Escuela
Politécnica
Superior

Social application with Pepper Robot

Bachelor's Degree in Computer Engineering

Bachelor's Thesis

Author:
Daniel Martínez Andreu
Supervisors:
Miguel Ángel Cazorla Quevedo
Félix Escalona Moncholi

July 2020

Universitat d'Alacant
Universidad de Alicante

Social application with Pepper Robot

Bachelor's Degree in Computer Engineering

Autor
Daniel Martínez Andreu

Tutor/es
Miguel Ángel Cazorla Quevedo
Ciencia de la Computación e Inteligencia Artificial
Félix Escalona Moncholi
Ciencia de la Computación e Inteligencia Artificial

Bachelor's Degree in Computer Engineering

Escuela
Politécnica
Superior

Universitat d'Alacant
Universidad de Alicante

ALICANTE, July 2020

Preamble

The original intention of this project is to design a social application based on the face detection and recognition which has to be integrated into Pepper Robot as a kind of interactive app for people, especially for children and people with disabilities. The robot has to be able to recognize the person who is in front of it, performing predefined movements and holding a basic real-time conversation based on the children's preferences (favourite color, hobbies, etc.). All these preferences are established and registered during the phase of face recognition training. Machine Learning (ML) and Robotic Operating System (ROS) are the main applied technologies in this project.

Acknowledgements

I greatly appreciate the golden opportunity that my supervisors Miguel and Félix have given to me by allowing me to make this wonderful project. They have helped me in doing a lot of research and have provided me with all the necessary facilities I needed. I have come to know about so many new things, so I am really thankful for them. Working with you has really been a pleasure.

I also take this opportunity to express my gratitude to all members of my family and friends for their help and support. They have been essential in my work to move this matter forward.

Dedicated to my family and friends who have helped me a lot throughout this project.
I also place on record, my sense of gratitude to one and all,
who directly or indirectly, have let their hand in this work.

Ninguna investigación humana
puede ser llamada ciencia real
si no puede demostrarse
matemáticamente

Leonardo da Vinci.

Contents

List of Figures

List of Tables

Listings

1 Introduction

Nowadays, there are many social applications with a view to help people. Applications ranging from mobile applications to robots that incorporate this kind of technology in their software, as in many countries that consider robotics a fundamental part of their society. These countries are, especially Japan, China and Korea. The aim of this project is to develop a type of social application that is able to assist people, particularly kids and children with disabilities. To tackle the issues, it would be necessary to use some leading edge techniques like ML and Deep Learning (DL).

The main challenge of this project is to obtain a model which will be able to recognize a face with the highest reliability as possible,by experimenting and testing the results. This requires a testing and validation process, adapting and testing different neural networks in order to determine which is the best one to resolve, in the best way possible, the initial challenge. This is an issue that has received increasing attention in the past few years by the reason of the advances in DL and face recognition in images.

The dataset we pretend to train and validate the model with is the *CelebFaces Attributes Dataset (CelebA)* provided by (Liu et al., 2015). It contains more than 200.000 images of more than 10.000 different identities. The classifier will receive a given image with a face and it will have to determine whether that face belongs to someone of the trained faces or if it is a new one. In case of belonging to someone known, there will be some actions that the robot will perform with that person. By contrast, if the detected person is someone unknown, Pepper will ask for their name and it will proceed to start the training process to store the given face (the robot will request them to take some pictures of themselves).

Once the best model for face recognition has been established, the next step will be integrating everything developed in the Pepper Robot, which is shown in Figure 1.1. This robot has got two RGB cameras which will do the detection process. As mentioned before, Pepper has to recognize a known person or store the new face detected. The utilized tool to integrate the resulting model in the Pepper robot has been a framework oriented to be used in robots called ROS. Moreover, having a basic spoken interaction with Pepper is the final step of the project. Due to the fact that this is not the main purpose of the project, this conversation is made 'by hand', writing a JSON file with basic movements and words that will run when the identified person says any pre-established word or expression beforehand.

Due to the COVID-19 pandemic it has been necessary to slightly adapt the last part of this project, since testing the resulting application into the robot has not been possible. Its workplace is at the University and it was not possible to go there during this period of time. Therefore, the places where everything has been developed and tested are my computer and the Jackson multi-GPU server located in the RoViT department (see section 4.3.2).

Figure 1.1: This Pepper is located in the RoViT department at the University of Alicante and the access was via Secure SHell protocol (SSH)

2 Aims and motivation

My personal motivation when I chose this Bachelor's Thesis (BT) proposal was the aim of learning to develop current applications in DL related and applied to the real world. Face detection and recognition is a technology trend that more and more people use. There are a lot of applications, from a mobile phone in its unlock system, to the software in surveillance cameras; it is a topic that appears everywhere in our current society. Another important factor that was intriguing to me was running everything that I mentioned above on a robot, since I consider that it could have many useful applications. The first thing I thought about was developing an application focused on helping children with disabilities in their education and integration.

The main objectives I would like to achieve during the development of this project are:

- Learning and implementing DL algorithms, including the use of Keras and Tensorflow.
- Enhancing what I have learned this year about ROS, developing and running an application under this framework into Pepper robot.
- Developing a real-time algorithm based on face recognition.
- Choosing the architecture that best adapted to address the problem, making an experimental comparison between all the different approached networks and analyzing the advantages and disadvantages of each one.

3 State of the art

In this chapter it is explained an extended analysis about the research that the student has done in this BT. This section is useful to prove the relevance that it has and to compare the given solution from others; since it is possible to have different approaches to a solution. In particular, this will be addressed to the face detection and recognition algorithm.

3.1 Architectures

When obtained the data, by using the architectures that best adapt to address the problem, it is possible to solve effectively and establish conclusions in many computer vision tasks. In this particular case, we have to talk about Convolutional Neural Network (CNN) due to the fact that this kind of Neural Network (NN) is used to extract features from deep images. There are three main approaches: DenseNet, SVM and EfficientNet.

3.1.1 DenseNet

Keras (see section 4.1.2) provides DenseNet which connects each layer to every other layer in a feed-forward fashion (Huang et al., 2017). The connections between layers are given by $\frac{L(L+1)}{2}$ direct connections. This means that for each layer, the feature-maps of all preceding layers are used as inputs, and their own feature-maps are used as inputs into all subsequent layers. This kind of connections suppose several advantages, such as the alleviation of the vanishing-gradient problem, the strength feature propagation, the encouragement of the feature reuse and the reduction of the number of parameters. This architecture provides additional inputs from preceding layers to each layer and passes on its own feature-maps to all the subsequent layers, using concatenation as it is shown in Figure 3.1. For that reason, DenseNet NNs can be thinner and compact, which leads to a reduction in the number of parameters.

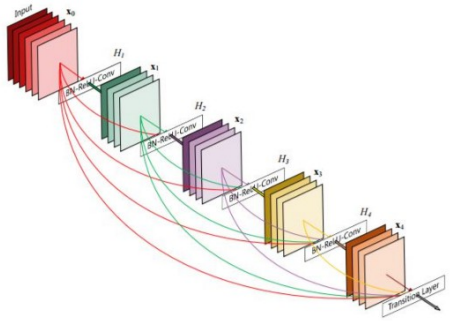

Figure 3.1: The DenseNet layers connection

This NN architecture could have many designs and variants depending on some factors like the number of parameters. Within these designs is DenseNet-121 which has been used in this project and explained in detail in section 5.2.1.2. The design of a DenseNet-121 is represented in figure 3.2.

Layers	Output Size	DenseNet-121
Convolution	112×112	7 x 7 conv, stride 2
Pooling	56×56	3 x 3 max pool, stride 2
Dense Block (1)	56×56	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$
Transition Layer (1)	56×56	1 x 1 conv
	28×28	2 x 2 average pool, stride 2
Dense Block (2)	28×28	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$
Transition Layer (2)	28×28	1 x 1 conv
	14×14	2 x 2 average pool, stride 2
Dense Block (3)	14×14	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 24$
Transition Layer (3)	14×14	1 x 1 conv
	7×7	2 x 2 average pool, stride 2
Dense Block (4)	7×7	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 16$
Classification Layer	1×1	7 x 7 global average pool
		1000D fully-connected, softmax

Figure 3.2: DenseNet-121 design

3.1.2 Support Vector Machine

Support Vector Machine (SVM) is an algorithm originally formulated to solve a classical two class pattern recognition problem (Bowyer & Phillips, 1998). With some adaptations, by modifying the interpolation of the output classifier obtained, it is possible to capture the dissimilarities between images of the same individual and dissimilarities between images of different people (this refers to the input). A SVM algorithm generates a decision surface separating the two classes. For face recognition, we re-interpret the decision surface to produce a similarity metric between two facial images.

The algorithm is a binary classification method that finds the optimal linear decision surface based on the concept of structural risk minimization. The decision surface is a weighted

combination of elements of the training set. These elements are called support vectors and characterize the boundary between the two classes. The input is a set x_i, y_i where x_i refers to the data (the image) and y_i is the training label, which says if the data corresponds to a given class or not (values 1 and -1). The output is a set of N support vectors (s_i), coefficient weights (α_i), class labels (y_i) and a constant term (b). The linear decision formula is given by the Equation 3.1.

$$w * z + b = 0 \ \textit{where} \ w = \sum_{i=1}^{N} \alpha_i * y_i * s_i \tag{3.1}$$

3.1.3 EfficientNet

It is a family of models which consists in scaling methods for CNN that uniformly scales all dimensions of depth, width and resolution, using a simple yet highly effective *compound coefficient* (Tan & Le, 2019). The effectiveness of this method is evident due to the accuracy and efficiency obtained which are much better than using a simple CNN.

