ISCTE IUL
Instituto Universitário de Lisboa

Department of Information Science and Technology

Integration of Mobile Devices in Home Automation with Use of Machine Learning for Object Recognition

Rui Jorge Silva Passinhas

A Dissertation presented in partial fulfillment of the Requirements
for the Degree of
Master in Telecommunications and Computer Engineering

Supervisor:
Rui Miguel Neto Marinheiro, Assistant Professor
ISCTE-IUL

Co-Supervisor:
Paulo Jorge Lourenço Nunes, Assistant Professor
ISCTE-IUL

October, 2019

Resumo

O conceito de casas inteligentes está cada vez mais em constante expansão e o número de objetos que temos em casa que estão conectados cresce exponencialmente. A tão chamada internet das coisas abrange cada vez mais dispositivos domésticos crescendo também a necessidade de os controlar. No entanto existem inúmeras plataformas que integram inúmeros protocolos e dispositivos, de inúmeras maneiras, muitas delas pouco intuitivas.

Algo que transportamos sempre connosco são os nossos dispositivos móveis e com a evolução da tecnologia, estes vieram-se tornando cada vez mais potentes e munidos de variados sensores. Uma das portas para o mundo real nestes dispositivos é a câmara e as suas inúmeras potencialidades. Uma temática que tem vindo também a ganhar enorme relevância é a Inteligência Artificial e os algoritmos de Aprendizagem Máquina. Assim, com o processamento correto os dados recolhidos pelos sensores poderiam ser utilizados de maneira intuitiva para interagir com os tais dispositivos presentes em casa.

Nesta dissertação é apresentado o protótipo de um sistema que integra os dispositivos móveis nas plataformas de automação de casas através da deteção de objetos na informação recolhida pela câmara dos mesmos, permitindo assim ao utilizador interagir com eles de forma intuitiva. A principal contribuição do trabalho desenvolvido é a integração não explorada até então, no contexto da automação de casas, de algoritmos de ponta capazes de superar facilmente os seres humanos na análise e processamento de dados adquiridos pelos nossos dispositivos móveis. Ao longo da dissertação são explorados os conceitos referidos, bem como a potencialidade dessa integração e os resultados obtidos.

Palavras Chave: Internet das Coisas, Casas Inteligentes, Computação Visual, Aprendizagem Máquina, Dispositivos Móveis

Abstract

The concept of smart homes is increasingly expanding and the number of objects we have at home that are connected grows exponentially. The so-called internet of things is increasingly englobing more home devices and the need to control them is also growing. However, there are numerous platforms that integrate numerous protocols and devices in many ways, many of them being unintuitive.

Something that we always carry with us is our mobile devices and with the evolution of technology, they have become increasingly powerful and equipped with lots of sensors. One of the bridges to the real world in these devices is the camera and its many potentials. The amount of information gathered can be used in a variety of ways and one topic that has also gathered tremendous relevance is Artificial Intelligence and Machine Learning algorithms. Thus, with the correct processing, data collected by the sensors could be used intuitively to interact with such devices present at home.

This dissertation presents the prototype of a system that integrates mobile devices in home automation platforms by detecting objects in the information collected by their cameras, consequently allowing the user to interact with them in an intuitive way. The main contribution of the work developed is the non-explored until then integration, in the home automation context, of cutting-edge algorithms capable of easily outperforming humans into analyzing and processing data acquired by our mobile devices. Throughout the dissertation the referred concepts are explored as well as the potentiality of this integration and the results obtained.

Keywords: Internet of Things, Smart Homes, Computer Vision, Machine Learning, Mobile Devices

Acknowledgements

Firstly, I would like to express my sincere thanks to my supervisors, Professor Rui Marinheiro and Professor Paulo Nunes for the availability and promptness in supporting me during the course of this dissertation.

I would also like to thank Instituto de Telecomunicações for providing me the conditions to the realization of all the work developed.

Finally, and not least important, I want to express my gratitude to all my family and friends which helped me through this journey and always kept my motivation levels high. Some of them accompanied me since the beginning of my academic journey and share with me all the difficulties and step backs we all had to deal with to accomplish this goal. Without all of them it would not be possible to complete this dissertation.

Contents

List of Tables

List of Figures

Abbreviations

AES	Advanced Encryption Standard
ANN	Android Neural Network
API	Application Programming Interface
CNN	Convolutional Neural Network
CV	Computer Vision
DL	Deep Learning
DSR	Design Science Research
ESH	Eclipse Smart Home
FHSS	Frequency Hopping Spread Spectrum
HAL	Hardware Abstraction Layer
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
IEEE	Institute of Electrical and Electronics Engineers
IoT	Internet of Things
ITU	International Telecommunication Union
JVM	Java Virtual Machine
ML	Machine Learning
MQTT	Message Queuing Telemetry Transport
OCR	Optical Character Recognition
OS	Operating System
QoS	Quality of Service
RSA	Rivest–Shamir–Adleman
TPU	Tensor Processing Unit
UI	User Interface
URI	Universal Resource Identifier

Chapter 1 – Introduction

In recent years we have been witnessing a technological change of a dimension comparable to the industrial revolution [1]. This is based on the principle that all the objects that surround us are likely to become connectable and form an intelligent network that we can call the Internet of Things (IoT), a network that while connecting all the devices to the Internet allows them to communicate with each other and with the people, changing information according to a set of defined protocols.

IoT can be seen as a bridge between the real and the virtual world since it allows “things” to gather information from the environment and share knowledge [2]. As a result, this translates into improved process efficiency and increased level of automation in tasks performed with these devices.

These developments provide the ability to develop applications that allow us to improve our life quality in the different environments we attend. Thus, by making devices “smart”, we can design solutions for diverse areas such as: logistics, farming, transportation, health, smart environments (home, office) and even at a personal and social level [3].

1.1. Motivation and Framework

Homes are precisely one of the most predominant areas in IoT. By 2020 the number of connected devices is expected to grow exponentially to around 8 billion, with a large slice of approximately 30% [4] being devices in our homes.

These devices, when used in a smart and dynamic way, form what we nowadays call "Smart Homes". With the right automation logic implemented, we can control lights, temperatures, appliances and other devices, set alarms and monitor systems in order to provide comfort and convenience to its residents.

However, with incessant search for innovation and a constant emergence of new technologies, we come across an endless number of protocols and platforms to integrate all these new devices. One of the main obstacles to the development of an IoT is the standardization of this integration. As mentioned in [5], if we imagine a scenario where each car manufacturer used different controls and their drivers were forced to use wheels on one type of car, joysticks on others and buttons on another type, we easily understand the complexity of the problem. If we apply the metaphor for Smart Homes, we can see that in a scenario where we have a vast set of devices connected at home, it becomes difficult to have a protocol or a bridge that connects all of them.

Currently the Institute of Electrical and Electronics Engineers (IEEE) Standards Association in joint efforts with the International Telecommunications Union (ITU) have been working to define a standards framework to help overcome these barriers that prevent IoT from reaching its full potential.

It is in this context that arises the opportunity for solutions based on Open Home Automation Bus (OpenHAB) [6] or OpenRemote [7]. Through the use of open standards, these platforms can integrate almost any type of device without despising the efficiency and benefits to the user. However, the configuration process and everyday use is time consuming and not so intuitive for the average user. This almost ends up narrowing the use of this platform to users with computational background or who are willing to take the time to learn how to use the system [8].

This scenario opens the door to the use of our personal mobile device to ease the interaction with the platform and make its usage more user friendly and intuitive. However, the integration of these devices on the automation platforms is still far from complete, especially considering the amount of information from the sensors that can be collected from the device.

One of the sensors that allows a greater margin of integration is the camera. With the help of Computer Vision (CV) libraries such as the Open Source Computer Vision Library (OpenCV) [9] or image detection APIs such as Cloud Vision [10], there is room to progress towards using the camera as a sensor on the automation platforms and perform image recognition tasks. Furthermore, there are also other sensors in the mobile device that can support intuitive tasks, such as the light sensor and the gyroscope.

1.2. Objectives

The main goal of this dissertation is to promote the integration of mobile devices into current home automation solutions, considering the device as a sensor, in particular the use of its camera in a user-friendly way in order to perform tasks based on image recognition.

Analyzing the related work in the smart homes sector, it is noticeable that an effort is being made in progressively using the mobile device to interact with the home automation platforms and control our home appliances in a user-friendly way. Still and all, there is no bridge between the mentioned subjects and this dissertation proposal aims to fill that gap, providing the device the ability to perform image recognition tasks from an intuitive application and consequently take actions in a smart home environment.

This integration strives to go beyond the scope of the normal home automation mobile applications as far as the interaction with the user happens and as well as the concept of object recognition with real time detection is attached.

Thus, the proposed system consists in a sequence of modules which provide the foundation to the whole work flow that needs to occur since back from the user to the end smart home devices. In a high-level system architecture overview, the system presented consists of a mobile application where the user takes interactions. This application is where the object recognition is performed in real time, analyzing the surrounding environment and enabling the interactions with the devices. Then, the application interacts with a message broker, which connects to the home automation platform and consequently triggers events in the devices themselves, according to the existing rules.

The final objective is validating the prototype implementation analyzing real case scenarios and identifying and interacting with the smart home devices, considering the obtained accuracy and viability.

1.3. Dissertation Organization

After contextualizing the motivation and introducing the scope of the dissertation, Chapter 2 reviews the work already developed, considering the existent solutions and the state of the art itself. In Chapter 3, a high-level system architecture is presented, introducing the modules of the system developed and the choices made for each of them. Later, Chapter 4 describes the major system development stages in order to successfully implement the proposed solution. In Chapter 5, the final solution achieved and its components are described as well as comparisons and tests on the implemented prototype are made. This analysis allows to draw the major conclusions depicted in Chapter 6, which also discusses possible system limitations and identifies topics for future work development.

Chapter 2 – Literature Review

IoT for Smart Homes and Computer Vision are some of the most emerging trends in the 21st century and consequently there are numerous solutions and concepts developed in both domains. However, there are not many proposals that put together these two topics taking advantage of the integration between them. Nevertheless, extensive related work can be found mostly in independent subjects.

Considering the context of IoT presented in Chapter 1 and the above statements, given the fact that home automation platforms provide a certain level of abstraction of the protocols implemented between the platform and the items, this Chapter will focus mainly on the home automation platforms and the integration with the mobile device itself, regarding all the tasks and integrations associated. Section 2.1 reviews the main home automation platforms considered. Section 2.2 discusses the integration with the mobile device and the mobile application as well as the communication with the platform. Section 2.3 focus on the image processing tasks taking in charge by the integration referred and gives an approach on different techniques to object detection, image labeling and processing.

2.1. Home Automation Platforms

With a constant expanding number of devices and sensors available to aid the task of automate and monitoring an everyday home, there is a growing market for solutions to tie them up together. Early 2019, the worldwide household penetration regarding smart devices is 9.5% and is expected to hit 22.1% by 2023 [11], being United States the major contributor, having 33.2% in the beginning of 2019 and expected to almost double that value by 2023.

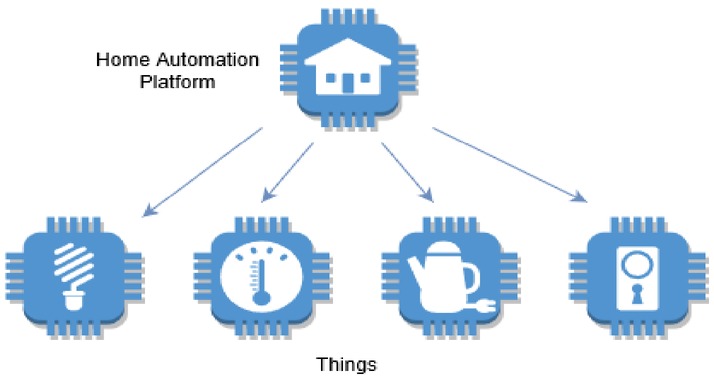


Figure 2.1 - Devices Connected to a Home Automation Platform

For this reason, it is crucial to have the ability to control lighting, air conditioning, heating and other connected devices from a single hub platform as shown in Figure 2.1.

2.1.1. Platform Solutions

There are currently countless solutions on the market that provide almost unlimited possibilities for what we can accomplish. Table 2.1 shows a comparison between some key features of the most popular home automation platforms. A more detailed approach on each one can be found in the subsections below.

Table 2.1 - Home Automation Platforms Key Features

Feature	**Home Automation Platform**					
	OpenHAB	**Domoticz**	**Home Assistant**	**OpenMotics**	**OpenRemote**	**Calaos**
Open-source	Yes	Yes	Yes	Yes	Yes	Yes
Backend Language	Java	C/C++	Python	Python, C	Java	C++, Shell
Web UI	Yes	Yes	Yes	Yes	Yes	Yes
Mobile Apps	Yes (Android and iOS)	Yes (Android and iOS)	Yes (iOS)	No (Under development)	Yes (Android and iOS)	Yes (Android and iOS)
Number of supported devices and integrations	Large	Limited	Large	Limited	Large (with retrofitting)	Medium
Installation	Easy	Medium	Easy	Medium (need to install the modules)	Medium	Easy
Automation Rules	Yes	Yes (with LUA scripting)	Yes	No	Yes	Yes
Community of Users	Large	Large	Large	Medium	Medium	Small
Updates	Slow (but stable)	Slow (for latest devices)	Fast (almost every week)	Average (mainly bug correction)	Regular	Slow
Can run on	Linux, Windows, Mac OS, Raspberry Pi	Linux, Windows, Mac OS, Raspberry Pi	Raspberry Pi (recommended), Windows, Mac OS, CentOS	Specific Modules	Mac OS, Windows, Raspberry Pi, NAS, Debian	x86 and x64 PC, Raspberry Pi, Mele, Cubieboard

2.1.1.1 OpenHAB

OpenHAB [6] is an 100% open-source automation platform built upon Eclipse SmartHome (ESH) IoT framework which supports more than 200 technologies/systems and thousands of devices. It is a flexible solution that allows to integrate these multiple devices and technologies into a single solution with a uniform and customizable Web User Interface (UI). This platform runs on any device capable of running Java Virtual Machine (JVM) and provides the ability to be integrated in other systems using its Application Programming Interfaces (APIs). Being one of the best-known home automation platforms, OpenHAB has a large and well stablished community of users.

2.1.1.2 Domoticz

Domoticz [12] is a lightweight open-source automation system that allows to integrate many devices such as lights, switches, multiple sensors and other third-party integrations. Written in C/C++ with a scalable HTML5 designed frontend, the platform can be accessed both in desktop and mobile devices running in different operating systems including Windows, Apple Unix and even on a Raspberry Pi. It is designed for simplicity and features to send notifications and alerts to any mobile device. The platform configuration is made through a Web interface and the functionalities can be extended with the use of plug-ins. Although being very stable, Domoticz interface is not that intuitive and the supported devices and configurations can be limited.

2.1.1.3 Home Assistant

Home Assistant [13] is an open source automation platform that will track the state of all the devices connected in a smart home using a single user-friendly interface. Putting privacy in first place, the platform stores data locally and away from the cloud. It is developed using Python 3 and Polymer and has frequent updates. The installation process is very simple and tries to connect to all your devices in the first run. With the set of advanced rules, it provides the ability to automate certain actions and simplify people needs in day to day life.

2.1.1.4 OpenMotics

OpenMotics [14] is a slightly different home automation system. Being open-source, this platform offers a complete solution with both software and hardware components that provide full control over devices instead of trying to combine and integrate multiple solutions from different manufacturers. The OpenMotics platform is composed of several hardware components called Modules and each Module has a designated role in the home automation process. It has intuitive interfaces for computers, tablets and smartphones and the data can be accessed in each one of these devices or anywhere in the cloud. Having community scope in mind, the platform aggregates data from different Module groups allowing improved building management.

2.1.1.5 OpenRemote

OpenRemote [7] is an open-source project to integrate, design and manage solutions focused on smart cities, buildings, home automation and health care. In the home automation department, OpenRemote provides the ability to integrate all the devices in a smart home and create a universal remote to control them from your smartphone or tablet. With the use of Open Remote Designer, it is possible to tailor a specific solution to satisfy each user needs, also giving the ability to retrofit devices that were not thought to be smart in the first place and to design specific rules which will control lighting, entertainment, climate and others.

2.1.1.6 Calaos

Calaos [15] is a full stack project designed by a French company built in several layers and including a full Linux Operating System (OS), Calaos OS. This solution includes a server application, a touchscreen interface, mobile apps developed natively both to iOS and Android and even a Web App. It allows to control switches and lights in different rooms, to manage security cameras and to share media across the entire house.

2.1.2. Remarks

The choice of the automation platform is a major concern because it can easily dictate the system limitations and capabilities accordingly to the platform features and possible implementations. Bearing in mind the integration intended and given the descriptions made in the past sections, OpenHAB, Home Assistant and Domoticz are good choices to do further tests because they are all open-source, developed in a well-known easy to learn language, have a strong base community of users, a companion mobile app and can be installed on common hardware like a Raspberry Pi.

2.2. Integration with Mobile Devices

Since the integration with the mobile device is the focus of this Thesis, the device needs to be able to interact with the surrounding environment through a mobile application and to communicate with the automation platform in order to trigger actions on each of the devices connected.

This section will discuss and compare the main related concepts and the solutions available to solve each of the situations stated.

2.2.1 Mobile Application

A mobile application is a software artifact designed to run on a mobile platform. Currently, the biggest players in the mobile operating systems are Android with approximately 75% and iOS with 22% market share worldwide being the remaining part distributed among other less known operating systems or proprietary solutions [16]. Additionally, given the fact that iOS and Android do not have a common framework development tool, there are solutions to multi-platform development like Web Applications, Hybrid Applications and proposals for new taxonomies to unify the cross-platform development [17].

