated that it will capture 18% of the market, compared to the 1% in 2022.

Table 2.5: Automotive Grade Linux vs Android Automotive.

| **Automotive Grade Linux** | **Android Automotive** |
|---|---|
| Open-source software. | Open-source software. |
| Linux based. | Android based. |
| More % of usage in the market in 2022. | Less % of usage in the market in 2022. |
| Allows developers to customize the infotainment's system design. | Allows developers to customize the infotainment's system design. |
| Needs to have a smartphone connected in order to have the user's information. | Does not need to have a smartphone connected to have the user's information. |
| Does not allow the use of apps such as Google Maps unless Android Auto is activated. | Allows the use of Google apps like Google Maps without any external connection. |
| Less appealing and hard to use design. | More appealing and easy-to-use design. |
| Customers are not used to Linux-based systems. | Customers are used to Android base systems. |
| Does not have a lot of support worldwide for the developers. | Has a big support community worldwide. |

However, some other car manufacturers like BMW will start to implement infotainment systems both in Linux and Android making it possible to individually configure the system, and quoting the Senior Vice-President Connected Company and Development Technical Operations at the BMW Group [44] the company is "integrating the best aspects of all worlds" making sure their customers "always enjoy a unique, customised digital experience in their vehicle".

# Chapter 3

# Study of the Problem

In this Chapter, the idea behind the project will be explained by giving an overview of Jakku, highlighting the way it works as well as what can be improved. After, the work that needs to be developed to improve the platform will be explained.

## 3.1 Work Previously Developed

The project hereby presented has its starting point in a system developed by the company, Jakku, with the aim of developing a new approach capable of solving some of its issues. Given that, this section has the purpose of explaining what is Jakku and how it works.

Jakku was developed using the Python framework for web development - Django - with the purpose of creating a tool that would allow the execution and management of both manual and automated tests on an Android Automotive platform remotely, so that the hardware could be kept in the customer facilities and the tester did not have the need to work near it. The fact that Jakku has its web app, Figure 3.1, makes user interaction more appealing and easier since it allows the user to create his profile, connect a new board, execute tests as well as other tasks very easily, where the only requirement to access the web app is to be connected to the company's Virtual Private Network (VPN).

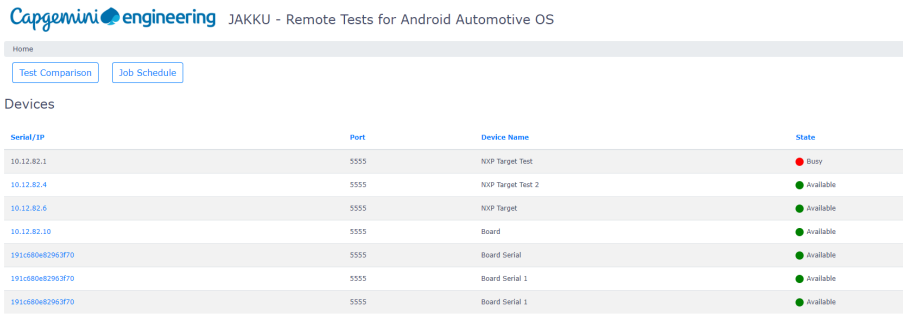



Figure 3.1: Jakku web app main page.

As visible in the previous figure, on the main page, can be seen all the configured boards as well as their status (busy or available), making it easy to perform the optimization of the target allocation since it can easily be seen on which boards we can trigger test executions.

The mentioned boards are NXP MCIMX8QXP-CPU boards [45] which contain a 4x Cortex-A35 processor and were built with the purpose of supporting graphics, video, image processing, audio and voice functions, while meeting efficient performance requirements. For the Jakku project, the boards will work as targets containing an Android Automotive image, previously built and flashed.

The platform allows the user to connect the target via serial or IP and select if the tests will be performed manually or automated using Robot Framework, which is a keyword-driven, Python-based open-source framework used for test automation. It is important to mention that the platform allows the connection to Jira, a tool used mainly for work management and GitLab, an open-source version control system, where both work as a database for the manual and automated tests, respectively.

The workflow of the execution of manual tests can be seen in Figure 3.2.

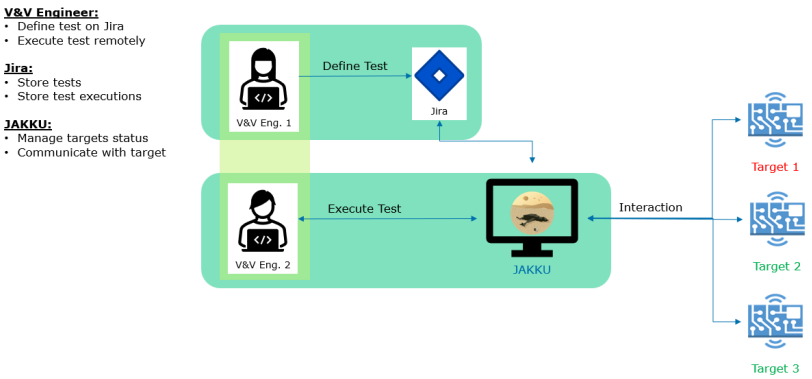



Figure 3.2: Workflow for manual tests in Jakku.

By analysing the previous scheme, it is possible to conclude that in order to execute manual tests three parties need to be involved:

- Jakku - used to communicate with the target showing its status and also to stream the Android Automotive display, so that Engineers are able to see it remotely;
- Engineers - in charge of writing manual tests in Jira and executing the tests remotely, by manually pressing the defined icon in each step of the test;
- Jira - the management tool used to store the tests previously written by the Engineers, the executions of the test cases, including the final result of the tests as well as pieces of evidence such as screenshots.

The automated test cases are done by the user, and the workflow shows some divergences when compared with the previous method, Figure 3.3, being the main differences the fact that more technologies need to be involved and that the user doesn't have interaction with the target while the tests are running.

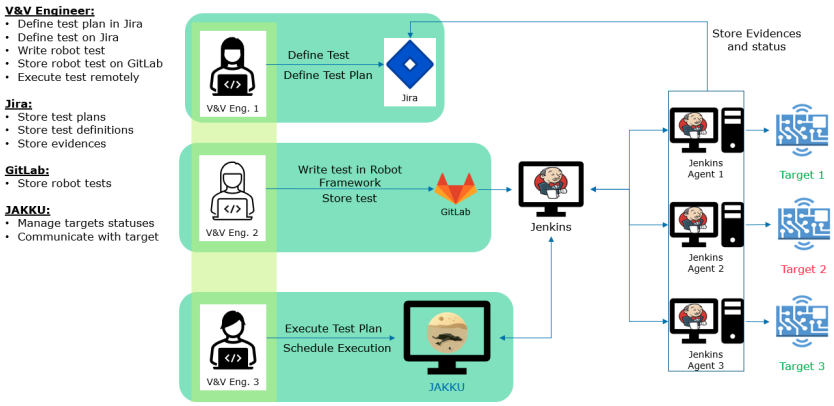



Figure 3.3: Workflow for automated tests in Jakku.

Upon examination of the image shown above, it is possible to conclude that at least three new technologies are used in this process, Robot Framework, GitLab and Jenkins. In order to execute automated tests, the Engineers besides writing the tests in Jira need to automate them using Robot Framework and then store them in GitLab.

However, even though Jakku was a great innovation made by the company, it still has some disadvantages, such as the fact that it is possible to connect only one Gitlab and Jira project to the platform, which makes it necessary to store all the information together, something that is not practical since the company works with different projects that cannot share information between them. Another great problem is resource usage because for executing manual tests, each Engineer needs to spend a lot of time whether to design the test, execute it, attaching evidence and then analyse it. Of course, automated tests started to be implemented more often to decrease the time spent, however for each automated test an Engineer needs to first,

have knowledge in Robot Framework and then develop the corresponding code for each step of the test. Also, in the particular case of Jakku, automated tests involve a lot of technologies which can become somewhat confusing to interact and mainly to analyse the reports.

Because of these, a proof of concept was thought to first overcome some of the mentioned disadvantages and also to include a new technique in the company to perform automated tests.

## 3.2 Work to be Developed

The architecture of the work to be developed can be seen in Figure 3.4.

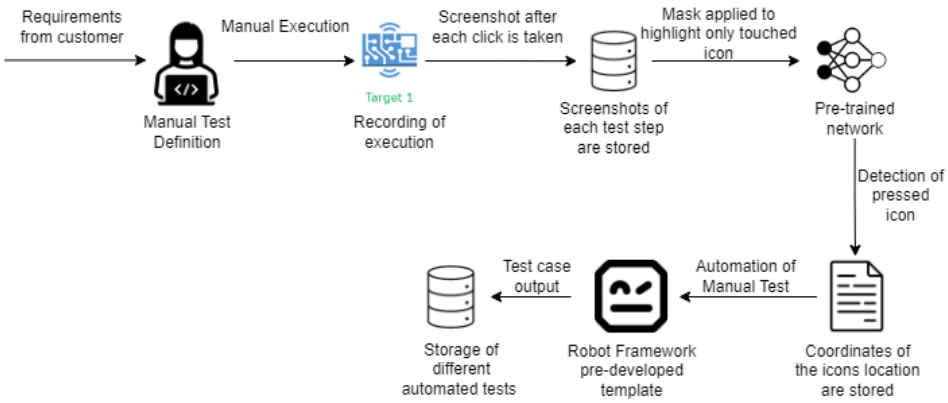



Figure 3.4: Work to be developed.

When comparing with the already existing approach, it is possible to verify that the need to use Jira, Jenkins and Gitlab is no longer mandatory. Jira can still be used as a way to read all the requirements that come from the customer and to write all the test steps, and Gitlab can also still be used as a way of storing the developed code. However, other similar tools can be utilized, which already makes this proof of concept more beneficial than Jakku, since the customers can select their preferred tools.

The presented architecture has three main steps: the recording of the manual tests' execution, the training of the CNN and the development of a single Robot Framework template that can be used in all test cases' execution. Once an Engineer receives the requirements from the customer, he can start defining the steps needed to execute the manual test and perform the execution on the target that will be recorded by developing a Python script capable of taking a screenshot after each click on the target's screen. All screenshots taken after a manual test is done will be stored to serve as an input of the pre-trained CNN. However, since each screen of the Android Automotive infotainment has several icons that the network will be trained to detect, a mask will be applied to each stored screenshot to highlight the pressed

menu and therefore serve as an input of the neural network, that will find the icon in the screen and give as an output the `x` and `y` coordinates of its location. Finally, the Robot Framework template will read through all the lines of the network's output file and perform the test in an automated way, to verify if each step is being executed correctly also a screenshot will be taken after each menu is touched and an image comparison between the taken image in the automated test and the stored image of the manual test will be made. Then, all the needed files to perform the automated execution will be stored, including the screenshots taken during the manual test and the output file of the network.

With the described architecture, Jakku's main problems would be overcome, starting with the fact that only one Engineer would be needed to execute both manual and automated tests and no previous knowledge of Robot Framework or any other tools would be needed. And, even though it is a complex workflow, the final user, would only need to do the manual test definition and execution and start the automated tests.

In case the described approach provides good results, it has the final objective of being integrated into Jakku's web app to make it even easier for the user to complete the testing of each target.

# Chapter 4

# Convolutional Neural Networks

In this Chapter a description of CNN applications will be presented, emphasizing existing networks for object detection since it represents the goal of the project. In addition to this, important performance metrics will be described, as well as different behaviours that can occur during the training process.

## 4.1 Applications of CNN

As mentioned in Chapter 2, these networks are the most used ones, both because no human supervision for relevant feature identification is needed and because of their high accuracy mainly in image classification, object detection and speech processing.

The usage of CNN on daily tasks like face recognition (to detect, for example, if a person is using a mask in a hospital or not), image classification (an example is medical imaging to detect anomalies) or object detection (when it comes to detecting road signals, for example) has been increasing because of great results that a CNN brings and also because of pre-trained networks already existing that can easily be adapted to each specific case by using transfer learning. Transfer learning consists of the reuse of an already existing model on a new problem, where the knowledge of a model is applied to a different, yet relatable task. The transfer learning process can be seen in Figure 4.1 and it is done by reutilizing the first layers (backbone) and then customising and adapting the final output layers.

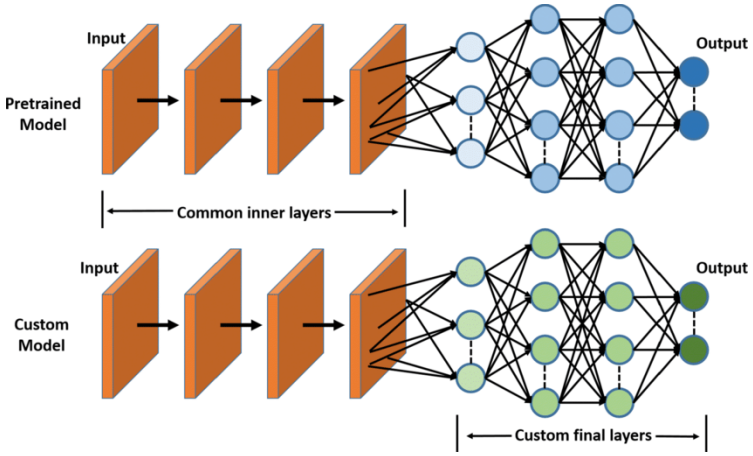



Figure 4.1: Transfer learning [46].

This technique has several advantages, such as the decrease of data, the accuracy increase and the faster training.

The most used approaches to transfer learning are [47]:

- Training a model and then reusing it - in a case where there is not a lot of data to train the model it is possible to find a similar problem, but with a large dataset, train the deep neural network for that problem and then reuse the model to solve the initial task;
- Using a pre-trained model - this corresponds to one of the most used approaches when it comes to transfer learning and consists of finding the model most suitable for the problem that needs to be solved, replacing the dataset in which the model was trained to the one of the problem in question and reuse some of the model's layers and retrain others;
- Feature extraction - or representation learning - consists of discovering a good features combination within a short period that can then be reused for other problems by using the first layers to recognize the right representation of features and use as the output layer one of the intermediate layers since the last ones are specific to the task that they were trained for.

When it comes to image classification, some pre-trained networks such as VGG-16 and Resnet are widely used due to their characteristics and results that present. The first one - VGG-16 (Figure 4.2) is a CNN model with a depth of 16 layers [48] that was trained over several weeks, achieving a test accuracy of 92.7% using the ImageNet dataset [49], which contains approximately 14 million training images across 1000 objects, making it widely used in classifying all types of images or objects.

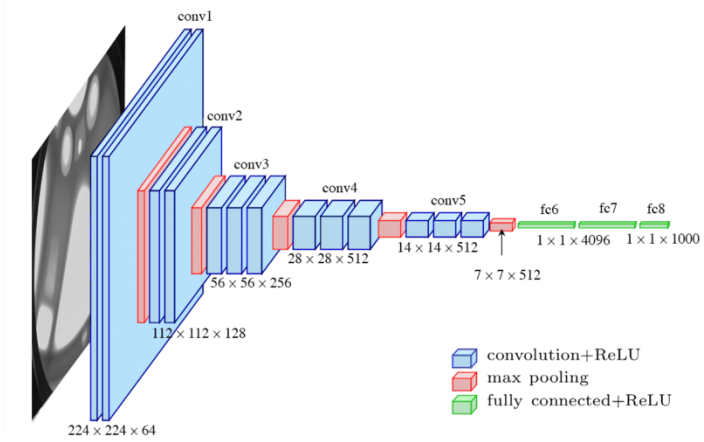



Figure 4.2: VGG-16 architecture [48].

The second one, Resnet [50] is an 18, 50, 101 or even deeper CNN that was also trained using the ImageNet dataset, being the difference between them in the number of layers, the way layers are connected and consequently the test accuracy. Figure 4.3 presents the base architecture for any of the Resnet101 models.

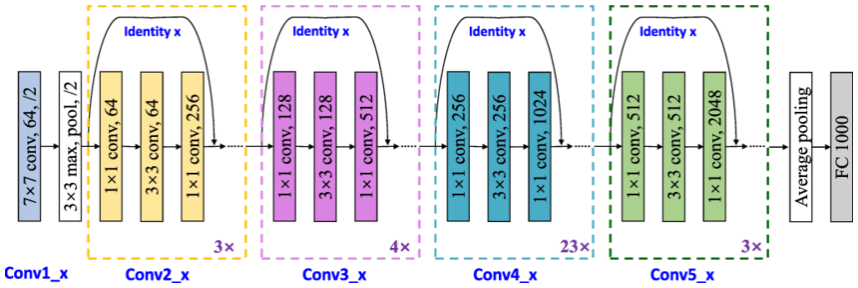



Figure 4.3: Resnet101 architecture [51].

With VGG-16 the concept of increasing the number of layers to achieve better accuracy was presented, however having more than 20 layers could prevent the model from converging, so, with Resnet the concept of skipping connections was introduced, which allows inputs to skip some convolutional layers leading to a reduction in training time, since once the model learns a feature it will not try to learn it again, instead it will learn another feature [48, 50]. Because these models offer good test accuracy, they are usually used as a backbone - which consists of the feature extractor of the network - for bigger networks such as object detection CNN like Retinanet which has Resnet as its backbone.

Like Retinanet, other CNN models such as YOLOv5 (or older YOLO versions), YOLOv8, RCNN, Faster R-CNN and Single Shot Detector (SSD) are used for object detection.

### 4.1.1 YOLOv5

You Only Look Once (YOLO) is a real-time object detection algorithm that was pre-trained with the COCO dataset [52]. It is widely used for object detection due to its speed, since it - like the name indicates - looks at the image only once, by breaking it into a grid and then classifying and locating each section of the grid, whereas other systems (the multi-stage networks) analyse the image pixel by pixel.

The first version of YOLO was developed in 2015, being YOLOv8 the latest version of the network, released in 2023. YOLOv5 was launched in 2020, and it was the version selected for the purpose of this project due to the amount of available information and also because it was where the biggest time improvement was made [53, 54].

Being YOLOv5 a single-stage model, its architecture is made of three main components: backbone, neck and head [53]:

- Backbone - corresponds to a pre-trained network, used to extract rich feature representation for images and it helps to reduce the image's spatial resolution and increase its feature resolution;
- Neck - used to extract feature pyramids, helping the model to adapt to objects of different sizes and scales;
- Head - applies anchor boxes on feature maps and renders the final output: labels, bounding boxes and scores.

In this version of the network, five different models were released, from the variant `n (nano)` for an extra small size model, until `x (extra)` for an extra large model. For the purpose of this work, the `s (small)` version of the model will be used, both because of the size of the dataset, and the speed-accuracy ratio given by the model, possible to see in Figure 4.4.

| Model | size (pixels) | $mAP^{val}$ 0.5:0.95 | $mAP^{val}$ 0.5 | Speed CPU b1 (ms) | Speed V100 b1 (ms) | Speed V100 b32 (ms) | params (M) | FLOPs @640 (B) |
|---|---|---|---|---|---|---|---|---|
| YOLOv5n | 640 | 28.0 | 45.7 | **45** | **6.3** | **0.6** | **1.9** | **4.5** |
| YOLOv5s | 640 | 37.4 | 56.8 | 98 | 6.4 | 0.9 | 7.2 | 16.5 |
| YOLOv5m | 640 | 45.4 | 64.1 | 224 | 8.2 | 1.7 | 21.2 | 49.0 |
| YOLOv5l | 640 | 49.0 | 67.3 | 430 | 10.1 | 2.7 | 46.5 | 109.1 |
| YOLOv5x | 640 | 50.7 | 68.9 | 766 | 12.1 | 4.8 | 86.7 | 205.7 |

Figure 4.4: Performance of YOLOv5's different sizes models on COCO dataset [55].

Besides the presented differences, the number of layers is another parameter that changes between the five models, being the first two (YOLOv5n and YOLOv5s)

made of 214 layers, whereas the medium version (YOLOv5m) is composed of 291 layers. This number significantly increases for the larger version (YOLOv5l) with 368 layers, and finally, the extra large model (YOLOv5x) is composed of 445 layers.

The base architecture of the network can be seen in Figure 4.5.

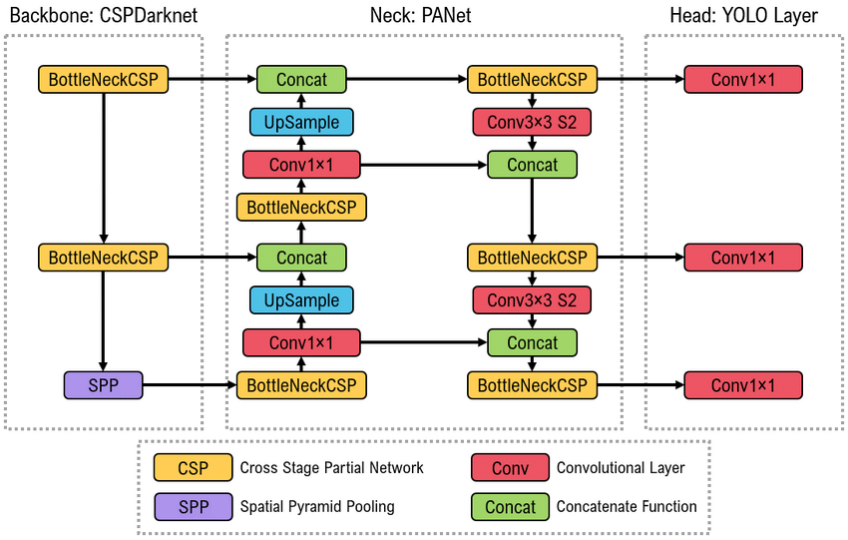



Figure 4.5: YOLOv5 base architecture [56].

