



INSTITUTO  
UNIVERSITÁRIO  
DE LISBOA

---

## **Mobile Application to Identify Recyclable Materials**

António Francisco Serol Sequeira

Master in Computer Engineering

Supervisor:

Prof. Dra. Ana Maria Carvalho de Almeida, Associate Professor,  
Iscte Instituto Universitário de Lisboa

Co-supervisor:

Prof. Dr. Luís Miguel Martins Nunes, Associate Professor,  
Iscte Instituto Universitário de Lisboa

October, 2020





TECNOLOGIAS  
E ARQUITETURA

---

## **Mobile Application to Identify Recyclable Materials**

António Francisco Serol Sequeira

Master in Computer Engineering

Supervisor:

Prof. Dra. Ana Maria Carvalho de Almeida, Associate Professor,  
Iscte Instituto Universitário de Lisboa

Co-supervisor:

Prof. Dr. Luís Miguel Martins Nunes, Associate Professor,  
Iscte Instituto Universitário de Lisboa

October, 2020

Direitos de cópia ou Copyright

©Copyright: António Francisco Serol Sequeira.

O Iscte - Instituto Universitário de Lisboa tem o direito, perpétuo e sem limites geográficos, de arquivar e publicitar este trabalho através de exemplares impressos reproduzidos em papel ou de forma digital, ou por qualquer outro meio conhecido ou que venha a ser inventado, de o divulgar através de repositórios científicos e de admitir a sua cópia e distribuição com objetivos educacionais ou de investigação, não comerciais, desde que seja dado crédito ao autor e editor.

## **Acknowledgments**

I would like to thank the following people, without whom I would not have been able to have made it through this dissertation.

To both my supervisors Professor Ana Almeida and Professor Luís Nunes for teaching me so much about machine learning and for their willingness for providing guidance and feedback that substantially improved the quality of this dissertation.

To my family, especially my parents for their constant support that they have shown me though this project and for always doing the possible and the impossible for me.

To my girlfriend, Raquel, for her love and all the support that she shows every day on both the small and big life problems.

Finally, to my friends, for understanding that I could not be there for some occasions due to the time required for this dissertation.



## Resumo

Nesta dissertação é proposto um sistema para ajudar o consumidor a reciclar eficientemente. O sistema é composto por uma aplicação móvel que captura imagens de lixo e classifica a sua categoria usando um modelo de aprendizagem automática. Conseguir também comunicar com um servidor para atualizar o modelo com versões melhoradas e enviar as imagens para o servidor para contribuir para a criação de modelos mais precisos.

Foi demonstrado através de um protótipo totalmente funcional que o sistema proposto funciona. Algumas imagens de lixo foram categorizadas correctamente, mas o modelo de aprendizagem automática produzido durante este projeto não é preciso o suficiente, em qualquer categoria de lixo, para usar em cenários da vida real.

As principais contribuições deste estudo são um compêndio de informação na área de visão de computador e aprendizagem automática para categorizar lixo, e um sistema protótipo funcional que utiliza elementos de contribuição colaborativa e aprendizagem automática para ajudar o consumidor a reciclar mais eficientemente.

**Palavras-Chave:** Visão de Computador; Aprendizagem Automática; Reciclagem; Contribuição Colaborativa; Gestão de lixo.





## **Abstract**

This dissertation proposes a system to help the consumer recycle efficiently. The system is composed by a mobile application that can capture images of waste and classify their category through the usage of a machine learning model. Furthermore, this application can communicate with a server to update the model with new improved versions and also upload the images to the server in order to contribute to the creation of more precise model versions.

The system has been validated by a fully working prototype. Although the proof of concept has been achieved, with some types of waste items correctly categorized, the machine learning model produced is not precise enough to be used in real-life scenarios, that is, for any type of waste.

The main contributions of this study are a compendium of information in the area of computer vision and machine learning to categorize waste, and a working prototype system that utilizes crowdsourcing and machine learning elements to help the consumer recycle more efficiently.

**Keywords:** Computer Vision; Machine Learning; Recycling; Crowdsourcing; Waste Management.



## Contents

<b>Acknowledgments</b> .....	<b>i</b>
<b>Resumo</b> .....	<b>iii</b>
<b>Abstract</b> .....	<b>v</b>
<b>Contents</b> .....	<b>vii</b>
<b>List of Tables</b> .....	<b>ix</b>
<b>List of Figures</b> .....	<b>xi</b>
<b>Abbreviations</b> .....	<b>xiii</b>
<b>Chapter 1 – Introduction</b> .....	<b>1</b>
1.1. Motivation.....	2
1.2. Research Questions .....	2
1.3. Objectives .....	3
1.4. Research methodology.....	3
1.5. Scientific Contribution.....	4
1.6. Structure and organization of the dissertation .....	4
<b>Chapter 2 – Literature Review</b> .....	<b>7</b>
2.1. Methodology .....	7
2.2. Related Work .....	8
2.2.1. Robotic Systems .....	8
2.2.2. Smart Trash Systems .....	10
2.2.3. Mobile Applications .....	11
2.2.4. Case Studies.....	12
2.3. Results from the Literature .....	12
<b>Chapter 3 – MobileNet</b> .....	<b>21</b>
<b>Chapter 4 – High-level Architecture</b> .....	<b>25</b>
<b>Chapter 5 – ML Model Processor</b> .....	<b>31</b>
5.1. MobileNet v2 .....	32
5.1.1. Methodology.....	33
5.1.2. Results .....	34
5.1.3. Converting to mobile .....	38
5.1.4. Repository.....	38
5.2. SSD MobileNet v2.....	39
5.2.1. Methodology.....	40
5.2.2. Results .....	42
5.2.3. Converting to mobile .....	48
5.2.4. Repository.....	48

<b>Chapter 6 – REST API.....</b>	<b>49</b>
6.1. Architecture .....	50
6.2. Endpoint Definition .....	51
6.3. Testing .....	51
6.4. Repository .....	51
<b>Chapter 7 – Mobile Application .....</b>	<b>53</b>
7.1. Architecture .....	54
7.2. Available Actions .....	54
7.3. Testing .....	61
7.4. Repository .....	62
<b>Chapter 8 – Conclusions and recommendations .....</b>	<b>63</b>
8.1. Main conclusions .....	63
8.2. Contributions to the scientific and business community .....	64
8.3. Limitation of the study.....	65
8.4. Future work.....	66
<b>Bibliography.....</b>	<b>67</b>
<b>Appendices .....</b>	<b>71</b>
Appendix A .....	73
<b>Model Endpoints.....</b>	<b>73</b>
/model/download/ .....	73
/model/version/ .....	73
/model/upload/ .....	74
<b>User Endpoints.....</b>	<b>74</b>
/users/login/ .....	74
/users/register/.....	75
/users/logout/ .....	75
Appendix B.....	77

## List of Tables

Table 2.1 – List of databases .....	8
Table 2.2 – Search criteria.....	8
Table 2.3 – Synthetized results from the literature.....	16
Table 4.1 – Trashnet dataset details.....	27
Table 4.2 – Real life images dataset distribution.....	29
Table 5.1 – Test dataset results confusion matrix with Mobilenet v2.....	36
Table 5.2 – Precision results for Test dataset by category with Mobilenet v2.....	36
Table 5.3 – Real dataset results confusion matrix with Mobilenet v2 .....	37
Table 5.4 – Precision results for Real dataset by category with Mobilenet v2 .....	37
Table 5.5 – Test dataset results confusion matrix with SSD Mobilenet v2.....	46
Table 5.6 – Precision results for Test dataset by category with SSD Mobilenet v2 .....	46
Table 5.7 – Real dataset results confusion matrix with SSD Mobilenet v2 .....	47
Table 5.8 – Precision results of Real dataset by category with SSD Mobilenet v2 .....	47



## List of Figures

Figure 2.1 – Search process .....	7
Figure 2.2 – Algorithm Distribution.....	13
Figure 2.3 – CNN Architecture Distribution .....	14
Figure 2.4 – Dataset Distribution .....	14
Figure 2.5 – Accuracy benchmarking results with Trashnet dataset.....	15
Figure 3.1 – The structure of an example CNN [41].....	21
Figure 3.2 – MobileNet body architecture [44].....	22
Figure 3.3 – Factorization of standard convolution [44].....	23
Figure 3.4 – Representation of MobileNet layer except the last layer [44].....	23
Figure 4.1 – High-level system architecture.....	25
Figure 4.2 – Cardboard category image samples .....	28
Figure 4.3 – Glass category image samples .....	28
Figure 4.4 – Metal category image samples .....	28
Figure 4.5 – Paper category image samples .....	28
Figure 4.6 – Plastic category image samples.....	28
Figure 4.7 – Trash category image samples .....	28
Figure 4.8 – Image samples for real life purpose test.....	29
Figure 5.1 – Example of fine-tuning [46].....	32
Figure 5.2 – Model training accuracy.....	35
Figure 5.3 – Model training loss.....	35
Figure 5.4 – Example of SSD MobileNet architecture [51] .....	39
Figure 5.5 – mAP results from Tensorboard .....	43
Figure 5.6 – mAP IOU results from Tensorboard .....	44
Figure 5.7 – Loss metrics from Tensorboard .....	45
Figure 6.1 – REST API architecture [56] .....	49
Figure 6.2 – Django project structure .....	50
Figure 7.1 – Mobile application architecture.....	54
Figure 7.2 – Mobile application experience flow .....	55
Figure 7.3 – Entry screen.....	56
Figure 7.4 – Select image screen .....	56
Figure 7.5 – Evaluation screen .....	57
Figure 7.6 – Capture image screen .....	58
Figure 7.7 – Confirm image screen .....	59
Figure 7.8 – Login screen .....	60
Figure 7.9 – Menu screen .....	61
Figure B.1 – User register testing.....	77
Figure B.2 – User login testing.....	77
Figure B.3 – Model version testing .....	77
Figure B.4 – Model download testing .....	78
Figure B.5 – Model upload testing .....	78

Figure B.6 – User logout testing..... 78



## **Abbreviations**

API – Application Programming Interface

COCO – Common Objects in Context

CNN – Convolutional Neural Network

CPU – Central Process Unit

CSV – Comma-Separated Value

GINI – Garbage in Images

GPU – Graphics Processing Unit

HTTP – Hypertext Transfer Protocol

HKNN – Hyperplane Nearest Neighbors

IoT – Internet of Things

IOU – Intersection Over Unit

KNN – K-Nearest Neighbors

mAP – Mean Average Precision

ML – Machine Learning

MVC – Model View Controller

PASCAL – Pattern Analysis, Statistical modelling and Computational Learning

RF – Random Forest

R-CNN – Regions Convolutional Neural Network

ReLU – Rectified Linear Union

REST – Representational State Transfer

SDK – Software Development Kit

SIFT – Scale Invariant Feature Transform

SSD – Single Shot Detector

SVM – Support Vector Machine

UI – User Interface

URL – Uniform Resource Locator

VOC – Visual Object Class

XML – Extensible Markup Language

## Chapter 1 – Introduction

Waste is defined as “any substance or object which the holders discards or intends or is required to discard”<sup>1</sup>. Our societies generate waste at an increasing and alarming rate. Each European Union inhabitant in 2016<sup>2</sup> generated on average 5.0 tons of waste, only 37.8 % of this waste was recycled with 45,7% being landfilled. This means that 3,11 tons of waste per inhabitant were landfilled, incinerated or deposited in the environment. These statistics show that recycling still has a great margin for improvement and that waste that does not get recycled may end up polluting the air, water and soil since landfills take up land space. Incineration is also an alternative way to treat waste, yet this solution may result in emissions of air pollutants. Increasing the efficiency of recycling is a mandatory step to reduce the pollution to the environment, reduce health impacts and to achieve better resource efficiency.

Recycling is defined as any operation with the aim of recovering waste materials, reprocessing them into products, materials or substances that can be used again after being processed<sup>3</sup>. Not all waste can be recycled in the same way, it is important to categorize and separate it. An important level of waste segregation is achieved at the end point of the recycling lifecycle, where factories specialized in waste segregation use semi-autonomous methods to separate the trash. However, waste segregation is achieved at multiple levels and, at the consumer level, it is achieved by using different containers. Yet, for this to be efficient, the consumer needs to know in which container he/she should put the waste. Moreover, the consumer needs to be motivated to do so mainly because this task can be more elaborated than what could be expected. For instance, if you have a paper bag that has a plastic window, where should you throw out this waste: the paper container, the plastic container?

Computers have proven to surpass humans at certain tasks, especially in repetitive tasks. Based on this premise, machine learning (ML) has been successfully used in the implementation of waste segregation systems for identification of waste categories based on image recognition as explained in Chapter 2. On the other hand, smartphones, at each

---

<sup>1</sup> Definition from Article 3 in website: <https://eur-lex.europa.eu/legal-content/EN/ALL/?uri=CELEX:32008L0098>

<sup>2</sup> From website: [https://ec.europa.eu/eurostat/statistics-explained/index.php/Waste\\_statistics](https://ec.europa.eu/eurostat/statistics-explained/index.php/Waste_statistics)

<sup>3</sup> Definition from website: [https://ec.europa.eu/eurostat/statistics-explained/index.php?title=Glossary:Recycling\\_of\\_waste](https://ec.europa.eu/eurostat/statistics-explained/index.php?title=Glossary:Recycling_of_waste)

generation, are increasing their processing power and camera's resolution. Joining together ML and smartphones seems like a natural match for present and future applications that can analyze images and classify them, thus helping the consumer to separate waste for recycling.

It is known that ML requires large amounts of data to produce reliable models. Depending on the type of problem solving to be achieved by using ML, these models might require a flow of new data to keep updating them in order to successfully solve an evolving problem of classification. Applying crowdsourcing concepts in such system can help secure this flow of data. Crowdsourcing can be defined as outsourcing work to a group of agents, where traditionally this work would be assigned to a designated agent [1].

### **1.1. Motivation**

As stated, there still exists a great margin for improving recycling rates at the consumer's level. As such, the creation of systems that help the users understand how to recycle is important. Since nowadays almost everyone has a smartphone, an application that helps users with the task in hand seems to be an idea. Yet, due to the large amounts of data required for training precise ML models and to maintain them updated, it is important to secure a flow of data. Using crowdsourcing techniques can help such requirements with a continuous flow of new data. For this a centralized server is required to receive the images captured by the application. These images will be used to train improved ML models that can be downloaded by the application.

### **1.2. Research Questions**

The purpose of this research is to create an ML model for usage in an application for smartphones, to analyze waste images and classify them into its specific recycling category. Nevertheless, several questions remain.

RQ1: Considering the requisites of an ML model, which types of ML algorithms are best suited for the classification of waste images within a smartphone application?

RQ2: Is it possible to create an accurate (that is, achieving over 90% precision) but yet light weight model (enabling a quick classification: under three seconds) able to run in smartphones?

RQ3: Is the most cited public available dataset of waste images diverse enough to produce a ML model that can classify waste into its specific recycling category in a real-life scenario?

### **1.3. Objectives**

This work's main objective is to create a mobile application that can help people to recycle. The system can work by image acquisition (capturing an image) and classifying it into a respective recycling category. The application will be able to send the captured images to a Representational State Transfer (REST) Application Programming Interface (API) and reinforce the model present in the application by updating it with the most recent ML model available through the API. For this, the main research questions will be investigated.

Furthermore, this work intends to synthesize the related work in the area of computer vision that uses ML to categorize waste in order to create a compendium for present and further investigation.

### **1.4. Research methodology**

The research methodology used in order to achieve the purposed objectives for this work was the Design Science Research model, focused on the design and development. The following are this dissertation's proposed developments:

- 1) Creation of a multi-classifier ML model capable of identifying waste in images and classify it into its specific recycling category.

- 2) Creation of a prototype REST API that will communicate with the mobile application to receive feedback from it (in the form of images captured in the application) and also update the ML model of the application.

3) Creation of a prototype Android application, capable of taking a picture and process it into the ML model to get a classification. It should be able to also communicate with a REST API.

4) Field test the prototype system with real life scenarios.

### **1.5. Scientific Contribution**

This dissertation presents the following contributions:

- Compendium of information in the area of computer vision and ML to categorize waste.
- Working prototype system to help people to recycle.

The research on the literature, that was conducted during the development of this dissertation, has been synthetized in a scientific paper submitted to the journal Environmental Technology ID TETR-TENT-2020-1177. At the moment of the conclusion of this dissertation it is still under review having been sent to the reviewers and undergoing the normal revision process.

### **1.6. Structure and organization of the dissertation**

The present study is organized in nine chapters. These chapters will reflect the different phases until its conclusion.

The first chapter introduces the theme of the study along with its motivation, research questions and objectives. This chapter also describes the research methodology utilized in the study.

The second chapter focuses on the literature related with the work that is being studied. It is subdivided in the methodology applied to the literature review, the categorization of the related work in the field and a synthetization of this work. It also presents some results based on the information that can be taken from the literature.

The third chapter reflects the ML architecture to be used in this dissertation. It describes the architecture and introduces concepts that will help understand the work done in upcoming chapters.

The fourth chapter reflects the architecture of the system. It defines a high-level overview of the system. Each module of the system is briefly described in this section while also specifying the technology used. This chapter also describes the dataset that will be used in the training of the ML model for the application. It contextualizes the origin, categorization, and format of the data.

The fifth chapter specifies how the ML model, to be used in the application, is trained. It describes all the steps taken, tests and results.

The sixth chapter presents the REST API information. This chapter will describe the technology used to build the API, the architecture that was used to model the code, the documentation of the code and how to use the API and a section that describes the tests that were done to make sure the API was functioning as expected.

