INSTITUTO
UNIVERSITÁRIO
DE LISBOA

IoT Home Automation with Mobile Devices Vision

Carlos Daniel Simões Jorge Guerra

Master in Telecommunications and Computer Engineering

Supervisor:
PhD Rui Neto Marinheiro, Associate Professor,
Iscte-IUL

Co-Supervisor:
PhD Tomás Gomes da Silva Serpa Brandão, Assistant Professor,
Iscte-IUL

November, 2022

TECNOLOGIAS
E ARQUITETURA

Department of Information Science and Technology

IoT Home Automation with Mobile Devices Vision

Carlos Daniel Simões Jorge Guerra

Master in Telecommunications and Computer Engineering

Supervisor:
PhD Rui Neto Marinheiro, Associate Professor,
Iscte-IUL

Co-Supervisor:
PhD Tomás Gomes da Silva Serpa Brandão, Assistant Professor,
Iscte-IUL

November, 2022

Acknowledgments

Firstly, I would like to thank my supervisors, Dr. Tomás Brandão and Dr. Rui Marinheiro, for their guidance over these years, for always keeping pushing me and never stopping believing that I would be able to finish this dissertation.

I would like to thank all my friends and family for always being a pillar that I could rely on during these 6 years of academic studies. I would like to especially thank, Maria and Beatriz, for being there on those long nights of study, projects, and everything in between.

Lastly, I would like to thank Instituto de Telecomunicações for all the support that was given to me during the writing of this dissertation.

Resumo

Nos últimos anos, o número de dispositivos da Internet das Coisas cresceu rapidamente, em particular os dispositivos utilizados num contexto doméstico inteligente. O mercado da domótica tem sido prejudicado por um número interminável de protocolos, muitas vezes difíceis de integrar, principalmente quando novos dispositivos são anexados ao sistema.

Também a integração de dispositivos móveis está incompleta, nomeadamente quando consideramos a integração e utilização de informação de sensores, em particular a câmara, que pode ser recolhida a partir do dispositivo.

Para tornar esta integração mais eficaz, desenvolvemos uma solução onde é possível, através de um dispositivo móvel, adicionar e controlar novos dispositivos da Internet das Coisas. Esta solução, suportada pela plataforma OpenHAB opensource, fará uso da câmara do dispositivo móvel do utilizador como um input para identificar qual o objeto que o utilizador está a tentar controlar. Para isso, o trabalho desenvolvido faz uso de inteligência artificial, especificamente uma rede siamesa, que tem a vantagem de partilhar os mesmos pesos entre as duas torres, tornando-a uma rede neural, fácil de treinar e leve, adequada para a possibilidade de operar o modelo num dispositivo móvel.

Este sistema obteve bons resultados ao tentar identificar objetos utilizando quer imagens já presentes quer novas imagens, para esses objetos. Foi também capaz de se adaptar corretamente a uma nova classe a ser adicionada ao conjunto de dados, sem necessidade de reciclar toda a rede.

Assim, ficou provado que é possível melhorar a integração de dispositivos móveis na domótica, utilizando o sensor de câmara de forma inovadora.

Palavras-chave: Internet das coisas, aprendizagem automática, reconhecimento de objetos, casa inteligente, Domótica

Abstract

In recent years, the number of Internet of Things devices has grown rapidly, in particular devices used in a smart home context. The home automation market has been hampered by an endless number of protocols, often difficult to integrate, especially when new devices are attached to the system.

Also, the integration of mobile devices is incomplete, namely when considering the integration and use of sensor information, in its camera, which can be collected from the device.

To make this integration more effective, we developed a solution where it's possible, through a mobile device, to add and control new Internet of Things devices. This solution, supported by the OpenHAB open-source platform, will use the camera of the user's mobile device as an input to identify which object the user is trying to control. For this, the work developed uses artificial intelligence, specifically a Siamese network, which has the advantage of sharing the same weights between the two towers, making it a neural network, easy to train and lightweight, suitable for the possibility of operating the model on a mobile device.

This system obtained good results when trying to identify objects using either already present images or new images, for those objects. It was also able to correctly adapt to a new class being added to the dataset, without the need to retrain the whole network.

Thus, it was proved that it is possible to improve the integration of mobile devices in home automation, innovatively using the camera sensor.

Keywords: Internet of things, machine learning, object recognition, smart home, Home automation

Contents

List of Figures

List of Tables

List of Acronyms

IoT	Internet of Things
R-CNN	Region-Based Convolutional Neural Networks
Std Dv	Standard Deviation
MQTT	MQ Telemetry Transport
JVM	Java Virtual Machine
CNN	Convolutional Neural Network

CHAPTER 1

1. Introduction

1.1.Motivation and Framework

In the last few years, there has been great technological development focusing on the Internet of Things (IoT). IoT devices have the potential to alter and change the way we can interact with the world around us. One of the environments in which IoT devices can make a difference is in our houses. The promise of a smart home has been around for nearly 30 years, however with the advancements that have been made recently it is becoming a reality. A smart home consists of a home environment in which appliances and devices can communicate with each other and can be controlled either over each other, over a single dashboard that can control all of them, or over a smartphone over the internet.

However, for a long time, the market has been saturated with countless protocols and platforms that can integrate these new devices. This market saturation creates the opportunity for platforms that can support multiple devices and protocols to appear and offer solutions that can be very useful for an end-user that wants to implement these devices at its home. Platforms like OpenHAB or OpenRemote, offer that market gap by using open standards which make them able to integrate a great number of devices without lacking benefits for the user. However, the configuration portion of these platforms can be time-consuming and not intuitive for an average user. In addition to the configuration portion, everyday usage can also be confusing. The combination of both these factors ends up limiting the number of users that are willing to learn and understand these platforms.

One area that platforms such as these are lacking, is in the usage of the user's mobile device. With the help of the sensors present in the personal mobile device, it is possible to facilitate the configuration process making it easier and more intuitive. One of the biggest sensors that can help make this possible is the camera, which can be used in combination with Object Recognition software to possibly integrate with automation platforms.

1.2. Research Questions

- Should the implemented system use a pre-trained model, or should be built a new one for this implementation?
- Is the implemented system capable of accurately telling which object is the user trying to control and in a rapid manner that won't affect the user experience?
- Is the implemented system capable of adapting to new classes without training the neural network?

1.3. Objectives

The objective of this dissertation is to improve the integration of the usage of mobile devices in current home automation platforms. To achieve this objective the mobile device will be considered as a sensor, especially its camera, to perform image-based object recognition tasks in a user-friendly way.

1.4. Research Method

Throughout this project the research method that will be used, the Design Science Research methodology consists of five phases, which can be seen in Figure 1.

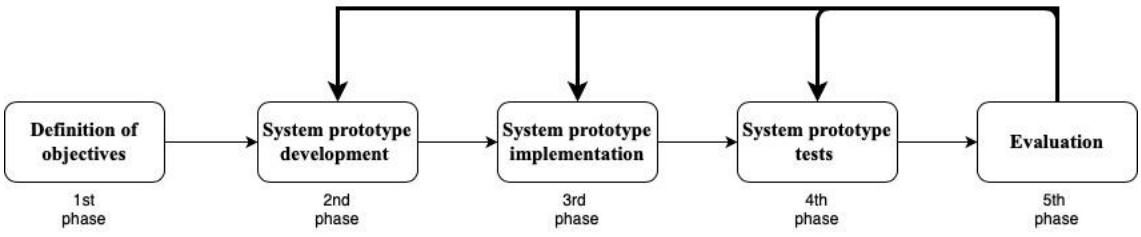

Figure 1: Research method flow

1st phase: Definition of objectives – At this stage, technologies and mechanisms are studied to help determine which platforms and methods offer the best features. This will guide the investigation and should be fulfilled.

2nd phase: System prototype development – In this phase, the different components of this project are developed to fulfill the objectives, such as:

- System architecture
- Implementation of the image recognition algorithm
- Creating a demonstrator of the system
- Development of the module for the platform

3rd phase: System prototype implementation – The components mentioned in the 2nd phase should be implemented together to implement a system prototype

4th phase: System prototype tests – The system is subject to tests to perform the necessary changes for achieving the final prototype

5th phase: Evaluation – The final prototype is evaluated to ensure it is the best solution

The 2nd, 3rd, and 4th phases are cyclical which means that, at the end of the 5th phase, if the prototype is not in accordance with the objectives defined in the first phase, the problem needs to be analysed and improvements need to be performed. Therefore, a return to the previous phases is required for overcoming the obstacles between what was initially defined and the final prototype. The project is finished when the final prototype is in accordance with the defined objectives.

1.5. Dissertation Structure

After the objectives, methodology and motivation for this dissertation have been defined, the presented structure for this dissertation is as follows:

In Chapter 2, it is conducted a literature review of the state-of-the-art, regarding the platforms that exist in the market for the management of IoT Devices in a home context, a literature review is also conducted for the methods of object recognition.

In Chapter 3, it is presented the design and development process that is not only expected to identify smart objects, but also control them, providing a high-level system description of the system and its development steps. For this purpose, it was developed a Siamese neural network in order to handle the identification of the smart objects.

In Chapter 4, the tests that were performed on this system in order to validate its operation are presented and the results of said tests are discussed. In this chapter it will also be discussed

how the system is able to identify correctly the objects that were present on the dataset, most importantly for new images that were presented to the network.

Finally in Chapter 5, this dissertation's final chapter, a conclusion for the implemented system is presented. In this chapter it is also presented a discussion regarding some of the future work that could be carried out in order to ensure the results of this research are improved. In this chapter it is also discussed how the purposed system was able to match the constraints that were presented during the previous chapters.

CHAPTER 2

2. Related Work

Mobile devices in the context of Home Automation and Object Recognition are topics being explored with great depth. However, a combination of both topics isn't that common. Furthermore, it is even less common that Home Automation takes advantage of all the sensors contained in mobile devices, such as its camera, and in particular Computer Vision-based solutions for Object Recognition. Nevertheless, there is a great number of publications and work related to implementing Object Recognition in the mobile environment.

To understand the aim of this work and all its components, it is first needed to understand prior work that has been developed in each of these research fields and how it will be used in the work developed for this dissertation. As already explained in Chapter 1, there is a great number of protocols and platforms for IoT, such is the case with Home Automation. This chapter will summarize and compare major relevant platforms that exist today for Home Automation and the selection of one of them for the present work will be justified. After this, this chapter will also explore how object recognition works and all the methods that it can use, and how it can be used on a mobile device.

