

TECHNICAL UNIVERSITY OF CRETE, GREECE
SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING

FACESiR:
Face and Speaker Identity Recognition
in Video Streams



Anastasios Karageorgiadis

Thesis Committee

Associate Professor Michail G. Lagoudakis (ECE)

Associate Professor Georgios Chalkiadakis (ECE)

Dr. Vassilios Diakouloukas (ECE)

Chania, November 2019

ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

FACESiR:
Αναγνώριση Προσώπου και Ομιλητή
σε Ροή Βίντεο



Αναστάσιος Καραγεωργιάδης

Εξεταστική Επιτροπή
Αναπληρωτής Καθηγητής Μιχαήλ Γ. Λαγουδάκης (ΗΜΜΥ)
Αναπληρωτής Καθηγητής Γεώργιος Χαλκιαδάκης (ΗΜΜΥ)
Δρ. Βασίλειος Διακολουκάς (ΕΔΙΠ ΗΜΜΥ)

Χανιά, Νοέμβριος 2019

Abstract

Person indexing in video streams requires first to recognize a person's identity and secondly finding the time slot in which a person appears. In this diploma thesis, we develop a method for identifying exposed speakers within a video stream using machine learning techniques. More specifically, with the help of Neural Networks, after we exploit the structure of a video as a sequence of images and sounds, we use these data for the identification of a speaker at each video frame. The above problem is divided into two sub-problems, *Face Recognition* and *Speaker Recognition*, where we use a top-down design to split them into smaller ones. Each sub-problem is solved individually, but the combination of their output probabilities per class leads to an improved final decision regarding classification. The method has been implemented in the Python programming language using the Tensorflow framework and the Keras API. The suggested approach is based on Convolutional Neural Network architectures for both Face and Speaker Recognition. As a result, the combination of image and sound leads to a better decision for the identity of a person who appears in a specific time slot of the video. In addition, the main advantage of the proposed method is that it can be utilized for many different use cases, such as search for missing persons, recognition of celebrities, or even promotion of public figures. It is also worth mentioning that with some minor changes it can be used for identifying any other entity in a video stream.

Περίληψη

Η εύρεση ενός προσώπου σε μία ροή βίντεο απαιτεί πρώτα να αναγνωριστεί η ταυτότητα του εικονιζόμενου και δεύτερον το χρονικό εκείνο διάστημα στο οποίο εμφανίζεται. Στην παρούσα διπλωματική εργασία, αναπτύσσουμε μία μέθοδο αναγνώρισης εμφανιζόμενων ομιλητών εντός μιας ροής βίντεο χρησιμοποιώντας τεχνικές μηχανικής μάθησης. Πιο συγκεκριμένα, αφού αξιοποιήσουμε την δομή ενός βίντεο ως μια ακολουθία εικόνων και ήχων, χρησιμοποιούμε αρχιτεκτονικές νευρωνικών δικτύων, για την ταυτοποίηση ομιλητών σε κάθε πλαίσιο εικόνας. Το παραπάνω πρόβλημα χωρίζεται σε δύο υπο-προβλήματα, την *Αναγνώριση Προσώπου* και την *Αναγνώριση Ομιλητή*, όπου με μία top-down σχεδίαση καταλήγουμε σε ακόμη μικρότερα προβλήματα προς επίλυση. Το κάθε υπο-πρόβλημα επιλύεται ξεχωριστά, ωστόσο συνδυάζοντας τις λύσεις τους με την χρήση των πιθανοτήτων εξόδου ανά κατηγορία, πετυχαίνουμε βελτίωση στην τελική μας απόφαση για κατηγοριοποίηση στη σωστή κλάση. Η εργασία έχει υλοποιηθεί στη γλώσσα προγραμματισμού Python με την χρήση του Tensorflow και του Keras. Η προτεινόμενη προσέγγιση έχει στηριχθεί σε Συνελικτικά Νευρωνικά Δίκτυα (Convolutional Neural Networks), τόσο για την αναγνώριση προσώπου, όσο και ομιλητή. Ως αποτέλεσμα, ο συνδυασμός εικόνας και ήχου οδηγεί σε ορθότερη απόφαση για την ταυτότητα ενός ατόμου που εμφανίζεται σε κάποιο χρονικό διάστημα του βίντεο. Επιπλέον το βασικό πλεονέκτημα της προτεινόμενης μεθόδου είναι ότι μπορεί να αξιοποιηθεί σε πολλές διαφορετικές εφαρμογές, όπως εύρεση αγνοουμένων, αναγνώριση διασημοτήτων, ή ακόμη και προώθηση δημοσίων προσώπων. Αξίζει να σημειωθεί ότι με κάποιες μικρές αλλαγές μπορεί να χρησιμοποιηθεί για ταυτοποίηση οποιασδήποτε άλλης οντότητας σε ροή βίντεο.

Acknowledgements

First, I would like to thank my advisor Mr. Michail Lagoudakis for giving me the opportunity to improve myself by working in his laboratory, while I was free to do what I love and learn from it.

I am also grateful to my family, which has been supporting me all the time, so as to achieve my dreams.

Furthermore, to my teammates at *Kouretes Robotics* team and to all my friends and coworkers at the University; it was an awesome journey. Special thanks,also, to Nvidia Corporation for supplying me with a high-end GPU which had a vital role in my work.

Contents

1	Introduction	1
1.1	Thesis Contribution	1
1.2	Thesis Outline	2
2	Background	5
2.1	Face Recognition	5
2.1.1	Face Geometry	5
2.1.2	Eigen Face - PCA	6
2.1.3	Fisher Face - LDA	8
2.1.4	Nowadays	9
2.2	Speech/Speaker Recognition	10
2.2.1	Feature Extraction	10
2.2.2	Hidden Markov Models	14
2.2.3	Gaussian Mixture Models	17
2.2.4	Nowadays	20
2.3	Neural Networks	21
2.3.1	Introduction to Neural Networks	21
2.3.2	Transfer Learning	24
2.3.3	Convolutional Neural Networks	26
3	Problem Statement	43
3.1	Person Indexing in Video Streams	43
3.2	Related Work	44
4	Our Approach	47
4.1	Face Recognition(FR)	49
4.1.1	Face Detection	49
4.1.2	Face Classification	56
4.2	Speaker Recognition (SR)	60
4.2.1	Voice Features Extraction	60
4.2.2	First Approach Model for Speech Recognition	61

CONTENTS

4.2.3	Speaker Recognition Classifier	63
4.3	FR & SR for Person Indexing in video	66
5	Results	69
6	Conclusion	83
6.1	Conclusion	83
6.2	Future Work	84
6.2.1	Multi Person Detection and Classification	84
6.2.2	More Testing and Evaluation	84
6.2.3	Outcome	84
6.3	Lessons Learned	84
	References	91

List of Figures

2.1	Hamming window [1]	12
2.2	Mel filters [1]	13
2.3	Spectrogram [1]	14
2.4	HMM model	16
2.5	Gaussian Mixture Model example	19
2.6	Neural Network	22
2.7	Single Neuron Back Propagation	23
2.8	Real Neuron Humans' Brain[2]	23
2.9	Transfer Learning method -pipeline[3]	25
2.10	Convolutional Neural Network example [4]	27
2.11	Convolution filtering	29
2.12	Max Pooling example	30
2.13	Max Pooling image example	31
2.14	Sigmoid plot	34
2.15	ReLU plot	35
2.16	Leaky and Parametric-ReLU plot	36
2.17	ELU plot	36
2.18	Softmax example	37
2.19	Softmax Cross Entropy Loss example	39
4.1	A face image	50
4.2	HoG filtered face image	51
4.3	Landmarks face image	52
4.4	Detection with CNN, side-view (profile)image	53
4.5	Detection Landmarks Crop	55
4.6	One Hot Encoding Example	56
4.7	CNN architecture of classifier [5]	58
4.8	Detailed FR CNN layers	58
4.9	FR Model's parameters summary	59
4.10	Tensorflow Graph of the model	62
4.11	CNN architecture layers [5]	62

LIST OF FIGURES

4.12	CNN architecture of SR classifier [5]	63
4.13	Detailed SR CNN layers	64
4.14	SR Model's parameters summary	65
4.15	Flow Chart of FACESIR system	67
5.1	Face classification Loss plot	70
5.2	Face classification Accuracy plot	71
5.3	First Speaker Recognition model Loss plot	71
5.4	First Speaker Recognition model Accuracy plot	72
5.5	Speaker Recognition model Loss plot	73
5.6	Speaker Recognition model Accuracy plot	74
5.7	Increased FR parameters example	75
5.8	Increased FR parameters Loss plot	76
5.9	Increased FR parameters Accuracy plot	77
5.10	Non Normalized input example	78
5.11	Normalized input example	79
5.12	Unknown class input example	80

List of Algorithms

1	Batch Normalization algorithm	32
---	---	--------------------

LIST OF ALGORITHMS

Chapter 1

Introduction

Audiovisual data consists of photos, sound, and videos. They are part of the synchronous everyday lifestyle, which demands better experiences mostly by the ease of use. For example, when we need to edit some photos or videos, we are trying to find out how to do it without spending too much time on the task and concurrently seeking the best results. As technology improves and as people produce more content of these kinds of data, quality has increased, but also their size at the same time. A single photo nowadays can have a size of a few hundred MegaBytes, while videos are reaching sizes of hundreds of GigaBytes. So, researchers and software engineers are putting efforts to discover new paths in order to handle these data. With this huge increase of information and with personal data flowing around the internet, needs like safety, faster and more reliable search across data, classification, and tagging of entities, have emerged.

A problem that combines all the above aspects in one piece is the problem of Person Indexing in video streams. Person indexing in video streams requires first to recognize a person's identity and secondly finding the time slot in which a person appears. This is the problem we are dealing with in this diploma thesis. This statement becomes clearer, if we view a video as a sequence of photos and sounds. Thus, we have different kinds of data to process in a fast and reliable way in order to identify and classify a person.

1.1 Thesis Contribution

In this diploma thesis, we develop a method for identifying exposed speakers within a video stream using machine learning techniques. More specifically, after we exploit the structure of a video as a sequence of images and sounds, we use these data for the identification of a speaker at each video frame.

1. INTRODUCTION

To deal with the complexity of this problem, we propose a top-down approach to split it into two smaller ones: *Face Recognition* and *Speaker Recognition*.

The proposed approach is based on Neural Networks and specifically Convolutional Neural Networks in order to solve both the Face and the Speaker Recognition tasks. We try to provide better results by combining different kinds of features (facial and voice) in such a way that we can easily deploy the final system and extract information about persons appearing in videos, adjusting appropriately the desired level of accuracy. For this purpose, we are filtering our final decision with appropriate thresholds, but we also keep the face and speaker recognition details separately, in case we need a more detailed analysis for the identities in a video frame. We try to take advantage of modern, state-of-the-art techniques that are already used for each sub-problem and, by adding our ideas and personal experience, to create a new solution.

Our solution uses two similar convolutional neural networks to solve the face and speaker recognition problems and the same pipeline to extract image and sound data from videos both during the training phase and during the testing phase. First, we are using a variety of pre-trained models for extracting image features and classifying person identities based on facial characteristics and then a set of algorithmic steps to extract auditory features and accomplish person identity classification based on sound. In the end, we combine the results of each classification to provide higher confidence as to the final result.

Recognition experiments have been conducted with three person identities (classes), specifically the actors Stan Lee, Jack Black, and Samuel L. Jackson, on a variety of free video clips from YouTube. The results from the combined recognition were more than promising for this small set of target identities. Interestingly, the training set for these identities was not formed video streams, but from independent set of images and sounds.

The proposed method has been implemented in the Python programming language using the Tensorflow framework and the Keras API. The main advantage of the proposed method is that it can be utilized for many different use cases, such as search for missing persons, recognition of celebrities, or even promotion of public figures. It is also worth mentioning that with some minor changes it can be used for identifying any other entity in a video stream, besides humans, with the only assumption that the target entity can be identified through images and sounds, for example, some species of animals.

1.2 Thesis Outline

In Chapter 2 we present all the background information needed for this thesis. We give an overview of Pattern Recognition algorithms, the evolution of the Face Recognition problem through the years,

the Speaker Recognition task, and the basic idea for Person Indexing in video streams. In Chapter 3 we state our Person Indexing problem and we refer to different approaches that have been used by others in the previous years. In Chapter 4 we describe our top-down design approach splitting the main problem to two smaller ones and emphasize on our implementation. In Chapter 5 the performance of the proposed approach is evaluated and compared to the individual performance of each sub problem. Finally, Chapter 6 acts as an epilogue for this thesis, presenting our conclusions along with future improvements and all the lessons learned throughout this process.

1. INTRODUCTION

Chapter 2

Background

2.1 Face Recognition

The face recognition problem has become a trend in our days, mainly for security reasons, but also as an extra level of authentication present on the latest smartphones in the past couple of years. This is just one example where face recognition is required for solving an everyday problem. Nevertheless face recognition didn't show up just now, it has been around, as a research problem, for many decades.

For us humans, face recognition problem is already solved, as we can easily use our visual system to recognize different faces. From the computer perspective however, it's not that easy. A machine has a lot of work to do in order to reach the recognition. In general, it is important to split the whole problem into two smaller tasks to be solved, namely Face Detection and then Face Recognition.

2.1.1 Face Geometry

The first approach [6, 7] to the problem relying geometric points of a face, such as eyes', (nose, mouth, ears, ...) position, which were used to build a feature vector; then by calculating the Euclidean distances between the extracted feature vector and that of reference image and by using a simple threshold you could decide how much two faces look alike. That was a great idea to start from, interestingly that the basic concept is still used. Even though this method is robust against illumination changes, there are some important drawbacks; first, is the difficulty of specifying the positions of those geometrical characteristics (marker points) on a face and, secondly, that this kind of information is not enough for efficient and reliable recognition.

2. BACKGROUND

2.1.2 Eigen Face - PCA

The next step was to extend this idea; this is where the Eigen Faces [8, 9] method was introduced. The *Eigen Face* method is based on linear algebra and pattern recognition theory; it uses the [Principal Component Analysis](#) technique to reduce the face-images' space size. Each component of the reduced space are the eigenvectors of the covariance matrix, also every component, in turn, has the highest variance possible under the constraint that it is orthogonal to the preceding components. With this method, we achieve to represent the distribution of face data in a very efficient way. After we have created the eigenface vectors of all face images in the training set, we use the Euclidean distance as a measurement of the difference between an eigenface-image (known face) and a test image –how much the two images look alike–. Let's review the Eigen face method step by step.

First, we assume that $\Gamma_1, \Gamma_2 \dots \Gamma_m$ are the images of the training set with each image denoted as $I(x, y)$. Then, we convert each image into a set of vectors and a new matrix of size $m \times p$ is formed, where m is the number of training images and p is equal to $x \times y$. So, we find the mean face by the following equation:

- Mean Face offset:

$$\Psi = \frac{1}{m} \sum_{n=1}^m \Gamma_n \quad (2.1)$$

where Γ_n is the Face image,

Next we calculate the mean-subtracted face,

- Face distance from average:

$$\Phi_i = \Gamma_i - \Psi \quad (2.2)$$

where $i = 1, 2 \dots m$, and $A = [\Phi_1, \Phi_2, \dots \Phi_m]$ is the mean-subtracted matrix vector with size $m \times p$.

Then, we implement the matrix transformations, and matrix A is reduced to

$$C_{m \times m} = A_{m \times p} \times A_{p \times m}^T \quad (2.3)$$

where C is the covariance matrix. If we view C in more detail we have :

• **Covariance Matrix:**

$$C = \frac{1}{m} \sum_{n=1}^m \Phi_n \Phi_n^T = AA^T \quad (2.4)$$

where C is $N^2 \times N^2$, and determining the N^2 eigenvectors and eigenvalues. This is an intractable task for typical image sizes; but with this analysis, the calculations are greatly reduced to $M \times M$. N^2 is the order of the number of pixels in the images, and M is the order of the number of images in the training set. So, the actually training set will be relatively small $M \ll N^2$.

Afterwards, we have to calculate the eigenvectors V_{mm} and the eigenvalues λ_m from the C matrix using *Jacobi* method and we have to order the eigenvectors by the highest eigenvalues. Jacobi's method is chosen because of its accuracy and reliability. We also apply the eigenvectors matrix V_{mm} and the adjusted matrices Φ_m ; these vectors determine linear combinations of the training set images to form the eigenfaces U_k .

• **Eigenfaces U_k :**

$$U_k = \sum_{n=1}^m \Phi_n V_{kn} \quad (2.5)$$

where $k = 1, 2, \dots, m'$

Instead of using all m eigenfaces, $m' < m$ we assume that the images provided for training are more than 1 for each individuals or class.

• **Each Image Face Vector:**

$$W_k = U_k^T (\Gamma - \Psi) \quad (2.6)$$

each image have its face vector W_k and mean subtracted vector of size $p \times 1$ and eigenfaces is $U_{pm'}$. The weights form a feature vector: $\Omega^T = [w_1, w_2, \dots, w_{m'}]$

It is worth mentioning that a face can be reconstructed by using its feature, Ω^T vector and the previous eigenfaces $U_{m'}$ as follows:

$$\Gamma' = \Psi + \Phi_f \quad (2.7)$$

where $\Phi_f = \sum_{i=1}^{m'} w_i U_i$.

The Eigen Faces method has its drawbacks such as the fact that no class labels are taken into account; it's only about comparing a pair of face-images, and because of the variance dependence, it's very sensitive to external light conditions. The only positive point about Eigen Faces, they are optimal for face reconstruction tasks.

2. BACKGROUND

2.1.3 Fisher Face - LDA

The next level approach, is Fisher Faces [10] a *Linear Discriminant Analysis (LDA)* method also well known in the *Pattern Recognition* field. Fisher Faces have the ability to take into account class labels; this is very helpful for face recognition, because we want to create a class per face id and separate it from the other. Here comes the main property of the Fisher method, which is that it tries to minimize the variance within a class, while trying to maximize the variance between the classes. *Fisher Face* extends and improves the *Eigen Face* method by dealing better with illumination variation in an image, while at the same time can handle face expression quite well.

Fisher Faces Equations:

- Between-class scatter matrix:

$$S_B = \sum_{i=1}^c N_i (\mu_i - \mu)(\mu_i - \mu)^T \quad (2.8)$$

- Within-class scatter matrix:

$$S_W = \sum_{i=1}^c \sum_{x_k \in X_i} (x_k - \mu_i)(x_k - \mu_i)^T \quad (2.9)$$

where $i = 1, 2, \dots, c$, c is the total number of classes, μ_i is the mean image of class X_i , μ is the overall mean of images, $x_k \in X_i$, x_k is the k^{th} image inside the class X_i , and N_i is the number of samples in this class.

- Optimal projection matrix:

$$\begin{aligned} W_{opt} &= \arg \max_W \frac{|W^T S_B W|}{|W^T S_W W|} \\ &= [w_1 w_2 \dots w_m] \end{aligned} \quad (2.10)$$

where $w_i | i = 1, 2, \dots, m$ is the set of generalized eigen-vectors of S_B and S_W corresponding to the m largest generalized eigenvalues $\lambda_i | i = 1, 2, \dots, m$, i.e.,

$$S_B \mathbf{w}_i = \lambda_i S_W \mathbf{w}_i$$

$i = 1, 2, \dots, m$. It is also important to say that there are at most $c - 1$ *nonzero* generalized eigenvalues, and so an upper bound is the $c - 1$ for m .

2.1.4 Nowadays

Nowadays, there are so many different solutions for robust Face Recognition to choose from, depending on how much accurate we want it to be and how fast it can be deployed. Just to reference some of them, mostly for feature extraction, these are *Haar*-like features [11], Local Binary Transformation [12, 13], and Histogram of Oriented Gradients [14, 15] which are implemented by libraries like *OpenCV* and *Dlib*, followed by a type of binary classifier, in order to separate faces from non-face regions in an image. And then, there are Neural Networks, like CNNs, which can solve both the Detection and Recognition part of the problem. We can also use a combination of classical Pattern Recognition algorithms along with Neural Networks to solve the whole aspect of the problem, with a more elegant way¹.

¹See also OpenCV Face Recognition [6]

2.2 Speech/Speaker Recognition

The speech recognition includes all the methodologies and technologies that enable the recognition and translation of a spoken language into text by computers. It is known more, like, automatic speech recognition (ASR). We are already enjoying this technology as the basic feature of chatbots or as voice assistants helping our everyday life, by executing many kinds of tasks in an easier way. For instance, we can make a call by using voice commands while driving.