The seventh chapter describes how the mobile application was built, explaining the technology that was used, the architecture of the code and how it was tested to make sure it was functional as required.

The eighth and last chapter presents the conclusions from this study as well as some recommendations, limitations and proposed future works.





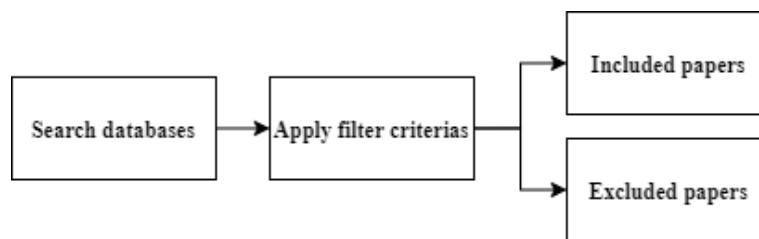
## Chapter 2 – Literature Review

This chapter will describe the literature review methodology and will provide a summary of all the related work found during the review.

### 2.1. Methodology

The present literature research was carried between September 2019 and October 2020, focusing on the theme of Computer Vision applications in the field of recycling systems. Mainly, the intent was to search for scientific papers that described developments or theories falling into the categorization of recycling waste with the help of ML.

The search method used in this study is described by Figure 2.1. The first step was to carry out a systematic search in the available scholar databases (Table 2.1). For this, a list of search criteria was used (Table 2.2). In step two, the results from step one were validated with filters that consisted in the following criteria: the language must be English, the work must be a scientific text, the content is relevant to the paper's topic, the work is not duplicated.



*Figure 2.1 – Search process*

Table 2.1 – List of databases

Databases
IEEE Xplore Digital Library
Google Scholar
B-On
Research Gate
ACM Digital Library
Springer Link

Table 2.2 – Search criteria

Search Strings
Machine learning waste classification
Computer vision waste classification
Computer vision recycling materials
Waste segregation
Recyclable goods
Image recognition recycling
Garbage classification

This search yielded 57 papers and the filtering processing ended with 39 scientific papers concerning the related work in the area of computer vision applied to the classification of recyclable waste. This related work will be described in Section 2.2 and the synthesis of the results as well as some conclusions will be described in Section 2.3.

## 2.2. Related Work

The literature has many examples of how image classification through ML may help to achieve automation in recycling systems. Some just use image classification others use it as part of a more complex system. This section will describe the examples found in the literature in order to clarify a generalized view of the possibilities in this area.

### 2.2.1. Robotic Systems

Some researchers propose robotic systems which enable the automation of garbage handling. The robotic systems can also be sub categorized in similarity of implementation.

### Conveyor Belt Systems

In this type of implementations, the system revolves around a conveyor belt where the waste is processed, such as “a robotic grasping system for automatically sorting garbage based on machine vision” [2], where the system is composed of three main parts: a camera, a conveyor belt and a manipulator for object grasping. The camera images are used to locate and classify the objects with Regions Convolutional Neural Network (RCNN) that identifies a subset of regions in the image that might contain an object and then tries to classify the objects in the images. Another study presents a method to classify waste in a conveyor belt, through the usage of a high-speed camera and the extraction of texture features combined with a probabilistic neural network to classify the waste [3]. Some researchers suggest another way for classification of plastic bottles in a conveyor belt system can be achieved through the usage of image classification by a Support Vector Machine (SVM) algorithm [4]. An implementation shows a conveyor belt system that is able to classify different grades of paper only through image processing techniques using the K-Nearest Neighbors (KNN) algorithm [5]. The authors of [6] suggest another design for a system that can identify solid waste in a conveyor belt and grab the waste with a robotic arm using a Hyperplane Nearest Neighbors (HKNN) algorithm.

### Mobile Systems

Here, the system is mobile and can move freely in an environment in order to reach static waste as proposed in the development [7] of a robot equipped with a thermal imaging camera, proximity sensor and a robotic arm. The robot employs a bag-of-features and a multi-class SVM in order to identify and classify recyclable materials. One other study proposes a design for a semi-autonomous robot that would segregate recyclable and non-recyclable waste using a content based image retrieval method that contains a bag-of-features [8]. Another work proposes to quantify littering in urban environments with a camera mounted on top of a vehicle and facing the ground. The images captured are then used to feed a deep Convolutional Neural Network (CNN) so it could classify the waste along with its position and timestamp [9]. Another work suggests a robot that can categorize items for recycling through the automatic localization of the items using depth image analysis, grab them with a mechanical arm and place them in the corresponding recycling bin. This last part is achieved with a CNN algorithm for recognition and categorization [10]. The authors of [11] demonstrate a robotic system that is able to collect

and segregate waste, through the usage of a CNN model for image classification. In [12] an idea for a construction waste recycling robot is present, with the aim of identifying and classifying nails and screws within a construction waste scene with the usage of a R-CNN algorithm. [13] propose a micro-unmanned aerial vehicle capable of detecting litter in real-time through the usage of a CNN with object detection algorithm and benchmarks three different detection algorithms in the study.

### Miscellaneous Systems

This category concerns systems that are abstract in its implementation, presenting a system that can recognize and classify waste, but not specifying how to do it. [14] mentions a computer that is able to classify what is biodegradable and non-biodegradable through image analysis, object detection and classification with CNN. A different approach proposes the identification of different type of plastics to conclude whether the plastic can be recycled or not [15]. To achieve this, both visual and physical properties are used in the classification process. The authors of [16] suggest a “a multilayer hybrid deep-learning system” that can automatically identify waste through the usage of a CNN algorithm.

#### 2.2.2. Smart Trash Systems

Some studies propose systems that can receive trash, analyze and classify it based on features of the item. These systems can be sub categorized in different types.

### Reverse Vending Machines

This type of systems allows the end user to submit some waste, such as drinking cans or water bottles and receive back for example, a voucher or money. [17] presents an Internet of Things (IoT) implementation along with a CNN algorithm to be able to recognize plastic waste, three different architectures of CNN were tested. In [18] a design for a Bottle Recycling Machine to collect used bottles and classify them through the usage of a CNN algorithm can be found. The authors of [19] suggest an IoT implementation along with a CNN algorithm in order to create a more efficient Reverse Vending Machine.

This study tested four different architectures of CNN, as well as two different programming languages that were benchmarked in the implementation of the system.

### Smart Bins

This type of implementation refers to systems that introduce technology to the concept of trash bins in order to sort the trash in an intelligent way. The authors of [20] present a smart trash bin system to receive waste, classify it and place it automatically in the correspondent container by using a KNN algorithm. Another work presents an intelligent waste separator, dubbed “*Trashcan*”, with the purpose of replacing the recycle bins. This system can identify incoming waste and place it into different containers by using a KNN algorithm [21]. [22] proposes the design for a robotic trash bin that can locate, identify and classify trash into organic waste, non-organic and non-waste by using an SVM algorithm. Another study suggests an IoT smart waste segregation bin that with the help of sensors can identify and categorize waste that is deposited inside it by using a KNN algorithm [23].

### 2.2.3. Mobile Applications

A type of implementation suggested in the literature is applications for mobile devices that can harness their image and processing power. This is the type of implementation that is closer to what is purposed in this work, more details on how the work differs from these implementations will be specified in Chapter 3. [24] proposes an Android application equipped with a ML model, based on the MobileNet architecture, that allows for a photo to be taken with the camera and classify waste present in the photo into glass, paper, cardboard, plastic, metal or other trash. Another study proposes a smartphone app named “*SpotGarbage*” that is able to detect, geo-tag and classify garbage in an image, taken with the camera, by using a CNN algorithm [25]. The authors of [26] also propose a mobile application that allows citizens to report uncollected garbage. They convey that this application “*has been successfully deployed and has seen more than a million complaints registered across many Indian cities*”. This system uses a CNN algorithm to classify the images. In [27] the authors present a CNN architecture “*CompostNet*” with the purpose of classifying compostable, recyclable and landfill materials. A mobile

application for IOS was created as a proof of concept to show the capabilities of the architecture.

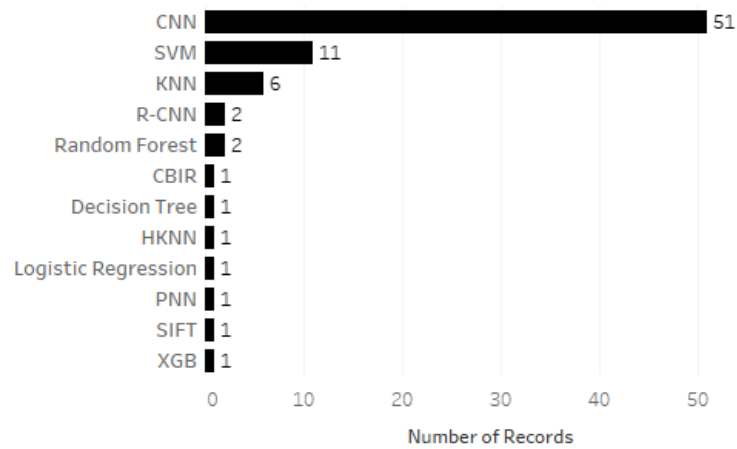
#### 2.2.4. Case Studies

In the literature survey several studies focused only in benchmarking of ML algorithms. However, the datasets generally used are specific to one study only. Such is the case with [28] that analyses the performance of two different algorithms in two different datasets from public and private environments. [29] tested several architectures of a CNN algorithm in order to understand the most efficient approach for the datasets. Another paper discusses the viability of optical identification of recyclable waste in construction and demolition sites and compares the SVM and Random Forest (RF) algorithms [30]. In [31] the authors benchmark multiple CNN algorithm architectures and propose an optimized deep convolutional neural network architecture to classify recyclable objects. One other study suggests the usage of an SVM algorithm for an intelligent sorting system for trash [32]. The authors of [33] propose the classification of garbage via image analysis through the usage of Scale invariant Feature Transform (SIFT) to extract the characteristics of the image garbage label. [34] suggest a method that is able to sort two types of materials, polycoat containers and Polyethylene Terephthalate bottles by using the SVM algorithm to classify the objects. One research describes a way of sorting polycoat containers from plastic bottles through the usage of image intensity data and a SVM algorithm [35]. In [36] the authors explore the SVM and CNN algorithms with the purpose of efficiently classifying garbage into six different recycling categories. Another work analyses multiple CNN algorithm architectures mixed with SoftMax and SVM classifiers to categorize garbage into different recycling types [37]. The author of [38] evaluates five different ML algorithms applied to the classification of trash for comparison. In a study the authors experiment with Transfer Learning and learning rates in a R-CNN algorithm with the objective of detecting waste objects in images [39].

### 2.3. Results from the Literature

Analyzing all the papers that reference ML algorithms usage for image classification applied to waste, we can synthesize this information and conclude that the CNN algorithm has a higher predominance with 51 entries followed by SVM with 11 as it can be seen in

Figure 2.2 which presents the distribution of the algorithm usage within the universe of papers that were reviewed. The number of entries is superior to the number of papers because some papers review multiple algorithms.



*Figure 2.2 – Algorithm Distribution*

Drilling down the analysis into the most used algorithm (CNN), we have a universe of different architectures as it can be seen in Figure 2.3, where the AlexNet architecture is the most predominant with eight entries followed by the MobileNet with five entries. AlexNet was introduced in 2012 and was a winner of the Imagenet Challenge [37]. MobileNet was introduced in 2017 as a class of efficient models for mobile and embedded vision applications [40]. For the analysis of the CNN architectures the entries where the architecture was not specified were excluded since they had no value for this specific analysis which was meant to identify the predominant architectures.

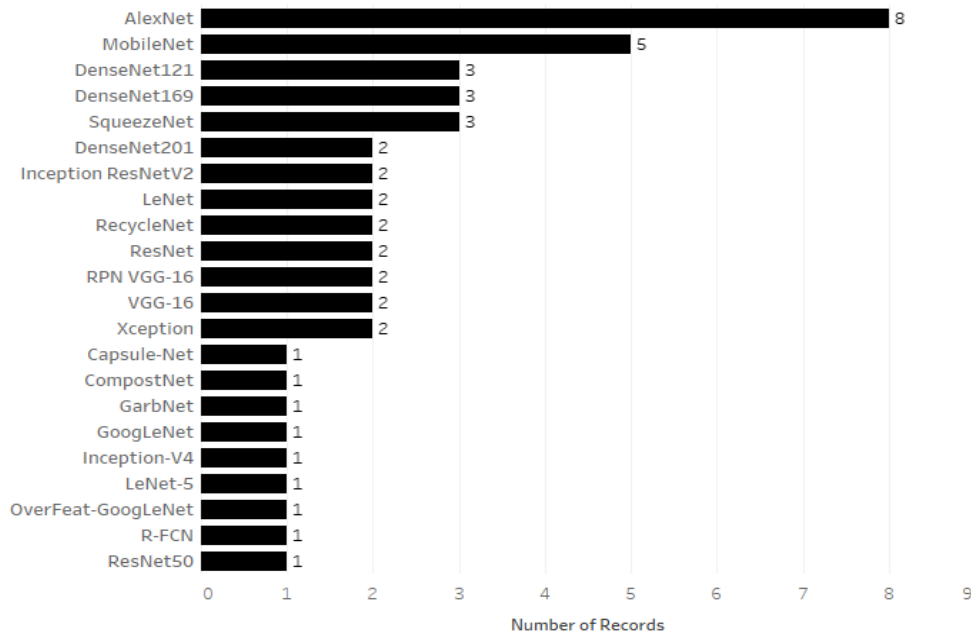


Figure 2.3 – CNN Architecture Distribution



Figure 2.4 – Dataset Distribution

It is not possible to objectively benchmark the majority of results present in the literature. While the predominant metric used to evaluate the results is Accuracy, surpassing by far any other measure. Only five studies used the same dataset (Trashnet) for testing their algorithms. The majority (Figure 2.4) uses datasets that either were left unspecified or were self-created for the study and with different image sizes. In the small universe of studies that used the Trashnet dataset and the same metric to evaluate results, therefore allowing for an objective benchmarking, the results (Figure 2.5) indicate that CNN algorithms yield the best results, with all the tested architectures surpassing the other algorithms. Within the CNN architectures universe, GoogleNet presents the best results with 97,86% accuracy, followed by VGG-16 with 97,46% and AlexNet with 97,23%. These accuracy results are specific to the classification of garbage in recycling categories.



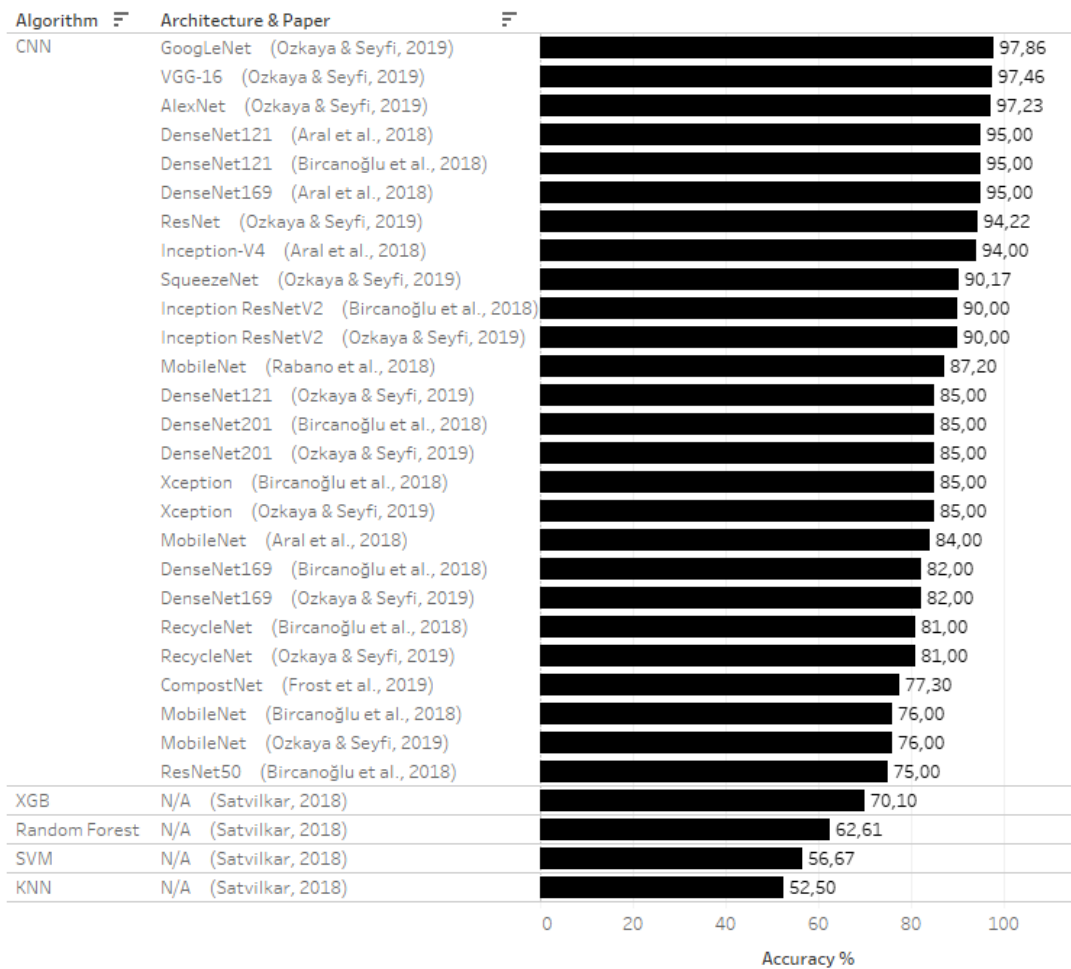


Figure 2.5 – Accuracy benchmarking results with Trashnet dataset

Table 2.3 presents a synthesis of the reviewed papers relevant information, that is related with ML algorithms used for image classification applied to waste. The “Algorithms and Results” column presents the ML algorithms used along with extra information, when available, such as the specific architecture, object detection algorithms, metrics used and results. The “Dataset” column defines the dataset that was used in the studies (when specified) along with its size and the images’ resolution.