2.1. IoT Devices and Home Automation Platforms

2.1.1. Platforms

Just like was mentioned before, the number of platforms that offer home automation management is growing, however, there are a few platforms that stand out from the rest due to either the fact that they are open source or not, the language in which it has been developed, the mobile platforms in which it can run and, maybe not less important, how easy it is to set up the platform or configure integration and automation of IoT devices. A detailed approach on each platform will be presented after a simple table, that can be seen in Table 1, showing a comparison between each platform based on these topics:

- Open-Source
- The language that has been developed on
- The difficulty of installation of new devices and the platform itself
- The number of supported devices

- Mobile App
- The devices in which it can be run
- The number of worldwide users
- If it allows automation rules

With these topics the platforms that will be in the study will be:

- Home Assistant[1]
- OpenRemote[2]
- Domoticz[3]
- OpenHAB[4]

Table 1: Comparison of different platforms (Adapted from[1])

Topic	**Platform**			
	Home Assistant	**OpenRemote**	**Domoticz**	**OpenHAB**
Open-Source	Yes	Yes	Yes	Yes
Developed Language	Python	Java	C/C++	Java
Installation	Easy	Medium	Medium	Easy
Number of supported devices	Large	Large	Limited	Large
Mobile App	Yes (iOS only)	Yes	Yes	Yes
Systems it can run on	macOS, Windows, Linux, Raspberry Pi	Linux, Windows, Mac OS X	macOS, Windows, Linux, Raspberry Pi	macOS, Windows, Linux, Raspberry Pi
Number of Worldwide users	Large	Medium	Large	Large
Automation Rules	Yes	Yes	Yes	Yes

[1] https://www.home-assistant.io/
[2] https://openremote.io/
[3] https://domoticz.com/
[4] https://www.openhab.org/

2.1.2. Home Assistant

Home Assistant is an open-source home automation software that was designed to be the central home automation control system. Both of its components, the core itself and its extensions, are written in Python focusing on main control and the privacy of its users, data is stored locally instead of stored in a server.

In November 2020 it supported over 1700 plug-ins or addons for different IoT technologies, systems, and services.

In 2020 on Github's state of the octoverse, Home Assistant was named the second Python package with the most contributors, which means that the community for this platform is very extensive.[1]

2.1.3. OpenRemote

OpenRemote is an open-source IoT solution that is available for smart buildings and smart cities. It integrates a lot of different protocols and solutions while offering visualization tools that can be used on a tablet or a smartphone.[2]

Just to give an example, OpenRemote is currently the solution that the city of Arnhem in the Netherlands has selected to develop an energy management system. The system can forecast the power generation, consumption, and carbon cost of a solar and wind park for the next 24 hours allowing the city to make a detailed analysis of its solar park.[3]

2.1.4. Domoticz

Domoticz is a free and open-source home automation system that allows a user to monitor various IoT devices such as lights, switches, etc. It has been developed using C/C++ with an HTML5 frontend that allows the system to adapt to either desktop or mobile devices. It allows for push notifications and the system can auto-learn switches/sensors. It was designed with a simplistic approach in mind. However, its community is small compared with other platforms, especially when compared with OpenHAB or Home Assistant.[4]

2.1.5. OpenHab

OpenHab is an open-source platform developed in Java, it is part of the Eclipse Foundation, and it supports integration for over 2000 devices and hundreds of technologies. It allows the integration and configuration of devices into a single solution either through text files or a Web Interface, which allows users to customize a multitude of automation scenarios. All the actions, like controlling light switches, controlling temperatures, etc. are triggered by rules, voice commands, or controls, and are done also either through text files or with the help of a Web Interface. It has a simple installation and it can be installed on almost any hardware that can run Java Virtual Machines (JVM).[5]

Since OpenHab is one of the most well-known and established solutions on the market it has a large community of worldwide users, it offers the possibility to work with either an Android or IOs application, and the fact that it is developed in java and previous experience with this platform, make it the best solution to use in this implementation.

2.2. Object Recognition

This topic is very relevant nowadays and with that in mind, there have been a lot of scientific contributions each discussing the way Object Recognition can be implemented. With that in mind and to synthesize all the information given in those scientific contributions, there have been a lot of literature review papers with that goal. [6][7][8]

2.2.1. Traditional Object Recognition

In [6], the authors of the paper focus primarily on object detection and tracking, however, some of the methods that they discuss in the paper can still be applied and used in object recognition, the authors present a table comparing the various methods which are presented in Table 2.

Table 2: Comparison of object detection methods [7]

Method	Pros	Cons
1. Background Subtraction Method	• A very widely used method that is simple to implement • Objects are allowed to become a part of the background without destroying the existing background. • It learns itself and does not need to be reprogrammed. • Can be implemented in any application. • Provide fast recovery. • Low memory requirement.	• Highly inaccurate • Cannot deal with quick changes. • Initializing the Gaussians is important. • Not a good subtraction when shadows, or any other obstacles, are there. • Gives false positives • It does not survive with a multimodal background.
2. Real Time Background Subtraction and Shadow Detection Technique Theory	• The accuracy of this method is higher than the frame difference • It detects shadows as well	• The algorithm based on this method is quite complex
3. Template Matching	• The best method for a specific environment	• Only occurs when there's a one-to-one match • Slow process for recognizing new variations of a pattern. • No scanning process is done on the percentage so there is no guaranteed accuracy • It only works if the object is always in the video, otherwise, it will create a false detects
4. Image Differencing	• Simple and straightforward • Easy to interpret the result	• A different value is absolute so the value may have a different meaning • Require atmospheric calibration • Requires selection of thresholds
5. Shape Based	• Simple pattern-matching approach • Having unable to moderate accuracy	• More striking technique • Often used as a replacement for local features • Does not work well in dynamic situations • Unable to determine internal movements well • Computational Time is low
6. Optical Flow	• It can produce complete object-moving information • Contain enough accuracy	• Require a large amount of calculation
7. Frame Differencing	• Perform well for static background • High accuracy • Easiest method	• It must require a background without moving objects

Method	Pros	Cons
		• Method having Computational time low to moderate
8. Motion-based	• Does not require predefined pattern motion detection	• Struggles to identify a non-moving human
9. Texture based	• Provides improved quality with the expense of additional time	• High computational time

A few of these methods, despite having a video-based approach in mind, can still be applied in an image-based or object recognition system, specifically, the Template Matching method that can be applied to an image which consists of matching each frame with a specific template of an object, even though it's one of the most accurate methods, it works best in a specific environment. In Figure 2 a flow chart of how this method works is presented [9]:

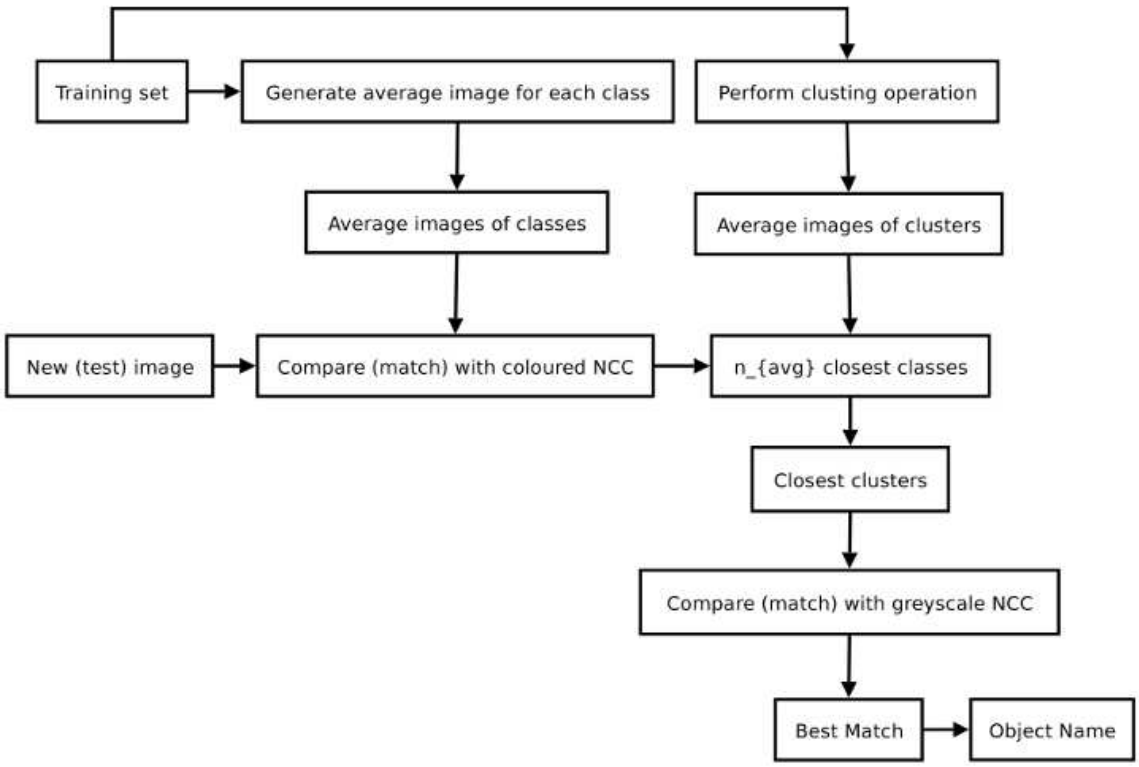

Figure 2: Flowchart of how Template Matching works [9]

From this simplified flow chart, we can understand a simplified version of the Template Matching method: the new image first gets compared with average images for each class, and then it gets compared with clustered images of other classes. In [10] it is presented a snippet of code on how to implement this method using OpenCV, an open-source library that contains an extensive number of computer vision algorithms [11].

2.2.2. Object Recognition with Deep Learning

Applying Deep Learning to object detection methods can help traditional object detection methods, by proving better results than the traditional object detection methods

2.2.2.1. Region-Based Convolutional Networks

The Region-Based Convolutional Networks (R-CNN) belong to one of the most used families of deep learning-based object recognition methods. Figure 3 presents a flowchart of how R-CNN can be applied to an image.