On the other end, we have a related recognition task referred to as speaker recognition. Speaker (or voice) recognition [16] is the ability of a computer to know the voice of a person speaking into it, in such a way that only the voices that the computer knows can be identified. This process can simplify the task of translating speech in systems that can deal with a specific person's voice in order to use them for authentication or verification as part of a security process. Speaker recognition can be split into two cases; the first is the *text-dependent* case, where a speaker is recognized by some predefined set of words or phrases, and the other is the *text-independent* case, which is based on identifying the speaker by his/her actual voice characteristics and can be used for a variety of applications of speaker recognition.

2.2.1 Feature Extraction

Filter banks [1] along with the MFCCs [17, 18] (*Mel-Frequency Cepstral Coefficients*) are two similar and widely known techniques for voice features extraction. MFCCs are also computing filter banks, but they add some more extra steps up to the final feature map of a voice or sound signal. Let's now take a look at the algorithm behind the filter banks technique and compare it with the MFCCs one.

Pre-emphasis filter:

First, we have to amplify the high frequencies of our signal because higher frequencies have smaller magnitudes. In order to do this, we use a pre-emphasis filter. The pre-emphasis filter applied to a signal x using the first-order filter is shown in the following equation:

$$y(t) = x(t) - \alpha x(t - 1) \quad (2.11)$$

where α is the filter coefficient, with typical values to be 0.95 or 0.97.

Split signal into frames:

Then we split our audio file into small overlapped frames due to frequency changes over time, and for that reason, it's not a good idea to apply FFT to the whole signal. So we assume that in a short period of time our frequencies are stationary and we can apply FFT to these short frames, without losing any frequencies contours of a signal over time. In general, frame sizes in speech processing range from 20 *msec* up to 40 *msec* with a 50% overlap between consecutive frames of our signal.

Window filtering:

The next step is to apply a kind of window filter as proposed by the initial theory of MFCCs. For example, in this step we use a Hamming window filter, which is more like a Gaussian filter. The Hamming window is given by this equation:

$$w[n] = 0.54 - 0.46\cos\left(\frac{2\pi n}{N-1}\right) \quad (2.12)$$

where $n = 0, 1, \dots, N-1$, with N being the length of the window or otherwise N is the total number of samples. Both $n, N \in \mathbb{Z}$.

In Figure 2.1 we have a hamming window of size $N = 200$, for each sample n we get a value (amplitude) $0 \leq w[n] \leq 1$. The main reason why we apply a Hamming window is to counterbalance the assumption of FFT that the data are infinite and also to reduce the spectral leakage.

2. BACKGROUND

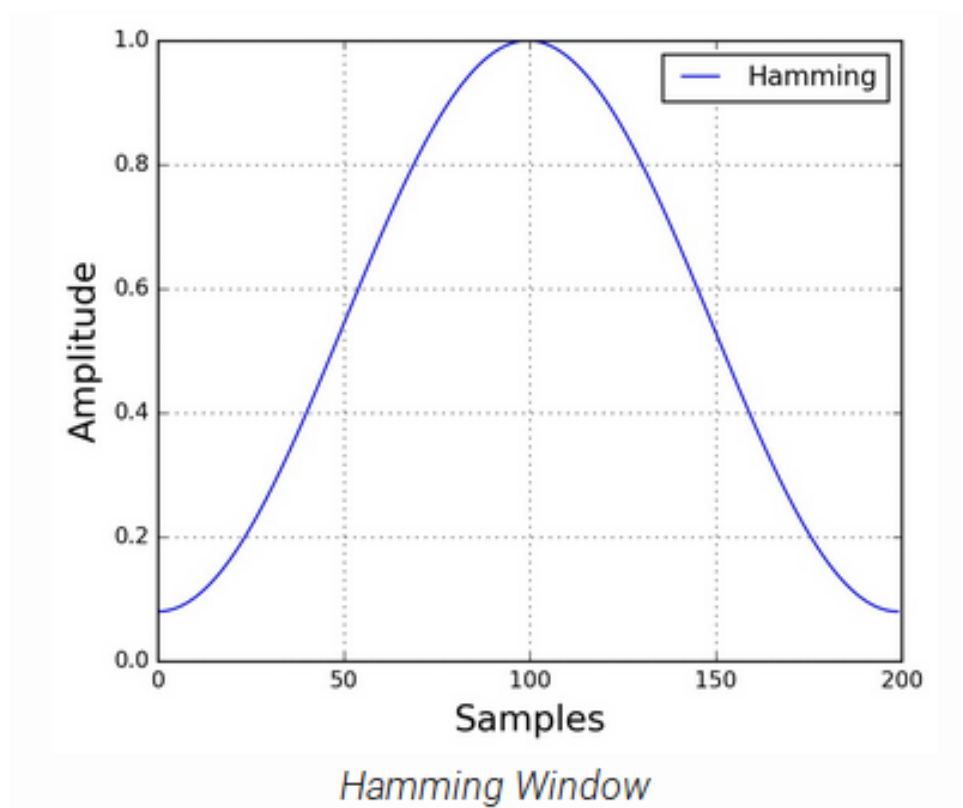


Figure 2.1: Hamming window [1]

FFT and Power Spectrum:

For this step we do a N -points FFT on each of the above frames to calculate the frequency spectrum. This method is referred to as *Short-Time Fourier-Transform (STFT)* and we are using a $N = 2048$. Afterwards we compute the power spectrum using the following type:

$$P = \frac{|FFT(x_i)|^2}{2N} \quad (2.13)$$

where x_i is the i -th frame of a signal x .

Mel-Coefficients:

In the end, we have to apply some triangular filters, shown in Figure 2.2, with 64 filters on a Mel-scale to the power spectrum, to extract the frequency bands of the voice signal. The Mel-scale is closer to simulating the non-linear human ear perception of a sound. Mel-scale is more discriminative at the lower frequencies and less at the higher ones. Converting Hertz to Mel is given by these equations:

$$m = 2595 * \log \left(1 + \frac{f}{700} \right) \quad (2.14)$$

$$f = 700 * \left(10^{m/2595} - 1 \right) \quad (2.15)$$

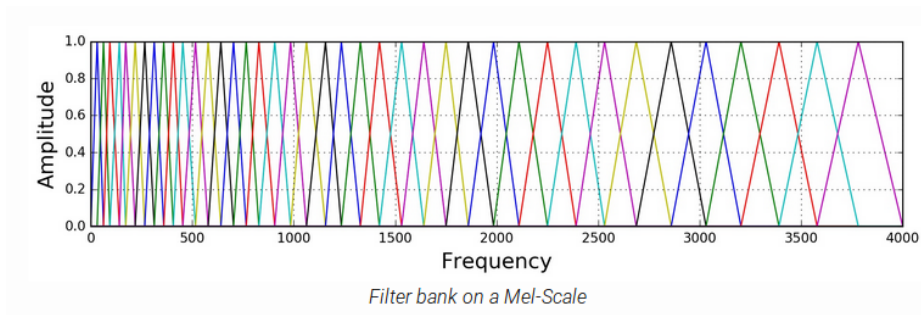


Figure 2.2: Mel filters [1]

$$H_m(x) = \begin{cases} 0, & x < f(m-1) \\ \frac{x-f(m-1)}{f(m)-f(m-1)}, & f(m-1) \leq x < f(m) \\ \frac{f(m+1)-x}{f(m+1)-f(m)}, & f(m) \leq x < f(m+1) \\ 0, & x \geq f(m+1) \end{cases}$$

2. BACKGROUND

After we have applied the filter bank to the power spectrum (periodogram) of the our signal, we obtain the spectrogram shown in Figure 2.3.

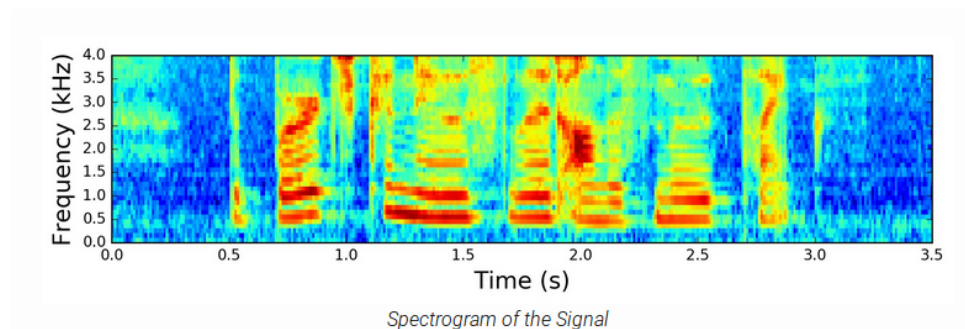


Figure 2.3: Spectrogram [1]

With these steps above, we have described how filter banks extract the required information from a voice signal. These steps are the same for MFCC technique also, but here comes the next step, which makes MFCC different from filter banks. Because of the high correlation between frames, sometimes this can cause problematic behavior in some machine learning algorithms. So, we have to remove the frames' correlation. There comes the MFCC technique by applying a DCT (*Discrete Cosine Transformation*) to de-correlate the filter banks coefficients. Also by this method, we manage to compress the size of the representation of filter banks.

Mean Normalization:

If we have to balance the spectrum and improve the *Signal-to-Noise Ratio (SNR)*, we can simply subtract the mean of each coefficient from all frames. In case that Mel-scaled filter banks are the desired features, then we can skip to mean normalization.

2.2.2 Hidden Markov Models

The Hidden Markov Model [19, 20] is an extension of the Markov Model, which uses Markov's chain rule to describe the dependency between the different states of a stochastic process. A Markov chain contains all the possible states of a system and the probabilities of transitioning from one state to the other. The most important feature of this approach is the assumption of Markov's rule that the next state depends only on the current state, which simplifies the description and the calculations of finding the probability of the next state.

But there are some cases, within a Markov model, where not all the states are observable by an external observer. We call these states *hidden* states, and the model is referred to as HMM. In order to describe and define these states, we need some indirect information about them. For example, let's say we have two states of a person's emotions $\{\mathbf{A}: \text{Angry or } \mathbf{C}: \text{Calm}\}$, which we cannot directly observe. In addition, we have the information that if $\mathbf{A}: \text{True}$, person's voice volume is high with probability **0.8**, but when $\mathbf{C}: \text{True}$ that probability is **0.1**. So, we can observe the voice's volume and figure out something about the emotional state of a person. The probability of observing an observation given a *hidden* state is called the **emission probability**, and then there is the probability of transitions between different states, which is referred to as **transition probability** (Section 2.4). In order to learn our HMM model and define its parameters (emission and transition probabilities) we use the *Baum-Welch* algorithm which is a form of the Expectation Maximization algorithm (E.M.) (Section 2.2.3);

Now, on the speech recognition field, the observation is the context in each audio frame; an audio frame is represented by using MFCC parameters (Section 2.2.1) and with a HMM we represent the sequence of these audio frames. The HMM approach though, is used mostly for text-dependent speech recognition, like when we need to recognize specific phrases or words. If we move one step forward, a state in a HMM model can be thought to have a mixture of distributions (e.g. Gaussian mixture) with the probability of belonging to a distribution being represented by the *emission* probability; each state in the HMM can have a unique set of *emission* probabilities; or in other words, that each state in a HMM can be thought of as a GMM (*Gaussian Mixture Model*, Section 2.2.3)¹. But when it comes to text-independent speech recognition HMM fails (or at least it is too complicated) to approach every kind of words' combination and phrases, due to the size of the whole set.

$$P(x) = \sum_s P(x, s) = \sum_s P(x|s)P(s) \quad (2.16)$$

¹emission probabilities representing the probability of association to a distribution

2. BACKGROUND

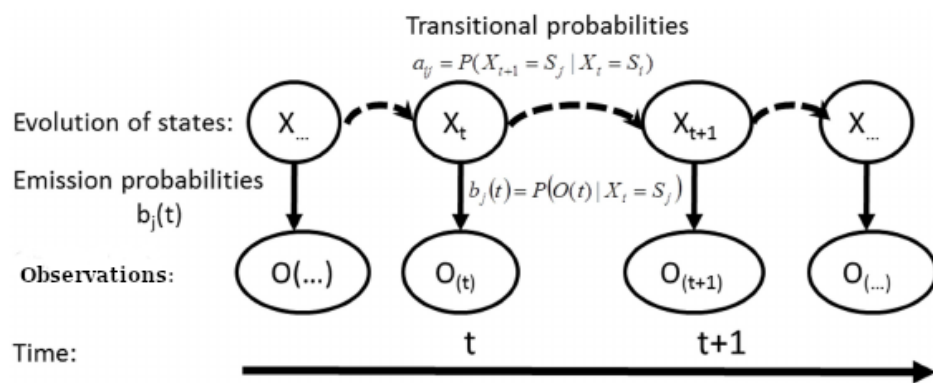


Figure 2.4: HMM model

2.2.3 Gaussian Mixture Models

A GMM or Gaussian Mixture Model [21] is a probabilistic model for representing normally distributed sub-populations within an overall population. Also as we mentioned before, it can be thought of as a single state HMM. In general, mixture models don't require knowing in which sub-population a data point belongs to, allowing that way, to learn automatically each class. In the case of Speaker Recognition by using a mixture of Gaussian distributions we try to model speakers identity by weighting each part of the Gaussian mixture with a specific probability in such a way that every speaker has his own model as a linear combination of Gaussian distributions. So, a GMM is parameterized by two types of values, the mixture **component weights** and the **component means** and **variances/covariances**. In a Gaussian mixture model with \mathbf{J} components, the j^{th} component has a mean of μ_j and variance of σ_j for the univariate case and a mean of $\vec{\mu}_j$ and covariance matrix of Σ_j for the multivariate case. The mixture component weights are defined as P_j for component C_j , with the constraint that $\sum_{j=1}^J P_j = 1$, so that the total probability distribution normalizes to $\mathbf{1}$.

A multi-dimensional Gaussian distribution is given by the following type:

$$\begin{aligned} \mathbf{N}(\mathbf{x}|\boldsymbol{\mu}_j, \Sigma_j) &= \frac{1}{\sqrt{(2\pi)^J |\Sigma_j|}} \exp \left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_j)^T \Sigma_j^{-1} (\mathbf{x} - \boldsymbol{\mu}_j) \right) \\ &= \sum_{j=1}^J \mathbf{N}(\mathbf{x}|\boldsymbol{\mu}_j) \\ &= \sum_{j=1}^J p(\mathbf{x}|j) \end{aligned} \quad (2.17)$$

where $p(\mathbf{x}|\boldsymbol{\mu}_j) \sim \mathbf{N}(\boldsymbol{\mu}_j, \Sigma_j)$,

An alternative way to model an unknown $p(\mathbf{x})$ is through a linear combination of p.d.f (probability density function) of,

$$p(\mathbf{x}) = \sum_{j=1}^J p(\mathbf{x}|j)P_j \quad (2.18)$$

where $\sum_{j=1}^J P_j = 1$, $\int_{\mathbf{x}} p(\mathbf{x}|j)dx = 1$ and \mathbf{J} distributions contribute in the $p(\mathbf{x})$.

To learn the parameters of a Gaussian Mixture Model we use the Expectation Maximization (E.M) algorithm,

Expectation Maximization Algorithm (EM):

2. BACKGROUND

In our case the entire set of data consists of the common events $\mathbf{x}_k, j_k, k = 1, 2, \dots, N$ where $j_k \in [1, J] \subset \mathbb{Z}$ and defines the mixture that was produced from \mathbf{x}_k . By applying the Maximum-Likelihood (ML) rule :

$$p(\mathbf{x}_k, j_k; \boldsymbol{\theta}) = p(\mathbf{x}_k | j_k; \boldsymbol{\theta}) P_{j_k} \quad (2.19)$$

Let's assume mutual exclusion between samples of the set, the log-likelihood probability function is:

$$L(\boldsymbol{\theta}) = \sum_{k=1}^N \ln(p(\mathbf{x}_k | j_k; \boldsymbol{\theta}) P_{j_k}) \quad (2.20)$$

If $\mathbf{P} = [P_1, P_2, \dots, P_J]^T$ and the vector of unknown parameters is $\boldsymbol{\Theta}^T = [\boldsymbol{\theta}^T, \mathbf{P}^T]^T$.

• Step E:

$$\begin{aligned} Q(\boldsymbol{\Theta}; \boldsymbol{\Theta}(t)) &= E \left[\sum_{k=1}^N \ln(p(\mathbf{x}_k | j_k; \boldsymbol{\theta}) P_{j_k}) \right] \\ &= \sum_{k=1}^N E [\ln(p(\mathbf{x}_k | j_k; \boldsymbol{\theta}) P_{j_k})] \\ &= \sum_{k=1}^N \sum_{j_k=1}^J P(j_k | \mathbf{x}_k; \boldsymbol{\theta}(t)) \ln(p(\mathbf{x}_k | j_k; \boldsymbol{\theta}) P_{j_k}) \end{aligned} \quad (2.21)$$

Now we can simplify our symbols in the above equation by removing the index k from j_k because $\forall k$ we sum all possible J values of j_k which are the same for each k . We can now show the algorithm of Gaussian mixture components with diagonal covariance, $\sum_j = \sigma_j^2 I$.

$$p(\mathbf{x}_k | j; \boldsymbol{\theta}) = \frac{1}{(2\pi\sigma_j^2)^{l/2}} \exp\left(-\frac{\|\mathbf{x}_k - \boldsymbol{\mu}_j\|^2}{2\sigma_j^2}\right) \quad (2.22)$$

Suppose that the a-priori probabilities P_j , the mean values $\boldsymbol{\mu}_j$ and the variances $\sigma_j^2, j = 1, 2, \dots, J$ of Gaussian mixture components are also unknown. So $\boldsymbol{\theta}$ is a vector of dimension $J(l+1)$ and from the above equations by removing constant term we have:

• Step E:

$$Q(\boldsymbol{\Theta}; \boldsymbol{\Theta}(t)) = \sum_{k=1}^N \sum_{j=1}^J P(j | \mathbf{x}_k; \boldsymbol{\theta}(t)) \left(-\frac{1}{2} \ln \sigma_j^2 - \frac{1}{2\sigma_j^2} \|\mathbf{x}_k - \boldsymbol{\mu}_j\|^2 + \ln P_j \right) \quad (2.23)$$

• **Step M:**

In this step we have to maximize the above function as to μ_j, σ_j^2 and P_j .

$$\mu_j(t+1) = \frac{\sum_{k=1}^N P(j|\mathbf{x}_k; \Theta(t)) \mathbf{x}_k}{\sum_{k=1}^N P(j|\mathbf{x}_k; \Theta(t))} \quad (2.24)$$

$$\sigma_j^2(t+1) = \frac{\sum_{k=1}^N P(j|\mathbf{x}_k; \Theta(t)) \|\mathbf{x}_k - \mu_j(t+1)\|^2}{l \sum_{k=1}^N P(j|\mathbf{x}_k; \Theta(t))} \quad (2.25)$$

$$P_j(t+1) = \frac{1}{N} \sum_{k=1}^N P(j|\mathbf{x}_k; \Theta(t)) \quad (2.26)$$

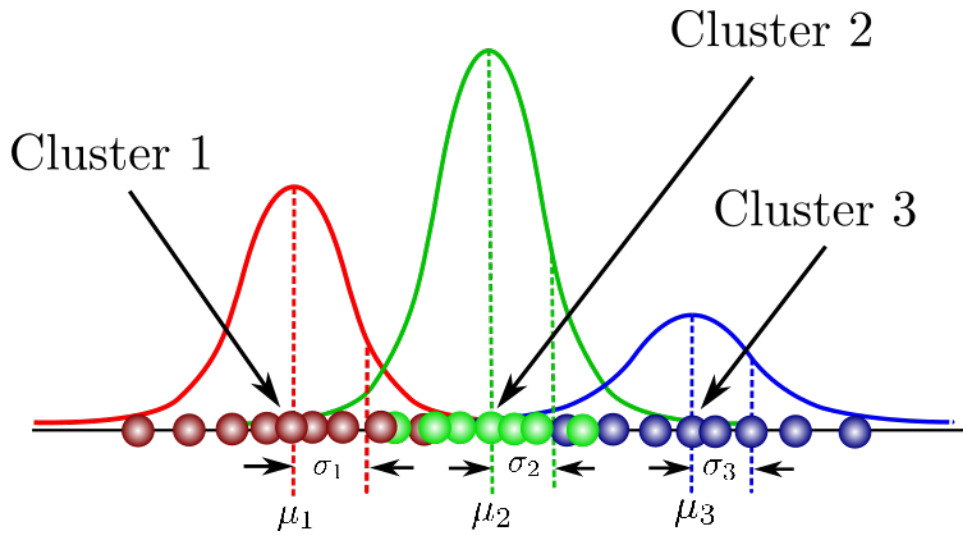


Figure 2.5: Gaussian Mixture Model example

2. BACKGROUND

2.2.4 Nowadays

Speech and Speaker recognition have a long history with many major innovations. In our days from the perspective of technology, the machine learning research, mostly by the use of neural networks and with the big data, both speaker and speech recognition have benefited and they have significantly improved. As a result, we have faster and more accurate systems that deal with both tasks; we can easily use neural networks for speech/speaker recognition. For instance [21], if we think of HMM's states as classes we can use a multilevel perceptron model (MLP [22]) as a non-linear classifier. The observations are given to the input nodes and the network will have as output the states; the number of outputs is equal to the number of HMM's states. The outputs of this network will be close to the a-posterior probabilities $P(i|\mathbf{x})$ ¹. From Bayes rule we can get,

$$p(\mathbf{x}|i) = \frac{P(i|\mathbf{x})p(\mathbf{x})}{P(i)} \quad (2.27)$$

where the a-priori probabilities of the states $P(i)$, are defined from the relative frequencies of appearance and $p(\mathbf{x})$ is the same for all the states during the recognition process. But this method cannot always split the audio frames in an accurate way, as the original HMM method; in other words, the boundaries between the frames may not be defined that well and an overlay among them might appear.

