UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN

MÁSTER UNIVERSITARIO EN
INGENIERÍA DE TELECOMUNICACIÓN

TRABAJO FIN DE MASTER

Design of a facial emotion recognition system using Deep Learning techniques

IGNACIO RAMOS GARCIA

2019

TRABAJO DE FIN DE MASTER

Título:	Diseño de un sistema de reconocimiento de emociones faciales utilizando técnicas de aprendizaje profundo
Título (inglés):	Design of a facial emotion recognition system using Deep Learning techniques
Autor:	Ignacio Ramos García
Tutor:	Carlos Ángel Iglesias Fernández
Departamento:	Departamento de Ingeniería de Sistemas Telemáticos

MIEMBROS DEL TRIBUNAL CALIFICADOR

Presidente:	—
Vocal:	—
Secretario:	—
Suplente:	—

FECHA DE LECTURA:

CALIFICACIÓN:

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE
INGENIEROS DE TELECOMUNICACIÓN

Departamento de Ingeniería de Sistemas Telemáticos
Grupo de Sistemas Inteligentes

TRABAJO DE FIN DE MASTER

Design of a facial emotion recognition system using Deep Learning techniques

Enero 2019

Resumen

Este trabajo se enmarca en la línea de investigación de 'NLP y Análisis de Sentimientos' del Grupo de Sistemas Inteligentes de la ETSIT-UPM. El principal objetivo de este proyecto es el diseñar y desarrollar un sistema que sea capaz de reconocer emociones a través de las expresiones faciales.

El proyecto propuesto se divide en tres fases:

- Diseño y desarrollo de un sistema de reconocimiento de emociones faciales utilizando técnicas de aprendizaje profundo. Para ello se elaborarán diferentes sistemas con diferentes arquitectura para su posterior comparación y evaluación.

- Diseño y desarrollo de una aplicación local para testear el modelo entrenado anteriormente. Esta aplicación recoge información en tiempo real de la web cam, y utiliza técnicas de visión artificial para detectar caras presentes en cada frame del vídeo. Una vez se ha aislado la cara de resto de la imagen, se realiza un procesado y se hace uso del algoritmo para realizar la predicción de la emoción.

- Implementación de una aplicación en la nube a modo de prueba para detectar las emociones en tiempo real. En este caso se trata también de una aplicación que recoge información de la webcam, pero con la peculiaridad de poder ser desplegado en cualquier plataforma o servidor web, puesto que se ha implementado con NodeJS, lenguaje ampliamente utilizado en proyectos web.

Palabras clave: Aprendizaje profundo, Redes Neuronales, visión artificial, predicción de emociones, aplicación web

Abstract

This project is within the researching line of 'NLP and Sentiment Analysis' of the Intelligent System group of the ETSIT-UPM. The main objective is to design and develop a system capable of recognizing emotions from facial expressions.

The project is divided into three phases:

- Design and development of a facial emotion recognition system using Deep Learning techiques. To do so, various models are designed with different architectures, so that a comparison can be done in order to select the one with the best performance.
- Design and development of a local app in order to test the model previously trained in a practical way. this app gathers information from the webcam in real time, and performs computer vision techniques in order to detect different faces presented in each frame of the video. Once the face has been isolated from the rest of the image, a preprocessing is done and then the model is used to recognized the emotion.
- Implementation of a cloud-deployable app to detect emotions in real time. As before, this app also gathers information served by the web cam, but with the aim of being deployed on the cloud, since it has been developed using NodeJS, a language widely used in web projects.

Keywords: Deep Learning, Neural Networks, Computer Vision, Emotion prediction, web app

Agradecimientos

Tras la finalización de este Trabajo Fin de Máster, pongo fin a una etapa muy importante en mi vida, en la que tengo que agradecer a numerosas personas la ayuda recibida.

En primer lugar a Carlos A. Iglesias por ofrecerme la posibilidad de realizar este interesante trabajo de una manera muy flexible, aportándome un rápido feedback en cada momento.

Especial mención a mi familia (mis padres y Helen), los cuales han estado presentes en los buenos, pero sobre todo en los malos momentos de estos seis años. Me gustaría mencionar a mi abuelo Ángel, el cual siempre ha sido una fuente de inspiración para mí.

Finalmente, agradecer a Laura, la cual también me ha apoyado en todos los momentos.

¡Muchas gracias!

Contents

List of Figures

CHAPTER 1

Introduction

This chapter is going to introduce the context of the project, including a brief overview of all the different parts that will be discussed in the project. It will also break down a series of objectives to be carried out during the realization of the project. Moreover, it will introduce the structure of the document with an overview of each chapter.

1.1 Context

Technology, nowadays is one of the pillars of industries, in which Digital Transformation plays a really important role. This transformation based on technology is leading to an increasing generation of data. Moreover, it can be also seen in society as something indispensable and necessary, such as the increasing tendency on the use and dependency of mobile phones. Thereby, this massive data must be stored somewhere, and increasingly more companies are processing this data to obtain further information about their clients or the company itself. The extra value obtained from from huge amount of data is called Big Data[1]. In addition, Big Data techniques are really useful because they help companies to find answers of questions that they didn't even know that they had.

Together with Big Data, another technology that is becoming more and more popular is Artificial Intelligence (AI)[2]. This term, englobes many techniques and algorithms that have in common one thing, the cognitive part. This means that a computer is able to learn from data in order to solve certain problems of many types. This solving in some cases is also known as predicting. Therefore the principal goal of AI is to create machines that can solve problems automatically and independently.

Within AI techniques family, there is a term called **Deep Learning**[3], which is a Machine Learning technique based on **Artificial Neural Network (ANN)**. ANNs are supervised approaches in which a model is trained with different examples [20]. This means that the algorithm learns from data. ANNs intend to simulate human being's brain functionality. Currently, they are a tendency because they are able to solve many ordinary mathematical problems such as classification or clustering. After training these models, a testing phase takes place in order to evaluate the performance, and assure good results will be obtained when predicting on new data. Meaning by new data, the one that has not been used for training, and thus, the algorithm that sees for the first time.

Regarding this technological revolution, Robotics have turned into a very popular engineering tool, owing to thanks to this, many type of processes have been automated and accelerated. Furthermore, as time goes by, is more typical to see, for example, smartphones with personal assistants, which is nothing but a robot that can learn from your habits and give you some advices. Therefore, **Emotion Analysis** is increasing popularity in order to build services more personalized to each user. Emotion analysis done by a computer is embedded under the line of research called Affective Computing [35], which is a domain

[1] https://www.investopedia.com/terms/b/big-data.asp

[2] https://www.sas.com/en_us/insights/analytics/what-is-artificial-intelligence.html

[3] https://www.investopedia.com/terms/d/deep-learning.asp

that focuses on user emotions while he interacts with computers. As mentioned before, one of the principal purposes of this study is the computer's ability of recognizing, interpreting and simulating humans state. Picard, the author of Affective Computing [35], believes that if we want to achieve a natural interaction with computer, we must provide them with the tools needed to understand us. There are many different ways to retrieve an emotion from a person, such as using speech recognition or face recognition. This work will be focused on the second technique. To implement this, a Deep Learning model will be built and trained, for the purpose to perform emotion predictions from facial expressions. In addition to this model, an application using video as an input will be developed.

Nevertheless, any of the technologies mentioned above wouldn't be possible without an infrastructure capable of processing the data with a certain speed. In this moment, it's where the term **Cloud** (Internet) comes up, and with this distributed applications, allowing users from different parts of the world to make use of these applications. Hence, Machine Learning algorithms are normally deployed in the Cloud as a web service. Thus, as a second part of the project, the application will be deployed in the cloud using Websockets [50].

1.2 Project goals

In relation to the previous section , this project pursues several objectives, which are related to the different technologies/trends already mentioned.

First of all, the implementation of a model capable of predicting emotions from facial expressions. To accomplish so, different state-of-the-art techniques will be compared in order to retrieve the one with the best performance. All these techniques are based on the term Deep Learning, which is nothing but a neural network with several layers. Therefore, different combinations of layers will be tested so that the best architecture will be used in further sections of the project.

Secondly, in order to test the model in a real-world scenario, a python app is implemented. This app retrieves real-time images from a laptop webcam, and performs a face detection follows by the prediction itself. Thus, in this part different techniques will be handled in parallel, such as, image processing and computer vision [45].

Finally, as the app developed in python is runnable only in a laptop, the need of deploy this app in the cloud come up. The main reason of that, is to make it accessible from any type of device. To do so, NodeJS [43] and Docker containers [33] are used.

1.3 Structure of this document

This section intends to give an overview of the structure of this document, in order to better understand each section.

Chapter 2 a state of the art of Sentiment and Emotion Analysis is detailed. Thus, it is giving the reader the necessary knowledge to fully understand the whole project, and thus further sections. Therefore, different concepts will be explained in a high level, without going into many details. However, some mathematical formulations will be added to support the explanations, so some mathematical previous knowledge is needed. Therefore, It provides an in-depth summary about some basic types of networks and all the parameters that are involved in their development.

Chapter 3 provides an overview of all the technologies that will be used on this project. Moreover, it will describe some resources that will be important. It covers all the technologies needed, from the programming language used to web tools used to train models,

Chapter 4 shows the overall methodology followed during the project.

Chapter 5 describes all the approaches carried out with the Neural Networks and a explanation of the advanced techniques that have been used for developing the classifiers. Moreover, a comparison of the different architectures implemented is done, in order to get the algorithm with the best performance

In chapter 6, a desktop app in python is explained. The main goal of this app, developed in Python, is to test the best model chose in previous chapter, with real-world data. In other words, implement an app which allows to test the model using real-time information captured by the webcam.

In chapter 7 a web app is implemented and described. The main objective of this app is to deploy the facial emotion recognition model in the cloud, so that it can be accessible from every device.

Chapter 8 summarizes the conclusions of each one of the phases.

1.4 Methodology

In this chapter the methodology followed to accomplish the project is presented. The order of the chapters corresponds to the line time and steps of the project. Thus, there are three main phases: model comparison, elaboration of a local application and the development of a similar cloud-deployable app.

Figure 1.1 represents the process followed. The first step is the design and implementation of a **Deep Learning model**. In this step, four architectures have been developed in order to try different combinations and achieve the best accuracy. Thus, after the design and training of these four architecture, a model comparison analysis is done, in order to use the model with best characteristics in following steps. Concerning the model implementation phase, some cutting-edge technologies, such as Collaboratory, will be used. Furthermore, some advances techniques while training will be implemented.

Once the best model has been selected, the next step is to develop a practical application so that it can be tried with real-time information. Hence, a **real-time application** to get users' emotions from faces expressions will be developed. The language used to develop it is **Python**, so that the integration of the model with the application itself can be eased. Moreover, in order to do some preprocessing in the image and to capture webcam information, the library **OpenCV** [7] is used. The app is developed, and afterwards, submitted to a set of tests. This tests are entirely carried out by the developer, as it was the final user.

Finally, the next goal is to deploy this application on the **cloud**. To do so, all the application will be entirely developed using **NodeJS**, and Websockets to perform client-server communication. The reason why this has been used, is to create a real-time communication. First of all the app is implemented followed by some user testing. As before, these tests are entirely carried out by the developer, as it was the final user. After this, the app is dockerized for deployment.

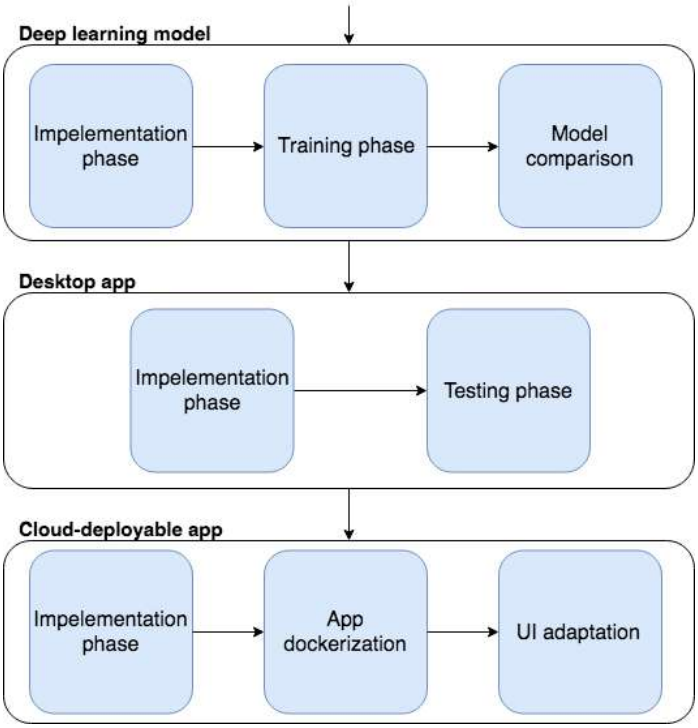

Figure 1.1: Methodology and project phases

CHAPTER 2

State of Art

This chapter provides the reader with the necessary knowledge to fully understand the whole project, and thus further sections. Therefore, different concepts are explained in a high level, without going into many details. However, some mathematical formulations are added to support the explanations, where previous mathematical knowledge is needed.

2.1 Emotion Analysis

Human beings emotions play an important role in communications between each other, because they help us to understand the intentions of the others. The face is an opened window of our internal emotion, that means that through our face expressions the emotions can be inferred. This is called non-verbal expression.

Concerning that most important emotions can describe people behaviors, there is no universal agreement by the researches and scientists. However the work done by Paul Ekman [16] in the 1960s has still a solid acceptance among this field. The model proposed in that work considers that there are six basic emotions that can provide universally clear reactions: Anger, Disgust,Fear, Happiness, Sadness and Surprise. In addition in that research there were given the facial expressions related to each emotion. These gestures are summarized in table 2.1.

Emotion	**Facial expressions**
Happiness	characterized by raising the extremes of the mouth and tighten the eyelids
Surprise	characterized by arching the eyebrows, open wide the eyes with a stronger white, and the jaw drops a bit
Anger	characterized by lower the eyebrows, press the lips and eyes bulging
Sadness	characterized by moving down the extremes of the mouth, the eyebrows descending to the inner corners and the eyelids drooping
Disgust	characterized by raising the upper lip, wrinkle the nose bridge, and raising the cheeks
Fear	characterized raising the upper lip, opening the eyes and stretching the lips horizontally

Table 2.1: Facial expressions that characterize each emotion

After the first research done, Ekman added new emotions such as contempt, which nowadays is grouped with the other ones. Moreover, he did further studies and he tried to give new groups of basic emotions, such as guilt, shame, interest, embarrassment, awe

and excitement. Finally in [17] he suggested the presence of sixteen positive, or enjoyable emotions each different from another.

In [15], Ekam exposed the concept of **micro expressions**. Micro expressions are very speedy (in a very small fraction of second) involuntary facial expressions that denote a person's true emotion made in certain circumstances. Micro expressions are really important because nobody can hide them, as they are involuntary, and everybody makes them. They can be a certain key to detect deception or awareness.

Learning these micro expressions can add several insights which can help us to better understand other people. For instance, when someone is trying to conceals an emotion, certain facial expressions are flashed in an unknowingly way. Hence, to support the idea mentioned above, learning them can enhance relationships because you can retrieve the emotion of other people easily and help you to increase the connection with the other, and thus, it helps you to develop your capacity of empathy.

2.2 Computer vision

Computer vision is the research field that aims at finding techniques to simulate the human vision in a computer [45]. This means that enabling a computer perceiving the environment as humans do. It is not only focused on this simulation, but also processes and provides useful results based on observation.

As humans, we perceive the three-dimensional structure of the world around us with apparent ease. Therefore, computer vision tries to develop mathematical techniques for recovering the three-dimensional shape and appearance of objects in imagery.

The reason of implementing a computer vision technique in this project, is that this field usually goes together with artificial intelligence, and one of the basics of AI are the neural nets. For example, in a cutting-edge technology such as robotics, first of all a vision algorithm is implemented to see what its surrounding it and then applies an algorithm to perform an appropriate analysis of it to better understand it, and to overcome different problems.

2.2.1 Face recognition

Face recognition is an object detection approach in which a human face must be detected. Obtect detection is a field that nowadays is really mature, in which there are plenty of robust algorithms that perform with high accuracy this object-detection. Regarding face detection, according to the author Yang[31], there are four methods: knowledge-based methods, feature invariant approach, template matching method and appearance based methods.

Regarding the first technique, the way to accomplish the detection is that the face is determined by defined rules, which describe a typical face. Usually these rules describe relations among face features. Yang and Huang [51] used a hierarchical knowledge-based method to detect faces, in which there were three levels of rules.

The second set of techniques aims at retrieving several structural features that exist even when external factors affects the image, such as the light or the pose(head rotation). Such features can be facial features (eyes, nose, etc.), facial texture or skin color. Kin Choong Yow et al. [53] proposed an algorithm that detects the feature or key points using spatial filtering techniques.

The third group of techniques compute a correlation of the input image with respect to several features or standard patterns previously stored which describe the face as a whole.According to Caifeng Shan et al. [18] face is represented based on statistical local features, local binary patterns(LBP) for person independent expression recognition. LBP operator takes a local neighborhood around each pixel, thresholds the pixels of the neighborhood at the value of the central pixel and uses the resulting binary valued image patch as a local image descriptor.

Finally in appearance-based methods the models are learned from a set of training images and the features must be learning during the training process.

In this project the technique that will be used is the one proposed by Paul Viola and Michael Jones in their paper [49]. In this paper they propose a general object detection based on Haar feature-based cascade classifiers. The Haar Features are the ones that all human faces have in common such as the eye region is darker than the upper-cheeks. Nevertheless, the comparison done at one time with all these features is computational inefficient. Hence, the Cascade term comes up, which aims at concatenating different stages of classifiers and apply one-by-one in the input image. Finally the outcome is a detector with more than 6000 features with 38 stages.

2.2.2 Facial Emotion Recognition

The goal of human emotion recognition is to automatically classify user's temporal emotional state basing on some input data. Automatic facial emotion recognition (FER) is an important field in Computer Vision, and the techniques to accomplish that has increased over the last few years with the increase of artificial intelligent techniques.

As a conventional facial emotion recognition, we can divide the techniques into three groups [30] based on the extracted features: geometric features, appearance features and hybrid. Geometric-based technique finds a set of representative features of geometric form to represent an object by collecting geometric features from images. In this technique, the

shape of the face and its components is described and emotional data is extracted from motion of facial muscles. Otherwise, appearance based extraction describes the texture of the face caused by expression, and extracts emotional data from skin changes. For hybrid features, some approaches have combined geometric and appearance features to complement the weaknesses of the two approaches and provide even better results in certain cases. The approaches mentioned above, especially, the apearance-based ones make use of Machine Learning techniques to perform classification tasks such as Support Vector Machines (SVM) [12]. Machine Learning tools are nothing but mathematical algorithms that use data to learn from and to predict new data from the input one. They are characterized, by being different as traditional static algorithms used in computations. Modern approaches make use of cutting-edge technologies such as the ones named as **Deep Learning** techniques. This set of techniques are commonly based on ANNs on conform the basis of the Artificial Intelligence. Thereby, Deep Learning tries to overcome new challenges of tedious feature engineering task. In the following chapters these techniques are explained in more details.

2.3 Neural Networks

Neural networks [20] are modeled after biological neural networks and attempt to allow computers to learn in a similar manner to humans: reinforcement learning.

The human brain has interconnected neurons with dendrites that receive inputs, and then based on those inputs, produce an electrical signal output through the axon. There are problems that are difficult for humans but easy for computers (e.g. calculating large arithmetic problems). Then there are problems easy for humans, but difficult for computers (e.g. recognizing a picture of a person from the side). Therefore, Neural Networks attempt to solve problems that would normally be easy for humans but hard for computers.

2.3.1 Single perceptron

Before going into details, the most important model of Neural Nets is explained: the perceptron. It was the first algorithmically described neural network and his inventor was Rosenblatt who made a first approach of it in a paper published in 1958 [39].