Like in YOLOv3 [57], the backbone used is Darknet53. It corresponds to a 53 layer-deep convolutional neural network that uses successive 3x3 and 1x1 convolutional layers, as illustrated in Figure 4.6.

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

Table 1. **Darknet-53.**

Figure 4.6: Darknet-53 [57].

However, what changes to the YOLOv5 version is the Cross Stage Partial (CSP) network strategy, visible in Figure 4.7, that divides the feature map of the base

layer into two parts. These are then merged using a cross-stage hierarchy, meaning that there are connections between layers that are not adjacent to each other. This strategy is used to allow the network to capture high-level, abstract features of the input data by combining information from multiple levels of abstraction and to reduce both the number of parameters as well as the amount of computation [53, 58].

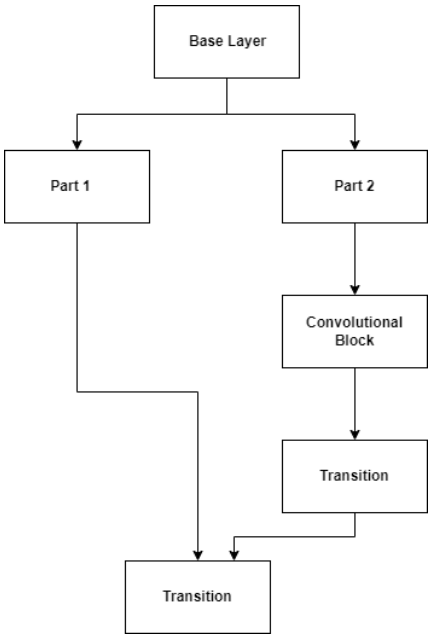



Figure 4.7: CSP network strategy.

YOLOv5's backbone works as follows [53, 59, 58, 60]:

- The input image is passed through a convolutional layer that will reduce its spatial resolution by a factor of two, both vertically and horizontally. This downsampling will help reduce the computational cost of the next convolutional layers, since there are fewer pixels to process, while it allows the network to capture larger-scale features;
- The output of the previous point will then pass through several convolutional blocks, each of which has two or more convolutional layers with residual connections that allow the network to bypass some of the layers, enabling a deeper architecture (the CSP strategy).

Besides the Darknet-53 network, it also includes a Spatial Pyramid Pooling (SPP) layer, visible in Figure 4.8, that applies multiple pooling operations with different kernel sizes to each spatial location in the feature maps allowing the layer to capture features at multiple scales without introducing additional parameters or computation. The result of this operation is then concatenated and passed through convolutional layers to produce the feature maps.

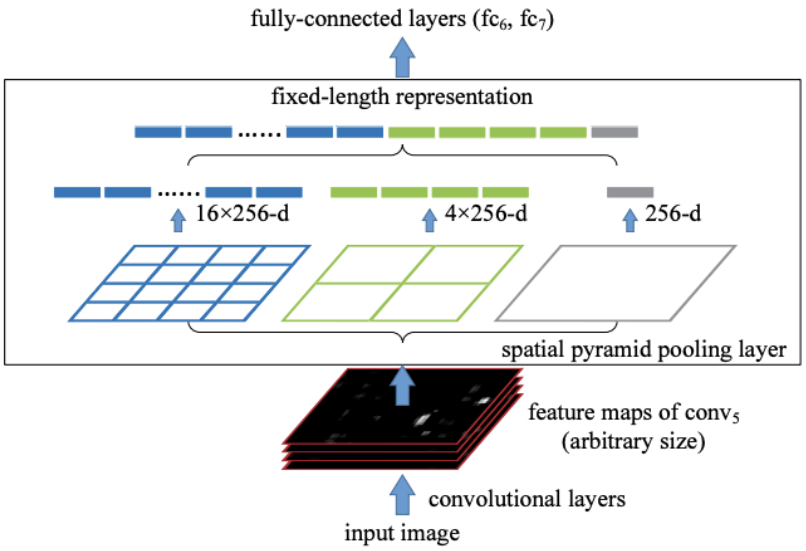



Figure 4.8: SPP layer [61].

The model neck of this network is composed of the Path Aggregation Network (PANet) module [53, 58, 62], a neural network designed for processing images or other multidimensional data at multiple scales or resolutions. It takes as input the output of the SPP layer and a batch of feature maps from an earlier stage of the backbone with higher spatial resolution.

The two sets of feature maps are first passed through convolutional layers to decrease their depth and increase the spatial resolution and then concatenated and passed by another series of layers to produce a single set of feature maps. This is done at each stage of the backbone, allowing the network to combine features from different scales.

In short, the idea behind the neck of YOLOv5 is to create a set of feature maps that capture information about the input data at different scales, and then combine these representations into a final output, improving the information flow and thus helping in the proper localization of pixels in the task of mask prediction [59, 63].

The final part of the YOLOv5 network corresponds to the head, which consists of three convolutional layers that take the fused feature maps from the neck as an input, and it is responsible for predicting the bounding boxes and object classes for each detected object [59, 60].

Figure 4.9 shows an example of how YOLOv5's architecture works in object detection for a given input image.

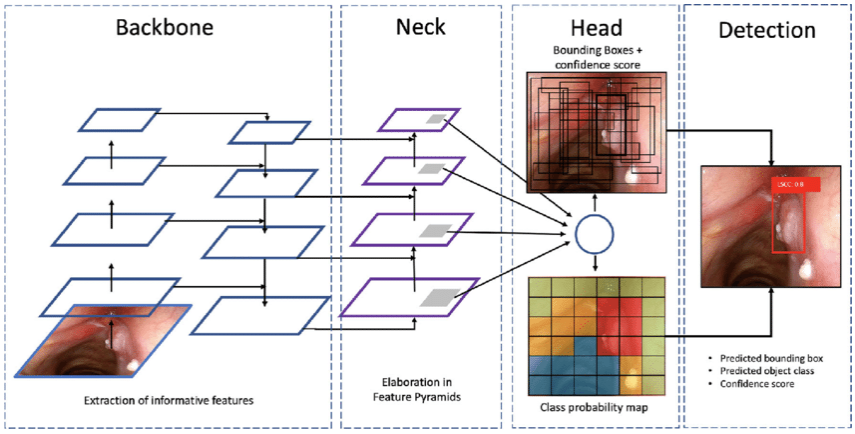



Figure 4.9: Example of YOLOv5 architecture's representation [64].

There are some key concepts to YOLOv5's head to better understand how the detection is made [53, 59]:

- Grid-cells - the location of objects is given through coordinates expressed at the centre of a grid-cell, where each cell is a vector of $(5+num\_classes)$ values that has information such as:

  1. $Objectness\_score$ - the probability of the existence of an object within the grid-cell;
  2. $X_c$ - the horizontal distance between the top left corner of the grid-cell and the centre of the object;
  3. $Y_c$ - the vertical distance between the top left corner of the grid-cell and the centre of the object;
  4. $Width$ - width of the grid-cell;
  5. $Height$ - height of the grid-cell;
  6. $Classes$ - vector of $len = len(n\_classes)$ where the values represent the chance of an object being part of a certain class.

- Detection layers - by default there are three detection layers:

  1. First detection layer - suitable for small objects. It outputs 80x80 grid cells, where each grid cell represents a width and height of 8 pixels of the image that was given as an input $(80 * 8 = 640)$;
  2. Second detection layer - suitable for medium objects. It outputs 40x40 grid cells, where each grid cell represents a width and height of 16 pixels of the input image $(40 * 16 = 640)$;
  3. Third detection layer - used for detecting large objects. It outputs 20x20 grid cells, where each grid cell represents a width and height of 32 pixels of the original input image $(20 * 32 = 640)$.

- Predictions per scale - each layer presented in the head of the network will be specialized at detecting, by default, three different aspect-ratios;
- Anchor boxes - represented by two integers that indicate the width and height in pixels.

As mentioned, one of the final outputs of the network will be a bounding box around the detected object. The centre of the `x` and `y` coordinates of the bounding box is given by the following equations, respectively [59, 63]:

$$b_x = (2 * \sigma(t_x) - 0.5) + c_x \tag{4.1}$$

$$b_y = (2 * \sigma(t_y) - 0.5) + c_y \tag{4.2}$$

Where $\sigma$ corresponds to the sigmoid function, $t_x$ and $t_y$ are the predicted offsets for the centre of the bounding box relative to the centre of the grid cell and $c_x$ and $c_y$ are the coordinates of the top-left corner of the grid-cell.

To calculate the width ($b_w$) and height ($b_h$) of the bounding box, the following operations are made [59, 63]:

$$b_w = p_w * (2 * \sigma(t_w))^2 \tag{4.3}$$

$$b_h = p_h * (2 * \sigma(t_h))^2 \tag{4.4}$$

Where $p_w$ and $p_h$ represent the width and height, respectively, of the anchor box and $t_w$, $t_h$ represents the predicted width and height relative to the width and height of the grid-cell.

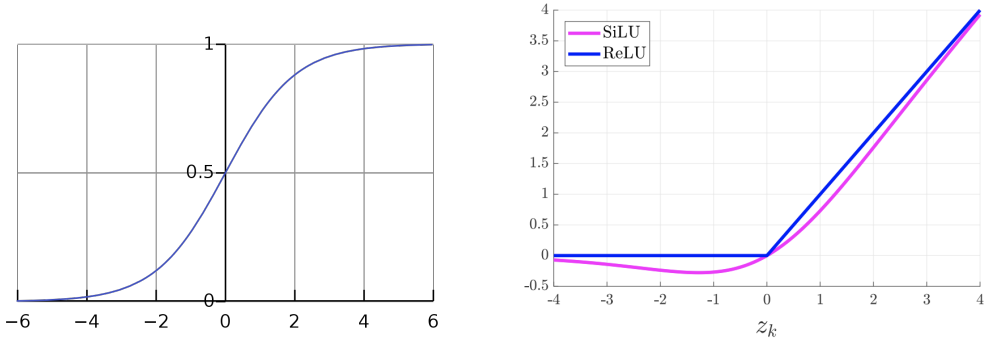
YOLOv5 also predicts the class of the detected object for all grid-cells present in the image, and after predicting the class probabilities, the algorithm does non-max suppression which helps it to discard unnecessary anchor boxes. It finds the Intersection over Union (IoU) for all the bounding boxes with respect to the one with the biggest probability and by doing this only one bounding box around the object is given as a final output. The IoU is given by the following equation [63]:

$$IoU = \frac{B1 \cap B2}{B1 \cup B2} \tag{4.5}$$

Being the numerator the area of overlap and the denominator the area of union. An example of how IoU is calculated can be seen in Figure 4.10.


Figure 4.10: IoU calculation [65].

Finally, and as seen in equations (4.1), (4.2), (4.3) and (4.4) the network uses the Sigmoid activation function (Figure 4.11a) in the output layer, which is a non-linear activation function that can transform any value in the domain $[-\infty, +\infty]$ to any value between [0,1] and in the particular case of YOLOv5 it is used to map the predicted values of bounding box dimensions and objectness score to the range of [0,1]. Besides this function, the model also uses the SiLU activation function in the convolution operations of the hidden layers, which is a function similar to the ReLU activation function but with a smooth curve, as visible in Figure 4.11b, that allows for a more stable gradient during backpropagation [53, 59, 63].




(a) Sigmoid function [53].

(b) SiLU vs ReLU function [53].

Figure 4.11: Activation functions used in YOLOv5.

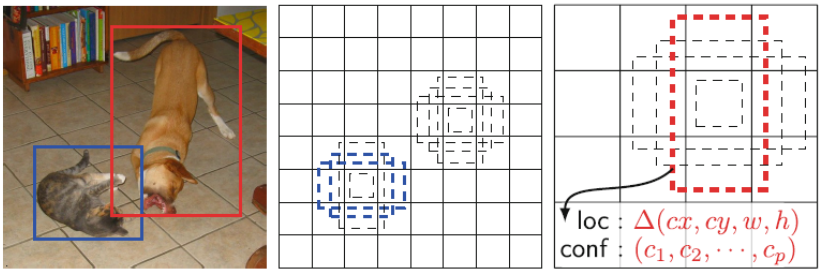
### 4.1.2 SSD

Just like YOLO, SSD is a one-stage detector, however, its approach differs from the previous network. SSD [66] parameterizes the output space of the bounding box into different default boxes for different aspect ratios and scales for each feature map location. At prediction, the network generates scores for each category in each default box and makes adjustments to the box so that it adapts to the shape of the object.

In other words, unlike YOLO which detects the object and based on resamples of pixels or features gives bounding boxes hypotheses, SSD defines a set of default

bounding boxes of different scales and scores the presence of the object in those boxes, that will say which one of the default boxes matches the detected object, improving the speed of the network.

This is done by including a convolutional filter to predict object categories and offsets in locations of bounding boxes. By using separate filters for different aspect ratio detections and applying them to multiple feature maps generated at later stages of a network, SSD is capable of performing detection at multiple scales.




(a) Image with GT boxes (b) $8 \times 8$ feature map (c) $4 \times 4$ feature map

Figure 4.12: SSD way of working [66].

Figure 4.12 shows an example of how SSD works, that only needs an input image and ground truth boxes for each class during training:

1. A small set of default boxes at each location in several feature maps will be evaluated;

2. For each default box, the shape offset and confidence for the object will be predicted;

3. The default boxes will be matched with the ground truth boxes - an example where four default boxes exist and only two boxes match the object, the two boxes that matched are treated as positives and the rest as negatives;

4. Finally, a non-maximum suppression, that removes the repeated detections, is applied to get the final detection.

The architecture of the model in Figure 5.18 is based on a feed-forward convolutional network, meaning that it processes data in a one-way flow, and it is composed of the backbone and the head.

For the backbone, the authors used VGG-16 to build the network, and the head consists of convolutional layers with some key features to make detections, such as multiscale feature maps, convolutional predictors for detection and default boxes and aspect ratios:

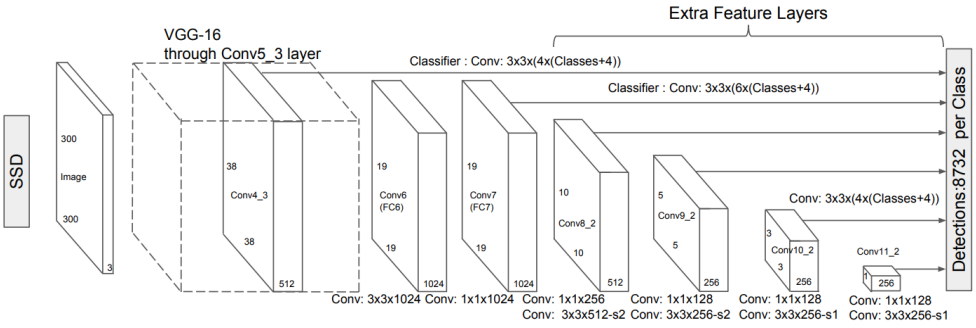



Figure 4.13: SSD architecture [66].

- Multiscale feature maps for detection - present in the convolutional layers that are added after the backbone. These layers progressively decrease their size allowing predictions of detections at multiple scales, as visible in Figure 4.14, being the convolutional model for predicting detections differently for each feature layer;

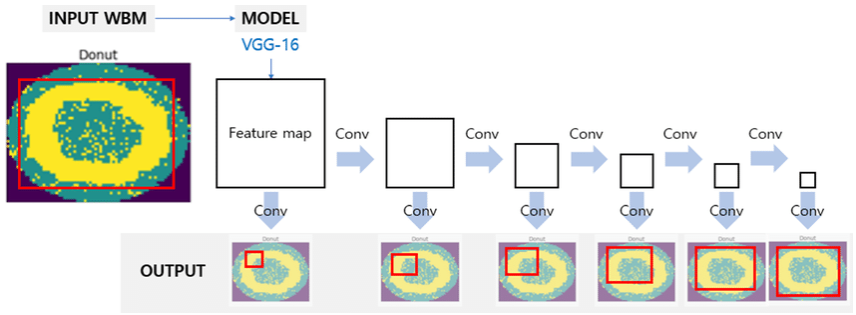



Figure 4.14: Multiscale feature maps for detection [67].

- Convolutional predictors for detection - each feature layer that is added after the backbone can output a fixed number of detection predictions using convolutional filters. For a feature layer with a size of $m * n$ using $p$ channels, the basic element for predicting potential detection parameters is a $3 * 3 * p$ small kernel that gives either a category score or the offset values of the bounding box relative to the default coordinates of the box;

- Default boxes and aspect ratios - a group of bounding boxes is associated with each feature map cell at the beginning of the network. The default boxes tile the feature map in a convolutional way, making each box's position relative to its corresponding cell fixed. Then, at each feature map cell, the offsets relative to the default box shapes are predicted, as well as the scores. Being k the total number of boxes at a given location, c class scores and 4 offsets are predicted for each box, which results in a total of $(c+4*k)$ filters that are applied around each location of the feature map, as visible in Figure 4.15.

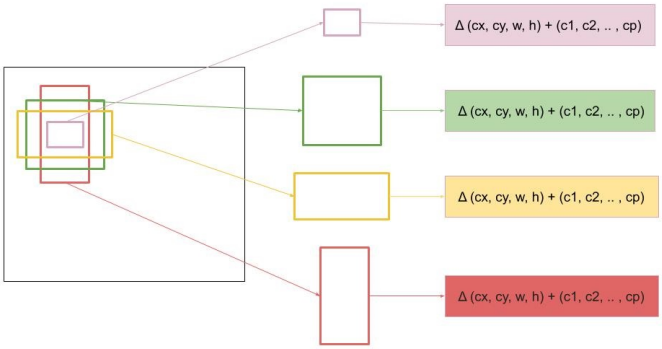



Figure 4.15: Predicted offsets for each box [68].

The training strategy at SSD also differs when compared to other networks and it involves the assignment of ground truth information to specific outputs, choosing the set of default boxes and scales, hard negative mining and data augmentation strategies:

- Matching strategy - this corresponds to the assignment of ground truth detection to the default boxes:

  1. Each ground truth box is matched with the default box to which has the best IoU;
  2. The default boxes are matched to any ground truth with an IoU>0.5;
  3. The two previous points allow the network to predict high scores for several overlapping default boxes and not need to choose only the one with maximum overlap.

- Choosing scales and aspect ratios for default boxes - the bounding boxes were designed so that specific feature maps learn to respond to certain scales of objects. Given $m$ as the number of feature maps for prediction, the scale of the default boxes for each feature map is calculated by expression (4.6):

$$s_k = s\_min + \frac{s\_max - s\_min}{m - 1} * (k - 1), k \in [1, m] \tag{4.6}$$

  where $s\_min$ and $s\_max$ correspond to the lowest and highest layer, respectively and have a scale of 0.2 and 0.9;

- Hard negative mining - after the matching strategy, some of the boxes are negative and, instead of using all negative examples, the network classifies them using the highest confidence loss for each default box. Then, the best ones are selected, maintaining a maximum ratio of 3:1 between the negatives and positives;

- Data augmentation - each training image suffers augmentation, that consists in doing modifications to the original data, such as crop, changes of colour, and rotation, among others, as observable in Figure 4.16. This technique makes the model more solid to different input object sizes and shapes without collecting new data and therefore reduces overfitting [69].

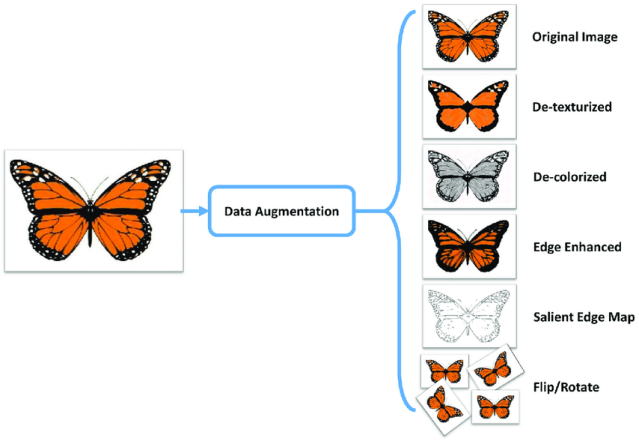



Figure 4.16: Data augmentation [69].

The network was trained on several datasets such as COCO and PASCAL VOC, and when comparing its performance, the authors were able to conclude that it outperforms Faster R-CNN in terms of accuracy and speed, but it does not match the performance of YOLOv5.

### 4.1.3 Retinanet

Retinanet is yet another one-stage network [70, 61], created by Facebook AI Research, it was developed to try to achieve the accuracy of two-stage detectors since single-stage networks are simpler and faster. For that, unlike SSD or YOLO, the focus was on the loss function - focal loss - with the goal of eliminating class imbalance during training. It is based on the cross-entropy loss function, which measures the difference between the predicted class probabilities and the true class labels.