Depending on the number of the scaling parameters, the final output can be given by different kind of levels which go from B0 to B7 moving from less to more complexity. Below this paragraph, it is Figure 3.3, which represents some performance results in which all EfficientNet models get their efficiency compared with the efficiency of some examples of CNN with similar accuracy.

Model	Top-1 Acc.	Top-5 Acc.	#Params	Ratio-to-EfficientNet	#FLOPS	Ratio-to-EfficientNet
EfficientNet-B0	**77.3%**	**93.5%**	**5.3M**	**1x**	**0.39B**	**1x**
ResNet-50 (He et al., 2016)	76.0%	93.0%	26M	4.9x	4.1B	11x
DenseNet-169 (Huang et al., 2017)	76.2%	93.2%	14M	2.6x	3.5B	8.9x
EfficientNet-B1	**79.2%**	**94.5%**	**7.8M**	**1x**	**0.70B**	**1x**
ResNet-152 (He et al., 2016)	77.8%	93.8%	60M	7.6x	11B	16x
DenseNet-264 (Huang et al., 2017)	77.9%	93.9%	34M	4.3x	6.0B	8.6x
Inception-v3 (Szegedy et al., 2016)	78.8%	94.4%	24M	3.0x	5.7B	8.1x
Xception (Chollet, 2017)	79.0%	94.5%	23M	3.0x	8.4B	12x
EfficientNet-B2	**80.3%**	**95.0%**	**9.2M**	**1x**	**1.0B**	**1x**
Inception-v4 (Szegedy et al., 2017)	80.0%	95.0%	48M	5.2x	13B	13x
Inception-resnet-v2 (Szegedy et al., 2017)	80.1%	95.1%	56M	6.1x	13B	13x
EfficientNet-B3	**81.7%**	**95.6%**	**12M**	**1x**	**1.8B**	**1x**
ResNeXt-101 (Xie et al., 2017)	80.9%	95.6%	84M	7.0x	32B	18x
PolyNet (Zhang et al., 2017)	81.3%	95.8%	92M	7.7x	35B	19x
EfficientNet-B4	**83.0%**	**96.3%**	**19M**	**1x**	**4.2B**	**1x**
SENet (Hu et al., 2018)	82.7%	96.2%	146M	7.7x	42B	10x
NASNet-A (Zoph et al., 2018)	82.7%	96.2%	89M	4.7x	24B	5.7x
AmoebaNet-A (Real et al., 2019)	82.8%	96.1%	87M	4.6x	23B	5.5x
PNASNet (Liu et al., 2018)	82.9%	96.2%	86M	4.5x	23B	6.0x
EfficientNet-B5	**83.7%**	**96.7%**	**30M**	**1x**	**9.9B**	**1x**
AmoebaNet-C (Cubuk et al., 2019)	83.5%	96.5%	155M	5.2x	41B	4.1x
EfficientNet-B6	**84.2%**	**96.8%**	**43M**	**1x**	**19B**	**1x**
EfficientNet-B7	**84.4%**	**97.1%**	**66M**	**1x**	**37B**	**1x**
GPipe (Huang et al., 2018)	84.3%	97.0%	557M	8.4x	-	-

Figure 3.3: EfficientNet Performance Results on ImageNet

3.2 Dataset

CelebFaces Attributes Dataset (most common known as *CelebA*), is a set of large-scale face attributes dataset with more than 200.000 celebrity images (Liu et al., 2015). Each of them has 40 binary attribute annotations in which are included, among other things, having (or not) eyeglasses, hair style, nose shape, etc. In Figure 3.4 there is an example of some of the attribute annotations that are included in the original dataset. The mentioned dataset is composed by the following:

- 10,177 identities.
- 202,599 face images.
- 5 landmark locations per image, including the mentioned 40 binary attributes.

The given dataset consists of a total of 202,599 images which have been used to train and validating the model. All of them have been first roughly aligned using similarity transformation according to the two eye locations and resized to 218*178. Furthermore, each image has been processed before the training process in order to delete unnecessary noise (e.g. blur the background), to zoom to enlarge the view of interest points in people faces, to adapt the light intensity and some other necessary adjustments like cropping and rotating the image.

Figure 3.4: Example of a piece of images of the *CelebA Dataset* to show how the annotations classify them by categories.

3.3 Learning face similarities

In this problem, learning face similarities is different from categorizing images into classes. What we pretend to do is not a classification task, instead, we want to learn a representa-

tion that can individually describe each picture, each input. The main objective is to find similarities between the given input images. For that reason, it would be necessary to find a representation capable of expressing a relationship between two comparable things as embedding vectors to represent relationships among people's face images. The vectors should have the following properties:

- Given the images X_1 and X_2, if they are of the same class, the distance between 2 output vectors has to be as small as possible.
- Otherwise, this distance has to be as large as we can make it.

Once having seen the vectors properties, let me introduce the concept of *contrastive loss*, which could be defined as a system composed by a pair of images (X_1, X_2) with the following properties:

- If X_1 is considered to be similar to X_2, the label value has to be 0.
- Otherwise, X_1 has to take the limit loss value established as a top margin, as it will be explained later on. In this case, this value is 1.

A parametric function G_w (NN) reduces the high dimensional input to a low dimensional representation mapping high-resolution inputs to low-resolution outputs as seen in Figure 3.5.

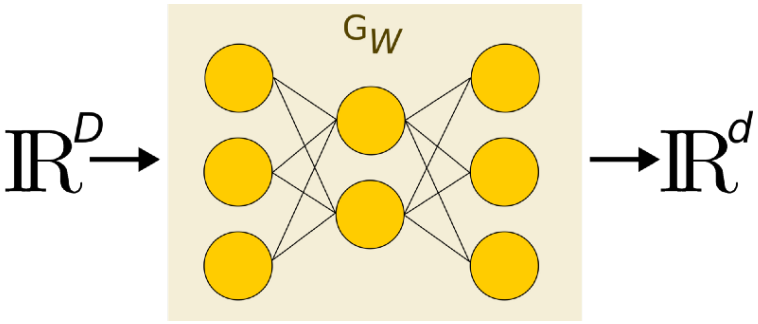

Figure 3.5: G_w as a parametric function. Reducing a high dimensional input to a low dimensional representation (D < d)

The mentioned margin (m) defines the radius around G_w and controls how the dissimilarity between images contributes to the total loss function. It only contributes to the loss function if the distance between a pair of images from different people is within the margin, which means $(m - D_w) > 0$. The final loss function and its implementation in TensorFlow are defined by the Equation 3.2.

$$(1 - Y)\frac{1}{2}(D_w)^2 + (Y)\frac{1}{2}[max(0, m - D_w)]^2 \tag{3.2}$$

On the other hand, when we talk about distance, we are referring to the Euclidean distance, which is given by the Equation 3.3.

$$D_w(\vec{X_1}, \vec{X_2}) = [G_w(\vec{X_1}) - G_w(\vec{X_2})]^2 \tag{3.3}$$

The whole process of learning face similarities could be summarized and represented as in Figure 3.6, where both representations are compared using the Euclidean distance.

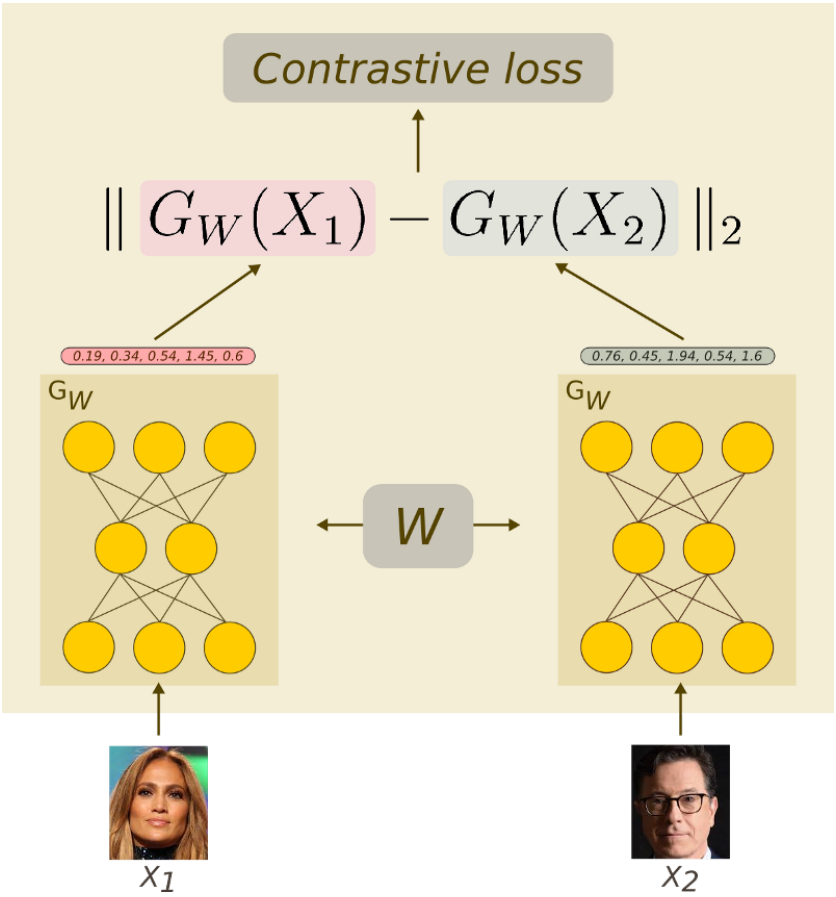

Figure 3.6: Learning face similarities algorithm

Applying everything mentioned above, we finally obtain a set of images similar to the one shown in Figure 3.7, which has to be analyzed by pair of rows.