The perceptron is the simplest form of a neural network used for the binary classification of patterns said to be linearly separable (i.e., patterns can be easily separate by an hyperplane). Binary classifiers are functions that can decide whether an input, represented by a vector of numbers, belongs to some specific class or not. Basically, it consists of a single neuron with adjustable weights and bias. Rosenblatt's perceptron is built around a nonlinear neuron, namely, the McCulloch &Pitts model of a neuron. The summing node of the neural model computes a linear combination of the inputs applied to its weights, as well

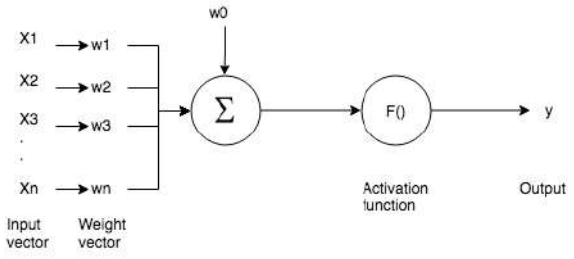

Figure 2.1: Signal-flow graph of the perceptron

as incorporates an externally applied bias. The resulting sum, that is, the induced local field, is applied to an activation function. This is depicted in Figure 2.1.

The activation unit is provided by eq. 2.1, where we redefine $x = [1, X_1, ..., X_p]^T$ as an augmented input vector with $x_0 = 1$ in order to include the bias term (w_0) in the weight vector: $w = [w_0, w_1, ..., w_p]^T$.

$$y = F[\sum_{i=1}^{p} w_i x_i + w_0] = F[\sum_{i=1}^{p} w_i x_i] = F[W^T X] \tag{2.1}$$

The connection weights are adjusted globally in order to optimize the performances of the network through a cost function.

2.3.2 Activation functions

The activation function takes the decision of whether or not to pass the signal and transform that signal to something with bounds. This subsection explains the different activation functions that are related to the development of the deep learning analysis.

2.3.2.1 Sigmoid function

It consists in a special case of the Logistic Activation Function. Basically, it transforms the input in an output with bounds between 0 and 1. The mathematical equation is the following [20]:

$$S(x) = \frac{1}{1 + exp[-(W^T X)]} \tag{2.2}$$

Sigmoids are one of the most widely used activations functions. However, they have vanishing gradients, which could affect at the learning process of the neural network. The representation of a sigmoid function can be appreciated in Figure 2.2. The order axis (Y) tends to respond very less to changes in the abscissas axis (X). This makes the neuron to enter in a saturated regime, making the network to refuse learning further or slowing the

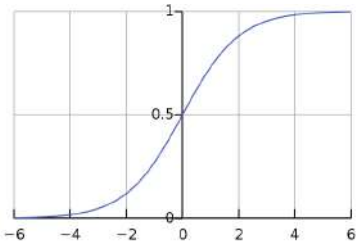

Figure 2.2: Sigmoid

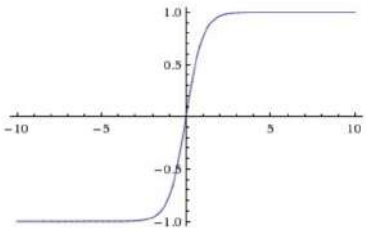

Figure 2.3: Tanh

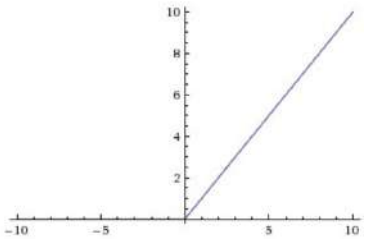

Figure 2.4: ReLu

process. Despite of this problem, there are ways to work around this problem and sigmoid is useful in classification tasks.

2.3.2.2 Tanh function

Tanh function is also known as the hyperbolic tangent activation function [20]. It has two main differences with the sigmoid function, as it can be seen in the image. The first one, it is the bounds. The tanh transform the outputs with bounds between -1 and 1. The results is that the representation is now centered at zero. The negative inputs considered as strongly negative, zero input values mapped near zero, and the positive inputs regarded as positive.

This function can be written as two sigmoid functions put togethers. The vanishing gradient is also present in this activation function.

2.3.2.3 ReLU function

ReLU function is also known as rectified linear unit activation function [20]. This functions gives the input as output if it is positive and 0 otherwise. The ReLu function is shown in Figure 2.4. This activation makes the network converge much faster. It does not saturate which means it is resistant to the vanishing gradient problem at least in the positive region. Due to the horizontal line in ReLu (for negative X), the gradient can go towards 0. For activations in that region of ReLu, gradient will be 0 because of which the weights will not get adjusted during descent. That means, those neurons which go into that state will stop responding to variations in error/ input (simply because gradient is 0, nothing changes).

2.3.2.4 Softmax function

Sigmoid, tanh, ReLUs are good activation functions for the neural networks. However, when you want to deal with classification problems, they seem to have some inconveniences. The softmax [20] (also called multinomial logistic) functions transform the input into an output with bounds between 0 and 1, just like a sigmoid function. Moreover, Softmax

functions divide each output such that the total sum must be equal to 1. This output is equivalent to a categorical probability distribution. When using a multilayer neural network for classification the output layer (output units) provides a degree of membership to each class, then a multinomial logistic is used just as for the logistic regression. Mathematically, the function is the following:

$$S(x) = \frac{exp(W_k^T X)}{\sum_{j=1}^{q} W_j^T X} \tag{2.3}$$

2.3.3 Loss functions

In the previous section 2.3.4.2 we have briefly mentioned the concept of cost function which is a way to measure the accuracy of our trained model. Thus, it is also an important hyperparameter that we should establish while training a model. The accuracy represents the percentage of predictions computed properly with respect to a known output. There are several cost functions available, such as [20]:

- Quadratic cost also known as sum squared error is the most common one, for example, in Least Square techniques.

$$C = \frac{(y - y')^2}{2} \tag{2.4}$$

  where y' is the output of the perceptron.

- Cross-entropy cost expression is

$$C = -\frac{1}{n} \sum_x [y \ln y' + (1 - y) \ln(1 - y')] \tag{2.5}$$

  where n is the total number of samples in the training data. The cross-entropy is positive and tends toward zero as the neuron gets better at computing the desired output, y, for all training inputs, x. A variant of cross-entropy is **categorical cross-entropy** which is used for multi-class classification where each example belongs to a single class.

2.3.4 Artificial Neural Nets

A neural network, as said before, is a mathematic tool that models, at a very low level, the functionality of the neurons in the brain, so it comes from a biological-inspiration [20]. However, unlike a biological brain where any neuron is connected to any other neuron by a physical distance, these ANNs have discrete layers, connections, and directions of data propagation. Therefore ANNs are mainly a link of many perceptrons together in layers, and are characterized by:

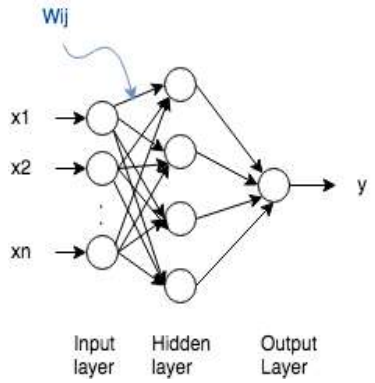

Figure 2.5: Architectural graph of a MLP with one hidden layer

- The units (artificial neurons) and their activation function, in terms of the connection weights. The connection weights are adjusted globally through a cost function in order to optimize the performances of the network.
- Its structure, architecture, or topology
- A cost function to be optimized in function of the connection weights
- An optimization or training algorithm to optimize the cost function in terms of the connection weights. Thus, to estimate the connection weights based on the training set

2.3.4.1 Structure

An example of the structure of the multilayer ANN is shown in Figure 2.5. The inputs to the network correspond to the values of the features measured for each training sample. Inputs are fed simultaneously into the units making up the input layer. They are then weighted and fed simultaneously to a hidden layer. The number of hidden layers is arbitrary, although usually only one, as in the example. Finally, the weighted outputs of the last hidden layer are input to units making up the output layer which emits the network's prediction

From a statistical point of view, artificial neural networks perform a nonlinear regression from the input into the output. Given enough hidden units and enough training samples, they can closely approximate any function.

When there are L layers (more precisely, (L + 1) layers), Layer 0 corresponds to the input units, Layer 1 corresponds to the first hidden layer and Layer L corresponds to the output units. The information propagates from the input units (layer 0) to the output units (layer L). In equation 2.6, we can see the mathematical expression of the activation of the jth unit of layer l+1 ($y_j^{(l+1)}(x)$), with respect to activation of the ith unit of layer l ($y_i^{(l)}(x)$) through the connection weight between unit i and j ($w_{ij}^{(l+1)}$).

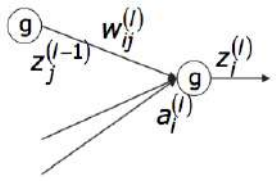

Figure 2.6: Some notations about connections between two layers

$$y_j^{(l+1)}(x) = F[\sum_{i=1}^{p(l)} w_{ij}^{(l+1)} y_i^{(l)}(x)] \tag{2.6}$$

2.3.4.2 Training algorithm

The estimation of the weights is performed thanks to a gradient-based iterative algorithm. It corresponds to the fitting, training or learning algorithm which will try to optimize the cost function in terms of the weights. It is usually called the **back-propagation algorithm**. The training proceeds in two phases [20]:

- In the **forward phase**, the synaptic weights of the network are fixed and the input signal is propagated through the network, layer by layer, until it reaches the output. Thus, in this phase, changes are confined to the activation potentials and outputs of the neurons in the network.

- In the **backward phase**, an error signal is produced by comparing the output of the network with a desired response. The resulting error signal is propagated through the network, again layer by layer, but this time the propagation is performed in the backward direction. In this second phase, successive adjustments are made to the weights.

Before going into some mathematical details, some notation is be given. In Figure 2.6 a graphic representation of those notations is given to further understand the brief mathematical explanation, where $a_i^{(l)} = \sum_j w_{ij}^{(l)} z_j^{(l-1)}$. Furthermore, the error criterions are the ones explained in section 2.3.3.

Finally, a distinction of the derivative of the error respect to the output of inner/hidden units and respect to the output of the last layer. The main difference is that, for output units the derivative of the error is proportional to the derivative of the activation function within the layer, whereas for hidden units this derivative is expressed as a combination of errors in the next layer.

From the mathematical viewpoint we can divide this algorithm in six steps:

- Apply an input vector x^k and propagate it through the network to evaluate all activations $z_j^{(l)}$ and neuron outputs $a_j^{(l)}$
- Propagate the activations forward from the input layer to the output layer and compute the activations of the output layer
- Evaluate error terms $\delta_i^{(o)}$ in output layer. This is nothing but the derivative of each error with respect to the each output of each neuron of the last layer, the output layer.
- Back-propagate error terms $\delta_i^{(l)}$ to find error terms $\delta_i^{(l-1)}$
- Evaluate all the derivatives $\frac{\delta E}{\delta w_{ij}^{(l)}} = \delta_i^{(l)} z_j^{(l-1)}$. In other words, compute the gradient for each observation k and cumulate the gradients.
- Adjust weights according to derivatives and a gradient descent scheme

However, the cost function can have many local maxima, so that the algorithm will only find a local maximum. Moreover, when starting with different initial values for the weights, we obtain different results.

2.3.4.3 Weight adjustment

Regarding the weight adjustment done during the backpropagation algorithm, we need some techniques to perform it. These techniques are also known as **Optimizers**.

Optimizers are about minimizing cost functions, and the most important ones are:

- Gradient descent (GD) [20] is the most common one. It searches the minima in a gradual way. It is also known as a first-order method and its more common mathematical expression is the following:

$$w(t+1) = w(t) - \alpha \frac{\delta E}{\delta w} \tag{2.7}$$

  where α is the learning rule. It is very usual to use an adaptive learning rule instead of fixing the same for all the steps. Even though this algorithm is the basic one, there are some problems. The main problem is its slow convergence to a minimum, due to the fact that they perform the Gradient of the whole dataset at once. Apart from the slow convergence, there are some problems related to compatibility. For example, if new data is added to the original dataset, a new training of all the weights is needed.

- SGD which is Stochastic gradient descent optimizer. This scheme is a variant of GD which aims to apply the gradient descent to different splittings of the original data [20]. This algorithm is much faster.

- RMSprop: during the learning process the magnitude of the gradient of some weights can differ a lot from others. To address this variation, applying the gradient on the full dataset, Geoff Hinton [23] proposed to take only the sign of the gradient together with the adaptation of the step size of each weight separately.

- Adagrad [21] enables using an adaptive learning rate. This optimizer addresses the problem of struggling with the learning rate in SDG. The learning rate is adapted component-wise, and for each parameter we store sum of squares oh the historical component-wise gradient. Considering G as the historical gradient $G^k = G^{k-1} + \nabla J(\theta^{k-1})^2$, and θ as each parameter, the general expression of the optimizer is:

$$\theta^k = \theta_{k-1} - \frac{\alpha}{sqrt(G^{k-1}}.\nabla J(\theta_{k-1}) \tag{2.8}$$

  Therefore Adagrad modifies the general learning rate α at each time step t for every parameter θ_i based on the past gradients that have been computed for θ_i.

- Adam, Adaptive Moment Estimation [29], can be seen as a generalization of Adagrad, in which it uses momentum estimations to update the rule. That means that in addition to storing the average of past squared gradients, it also keeps the average of past gradients similar to momentum.

For this project Adam is chosen because it is straightforward, efficient and only consume little memory.

2.4 Deep Learning

Deep Learning can be considered as a powerful set of techniques for learning in neural networks, which currently provides the best solution in image and speech recognition as well as language processing, main trends and crucial issues for the years to come in technology. A Deep Neural Network (DNN) is nothing but a ANN with multiple hidden layers.

2.4.1 Convolutional Neural Networks

CNNs [48] is a type of ANN which make the explicit assumption that the inputs are images and try to take advantage of the spatial structure. Therefore, they are very suitable for image recognition or classification. There are different types of layers in a CNN.

2.4.1.1 Convolutional layer

This type of layers are the core of the ConvNets and will perform the highest number of operations. The main operator of the CNN is the convolution. This aims at retrieving or extracting some features from the input image. As metioned before, the principal aspect of the convolution is the preservance of the spatial relationships between pixels by learning those image features using small squares of input data. Therefore, an important aspect in these architectures are the ***filters*** or *kernels* to perform the convolution, which are smaller matrixes.

In images the convolution is performed by sliding the kernel through the original image, and compute the element wise multiplication obtaining another matrix. The resultant matrix of this dot product is called *Convolved Feature* or ***Feature Map***. In images this convolution is widely used to Edge detection, Sharpen and blur depending on the filter used in each case.

Regarding the number of neurons of the output matrix there are three hyperparameters that control the size [48].

- The depth which corresponds to the number of filters we use for the convolution operation. In future sections this number is given for the architectures proposed for this project. Nevertheless, commonly 64 filters is used.

- The stride is the number of pixels by which we slide our filter matrix over the input matrix. Therefore, when this value is one we will move the filter one pixel at a time. For this project value 1 is mostly used.

- Zero-padding is the size of the padding that is applied in some cases around the border, to perform properly the convolution. In addition, this parameter will allow us to control the size of the output volumes. If the zero-padding technique is implemented all the elements that would fall outside of the matrix are taken to be zero. Adding this padding is also called wide convolution, and not using it is called narrow convolution.

In practice, when we train a CNN the convolution is automatically done, but different parameters must be specified. Some of the most important are the number of filter and the filter size.

Regarding the connections between neurons, in these architectures each neuron will be connected to only a local region of the input volume. This local region is a hyperparameter called **receptive field** which is equal to the filter size.

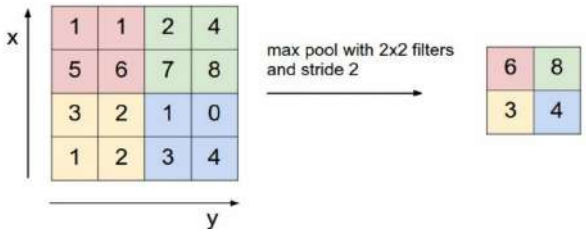

Figure 2.7: An example of a max pooling technique extracted from [48]

2.4.1.2 Pooling Layers

This type of layers is usually implemented just after the convolutional one. It conducts a downsampling, to reduce the dimensionality of the feature maps retaining the most important information. There are different types of pooling techniques such as max or average. A pooling function replaces the output of the net at a certain location with a with a summary statistic of the nearby outputs [19]. The most common one is the max, which takes the maximum of the results of each filter. For a ilustrated explanation, see Figure 2.7.

The principal goal of this type of layers is the progressively reduction of the spatial size of the input representation, and thus making them more manageable. In addition, it helps to make the matrix approximately translation-invariant of the input. This means that if the input suffered a small translation(shifting) and/or rotation the pooled output does not change.

Regarding the backpropagation algorithm to train ANN, in the pooling layers using max function only the gradient of the input that had highest value will be forwarding. Hence, a very common practice is to keep track of the index of the max activation, in order to route the gradient efficiently.

Due to the fact that this layer summarizes the output value over a whole neighborhood, there is the possibility of using less pooling units than detector ones. The way that this is addressed is by reporting summary statistics for pooling regions spaced k pixels instead of 1 pixel apart. Therefore they are very suitable to avoid **overfitting**.

2.4.1.3 Fully connected layers

These type of layers are the ones of the MLP already mentioned in chapter 2.3.4.That means that, every neuron in the previous layer is connected to every neuron on the next one. The activation function typically used in these layers is the Softmax, which as mentioned, widely used for classification.

As a summary of this chapter, the CNN structure can be divided into two parts: Convolution + Pooling layers that act as Feature Extractors of the input image and the Fully

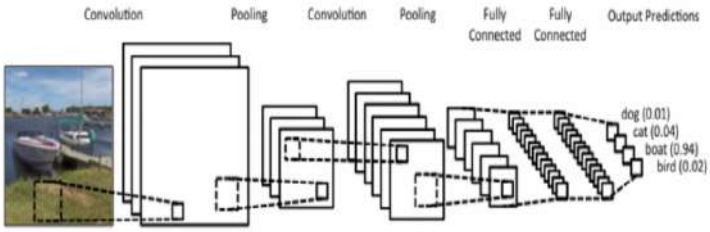

Figure 2.8: An example of a CNN architecture extracted from [48]

connected layer performs the classification as such. In Figure 2.8 the general architecture of a CNN is ilustrated.

CHAPTER 3

Enabling Technolgies

This chapter offers a brief review of the main technologies that have made possible this project, as well as some of the related published works.

3.1 Machine Learning Technologies

3.1.1 TensorFlow

TensorFlow (TF) [1] is an open source software library for numerical computation using data flow graphs. TensorFlow originated as an internal library that Google developers used to build models in-house, and we expect additional functionality to be added to the open source version as they are tested and vetted in the internal flavor. It was released on the last semester of 2015.It reached version 1.0 in February 2017, continuing with a rapid increase, reaching more than 21 thousand commits.

The main characteristic of TF is the flexibility due to the fact that it is a cross-platform. in other words, it can be run on nearly every device, ranging from mobile devices such as phones and tablets up to large-scale distributed systems of hundreds of machines and thousands of computational devices. It is executed over GPU or CPU, and even TPUs (tensor processing units) which are a specialized hardware.

There are many APIs already included in TensorFlow which ease the understanding of the behavior of the Neural Networks implemented. One important tool, is TensorBoard, which are a suite of visualization tools in order to make easier the understanding, debugging and to better optimize TensorFlow graphs. Moreover, different metrics can be plot in order to show additional information. Another important issue about this tool, is the display of this metrics in real time.

Nowadays, it is one of the most popular libraries to develop Deep Learning algorithms, due to two main reasons: the ease with which this models can be developed using Python and the speed of doing this; and as it has been developed by Google, there is a lot of documentation available.