However, in some images a big part of the pixels or regions do not contain any objects of interest which can lead to numerous negative examples, resulting in a class imbalance and therefore the authors modified the original cross-entropy loss function to give more weight to hard examples, that are difficult to classify correctly. The focal loss function is dynamically scaled based on the confidence of the model's prediction and is multiplied by a scaling factor, that will decrease as the confidence in the correct class increases, meaning that the loss for easy examples is down-weighted, while the loss for hard examples is up-weighted, helping the model to focus on difficult examples and improve its overall performance.

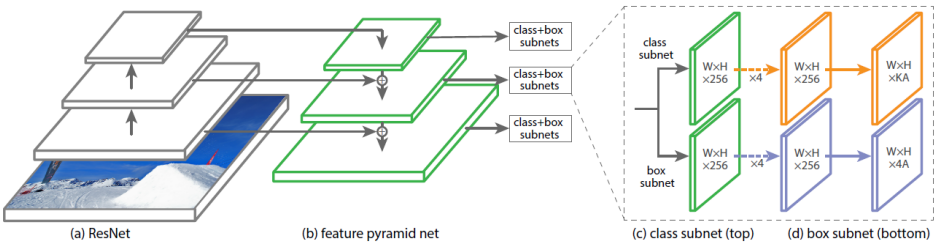



Figure 4.17: Retinanet architecture [70].

Figure 4.17 shows Retinanet's architecture, which is a single, unified network, made by a backbone network: ResNET-101, a Feature Pyramid Network (FPN) and two task-specific subnetworks, one for classifying anchor boxes (c) and one for receding from anchor boxes to ground-truth object boxes (d). The network works as follows:

1. The backbone does the computing of a convolutional feature map over an entire image;
2. The first network performs convolutional object classification of the output of the backbone;
3. The last subnet does convolutional bounding box regression.

Besides the focal loss function, there are some components inherent to this network:

- FPN - it provides a methodology for object detection at different scales, as each stage of the network corresponds to a sequence of pyramid levels and has multiple convolutional layers of the same size scaled down by a factor of two at every stage. The idea behind FPN can be seen in Figure 4.18;

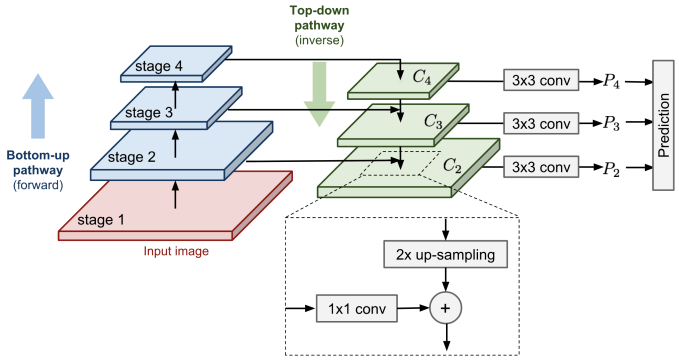



Figure 4.18: FPN [61].

- Anchors - the authors used translation-invariant anchor boxes with different areas on the different pyramid levels which are assigned to ground-truth object

boxes using an IoU threshold of 0.5. A maximum of one object box is assigned to each anchor, and if any is unassigned it is ignored during training;

- Classification subnet - it predicts the probability of the presence of an object at each spatial position for every of the anchor boxes and object classes. The subnet is a small fully convolutional network attached to each FPN level and its parameters are shared among all pyramid levels. An example of how this subnetwork works would be:

  1. Taking an input feature map with 256 channels, the subnet applies four $3 * 3$ convolutional layers;
  2. Each convolutional layer has 256 filters (the same number as channels) that are followed by ReLU activations;
  3. After this, the subnetwork has another $3 * 3$ convolutional layer with $KA$ filters, where $K$ is the number of object classes and $A$ the number of anchor boxes;
  4. Finally, sigmoid activations are attached to output the $KA$ binary predictions per spatial location.

- Box regression subnet - in parallel with the previous subnet, the authors included another small full convolutional network to every pyramid level with the purpose of regressing the offset from each anchor box to an adjacent ground-truth object, if existing. The four outputs of the subnet predict the relative offset between the anchor and the ground-truth box, for each anchor per spatial location.

The authors compared Retinanet with SSD, YOLOv2 and Faster R-CNN on the COCO dataset and, as seen in Figure 4.19, Retinanet outperforms the mentioned network in terms of accuracy.

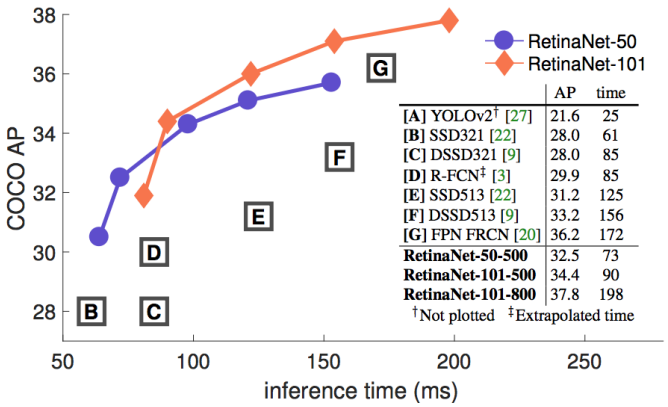



Figure 4.19: Retinanet vs. other networks [70].

### 4.1.4 Faster R-CNN

Unlike the networks presented before, which directly predict the class and bounding box coordinates of objects in a single pass through the network, without a separate proposal generation stage, Faster R-CNN is a two-stage detector [71] since it first generates a set of object proposals (in this particular case using a Region Proposal Network (RPN)) and then classifies the object within the proposals.

Faster R-CNN is a single, unified network for object detection that is composed of two modules: the first one, a deep fully convolutional network that proposes regions (RPN) and the second one the Fast R-CNN detector that uses the proposed regions. The training scheme of this network consists of the alternation between fine-tuning for the region proposal task and then fine-tuning for the object detection, while the proposals remain fixed.

The architecture of Faster R-CNN can be seen in Figure 4.20.

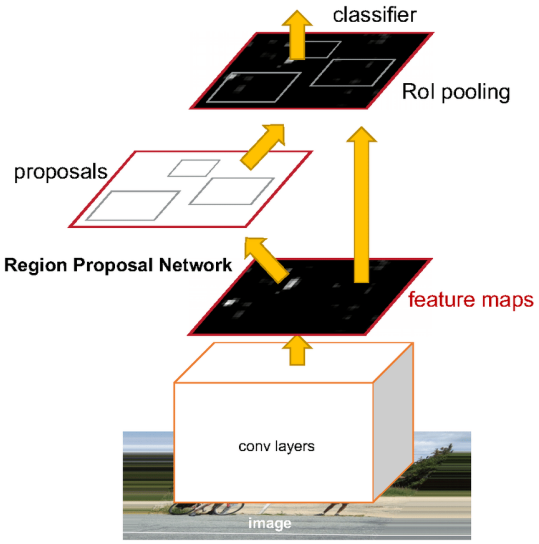



Figure 4.20: Faster R-CNN architecture [71].

The first module - RPN - visible in Figure 4.21, has as an input an image of any size and then outputs a group of rectangular object proposals, each with an objectness score. It corresponds to a vector of two elements for each region proposal that can classify the region as background or object.

It is used since the final goal is to share computation with the second model, so both networks share some convolutional layers. Given that, to generate region proposals, the authors slide a small network over the convolutional feature map output by the last shared convolutional layer and each sliding window is mapped to a lower dimensional feature that serves as an input of two sibling fully-connected layers - a box-regression layer (`reg`) and a box-classification layer (`cls`).

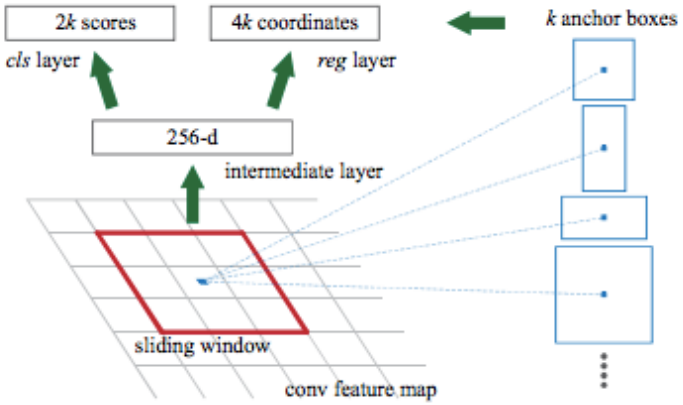



Figure 4.21: RPN architecture [71].

There are some key concepts of RPN:

- Anchors - in each location of the sliding-windows, there is the prediction of multiple region proposals, where the `reg` layer has $4k$ outputs encoding the coordinates of $k$ boxes and `cls` layers outputs $2k$ scores that give an estimation of the probability of object or background for each proposed region ($k$ corresponds to the maximum possible proposals for each location):

    1. The anchors are translation invariant, a property that helps to reduce the model size, which means that if the translation of an object in an image occurs, the proposal should also translate and the same function should be capable of predicting the proposal in either location;
    2. The anchors are multiscale, meaning that the classification and regression of bounding boxes occur with reference to anchor boxes of multiple scales and aspect ratios. The network only uses images and feature maps of a single scale and filters of a single size, which allows the use of the convolutional features computed on a single-scale image.

The second module, Fast R-CNN (Figure 4.22) is a network developed to overcome some of the main issues of R-CNN [72] such as the time spent in training.

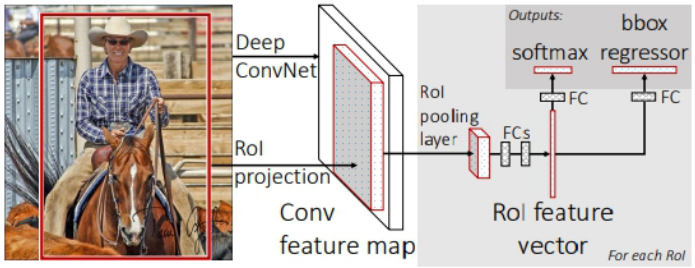



Figure 4.22: Fast R-CNN architecture [72].

Fast R-CNN works as follows:

1. From the generated feature map of an image, region proposals are identified and resized to a fixed size by a Region of Interest (RoI) pooling layer;

2. Softmax layer is used to identify the objects within each of the proposed regions and to predict four offsets.

The architecture of Faster R-CNN was designed to overcome the main issue of Fast R-CNN [71] - the fact that the network depends on Selective Search to generate region proposals. It consists of a time-consuming algorithm that cannot be customized for specific object detection tasks and therefore presents a lack of accuracy in detecting all target objects.

To achieve Faster R-CNN, the authors developed a technique that is capable of sharing convolutional layers between the two networks (RPN and Fast R-CNN) named alternating training. Faster R-CNN is trained in four different steps:

1. RPN is trained. It is initialized with a pre-trained model and fine-tuned end-to-end for the region proposal;

2. Fast R-CNN is trained using the proposals generated by the first step;

3. The detector network is used to initialize RPN training and the shared convolutional layers are fixed, fine-tuning the layers that are exclusively unique to RPN;

4. Keeping the shared convolutional layers fixed, the individual layers of Fast R-CNN are fine-tuned.

With this four-step method, both networks share the same convolutional layers and form a unified network. Faster R-CNN, is one of the most accurate networks when compared with YOLOv5, SSD and Retinanet, however, it is the slowest network and in which the training relies on more computation resources.

## 4.2 Performance Metrics

When it comes to DL some concepts such as batch, epoch, iterations, learning rate, overfitting and underfitting are needed to take into consideration while training the network [73, 74, 75, 76, 77]:

- Batch - corresponds to the number of samples (a sample is a single instance of data) used during training, being the model updated at the end of the batch and the predictions compared to the expected output;

- Epochs - defines the number of times that the algorithm will go through the entire dataset, corresponding, in the case of neural networks, to the forward propagation and back-propagation;
- Iteration - it refers to the number of batches needed to finish one epoch. Figure 4.23 shows the comparison between batch, epoch and iteration;

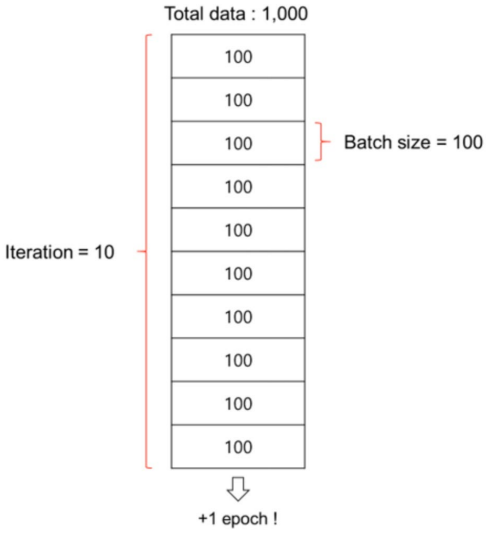



Figure 4.23: Batch vs epoch vs iteration [78].

- Learning Rate - one of the most important hyperparameters and it is used to set the rhythm at which the algorithm updates the weights of the neural network during training;
- Overfitting - it occurs when the train error is small, but the test error is big, happening usually when the model gets trained with an excessive amount of data and, because of that, it starts learning from the noise and inaccurate data entries in the dataset;
- Underfitting - the opposite of overfitting. It happens when both the training and testing errors are large and the main causes of it are the use of not enough data to train or the usage of a model that is too simple for the data. Figure 4.24 shows the difference between overfitting and underfitting.

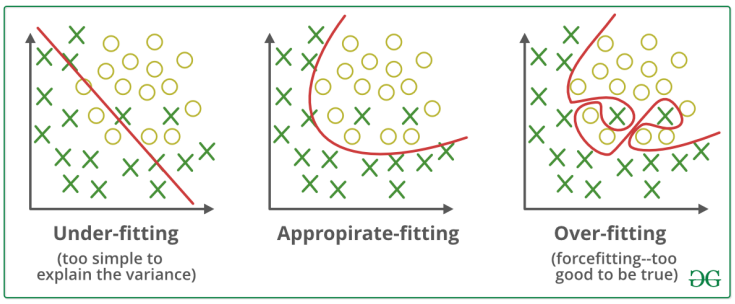



Figure 4.24: Overfitting vs underfitting [77].

After the training is completed, some metrics need to be analysed in order to evaluate the used model, being the most used ones, for object detection, precision, recall, F1 score, Average Precision (AP), Mean Average Precision (mAP), confusion matrix and accuracy.

### 4.2.1 Precision, Recall and F1 Score

Both precision and recall are based on other four concepts [79]:

1. True Positive (TP) - corresponds to the correct detection of a ground-truth bounding box;
2. False Positive (FP) - represents an incorrect detection of an object that does not exist or a misplaced detection of an object that exists;
3. False Negative (FN) - corresponds to the non-detection of a ground-truth bound box;
4. True Negative (TN) - corresponds to the correct detection of an object that should not be detected.

The correct and incorrect detection are defined using the IoU, as mentioned previously and it measures the area between the predicted bounding box and the ground-truth bounding box that is overlapped, dividing it by the area of union between them. The classification of a detection as correct or incorrect is made by comparing the IoU with a given threshold, therefore if $IoU \geq threshold$ the detection is correct, oppositely if $IoU < threshold$ the detection is wrong [79].

The equations for the precision $P$ and recall $R$ can be calculated by the following expressions (4.7) and (4.8), respectively:

$$P = \frac{TP}{TP + FP} = \frac{TP}{all\_detections} \tag{4.7}$$

$$R = \frac{TP}{TP + FN} = \frac{TP}{all\_ground\_truths} \tag{4.8}$$

Precision is defined as the model's capability of identifying only relevant objects and so it is the percentage of correct positive predictions. On the other hand, recall is the model's aptitude to find all relevant cases (all ground-truth bounding boxes), therefore it is the percentage of correct positive predictions in between all ground truths [79].

F1 score [80] corresponds to the harmonic mean of precision and recall and can be calculated with the following formula (4.9):

$$F1score = 2 * \frac{P * R}{P + R} \tag{4.9}$$

Since the F1 score is an average of both precision and recall, a model will obtain a high F1 score if the other two parameters are high and, in contrast, it will obtain a low F1 score if the two parameters, $P$ and $R$, are low.

### 4.2.2 Average Precision and Mean Average Precision

Average Precision (AP) is the area under the precision-recall curve evaluated at an IoU threshold, usually 50% (AP50) or 75% (AP75). Being a particular object detector considered good if the precision remains high as the recall increases [79, 81].

Since the Precision x Recall curve has a zigzag behaviour, interpolation methods are used to remove it:

- 11-point interpolation method - is a plot of interpolated precision scores for model results, equally spaced, from 0.0 until 1.0 (11 points), as seen in Figure 4.25;

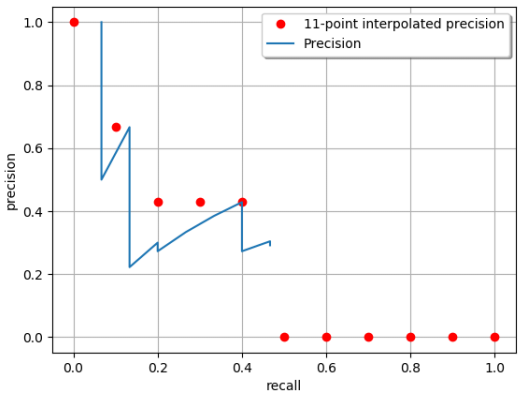



Figure 4.25: 11-point interpolation method [79].

- All-point interpolation method (Figure 4.26) - the interpolation is done for all the recall values.

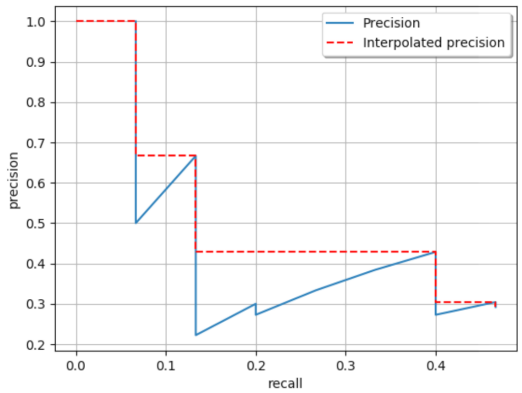



Figure 4.26: All-point interpolation method [79].

The mAP is one of the most used metrics for object detectors and it is used to measure object detectors' accuracy over all classes in a specific database. The formula of mAP is:

$$mAP = \frac{1}{N} * \sum_{i=1}^{N} AP_i \tag{4.10}$$

Being the mAP the mean of AP, $AP_i$ corresponds to the AP in the $i^{th}$ class and $N$ represents the total classes being evaluated.

### 4.2.3 Confusion Matrix and Accuracy

The confusion matrix [82] also uses the concepts of TP, FP, FN and TN and it is a visual representation of the ground-truth labels in contrast to the model predictions. Each row of the matrix corresponds to the instances in a predicted class and each column represents the instances in an actual class, as visible in Figure 4.27.

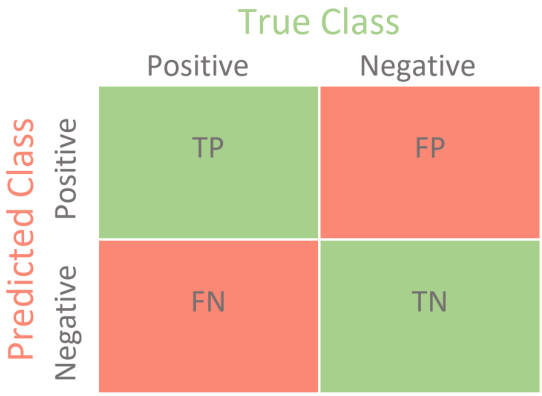



Figure 4.27: Confusion matrix example [82].

Accuracy [82] is the most known metric and it gives the percentage of correct predictions in all total predictions, as:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.11}$$

## 4.3 Conclusions

The main characteristics of the four presented networks can be seen in Table 4.1.

Table 4.1: Convolutional Neural Networks.

| **YOLOv5s** | **SSD** | **Retinanet** | **Faster R-CNN** |
|---|---|---|---|
| One-stage detector. | One-stage detector. | One-stage detector. | Two-stage detector. |
| Made by: backbone (Darknet53 with CSP network strategy), neck (PANet) and head (3 convolutional layers). | Composed by backbone (original is VGG-16) and head (convolutional layers with some key features for detection). | Made by backbone (ResNET-101), a FPN and two task-specific subnetworks. | Made by a deep fully convolutional network (RPN) and the Fast R-CNN detector. |
| Used mainly due to its speed, available community and easy utilization. | Used in fields of object detection where real-time processing is needed due to its flexibility and accuracy. | Outperforms SSD, YOLOv2 and Faster R-CNN in terms of accuracy for the COCO dataset. | Designed to overcome the main issue of Fast R-CNN: the dependence on Selective Search to generate region proposals. |
| Activation functions: sigmoid and SiLU. | Outperforms Faster R-CNN in terms of accuracy and speed, but does not match the performance of YOLOv5. | Activation functions: sigmoid and ReLU. | One of the most accurate networks, however slower and depends on more computational resources. |
| Way of working: the network detects the object and based on resamples of pixels or features gives bounding boxes hypothesis. | Detection is made by defining a set of default bounding boxes with different scales and then scores the presence of the object in those boxes. | Way of working: the backbone does the feature map, the first network performs object classification of the backbone's output and the last subnetwork does bounding box regression. | Detection is made in two different stages: first a set of object proposals is generated and then the object is classified within the proposals. |

In summary, DL networks are currently used mostly in day-to-day activities that require image processing, such as surveillance systems or medical purposes, due to

its accuracy and fast training mostly because of the use of the transfer learning technique.