There are 8 pictures per row and each one matches the photo that is located in the same position of the row below. The attribute 'sim' indicates the similarity value between the given images. The smaller is the value, the higher is the similarity. On the other hand, there are only 2 possible values for the attribute 'Label': '0' and '1'. The first value means that the pair of images are from the same person and the second one denotes we are comparing two different people.

Figure 3.7: Set of images of celebA dataset

4 Methodology

In this section, we will present an explicit explanation of the basic and the applied issues during the research.

4.1 Deep Learning Libraries and Frameworks

As a consequence of the interest that ML has awaken in many people during these past few years, a large number of open source frameworks specialized in ML and NN like TensorFlow (Abadi et al., 2016), Keras (Gulli & Pal, 2017), CNTK (Seide & Agarwal, 2016), PyTorch (Ketkar, 2017), Darknet (Redmon, 2013–2016) and so on, have been developed and distributed.

In our case, before starting with the development of this project, the first approach with NN, in order to familiarize with these kind of algorithms that are largely based on mathematics, was working with the framework called *You Only Look Once (YOLO)* to solve basic object recognition tasks. The aim of dealing with this framework was proving it and getting results in order to compare them (see Section 6) with the results obtained from some other algorithms in human face recognition tests. This is one of the purposes of our BT.

4.1.1 You Only Look Once

YOLO is an object detection framework which is related to computer vision and image processing (Huang et al., 2018). The most characteristic of this framework is its real time processing and execution from pre-trained model detectors (see Section 4.1.1.1). It is widely used in a large variety of detection and recognition tasks such as humans, buildings, cars, etc. These applications could be processed both as digital images and videos. The way that the framework deals with the object detection is as a regression problem to spatially separated bounding boxes and associated class probabilities (Redmon et al., 2016).

The YOLO concept, which is represented in Figure 4.1, is based on the way that the human visual system works. When given an image, it takes a classifier for the represented object and evaluates it at various locations and scales.

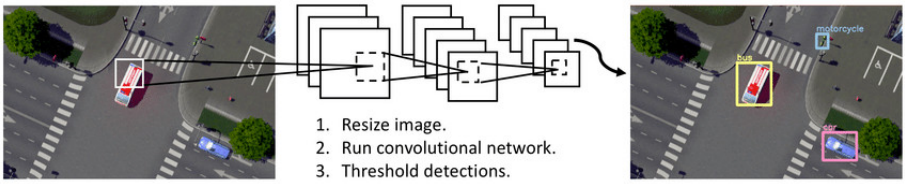

Figure 4.1: The YOLO Detection System. 1) Resize input image to the established size. 2) Run a single convolutional network. 3) Print resulting detections on the recognized objects of the image (Redmon et al., 2016)

It is said that YOLO is extremely fast, more than 1000x faster than Regions with Convolutional Neural Network (R-CNN) and 100x faster than Fast R-CNN (Redmon et al., 2016).

4.1.1.1 Darknet

Darknet is a framework to train NNs using the YOLO system from a pre-trained model (Chen et al., 2019). It prints out the detected object (as it could be seen in Figure 4.2) and how long it took to find it. The CPU takes around 6-12 seconds per given image without using the Graphics Processing Unit (GPU) version (this version makes the performance and execution process much faster). The great advantage of using YOLO, due to the use of a pre-trained model, is that given a new image, it is not necessary to retrain the model again.

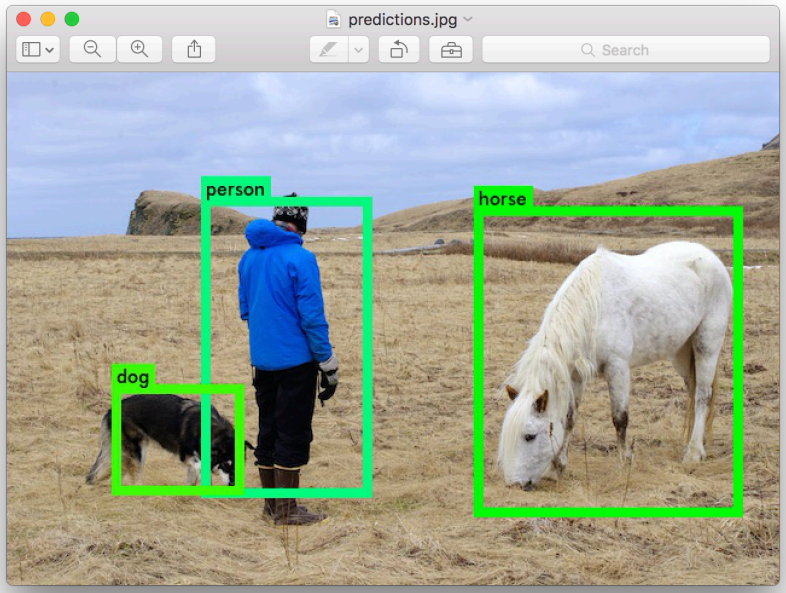

Figure 4.2: Darknet detector is able to identify and classify in different categories the objects located in the picture

4.1.2 Keras

This open source neural network library was designed to enable fast experimentation with deep NN. It was developed by François Chollet, a Google engineer who took part in a research project in Open-ended Neuro-Electronic Intelligent Robot Operating System (ONEIROS) in 2015 (Shatnawi et al., 2018). At present, Keras is supported by TensorFlow's core library since 2017. Many implementations of common issues when building NN such as layers, activation functions and optimizers are included in this framework so that working with image and text data is easier and also to simplify the coding necessary to write deep neural network code.

4.1.3 TensorFlow

TensorFlow is an open-source library for numerical computation which uses data flow graphs. It was developed by Google Brain in 2015 with the initial aim of encouraging research in ML (Abadi et al., 2016). The software developed with this library is easily deployed in a variety of hardware platforms like Central Processing Unit (CPU), GPU, mobile devices, etc. In this project in particular, the main learning objective is to acquire enough knowledge to be capable of developing NN models working with sequences of variable length, like the different characteristics present in the face recognition problem that differentiates one person from the others.

4.2 Software

Once having seen which libraries and frameworks we have used, for developing and executing code in a robot environment, it is necessary to use some more issues. In this section, there is a brief description of the software resources we needed to use and have played an important role in the project development.

4.2.1 Python

Python is an interpreted, object-oriented and high-level programming language which has dynamic semantics. This could be one of the reasons why Python is gradually becoming more popular as time goes by. Another feature is that Python does not have a compilation step, making incredibly fast the edit-test-debug cycle, so, in contrast with other programming languages, in Python there will never be a segmentation fault. In addition, Python has become the main programming language for ML due to its usefulness and ease not only for programmers, but also for mathematicians and scientists. It is not a coincidence that the majority of packages, particularly, ML libraries and frameworks, are only developed in Python.

4.2.2 Docker

The aim of working with Docker (Merkel, 2014) is solving incompatibilities with devices, due to the fact that Pepper Robot works under ROS Kinetic version which is only available on Ubuntu 16.04 and lower. The current version of that Linux distribution, however, is 18.04. For this reason, one of the options was refusing the new version and installing a version that might be unsupported in a short period of time. The other alternative was using some kind of virtualization that could allow different users to work with Pepper independently of the version or the Operating System installed in their devices. That is one of the purposes of Docker. Docker is a set of open-source platform that provides a service that produces some kind of OS-level virtualization to deliver software in packages called containers (Merkel, 2014). Those containers have many libraries and configuration files that can communicate with each other working in a way that is close to virtual machines. In order to understand this, it is interesting to check Figure 4.3 where are these concepts are explained graphically.

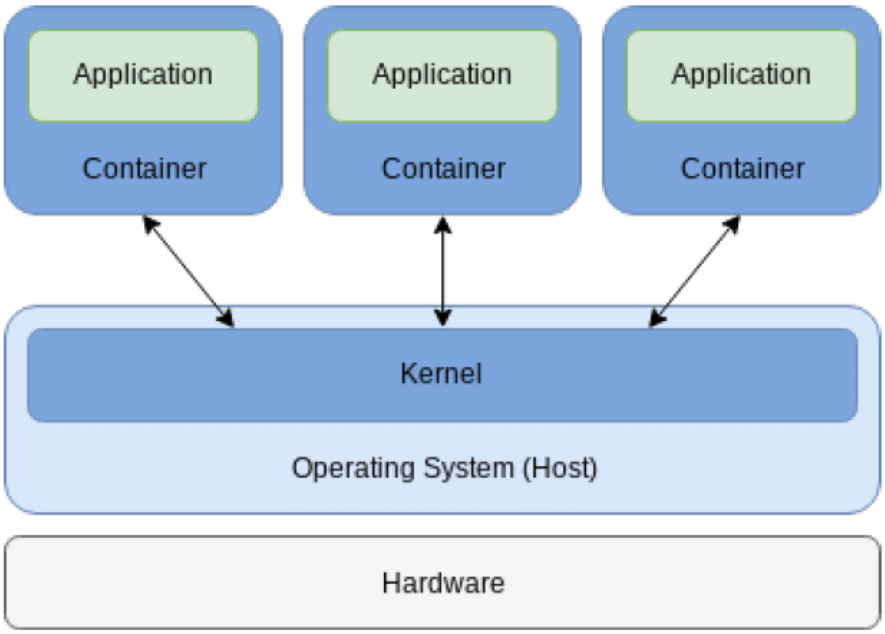

Figure 4.3: Docker container architecture (Scolati et al., 2019)

4.2.3 ROS

As mentioned before, Kinetic has been the used version, since Pepper Robot is below that version. ROS is a framework oriented to be used in robots that provides libraries and implements with some mechanisms of communications between programs, hardware abstraction, messages passing and so on. Those tools make possible to develop and create robotic applications (Anvari, 2011). The main concepts needed to be able to understand how ROS works are shown in Figure 4.4 and are the following:

- ***ROS Master***: It manages names and registration services to the nodes within a ROS system. Publishers and subscribers are monitored by the ROS Master to ensure associated topics, as well as services, are provided within the Robotic System.

