

University POLITEHNICA of Bucharest
FACULTY OF ELECTRONICS, TELECOMMUNICATIONS AND
INFORMATION TECHNOLOGY

Multimodal Person Identification using Deep Neural Networks

Diploma Thesis

submitted in partial fulfillment of the requirements for the
degree of *Engineer*
in the domain *FACULTY OF ELECTRONICS,*
TELECOMMUNICATIONS AND INFORMATION TECHNOLOGY
study program *Applied Electronics*

Thesis advisor(s)
Conf. Dr. Ing. Horia Cucu

Student
Sandu Marian-Gabriel

Year 2020

University "Politehnica" of Bucharest
Faculty of Electronics, Telecommunications and Information Technology
Department **EAI**

Anexa 1

DIPLOMA THESIS
of student **SANDU Ș.M. Marian-Gabriel , 441F-ELA.**

1. Thesis title: Multimodal Audio-Video Person Recognition using Deep Neural Networks

2. The student's original contribution will consist of (not including the documentation part) and design specifications:

At first, I have a database with over 5000 videos and transcripts containing politicians talking in The Chamber of Deputies. This project aims to use this database in order to apply person detection not using only image or audio recognition, but both. This database needs to be processed properly in order to get relevant frontal images with all the speakers in order to be fed to the neural networks. To do this, I first need to process the html files found on the CDEP website, where the videos are found, in order to find the timestamps for every video and every speaker and to find the parts of the videos where every speaker is.

Secondly, after the preprocessing is done and the database with images exists and is quite clean, I will have to come up with an architecture for only the image part of the learning, in order to identify the speakers by their frontal face. For this, it is required to build the neural network, meaning to choose the different layers in order to achieve the goal of face recognition. After the neural network is designed, I will train the neural network with the images I have collected from the CDEP database and then it will be seen whether the network did well, or the hyperparameters of it need to be tuned in order to have a better accuracy. Then, having the weights calculated by the previous network, we will try other architectures in order to try to increase the accuracy of the face recognition algorithm.

Finally, the multimodal learning will consist of trying to implement both audio and video recognition using a complex neural network containing many hidden layers in order to calculate complex parameters. This has the purpose of trying to increase the accuracy of the previous neural networks using only image recognition.

3. Pre-existent materials and resources used for the project's development:

Python, Java, Bash Scripting, Keras, Tensorflow, OpenCv, Ffmpeg, Haar-Cascade Classifiers, Anaconda virtual environment, Visual Studio Code, CDEP database.

4. The project is based on knowledge mainly from the following 3-4 courses:


Computer Programming, Data Structures and Algorithms, Object-Oriented Programming, Neural Networks

5. The Intellectual Property upon the project belongs to: U.P.B.

6. Thesis registration date: 2019-11-29 13:42:19

Thesis advisor(s),
Conf. dr. ing. Horia CUCU

Department director,
Prof. dr. ing Sever PAȘCA

Student,

Dean,

Prof. dr. ing. Mihnea UDREA

Validation code: **9721cd7f1f**

Statement of Academic Honesty

I hereby declare that the thesis *Multimodal Person Identification using Deep Neural Networks*, submitted to the Faculty of Electronics, Telecommunications and Information Technology in partial fulfillment of the requirements for the degree of *Engineer* in the domain Electronic Engineering, study program *Applied Electronics* is written by myself and was never submitted to any other faculty or higher learning institution in Romania or any other country.

I declare that all information sources I used, including the ones I found on the Internet, are properly cited in the thesis as bibliographical references. Text fragments cited "as is" or translated from other languages are written between quotes and are referenced to the source. Reformulation using different words of a certain text is also properly referenced. I understand plagiarism constitutes an offence punishable by law.

I declare that all the results I present as coming from simulations and measurements I performed, together with the procedures used to obtain them, are real and indeed come from the respective simulations and measurements. I understand that data faking is an offence punishable according to the University regulations.

Bucharest, June 2020.

Student: Sandu Marian-Gabriel



.....

Table of Contents

List of figures	iv
List of tables	vi
List of abbreviations	viii
1. Introduction	1
1.1. Motivation and objectives	1
1.2. Paper structure	2
2. Theoretic insights	3
2.1. Spectrogram	3
2.2. Voice Activity Detection	3
2.3. Agglomerative Hierarchical Clustering	4
2.4. Jupyter Notebooks	5
2.5. Facial Detection	5
2.6. Deep Learning	5
2.6.1. Deep Neural Networks	6
2.6.2. Convolutional Neural Networks	7
2.6.2.1. Convolution layer	7
2.6.2.2. Max-pooling layer	8
2.6.2.3. Fully-connected layer	9
2.6.3. Residual Neural Networks	9
2.6.4. Cost functions	10
2.6.5. Optimization algorithms	10
2.6.6. Regularization methods	11
2.6.7. Transfer Learning	12
2.6.8. Performance metrics	12
3. State of the art on face recognition	13
3.1. Face recognition datasets	13
3.1.1. ImageNet Dataset	13
3.1.2. VGGFace2 Dataset	13
3.1.3. MS-Celeb-1M Dataset	14
3.2. Previous work	14
3.3. Network architectures	15
3.3.1. VGG16 architecture	15
3.3.2. FaceNet architecture	16
3.3.3. GoogLeNet architecture	17

4. State of the art on speaker recognition	19
4.1. Speaker recognition datasets	19
4.1.1. VoxCeleb2 Dataset	19
4.2. Previous work	19
4.3. Network architectures	20
4.3.1. VGGVOX architecture	20
5. Technologies	23
5.1. Programming languages	23
5.1.1. Python	23
5.1.2. Java	23
5.2. BeautifulSoup (JSoup)	23
5.3. Numpy	24
5.4. Pandas	24
5.5. Deep learning frameworks	24
5.5.1. Keras and Tensorflow	24
5.5.2. PyTorch	25
5.5.3. SKlearn	25
5.6. Ffmpeg	26
5.7. OpenCV	26
5.8. LibROSA	26
5.9. CUDA	27
6. CDEP Dataset development	29
6.1. Previous work	29
6.2. Overview of data processing tasks	29
6.3. Image dataset development	30
6.3.1. Image extraction	30
6.3.2. Obtaining the similarity matrix	31
6.3.3. Algorithm development	33
6.4. Audio dataset development	40
6.5. Multimodal dataset development	41
7. Face recognition	43
7.1. Experimental setup	43
7.2. FaceNet architecture	43
7.3. GoogLeNet architecture	46
7.4. VGG16 architecture	47
7.5. Results and conclusions	49
8. Speaker recognition	51
8.1. Experimental setup	51
8.2. VGGVOX	51
8.3. Results and conclusions	53

9. Multimodal person identification	55
9.1. Previous work	55
9.2. Experimental setup	55
9.3. Multimodal architecture	55
9.4. Experiments	56
9.5. Conclusions	58
9.6. Multimodal network validation	59
10. Conclusions and further development	61
10.1. General conclusions	61
10.2. Personal contributions	61
10.3. Further development	62
Bibliography	63
Annex A. Similarity computation	67
Annex B. Image dataset validation algorithm	68
Annex C. Multimodal data generator	70