Table 2.3 – Synthesized results from the literature

Paper	Algorithms and Results	Dataset
[9]	CNN (OverFeat-GoogLeNet architecture) - 63.2% Precision in cigarette butts' class - 77,35% Precision in leaves class	Self-created dataset with 25 different types of waste and 18.676 images at 640x480 pixels
[2]	RPN + CNN (VGG-16 architecture) - 3% False negative rate - 9% False positive rate	Self-created dataset with 1999 images at 600x1200 pixels
[14]	CNN (does not specify the architecture) - No results presented	Nonexistent dataset
[7]	SVM - 94.3% Accuracy	Self-created dataset with 500 images at 320x240 pixels
[28]	CNN (capsule-Net architecture) - 96% Accuracy	Self-created dataset with 19046 images at 256x256 pixels
[29]	CNN (multiple architectures) - DenseNet121 – 95% Accuracy - DenseNet169 – 95% Accuracy - Inception-V4 -94% Accuracy - MobileNet – 84% Accuracy	Trashnet dataset with 2527 images at 512x384 pixels
[24]	CNN (MobileNet architecture with transfer learning from model trained on ImageNet Large dataset) - 87.2% Accuracy	Trashnet dataset with 2527 images at 512x384 pixels
[41]	CNN (AlexNet architecture) - 83% Accuracy SVM (with bag of features) - 94.8% Accuracy	Self-created dataset with 2000 images at 256x256 pixels
[17]	CNN (multiple architectures) - LeNet – 93% Accuracy - AlexNet – 93% Accuracy - SqueezeNet – 87% Accuracy	Not specified
[8]	CBIR - No results presented.	Not specified

[18]	CNN (architecture not specified) - 96% Accuracy	Self-created dataset with 500 images with unspecified resolution
[3]	PNN - 98% Accuracy	Not specified
[20]	KNN - 98.33% Accuracy	Self-created dataset with 60 images with unspecified resolution
[16]	CNN (AlexNet architecture) - 98.2% Accuracy	Self-created dataset with 5000 images at 640x480 pixels
[21]	KNN - 98% Accuracy	Self-created dataset with 60 images with unspecified resolution
[30]	SVM - 98.7% Accuracy Random Forest - 97.8% Accuracy	Self-created dataset with 1000 images with unspecified resolution
[31]	CNN (multiple architectures) - ResNet50 – 75% Accuracy - MobileNet – 76% Accuracy - InceptionResNetV2 – 90% Accuracy - DenseNet121 – 95% Accuracy - DenseNet169 – 82% Accuracy - DenseNet201 – 85% Accuracy - Xception – 85% Accuracy - RecycleNet – 81% Accuracy	Trashnet dataset with 2527 images at 512x384 pixels
[32]	SVM - 97.3% Accuracy	Not specified
[10]	CNN (AlexNet architecture) - 77% Accuracy	Not specified
[33]	SIFT - 89.9% Accuracy	Self-created dataset with 192 images at 500x375 pixels
[34]	SVM - 92.85% Accuracy	Not specified
[35]	SVM - 96% Accuracy	Not specified
[22]	SVM - 82.7% Accuracy	Self-created dataset with 1000 images with unspecified resolution

[25]	CNN (GarbNet architecture) - 87.69% Accuracy	Garbage in Images (GINI) dataset with 2561 images with unspecified resolution
[4]	SVM - 94.7% Accuracy	Self-created dataset with 1446 images at 672x512 pixels
[15]	SVM - 75.6% Accuracy KNN - 95.1% Accuracy Decision Tree - 92.6% Accuracy Logistic Regression - 73.1% Accuracy	Not specified
(Yang & Thung, 2016)	SVM - 63% Accuracy CNN (AlexNet architecture) - 22% Accuracy	Self-created dataset with 2400 images at 384x384 pixels
[12]	R-CNN - 89.1% Precision	Not specified
[11]	CNN (LeNet-5 architecture) - No Results Presented	Not specified
[37]	CNN (Multiple architectures + SoftMax & SVM classifiers) - MobileNet – 76% Accuracy - Inception ResNetV2 – 90% Accuracy - DenseNet121 – 85% Accuracy - DenseNet169 – 82% Accuracy - DenseNet201 – 85% Accuracy - Xception – 85% Accuracy - AlexNet – 97.23% Accuracy - GoogLeNet – 97.86% Accuracy - ResNet – 94.22% Accuracy - VGG-16 – 97.46% Accuracy - SqueezeNet – 90.17% Accuracy - RecycleNet – 81% Accuracy	Trashnet dataset with 2527 images at 512x384 pixels
[26]	CNN (AlexNet architecture) - 85% Accuracy	Self-created dataset with 21000 images at 128x128 pixels

[38]	<p>CNN - 89.81% Accuracy</p> <p>SVM - 56.67% Accuracy</p> <p>XGB - 70,1% Accuracy</p> <p>RF - 62,61% Accuracy</p> <p>KNN - 52,5% Accuracy</p>	Trashnet dataset with 2527 images at 512x384 pixels
[5]	<p>KNN - 93% Accuracy</p>	Not specified
[6]	<p>HKNN - No Results Presented</p>	Not specified
[23]	<p>KNN - 99% Accuracy</p>	Not specified
[19]	<p>CNN (multiple architectures)</p> <ul style="list-style-type: none"> <li>- LeNet – 93% Accuracy</li> <li>- AlexNet – 93% Accuracy</li> <li>- SqueezeNet – 87% Accuracy</li> <li>- MobileNet – 88% Accuracy</li> </ul>	Not specified
[13]	<p>CNN (multiple architectures + detectors)</p> <ul style="list-style-type: none"> <li>- VGG-16 with SSD detector – 56% Precision</li> <li>- R-FCN – 53% Precision</li> <li>- YOLO - 40% Precision</li> </ul>	Trashnet dataset with 2527 images at 512x384 pixels
[27]	<p>CNN (CompostNet architecture)</p> <ul style="list-style-type: none"> <li>- 77.3% Accuracy</li> </ul>	Trashnet dataset with 2527 images at 512x384 pixels
[39]	<p>R-CNN - 81,6% Precision</p>	Trashnet dataset with 2527 images at 512x384 pixels

Some conclusions can be taken from the 39 papers that were validated as relevant for the topic and were thoroughly analyzed in this review.

One of the most important conclusions to highlight is the fact that some studies present self-built datasets, with different image sizes and categories, while others use datasets

available in the community (Trashnet and GINI), making it extremely difficult to compare approaches. With regards to the ML algorithms used, it can be concluded that Neural Networks and SVM are the types of algorithms most used in this field of study. A recurring factor, the inconsistency between results, even considering the same algorithm, is highlighted by multiple studies, with the authors concluding that further work using more available data and real-life scenarios is needed to support their findings.

Next, regarding the used datasets, the studies can be divided into four categories: Self-created dataset (15 papers), dataset Not specified (15 papers), GINI (1 paper) and Trashnet (8 papers). For the latter, some kind of benchmark can begin to be achieved since the same dataset was used for all the eight approaches. In this last scenario, the CNN algorithm with a GoogLeNet architecture was the one presenting the best results.

It was concluded from the literature review that the most appropriate ML algorithm to be used in this project is the MobileNet. Even though it isn't the most accurate as highlighted by the results from the literature in Figure 2.5, it was designed to effectively maximize accuracy while dealing with the restricted resources of a mobile device.

During the development of this work a journal paper with the synthetization of the literature analysis was submitted to a journal, this will be useful for further researchers that are interested in the field as it works as a compendium of the information available until the point in time when it was written. As future work it is important to keep updating this study with all new scientific work that gets released, and to merge all the available datasets into one that can be used by multiple researchers to further their studies in the field and enable proper benchmarking between studies. It would also be helpful for further researchers to benchmark the speed of classification of the algorithms.

## Chapter 3 – MobileNet

The MobileNet architecture has been introduced as a sub class of CNN and are specialized in efficient models for mobile and embedded vision applications. As seen in Chapter 2, within this area, MobileNet seems to be the most adequate for image recognition and classification. As such, this will be the modelling choice for this dissertation and this chapter is dedicated to defining the ML modelling technique that will be used in this project.

CNN is a widely used class of ML models for image classification, having been recognized in the 2012 ImageNet competition which made it one of the most promising ML techniques [42]. This type of class makes strong and generally correct assumptions about the nature of images while having fewer connections and parameters which makes them easier to train compared to standard feedforward neural networks [43]. One of the downsides of this type of architecture is the high demand for processing when dealing with high resolution images. Graphical Processing Units (GPU) help accelerate the training of CNN models, with many vendors like NVIDIA creating hardware and software specially for that [42]. A CNN is constituted by several types of convolution layers. A CNN has less parameters than a classic neural network due to the fact that the latter use fully connected layers exclusively while the former have each neuron connected to only a few neurons.

A convolution is a linear transformation that has the purpose of extracting useful features from an input [44]. A convolutional layer combines the extracted features into feature maps as shown in Figure 3.1.

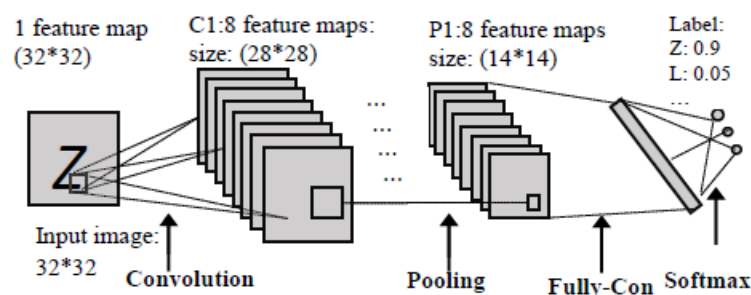


Figure 3.1 – The structure of an example CNN [42]

The convolution layer can be defined by the following parameters:

- **Kernel:** This parameter defines the field size (number of pixels, e.g. 3x3) in a convolution.

- **Stride:** This parameter refers to the step size of the Kernel when processing an image. The number means the number of pixels it will stride in each step.
- **Padding:** This parameter defines the way it will handle the borders of a sample.
- **Channels:** This can refer to Input and Output, as the layer takes a number of input channels to calculate a determined number of output channels.

The MobileNet architecture is “based on a streamlined architecture that uses depthwise separable convolutions to build light weight deep neural networks” [40] and it can be seen in Figure 3.2.

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
5× Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool $7 \times 7$	$7 \times 7 \times 1024$
FC / s1	$1024 \times 1000$	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Figure 3.2 – MobileNet body architecture [40]

A depthwise separable convolution factorizes a standard convolution into a depthwise convolution, which applies a single filter to each input channel, and a pointwise convolution, which combines the outputs of the depthwise convolution. An example of the factorization can be seen in Figure 3.3 where (a) corresponds to a standard convolution and (b) corresponds to a depthwise convolution and (c) corresponds to a pointwise convolution. This factorization reduces the computation needed and the model size [40].



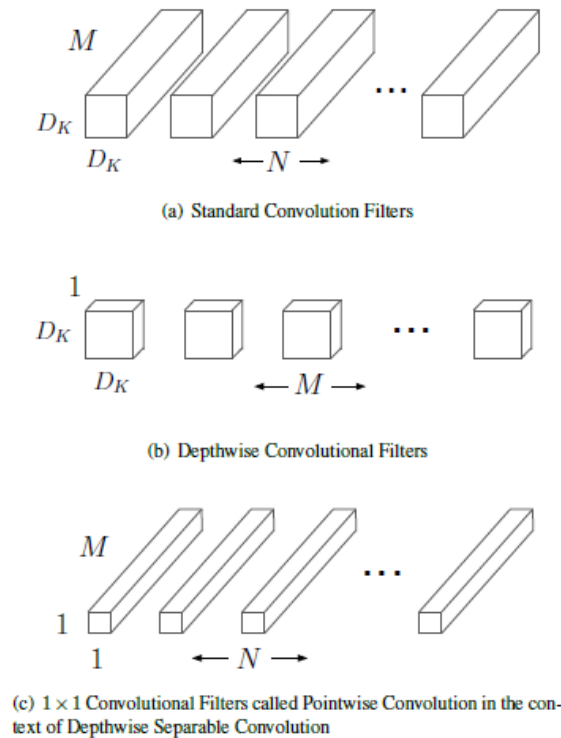


Figure 3.3 – Factorization of standard convolution [40]

Almost all convolutional layers in the architecture are followed by a batch normalization and a rectified linear unit (ReLU) as it can be seen in Figure 3.4.

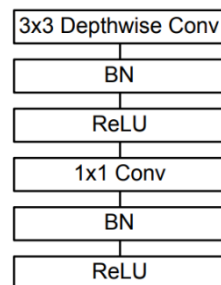


Figure 3.4 – Representation of MobileNet layer except the last layer [40]

Batch normalization standardizes the inputs to a layer for each mini batch. This stabilizes the learning process and reduces the number of training epochs required to train deep networks. An epoch is a training iteration. In each epoch a different model is built with a different set of weights. A neural network will build  $N$  different models for  $N$  epochs. Each model created for a corresponding epoch should be validated in generalization performance. A higher epoch number does not necessarily mean a better generalization performance [45].

The ReLU is “the standard way to model a neuron’s output  $f$  as a function of its input  $x$ ” [43]. It is a unit that uses an activation function that outputs the input directly if it is

positive, else it will output zero. The function used in the unit can be considered both linear, for value greater than zero, and nonlinear as negative values always output as zero.

The last layer in this architecture is not followed by batch normalization and ReLU like the previous ones and it is a fully connected layer. The purpose of this layer is to flatten the feature maps into a two-dimension matrix, or vector. This layer performs classification taking into account the features extracted by the previous layers [42]. This fully connected layer feeds into a softmax layer which is the final layer of the network. It's main purpose is classification and it achieves that through the computation of the possibility distribution over different labels [42].

The architecture also uses a parameter  $\alpha \in (0, 1]$ , width multiplier, with the purpose of thinning the network uniformly at each layer and reducing the computational cost, this means that for a given layer and width multiplier  $\alpha$ , the number of channels in the input  $M$  becomes  $\alpha M$  while the number  $N$  becomes  $\alpha N$ .

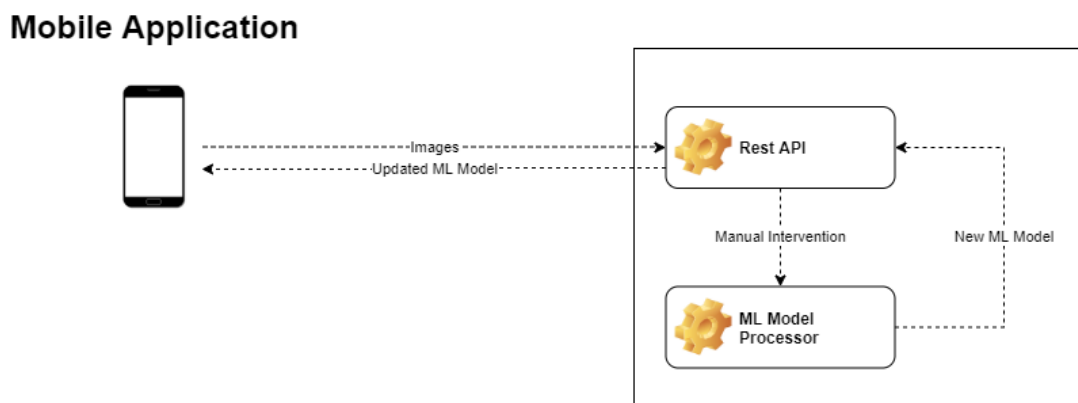
Another parameter used is the resolution multiplier  $p \in (0, 1]$ , this is used to also reduce the computational cost of the neural network. When this parameter is applied to the input image, the internal representation in each layer is reduced, this happens because the input resolution is implicitly set by the resolution multiplier.

At the time of the development of this project, three versions of this architecture exist. In theory each iteration of the algorithm increases its accuracy. In this project the v2 of the algorithm will be used due to limitations that will be explained in Chapter 5.

## Chapter 4 – High-level Architecture

As stated in Chapter 1, the objective of this dissertation is to create a system that can help people to recycle via a mobile application installed in a smartphone. This application will utilize some features described in the literature surveyed in Section 2.2.3. For example the application itself is similar to the one in [24], but it will add extra features that will distinguish it such as the capacity to upload the images to a server and that these images can be utilized to potentially create better performing ML models, thus capable to adapt or learn with the types of waste that is presented in the universe of images that are captured with the application and that might be either slightly different or completely new from the previous dataset. This type of crowdsourcing feature in a mobile application that classifies waste seems to be unique in the literature and the potentially long-life learning system makes it also new within the application for recycling realm.

The application should be able to capture an image, analyze it and classify it according to a recycling category in just a few seconds. The classification should be based in specific recycling categories. The mobile application can also communicate with a REST API to (1) update the ML model when there is a new version available, (2) send images in order to help reinforce the ML model in the application. This reinforcement should not be automated and requires manual classification of the images by an operator that is specialized in the categorization of waste. The system is composed by three main modules as described in Figure 4.1.



*Figure 4.1 – High-level system architecture*

The mobile application is the part of the project that the users will interact with. The main features of the application are:

- Capture images and store them in the smartphone storage.
- Load images previously stored within the smartphone storage.
- Classify images by waste category and returns the result to the user.
- Communicate with the REST API to send the images that were captured with the application.
- Communicate with the REST API to update the ML model that is installed in the application, if there is a newer model.