Even tough cross-platforms solutions represent an efficient approach to develop software for multiple operating systems [18], the state-of-the-art work in this area acts as a limitation to integrations with already existing image processing APIs and frameworks which are not supported. As a result, the best solution in the context of this dissertation is to natively develop apps for each of the major operating systems in the market, respectively Android and iOS.

2.2.2 Communication

As mentioned above, communication plays a major part in integrating the mobile device with the home automation platform. The communication protocol to use between the device and the platform (see Figure 2.2) depends heavily on the choice of the platform since each of the platforms referred on Section 2.1 supports a different set of protocols and integrations.

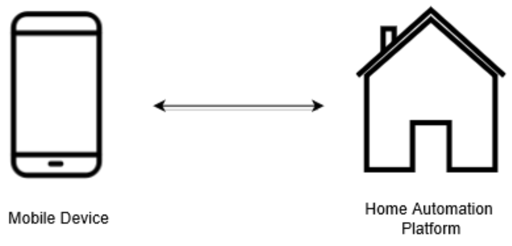


Figure 2.2 - Communication between Mobile Device and Home Automation Platform

Notwithstanding the choice of the platform, the most common binding supported are MQTT, HTTP and Bluetooth. The use of Bluetooth requires the development of a specific implementation to each platform tested as opposite to MQTT and HTTP which have already developed integrations with most of the platforms approached in Section 2.1.

2.2.2.1 MQTT

Message Queuing Telemetry Transport (MQTT) [47] is a lightweight Machine-to-Machine communication protocol designed to be easily implemented on low bandwidth networks. It works on a publish/subscriber architecture (see Figure 2.3) and normally runs over TCP/IP. It was created back in 1999 and maintained its focus on being energy efficient and running on embedded devices using small data packets escalated to many receivers.

MQTT considers two different entities, a broker and a variable number of clients. An MQTT client can act as a publisher or as a subscriber and both connect to the broker. Information is organized by topics and the first one will publish information to a given topic, sending a message to the broker which therefore is responsible to distribute the information to all the clients who subscribed that topic (Figure 2.3).

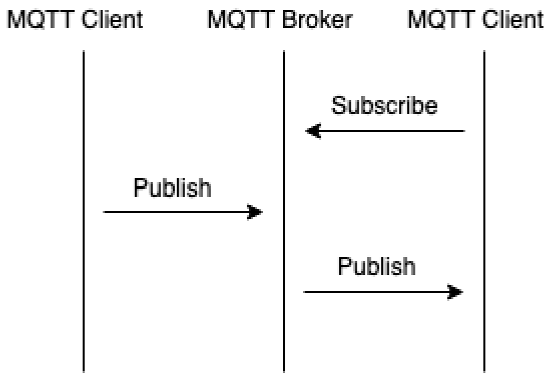


Figure 2.3 - MQTT Publish/Subscribe Architecture

This protocol can specify a Quality of Service (QoS) for each connection which quantifies the guarantee of a specific message being delivered divided by 3 levels: At Most Once (0), At Least Once (1), Exactly Once (2). This QoS considers the two steps of the delivery which are the delivery from the publisher to the broker and the delivery from the broker to the subscriber.

2.2.2.2 Bluetooth

Bluetooth [48] was created in 1994 with the objective of transmitting data in a master-slave architecture based in packets. It is an open technology which uses 79 radio frequency channels in a Frequency Hopping Spreading Spectrum (FHSS) with a changing rate of 1600 times per second. It established as an IEEE standard (802.15.1) in 2005 operating in 2.4 GHz band.

The Bluetooth enabled devices communicate in a master-slave architecture and together form an ad-hoc network of up to 7 slaves per master where the device working as master is responsible to initiate the communication while the slaves listen, letting the master know their address. The master can be communicating with one or more slaves, this way, the communication established can be point to point or point to multipoint, respectively.

2.2.2.3 HTTP

Hypertext Transfer Protocol (HTTP) [19] is an application-level communication protocol based on TCP/IP used since 1990 for communication on the Web. It is a protocol based on a request-response architecture where the Web server responds to requests from the HTTP clients, as presented in Figure 2.4.

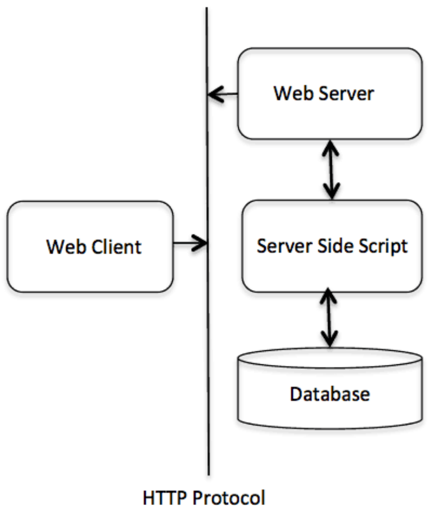


Figure 2.4 - HTTP Client/Server Architecture [20]

The workflow consists on a request sent by the client which includes the Universal Resource Identifier (URI), the protocol version, request modifiers, among other parameters over a TCP/IP connection. Then, the HTTP server is responsible for fiving a response containing the status, message protocol version and informing the success or fail of the operation alongside the meta information and body content of the response. HTTP, despite being simple, is also powerful since it is a stateless, media independent and connectionless protocol.

2.2.2.4 Remarks

As discussed in [21, 22, 23] and stated above, MQTT is proven to be less power consumption, require less resources to be implemented, and less bandwidth providing better quality of service as opposite to HTTP. On the other hand, HTTP provides the ability to compose longer messages, has a larger specification and higher levels of security and interoperability. Thus, regarding the advantages and disadvantages presented, both protocols can be suited to testing.

2.3. Image Processing

This interaction process mentioned in Section 2.2, based on image recognition, is one of the dissertation cores and can be taken in charge with two different approaches. In one hand, it can be implemented with more traditional Computer Vision techniques performing feature extraction over an image, using methods based on edge, corner, color schemes and texture detection to extract as many information as possible and provide the ability to discriminate objects. On the other hand, Machine Learning (ML) algorithms can come in handy, teaching a neural network by uploading a big number of images with lamps and other devices in it and let it do all the work.

2.3.1 Computer Vision

Conventionally, Computer Vision [24] is formally described as the construction of explicit and meaningful descriptions of objects from images. In Figure 2.5 is described an usual workflow using traditional CV algorithms. It implies performing visual recognition tasks in order to classify images and detect objects.

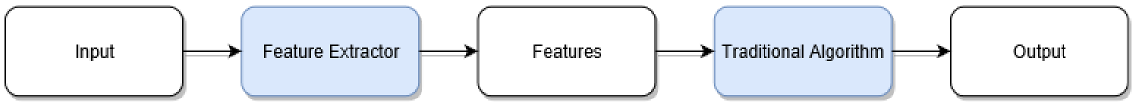


Figure 2.5 - Traditional Computer Vision Algorithm Flow

A recurrent problem in Computer Vision tasks is defining if the images contain a given object or specific feature and this task can be performed with several levels of granularity. These visual recognition tasks can be divided in different categories which are described in detail in the next sub chapters.

2.3.1.1 Image Classification

Image classification [25] can be defined as the process of attributing a classification to an image, based on its content. Given an image as an input, an image classificator will output a class name identified or in some cases the probability of the input passed being a specific class, as exemplified in Figure 2.6. Multiple label classifiers can be created to classify an instance into one of *n* possible classes.

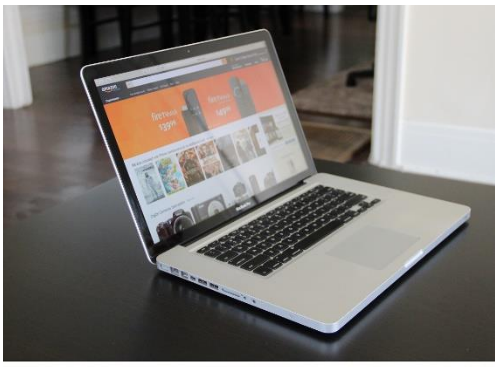


Figure 2.6 - Image Classification Example [25]

2.3.1.2 Object Detection

If the images considered for identification only have one object, a classification model can be easily ran and give right predictions and even if there are two different object intended to be classified inside the same image there are also solutions based on multi-label classifiers but a drawback is not knowing where these objects are inside the image. This scenario is where Image Localization is needed and the concept of Object Detection [26] is introduced.

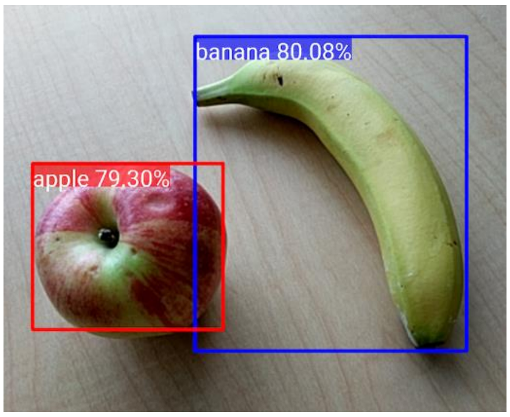


Figure 2.7 - Object Detection Example [26]

As exemplified in Figure 2.7 above, the concept of object detection goes by the ability of knowing the location of the objects detected inside the image alongside the class for each of them. Object detection techniques are usually associated with defining rectangular bounding boxes around the objects identified.

2.3.1.3 Other Techniques

Additionally, to image classification and object detection, there are other more complex computer vision techniques [25] such as object tracking, image segmentation and even semantic segmentation.

For instance, image segmentation is the partition of a given image in various segments which allow to group together a number of pixels with similar properties that represent a particular segment of that image, as exemplified bellow in Figure 2.8. Image segmentation gives thus a better understanding of the different objects inside the image and its real shapes instead of rectangular boxes.

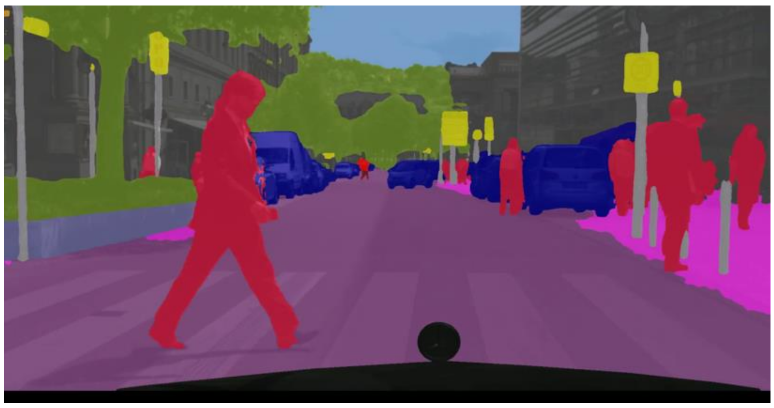


Figure 2.8 - Image Segmentation Example [25]

2.3.2 Machine Learning

The problem with feature extracting to classify an image is that you need to know in advance what to look for and which feature to identify in each one of the images under analysis. This situation grows exponentially with the number of different types of objects.

Here is where Machine Learning algorithms come in handy. For example, if someone wants to identify a lamp in a given image, instead of manually deciding the methods to detect the features desired, like round shapes and filaments, it is easier to teach a neural network by uploading a big number of images with lamps in it and let it perform all the tasks, as shown in Figure 2.9, in comparison to the previous method described in Section 2.3.1.

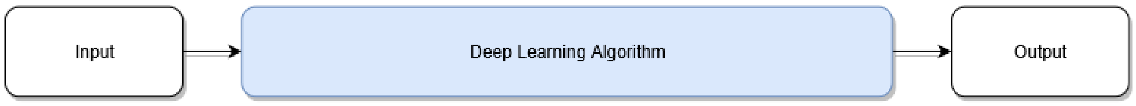


Figure 2.9- Deep Learning Algorithm Flow

Despite these benefits, deep learning algorithms require a large amount of computer resources and need to be trained and tweaked with huge annotated datasets in order to obtain the highest accuracy possible [27].

2.3.2.1 Neural Networks

The main objective of a neural network is to understand data patterns and making decisions based on that pattern detections after analysis. Neural networks are designed to mimic the structure of the human brain, getting predictions from accretions of small data abstractions. These networks underly most of the so-called artificial intelligence systems and have the ability of detecting very complex relations between structured or unstructured data.

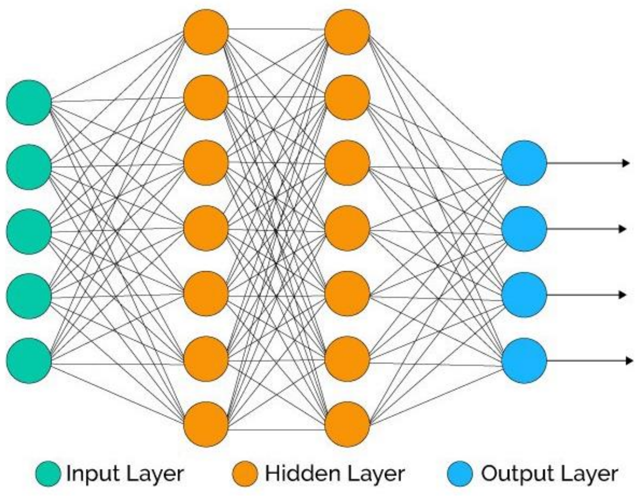


Figure 2.10 - Neural Network Architecture

The base unit on these networks are the nodes, or neurons, which are densely interconnected as depicted in Figure 2.10. The nodes are organized into layers and the network can have different topologies according to the number of layers and the number of nodes per layer. A node or neuron has multiple connections, including one or more weighted inputs which lead to an output generated by internal functions.

2.3.2.2 Deep Learning

Deep learning is a subset class inside machine learning that uses the hierarchical multiple layers of neural networks to handle machine learning processes, enabling this way a nonlinear approach on unstructured data processing. Deep learning is based on artificial neural networks and implements deep neural networks to fields like computer vison, language processing, among others.

A deep neural network is basically an Artificial Neural Network (ANN) with a higher number of layers between the layers that receives the input and the final output layer. Usually Deep Neural Networks are feedforward, which means no data loops back at the layer, flowing forward to the output.

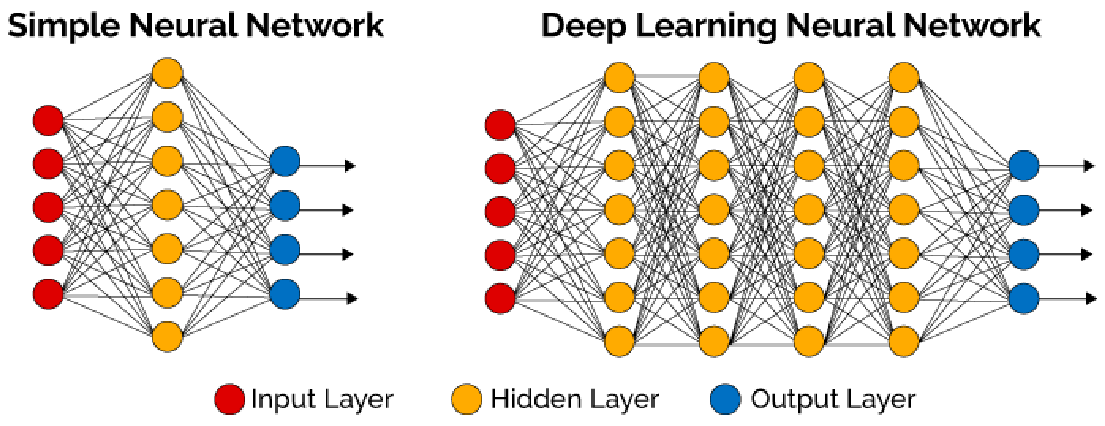


Figure 2.11 - Deep Learning Neural Network Example

A specific class of deep neural networks is the Convolutional Neural Networks (CNNs) which are fully connected multi-layer networks that do not vary in space and meaning that one layer node is connected to all the nodes in the next layer, as presented above in Figure 2.11.

2.3.3 Frameworks and Libraries

Given the advantages and tradeoffs of both approaches [28], state-of-the-art frameworks and libraries used in major implementations for classification, localization, object detection and image segmentation are described below.

2.3.3.1 OpenCV

OpenCV [9] is one of the best-known libraries for computer vision and it has most of the traditional image processing algorithms and methods already built-in. It has Java, C++ and Python interfaces and supports major platforms like Windows, Linux, MacOS and even iOS and Android. Being free to use, it is a mature and fast platform with a large community of users and extensive documentation containing examples for all the platforms.

2.3.3.2 Tensor Flow

Tensor Flow [29] is an open-source machine learning library with a flexible architecture that has implementations for a variety of platforms like Windows, Linux, MacOS and even for Android and iOS. Developed by Google Brain Team, this software allows to develop and train neural networks, providing both high and low level stable APIs in Python and C but also in other languages like Java, Go, or C++.

In 2017, Google launched Tensor Flow Lite allowing to run machine learning models on mobile devices with low latency and fast performance. With Tensor Flow Lite, it is possible to build a new model, retrain an already existing one or even converting an original Tensor Flow model to a compressed and mobile solution. It has portability for iOS, Android and other IoT devices.

2.3.3.3 Cloud Vision

Cloud Vision [10] is a solution developed by google to integrate machine learning vision models with an application. Vision API allows to implement detection features like face detection, optical character recognition (OCR) or content tagging within applications and Auto ML Vision provides the ability to train a custom machine learning model that performs the desired task.