Several networks, like Resnet or VGG-16 that are pre-trained for object detection, are the base of object detection networks, making these networks very accurate. However, not all algorithms can be used for the same purpose, when it comes to real-time image processing, one-stage detectors are normally used due to their speed, however when that is not the goal and there are good computational resources a two-stage detector might be a good choice as it is, usually, more accurate than the single-stage networks.

When it comes to performance metrics, there are two key concepts: accuracy and mAP, being the last one particularly important in object detectors.

# Chapter 5

# Developed Work and Results

In this Chapter, all the developed work, which was divided into three main phases: proof of concept to understand the viability of the project, training of the DL networks and improvements to the initial developments, will be described. Also, all the tests made will be presented as well as the results' analysis and comparison.

## 5.1 First Phase

For a first step, the full cycle was made, as a proof of concept, in order to understand if the implementation of the project was possible or not. It consisted of the following points:

- Development of a Python script to record the manual tests;
- Train and test a DL algorithm;
- Development and test of the Robot Framework template.

### 5.1.1 Recording Tool

The process flowchart illustrated in Figure 5.1 delineates the sequential actions performed for this specific step.

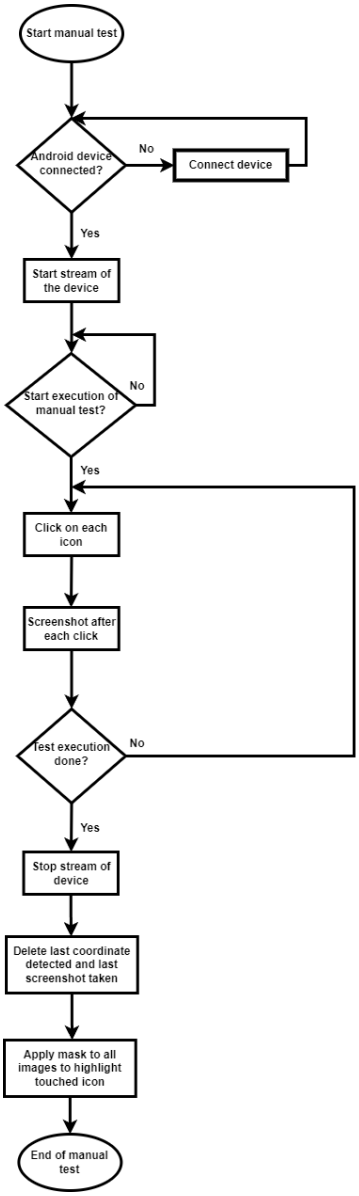



Figure 5.1: Process flowchart of the recording tool.

The recording tool implementation involved the development of a Python script designed to detect manual clicks on the Android Automotive infotainment and subsequently capture screenshots. Each manual interaction resulted in the creation of a dedicated folder that stored the screenshots in the order they were taken.

To facilitate the device streaming process, the `scrcpy` tool was used, as it consists of an app capable of mirroring any Android device, as depicted in Figure 5.2.


Figure 5.2: Target streaming.

After the stream was initiated, every mouse click was detected and after each detection, a screenshot was taken. Additionally, the pair of coordinates `x,y` of each touch event was saved in a text file, as illustrated in Listing 5.1.

```
1  def click(x, y, button, pressed):
2      date = datetime.now()
3      hour = date.strftime("%Y-%m-%d_%H%M%S")
4      logging.basicConfig(filename="../coordinates.txt", level=
           logging.INFO, format='%(message)s')
5      # if button number 1
6      if pressed:
7          time.sleep(1)
8          subprocess.call("adb exec-out screencap -p > Screenshots/"
               + hour +".png", shell=True) #screenshot after click
9          logging.info('{0}, {1}'.format(x,y)) #logs all touched
               coordinates to text file
10         Event().wait(1) #waits 1 second to check if the subprocess
               is still alive
11         if stream.poll() is not None:
12             listener.stop()
13             # if button number 1 is pressed (clicked)
14         else:
15             return False
16     while stream.poll() is None:
17         listener = mouse.Listener(on_click=click)
18         listener.start()
19         listener.join()
```

Listing 5.1: Capture of each mouse click.

The detection of each mouse click was made using the `pyinput` library that allows the control and monitoring of input devices, such as the computer mouse.

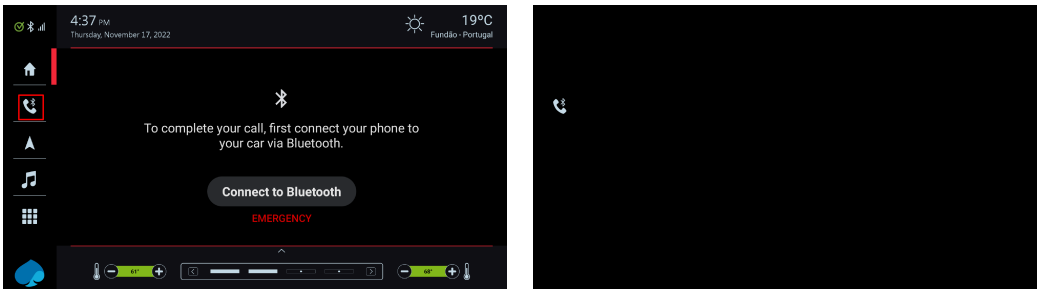
Upon detecting a mouse click event, the script executed a subprocess to invoke an Android Debug Bridge (adb) command, which captured a screenshot of the Android device's screen.

The main function continuously listens for user inputs, specifically mouse clicks, while the device streaming process is ongoing. The streaming process is managed by a parallel subprocess referenced as the `stream` variable. As long as the `stream` variable holds a value of `None`, indicating that the streaming is still active, the function remains in listening mode, capturing and logging every mouse click event. However, once the streaming subprocess is terminated, the `stream` variable returns `0`, signifying the end of the streaming.

To prevent any false inputs caused by detecting the closing of the mirroring window, the developed tool includes a step to delete the last screenshot and its corresponding coordinates entry. By doing this, the script guarantees that only valid and relevant data is used for the next steps, improving the overall reliability and precision of the system.

Finally, the pressed icon in each step will be highlighted and a mask will be applied. This masked image is then utilized as input for the pre-trained neural network.

Figure 5.3a displays a screenshot taken during a manual test with the highlighted pressed icon, and Figure 5.3b exhibits the same image with the mask applied.




(a) Pressed icon highlighted. (b) Mask applied in image.

Figure 5.3: Image processing.

In this project, the mask serves the specific purpose of isolating the pressed icon within the image, ensuring that it is the only visible icon. Since the network will be trained to recognize the majority of the icons present in the infotainment system, if the raw image was used as input for the neural network, the output would contain the positions and labels of all detected icons in the image. By doing this strategy, as the neural network's input focuses only on this isolated region, the output will be limited to recognize only the specific pressed icon, rather than identifying all present in the image.

In order to be able to apply the mask to the original image, `opencv` - a Python open-source computer vision library - was used. Listing 5.2 represents an extract of the developed code in order to achieve the final image.

```
1     image = np.array(screenshots)
2     mask = np.zeros(image.shape[:2], np.uint8)
3     x_corner = x - 75
4     y_corner = y - 75
5     roi = (x_corner, y_corner, 150, 150)  # (x, y, width, height)
6     cv2.rectangle(mask, (roi[0], roi[1]), (roi[0] + roi[2], roi[1]
          + roi[3]), 255, -1)
7     masked_image = cv2.bitwise_and(image, image, mask=mask) #
          Apply the mask to the image
```

Listing 5.2: Mask of the original image.

A `for` cycle was used in order to pass through all the images in a folder, and in each iteration of the cycle the image to be processed was stored in an `image` variable.

The first line of Listing 5.2 creates a blank black mask image with the same dimensions as the `image` variable. The `image.shape[:2]` returns the height and width of the image, and `np.zeros()` creates a `NumPy` array filled with zeros, forming the initial blank mask.

Subsequently, the corner coordinates are calculated to define the Region of Interest (RoI) that includes the entire icon. The values `x_corner` and `y_corner` are calculated by subtracting 75 from the centre coordinates (`x, y`), ensuring that the RoI includes the complete icon. The RoI is then calculated with the dimensions `(x_corner, y_corner, 150, 150)`. The choice of 150 for both width and height ensures a square that delimits only the icon, as it is twice the distance from the centre to the corner.

Finally, the last two lines create the masked image. The code first draws a rectangle on the output with the specified dimensions, colour and thickness, effectively creating the mask. Then, the mask is applied to the original image using a bitwise AND operation: `masked_image = cv2.bitwise_and(image, image, mask=mask)`. This operation results in a new image that only contains the pixels of the original screenshot within the specified RoI with the rest of the image black, as visible in Figure 5.3b.

### 5.1.2 YOLOv5s Training

For the first phase of the project, the train was only made for two classes using the YOLOv5s network implemented with PyTorch. The selection of this network was made due to the available information and tutorials available from the authors [55]. As mentioned previously, the main goal of this phase was to see the possibility of

continuing with the project or not, therefore no type of fine-tuning or changes to the available network were made, except the use of a custom dataset.

The decision to opt for the smaller version of the YOLOv5 model (YOLOv5s) was driven by resource limitations. Since the company's computer lacked a dedicated Graphics Processing Unit (GPU), training a bigger version of YOLOv5 on it would be highly time-consuming and, in some cases, not even possible. To overcome this limitation, the project made use of Google Colab [83], a product from Google Research that consists of a hosted Jupyter notebook service that requires no setup to use and offers free resources such as GPU, being that widely used for the training of ML and DL networks. For this part of the project, Google Colab free was used which offers a T4 GPU for a limited time.

At this stage, the custom dataset used in the majority of the trains was composed of 194 images, with a split of 70% training, 20% validation and 10% test. Among the 194 images, only 10 were original screenshots, directly taken from the target, as the infotainment display consists of a static image.

To augment the dataset and increase the number of images available for training, a Python script was developed. This script utilized various data augmentation techniques, such as cropping, rotating, blurring, and changing colours. The `Augmentor` package was used to implement these augmentation techniques, as shown in Listing 5.3.

```
1 p = Augmentor.Pipeline("Screenshots/")
2 p.rotate(probability=0.7, max_left_rotation=10,
      max_right_rotation=10)
3 p.zoom(probability=0.2, min_factor=1.1, max_factor=1.6)
4 p.random_color(probability=0.8, min_factor=0.2, max_factor
      =0.9)
5 p.random_brightness(probability=0.6, min_factor=0.3,
      max_factor=1.5)
6 p.random_contrast(probability=0.4, min_factor=0.4, max_factor
      =1.0)
7 p.random_distortion(probability=1.0, grid_width=7, grid_height
      =7, magnitude=8)
8 p.sample(300)
```

Listing 5.3: Data augmentation.

The `Augmentor` library [84] allows several augmentation techniques with different parameters, and as seen in the previous code excerpt, 300 images were generated, however, only 184 could be used as the rest were either repeated images or black images.

The annotations were made using Roboflow [85], a platform that besides empowering developers to create their computer vision model, also provides all tools

to go from raw images to a custom dataset for the specific model to train. For this case, the images were annotated with two different classes: *connectivity* and *home* and the dataset was exported in the model accepted by the YOLOv5s network - YOLOv5 PyTorch.

The first training of the YOLOv5s network was conducted using the available network weights for 100 epochs. The training process involved using a batch size of 16, and the input images were resized to a resolution of 384x640 pixels. The dataset utilized for training contained 151 images, which were split into three subsets: 70% for training, 20% for validation, and 10% for testing.

After cloning the network's repository, the train was initialized by doing: `python train.py --img 640 --batch 16 --epochs 100 --data data.yaml --weights yolov5s.pt`, where:

- `--img 640` - specifies that the input images will be resized to a resolution of 640 pixels in the width and an appropriate height to maintain the aspect ratio;
- `--batch 16` - sets the batch size to 16;
- `--epochs 100` - sets the number of training epochs to 100;
- `--data data.yaml` - refers to the `data.yaml` file, which provides necessary information about the dataset like the number and name of the classes and paths to the train, validation and test images. This file is automatically generated when exporting the dataset using Roboflow and it is essential for YOLOv5;
- `--weights yolov5s.pt` - specifies the initial network weights used for training. In this case, the weights for the YOLOv5s model were loaded to begin training.

The first train presented good training and validation results, as illustrated in Figure 5.4.

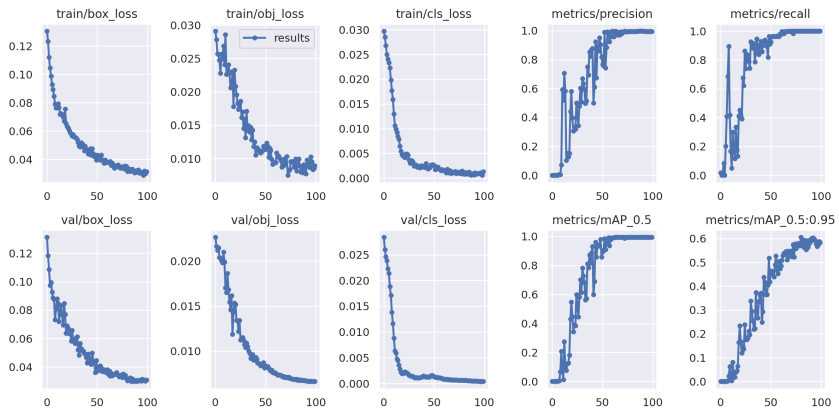



Figure 5.4: Results of the first train.

The detailed values of each parameter, at the last epoch, are visible in Table 5.1.

Table 5.1: Results of the first training at the $100^{th}$ epoch.

| **Metric** | **Value** |
|---|---|
| Train Box Loss | 0.031198 |
| Train Object Loss | 0.0089688 |
| Train Classification Loss | 0.0012936 |
| Precision | 0.99457 |
| Recall | 1 |
| mAP_0.5 | 0.995 |
| mAP_0.5:0.95 | 0.58306 |
| Validation Box Loss | 0.006644 |
| Validation Object Loss | 0.00039883 |
| Validation Classification Loss | 0.000298 |

By analysing the results some conclusions can be taken:

- Train Box Loss - this metric represents the average error in predicting the bounding box coordinates. During the training this value was continuously decreasing, achieving a low and stable value, which indicates better accuracy in predicting the bounding boxes around the detected icons;
- Train Object Loss - it measures how well the model detects objects within the predicted bounding boxes;
- Train Classification Loss - it represents the average loss in classifying the objects during training and it measures the accuracy during classification. With the obtained results it is also possible to see that this value was decreasing over the epochs achieving a low classification loss by the end of the training, indicating a good performance of the model;
- Precision - the obtained value indicates that approximately 99.46% of the detected icons on the validation dataset were correctly identified by the model;
- Recall - the obtained value indicates that all the positive instances in the dataset were correctly detected by the model;
- mAP - it assesses the model's accuracy in localizing the detected objects with a certain level of overlap. The second mAP presented at IoU thresholds from 0.5 to 0.95 provides a more comprehensive evaluation of the model and the obtained value of 58.31 % shows that the model's detections have moderate accuracy in localizing objects, meaning that even though the model is capable of identifying the objects it can still miss certain objects or have false detections;
- Validation Box Loss - similar to the first metric it represents the average error in predicting the bounding box coordinates but in this case for unseen data.

The obtained value was close to the obtained during training, indicating a low error even on images not used in training;

- Validation Object Loss - like the training object loss it indicates the average loss in predicting the presence of the icon in the bounding boxes. Like in the previous metric, the validation value was similar to the one obtained during training, indicating again a good performance of the model;
- Validation Classification Loss - the same goal as the train classification loss metric. Unlike the two previous metrics, this particular one behaves well for unseen data as the value is lower than for training data.

The confusion matrix for the validation dataset can be seen in Figure 5.5.

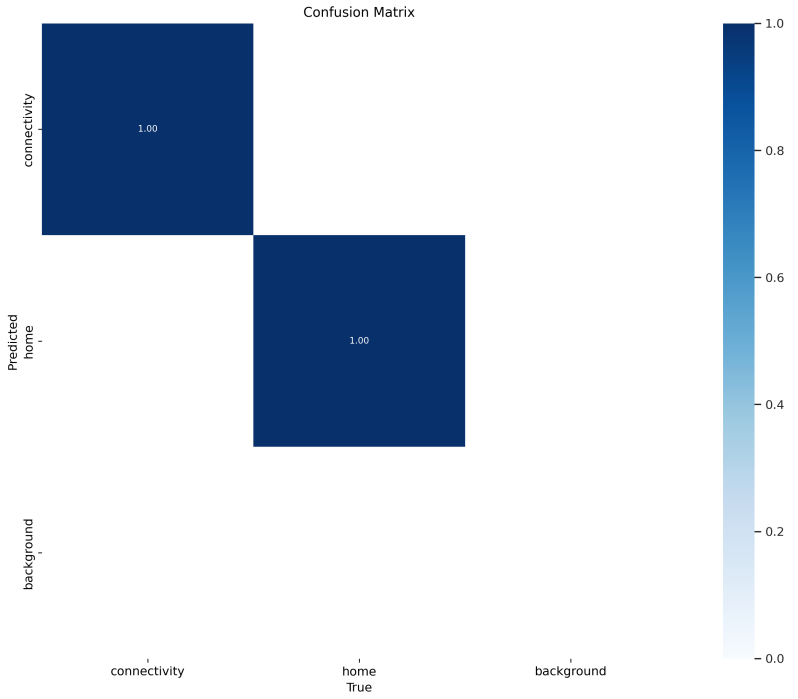



Figure 5.5: Obtained confusion matrix.

The obtained matrix confirms the obtained value for the recall parameter, as it correctly identifies all relevant instances for each class. This was again observed during testing, since for the 15 images used in testing the network correctly identified all instances, giving an accuracy of 100% for this first stage and with a maximum confidence of 85% for the *connectivity* class and 90% for *home*, as visible in Figure 5.6.


Figure 5.6: Detection in a test image.

However, when using one of the masked images, which would be the input of the network, the model was not capable of performing detections since there were no images of the same type were used in training. This caused the need to increase the training dataset to include masked images, reaching the final dataset for this phase with 194 images, as mentioned.

For this new dataset, several other trainings were made, where the changes consisted of the number of epochs (from 150 to 200) and batch size (16 and 32). The best results were achieved by training the network during 170 epochs with a batch of 32.

Mostly the metrics of validation box loss (as the final goal of the project was to have a correct detection of the icons' positions), mAP, confidence in the detection of the masked images and accuracy in the detection were taken into consideration to select the mentioned values for number of epochs and batch size as the best ones for the training of the model.

Table 5.2 represents some of the different tests made and their respective results on the mentioned metrics, using a test dataset with 21 images (both masked images and original screenshots) and 35 icons of the 2 classes, in total.

Table 5.2: Tests on new dataset and respective results.

| **Test** | **mAP** | **Validation Box Loss** | **Accuracy** | **Maximum Confidence** |
|---|---|---|---|---|
| 150 epochs and batch 16 | 0.60122 | 0.028979 | 100% | 80% *home* and 87% *connectivity* |
| 150 epochs and batch 32 | 0.62482 | 0.029116 | 95% | 71% *home* and 62% *connectivity* |
| 170 epochs and batch 16 | 0.60355 | 0.028548 | 57% | No detections in masked images. |
| 170 epochs and batch 32 | 0.61972 | 0.028752 | 100% | 89% *home* and 89% *connectivity* |
| 200 epochs and batch 16 | 0.6001 | 0.030054 | 100% | 84% *home* and 70% *connectivity* |
| 200 epochs and batch 32 | 0.60238 | 0.029135 | 89% | 66% *home* and 74% *connectivity* |

Based on the analysis of the presented results, it is possible to conclude that even though there were no major differences between the first and fourth tests, this last one presented better overall values. Figure 5.7 shows all obtained results during this particular training.

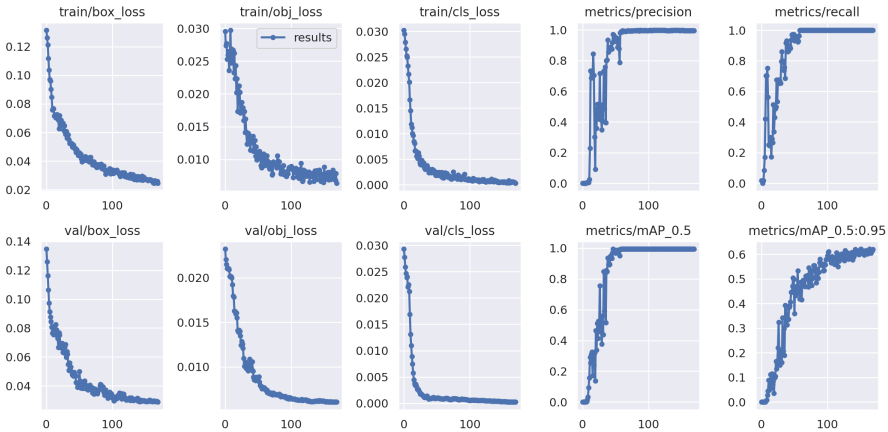



Figure 5.7: Results of the best train.