An alternative is Convolutional Neural networks, a trending solution for speech/speaker research [23, 24] problems like those above, as they manage to use the speech spectrogram as an input image. Furthermore, many big companies like Google have launched a few software tools helping developers to build their own applications based on speech/speaker recognition tasks relying on AI and machine learning field ([Google cloud platform](#)).

¹See example [21] page-472

2.3 Neural Networks

2.3.1 Introduction to Neural Networks

The first idea of a mathematical model close to the Neural Network (NN) (Figure 2.6) was introduced back in 1943 by Warren McCulloch and Walter Pitts. They used some threshold technique to simulate the way a real neuron works, on the humans' brain (Figure 2.8). It was a few years later when Frank Rosenblatt in 1958 created the Perceptron [25] Model, the simplest model that is used in neural networks till our days. For example, Iris classification problem is actually solved with a Perceptron model and it constitutes an introduction to Neural Networks for beginners. But then, it was Marvin Minsky and Seymour Papert [26] who showed the limitations and problems of this model, many of which were solved by Paul Werbos [27] around 1975 by creating the Back Propagation technique to solve the XOR problem. Back Propagation is the way of a neural network to understand its "mistakes" and learn from them by feeding its output back to its inputs and by calculating the error from a set point (real data), as shown in Figure 2.7. This was a fundamental step on the evolution of Neural Networks as it is being used from that day to all the Neural Network algorithms we have come up with.

For the following decades, there wasn't any improvement in the field, except, maybe, from the Max-Pooling method, introduced back in 1992 [28] for 3D object recognition. This method helped to the least shift invariance and the tolerance to deformation. Things kept going on for Neural Network models with small improving steps through the following years.

It was around 2009, when research interest was renewed, mostly because the big data and the Internet of things evolved very fast. Progress continues till nowadays, where NN are considered to be the state of the art mathematical model for solving almost every difficult and complex problem in Computer science, and especially in the Artificial Intelligence field. Many models were created in the past ten years, like *Recurrent Neural Network (RNN, LSTM)* [29], *Convolutional Neural Network [30] (CNN)* or *Residual Neural Networks [31] (ResNet)*.

2. BACKGROUND

It is also good to mention that nothing would have ever happened, if there wasn't the evolution of the hardware technology, like GPU acceleration techniques. Big Data was one ingredient, but without the hardware improvements that helped the processing of data, by making it faster and more efficient, possibly Neural Networks would have stayed in the closet. Companies, like Nvidia, have contributed so much to the field and they still do it with an awesome pace that does not leave you indifferent and also makes you curious for the future.

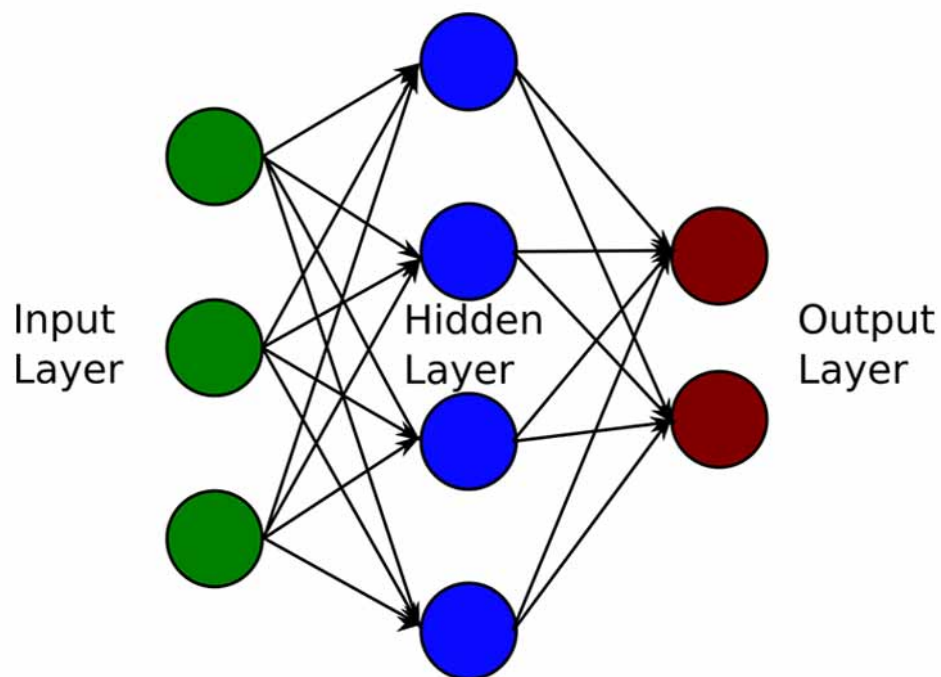


Figure 2.6: Neural Network

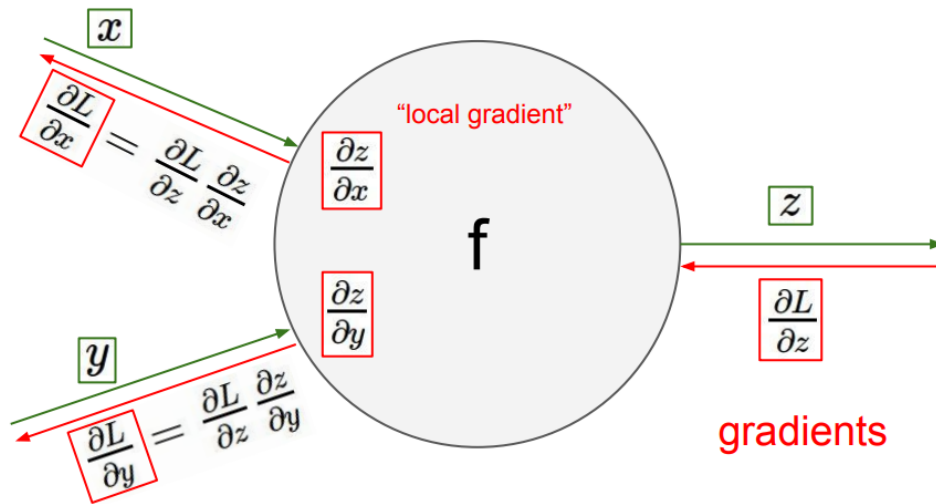
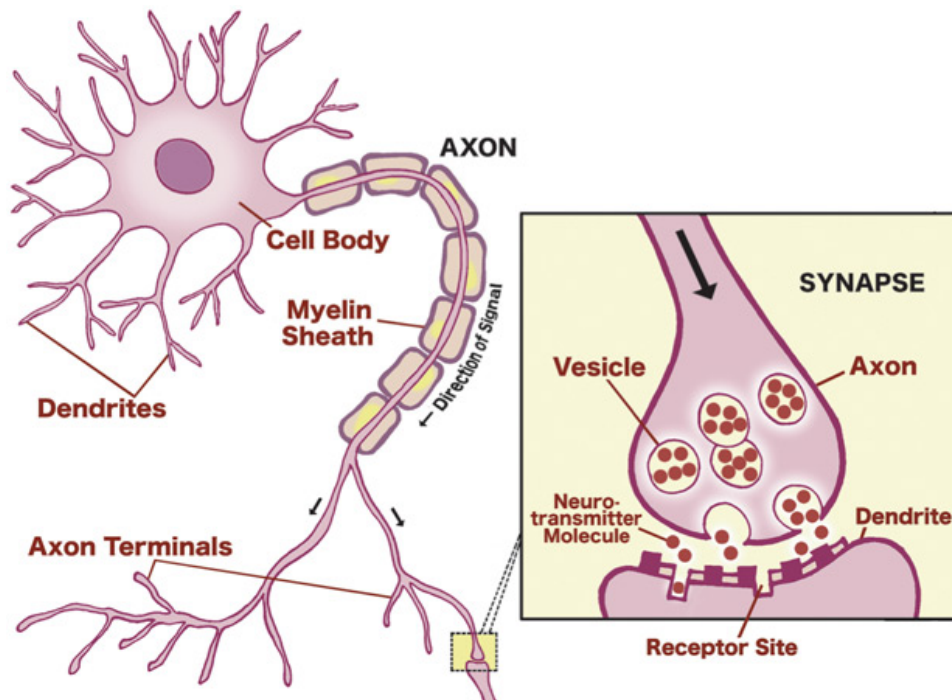


Figure 2.7: Single Neuron Back Propagation

Figure 2.8: Real Neuron Humans' Brain^[2]

2. BACKGROUND

2.3.2 Transfer Learning

Observing the way we humans try to deal with tasks, an interesting behavior emerges; the behavior of seeking previous similar experiences or any kind of useful knowledge from the past, that will help to find solutions to new tasks (or problems). This behavior, if we see it as an idea, is the cornerstone of the transfer learning [\[32\]](#) technique, used in machine learning.

So, in transfer learning, the knowledge learned in one or more source tasks is transferred and used to improve learning of a related target task. How is this applied to neural networks? As we know, the neural networks training phase requires a lot of time and it is not very efficient to create models from scratch each time we have a different use-case for neural networks. For that purpose, we demand to reuse some NN models that have been trained, for solving similar problems. Let's take as an example a convolutional neural network that is used for classifying objects through images (e.g. tables, chairs, etc), after we create this model and we train it for some object-classes. If then we need to recognize a different kind of object, like cars, creating a new model from scratch to recognize cars in images is not the best approach. Instead, we try to use the base model's knowledge for object classification in order to recognize cars too. This approach at least will save us time and effort. In convolutional neural networks, the early layers extract mostly generic features from images –color, illumination, etc – while the later layers extract mostly features specific to the original-dataset . Applying transfer learning to our example, we will keep the weights of the first few layers of the object classifier and we will re-train the last layers in order to recognize car-like features. The idea of transfer learning is visualized in [Figure 2.9](#).

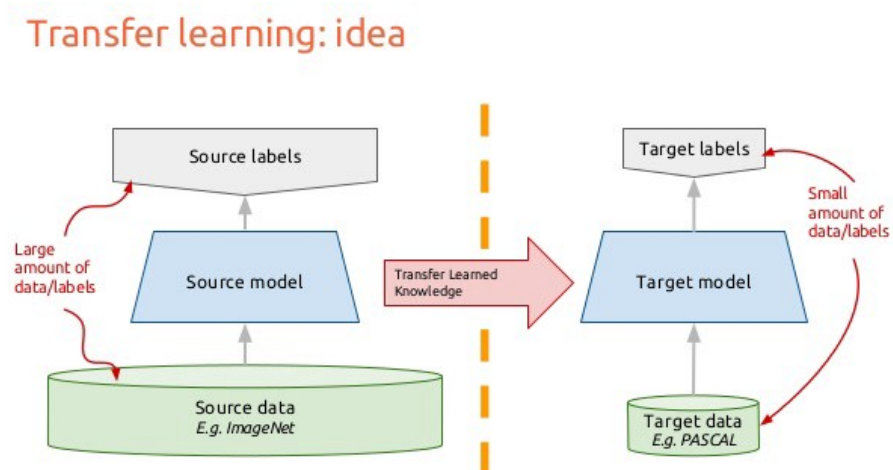


Figure 2.9: Transfer Learning method -pipeline[3]

2. BACKGROUND

2.3.3 Convolutional Neural Networks

Convolutional Neural Network [30] (CNN) is a type of *Artificial Neural Network (ANN)* mostly used for image recognition and classification tasks. In general, a CNN model can be seen as a regularization of multi-layer Perceptrons, which actually are a kind of fully connected networks. By fully connected we mean that each layer's node is linked with every node of the neighbor layer. This property attributed to multi-layer Perceptron is prone to overfitting data. So, by taking advantage of hierarchical patterns in data, CNN tries to assemble the most complex patterns into simpler and smaller ones, achieving with that, reduction of complexity and connectedness of the model.

From the aspect of image classification tasks, CNNs demand, relatively, less pre-processing on data, compared to classic pattern recognition algorithms, because they manage to learn the filters that fit to the extraction of feature sets that otherwise would need to be hand-crafted. This is a huge plus for the engineers, since it requires slighter human effort for the design and development of complicated solutions for such kind of problems.

By the time this text is written, CNNs are the main architecture (Figure 2.10) that is used for a wide range of problems, besides image classification. For example, CNNs are used now for Speech and Speaker Recognition, also gradually replacing Recurrent Neural Networks, which are used for input data that are based on time-sequence, an example of this kind of data being sound (or voice). Also we should not forget to mention that convolutional networks inspired and are fundamental for some of the state of the art models, like ResNets [31].

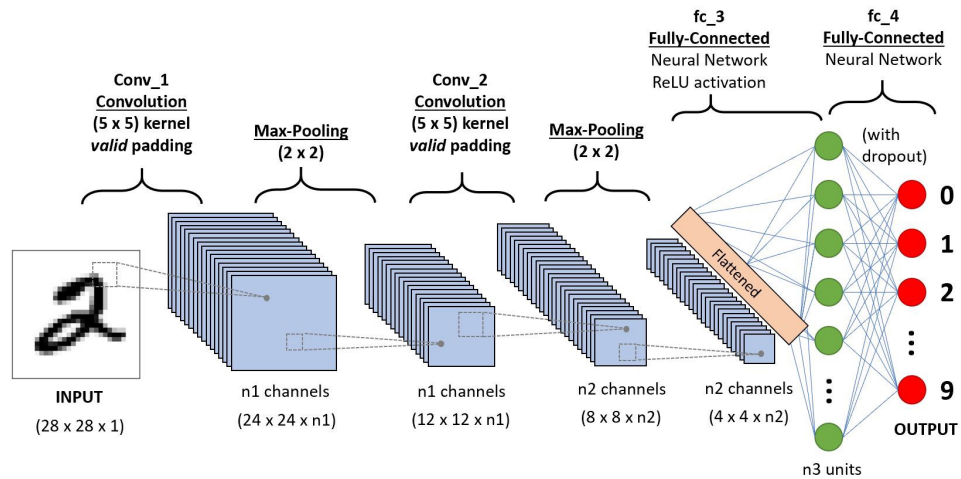


Figure 2.10: Convolutional Neural Network example [4]

2. BACKGROUND

2.3.3.1 Convolution Layer

A convolutional layer [30, 4] is the main part of a *Convolutional Neural Network*. The purpose of this layer is to be applied to the input in such a way that filters the whole input across every dimension in order to extract features (Figure 2.11). The convolution layer's parameters consist of a set of learnable filters, which have a small receptive field, but extend through the full depth of the input volume. During the forward pass, each filter is convoluted across the width and height of the input volume for each channel. For example, an RGB image volume is an N-dim-array and has a size of shape (height, width, 3), so convolution filters each sub-array with respect to channel size. This layer computes the dot product between the entries of the filter and the input and produces a 2-dimensional activation map of that filter. As a result, the network learns filters that activate, when it detects some specific type of feature at some spatial position in the input. In each machine learning framework, there are two types of Convolutional Layers implemented; one is for 2-D and the other is for 3-D input volumes.

The Convolution layer is specified by the kernel and the stride size which actually set the shape of the filter mask that we want to pass through an input matrix. Kernel refers to the actual filter size; usually a kernel of 3×3 is used on most CNN model implementations. The Stride option determines if there will or will not be an overlap between masks that filter the input matrix.

$$G[m, n] = (f * h)[m, n] = \sum_j \sum_k h[j, k] f[m - j, n - k] \quad (2.28)$$

where h is the filter(kernel), f is the image-vector, and m, n is number of rows and columns or image vector of size $m \times n$.

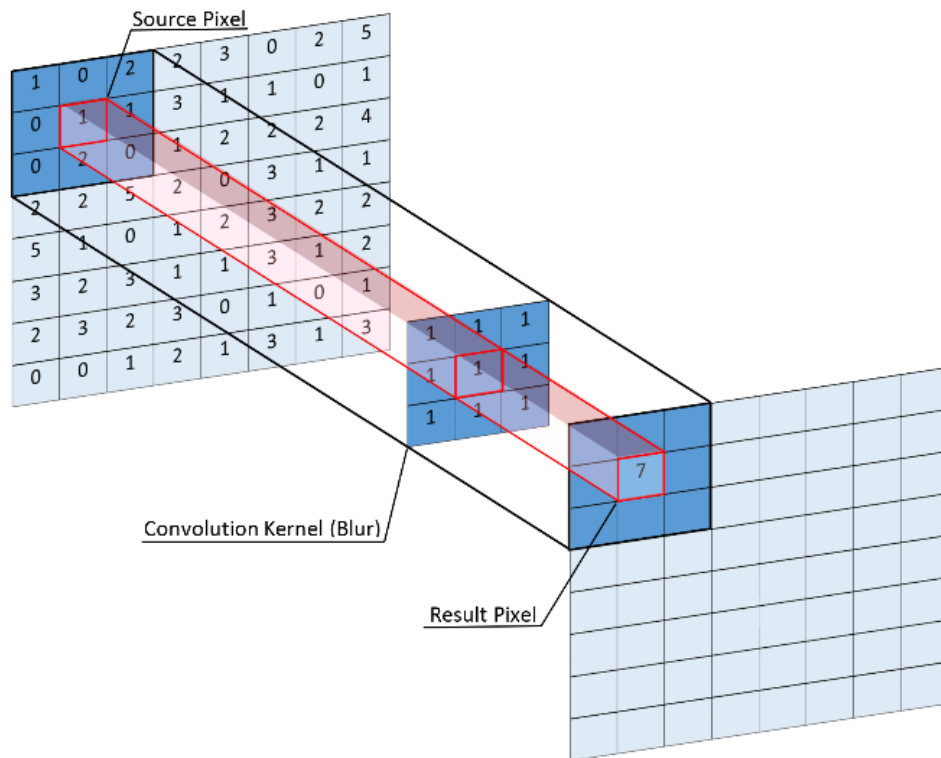


Figure 2.11: Convolution filtering

2. BACKGROUND

2.3.3.2 Max Pooling

Max pooling [33, 28, 30] is a standard way of down-sampling the outputs of a layer in a Neural Network. It is most commonly used in Convolutional Neural Networks after each convolution layer. Its main purpose is reducing the dimensionality of data and allowing for assumptions to be made about features contained in the sub-regions binned, for example, in an image to keep some specific Regions of Interest. The way it is done is by applying a max filter to (usually) non-overlapping subregions of the initial representation, it also helps as to avoid the overfitting of our model.

For example, in Figure 2.12 suppose that we have a 4×4 sized input; we apply some 2×2 convolution filters and we also have a stride of 2. This example shows that our max-pooling layer will only keep the highest values for each region, and as you can see it reduces the size back to 2×2 from 4×4 . In Figure 2.13 there is a max pooling example in a real image.