2.3.4 Remarks

Even though Machine Learning (ML) processes are making a revolution in computer vision IoT applications, classic Computer Vision are still very useful and a combined use of both can offer much better results. For example, using basic computer vision techniques for image segmentation and then using deep learning to process those segments extracted instead of the whole frames can result on saving computing resources and reduce identification time. Therefore, there is a lot of space to improve and innumerous scenarios and use cases that can be adapted to the user needs.

2.4. Related Work

In the field of IoT, the authors in [30] have taken a more mobile approach to Smart Homes using IoT. The proposed system aims to access and control devices in a Smart Home using a smartphone app. With the integration of wireless communication and cloud networking, the goal is to provide users with the possibility to control all the electrical smart appliances, devices and sensors using a friendly interface in a smartphone from remote locations. The proposed system is composed of a base station implemented in an Arduino Mega connected to Wi-Fi and multiple satellite stations based in Arduino Uno boards with Radio Frequency modules to communicate with the base station.

The authors in [31] developed a mobile Android app than can access information of all the appliances in a smart home and allows to interact with them, manually or automatically from scheduled events. Implementing both AES and RSA algorithms, the app was designed taking into concern common security issues. The system is composed of two main blocks, an outdoor environment and an indoor environment. The outdoor one consists of the end user and the application cloud server whereas the indoor one has the access point, the hosts and all the nodes. Even though communication between the two environments is done using the Internet with encrypted information, communication inside indoor environment uses Zigbee. The application can be accessed in real-time in any remote location and has notifications, QR Code and Auto-Lock features.

In the matter of Computer Vision, in [32] the authors present an application for object detection based on OpenCV libraries. The object detection system was developed and trained on a Windows machine and implemented on a Texas Instruments embedded platform. It adopted a cascade classifier based on Haar-like feature in order to reduce the computational time and to increase the speed of object detection. The system was trained with a dataset of around 4000 images from different angles and positions.

In [33], even though not related directly to IoT, the author trained his own object detector to accurately detect Raccoons around his house. The system was developed using TensorFlow Object Detection API and was trained with a specific dataset of about 200 racoon images collected and labeled by the author. The training process was done with an object detection training pipeline based on a Single Shot MultiBox Detector [33] network with default settings and adapted to only one class. Since the image dataset was small and few training time was used, the detector does not recognize every single racoon but it can deliver decent results with relatively good accuracy.

Analyzing the related work in the smart homes sector, it is noticeable that an effort is being made in progressively using the mobile device to interact with the home automation platforms and control our home appliances in a user-friendly way. Still and all, there is no bridge between the mentioned subjects and this dissertation proposal aims to fill that gap, providing the device the ability to perform image recognition tasks from an intuitive application and consequently take actions in a smart home environment.

Chapter 3 – System Architecture

The main goal of this dissertation is to promote the integration of the mobile devices into the home automation segment, providing the user the ability to interact intuitively with the system. This integration strives to go beyond the scope of the normal home automation mobile applications as far as the interaction with the user happens and as well as the concept of object recognition with real time detection is attached.

As referred in Chapter 2 there is a lot of work already develop in the subjects of home automation, from energy efficient to fully automated solutions, but the introduction of machine learning in the user interaction with the system is yet to be made and a fair long way from being established.

Thus, the proposed system consists in a sequence of modules which provide the foundation to the whole workflow that needs to occur since back from the user to the end devices. In Figure 3.1, a high-level system architecture is presented in which can be identified the user representation, the mobile cluster, the automation platform aggregate and int the opposite end, the smart devices connected inside home. Each of these modules choices and their respective roles in the entire process will be detailed in the sections bellow whereas their implementation will be fully described in Chapter 4.

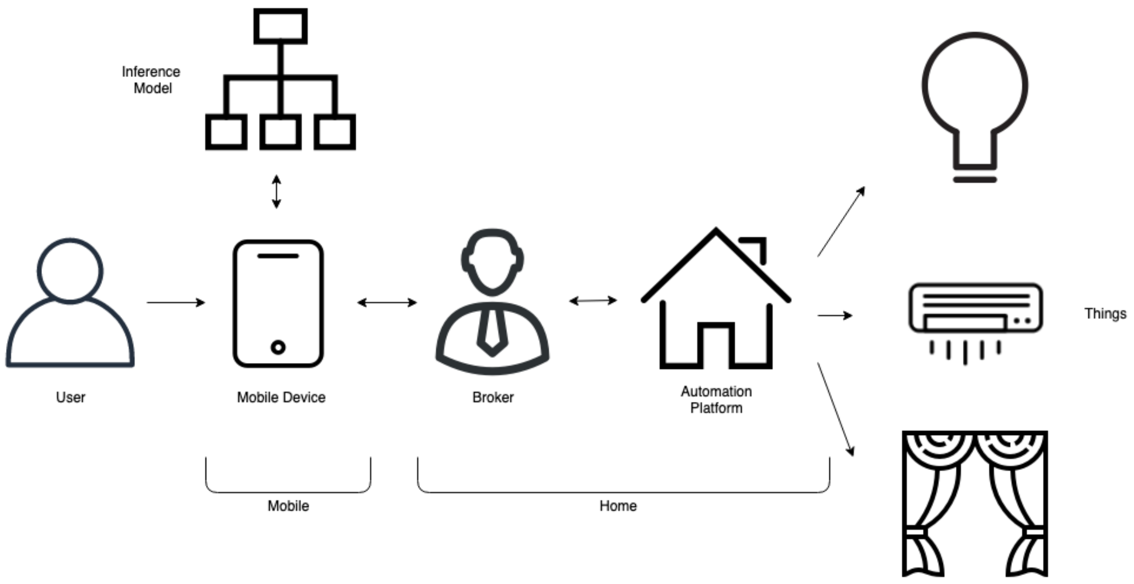


Figure 3.1 - High-Level System Architecture

3.1. Mobile Module

This module serves has the entry point to the user in the whole workflow. All the interactions will happen within an application installed in the mobile device and the actions taken inside it will trigger series of events throughout the system.

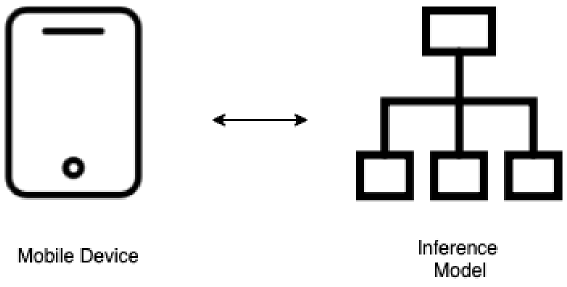


Figure 3.2 - Mobile Module Architecture

As illustrated in Figure 3.2, the mobile component is composed by a mobile application and a re-trained model integrated into the application. This integration occurs in real time providing the user instant feedback and therefore, according to the scenario, specific commands and actions will be sent through the channel of communication existing between this module and the home automation one. Both components play a major role in the system responsiveness and utility and are described in the subsections ahead.

3.1.1. Mobile Application

The mobile application will be installed in the user device and its main role is to collect user interactions, interpret the data collected and stablish the communication with the automation platform.

As discussed in Section 2.2 and having in mind the market share, ease of use and accessibility, the most reasonable choice in the given context is to have an Android application. Developing natively an iOS application would require specific hardware and would reach a smaller community of users than the Android. Moreover, the choice of a hybrid solution would create limitations to future integrations of frameworks and consequently narrow the use case scenarios.

Therefore, with the choice of the Android operating system to the developments, it can be guaranteed that a large number of devices would be compatible, a vast number of users could be reached, and the communication process would not be limited by software restrictions.

This application must furthermore allow the communication to be established between the mobile device and the home automation platform. To do so, it will need to play the client role in a communication protocol between both, sending information to the platform. Since Android supports a wide variety of integrations with communication plugins and software, this choice guaranteed that a close to optimal solution could be found. The communication scheme and implementation will be further detailed.

3.1.2. Inference Model

Being the main purpose of the integration proposed, the ability to use the mobile device sensors to interact intuitively with the smart devices. One of the major and biggest information collectors which can be found in every mobile device is the camera. So, to take advantage of all the potentialities, a way of interpreting the data collected by the camera sensor needed to be found.

As discussed in Section 2.3, a possible solution was to implement traditional computer vision algorithms, using feature extraction and pattern recognition processes to manually identify objects. Notwithstanding the fact of these techniques have proven themselves to be reliable and effective, a more futuristic approach was taken in consideration.

Here is where the Machine Learning component comes into play. The use of a machine learning library compatible with the architecture proposed, able to load and process data, build, train and re-use models with easy deployments was the solution. This way, the solution used in the interpretation component relies on the Tensor Flow framework [25].

In addition to providing the ability to re-train a neural network without the need of sophisticated and powerful hardware, the biggest overall advantage of this tool is the allowance to run machine learning models on mobile devices within a compressed and mobile solution, Tensor Flow Lite.

The use of on-device machine learning allows a simpler system architecture, without the need of executing consecutive server calls to evaluate information. That would require constant data streaming resulting in more energy consumption, higher latency and extra processing with back and forth communication also having to consider the possibility of data loss in it.

As far as the solution architecture goes, Figure 3.3 presents the components stack involved in the process. On top, the Java and C++ application programable interfaces are responsible for loading and invoking the interpreter which in turn executes the model using a set of kernels. In Android versions superior to 8.1, Android Neural Networks API (NN API) [34] is supported and it allows to efficiently distribute the computation across device processors and benefit of running hardware acceleration through Android Neural Network Hardware Abstraction Layer (NN HAL). If none of these are available, normal CPU execution will run.

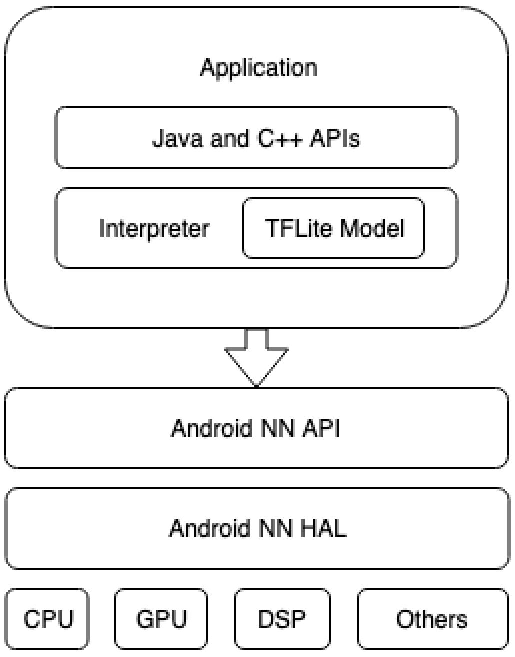


Figure 3.3 - Inference Model Architecture Stack Integration

The process of obtaining the Tensor Flow Lite model which will be loaded into the interpreter will be described in de system implementation sections ahead.

3.2. Communication

The connecting point between the main module described in Section 3.1 and the later one described in Section 3.3 is the communication established between both. As stated in Section 2.2.2, the communication protocol to use between the device and the platform depended heavily on the choice of the automation platform and the application ecosystem since each platform and operating system supports a different set of protocols and integrations.

This way, considering the advantages and disadvantages already presented, MQTT is the most suitable choice in this scenario, providing a solution to assure communication between the two modules that is widely used across the IoT environments with low bandwidth, low latency and good performance.

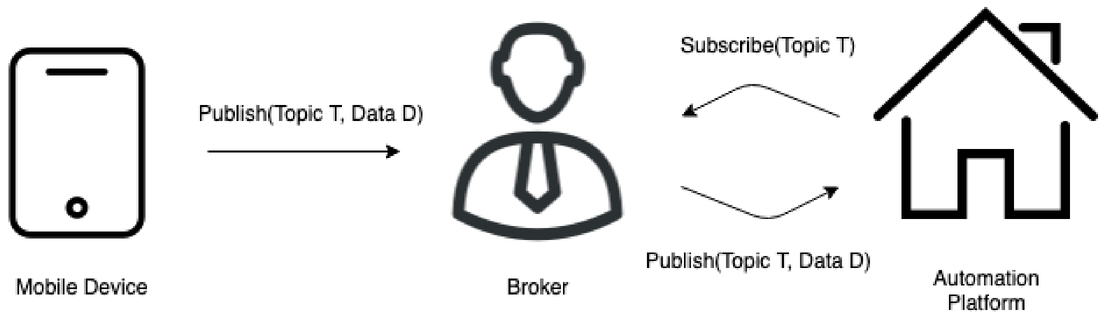


Figure 3.4 - Communication Architecture Workflow

Within the system architecture context, and given the MQTT principles, one component will act as a publisher, one as a broker and the other as a subscriber. The most logical way of implementing the protocol architecture in this scenario is being the mobile device the subscriber responsible of publishing data to a certain topic which is subscribed at the automation platform end. By doing so, the mobile device can constantly push updates on state changes and user interactions knowing that these will be received on the automation platform listening for data in the specific topics subscribed.

3.3. Home Module

The last but not the least important element in the proposed system architecture is the home module. This one houses two main components, the broker, responsible for the communication, and the home automation instance, responsible for the integration of the smart devices and sensors. As displayed in the Figure 3.5, both of them are housed inside the same installation platform.

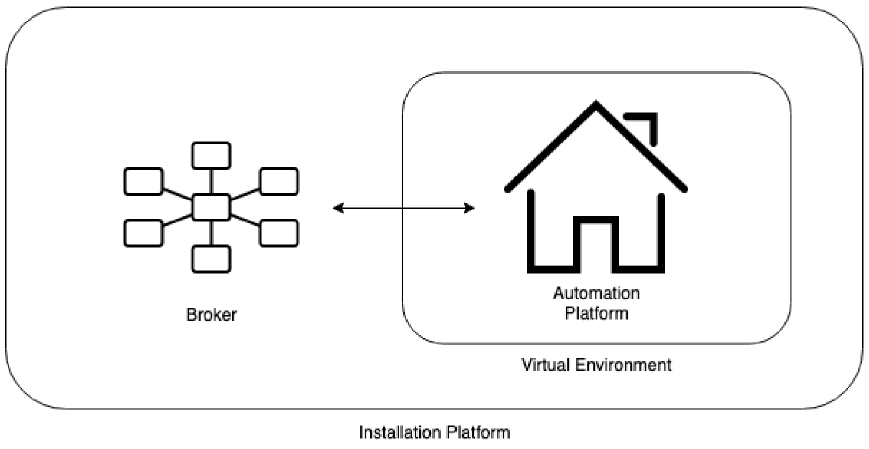


Figure 3.5 - Home Module Architecture

One major concern was to have the broker and the automation platform installed in the same hardware so the need of extra hardware could be avoided. Consequently, the choice of all three components was made having in mind the need of each one being compatible between them.

Keeping in mind the comparisons performed in Section 2.1 and the choice of the communication protocol made, a suitable solution to integrate both elements was to install them inside a Raspberry Pi. Therefore, giving the reduced cost and the ease of installation and maintainability, this whole module is based on a Raspberry Pi 3B+ [35] where the MQTT broker and the Automation Platform live. A detailed description of them is made in the subsections bellow.

3.3.1. Broker

The choice of MQTT as the communication protocol resulted in the need of having a broker to manage all the publish and subscribe calls and therefore maintain the communication between the elements involved.

The MQTT Broker solution could be managed in three different ways: the first one relies on having a public broker providing the services, the second one is to have a private broker and the third one implies using an in-platform broker, when available. Since in a public broker, any device or entity can publish and subscribe to any topic on it, and that most home automation platforms analyzed in Section 2.1.1. did not have a broker by default, the securest and easiest way to integrate this node into the architecture was relying on a private broker where only the devices with given permission can publish and subscribe to the topics managed by that broker.

In order to do this integration, the chosen MQTT broker was Mosquitto [36]. As an open-source lightweight solution widely used on IoT messaging and additionally able to be easily installed on a Raspberry Pi, Mosquitto provides the tools needed to act as the communications mediator.

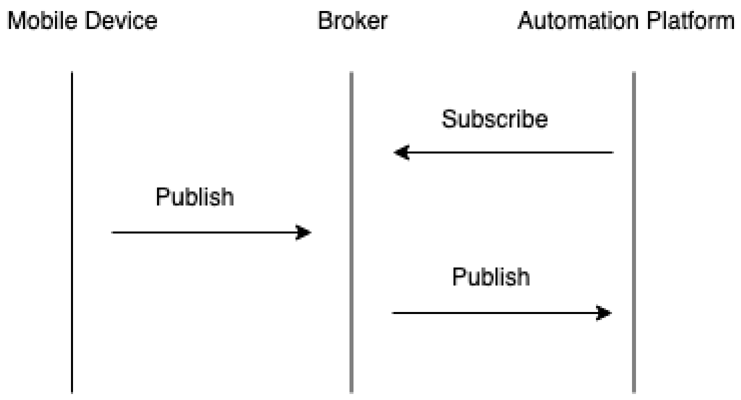


Figure 3.6 - Broker Communication Role In System Architecture

Figure 3.6 represents the Broker placement within the system architecture, receiving subscribing requests from the Automation Platform and therefore delivering the messages there upon a data publish receipt from the Mobile Device, consequently establishing the connection for data transfer between both.

3.3.2. Automation Platform

Since the automation platform is a core element of the system, it plays a key role on its potential and limitations. Architecture wise, as referred in the beginning of this chapter, the home automation platform is installed inside the Raspberry Pi alongside the MQTT broker also already mentioned.

Bearing in mind the integration intended and given the statements conclusions obtained in Section 2.1, Home Assistant [37] is a good choice to implement because it checks all the boxes: is open-source, developed in a well-known easy to learn language, has a strong base community of users and can be installed on a low processing power and costless device like the Raspberry Pi.