The detailed metrics' values of the last epoch can be seen in Table 5.3.

Table 5.3: Results of the best training for 2 classes at the $170^{th}$ epoch.

| **Metric** | **Value** |
|---|---|
| Train Box Loss | 0.024501 |
| Train Object Loss | 0.0063505 |
| Train Classification Loss | 0.00028048 |
| Precision | 0.99659 |
| Recall | 1 |
| mAP_0.5 | 0.995 |
| mAP_0.5:0.95 | 0.61972 |
| Validation Box Loss | 0.028752 |
| Validation Object Loss | 0.0060754 |
| Validation Classification Loss | 0.00024662 |

When comparing to the results obtained in the first training, it is possible to see that there was an improvement in the training results as the values of the box and object loss decreased. During validation it was possible to observe a slight increase in the mAP, leading to the conclusion that the accuracy in locating the objects in the dataset for different values of IoU thresholds increased, even though the values of the validation box, object and classification loss were higher.

Also, it was possible to conclude during testing, that the accuracy continued at 100% as the network was capable of correctly identifying all icons in the images, including masked images. There was also a slight difference, as possible to see in Figure 5.8 in the confidence levels, particularly in the *connectivity* icon, when compared to the first training.


Figure 5.8: Detection results in a test image.

### 5.1.3 Robot Framework Template

To understand how the final part of the project could be approached, in this third phase a general Robot Framework template was created with the goal of being used to execute all test cases.

Robot Framework [86] is a Python-based open-source framework for test automation, that is mostly used for acceptance testing where its main difference and advantage when compared to other test automation frameworks is the fact that Robot Framework is a keyword-driven framework that provides a readable and expressive syntax making it easy to create and maintain tests. Besides having a built-in library with different keywords, it also supports different libraries, being the most used Selenium for web testing. For the project, Robot Framework was used as it is a framework that is widely utilized in all department initiatives.

Listing 5.4 shows a test case example in Robot Framework.

```
1     Login User with Password
2     Connect to Server
3     Login User            ironman    1234567890
4     Verify Valid Login    Tony Stark
5     [Teardown]    Close Server Connection
```

Listing 5.4: Robot Framework test case example [86].

The developed template in Robot Framework uses the centre coordinates of the detected icons since, besides detecting the object and drawing a bounding box around it, YOLOv5 has the option of generating a text file with all the detected labels in each image as well as the centre coordinates, width and height of the bounding box. An example of an output file can be observed in Listing 5.5.

```
1     0 0.35463 0.0552083 0.0740741 0.0354167
2     1 0.227315 0.0513021 0.0675926 0.0317708
```

Listing 5.5: Example of output file.

Where:

- First column - represents the detected label: 0 - *home*, 1- *connectivity*;
- Second and third columns - the `x` and `y` coordinates of the centre of the icon;
- Fourth and fifth columns - width and height of the bounding box;

At this point, one important conversion needed to be made: as the network was trained with the input images resized from 1920x1080 to 384x640 and the detection also was made with the same image resolution, there was the need to convert the output coordinates to match the original image.

For that, a Python script (Listing 5.6) that goes through all the text files generated on each detection and does the conversion to the correct coordinates, was developed.

```
    for file_name in os.listdir(folder_path):
        with open(os.path.join(folder_path, file_name), 'r') as f:
            for line in f.readlines():
                # split the line into columns
                columns = line.split()
                # extract the second and third column
                old_x = columns[1]
                old_y = columns[2]
                new_x = int(float(old_x) * 1920)
                new_y = int(float(old_y) * 1080)
                # add the columns to the result list
                result.append((old_x, old_y))
                new_results.append((new_x,new_y))
```

Listing 5.6: Extract of the script for the coordinates' conversion.

Then, all the converted coordinates are stored in a text file that will be used in the test cases. The developed template can be seen in Listing 5.7.

```
My first test case

    ${file_content}=    Get File    ${CURDIR}/output.txt
    @{lines}=    Split String    ${file_content}    \n
    ${line_count}=    Get Length    ${lines}
    ${lines_to_process}=    Get Slice From List    ${lines}    0
           -1    # exclude last line

    FOR    ${line}    IN    @{lines_to_process}
        ${columns}=    Split String    ${line}    separator=,
        ${x}=    Get From List    ${columns}    0
        ${y}=    Get From List    ${columns}    1
        Tap Coordinates    ${x}   ${y}
        Sleep    5
        Take Screenshots
        Verify if Images are Equal

    END
```

Listing 5.7: Robot Framework developed template.

Here the test case begins by reading the text file containing all the coordinates of the detected icons. The file is then split into lines, where each line corresponds to the coordinates of one icon. To ensure accurate execution, the last line (representing a blank space) is excluded.

The template uses a loop to iterate through each line of the file. Within the loop, each line is further divided into columns. The first column holds the `x` coordinate,

and the second column holds the `y` coordinate of the centre of the detected icon. These coordinates are then used as arguments for the `Tap Coordinates` keyword, which simulates tapping on the given coordinates on the Android device.

After tapping, a screenshot is taken, and the template proceeds to verify if the captured image is equal to the images taken during the manual execution. To perform this image comparison, the template uses `opencv` and applies a template matching approach with a defined threshold, set to 95% during testing. This threshold was chosen to accommodate minor variations in the infotainment display, such as changes in the current timing shown in the images.

For this template, four Python functions were developed capable of connecting the Android device, capturing screenshots, comparing images, and tapping coordinates. Corresponding Robot Framework keywords were created for each function.

In this particular step of the project, the goal was not to test and verify the functionalities of the Android Automotive system, but to see if the template was capable of executing the same test that had been manually done. So, even though the developed template worked, some big changes needed to be implemented, such as: using the first detected label to define the type of the test (so that more functionalities of the DL network could be used), organizing manual test images into folders according to the respective features being tested and ensuring that duplicate manual executions resulted in only one automated test creation.

## 5.2 Second Phase

By achieving good results in the first phase, it was possible to continue with the work and consequently do the next step: training four different networks with the entire dataset and comparing the results to understand which one was more suitable for the project.

At this stage, the dataset consisted of 14728 images, comprising both original and masked versions of 25 icons. The dataset was created by taking screenshots of the infotainment system, applying a mask to the images, using data augmentation and tile techniques as the final goal was the detection of small objects.

The same dataset was used for training all the networks, with the same split: 70% training (10304 images), 15% validation (2208 images) and 15% test (2216 images). This division was made taking into consideration the need to have sufficient training data, therefore the majority of images being used for training and the equal ratio between test and validation was done so that the model had enough unseen data to test and validate the fine-tuning done in the hyperparameters.

Moreover, all the presented confusion matrices for the test dataset shown in this subchapter were obtained using an IoU threshold of 0.6 for detection, meaning that

for each correct prediction, at least 60% of the predicted bounding box overlaps with the ground-truth bounding box.

Finally, the same setup was used to train all the models, that were not developed from scratch: Google Colab PRO which in comparison to the free version offers a stronger GPU (A100).

### 5.2.1 YOLOv5s Training

Just like previously, YOLOv5s was used [87]. However, at this stage, some hyperparameters such as the learning rate, decay, backbone and output layers were changed.

Before starting fine-tuning the network it was necessary to understand the behaviour of the network with the current dataset, so first some training was done only changing the number of epochs (30, 40, 50 and 60) with a batch of 32. The network behaved well during training for all, only with slight changes in the output metrics. However, during the test, it was possible to conclude that the network had its best results for 50 epochs.

At this point, each training of the network took between 4 to 5 hours, so the change of the batch size was only tested for the 50 epochs, achieving the conclusion that 50 epochs and a batch size of 32 were the best options for the used dataset. Therefore, all other tests were done using these values.

The initial change made to the hyperparameters was reducing the final learning rate from 0.01 to 0.0001, to ensure that the last layers of the network could effectively learn the specifics of the target task. To support this training, a weight decay of 0.0005 was used, along with 3 warmup epochs and no weights were initialized.

The obtained results from the described training can be seen in Figure 5.9.

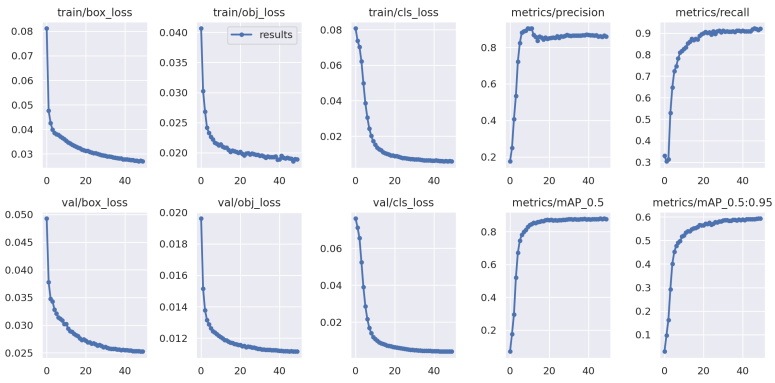



Figure 5.9: First training results.

When analysing the values of precision and recall it is possible to conclude that even during training, out of all positive predictions made by the model approximately

85% were correct predictions and that out of all the ground-truth bounding boxes, the model was able to correctly predict around 92%.

Also, by looking into the second mAP value of almost 60%, it is possible to verify that even in training, the model performed reasonably well, however improvements needed to be made, which is possible to confirm in Figure 5.10 that represents the confusion matrix for the test dataset with the obtained train weights.

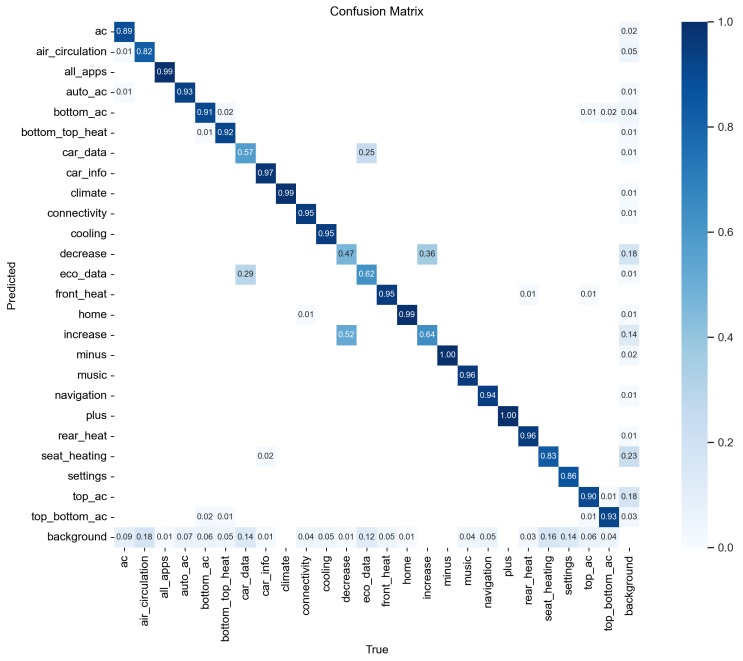



Figure 5.10: Confusion matrix for the test dataset.

By observing the previous figure it is possible to understand that there are some false predictions, mostly in: *car_data*, *decrease*, *eco_data* and *increase* icons.

In order to try to improve the obtained results, the network was trained using the configurations of YOLOv5s6 instead of YOLOv5s and changing the initial learning rate from 0.01 to 0.001, maintaining the other hyperparameters described previously. The difference between them lies in the number of anchor boxes and backbone connections, meaning that:

- YOLOv5s - uses three anchor boxes per prediction head and the head is connected to three feature maps (P3, P4 and P5);
- YOLOv5s6 - uses four anchors per prediction head and the head is connected to four feature maps (P3, P4, P5 and P6).

The difference in the number of anchor boxes and feature map connections can affect the model's ability to detect objects of various sizes and aspect ratios, which means that the second configuration is able to detect more objects as it detects more sizes.

In Figure 5.11 it is possible to see the obtained training results.

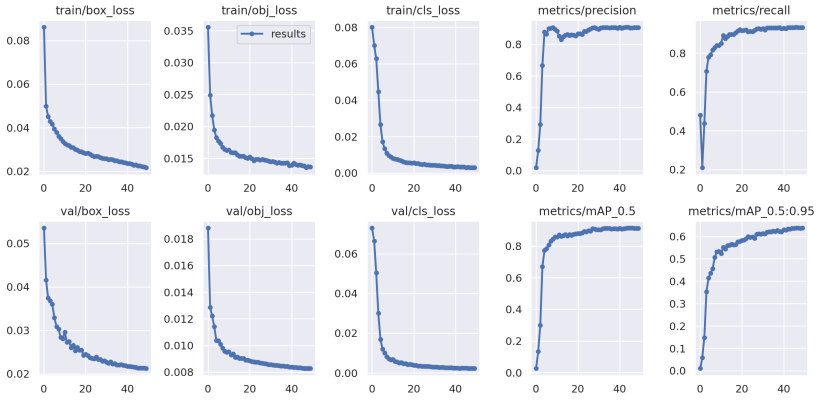



Figure 5.11: Training results of the network using YOLOv5s6 configurations.

By observing the obtained results it is possible to conclude that there is an improvement in the precision and recall of the model, where now almost 91% of the positive predictions made by the system are correct and around 93% of all positive instances of the dataset were found by the model. Also, in terms of mAP with an IoU threshold from 0.5 to 0.95, there was an increase, leading to the conclusion that the model's performance improved. This can be confirmed when looking at the confusion matrix obtained during testing, represented in Figure 5.12.

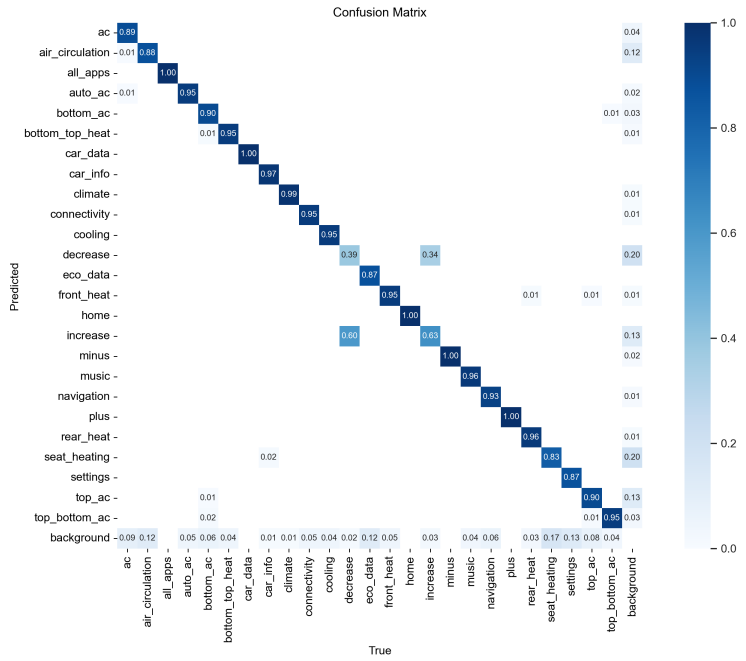



Figure 5.12: Confusion matrix for the test dataset.

With the new confusion matrix, it can be seen that the model still makes some false predictions, however now mostly for two icons: *increase* and *decrease*. Also, the main diagonal values mainly increased, meaning that the number of correct predictions was even higher than before.

To further improve the model's performance, a subsequent training stage was made by reusing the weights obtained from the previous training. In this stage, the first 8 layers of the YOLOv5s6 backbone were frozen, whilst the configuration and hyperparameters used were the same as in the train previously described. This decision was made since typically the first layers capture low-level features such as edges, textures and simple patterns and, by using the best weights from the previous training, the model could benefit from the knowledge of these generic features while focusing more on learning task-specific features in the later layers.

YOLOv5s6 backbone consists of 10 layers, and freezing the first 8 layers allowed the model to prioritize learning higher-level features responsible for generating accurate detections (layer 9) and handling objects of various sizes through the Spatial Pyramid Pooling (SPP) layer (layer 10).

The training results are possible to see in the following output graphics represented by Figure 5.13.

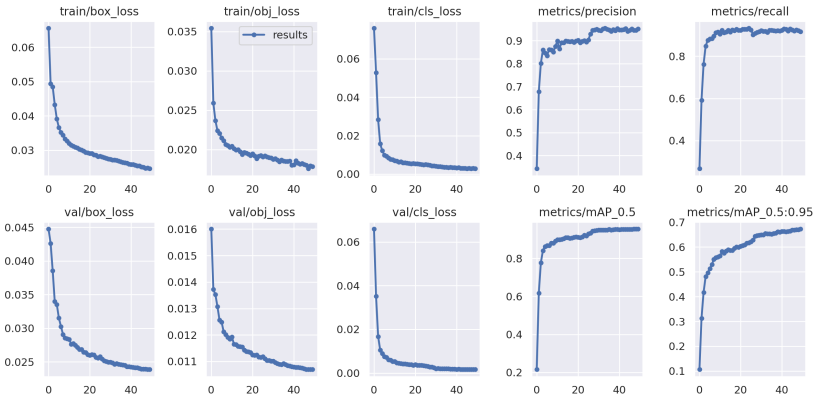



Figure 5.13: Training results of the network with 8 backbone layers frozen.

After this training was completed, it was possible to see a good improvement both on the mAP value and precision, meaning that the model was performing well and demonstrating a good ability to detect objects accurately across different IoU thresholds and doing more than 95% of the predictions correctly. However, there was a slight decrease in the recall value, suggesting a decrease in the positive instances found by the model.

This refinement of the metrics during training can be sustained by Figure 5.14 which shows the confusion matrix obtained for the test dataset.

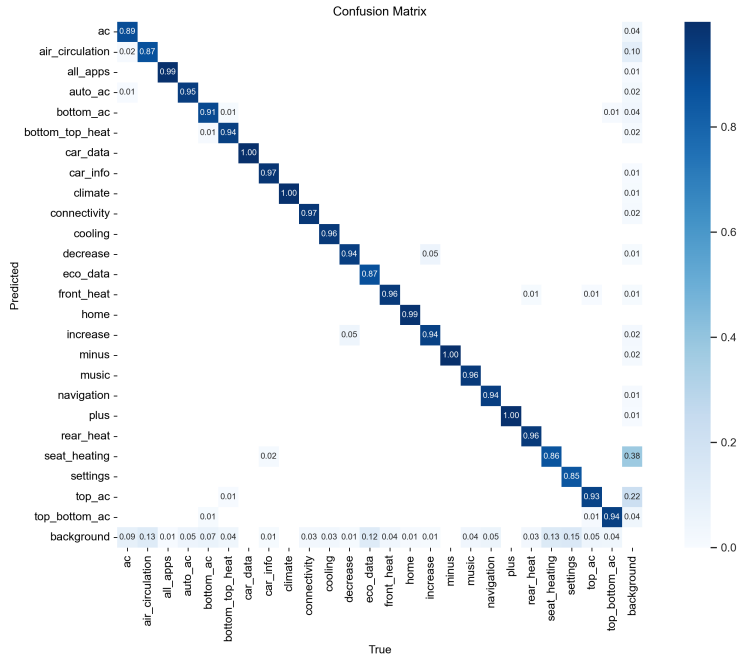



Figure 5.14: Confusion matrix for the test dataset with 8 frozen backbone layers.

It is possible to verify that the false predictions significantly decreased when comparing with the two previous experiences and the network is correctly detecting the objects at least 85% of the time.

For a final experience, to understand if it would be possible to upgrade the results, 6 layers of the backbone were frozen. The decision to train 4 layers of the backbone instead of only 2 was based on the fact that layers number 7 and 8 are convolutional layers continuing with feature extraction and so, this experiment was done to understand the impact on the model in terms of accuracy in detection. The final results of the training can be seen in Figure 5.15.

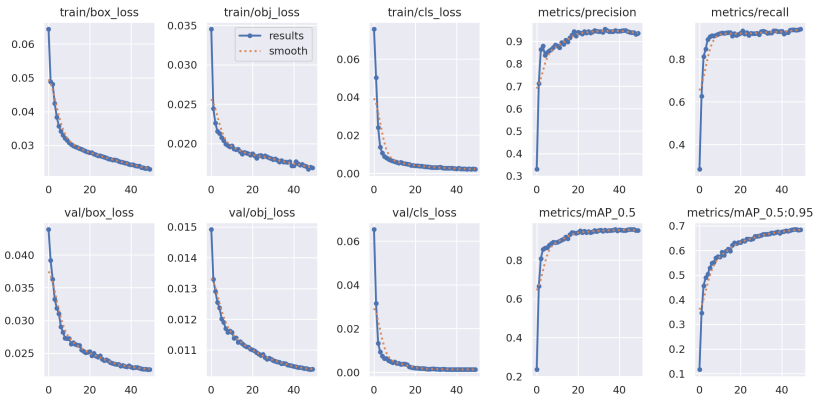



Figure 5.15: Training results of the network with 6 backbone layers frozen.

It is then possible to see that the value of mAP and recall slightly increased whilst the precision value decreased when comparing with the previous test. With this, and as possible to confirm in the confusion matrix represented by Figure 5.16, it is possible to conclude that the 2 last layers of the backbone are the ones that have the biggest impact while training the network.