The mobile application module will be further explained in Chapter 7.

The REST API is the module that is responsible with the communication with the mobile application. It should be able to:

- Authenticate users that are pre-registered in the system.
- Receive images from the mobile application.
- Send new versions of the ML model to the mobile application when requested.

Further information for this module will be explained in Chapter 6.

The ML model processor module is responsible for training new ML models. The transition of images that were received in the REST API to this module is not automated and requires human intervention. The required human steps for this module are to transfer any new images received from the mobile application and merge them into the dataset already available to him. Images should be categorized properly in this step. With this enriched dataset it is responsible for processing newer versions of the ML model, comparing it to previous versions and decide which version should be available for the REST API. The newer versions of the ML model should be compared through specific measures in order to validate the overall capacity of identifying waste materials. Only the versions with better results should be available for the REST API to propagate to the mobile application.

For a ML model to be accurate there is the need for training it with images of recyclable and non-recyclable materials, the more images the better as it is expected that the model will be able to generalize the results better. For this project two sources were used as the main sources of images for the training. The first dataset was Trashnet<sup>4</sup> which is

---

<sup>4</sup> Website link: <https://github.com/garythung/trashnet>

composed by 2527 images at 512x384 pixel resolution. The images are segregated in six categories, cardboard, glass, metal, paper, plastic and trash. The distribution of images in each category is presented in Table 4.1.

*Table 4.1 – Trashnet dataset details*

Category	Number of Images	Definition
Cardboard	403	Class containing only cardboard items. Cardboard differs from paper because the former is a hard material and has multiple layers of types of papers.
Glass	501	Class containing glass items some examples are bottles and jars.
Metal	410	Class containing metallic items such as cans for example.
Paper	594	Class containing only paper items. This differs from the cardboard class because it only has a single layer of paper.
Plastic	482	Class containing plastic materials. Plastic bottles and containers are some examples of this class.
Trash	137	Class containing non-recycling waste.

It is important to denote that the Trashnet dataset was created in a laboratory environment, where the background of the images is white even though there is some illumination variance. Some examples of images present in each category can be seen in Figures 4.2 to 4.7.



Figure 4.2 – Cardboard category image samples



Figure 4.3 – Glass category image samples



Figure 4.4 – Metal category image samples



Figure 4.5 – Paper category image samples



Figure 4.6 – Plastic category image samples



Figure 4.7 – Trash category image samples

The second dataset was created during this project to test the results in a real-life scenario. A total of 290 images were captured with the mobile application and their distribution within the Trashnet original classes can be seen in Table 4.2.

Table 4.2 – Real life images dataset distribution

Category	Number of Images
Cardboard	25
Glass	22
Metal	40
Paper	78
Plastic	108
Trash	17

These images were taken in different backgrounds, angles and distance to try to simulate how a user of the application would take a photo. Some examples of these images can be seen in Figure 4.8.



Figure 4.8 – Image samples for real life purpose test





## Chapter 5 – ML Model Processor

Even though the objective of this project is to create a prototype system that will evolve over time with the images provided by the users of the mobile application, it is still required to have an initial state where a ML model that has been trained can classify some images. This chapter will focus on the description on how the ML model processor does the training of the first model that will accompany the prototype system at launch. Two types of architecture were trained and tested. These types correspond to the iterations of the mobile application. The first iteration will be described in Section 5.1 and the architecture used was MobileNet v2 without any object detection technology. This iteration was to test if the proposed system would work. The second iteration will be described in Section 5.2 and it was created to further help the possible users of the system. An SSD MobileNet v2 architecture was used, which extends the first iteration with object detection technology. The final objective is to use an object detection architecture because the user might capture images in a real-life scenario where more than one object may appear in the image.

Both implementations used transfer learning. This technique uses models that were pre-trained in a dataset. These models are then trained with another dataset, resulting in changes in the architecture of the model. The weights, or learned features, from the pre-trained model are transferred to this new model, instead of training the model from scratch [24]. A pre-trained model might not be 100% accurate in its current application but in theory it shortens the training time and might yield a higher accuracy [37]. This will avoid excess training time which will be useful due to the limited dataset size and available computing power to train the models. Fine-tuning, an approach to transfer learning, was also used. There are multiple ways of realizing fine-tuning. One of the approaches is to optimize all the parameters of the pre-trained model on a training dataset. Alternatively, only some layers can be fine-tuned, the last few ones, while the parameters from the initial layers can be frozen. The theory behind the latter approach is that the initial layers of a neural network learn low-level features that are common to many computer vision challenges [46].

The dataset used in both implementations for training and first tests was the Trashnet dataset. The second dataset of real-life images was used to test the models in a real-life scenario.

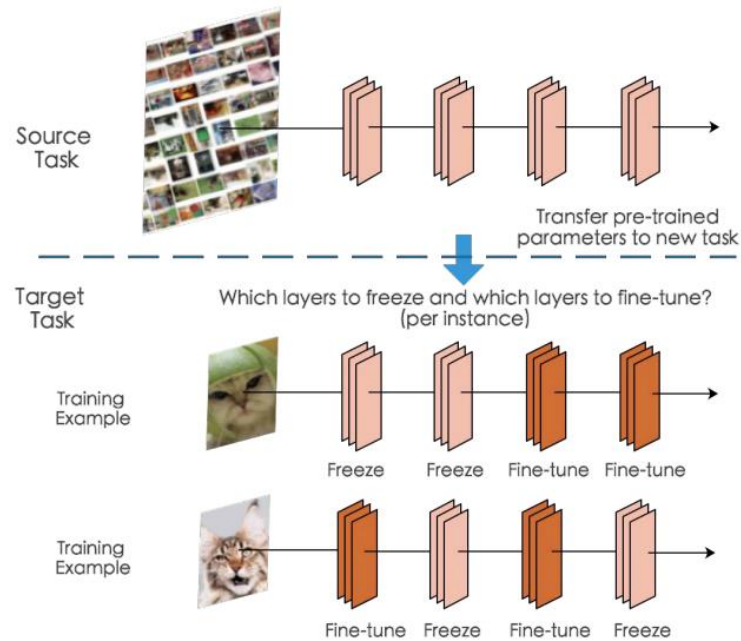


Figure 5.1 – Example of fine-tuning [46]

The hardware used for the training of the two models was: Intel Core i5-6600 CPU @ 3.30 GHz (4 CPUs) processor, Kingston 16 GB DDR4-2400 of Memory Ram, Samsung 850 EVO 250GB Solid State Drive disk and a NVIDIA GeForce GTX 970 graphic card.

### 5.1. MobileNet v2

In this implementation, a MobileNet v2 architecture was used. This type of architecture does not support any object detection and only allows to classify the entirety of an image without segmenting it.

This module was built in Python and Keras<sup>5</sup>, an open-source neural-network library written in Python, capable of running on top of TensorFlow. This library allows to quickly create code to train and test ML models. The version of Python used was 3.6. The deep learning API Keras version was 2.2.4. Since there was a GPU with dedicated memory available for this project, the Tensorflow-GPU library was used, this allows to accelerate the training of ML models when compared to training in a CPU.

<sup>5</sup> Website link: <https://keras.io/>

### 5.1.1. Methodology

A single python file “train.py”<sup>6</sup> was created to implement the training of the model, this implementation can be categorized in five major steps.

The first step is to load the images that are to be used in the training, these images are resized into 224x224 pixels format which is the maximum size this model architecture accepts in the Keras implementation when using transfer learning with Imagenet<sup>7</sup> weights. Imagenet is an image database of more than 14.000.000 images and more than 80.000 classifications. Transfer learning was used to accelerate the training of the ML model. The dataset is then split into a training set, with 80% of the images. For the resulting 20% the same split is applied again, dividing it into 80% for a validation set and 20% for a test set.

The second step involves applying standardization and normalization to the images in order to prepare the data to be consumed by the classifier in the next step. The method of standardization means that the features of the data are being transformed in such manner that they have a mean of zero and a standard deviation of one. This type of data transformation is useful when the data has differing scales. Data normalization is the rescaling of the original data so that all values are normalized within a range of 0 and 1.

In the third step, the type of architecture is defined. In this case the MobileNet v2 architecture is defined using a preconfigured architecture available in Keras, this speeds up the coding time by avoiding the creation of a full architecture from zero. The classifier is defined as well using a softmax layer, which is the default classifier for this architecture. An optimization algorithm is used as well, the Adam optimizer which is used to update network weights iteratively based in training data [47]. This optimizer was used because it reported higher accuracy in the literature [29]. Since the classification in this project is multiclass, the type of loss to be computed is the categorical cross entropy. This computes the cross entropy loss between the labels and predictions. [48].

The fourth step is to prepare a data augmentation generator. This is an important technique because by artificially increasing the number of training examples it can help preventing overfitting in neural networks and improve performance in imbalanced datasets [49]. Image altering augmentations were used randomly, such as rotating,

---

<sup>6</sup> Github link: [https://github.com/antoniosequeira/trainer\\_mobilenet\\_v2/blob/main/train.py](https://github.com/antoniosequeira/trainer_mobilenet_v2/blob/main/train.py)

<sup>7</sup> Website link: <http://www.image-net.org/>

applying zoom, shifting in terms of width and height, shearing transformation and flipping horizontally. What the generator does is for each epoch it provides slightly altered images based on the original images, so that each epoch model will be trained on the same number of images than the previous epoch but these images are slightly different.

In the fifth and final step, the model is trained. An option of saving the best checkpoint based on the validation loss metric is used, this means that when a minimum loss when predicting the validation set is found, a checkpoint is created. The best checkpoint is the model to be used in predicting the test dataset.

For the training, a batch size of 12 was used due to memory limitations as increasing the size led to memory errors. The maximum number of epochs was 500 but as explained in the previous paragraph, due to the usage of the best checkpoint it might mean that the best model (with the minimum validation loss) was created before the 500<sup>th</sup> epoch. For each epoch it is important that the training uses all the available training data, to achieve this the number of steps for each epoch is calculated according the formula steps:

$$\text{steps} = \text{number of images} / \text{batch size} \quad (1)$$

### 5.1.2. Results

The training of the model was achieved in 5 hours with the available hardware and software version. The results of the training set and validation set can be seen in Figure 5.1 and 5.2 where the accuracy and loss metrics are measured. The accuracy metric calculates how often predictions equal labels, while the loss metric is the result of a function that computes the cross-entropy loss between the labels and the predictions. The former metric is best when it is higher while the latter is best when is lower. The results in this part are using the accuracy metric because due to a limitation on the version of Tensorflow that was being used, only this metric was available.

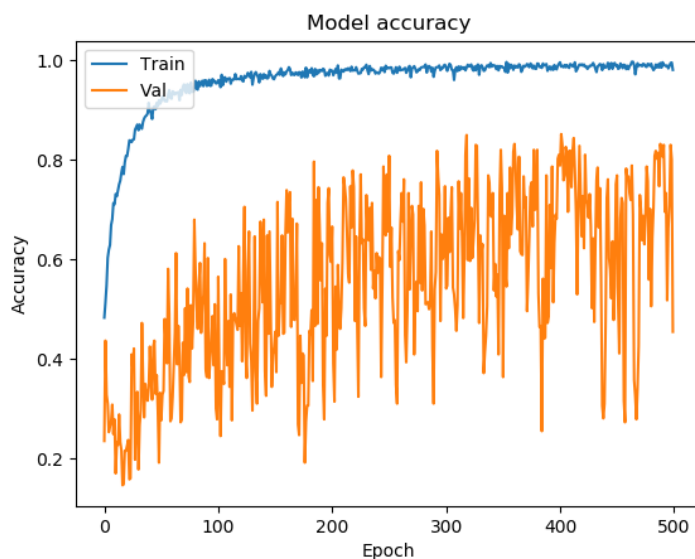


Figure 5.2 – Model training accuracy

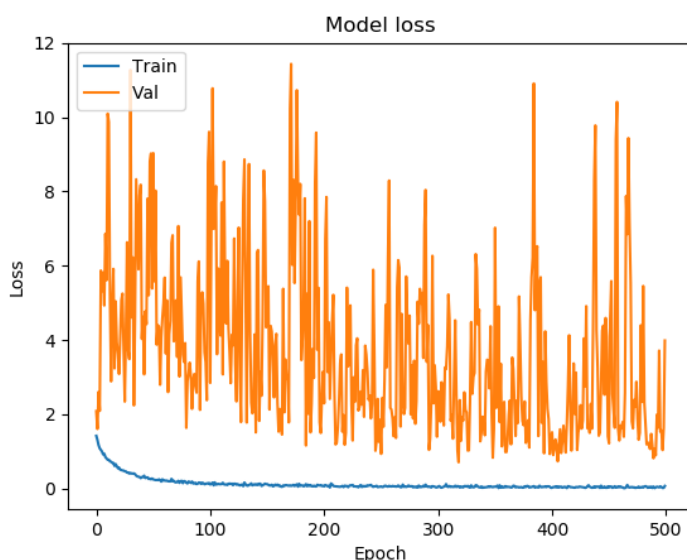


Figure 5.3 – Model training loss

Due to the usage of the best checkpoint as specified in Section 5.1.1, the model was saved in the epoch 318 where the validation loss achieved the lowest number of 0,7072. The corresponding validation accuracy for this epoch was 84,98%. This loss and accuracy are averages for all classes. Both Figure 5.2 and Figure 5.3 show that the validation results are varying frequently, and it doesn't stabilize. These results indicate that the model might have problems solving the classifications correctly.

Testing the trained model in the test set we have the following predictions against the ground truth in Table 5.1. For a deeper analysis, a confusion matrix was created. This

type of table layout allows for the visualization of the performance of the model. It is comprised of two dimensions, the “Ground Truth” which is the original classification of an object in an image, and the “Predicted” which is the prediction generated by the model. Each dataset class is under both dimensions.

*Table 5.1 – Test dataset results confusion matrix with Mobilenet v2*

		Predicted					
		cardboard	glass	metal	paper	plastic	trash
Ground Truth	cardboard	84	0	0	0	0	0
	glass	1	96	2	2	1	0
	metal	0	1	83	2	1	0
	paper	2	0	1	115	0	0
	plastic	1	0	4	1	85	0
	trash	0	0	1	2	0	21

The precision metric was used for a better perception of the results. Even though this implementation doesn’t have any object detection, this metric seems to be the most used metric in object detection problems as it will be seen in Section 5.2. Due to this fact the same metric will be used in this implementation. This metric is calculated as:

$$\text{Precision} = \text{Number of correctly predicted images} / \text{All images of that class} \quad (2)$$

The precision result for each class can be analyzed in Table 5.2. Cardboard was the class with the highest precision score. Trash was the class with the worst result, even though the precision value was very high.

*Table 5.2 – Precision results for Test dataset by category with Mobilenet v2*

Category	Precision
cardboard	100,00%
glass	94,12%
metal	95,40%
paper	97,46%
plastic	93,41%
trash	87,50%
<b>Average</b>	94,65%

The results on the test set show that the precision value for each class is very high, with one the classes achieving a perfect score. This test was done in images from the same dataset. Even though the images used to test the model were not used to train the model, they were from a similar source. This might create situations of overfitting, where the

model is good at predicting a certain dataset but is not good at generalizing. To test if the model is good at generalizing a new test will be done with images of waste captured with a smartphone during the development of this project and that was described in Chapter 4.

The results of the test in the real dataset (Table 5.3) reveal that the model was not capable of predicting most of the images correctly. With half of the classes having zero correct predictions.

*Table 5.3 – Real dataset results confusion matrix with Mobilenet v2*

		Predicted					
		cardboard	glass	metal	paper	plastic	trash
Ground Truth	cardboard	22	0	0	1	2	0
	glass	7	0	0	8	7	0
	metal	22	0	0	12	5	1
	paper	40	0	0	30	8	0
	plastic	45	0	0	37	25	1
	trash	9	0	0	5	3	0

The precision metric was used to evaluate the results of the predictions. The result of this metric for each class can be analyzed in Table 5.4.

*Table 5.4 – Precision results for Real dataset by category with Mobilenet v2*

Category	Precision
cardboard	88,00%
glass	0,00%
metal	0,00%
paper	38,46%
plastic	23,15%
trash	0,00%
<b>Average</b>	24,94%

The precision metric results are all lower in the real dataset when compared to the test dataset. Only the cardboard class had a closer result while the rest of the classes had very bad results. This might mean that the model is overfitting and cannot generalize properly. As reported by [29] the small amount of data and the white background of images in the Trashnet dataset might be causing the poor results in the Real dataset because the source of the data for both datasets is different. One possible explanation for the fact that the model is classifying a big portion of the items as cardboard might be due to the environment where the images were captured. A wooden board is in the background of a

majority of the images from the Real dataset, and the color and texture in low light is similar to the color and texture of the cardboard. Another explanation for such results is human error in the classification of the Real dataset images due to misinterpretation of the original Trashnet dataset classes.

### 5.1.3. Converting to mobile

The final version of the mobile application only uses the model produced in Section 5.2. The code for a version of the mobile application that uses the type of model produced in Section 5.1.1 is linked in the Section 5.1.4. The result ML model was used as in the first iteration of the mobile application in the same way as it was previously explained in Chapter 4 and demonstrated in Figure 4.1, which represents the high-level system architecture for the final version of the system. This model would be downloaded in the mobile application at the installation and each new updated ML model created will be available for download using the REST API. However, the output of the Keras framework is not compatible with the framework Tensorflow Lite<sup>8</sup> which is the framework to run ML models in a smartphone. A conversion of the output model is required in order for the model to be run in a mobile environment. An extra step is required to convert the trained model into a format that can run in a smartphone application that uses Tensorflow Lite.