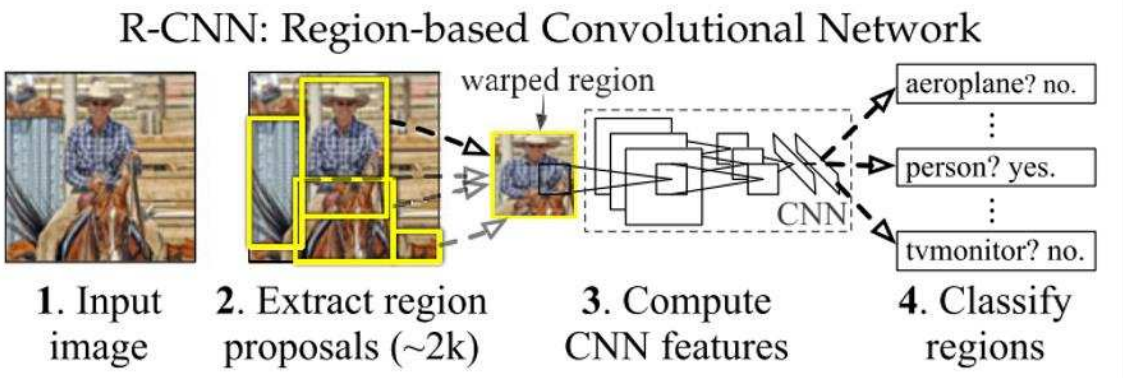

Figure 3: Flowchart of how R-CNN works

The system takes an image, then takes around 2000 region proposals, computes features for each proposal and after that, it classifies each region [12].

2.2.3. MobileNets

This method was developed contradicting the trend that was happening in this subject, which was of creating deeper and more complicated networks, which not always meant that the performance and accuracy would be better. This method uses an efficient network architecture and two hyper-parameters to build small and low-latency models that can be implemented in mobile devices [13]. An example of this can be seen in Figure 4.

Figure 4: Object detection using MobileNet

2.2.4. Applying Object Recognition on Mobile Devices using TensorFlow

Using an Object Recognition System on a mobile device is a relatively small concept, however, there is still a small percentage of literature explaining how it can be applied to a mobile application making use of TensorFlow [14], which is an open-source library developed by Google for machine learning, its focus on training deep neural networks. Next is presented a figure that will display the architecture of a Mobile App using the TensorFlow API [15]:

Figure 5: Object Recognition with TensorFlow

The Mobile Application will send an image to the TensorFlow API, which can categorize the data into 1000 classes, those classes are linked with the name of the detected object, returning its name and a score that represents the accuracy of the recognition. To implement the API into Android the API uses Android Camera2, a package that supports Java Native Interface to interact with the TensorFlow engine.

CHAPTER 3

3. Design and Development

This chapter presents the design and development process for creating a solution that is expected to identify and control smart objects. To achieve the goal of this implementation this solution was split into the two parts depicted on the diagram shown in Figure 6: Smart System and IoT Devices Hub. The first part, the Smart System, addresses how it is possible to identify a certain object in a room using an image acquired by the mobile device camera sensor as input. The second part, the IoT Devices Hub, addresses how it is possible to control the identified object in a Home Automation platform.

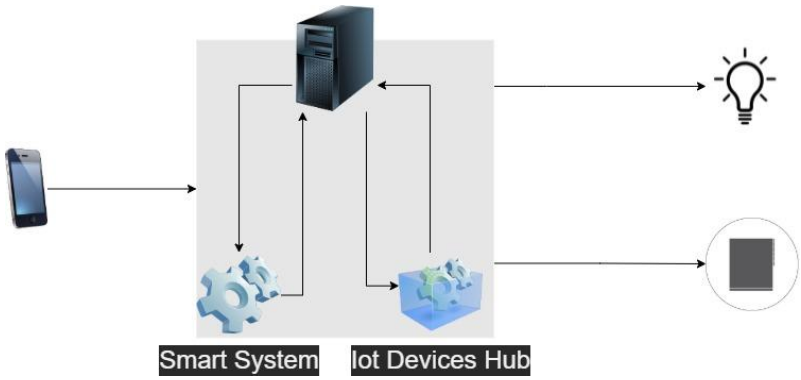

Figure 6: Diagram of the final architecture of the proposed system

Regarding the first part, the system must be capable of receiving an image sent from the user's mobile device, saving it in its memory, and then using the image as input for the smart system. This system will then return the class to which a major object in the image belongs, as illustrated in the interaction illustrated in Figure 7.

The second part of this system, detailed in Figure 8, must be capable of receiving the class of the object that the user wants to control and a command, example ON or OFF, to be sent to the IoT device. The IoT Devices Hub will be responsible for understanding which objects to control and to send the command to it.

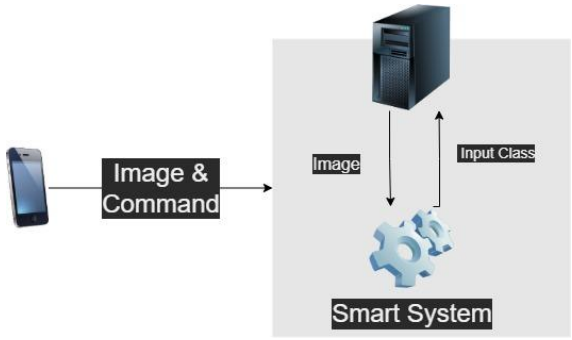

Figure 7: Diagram of the communication between the user's mobile device and the local system

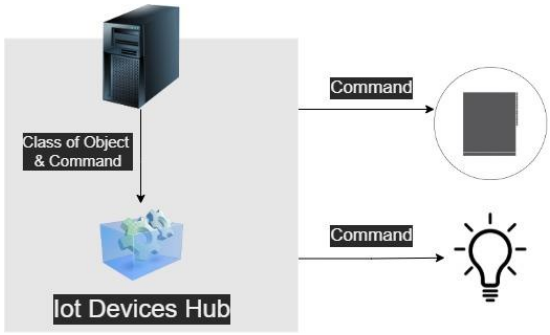

Figure 8: Diagram of the communication between the local system and the IoT devices

3.1. High-Level System Description

For this system, when the user intends to add another object that he wishes to control he must add several images to the dataset also with the name of the class that he wants to add, this will be done through an App that was developed for this implementation in order to aid the end user. The interaction with the user can be seen in Figure 9.

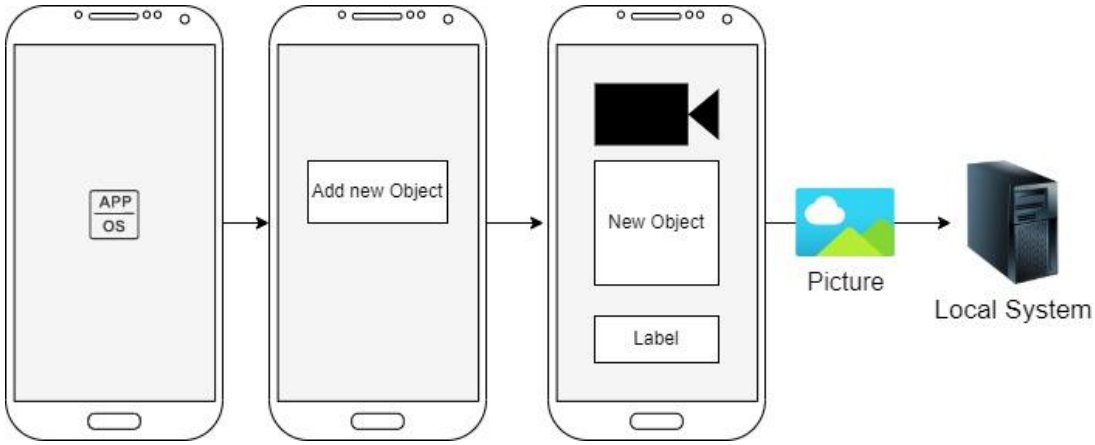

Figure 9: Diagram of how the developed App will interact with the System

When the user wants to control an object, a new one, or an existing one, the user needs to take a picture of the object that he is trying to control. Afterward, there are two possible routes for this architecture:

1. One possibility for this architecture is to run the smart System inside the developed App. With this approach, the system would be faster, without the need to communicate with external devices, but the smart system would need to be adapted in order to be able to run it on a device that is powerful but with limited capacities, such as a smartphone. This architecture can be viewed in Figure 10.

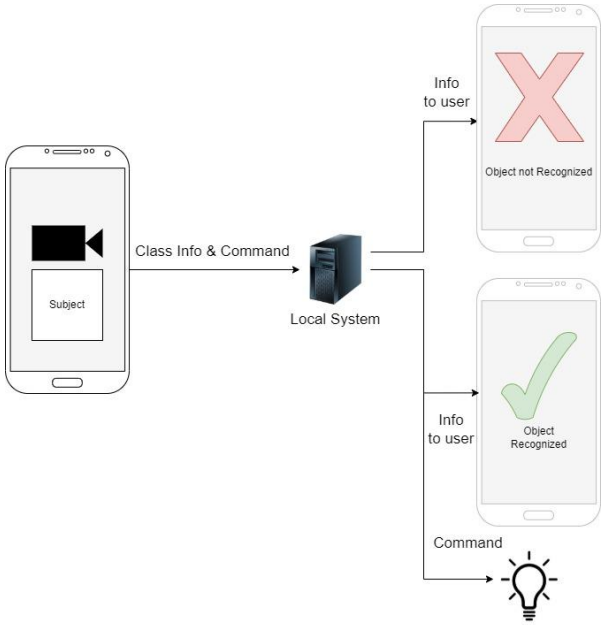

Figure 10: Possible Architecture for the system, where the Smart System runs on the mobile device

2. The alternative to the first architecture was to run the smart system inside a server and then communicate the results to the user's smartphone. For this architecture, the smart system would not have to be adapted and so it would preserve the execution time of the system. However, with this approach, it is necessary to maintain communication with external devices for a period of time, which would not only add to the latency of the communications but also present several points of failure that may disrupt the capabilities of the integration such as one point of communication fails the entire system will fail, for example, if the communication between the user's mobile device and the local system is broken, the system cannot proceed. A diagram for this type of architecture can be seen in Figure 11.