List of figures

1.1. Implementation steps	1
2.1. Spectrogram example [1]	3
2.2. Voice Activity Detection overview	3
2.3. Example of a Dendrogram [2]	4
2.4. Example of haar features [3]	5
2.5. Artificial Intelligence, Machine Learning, and Deep Learning [4]	6
2.6. ReLU activation function	6
2.7. Convolutional Neural Network [5]	7
2.8. Example of a filter's output as a function of the number of neurons [6]	8
2.9. Example of max-pooling with the extent and stride 2	8
2.10. Example of a residual block	9
2.11. Example of a residual block with an operation on the shortcut	10
2.12. Example of the Dropout algorithm [7]	12
3.1. ImageNet example images	13
3.2. Example of images from the VGGFace2 dataset	13
3.3. Example of images from the MS-Celeb-1M dataset	14
3.4. State-of-the-art methods of face recognition [8]	15
3.5. VGG16 layer architecture [9]	15
3.6. FaceNet detailed architecture [10]	16
3.7. Triplet Loss concept training example [11]	17
3.8. GoogLeNet detailed architecture [12]	17
4.1. (left) Distribution of utterance lengths; (middle) Gender distribution; (right Nationality distribution of speakers) [13]	19
4.2. Examples of x-vector architectures [14]	20
4.3. VggVox architecture	21
6.1. Flowchart of the data processing task	30
6.2. Flowchart of the extraction of images algorithm	31
6.3. Flowchart of the creation of the similarity matrix	32
6.4. Example images from VGGFace dataset	33
6.5. Flowchart for the image dataset validation algorithm	34
6.6. Example of bad-labeled images	35
6.7. List of classes, their average similarity and observations	36
6.8. Example of a similarity matrix for a class	36
6.9. Clustering example of a folder with a dominant person	37
6.10. Clustering example of a folder with multiple non-dominant persons	37
6.11. Manual verification results example	38
6.12. Examples of data from the validated face dataset	39
6.13. Person histogram	39
6.14. Flowchart explaining the audio database creation algorithm	40
7.1. Experimental setup for face recognition	43

7.2.	FaceNet fine-tuning	44
7.3.	FaceNet experiment for 10 images/class dataset	44
7.4.	FaceNet experiment for 50 images/class dataset	45
7.5.	FaceNet experiment for 100 images/class dataset	45
7.6.	GoogLeNet detailed architecture with emphasis on fine-tuning	46
7.7.	GoogLeNet experiment for 10 images/class dataset	46
7.8.	GoogLeNet experiment for 50 images/class dataset	47
7.9.	GoogLeNet experiment for 100 images/class dataset	47
7.10.	VGG16 detailed architecture with emphasis on fine-tuning	48
7.11.	VGG16 experiment for 10 images/class dataset	48
7.12.	VGG16 experiment for 50 images/class dataset	49
7.13.	VGG16 experiment for 100 images/class dataset	49
7.14.	Results for the face recognition task	50
8.1.	Experimental setup for the speaker recognition tasks	51
8.2.	VGGVOX architecture with an emphasis on fine-tuning	52
8.3.	VGGVOX experiment for 10 images/class dataset	53
8.4.	VGGVOX experiment for 50 images/class dataset	53
8.5.	Results of the speaker recognition experiments	53
9.1.	Fusion process	56
9.2.	multimodal neural network architecture	56
9.3.	multimodal experiment for 10 images/class dataset	57
9.4.	Multimodal experiment for 50 images/class dataset	57
9.5.	multimodal experimental results	58
9.6.	Test waveforms of the "01724" class	59
9.7.	Test images of the "01724" class	59
9.8.	Examples of images from the VidTIMIT database	60
9.9.	VidTIMIT experiment accuracy and loss curves	60

List of tables

6.1.	Image dataset configurations	39
6.2.	Audio database configuration	40
6.3.	Derived audio dataset configurations	41
6.4.	Multimodal dataset configurations	41
7.1.	Derived image datasets split numbers of examples	43
7.2.	Mislabeled classes from the image experiments	50
8.1.	Audio derived datasets split number of examples	51
8.2.	Mislabeled classes for the audio experiments	54
9.1.	Mislabeled classes from the multimodal experiments	58

List of abbreviations

ML = Machine Learning
PCA = Principal Component Analysis
DNN = Deep Neural Networks
VAD = Voice Activity Detector
CNN = Convolutional Neural Network
RNN = Residual Neural Network
SGD = Stochastic Gradient Descent
BGD = Batch Gradient Descent
TP = True Positives
TN = True Negatives
FP = False Positives
FN = False Negatives
MSE = Mean Squared Error
FFT = Fast Fourier Transform
GCP = Google Cloud Compute
ReLU = Rectified Linear Unit
MFCC = Mel Frequency Cepstrum Coefficients
LPC = Linear Prediction Coefficients
SNR = Signal to Noise Ratio
CDEP = Camera Deputatilor
DOM = Document Object Model
CSS = Cascade Style Sheet
URL = Uniform Resource Locator
HTML = HyperText Markup Language

Chapter 1

Introduction

1.1 Motivation and objectives

The technological advance in the recent years had a major impact on almost all domains. Until recent years, Artificial Intelligence was not known to the large population. Nearly every computer program that we interacted with before was coded by software developers from scratch. By the increase of ML engineers in the next years, many tasks that were not be so easily developed by humans started to be created by using machine learning algorithms and methods.

Automatic identification of people has many applications in different areas, such as securing access to sites, automatic banking, password-free computer login and also analysis of human behaviour. As a motivation for building this automated multimodal person recognition system, we can take into consideration the fact that Deep Neural Networks were used recently in hacking. This networks would be learnt to break security barriers such as face recognition systems, fingerprint scanners or audio recognition systems. By combining two of these methods, the metadata of the neural network would be a lot more complex and in order to train another network to hack into the system would be much harder.

This project implies a system that automatically identify people in a video sequence using deep learning algorithms and network architectures. The steps required to complete these objective are shown below:

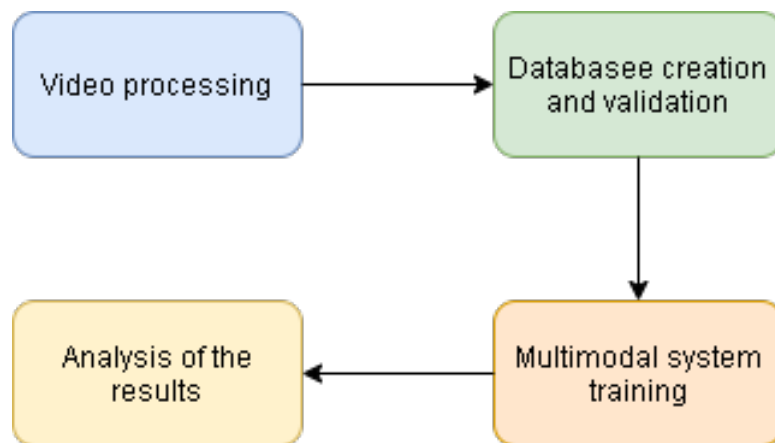


Figure 1.1: Implementation steps

By looking at Figure 1.1, we can next list the specific objectives for creating the multimodal recognition system:

- Developing an algorithm that automatically perform crawling on the website "www.cdep.ro" in order to obtain the videos and the metadata associated with it.
- Developing algorithms to create databases with faces and audio corresponding to the images.

- Validating the databases with originally developed algorithms specifically built for our task.
- Finding a couple of state-of-the-art pre-trained neural networks for the task of face recognition which would fit our problem well - After this, retrain them and find out which one has the best results.
- Finding a state-of-the-art pre-trained neural network for the task of speaker recognition which would fit our problem well, and evaluate it.
- Developing a multimodal system based on the previous neural network and analyse the results.