- ***Topic***: Channel of communication between nodes (messages). Any node can be advertised or subscribed to any topic. The information is asynchronous.

- ***Package***: This is how its software is organized. Any package can contain a node, a library, a set of data or something that builds a module. Packages can be organized in stacks.

- ***Node***: Process that does some kind of computation, sharing information between them in order to create complex executions.

- ***Stack***: A set of nodes that all together provides some functionality.

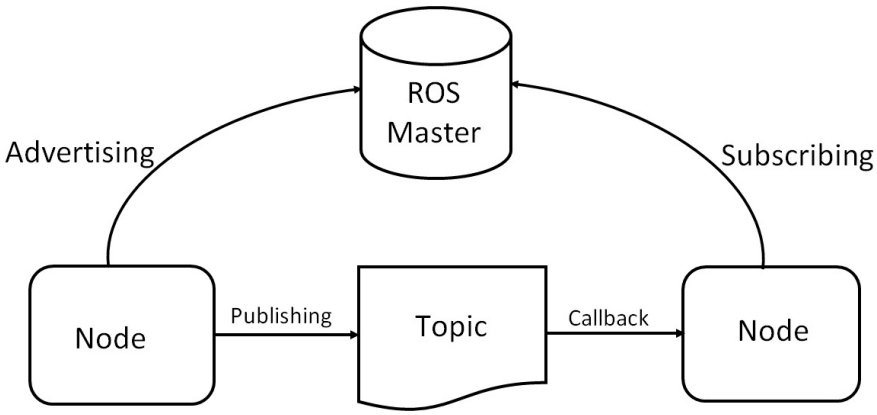

Figure 4.4: ROS system components

4.3 Hardware

When working with DL systems, it is necessary to make use of devices with high performance in GPU in order to allow us to make calculations and obtaining results in a permissible returning time. The quantity of data is usually enormous, so, if we do not have that kind of devices, the returning time could become a handicap for our project. That is precisely why the following hardware devices have been used:

4.3.1 Google Colaboratory

Google Colaboratory, also known as Google Colab, is a free Jupyter notebook environment provided by Google that runs entirely in the cloud (Bisong, 2019). It supports many ML libraries which can be easily loaded in a notebook without the necessity of installing anything. The integrated programming language is Python. The main feature that makes this environment so attractive to use is the GPU. It is totally free and it allows you to execute multiple data and also sharing your code with others.

The main tasks that Colab offers to any programmer are the following:

- Write and execute code in Python (any version of Python is available)
- Document the code
- Create/Upload/Share notebooks
- Import/Save notebook from/to Drive and GitHub
- Import external datasets (e.g. from Kaggle)
- Integrate many ML libraries and frameworks sucs as PyTorch, TensorFlow, Keras and OpenCV
- Free cloud service with free GPU

4.3.2 Jackson Server

Jackson is a multi-GPU server located in the RoViT department at the University of Alicante which is accessed via SSH. My research team has offered me the access to this powerful machine. Its features are:

- Operating System: Ubuntu 16.04
- RAM: 16GB DDR4
- CPU: Intel(R) Core(TM) i7-7700 CPU @ 3.60GHz
- GPU1: NVIDIA QUADRO P6600
- GPU2: NVIDIA RTX 2080ti

4.3.3 Pepper Robot

Pepper is a semi-humanoid robot manufactured by SoftBank Robotics designed with the ability to read emotions (Son, 2014). This robot is already in use in many countries leading to an occupation in various sectors like receptionist at several places such as hotels, banks and airports and it looks as shown in Figure 4.5.

The main specifications and the technical information from this robot are the following:

- Weight: 28 kg
- Height: 120 cm
- Width: 42.5 cm
- Battery: Lithium, 30.0Ah / 795Wh
- Autonomy: Until 12 hours
- Networking: Wi-Fi / Ethernet
- Speed: Up to 3 km/h
- Motor: 20
- Moving parts: Head (1), Shoulders (2), Elbows (2), Wrists (2), Fingers (10), Hip (1) and knees (1)
- Wheels: 3 (omnidirectionals)
- Degrees of motion: 360º
- Display: 10.1-inch touch display

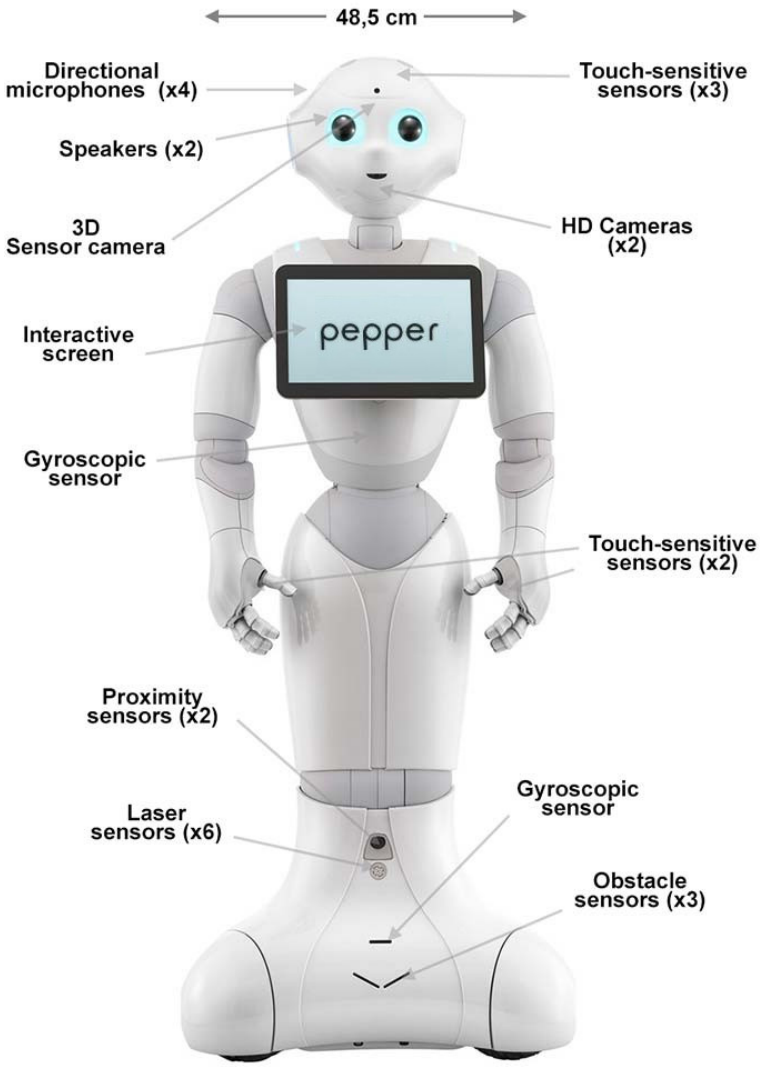

Figure 4.5: Technical information from Pepper Robot

5 Development

In this chapter it will be shown some of the developed algorithms which have been essential for the realization of the face recognition part of the project. Their theoretical foundations have been explained in section 3.

5.1 Image pre-processing

Before proceeding with the face detection and recognition, which are explained in Section 5.2, it would be necessary to do some modifications and transformations on the *CelebA* images. Exploring the mentioned dataset, we realized that not all the images had the same size and picture quality. Section 5.1.1 explains some techniques we did to soften these negative effects in the pictures.

5.1.1 Cropping and resizing

Although all the images in the *CelebA* dataset were supposed to be scaled to 218*178, there were some of them whose size did not fit in the mentioned measures. Moreover, we found them a bit big to apply DL on them. For that reason, after we ensured every picture had the same size, we scaled all pictures to 120*96 pixels.

Figure 5.1: A picture of Gaten Matarazzo from the *CelebA* dataset with the original picture (left) and the result of applying cropping and resizing (right)

Another difficulty found was the off-centred face position on images. Some pictures as the one shown on the left in image 5.1 were corrected. However, after applying these techniques, we found others that did not show the whole face, as shown in Figure 5.2. For that reason, as there were more well centred pictures than off-centred, we discarded to apply this correction, being aware that this problem could suppose bad results on images of this type.

Mention that all these modifications and transformations have been implemented using the Pillow Image processing library in Python. In particular, I used its Image class.