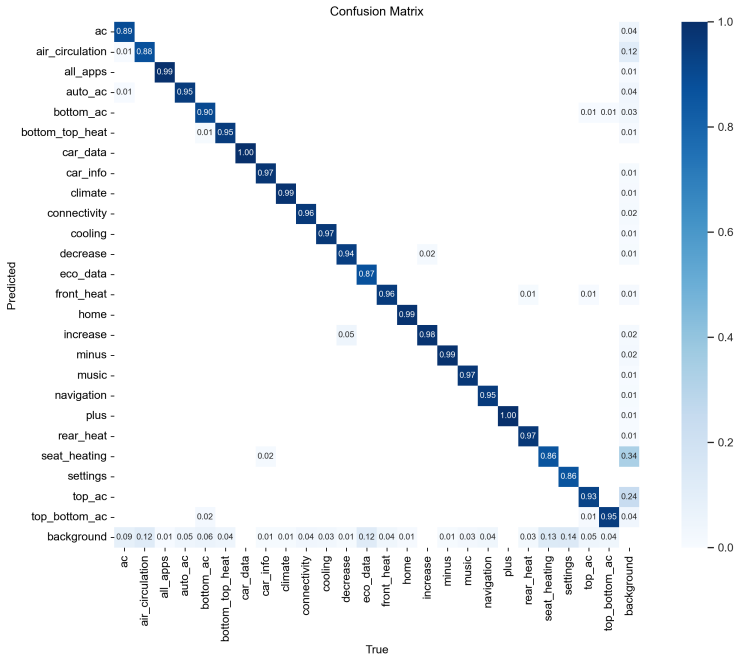



Figure 5.16: Confusion matrix for the test dataset with 6 frozen backbone layers.

To summarize, several tests were done with changes to the hyperparameters of the network: changing the initial and final learning rate, configuration of the network and backbone layers. With the conducted tests, where each took an average of 4 hours and 30 minutes to complete, it was possible to verify that the network performed well for the used dataset, achieving good results of precision and recall.

In addition, YOLOv5's inference time, meaning the amount of time the model takes to make predictions on unseen data, is one of its biggest advantages as it takes 1.3 ms per image with 3 channels and a resolution of 640x640 to detect the object and make the prediction, making it a good option to use in a real-time detection system.

### 5.2.2 SSD Training

Next, the SSD network was trained. As previously mentioned, the network was originally developed with VGG-16 as its backbone however, due to the common resources limitation, MobileNet [88] has become more and more used to replace VGG-16 as SSD's backbone. Even though the accuracy results are not as good as

the ones obtained by the authors of the original network, its efficiency makes it a good choice for practical applications. So, for the purpose of the project, MobileNet v2 was used as the model's backbone and Tensorflow as its framework [89].

The used model had the particularity that the train was done in steps, meaning that the model only processes one batch during each step. Four tests were conducted where the model was trained during 40 000 steps, with 2000 warmup steps, either with a batch of 16 or 32. For a fair comparison, the changed hyperparameters were the same as YOLOv5.

So, firstly a batch of 32 was used, with existing weights initialized and the initial learning rate was changed from 0.01 to 0.001 and the final from 0.001 to 0.0001. After 40 000 steps the loss graphic, which indicates how well the network is performing during the training process, as it represents a numeric value that quantifies the discrepancy between the predicted output of the neural network and the ground truth labels for the training data, can be seen in Figure 5.17.

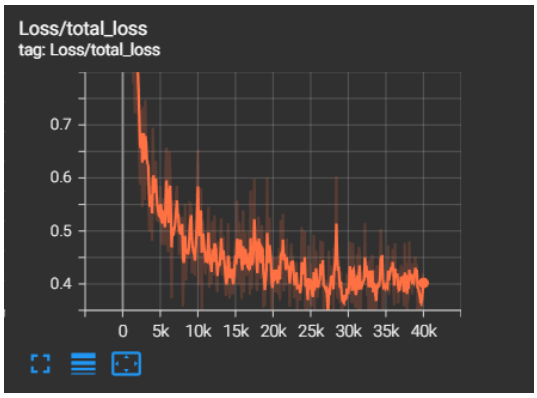



Figure 5.17: Total loss for the first train of SSD network.

It is possible to observe that the loss value had some fluctuations during the training, however, it achieved a significantly stable value in the last 5000 steps. Based on the testing results of the obtained mAP for different thresholds, the model had a mAP of 53.21%, indicating that it performed reasonably and that the loss value was a relatively high value for the used dataset.

The average precision of the model on the test dataset was 25.7% and the average recall was 60.7%, indicating that almost 61% of all ground-truth bounding boxes were correctly predicted, however, the majority of positive predictions were false ones.

As an example, Figure 5.18 shows the obtained values for precision and recall per class for an IoU threshold of 60%. All used data to calculate these values can be seen in Appendix A.

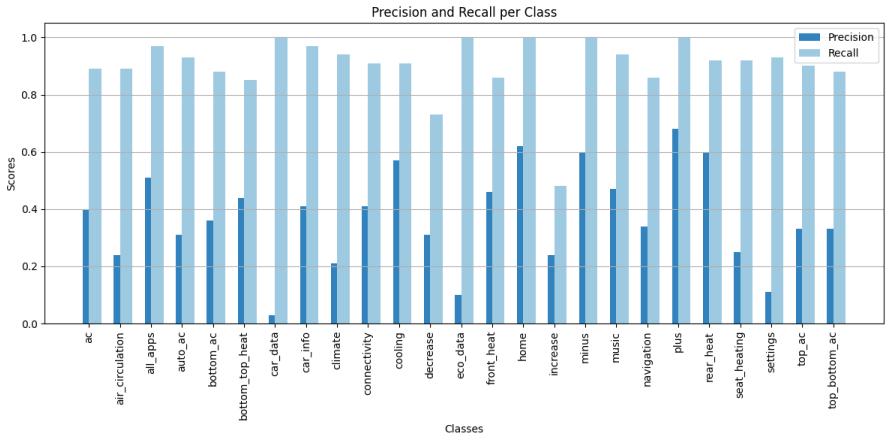
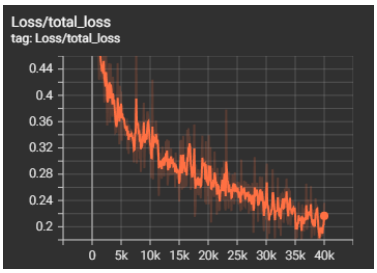


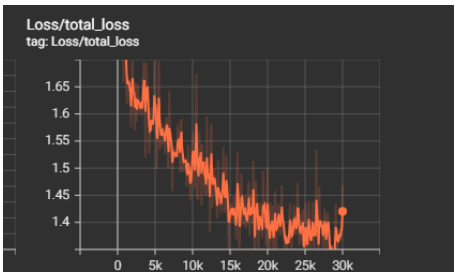

Figure 5.18: Precision and recall values per class.

The first results showed that there was a need to improve the network, so for the following tests, first, the batch size was decreased to 16 so that the network could analyse fewer images per step and then the backbone was frozen.

This first change resulted in a loss of 0.22 by the end of the 40 000 steps, as visible in Figure 5.19a and an increase of the mAP to 63.54%, however, the second change resulted in an increase of the loss - Figure 5.19b - and the training was automatically stopped by the network as the loss started to grow.




(a) Total loss for the second train of SSD network.




(b) Total loss for the third train of SSD network.

Figure 5.19: Total loss in SSD training.

As a final test, the 40 000 steps ran with a batch of 32 and all hyperparameters default. The loss during training decreased, as visible in Figure 5.20.

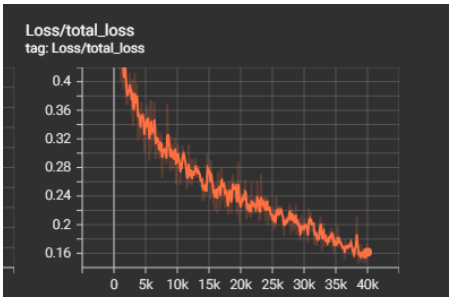



Figure 5.20: Total loss for the last train of SSD network.

With this last train, the lost value achieved its lowest (0.16), but in comparison to the second one, the mAP decreased to 59.7%. All relevant outputs for the 3 successful trainings can be seen in Table 5.4. The values of precision and recall presented in the table were calculated as an average of the precision and recall achieved for each class from an IoU threshold of 50% to 95%. Also, the values of precision and recall for each class and each IoU were determined based on the ground-truth objects, detected objects, true positives and false positives (per class).

Table 5.4: Testing results of all trains.

| **Train Number** | **mAP** | **Precision** | **Recall** |
|---|---|---|---|
| 1 | 53.21% | 25.7% | 60.7% |
| 2 | 63.54% | 51.8% | 65.8% |
| 4 | 59.7% | 53.8% | 64% |

With all the results it is possible to conclude that the changes done in the model were not effective as the last train, which ran with all default parameters presented the best loss value as well as precision. Each training of the network took from 3 hours to almost 7 hours, making this network slow to train, at least with the available resources, to achieve good results. As possible to see in the last table, only average results for precision and recall were achieved. However, one important aspect of SSD is the fact that when correctly detecting, the confidence is high, as possible to verify in Figure 5.21, and that as a one-stage detector presents a good inference time of about 22 ms per image.

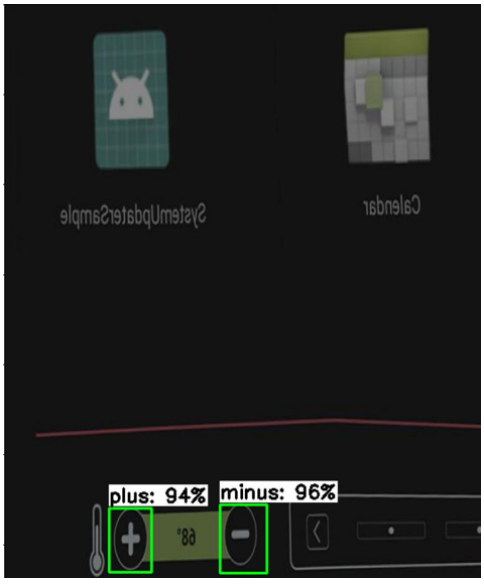



Figure 5.21: Detection results for the second training.

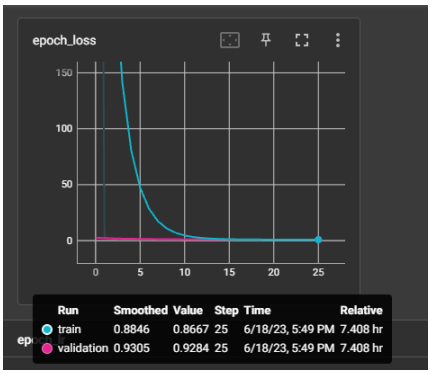
### 5.2.3 Retinanet Training

The final one-stage detector to be tested was Retinanet, however, for this model, the Colab Notebook was developed from scratch, based on the information provided by the author of the model to be used in [90]. The model was created using Keras as a framework and ResNet50 as its backbone.

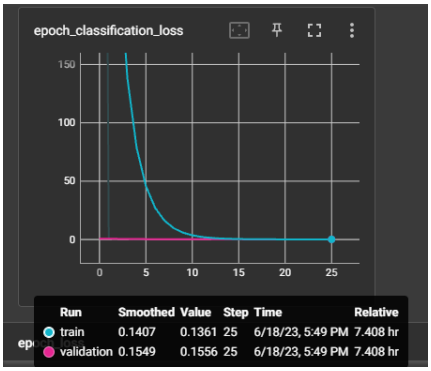
In order to try to do an equivalent comparison between all models, the first attempt to train the network was with a batch of 32 but due to the lack of resources, the model was not able to start its training due to memory allocation issues and the network initialization was only possible when using a batch of 8. With this, it was already possible to understand that the used model required a high usage of resources.

To start, a batch of 8 was used and the model was trained with no initialized weights during 25 epochs with 400 steps per epoch, meaning that in each epoch 400 batches of data would be processed, also, the learning rate was changed from 0.1 to 0.01.

By the end of the 25 epochs the model's loss was 0.8667 on the training dataset and 0.9284. As can be seen in Figure 5.22a, practically there was not a decrease in the loss for the validation dataset and on the last 10 epochs the loss for the training dataset was also constant, which could indicate that the learning rate was too low. As expected, this same convergence of values happened on the classification loss, as depicted in Figure 5.22b.




(a) Total loss for the first train of Retinanet network.




(b) Classification loss for the first train of Retinanet network.

Figure 5.22: Loss in the first Retinanet training.

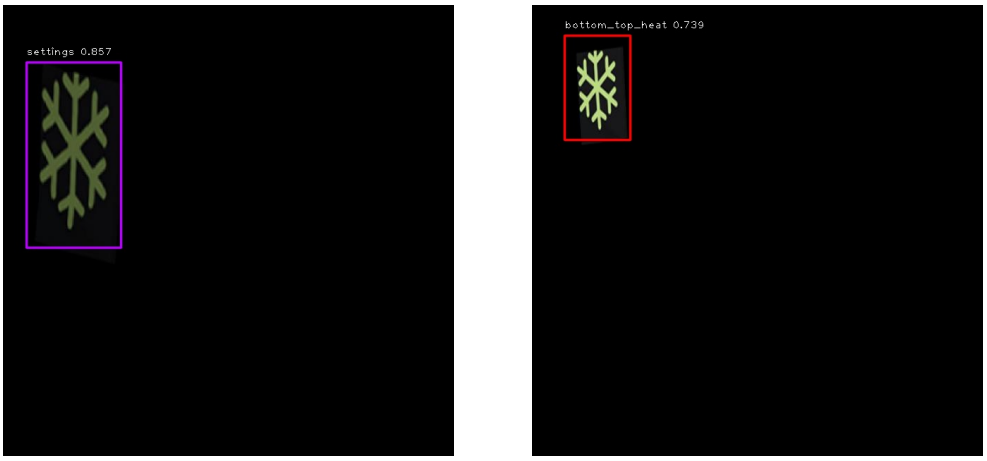
The mAP was constantly increasing, and by the end of the 25 epochs, it achieved the value of 0.8505, which was a high value, even when compared with the previous networks, indicating that the model's performance was quite good.

However, when testing the network, even though there were some good results, as represented in Figure 5.23a, there was also a clear confusion of the model, as seen in Figure 5.23b, where it wrongly detected the objects with high confidence.


(a) Correct detection during testing. (b) Wrong detection in testing.

Figure 5.23: Detection using Retinanet.

This confusion was even more clear when testing with the masked images, where in the majority of the cases the wrong detection was made and in others when giving as an input the same icon to the network, the output would be two completely different labels, as visible in Figures 5.24a and 5.24b, and with an high confidence.




(a) Wrong detection of icon in masked image. (b) Different detection of icon in masked image.

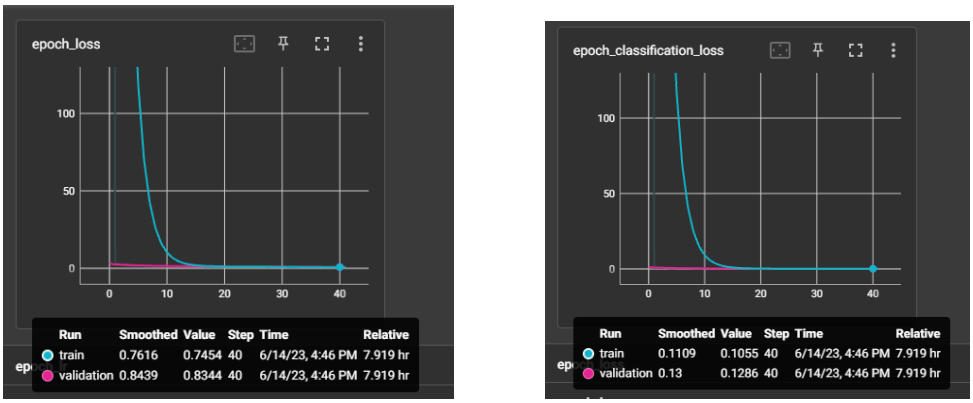
Figure 5.24: Wrong detection of the cooling icon.

So, to try and obtain better results some changes were made:

1. Weights were initialized: ResNet50 trained with the COCO dataset;
2. Learning rate was changed to the default value during the first epochs and then started to gradually decrease until 0.001 for the final epochs;
3. The model was trained during more epochs but with less step value.

The weights were initialized so that the network was not as dependent on high resources as in the previous train and also to have less data dependency. The third

change was made so that the model could process fewer data per epoch, but could improve its accuracy. So, the second training of the model was done with a batch of 8 during 40 epochs and with a step of 200. The obtained results for the total loss and classification loss can be seen in Figures 5.25a and 5.25b, respectively.

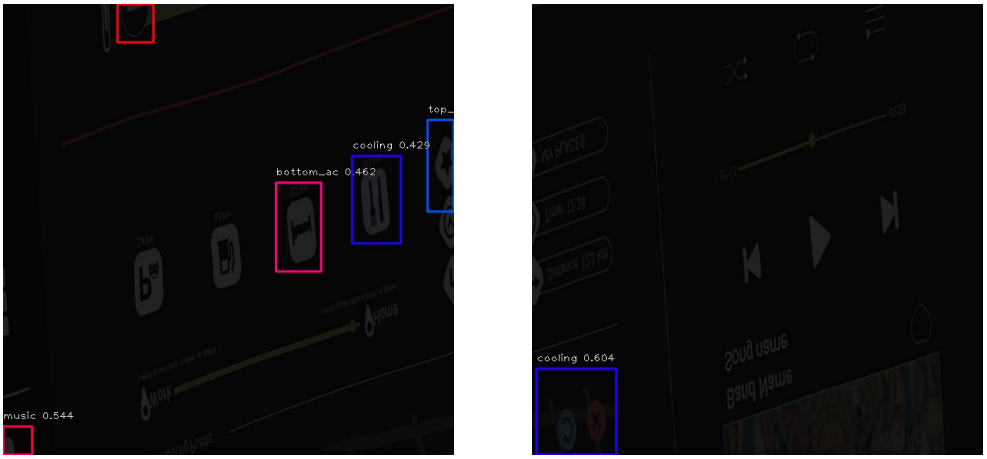



(a) Total loss for the second train of Retinanet network.

(b) Classification loss for the second train of Retinanet network.

Figure 5.25: Results of second training.

It is possible to confirm that there is a decrease in the loss values when comparing with the first results, as the model is now capable of achieving a total loss value of 0.7454 on train and 0.8344 on validation, and values of classification loss of 0.1055 and 0.1286 on train and validation, respectively. Additionally, as expected based on the loss results, the mAP increased to 0.8657. Despite that, the train values, after some time remained idle and the validation values practically did not fluctuate, just like previously. This convergence could again be sustained by the testing results, where faulty detections with high confidence were being made. Two examples are represented in Figures 5.26a and 5.26b.




(a) Example of wrong detection results.

(b) Example of wrong detection.

Figure 5.26: Detections during the second training.

As each train of the model took more than 7 hours, a last attempt was made by freezing the backbone's network, so that no feature extraction was again made. However, after 8 epochs, the model's loss was again starting to become unchanged and, because of that, the training was interrupted.

With this network, there were clear signs of overfitting as good training results were being obtained, especially the mAP, but then when dealing with unseen data the model was not capable of performing well. With this, it was possible to conclude that the model was not suitable for the dataset in use, at least with the changes on the hyperparameters done for each training.

### 5.2.4 Faster R-CNN Training

To better understand the difference between two-stage and one-stage detectors and to be able to do clear comparisons in what would work better for the system, the dataset was lastly trained with Faster R-CNN implemented with ResNet50 as the backbone. The used model [91] was implemented in Detectron2, a library developed by Facebook that provides state-of-the-art detection and segmentation algorithms.

For the 3 first trainings, the network ran for 1000, 2000 and 3000 steps with all the default values, which included a learning rate of 0.002, initialized weights of ResNet50 trained on the COCO dataset and a batch of 2. The model finalized the 3000 steps in 15 minutes and the maximum obtained classification accuracy was 95%, however, the maximum AP was approximately 46%, indicating that the model could be improved.

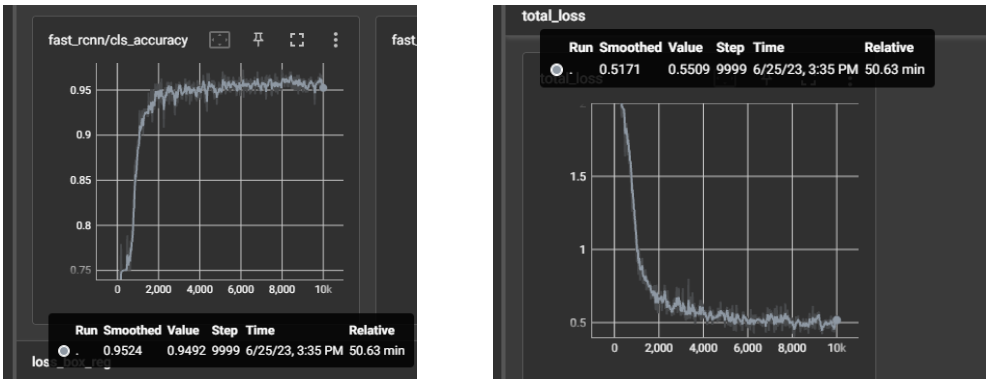
One example of the network's detection with high accuracy with only 3000 steps can be seen in Figure 5.27a. On the other hand, Figure 5.27b shows the relatively low precision of the network with different detections for the same icon.