The installation consists in a Home Assistant instance, housed in the Raspberry Pi, running natively on a virtual Python environment over the Raspian OS. As illustrated in Figure 3.7, the platform will also have the Smart Home Devices connected to itself, being able to send commands, perform operations and state changes accordingly to the information sent by the user from the mobile device and transmitted through the broker. The whole implementation and integration process are described in detail in Chapter 4.

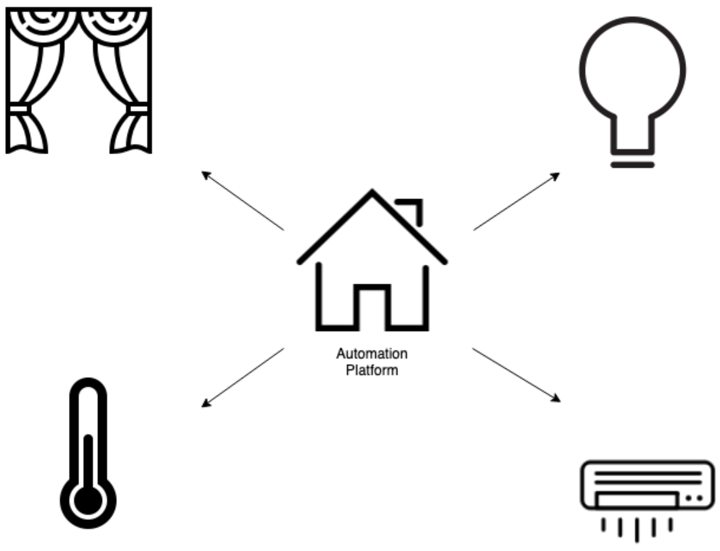


Figure 3.7 - Smart Devices Connected to Home Automation Platform

Chapter 4 – System Implementation

In order to reach the concept solution proposed and implement the system architecture described in the previous chapter, an iterative and objective-centered approach development was taken. This way, having the desired solution defined and bearing in mind what already existed and the limitations and possibilities of the integration proposed, the design and development process was divided into a set of stages which are stated along this chapter.

At first, the machine learning component was developed and converted into the desired format. After that, the mobile application responsible for integrating the model in real time was built. Therefore, the automation platform was installed and configured and consequently the communication processes where implemented. These four major stages of system development and implementation are the keys to successful operation of the solution proposed and are described in the Sections bellow.

4.1. Inference Model

To implement the architecture solution described in Section 3.1.2 and having in mind the conclusions made in Section 2.3, a machine learning model had to be trained, converted and later deployed on the mobile device.

Considering the visual perception task categories already described, it is clearer that a more complex approach like object detection or even image segmentation would generate more precise results than image classification only. As the complexity of the approach increases, the complexity of the training process also grows, and image segmentation would be overkill to have in a context like the one presented. This way, for the sake of this dissertation, an object detection first approach was considered ideal to fulfill the requirements implicit.

Unfortunately, even though the use of object detection instead of image classification would result in a better overall solution, this approach has come not to be feasible since the cost and hardware requirements attached. In any case, the first training approach will be briefly described before the one taken in the sub sections bellow.

4.1.1. Collecting and Preparing the Dataset Broker

Regardless of the final solution, a custom ML model training process requires an image dataset as an input. As stated before, for the model to perform in a certain scenario, it will have to be trained with enough data to learn the patterns and features desired. The amount of images provided, the number of classes existent and variety of the images inside the same class are all going to influence the final model precision and accuracy. Logically, a larger number of images and various angles, brightness and scales of the same object would return better results during the training processes applied and described along Section 2.3.

In order to start the initial approach considering object detection, a dataset of images divided in 3 groups, exemplified in Figure 4.1, was gathered. The training process could have been done with many classes and devices but for the purpose of the implementation and to prove the main concept, the choice of 3 objects that can be found in most of so-called smart homes was made:

- Bulbs – 608 images
- Air Conditioners – 508 images
- Window Shutters – 808 images

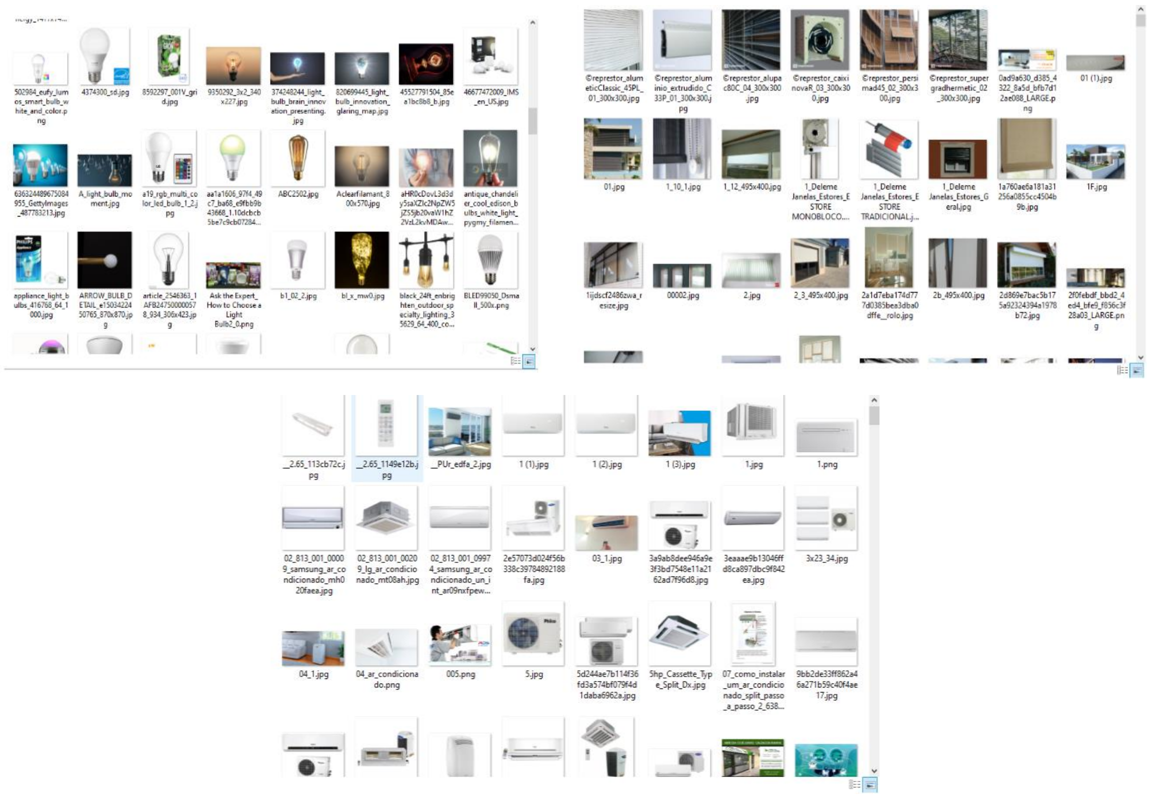


Figure 4.1 - Image Dataset for Training

Manually preparing the dataset for the training process is the most time-consuming part since after collecting all the images, all of them have to be filtered, labeled and consequently exported into TFRecord format to be interpreted by TensorFlow.

The filtration consists in verifying if all the images are not too large for the training pipeline, maintaining the average size below 600x600 to prevent memory related problems and that all of them are in PNG or JPEG format which are the supported ones. The labeling part is done by identifying inside the image with a surrounding box where the object is, in other words, defining the minimum and maximum x and y coordinates, as illustrated in Figure 4.2, which will be therefore passed with the image to the model.

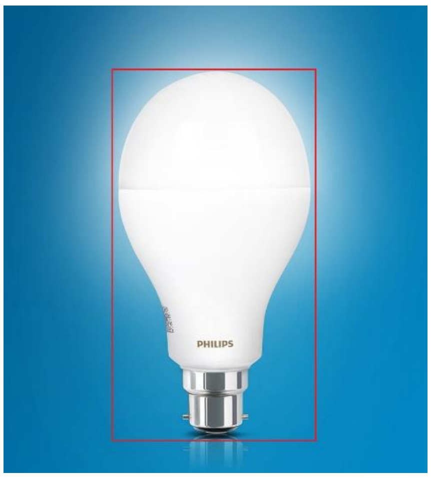


Figure 4.2 - Image Labeling Example

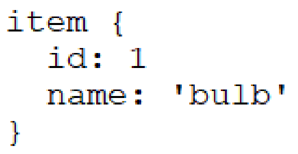
Since the labeling part is a very slow process, a tool called Labelbox [38] was used. This tool basically provides an in-browser user interface to draw rectangles along the dataset uploaded and after that allows to export the information already prepared into TFRecord files.

4.1.2. Training Elements

After gathering all the data needed and obtained the final TFRecord files, there were several choices to be made that would outline the path from here forward. To begin the training process, the following elements were needed:

- TFRecord Files – Containing the dataset provided and labeled above;

- Label Map – Containing the classes used in the training, as illustrated partially in Figure 4.3;



```
item {
  id: 1
  name: 'bulb'
}
```

Figure 4.3 - Label Map Example

- Model Config File – A training pipeline is needed to define the type of model being trained, the parameters utilized in the training, the evaluation metrics and the dataset inputs. This is where are also defined, for example, the learning rate, memory configurations, training steps and batch size;
- Pre-Trained Model – Training a model from scratch could take several days and a lot more images that the ones used. An already trained model can be used as a checkpoint for transfer learning and to retrain the final layers providing the data wanted for the model to perform the recognition process it was already intended to;

Having the elements needed, the next implementation task was to train the actual model. The training process could be done in 2 different ways: the first one is locally, and the second use is using Cloud Tensor Processing Units (TPUs) based solutions. Subsequently, the training can be done from scratch or using a pre-trained model as a base.

4.1.3. Training Process

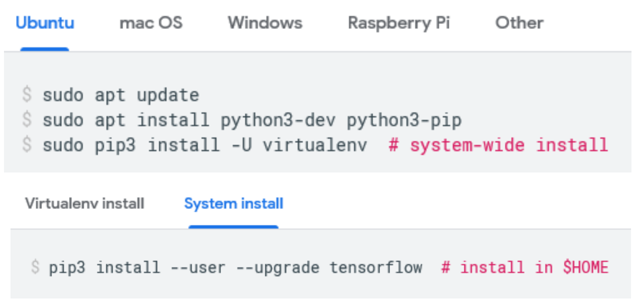
Here is where the unfeasible approach due to the limitations referred in the beginning of the chapter made an alternative but also functional path to be taken. To train the model locally, a lot of computational power was required for intense GPU in order to obtain an acceptable and realistic time and precision of training. Using a Cloud solution those hardware limitations are no longer real since options like Google Cloud TPU [39] provide the ability to run state of the art machine learning models with performances reaching the 100 Petaflops mark with the latest versions.

Despite being a step back compared to object detection, an approach using image classification solved both the problems since could be trained in a less powerful machine, using the dataset and files already prepared but with an equivalent use of the TensorFlow API returning similar results. The result of the model evaluation thus became more limited giving the fact that it returns only a label from an image instead of the coordinates of that same object identified inside the image.

4.1.3.1 Environment Configuration

In order to advance in the implementation, a training environment had to be set. TensorFlow can run both on Windows and Linux but since the installation process, dependency management and execution are easier and smoother on Linux systems, the base OS working environment used to support the implementation was Ubuntu – Version 14.0.

Therefore, TensorFlow was installed via normal Python development environment with Pip Installs Python (pip) which is a cross-platform package manager for installation of Python packages, as presented in Figure 4.4. Although it is also recommended to configure TensorFlow inside a virtual Python environment to isolate package installations from the system, there was no need to have that precaution here because the environment is only intended to perform the tasks related to this dissertation.



```
$ sudo apt update
$ sudo apt install python3-dev python3-pip
$ sudo pip3 install -U virtualenv  # system-wide install
```

```
$ pip3 install --user --upgrade tensorflow  # install in $HOME
```

Figure 4.4 - Python Environment Installation Commands [40]

At the time of the implementation, the TensorFlow version installed was 1.12.0 although there is now a 2.0 stable version already released bringing some overall improvements to the process.

4.1.3.2 Network Re-training

After setting up the environment, the next step was the re-training process. This one was done using MobileNets [41] which are a set of computer vision models optimized to TensorFlow, designed to obtain high accuracy using limited computation power and restricted resources, building this way light weight convolutional neural networks.

A MobileNet is configurable with 2 hyper-parameters, input image resolution and relative size compared to the largest MobileNet, that scale the relations between accuracy and latency. Logically, with the choice of a bigger image resolution results in a more time consuming but more accurate model. Under this dissertation scenario, the default parameters were maintained, having an input image resolution of 224px and a 0.5 fraction of the model. These 2 parameters were passed inside Linux shell variables, as presented in Figure 4.5.

```
$ IMAGE_SIZE=224
$ ARCHITECTURE="mobilenet_0.50_${IMAGE_SIZE}"
```

Figure 4.5- Linux Shell Variables Representing the Hyper-Parameters

Therefore, the model used was *MobileNet_v1_0.50_224*, an intermediate solution, based on an ImageNet pre-trained classification checkpoint and considering the tradeoff between accuracy and latency with 150 Million Multiply-Accumulates (MACs) and 1.4 Million Parameters [42].

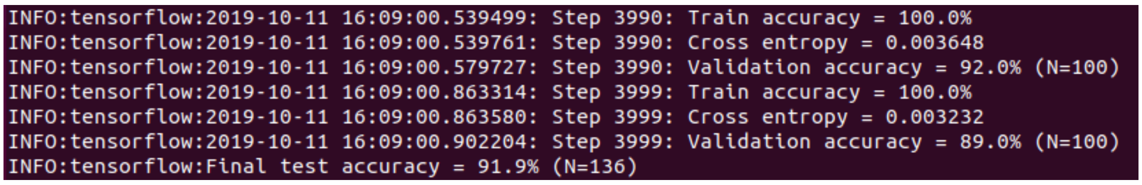
To begin the training, a Python Script obtained from TensorFlow repository was used. The script *retrain.py* is responsible to download the pre-trained model and consequently add the new layer to be trained on the dataset given. The default number of iterations (4000) was used and the script was executed (see Figure 4.6) with the remaining parameters passed:

```
ubuntu:~/Documents/Final/$ python -m scripts.retrain \
>   --bottleneck_dir=tf_files/bottlenecks \
>   --model_dir=tf_files/models/"${ARCHITECTURE}" \
>   --summaries_dir=tf_files/training_summaries/"${ARCHITECTURE}" \
>   --output_graph=tf_files/retrained_graph.pb \
>   --output_labels=tf_files/retrained_labels.txt \
>   --architecture="${ARCHITECTURE}" \
>   --image_dir=tf_files/iot_photos/
```

Figure 4.6 - Command to Run the Training Process

- bottleneck_dir – Path to bottleneck layer files;
- model_dir – Path to *pb* file, label map and *pbtxt* file;
- summaries_dir – Path to TensorBoard summaries log;
- output_graph – Location to save the trained graph;
- output_labels – Location to save the trained graph labels;
- architecture – Model architecture used;
- image_dir – Path to the labeled images folders;

The re-training process took long time to complete but at the end, after analyzing all the images, calculating the bottleneck values and feeding the input to the final classification layer, the Script output reported a Final test accuracy of 91.9%, as presented in Figure 4.7 below.



```
INFO:tensorflow:2019-10-11 16:09:00.539499: Step 3990: Train accuracy = 100.0%
INFO:tensorflow:2019-10-11 16:09:00.539761: Step 3990: Cross entropy = 0.003648
INFO:tensorflow:2019-10-11 16:09:00.579727: Step 3990: Validation accuracy = 92.0% (N=100)
INFO:tensorflow:2019-10-11 16:09:00.863314: Step 3999: Train accuracy = 100.0%
INFO:tensorflow:2019-10-11 16:09:00.863580: Step 3999: Cross entropy = 0.003232
INFO:tensorflow:2019-10-11 16:09:00.902204: Step 3999: Validation accuracy = 89.0% (N=100)
INFO:tensorflow:Final test accuracy = 91.9% (N=136)
```

Figure 4.7 - retrain.py Final Output

For each of the training steps (4000 in this case) a 10 images set is chosen randomly to be fed into the final layer to obtain predictions, which are afterwards compared to the initial training labels and therefore updated with a backpropagation method. The idea of the backpropagation algorithm is, based on the calculation of the error occurred in the output layer of the neural network, to recalculate the value of the weights of the last layer of neurons and thus proceed to the previous layers, from back to front, that is, to update all the weights of the layers from the last one until reaching the input layer of the network, for this doing back-propagation the error obtained by the network.

4.1.3.3 Training Summary

Before the training execution, a monitoring tool included in TensorFlow called TensorBoard was launched in background. This process was running alongside the training one to monitor a series of training parameters.

Consulting TensorBoard, during and after the training, a set of outputs could be evaluated:

- Accuracy – Divided in training accuracy and validation accuracy, these values represent, respectively, the percentage of images labeled correctly and the validation precision on a set of images chosen. In Figure 4.8, the accuracy, represented in the y-axis, is a function of the training progress, represented in the x-axis. The orange line represents the training accuracy of the model while the blue line exhibits the validation accuracy. As the validation accuracy remained the same as the training accuracy increases, we can say that the model did not entered in overfitting which is a scenario when the model is learning more of the training data proprieties than the data patterns itself.