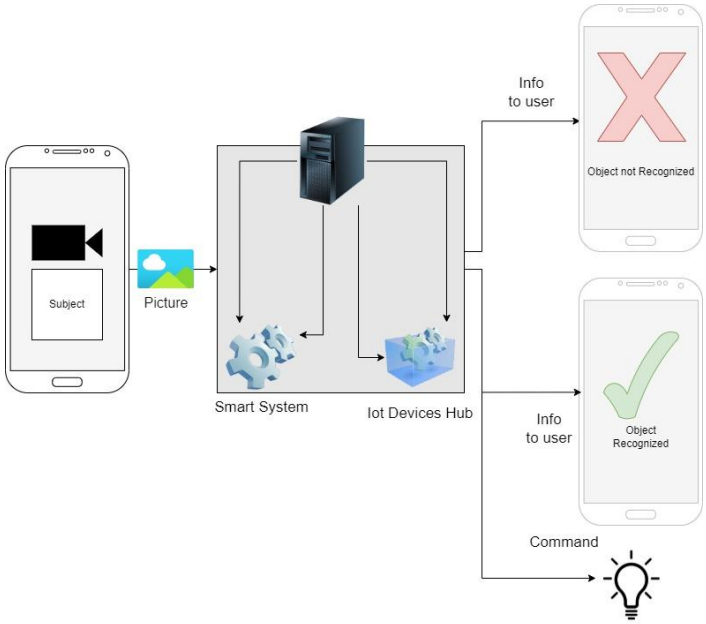

Figure 11: Another possible architecture for the system, where the Smart System is running on a Local System

3.2. Development

This section will focus on the development[5] of the system presented in the former section. To achieve this purpose, the used dataset will first be presented. After this, the type of Siamese network that was designed and implemented will be presented, and then explained how they were constructed and the limitations that are imposed by using the type of architecture that was chosen. At last, this section will also present the training parameters that were applied to this model in order to train it to be able to achieve the desired purpose.

3.2.1. Collecting and Preparing the Dataset

In every approach using a Machine Learning model, it is necessary to build a dataset from which the model will optimize its parameters from the training process. The machine learning model for this implementation will need to determine which class of images is the most similar to the image provided as input for the network. Since the model is classifying the input image based on similarity, a Siamese network approach was followed. In this type of network, the inputs are a pair of images and the output is a similarity measurement between those images. For the training process, the number of images in the dataset does not need to be large, since with a small number of images it is possible to generate a large number of image pairs by

[5] https://github.com/cdsjg-iscteiul/Tese_Flutter

combining them. For instance, with eighteen images it is possible to form 153 different pairs of images (without repetitions).

To start the process of creating the dataset it was decided on the appropriate approach to create it, considering the machine learning model is expected to determine which class is the most similar image to the input, the dataset must contain in each class a type of object that can be used.

Considering the proposed utility of the model and the structure discussed above, the dataset was built according to the illustrated in Figure 12. Even though the Siamese neural networks do not require a lot of images for training when compared with direct image classification approaches, the images acquired for each object should cover a wide number of viewpoint angles, especially when the images do not have a lot of differences between them. The dataset was therefore built with that in mind, making each class have almost a 360º view of each object.

Figure 12: Representation of the dataset used

3.2.2. Network Design

Before we can discuss the architecture that was chosen for this network, it is necessary to consider the requirements that this system must meet, which are:

- The system should not be trained every time a new device is added.
- As mentioned in section 3.1 this system can have two possible architectures, one of which requires the neural network to run on a mobile device. As such, the neural network should be lightweight and capable of running on a mobile device or a device with little processing power.

- The network has to be modular, introducing modularity in the network could significantly improve the time required to make a classification, which would significantly improve the end-user experience.

Considering all the requirements presented, the neural network architecture that best fits this system is that of a Siamese network. The defining characteristic of a Siamese network is the fact that it counts 2 inputs that converge into only one output, as shown in Figure 13, this output is used to make a comparison between the two inputs resulting in a value that can tell us if the two images are of the same or from the same object.

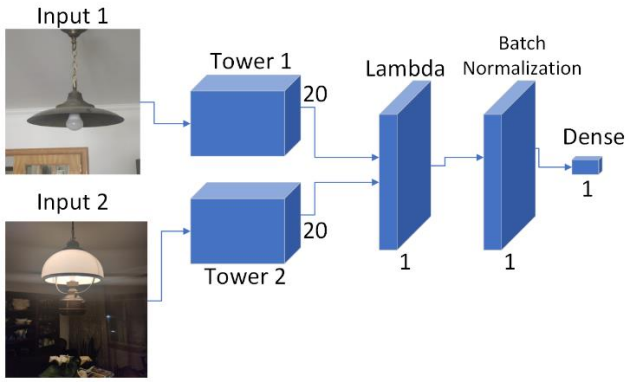

Figure 13: Diagram of the architecture of the final Siamese network

This system created the network that is shown in Figure 13. Within this network, there are two equal towers, that correspond to convolutional neural networks (CNN) without their top classification layers. Two different CNN architectures have been experimented.

One architecture uses a pre-trained MobileNet model, represented in Figure 14, whose network weights were already trained using a very large image dataset – using the pre-trained model, each tower is already trained, and therefore the overall model takes less time to be trained.

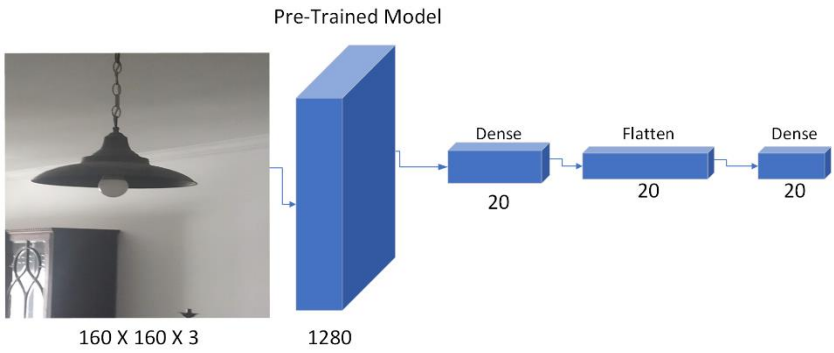

Figure 14: Diagram of the architecture for the towers that are part of the Siamese network, using the pre-trained network

The second architecture that was built from scratch, represented in Figure 15, where each network weight can be customized to the task at hand – it could potentially achieve better results at the expense of more processing power and a larger training time. These questions will be approached and answered further in the dissertation.

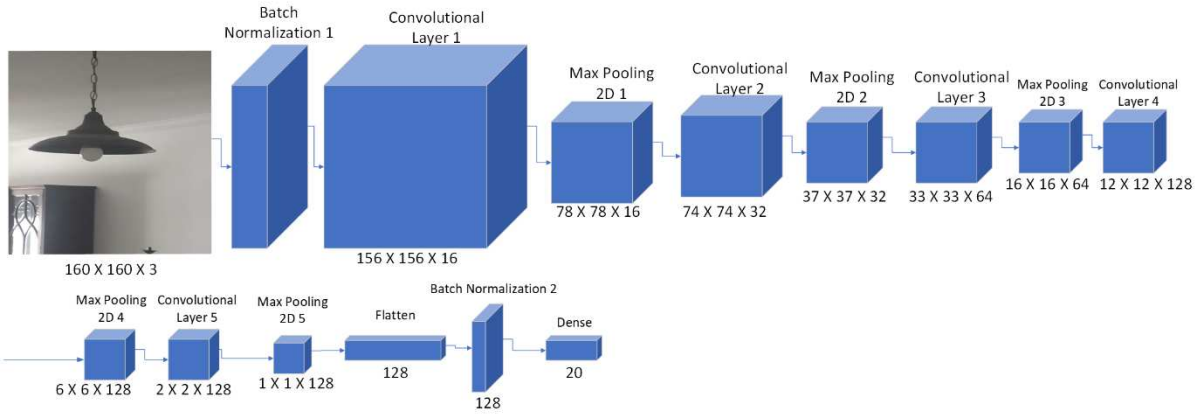

Figure 15: Diagram of a tower that will be used on the final architecture of the system using the network built for this purpose

This module consists of a lambda layer, a batch normalization layer, and a Dense layer. The lambda layer is responsible for using the features of each image and performing the calculation of the Euclidean distance that provides the degree of similarity between the two images. It outputs a value between 0 and 1 depending on the degree of similarity of images: 0 is for images that are rather different, and therefore belong to different classes; while 1 is for images that are very similar, and therefore should belong to the same class. An example of this classification can be observed in Figure 16.

Figure 16: Example of how the labels are associated with different pairs of images, 1 for images with the same class and 0 for different

Besides the tower, the Siamese network architecture also includes a module containing the layers that receive the tower's output which can be interpreted as image features. As stated before, it is required that the system is sufficiently light to run on a system with little

computational power. In order to save computational power, this module has the ability to receive the features from the towers' module of the network as well as receive them from a file that contains the information stored for the image features. An example of this architecture can also be viewed in Figure 17.

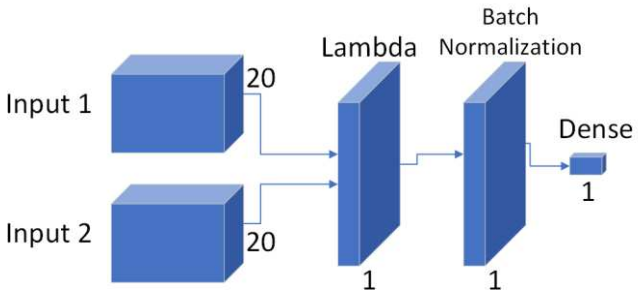

Figure 17: Diagram of the proposed architecture for the network responsible for outputting the similarity between two images

3.2.3. Network Training Process

This section presents how the Siamese network was trained. Due to the particularity of this system, this network must be trained following the network diagrams that were presented in the previous section. The system must be trained following a set number of parameters summarized in Table 3

Table 3: Summary of the training parameters for this system

Number of classes for training	24
Number of images for each class	13
Number of pairs	312 samples to form pairs, forming 48516 pairs
Training and validation split	0.7
Loss Function	contrastive loss
Batch size	32
Epochs	100
Activation Function	Sigmoid

Some parameters are specific to this type of network, here detailed:

- The number of classes for training – this parameter specifies how many different classes will be used for training. Each class represents an object that can be identified when the system is completed. All training trials use 24 classes.