1.2 Paper structure

The thesis is organized in nine chapters, as follows: Chapter 1 presents the motivation, the objectives and the summary of the thesis. In Chapter 2, the theoretical aspects of the thesis are outlined. Chapter 3 and 4 presents the state-of-the-art technologies regarding the face recognition and the audio recognition. In Chapter 5, the technologies used in developing this thesis are detailed and explained, with an emphasis to where these technologies were used. Chapter 6 is the first chapter that illustrates contributions of the author of the thesis and deals with the development of the face and audio dataset algorithm and its validation. In Chapters 7,8 and 9 the deep learning experiments are explained and analysed. Finally, Chapter 10 summarizes the main conclusions of the thesis and underlines the author's contribution.

Chapter 2

Theoretic insights

2.1 Spectrogram

A spectrogram is a visual representation of a signal's frequency spectrum over time, as seen in Figure 2.1. This notion is used especially in the musical domain, radars and sound processing. It is represented under the form of an image with the X-axis representing time and the frequency on the Y-axis. Also, the amplitude of the signal is represented by the color of the pixels of the graph. [15]

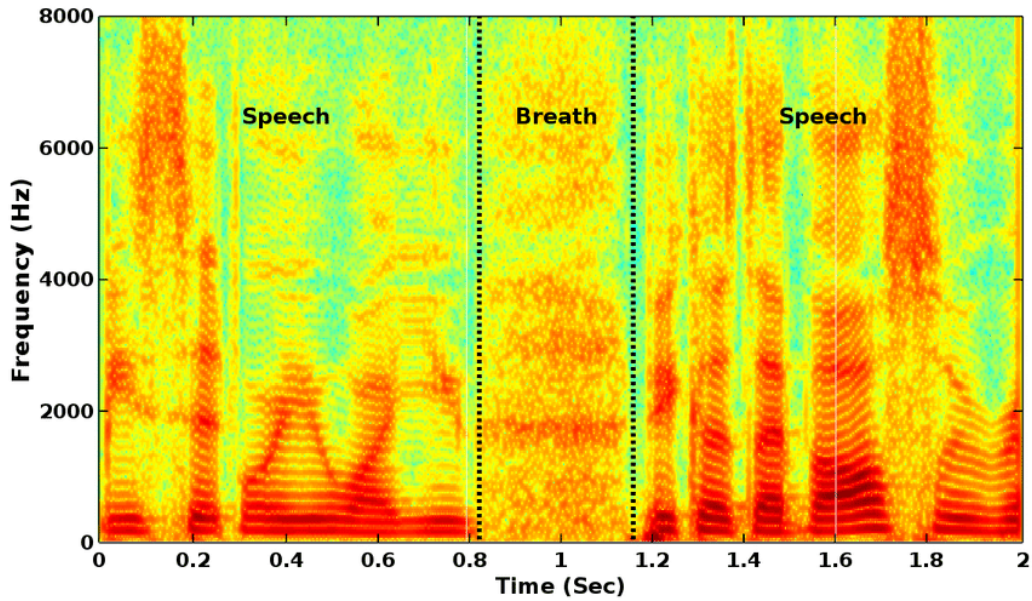


Figure 2.1: Spectrogram example [1]

2.2 Voice Activity Detection

Voice Activity Detection (VAD) is a signal processing algorithm that is used in order to distinguish speech segments from background noise in an audio stream.

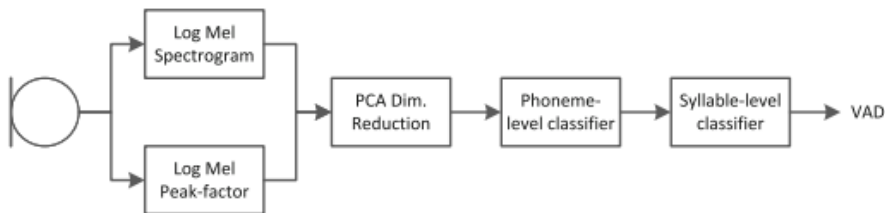


Figure 2.2: Voice Activity Detection overview

The algorithm in Figure 2.2 consists of feature extraction, feature space dimensionality

reduction and two level classifier. This algorithm uses the Mel band spectral envelope and Mel band peak factor as features.

The spectral envelope is a feature manifesting as Mel Frequency Cepstrum Coefficients (MFCC) or Linear Prediction Coefficients (LPC). According to Gunnar Fant's study of the acoustic theory of speech production [16], the harmonics of the fundamental frequency contain most speech energy, which is distinguished from noise due to high SNR (Signal to Noise Ratio). The spectral envelope does not capture the pitch, and because of this reason band peak factors are added. The next step involves the improvement of noise robustness. This step is described as the addition of tonal and temporal auditory masks processing to the spectral envelope.

Extracted features are represented by a column per each frame. These columns are aggregated by sliding a frame window which form a feature space of the extracted features. The final scores are then calculated by the help of the Principal Component Analysis (PCA), which takes into consideration only the most important components.

Finally, features are classified using a Softmax classifier using a modified version of deep decision trees [17].

$$H_0(f) = \sum_{i=1}^N \omega_i h_i(f)$$

2.3 Agglomerative Hierarchical Clustering

The Agglomerative Clustering is the most common hierarchical clustering method used to group objects based on the similarity between each other. This algorithm starts by taking into consideration each object as a cluster containing a single object. Then, sequentially, pairs of clusters are successively merged together until all clusters have been merged into a fixed number of clusters. The result is called a dendrogram (Figure 2.3), and is based on a tree representation of the objects. [18]

The Agglomerative Clustering algorithm works "bottom-up". Initially, each object is considered as a singleton cluster. At each step of the algorithm, the two clusters that are the most similar are merged into a new bigger cluster.

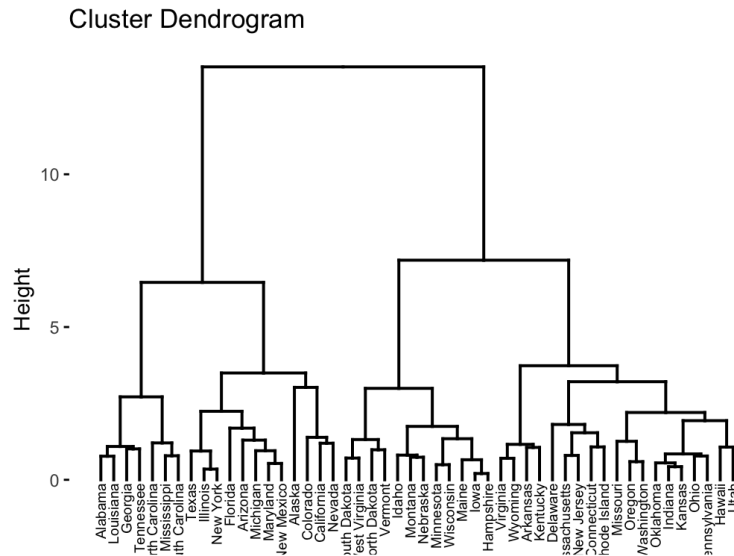


Figure 2.3: Example of a Dendrogram [2]

2.4 Jupyter Notebooks

Jupyter Notebook is an open-source web application that allows the user to create and share documents that contain live code, equations, visualisations and narrative text. Also, by using this web application, we were able to debug the training and validation of the networks much easily, since the code can be split in cells and each cell can be run separately. Also, these notebooks were easily shareable between different machines, such as my own computer and Google Colab. [19]

2.5 Facial Detection

The first step in face recognition using CNNs is to extract the face of the subject from the rest of the image, so the neural network does not learn useless information. There are two big categories of facial detection: static and real-time. The approach we are using in this project is static.

In order to explain the facial detection in the below chapters, we have to describe the Haar-Cascade classifier which represents a machine learning algorithm for detection of different objects. This algorithm can be easily used in order to detect faces if it is trained on the proper data.