The process to convert a Keras ML model is to use the Tensorflow lite converter function to convert the chosen ML model into tflite format. For this step, the python file “liteconverter.py”<sup>9</sup> is used.

### 5.1.4. Repository

All the code produced in this module was versioned within GitHub the repository is public and under MIT license in the following link:

[https://github.com/antoniosequeira/trainer\\_mobilenet\\_v2](https://github.com/antoniosequeira/trainer_mobilenet_v2).

[https://github.com/antoniosequeira/mobile\\_application\\_old](https://github.com/antoniosequeira/mobile_application_old).

---

<sup>8</sup> Website link: <https://www.tensorflow.org/lite>

<sup>9</sup> Github link:

[https://github.com/antoniosequeira/trainer\\_mobilenet\\_v2/blob/main/liteconverter.py](https://github.com/antoniosequeira/trainer_mobilenet_v2/blob/main/liteconverter.py)



## 5.2. SSD MobileNet v2

For this implementation, an SSD MobileNet v2 architecture was used. This architecture incorporates an object detection algorithm called single shot detector (SSD) which allows to detect multiple objects in an image and classify them distinctly. The SSD architecture is a convolution network that learns to predict bounding box locations and their classification in one pass. The early network layers use a standard architecture, in this case we are using MobileNet v2 architecture. These early layers are followed by several convolution layers that decrease in size progressively and allow predictions of detections at multiple scales [50].

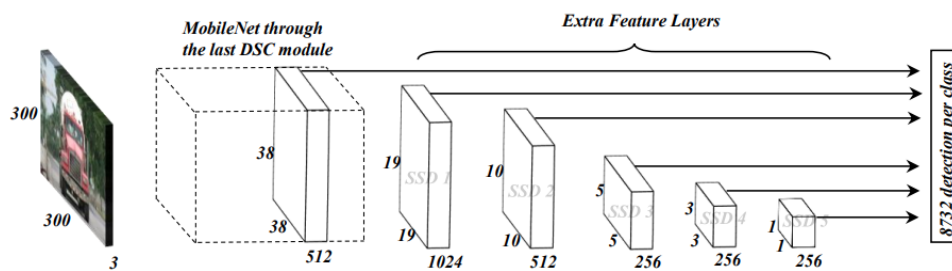


Figure 5.4 – Example of SSD MobileNet architecture [51]

This type of architecture usually assumes the name of the early network layers standard architecture and the SSD, so in this case the full name of architecture is SSD MobileNet v2.

The dataset that was used for this architecture was Trashnet but now an annotated version was used due to the object detection technique. For this type of architecture, the techniques of transfer learning and fine-tuning were used to try to reduce the amount of time required to train the model. The pre-model used for transfer learning was the `ssd_mobilenet_v2_coco`, obtained from the Tensorflow 1 Detection Model Zoo<sup>10</sup>. As stated before, due to GPU incompatibilities, version of 1 of Tensorflow is being used in this project, otherwise the Tensorflow 2 Detection Model Zoo<sup>11</sup> could have been used. This model is an object-detection model pre-trained on the Common Objects in Context (COCO) dataset. This specific model was used to perform transfer learning because in the original dataset contains much of the objects we are trying to classify in our

<sup>10</sup> Website link:

[https://github.com/tensorflow/models/blob/master/research/object\\_detection/g3doc/tf1\\_detection\\_zoo.md](https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf1_detection_zoo.md)

<sup>11</sup> Website link: <https://cocodataset.org/#home>

implementation. Additionally, fine-tuning was applied to all the parameters of the pre-trained model as no layer was chosen to freeze parameters.

Even though it would be interesting to compare results between implementations, it was not possible because the implementation and configuration differ. Keras provides a high-level API written on top of the Tensorflow backend, this means that the former provides less options than the latter. One of those limitations is the lack of out-of-the-box object detection for the MobileNet v2 architecture. In theory, it is possible to develop such a project but due to time constraints it was not possible to do so within this project. For example, the Keras implementation with transfer learning only allowed for a max size of image for training to be 224x224 pixels, while the Tensorflow object detection API<sup>12</sup> did not have any limitations for the size of the images when using transfer learning. A limitation that is out of the scope of the framework differences is that, since the latter used a pre-trained model in object detection for transfer learning, which is not available in the same format for the former, comparison between both trainings results is not possible since each one used a different source for the transfer learning technique.

The present MobileNet implementation can be viewed as an evolution from the one presented in Section 5.1, as the former is simpler and with sole focus on the classification of images, while this one not only classifies images, but also adds object detection into the picture, which is an entirely different challenge.

To avoid programming all code from scratch, the TensorFlow object detection API was used. This API is an open source framework allows the creation of ML models capable of locating and identifying multiple objects in a single image [52]. Even though, by the time of the development of this project, a version 2 of this API is generally available, due to limitations with the available GPU, this project still used version 1 of the API. The programming language used was Python.

### 5.2.1. Methodology

This methodology is very different from the previous one, because it revolves around the Tensorflow object detection API. Multiple Python files were used in this implementation, along with additional software.

---

<sup>12</sup> Website link: <https://github.com/tensorflow/models>

The first step is to annotate the images of the dataset with the location of the objects and respective categories. This was achieved with the LabelImg graphical image annotation tool [53]. The annotation of an image requires the selection of an area in the image, called a bounding box, and manually label that bounding box. This is a time-consuming activity because, for datasets that have not been previously annotated, or where the annotations are not fit for the problem being solved, this is a completely manual task. The information is saved in Extensible Markup Language (XML) files in Pattern Analysis, Statistical modelling and Computational Learning (PASCAL) Visual Object Class (VOC) format [54].

The second step is to partition the dataset into training, validation and test datasets. For this action, the “dataset\_partition.py”<sup>13</sup> file was used for each class. The first round of partitions creates the training set using 80% of the images. The second round of partitions uses the remaining 20% and splits it into the validation set (80%) and the test set (20%).

The third step requires the transformation of the XML files of the first step into Comma-Separated Value (CSV) files since the next step only accepts CSV formats. The conversion from XML to CSV is done by the xml\_to\_csv.py file. A CSV file is required for each dataset, in this case three files will be required.

The fourth step creates a tfrecord<sup>14</sup> file based on the CSV files of the previous step. This tfrecord file is the format required by the object detection API to train the model. This format stores a sequence of binary records. The conversion is done in the “generate\_tfrecord.py”<sup>15</sup> file. A tfrecord file is required for each dataset, in this case three files will be required.

The fifth step is the configuration of the pipeline file. This file is the main configuration for the API to run the training of the model. Information such as the location of the training and validation set, batch size, number of steps, data augmentation to be applied are configured here. This file was downloaded from the same source as the pre-trained model used for transfer learning and fine-tuning.

---

<sup>13</sup> Github link:

[https://github.com/antoniosequeira/trainer\\_ssd\\_mobilenet\\_v2/blob/main/dataset\\_partition.py](https://github.com/antoniosequeira/trainer_ssd_mobilenet_v2/blob/main/dataset_partition.py)

<sup>14</sup> Definition link: [https://www.tensorflow.org/tutorials/load\\_data/tfrecord](https://www.tensorflow.org/tutorials/load_data/tfrecord)

<sup>15</sup> Github link:

[https://github.com/antoniosequeira/trainer\\_ssd\\_mobilenet\\_v2/blob/main/generate\\_tfrecord.py](https://github.com/antoniosequeira/trainer_ssd_mobilenet_v2/blob/main/generate_tfrecord.py)

The sixth and final step is the training of the model. The script that runs the training is the “model\_main.py”<sup>16</sup>. The results of the training and validation can be monitored using Tensorboard<sup>17</sup> which provides the visualization and tooling needed for ML experimentation. Tensorboard is installed by default when Tensorflow is installed in the computer.

For training, due to memory limitations, a batch size of 12 was used. In fact, when trying to use a higher batch size, the object detection API reported out of memory errors. In this type of training the previous concept of epoch is equal to the number of steps in this training. The number of steps used was 100000, even though the default number configured in the pre-model pipeline was 200000, after a few tests it was noted in the results (Section 5.2.2) that with less than half this number the results do not significantly change afterwards.

In terms of data augmentation, the object detection API differs from the previous framework offering a few different options. For this implementation only two data augmentation types were used, the ones that were set as default with the pipeline configuration file of the pre-trained model. The types are `random_horizontal_flip`, which randomly flips inputs horizontally, and `ssd_random_crop` that randomly removes part of the images’ outer areas.

### 5.2.2. Results

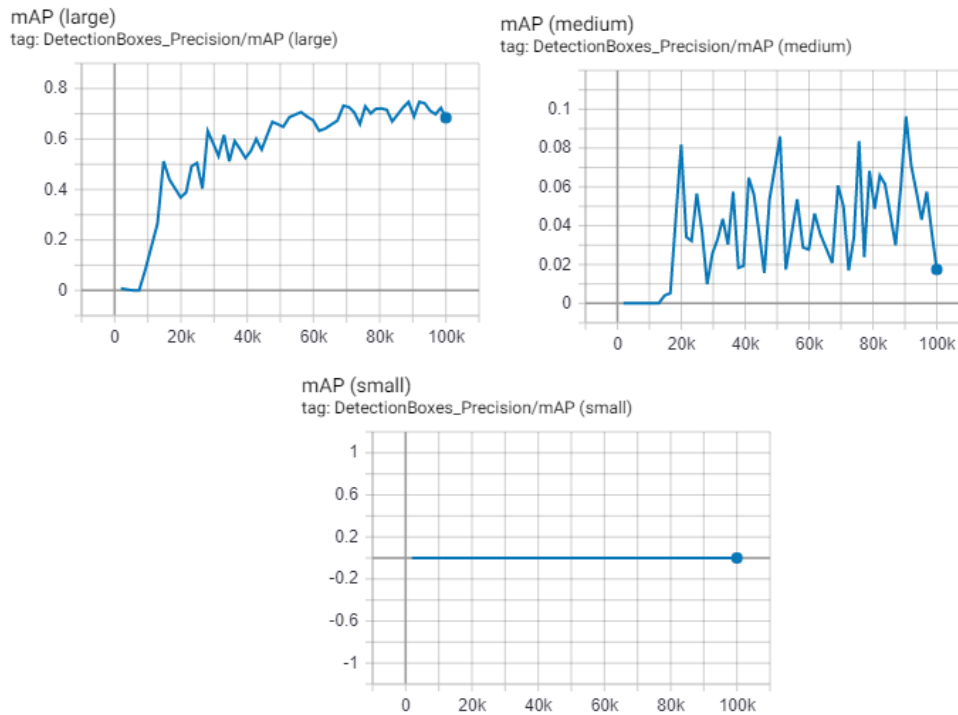
During training of the model, the results were monitored in Tensorboard. This tool allows for the visualization of the different detection evaluation metrics used by COCO. The first available metric is mean average precision (mAP) [54]. This metric represents the average precision averaged over all classes. It is important to note that this implementation has a limitation: the API does not have an option by default to save the best checkpoint or step in the training. After several tries, it revealed to be hard to program a way to do this due to the lack of time and experience in the framework, so it was decided to skip it. This means that the trained model will correspond to the final step of the training and this will be the one to be analyzed.

---

<sup>16</sup> Github link:

[https://github.com/antoniosequeira/trainer\\_ssd\\_mobilenet\\_v2/blob/main/model\\_main.py](https://github.com/antoniosequeira/trainer_ssd_mobilenet_v2/blob/main/model_main.py)

<sup>17</sup> Website link: <https://www.tensorflow.org/tensorboard>



*Figure 5.5 – mAP results from Tensorboard*

The mAP results are divided into three different graphics. Each graphic represents the average precision for different sized objects. In Figure 5.5, from left to right and top to bottom, the first graphic presents the mAP for large objects, that is, bigger than 96x96 pixels. The second graph presents the mAP for medium objects, or smaller than 96x96 and bigger than 32x32 pixels. The third and last graph presents the mAP for small objects, smaller than 32x32 pixels.

The best mAP corresponds to the one where the large objects are recognized, with a 0.6618 score at step 100000. The medium and small objects evaluated very poorly, one possible reason for this is due to the type of images present in the Trashnet dataset. This dataset is mainly comprised of close ups of objects, which means that most of the bounding boxes annotated in the dataset will have a size bigger than 96x96 pixels and therefore the model will not be able to be trained with almost any object smaller than that. To be more specific there are 2069 large objects, 130 medium objects and 1 small object in the dataset. The number of objects does not equal to the number of images because some images have more than one object.

One other metric available in Tensorboard is the mAP Intersection over Union (IOU). This metric represents the intersection of the predicted and ground truth bounding boxes.

The formula for this metric is according to IOU, where A is the predicted bounding box and B is the ground truth bounding box annotated previously in the dataset.

$$IOU = \frac{|A \cap B|}{|A \cup B|} \quad (3)$$

From left to right, in Figure 5.6, the first graph presents the mAP IOU when at least 50% of the predicted bounding box intersects with the ground truth bounding box. For this metric we have a mAP IOU of 0.7785. The second graph presents the mAP IOU when at least 75% of the predicted bounding box intersects with the ground truth bounding box and the result is 0.742, a slightly lower value than the previous metric, which is expected since the latter is more precise than the former. It is not expected for the former metric to have a higher mAP IOU than the latter because the former metric is less precise than the latter.

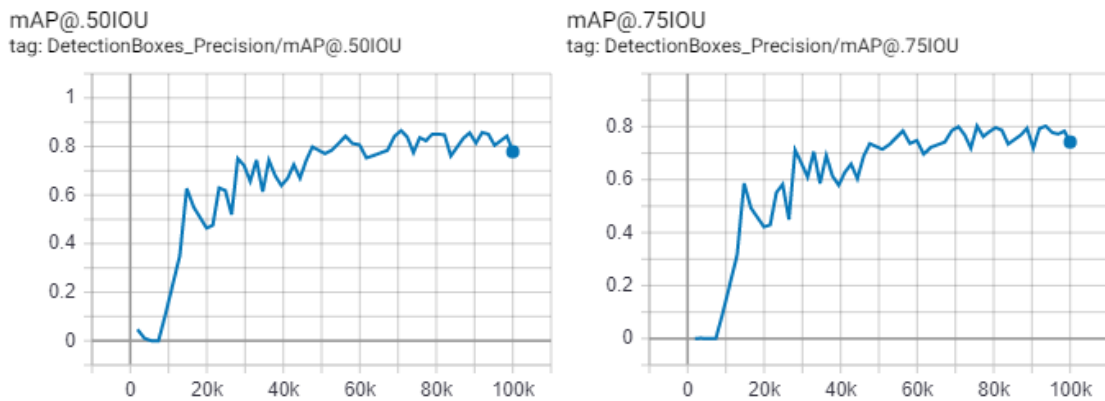


Figure 5.6 – mAP IOU results from Tensorboard

Two other important metrics are the classification loss and localization loss, as seen in Figure 5.7. Loss compares the output of the predicted with the ground truth. The main objective is for a model to minimize loss as much as possible. In this implementation two types of losses are considered, equal to the two challenges being solved. The first type of loss is classification. This type of loss measures how well the model is able to correctly categorize an object. The pipeline configuration file had the function weighted sigmoid configured by default. The result for the classification loss was 2.241. The second type of loss is localization. This type measures how well the model is able to locate objects. For this type of loss, the default configured function (weighted smooth L1) was maintained as well. The result for the localization loss was 0.1716. The difference of values in both losses indicates that the model is better at localizing objects than it is at classifying them.

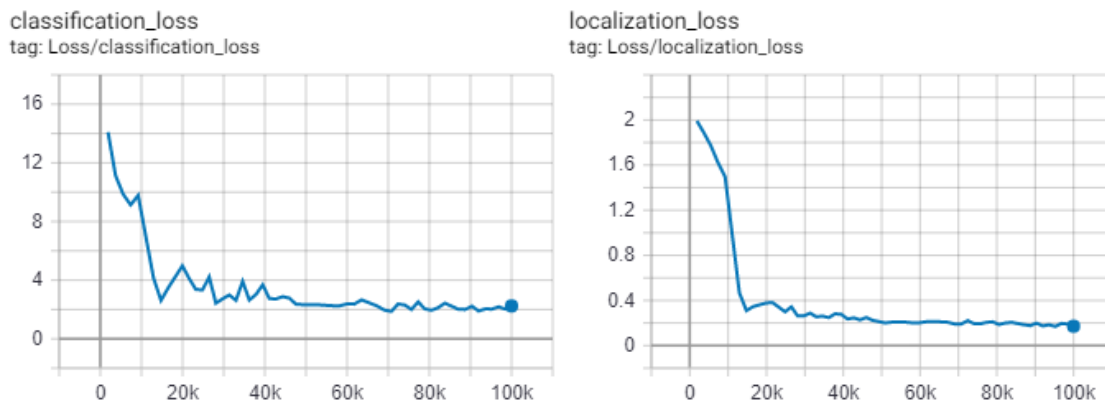


Figure 5.7 – Loss metrics from Tensorboard

Testing the trained model in the test set we have the following predictions against the ground truth in Table 5.5. To analyze the results further, a confusion matrix was created. An extra class was added by the Python code that generated the confusion matrix. The class “nothing” in the “Ground Truth” dimension represents areas of the images that were not annotated in the ground truth but were predicted as one of the classes of the dataset. In the dimension “Predicted” this class means that an object, that was labeled as one of the other classes in the ground truth, was not classified by the model. An IOU threshold of 50% was used for the detection of objects in these results, which means that the ground truth bounding box had to have at least 50% of its area under the predicted bounding box, else the classification was considered as “nothing”. A confidence threshold of 50% was used for the classification of the objects, which means that the model had to have at least 50% of confidence that an object belonged to a specific class, else the classification was considered as “nothing”.