Figure 5.2: Bad results after applying cropping and resizing. The resulting picture of Peter Sellers is now a non-trainable image

5.2 Face detection and recognition

In this section it is described how the main part of the face detection and recognition algorithm was attained. We will start by introducing the NN architectures employed, as well as the developed modifications and adaptations (see section 5.2.1.2) and we will finish by explaining how it is possible to add new faces in the CelebA dataset (see Section 3.2) without the necessity of retraining the model and still having satisfactory results (see section 5.2.2).

5.2.1 Features extraction

5.2.1.1 Training set

We should proceed to split the data into two sets. One will be for training and the other one for testing the model.

Dataset	
Training	Testing
141,820	60,779

Table 5.1: Quantity of samples for each set of data

As seen in Table 5.1, the dataset is approximately split in 70% for training and the remaining 30% is for testing the model.

5.2.1.2 Models

There are many models specially designed to work with detection and recognition patterns with successful results. The most difficult part is to choose the best one and adapt the model in case it is required, to obtain the highest accuracy as possible. Notice that once having the best model with the highest results, the objective is to use this one as a pre-trained model in order to recognize new faces, making some modifications and without the necessity of retraining the model again (see section 5.2.2).

NN architectures used	
Name	**Number of parameters**
DenseNet-121	8,062,504
SVM	5,963,722
EfficientNetB0	5,330,571
EfficientNetB3	12,320,535

Table 5.2: Models for face recognition

To solve this problem, firstly some existing NN models were taken. In Table 5.2 are the different tested models with the parameters of each one enumerated. Then, it will be determined which is the one with the highest accuracy. Notice that the parameter of SVM (5,963,722) is given by the total number of hyperplanes (10,177), which are the same number as the quantity of identities than are in the *CelebA* dataset, multiplied by the dimension of each one (this dimension number is given by the dimension of the hyperplane features vector, which is 586).

Furthermore, one thing I would like to emphasize in this section is the weights parameter of each model. Using these architectures under Keras, the default weights taken are those from *ImageNet*. This is a dataset with more than 14,000,000 of images and 20,000 different categories so, it is one of the referents in image recognition because these weights are suitable for the majority of recognition problems. As an alternative to ImageNet, these models have also been tested with 'random' weights. What that means is that some different number of parameters without any criteria have been tested in order to find the most suitable one to address the problem. Finally, the used parameters are those that appear in Table 5.2.

5.2.1.3 KNN

In this part, a K-Nearest Neighbors (KNN) classifier was implemented in order to find similarities between two faces. To understand and imagine how it works, the algorithm could be summarized as follows:

Listing 5.1: KNN algorithm pseudocode

```
for each testingImage as a vector:
    for each trainingImage as a vector:
        EuclideanDistance between testingImage and trainingImage vectors
    end for
    Get the k−closest training vectors to testingImage vector
    trainingImage class = the most frequent class
end for
```

It has been used the 'EuclideanDistance' method to calculate the Euclidean distance between two vectors because this distance is the one which has the best results in these tasks. Then, this approach will determine if two given vectors (images) are similar or are from different classes.

5.2.1.4 SHAP

The SHapley Additive exPlanations (SHAP) method allows us to explain the obtained predictions by the model. With this approach, it is possible to detect in which places of the image the model is concentrated. When we began with the experimentation using random weights, our models predictions were quite variable. Whereas in some epochs the accuracy values were quite successful, the accuracy was so poor in others. For example, the oscillation between the maximum and the minimum accuracy during the training process using a DenseNet-121 was a value of 0.684, giving us reasons to believe that there would be something wrong in the model parameters. Then, we discovered that the model was concentrating more in the pictures background than in the given person. The way we solved this problem was using the trial and error method.

5.2.2 Transfer Learning

From the 'checkpoint' of a model, it is possible to add new faces to our dataset without training the whole model again. The way in which this process has been implemented is establishing the Keras parameter *include-top* as 'False' for not including the fully-connected layer at the top of the network. What we pretend to do is to apply this ML technique to take advantage of data from the first setting (celebA dataset) to extract information that may be useful to make predictions in the second setting (images from people who do not appear in the celebA dataset). In other words, the output of the model from a layer prior to the output layer of the model will be used as input to a new classifier model (see Figure 5.3).

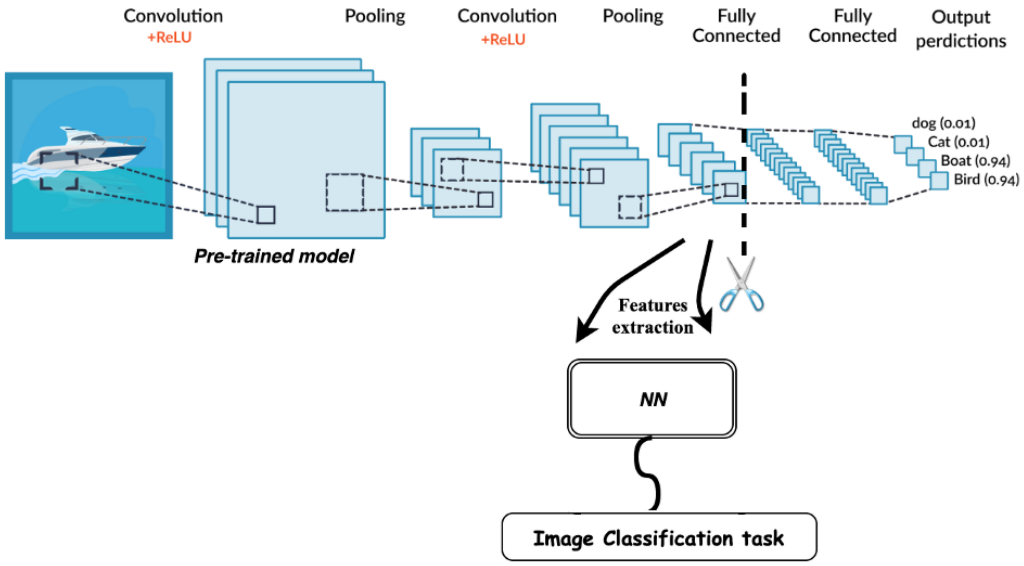

Figure 5.3: Transfer Learning

In Figure 5.3 there is an example of how *Transfer Learning* works. In order to clarify this concept, I would like to enumerate the steps I followed to apply this technique to my project and explain how new faces can be distinguished from known faces:

1. Take the last convolutional layer, the layer that is just before the classifier of the implemented NN.

2. Represent every image as a vector. Then, save all the resulting vectors in a way that we have one vector per image.

3. Apply KNN to the image vectors in order to find the highest number of correspondences between vectors as it is explained in Section 5.2.1.3.

4. Analyze the person's identity:

   a) If the given face is from a known person, move to the stage of the interaction between Pepper and him/her with the pre-established actions that were defined the first time the robot saw this person.

   b) Otherwise, add that face to the dataset and define some interactions for Pepper into a JSON file. Those actions will be performed by the robot next time he sees that person.

6 Experiments

In this part it will be analyzed and shown the different results obtained during the development of the project (see Section 5).

6.1 Features extraction

6.1.1 Models

Firstly, it will be evaluated each architecture considering the accuracy and loss results obtained. This will be useful to determine which one is the best model to use for *Transfer Learning* (see Section 6.2). For testing all models, it has been used the *CelebA* dataset (see Section 3.2) so, the results obtained are based on this dataset. The process to select the best model for this problem is enumerating each architecture with all its parameters and make a final comparison between them.

DenseNet-121: In Figure 6.1 are the results obtained using a DenseNet-121 with ImageNet weights and *Adam* Optimizer for optimizing its parameters. In addition, the initial model proposed by Keras has been slightly modified in some of its layers as follows:

- Growth rate (k): 32
- Embedding layer: 32D fully-connected

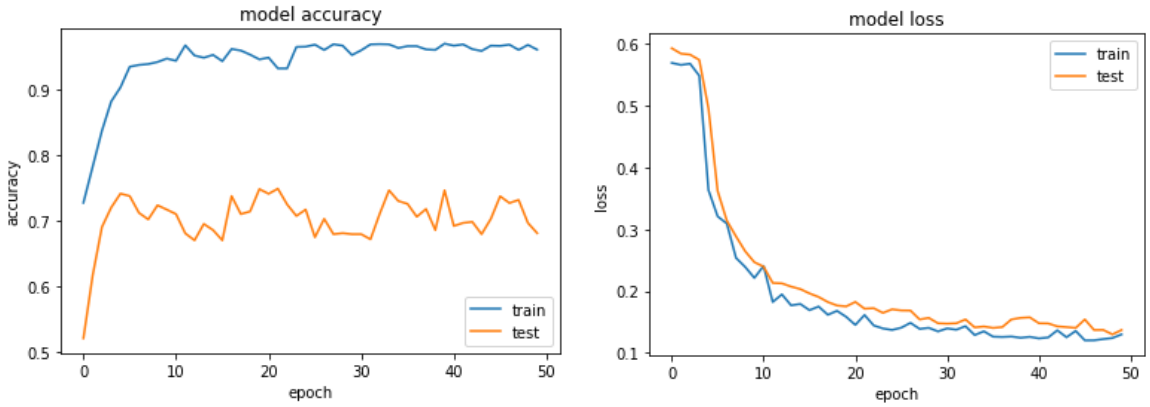

Figure 6.1: Accuracy and loss results using DenseNet-121 with k=32

This first experiment was the one with the lowest accuracy obtained, in spite of having improved the initial architecture. Table 6.1 shows the whole progress that the model experimented until the best results were obtained and represented in Figure 6.1.