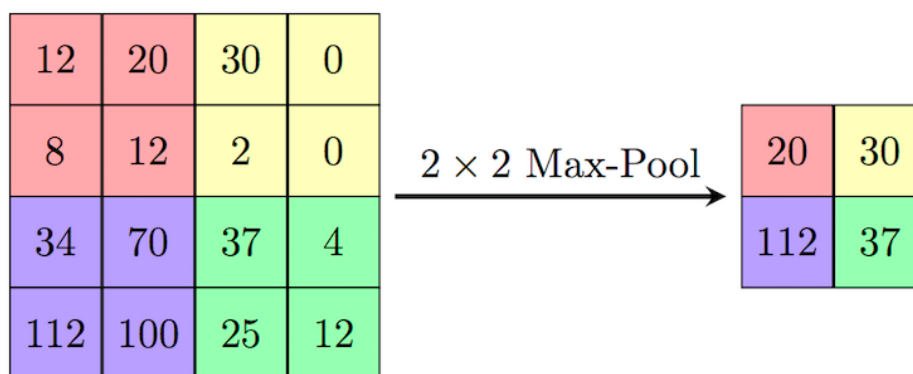


Figure 2.12: Max Pooling example

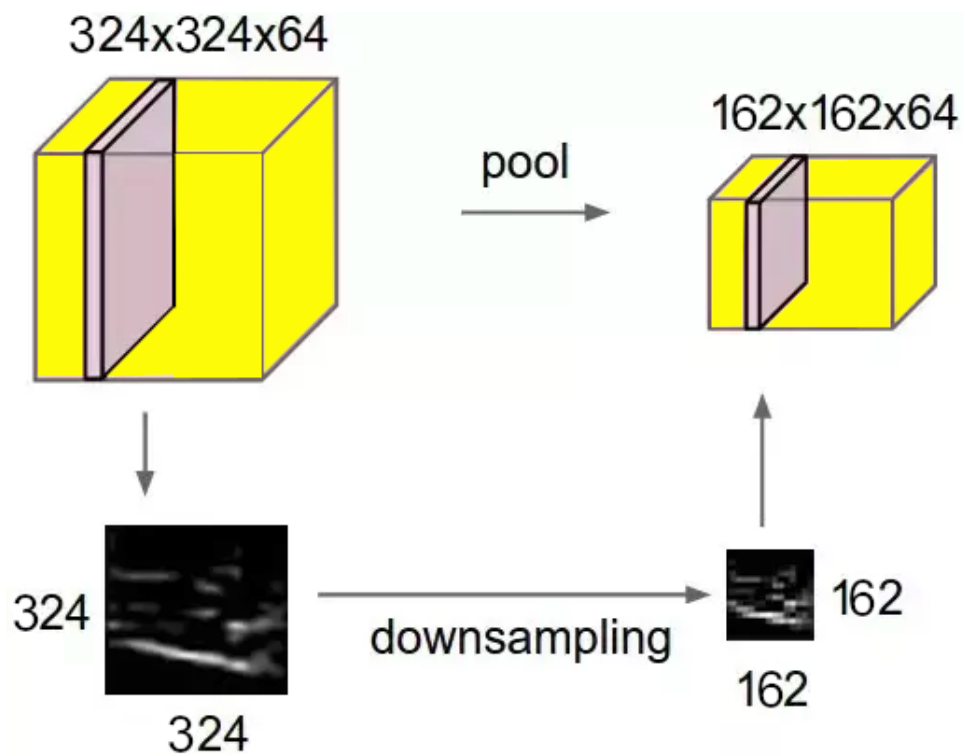


Figure 2.13: Max Pooling image example

2. BACKGROUND

2.3.3.3 Batch Normalization

Batch normalization [34] is a technique for finding and normalizing the output data from a layer in Neural Networks. More precisely, we apply batch normalization after a convolutional layer, in our case of a CNN model. This technique helps our network to define the appropriate distribution of the layer's output, which may differ from the standard *Gaussian* distribution with zero mean and variance equal to one. This problem is also called covariate shifting and our goal is to reduce it by applying batch normalization. Defining the distribution of input/output data on each layer helps our model to learn faster and be more robust to different initialization schemes. In our case, we used batch normalization during the development of the first model approach for speaker recognition in order to be able to use higher learning rates without vanishing or exploding gradients and we could also remove dropout layers to mitigate overfitting. Thus, we were more flexible to test and improve our model's training process by observing how it responds to different learning rates, dropout factors, etc. But, how batch normalization algorithm work? It starts from the point, where we have zero mean and one variance, and tries to specify the mean and the variance of data in each layer by learning some extra variables, which will be trained along with all the other variables (weight, biases, etc). With this process, we manage to learn a distribution closer to the actual output data. Actually, these extra variables are linear equation scaling factors.

$$y_i = \gamma \bar{x}_i + \beta \quad (2.29)$$

Algorithm 1 Batch Normalization algorithm

INPUT: Values of x over a mini-batch: $\mathcal{B} = \{x_1 \dots, x_m\}$

Parameters to be learned γ, β

OUTPUT: $\{y_i = BN_{\gamma, \beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i \quad (2.30)$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad (2.31)$$

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad (2.32)$$

$$y_i = \gamma \hat{x}_i + \beta \equiv BN_{\gamma, \beta}(x_i) \quad (2.33)$$

There are many questions about what the right place in a model to apply batch normalization whether it is necessary, while a lot of research through out the last years is made in order to find out the positives and consequences of every possible scenario.

2. BACKGROUND

2.3.3.4 Activation Functions

Activation functions [35, 30, 36] are used to obtain the output of a neural network layer's node. It actually maps the node's result values in $[0, 1]$ or $[-1, +1]$ depending on the function. There are two types of activation functions, Linear and Non-Linear ones. For example, a linear function would be something like $y = x$. A non-linear example is an exponential or logarithmic function.

In neural networks, we mostly use non-linear functions, one of this type is the **Sigmoid**(Figure 2.14). But we mostly use a variant of the **Sigmoid**, the **ReLU** –*Rectify Linear Unit*(Figure 2.15)–, which is half rectified in $x \in [-\infty, 0]$. The function and its directive are both monotonic (increasing).

•The Sigmoid function:

$$\sigma(x) = \frac{1}{1 + \exp^{-x}} \quad (2.34)$$

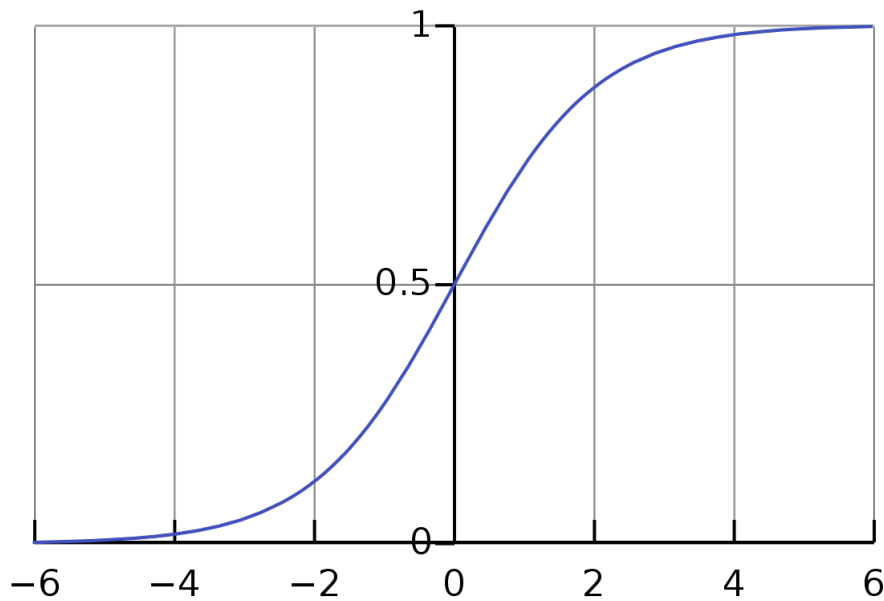


Figure 2.14: Sigmoid plot

•The ReLU function:

$$R(z) = \max(0, x) \quad (2.35)$$

or otherwise we can say:

$$f(x) = \begin{cases} 0, & \text{if } x < 0 \\ x, & \text{if } x \geq 0 \end{cases} \quad (2.36)$$

$$(2.37)$$

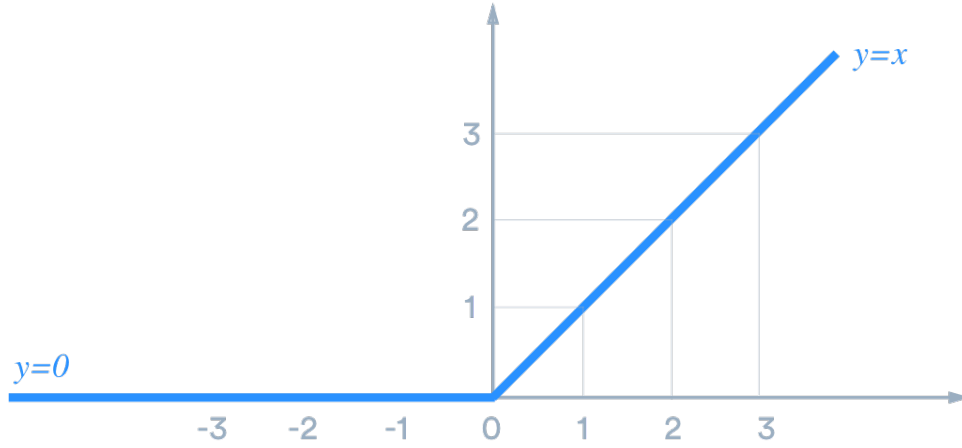


Figure 2.15: ReLU plot

The **ReLU**'s drawback is that all negative values become zero immediately, which decreases the model's ability to fit or train from the data properly. That means any negative input given to the ReLU activation function turns the value into zero immediately in the graph, which in turn affects the resulting graph by not mapping the negative values appropriately. It is also known as zero-stacked neurons problem and its more likely to occur when we have a high learning rate or there is a large negative bias. This problem means that many neurons will not add useful information to our network, so a solution to that is adding a dropout parameter, where neurons are disabled with a random probability in order to retrieve information from different neurons each time and improve the fitting of the model.

Engineers have come up with some other activation functions to avoid this problem, like **LReLU**, **PReLU**, and **ELU**. Which are shown in Figures 2.16, 2.17 along with their equations:

- Leaky-ReLU equation:

$$f(x) = \begin{cases} 0.01 \times x, & \text{if } x < 0 \\ x, & \text{if } x \geq 0 \end{cases} \quad (2.38)$$

$$(2.39)$$

if we generalize the equation of LReLU it comes to PReLU,

- Parametric-PReLU equation:

$$f(x, a) = \begin{cases} a \times x, & \text{if } x < 0 \\ x, & \text{if } x \geq 0 \end{cases} \quad (2.40)$$

$$(2.41)$$

where a is added to set of the trainable variables of the model.

2. BACKGROUND

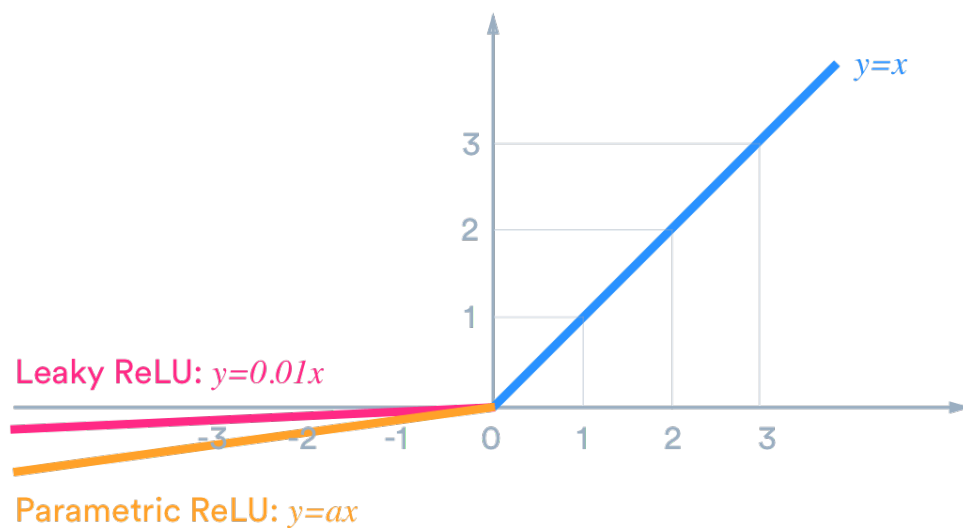


Figure 2.16: Leaky and Parametric-ReLU plot

- Exponential Linear Unit (ELU) equation:

$$f(x, a) = \begin{cases} a \times (\exp^x - 1), & \text{if } x \leq 0 \\ x, & \text{if } x > 0 \end{cases} \quad (2.42)$$

$$(2.43)$$

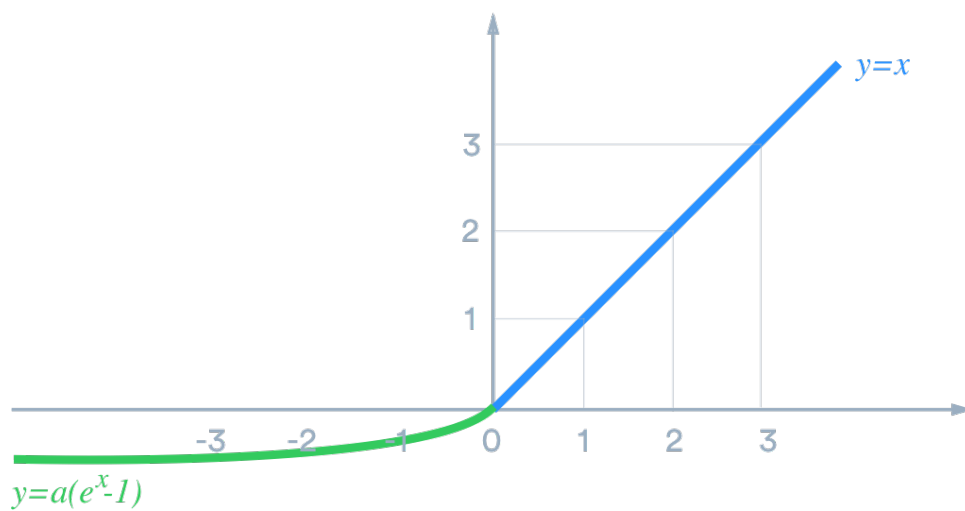


Figure 2.17: ELU plot

An other well known activation function is the **Softmax**. **Softmax** is a function that takes as argument a vector of N real numbers and convertes it into a probability distribution consisting of N probabilities proportional to the exponentials of the input numbers (Figure 2.18). This method actually normalizes the input between $[0,1]$ by respecting the main rule of probability theory that $\sum_i^N P(N_i) = 1$. In neural networks **Softmax** is used in the most cases after the last layer –logits– to map the non-normalized output to a probability distribution over the predicted logits or classes. Also Softmax is used into calculation of the loss in such cases by using softmax–cross-entropy loss.

- Softmax Equation :

$$\sigma(x_i) = \frac{\exp^{x_i}}{\sum_{j=1}^N \exp^{x_j}} \quad (2.44)$$

this is the standard softmax function $\sigma: \mathbb{R}^N \rightarrow \mathbb{R}^N$, where $i = 1, \dots, N$ and $\mathbf{x} = (x_1, \dots, x_N) \in \mathbb{R}^N$.

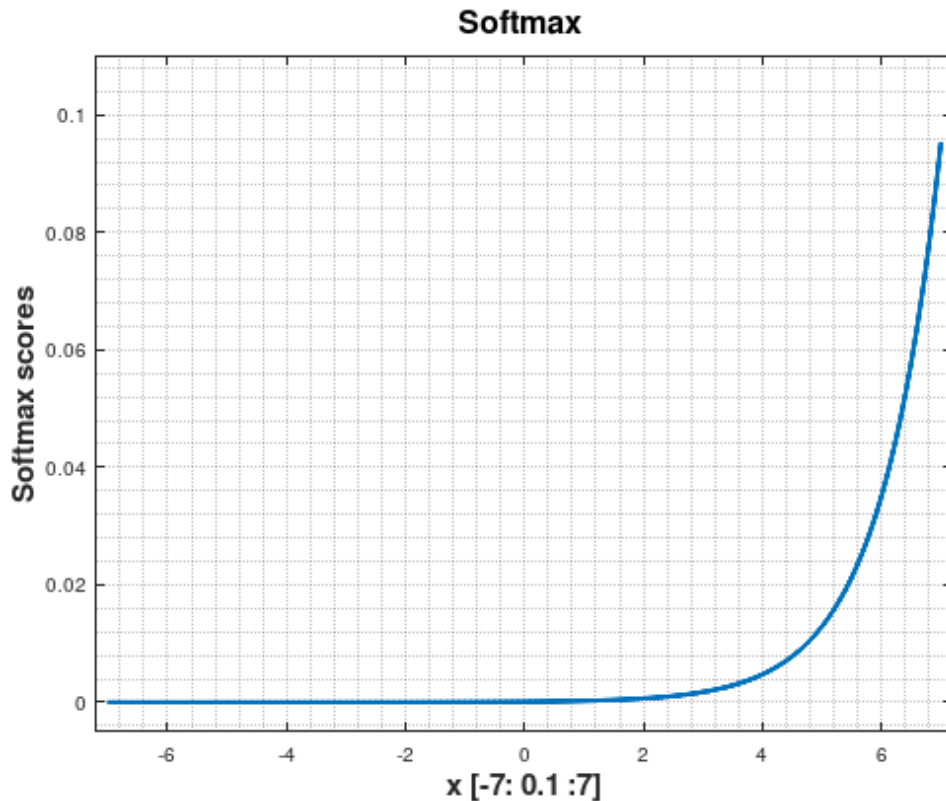


Figure 2.18: Softmax example

2. BACKGROUND

2.3.3.5 Loss Function

The most used and reliable loss function used in Neural Networks is *Cross entropy loss* [37, 38]. *Cross entropy loss* has its root based on information theory, where cross entropy is the average number of bits needed to state that two probability distributions (p,q) are different from each other. In machine learning this idea refers to the distance between the predicted output (labels) of our network and the true labels of the data-set.

- Cross Entropy Definition:

$$H(p, q) = E_p[-\log(q)] \quad (2.45)$$

which can be formatted using the *Kullback-Leibler* divergence (known as the relative entropy of p with respect to q).

$$H(p, q) = H(p) + D_{KL}(p||q) \quad (2.46)$$

- Cross entropy Equation:

$$H(p, q) = - \sum_{x \in X} p(x) \log q(x) \quad (2.47)$$

where p, q are the two discrete probability distributions for the same support X .

So, in neural networks q_i are the predicted labels and p_i Re the true labels (p_i and q_i referred to the labels of a trianing sample i). The purpose of a machine learning algorithm – like Neural Net – is to minimize the cross entropy until its optimal point. That's the reason we need a kind of optimizer, so that cross entropy converges faster and avoid local minimal points – (see Section 2.3.3.6).

Then, there is the softmax-cross entropy loss (see Section 2.3.3.4) to calculate the cross-entropy loss of the logits layer when we need to apply softmax as the final layer of our network's output.

- Softmax-Cross Entropy Loss:

$$\mathbb{L}_i = - \sum_{c=1}^C y_c \log(p_c) \quad (2.48)$$

or otherwise

$$\mathbb{L}_i = - \sum_{c=1}^C y_c \log(p_c) + (1 - y_c) \log(1 - p_c) \quad (2.49)$$

where p_c is the output probability for the class c , of a single training sample i

if we have M training samples it can be generalized to:

$$\mathbb{L} = - \sum_{i=1}^M \sum_{c=1}^C y_{ci} \log(p_c) \quad (2.50)$$

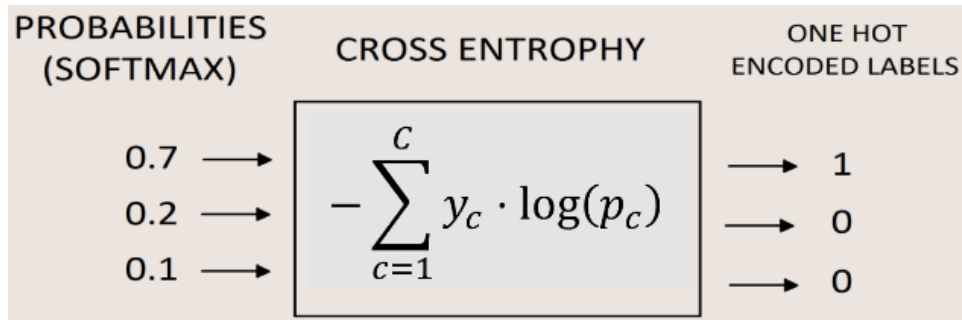


Figure 2.19: Softmax Cross Entropy Loss example

There are also some other methods to calculate loss (or regression) of a model, like mean square error, but cross entropy seems to be a better approach for most of the cases.

2. BACKGROUND

2.3.3.6 Adam Optimizer

Adam [39] is an acronym that derives from Adaptive Moment estimation. It is a widely common optimization algorithm that extends *Stochastic Gradient Descent* and *RMSprop* [40] and its goal is to update the network weights iteratively based on the training data. In a nutshell, it is an adaptive learning rate method, which means, it computes individual learning rates for different parameters and it uses the estimations of first and second moments of gradient in order to adapt the learning rate for each weight of the neural network.

By the term moment we mean the N -th value of a random variable which is defined as the value of that variable to the power of N . This random variable in a neural network represents the gradient of the loss function.

$$m_n = E[X^n] \quad (2.51)$$

Let's see how the Adam algorithm works and what math is behind each step. In the very first stage, all the vectors of moving averages are initialized with zeros so $m_0 = 0$. Then we have the first moment, which is simply the mean of the gradient, and after there is the second moment, that is, the uncentered variance of it. So, Adam optimizer tries to estimate moments by utilizing exponentially the moving averages, which are computed on the gradient evaluated on the current mini-batch.