It is important to note that the code used to generate the confusion matrix was reused from an open source code available on GitHub [55].

Table 5.5 – Test dataset results confusion matrix with SSD Mobilenet v2

		Predicted						
		cardboard	glass	metal	paper	plastic	trash	nothing
Ground Truth	cardboard	56	4	0	4	0	0	0
	glass	0	68	6	0	5	0	0
	metal	1	11	54	0	5	0	8
	paper	7	6	7	71	2	0	0
	plastic	1	13	6	3	66	0	24
	trash	1	0	0	2	0	5	1
	nothing	4	20	22	11	24	1	0

The precision results for each class can be analyzed in Table 6.6. Cardboard and glass were the classes with the highest precision score. Trash and plastic were the classes with the worst results.

Table 5.6 – Precision results for Test dataset by category with SSD Mobilenet v2

Category	Precision
cardboard	87,50%
glass	86,08%
metal	68,35%
paper	76,34%
plastic	58,41%
trash	55,55%
<b>Average</b>	72,04%

So far, all the tests were done in images from the same dataset. Even though the images used to test the model were not used to train the model, they were from a similar source. This might create situations of overfitting, where the model is good at predicting a certain dataset but is not good at generalizing. Once again, to test if the model is good at generalizing, a new validation will be performed employing the images of waste captured with a smartphone during the development of this project. This dataset is described in Chapter 4.

The results of the test in the real dataset (Table 5.7 and 5.8) reveal that the model has not been able to classify properly the objects in most of the images, with a very poor result in almost all classes. Only the cardboard class had a precision result that was close with the test dataset results, with a difference of 7,5%. The rest of the classes had worst results.



Table 5.7 – Real dataset results confusion matrix with SSD Mobilenet v2

		Predicted						
		cardboard	glass	metal	paper	plastic	trash	nothing
Ground Truth	cardboard	20	0	1	3	0	0	1
	glass	10	2	2	3	0	0	5
	metal	21	1	12	1	0	0	5
	paper	53	2	8	6	1	0	8
	plastic	54	7	17	9	9	0	12
	trash	11	0	2	0	0	0	4
	nothing	8	0	6	1	1	0	0

Table 5.8 – Precision results of Real dataset by category with SSD Mobilenet v2

Category	Precision
cardboard	80,00%
glass	9,09%
metal	30,00%
paper	7,69%
plastic	8,33%
trash	0%
Average	22,52%

The explanation for the poor results might be related with the results of training where it could be seen that the model was only able to detect large objects with precision. This might happen because the Trashnet dataset is mostly comprised of image closeups of objects and therefore the model only learns how to create large bounding boxes which occupy most of the image. Due to this the area of intersection between the ground truth bounding box and the predicted bounding box fails because it exceeds the threshold of 50%. On top of this problem the previous problems reported in Section 5.1.2 also apply here. For example, due to the fact that the model is predicting large bounding boxes that occupy a large part of the image, it means that the background of the image will affect the classification of the object just like it was affecting the MobileNet v2 model of Section 5.1. This explains the slighter worst results in this model.

The training of the model took 8 hours with the hardware and software available for the project. Some tests were done with different parameters in the training to see what affected the most the time of training. It was concluded that batch size and image size highly affected the time elapsed for the training of a model.

### 5.2.3. Converting to mobile

The result ML model will be used in the mobile application as previously explained in Chapter 4 and demonstrated in Figure 4.1 which represents the high-level system architecture. This model will be present in the mobile application at first and each new ML model that is created will be available for download in the REST API. But the output of the Tensorflow Object Detection API is not compatible with the framework Tensorflow Lite, the framework to run ML models in a smartphone. A conversion of the output model is required in order for the model to be run in a mobile environment. Two steps are required to convert the output model of the API.

The first step creates a quantized model based on the model created by the API. This post-training quantization is a conversion technique to reduce the model size and improve the CPU and hardware accelerator latency with the smallest degradation possible in the model accuracy. For this step, the python file `export_tflite_ssd_graph.py` is used.

The second and final step is the conversion of the quantized model into the tflite format. This format is the Tensorflow Lite required format to run ML models in a smartphone. For this operation, the command line “`tflite_convert`” command is used. This command is available when Tensorflow is installed in a computer.

After this step, a tflite file with the ML model is ready to be transferred to a smartphone application that uses Tensorflow Lite to execute it.

### 5.2.4. Repository

All the code produced in this module was versioned within GitHub, the repository is public and under MIT license in the following link:

[https://github.com/antoniosequeira/trainer\\_ssd\\_mobilenet\\_v2](https://github.com/antoniosequeira/trainer_ssd_mobilenet_v2).

## Chapter 6 – REST API

This chapter serves as the documentation for the REST API used in the project, information such as the technology and architecture being used will be found here.

A REST API is a method of communication between electronic devices using the world wide web, its architecture is modeled through the way data is presented, accessed or modified. Its principles allow the service to be simple and lightweight, having high performance [56]. This type of service typically allows for the main hypertext transfer protocol (HTTP) methods, the create, retrieve, update and delete actions. In Figure 6.1 an example of the REST API architecture can be seen.

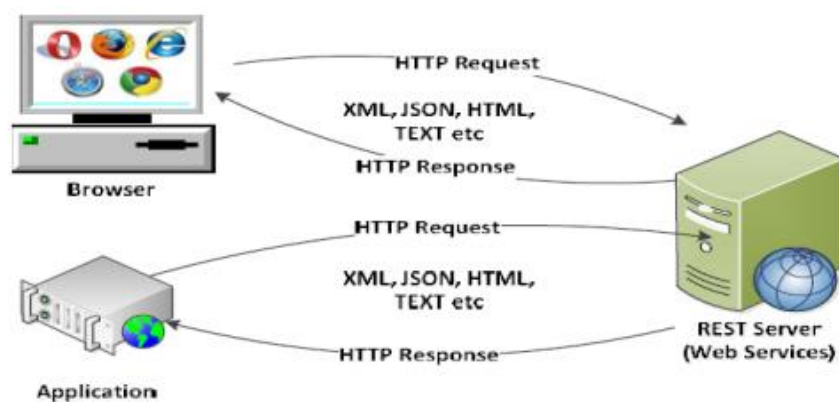


Figure 6.1 – REST API architecture [56]

In this project the REST API will serve as the handler of actions requested through the Internet by the mobile application.

For this module, it was decided to use the Django<sup>18</sup> framework because it is open source, meaning that a low cost solution could be built, it is based on Python which is widely used, so online assistance is available to help solve most of the problems, and it includes a lot of reusable components to quickly create web-based applications like this REST API. The version of Django used was the 3.0.4. The Python environment where the module was built was based on the version 3.6.7 of Python. Besides that, an extension of Django named Django Rest Framework<sup>19</sup> was used to create the REST methods, the version of the extension is the 3.11.0.

<sup>18</sup> Website link: <https://www.djangoproject.com/>

<sup>19</sup> Website link: <https://www.django-rest-framework.org/>

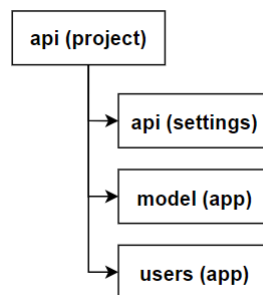
All the code of the module was versioned in GitHub in order to safeguard it and share it easily. The module was developed in Visual Studio Code<sup>20</sup>.

## 6.1. Architecture

The typical Django architecture follows the model view controller (MVC) pattern where the model corresponds to the component that acts as the mediator between the web application interface and the database. The view contains the logic for the user interface (UI). The controller, which is the main control component, has the task of selecting the view component that corresponds to the UI interaction.

For this project, the MVC pattern was used as well but with a slight variation since there is no UI, given that it is a REST API. For example, the controller instead of receiving UI interactions, it will receive REST requests.

The structure of the Django project is presented in Figure 6.2, where we can see that the project contains two apps inside of it. An app in Django is a module that has one purpose and can be reused in multiple projects, in this case we have two modules, the model app that is responsible for any logic related with the ML model, the users app is responsible for the authentication logic of the project.



*Figure 6.2 – Django project structure*

The authentication chosen for this project was token based, so the users will have to register and login their users to be able to get a token that will be necessary to access any other endpoint. No UI for user registration has been created in the project because it was out of the scope. Manually registrations were done via the native Django administrator module.

---

<sup>20</sup> Website link: <https://code.visualstudio.com/>

## 6.2. Endpoint Definition

The way the mobile application will interact with the REST API is through the API endpoints. Each of these endpoints are uniform resource locators (URL) within the REST API and are accessible through HTTP request methods which enable communication between clients and server. The methods that are used within this project are the following:

- **GET** – Used to request data from a specified resource.
- **POST** – Utilized to send data to a server.

The detailed information for the endpoints that were created can be found in Appendix A.

## 6.3. Testing

It is important when developing a feature, to be able to test and prove that it works. Multiple tests were done to ensure that all features worked as intended. The detailed information of the tests for this module are available in Appendix B.

## 6.4. Repository

All the code produced in this module was versioned within GitHub. The repository is public and under MIT license in the following link:

[https://github.com/antoniosequeira/rest\\_api](https://github.com/antoniosequeira/rest_api).



## Chapter 7 – Mobile Application

In this chapter all the information, related with the mobile application, will be described. Originally it was planned to use React-Native<sup>21</sup> as the technology to build the application. An initial version of the application was built in this technology with just the feature of capturing images, but it had to be scrapped and the technology changed due to limitations and incompatibilities with the framework Tensorflow Lite which is required to run the ML model in a mobile device. The adopted technology to build the mobile application was the mobile framework Flutter<sup>22</sup>, a JavaScript framework that allows the writing of natively rendering mobile applications for both Android and iOS. The main reasons to choose this technology are because of its versatility in terms of allowing the developer to build a web application for both Android and iOS with the same code, and it is one of the most popular frameworks for mobile applications development while being open source.

The application was developed for Android and it was only tested (as shown in Section 7.3) in the version 10 and 11 of this operating system (OS), yet the technology that was used, in theory allows to rapidly create an iOS application as well, with the same code, yet this was not part of the scope of the project and therefore was not tested or implemented.

As stated, the main technology used to build the mobile application was Flutter, because this portable UI toolkit allows the developer to use nonnative programming language to create cross-platform applications. This framework was developed by Google and it utilizes as its programming language the Dart language. This project used version 1.18.0-11.1.pre. As a pre-requisite for Flutter, Android Studio<sup>23</sup> is required in order to get an Android Software Development Kit (SDK) to be able to develop Android applications. The version of Android Studio used in the project was version 4.0 and the Android SDK version was the 29.0.3. It is important to mention that tflite<sup>24</sup> a Flutter plugin for accessing TensorFlow Lite API was utilized and the version was 1.0.6.

For Flutter development, Visual Studio Code was used.

---

<sup>21</sup> Website link: <https://reactnative.dev/>

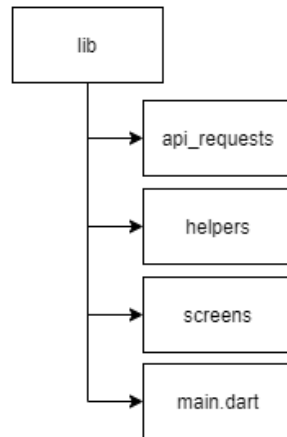
<sup>22</sup> Website link: <https://flutter.dev/>

<sup>23</sup> Website link: <https://developer.android.com/studio>

<sup>24</sup> Website link: <https://pub.dev/packages/tflite>

## 7.1. Architecture

The code for the mobile application was organized in logical modules as shown in Figure 7.1.



*Figure 7.1 – Mobile application architecture*

The lib folder contains all the public code, and it is created by default when generating a new Flutter project. The api\_requests folder contains all the logic that the app requires to interact with the REST API described in Chapter 6. The helpers folder contains required logic to interact with the tflite plugin. The screens folder contains the logic for secondary screens in the application. At last the “main.dart”<sup>25</sup> class is the entry point of the application and contains the necessary code for the main actions and navigation for secondary screens.

## 7.2. Available Actions

The mobile application allows the user to perform multiple actions. For a better understanding of the flow the user experience in the application, Figure 7.2 was created.

<sup>25</sup> Github link: [https://github.com/antoniosequeira/mobile\\_application/blob/main/lib/main.dart](https://github.com/antoniosequeira/mobile_application/blob/main/lib/main.dart)



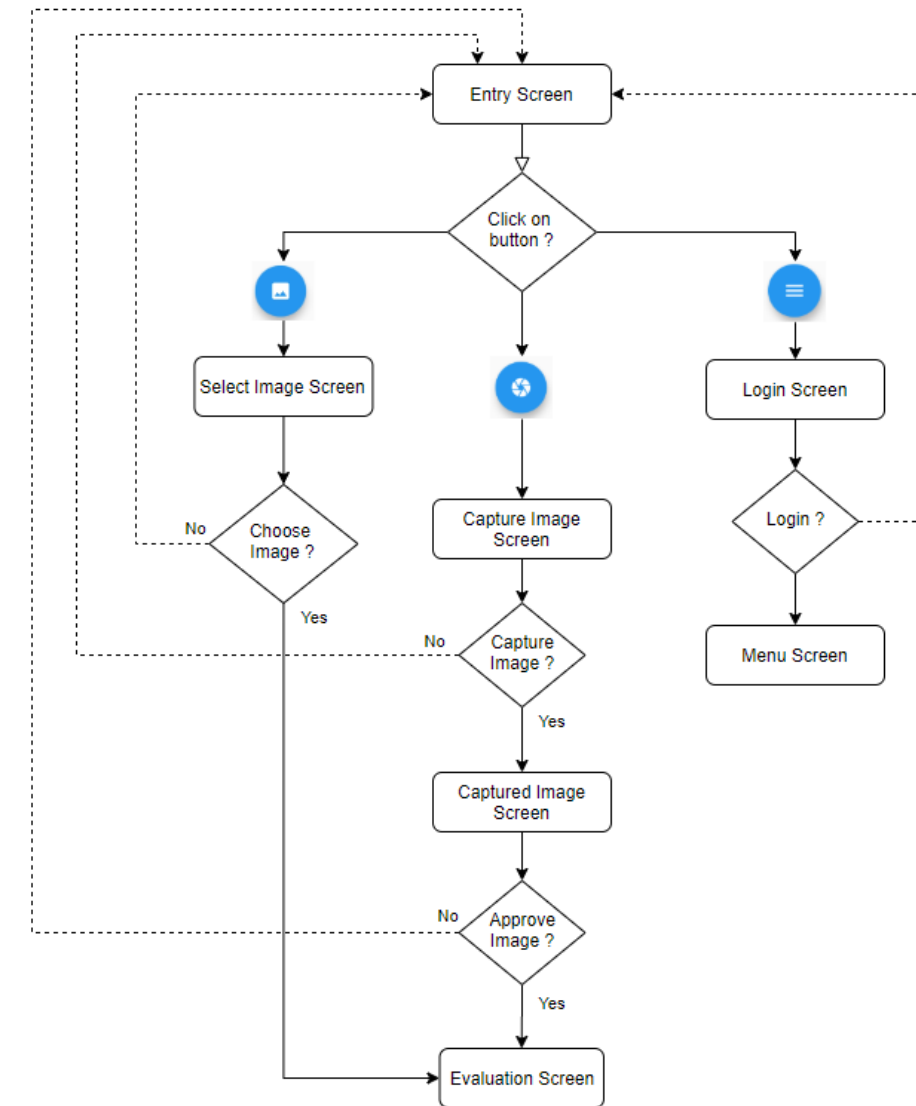
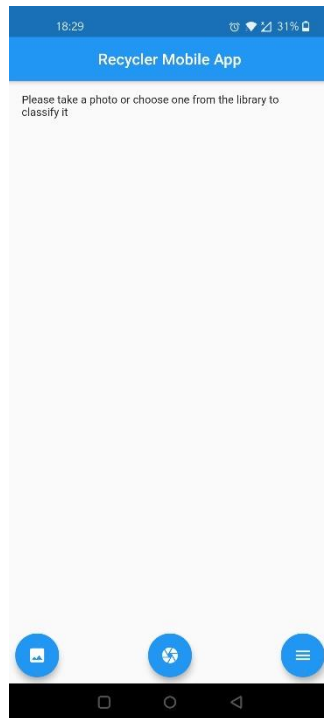


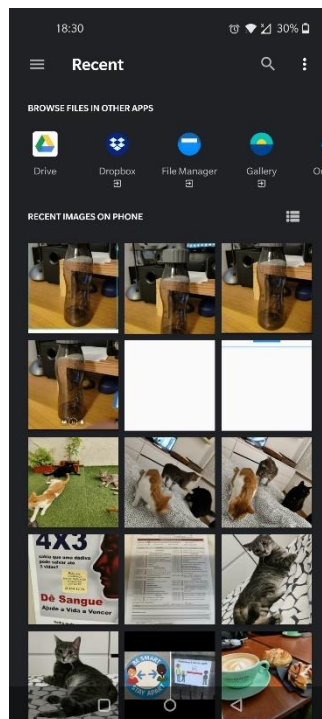
Figure 7.2 – Mobile application experience flow

The user experience is comprised of three main sequences. The first sequence begins when a user enters the entry screen (Figure 7.3) and clicks on the lower left button of the application.



*Figure 7.3 – Entry screen*

This action will direct the application to the select image screen (Figure 7.4). This screen may vary in aspect in each smartphone due to the utilization of native functionalities of the mobile operating system.