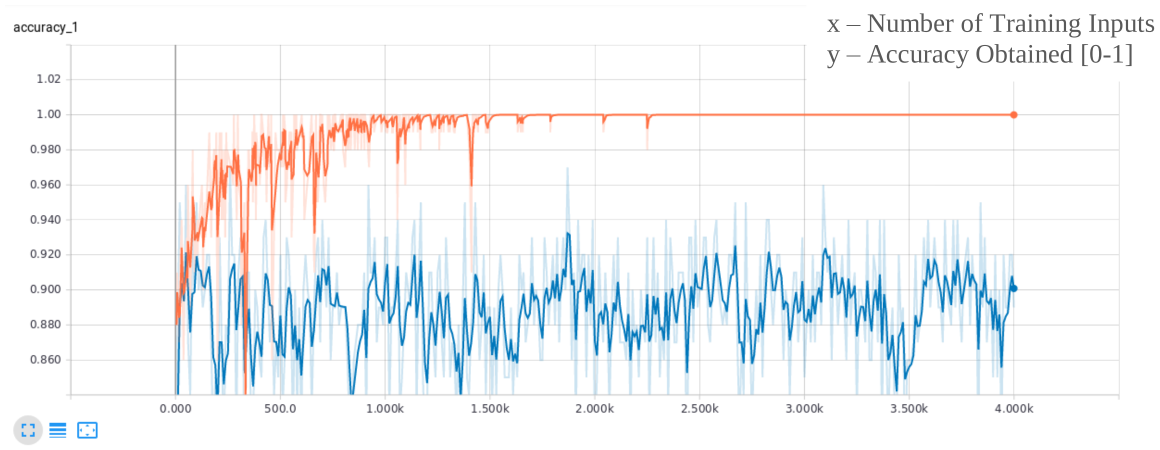


Figure 4.8 - Training Accuracy (TensorBoard)

- Cross Entropy – In short, cross entropy is a positive loss function which tends to zero as the neuron improves computation of the desired output, *y*, for all training inputs, *x*, as represented in Figure 4.9;

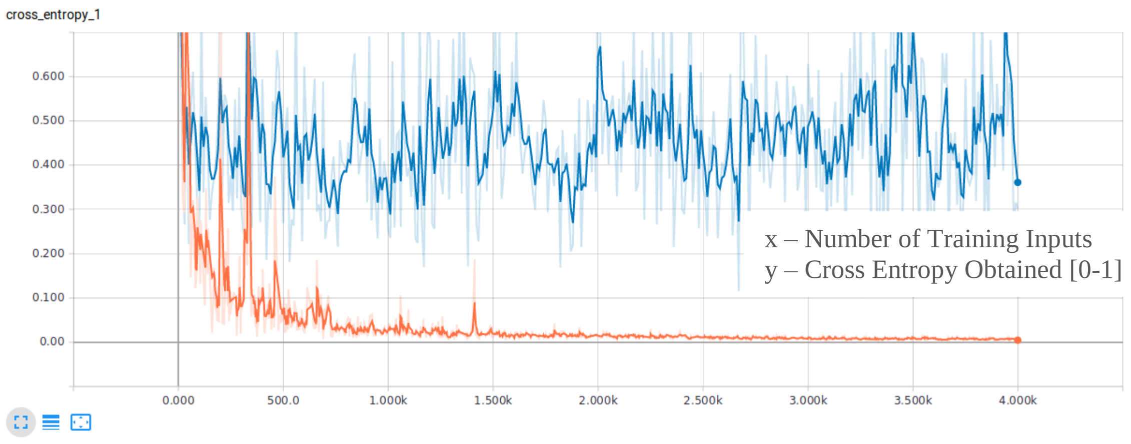


Figure 4.9 - Training Cross Entropy (TensoarBoard)

After running all the training process, a final script was run to test the accuracy of the model evaluation which returned the value of 91.9% as referred in the previous section. This number translates the overall performance of the model in a real classification scenario and since the training was done on only 3 classes, a high accuracy could be obtained.

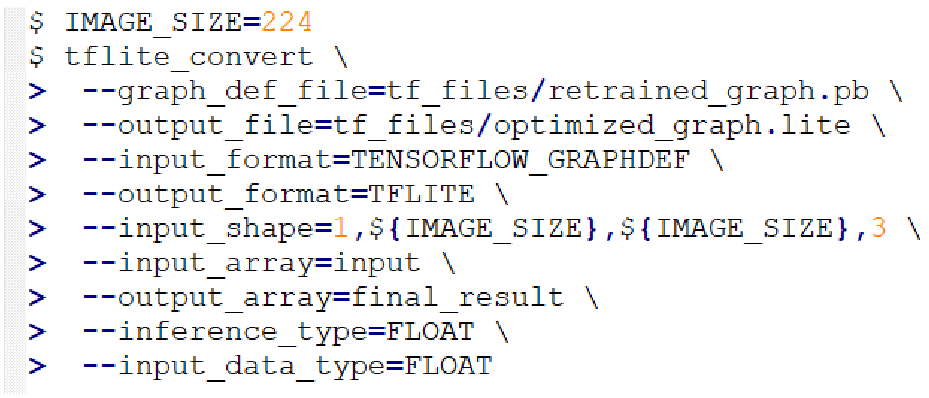
4.1.4. Exportation and Conversion

After obtaining the final re-trained graph from the previous training output, the model needed to be converted and optimized to run on the mobile device. As mentioned before, the process will use TensorFlow Lite and its tools, namely, a TFLite Converter and a TFLite interpreter. To set the python environment for the conversion, an image manipulation tool built over Python Image Library (PIL), PILLOW [43] had to be installed.

The inference graph was then converted using the TensorFlow Lite Optimizing Converter, tflite_convert. It is part of the TensorFlow installation and is easily ran as a command line script. This tool was responsible for optimizing and converting the model, consequently outputting a model in TFLite format.

Concerning the optimization part, while TensorFlow uses Prtotocol Buffers to optimize the generate ProtoBuffer file, TFLite uses FlatBuffers to do so. FlatBuffers [44] is an efficient cross platform serialization library which does not need a parsing/unpacking step to directly access data, allowing them to be memory mapped and consequently achieve faster speed retrieving pages from the model file and without killing the process when low on memory.

Therefore, the tflite_convert program was ran with the command presented in Figure 4.10 bellow and the remaining parameters passed:



```
$ IMAGE_SIZE=224
$ tflite_convert \
>  --graph_def_file=tf_files/retrained_graph.pb \
>  --output_file=tf_files/optimized_graph.lite \
>  --input_format=TENSORFLOW_GRAPHDEF \
>  --output_format=TFLITE \
>  --input_shape=1,${IMAGE_SIZE},${IMAGE_SIZE},3 \
>  --input_array=input \
>  --output_array=final_result \
>  --inference_type=FLOAT \
>  --input_data_type=FLOAT
```

Figure 4.10 - Model Conversion Using TFLite

- graph_def_file – Path to the file containing the model generate;
- output_file – Path to the output file;
- input_format – Input file format;
- output_format - Output file format;
- input_shape - Shapes corresponding to --input_arrays, colon separated;
- input_array - Names of the input arrays, comma-separated;
- output_array - Names of the output arrays, comma-separated;
- inference_type - Target data type of real-number arrays in the output file;
- input_data_type - Target data type of real-number input arrays;

After the script execution, the optimized_graph.lite file was generated under the output path defined.

4.2. Mobile Application

After generating the .lite file, the customized model was then prepared to be integrated inside the Application. As referred above in Chapter 3, the application is an Android Application and it was developed using Android Studio, Version 3.2, later updated to Version 3.4. It has a minSdkVersion of 21, which stands for Android 5.0 – Lolipop, granting minimum compatibility of approximately 88.2% of the devices [50].

The official TensorFlow example application [45] was used as a base since it contained all the Classes and Libraries needed to implement the image classifier pretended. Therefore, in a general overview, the app was then built to classify what is captured from the device back camera based on the inference model interpreted.

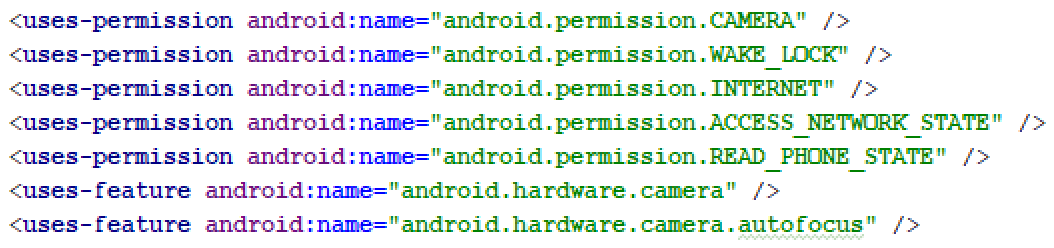
The application has to handle two main tasks: the interpretation of the model in real time and the communication with the forward elements of the architecture. Both of these and their implementation are described in the subsections bellow.

4.2.1. Model Interpretation

The first step was to integrate the model already trained, optimized and converted into the TFLite interpreter. The development process could be tested either on a real Android device or in an Android emulator of choice (including the Android Studio Emulator) but using the emulator would imply to use the computer camera and to simulate other real aspects so, a physical device was used during the whole process, making easier to capture images from the camera sensor and handle that information.

4.2.1.1 Configuration

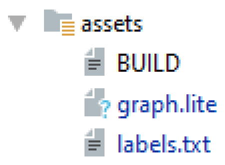
Initially, the permissions required were (shown in Figure 4.11) placed under AndroidManifes.xml. These configurations were needed to allow the device to build TensorFlow dependencies and to enable Camera, File System and Internet access, among others.



```
<uses-permission android:name="android.permission.CAMERA" />
<uses-permission android:name="android.permission.WAKE_LOCK" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-feature android:name="android.hardware.camera" />
<uses-feature android:name="android.hardware.camera.autofocus" />
```

Figure 4.11 - Permissions and Features in AndroidManifest.xml

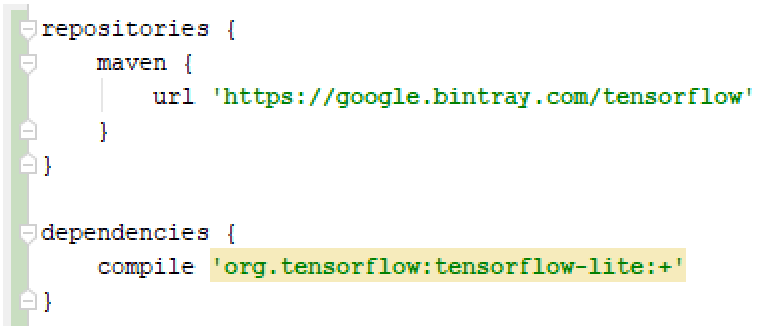
The model file in TFLite format optimized and converted in the previous section, alongside the retrained labels file also generated were placed inside the main Assets folder (see Figure 4.12) from where they will be loaded to the application execution.



```
assets
    BUILD
    graph.lite
    labels.txt
```

Figure 4.12 - Resources in Android Assets Folder

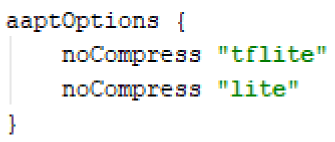
Since the application needs to use a pre-compiled version of the TFLite Android Archive (AAR) the dependencies and the Maven TensorFlow bintray Repository where the archives are hosted (see Figure 4.13) need to be added to the app Module build.gradle file. These will import the AAR which is similar to a JAR file import but in addition to Java classes and methods it also allows to include activities, drawables and layout resources.



```
repositories {
    maven {
        url 'https://google.bintray.com/tensorflow'
    }
}

dependencies {
    compile 'org.tensorflow:tensorflow-lite:+'
}
```

Figure 4.13 - App Module Gradle Dependencies and Repositories

Giving the FlatBuffers serialization, the inference model will be mapped into memory and cannot be compressed. Therefore, the project had to be instructed not to compress the model or the model related files. To do so, the instructions presented below in Figure 4.14 also had to be added to the app Module build.gradle file. The instruction block was placed inside the Android brackets and uses the Android Asset Packaging Tool Options (aaptOptions) containing the instructions not to compress neither the tflite nor the lite file formats.



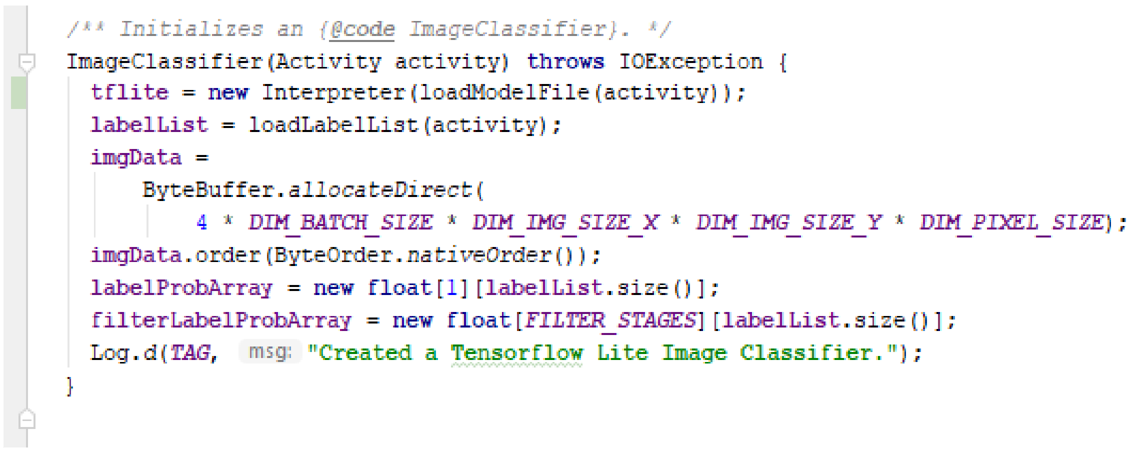
```
aaptOptions {
    noCompress "tflite"
    noCompress "lite"
}
```

Figure 4.14 - App Module Gradle AAPT Options

4.2.1.2 TFLite API

The implementation of the TFLite API into project is contained inside ImageClassifer.java. As the name suggests, this class is going to be responsible for classifying images using TensorFlow Lite and there are two main focus points in the API implementation: (i) the initialization process when instantiated the class, and (ii) the model run itself.

In the first one (see Figure 4.15), a TFLite Interpreter is initially created with a MappedByteBuffer passed as an argument. The MappedByteBuffer is generated in the method loadModelFile(Activity activity) where the model in .lite format already placed in the assets folder is read as a input stream and then mapped in the file channel.

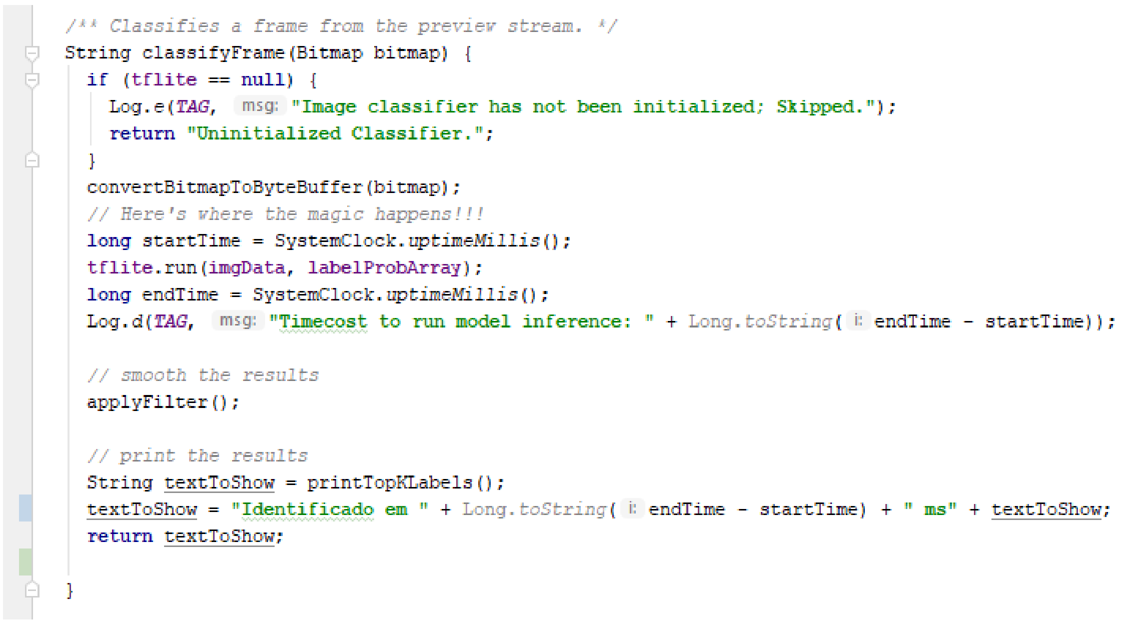


```
/** Initializes an {@code ImageClassifier}. */
ImageClassifier(Activity activity) throws IOException {
  tflite = new Interpreter(loadModelFile(activity));
  labelList = loadLabelList(activity);
  imgData =
      ByteBuffer.allocateDirect(
          4 * DIM_BATCH_SIZE * DIM_IMG_SIZE_X * DIM_IMG_SIZE_Y * DIM_PIXEL_SIZE);
  imgData.order(ByteOrder.nativeOrder());
  labelProbArray = new float[1][labelList.size()];
  filterLabelProbArray = new float[FILTER_STAGES][labelList.size()];
  Log.d(TAG,  msg: "Created a Tensorflow Lite Image Classifier.");
}
```

Figure 4.15 - ImageClassifier.java Constructor

Afterwards, the categories labels are loaded to a list, an input data buffer is created to receive image data and an output buffer is created as a float array to output the probability generated by the model for each label.