- The number of images for each class – represents how many pictures for each object are present in the dataset. In this case, each object has 13 associated images. Remember that this type of architecture does not require a large number of samples for each class, which justifies this small number of images per class.
- The number of pairs – this parameter depends directly on the previous parameter, since this architecture takes a pair of images as input, it is possible to multiply the number of samples greatly from the number of inputs of samples in each class. For example with 13 samples in each class that was presented previously, means that we have 312 unique samples to form pairs with.
- Training and validation split – this parameter specifies the split between how many samples are provided for training and validation, for this system the split was set at a proportion of 70:30, this means that 70% of samples were used for training and 30% for validation. In absolute numbers, there are 218 samples to form training pairs and 94 samples to form validation pairs
- Loss Function – this is a function that will compare the target and predicted output of a network. This function measures how well the network performs with the training data. The goal of the training process this function is to minimize this function. The loss function that was used was the contrastive loss function, which is commonly used for training Siamese networks. It calculates the distance between the two images, using the following formula [16]:

$$\ell_{i.j} = -\log \frac{\exp\left(\frac{sim(z_i, z_j)}{\tau}\right)}{\sum_{\substack{z_k \in \{z^P \cup z^q\}, \\ z_k \neq z_j}} \exp\left(sim(z_i, z_k)/\mathcal{T}\right)} \qquad (1)$$

  where Z is the set of representations in the projection space, z_j is the representation of the other view of the sample, τ denotes a temperature parameter.
- Batch size – the value of this parameter specifies the number of samples that the network uses for the training on each optimization step. The batch size was set to 32, meaning that the system will train with 32 pairs of images at each step, this value was achieved comparing the performance of the network with different values, arriving in the final value of 32.
- Epochs – how many times the network uses all training samples to produce an optimization process iteration. The higher this value the more time the network will spend during the training process and a better overall result for the training is produced.

However, after a certain number of training epochs, it is possible to observe that the network does not improve the results on the validation set, meaning that further training may lead to overfitting, the final value of 100 was achieved measuring the performance during various number of epochs, and noting that after that value the system would not benefit from additional epochs.

- Activation Function – this function defines the output of that node given an input or set of outputs. The chosen activation function was the sigmoid since the output of this function is between 0 and 1 which is adequate for representing the similarity of the two images provided as input. An equation and plot for the sigmoid activation can be observed in eq. (2) and Figure 18:

$$f(x) = \frac{1}{1 + e^{-x}} \qquad (2)$$

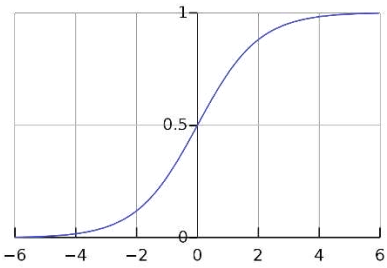

Figure 18: Graph for the sigmoid activation function

In Figure 19, it's possible to see the results from the training that was done using the parameters that were mentioned in the table before, in the architecture using the model build for this system, while the results using the pre-trained model can be seen in Figure 20.

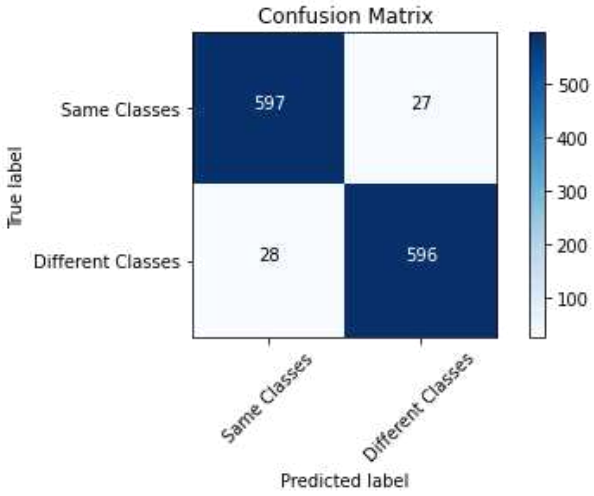

Figure 19: Confusion Matrix for all the sets of training using the model built for this system

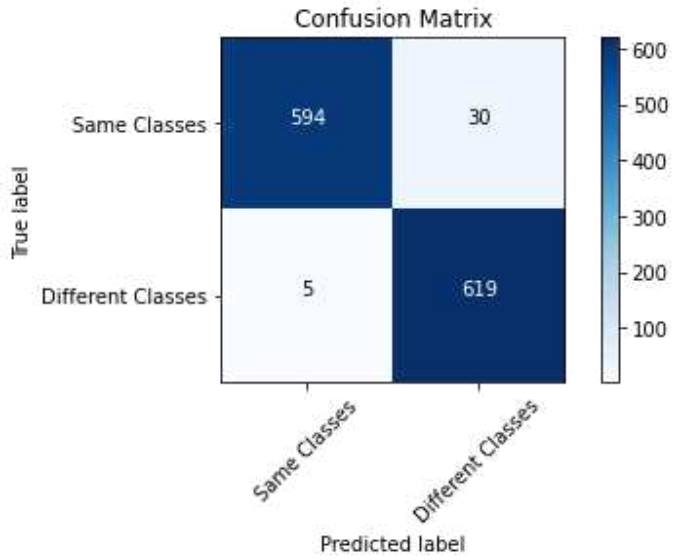

Figure 20: Confusion Matrix for all the sets of training using the pre-trained model

These Confusion Matrixes represent how the system performed for all the sets in the training, the training set, the validation set, and finally the test set. These results will now need to be used to perform tests and see how the network after being trained will perform when faced with the problem where this system is expected to act.

CHAPTER 4

4. Tests and Results

This chapter presents the tests performed on this system to validate its operation and shows and discusses the results of those tests. Throughout this dissertation, especially in Chapter 3, several requirements that the system must satisfy were presented. A summarized list of those requirements can be seen in the following list:

- The network must be able to detect with high accuracy if an image given as input is a part of a class that was trained with and when presented with new images, different from the ones that were trained with, it must be able to depict which class is a part of.
- The network must be lightweight, as talked before this network must be able to run in an environment that has low processing capabilities, which means that in an environment with a great capability of processing it should be accessed how much time it takes to train the network.
- The network must be able to adapt to new classes, when adding a new class it must be able to distinguish the new class from every other single class that was already present in the dataset, this action will be triggered by the user when adding a new class.

4.1. High Accuracy Detection

Regarding the first requirement that was presented, and with every network of this kind we must first validate the networks training and the values of its accuracy and the loss, this step is important in every machine learning model because this will determine how well the network has trained and how well the network will be able to achieve the desired result.

As discussed in the previous chapter there are two possible architecture models that we can follow for this network, the first one is the architecture that was made from scratch, the results for this architecture's accuracy and be viewed in Figure 21, and the results for its loss in Figure 22, on the other hand, the results for the architecture where we used the pre-trained network can be viewed in Figure 24 for its accuracy and Figure 25 for its loss. For these results a good

target to aim for is a very high value of accuracy and a very low value of loss, this means that our network can be able to make a great number of predictions with very small errors.

As we can see in both Figure 21 and Figure 22, the results are very good since the model has high accuracy and low loss, meaning that the model is able to perform predictions with a very low level of errors. An example of those predictions can be observed in Figure 23, where it is possible to see the true and predicted values of those images' pairs, 1 if they are from the same class (i.e., same object) and 0 if they are not.

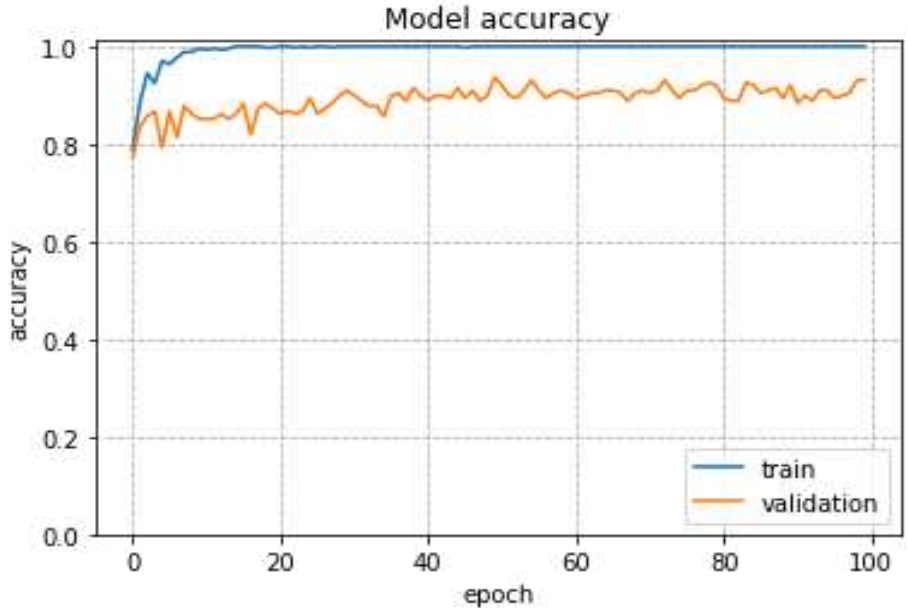

Figure 21: Results for the models' accuracy with the model that was made for this system

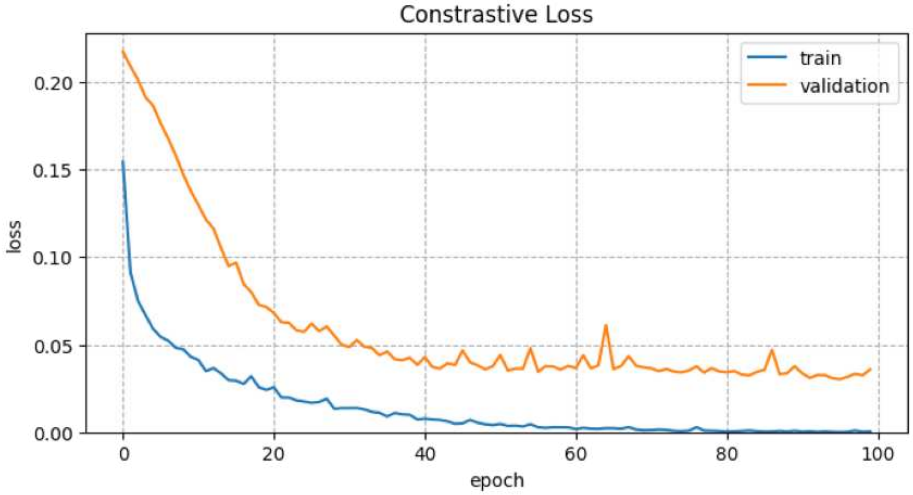

Figure 22: Results for the models' loss with the model that was made for this system

Figure 23: Prediction values for the inputs with the model that was made for this system

Regarding the other architecture where we employ the use of a pre-trained model, the results for its accuracy and its loss can be viewed in Figure 24 and Figure 25, respectively also the same example of predictions can be viewed in Figure 26. It is possible to conclude that as the same as the previous architecture the results are very good meaning that both architectures are viable to implement on this system.