- Moving averages equations:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \quad (2.52)$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \quad (2.53)$$

where β 's are the hyper-parameters of the algorithm and also are defined as $\beta_1 = 0.9$, $\beta_2 = 0.999$.

Then, we have the expected values of the estimators, which are described by the following equations:

$$E[m_t] = E[g_t] \quad (2.54)$$

$$E[v_t] = E[g_t^2] \quad (2.55)$$

If these expected values are equal to the parameter we are trying to estimate, then our estimator is unbiased, but in our case, since we initialize the average with zeros, the estimator becomes biased towards zero.

• Unroll step by step equations:

$$m_0 = 0 \quad (2.56)$$

$$m_1 = \beta_1 m_0 + (1 - \beta_1)g_1 = (1 - \beta_1)g_1 \quad (2.57)$$

$$m_2 = \beta_1 m_1 + (1 - \beta_1)g_2 = \beta_1(1 - \beta_1)g_1 + (1 - \beta_1)g_2 \quad (2.58)$$

$$m_3 = \beta_1 m_2 + (1 - \beta_1)g_3 = \beta_1^2 + (1 - \beta_1)g_1 + \beta_1(1 - \beta_1)g_2 + (1 - \beta_1)g_3 \quad (2.59)$$

As is shown from the above steps, we can understand that as further we go expanding the value of m the less the first values of the gradient contribute, to the overall value as they get multiplied by smaller β in each step. In general, we can transform the previous equation to this form:

$$m_t = (1 - \beta_1) \sum_{i=0}^t \beta_1^{t-i} g_i \quad (2.60)$$

But, still we have to correct our estimation and this step is called bias correction

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \quad (2.61)$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \quad (2.62)$$

The last step is to scale the learning rate for each parameter individually:

$$W_t = W_{t-1} - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \quad (2.63)$$

2. BACKGROUND

There are also some variants of Adam, like *Adamax* which uses the $L2$ norm of the current gradient but is a little bit unstable and then there is *Nadam*, that uses Nesterov momentum. At the end Adam is one of the trend optimizers in Neural Networks in the last couple of years.

Chapter 3

Problem Statement

3.1 Person Indexing in Video Streams

Person indexing in video streams is the problem of recognizing a person's identity and finding the time slot in which that person appears in a video. This is the problem we are dealing with in this diploma thesis and it is among the trend problems in the Artificial Intelligence (AI) field, mostly for human identification and verification. As our needs for more efficient systems of this kind are increasing, several hybrid recognition methods are proposed, in our case a combination of image and voice data. This statement becomes clearer, if we view a video as a sequence of photos and sounds. Thus, we have different kinds of data to process in a fast and reliable way in order to identify and classify a person.

Image and Voice data in a video stream seem to be uncorrelated at first sight. If we step back for a moment and think, we will find out that they share a common feature within the video, namely the time. The time refers to the actual time-stamp of a set of frames where images and sound coexist. This simple observation implies that time will have to play a critical role in helping us to solve the problem.

Another obvious observation is that image data will be used to address the problem of face recognition (when the face of the target person appears). Likewise, sound data will be used to address the problem of speaker recognition (when the target person speaks). As a result, to obtain an effective solution to the problem we are dealing with in this thesis, these two sub-problems, *Face Recognition* and *Speaker Recognition*, have to be studied separately first and finally be fused.

Our goal is to build a system that can be easily deployed and is also as fast and accurate as it can be. Furthermore, we would like our work to be future-proof, in a way that we can maintain and expand it with little effort. For example, we would like to be able to add new persons' identities or even more other types of identities, if we intend to use the system, for recognizing different kinds of entities besides

3. PROBLEM STATEMENT

humans.

3.2 Related Work

Our work was inspired by a previous approach for Person Indexing based on audio-visual data, titled "*A fully automatic face recognition system using a combined audio-visual approach*" and authored by Albiol, Torres, and Delp [41]. In that paper, audio and visual information is used independently, so as to obtain confidence values that indicate the likelihood of a known person to appear in a video stream. Then, a *post-classifier* (a kind of late integration) is used in order to achieve the fusion between visual and audio confidence values. As a result of this hybrid system, a better recognition rate is achieved. In this section we will discuss the above-referenced paper in order to explain the techniques it uses. We will also point out aspects of this work, which can be improved and will be used as starting points in our approach.

The PCA method (Section 2.1.2) is one of the most successful approaches used for face recognition. In the above work, the face recognition problem is solved with the help of the *self-eigenfaces*, which is a variant of the PCA algorithm. In self-eigenfaces, the PCA is executed for each person using her/his subset of training faces (images); their training set includes multiple training faces per person. In a nutshell, the PCA is performed independently for every single person-class, while in the original PCA algorithm it is only performed once across all the faces (one face image per person). The outcome of this analysis is a set of eigenfaces for every identity (person). These types of eigenfaces are referred to as self-eigenfaces. To model each person they use a set of parameters,

$$\mathbf{F}_m = \{\mathbf{x}_\mu^m, \mathbf{V}_m\} \quad (3.1)$$

where \mathbf{x}_μ^m is the average face of person m and \mathbf{V}_m is a matrix whose columns are the K_m principal self-eigenfaces \mathbf{x}_μ^m of person m . For the purpose of dimensionality reduction, K_m is chosen to be much smaller than the number of pixels of the face patterns. The location of the eyes is taken as the normalization parameter for the size of each training face.

For the test stage, with the self-eigenfaces method, each face candidate is projected and then reconstructed utilizing each \mathbf{F}_m (self-eigenfaces). At the end, the metric of confidence for the possibility the identity of the face candidate to be person m is the *reconstruction error*. More specifically, if \mathbf{x} is a test face, represented as a column vector, the equation

$$\mathbf{y}^m = \mathbf{V}_m^T (\mathbf{x} - \mathbf{x}_\mu^m) \quad (3.2)$$

gives the projection coefficients of a person m . Then, the original test pattern \mathbf{x}^m can be calculated approximately by the following reconstruction:

$$\tilde{\mathbf{x}}^m \approx \mathbf{V}_m \cdot \mathbf{y}^m + \mathbf{x}_\mu^m \quad (3.3)$$

After reconstruction of the test image, the *reconstruction error* of person m is given by:

$$\epsilon = \frac{1}{255} \sqrt{\frac{1}{RC} \sum_{j=1}^{RC} |\mathbf{x}(j) - \tilde{\mathbf{x}}^m(j)|^2} \quad (3.4)$$

where j is the vector index and R, C are the number of image rows and columns respectively. When this error is the smallest, the identity of self-eigenfaces is the same (or close enough) to the test face. The global confidence measurement is obtained by taking the median value of that error; if ϵ_i is the minimum error of reconstruction for all the faces appearing in a frame i , then

$$\mathbf{FC}_m = \text{median}\{\epsilon_0, \epsilon_1, \dots, \epsilon_{N-1}\} \quad (3.5)$$

represents the frame-based confidence of a person (**Face Candidate**) m to appear in a particular frame.

Inside the face recognition part, there is also a simple face detection method. For the purpose of face recognition, the location of facial features must match those of the training faces. Therefore, the location of facial features is compatible to the distance of the face model inside the detection phase. Facial features are extracted by applying a zero-mean *Laplacian filter*¹ to enhance facial features, followed by a morphological *erosion* of size 3×3 to remove the white areas from teeth and eyes. Moreover, to remove some non-useful features, an intensity thresholding is applied (20% of dark pixels are below the threshold). If W_p are the pixels of the 3×3 window at a pixel location p , after intensity normalization and thresholding, then the distance to the face model is given by:

$$D(\mathbf{F}_{av}, W_p) = \|\mathbf{F}_{av} - W_p\| \quad (3.6)$$

The above distance is only computed when skin color can be used to reduce the search of face candidates to skin-colored areas, so as to reduce the computational burden, when flat skin-colored areas can be discarded. The \mathbf{F}_{av} is the average face model, which was obtained by averaging 280 patterns. Moreover, this face detection method is used to extract faces from head and shoulders sequences (frontal view in an upright position for a person) and it does not take into account face rotations or pose variations. Because of the above, it leads to some missing faces with these circumstances.

The drawbacks of above approach are that (a) it uses only front-facial images and videos, like interviews at a TV show, (b) it is very sensitive to illumination changes, and (c) the selected region of

¹Laplacian filtering

3. PROBLEM STATEMENT

interest (ROI) contains a lot of useless information (e.g. neck, shoulders, etc) during the face detection. In our approach we try to deal with these drawbacks.

To perform speaker recognition, features must be extracted from the audio signal with respect to the speaker. Once again, the *MFCC method* (Section 2.2.1) is applied to define the features per speaker across all the train voice data. In particular, the first 12 cepstral vector coefficients and their corresponding deltas are extracted every 17ms and a *Hamming* window of size 34ms is used. Then classification is formulated, as a classical hypothesis test using the *log-likelihood* ratio:

$$\mathbf{AC}_m = \log \left\{ \frac{p(S/m)}{p(s/BM)} \right\} \quad (3.7)$$

where S is the set of speaker-dependent features and $p(S/m)$, $p(S/BM)$ are the conditional p.d.f. of the person m and the background BM respectively. The AC_m is the confidence that a person m speaks in a speech segment. Speaker recognition demands for *text-independence* is fulfilled by modeling the p.d.f. with GMMs (Section 2.2.3). During the training phase, GMMs are built from 2 – 3min of clear speech. The drawback of the above approach is that the small number (12) of cepstral coefficients leads to relatively lower accuracy. This is another point we try to improve with our approach.

In the last step of the work of Albiol, Torres, and Delp, a *Bayesian* post-classifier is applied to make the final confidence decision from the two independent modalities (face and speaker recognition). A *Bayesian* classifier is based on the likelihood ratio of the conditional probability density functions (p.d.f), $p(\mathbf{C}_m|tc)$ and $p(\mathbf{C}_m|im)$ (tc : true claimant, im : impostor classes). The p.d.f can be modeled by GMMs, because they can deal with arbitrary densities.

Chapter 4

Our Approach

In this diploma thesis, we develop a method for identifying exposed speakers within a video stream using machine learning techniques. More specifically, after we exploit the structure of a video as a sequence of images and sounds, we use these data for the identification of a speaker at each video frame. To deal with the complexity of this problem, we propose a top-down approach to split it into two smaller ones: *Face Recognition* and *Speaker Recognition*.

The proposed approach is based on Neural Networks and specifically Convolutional Neural Networks in order to solve both the Face and the Speaker Recognition tasks. We try to provide better results by combining different kinds of features (facial and voice) in such a way that we can easily deploy the final system and extract information about persons appearing in videos, adjusting appropriately the desired level of accuracy. For this purpose, we are filtering our final decision with appropriate thresholds, but we also keep the face and speaker recognition details separately, in case we need a more detailed analysis for the identities in a video frame. We try to take advantage of modern, state-of-the-art techniques that are already used for each sub-problem and, by adding our ideas and personal experience, to create a new solution.

Another crucial step for our implementation is to define our data and what our system's input will be. Video streams can be analyzed as sequences of images combined with sequences of sound. So, splitting a video into frames and saving these images, generates the data set we need, to test on our *Face Recognition module*. Then, we extract sound windows from the same video in order to create the sound data for the *Speaker Recognition module*. In order to recognize a face identity, we need a training data-set with all the possible classes (ids) and for each class, we must have a large enough variety of face images. By variety we mean images that contain face poses of a person with different angles, expressions, and resolution.

4. OUR APPROACH

The Face Recognition problem has two stages; the one of face detection and the other of recognition of the detected face in an image (or video frame). The purpose of the face detection module (or stage) is to detect faces into images and classify them into those which contain face and those with no faces. Then, we keep the images where a face was detected and we crop each image around the face-ROI¹. Afterwards, we use them as input images to feed our face-recognizer, which will tell the id of that face –if it is a known person’s face–. After that we resize our images to a specific size, common for all of them, let’s say 256×256 and feed them in our model during the training phase. In the first phase, we train our networks for Face Recognition by picking up Google images from persons that are contained also in the speech data [42] set. Then we restore our trained models and run predictions by using a test data-set, to check the real accuracy of the model on new unknown data.

The Speaker Recognition on the other hand tries to match the voice characteristics of a person with the already known voices in the data set. For this task, we used a similar older project, which also has the same feature size (image size of 64×17), but was developed for Anti-spoofing verification on speech utterances (*source data*). Our goal was to re-train it, by relying on the Transfer Learning (Section 2.3.2) idea, in order to fine-tune it over new target data taken from the VoxCeleb dataset [42]. That was only the first approach we took but we will see later on what the final speaker recognition model was. As for the data pre-process we used the filter banks technique (see Sections 2.2.1, 4.2.1), to extract voice features from all speech data.

Our solution uses two similar convolutional neural networks to solve the face and speaker recognition problems and the same pipeline to extract image and sound data from videos both during the training phase and during the testing phase. First, we are using a variety of pre-trained models for extracting image features and classifying person identities based on facial characteristics and then a set of algorithmic steps to extract auditory features and accomplish person identity classification based on sound. In the end, we combine the results of each classification to provide higher confidence as to the final result.

The proposed method has been implemented in the Python programming language using the Tensorflow [43] framework and the Keras API [44]. The main advantage of the proposed method is that it can be utilized for many different use cases, such as search for missing persons, recognition of celebrities, or even promotion of public figures. It is also worth mentioning that with some minor changes it can be used for identifying any other entity in a video stream, besides humans, with the only assumption that the target entity can be identified through images and sounds, for example, some species of animals.

¹ROI: Region of Interest

4.1 Face Recognition(FR)

In the Face Recognition problem we have to recognize a person from an image input. The face of that person must be learned before and must be added to our known faces base. Known face base is created during the training phase of our models by specifying the size of it, –how many images per class we need– the quality and the dimensions of each image as well. As to the actual recognition process the standard approach is first to detect if there is a face in this image and then try to recognize it. That is the way we divide the main problem into Detection and Recognition.

4.1.1 Face Detection

4.1.1.1 First Approach on face detection

The first attempt for Face Detection was done using the *Dlib* [45] library’s module –`face_landmark_detection.py`– which uses a pre-trained data file of 68 face landmarks as input along with the input face-image to be detected. These 68-face landmarks were extracted from million face-images and they are representing points on faces like eyes’ shape and position, nose and mouth positions of a human face. In other words, they are facial features.

Inside the module we have **HoG**–Histogram of Oriented Gradients [15, 14] (Figures 4.1, 4.2) filter which is applied to the input image in order to extract face features and to remove the color and the background, because they are not useful information for the detection. Then we compare this HoG filtered image with the landmarks (Figure 4.3) file. In order to achieve this comparison and to decide whether a face exists, a **SVM** –*Support Vector Machine* [46]– classifier is used. With the SVM method, the detection module manages to classify image’s regions to those which contain faces and those which don’t. This is a pretty straight forward approach for accurate and reliable face detection.

The advantage of this method its invariance to light conditions, face expressions and face accessories like glasses. In addition its deployment speed is quite fast and it consumes a small number of system resources.

The only drawback of the above module is the need for a front-view (portrait) images, namely it does not perform well when there is a side-view (profile) image .

4. OUR APPROACH



Figure 4.1: A face image

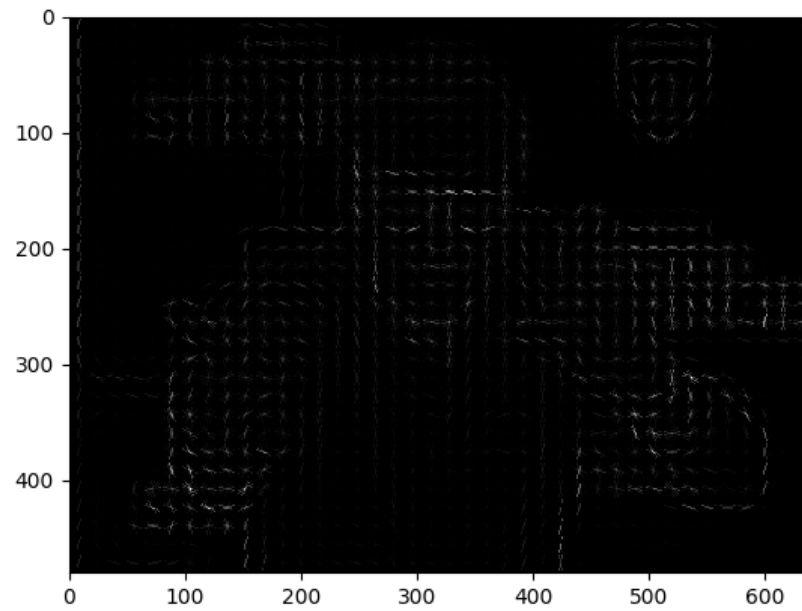


Figure 4.2: HoG filtered face image

4. OUR APPROACH

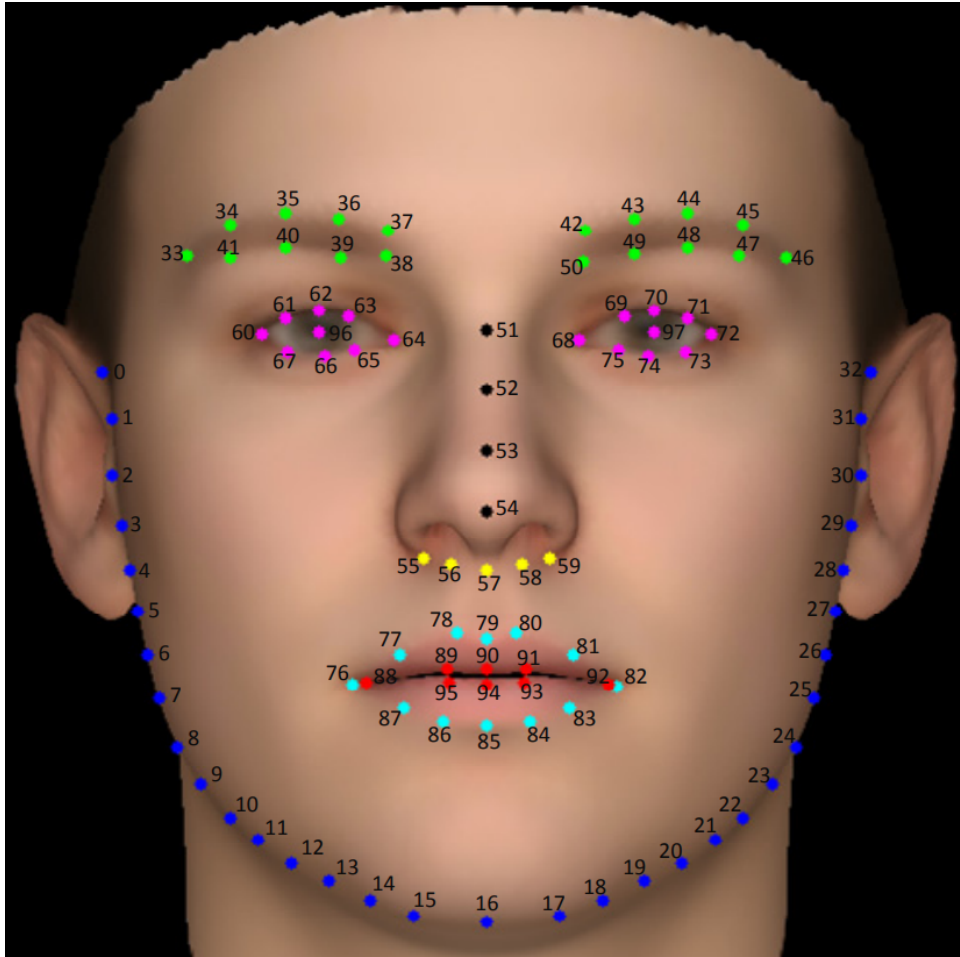


Figure 4.3: Landmarks face image

4.1.1.2 Second Approach on face detection

We tried a second approach to resolve the drawback of the previous one. In general, in video streams a person does not always stand in front of the camera looking straight at it, but there is a motion of the head.

The method used employed this time is based on CNN model implemented in *Dlib*'s module – [face_detection_cnn.py](#)– . More specifically, the CNN detection (Figure 4.4) module takes a pre-trained file as input –`mmod_human_face_detector.dat`– alongside with the image and detects the faces appearing inside. In this way, we have a more generalized purpose detector which manages to find faces with different angles. By using the convolutional neural network method we need larger execution time but we have better detection accuracy.