DenseNet-121		
Weights	*Modifications*	*Accuracy*
ImageNet	Yes	**0.729**
ImageNet	No	0.697
Random	Yes	0.524
Random	No	0.521

Table 6.1: DenseNet-121 experimentation

SVM: For this this type of architecture, it has been used the 'sklearn.svm.SVC' class from the *Scikit-Learn* library. What we are trying to do is to separate images in hyperplanes. For that reason, we will have as many hyperplanes as identities. Notice that the results obtained have slightly been better than the results with DenseNet-121, reaching about **0.784** of accuracy. Even so, SVM is still far from being the best solution to address the problem. The margin of the hyperplanes between support vectors are not large enough to separate the dataset into classes as much as we hoped.

EfficientNetB0: With a lower number of parameters than in DenseNet, this model has obtained by far better results. In comparison with SVM, the execution time was quite longer and the results slightly better as well, arriving at **0.820** of accuracy as shows in Figure 6.2.

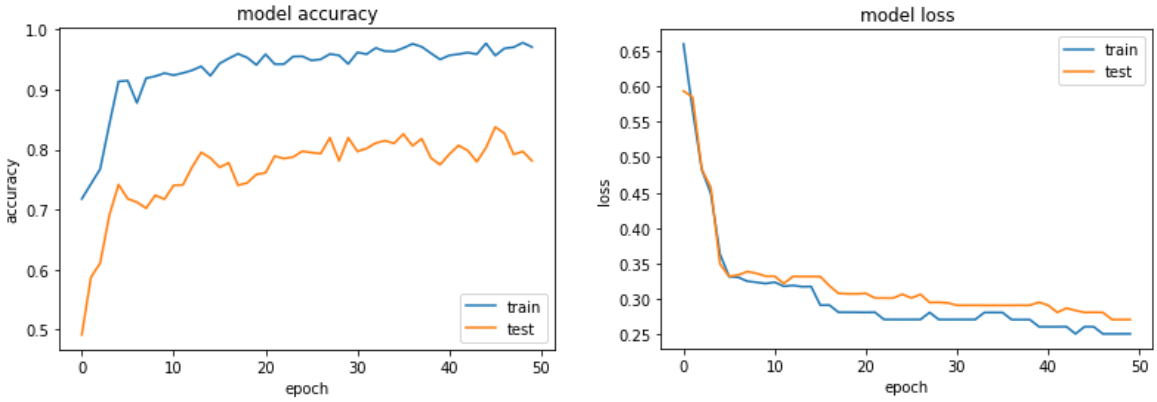

Figure 6.2: Accuracy and loss results using EfficientNetB0

In order to obtain the combination with the highest accuracy, it has been necessary to adapt and experiment with the parameters of the initial NN proposed by Keras. Table 6.2 represents this process of experimentation.

EfficientNetB0	
Weights	*Accuracy*
ImageNet	**0.820**
Random	0.766

Table 6.2: EfficientNetB0 experimentation

EfficientNetB3: As shown in Figure 6.2, the accuracy score for this model is the highest of the four models (**0.841**). For this reason, EfficientNetB3 will be the used architecture for *Transfer Learning* (see Section 6.2).

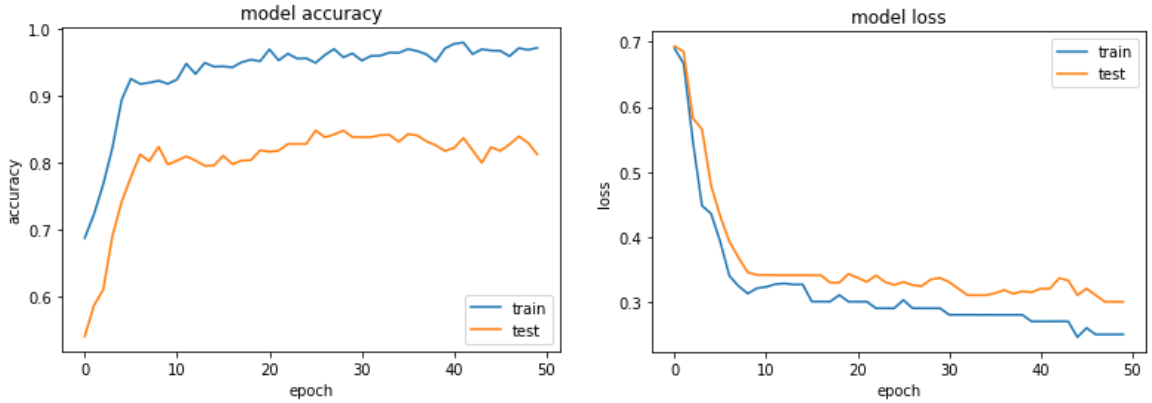

Figure 6.3: Accuracy and loss results using EfficientNetB3

On the other hand, its execution time has been longer than the others due to the number of parameters which the model has been trained. As it has done with each architecture, we have trained the model with many different configurations and weights which are shown in Table 6.3.

EfficientNetB3	
Weights	*Accuracy*
ImageNet	**0.841**
Random	0.711

Table 6.3: EfficientNetB3 experimentation

6.2 Transfer Learning

Once having the pre-trained EfficientNetB3 model as the best model to address the face recognition problem, we proceed to simulate a real situation, saving new faces and comparing and finding similarities between pictures from different people. For this experiment, in addition to myself, pictures of 6 more people have been used, where are included my sister and my girlfriend, who have given their consent to take part in this BT. The rest of the pictures are taken from famous actors and singers whose faces are widely known. The testing dataset is structured as follows:

- A new picture which appears in this experiment for the first time (the person who appears in the given image has been saved into the system previously taking some other photos of themselves).
- A list of five pictures ordered by decreasing similarity to the testing image. Above of

each one, the person label and the value of how similar they are with the tested person are indicated. Notice that once working this part in the robot, the label to guess who the person in front of Pepper is, will be the one with the highest similarity found, which, at the same time, will be the one with the lowest score.

Figure 6.4 shows the people who take part in this dataset. We are trying to simulate a real situation where new faces are added to the dataset little by little; therefore, this experiment is divided in three sets of data. The first one only compares famous people in labels 0, 1, 2 and 7. The second one only takes people with labels 3, 4 and 5. The other test compares all faces in Figure 6.4.

Figure 6.4: Dataset for face recognition testing with Transfer Learning

- **Experiment 1**: Labels 0, 1, 2 and 7. This experiment consists in four sets of famous people images. Once a person is detected by the robot, which we are simulating providing a new picture, the robot will have to find the identity of that person comparing and finding similarities in all the images stored in the system.
  - Label 0: As it could be seen in figure 6.5, in spite of appearing with a different hair color in some pictures, the model still recognized her as the same person. However, some similarities were also found with *Label 2*, since both women share the same lipstick color and have similar face features and hair color.
  - Label 1: Despite the fact of having a very particular face and finding perfectly his correspondence, there are many similarities with *Label 7*, such as their skin color or their goatee (see Figure 6.6).

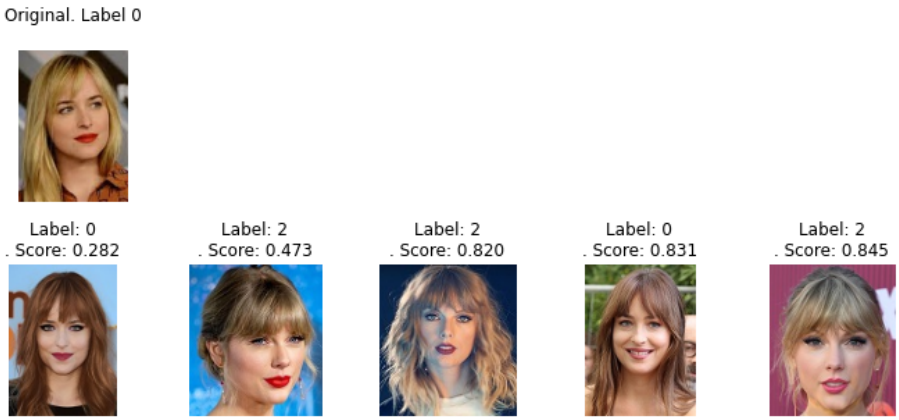

Figure 6.5: First set of data. Label 0

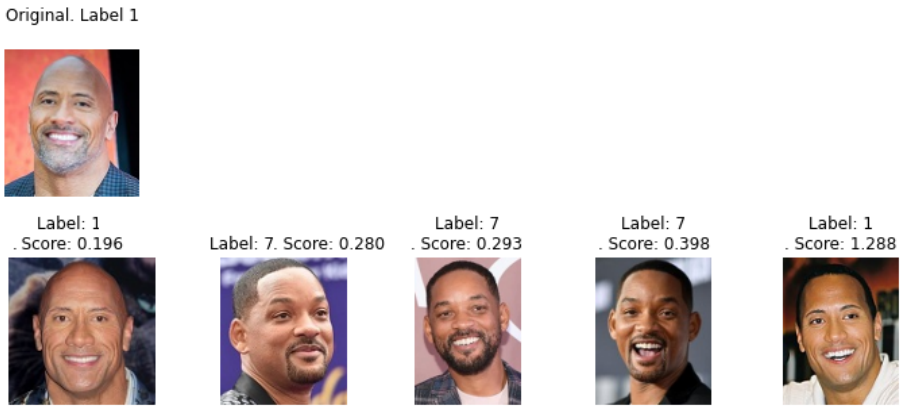

Figure 6.6: First set of data. Label 1

  - Label 2: The model classifies this label quite properly but, as mentioned in Figure 6.5, there are many correspondences with *Label 0*. For this reason, *Label 0* appears on the top three of the highest likeness (see Figure 6.7).
  - Label 7: The man in Figure 6.8 might be the person who differs the most from the rest, as it can be seen in the accuracy of the results obtained.