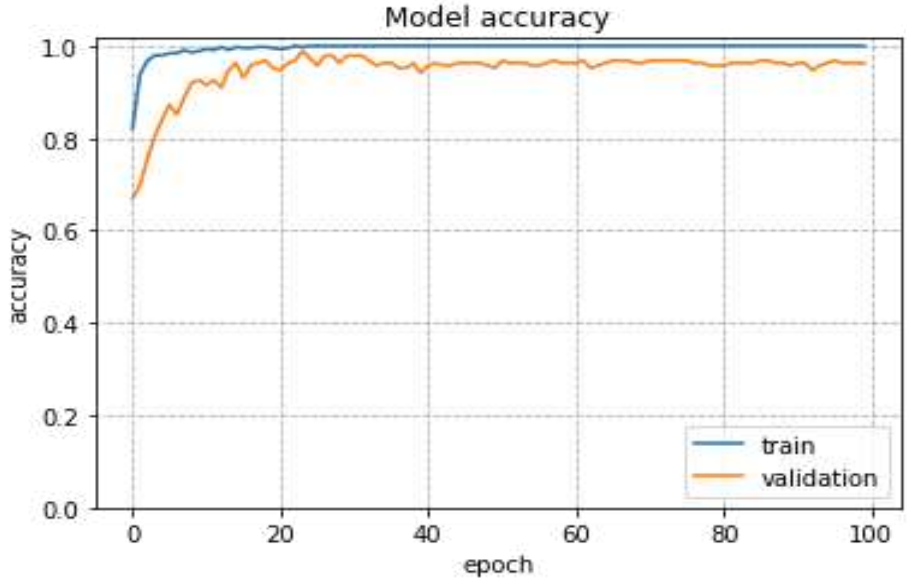

Figure 24: Results for the models' accuracy using the pre-trained model

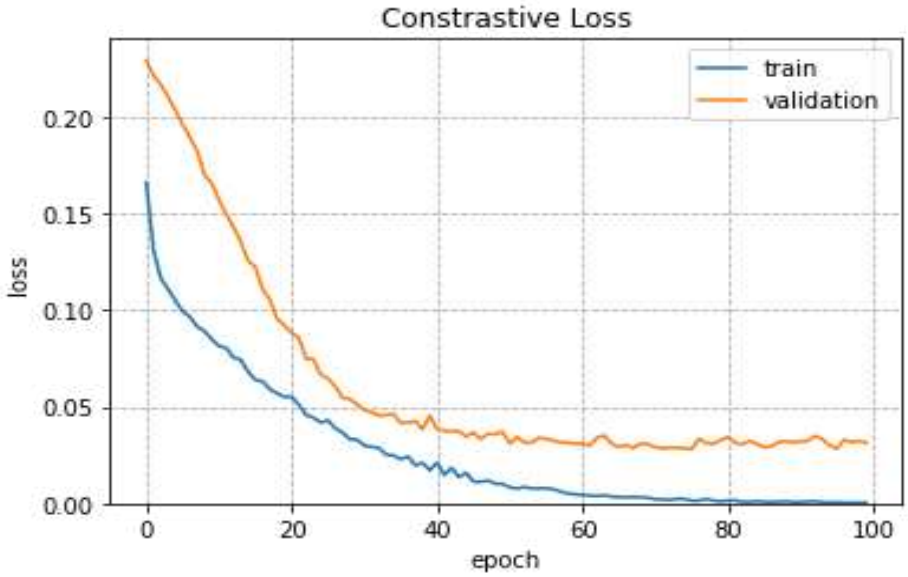

Figure 25: Results for the models' loss using the pre-trained model

Figure 26: Prediction values for the inputs with the pre-trained model

For both models, we can compare the training results in Table 4. Even though both architectures have similar results, the architecture with the pre-trained model has a slightly better one, consequently, we will consider from this point forward using the pre-trained model in the system.

Table 4: Training results for each architecture

	Model Built for the System	Pre-Trained Model
Training Loss	0.0017	0.0010
Training Accuracy	1.000	1.000
Validation Loss	0.0586	0.0313
Validation Accuracy	0.9309	0.963
Test Loss	0.0512	0.0437
Test Accuracy	0.9391	0.953

4.2. System Performance

After being able to process the images proved as inputs and being able to extract their features and determining if they are similar, it is required for the system to be able to determine in which class a certain image given by the user belongs, for this and using the limitations for this system, for this and to improve the processing time and to attend to the limitations, all the features of the images dataset are pre-loaded using a pickle (.pkl) file and every time the user adds new images they will be added to this file. For making the processing of the dataset easier on the system in which it will run, the structure of this file is [Path, Class, Name, Features]. For example, for a file that has path "\Database\Data_Test_2\Bedroom_Lamp\6" in the dataset, the system will determine the parameters and create a line in the pickle file with the structure ["\Database\Data_Test_2\Bedroom_Lamp\6", Bedroom_Lamp, 6, Image's Features].

For the system to determine to which class the image submitted by the user belongs, the submitted image is first passed through a network tower to extract its features. These features will then be compared with every reference image features present in the dataset (using the pickle file mentioned above). After that it is possible to determine to which class the image belongs to with multiple ways. Two ways were implemented in the context of this work. The simplest way was to check which reference image has higher similarity score when compared with the image submitted by the user. The predicted class for the user's image will be the class of the most similar reference image. Another way to determine the class of the image submitted by the user is using the k-most similar images. In this case a parameter k is provided and the system determines the k most similar images and their classes. The class that occurs most frequently in this k-sized set is the class predicted for the users' submitted image. If in case of ties, the predicted class will be the one showing the pair with highest similarity value.

For the method that uses the most similar classes, i.e., K Similar, the confusion matrix, that resulted from using this method to predict the class to which all the images in the dataset belong to using the value of k as 7 can be observed in Figure 27.

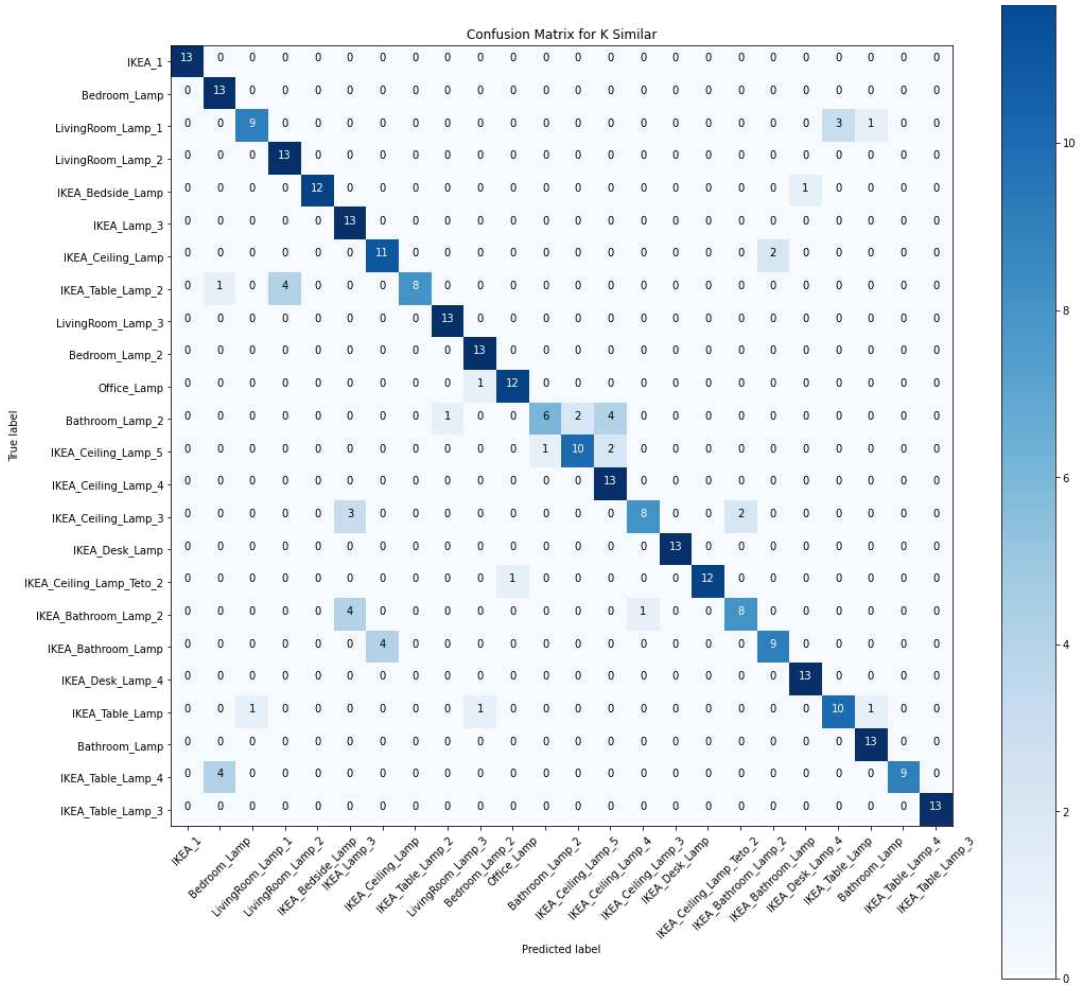

Figure 27: Confusion matrix using the K Similar Method

The confusion matrix shows that most of the performed predictions were correct. However, there were a few inaccurate predictions, especially for the class IKEA_Bathroom_Lamp_2, where the system could only correctly identify 8 out of 13 samples, predicting it 4 times as the class IKEA_Bathroom_Lamp. A reason for this could be due a high similarity between the photos acquired for these objects. One possible solution for this is to acquire additional photos of different angles or using different lighting conditions.