Figure 4.4: Detection with CNN, side-view (profile)image

4. OUR APPROACH

In conclusion, there is a minor disadvantage of both *Dlib*'s detection modules and that is they fail to detect faces that are smaller than 80×80 pixels and also they cannot handle very large size images. This case appears in a video stream when a person moves far away from the camera and that causes the face region of a frame to shrink. On the other hand, *OpenCV* detection algorithms recognize almost any size of the face in a frame but they lack accuracy as the detection bounding box is less related to the actual face region. So, in our case we made the assumption that there is no a person who walks away from the camera in a video stream and by filtering or resizing the images which are greater than 640×480 , we deliver a reliable and robust face detection approach.

4.1.1.3 Crop and Resize Image

The common step after detection of faces for each class –independently from the detection method– is to crop the detected face sub-region of the actual image and store it in the training set images (Figure 4.5). Then –in the training phase– because in an image there is a chance to find additional faces that are irrelevant to the class id of a person, we have to remove them manually from the data-set. If we want to avoid this process we have to use images with only one person appearing in them, but in our case, we use random images of a person from Google so this is almost impossible to happen.

After we have cleared our classes' training data there is one more step to perform before we feed them in our model. By cropping the detected face there is no guarantee that the output faces' images will all have the same dimensions! Feeding a Neural Network requires to specify the input matrix size, which is the same for all input data. So, we resize every face image to have a size of 256×256 . The choice of that image width and height was made by testing different numbers, and having as goal to minimize the input size while keeping as much quality information. At this point, it is good to mention that these steps are repeated following the detection on the test set also, except for the step of removing the unknown faces from the set.

The cropping is made inside the face detection script. When we detect a face we use the detection box margins and we save the ROI of the image where a face is. Then we use a resize module –*resize_img.py*– that we developed using *OpenCV* library to resize –to the specified height and width– all the images of a folder.

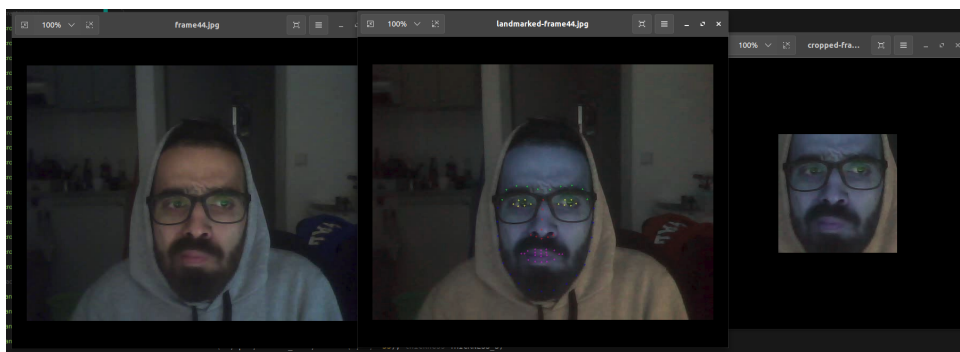


Figure 4.5: Detection Landmarks Crop

4. OUR APPROACH

4.1.2 Face Classification

Here comes the last step for the face recognition problem. Since we have created our training set from detected faces per class, now we have to develop a model which will be fitted to the data and will make predictions for the identity of the class. Before describing the implementation of our model we have to organize our data into a `.csv` file where we keep the file path, the name of the class and a label for it starting from '0'. This action makes our data-handling easier and faster by using the *pandas* library to read the actual class label and to load each image file. Last but not least, we have to encode our labels with the *one hot encoding* method. In a nutshell, a label is an integer number from 0 till $K - 1$, where K is the number of different classes.

One hot encoding (Figure 4.6) is a process by which categorical variables like labels, are converted into a form that could be provided to Machine Learning algorithms to do a better job in prediction. In short, this method produces a vector with length equal to the number of categories in the data set and the predicted class has the value '1' while all the others have '0'. This makes the result of our classifier more understandable for the model.

Label Encoding			One Hot Encoding			
Food Name	Categorical #	Calories	→			
Apple	1	95	Apple	Chicken	Broccoli	Calories
Chicken	2	231	1	0	0	95
Broccoli	3	50	0	1	0	231
			0	0	1	50

Figure 4.6: One Hot Encoding Example

Let's now talk about the Convolutional Neural Network model (Figures 4.7, 4.8) ¹ which plays the role of the classifier. After we made the necessary pre-processing of the data we load images to a large *Numpy* array of shape

$\#images, 256, 256, 3$

. Since our images are represented in *RGB* scale our input has 3 color channels– the array of data is called X_{input} and the one hot encoded labels are stored Y_{input} *Numpy* array. The first block of our

¹Model developed with Keras

model will have a convolution layer with kernel shape 3×3 , and a number of output channels of 32. After that we have a *Rectified linear Unit –ReLU* as the activation method of our layers neurons. At last we have a Max Pool layer with strides and kernel shape $[1, 1, 1, 1]$. The second block is the same as the first one with the only change being the stride size, which here is $[1, 2, 2, 1]$.

The third and the fourth blocks are identical, they are composed of a convolutional layer with kernel 3×3 and output channels number 64, followed by a ReLU activation function and then the Max Pool with $[1, 2, 2, 1]$ shape for strides and kernel shape as the second block. Next we flatten out our fourth of Max Pool's output by reshaping to a $2 - D$ array from a $4 - D$. The reshaped array is moved forward to the pipeline as input to the first Dense layer which has 64 for output channels and a ReLU activation. One more step is to add the next Dense Layer which has K channels as outcome, in our test case $K = 2$ –number of classes to predict–. Between the Dense layers we have a dropout layer with a factor of dropped neurons equals to 0.4. The last layer of our model is the *Softmax* layer, which converts the prediction result to normalized probabilities per class.

As for the back-propagation method of the model where loss between the predicted and the true classes (labels) is computed, we use *categorical-cross entropy* loss with *Adam* as optimization method and a learning rate of 0.001 –the default of Adam optimizer. While training with our data we use a validation set which is actually 25% of `train_data_size`, which improves the model fit and reduces the changes of over-fitting.

4. OUR APPROACH

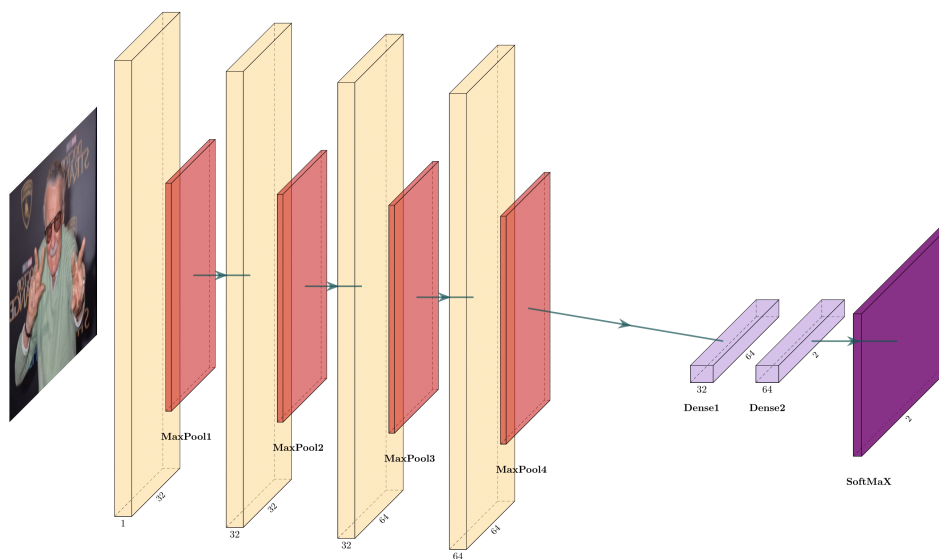


Figure 4.7: CNN architecture of classifier [5]

More Detailed Model Description:

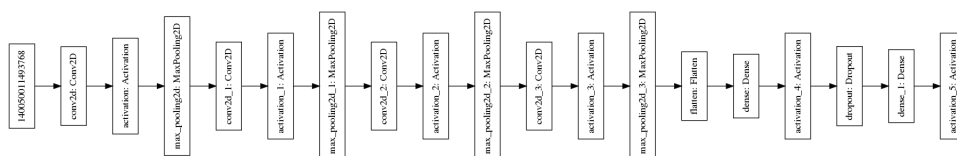


Figure 4.8: Detailed FR CNN layers

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 254, 254, 32)	896
activation (Activation)	(None, 254, 254, 32)	0
max_pooling2d (MaxPooling2D)	(None, 254, 254, 32)	0
conv2d_1 (Conv2D)	(None, 252, 252, 32)	9248
activation_1 (Activation)	(None, 252, 252, 32)	0
max_pooling2d_1 (MaxPooling2D)	(None, 126, 126, 32)	0
conv2d_2 (Conv2D)	(None, 124, 124, 64)	18496
activation_2 (Activation)	(None, 124, 124, 64)	0
max_pooling2d_2 (MaxPooling2D)	(None, 62, 62, 64)	0
conv2d_3 (Conv2D)	(None, 60, 60, 64)	36928
activation_3 (Activation)	(None, 60, 60, 64)	0
max_pooling2d_3 (MaxPooling2D)	(None, 30, 30, 64)	0
flatten (Flatten)	(None, 57600)	0
dense (Dense)	(None, 64)	3686464
activation_4 (Activation)	(None, 64)	0
dropout (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 2)	130
activation_5 (Activation)	(None, 2)	0
Total params: 3,752,162		
Trainable params: 3,752,162		
Non-trainable params: 0		

Figure 4.9: FR Model's parameters summary

In Figure 4.9 we view the number of model's parameters per layer, and the total number of parameters to be trained.

4.2 Speaker Recognition (SR)

In this section we describe the methods followed all the way down to the actual Speaker classification module. We analyze the steps that went wrong in order to explain how through this process we managed to gain experience to finally solve the problem. At first, it is good to mention that for our speech data we use some speaker classes that are included in the VoxCeleb [42] data-set.

4.2.1 Voice Features Extraction

The generic method used for voice features extraction is described in the Section 2.2.1. Here we would like to view the differences in detail, for the settings we used in filter banks for speaker recognition. The first difference is that the size of each frame is 0.025 sec, with a frame stride of 0.05 sec, which is referred in the filter banks algorithm (*split signal step*) in Section 2.2.1.

In practice for our design we use the *Filter banks* method with 64 filters to split a sound file in windows with sampling time *5ms*. Then in order to represent the correlation between those windows –as sound is a sequence of current, past and future windows–, we stacked these sampled windows of *5ms* into 8 past plus 1 current plus 8 future windows. So, we form small groups of total size 17 windows. For the first utterance the past frames are repetitions to the left of the first frame, while for the last utterance we repeat it to the right to complete the number of future-stacked windows. Due to all this process each group of 17 forms a 2 – *D* image of size 64.

As a result from all the above steps we get the features of a sound file *.wav* and convert it to an *Numpy* array with shape

$$\#windows.in.wav, 64, 17, 1$$

(sound files, are represented as "spectral-image"-vectors that have one channel) and store them as binary *.cmp* files. All these binary files will be the input data for the Speech Recognizer models. Also, it is important to normalize our data with the mean and standard deviation of the training set, before we actually feed them to our models. This is crucial both for the training, validation and for the prediction phase in Machine Learning with Neural Networks.

4.2.2 First Approach Model for Speech Recognition

An initial thought for Speech Recognition was to use a previous project’s model developed for *Anti-Spoofing Verification* (ASV). This model was an approach for the ASV contest where we tried to use a Convolutional Neural Network (Figures 4.10, 4.11) to predict if a sound utterance is genuine speaker or a spoof attack. The main concept was to modify the model based on the Transfer Learning approach to fine tune the fully connected layers to a new data set –speakers– and predict the identity of them.

We were inspired to choose a model architecture similar to paper [47] which suggests to use CNN for robust speech recognition instead of a Recurrent Neural Network. We kept the number of layers proposed in this publication but we changed the output channels per layer according to the project requirements.

Our development of the CNN architecture was customized at each step by us, so we used native Tensorflow framework and not some high level API like Keras. The architecture itself has 10 convolution layers namely it is a Deep Neural Network and that could add complexity to the model. In general, a Tensorflow model is split to the code that defines the graph of the Neural Network and to the session part which is the execution of that code. We will focus on the graph design analysis in the next paragraph.

We start by saying that the graph is divided in two parts; to the part used for training and the part is used for validation. The only difference is that in the part used for validation there is no any parameter update, it is strictly used to get along with the training process and check if we dealing with under-fitting or over-fitting problems. Both parts of the graph include the model architecture.

Within the model architecture we have 10 *convolution* layers with kernel size 3×3 and we gradually increase output channels from 1 up to 64. A convolution layer is followed by a *batch normalization* layer and then we group each convolution–batch normalization pair of layer to a block with a ReLU activation method. A *Max Pool* layer succeeds each block. We have 5 blocks of this type, so there are 5 max pool layers. The first two max pool layers have a stride and kernel shape of $[1, 2, 1, 1]$ and the remaining three have a $[1, 2, 2, 1]$ shape for both attributes. When our pipeline reaches the last max pool layer we reshape its output –Flatten Out– to reduce dimensionality to $2 - D$.

Afterwards we have two *Fully Connected* layers one with 128 and the other with 256 output channels. Between those layers there is a dropout factor of 0.4 and *ReLU* is used in each one for neurons activation. Before reaching the final layer we add a *Dense* layer with 2 output channels. The final step once again is to take the *Softmax* of predicted labels –logits–.

Regarding the loss function we used the softmax cross entropy loss and took as input the output of the logit layer, which is the penultimate layer of our CNN model (the last one is the softmax layer,

4. OUR APPROACH

which converts the predicted labels into probabilities per class). We also used the Adam optimizer with a learning rate of 0.0001.

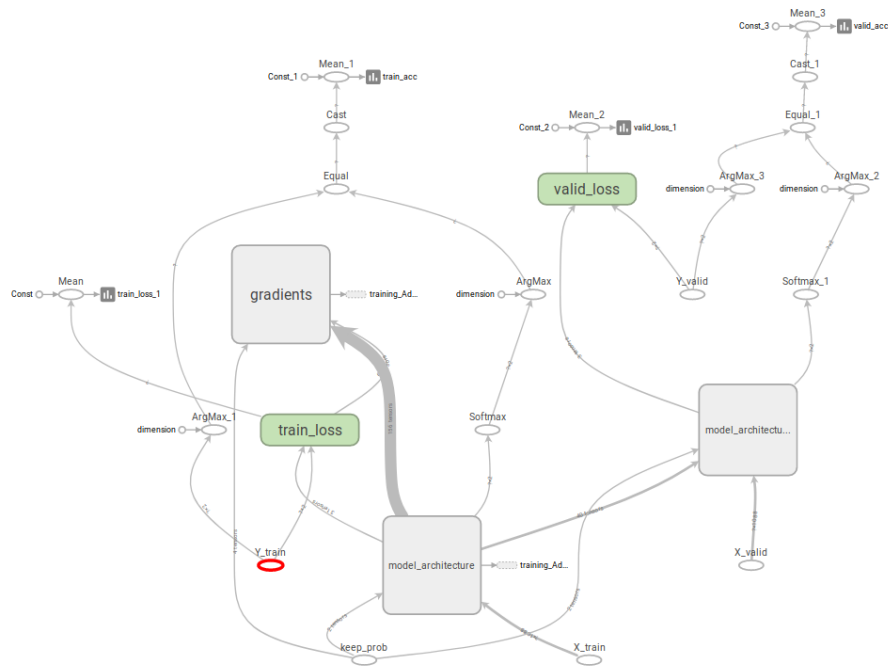


Figure 4.10: Tensorflow Graph of the model

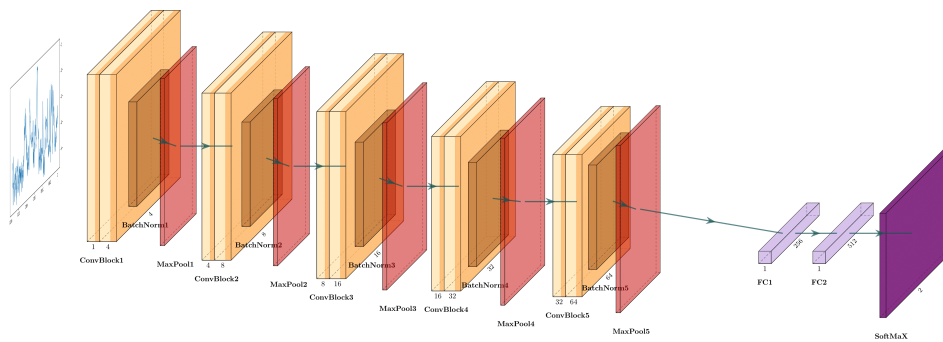


Figure 4.11: CNN architecture layers [5]

4.2.3 Speaker Recognition Classifier

According to the previous section the model we already have trained did not manage solving the recognition problem as expected and that is why we designed a new model (Figures 4.12, 4.13, 4.14). This model is exactly the same architecture as the Face Classifier but we removed the last convolution layer and reduced the layers from 4 to 3. As mentioned in speech feature extraction (Section 2.2.1) a frame has a size of 64×17 which is much smaller than the 256×256 of the face model and for that reason we have to use fewer layers to our model. From this module we get as result a class –the class with the higher probability value– and the probabilities per class in which later a threshold will be applied to make the final decision.

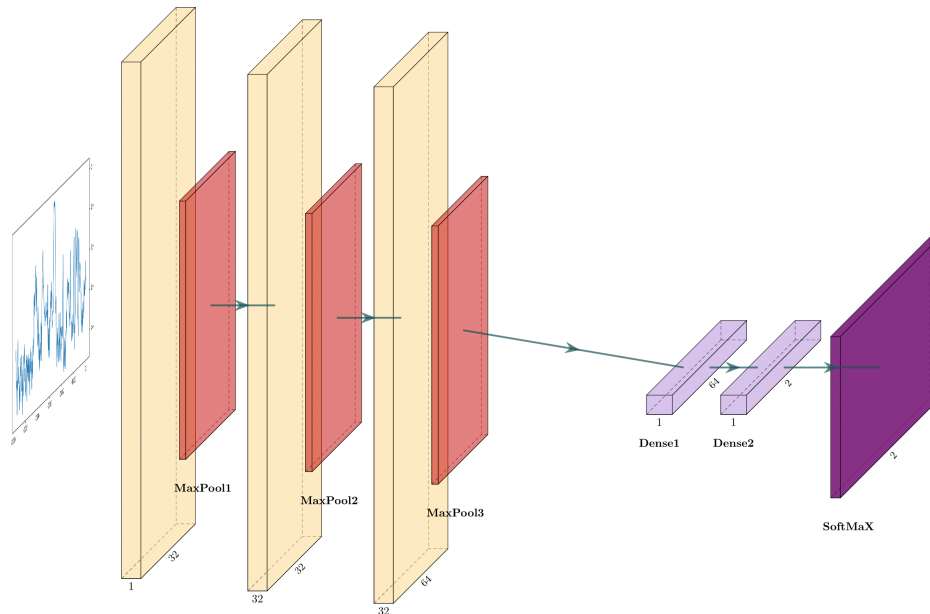


Figure 4.12: CNN architecture of SR classifier [5]

4. OUR APPROACH

More Detailed Model Description:

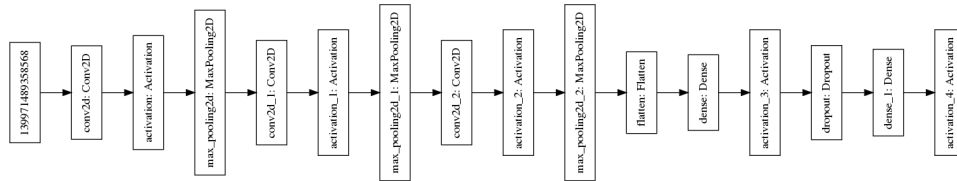


Figure 4.13: Detailed SR CNN layers

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 28, 28, 15, 32)	320
activation (Activation)	(None, 28, 28, 15, 32)	0
max_pooling2d (MaxPooling2D)	(None, 14, 14, 15, 32)	0
conv2d_1 (Conv2D)	(None, 10, 10, 13, 32)	1664
activation_1 (Activation)	(None, 10, 10, 13, 32)	0
max_pooling2d_1 (MaxPooling2D)	(None, 5, 5, 6, 32)	0
conv2d_2 (Conv2D)	(None, 28, 4, 64)	18496
activation_2 (Activation)	(None, 28, 4, 64)	0
max_pooling2d_2 (MaxPooling2D)	(None, 14, 2, 64)	0
flatten (Flatten)	(None, 1792)	0
dense (Dense)	(None, 64)	114752
activation_3 (Activation)	(None, 64)	0
dropout (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 3)	195
activation_4 (Activation)	(None, 3)	0
Total params: 143,011		
Trainable params: 143,011		
Non-trainable params: 0		

Figure 4.14: SR Model's parameters summary

4. OUR APPROACH

The Figure 4.14 is the speaker recognition’s model parameters to be trained (per layer and the total number of parameters).