After the training of the classifier, the features are extracted from the data. For this, the Haar features are extracted, which can be seen in Figure 2.4. They are just like a convolutional kernel, meaning each feature is a single value obtained by subtracting a sum of pixels under a white rectangle from sum of pixels under a black rectangle. [20]

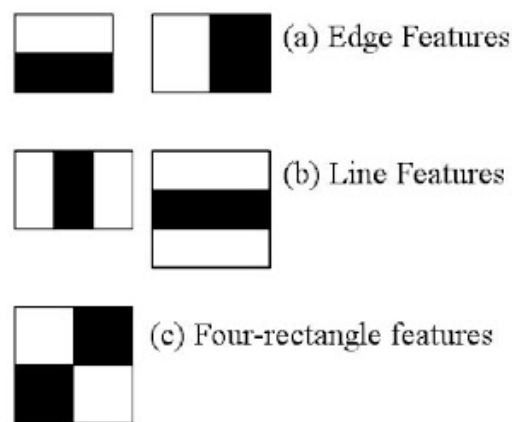


Figure 2.4: Example of haar features [3]

2.6 Deep Learning

As an introduction, artificial intelligence is the intelligence that is simulated with a computer and programmed so it will think like a human brain and mimic its actions.

Deep Learning is basically a sub-category of Machine Learning, which in fact is a subdomain of Artificial Intelligence. Figure 2.5 shows the connections between Artificial Intelligence, Machine Learning and Deep Learning fields of work.

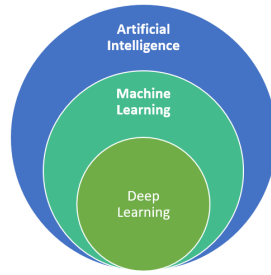


Figure 1: artificial intelligence, machine learning and deep learning Source: Nadia BERCHANE (M2 IESCI, 2018)

Figure 2.5: Artificial Intelligence, Machine Learning, and Deep Learning [4]

As a definition, Deep Learning is an artificial intelligence function that imitates the human brain in processing data and creating patterns for use in decision making. The learning process can be of two types: supervised (when the data fed to the network is composed of input data and also the ideal output for each example) and unsupervised (only input data is given, without labels). [21]

Deep Learning has the following features:

- a sequence of layers containing neurons or units which process information non-linearly in order to extract certain characteristics and transform data; to each layer's input, the output data of the previous layer is given.
- a network learns multiple representations of the data, each of them corresponding to a certain layer of abstracting (hierarchical mode).

2.6.1 Deep Neural Networks

A neural network can be described by a graph in which each node corresponds to a neuron and every connection to a weight. Each neuron calculates the pondered sum of the previous output neurons connected to it. The result is then fed into an activation function and an output of the neuron is obtained. Statistically, the neuron, which is the building block of a neural network, models a linear regression on which an activation is added (non-linear part). [21]

As an example of an activation function, in this project ReLU (Rectified Linear Unit) was used. The equation (3,3) presents the mathematical formula for calculating the output of a neuron, for which the input is a vector of real numbers X , and Figure 2.6 shows how the graph of the ReLU function.

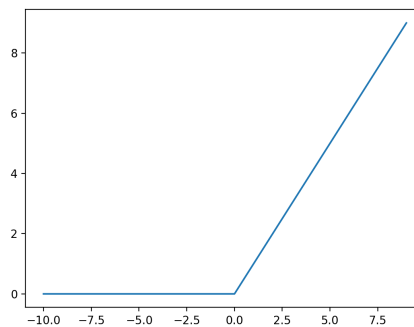


Figure 2.6: ReLU activation function

The input data fed into a neural network is typically represented by a set of vectors of real numbers. In the context of supervised learning, the optimal output results are given with the input.

The passing of an example from the input data and obtaining an output is called forward propagation. Because of the fact we want to minimize the error, after the forward propagation is completed, a backward propagation is necessary in order to update the weights of the neural network. By using the value of the error, the network will start the process of backward propagation. This process consists of using an optimization algorithm of the cost function and the state variables are the weights.

A forward-backward propagation cycle of an example from the input data is called an iteration. The passing of all the training example is called an epoch. Furthermore, the training of a neural network is done in many epoch, such that the error can be decreased. A drawback of a large number of training epochs is the overfitting on the training data. This phenomenon can be described as the incapability of a neural network of generalization. Due to overfitting, the network will have great results on the training examples, and not on new ones. [21]

2.6.2 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are multi-layer neural networks that are specialized in pattern recognition tasks, variants of the Multi-Layer Perceptrons (MLPs) inspired from biology.

CNNs are hierarchical neural networks based on convolution combined with pooling layers, for automated feature extraction and a series of fully-connected layers that will perform the final classification of the data. [22] CNNs are not like a regular neural network. They take as advantage the fact that the input consists of images. As a result, the neurons in a CNN are arranged in 3 dimensions: width, height and depth. Figure 2.7 depicts the general architecture of a CNN in a form that is understandable to the human eye.

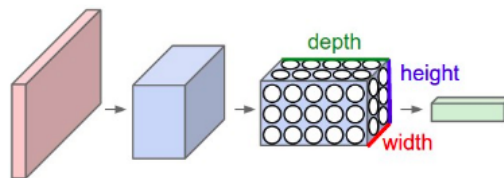


Figure 2.7: Convolutional Neural Network [5]

2.6.2.1 Convolution layer

The Convolution layer is the core building block of a CNN, because of the fact that it does the most computationally heavy work.

The convolution layer consists of a set of learnable filters. Each filter has a small dimension, but during the forward pass through the network it slides on the input at any position, creating the so-called extension through the full depth of the input volume.

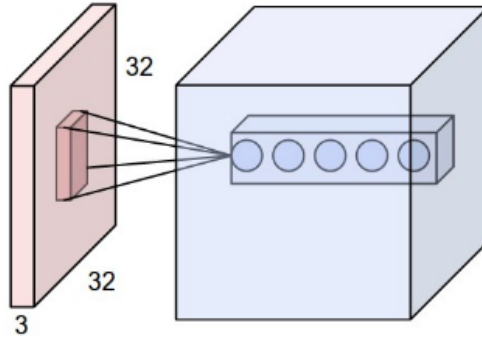


Figure 2.8: Example of a filter's output as a function of the number of neurons [6]

Figure 2.8 shows a single step of the extension described in this paragraph. [22]

The process of obtaining the output volume is fairly straightforward: we center the convolution filters on a pixel from the input image, calculate the scalar product between the filter's parameters and the pixels' values onto which the filter is, and obtaining the corresponding result. This process is sequentially repeated until the image is finished. This type of layer has three parameters to take into consideration:

- depth of the output volume, which corresponds to the number of filters we would like to use
- stride of the filters, which can be explained as the distance we want to move the filter at a time
- zero-padding, which is a convenient way to control the spatial size of the output volumes, by padding the input data with zeros [22]

2.6.2.2 Max-pooling layer

As an intuition, because of the fact that the size of the a CNN is large regarding the learnable parameters. After each convolution layer, we need a layer that reduces the dimensionality of the parameters. Furthermore, it also extracts and focuses onto the most important features. This layer is called the Max-pooling layer and its function is shown in Figure 2.9. [22]

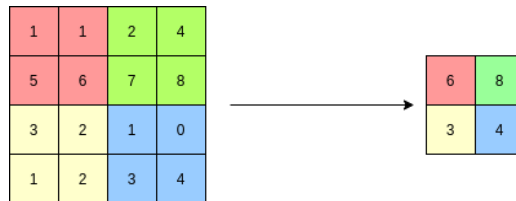


Figure 2.9: Example of max-pooling with the extent and stride 2

This type of layer accepts a volume of size $W1 \times H1 \times D1$, and requires two hyperparameters: their spatial extent F , and the stride S . After the hyperparameters are given, the pooling layer produces a volume of $W2 \times H2 \times D2$, where:

- $W2 = (W1 - F)/S + 1$
- $H2 = (H1 - F)/S + 1$
- $D2 = D1$

2.6.2.3 Fully-connected layer

The fully connected layers are placed in the final part of the CNNs, because they have a large number of learnable parameters. Their main feature is that they have connections to all the neurons from the previous layer. These connections can be seen as a multiplication of matrices and then an addition of a mean. [22]

2.6.3 Residual Neural Networks

Residual Neural Networks (RNNs) have been developed due to a certain overwhelming problem in image recognition tasks named the vanishing gradient problem. This issue occurs while training artificial neural networks that involved gradient-based learning and propagation. In the step of backward propagation, gradients are used to update the weights in our network, but sometimes it happens that the gradient becomes very small (vanishing), and prevents the weights from being updated. To solve this problem, Residual Neural Networks were introduced. [23]

RNNs or commonly known as ResNets are a type of neural network that is derived from the CNNs, but also applies identity mapping.