For the method that considers the class of the image with the highest similarity value when compared with the image submitted by the user, i.e., Highest Similarity method, the confusion matrix produced is depicted in Figure 28. From the matrix, a different conclusion from the previous method can be drawn. Using the Highest Similarity method, the network can correctly predict all samples. Considering these results and comparing them with the previous method, it is possible to conclude that this method performs better for classifying the dataset images based on similarity measurements.

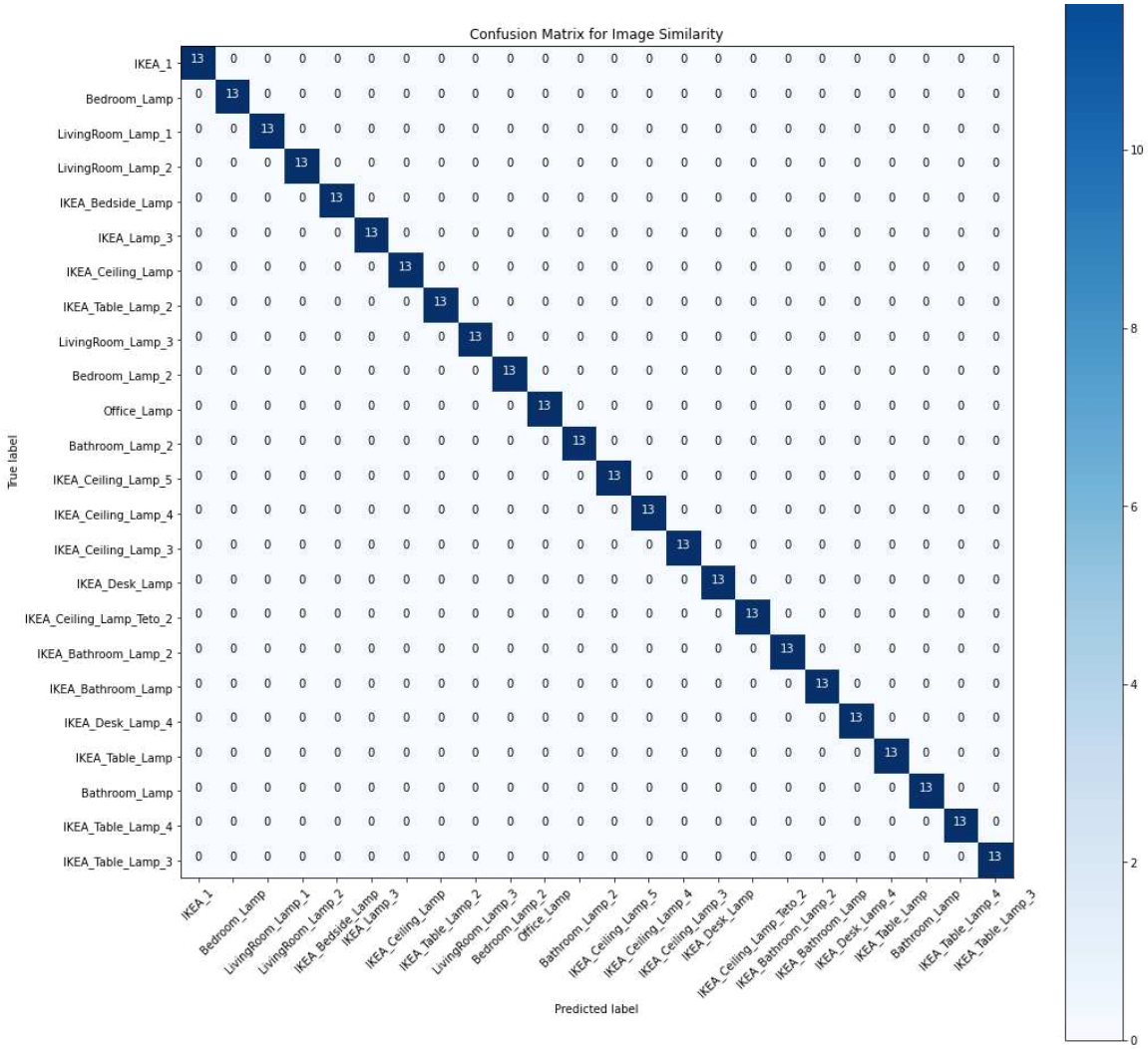

Figure 28: Confusion matrix using the method based on the Highest Similarity method

For this system to work properly and to provide a good user experience for end users, the system must also be able to handle all communication and all processing in a fast manner. Since the time it takes to communicate with the local system is something that is only possible to minimize to a certain extent, we must ensure that the time for processing the image is as low as possible. One hypothesis that was already addressed in Chapter 3 was the possibility of splitting the neural network into two modules, one that would extract the features of each input and another that would take the extracted features and measure the distance between the two features that were extracted on the first module.

In Table 5 it is possible to analyse the measured time for each step of the process that was described before, from extracting the features of one image to running the entire system using the pre-trained model that was used during the system's development. These time measurements were performed using a computer equipped with a RTX 2060 GPU, a Ryzen 5 3600X 6-Core CPU and 32 GB of RAM under the Windows 10 operating system.

Table 5: Different times the system takes for different scenarios using the pre-trained model

	Run 1 (s)	Run 2 (s)	Run 3 (s)	Run 4 (s)	Run 5 (s)	Avg. (s)	Std Dv
Run the whole system with 2 images as input	0.372	0.366	0.373	0.357	0.372	**0.368**	0.00603
Extract Features of a Single Picture	0.199	0.203	0.198	0.190	0.193	**0.197**	0.00459
Save features of the entire dataset on a file	39.158	39.207	39.616	38.860	39.745	**39.317**	0.322
Import the file with the features of the entire dataset	0.037	0.038	0.043	0.037	0.037	**0.038**	0.00233
Import the Dataset's Features file, read it, compare them with an image and returning the class of the most similar image	0.924	0.898	0.953	0.891	0.901	**0.913**	0.0227
Import the Dataset's Features file, read it and compare them with an image and returning the class of the K most similar images	0.931	0.892	0.885	0.908	0.902	**0.904**	0.0158

As previously presented in Chapter 3, there were two possible architectures for this system. One of those would consider that the neural network would be contained inside the mobile device and, as such, it would only be necessary to communicate with the external IoT Devices Hub sending the class of object that has been identified and that the user wishes to control. For this, we would have to use TensorFlow, in order to make it possible for the neural network to run inside a mobile app. However, in this implementation, it was not possible to make use of TensorFlow inside an Android application, and, for that reason, the architecture that was implemented for this dissertation only considers the second variant. In this variant, it is necessary to consider multiple communications externally, the MQTT communications that will be needed to connect the user's mobile device to the system where the processing will occur. This means that it will be necessary to consider the added time for these communications. However, since it is expected that the mobile device will be in the same network as the local system where the image processing will occur, the additional time required for those communications is expected to be small, without noticeable impact in the usability of the application. A diagram of the communications needed for this implementation can be observed in Figure 29. Since OpenHAB is compatible with various types of protocols, the communication between the Hub and the final object may vary depending on the type of protocol the final object uses.

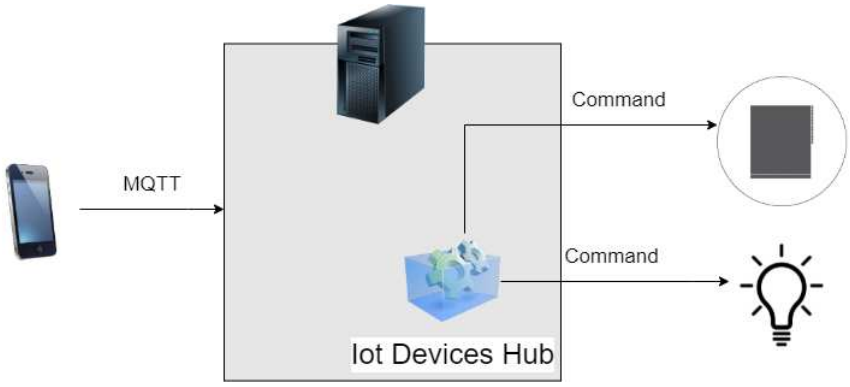

Figure 29: Diagram of the communications needed for this system

4.3. Network Adaptability

As discussed previously in this chapter, one of the requirements for this network is the ability to adapt to new classes without having to re-train the entire network. The test described in this section will allow to validate that requirement in an end-user scenario. Adding new elements to the dataset is very easy needing only to re-train the network from time to time and

not every time a new class is added. One way that we can test this scenario if to add an entire new class to the dataset and after running the same scenario tests, that were made on the previous sections.

4.3.1. Adding the New Class

As discussed, in order to validate this network as a solution for this implementation, we need to see how the network will perform when adding a new class without training for it. With that purpose in mind we first need to add the images from the class that we wish to add to the dataset, for that the user can directly access the dataset and add the new class as folder using the same structure as discussed before, in this example, the class added will be the class "Kitchen_Lamp", which for test purposes is very different from the objects in the other classes, a example of this class can be seen in Figure 30.

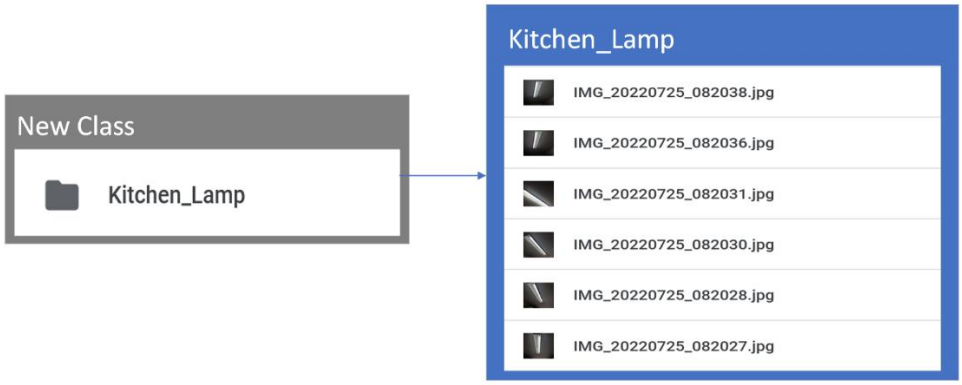

Figure 30: View of the new class added

4.3.2. Measuring the results of the New Class

After adding the new class to the dataset, it is necessary to validate if the network can adapt to this new class, for that it is necessary not only that the system is able to identify an image submitted as belonging to this class, but also that when the user is trying to identify another image from another class the system does not mistake that image as belonging to this class. To validate the addition of a new class we should consider how the prediction of the class will behave when added a new class, making the validation test of this new class the Confusion Matrix. Since we already got to the conclusion that the best method of deducing the class of a submitted image is using the Image Similarity, only that method will be used to validate the new class The new result of the confusion matrix with the new class, can be seen in Figure 31.