- **Experiment 2**: Labels 3, 4 and 5. This set of images contains pictures from people with a family connection. In contrast to the images used in *Experiment 1*, which were taken in different events with specialized cameras and were edited by professional photographers, these pictures were taken in different moments of our lives and the have been resized and cropped by myself for this experiment. For this reason, in this case, the difficulties the model has found in the process of analyzing the pictures has increased in comparison with *Experiment 1*.
  - Label 3: In Figure 6.9, the test has been successful, although it is true that the model has found many similarities between siblings.
  - Label 4: In Figure 6.10, the person with *Label 4* is successfully differentiated from

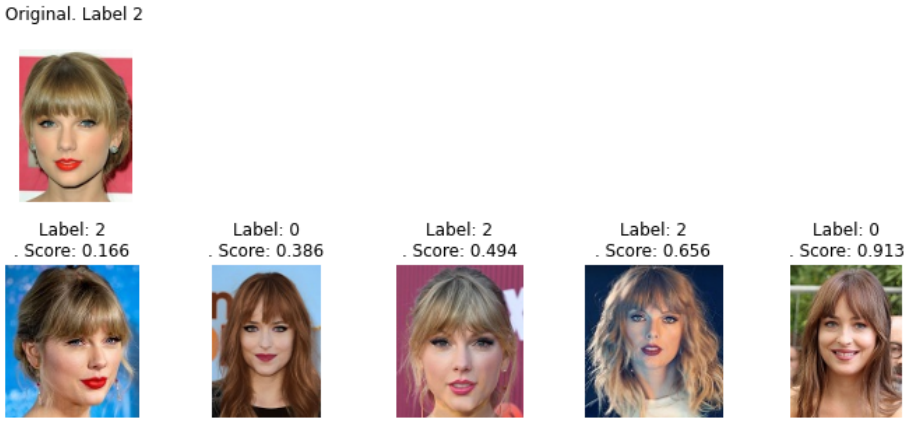

Figure 6.7: First set of data. Label 2

Figure 6.8: First set of data. Label 7

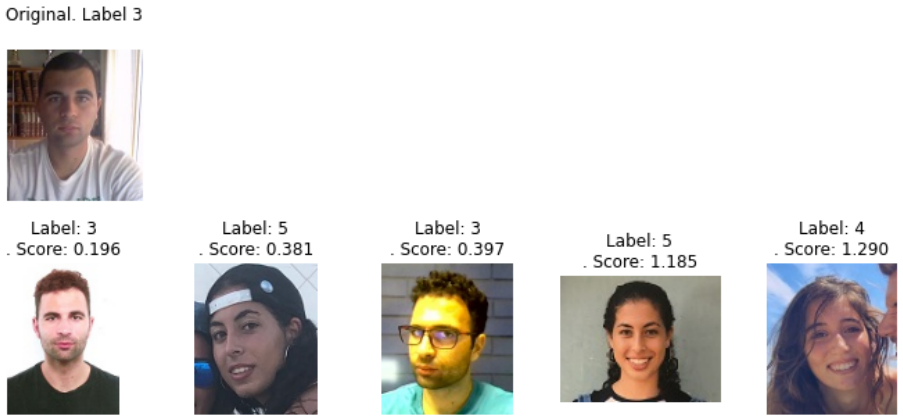

Figure 6.9: Second set of data. Label 3

the rest. The model is able to locate her when she appears with and without sunglasses.

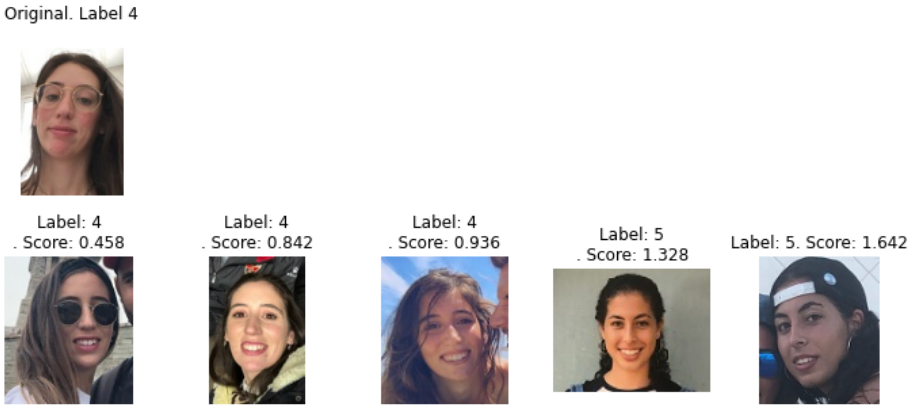

Figure 6.10: Second set of data. Label 4

– Label 5: In this case, in Figure 6.11, the model has identified the face properly and it has found the similarities mentioned in Figure 6.9.

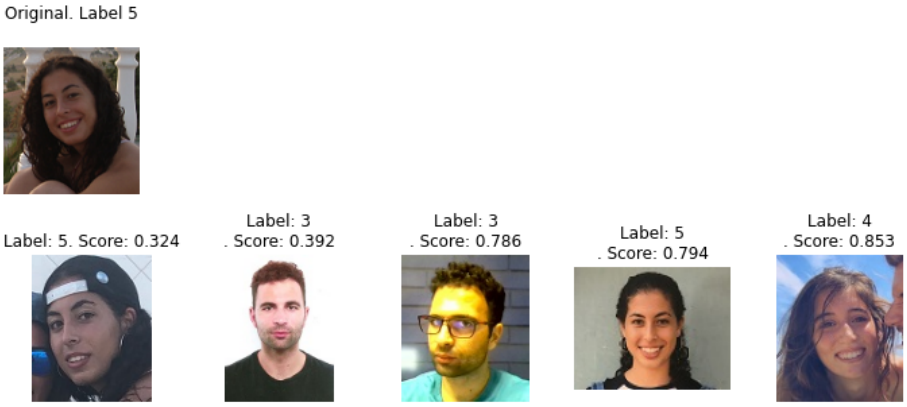

Figure 6.11: Second set of data. Label 5

- **Experiment 3**: In this experiment we are comparing new photos of each person that appears in Figure 6.4, with the rest of people. A fact that has been noticed is that as more faces from different people are added into the dataset, it gets more difficult to identify a person. Even so, Figures 6.12, 6.13 and 6.14 show satisfactory results.
  - Label 2: In Figure 6.12, it is shown a face expression of the person with *Label 2* quite different from the usual. This fact, however, is not a problem to distinguish her from the rest of people.
  - Label 4: In spite of wearing makeup, the model is able to identify her properly in some other pictures as shown in Figure 6.13.
  - Label 7: Even having a different goatee color and wearing a cap, as it is represented in Figure 6.14, this man is easy to distinguish from the rest.

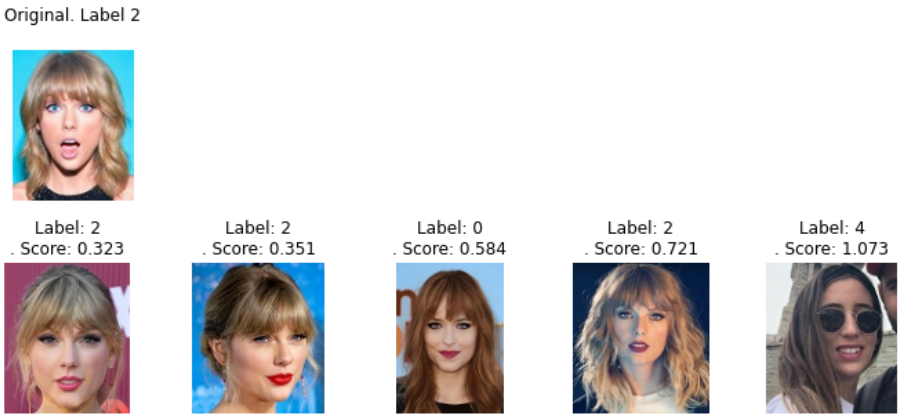

Figure 6.12: Third set of data. Result 1

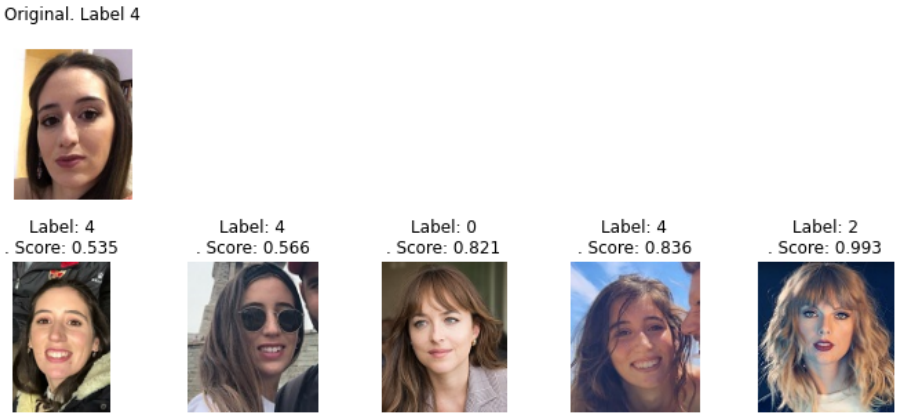

Figure 6.13: Third set of data. Result 2

Figure 6.14: Third set of data. Result 3

By contrast, there are many tests where the results obtained were not as accurate as expected. As it could be seen in Figures 6.15, 6.16 and 6.17, different people with similar facial expression and face features could lead to cause confusions in identifying people.