3.1.2 Keras

Keras [10] is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result with the least possible delay is key to doing good research. The main properties of Keras are:

- Easy-to-use: Keras is an API designed for human beings, not machines. Keras follows best practices for reducing cognitive load. It offers consistent and simple APIs, it minimizes the number of user actions required for common use cases, and it provides clear and actionable feedback upon user error. This ease of use does not come at the cost of reduced flexibility: because Keras integrates with lower-level deep learning languages (in particular TensorFlow).

- Modularity: every Deep Learning model is composed of different elements such as, layers, activation functions, costs functions optimizers or regularization schemes. Thus, Keras offers these elements as modules, in order to use them as required. The user can combine these elements to create models with slight differences, changing only one of these modules, resulting in a very simple process.

- Concerning the modularity, new modules can be created and added seamlessly . Therefore, it is easily extensible.

- Finally, as Keras core is written in Python, all the models will be implemented with this language,which is compact and easier to debug,

As Keras is usually over TensorFlow, using different functions you can integrate your model with your TensorFlow workflow.

The main idea is to use the Keras API with Tensorflow as backend, to implement the Deep Learning models.

3.1.3 Colaboratory

Collaboratory[1] is a Google research project created for dissemination and training on machine learning. It is a Jupyter Notebook environment that does not require setting up and runs completely in the cloud.

Collaboratory notebooks are stored in Google Drive, and you can share them as you would with spreadsheets or Google Docs. Therefore, it is a very suitable tool while working in a project with other developers, as everybody can be working on the same document. As Google Docs, it offers an online tool to develop your project everywhere, without any further installation.As every Google tool, is a free service.

Even though the main characteristic of Collab is the cloud execution , it also allows you to connect to a Jupyter runtime on local device.

In this platform, you can run programs written either on Python2 or Python3. Changing this language whenever is required. As mentioned before any complex installation is needed, as it allows you to run your TensorFlow code in the browser. Furthermore, it is also included many useful Python libraries, such as *matplotlib* to simplify the visualization of the data.

Other characteristics is that you can use a free **GPU** (k80) to train faster your models. In addition, you can upload code from GitHub or from your local computer.

To conclude, this is a new and very interested tool to use to train Deep Learning models. This models usually consume a lot of training time and resources. Therefore, this can be

[1] `https://colab.research.google.com/notebooks/welcome.ipynb`

shortcut with this tool, as you can train a complex model in your browser in less than an hour, whereas in a normal device it will spend more than 20 hours.

3.2 OpenCV

OpenCV [7] (Open Source Computer Vision Library) is released under a BSD license and hence it's free for both academic and commercial use. It has C++, Python and Java interfaces and supports Windows, Linux, Mac OS, iOS and Android This library helped the development of Computer Vision field due to the fact there are more than 2500 optimized algorithms, which includes both classic and state-of-the-art computer vision and machine learning algorithms [7]. Therefore, it was an important milestone because thanks to this complex applications, focused for instance on detecting objects , can be implemented nowadays with few lines of code, and with a wide option of programming languages.

The first version was released in 1999 by Intel, and after several releases nowadays there are more than 47 thousand people collaborating, and the number of downloads since the very beginning exceeds 14 million. Moreover, the library is widely used in companies, research groups and in government institutions. In spite of being created by Intel, currently important technological companies such as Google, Yahoo, Microsoft or IBM are helping to the improvement of this library.

3.3 NodeJS

This section of the report delves into one of the most important Frameworks in web. NodeJS [43] is the framework used to implement the logic of the application, on the server side. One of its peculiarities is the use of **Javascript**[2] as the programming language to develop the application.

Javascript is a scripting language. At the beginning its main use was to develop small portions of code without a clear structure, called scripts. These scripts executed some routines, with the purpose of providing dynamism to the first webpages, implemented entirely with static HTML. Therefore, it was only use to develop webpages, but in 2008 Google release the first Google Chrome, in which there was a Javascript compiler. From that moment it was possible to execute Javascript source code, in any other environment that was not a web browser. Ryan Dahl, a programmer, together with his development team created NodeJS after this. In 2009 this framework was presented.

There are two main issues to highlight in NodeJS, as one of he best technologies with Javascript on server side:

[2] `https://www.javascript.com/`

- It solves the eternal problematic of the Javascript language, since this is a scripting language. All scripting language can not be grouped into blocks, modules, or portions of code that can be properly organized, and can be adjusted to a specific SW architecture. NodeJS allows to make small code modules that can collaborate with each other, in order to organize the source code according to their specialization or other criteria.

- Possibility of expanding the standard module package of NodeJS, with modules implemented by third parties, from companies to individuals. Since NodeJS is a framework, it saves the need to implement some basic functionalities in any web application, with the help of provided modules . If it is also possible to add more modules to the set of existing ones, in a simple way and without extra costs.

3.4 Express

Express [3] is a NodeJS module implemented by third parties. Is one of the principals modules to develop web projects, providing a large API, with multiple functionalities already implemented. It is such a powerful tool that sometimes nowadays is frequently used as an independent framework specialized in web environments.

It provides many efficient functionalities required in ever web project. All the web connections are done through HTTP protocol. An important characteristic of Express is that it can intercept the requests made by an user through the browser, during the execution on the server. This is also capable of processing the request, assigning a function so that it carries out the corresponding task. Once a HTPP request is done, Express review the URL associated to it and its parameters

The integration with NodeJS is so simple. Firstly, the dependency must be specified in the **package.json**[4] file and then download the source code of this module using **npm**[5]. npm iss the world's largest software registry, containing over 600000 packages. Package.json is the best way to manage locally installed npm packages since it lists the packages that your project depends on and allows us to specify the versions of a package that your project can use.

[3] `https://expressjs.com/`
[4] `https://docs.npmjs.com/getting-started/using-a-package.json`
[5] `https://docs.npmjs.com/getting-started/what-is-npm`

3.5 Websockets

One of the main functionalities within NodeJS are websockets [50]. A websocket is nothing but a way of client-server communication over a unique TCP socket. Before the release of such an appealing technique, there was the technology called Ajax, which was the HTTP's basic request/response mechanism. The main difference between them, is that with wesocket some non-bloquing and bidirectional communication threads can be created.

Initially, the programming language adopted to implement them was C, due to it is the most efficient language to perform some complex algorithms. After the release of the V8 Google's compiler for Javascript, the NodeJS creator decided that Javascript could be the language to implement these websocket.

The prior HTTP request/response mechanism had some limitations such as constant opening and closing of connections, which hindered the creation of real time applications. Nevertheless, as webSockets are a full-duplex communication between server and client, they are really suitable for this type of applications.

The communication protocol via websocket is represented in Figure 3.1. To initiate the communication via webscoket, first of all, the connection must be established through a handshake between the server and the client, and at this moment the socket is created. From that point onwards, each side can modify the state or information of the other side. That means that, both the server and client can simultaneously send new data without the other asking for it.

As in traditional way of client-server communication, the client can modify server resources at data level. Javascript is very suitable for this type of communication due to the fact that it is event-oriented. There are some events in client side, with which it is established the communication with the server and transfer data to it. Furthermore, in the server side there are other events in order to transfer them to the server as well.

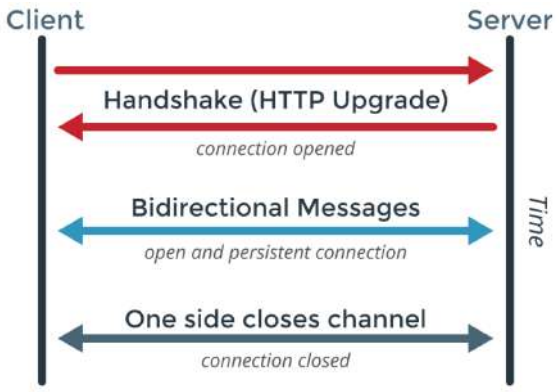

Figure 3.1: Websocket connection protocol seen in [50]

3.6 Docker

Docker [33] is a technology that is gaining popularity in IT field and it is considered one of the fastest growing technologies in the recent software history. Docker is nothing but a platform that allow developers to easily pack, distribute and manage apps within containers. In other words, It is an open-source project that automates the deployment of applications inside software containers.

A **container** is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another. Furthermore, containers isolate the software from its environment ensuring that it works uniformly. The functional architecture of containers is shown in Figure 3.2. In this image is represented the traditional way of virtualize an app using virtual machines (VMs) and with containers.

As can be seen in traditional way of virtualization, a limitation is the kernel resource duplication. This is due to the fact each virtual machines needs to have an operating system installed. In other words, an entire guest operation system with its own memory management, device drivers, etc. Nevertheless, with containers this problem disappears and a cost-efficiency is achieved, as this does not create an entire virtual operating system, packing up only the required components inside the container with the application. So containers consume less CPU, RAM and storage space. Meaning that we can have more containers running on one physical machine than VMs.

Another limitation of virtual machines is that the application portability is not guaranteed, but with containers this portability is assured, because containers are essentially independent self-sufficient application bundles, they can be run across machines without compatibility issues.

In addition, another advantage of using a container-based virtualization is the runtime isolation. For instance, two applications that need two different libraries or just two different versions of the same library, each app can be run in a different container. Concerning this runtime isolation, with Docker a fast deployment is achieved. This is due to the fact that it creates a container for every process and does not boot an OS. Data can be created and destroyed without worry that the cost to bring it up again would be higher than what is affordable.

Continuing with concepts of Docker the term of **images** comes out. Images are read only templates used to create containers. Images are created with the docker build command, either by us or by other docker users.

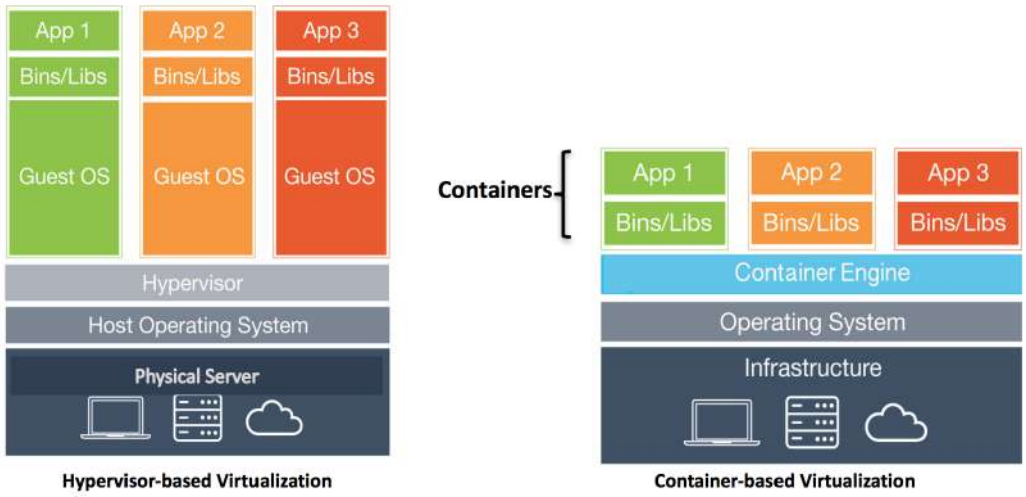

Figure 3.2: Comparison between hypervisor-based and container-based virtualization [33]

CHAPTER 4

Deep learning Model for Emotion Analysis

In this chapter a description of the steps followed to train the different models is detailed. First of all different Deep Learning architectures are presented, followed by an experimentation section.

4.1 Introduction

In this chapter all the process followed to develop a Deep Learning model is explained. The very first step is to define the architecture of the model, meaning by architecture, the combination and number of layers explained in section 2.4.1. Four architectures are detailed.

Secondly an experimentation phase is carried out. The first step to perform a test is the setting-up of the hyperparameters that define each architecture. This is explained in section 4.3.1. The second step is the choice of the dataset used to train the model (section 4.3.2). Thirdly, in section 4.3.3, a model selection is done in order to retrieve the model with the best trade-off between accuracy and complexity. Finally, in section 4.3.4 a confusion matrix of the selected model is given, in order to better understand the behavior of this model.

4.2 Architectures

In this chapter, the architecture of four networks is detailed. It is important to note that there is no specific formula to building a neural network that would guarantee to work well. Different problems would require different network architecture and a lot of trail and errors to produce desirable validation accuracy. This is the reason why neural nets are often perceived as *black box algorithms*. Therefore, this approach of trial-and-error will be adopted an different combinations will be tested. As mentioned in previous chapters, the library **Keras** upon **Tensorflow** has been used to implement all the models.

4.2.1 First architecture

This architecture is inspired in a GitHub repository[1]. The main idea behind this architecture is to introduce as much convolutional layers as possible without loosing information or leading to overfitting, so that several feature maps can be obtained that describe thoroughly the main points of the face for each emotion. Thus, the model will learn automatically what was explained by Ekman in [16].

The combination of layers is illustrated in Figure 4.1, and basically consist of : 3 sequential convolutional layers with 32 filters of size 3x3; a pooling layer using the max function to pool; 3 sequential convolutional layers of 64 filters of 3x3; a pooling layer using the max function to pool; 3 sequential convolutional layers of 128 filters of 3x3; a pooling layer using the max function to pool; a Fully Connected layer of 64 units with a dropout of 20% ; a Fully Connected layer of 64 units with a dropout of 20% and finally a Fully Connected layer

[1] `https://github.com/JostineHo/mememoji`

with Softmax activation function. In addition, all the layers use a ReLu activation function, except the last one, which is the one that performs the classification itself.

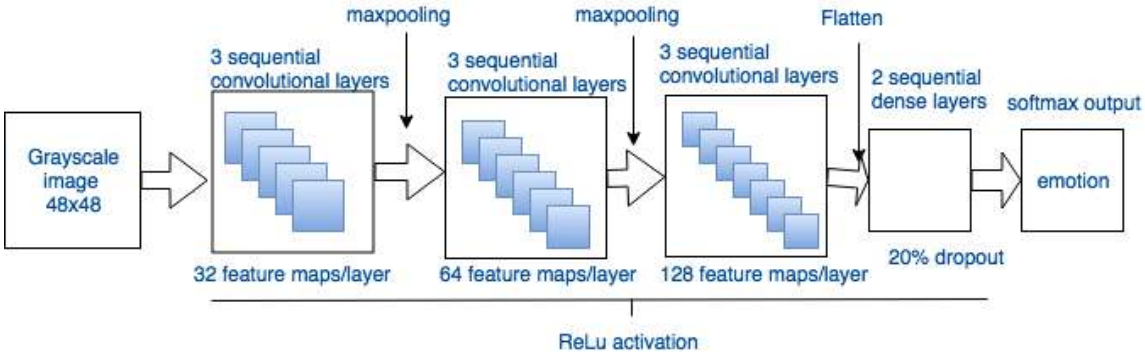

Figure 4.1: First CNN architecture

4.2.2 Second architecture

In this case, a smaller structure was chosen. The architecture is proposed in [13], and this CNN is designed with some modification on LeNet Architecture. The main difference with respect to the previous one is the size, and thus the number of trainable parameters. It consists of 8-layer CNN with three convolutional layers, three pooling layers, and two fully connected layers. The activation function used was ReLu for all the layers except the last one which was a Softmax.

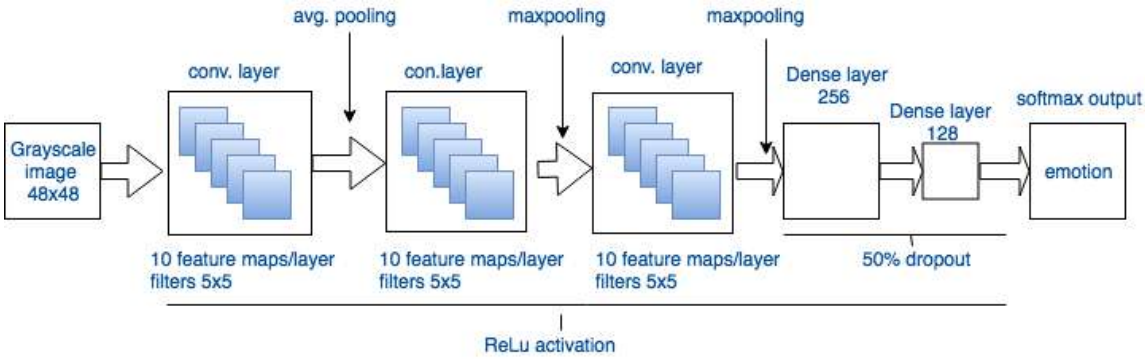

Figure 4.2: Second CNN architecture

4.2.3 Third architecture

As suggested in [37], this architecture is a CNN with a fixed size of five convolutional layers: 64 filters of 5x5, 64 filters 3x3, 64 filters 3x3, 128 filters 3x3 and 128 filters 3x3. After this convolutions, two fully connected layers of 1024 units, each of them followed by a dropout of 20%. All these layers have a ReLu activation. Finally the softmax layer is

implemented. This is represented in Figure 4.3. The difference with respect to the first and second architecture is the size. This architecture has five convolutional layers, instead of 9 in the first one, and 8 in the second one. The reason to do so, is to try different combinations of different types of layers looking for the best performance, together with less parameters.

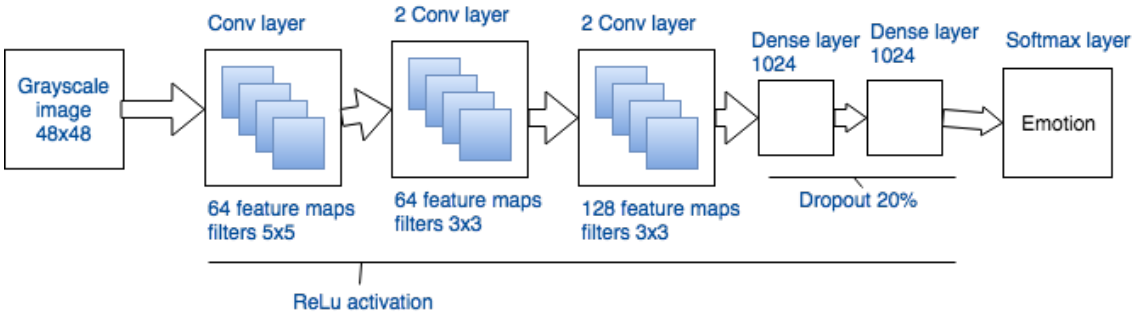

Figure 4.3: Third CNN architecture

4.2.4 Fourth architecture

In this architecture different implementation techniques were carried out. The reason behind is the poor accuracy obtained with previous architectures, which will be explained in future sections, and less trainable parameters, enhancing the speed performance. Therefore, some *innovations* were done as proposed in [36] which was inspired in Xception architecture [11].

First of all, is the concept of **residual modules**. Residual modules is a technique to ease the training of networks, and they modify the desired mapping between two subsequent layers, so that the learned features become the difference of the original feature map and the desired features[38]. Consequently, the desired features H(x) are modified in order to solve an easier learning problem F(X) such that:

$$H(x) = F(x) + x \tag{4.1}$$

A graphical representation of this technique is given in Figure 4.4 , so that the reader can fully understand the implemented network.

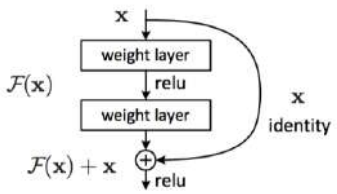

Figure 4.4: A representation of the residual modules technique extracted from [38]

Secondly, **depth-wise separable convolution** which is a form of factorized convolutions which factorize a standard convolution into a depthwise convolution and a 1 x 1 convolution called a pointwise convolution. Therefore, there are two main layers in each convolution, which aim to to separate the spatial cross-correlations from the channel cross-correlations. Depthwise convolutions are used to apply a single filter per each input channel (input depth). Pointwise convolution, a simple 1 x 1 convolution, is then used to create a linear combination of the output of the depthwise layer [3]. In spite of they are extremely efficient compare to standard convolutions, it does not combine the filtered channels to create new features. That's where the new layer comes up, the point-wise layer is used to compute a linear combination of the outputs of the depth-wise convolution. In a nutshell, depth-wise separable convolutions reduces the computation with respect to the standard convolutions, so that the training phase is faster.