(a) Example of high accuracy detection.

(b) Example low precision detection.

Figure 5.27: Predictions after 3000 steps.

Next, the number of steps was increased to 10 000, where the AP increased to 54% and the accuracy remained around 95%, as visible in Figure 5.28a. At this point, the total loss, visible in Figure 5.28b, was at 0.5 and, when compared to previous results, this is a high value.




(a) Accuracy results.

(b) Total loss results.

Figure 5.28: Results after 10 000 steps.

As the results were improving, the next training of the model was done with the same parameters as previously, but during 20 000 steps, however, there was no significant difference in the results. Therefore, with the 10 000 steps, changes on the hyperparameters were done, the batch size was increased from 2 to 16 and the initial learning rate was set to 0.001 and the final to 0.002. With this, the total loss, Figure 5.29, decreased to around 0.44 and the AP increased to 58%, however there was no change in the accuracy.

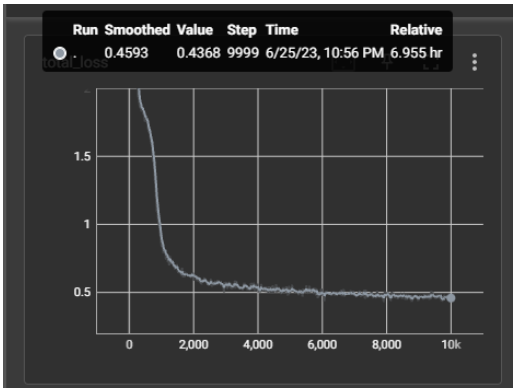



Figure 5.29: Total loss for the train with a batch of 16 during 10 000 steps.

As can be seen in the previous figure, the training time increased to almost 7 hours and there was not a big change in the results. So, as a last attempt, the backbone of the network was frozen, but with this last change, both accuracy and AP decreased to 82% and 36%, respectively.

With the use of this model, it was possible to conclude that even though it presented reasonable results, it needed to be trained for much more time to achieve high precision, which would be time-consuming. Also, as expected, the inference time was high (86.96 ms per image) when compared with the one-stage detectors.

### 5.2.5 Results Comparison

After training the same dataset on four models it was possible to select the best one, suitable for the project.

As the final goal of the project is to build a real-time detection system that can be easily re-trained in case new images need to be added to the dataset, Faster R-CNN is not a good choice as first, a lot of time needs to be spent to achieve similar results to one-stage detector, but mainly to its inference time, that is almost 86 times higher than the one in YOLOv5.

Then, when comparing the one-stage detectors, Retinanet was the one with the poorest performance, as it showed signs of overfitting. Finally, when comparing YOLOv5s with SSD, it is possible to determine that YOLOv5s presented better results due to several reasons:

- Training time - YOLOv5s average training time was 4 hours and 30 minutes, while SSD training took up to almost 7 hours;
- Precision and recall results - YOLOv5s had values of more than 90% for both metrics, whilst SSD had a maximum of 53.8% for precision and 64% for recall;
- mAP - it reached 67% with YOLOv5s and 65.8% with SSD;
- Inference time - 1.3 ms per image in YOLOv5s against 22 ms per image in SSD;
- Available community and information - The YOLO network is in constant development with an active community, while SSD authors only released one article upon its creation;
- Output metrics - YOLOv5s outputs, by default, all the needed metrics to correctly analyse the model's performance, while SSD outputs only loss graphics;
- Transfer learning - this technique can be easily used in YOLOv5 as the network was built so that the users could reuse what was done by the authors, but also make their changes. The same does not happen in SSD since the model was created to use everything by default and change only the used dataset, making it difficult to change the hyperparameters.

Due to all the mentioned points and also obtained training and testing results, for the elaboration of the project, the weights acquired during the training of YOLOv5s6 with 8 backbone layers frozen will be used to execute the object detection.

## 5.3 Third Phase

The last phase involved integrating the model trained in the previous step to perform detection and enhancing the existing work to add more features, making it more accessible and user-friendly.

The execution of the manual test remained the same, meaning:

1. The stream of the device starts;
2. After each click a screenshot is taken and stored;
3. A mask is applied to highlight the touched icon;
4. The masked image serves as an input of the DL model.

After the detection was made, the following features were included:

- The output file given by the network with the labels and detected coordinates is divided into two different files: one with the coordinates and one with the labels;
- A window will pop up so that the user can define certain aspects of the test, such as name, description and associated tag;
- The first label of the test will be responsible for giving the tag and test name suggestion (that can be changed by the user);
- If the same first label is detected, a different test name is suggested, but if the user selects an existing name all content is replaced;
- All important data regarding the manual test, such as: screenshots path, coordinates file, tag name, test name and description is stored in a `JSON` file that serves as a database;
- After the manual test is finished a "parent" folder with the name of the tag will be created and inside of it the coordinates, manual test screenshots, and the executed steps will be stored.

An example of how the data is stored in the `JSON` file can be seen in Listing 5.8.

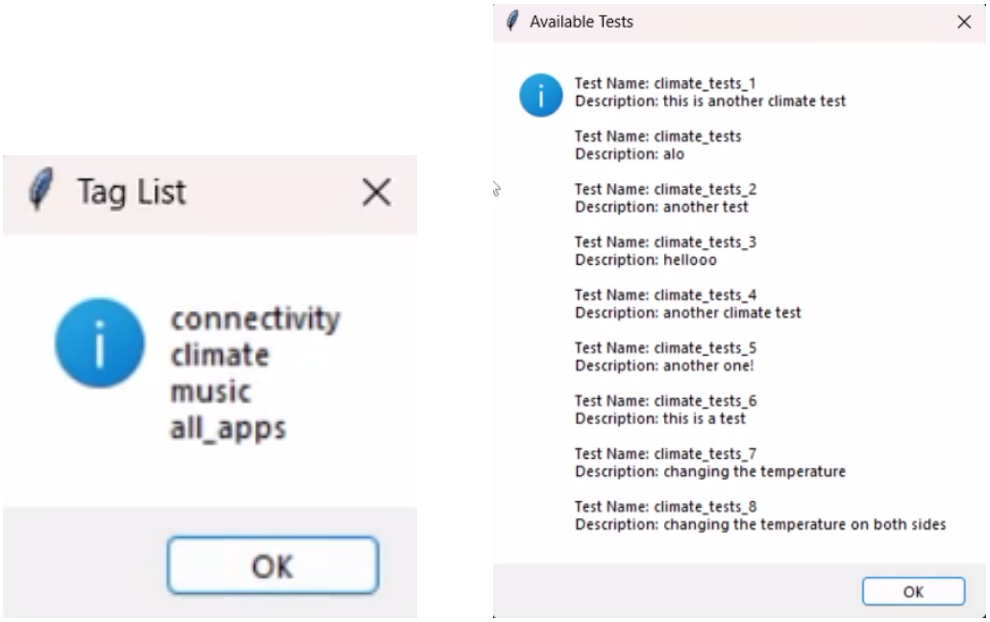
```
1         "tag_name": "navigation",
2         "test_name": "navigation_tests",
3         "coordinates_file": "../Robot\\navigation\\
              navigation_tests\\coordinates.txt",
4         "description": "This is a basic navigation test",
5         "screenshots_path": [
6             "../Robot\\navigation\\navigation_tests\\Manual_Test
                  \\2023-07-25_120758.png",
```

```
7                "../Robot\\navigation\\navigation_tests\\Manual_Test
                     \\2023-07-25_120809.png"]
```

Listing 5.8: Example of stored data in the file.

With the stored data, it is possible to replicate all previously executed manual tests automatically. The final template and keywords can be seen in Appendix B and resulted in an improvement of the work elaborated in the first phase of the project. It is important to mention that the keywords were created based on Python functions that were developed for this purpose.

To be able to select which automated tests to execute, a simple interface was created that allows the user to see information such as the available tags - Figure 5.30a, available tests for each tag and the description of each test - Figure 5.30b. The interface also serves the purpose of allowing the user to select which tests to run.




(a) Tag list. (b) Tests information.

Figure 5.30: Developed interface.

To execute an automated test, the user will need to do the following steps:

1. Select which tag or tags he wants to run - the tags work as a split of categories of the tests (climate, navigation, connectivity...);
2. Once the tags are selected, choose which test or tests to run - all available tests for the selected tags will be displayed;
3. Start the execution.

After the execution is finished, the screenshots taken during each test will be saved in the respective folder and each report will be saved in a folder with the corresponding date and time of the test's execution, meaning that if the same test is executed four times there will be four different folders with the taken screenshots and four different reports. The final folder structure can be seen in Figure 5.31.

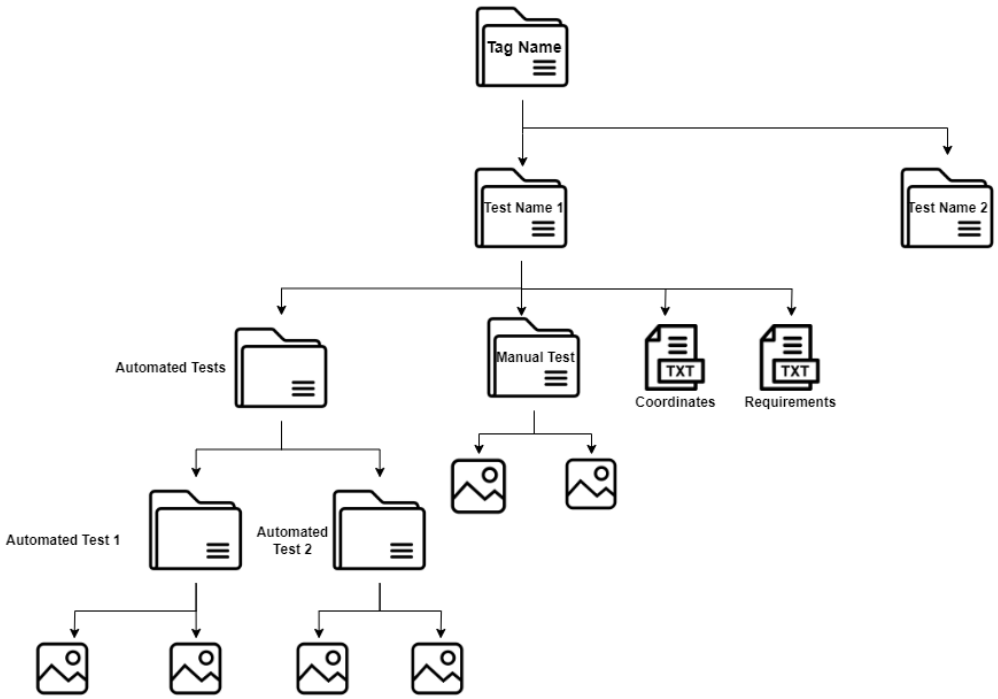



Figure 5.31: Final folder structure.

The structure of the reports' organization can be seen in Figure 5.32.

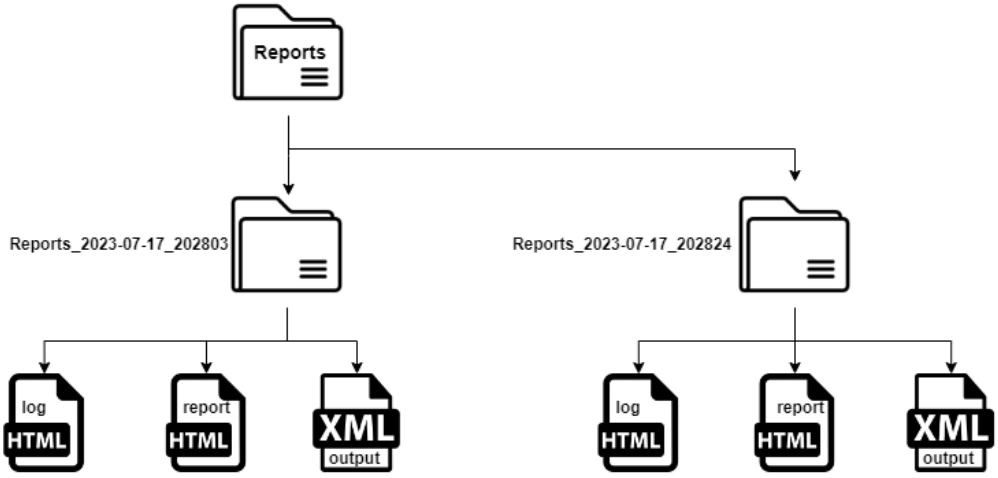



Figure 5.32: Final folder structure of Robot Framework reports.

Finally, to ease the handling of the developed tool, two executables were created,

one to start the execution of the manual tests, including the detection of icons using the pre-trained model, and one to start the execution of the automated test cases.

The division of the project into three distinct phases was important for its development, since in the first phase it was possible to understand that the project was capable of working, in the second phase it was possible to do several tests with several models in order to comprehend which one was more suitable for the application and lastly in the third phase, there was room for improvements so that the project becomes as user-friendly as possible.

# Chapter 6

# Conclusions

The integration of AI within the automotive sector has progressively surged, owing to its user convenience and safety enhancements. Consequently, there has been a growing interest in employing AI techniques, like DL, for software testing within the automotive industry. Despite not being widely prevalent, companies such as Capgemini Engineering are recognizing the potential of investing in such approaches.

With the developed work, all proposed objectives were achieved and it is possible to conclude that the use of DL in automated tests for the automotive industry has potential and will be a great asset for the future.

The primary aim was to transition from manual testing on an Android Automotive system to automated testing, without needing the creation of distinct Robot Framework test cases and recurring to the use of a DL model. For that, there was the need to study and comprehend several networks and, after all the research, YOLOv5 was used as it presented better results for the project itself. Besides all the already presented reasons, the fact that the used model was the one developed by the authors of the network could had some impact on the obtained results.

During the study and implementation, there were several drawbacks, especially in the beginning, as it was necessary to understand how Android works and for that, an attempt to deploy an Android Automotive image into a Beagle Board was made. However, due to the lack of knowledge inside the company and as it was not the main goal of the project, an already existing setup was adopted instead. During the DL, study there were also some challenges: the annotation process, which was very

time-consuming due to the size of the dataset, the difficulty of the subject itself and the low expertise inside the company.

In summary, all the encountered challenges were overcome and in the end, a working proof of concept that included the use of the DL algorithm with the minimum interaction and knowledge from the user was achieved. A test for each icon was made as well as several tests with different icon combinations, giving a total of 30 manual tests transformed into automated ones, where the developed tool was able to correctly replicate all of them. Also, some of the automated tests were executed more than once and the output was always the same, meaning that the tool had a consistent behaviour.

## 6.1 Future Work

Even though the proposed objectives were achieved and the tool was tested in different Android Automotive infotainments, the project remains as a proof of concept and so some aspects need to be improved so that it can be used in future projects.

To begin with, the project still needs to be implemented in the Jakku platform so that this new approach replaces the old one which still requires the development of different Robot Framework test cases, the use of Jenkins and Jira. Also, some improvements could be made, such as the suggestion of test executions, meaning that based on the last executed tests the tool could be capable of suggesting tests to be executed that were not executed in some time. This could be helpful to guarantee that all features were always tested.

Besides this, the adaptation of the reports' comparison that already existed in the Jakku platform would be useful, since with that the testers could verify if new bugs were either fixed or discovered in between executions.

Another improvement would be the increase of the dataset to understand if the pre-trained model could be again trained with a new dataset and if it was able to achieve good results quickly.

Finally, once the tool is totally developed and working in Android projects, one last upgrade would be the adaptation of the tool to work in different infotainments that are not Android-based.

As mentioned, this is an area that will improve the quality of software testing in the Automotive industry, however, it still needs a lot of work and study inside the company to be able to do the maintenance of this tool so that it is always up-to-date and ready to use in any kind of Automotive project.

# Bibliography


[1] A. Tierno, M. M. Santos, B. A. Arruda, and J. N. H. da Rosa, "Open issues for the automotive software testing," in *2016 12th IEEE International Conference on Industry Applications (INDUSCON)*, pp. 1–8, 2016. [Cited on page 2]

[2] C. Engineering, "Capgemini engineering." Available at `https://capgemini-engineering.com/pt/pt-pt//`, 2022. (Last accessed in 20/11/2022). [Cited on page 2]

[3] M. Polo, P. Reales, M. Piattini, and C. Ebert, "Test automation," *IEEE Software*, vol. 30, no. 1, pp. 84–89, 2013. [Cited on page 7]

[4] M. A. Umar and Z. Chen, "A study of automated software testing: Automation tools and frameworks," vol. 8, pp. 217–225, 12 2019. [Cited on pages 7, 9, and 10]

[5] C. Jones, "Software quality in 2012: A survey of the state of the art," pp. 1–4, 5 2012. [Cited on page 7]

[6] L. Khong, L. Yu Beng, T. Yip, and T. Soofun, "Software development life cycle agile vs traditional approaches," 02 2012. [Cited on page 8]

[7] O. Anurina, "Agile sdlc: Skyrocketing your project with agile principles." Available at `https://mlsdev.com/blog/agile-sdlc`, 2021. (Last accessed in 11/12/2022). [Cited on pages vii and 9]

[8] D. Huizinga and A. Kolawa, eds., *Automated Defect Prevenction: Best Practices in Software Management*. New Jersey: Hoboken, 2007. [Cited on page 9]

[9] EL-PRO-CUS, "What are testing techniques : Types, advantages disadvantages." Available at `https://www.elprocus.com/what-are-testing-techniques-types-advantages-disadvantages/`. (Last accessed in 08/01/2023). [Cited on page 9]

[10] D. M. Rafi, K. R. K. Moses, K. Petersen, and M. V. Mäntylä, "Benefits and limitations of automated software testing: Systematic literature review and practitioner survey," in *2012 7th International Workshop on Automation of Software Test (AST)*, pp. 36–42, 2012. [Cited on page 10]

[11] D. Ganea, R. Bogdan, V. Ancusa, and M. Popa, "A case study of automated testing implementation in the automotive industry," pp. 471–475, 11 2013. [Cited on page 11]

[12] E. Bringmann and A. Krämer, "Model-based testing of automotive systems," in *2008 1st International Conference on Software Testing, Verification, and Validation*, pp. 485–493, 2008. [Cited on pages 11, 12, and 13]

[13] M. Markthaler, S. Kriebel, K. S. Salman, T. Greifenberg, S. Hillemacher, B. Rumpe, C. Schulze, A. Wortmann, P. Orth, and J. Richenhagen, "Improving model-based testing in automotive software engineering," in *2018 IEEE/ACM 40th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP)*, pp. 172–180, 2018. [Cited on pages vii, 11, 14, and 15]

[14] H. Altinger, F. Wotawa, and M. Schurius, "Testing methods used in the automotive industry: Results from a survey," in *Proceedings of the 2014 Workshop on Joining AcadeMiA and Industry Contributions to Test Automation and Model-Based Testing*, JAMAICA 2014, (New York, NY, USA), p. 1–6, Association for Computing Machinery, 2014. [Cited on page 11]

[15] T. Hamilton, "What is model based testing?." Available at `https://www.guru99.com/model-based-testing-tutorial.htmlc`, 2022. (Last accessed in 21/01/2023). [Cited on page 12]

[16] A. C. Team, "Waterfall methodology: A complete guide." Available at `https://business.adobe.com/blog/basics/waterfall`. (Last accessed in 08/06/2023). [Cited on pages vii and 12]

[17] M. S. Team, "What are mil, sil, pil, and hil, and how do they integrate with the model-based design approach?." Available at `https://www.mathworks.com/matlabcentral/answers/440277-what-are-mil-sil-pil-and-hil-and-how-do-they-integrate-with-the-model-based-design-approach`, 2022. (Last accessed in 21/01/2023). [Cited on page 13]

[18] H. Altinger, F. Wotawa, and M. Schurius, "Testing methods used in the automotive industry: Results from a survey," in *Proceedings of the 2014 Workshop on Joining AcadeMiA and Industry Contributions to Test Automation and Model-Based Testing*, JAMAICA 2014, (New York, NY, USA), p. 1–6, Association for Computing Machinery, 2014. [Cited on page 14]

[19] MathWorks, "Polyspace making critical code safe and secure." Available at `https://www.mathworks.com/products/polyspace.html`. (Last accessed in 21/01/2023). [Cited on page 14]

[20] V. I. GmbH, "Testing ecus and networks with canoe." Available at `https://www.vector.com/int/en/products/products-a-z/software/canoe/`. (Last accessed in 21/01/2023). [Cited on page 14]

[21] C. Janiesch, P. Zschech, and K. Heinrich, "Machine learning and deep learning," *Electronic Markets*, vol. 31, no. 3, pp. 685–695, 2021. [Cited on page 15]

[22] L. Alzubaidi, J. Zhang, A. J. Humaidi, A. Al-dujaili, Y. Duan, O. Al-Shamma, J. Santamaría, M. A. Fadhel, M. Al-Amidie, and L. Farhan, "Review of deep learning: concepts, cnn architectures, challenges, applications, future directions," *Journal of Big Data*, vol. 8, 2021. [Cited on pages vii, 16, 17, and 18]

[23] F. He, P. Olia, R. Oskouei, M. Hosseini, Z. Peng, and T. Banirostam, "Applications of deep learning techniques for pedestrian detection in smart environments: A comprehensive study," *Journal of Advanced Transportation*, vol. 2021, pp. 1–14, 10 2021. [Cited on pages vii and 16]