In this circumstance, identity mapping means that the input to some layer is passed directly to some other layer with the help of a skip connection. Figure 2.10 shows the basic residual block's structure:

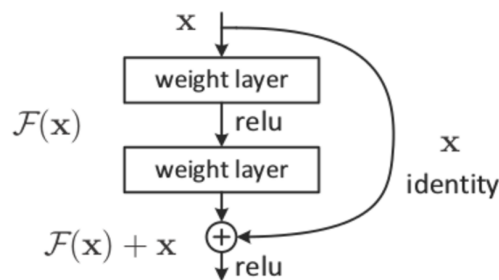


Figure 2.10: Example of a residual block

Because of the fact that the identity input and the output of one layer may have different dimensions, an operation or function can be added to the skip connection in order for the both entities to have the same dimensions. Figure 2.11 shows this concept.

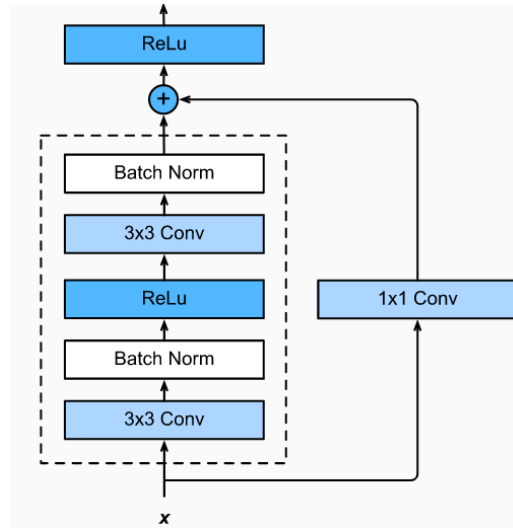


Figure 2.11: Example of a residual block with an operation on the shortcut

2.6.4 Cost functions

In the decision, estimation and probability field, the cost function makes a connection between an input even and a real number which represents an associated cost to that event. In the training event of a neural network, the optimization problem consists of minimizing a chosen cost function.

As a summary, the cost function reduces a complex system to a single scalar value such that it can be compared and ordered. Based on the used architecture of neural network and the type of the output, there can be defined a few well-known cost functions. [24]

- If the network performs regression, the mean squared error can be used as a cost functions.

$$MSE = \frac{1}{n} \sum_{i=1}^n (Y_i - \hat{Y}_i)^2$$

- For a classification problem, the most used cost function is the cross-entropy based cost function.

$$J(w) = -\frac{1}{N} \sum_{n=1}^N [y_n \log \hat{y}_n + (1 - y_n) \log 1 - \hat{y}_n]$$

2.6.5 Optimization algorithms

Gradient descent is the most popular algorithm to perform optimization and also the most common way to optimize a neural network. This algorithm is a way to minimize an objective function $J(\theta)$ parameterized by a model's parameters by updating the parameters in the opposite direction of the gradient of the objective function with respect to the parameters. [25]

An important hyperparameter of a neural network, the learning rate, determines the size of the steps the algorithm takes to reach a local minimum of the cost function.

At the time being, there are three variants of gradient descent: batch gradient descent, stochastic gradient descent and mini-batch gradient descent. These three optimization algorithms differ from each other by the amount of data taken into consideration when updating the parameters.

In our case, the optimization function used is the Stochastic Gradient Descent. SGD performs a parameter update for each training example and its label. This fact explains that this algorithm is much faster but it also can make the objective function fluctuate heavily.

Stochastic Gradient Descent performs a parameter update for each training example x and label y by the formula:

$$\theta = \theta - \eta \cdot \nabla_{\theta} J(\theta; x^{(i)}; y^{(i)})$$

Comparing SGD to the Batch Gradient Descent, BGD performs redundant computations for large datasets, because it recomputes gradients for similar examples before each parameters update. While BGD converges to the minimum the parameters are in, SGD fluctuates so much, allowing it to jump to a new and potentially better global minimum. A problem of this would be the problem of overshooting, which can be solved by lowering and adjusting the learning rate. [25]

2.6.6 Regularization methods

The learning algorithms based on neural networks are managing to learn very high level functions with a very high grade of non-linearity. Still, this thing can lead to the situation in which the learned function fits the training data too well. This is the reason why the learned function won't behave well on the test data or another data set. This phenomenon is called overfitting and represents an active problem in the Deep Learning domain. [26]

There can be two methods of regularization:

- Cost function regularization
- Neuron regularization

The cost function regularization is the most common regularization method which consists in the cost function being modified by truncating the biggest values of the function. The modification consists of the introduction of a regularization term in different ways. There are two types of cost function regularization:

- L2 regularization

$$J_n(W) = J(W) + \lambda \sum_{i=1}^n w_i^2$$

- L1 regularization

$$J_n(W) = J(W) + \lambda \sum_{i=1}^n |w_i|$$

Also, another regularization method that is very popular in the Deep Learning domain is the neuron regularization method, or Dropout method.

Dropout is a method which helps avoid overfitting by making neurons be more adaptive and not depend on the others. It consists in cutting certain connections between neurons during training, chosen randomly. Every connection is cut with a certain probability, which is called the dropout rate. This rate is a hyperparameter for our network architectures and need to be chosen properly because it can affect negatively the performances. [27]

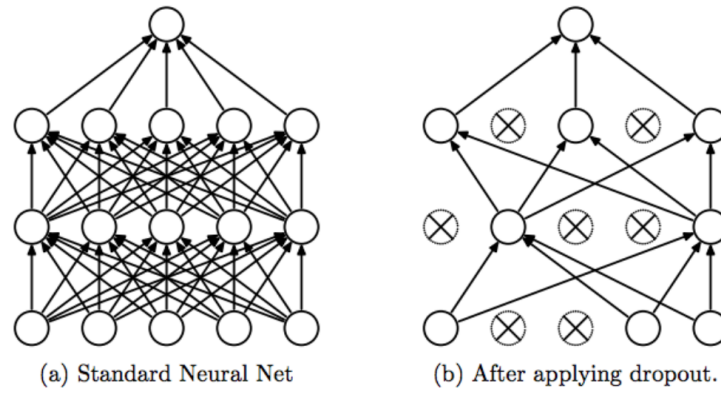


Figure 2.12: Example of the Dropout algorithm [7]

In Figure 2.12 the process of dropout is shown in two steps, therefore the two images. The left image shows the neurons in the layers interconnected through the weights. After applying dropout, we can see that some neurons have lost their connection, therefore trying to increase the ability of the network to generalize.