Thirdly, as the training process with previous network was notoriously hard and thus saturated with a nonlinearities, also known as *internal covariate shift*, a solution found is normalizing the inputs of the layers. This is called **Batch-normalization** [26], and it also acts as a regularizer avoiding in certain cases the use of dropouts. This technique performs an easy operation which is applied to activation x over a mini-batch, as shown in eq. 4.2. The term μ_B refers to the mini-batch mean, σ^2 mini-batch variance and ϵ is a constant added to the mini-batch variance for numerical stability.

$$x_i' = \frac{x_i - \mu_B}{\sqrt{\sigma^2 + \epsilon}} \tag{4.2}$$

Finally the concept of regularization applied in some layers. Regularizers aim at generalize the behaviour of the net. Specifically in this architecture, together with other already mentioned, **l2-regularization** (aka weight decay) will be implemented in some layers. The idea of L2 regularization is to add an extra term to the cost function, called the regularization term. As an example, using the cross-entropy function seen in 2.3.3 the generalization term is added as follows:

$$C = -\frac{1}{n}\sum_x [y \ln y' + (1-y)\ln(1-y')] + \frac{\lambda}{2n}\sum_w w^2 \tag{4.3}$$

The first term corresponds to the function itself, and the second one is the term, in which the sum of the squares of all the weights in the network is done, scaled by a factor, where λ is the regularization parameter. In this case the scale factor used is 0.01 .

Our final architecture, as shown in Figure 4.5 is a fully-convolutional neural network consisting of 2 convolutional layers of 8 features. ReLu activation, 3x3 with a kernel regularizer, followed by a batch normalization. Then there are four modules in which a residual is implemented (convolutional layer, 1x1, followed by a batch normalization) and added

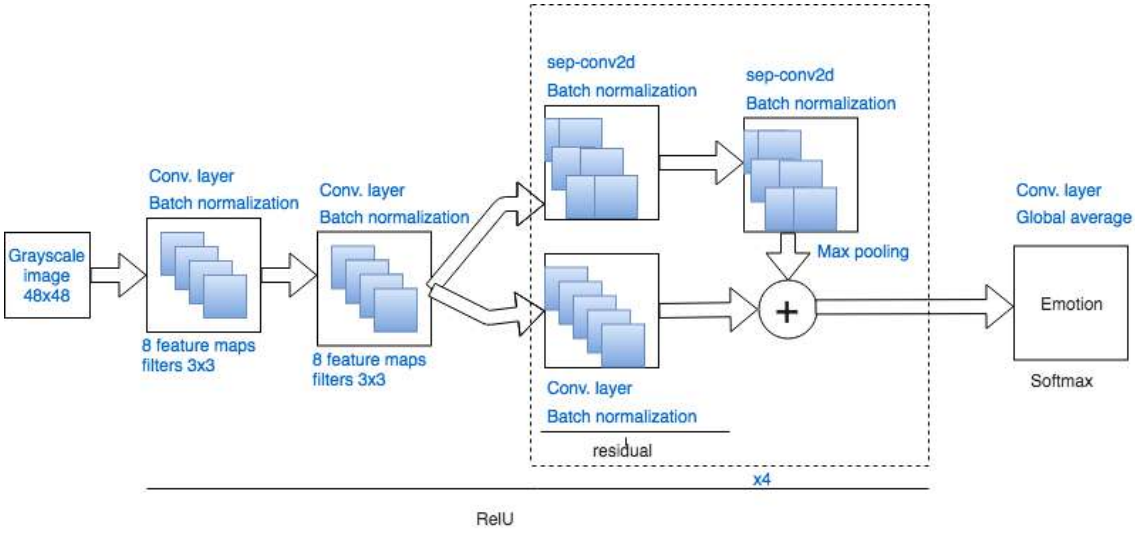

Figure 4.5: Fourth CNN architecture

to the separable convolutional layer, 3x3 with kernel regularization, followed by a batch normalization as well. The size of these convolutional kernels is increasing in each module, starting in 16 and ending in 128. Finally the classification itself is carried out through a convolutional layer with the size equal to the number of emotions to be predicted, followed by a Global average pooling layer with Softmax activation.

4.3 Experimentation

4.3.1 Setting-up

Some hyperparameters must be defined in advance to train a model using Keras. The main reason behind this selection is to avoid overffitng. This term means that the model predicts properly on training data, but on new data the performance decreases considerably. This is one of the main problems in ANN, and especially in Deep Learning architectures. This is due to a high number of free parameters that can lead to a final unexpected solution. As explained in section 2.3, these parameters are:

First of all, **Batch Size** defines number of samples that are going to be propagated through the network. A batch is a set of N samples, being a sample one element of a dataset. The samples in a batch are processed independently, in parallel. If training, a batch results in only one update to the model. On the one hand, some advantages of using batch-size is that it requires less memory to train. this is due to the fact that, using less number of samples rather than using the whole dataset, consumes less resources. This can be a big issue while trying to train the model in a computer with weak resources, such as local computer. Furtermore, another advantage is that the training phase of the network takes less time, as all the weights are updated after each propagation. Therefore, if all the training samples are used the weights will be fixed only once. Moreover, as stated in [46], it has been observed in practice, that when using a larger batch there is a significant degradation in the quality of the model, as measured by its ability to generalize. To support this idea they tested different models using different sizes and they conclude that using large-batch methods tend to converge to sharp minimizers, and thus to poorer generalization. In other words, the model will be overfitted.

Nevertheless, there are earlier researchers which state that in some cases is not useful to use small Batch size to decrease the overfitting. For instance, in [52] some researchers from Google explained some findings which enabled the efficient use of vast batch sizes, significantly reducing the number of parameter updatings required to train a model. Hence, in this project different batch sizes **(32, 64, 80,128)** will be used for each architecture, to compare the final accuracy obtained.

Secondly, another parameter that must be specified is the **number of epochs** which will determine the number of iterations during the training process. One epoch is one forward pass and one backward pass of all the training examples, and iterations is the number of passes, each pass using [batch size] number of examples. As an example, if you have 1000 training examples, and your batch size is 500, then it will take 2 iterations to complete 1 epoch. Normally, if the number of epochs is too high, at a certain point, the accuracy will

top increasing and therefore the weights will be too fixed to the training data, overfitting. However, there are additional techniques that allow you to avoid choosing in advance this hyperparameter, due to the fact that they are built in order to avoid this overfitting. One important technique that will be implemented in all the training processes is called **Early Stoppings**. This is nothing but a form of regularization used to avoid overfitting when training a learner with an iterative method, such as gradient descent.

Another important hyperparameter is the **Loss function** used in the backpropagation method. As mentioned in chapter 2.3.3 in terms of classification the most used is cross-entropy, and especially **categorical cross-entropy**, used for multi-class classification where each example belongs to a single class. This parameter is specified while compiling the model architecture.

Furthermore, the type of **Activation function** must be selected for each layer. In general terms, for the first layers will be implemented using **ReLu** functions, and the last layer, the Fully connected, will be implemented using **Softmax**. In the last layer this choice has been done due to is has been commonly used in many of the most popular Image Classification Deep Learning architectures such as ImageNet [22], VGG16 [40] or Inception [25]. Optimizers, as explained in 2.3.4.3 play a very crucial role in increasing the accuracy of the model. For all the models Adam optimizer will be selected. These parameters are specified while compiling the model as well.

Essentially, the more layers/nodes we add to the network the better it can pick up signals. As good as it may sound, the model also becomes increasingly prone to be overffited with respect to the training data. Another good method to prevent overfitting and generalize on unseen data, apart from the already mentioned, is to apply dropouts. Dropout randomly selects a portion (usually less than 50%) of nodes to set their weights to zero during training. This method can effectively control the model's sensitivity to noise during training while maintaining the necessary complexity of the architecture.

Figure 4.6: Example of FER2013 dataset extracted from [24]

.

4.3.2 Datasets

In every project involving Machine Learning algorithms whereby a processing of data, a previous stage must be done: data exploration. Data Exploratory Analysis is an important phase whence a Data Scientist can fully understand the context and the incoming data for a subsequent correct model selection.

The dataset used for training the model is from a Kaggle Facial Expression Recognition Challenge [24] a few years back **(FER2013)**. It compromises a total of 35887, 48-by-48-pixel grayscale images of faces each labeled with one of the emotions defined in section 2.1 plus a neutral one. Therefore the final result is: anger, disgust, fear, happiness, sadness, surprise, and neutral. An example of some emotions is represented in Figure 4.6.

These faces have been preprocessed so that they occupy more or less the same space in each image and have the same size. The set contains two columns, “emotion” and “pixels”. The emotions have been labeled using categorical classes, thus from 0 to 6 instead of the name itself. The column pixels contains a string.

This dataset was prepared by Pierre-Luc Carrier and Aaron Courville, as part of an ongoing research project. They have graciously provided the workshop organizers with a preliminary version of their dataset to use for this contest.

The distribution of the images is the one shown in Figure 4.7. It is notorious the imbalance of the emotion *disgust* compared to other classes. Thus, this may result in a data leakage, leading to a low accuracy predicting classes of this type. As opposed with the emotion *happiness* there is slightly higher difference with respect the others. Therefore, overfitting must be treated carefully to avoid predictions principally on *happiness*.

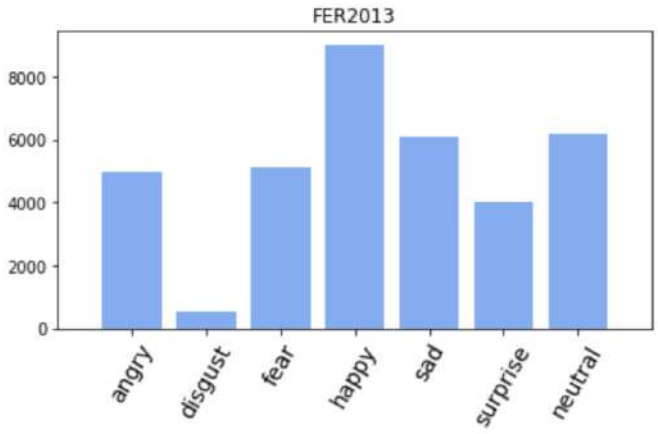

Figure 4.7: Distribution of number of images per emotion

Architecture	Non-trainable params	Trainable params	Total params
1	0	329,031	329,031
2	0	101,735	101,735
3	0	1,485,831	1,485,831
4	1,472	56,951	58,423

Table 4.1: Comparison of the number of parameters of each network architecture

4.3.3 Model selection

In this section, the testing process will be detailed. Through all this section a continuous comparison of the four architectures implemented will be done.

First of all, before going into performance details, the number of network parameters will be taken into account. In table 4.1, these parameters are specified. One of the objectives sought was the reduction of total parameters so that the real-time prediction can be performed faster. At first glance, it can be seen that the only architecture with non-trainable parameters is the fourth one. This is due to the elimination of dense or fully connected connected layers in this implementation, being this, therefore, the one with less total parameters.

Secondly, as mentioned in Section 4.3.1, once the architecture has been established, the training phase will pursue with some hyperparamenters previously fixed. All the models

have been compiled using Adam optimizer and categorical cross-entropy as a loss function. In addition, several batch sizes and validation method will be used. To do so, a validation set of 20% will be set, in order to compute the validation accuracy on each epoch.

Thirdly, some modifications in the data will be done in order to prevent overfitting. Thus **data augmentation** is performed. This is another regularization method which aims at performing some variations in the original images in order to obtain a higher dataset, and thus to have more different data avoiding overfitting. There are many type of variations, such as rotations or zooming, but the ones implemented are:

- Random rotations with 10 degrees range. This is done in order to train the model to better handle rotations of images in real applications.
- Random shifts, both vertical and horizontal. This is done because detected faces may be not centered, they may be off-center in a variety of different ways. Therefore a range of 0.1 shift is done.
- Random Flips only on horizontal axis.
- Zoom of 0.1 range.

Thereupon, to avoid the overfitting caused by the number of epochs, different solutions has been applied while training these models. The first one is **early stopping**, which attempts to remove the need to manually set this value. This is a form of regularization used to avoid overfitting when training a learner with an iterative method, such as gradient descent. It can also be considered a type of regularization method in that it can stop the network from overfitting. The main idea of this technique is that at the end of each epoch the network will be evaluated with the test set, and if the network outperforms the previous best model, a copy of the network at the current epoch is saved. Hence, the final model will be the one with the best test performance.

The second technique implemented **reduction of learning rate**. This means that the learning rate is reduced when a metric has stopped improving. This callback monitors a quantity and if no improvement is seen for a 'patience' number of epochs, the learning rate is reduced.

Finally, the training phase is carried out. The four models have been trained using different batch-sizes. All the accuracies obtained, are shown in Table 4.2. Model accuracy is a percentage indicating the number of correct classifications done by the network during the test phase.To perform this test, an amount of samples have been separated before the training phase. More precisely, as mentioned before, this amount is the 20% of the total set, that means that, 7178 samples are used for model validation.

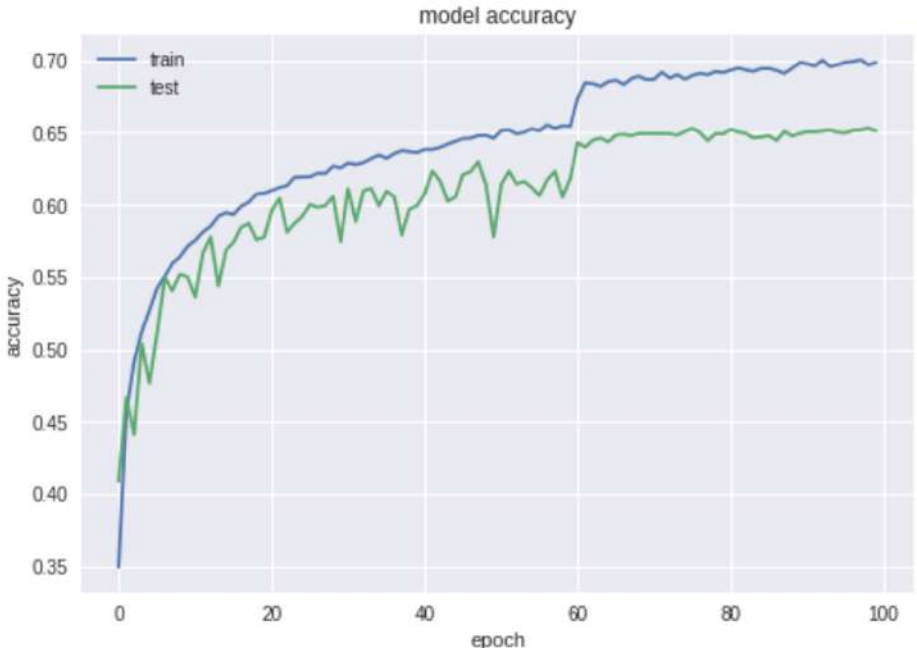

Figure 4.8: Model accuracy of the fourth CNN architecture with 100 epochs training

Regarding all these results, an overfitting can be appreciated in some cases. As an example, with the second architecture using a batch-size of 60 units, the training and test accuracies obtained are 82.27% and 50.72% respectively, differing notoriously from each other. Thus, testing with new samples plays an important role in model validation, because it helps you to simulate a real-world scenario, in which the data used to test has not been seen previously by the algorithm, avoiding models with poor performance.

The **best** trade-off between **training** and **test accuracy** is obtained with the **fourth architecture** and a **batch-size of 32**. These accuracies are **71.57 %** and **65.13 %** respectively. The difference between these two accuracies is not too high, being almost the same. This is due to all the generalization techniques applied with which overfitting has been avoid.

Moreover, the evolution of this accuracy and model loss are shown in Figures 4.8 and 4.9. Glancing at these graphics it can be observed this effect of non-overfitting, as the training accuracy is increasing considerably as well as the test one. The same is happening with loss, as training loss is decreasing, test loss decreases too. Nevertheless, it is true that test accuracy over all the epochs remains lower, whereas test loss remains higher than the training one.

Finally, comparing the training times, the fourth architecture takes more time to do it. The reason is the introduction of different filter techniques as well as more layers to extract

Architecture	Batch size	Train accuracy	Test accuracy
1	256	88.23	55.698
1	128	89.7558	56.018
1	80	89.275	56.018
1	60	89.24	55.29
1	32	89.12	54.29
2	256	68.97	50.68
2	128	75.84	50.33
2	80	56.64	49.13
2	60	82.27	50.72
2	32	81.01	50.68
3	256	64.84	55.64
3	128	81.63	55.53
3	80	70.01	53.94
3	60	83.72	52.54
3	32	86.16	52.55
4	**32**	**71.57**	**65.13**
4	64	70.84	64.28
4	80	70.83	64.21
4	128	63.31	57.58
4	256	70.86	63.03

Table 4.2: Performance comparison of each architecture highlighting the best model

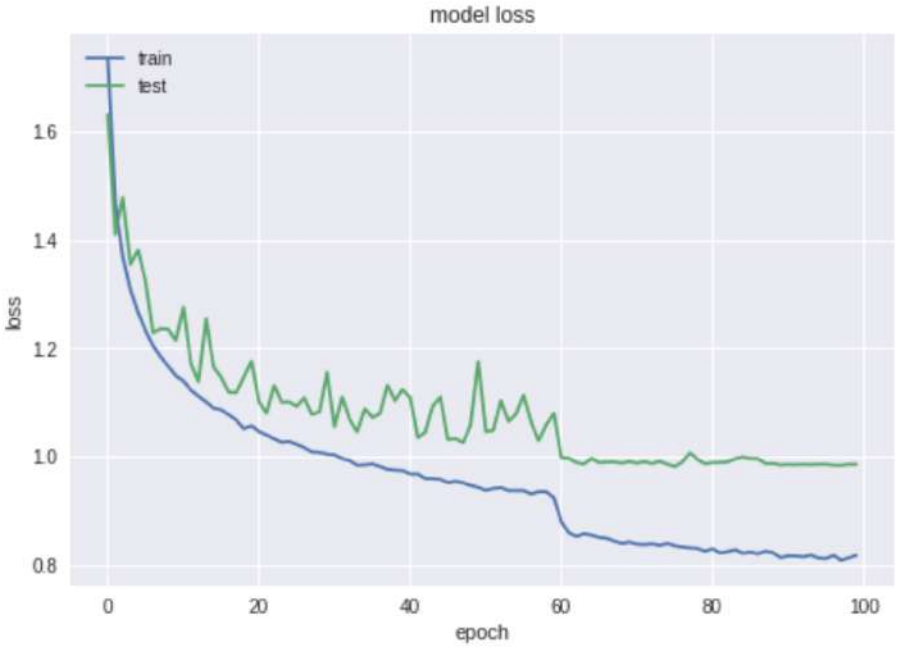

Figure 4.9: Model loss of the fourth CNN architecture with 100 epochs training

features. Moreover, the residual techniques takes more computational time to train. For example, training network 1 and 4 with the same hyperparameters (256 batch size and early stopping) the second took more than one hour using Colab. On the opposite, the other network took only ten minutes.

4.3.4 Confusion matrix

Concerning data visualization techniques, apart from the traditional plots, a wide-used one is the plot of the confusion matrix. Confusion matrix is also known as error matrix in statistical field, due to the fact that it represents the performance of an algorithm. Each row of the matrix represents the instances in a predicted class while each column represents the instances in an actual class (or vice versa). In Figure 4.10 it is represented the confusion matrix of the model selected on test data. As explained before, columns stand for the number of prediction done of each emotion.

In order to better understand this figure, it is better to take an example. Concerning the large of images with a happy face, the model has predicted correctly 1453 images out of 1620. This means that only 167 images are misclassified. Keeping on this example. it can be seen that the model confuses more happy with neutral faces. Thinking about a neutral face and a happy face it makes sense, since there are slightly differences between these two facial expressions.