Concerning the model execution, the method classifyFrame (see Figure 4.16) is where the inference is run and the image classification is obtained. Inside the method, after converting the Bitmap received as an input to the ByteBuffer, the interpreter's run method is called with 2 parameters: the ByteBuffer converted and the output label array to be populated with the generated results of the execution. Therefore, for each frame of the preview stream, an image classification is generated by the model and presented in real time.



```
/** Classifies a frame from the preview stream. */
String classifyFrame(Bitmap bitmap) {
  if (tflite == null) {
    Log.e(TAG, msg: "Image classifier has not been initialized; Skipped.");
    return "Uninitialized Classifier.";
  }
  convertBitmapToByteBuffer(bitmap);
  // Here's where the magic happens!!!
  long startTime = SystemClock.uptimeMillis();
  tflite.run(imgData, labelProbArray);
  long endTime = SystemClock.uptimeMillis();
  Log.d(TAG, msg: "Timecost to run model inference: " + Long.toString( i: endTime - startTime));

  // smooth the results
  applyFilter();

  // print the results
  String textToShow = printTopKLabels();
  textToShow = "Identificado em " + Long.toString( i: endTime - startTime) + " ms" + textToShow;
  return textToShow;

}
```

Figure 4.16 - Frame Classification Method

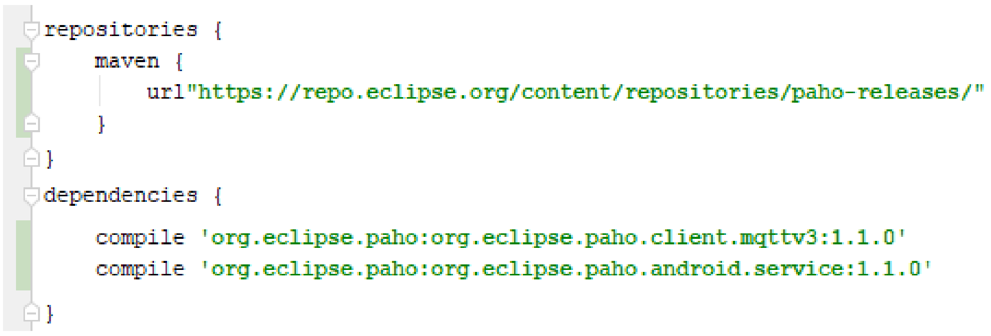
4.2.2. MQTT Communication

After successfully integrating the inference model in real-time with data collected from the camera main sensor, the device needed to send the commands to the platform in order to trigger the user desired actions. As already described in System Architecture, this communication process was implemented using MQTT.

The solution was to implement Paho Android Service [46], which provides an interface to the Original Paho Java MQTT Client. This allows to encapsulate the connection inside a service and to run it in background along the Android Activities providing reliability in MQTT connections and message receiving and sending.

4.2.2.1 Configuration

Since, in the Android system, dependencies and build are managed through the app Module build Gradle File, the first step was to add the respective Paho service inside it, as presented in Figure 4.17.

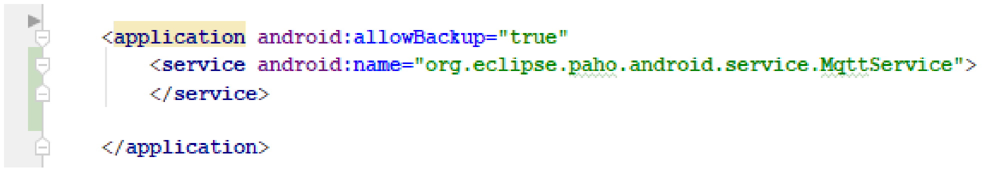


```
repositories {
    maven {
        url"https://repo.eclipse.org/content/repositories/paho-releases/"
    }
}
dependencies {

    compile 'org.eclipse.paho:org.eclipse.paho.client.mqttv3:1.1.0'
    compile 'org.eclipse.paho:org.eclipse.paho.android.service:1.1.0'
}
```

Figure 4.17 - App Module Grade Paho Dependencies

In the repositories section, the repository containing the Paho releases is added to the configuration so that the required JAR files can be downloaded. Therefore, in the dependencies section, the latest Paho release as a dependency to the present application runtime.

Concerning the connection itself, for Paho to be able to create the binding needed to the MQTT connection encapsulation, the service also needs to be declared as a service tag inside the Android Manifest file (see Figure 4.18). Paho will also need the already declared permissions to access phone state, network state and the Internet.

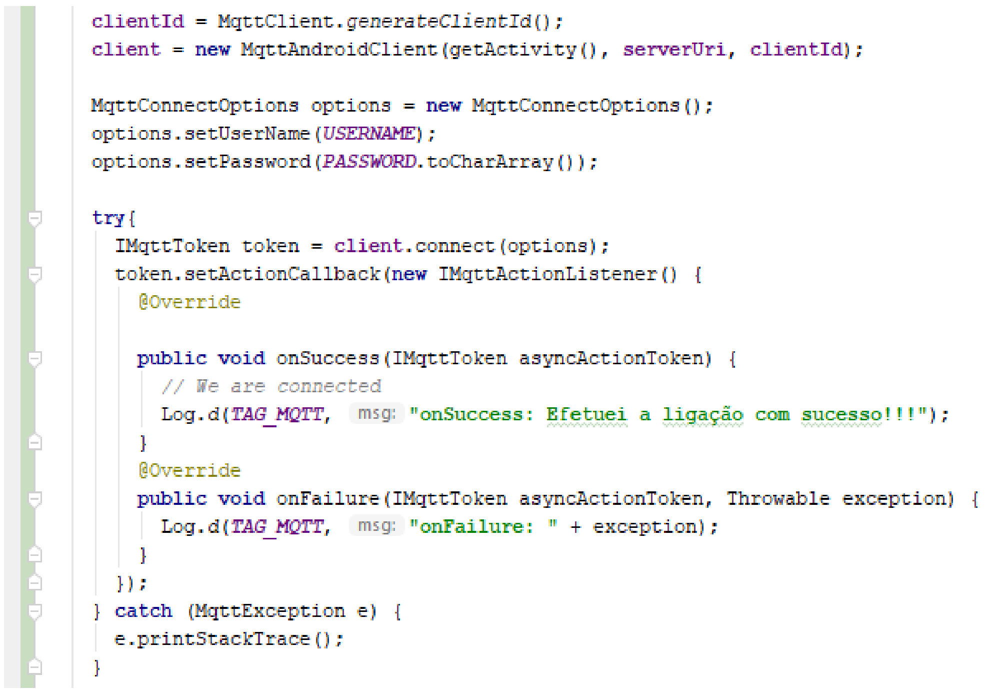


```
<application android:allowBackup="true"
    <service android:name="org.eclipse.paho.android.service.MqttService">
    </service>

</application>
```

Figure 4.18 - Paho Service Decalration inside Android Manifest

4.2.2.2 Message Sending

Before being able to publish messages, a connection to the broker needs to be established. To enable this connection, the service is going to bind through an interface called MttAndroidClient. In Figure 4.19 below is the code containing the described action.

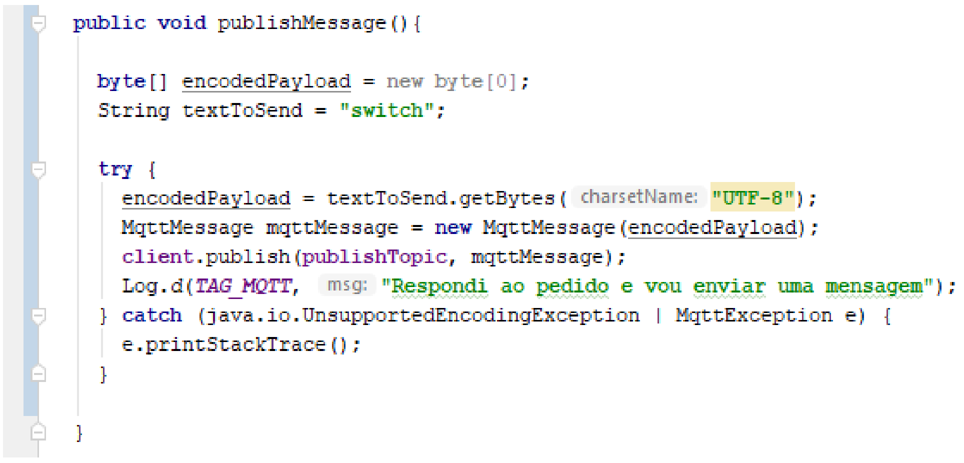


```
clientId = MqttClient.generateClientId();
client = new MqttAndroidClient(getActivity(), serverUri, clientId);

MqttConnectOptions options = new MqttConnectOptions();
options.setUserName(USERNAME);
options.setPassword(PASSWORD.toCharArray());

try{
  IMqttToken token = client.connect(options);
  token.setActionCallback(new IMqttActionListener() {
    @Override

    public void onSuccess(IMqttToken asyncActionToken) {
      // We are connected
      Log.d(TAG_MQTT, msg: "onSuccess: Efetuei a ligação com sucesso!!!");
    }
    @Override
    public void onFailure(IMqttToken asyncActionToken, Throwable exception) {
      Log.d(TAG_MQTT, msg: "onFailure: " + exception);
    }
  });
} catch (MqttException e) {
  e.printStackTrace();
}
```

Figure 4.19 - Paho Connection Establishment

Initially, a Client Id is randomly generated by the MqttClient object and the referred MqttAndroidClient responsible for the connection is instantiated with the server and client attributes. The username and respective password are also defined as an MqttConnectOptions attribute. After that, the MqttAndroidClient will try to connect with the MQTT broker, returning a token which is used to define Listener Callbacks and successfully or unsuccessfully establish the connection.

After establishing the connection, the client allows the device to send messages via the publish method. The method is part of the MqttAndroidClient and is called (see Figure 4.20) passing the encoded message payload and the topic desired to send the message.



```
public void publishMessage(){

  byte[] encodedPayload = new byte[0];
  String textToSend = "switch";

  try {
    encodedPayload = textToSend.getBytes( charsetName: "UTF-8");
    MqttMessage mqttMessage = new MqttMessage(encodedPayload);
    client.publish(publishTopic, mqttMessage);
    Log.d(TAG_MQTT, msg: "Respondi ao pedido e vou enviar uma mensagem");
  } catch (java.io.UnsupportedEncodingException | MqttException e) {
    e.printStackTrace();
  }

}
```

Figure 4.20 - Paho Message Publish

The implementation of the topics subscription and message reception will be explained in the following section.

4.3. Home Automation Platform

Having the inference model trained and integrated inside the android application, the third and last step to complete the system implementation was the installation of the automation platform on the Raspberry Pi, including the MQTT broker to allow the platform to receive the messages originated in the mobile device and consequently trigger automation rules and perform actions.

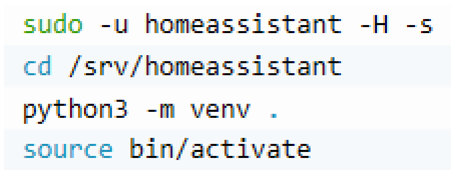
As referred in Chapter 3, the components were installed in a Raspberry Pi 3B+ and all the 3 implementation steps are described in the subsections bellow.

4.3.1. Platform Installation

A common solution to install Home Assistant in a Raspberry Pi is to install Hass.io [37] inside a Docker container but that would restrain the installation of other components inside the raspberry and complicate the communication with the broker which also needs to be installed. The solution used was to install Home Assistant inside a python virtual environment, providing the flexibility needed to the system implementation.

The major concern to have in mind using the virtual environment is the fact that any update or changes under the home assistant installation have to be made inside the environment or it can cause the duplication of the install, one inside the virtual environment and other in the host, and possibly generate conflicts between both of them.

Before the installation process, all the dependencies needed were installed using pip package manager and a system account and a home directory for Home Assistant were created.

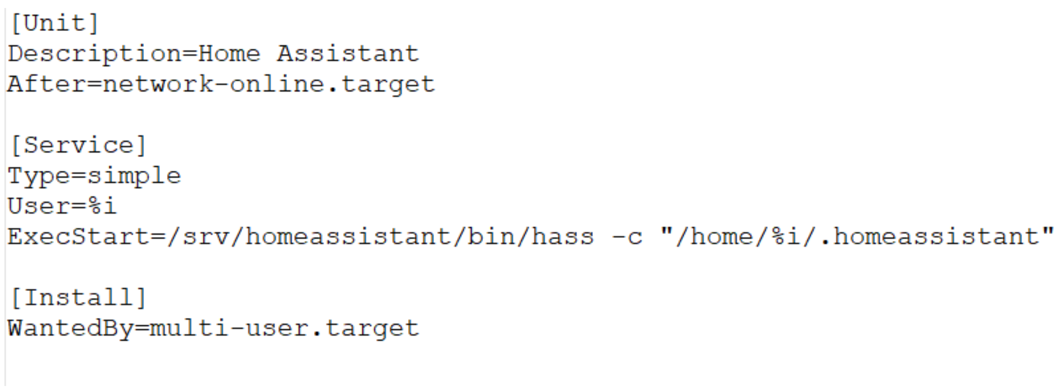


```
sudo -u homeassistant -H -s
cd /srv/homeassistant
python3 -m venv .
source bin/activate
```

Figure 4.21 - Home Assistant Virtual Environment Activation

Figure 4.21 shows the Virtual environment creation and activation. This is the environment where the platform was installed and where it runs over. After running the installation, the directories, configurations and other libraries installation were completed and the service was ready to be started and accessed via the Web interface on port 8123 of the device.

Each time the raspberry pi is powered on, the Home Assistant instance needs to be initiated. This way, hass service has been set has a daemon and defined to autostart on raspberry boot using system which is a tool for daemons managing on Debian based systems. The necessary system file, presented in Figure 4.22, was created under the system directory containing the indication to execute the service after the machine is successfully started and connected.



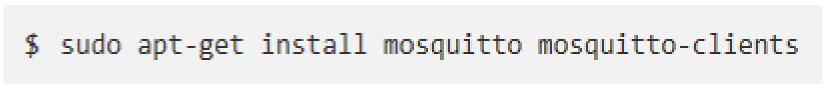
```
[Unit]
Description=Home Assistant
After=network-online.target

[Service]
Type=simple
User=%i
ExecStart=/srv/homeassistant/bin/hass -c "/home/%i/.homeassistant"

[Install]
WantedBy=multi-user.target
```

Figure 4.22 - Home Assistant Service File

4.3.2. MQTT Communication

As already referred, despite HomeAssistant has the possibility to define an in-platform broker and the existence of public brokers, a private MQTT broker was used to implement the MQTT communication on the server side.

The choice relied on Mosquitto [36] as the solution to implement the machine-to-machine messaging protocol. The service was easily installed (see Figure 4.23) in the raspberry and afterwards protected with username and password inside a system file.

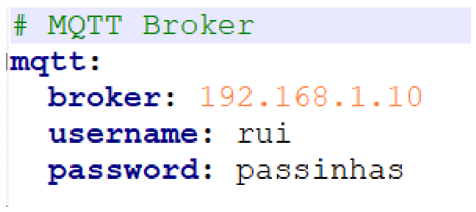


```
$ sudo apt-get install mosquitto mosquitto-clients
```

Figure 4.23 - Mosquitto MQTT Broker Installation

After the installation, the system can be started and is able to receive publish and subscribe requests. Again, the broker runs as a service and needs to be started each time the system boots so, to enable automatic start two things needed to be defined. First, the mosquito service was indicated to run at system boot in the service file under /etc/init.d directory, then, the HomeAssistant Service file was modified to instead of starting after the network is connected, starting after the mosquito broker starts. Therefore, the broker could be integrated inside the automation platform and making sure it is always running and listening for messages.

In order to integrate the MQTT broker in the automation platform, an entry containing the broker address, username and password was added to the configuration.yaml file, as shown in figure 4.24.



```
# MQTT Broker
mqtt:
  broker: 192.168.1.10
  username: rui
  password: passinhas
```

Figure 4.24 - MQTT Broker Declaration in configurations.yaml

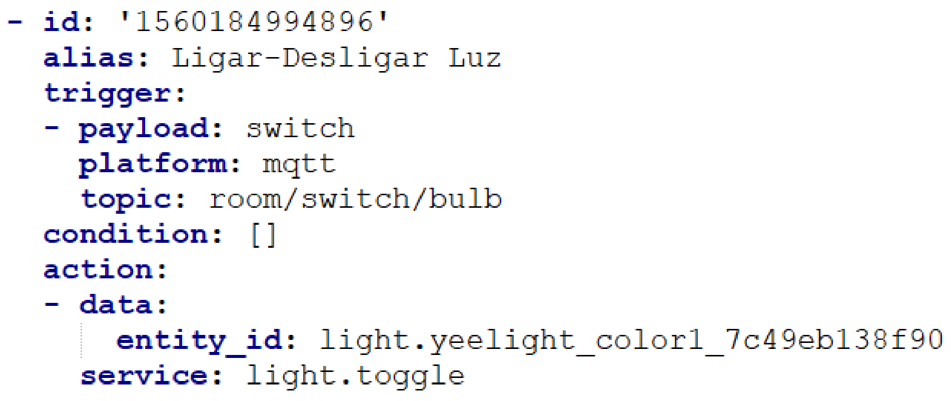
4.3.3. Automation Rules

One big advantage of Home Assistant is the ability to define automation rules. An automation rule contains 3 main blocks:

- Trigger – Describes the event which should fire the automation rule. There are many types and it is possible to have multiple triggers in the same rule;
- Condition – Is an optional field where can be defined specific conditions where the automation rule should work. When a condition does not return true, the action will not perform can also exist multiple conditions for the same rule;
- Action – Is the action that it will be performed after a trigger has been started and the conditions validated.