2.6.7 Transfer Learning

Transfer learning is a machine learning technique where a deep neural network model trained on one task is re-purposed on a similar second task by transferring the weights and the architecture of the model.

The form of transfer learning used in deep learning and in our problem is called an inductive transfer. This type of transfer can be explained as where the scope of possible models is narrowed in a good way by using a model fit on a different but related task. [28]

2.6.8 Performance metrics

The effectiveness of a neural network architecture is measured by using some performance metrics, such as confusion matrices, accuracy, precision, recall or sensitivity. Next, we will describe each one of them separately.

The confusion matrix is one of the most intuitive and easiest metrics used in order to find out which classes are not learned properly, though we do not use it because of the fact that we have a big number of classes and it is impractical.

The accuracy of a model is defined as the number of correct predictions made by the model out of the total number of predictions. Accuracy has a good usage in classification problems if the classes are balanced, so this metric has a great importance in our project.

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}$$

Precision is a metric that points out what percentage of positives were correctly classified.

$$Precision = \frac{TP}{TP+FP}$$

Recall is also a metric that tells us what fraction of the total amount of relevant data were actually retrieved.

$$Recall = \frac{TP}{TP+FN}$$

By using these metrics we can correctly evaluate whether if a model predicts correctly or not, and also on which classes and data the model underperforms.

Chapter 3

State of the art on face recognition

3.1 Face recognition datasets

3.1.1 ImageNet Dataset

ImageNet was designed especially because of the fact that in this era of Deep Learning, the size of a dataset can make the difference in the success of a neural network. [29]

In terms of organization, this dataset was organized with respect to the WordNet hierarchy. This WordNet is built out of synonym sets or synets. There are more than 100,000 synets in ImageNet.

Regarding the technical details, ImageNet consists of 14,197,122 images organized into 21,841 subcategories. These subcategories can be considered as a sub-tree of 27 high level categories.

On average, there are approximately 500 images per class. For example, the category "animal" is the most widely covered with 3822 classes and 2799K images.

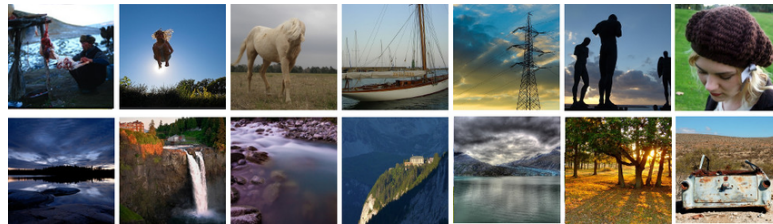


Figure 3.1: ImageNet example images

It can be seen from Figure 3.1 that this dataset is not built especially for face recognition, so a proper design of a neural network architecture needs to be created such that transfer learning can work.

3.1.2 VGGFace2 Dataset

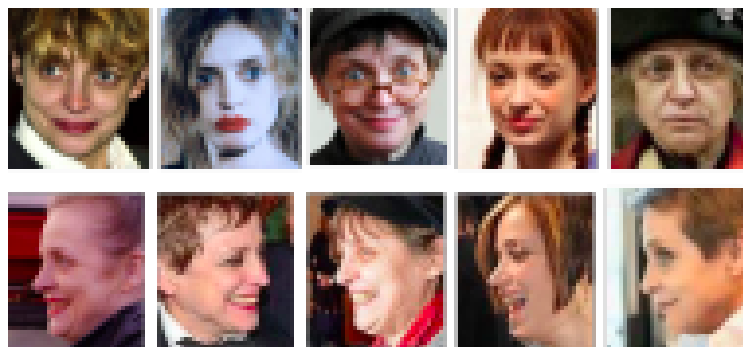


Figure 3.2: Example of images from the VGGFace2 dataset

The VGGFace2 Dataset was developed by the University of Oxford’s Department of Engineering Science and consists of 3.31 million images of 9131 subjects, with an average of 362.6 images per subject. The images are procured from Google Image Search. This dataset’s strong point is the diversity of the images, having high variations of age, light, pose and ethnicity. Examples from this dataset can be seen in Figure 3.2. [30]

3.1.3 MS-Celeb-1M Dataset

Microsoft Celeb (MS-Celeb-1M) is a dataset comprising 10 million images collected from the internet for the sole purpose of person recognition tasks. Microsoft Research created and published this dataset in 2016, and according to them is the largest publicly available face recognition dataset in the world, containing over 10 million images of nearly 100,000 individuals. Examples from this dataset can be seen in Figure 3.3. The majority of identities in the MS-Celeb-1M dataset are American and British actors. In contrast, many of the names in the MS Celeb dataset are normal people who must maintain an online presence due to their profession. After the MS Celeb dataset was first introduced in 2016, researchers from Microsoft Asia worked with researchers from China’s National University of Defense Technology (NUDT) and used this dataset for research purposes. [31]

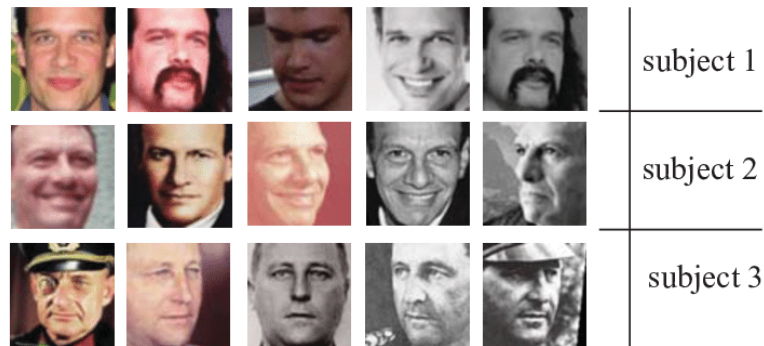


Figure 3.3: Example of images from the MS-Celeb-1M dataset

3.2 Previous work

The facial recognition as an application of deep and representation learning has been the important factor in the development of the state-of-the-art in deep learning. The most advanced and accurate facial recognition systems are built with the help of big datasets which are becoming bigger and CNNs.

As a background introduction in this subchapter, we can say that previous image recognition and facial recognition systems were built using hand-engineered features such as SIFT, LBP and Fisher vectors. By looking at the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2011, one can see the fact that the top of the chart is constituted by engineered features, and not hand-engineered features. [8]

It can be seen that the algorithm for facial recognition has drastically changed through years. There have been a lot of changes, from hand-engineered features to learned features, from face-specific alignment to rough alignment and centering, and a transition regarding the size of the datasets used to train such networks for face recognition.

Name	Method	Images (Millions)	Accuracy
Baidu ²³ (Announced)	CNN	-	0.9985 \pm -
Google FaceNet⁵	CNN	200.0	0.9963 \pm 0.0009
DeepID3 ²⁴	CNN	0.29	0.9953 \pm 0.0010
MFRS ²⁵	CNN	5.0	0.9950 \pm 0.0036
DeepID2+ ²⁶	CNN	0.29	0.9947 \pm 0.0012
DeepID2 ³	CNN	0.16	0.9915 \pm 0.0013
DeepID ²⁷	CNN	0.2	0.9745 \pm 0.0026
DeepFace ⁶	CNN	4.4	0.9735 \pm 0.0025
FR+FCN ²⁸	CNN	0.087	0.9645 \pm 0.0025
TL Joint Bayesian ²⁹	Joint Bayesian	0.099	0.9633 \pm 0.0108
High-dim LBP ³⁰	LBP	0.099	0.9517 \pm 0.0113

Figure 3.4: State-of-the-art methods of face recognition [8]

3.3 Network architectures

3.3.1 VGG16 architecture

VGG16 is a plane convolutional neural network model proposed by K. Simonyan and A. Zisserman from the University of Oxford in the paper “Very Deep Convolutional Networks for Large-Scale Image Recognition”. [32]

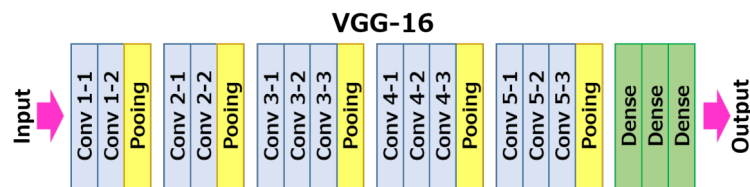


Figure 3.5: VGG16 layer architecture [9]

A Convolutional Neural Network can be explained as an algorithm that takes as an input an image, assign importance to different aspects in the image, called weights and biases. In Figure 3.5, it can be seen that there are 3 types of layers in this architecture: convolutional, pooling and dense layers.