Furthermore, it can be seen the emotion with the poorest performance is anger. Only

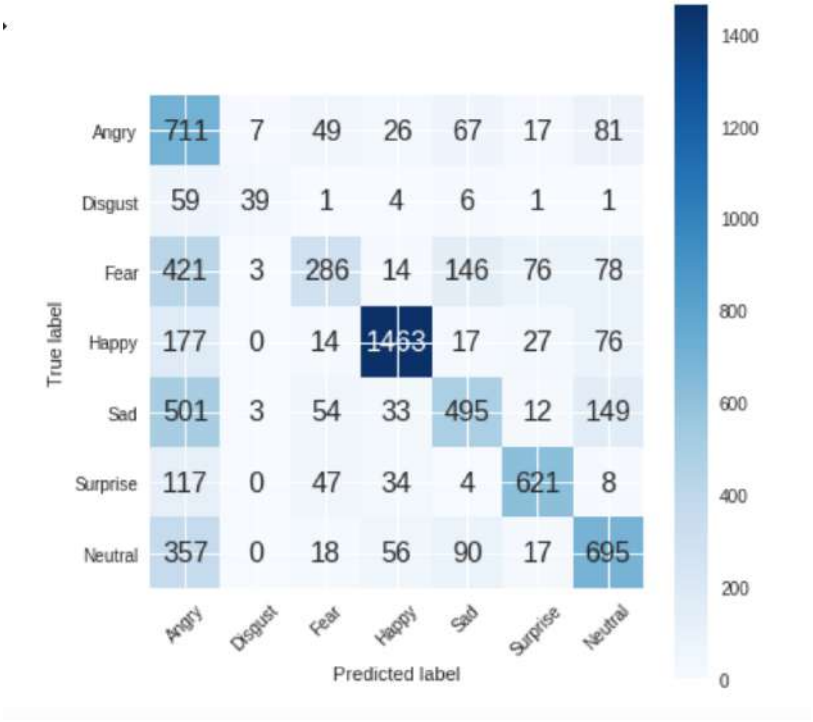

Figure 4.10: Confusion matrix with test data using the best model

711 correct predictions out of 2166. The algorithm predicts an angry face as a sad face. As previously this is due to similarities between these two expressions.

4.4 Conclusion

To conclude, after the comparison of different relevant aspects, the fourth architecture will take into account to develop the final application and its deployment on the cloud (see further sections), the highest trade-off obtained between train and test accuracies, meaning that the less overfitting. Another reason of using this implementation is the latency needed to perform the predictions. With the structure chosen, the computational cost has been considerably decreased. Therefore, when applied this model to real-time systems there won't be any serious time reduction.

CHAPTER 5

Emotion Analysis Desktop Application

This chapter describes a real-time application developed to test the model using the information of the webcam .

5.1 Introduction

In this chapter a Real-Time facial emotion recognition application is described. Before going into details few constraints must be respected:

- *Real time processing* . The principal aim of this application is to analyze real time information, that means processing video captured instantaneously. Therefore, this supposes a major challenge for emotion recognition systems where the analysis to be done requires complex techniques that often spend too much time on obtaining the final results. The architecture proposed will integrate real time emotion recognition by mean of face expressions analysis. The real time information will be the one captured from the laptop's webcam, so the model will process each of the frames of the video captured predicting the emotion of the face detected. Thus, further Machine Learning techniques to detect faces will be implemented, in order to retrieve the information that will feed the Deep Learning algorithm.

- *Accuracy.* When analyzing and detecting emotions, the accuracy obtained by the application results a crucial factor. In this project, the correct behavior of the system depends on the correct recognition of emotions. It is important to point out the major effort made by improving this task in the architecture design. The accuracy of the emotion recognition depends entirely on the model.

The application will be focused on these two issues. The model used in this application could change if required. Nevertheless, the principal aim is to test the model described in chapter 4 in a practical way by developing a functional application. To accomplish this, the service will be programmed entirely with **Python**, making use of the library **OpenCv**. The important tools or technologies used for this application are defined:

First of all, as explained in section 3.2, OpenCV provides many functionalities to perform a real time image processing. Moreover, it also allow us to start a video streaming with respect to the laptop webcam. In the following sections this processing will be thoroughly explained, but in a nutshell, as it will be used in order to "transform" the original frame captured from the webcam into a similar image to the ones used to train the Neural Network. Furthermore, there are many Computer Vision algorithms already implemented, so that complex operations such as objects detection, are easily implementable, using few lines of code.

Secondly, the decision made of using Python as the tool to develop this application was held by the use of this to implement the Deep Learning algorithm, so that it can ease the integration of such algorithm with the functionalities provided by OpenCV library.

Furthermore, with Python support, this library provides several features resulting particularly interesting in this thesis, such as image processing, video analysis, and an easy-to-use interface to video capturing and video codecs.

Scikit-learn[1]. Is an open source library with the BSD license that contains various Machine Learning algorithms such as classifications, regression and clustering ones, which interoperates with two important Python libraries: NumPy and SciPy. The first one is a numerical library that provides with many mathematical functions implemented, and the second one is the direct implementation of these algorithms mentioned above. Thus, Scikit-learn provides tools for joining features, converting features and labels into matrices and classifiers.

Finallly, **H5PY**[2], which is a Pythonic interface to the HDF5 binary data format, and It lets you store huge amounts of numerical data, and easily manipulate that data from NumPy. HDF5 is a data model, library, and file format that support widely-used standard binary format, designed for, among others, flexible and efficient I/O. Therefore, this library will be used to access the data of the Deep Learning model stored in HDF5 format.

[1] `http://scikit-learn.org/stable/`
[2] `https://www.h5py.org/`

5.2 Architecture

In this section the architecture of the application designed will be fully detailed. As mentioned before, this app is implemented with Python, in order to maximize the compatibility of the modules with any current computer. Furthermore, the main application scripts are supported by other submodules or scripts containing each of the needed functions, so that the code is more ordered in order to ease the understanding. All the modules are represented in Figure 5.1

The main goal of this is the extraction of users' emotions by mean of real time recognition of facial expression, carried out with Convolutional Neural Networks (CNN). Thus, different functions are provided to enable the access to the webcam information in which the user information will be presented, for the further process and face detection, that will be the input of the model.

Once an overview has been given, in the following subsections, different submodules are detailed, explaining their main features and design aspects.

The principal module (*app.py*) is divided into two submodules: video capture functions and face inference; each of them with different functions that help to achieve the goal of the application: facial emotion recognition.

5.2.1 Video information

The principal characteristic of this program is the real time information, which in this case will be the information captured from the user. The way to overcome this is using the information captured by the webcam of the laptop as said before.

In Figure 5.2, the flow followed by the information is represented. The mechanism implemented is based on OpenCV functions that allow us to open a connection with the webcam. The first step is to get the video streaming by mean of the creation of an OpenCV *VideoCapture* object as detailed in Listing 5.1. Then a pop-up is launched to show the video captured depending on the variable *Show_camera* variable, which its main purpose is give to the user some freedom of configuration.

Listing 5.1: Function to initialize VideCapture object using OpenCV library

```
if video_settings.show_cam:
  cv2.namedWindow(video_settings.webcam_window_name)
  video_capture = cv2.VideoCapture(0)
  return video_capture
```

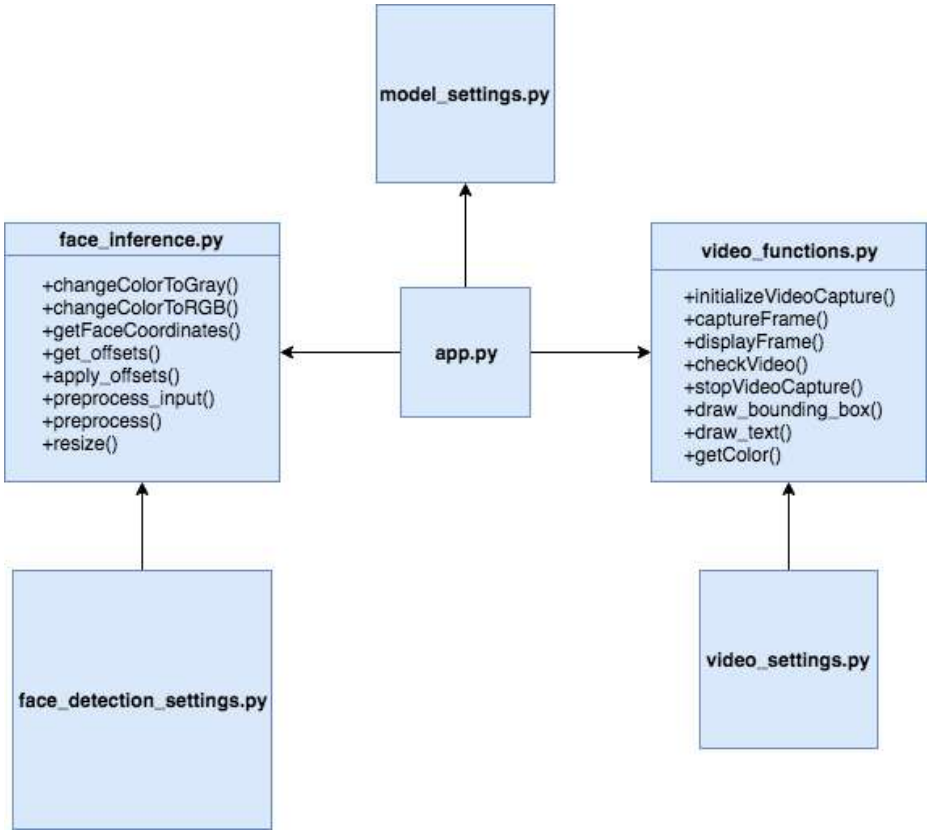

Figure 5.1: Python application modules architecture

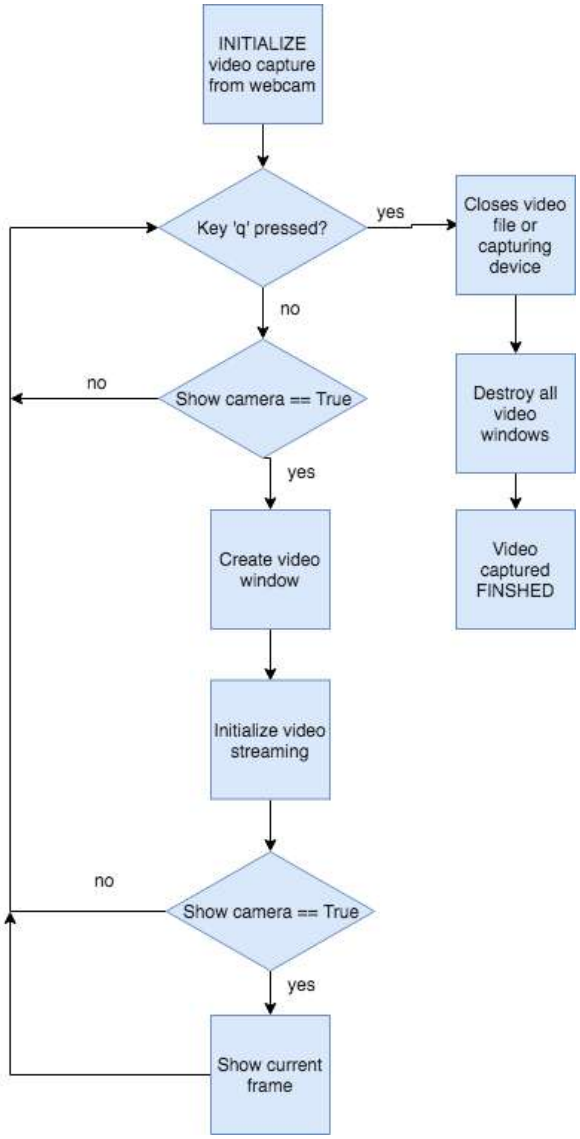

Figure 5.2: Flow chart of video capture from webcam

Finally, after created the Video Capture object, it is possible to capture frame by frame using the read method of the VideoCapture object (Listing 5.2), and show it in the pop-up launched previously, creating a streaming with the camera.

Listing 5.2: Capturing current frame

```
ret, bgr_image = video_capture.read()
```

Finally, the way to close the window opened to show the video streaming, is by pressing the key 'q' or stoping it from the terminal where it was launched.

Once the way to get the user information is overcome, some preprocessing must be done

in order to filter only the useful information to predict the emotion from facial expressions.

5.2.2 Face inference and preprocessing

Once the video streaming has been stablished and configured, the face detection phase takes place. To accomplish it a preprocessing of the image has to be done in order to retrieve the face from the frame as a whole, making use of different OpenCV functions. This face will be the input of the model trained in previous chapters. An overview of the main steps in this process process is shown in Figure 5.3, which is not straightforward, so details detailed will be given hereafter.

The main purpose of this face detection and preprocessing of the image is to obtain an image similar to the ones used to train the model, so that it can perform a proper prediction of the emotion from the face detected. Therefore, first of all the program takes each of the frames contained in the video, which are represented in RGB scale, and convert this scale to a Gray one, in which there are only values within black and white range. This scale ease the preprocessing making it much faster. Then, an equalization of the histogram of the image is done. The histogram is nothing but a representation of the intensity distribution of an image, so the equalization is done to avoid peaks in certain values that can cause some problems in the prediction. In addition this equalization improves the contrast in an image, in order to stretch out the intensity range and facilitate the face detection. Finally after the equalization of the image, the **face detection** itself takes place. The way to do so, is implementing an algorithm based on **cascade classifiers** as explained at the end of the section 2.2.1. Fortunately, in OpenCV there are many pre-trained classifiers for faces, eyes, mouthes, etc. In this program this set of classifiers is loaded from a .xml file, haardcascade_frontalface_default.xml [3]) which is a stump-based 24x24 discrete adaboost frontal face detector, and contains a ton of features that have been decided as belonging to a front-facing face. Thus, after loading the classifiers, the OpenCV function *detectMultiscale* is used to retrieve the coordinates of all the faces appearing at the video. The code implemented to detect faces is presented in Listing 5.3.

Listing 5.3: Function used to detect faces using multiscale filters

```
def getFaceCoordinates(frame):

    face_cascade = cv2.CascadeClassifier(face_detection_settings.
        cascade_path)
```

[3] https://github.com/opencv/opencv/blob/master/data/haarcascades/haarcascade_frontalface_default.xml

```
    img_equalized = cv2.equalizeHist(frame)

    faces = face_cascade.detectMultiScale(img_equalized,
        scaleFactor  = face_detection_settings.scale_factor,
        minNeighbors = face_detection_settings.min_neighbors,
        minSize      = face_detection_settings.min_size,
        flags        = cv2.CASCADE_SCALE_IMAGE
    )
    return faces
```

The second part of this process consists in **preprocessing** each of the faces extracted from the original frame. As the face retrieved by the function implemented in OpenCV, is a bit smaller than the ones used to train the model, specially in terms of height, some offsets must be applied. In other words, the face area is heightened or augmented in both axis corresponding to some factors. After this, the face is resized to 48x48 (size of the images used to train the model), and converted to Float32. This is detailed in Listing 5.4.

Listing 5.4: Preprocessing the image

```
gray_face = face_inference.apply_offsets(gray_image,face_coordinates)

try:
  gray_face = face_inference.resize(gray_face, (emotion_target_size))
except:
  continue
```

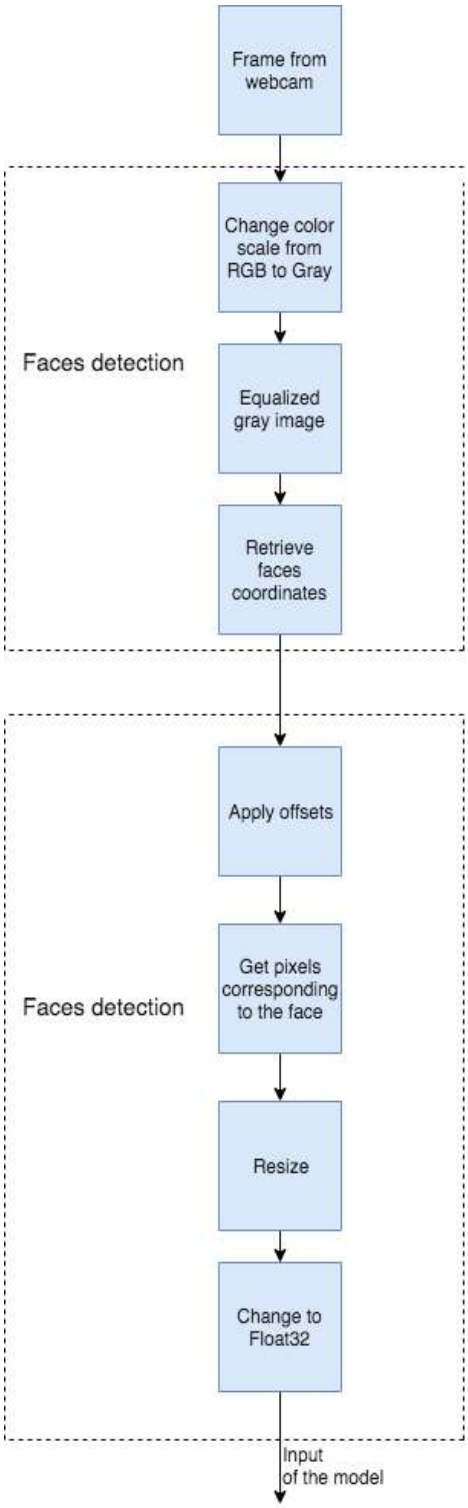

Figure 5.3: Flow chart of face detection and image preprocessing from video captured

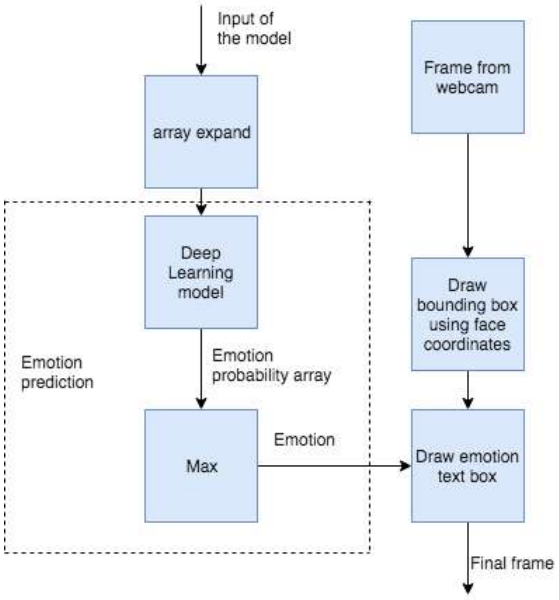

Figure 5.4: Flow chart of emotion prediction and drawing of bounding box and emotion text

5.2.3 Emotion prediction and face bounding

At this point a face has been extracted and isolated from the original frame, so the next step is to feed the Deep Learning algorithm to perform the emotion prediction. To do so, Keras library (see section 3.1.2) gives some useful functions to *load* the weights of a pre-trained model. In addition, there is also *predict* function, to perform the prediction itself. The process chart can be seen in Figure 5.4.

Nevertheless, before carrying out this prediction, another dimension modification must be done in order to adjust it to the one expected by the algorithm, as it was trained using Python arrays, and at this point a matrix (the way of representing images in Python) is available. The input should be 4-d, with the 1st dimension used to enumerate the samples. Thus, as the matrix has only two dimensions, two more dimensions will be added by mean of *expand_dims* function of NumPy library (Listing 5.5).

Listing 5.5: Image expansion

```
gray_face  = np.expand_dims(gray_face, axis = 0)
gray_face  = np.expand_dims(gray_face, axis = -1)
```

Once the **prediction** has been carried through, the result obtained is an array containing the probabilities for each emotion, thereby the emotion with maximum value is extracted. The code is detailed in Listing 5.6.

Listing 5.6: Performing emotion prediction

```
emotion_prediction = emotion_classifier.predict(gray_face)[0]
emotion_label_arg = np.argmax(emotion_prediction)
emotion_text = emotion_labels[emotion_label_arg]
```

The final step consist on drawing a rectangle **bounding** the face, with text-box above containing the emotion predicted formerly. Furthermore, the color of those elements depends on each emotion, which are showed in Table 5.1. It is important to highlight that these drawings are done in the original frame captured by the webcam, with original colors as well. In other words, the bounding box and emotion text are introduced in the frame previous to all the preprocessing, the one obtained at the end of section 5.2.1.