One of the trigger types is the MQTT trigger and is fired when a specific message is received in a specific topic. This way, automation rules could be created to make the desired system actions according to the MQTT messages received. For example, Figure

4.25 bellow shows an automation rule where an action is triggered when a message with the payload “switch” is received on topic “room/switch/bulb” with no conditions implicit.



```
- id: '1560184994896'
  alias: Ligar-Desligar Luz
  trigger:
  - payload: switch
    platform: mqtt
    topic: room/switch/bulb
  condition: []
  action:
  - data:
      entity_id: light.yeelight_color1_7c49eb138f90
    service: light.toggle
```

Figure 4.25 - Bulb Automation Rule

The action will take place on the bulb described in the entity tag with the service light.toggle, which is responsible for switching the bulb. Additionally, the system Web interface has a tool to create the automations, displaying the list of triggers, entities available, actions and the respective services.

The automation rules created to fulfill the desired system implementation were placed inside the automations .yaml file and loaded to the system at startup.

Chapter 5 – Validation Tests

Having the implementation completed, a prototype of the system proposed was obtained. This Chapter, describes the solution achieved and its components. A comparison between the manual inference and the on-device machine learning final solution will also be made, considering the evaluation times obtained and accuracy losses and finally tests over real application scenarios, which will be followed by a discussion over the overall scenario provisioned by the implementation made and its results.

5.1. Implemented Prototype

The developments made across this dissertation aim at a hypothetical scenario where a user can control the devices existent in his house through the camera user interface having visual representation of the actions and providing intuitive interactions. Looking at the implementation obtained, the major visual results come from the mobile application and the device connected to the platform.

The implementation described in Section 4.2 resulted in a mobile application which after installed in and android device is where the user will interact. Figure 5.1 bellow represents the application main screen, containing the camera viewfinder, a label containing the object identified and the accuracy of the evaluation alongside the buttons to take actions on that device when evaluated with accuracy over a pre-defined threshold.

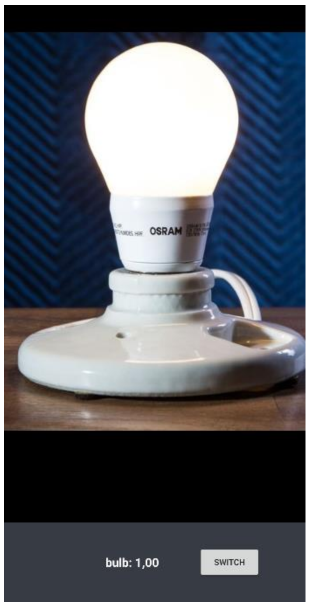


Figure 5.1 - Application Main Screen Example

In the screenshot presented in Figure 5.1, it is visible that the user was pointing the device at a light bulb and since the evaluation result is constantly returning values close to 100% the button to switch on the device is presented, which represents the action passible to be taken at that moment. When the button is pressed, the whole process described in Chapter 3 takes place and the bulb will then light up. Additionally to the evident visual feedback on the bulb, looking at HomeAssistant Web platform it is possible to see the event occurrence, as illustrated ahead in Figure 5.2.

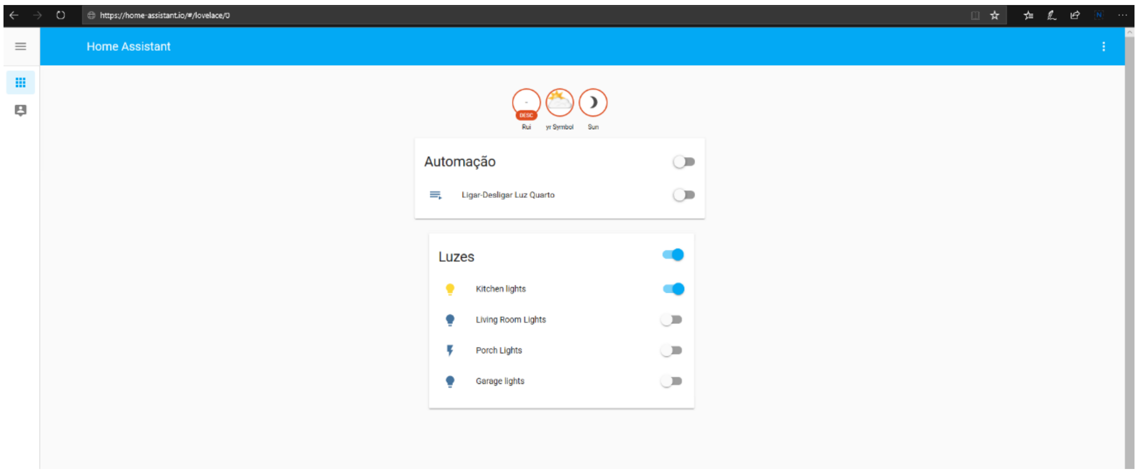
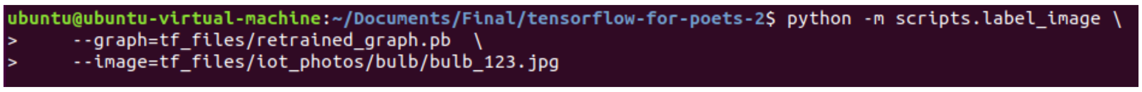


Figure 5.2 - Automation Platform with Light Turned On

5.2. Server vs On-Device Machine Learning Performance Tests

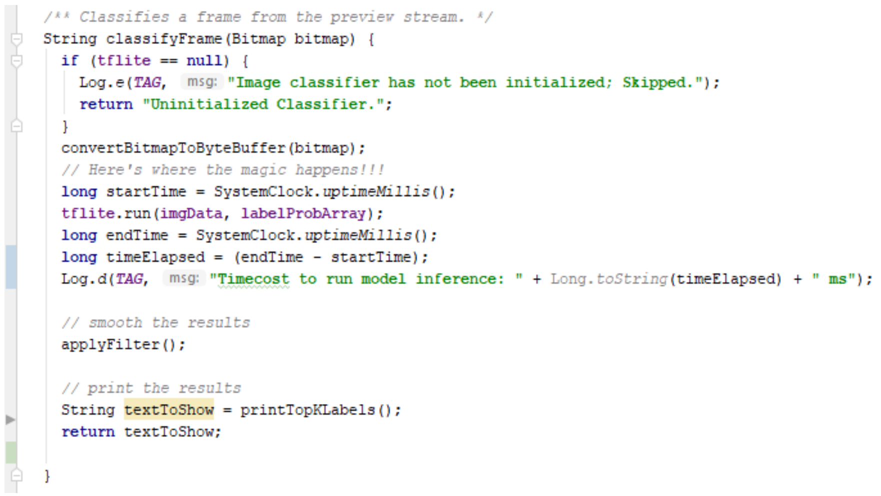
Even though traditionally, machine learning and neural networks are concepts associated with increased computation power and robust hardware, the scope of this dissertation addresses the on-device artificial intelligence arising ubiquitousness and its major potential. Therefore, an interesting result to analyze is the accuracy obtained with lighter models and the existence of latency or performance decreasing when running on less powerful devices.

This way, a metric that can give indicators of both benefits and drawbacks of this approach is the time spent during the execution of the model trained to produce an output and effectively label the input image. A possible way of analyzing this is by running the inference model at the machine where it was trained with a python script, label_image.py shown in Figure 5.3, and after that running the same model already integrated in the mobile application, measuring the times before and after the run (see Figure 5.4).



```
ubuntu@ubuntu-virtual-machine:~/Documents/Final/tensorflow-for-poets-2$ python -m scripts.label_image \
>     --graph=tf_files/retrained_graph.pb  \
>     --image=tf_files/iot_photos/bulb/bulb_123.jpg
```

Figure 5.3 - Python Script to Evaluate an Image using the Inference Model



```
/** Classifies a frame from the preview stream. */
String classifyFrame(Bitmap bitmap) {
  if (tflite == null) {
    Log.e(TAG,  msg: "Image classifier has not been initialized; Skipped.");
    return "Uninitialized Classifier.";
  }
  convertBitmapToByteBuffer(bitmap);
  // Here's where the magic happens!!!
  long startTime = SystemClock.uptimeMillis();
  tflite.run(imgData, labelProbArray);
  long endTime = SystemClock.uptimeMillis();
  long timeElapsed = (endTime - startTime);
  Log.d(TAG,  msg: "Timecost to run model inference: " + Long.toString(timeElapsed) + " ms");

  // smooth the results
  applyFilter();

  // print the results
  String textToShow = printTopKLabels();
  return textToShow;

}
```

Figure 5.4 - Image Classification inside the Application Measuring Time Elapsed

This analysis was performed using three different images (see Figure 5.5) and ran three times in each of them to guarantee a minimum coherence in the results. In the case of the mobile device, since the input comes as a video stream, the three images were represented by three different scenarios with the three different bulbs. The results are presented bellow in Table 5.1

Table 5.1 - Comparison between Manual Inference versus On-Device Run

	Server Run			On-Device		
Times (ms)	Run 1	Run 2	Run 3	Run 1	Run 2	Run 3
Image 1	12.8	12.7	12.9	16.0	15.0	16.0
Image 2	13.5	13.7	13.6	20.0	19.0	17.0
Image 3	12.4	12.4	12.5	30.0	30.0	30.0

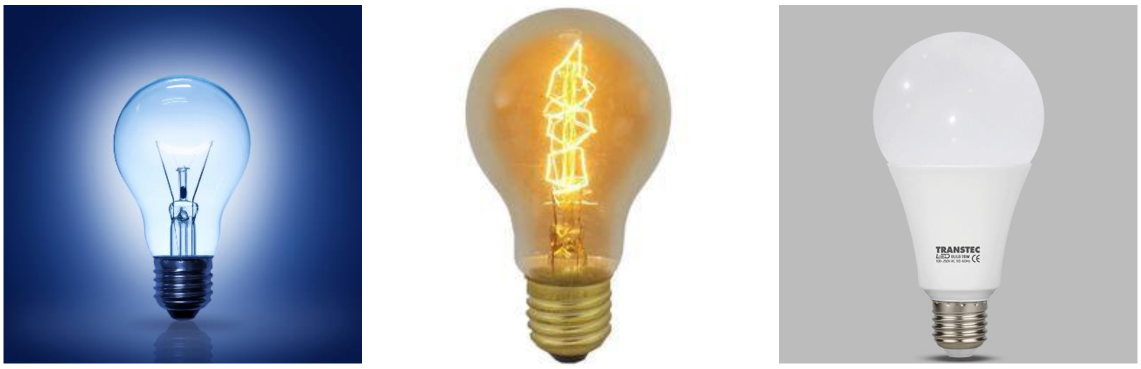


Figure 5.5 - Bulb Images Used for Testing

Figure 5.5 above shows the 3 example images used to do the testing. The test results obtained are presented in milliseconds and are further ahead discussed in section 5.4.

5.3. Confidence Tests

As referred in Section 4.1, the system implemented is based on a neural network retrained to identify smart household objects, particularly bulbs, air conditioners and electrical window shutters. The logical scenarios to validate the final solution was to put the system to the test against these items, therefore, each of the next three sections presents three scenarios to one of the three classes trained.

5.3.1. Bulb

The first test represents a use case scenario where the user points de device at a light bulb. In this case, when the evaluation output is higher than the 95% confidence threshold, the switch option appears on the screen and the user can interact with the device. To obtain maximum results and since the image dataset used for testing was simple light bulbs, the 3 tests performed was on simple lightbulbs connected to power (see Figure 5.6).

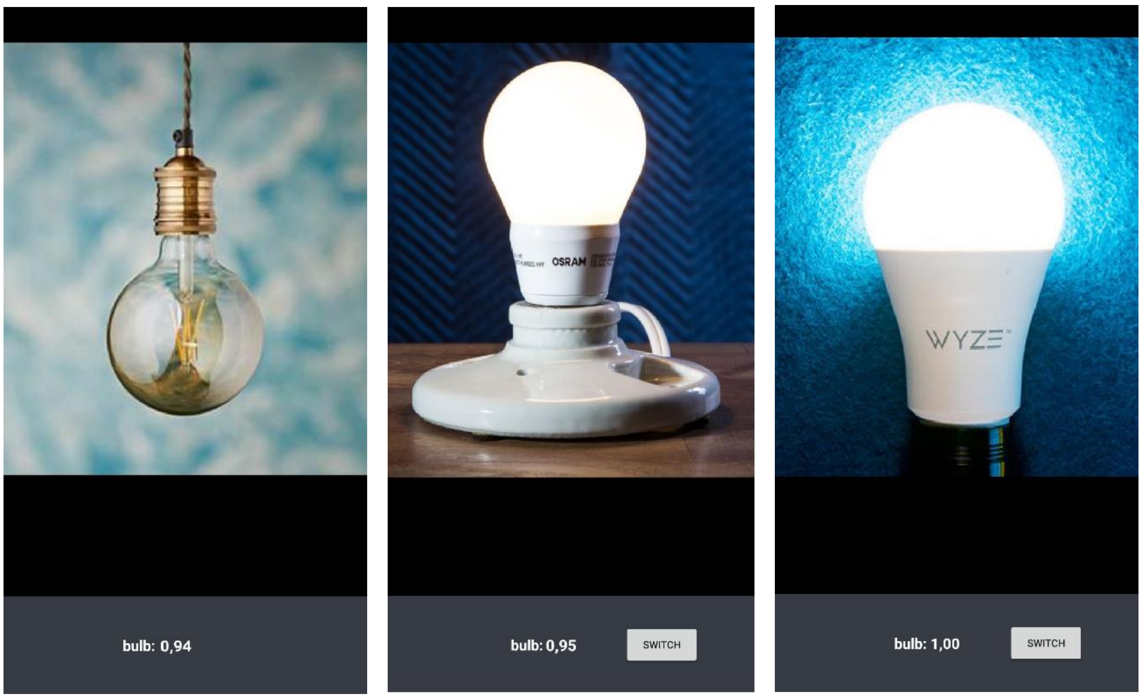


Figure 5.6 - Bulb Tested Scenarios

Table 5.2 presents a summary of the results obtained after executing the three test scenarios. Looking at the confidence obtained, it is possible to see that in the first test, the inference method evaluated the object with a conviction of 94%, in the second one, 95% certainly and the last one with maximum confidence. Again, only the identifications superior to the threshold provide the user the ability to interact with the device.

Table 5.2 - Bulb Tested Scenarios confidence

	Bulb 1	Bulb 2	Bulb 3
Confidence	94%	95%	100%

5.3.2. Air Conditioner

The second test represents a use case scenario where the user points the device at an air conditioner. In this case, when the evaluation output is higher than the 95% confidence threshold, the hotter/colder options, represented by the options up and down appear on the screen and the user can interact with the device. To obtain maximum results and since the image dataset used for testing was traditional house air conditioners, the 3 tests performed were on those instead of industrial or bigger ones. (see Figure 5.7).

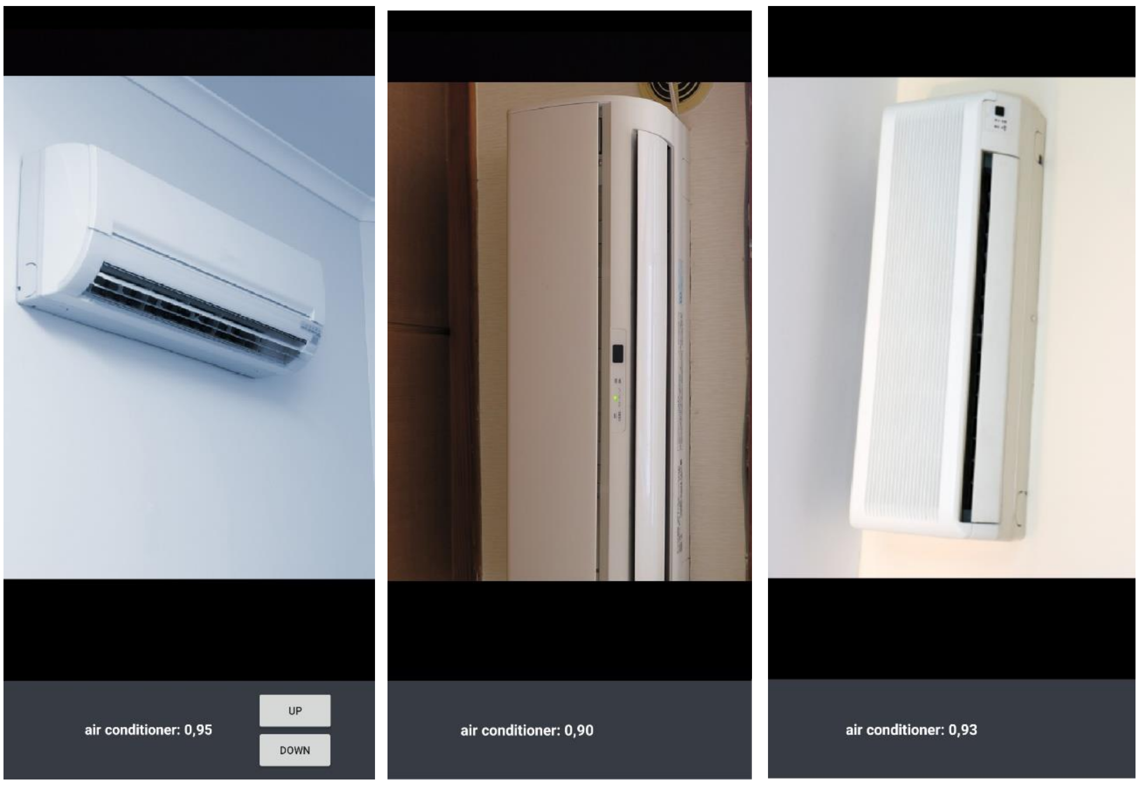


Figure 5.7 - Air Conditioner Tested Scenarios

Table 5.3 presents a summary of the results obtained after executing the three test scenarios. Looking at the confidence obtained, it is possible to see that in the first test, the inference method evaluated the object with a conviction of 95%, in the second one, 90% certainly and the last one with 93% confidence. Again, only the identifications superior to the threshold provide the user the ability to interact with the device.