The convolutional layer, also called the kernel, functions as a moving filter of a certain dimension. This “square” traverses the image and for each position it calculates a value based on the filter applied. The main objective of a convolutional layer is to minimise the dimensions of an image in order for it to be easier to process. Furthermore, it can be said that the objective of this layer is to extract features (low-level and high-level) depending on the depth in which the layer is found.

The pooling layer is responsible for reducing the spatial size of the convoluted features. Due to this, the computational power needed is reduced too. This type of layers acts as a kernel too, but instead of relying on the form of the filter, it makes either a maximum between the elements in the window, or the average between them.

The dense layer, also called a fully-connected layer, learns a non-linear combination of the high-level features which are the output of the convolutional blocks.

The multi-class classification step is done by adding to the top model a Softmax classifier. This name is inspired by the Softmax function, which takes a vector of scores and quantize them into a vector of values between 0 and 1. Therefore, these values represent a true probability distribution. The architecture described above can be seen as a visual representation in Figure 3.5. [32]

3.3.2 FaceNet architecture

The FaceNet model we used is an adaptation of Inception-Resnet-V1 architecture. This model is said to be the stronger modified version of GoogLeNet by combining the inception and residual principles for a network architecture [33]. A detailed version with every type of block is shown in Figure 3.6:

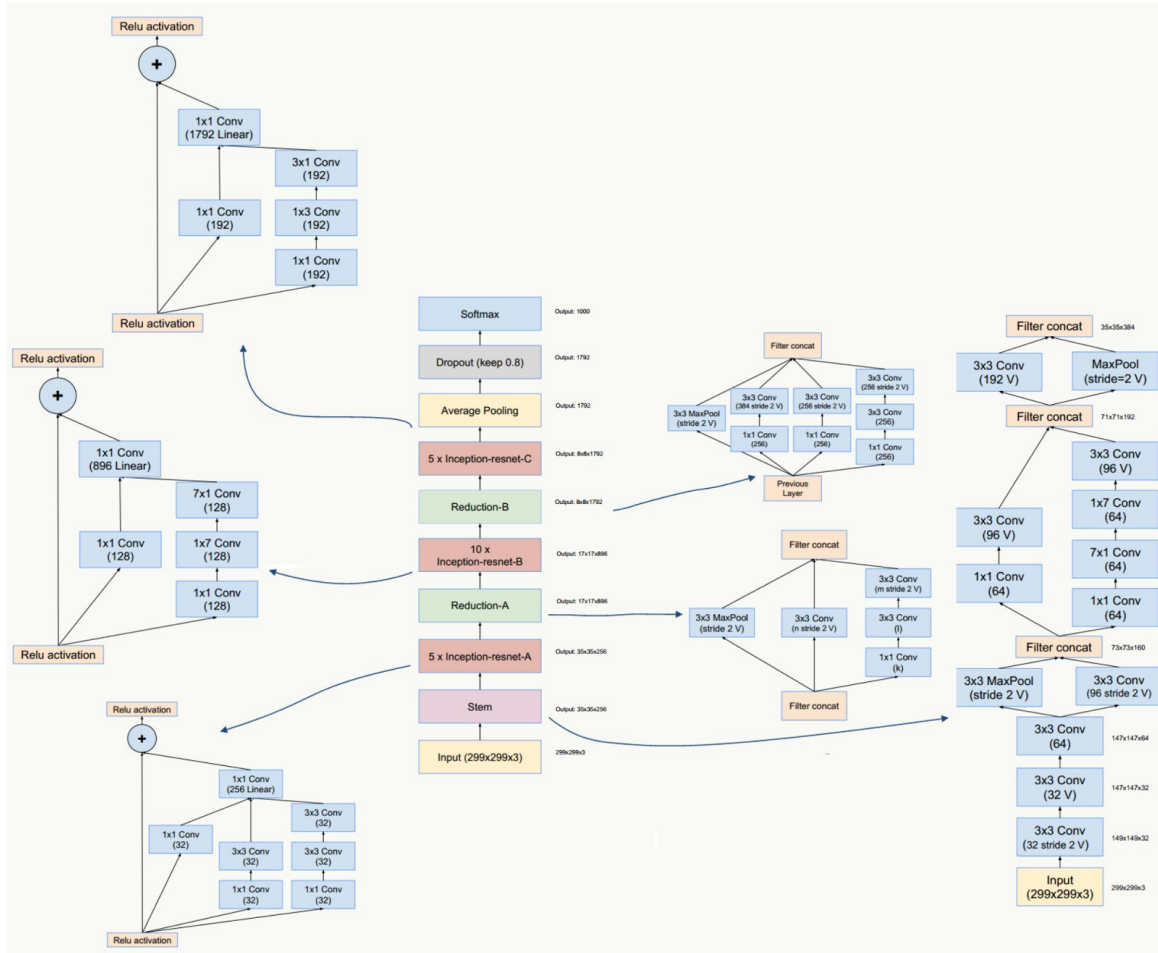


Figure 3.6: FaceNet detailed architecture [10]

The Inception blocks A,B and C are almost the same. What differs about them is the size of the convolutional kernels. On these blocks' inputs and outputs a ReLU (Rectified Linear Unit) activation function is applied, such that the negative weights can be discarded and also to help the backpropagation steps.

The Reduction blocks A,B and C are used in order to reduce the dimensionality of the feature vectors in order to select only the important features and also to reduce the overflow of information that needs to be processed by the computing machine. It's functionality is the same almost the same as of an inception block, the difference being the residual parameter passed without any processing.

Triplet Loss is a loss function where a baseline input is compared to a true input and a false input. Furthermore, this function tries to minimize the distance between the anchor and the positive input, and maximize the distance between the anchor and the false input. This concept is explained in Figure 3.7

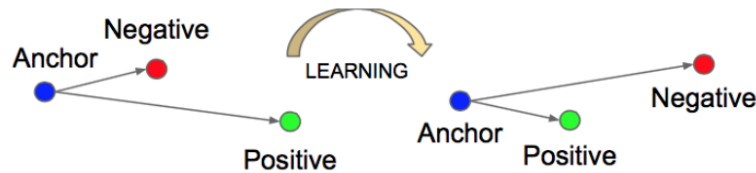


Figure 3.7: Triplet Loss concept training example [11]

This loss function can be described as an Euclidean distance function:

$$L(A, P, N) = \max(\|f(A) - f(P)\|^2 - \|f(A) - f(N)\|^2 + \alpha, 0)$$

The L2 Normalization layer, which is the difference in architectures between GoogLeNet and FaceNet, can be defined as the normalization technique that modifies the dataset such as each row has elements that squared give the sum maximum 1. This layer is very useful such that correct vector embeddings of the faces are created before the triplet loss step.