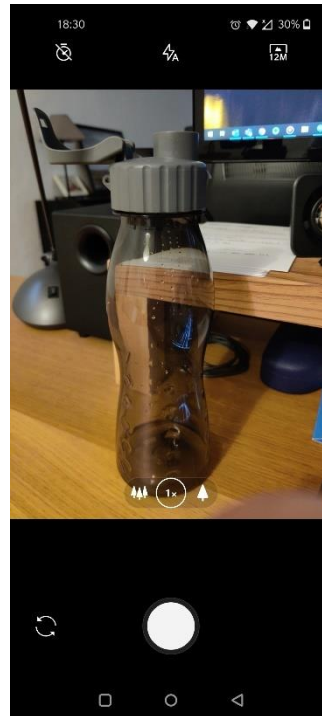
*Figure 7.4 – Select image screen*

If the user chooses an image the application will redirect to the evaluation screen (Figure 7.5) with the result of the analysis of the selected image. The analysis is an output of the image with a bounding box around the object, the name of the class for that object and the probability of the classification. Else it will be redirected to the entry screen (Figure 7.3).



*Figure 7.5 – Evaluation screen*

The second sequence begins when a user clicks on the middle lower button of the entry screen (Figure 7.3). This will result into the redirection to the capture image screen (Figure 7.6). This screen may vary in aspect in each smartphone due to the utilization of native functionalities of the mobile operation system.



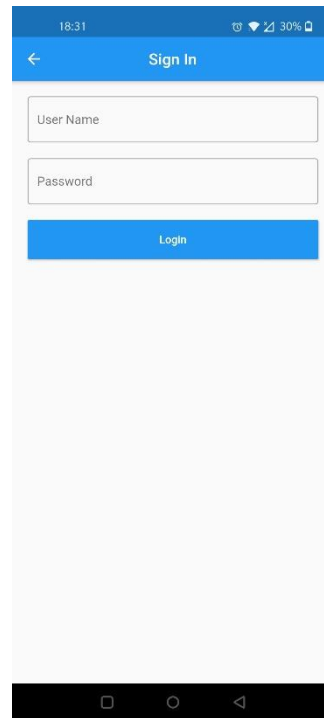
*Figure 7.6 – Capture image screen*

In this screen the user can capture a picture of an object. After that it will be redirected to the Confirm image screen (Figure 7.7). In this screen the user can overview the image that has just captured and it can accept or reject the image. If it accepts the image it will be redirected to the evaluation screen (Figure 7.5) just like the previous sequence. If it rejects the image it will be redirected to the capture image screen (Figure 7.6) to be able to capture another image.



*Figure 7.7 – Confirm image screen*

The third and final sequence happens when a user clicks on the lower left button of the entry screen (Figure 7.3). The user will be redirected to the login screen (Figure 7.8). In this screen a user that is previously registered with the platform will be able to login with a username and password.

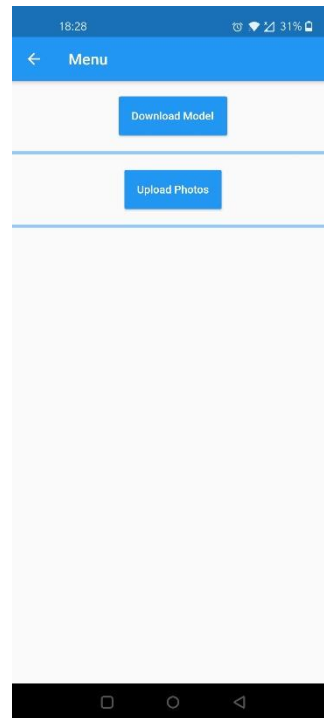


*Figure 7.8 – Login screen*

After a successful login, the user will be redirected to the menu screen (Figure 7.9). In this screen the user has two options.

The first option is called download model. This allows for the user to download a new version of the ML model that classifies images. The download will only happen if the present model in the application has a different version of the most recent one present in the server.

The second and final option is called upload photos. This option allows the user to upload images, that were captured by the application, into the server. These images will be taken into consideration for the training of new versions of the ML model. Only the images are uploaded, without any annotation.



*Figure 7.9 – Menu screen*

### 7.3. Testing

The mobile application was tested in two different devices the first one was through a virtual device created in Android Virtual Device Manager module from Android Studio, this virtual device works as an emulator of a mobile device, the device that was emulated was a Pixel 3 with Android 10. The second device where it was tested was a OnePlus 8 smartphone, running Android 11. Each available action was tested to make sure it was working properly. Due to the nature of the tests (actions in a smartphone application) it was not possible to register these results in this document. Due to this limitation, only the actions will be described in the section.

The actions used in the tests followed the three user experience paths described in Figure 7.2. Two types of tests were done, the first one involved testing the three sequence paths within the application with a reset of the application within each sequence to clean any saved state in the application. This made sure that no sequence was dependent of any other and that each feature was functional by itself. The second test repeated the testing of the three sequence paths within the application but without a reset of the application within each sequence. This was done to ensure that no error state was introduced by any path sequence. In both types of tests, the order of the path sequences was alternated to

make sure that every type of flow of sequences did not introduce any error state in the application.

#### **7.4. Repository**

All the code produced in this module was versioned within GitHub. The repository is public and under MIT license in the following link:

[https://github.com/antoniosequeira/mobile\\_application](https://github.com/antoniosequeira/mobile_application).



## Chapter 8 – Conclusions and recommendations

### 8.1. Main conclusions

This project's main objective was the creation of a mobile application that can help people to recycle. The secondary objectives were the creation of an updated compendium containing related work in the area of computer vision that uses ML to categorize waste, and the contribution to current datasets in the area.

Three research questions were answered during the development of the project.

RQ1: Considering the requisites of an ML model, which types of ML algorithms are best suited for the classification of waste images within a smartphone application?

This question was answered by researching the available literature and analyzing the results of previous studies. The MobileNet architecture had the best results due to its lightweight characteristics. Even though it was reported that the architecture had a lower accuracy generally when compared to others. Within the MobileNet architecture there are three iterations, from v1 to v3. Each iteration has improvements in the reported accuracy when compared to previous iterations. It was important that the reported results were related with the task of classification waste. Due to this the v3 architecture was excluded because there were no scientific papers reporting results in this architecture. Another point was considered, the availability of the architecture in existing ML frameworks that allowed quick development of the project. Only the v1 and v2 architectures were available at the moment of the creation of this project. Taking all this information into account, the MobileNet v2 architecture was the choice for the answer to this question yet the performance of this architecture in real images was shown that it is much lower to what is registered in the literature.

RQ2: Is it possible to create an accurate (that is, achieving over 90% precision) but yet light weight model (enabling a quick classification: under three seconds) able to run in smartphones?

This answer is not completely clear, due to the different types of tests that were done which returned mixed results. The MobileNet v2 model without object detection had an average precision of 94,65% on the test dataset, yet the test on the real dataset returned an average precision of 24,94%. The classification for this model when tested in the two environments was executed under three seconds. These results indicate that the ML model

isn't good enough for real life scenarios, but it is good enough to run in a smartphone within the purposed classification time. The SSD MobileNet v2 model that contains object detection failed the precision test with an average precision of 72,04% in the test dataset and go an even worse result in the real dataset, with an average precision of 22,52%. This ML model was able to classify images under three seconds as well in the mobile application, when tested in the two available environments.

RQ3: Is the most cited public available dataset of waste images diverse enough to produce a ML model that can classify waste into its specific recycling category in a real-life scenario?

The answer to this question is that the data is not diverse enough. The most cited public available dataset of waste images is Trashnet. In a MobileNet v2 architecture without object detection the test results in the test dataset returned an average precision of 94,65% which is very high, yet when these tests were done in the real dataset the ML model only had an average precision result of 24,94%. In an SSD MobileNet v2 architecture with object detection the test results in the test dataset returned an average precision of 72,04%, while these tests when done in the real dataset revealed an average precision of 22,52%. These results corroborate the lack of diversity in Trashnet dataset to produce a ML that can classify waste in a real-life scenario.

In terms of development this project demonstrated that a system that connects a mobile application to a server in order to transfer information for the production of new ML models is possible. All modules of the system were successfully produced. The mobile application that can use a ML model to classify waste images, upload images to the API and download from the API new versions of the ML model. The API that can authenticate users, receive images and upload ML models. The ML server module that can train new ML models.

## **8.2. Contributions to the scientific and business community**

The project contributed to the scientific and business community with a compendium of information in the area of computer vision and ML to categorize waste. This information can serve as a point of departure for present and future investigation in this area of expertise.

A working prototype system was created, showing that, both in theory and empirically, a system that uses crowdsourcing and ML elements can be further explored as a possible system for developing mobile applications in the area of waste management and classification.

### **8.3. Limitations of the study**

This project had multiple limitations that affected its outcome. Starting on the ML architecture options. Due to the amount of work required for this project a ML architecture had to be selected based on previous studies and it wasn't timely possible to compare multiple architectures to compare results. Due to the limitations present in the available ML frameworks that facilitate the development of code to train models, the most recent version of the chosen architecture wasn't used, because it wasn't available at the time when this project was developed.

The amount of data used in the training of the ML models has also stand as a serious limitation. Only two datasets from previous scientific studies were available. None of these two datasets were annotated with bounding boxes that allowed for the training of a ML model with object detection. Due to the limited time of this project, only the most cited dataset (Trashnet) was annotated with bounding boxes and therefore used as data for training.

Even though the study produced a working system prototype, the precision of the classification of the mobile application is very limited in real life as concluded. The system was not tested by third party subjects and therefore was limited to testing within the development environment.

The diversity of technologies required for the scope of the project brought the limitation of the best development practices in each module due to lack of knowledge in these technologies. This might have caused unnecessary delays and performance degradation in each module.

#### **8.4. Future work**

Multiple possible iterations of this project have been left for the future due to lack of time. Future work could focus on the limitations that were highlighted in Section 8.3 but not only necessarily.

It would be important for example to make a further study on ML architectures for mobile applications, due to the increasing smartphone computing power there is the probability that more process demanding architectures might be able to be used successfully in smartphones.

Another idea is to gather or produce more data that can be used to train the ML models. This can be achieved through direct gathering by the further researchers or through crowdsourcing techniques where a prototype system like the one that was developed in this project can be released to more users in order to gather data through its crowdsourcing capabilities. This would be important to build more precise ML models for the application.

A better user experience for the user should be developed for the mobile application. For example, more information could be returned to the user when it uses the application to classify an image. Information on how the material is recycled and the possible usages for that material. Techniques of gamification might also be an option to further engage the users to explore the application.

In theory the technology used to build the mobile application allows for the creation of code that can be compiled for both Android and IOS operating systems. Since this project scope was limited to Android, it would be interesting to also test or if needed adapt the system to fully support both mobile operating systems.

## Bibliography

- [1] S.-A. Barnes, A. Green, and M. de Hoyos, "Crowdsourcing and work: individual factors and circumstances influencing employability: Crowdsourcing and work," *New Technology, Work and Employment*, vol. 30, no. 1, pp. 16–31, Mar. 2015, doi: 10.1111/ntwe.12043.
- [2] C. Zhihong, Z. Hebin, W. Yanbo, L. Binyan, and L. Yu, "A vision-based robotic grasping system using deep learning for garbage sorting," in *2017 36th Chinese Control Conference (CCC)*, Jul. 2017, pp. 11223–11226, doi: 10.23919/ChiCC.2017.8029147.
- [3] Wang Kun and Kong Songtao, "Identification method of waste based on gray level co-occurrence matrix and neural network," in *2011 International Conference on Materials for Renewable Energy Environment*, May 2011, vol. 1, pp. 929–931, doi: 10.1109/ICMREE.2011.5930954.
- [4] Z. Wang, B. Peng, Y. Huang, and G. Sun, "Classification for plastic bottles recycling based on image recognition," *Waste Management*, vol. 88, pp. 170–181, Apr. 2019, doi: 10.1016/j.wasman.2019.03.032.
- [5] Mohammad Osiur Rahman, Aini Hussain, Edgar Scavino, Hassan Basri, and M.A. Hannan, "Intelligent computer vision system for segregating recyclable waste papers," 2011. .
- [6] A. Salmador, J. Pérez Cid, and I. Rodríguez Novelle, "Intelligent Garbage Classifier," Jan. 2008, Accessed: Oct. 24, 2019. [Online]. Available: <http://dialnet.unirioja.es/servlet/oaiart?codigo=4035318>.
- [7] S. Gundupalli Paulraj, S. Hait, and A. Thakur, "Automated Municipal Solid Waste Sorting for Recycling Using a Mobile Manipulator," p. V05AT07A045, Aug. 2016, doi: 10.1115/DETC2016-59842.
- [8] B. M. Chinnathurai, R. Sivakumar, S. Sadagopan, and J. M. Conrad, "Design and implementation of a semi-autonomous waste segregation robot," in *SoutheastCon 2016*, Mar. 2016, pp. 1–6, doi: 10.1109/SECON.2016.7506679.
- [9] M. S. Rad *et al.*, "A Computer Vision System to Localize and Classify Wastes on the Streets," in *Computer Vision Systems*, 2017, pp. 195–204.
- [10] Y. Liao, R. Lu, S. Wu, P. Cheng, and G. Xu, "The robot for recycling based on machine learning," in *2018 International Automatic Control Conference (CACCS)*, Nov. 2018, pp. 1–6, doi: 10.1109/CACCS.2018.8606758.
- [11] S. Nandhini, S. S. Mrinal, N. Balachandran, K. Suryanarayana, and D. S. H. Ram, "Electronically assisted automatic waste segregation," in *2019 3rd International Conference on Trends in Electronics and Informatics (ICOEI)*, Apr. 2019, pp. 846–850, doi: 10.1109/ICOEI.2019.8862666.
- [12] Z. Wang, H. Li, and X. Zhang, "Construction waste recycling robot for nails and screws: Computer vision technology and neural network approach," *Automation in Construction*, vol. 97, pp. 220–228, Jan. 2019, doi: 10.1016/j.autcon.2018.11.009.
- [13] A. Chung, S. Kim, E. Kwok, M. Ryan, E. Tan, and R. Gamadia, "Cloud Computed Machine Learning Based Real-Time Litter Detection using Micro-UAV Surveillance," 2018, p. 10.
- [14] S. Sudha, M. Vidhyalakshmi, K. Pavithra, K. Sangeetha, and V. Swaathi, "An automatic classification method for environment: Friendly waste segregation using deep learning," in *2016 IEEE Technological Innovations in ICT for Agriculture and Rural Development (TIAR)*, Jul. 2016, pp. 65–70, doi: 10.1109/TIAR.2016.7801215.
- [15] L. R. Kambam and A. R., "Classification of plastic bottles based on visual and physical features for waste management," in *2019 IEEE International Conference on Electrical*,

- Computer and Communication Technologies (ICECCT)*, Feb. 2019, pp. 1–6, doi: 10.1109/ICECCT.2019.8869191.
- [16] Y. Chu, C. Huang, X. Xie, B. Tan, S. Kamal, and X. Xiong, “Multilayer Hybrid Deep-Learning Method for Waste Classification and Recycling,” *Computational Intelligence and Neuroscience*, vol. 2018, pp. 1–9, Nov. 2018, doi: 10.1155/2018/5060857.
- [17] A. N. Kokoulin, A. I. Tur, and A. A. Yuzhakov, “Convolutional neural networks application in plastic waste recognition and sorting,” in *2018 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIconRus)*, Jan. 2018, pp. 1094–1098, doi: 10.1109/EIconRus.2018.8317281.
- [18] P. Dhulekar, S. T. Gandhe, and U. P. Mahajan, “Development of Bottle Recycling Machine Using Machine Learning Algorithm,” in *2018 International Conference On Advances in Communication and Computing Technology (ICACCT)*, Feb. 2018, pp. 515–519, doi: 10.1109/ICACCT.2018.8529483.
- [19] A. N. Kokoulin and D. A. Kiryanov, “The Optical Subsystem for the Empty Containers Recognition and Sorting in a Reverse Vending Machine,” in *2019 4th International Conference on Smart and Sustainable Technologies (SpliTech)*, Jun. 2019, pp. 1–6, doi: 10.23919/SpliTech.2019.8782990.
- [20] A. Torres-García, O. Rodea-Aragón, O. Longoria-Gandara, F. Sánchez-García, and L. Enrique González-Jiménez, “Intelligent Waste Separator,” *Computacion y Sistemas*, vol. 19, pp. 487–500, Sep. 2015, doi: 10.13053/CyS-19-3-2254.
- [21] L. Omar, R. Oscar, T. Andres, and S. Francisco, “Multimedia inorganic waste separator,” in *2013 IEEE International Conference on Multimedia and Expo Workshops (ICMEW)*, Jul. 2013, pp. 1–4, doi: 10.1109/ICMEW.2013.6618314.
- [22] I. Salimi, B. S. B. Dewantara, and I. K. Wibowo, “Visual-based trash detection and classification system for smart trash bin robot,” in *2018 International Electronics Symposium on Knowledge Creation and Intelligent Computing (IES-KCIC)*, Oct. 2018, pp. 378–383, doi: 10.1109/KCIC.2018.8628499.
- [23] Shamin N, P. M. Fathimal, R. R, and K. Prakash, “Smart Garbage Segregation Management System Using Internet of Things(IoT) Machine Learning(ML),” in *2019 1st International Conference on Innovations in Information and Communication Technology (ICIICT)*, Apr. 2019, pp. 1–6, doi: 10.1109/ICIICT1.2019.8741443.
- [24] S. L. Rabano, M. K. Cabatuan, E. Sybingco, E. P. Dadios, and E. J. Calilung, “Common Garbage Classification Using MobileNet,” in *2018 IEEE 10th International Conference on Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment and Management (HNICEM)*, Nov. 2018, pp. 1–4, doi: 10.1109/HNICEM.2018.8666300.
- [25] G. Mittal, K. B. Yagnik, M. Garg, and N. C. Krishnan, “SpotGarbage: smartphone app to detect garbage using deep learning,” in *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing - UbiComp '16*, Heidelberg, Germany, 2016, pp. 940–945, doi: 10.1145/2971648.2971731.
- [26] S. Singh *et al.*, “Identifying uncollected garbage in urban areas using crowdsourcing and machine learning,” in *2017 IEEE Region 10 Symposium (TENSYP)*, Jul. 2017, pp. 1–5, doi: 10.1109/TENCONSpring.2017.8070078.
- [27] S. Frost, B. Tor, R. Agrawal, and A. G. Forbes, “CompostNet: An Image Classifier for Meal Waste,” in *2019 IEEE Global Humanitarian Technology Conference (GHTC)*, Seattle, WA, USA, Oct. 2019, pp. 1–4, doi: 10.1109/GHTC46095.2019.9033130.
- [28] Sreelakshmi K, Akarsh S, Vinayakumar R, and P. Soman K., “Capsule Neural Networks and Visualization for Segregation of Plastic and Non-Plastic Wastes,” in *2019 5th International Conference on Advanced Computing Communication Systems (ICACCS)*, Mar. 2019, pp. 631–636, doi: 10.1109/ICACCS.2019.8728405.