Table 5.3 - Air Conditioner Tested Scenarios Confidence

	Air Conditioner 1	Air Conditioner 2	Air Conditioner 3
Confidence	95%	90%	93%

5.3.3. Window Shutter

The last test represents a use case scenario where the user points the device at the window shutters. In this case, when the evaluation output is higher than the 95% confidence threshold, the up and down options appear on the screen and the user can interact with the device, respectively opening or closing the shutters. The 3 scenarios tested are presented bellow in Figure 5.8.

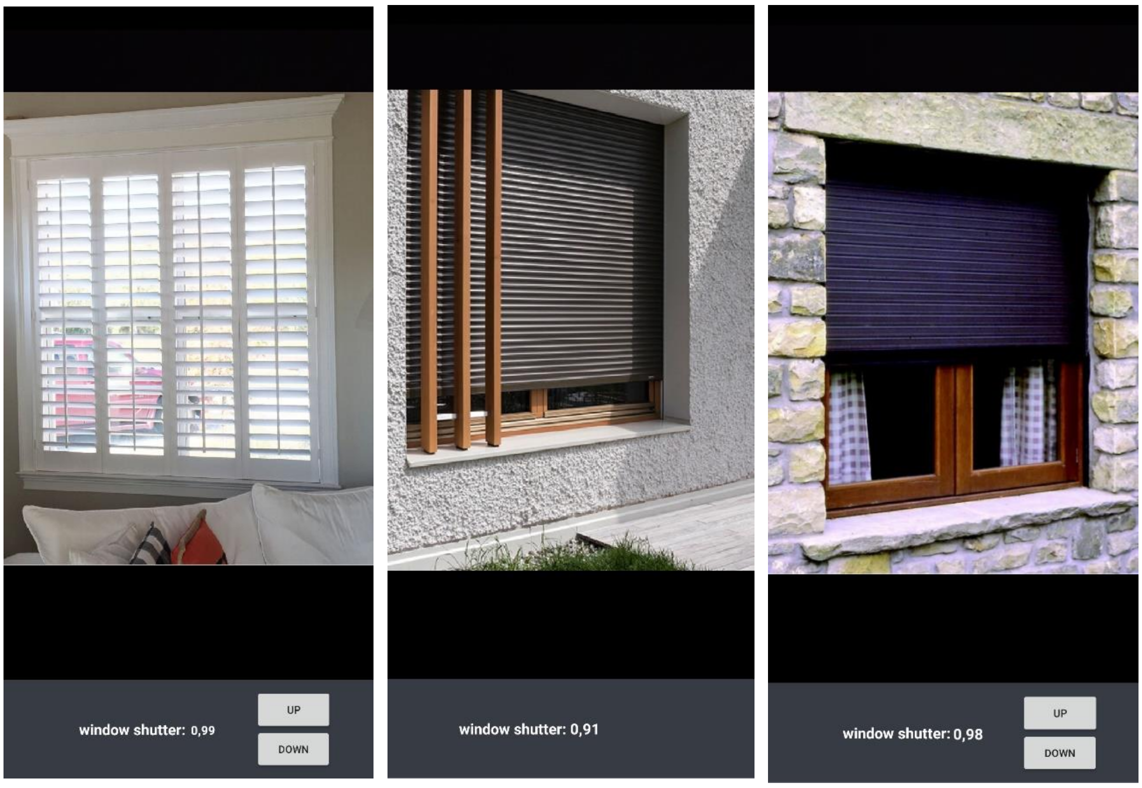


Figure 5.8 - Window Shutters Tested Scenarios

Table 5.4 presents a summary of the results obtained after executing the three test scenarios. Looking at the confidence obtained, it is possible to see that in the first test, the inference method evaluated the object with a conviction of 99%, in the second one, 91% certainly and the last one with 98% confidence. Again, only the identifications superior to the threshold provide the user the ability to interact with the device.

Table 5.4 - Window Shutter Tested Scenarios Confidence

	Window Shutter 1	Window Shutter 2	Window Shutter 3
Confidence	99%	91%	98%

5.4. Discussion

This chapter was focused on showing the final prototype obtained, demonstrating the potentiality of on-device machine learning and generally showing the results obtained when effectively testing the developed solution confidence and consistency.

Analyzing the results obtained, concerning the comparison made between manually running the inference model on the training environment and the inference happening on the mobile device, we can see that the results were not that distant and therefore, the system can produce fast results in real time without compromising the performance obtained. In terms of confidence obtained, every test provided result higher than 90%, which is positive and allowed to control the devices in almost every situation with confidence.

Chapter 6 – Conclusions and Future Work

This chapter presents the main conclusions, considering the work developed, the obstacles and limitations encountered as well as the future work that can be developed to add value to the already proposed solution.

6.1. Main Conclusions

As an overall solution, the developed system worked as a proof of concept of the integration of mobile devices in home automation with use of machine learning for object recognition. Throughout the dissertation, several obstacles were found due to the nature of the integration proposed, however, the initial objectives were achieved and a stable work foundation and architecture for future developments were set.

An initial envisaged solution was to use traditional computer vision manual methods to fulfill the proposed objectives, but the use of machine learning techniques to replace these more traditional algorithms proved to be a great evolutionary step forward and to give the robustness needed to a more futureproof system. Even though machine learning and deep learning-based systems often need massive hardware to be trained and developed, the work developed in this dissertation proves that lightweight alternatives can be found and adapted to meet the desired goals.

The choice of the MQTT for communication allowed to maintain a low resource lightweight communication between the modules given the small amount of data sent in each transmission. This way, the system relies on simple machine to machine interaction with acceptable latency in order to maintain the possibility to use in a real-time integration.

Being Home Assistant the automation hub for all the integrations allowed multiple advantages. To begin, the platform was installed in a Raspberry Pi 3B+, an embedded platform in which the software was optimized to run and consequently resulted in an easy installation and maintenance. Additionally, the ability to define automation rules revealed to be crucial in the automation process, providing the ability to implement complex scenarios depending on the command received via the MQTT messages.

Looking at the results obtained, an important conclusion to bear in mind is that the on-device classification based on machine learning proved to be surprisingly accurate and able of handling the tasks defined, confirming this way the already arising paradigm of AI increasingly moving to the edge devices without compromising functionality. Despite the fact that the last layer of the neural network was only retrained with 3 classes but also giving the fact that the model was trained in a normal computer without powerful hardware and that the inference is running on a mobile device, this could be proven analyzing the confidence obtained which was positive and higher than expected.

This way, the final solution developed in this dissertation proves that an inference machine learning model can be used in real time evaluation, integrated inside an android application providing instantaneous visual feedback. This leads to also real-time action triggering in the automation platform making use of the integration and consequently built a seamless and uninterrupted information workflow.

6.2. System Limitations

Even though all the work developed in this dissertation culminated in a functional prototype, certain assumptions were made to make the integration easier to prove.

For instance, the existence of two identical elements inside home would make the inference model return the same result for both, not being able to distinguish between them. In order to distinguish different instances of the same object, a solution like indoor location would have to be implemented, giving the device the ability to know where each of the objects were placed inside home and consequently know which object was being analyzed.

In order to make the integration possible, it was also assumed that the devices were already connected to the actuation platform. This required pre-configuration is a system limitation because the application developed does not offer a solution to configure each user specific smart devices.

6.3. Future Work

Despite the integration proposed in the solution was implemented and considering the existing limitations, the room for improvement is notorious. The system acts as a proof of concept and as a basis to major breakthroughs in future developments.

First of all, there are a lot of sensors in the mobile device which were not used in the course of this dissertation that can be further explored. The mobile device has the ability to measure ambient light, temperature, proximity and even magnetic fields. This enables the possibility to control a range of smart devices, from air humidifiers to door locks, in new and innovative ways.

Concerning the computer vision topic, the evolution from image classification to object detection would also provide the ability to track multiple objects at the same time and allow better precision and control of the devices. Even though implying major developments in the application and a whole new recourse consuming re-training process, this improvement would unlock more possibilities of user interaction and functionalities to the application.

Additionally, a contribution than can be remarking in the potentiality of the concept introduced is the re-training of the inference model based on user feedback. Even though a solution can be prepared for a general use case scenario, each case is unique, and each user will have different needs. This way, if the mobile application provided a way of collecting user feedback according to his respective scenario and reality, this input could be used to re-train the model with improved accuracy and consequently to obtain flawless and consistent results.

References

[1] D. Hanes, G. Salgueiro, P. Grossetete, R. Barton and J. Henry, IoT Fundamentals: Networking Technologies, Protocols, and Use Cases for the Internet of Things. Cisco Press, Indianapolis, USA, 2017.

[2] S. Chen, H. Xu, D. Liu, B. Hu and H. Wang, "A Vision of IoT: Applications, Challenges, and Opportunities With China Perspective," in IEEE Internet of Things Journal, vol. 1, no. 4, pp. 349-359, Aug. 2014.

[3] L. Atzori, A. Iera, G. Morabito, "The Internet of Things: A survey", Computer network 54.15.2010, pp. 2787-2805.

[4] J. Y. Kim, H. Lee, J. Son and J. Park, "Smart home web of objects-based IoT management model and methods for home data mining," 2015 17th Asia-Pacific Network Operations and Management Symposium (APNOMS), Busan, 2015, pp. 327-331.

[5] S. Greengard, The Internet of Things. The MIT Press, London, England, 2015.

[6] OpenHAB Community and OpenHAB Foundation e.V., “openHAB – empowering the smart home,” 2018, [Online] Available: https://www.openhab.org/, (visited 19/11/2018).

[7] OpenRemote Inc, “OpenRemote | Open Source for Internet of Things,” 2016, [Online] Available: http://www.openremote.com/, (visited 20/11/2018).

[8] N. Gyory and M. Chuah, "IoTOne: Integrated platform for heterogeneous IoT devices," 2017 International Conference on Computing, Networking and Communications (ICNC), Santa Clara, CA, 2017, pp. 783-787.

[9] OpenCV team, “OpenCV library,” 2018, [Online] Available: https://opencv.org/, (visited 21/11/2018).

[10] Cloud Vision, “Vision API - Image Content Analysis,” [Online] Available: https://cloud.google.com/vision/, (visited 22/11/2018).

[11] The Statistics Portal, “Market Directory – Smart Home – Worldwide,” [Online] Available: https://www.statista.com/outlook/279/100/smart-home/worldwide, (visited 10/01/2018)

[12] Domoticz, “Domoticz - Control at your finger tips,” 2017, [Online] Available: https://domoticz.com/, (visited 27/12/2018).

[13] Home Assistant, “Awaken your home,” [Online] Available: https://www.home-assistant.io/, (visited 27/12/2018).

[14] OpenMotics, “Smart Building Automation as Basic as Tap Water!,” [Online] Available: https://www.openmotics.com/, (visited: 28/12/2018)

[15] Calaos Team, “Calaos | Open Source Home Automation.,” [Online] Available: https://calaos.fr/en/, (visited: 28/12/2018)

[16] StatCounter, “Mobile Operating System Market Share Worldwide,” [Online] Available: http://gs.statcounter.com/os-market-share/mobile/worldwide (visited: 11/01/2018)

[17] R. Nunkesser, "Beyond Web/Native/Hybrid: A New Taxonomy for Mobile App Development," 2018 IEEE/ACM 5th International Conference on Mobile Software Engineering and Systems (MOBILESoft), Gothenburg, 2018, pp. 214-218.

[18] C. M. S. Ferreira et al., "An Evaluation of Cross-Platform Frameworks for Multimedia Mobile Applications Development," in IEEE Latin America Transactions, vol. 16, no. 4, pp. 1206-1212, April 2018.

[19] The IETF Trust, “Hypertext Transfer Protocol -- HTTP/1.1”, 2007, [Online] Available: https://www.w3.org/Protocols/HTTP/1.1/rfc2616bis/draft-lafon-rfc2616bis-03.html, (visited 10/07/2019)

[20] TutorialsPoints, “HTTP - Overview”, 2019, [Online] Available: https://www.tutorialspoint.com/http/http_overview.htm, (visited 11/07/2019)

[21] B. Wukkadada, K. Wankhede, R. Nambiar and A. Nair, "Comparison with HTTP and MQTT In Internet of Things (IoT)," 2018 International Conference on Inventive Research in Computing Applications (ICIRCA), Coimbatore, India, 2018, pp. 249-253.

[22] N. Naik, "Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP," 2017 IEEE International Systems Engineering Symposium (ISSE), Vienna, 2017, pp. 1-7.

[23] T. Yokotani and Y. Sasaki, "Comparison with HTTP and MQTT on required network resources for IoT," 2016 International Conference on Control, Electronics, Renewable Energy and Communications (ICCEREC), Bandung, 2016, pp. 1-6.

[24] D. Ballard and C. Brown, Computer Vision, Prentice Hall, First edition, 1982.

[25] James Le, ” The 5 Computer Vision Techniques That Will Change How You See The World,” 2018, [Online] Available: https://heartbeat.fritz.ai/the-5-computer-vision-techniques-that-will-change-how-you-see-the-world-1ee19334354b, (visited 10/07/2019)

[26] Google Developer Team, “Object detection,” 2019, [Online] Available: https://www.tensorflow.org/lite/models/object_detection/overview, (visited 20/08/2019)

[27] A.Canziani, A. Paszke and E. Culurciello, “An Analysis of Deep Neural Network Models for Practical Applications,” CoRR, 2018.

[28] A. Lee, “Comparing Deep Neural Networks and Traditional Vision Algorithms in Mobile Robotics,” 2016.

[29] TensorFlow, “Tensor Flow - An open source machine learning framework for everyone,” [Online] Available: https://www.tensorflow.org/, (visited 27/12/2018).

[30] V. Govindraj, M. Sathiyanarayanan and B. Abubakar, "Customary homes to smart homes using Internet of Things (IoT) and mobile application," 2017 International Conference On Smart Technologies For Smart Nation (SmartTechCon), Bangalore, 2017, pp. 1059-1063.

[31] T. Adiono, S. Harimurti, B. A. Manangkalangi and W. Adijarto, "Design of smart home mobile application with high security and automatic features," 2018 3rd International Conference on Intelligent Green Building and Smart Grid (IGBSG), Yi-Lan, 2018, pp. 1-4.

[32] S. Guennouni, A. Ahaitouf and A. Mansouri, "Multiple object detection using OpenCV on an embedded platform," 2014 Third IEEE International Colloquium in Information Science and Technology (CIST), Tetouan, 2014, pp. 374-377.

[33] Dat Tran, "How to train your own Object Detector with TensorFlow's Object Detector API," 2017, [Online] Available: https://towardsdatascience.com/how-to-train-your-own-object-detector-with-tensorflows-object-detector-api-bec72ecfe1d9, (visited 22/12/2018).

[34] Android Developer Team, "NeuralNetworks," 2019, [Online], Available: https://developer.android.com/ndk/reference/group/neural-networks, (visited 20/08/19)

[35] Raspberry Pi Foundation, "Raspberry Pi 3 Model B+," 2018, [Online] Available: https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/, (visited 17/23/2019)

[36] Eclipse Foundation, Inc, "Eclipse Mosquitto, An open source MQTT broker," 2019, [Online] Available: https://mosquitto.org/, (visited 24/04/2019)

[37] Fredrik Lindqvist, "Hass.io," 2019, [Online] Available: https://www.home-assistant.io/hassio/, (visited 10/07/2019)

[38] Labelbox, Inc, "Labelbox: The leading training data solution," 2019, [Online] Available: https://labelbox.com/, (visited 17/08/2019)

[39] Google Cloud Development Team, "Cloud TPU," 2019, [Online] Available: https://cloud.google.com/tpu/, (visited 14/07/2019)

[40] Google Developers Team, "Install Python Packages," 2019, [Online] Available: https://codelabs.developers.google.com/, (visited 29/07/2019)

[41] Howard, Andrew G, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications", 2017

[42] Andrew G. Howard, "MobileNets: Open-Source Models for Efficient On-Device Vision," 2019, [Online] Available: https://ai.googleblog.com/2017/06/mobilenets-open-source-models-for.html, (visited 10/07/2019)

[43] Fredrik Lundh, "Pillow," 2019, [Online] Available: https://pillow.readthedocs.io/en/stable/, (visited 12/05/2019)

[44] FPL, "FlatBuffers," 2019, [Online], Available: https://google.github.io/flatbuffers/, (visited 10/07/2019)

[45] Tensor Flow Team, "TensorFlow Lite image classification Android example application," 2019, [Online] Available: https://github.com/tensorflow/examples/tree/master/lite/examples/image_classification/android, (visited 10/07/2019)

[46] Eclipse Foundation, "Eclipse Paho Android Service," 2019, [Online] Available: https://github.com/eclipse/paho.mqtt.android, (visited 10/04/2019)

[47] OASIS, "MQTT,", 2018, [Online] Available: http://mqtt.org/faq, (visited 20/06/2019)

[48] M. Kumar and B. K. Gupta, "Security for Bluetooth enabled devices using BlipTrack Bluetooth detector," 2015 International Conference on Advances in Computer Engineering and Applications, Ghaziabad, 2015, pp. 155-158.

[49] StatCounter, “Mobile & Tablet Android Version Market Share Worldwide,” 2019, [Online] Available: https://gs.statcounter.com/android-version-market-share/mobile-tablet/worldwide