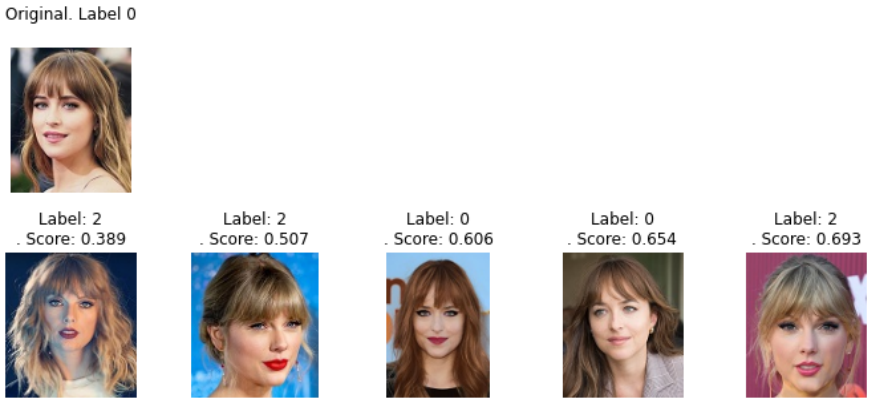

Figure 6.15: Face recognition error 1

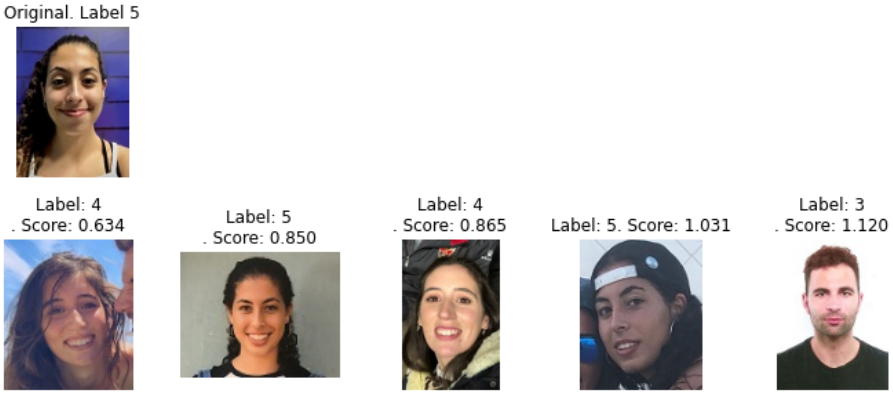

Figure 6.16: Face recognition error 2

Figure 6.17: Face recognition error 3

7 Conclusions

In this last chapter, we will present a deep reflection on the work done, the results obtained and we will also focus on everything learned.

First of all, I would like to emphasize the importance of ML and DL technologies. They are and will be essential to approach many complex problems that cannot be solved by humans or run under many machines either, due to the high amount of data that they require. Furthermore, something to point out is that before doing this research work, I could not imagine how a face recognition problem could be solved successfully, applying complex mathematical algorithms in neural form. Obviously, this requires investing a lot of time and effort in understanding all their basis and learning about the different architectures and their implementations.

Secondly, before starting with the DL experimentation and in order to find the highest accuracy in our results, an editing process was applied to every picture from the *CelebA* dataset, such as cropping and resizing, editing the brightness range and trying to reduce blurring on images. Nevertheless, in spite of applying those techniques, it was not possible to solve the problem that some faces of the dataset had. When someone appears facing sideways, their face features, which are the key to identify a person, cannot be clearly seen; therefore some results were not as accurate as expected. In fact, this is what caused most of the model errors. Moreover, the SHAP method was also applied on the given model after its training process, to check and detect in which places of the image the model was concentrated. This method was useful to find out that there were some errors with the NN parameters.

Finally, as seen in Section 6, some experiments and tests later, we have been able to determine which architecture was the best to address the face recognition problem. We concluded that it was the EfficientNetB3. The accuracy obtained (0.841) allowed us to implement and apply *Transfer Learning* on the model checkpoint to avoid re-training it continuously, which means identifying known or unknown people within an acceptable time range. Furthermore, in case we are introducing a new person, he/she will be saved into the system.

7.1 Future Work

Due to the time constraint imposed on this project and the unexpected pandemic of COVID-19, it was necessary to adapt or dispense with the implementation of some original ideas founded when this BT began. For those reasons, I would like to summarize some of my future aims:

- Improve face recognition results. In spite of having 0.841 of hit rate, this is not as successful as we desire. I am quite sure I could achieve better results increasing my knowledge on ML and DL, which would allow me to explore other approaches to rec-

ognize faces. This is one of the factors that encourages me to keep learning about the matter.

- Implement and test everything developed in this project into Pepper Robot which would be close to the real world. One of my aspirations when I chose the topic of this project were testing and witnessing the interaction between the robot and people.

In spite of everything mentioned above, I consider this project quite successful since I have achieved compliance with most of the proposed aims in Section 2. This project gives me personal satisfaction and an increasing desire to continue learning.

Bibliography

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., … others (2016). Tensorflow: A system for large-scale machine learning. In *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)* (pp. 265–283).

Anvari, M. (2011, July 12). *Multi-purpose robotic operating system and method.* Google Patents. (US Patent 7,979,157)

Bisong, E. (2019). Google colaboratory. In *Building machine learning and deep learning models on google cloud platform* (pp. 59–64). Springer.

Bowyer, K., & Phillips, P. J. (1998). *Empirical evaluation techniques in computer vision.* IEEE Computer Society Press.

Chen, R.-C., et al. (2019). Automatic license plate recognition via sliding-window darknet-yolo deep learning. *Image and Vision Computing*, *87*, 47–56.

Fairchild, C., & Harman, T. L. (2017). *Ros robotics by example: Learning to control wheeled, limbed, and flying robots using ros kinetic kame.* Packt Publishing Ltd.

Gulli, A., & Pal, S. (2017). *Deep learning with keras.* Packt Publishing Ltd.

Huang, G., Liu, Z., Van Der Maaten, L., & Weinberger, K. Q. (2017). Densely connected convolutional networks. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 4700–4708).

Huang, R., Pedoeem, J., & Chen, C. (2018). Yolo-lite: A real-time object detection algorithm optimized for non-gpu computers. In *2018 ieee international conference on big data (big data)* (p. 2503-2510).

Ketkar, N. (2017). Introduction to pytorch. In *Deep learning with python* (pp. 195–208). Springer.

Liu, Z., Luo, P., Wang, X., & Tang, X. (2015). Deep learning face attributes in the wild. In *Proceedings of international conference on computer vision (iccv).*

Merkel, D. (2014). Docker: lightweight linux containers for consistent development and deployment. *Linux journal*, *2014*(239), 2.

Redmon, J. (2013–2016). *Darknet: Open source neural networks in c.* `http://pjreddie.com/darknet/`.

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *Proceedings of the ieee conference on computer vision and pattern recognition* (pp. 779–788).

Scolati, R., Fronza, I., El Ioini, N., Samir, A., & Pahl, C. (2019, 05). A containerized big data streaming architecture for edge cloud computing on clustered single-board devices.. doi: 10.5220/0007695000680080

Seide, F., & Agarwal, A. (2016). Cntk: Microsoft's open-source deep-learning toolkit. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining* (pp. 2135–2135).

Shatnawi, A., Al-Bdour, G., Al-Qurran, R., & Al-Ayyoub, M. (2018). A comparative study of open source deep learning frameworks. In *2018 9th international conference on information and communication systems (icics)* (pp. 72–77).

Son, M. (2014). *A robot designed to interact with humans.* "https://www.softbankrobotics.com/emea/en/pepper". (Online; accessed 15 July 2020)

Tan, M., & Le, Q. V. (2019). Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*.

List of Acronyms

BT	Bachelor's Thesis.
CNN	Convolutional Neural Network.
CPU	Central Processing Unit.
CUDA	Compute Unfined Device Architecture.
DenseNet	Dense Convolutional Network.
DL	Deep Learning.
GPU	Graphics Processing Unit.
KNN	K-Nearest Neighbors.
ML	Machine Learning.
NN	Neural Network.
ONEIROS	Open-ended Neuro-Electronic Intelligent Robot Operating System.
R-CNN	Regions with Convolutional Neural Network.
ROS	Robotic Operating System.
SHAP	SHapley Additive exPlanations.
SSH	Secure SHell protocol.
SVM	Support Vector Machine.
TFG	Trabajo Final de Grado.
YOLO	You Only Look Once.