- [29] R. A. Aral, Ş. R. Keskin, M. Kaya, and M. Hacıömeroğlu, “Classification of TrashNet Dataset Based on Deep Learning Models,” in *2018 IEEE International Conference on Big Data (Big Data)*, Dec. 2018, pp. 2058–2062, doi: 10.1109/BigData.2018.8622212.
- [30] K. Anding, E. Linß, H. Träger, M. Rückwardt, and A. Göpfert, “Optical identification of construction and demolition waste by using image processing and machine learning methods,” *Proceedings of the 14th Joint International IMEKO TC1 + TC7 + TC 13 Symposium: “Intelligent quality measurements - theory, education and training” ; in conjunction with the 56th IWK, Ilmenau University of Technology and the 11th SpectroNet Collaboration Forum ; 31. August - 2. September 2011, JenTower Jena, Germany*, vol. 56, 2011, Dec. 2011, Accessed: Jun. 23, 2020. [Online]. Available: [https://www.db-thueringen.de/receive/dbt\\_mods\\_00019542](https://www.db-thueringen.de/receive/dbt_mods_00019542).
- [31] C. Bircanoğlu, M. Atay, F. Beşer, Ö. Genç, and M. A. Kızrak, “RecycleNet: Intelligent Waste Sorting Using Deep Neural Networks,” in *2018 Innovations in Intelligent Systems and Applications (INISTA)*, Jul. 2018, pp. 1–7, doi: 10.1109/INISTA.2018.8466276.
- [32] Shahrani Shahbudin, Aini Hussain, Dzuraidah Abdul Wahab, Mohd Marzuki Mustafa, and Suzaimah Ramli, “Support Vector Machines for automated classification of plastic bottles,” in *2010 6th International Colloquium on Signal Processing its Applications*, May 2010, pp. 1–5, doi: 10.1109/CSPA.2010.5545265.
- [33] W. Setiawan, A. Wahyudin, and G. R. Widiyanto, “The use of scale invariant feature transform (SIFT) algorithms to identification garbage images based on product label,” in *2017 3rd International Conference on Science in Information Technology (ICSITech)*, Oct. 2017, pp. 336–341, doi: 10.1109/ICSITech.2017.8257135.
- [34] B. W. House, D. W. Capson, and D. C. Schuurman, “Towards real-time sorting of recyclable goods using support vector machines,” in *Proceedings of the 2011 IEEE International Symposium on Sustainable Systems and Technology*, May 2011, pp. 1–6, doi: 10.1109/ISSST.2011.5936845.
- [35] M. Nawrocky, D. C. Schuurman, and J. Fortuna, “Visual sorting of recyclable goods using a support vector machine,” in *CCECE 2010*, May 2010, pp. 1–4, doi: 10.1109/CCECE.2010.5575231.
- [36] M. Yang and G. Thung, “Classification of Trash for Recyclability Status,” p. 6, 2016.
- [37] U. Ozkaya and L. Seyfi, “Fine-Tuning Models Comparisons on Garbage Classification for Recyclability,” *arXiv:1908.04393 [cs]*, Aug. 2019, Accessed: Oct. 27, 2019. [Online]. Available: <http://arxiv.org/abs/1908.04393>.
- [38] M. Satvilkar, “Image Based Trash Classification using Machine Learning Algorithms for Recyclability Status,” masters, Dublin, National College of Ireland, 2018.
- [39] H. N. Kulkarni and N. Kannamangalam, “Waste Object Detection and Classification,” 2019.
- [40] A. G. Howard *et al.*, “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications,” *arXiv:1704.04861 [cs]*, Apr. 2017, Accessed: Oct. 21, 2019. [Online]. Available: <http://arxiv.org/abs/1704.04861>.
- [41] G. E. Sakr, M. Mokbel, A. Darwich, M. N. Khneisser, and A. Hadi, “Comparing deep learning and support vector machines for autonomous waste sorting,” in *2016 IEEE International Multidisciplinary Conference on Engineering Technology (IMCET)*, Nov. 2016, pp. 207–212, doi: 10.1109/IMCET.2016.7777453.
- [42] C. Li, Y. Yang, M. Feng, S. Chakradhar, and H. Zhou, “Optimizing Memory Efficiency for Deep Convolutional Neural Networks on GPUs,” in *SC16: International Conference for High Performance Computing, Networking, Storage and Analysis*, Salt Lake City, UT, USA, Nov. 2016, pp. 633–644, doi: 10.1109/SC.2016.53.

- [43] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” *Commun. ACM*, vol. 60, no. 6, pp. 84–90, May 2017, doi: 10.1145/3065386.
- [44] V. Dumoulin and F. Visin, “A guide to convolution arithmetic for deep learning,” *arXiv:1603.07285 [cs, stat]*, Jan. 2018, Accessed: Dec. 09, 2019. [Online]. Available: <http://arxiv.org/abs/1603.07285>.
- [45] J. G. Carney and P. Cunningham, “The Epoch Interpretation of Learning,” p. 5.
- [46] Y. Guo, H. Shi, A. Kumar, K. Grauman, T. Rosing, and R. Feris, “SpotTune: Transfer Learning Through Adaptive Fine-Tuning,” p. 10.
- [47] D. P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization,” *arXiv:1412.6980 [cs]*, Jan. 2017, Accessed: Oct. 03, 2020. [Online]. Available: <http://arxiv.org/abs/1412.6980>.
- [48] K. Team, “Keras documentation: Probabilistic losses.” [https://keras.io/api/losses/probabilistic\\_losses/#categorical\\_crossentropy-class](https://keras.io/api/losses/probabilistic_losses/#categorical_crossentropy-class) (accessed Oct. 03, 2020).
- [49] S. C. Wong, A. Gatt, V. Stamatescu, and M. D. McDonnell, “Understanding data augmentation for classification: when to warp?,” *arXiv:1609.08764 [cs]*, Nov. 2016, Accessed: Oct. 03, 2020. [Online]. Available: <http://arxiv.org/abs/1609.08764>.
- [50] W. Liu *et al.*, “SSD: Single Shot MultiBox Detector,” *arXiv:1512.02325 [cs]*, vol. 9905, pp. 21–37, 2016, doi: 10.1007/978-3-319-46448-0\_2.
- [51] S. Arabi, A. Haghghat, and A. Sharma, “A deep learning based solution for construction equipment detection: from development to deployment,” p. 18.
- [52] J. Huang *et al.*, “Speed/accuracy trade-offs for modern convolutional object detectors,” *arXiv:1611.10012 [cs]*, Apr. 2017, Accessed: Oct. 03, 2020. [Online]. Available: <http://arxiv.org/abs/1611.10012>.
- [53] darrenl, *tzutalin/labelImg*. 2020.
- [54] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, “The Pascal Visual Object Classes (VOC) Challenge,” *Int J Comput Vis*, vol. 88, no. 2, pp. 303–338, Jun. 2010, doi: 10.1007/s11263-009-0275-4.
- [55] S. Valdarrama, *svpino/tf\_object\_detection\_cm*. 2020.
- [56] E. Thu and T. Aung, “Developing mobile application framework by using RESTFuL web service with JSON parser,” Aug. 2015, vol. 388, doi: 10.1007/978-3-319-23207-2\_18.



## Appendices



## Appendix A

This appendix serves as a detailed documentation for the endpoints that were mentioned in Section 6.2. These endpoints are organized in the following categories:

### Model Endpoints

These endpoints are related with the logic associated with the ML model.

`/model/download/`

This endpoint is for users that require to download the latest version of the ML model. The HTTP method to be used with this endpoint is the GET method.

Sample endpoint:

**`https://recycler-api.herokuapp.com/model/download/`**

Sample Request body:

**`curl -X GET -H "Authorization: <token>" https://recycler-api.herokuapp.com/model/download/`**

The response contains a ML model. Which is to be used within the mobile application environment.

`/model/version/`

This endpoint is for users that require to check the number of the latest version of the ML model available. The HTTP method to be used with this endpoint is the GET method.

Sample endpoint:

**`https://recycler-api.herokuapp.com/model/version/`**

Sample Request body:

**`curl -X GET -H "Authorization: <token>" https://recycler-api.herokuapp.com/model/version/`**

The response contains a json format like the following example:

**`{ "version": "1.0" }`**

/model/upload/

This endpoint is for users that require to upload images to the server. The HTTP method to be used with this endpoint is the POST method.

Sample endpoint:

**https://recycler-api.herokuapp.com/model/upload/**

Sample Request body:

**curl -X POST -H "Authorization: <token>" -F "file=@/<filepath>.<filetype>"**

**https://recycler-api.herokuapp.com/model/upload/**

The response contains a json format like the following example:

```
{“id”: 1, “file”: “/media/image.jpg”}
```

## User Endpoints

These endpoints are related with the logic associated with the management of user accounts and their login/logout.

/users/login/

This endpoint is for users that require to login with their user to retrieve the Token that is necessary to access the Model endpoints. The HTTP method to be used with this endpoint is the POST method.

Sample endpoint:

**https://recycler-api.herokuapp.com/users/login/**

Sample Request body:

**curl -X POST -d "username=<username>&password=<password>" https://recycler-api.herokuapp.com/users/login/**

The response contains a json format like the following example:

```
{“token”: “a64824a0dca3cf8f1c4fc0ccd4eccb635a001346”}
```

/users/register/

This endpoint is for users that require to login, but they still haven't created a user, so they will need to register one user first before being to login. A token will be automatically generated for the user being created. The HTTP method to be used with this endpoint is the POST method.

Sample endpoint:

**<https://recycler-api.herokuapp.com/users/register/>**

Sample Request body:

**curl -X POST -d "username=<username>&password=<password>" https://recycler-api.herokuapp.com/users/register/**

The response contains a json format like the following example:

```
{“user”: “test”, “first_name”: “”, “last_name”: “”, “token”:  
“a64824a0dca3cf8f1c4fc0ccd4eccb635a001346”}
```

/users/logout/

This endpoint is for users that require to logout their users. The HTTP method to be used with this endpoint is the GET method.

Sample endpoint:

**<https://recycler-api.herokuapp.com/users/logout/>**

Sample Request body:

**curl -X GET -H "Authorization: <Token>" https://recycler-api.herokuapp.com/users/logout/**

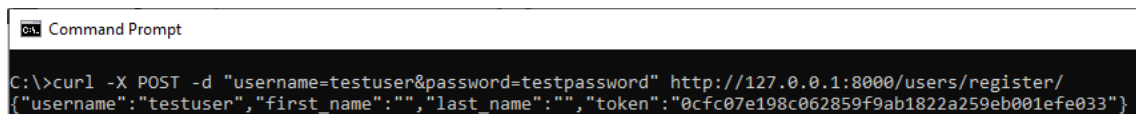
There is no response in this endpoint.



## Appendix B

It is important when developing a feature, to be able to test and prove that it works, as such the method that was chosen to test the endpoints was through the command line using the open source/free software Curl<sup>26</sup> because it allows to transfer data by using commands in the command line. Curl is available for multiple Operating Systems such as Windows and Linux. This batch of tests the REST API was executed in the localhost.

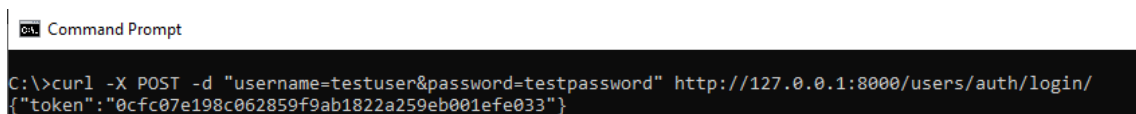
In Figure B.1 it can be seen the successful testing for the “users/register” endpoint, in line 1 the request for the creation of a user is done with the name “testuser”, the password “testpassword” and the URL where the endpoint is located. In line 2 the response from the API with the username and token. The “first\_name” and “last\_name” variables are redundant.



```
ca Command Prompt
C:\>curl -X POST -d "username=testuser&password=testpassword" http://127.0.0.1:8000/users/register/
{"username": "testuser", "first_name": "", "last_name": "", "token": "0cfc07e198c062859f9ab1822a259eb001efe033"}
```

*Figure B.1 – User register testing*

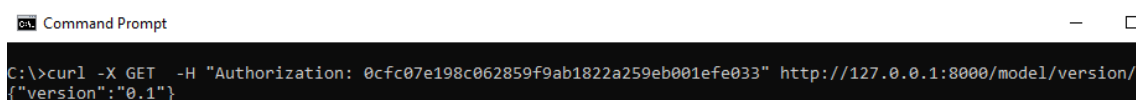
In Figure B.2, the endpoint “users/login” is successfully tested, with the login of the user that was created in the previous test and corresponding password, a token is returned which should be used for any model related endpoints.



```
ca Command Prompt
C:\>curl -X POST -d "username=testuser&password=testpassword" http://127.0.0.1:8000/users/auth/login/
{"token": "0cfc07e198c062859f9ab1822a259eb001efe033"}
```

*Figure B.2 – User login testing*

In Figure B.3, the endpoint “model/version” is tested by passing in line 1 the token that we received previously in the user login and the URL of the endpoint. The test is successful and in line 2 we get a json with the version number of the current model present in the server.



```
ca Command Prompt
C:\>curl -X GET -H "Authorization: 0cfc07e198c062859f9ab1822a259eb001efe033" http://127.0.0.1:8000/model/version/
{"version": "0.1"}
```

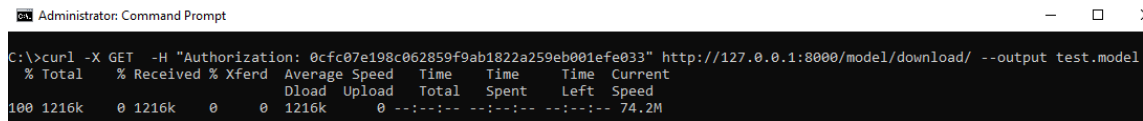
*Figure B.3 – Model version testing*

In Figure B.4, the endpoint “model/download” is tested by requesting (in line 1) to download ML model and save in the file “test.model”, the authentication token is given and

---

<sup>26</sup> Website link: <https://curl.haxx.se/>

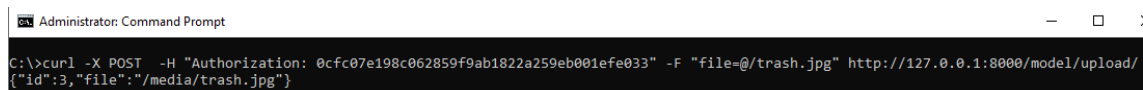
the URL for the endpoint is specified. The following lines are the result of downloading the specified file, the test was done with success.



```
Administrator: Command Prompt
C:\>curl -X GET -H "Authorization: 0cfc07e198c062859f9ab1822a259eb001efe033" http://127.0.0.1:8000/model/download/ --output test.model
% Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
           Dload  Upload   Total     Spent    Left  Speed
100 1216k    0 1216k    0     0  1216k    0  --:--:--  --:--:--  --:--:--  74.2M
```

*Figure B.4 – Model download testing*

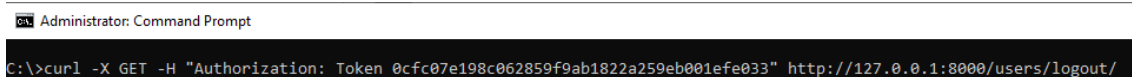
In Figure B.5, the endpoint “model/upload” is tested. In line 1 a request is created by specifying the authentication token, the file to be uploaded and the URL of the endpoint. In line 2 the response specifies that the new file has been uploaded with success and it was given the id 3 and it is located in the specified path, in this case in the path “/media/trash.jpg”.



```
Administrator: Command Prompt
C:\>curl -X POST -H "Authorization: 0cfc07e198c062859f9ab1822a259eb001efe033" -F "file=@/trash.jpg" http://127.0.0.1:8000/model/upload/ {"id":3,"file":"/media/trash.jpg"}
```

*Figure B.5 – Model upload testing*

In Figure B.6, the endpoint “users/logout” is tested successfully by passing the authorization token and the endpoint URL as it can be seen in line 1, this method will not give a response.



```
Administrator: Command Prompt
C:\>curl -X GET -H "Authorization: Token 0cfc07e198c062859f9ab1822a259eb001efe033" http://127.0.0.1:8000/users/logout/
```

*Figure B.6 – User logout testing*