4.3 FR & SR for Person Indexing in video

The last part of this project was to combine the two subsystems *Face and Speaker Recognition* on a video stream for identifying a person in a specific time slot of the video. Hence, we use the late integration for our two modes of features instead of the early integration at the feature level. On one hand early integration is not straightforward and is quite complex to fuse facial and speech features in general; in our use case, the two feature types are very different and independent and also might cause some interference if we try to fuse them. In the contrary, with late integration we managed to combine the features at the hypothesis level, as a kind of system combination. Each component system generates a hypothesis based on one feature type and then we combine the results of both subsystems to generate a better decision. Furthermore, if one of the two subsystems fails to recognize efficiently a person we can still use the second, to obtain an accurate result. We basically, concatenate the outputs of the two higher-level modules to make the predictions for classes per time slot –from the sound utterances– correlated with the time stamp of an image-frame for which also a class is predicted. In order to separate the decision into known and unknown identity, we take a threshold value chosen accordingly to the predict probabilities of the FR or SR module. The output of our system is a `.txt` file with the information of indexing and class identification per video’s time slot. The output example is shown into the flow chart in Figure 4.15.

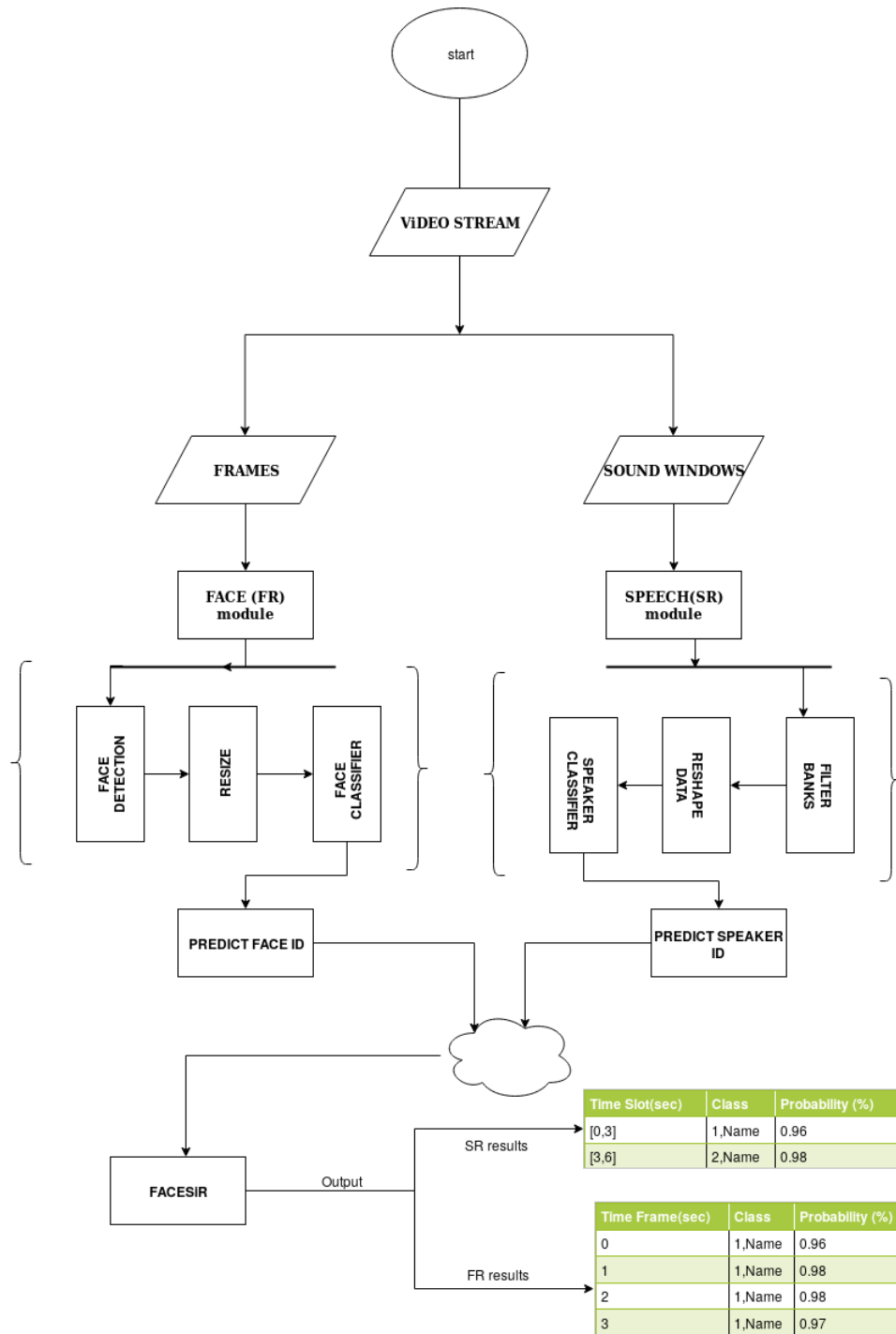


Figure 4.15: Flow Chart of FACESIR system

4. OUR APPROACH

Chapter 5

Results

In this chapter, we present the results of our work according to each part of the entire project. First, we need to clarify that our approach gives accurate predictions both on Facial and Voice data, however it is tested only on a small number of classes (2 or 3). Our work has not been optimized for online processing (recognition) of videos. Instead, it provides a promising proof-of-concept, but it is not the final solution to the problem.

Recognition experiments have been conducted with three person identities (classes), specifically the actors Stan Lee, Jack Black, and Samuel L. Jackson, on a variety of free video clips from YouTube. The results from the combined recognition were more than promising for this small set of target identities. Interestingly, the training set for these identities was not formed using video streams, but from independent set of images and sounds.

When it comes to hardware specifications to solve these kind of problems, the answer is GPUs. Because of the complexity of the project, a high-end GPU was necessary. Nvidia Titan Xp (11 GB)([Grant Program](#)) was used to reach our goal and solve the problem, saving significant time.

The Face classifier module was trained for a number of 60 epochs and with a batch size equal to 32. As for the Speaker classifier due to the smaller size of input a batch size of 256 was used for 15 epochs.

5. RESULTS

Face Classifier Plots:

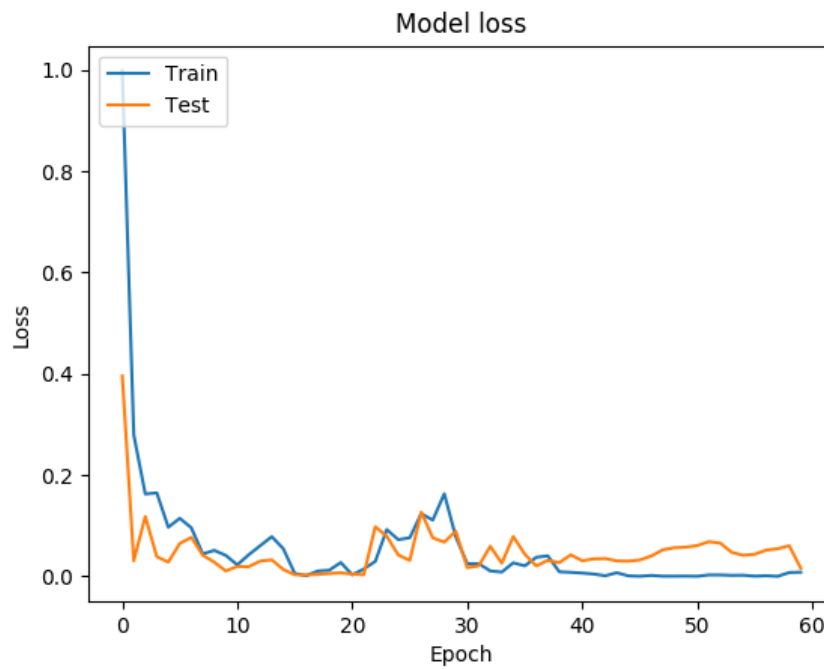


Figure 5.1: Face classification Loss plot

In Figures 5.1, 5.2 we view the performance of the classifier which is about **97%** on the validation set feed (see "Test" line in the plot). We also evaluated the model after training and we kept the accuracy percent of the classifier, showing a value of **94%**, as a measure.

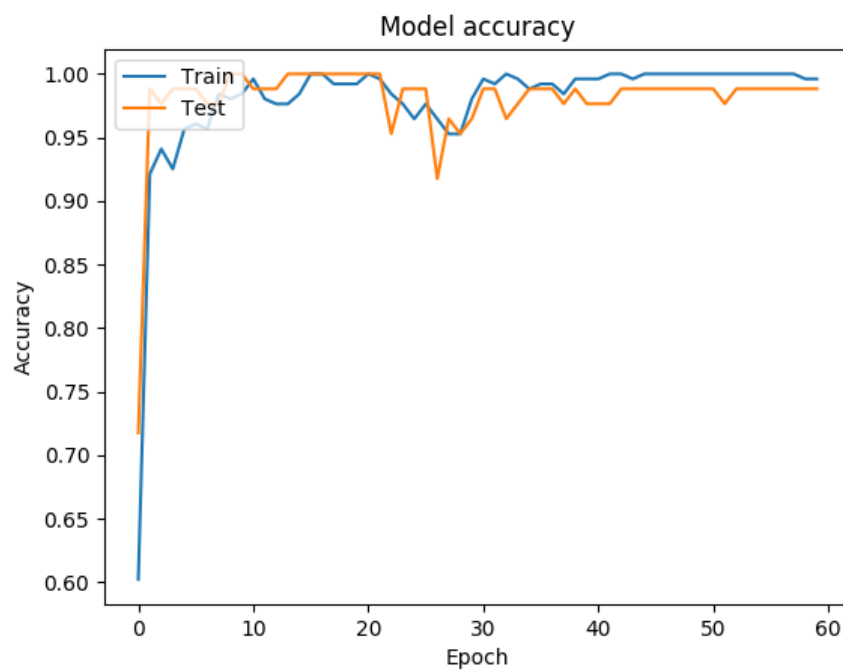


Figure 5.2: Face classification Accuracy plot

Failed Speech ASV Model Plots:

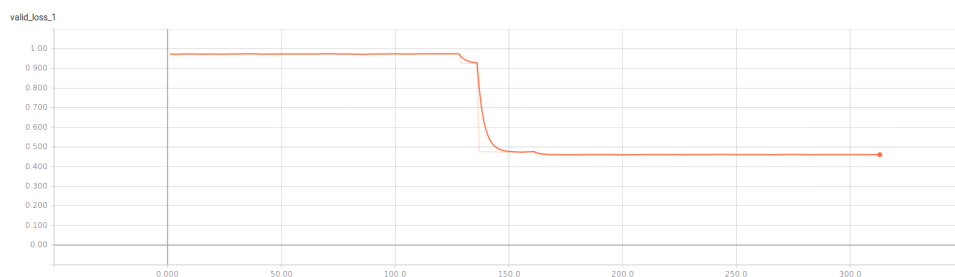


Figure 5.3: First Speaker Recognition model Loss plot

5. RESULTS

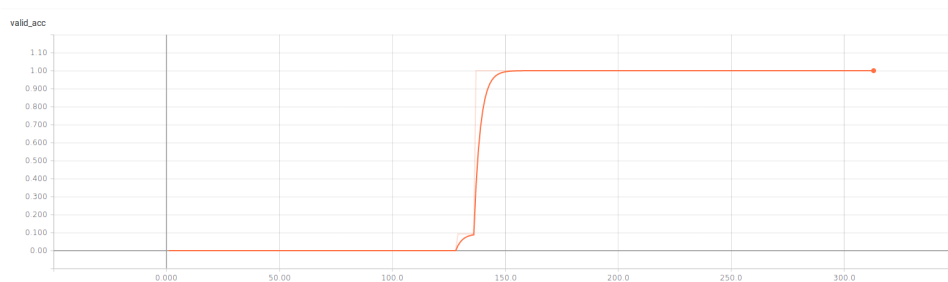


Figure 5.4: First Speaker Recognition model Accuracy plot

The first approach for speaker recognition is shown in Figures 5.3, 5.4. If we check carefully the validation loss and accuracy, the model seems to respond well; actually, the loss value was around 0.45 in Figure 5.3. The best loss value we managed to get was 0.25 (after few runs) and it was achieved after fine tuning the last layers. Before the transfer learning step (fine tuning) the best loss accuracy was around 60% . The main cause of these poor performance after hours of testing and development seemed to be the complexity of the model (too many parameters to be learned). So, for all these reasons we moved to the next approach which proved to be much better.

Speaker Recognition Model Plots:

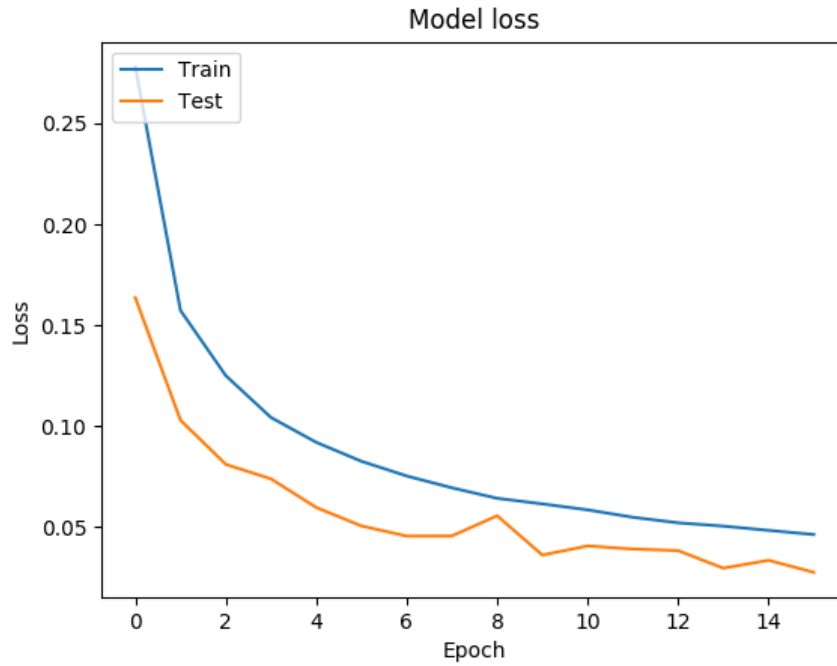


Figure 5.5: Speaker Recognition model Loss plot

As we see in Figures 5.5, 5.6 the validation accuracy of the speaker model is reaching **99%**, while the evaluation accuracy after training, was found to be **98%**. For both the FR and the SR subsystems the accuracy value –after training– is calculated as the mean accuracy from 10 independent runs.

5. RESULTS

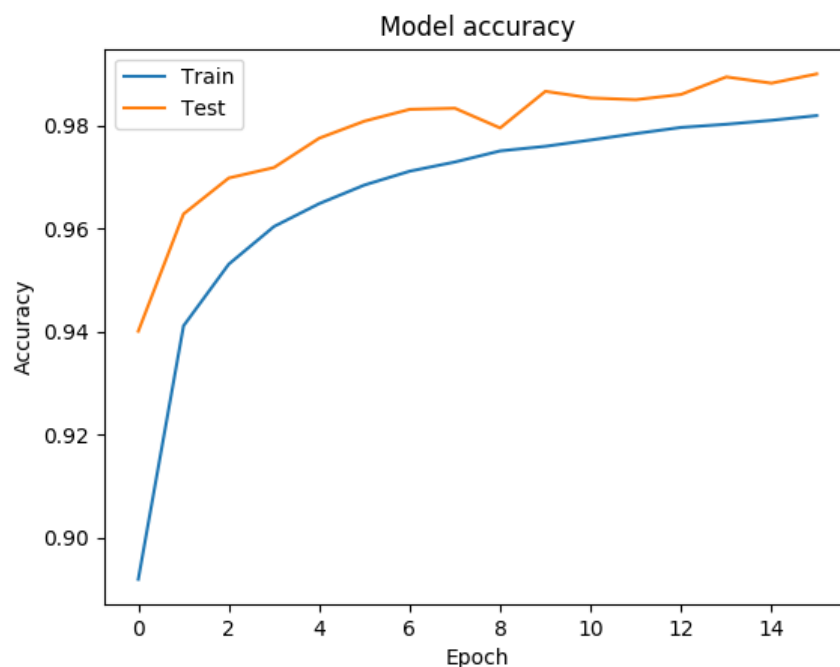


Figure 5.6: Speaker Recognition model Accuracy plot

Example depth-complexity reduces performance:

In Figure 5.7 we try to show how the increase of depth or size of output channel in a layer of the model can reduce its performance. In this example, we used the Face classifier to show this info because it is larger, referring to the total number of parameters it has. Figure 5.7 must be compared to Figure 4.9.

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 254, 254, 32)	896
activation (Activation)	(None, 254, 254, 32)	0
max_pooling2d (MaxPooling2D)	(None, 254, 254, 32)	0
conv2d_1 (Conv2D)	(None, 252, 252, 32)	9248
activation_1 (Activation)	(None, 252, 252, 32)	0
max_pooling2d_1 (MaxPooling2D)	(None, 126, 126, 32)	0
conv2d_2 (Conv2D)	(None, 124, 124, 64)	18496
activation_2 (Activation)	(None, 124, 124, 64)	0
max_pooling2d_2 (MaxPooling2D)	(None, 62, 62, 64)	0
conv2d_3 (Conv2D)	(None, 60, 60, 64)	36928
activation_3 (Activation)	(None, 60, 60, 64)	0
max_pooling2d_3 (MaxPooling2D)	(None, 30, 30, 64)	0
flatten (Flatten)	(None, 57600)	0
dense (Dense)	(None, 128)	7372928
activation_4 (Activation)	(None, 128)	0
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 64)	8256
activation_5 (Activation)	(None, 64)	0
dense_2 (Dense)	(None, 2)	130
activation_6 (Activation)	(None, 2)	0
Total params: 7,446,882		
Trainable params: 7,446,882		
Non-trainable params: 0		

Figure 5.7: Increased FR parameters example

5. RESULTS

In Figure 5.7 we can see which layer has a slightly change by increasing the output channels from 64 to 128 in comparison to Figure 4.9. We also see that this change increases the parameters to be learned up to 7,446,882 from 3,752,162; they are almost doubled. Then the loss and accuracy plots (Figures 5.8, 5.9) are included to spot out the difference in performance.