[24] Q. Xu, Q. Wang, C. Xu, and L. Qu, "Collective vertex classification using recursive neural network," *ArXiv*, vol. abs/1701.06751, 2017. [Cited on pages vii and 17]

[25] D. Johnson, "Supervised machine learning: What is, algorithms with examples." Available at `https://www.guru99.com/supervised-machine-learning.html`. (Last accessed in 27/01/2023). [Cited on page 19]

[26] D. Johnson, "Unsupervised machine learning: Algorithms, types with example." Available at `https://www.guru99.com/unsupervised-machine-learning.html`. (Last accessed in 27/01/2023). [Cited on page 19]

[27] DataRobot, "Semi-supervised learning." Available at `https://www.datarobot.com/blog/semi-supervised-learning/`. (Last accessed in 29/01/2023). [Cited on page 19]

[28] D. Johnson, "Reinforcement learning: What is, algorithms, types examples." Available at `https://www.guru99.com/reinforcement-learning-tutorial.html`. (Last accessed in 29/01/2023). [Cited on pages vii and 19]

[29] A. Luckow, M. Cook, N. Ashcraft, E. Weill, E. Djerekarov, and B. Vorster, "Deep learning in the automotive industry: Applications and tools," in *2016 IEEE International Conference on Big Data (Big Data)*, pp. 3759–3768, 2016. [Cited on page 21]

[30] F. Falcini, G. Lami, and A. M. Costanza, "Deep learning in automotive software," *IEEE Software*, vol. 34, no. 3, pp. 56–63, 2017. [Cited on page 21]

[31] M. Cheng, "Deep learning methods towards infotainment systems," Master's thesis, Chalmers University of Technology, 2019. [Cited on pages vii, 21, and 22]

[32] M. Dewalegama, A. de Zoysa, L. Kodikara, D. Dissanayake, T. A. Kuruppu, and S. Rupasinghe, "Deep learning-based smart infotainment system for taxi vehicles," in *2022 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA)*, pp. 1–6, 2022. [Cited on pages vii, 22, 23, and 24]

[33] M. N. Gevorkyan, A. V. Demidova, T. S. Demidova, and A. A. Sobolev, "Review and comparative analysis of machine learning libraries for machine learning," *Discrete and Continuous Models and Applied Computational Science*, vol. 27, no. 4, pp. 305–315, 2019. [Cited on pages vii, xi, 24, 25, and 26]

[34] TensorFlow, "Tensorflow." Available at `https://www.tensorflow.org/?hl=pt-br`. (Last accessed in 04/02/2023). [Cited on page 25]

[35] Keras, "Keras." Available at `https://keras.io/`. (Last accessed in 04/02/2023). [Cited on page 25]

[36] C. editorial team, "What is a car infotainment system?." Available at `https://www.cazoo.co.uk/the-view/buying/what-is-a-car-infotainment-system/`. (Last accessed in 05/02/2023). [Cited on page 26]

[37] P. Sivakumar, R. S. Sandhya Devi, A. Neeraja Lakshmi, B. VinothKumar, and B. Vinod, "Automotive grade linux software architecture for automotive infotainment system," in *2020 International Conference on Inventive Computation Technologies (ICICT)*, pp. 391–395, 2020. [Cited on pages 26 and 27]

[38] APTIV, "Android automotive transforms vehicle infotainment." Available at `https://www.aptiv.com/docs/default-source/white-papers/2020-aptiv-whitepaper-native-google-android-v.pdf?sfvrsn=833c43d_15`. (Last accessed in 05/02/2023). [Cited on pages 27 and 28]

[39] T. L. Foundation, "About automotive grade linux (agl)." Available at `https://www.automotivelinux.org/about/`. (Last accessed in 05/02/2023). [Cited on page 27]

[40] L. Anavi, "Automotive grade linux on raspberry pi: How does it work?." Available at `https://elinux.org/images/b/b6/Automotive_Grade_Linux_on_raspberry_pi.pdf`. (Last accessed in 05/02/2023). [Cited on pages vii and 27]

[41] A. Source, "What is android automotive?." Available at `https://source.android.com/docs/devices/automotive/start/what_automotive`. (Last accessed in 05/02/2023). [Cited on page 28]

[42] B. Bennett, "Android automotive 12 is all i want on my dashboard." Available at `https://mobilesyrup.com/2021/12/09/android-automotive-12-is-all-i-want-on-my-dashboard/`. (Last accessed in 05/02/2023). [Cited on pages vii and 28]

[43] S. G. Mobility, "Android automotive is taking over, but what about google automotive services (gas)?." Available at `https://www.spglobal.com/mobility/en/research-analysis/android-automotive-is-taking-over-but-what-about-google.html`. (Last accessed in 05/02/2023). [Cited on page 28]

[44] B. Group, "Bmw group expands bmw operating system 8, integrates android automotive os.." Available at `https://www.press.bmwgroup.com/portugal/article/detail/T0401912PT/bmw-group-expands-bmw-operating-system-8-integrates-android-automotive-os?language=pt`. (Last accessed in 05/02/2023). [Cited on page 29]

[45] NXP, "i.mx 8quadxplus multisensory enablement kit (mek)." Available at `https://www.nxp.com/design/development-boards/i-mx-evaluation-and-development-boards/i-mx-8quadxplus-multisensory-enablement-kit-mek:MCIMX8QXP-CPU`, 2022. (Last accessed in 21/11/2022). [Cited on page 32]

[46] A. Gilik, S. Ogrenci, and A. Özmen, "Air quality prediction using cnn+lstm-based hybrid deep learning architecture," *Environmental Science and Pollution Research*, vol. 29, pp. 1–19, 02 2022. [Cited on pages vii and 38]

[47] N. Donges, "What is transfer learning? exploring the popular deep learning approach.." Available at `https://builtin.com/data-science/transfer-learning`. (Last accessed in 16/04/2023). [Cited on page 38]

[48] Datagen, "Understanding vgg16: Concepts, architecture, and performance." Available at `https://datagen.tech/guides/computer-vision/vgg16/`. (Last accessed in 25/04/2023). [Cited on pages vii, 38, and 39]

[49] P. U. Stanford Vision Lab, Stanford University, "Imagenet." Available at `https://www.image-net.org/`. (Last accessed in 20/05/2023). [Cited on page 38]

[50] P. Ruiz, "Understanding and visualizing resnets." Available at `https://towardsdatascience.com/understanding-and-visualizing-resnets-442284831be8`. (Last accessed in 25/04/2023). [Cited on page 39]

[51] J. Chen, M. Zhou, D. Zhang, H. Huang, and F. Zhang, "Quantification of water inflow in rock tunnel faces via convolutional neural network approach," *Automation in Construction*, vol. 123, p. 103526, 03 2021. [Cited on pages vii and 39]

[52] T.-Y. L. et al, "Coco - common objects in context." Available at https://cocodataset.org/#home. (Last accessed in 21/05/2023). [Cited on page 40]

[53] O. IQ, "Yolo v5 model architecture [explained]." Available at https://iq.opengenus.org/yolov5/. (Last accessed in 30/04/2023). [Cited on pages 40, 42, 43, 44, and 46]

[54] S. Gutta, "Object detection algorithm — yolo v5 architecture." Available at https://medium.com/analytics-vidhya/object-detection-algorithm-yolo-v5-architecture-89e0a35472ef. (Last accessed in 30/04/2023). [Cited on page 40]

[55] Ultralytics, "Yolov5." Available at https://github.com/ultralytics/yolov5. (Last accessed in 30/04/2023). [Cited on pages vii, 40, and 67]

[56] I. Katsamenis, E. Karolou, A. Davradou, E. Protopapadakis, A. Doulamis, N. Doulamis, and D. Kalogeras, "Tracon: A novel dataset for real-time traffic cones detection using deep learning," 05 2022. [Cited on pages vii and 41]

[57] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," 2018. [Cited on pages viii and 41]

[58] Z. Tian, J. Huang, Y. Yang, and W. Nie, "Kcfs-yolov5: A high-precision detection method for object detection in aerial remote sensing images," *Applied Sciences*, vol. 13, no. 1, 2023. [Cited on pages 42 and 43]

[59] A. Mondin, "Yolov5(m): Implementation from scratch with pytorch." Available at https://pub.towardsai.net/yolov5-m-implementation-from-scratch-with-pytorch-c8f84a66c98b. (Last accessed in 01/05/2023). [Cited on pages 42, 43, 44, 45, and 46]

[60] WZMIAOMIAO, "Yolov5 (6.0/6.1) brief summary." Available at https://github.com/ultralytics/yolov5/issues/6998. (Last accessed in 01/05/2023). [Cited on pages 42 and 43]

[61] A. M. Fawzy, *A deep learning based real-time object detection implementation in a virtual instrument cluster*. PhD thesis, DSpace, 2019. [Cited on pages viii, 43, 50, and 51]

[62] M. R, "Panet: Path aggregation network in yolov4." Available at https://medium.com/clique-org/panet-path-aggregation-network-in-yolov4-b1a6dd09d158. (Last accessed in 30/04/2023). [Cited on page 43]

[63] L. Gilani, S. F. Tahir, U. Rasheed, H. Saqib, M. Hassan, and H. Alquhayz, "Fruits and vegetables freshness categorization using deep learning," *Computers, Materials Continua*, vol. 71, pp. 5083–5098, 01 2022. [Cited on pages 43, 45, and 46]

[64] M. Azam, C. Sampieri, A. Ioppi, S. Africano, A. Vallin, D. Mocellin, M. Fragale, L. Guastini, S. Moccia, C. Piazza, L. Mattos, and G. Peretti, "Deep learning applied to white light and narrow band imaging videolaryngoscopy: Toward real-time laryngeal cancer detection," *The Laryngoscope*, vol. 132, 11 2021. [Cited on pages viii and 44]

[65] J. Cowton, I. Kyriazakis, and J. Bacardit, "Automated individual pig localisation, tracking and behaviour metric extraction using deep learning," *IEEE Access*, vol. PP, 08 2019. [Cited on pages viii and 46]

[66] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. E. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector.," in *ECCV (1)* (B. Leibe, J. Matas, N. Sebe, and M. Welling, eds.), vol. 9905 of *Lecture Notes in Computer Science*, pp. 21–37, Springer, 2016. [Cited on pages viii, 46, 47, and 48]

[67] T. Kim, J. Lee, W. Lee, and S. Sohn, "Novel method for detection of mixed-type defect patterns in wafer maps based on a single shot detector algorithm," *Journal of Intelligent Manufacturing*, vol. 33, 08 2022. [Cited on pages viii and 48]

[68] M. E. Aidouni, "Ssd : Understanding single shot object detection." Available at `https://manalelaidouni.github.io/Single%20shot%20object%20detection.html`. (Last accessed in 25/05/2023). [Cited on pages viii and 49]

[69] S. Kumar, "Data augmentation increases accuracy of your model — but how?." Available at `https://medium.com/secure-and-private-ai-writing-challenge/data-augmentation-increases-accuracy-of-your-model-but-how-aa1913468722`. (Last accessed in 25/05/2023). [Cited on pages viii and 50]

[70] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 2999–3007, 2017. [Cited on pages viii, 50, 51, and 52]

[71] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, (Cambridge, MA, USA), p. 91–99, MIT Press, 2015. [Cited on pages viii, 53, 54, and 55]

[72] M. Maity, S. Banerjee, and S. S. Chaudhuri, "Faster r-cnn and yolo based vehicle detection: A survey," *2021 5th International Conference on Computing Methodologies and Communication (ICCMC)*, pp. 1442–1447, 2021. [Cited on pages viii and 54]

[73] J. Brownlee, "Difference between a batch and an epoch in a neural network." Available at `https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/`. (Last accessed in 14/05/2023). [Cited on page 55]

[74] C. Delteil, "Mastering the fundamentals: Differences between sample, batch, iteration, and epoch." Available at `https://pub.towardsai.net/mastering-the-fundamentals-differences-between-sample-batch-iteration-and-epoch-bd5a33b2c5fd`. (Last accessed in 14/05/2023). [Cited on page 55]

[75] deepchecks, "What is the learning rate in machine learning?." Available at `https://deepchecks.com/glossary/learning-rate-in-machine-learning/`. (Last accessed in 14/05/2023). [Cited on page 55]

[76] D. Nikolaiev, "Overfitting and underfitting principles." Available at `https://towardsdatascience.com/overfitting-and-underfitting-principles-ea8964d9c45c`. (Last accessed in 11/05/2023). [Cited on page 55]

[77] dewangNautiyal, "https://www.geeksforgeeks.org/underfitting-and-overfitting-in-machine-learning/." Available at `https://www.geeksforgeeks.org/underfitting-and-overfitting-in-machine-learning/`. (Last accessed in 11/05/2023). [Cited on pages viii, 55, and 57]

[78] J. An, "Batch size vs epochs." Available at `https://jerryan.medium.com/batch-size-a15958708a6`. (Last accessed in 28/05/2023). [Cited on pages viii and 56]

[79] R. Padilla, S. L. Netto, and E. A. B. da Silva, "A survey on performance metrics for object-detection algorithms," in *2020 International Conference on Systems, Signals and Image Processing (IWSSIP)*, pp. 237–242, 2020. [Cited on pages viii, 57, 58, and 59]

[80] J. Korstanje, "The f1 score." Available at `https://towardsdatascience.com/the-f1-score-bec2bbc38aa6`. (Last accessed in 14/05/2023). [Cited on page 58]

[81] K. E. Koech, "Object detection metrics with worked example." Available at `https://towardsdatascience.com/on-object-detection-metrics-with-worked-example-216f173ed31e`. (Last accessed in 14/05/2023). [Cited on page 58]

[82] A. Bajaj, "Performance metrics in machine learning [complete guide]." Available at `https://neptune.ai/blog/performance-metrics-in-machine-learning-complete-guide`. (Last accessed in 14/05/2023). [Cited on pages viii, 59, and 60]

[83] Google, "Colaboratory." Available at `https://research.google.com/colaboratory/faq.html`. (Last accessed in 15/06/2023). [Cited on page 68]

[84] M. D. Bloice, "Augmentor." Available at `https://augmentor.readthedocs.io/en/stable/`. (Last accessed in 15/06/2023). [Cited on page 68]

[85] Roboflow, "Roboflow - build better computer vision models faster." Available at `https://docs.roboflow.com/`. (Last accessed in 15/06/2023). [Cited on page 68]

[86] R. Framework, "Robot framework ua." Available at `https://robotframework.org/`. (Last accessed in 17/06/2023). [Cited on pages xiii and 75]

[87] Ultralytics, "Yolov5 tutorial." Available at `https://colab.research.google.com/github/ultralytics/yolov5/blob/master/tutorial.ipynb`. (Last accessed in 29/07/2023). [Cited on page 78]

[88] TensorFlow, "tf.keras.applications.mobilenet_v2.mobilenetv2." Available at `https://www.tensorflow.org/api_docs/python/tf/keras/applications/mobilenet_v2/MobileNetV2`. (Last accessed in 29/07/2023). [Cited on page 83]

[89] E. Juras, "Tensorflow lite object detection api in colab." Available at `https://colab.research.google.com/drive/1BxhHWRB5R4Em_NwWFG5kdrzvGiCkfDan#scrollTo=HEsOLOMHzBqF`. (Last accessed in 29/07/2023). [Cited on page 84]

[90] Samjith888, "Keras-retinanet-training-on-custom-datasets-for-object-detection-." Available at `https://github.com/Samjith888/Keras-retinanet-Training-on-custom-datasets-for-Object-Detection-/tree/master/keras_retinanet`. (Last accessed in 03/08/2023). [Cited on page 87]

[91] F. A. Research, "Faster r-cnn on detectron2." Available at `https://colab.research.google.com/drive/1otd2Ew89q1QlbUgz8fzhrSkRlsiiDMCt#scrollTo=v8oCoXRj9Xd5`. (Last accessed in 06/08/2023). [Cited on page 90]

## Appendix A

# SSD Results

Table A.1: mAP results of the first training.

| **Class** | **Ground-Truth Objects** | **Number of Detected Objects** | **True Positives** | **True Negatives** |
|---|---|---|---|---|
| AC | 62 | 139 | 55 | 84 |
| Air Circulation | 56 | 210 | 50 | 160 |
| All Apps | 39 | 75 | 38 | 37 |
| Auto AC | 82 | 249 | 76 | 173 |
| Bottom AC | 82 | 200 | 72 | 128 |
| Bottom Top Heat | 74 | 144 | 63 | 81 |
| Car Data | 1 | 40 | 1 | 39 |
| Car Info | 30 | 71 | 29 | 42 |
| Climate | 48 | 218 | 45 | 173 |
| Connectivity | 66 | 147 | 60 | 7 |
| Cooling | 86 | 136 | 78 | 58 |
| Decrease | 22 | 52 | 32 | 20 |
| Eco Data | 1 | 1 | 1 | 9 |
| Front Heat | 71 | 132 | 61 | 71 |
| Home | 32 | 52 | 32 | 20 |
| Increase | 25 | 51 | 12 | 39 |
| Minus | 55 | 92 | 55 | 37 |
| Music | 48 | 95 | 45 | 50 |
| Navigation | 59 | 151 | 51 | 100 |
| Plus | 49 | 72 | 49 | 23 |
| Rear Heat | 72 | 110 | 66 | 44 |
| Seat Heating | 151 | 553 | 139 | 414 |
| Settings | 43 | 371 | 40 | 331 |
| Top AC | 59 | 161 | 53 | 108 |
| Top Bottom AC | 117 | 314 | 103 | 211 |

# Appendix B

# Robot Framework

## B.1 Template

```
*** Settings ***
Library    OperatingSystem
Library    String
Resource    keywords.robot

Test Setup    Get Devices

*** Test Cases ***
Run Selected Tests
    Get Tag From User
    [Documentation]    This test will allow the user to select the
        tests to run.
    ${json_data}    Parse JSON File    selected_data.json
        FOR    ${data}    IN    @{json_data}
            ${file}        Get Coordinates File    ${data}
            ${lines}       Get Lines From Coordinates File    ${
                file}
            Log Test Name    ${data}
            ${screenshots_path}    Get Screenshots Path    ${data}
                FOR    ${line}    IN    @{lines}
                    ${columns}=    Split String    ${line}
                        separator=,
                    ${x}=    Get From List    ${columns}    0
```

```
                    ${y}=    Get From List    ${columns}    1
                    Touch Coordinates    ${x}    ${y}
                    Sleep    5
                    Take Screenshots
                    ${folder_path}    Set Variable    ../Robot/
                        Robot_Screenshots
                END
            Run Keyword And Continue On Failure    Verify Matching
                 Images   ${screenshots_path}   ${folder_path}
                0.98
            Save Automate Tests Screenshots    ${data}
        END

    [Teardown]    Delete Temporary Json File    selected_data.json
```

Listing B.1: Final Template.

## B.2 Keywords

```
*** Settings ***
Library    JSONLibrary
Library    api.py
Library    Collections

*** Keywords ***
Get Tag From User
    [Documentation]    This keyword will show a prompt where the
        user can see the available tags and write the tags and
        tests to execute. It will also verify if the tag name
        exists and in case it doesn't it will fail.
    Get Tag Names From Json    data.json
    Verify Tag Names In Json    data.json

Parse JSON File
    [Documentation]    This keyword will parse through the json
        file.
    [Arguments]    ${json_file_path}
    ${json_data}    Load JSON From File    ${json_file_path}
    [Return]    ${json_data}

Get Screenshots Path
    [Documentation]    This keyword will get the manual
        screenshots' path of the selected tests.
    [Arguments]    ${data}
    ${screenshots}    Get Value From JSON    ${data}
        screenshots_path
```

```
22      ${screenshots_path}    Get From List    ${screenshots}    0
23      [Return]    ${screenshots_path}
24
25  Log Test Name
26      [Documentation]    This keyword will log into console the name
            of the test that is running.
27      [Arguments]    ${data}
28      ${name}    Get Value From JSON    ${data}    test_name
29      ${test_name}    Get From List    ${name}    0
30      Log To Console    Running ${test_name}
31      [Return]    ${test_name}
32
33  Save Automate Tests Screenshots
34      [Documentation]    This keyword will save all the screenshots
            taken during the automated tests and save them in the
            corresponding folder of the content of the test in
            execution.
35      [Arguments]    ${data}
36      ${name}    Get Value From JSON    ${data}    test_name
37      ${test_name}    Get From List    ${name}    0
38      ${tag}    Get Value From JSON    ${data}    tag_name
39      ${tag_name}    Get From List    ${tag}    0
40      Automated Test Screenshots    ${tag_name}    ${test_name}
41
42  Tap Coordinates
43      [Documentation]    This keyword clicks on the coordinates
            detected by the algorithm and prints a message saying in
            which icon it is clicking.
44      [Arguments]    ${x}    ${y}
45      Touch Coordinates    ${x}    ${y}
46
47  Get Coordinates File
48      [Documentation]    This keyword will get the coordinates file
            of the test that is running.
49      [Arguments]    ${data}
50      ${file}    Get Value From JSON    ${data}    coordinates_file
51      ${coordinates}    Get From List    ${file}    0
52      [Return]    ${coordinates}
```

Listing B.2: Developed Keywords.