Emotion	**Color code (R, G ,B)**
Happiness	(255, 255, 0) = Yellow
Surprise	(0, 255, 255) = Light blue
Anger	(255, 0, 0) = Red
Sadness	(0, 0, 255) = Blue
Neutral and Fear	(0, 255, 0) = Green

Table 5.1: Color code to represent the face bounding rectangle and emotion text

5.3 Case study

In this section a real example using images captured with the computer used in the implementation of the desktop application will be shown. As mentioned previously, the main goal of this app is to process and analyze real-time data captured with a webcam. Therefore, four emotion detections will be presented, in order to visualize principally, colors used in each emotion.

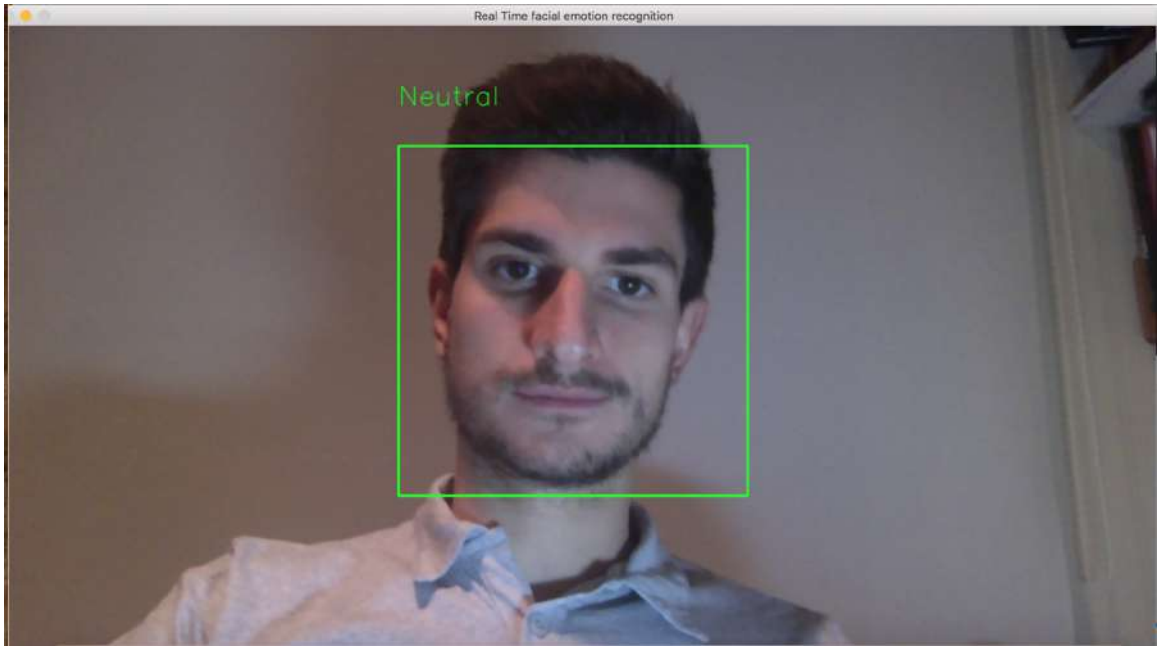

Figure 5.5: Neutral emotion detection with desktop app

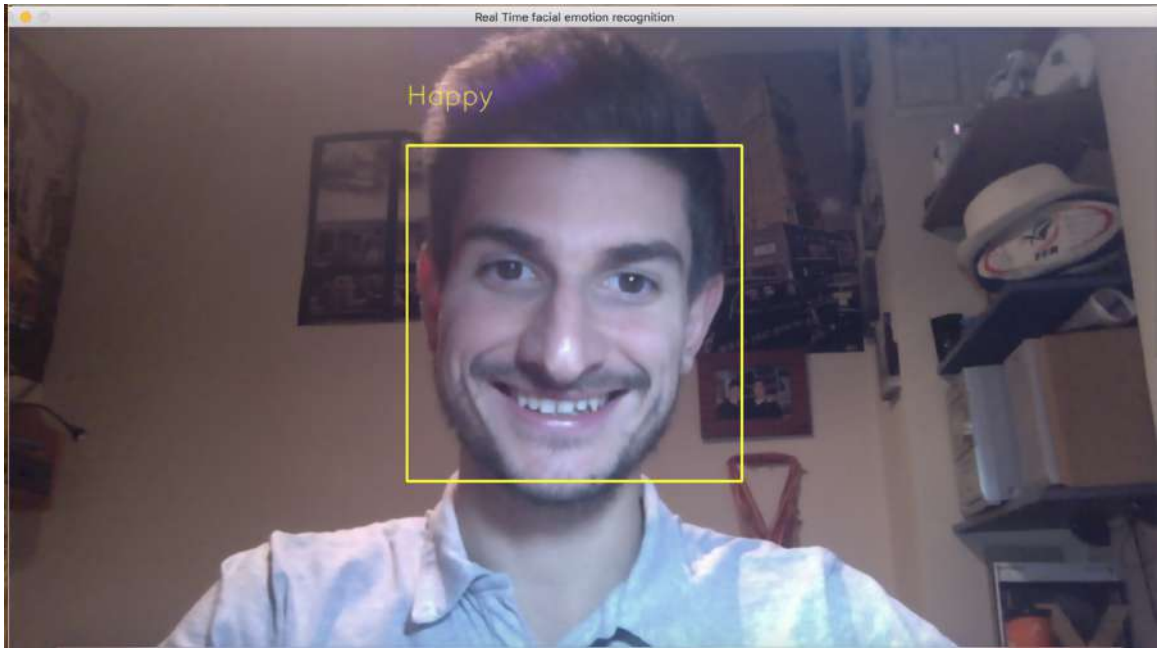

Figure 5.6: Happy emotion detection with desktop app

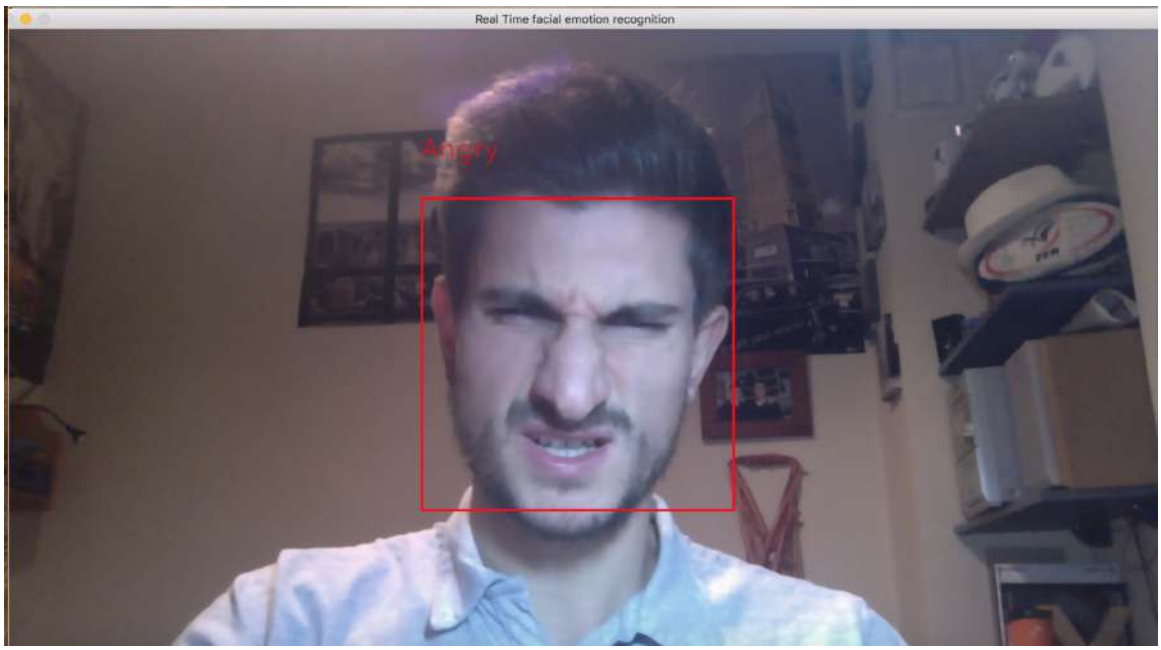

Figure 5.7: Anger emotion detection with desktop app

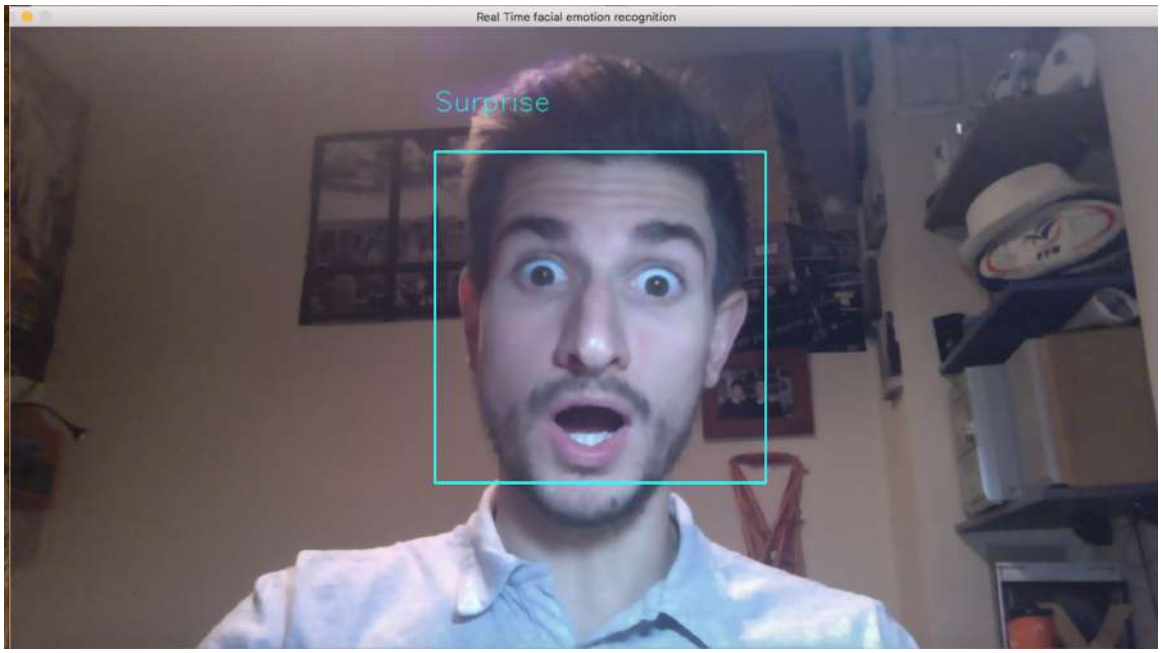

Figure 5.8: Surprise emotion detection with desktop app

5.4 Conclusion

This application has fulfilled the proposed objectives, which can be summarize in one sentence: **Develop a real-time application to get users' emotions from faces expressions**.

- Real time: this goal has been fulfilled with the use of OpenCv library, which allow us to capture webcam frames.
- Emotion prediction: this has done making use of the Neural Network model trained in chapter 4, and loaded using Keras functionalities. However, this model could be a different one, if a better accuracy is achieved with another architecture.

CHAPTER 6

Emotion Analysis Web Application

This chapter will describe the service created to deploy in the cloud an application for facial emotion recognition in real time. The main objective of this app is to deploy the facial emotion recognition model in the cloud, so that it can be accessible from every device.

6.1 Introduction

The previous chapter has introduced a Python application to detect emotions from facial expressions, which should be deployed in a local laptop. However, nowadays every application should be cloud-deployable. Therefore, after implementing that application locally, the need to deploy it in the cloud arose, so the application that will be explained in details in following subsections aims to simulate the same process deployed in the cloud.

As the previous application was implemented in Python, the first impulse was to build an application using the *Flask* Framework over Python, for a greater compatibility together with the reason that a proper domain of Python language was acquired during the mentioned implementation. Nevertheless, after thinking thoroughly the global architecture the final decision was to use **Node.js** (see section 3.3) with Express Framework (see section 3.4).

This decision was mainly propitiated by the preprocessing that has to be done in the image captured with the webcam. As there are many mathematical operations (resize, expand, change scale, etc), and also a Deep Learning model is loaded and computed, for latency reasons, it is better if those are carried out on the server side. Furthermore, thinking about storage resources, the browser must download many images and a model, that will overload user's resources.

Thereby, the way of communication offered by NodeJS is based on **websockets**. In the following sections this concept will be explained and how it has been implemented in the application. Nevertheless, before going into details about websocket, it is better to have an overview of the architecture of the application.

6.2 Architecture

The architecture commonly used in software implementations is the one called MVC (model, view, controller). As mentioned before the language chosen to implement this application is Javascript, using the framework NodeJS and Express.

There are many different ways of organizing files within the same model. Corresponding to different criteria, one is more suitable than the other. The main criteria is the complexity of the project. The greater the complexity of the project is, the more refined the structure of the code should be, in order to ease the maintenance.

For projects with little source code and more simplicity, it is more appropriate to use a linear data structuring. In this way all maintaining and documenting tasks are simplified.

For this project a **vertical structuring** has been chosen. Even thought this type of structuring is used principally in bigger projects, it is more intuitive, since it is based on the distribution of the modules depending on the app process in which they take place.

Furthermore, it is intuitive due to the splitting of the scripts depending on the functionality. In other words, all the scripts run on the client side, are under the client folder, and the same with the server. In following sections, more details about this implementation are given.

To sum up, the app architecture chosen has been organized in modules as shown in Figure 6.1, with a folder distribution based on the functionality of each of these modules.

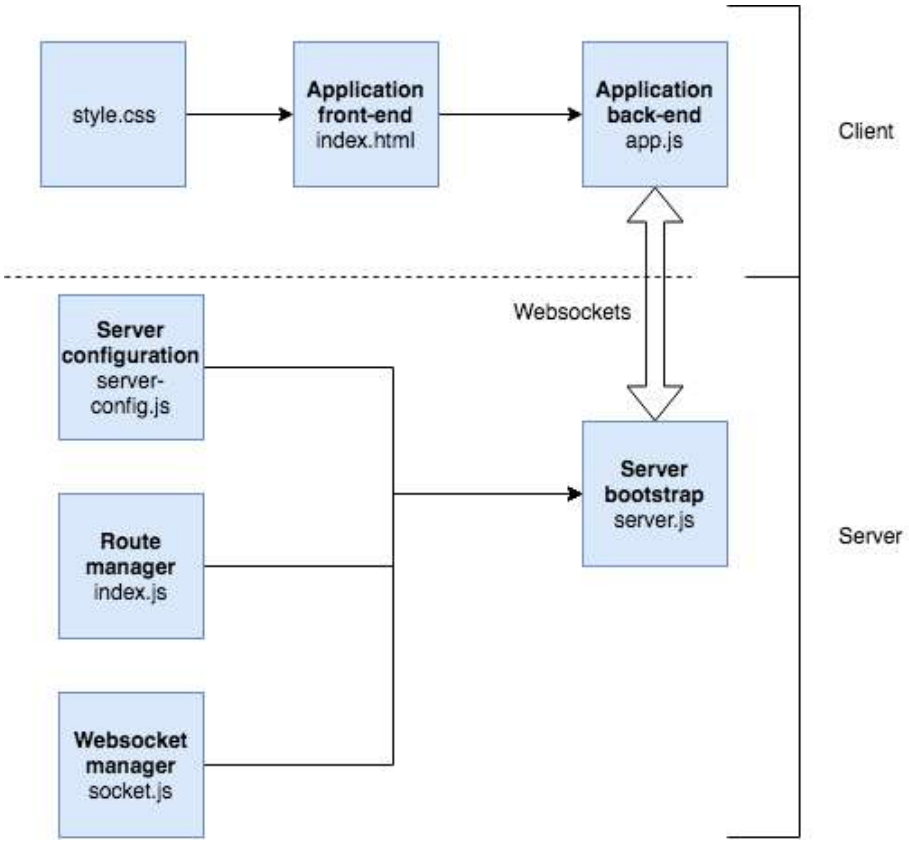

Figure 6.1: Module diagram of the NodeJS application

6.3 Client side

In this section the details about the information shown in the client are explained. Regarding the architecture explained in previous section, client side corresponds to the front-end of the application. This means that the files explained in this section will be in charge of showing the information processed in the back-end, the server.

Therefore three different files can be distinguish. In almost every front-end development there are three different types of files in order to create a dynamic view. In this project the names of these files are: *index.html, styles.css* and *app.js*.

Therefore, the file *app.js* contains all the logic of the front-end. In this file the websocket connection from the client is established, and thus, the one in charge of sending to the server the frames to be processed. The flow-chart of this operation is represented in Figure 6.2.

First of all, one the webpage is loaded the app asks the user for **permission** to use a multimedia device such as a camera or microphone. This is done with the function *getUserMedia*. If the user allows it, then the information retrieved is shown through a video element, thus the **video streaming** has begun. After this the program is going to do the sending of each of the frames. A periodical function is started, in which every 0,7s all the code inside is executed.

The code inside that function, performs a **screenshot** of the frame in each moment using a canvas element. Finally, this frame is sent to the server over the *frame* event (see section 6.5) in a socket, after encapsulating the image information in a buffer with *.png* format.

On the other hand, the frame with the emotion computed must be shown. This is done in the back-end, and sent to the front-end via websocket, as explained in followings sections. Thus, in the logic part of the client-side it is implemented the reception of this processed frame. Once the frame is received, it is loaded in a canvas element just besides the video element in which the original streaming is taking place.

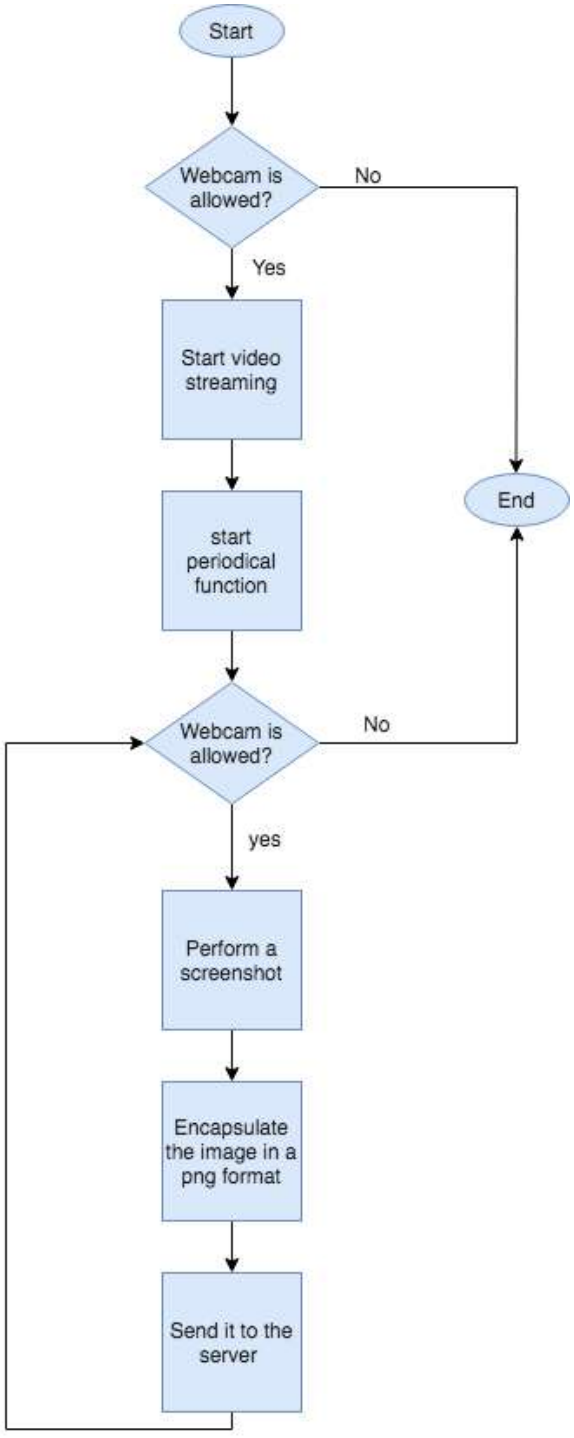

Figure 6.2: Flow-chart of webcam streaming and frame sending to the server

6.4 Server side

In this section all the processes carried out on the server side will be explained. This section is the core of the app, due to the fact that all the image preprocessing and predictions are done on the server. As explained before, the main reason of doing this, is the availability of more resources on this side to accomplish all the operations needed to obtain the desired results.