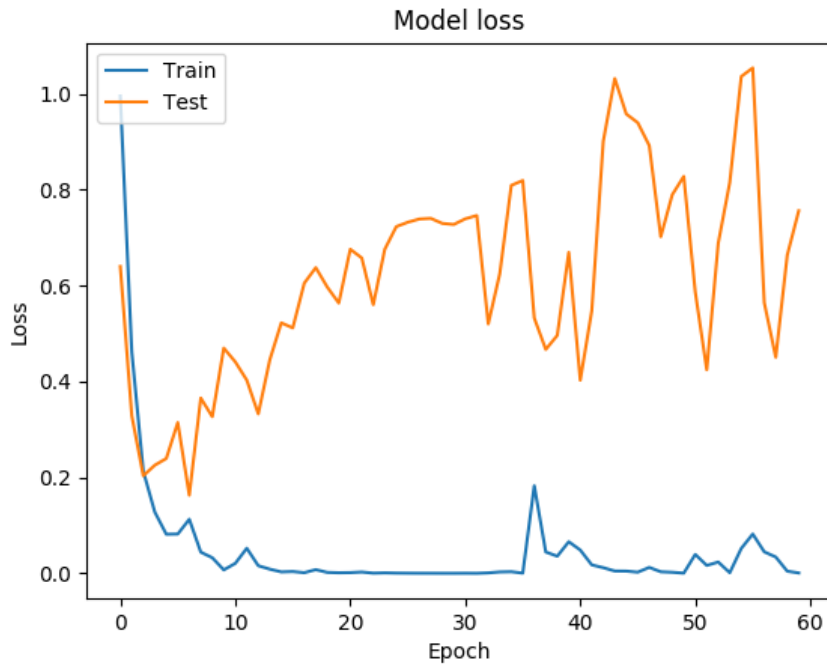


Figure 5.8: Increased FR parameters Loss plot

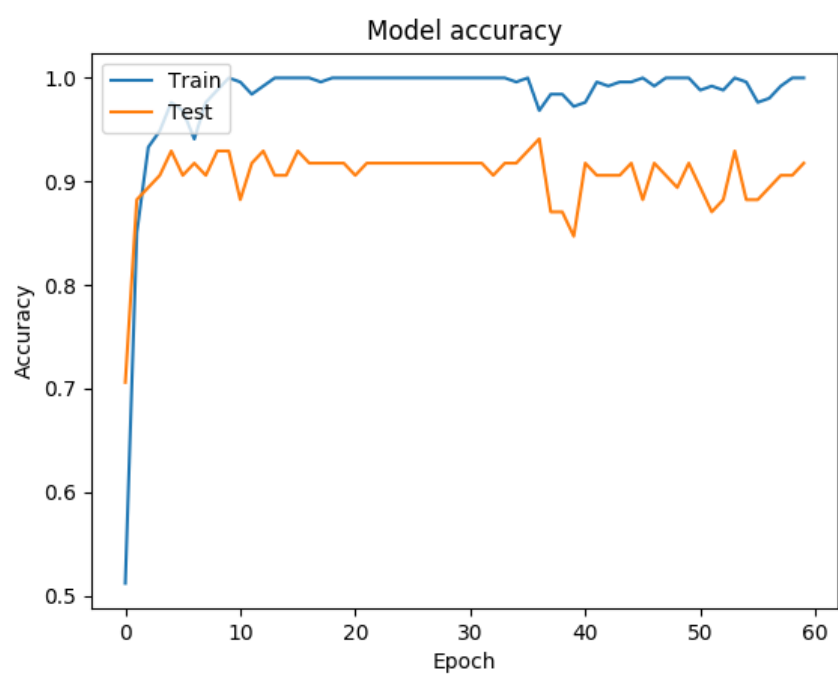
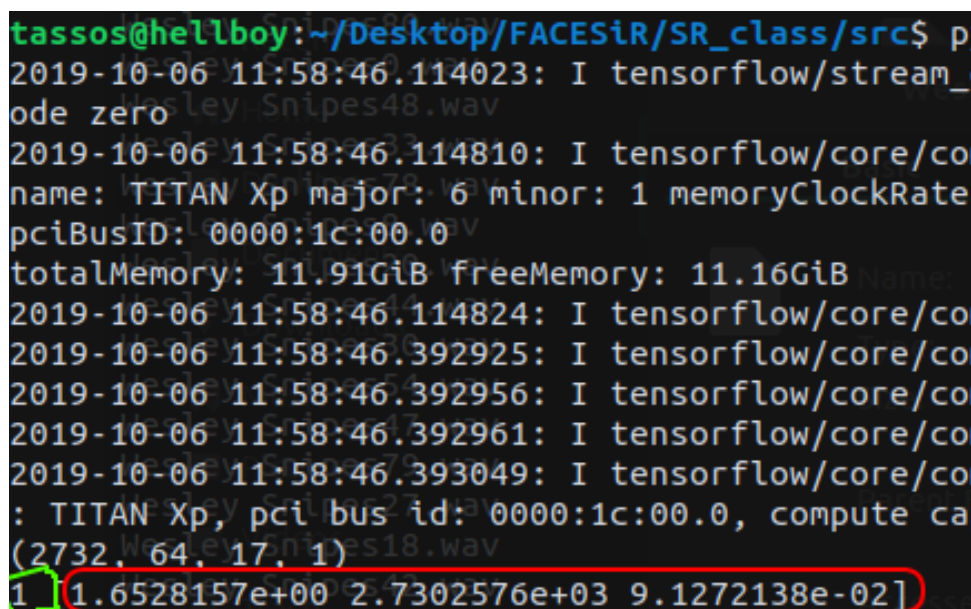


Figure 5.9: Increased FR parameters Accuracy plot

5. RESULTS

How predictions are made

In this case, we used the SR module as example, to display how predictions are made and how we obtain the final decision; a similar process is used with the FR module. With the red circle we highlight the sum of prediction probabilities –cause a sound file is composed of many frames with the same label, we take with the sum probabilities per column–. Alternatively, we could use one frame from each sound file to make the prediction instead of the whole file but this way the difference between known and unknown identity is clearer. As for the green circle, this is the predicted class label.



```
tassos@hellboy:~/Desktop/FACESiR/SR_class/src$ p
2019-10-06 11:58:46.114023: I tensorflow/stream_
ode zero
2019-10-06 11:58:46.114810: I tensorflow/core/co
name: TITAN Xp major: 6 minor: 1 memoryClockRate
pciBusID: 0000:1c:00.0
totalMemory: 11.91GiB freeMemory: 11.16GiB
2019-10-06 11:58:46.114824: I tensorflow/core/co
2019-10-06 11:58:46.392925: I tensorflow/core/co
2019-10-06 11:58:46.392956: I tensorflow/core/co
2019-10-06 11:58:46.392961: I tensorflow/core/co
2019-10-06 11:58:46.393049: I tensorflow/core/co
: TITAN Xp, pci bus id: 0000:1c:00.0, compute ca
(2732, 64, 17, 1)
1 [1.6528157e+00 2.7302576e+03 9.1272138e-02]
```

Figure 5.10: Non Normalized input example

Figure 5.10 shows that in case we have not normalized our input data before inserting them to the system for prediction we get a miss-classification result.

```
tassos@hellboy:~/Desktop/FACESiR/SR_class/src$  
2019-10-06 11:59:04.154669: I tensorflow/stream_executor/cuda/cuda_device_reporter.cc:249: Found 1 NVIDIA GeForce GTX 1080 Ti device with 11.51GiB of memory: device (0) = 0  
2019-10-06 11:59:04.155878: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1499: Found device TITAN Xp major: 6 minor: 1 memoryClockRate (GHz) = 1.5925. DeviceMemoryLimit = 11.51GiB  
pciBusID: 0000:1c:00.0  
totalMemory: 11.91GiB freeMemory: 11.16GiB  
2019-10-06 11:59:04.155899: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1626: Successfully initialized GPU device: TITAN Xp, pci bus id: 0000:1c:00.0, compute capability: 3.7  
2019-10-06 11:59:04.404289: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1499: Found device TITAN Xp major: 6 minor: 1 memoryClockRate (GHz) = 1.5925. DeviceMemoryLimit = 11.51GiB  
2019-10-06 11:59:04.404320: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1626: Successfully initialized GPU device: TITAN Xp, pci bus id: 0000:1c:00.0, compute capability: 3.7  
2019-10-06 11:59:04.404328: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1499: Found device TITAN Xp major: 6 minor: 1 memoryClockRate (GHz) = 1.5925. DeviceMemoryLimit = 11.51GiB  
2019-10-06 11:59:04.404424: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1626: Successfully initialized GPU device: TITAN Xp, pci bus id: 0000:1c:00.0, compute capability: 3.7  
(868, 64, 17, 1)  
2 [ 16.698378 9.715815 841.58624 ]
```

Figure 5.11: Normalized input example

Figure 5.11 shows the right classification result after normalizing the input data. These examples were added to point out the importance of input normalization (*use of the same mean $\mathbf{E}(\mathbf{X}_{\text{train}})$ and variation $\text{Var}(\mathbf{X}_{\text{train}})$ values*) both at the training and the testing phase.

5. RESULTS

```
tassos@hellboy:~/Desktop/FACESiR/SR_class/src$  
2019-10-06 13:28:46.209231: I tensorflow/stream_executor/cuda/cuda_device_reporter.cc:38: #createDevice zero  
2019-10-06 13:28:46.210437: I tensorflow/core/common_runtime/gpu/gpu_device.cc:49: name: TITAN Xp major: 6 minor: 1 memoryClockRate(GHz): 1.59  
pciBusID: 0000:1c:00.0  
totalMemory: 11.91GiB freeMemory: 11.15GiB  
2019-10-06 13:28:46.210457: I tensorflow/core/common_runtime/gpu/gpu_device.cc:49: TITAN Xp, pci bus id: 0000:1c:00.0, compute capability: 6.0  
2019-10-06 13:28:46.464432: I tensorflow/core/common_runtime/gpu/gpu_device.cc:49: TITAN Xp, pci bus id: 0000:1c:00.0, compute capability: 6.0  
2019-10-06 13:28:46.464482: I tensorflow/core/common_runtime/gpu/gpu_device.cc:49: TITAN Xp, pci bus id: 0000:1c:00.0, compute capability: 6.0  
2019-10-06 13:28:46.464490: I tensorflow/core/common_runtime/gpu/gpu_device.cc:49: TITAN Xp, pci bus id: 0000:1c:00.0, compute capability: 6.0  
2019-10-06 13:28:46.464581: I tensorflow/core/common_runtime/gpu/gpu_device.cc:49: TITAN Xp, pci bus id: 0000:1c:00.0, compute capability: 6.0  
(804, 64, 17, 1)  
1 [173.8095 458.75537 171.4348]
```

Figure 5.12: Unknown class input example

Comparing the sum of probabilities between Figure 5.11 with the one in Figure 5.12 helps to understand how we find the threshold we need to apply. Actually the way we suggest to define a global threshold is by using only one frame per sound file to make the prediction because each file has a different number of frames and that gives a variation to the sum of the predicted probabilities. In addition, our work demonstrates that we need a class number larger than or equal to 3 ($K \geq 3$) (each class is a person's identity) if we want to detect also the case of unknown identity. This is true, because in the binary classification case where $K = 2$ an unknown class will be matched to one of the known with a high probability and we can't define a threshold. In the end by keeping the information for persons appearing or speaking in a specific time slot of the video helps us to make better decisions. In more detail with this hybrid system we are able to overcome the drawback of each subsystem when standing alone. Take as an example the case of a small (less than 70×70) face ROI which our face detection module fails to detect; in such a case the speaker recognition is called to give an answer.

5. RESULTS

Chapter 6

Conclusion

6.1 Conclusion

This thesis describes a new person indexing approach for the video streams implemented in *Tensorflow* framework and *Keras API* based on Convolutional Neural Networks. The person indexing application takes as input a video stream and splits it into image frames and then into sound windows, which are correlated with the images using timestamps. The software is divided into two smaller applications the Face Recognition and the Speaker Recognition. The Face Recognition part of the code gives a confidence value (%) for each frame while the Speaker Recognition gives confidence for more than one frames (a possibility of a speaker to be in more than two continuous image frames – a sound window). After that, the two outputs of each sub-system are combined using a post classifier. Finally, we have as an outcome a probability value for each known identity and recognized inside the video. The proposed person indexing approach gives more accurate results than previous implementation methods based only on facial features and tries to improve confidence when there is a lack of efficient data for one of the two sub-systems.

6.2 Future Work

6.2.1 Multi Person Detection and Classification

The next idea to be added as a feature in our implementation is to detect and recognize more than one face in a frame of a video stream. From the side of the face detection task, this feature already exists, in the framework that is used by *Dlib* or *OpenCV*; but, from the side of simultaneously multi-face and multi-speaker recognition, it is quite a tricky task and demands extra time of processing, testing and mostly it adds complexity to our problem.

Another must-do improvement of our work would be to use another CNN, or some other type of Artificial Neural Network, as a post-classifier for the final decision in this hybrid system. That was a goal from the start of the project, but, we keep it in progress for the upcoming versions of our software.

Also, a much needed feature for improvement would be to find out a more elegant way to combine and present the *FACESiR* module's results instead of saving them into a `.txt` file or even further, to develop a Graphical User Interface (GUI) application.

6.2.2 More Testing and Evaluation

The main concern for future work and improvements in this project, will be the further testing on new data sets with a bigger variety and complexity. At the same time, we would like to test some more cases in our models' architecture, like trying different kinds of optimizers, loss functions, and activation functions. This will give us results for comparison to mark how good our implementation is and of course to provide more robust and accurate confidence values.

6.2.3 Outcome

If we view beyond the implementation of this project, the next goal to chase is to create a software application or a platform, as a long-term product from this diploma thesis. An actual software product that will be useful to people, like content creators, who would like to search for a person in video streams and they demand to reduce editing time, or a tool to be used for promotion tasks by advertising and marketing companies.

6.3 Lessons Learned

Throughout the last year spent on my diploma thesis, I have learned how important it is to have a clear mindset and be patient, in order to overcome difficulties that will come in your way when you have

a goal. In addition, I learned that you have to move one step at a time, solving a smaller problem first and then moving to the next one. But, the biggest lesson that I got from it, was not to over-think about the design and analysis of the solution; you have to act as soon as you think something. Then, spend the rest of the time to prove if your approach is either right or wrong. This is what will keep you moving forward and make progress.

6. CONCLUSION

References

- [1] Fayek, H.: Speech Processing for Machine Learning: Filter banks, Mel-Frequency Cepstral Coefficients (MFCCs) and What's In-Between. (Apr 21, 2016) <https://haythamfayek.com/2016/04/21/speech-processing-for-machine-learning.html>. , , , 10, 12, 13, 14
- [2] Administrator: Communication between neurons. ThreeKeyYears. (January 2017) <https://threekeyyears.org/portfolio-items/babys-brain-begins-now-conception-to-age-3/ch1-fg2-communication-between-neurons/>. , 23
- [3] Seif, G.: Towards Data Science: 7 Practical Deep Learning Tips. Medium Towards Data Science. (February 2018) <https://towardsdatascience.com/7-practical-deep-learning-tips-97a9f514100e>. , 25
- [4] Saha, S.: A Comprehensive Guide to Convolutional Neural Networks. Medium Towards Data Science. (December 2018) <https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>. , 27, 28
- [5] Iqbal, H.: Harisiqbal88/plotneuralnet v1.0.0 (December 2018) , , , 58, 62, 63
- [6] dev team, O.: Face Recognition with OpenCV. OpenCV. https://docs.opencv.org/2.4/modules/contrib/doc/facerec/facerec_tutorial.html. 5, 9
- [7] Kanade, T.: Picture processing system by computer complex and recognition of human faces (November 1973) 5
- [8] Turk, M.A., Pentland, A.P.: Face recognition using eigenfaces. In: Proceedings. 1991 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. (June 1991) 586–591 6
- [9] Kshirsagar, V.P., Baviskar, M.R., Gaikwad, M.E.: Face recognition using eigenfaces. In: 2011 3rd International Conference on Computer Research and Development. Volume 2. (March 2011) 302–306 6

REFERENCES

- [10] Belhumeur, P.N., Hespanha, J.P., Kriegman, D.J.: Eigenfaces vs. fisherfaces: recognition using class specific linear projection. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **19**(7) (July 1997) 711–720 [8](#)
- [11] Viola, P., Jones, M.: Rapid object detection using a boosted cascade of simple features. In: *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001. Volume 1.* (Dec 2001) I–I [9](#)
- [12] Rosebrock, A.: Local Binary Patterns with Python and OpenCV. Pyimagesearch. (December 7 2015) <https://www.pyimagesearch.com/2015/12/07/local-binary-patterns-with-python-opencv/>. [9](#)
- [13] Ojala, T., Pietikäinen, M., Mäenpää, T.: Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *IEEE Transactions on Pattern Analysis & Machine Intelligence* (7) (2002) 971–987 [9](#)
- [14] Dalal, N., Triggs, B.: Histograms of oriented gradients for human detection. In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05).* Volume 1. (June 2005) 886–893 vol. 1 [9](#), [49](#)
- [15] Mallick, S.: Histogram of Oriented Gradients. Learn OpenCV. (December 2016) <https://www.learnopencv.com/histogram-of-oriented-gradients/>. [9](#), [49](#)
- [16] Macmillan Dictionary: Voice recognition definition. <https://www.macmillandictionary.com/dictionary/british/voice-recognition>. [10](#)
- [17] Chakraborty, K., Talele, A., Upadhyay, S.: Voice recognition using mfcc algorithm. *International Journal of Innovation Research in Advanced Engineer (IJIRAE)* ISSN 2349–2163 **1**(10) (2014) 158–161 [10](#)
- [18] Prahallad, K.: Speech Technology: A Practical IntroductionTopic: Spectrogram, Cepstrum and Mel-Frequency Analysis. Carnegie Mellon University and International Institute of Information Technology Hyderabad. (September 5 2008) http://www.speech.cs.cmu.edu/15-492/slides/03_mfcc.pdf. [10](#)
- [19] Hui, J.: Speech recognition HMM and GMM. Medium Towards Data Science. (September 2019) https://medium.com/@jonathan_hui/speech-recognition-gmm-hmm-8bb5eff8b196. [14](#)

-
- [20] Shimodaira, H., Renal, S.: HMM and GMM. The University of Edinburgh. (January 2017) <https://www.inf.ed.ac.uk/teaching/courses/asr/2016-17/asr03-hmmgmm-handout.pdf>. 14
- [21] Bishop, C.M.: Pattern Recognition and Machine Learning (Information Science and Statistics). Springer-Verlag, Berlin, Heidelberg (2006) 17, 20
- [22] Deep Learning: MultiLevel Perceptron. <http://deeplearning.net/tutorial/mlp.html>. 20
- [23] McLaren, M., Yun, L., Scheffer, N., Ferrer, L.: Application of convolutional neural networks to speaker recognition in noisy conditions. INTERSPEECH-2014 (2014) 686–690 20
- [24] Chung, J.S., Nagrani, A., Zisserman, A.: Voxceleb2: Deep speaker recognition. CoRR **abs/1806.05622** (2018) 20
- [25] Jantzen, J.: Introduction to Perceptron Networks. Technical University of Denmark (1998) 21
- [26] Minsky, M., Papert, S.: An introduction to computational geometry. Cambridge tiass., HIT (1969) 21
- [27] Werbos, P.: Beyond regression:” new tools for prediction and analysis in the behavioral sciences. Ph. D. dissertation, Harvard University (1974) 21
- [28] Foulds, L.R., Haugland, D., Jørnsten, K.: A bilinear approach to the pooling problem. Optimization **24**(1-2) (1992) 165–180 21, 30
- [29] Sherstinsky, A.: Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network (2018) cite arxiv:1808.03314Comment: 39 pages, 10 figures, 66 references. 21
- [30] O’Shea, K., Nash, R.: An introduction to convolutional neural networks. CoRR **abs/1511.08458** (2015) 21, 26, 28, 30, 34
- [31] He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. CoRR **abs/1512.03385** (2015) 21, 26
- [32] Hacker Earth: Introduction in tranfer learning. <https://www.hackerearth.com/practice/machine-learning/transfer-learning/transfer-learning-intro/tutorial/>. 24
- [33] Brownlee, J.: A Gentle Introduction to Pooling Layers for Convolutional Neural Network. Machine Learning Mastery. (April 2019) <https://machinelearningmastery.com/pooling-layers-for-convolutional-neural-networks/>. 30

REFERENCES

- [34] D, F.: Batch Normalization in Neural Networks. Medium Towards Data Science. (October 2017) <https://towardsdatascience.com/batch-normalization-in-neural-networks-1ac91516821c>. 32
- [35] MissingLink.AI: 7 Types of Neural Network Activation Functions: How to Choose? <https://missinglink.ai/guides/neural-network-concepts/7-types-neural-network-activation-functions-right/>. 34
- [36] Karlik, B., Olgac, A.V.: Performance analysis of various activation functions in generalized mlp architectures of neural networks. *International Journal of Artificial Intelligence and Expert Systems* **1**(4) (2011) 111–122 34
- [37] Zhang, Z., Sabuncu, M.: Generalized cross entropy loss for training deep neural networks with noisy labels. In: *Advances in neural information processing systems*. (2018) 8778–8788 38
- [38] Zhao, H., Gallo, O., Frosio, I., Kautz, J.: Loss functions for neural networks for image processing. *arXiv preprint arXiv:1511.08861* (2015) 38
- [39] Kingma, D., Ba, J.: Adam: A method for stochastic optimization. *International Conference on Learning Representations* (12 2014) 40
- [40] Gandhi, R.: A Look at Gradient Descent and RMSprop Optimizers. Medium Towards Data Science. (June 2018) <https://towardsdatascience.com/a-look-at-gradient-descent-and-rmsprop-optimizers-f77d483ef08b>. 40
- [41] Albiol, A., Torres, L., Delp, E.J.: Fully automatic face recognition system using a combined audio-visual approach. *IEE Proceedings - Vision, Image and Signal Processing* **152**(3) (June 2005) 318–326 44
- [42] Nagrani, A., Chung, J.S., Zisserman, A.: Voxceleb: a large-scale speaker identification dataset. *CoRR* **abs/1706.08612** (2017) 48, 60
- [43] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X.: TensorFlow: Large-scale machine learning on heterogeneous systems (2015) Software available from tensorflow.org. 48

- [44] Chollet, F., et al.: Keras. <https://keras.io> (2015) 48
- [45] Dlib: DLib Library. <http://dlib.net>. 49
- [46] Gandhi, R.: Support Vector Machine-Introduction to Machine Learning Algorithms. Medium Towards Data Science. (June 2018) <https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47>. 49
- [47] Qian, Y., Bi, M., Tan, T., Yu, K.: Very deep convolutional neural networks for noise robust speech recognition. IEEE/ACM Transactions on Audio, Speech, and Language Processing **24**(12) (Dec 2016) 2263–2276 61