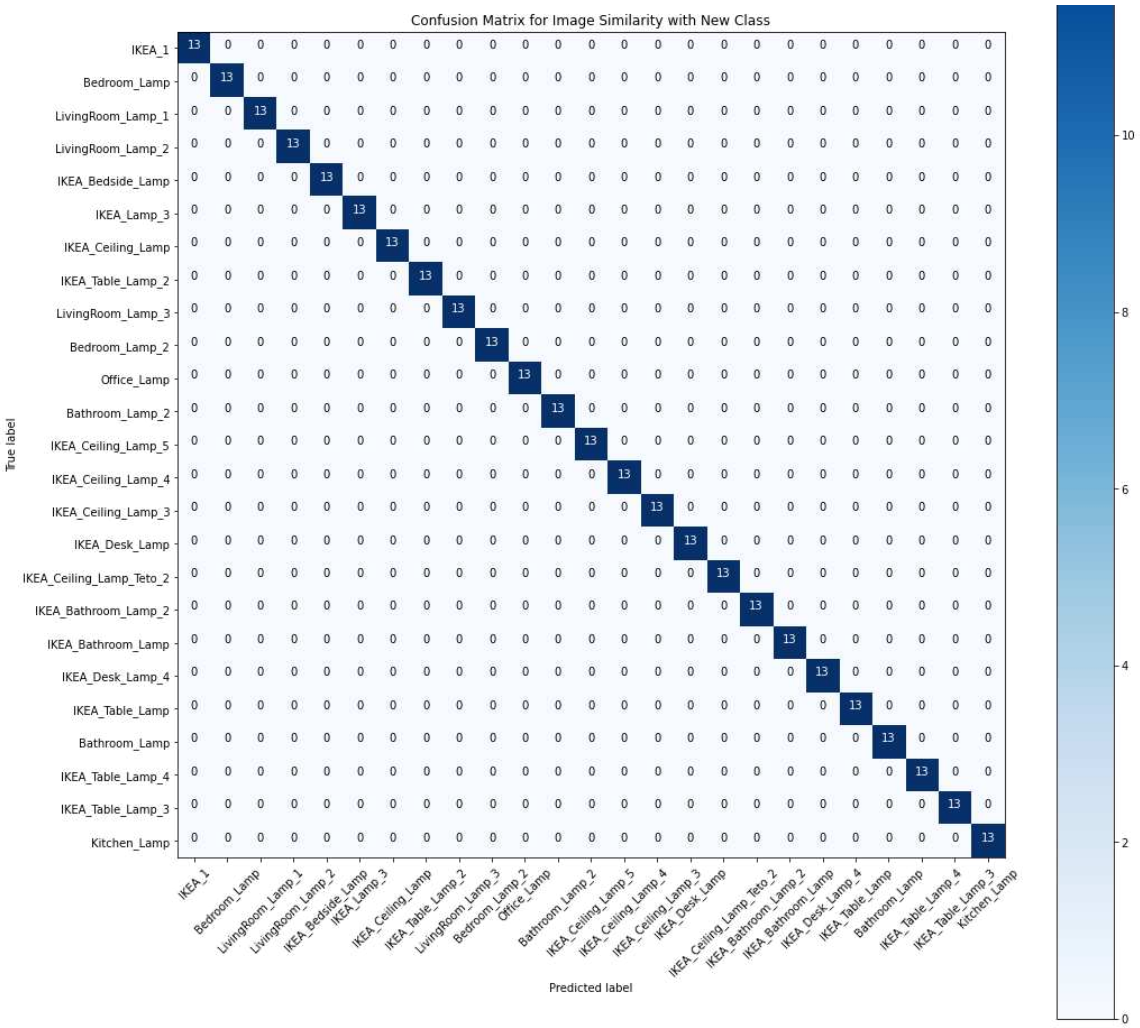

Figure 31: Confusion Matrix using the Image Similarity method after adding the new class

CHAPTER 5

5. Conclusion

In this dissertation, we aimed to present an alternative to the current systems of control of Internet of Things devices applied in a housing context, better known as home automation. Regarding the first question, whether the system should use a pre-trained network or one that was built on purpose to be applied in this implementation, after setting the training parameters in section 3.2.3 and getting its results, it was possible to understand that despite having very similar results the pre-trained network has slightly better results, making so that this should be architecture used throughout the rest of the dissertation .

In the second question we are exposed to the need for the system to be able to understand which class is part of a possible photo that is sent from the user, in order to make a correct assessment of this capability we firstly need to create functions that can help the system make those prediction, with that we were faced with two possibilities: one was to use a method based on the image's similarity between every single other image on the dataset and where the class attributed to the one submitted by the user was the class that had the image with the highest similarity with it, or to use a method that also is based on the images similarity, but instead of taking into account only one image of the dataset we use the K most similar images, after detailing the images that have the highest similarity with the submitted image we check the to which classes they belong and the class that has the most images in the most similar is the class of the submitted image, to examine the results of these two methods, confusion matrixes were built in section 4.2, where it's possible to access the performance of the system when trying to predict the class, where we can see that the method that has the best performance is the one based only on a single image similarity.

The third and final question is in relation to the system's ability to adapt to new classes being added to the dataset without the need to retrain the entire network, when the end user wants to expand these systems features and add a new object that is able to be recognized by the system, it needs to add the new object to the dataset after that the system should perform the same as before but containing the new class, as we can see in section 4.3, the confusion matrix shows that the system was able to adapt to the new class that was added.

Having responded to all the research questions that were established on the first phases of this dissertation, we can confirm that the system that was designed for this can be useful on

today society as a new alternative to manage and control new IoT devices in a home automation environment.

Some work could be done in the future in order either to validate even further this solution or either to improve upon it, some of those could be:

- Be able to implement the architecture where the neural network would be running on the user's mobile device, either to see if it's possible for this implementation to run on such devices or to improve and retrieve limitations.
- Another improvement that could be made upon this implementation is the possibility of integrating this solution using another platform to control the IoT devices, another platform (Home Assistant) has been growing in great numbers in the recent years, having this solution adapted to be able to run on it could improve its longevity.
- Make a survey with end users where it is possible to understand if the platform that was developed is an improvement on the existing solutions.

Bibliography

[1] "The State of the Octoverse | The State of the Octoverse explores a year of change with new deep dives into developer productivity, security, and how we build communities on GitHub." https://octoverse.github.com/ (accessed Jan. 12, 2021).

[2] "OpenRemote | The 100% Open Source IoT Platform." https://openremote.io/ (accessed Jan. 11, 2021).

[3] "Arnhem Sustainable Energy | OpenRemote." https://openremote.io/solution/arnhem-sustainable-energy/ (accessed Jan. 12, 2021).

[4] "Domoticz." https://www.domoticz.com/ (accessed Jan. 11, 2021).

[5] "Introduction | openHAB." http://www.openhab.org/docs/ (accessed Jan. 11, 2021).

[6] P. Panchal, G. Prajapati, S. Patel, and H. Shah, "a Review on Moving Object Detection and Tracking Methods," *Int. J. Adv. Eng. Res. Dev.*, vol. 3, no. 02, pp. 7–12, 2015, doi: 10.21090/ijaerd.c1021308.

[7] Z. Q. Zhao, P. Zheng, S. T. Xu, and X. Wu, "Object Detection with Deep Learning: A Review," *IEEE Trans. Neural Networks Learn. Syst.*, vol. 30, no. 11, pp. 3212–3232, 2019, doi: 10.1109/TNNLS.2018.2876865.

[8] G. Singh and J. Kaur, "Survey of Image Object Recognition Techniques," *Int. Res. J. Eng. Technol.*, vol. 4, no. 1, pp. 194–200, 2017, [Online]. Available: https://irjet.net/archives/V4/i1/IRJET-V4I136.pdf.

[9] L. Cole, D. Austin, and L. Cole, "Visual Object Recognition using Template Matching," *Australas. Conf. Robot. Autom.*, 2004, [Online]. Available: http://www.araa.asn.au/acra/acra2004/papers/cole.pdf.

[10] "OpenCV: Template Matching." https://docs.opencv.org/master/d4/dc6/tutorial_py_template_matching.html (accessed Jan. 10, 2021).

[11] "Home - OpenCV." https://opencv.org/ (accessed Jan. 11, 2021).

[12] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Region-Based Convolutional Networks for Accurate Object Detection and Segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 38, no. 1, pp. 142–158, 2016, doi: 10.1109/TPAMI.2015.2437384.

[13] A. G. Howard *et al.*, "MobileNets: Efficient convolutional neural networks for mobile vision applications," *arXiv*, 2017.

[14] "TensorFlow." https://www.tensorflow.org/ (accessed Jan. 12, 2021).

[15] R. Jafri and S. A. Ali, "A multimodal tablet-based application for the visually impaired for detecting and recognizing objects in a home environment," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 8547 LNCS, no. PART 1, pp. 356–359, 2014, doi: 10.1007/978-3-319-08596-8_55.

[16] I. Melekhov, J. Kannala, and E. Rahtu, "Siamese network features for image matching," *Proc. - Int. Conf. Pattern Recognit.*, vol. 0, pp. 378–383, 2016, doi: 10.1109/ICPR.2016.7899663.

[17] S. Mo, Z. Sun, and C. Li, "Siamese Prototypical Contrastive Learning," pp. 1–13, Aug. 2022, [Online]. Available: http://arxiv.org/abs/2208.08819.