First of all, the folder organization is going to be shown, in order to proceed to explain the details. As shown in Figure 6.3, one principal folder called *lib* together with some files (package.json and server.js) are under the principal root.

The file *package.json*, as explained in section 3.4, contains all the dependencies with external packages needed. Specifically there are four dependencies:

- **express**: version 4.10.1 or upper. This package allows us to access to all the functionalities provided by the Express. Hereafter, the most important ones will be explained in more details.

- **KerasJS** [9]: version 1.0.3 or upper. It is a library used to load Keras model in NodeJS. In other words, to load the model trained in previous sections to predict emotions.

- **morgan**[1]: version 1.4.1 or upper. It is a HTTP request logger middleware for NodeJS. This provides different options to overcome error debugging. For example, changing outputs colors to ease the identification.

- **socket.io**[2]: version 1.2.1 or upper enables real-time, bidirectional and event-based communication.It works on every platform, browser or device, focusing equally on reliability and speed.

6.4.1 Setting up

In this section all the process to configure the back-end part is going to be explained. Some files from the folder representation shown in Figure 6.3, will be detailed in order to further understand this configuration.

The file *server.js*, under the main folder *server*, contains all the functions needed to perform this configuration. These functions are mainly the ones provided by Express module. Specially to define the middlewares in charge of attending requests, and even to associate

[1] `https://www.npmjs.com/package/morgan`
[2] `https://github.com/socketio/socket.io`

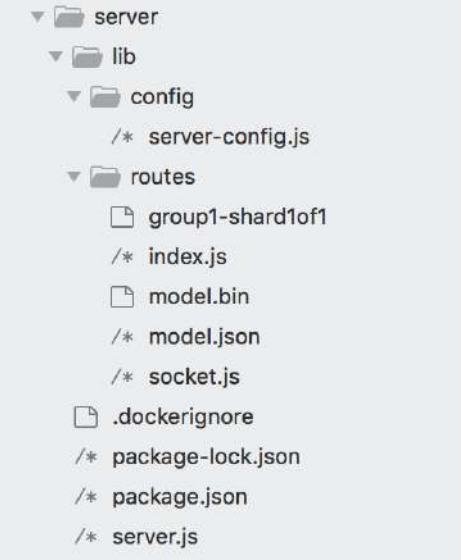

Figure 6.3: Vertical structuring of the server side

each middleware a specific URL from which perform the request. The setting up flow chart is represented in Figure 6.4.

Corresponding to these settings, first of all, a reference is defined called *app* , as detailed in Listing 6.1, from which all the functionalities already mentioned of Express can be used.

Listing 6.1: Creation of app reference

```
// configuration files
var configServer = require('./lib/config/server-config');

// app parameters
var app = express();
app.set('port', configServer.httpPort);
```

This is an instance or reference associated to Express module, used to configure the web app or to obtain characteristics of it. The script *config/server-config.js* contains different variables (Listing 6.2). The reason of isolating them is to follow some good programming techniques that ease app understanding.

Listing 6.2: Value of different variables included in server-config.js

```
var path = require('path')

module.exports = {
  httpPort: 8080,
```

```
  staticFolder: path.join(__dirname + '/../../../client')
};
```

The **port** used in this server configuration is **8080** as can be seen.
Regarding the *app* reference, the more common methods used in this configuration are:

- *app.set(name, value)*: this method is used to initialize the environment variables.
- *app.get(name)*: this method is used to obtain the value of the variables defined with the previous method.
- *app.use([path], callback)*: this method is used to create a middleware, in order to manage HTPP requests. If an URL is added in the path parameter, the middleware is associated to it, and this only will response to requests coming from this URL.

In this application only two middlewares are created using *app.use* function, as shown in Listing 6.3:

Listing 6.3: Creation of NodeJs middlewares

```
app.use(express.static(configServer.staticFolder));
app.use(morgan('dev'));
```

Firstly, a middleware already included in Express, and is responsible for the static asset service of an application. There must be an argument root which specifies the root directory from which the static service is done, which is included in *config/server-config.js*. After this, the creation of a route manager is implemented, as specified in Listing 6.4.

Listing 6.4: Importation of the route manager

```
// serve index
require('./lib/routes').serveIndex(app, configServer.staticFolder);
```

This is done in order to specify the root directory of the static service. In other words, the path to the client-side part is imported, specifically the path to the *index.html* file explained before.

In the script *route/index.js* a res object is created in order to return the HTML file whose name is specified in the *res.send* argument. This code is presented in Listing 6.5.

Listing 6.5: serveIndex function in index.js

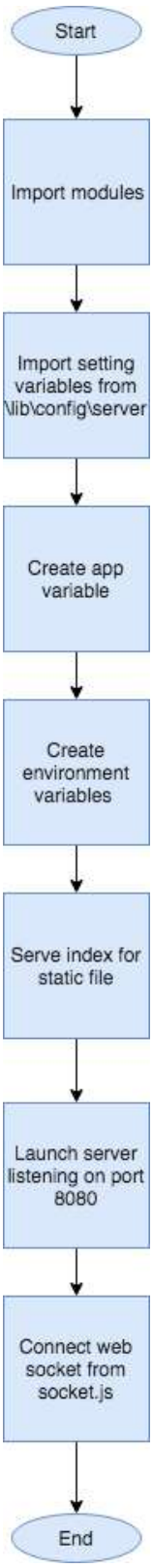

Figure 6.4: Flow chart of the server setting-up

```
exports.serveIndex = function (app, staticFolder) {
  app.get('*', function (req, res) {
    res.sendFile('index.html', { root: staticFolder });
  });};
```

In order to better understand this procedure, a brief theoretical explanation is going to be given.

- the object *res* is the encapsulation of the response module of NodeJS. It stores the information to be sent as a response of the last request done.
- *res.send* is a method which generates a response from the server side, in which the information to be sent can be either an array, or a buffer.

Secondly, another middleware is created, related to the Morgan package. As said before, this is a logger middleware that provides different functionalities to ease the debugging. Specifically, in this project a pre-defined format, called *dev* has been used, which concise output colored by response status for development use.

Then the server itself is launched listening on the port specified. To do so, firstly a server object is created using a function provided by the HTTP packet of NodeJS. Secondly, the server starts to listen at the **port** specified, which in this project is **8080**. The code implemented to do so is shown in Listing 6.6. Nevertheless, if the app is launched using Docker containers it will change. In further sections this is explained.

Listing 6.6: HTTP server creation

```
// HTTP server
var server = http.createServer(app);
server.listen(app.get('port'), function () {
  console.log('HTTP server listening on port ' + app.get('port'));
});
```

Finally, the websocket connection is established as detailed in Listing 6.7. This connection will be explained thoroughly in the next section.

Listing 6.7: Websocket connection establishment

```
// WebSocket server
var io = require('socket.io')(server);
io.on('connection', require('./lib/routes/socket'));
```

6.5 Websockets implementation

The way in which websockets are implemented in this project is the following. First of all, the connection is opened from the server side, through the function *on* available in socket.io library. In the argument of this function the file *server/routes/socket.js* is passed. In this file all the logic of the server socket is implemented.

On the other hand, in the client a step-up must be done as well. In this case, all the app logic is written in the file *client/app.js* as shown in Figure 6.2. In this file, first, in order to carry out the websocket connection, the socket must be connected to the server URL. In the case that the server was launched locally, it would only be enough to put as an argument *'http: // localhost'* . However, as Docker is being used, the argument will be the URL of the server. This is done as shown in Listing 6.8.

Listing 6.8: Websocket connection establishment in client side

```
//URL of the host to be connected
 var host = window.location.hostname;
 var socket = io.connect('http://'+host);
```

Concerning the communication itself, two different events has been implemented. To create events using websocket, you must specify in the argument of *emit* function the namespace of the event. As said, before, it has been decided to distinguish two different events:

- 'frame': this is the event used by the client to send the frame or image captured by the webcam.
- 'response': this is the event used by the server to send the response or the frame after the preprocessing and emotion prediction.

The main reason of this distinction is to ease the readability of the code, and to avoid collisions between each other.

6.6 Emotion prediction

6.6.1 Introduction

In this section the process followed to accomplish the emotion prediction is going to be explained. This process will be similar to the one explained in sections 5.2.2 and 5.2.3 .

First of all, before going into details, a brief introduction to technologies needed to overcome the problem will be given. To perform all the image preprocessing, in previous sections, OpenCV library over python was used. In this case, as Node.js is being used, all the initial efforts went to the search for a possible solution, as the library does not provide support for Javascript. Therefore, after some research two libraries came up: **node-opencv**[6] and **opencv4nodejs** [34].

- node-opencv: this is a NodeJS library created by Peter Braden, which provides bindings for NodeJS. Currently people are using node-opencv to fly control quadrocoptors, detect faces from webcam images and annotate video streams. The GitHub repository of this library has more than 3500 starts and 640 forks.

- opencv4nodejs: this is a library created by Vincent Muhler, which is an asynchronous OpenCV 3.x NodeJS bindings with JavaScript and TypeScript API. The GitHub repository of this library has more than 1900 starts and 200 forks.

At first glance both libraries seem to offer the same functionalities. Nevertheless, the first one seems to be better as it has more stars and forks, denoting higher popularity. This is due to the fact that the first one is older. Furthermore, being an older library can lead to a deprecation, thus the frequency of commits plays an important role in the final decision. From the commits viewpoint the second library seems to be better, as this frequency is higher. Finally, the documentation provided by the second author is enormously better, but there were more examples provided by the first one.

In spite of all these arguments, both libraries have been tried in order to try possible advantages and drawbacks of each of them. After the trial-and-error phase, indeed with **opencv4nodejs** more efficient results were obtained.

Furthermore, there was another problem with the cloud-deployment of the Python app using NodeJS needed to be solved. This problem has to do with the interaction with the Deep Learning model. In Python app, the function *load* provided by Keras was used. Therefore, somehow it was necessary to find this load function using NodeJS. The solution was making used of **KerasJS**, which is a straightforward binding of Keras, in order to run Keras models in the browser, with GPU support provided by WebGL 2 [9]. In addition, it allows you load and run a pretrained model on server side. The only requirement to do so,

is that the model must be saved in a .bin format, instead of .h5 as before. This library will be called in an asynchronous way.

6.6.2 Process

In this section all the process followed to perform the prediction itself will be explained. In Figure 6.5 the flow-chart is represented.

This process starts with the reception of the frame sent with a socket by the client captured through the webcam, over the *frame* event. The received frame is in base 64, thus this format will be used all over the preprocess. Then it is needed to convert the data to a buffer in order to proceed to de decoding of the image, in *.png* format, through *imdecode* function provided by the **opencv4nodejs** library, obtaining a Mat representation.

Once Mat representation is obtained, all the functionalities in the library to preprocess the image. The next step is the **face detection**. The method or technique implemented is the same as explained in Section 5.2.2, consisting on a classifier composed of several **classifiers in cascade**. Then, after retrieving faces coordinates, for each face, the offsets are applied and then the face is cropped from the frame.

At this point the face region is split from the original frame, so only on this region the **color scale** is changed to **gray**, the **histogram** is **equalized** and then **resized to 48x48** as the images used to train the model. Finally, the image is converted to an array and converted to Float 32.

After all this preprocessing, the face image is ready to feed the model, in order to compute the prediction. The way implemented to pass the image to the model, is asynchronous, due to the fact that asynchronous workflow processes might process faster. To accomplish this, as mentioned before, **Keras-js** library is used. As in section 5.2.3, the result obtained is an array containing the probabilities for each emotion, thereby the emotion with maximum value is extracted.

Once the emotion is retrieved, **face bounding** with a rectangle takes place. Using functionalities of Opencv for NodeJS, a rectangle around the face is drawn. The perimeter of this rectangle, corresponds to the region without the offsets, applied before, covering strictly the face. Then, the **emotion** is output in **text format**, just above this rectangle, using a function of the library as well.

In addition, in this app the color code used previously is followed. That means that, the rectangle and color text will depend on each emotion. This code is the one explained in Table 5.1.

Conclusively the image is **encoded** in *.png* format, in base 64. Then it is **sent** to the client via *response* event.

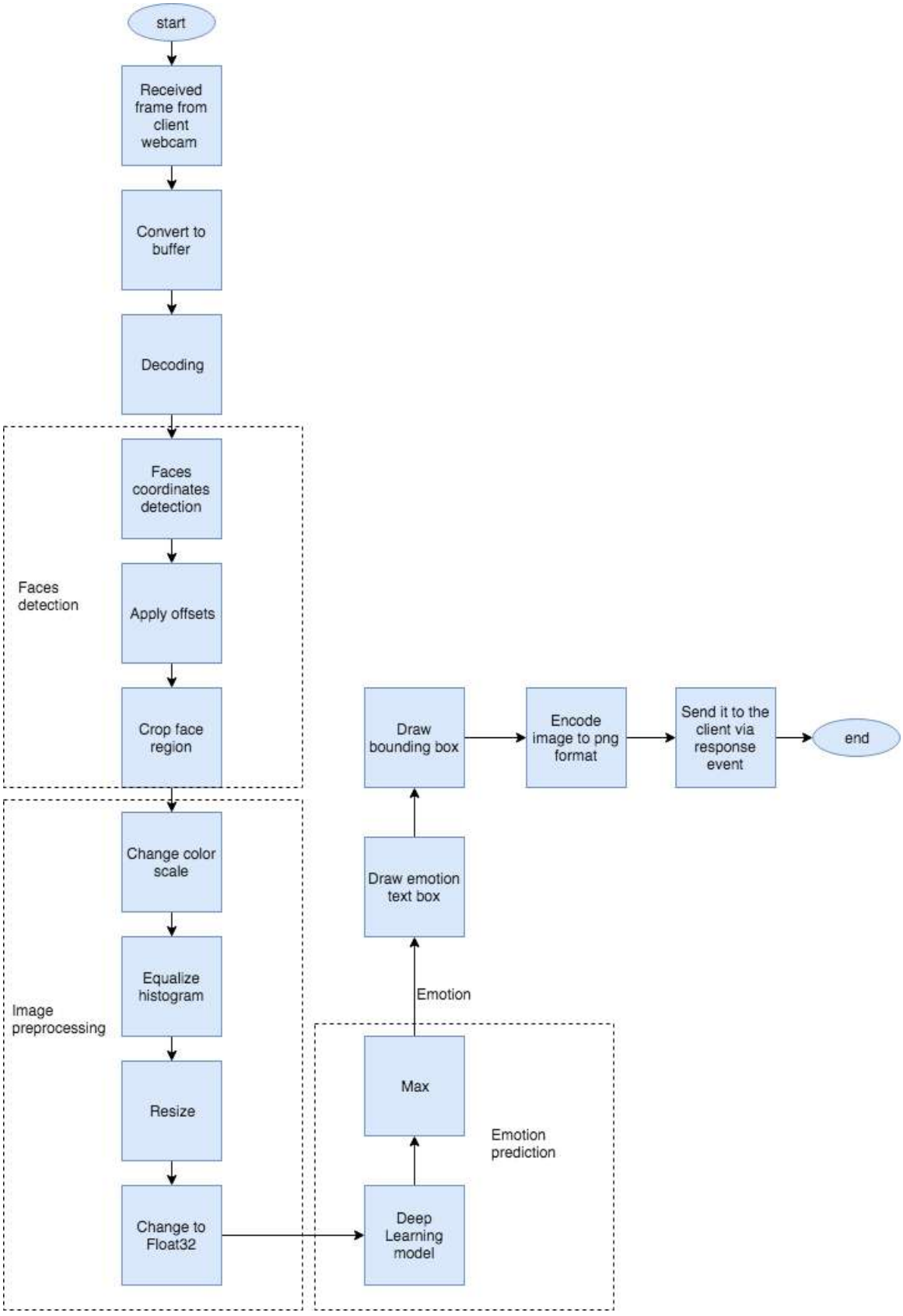

Figure 6.5: Flow chart of emotion prediction carried out in NodeJS app

6.7 Case study

In this section some visual real-examples are shown. The face recognizer GUI shows two screens, one with the real captured image, and the other shows the real captured image with a bounding box in each face detected, with the correspondent color of each emotion. Colors used to represent each emotion are specified in Table 6.1. Codes have changed with respect to the desktop app. In the library used to get the bindings of Opencv for NodeJS (opencv4nodejs), color codes are expressed in a different format, (B, G, R) instead of the traditional one (R. G. B).

It can be seen in Figures 6.6 and 6.7, two emotions predicted, happy and anger respectively. Moreover, it can be appreciated that the app is running on a local server, as the URL shown is **localhost:8080**.

Emotion	**Color code (B, G ,R)**
Happiness	(0, 255, 255) = Yellow
Surprise	(255, 255, 0) = Light blue
Anger	(0, 0, 255) = Red
Sadness	(255, 0, 0) = Blue
Neutral and Fear	(0, 255, 0) = Green

Table 6.1: Color code to represent the face bounding rectangle and emotion text

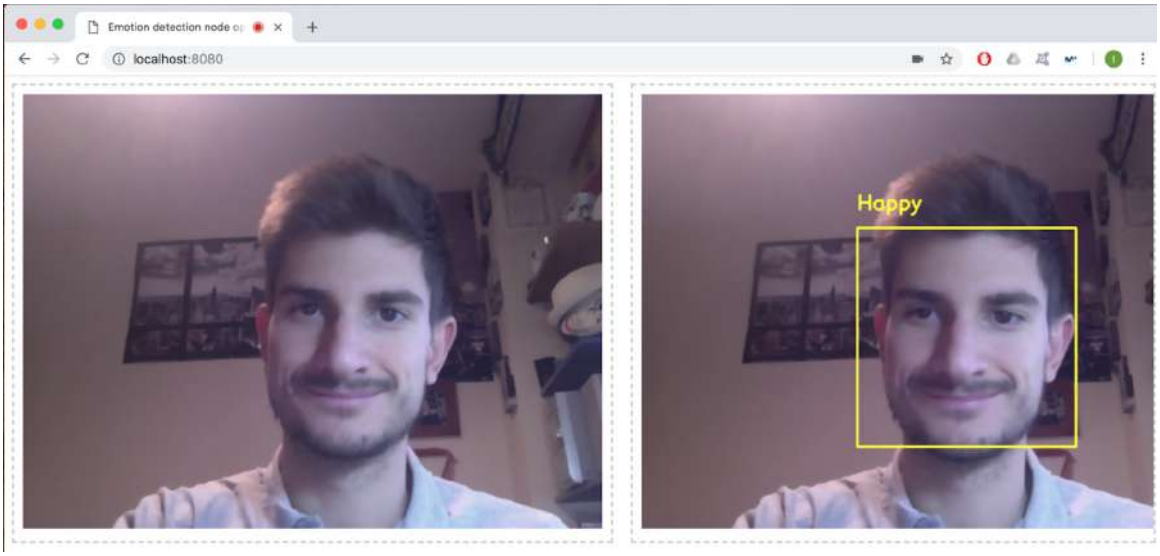

Figure 6.6: Happy emotion detection with web app

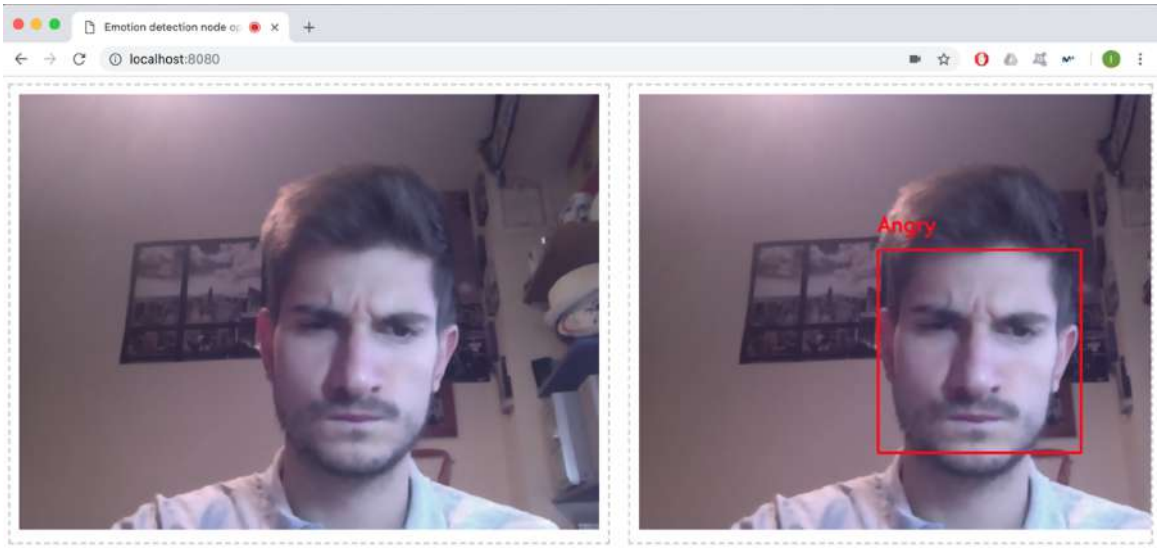

Figure 6.7: Anger emotion detection with web app

6.8 Conclusion

This application has fulfilled the proposed objectives, which can be summarize in one sentence: **Develop a real-time cloud-deployable application to get users' emotions from faces expressions**.

- Real time: this goal has been fulfilled with the use of websockets, which allow us to set up a bidirectional client-server communication, in order to process on the server frames captured with the client webcam.
- Cloud-deployable: this has been achieved with the use of a Popular web-programing language such as NodeJS, with the use of Express framework. Furthermore, as the application has been embedded in a Docker container, best cloud-deployment trends has been followed.
- Emotion prediction: this has done making use of the Neural Network model trained in previous sections, and loaded on server-side.

Nevertheless, in this app the performance of the Deep Learning model has decreased. It can only detect two emotions, the ones shown in section 6.7. This is due to the fact that a compression of the model has been carried out in order to load it in the NodeJS app using Keras-JS. Thus, all the weights have been rounded, changing completely the behavior of the algorithm. In the following chapter a possible solution to this problem is going to be detailed.

CHAPTER 7

Conclusions

This chapter will describe the achieved goals done by the master thesis following some the key points developed in the project.

7.1 Achieved Goals

The project has fulfilled the proposed objectives, but we must refer to each one of these objectives in to draw a general conclusion.

- **Implementation of a deep learning model capable of predicting emotions from facial expressions**. This algorithm was the one with the highest trade-off obtained between train and test accuracies, meaning that the less overfitting. Furthermore, with the structure implemented, the computational cost has been considerably decreased. Therefore, in a real-time system the latency is almost negligible.

- **Development of a a real-time application to get users' emotions from faces expressions**. Real-time has been fulfilled with the use of OpenCv library, which allow us to capture webcam frames. Emotion prediction has done making use of the Neural Network model trained in previous sections, and loaded using Keras functionalities.

- **Development a real-time cloud-deployable application to get users' emotions from faces expressions**. Real time has been fulfilled with the use of websockets, which allow us to set up a bidirectional client-server communication, in order to process on the server frames captured with the client webcam. Cloud-deployable has been achieved with the use of a popular web-programing language such as NodeJS, with the use of Express framework. Furthermore, as the application has been embedded in a Docker container, best cloud-deployment trends has been followed. Finally emotion prediction has done making use of the Neural Network model trained in previous sections, and loaded on server-side.

Nevertheless, there was a reduction in accuracy in the web app due to some limitations of the JavaScript implementation of Keras. To solve this, some possible solutions are given in the following section.

7.2 Future work

Once the project is finished, new lines emerge on which to continue working and implementing improvements. This is also known as future work.

To start with, some research can be done in order to try to enhance the model performance. As stated in [30], one possible solutions is the implementation of a a RNN (Recurrent Neural Network) just after the CNN in order to get some time characteristics together with the spatial characteristics learned by the CNN. Therefore, In the first part, the spatial image characteristics of the representative expression-state frames are learned using a CNN. In the second part, the temporal characteristics of the spatial feature representation in the first part are learned. This technique is also known as LSTM (Long short-term Neural Networks). LSTMs are a special kind of RNN, capable of learning long-term dependencies, this means remembering information for long periods of time. All recurrent neural networks have the form of a chain of repeating modules of neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer. LSTMs also have this chain like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four, interacting in a very special way. In [28] using this technique they could obtain an accuracy of more than 90%.

Another improvement that can be done, is related with the websockets. The main problem of this technology is the latency present in the communication. Therefore, a possible solution can be the implementation of HTTP/2.0 with push functionalities as the way of communicating both sides (client and server).

Furthermore, the library used to load and perform predictions in the cloud-deployable app with NodeJs, Keras-js, is no longer supported. By now, it was the only way to load keras models, but Google has recently released a new library for web apps and all the functionalities are being migrated to TensorFlow.js[1] . This library brings all the boundaries needed to use TensorFlow with JavaScript language. Moreover, in order to load a model using Keras-js it has to be compressed in advanced, causing some roundings in some weights of the model that reduces the accuracy. With the use of TensorFlow.js, this problem will be avoided, as it allows you to load the model directly, without any preprocessing.

Finally, in order to increase the model performance, the dataset used to train it (FER2013) can be enhanced by labelling the emotions in a different way. In FER+[2] each image has been labeled by 10 crowd-sourced taggers, which provides better quality ground truth for still image emotion than the original FER labels. Having 10 taggers for each image enables researchers to estimate an emotion probability distribution per face. This allows construct-

[1] https://js.tensorflow.org/

[2] https://github.com/Microsoft/FERPlus

Figure 7.1: Comparison between FER and FER+ labels extracted from [54]

ing algorithms that produce statistical distributions or multi-label outputs instead of the conventional single-label output, as described in [54].

APPENDIX A

Project impact

This appendix will show the Ethical, Economical, Social and Environmental impacts of this project. As it has been described in the document, the project is mainly centered in the extraction and processing of information extracted from videos, specifically people's faces. Therefore it is important to reflect on the handling of such a critical information.

A.1 Ethical impact

In this chapter all the ethical issues related to the technologies used in this project are going to be studied. As stated in [32] for each computer vision topic there are some ethical issues that must be covered.

Concerning all technologies covered in this project, the main ethical issues identified are: Espionage, Discrimination and Copyright Infringement.

Espionage is found to be the most concerning problem in the field of computer vision, as it concerns the largest amount of different computer vision applications. Espionage consist in gathering data on individuals without their permission. Thus, from the ethical viewpoint it is a serious breach in privacy. Emotion recognition from faces can be seen as a way to read people's mind or access directly to their thoughts. Therefore from the ethical standpoint it is important to preserve the privacy of individual by protecting their identity or sensitive information.

Discrimination is a another ethic topic that must be explained. There are different types of discrimination. As an example, there is a differentiation between the skin coloration and facial features related to ethnicity, or a gender differentiation. This means that, an algorithm can predict differently emotions from a male and a woman. This is also known as segmentation. To avoid this, the dataset used to train the model must contain a wide variety of face characteristics, in order to obtain a general result.

The last ethical concept, **Copyright Infringement**, has to do with the data used to train the model, specifically the source of it. It is defined to be an illegal use of intellectual property without rights or permission. The dataset used must be open source. Moreover, this problem in these databases also lies in avoiding images with scenes that might be susceptible to either copyright violations

A.2 Economical impact

The economical impact is mainly encouraged by the technology itself. It is well-known that Artificial Intelligence is a trend now, and more companies are investing on this type of solutions, such as automatizing processes. With this app, some of these processes can be automatized. For example, a certain company can implemented it in order to know the level of happiness of its employees from the emotions captured from their faces. This will reduce the number of costs caused by company abandonments, so that the unhappiness can be predicted and remedied.

A.3 Social impact

The social impact of this project could be strictly linked to the ethic aspects treated before. In may 2018 General Data Protection Regulation (GDPR) law entered into forced. It is a regulation in EU law on data protection and privacy for all individuals within the European Union (EU) and the European Economic Area (EEA). The regulation contains provisions and requirements pertaining to the processing of personal data of individuals. This requirements can be summarized in an anonymisation of the data used. This means that faces images used to train the model cannot contain the person name, or any other critical data with which the user can be identified.

In addition, regarding the ethical concept of espionage and GDPR law, there must be an allowance/acceptance of the user to treat and process the data. Concerning the desktop application, there is no acceptance pop-up implemented, due to the fact that the user is the one who triggers the app, in which this acceptance is embedded. Nevertheless, in the case of the web application, a pop-up has been implemented in order to ask for user permission to retrieve webcam information.

A.4 Environmental impact

This project does not have a clear environmental impact, except for the computing power needed for having the described systems running. The most common way to carry out those experiments is using a shared pool of resources, and so the host machines are being use for multiple applications at the same time. If those resources are allocated on a big cloud service, the whole system would be elastic with respect to demand, and so power consumption could be optimized in a way that only the needed machines would be working at any time. It is true that the power cost of a whole data center is really intensive, but the needed power to execute all the services in independent machines would be several times higher.

APPENDIX B

Project costs

This appendix describes the economic budget regarding the design and development of the project, considering material and human resources. Finally, a taxation involved in software selling is explained, as part of the final budget.

B.1 Material resources

Due to the complexity of training a deep learning model, which requires a huge number of calculations a high quality computer to accomplish this will be needed. Moreover, just in case, the web app is deployed on a server which has additional computing resources, the model can be trained there so the requisites of the computer could be relaxed.

As said before, the most resource-demanding activity is the training of the neural network.We estimate the need of 16 GB of RAM and optionally a GPU for accelerating the model training process. A computer of around 1,000 euros would be sufficient for this task, and the GPU could even double this budget. However, as all the models have been trained on Google cloud, through Google Collaboratory, this budget has been decreased.

Regarding the deployment server for the service, there isn't a exhaustive need of resources and so a common server could be used, being its price from 1,000 euros on. This server doesn't need to be fully dedicated to the deployed service, and even a contracted VPS service solution could be used. The diversity of VPS vendors is really high, but an average reasonable price could be 20 euros per month.

B.2 Human resources

In this section, the cost to carry out this project in terms of time is going to be explained. Meaning by time, is the time period required to entirely design and develop this project. In order to give a cost approximation an average Software Engineer salary will take into account.

The time estimation of this project is around 900 hours. This approximation is relying on the ECTS of this Master Thesis. As this thesis is validated with 30 ECTS, and 1 ECTS stands for 25-30 working hours, thus 900 hours is the maximum time computed. Within this amount, two differentiations must be done. One for the designed and implementation time, which can be 750 hours, and the other related to the writing of this document which can be 150. Considering that the salary of a Software Engineer is around 2.000 euros gross per month, 23 working days per month, and 8 working hour per day, the total development budget is around **8.000€**.

There would also be needed another engineer executing tasks related with the system deployment, availability and security. This person should be full time hired with the purpose of fixing any incidence happening at any time, but as this would not happen frequently, he could also be focused on similar tasks for other projects. Its associated cost would also be around 1.500 euros each month the system is up.

B.3 Taxes

Once the development of the project is accomplish a possible action to do is to sell it to another company. To do so, local regulation must be taken into account. In this case, as the software has been developed in Spain, the adjustment must be done in relation to Spanish law, specifically to the one regulating the selling of software products.

According to [14], there is a tax of 15% over the final price of the product, as regulated by the Statue 4/2008 of the Spanish law. Thus, considering the app price, the development cost, the final amount will be 9200€.

In the case of the willingness of selling it to a foreign country a law research must be done, increasing the price, and what's more, consider a double taxation.

APPENDIX C

Apps instalation

In this appendix a brief explanation will be given of how to install and use the two applications developed and explained in details previously.

C.1 Python app

This is the simplest app, and the installation is quite easy. The only requirements are the libraries needed to launch the app. In a nutshell, the principal libraries are: TensorFlow, Keras, Numpy and OpenCV. The libraries can be downloaded in a Python environment which can be created and activated with Anaconda. Once this dependencies are up, it is time to launch the app with the preferred Python IDE. To do so, just launch the file app.py

C.2 NodeJs app

C.2.1 Launching locally without Docker containers

This first way f launching the app is without the use of Docker containers. That means that, a local NodeJs server is created and running the app on it. To do so, first of all, all the node modules must be installed. Therefore, from the server directory, write the following command:

Listing C.1: Node modules installation

```
npm install
```

Once the modules have been installed, it is time to run the app. Thus, from the server directory, run the following command:

Listing C.2: Running NodeJs app

```
node server.js
```

At this point the app is up and running, so to test it, open a browser and go to **localhost:8080**

C.2.2 Launch it locally with Docker containers

As explained in previous sections, the app has been embedded in a Docker container, so a lighter app can be launch, without the need of installing the libraries directly. To run Docker containers, first of all the image must be built. In this case the Dockerfile is saved over the root, src file. From this, run the following commands to built the Docker image and to launch the app using Docker compose:

Listing C.3: Running NodeJs app using Docker containers

```
docker-compose build
docker-compose up
```

Again the app should be up and running on **localhost:8080**.
The Docker imaged created has the following characteristics:

- compressed size: 124mb
- ubuntu: 16.04
- nodejs: v9.8.0
- npm: v5.6.0
- opencv4nodejs: 3.3.1

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Shima Alizadeh and Azar Fazel. Convolutional neural networks for facial expression recognition. *CoRR*, 2016.

[3] Andrew G.Howard , Menglong Zhu, Bo Chen , Dmitry Kalenichenko,Weijun Wang , Tobias Weyand , Marco Andreetto and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, 2017.

[4] Aliaa A. A. Youssif, Wesam A. A. Asker. Automatic facial expression recognition system based on geometric and appearance features. *Computer and Information Science*, 2011.

[5] Cristopher M. Bishop. *Pattern recognition and machine learning*. Springer, 2006.

[6] Peter Braden. node-opencv, 2012.

[7] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[8] Ali Mollahosseini, David Chan, and Mohammad H. Mahoor. Going deeper in facial expression recognition using deep neural networks. *CoRR*, 2015.

[9] Leon Chen. Keras-js, 2017.

[10] François Chollet et al. Keras. `https://keras.io`, 2015.

[11] François Chollet. Xception: Deep learning with depthwise separable convolution. *CoRR*, 2016.

[12] Cortes, Corinna and Vapnik, Vladimir N. Support-vector networks. *Machine Learning*, pages 273–297, 1995.

[13] Prudhvi Raj Dachapally. Facial emotion detection using convolutional neural networks and representational autoencoder units. *CoRR*, 2016.

[14] Francisco de la Torre Diaz. *La tributacion del software en el IRNR. Algunos aspectos conflictivos*. Agencia Estatal de Administracion Tributaria, 2014.

[15] Paul Ekamn. Lie catching and micro expressions. *The Philosophy of Deception*, 2009.

[16] Paul Ekman. An argument for basic emotions. *Cognition and Emotion*, pages 169–200, 1992.

[17] Paul Ekman. Enjoyable emotions. *emotion research*, 2003.

[18] Caifeng Shan , Shaogang Gong and Peter W. McOwan. Facial expression recognition based on local binary patterns: A comprehensive study. *Image and Vision Computing*, 2009.

[19] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[20] Simon Haykin. *Neural networks and Learning machines*. Pearson, 1993.

[21] John Duchi,Elad Hazan and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 2011.

[22] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems 25 (NIPS)*, 2012.

[23] Geoff Hinton. Lecture 6e of his coursera class.

[24] M. Mirza, B. Hamner, W. Cukierski, Y. Tang, D. Thaler, D.-H. Lee, et al. I. J. Goodfellow, D. Erhan, P. L. Carrier, A. Courville. Challenges in representation learning: A report on three machine learning contests. *Neural information processing*, pages 117–124, 2013.

[25] Christian Szegedy,Vincent Vanhoucke,Sergey Ioffe and Jonathon Shlens. Rethinking the inception architecture for computer vision. *CoRR*, 2015.

[26] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, pages 448–456. JMLR.org, 2015.

[27] Alan Julian Izenman. *Modern Multivariate Statistical Techniques Regression, Classification, and Manifold Learning*. Springer, 2006.

[28] Y.M. Kim, D.H.; Baddar, W.; Jang, J.; Ro. Multi-objective based spatio-temporal feature representation learning robust to expression intensity variations for facial expression recognition. *IEEE Transactions on Affective Computing*, 2018.

[29] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, 2014.

[30] Byoung Chul Ko. A brief review of facial emotion recognition based on visual information. In *Sensors*, 2018.

[31] Ming-Hsuan Yang,David J. Kriegman and Nerendra Ahuja. Detecting faces in images : a survey. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24:34–58, 2002.

[32] Mikael Lauronen. Ethical issues in topical computer vision applications. Master's thesis, university of Jyvaskyla, 2017.

[33] Dirk Merkel. Docker: Lightweight linux containers for consistent development and deployment. *Linux J.*, 2014.

[34] Vincent Muhler. opencv4nodejs, 2017.

[35] Rosalind W. Picard. *Affective Computing.* MIT Press, 1997.

[36] Octavio Arriaga , Paul G. Ploger and Matias Valdenegro. Real-time convolutional neural networks for emotion and gender classification. *CoRR*, 2017.

[37] Arushi Raghuvanshi and Vivek Choksi. Facial expression recognition with convolutional neural networks. 2016.

[38] Kaiming He, Xiangyu Zhang ,Shaoqing Ren and Jian Sun. Deep residual learning for image recognition. *In Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016.

[39] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, pages 65–386, 1958.

[40] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, 2015.

[41] Maxime Oquab, Leon Bottou, Ivan Laptev, Josef Sivic. Decaf : A deep convolutional activation feature for generic visual recognition. *Proceedings of the 31st International Conference on Machine Learning*, 2014.

[42] Maxime Oquab, Leon Bottou, Ivan Laptev, Josef Sivic. Learning and transferring mid-level image representations using convolutional neural networks. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014.

[43] Lambert M. Surhone, Mariam T. Tennoe, and Susan F. Henssonow. *Node.Js.* Betascript Publishing, Mauritius, 2010.

[44] Nitish Srivastava,Geoffrey Hinton,Alex Krizhevsky,Ilya Sutskever and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, pages 1929–1958, 2014.

[45] Richard Szeliski. *Computer Vision: Algorithms and Applications.* Springer, 2011.

[46] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, P. T. P Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *ICLR*, 2017.

[47] Al-Rodhaan, M., Al-Dhelaan, A., Tian, Y., Song, B. and Huh, E. N. Threat-based evaluation for context-aware multimedia surveillance system. In *Proceedings of the 8th International Conference on Ubiquitous Information Management and Communication*, 2014.

[48] Standford university. *CS231n: Convolutional Neural Networks for Visual Recognition.*

[49] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. *IEEE Conf Comput Vis Pattern Recognit*, 2001.

[50] Pubnub.com: what is a websocket?

[51] G. Yang and T.S Huang. Human face detection in complex background. *Pattern Recognition*, 1994.

[52] Samuel L. Smith, Pieter-Jan Kindermans, Chris Ying and Quoc V. Le. Don't decay the learning rate, increase the batch size. *CoRR*, 2017.

[53] Yow and Cipolla. Feature based human face detection. *Image and Vision Computing*, 1997.

[54] Emad Barsoum, Cha Zhang, Cristian Canton Ferrer, Zhengyou Zhang. Training deep networks for facial expression recognition with crowd-sourced label distribution. In *Proceedings of the 18th ACM International Conference on Multimodal Interaction*, pages 279–283, 2016.