3.3.3 GoogLeNet architecture

Inception neural networks were developed because it was found that by building a plane neural network (densely-connected), the bigger the model would get, you would need more and more computation power. This was a major drawback before these Inception neural networks were developed. [22]

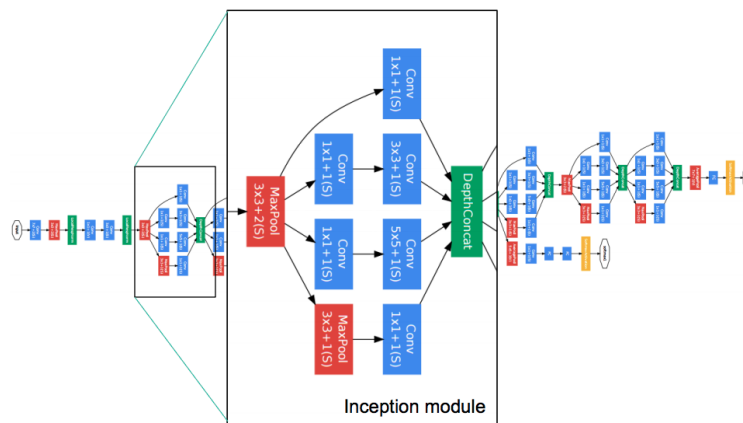


Figure 3.8: GoogLeNet detailed architecture [12]

This kind of neural network is built out of inception modules, which are built out of a max-pooling layers, a couple of convolutional layers, and a depth concatenation layer. Because of the fact that the layers are not connected one after another, they “fire together” and there is an amazing increase in performance and also in reductionality of the trainable parameters of the network. The 1x1 convolutional layers are used to reduce the dimensionality in order to make the training of the model faster. In comparison to the original convolutional neural networks, inception network tend to not get progressively bigger, but wider.

Also, in order to avoid overfitting, a drop-out layer is used before the classifier layers. Dropout refers to randomly ignore neurons, meaning that some parameters are not taken into consideration during training. A fully connected layer occupies most of the parameters, and because of that the neurons develop co-dependency between each other during training. This fact is the obvious reason why the network will overfit, because this codependency cuts the individual power of each neuron.

Regarding the activation function, all the convolutions, including those inside the Inception modules, use the rectified linear activation (ReLU).

The network also contains two extra Softmax branches. These branches are called classifiers and are located at the end of the Inception4a and Inception4d modules. During the training phase, their loss is added to the total loss of the network with a discount weight (weighted by 0.3). These branches are built out of an average pooling layer, a 1x1 convolutional layer, a fully-connected layer, a dropout layer (0.7 weight discount) and a fully-connected layer with Softmax classification (removed at inference time). The architecture described in this chapter can be visualised in Figure 3.8.

Chapter 4

State of the art on speaker recognition

4.1 Speaker recognition datasets

4.1.1 VoxCeleb2 Dataset

VoxCeleb2 is a dataset built specifically for speaker recognition and contains over 1 million utterances of over 6,000 celebrities, extracted from videos uploaded on YouTube. The dataset is generally gender-balanced, with male persons comprising 61 percent of the utterances. The speakers are widely spread among all categories of ethnicity, age, gender and profession. The data is taken from various environments, such as outdoor stadiums, quiet indoor studios, speeches, interviews from red carpets etc. [13]

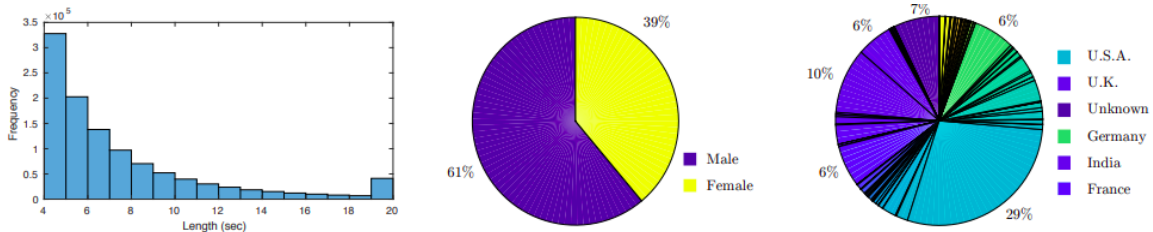


Figure 4.1: (left) Distribution of utterance lengths; (middle) Gender distribution; (right Nationality distribution of speakers) [13]

Figure 4.1 tries to build a visual statistical description of this dataset and has three parts: the first part is technically a histogram describing the distribution of utterance lengths, the second part shows the gender distribution in this dataset and the third part shows the nationality distribution of people appearing in this dataset.

The audio segments contained by the VoxCeleb2 dataset are loaded with background chatter, laughter, overlapping speech and room acoustics, such that the trained networks could generalize better, and the phenomenon of overfitting not to occur.

4.2 Previous work

The rush for speaker recognition began almost 20 years ago, when state-of-the-art was based on the GMM-UBM approach. This approach consists of using a universal background model (UBM) which would represent the acoustic features of a large set of persons. In this particular case, there were two phases: enrollment and test. In the first phase, GMM of individual speakers were adapted from the UBM. The second phase consisted of computing the similarity or the likelihood ratio between the test acoustic features and the enrollment. Although robust, this method seemed very sensitive to inter-session variability. [14]

Moreover, because of the fact that the utterances had different lengths, the algorithm would produce badly calibrated likelihoods. These results forced the neural networks community to approach a normalization method. The joint factor analysis (JFA) makes the algorithm better

by estimating the target GMM better and the test experiment more robust to the problem stated before. However, a score normalization is needed in order to achieve better results. [14]

The big advancement came in 2011, when the i-vector approach was introduced. The i-vector approach can be described as a compression of each recording into a unique fixed-length embedding. This embedding is then fed to a downstream classifier. In order to see the results of such a classifier, the first proposed scoring technique was LDA (linear discriminant analysis) to compare the enrollment vectors with the test i-vectors. Later, the authors also introduce the PLDA (generative probabilistic LDA), which produced more balanced results and likelihood ratios without normalization. [14]

Recently, there has been a very large interest in applying neural networks to the speaker recognition task. Such a task can be accomplished by using a DNN trained for ASR to compute the Gaussian component responsibilities of the i-vector model. [14]

Finally, very recent work imply the removal of both GMM and i-vector and only using Deep Neural Networks to compute sequence embeddings. These DNNs are used to classify acoustic features into a set of training speakers. In the evaluation step, an embedding is obtained by taking the output of the final hidden layer of the network. Also, in 2017, Zhang and Koishida [34] propose using Triplet Loss to increase the margin of the between embeddings of different speakers. The last innovation, called x-vector, can be described as the usage of embeddings extracted from networks using pooling and convolutional layers, these networks being trained using categorical or binary cross-entropy cost functions. Several variants of the x-vector approach have been published in which the emphasis is put on different studying different pooling methods and training objectives. [14]

Layer	Layer Type	Context	Size
1	TDNN-ReLU	t-2:t+2	512
2	TDNN-ReLU	t-2, t, t+2	512
3	TDNN-ReLU	t-3, t, t+3	512
4	Dense-ReLU	t	512
5	Dense-ReLU	t	1500
6	Pooling (mean+stddev)	Full-seq	2x1500
7	Dense (x-vector)-ReLU		512
8	Dense-ReLU		512
9	Dense-Softmax		Num. spks.

Figure 4.2: Examples of x-vector architectures [14]

Figure 4.2 is a summary of what has been talked about in this chapter. It describes the layers and their ordering in a state-of-the-art network based on x-vectors.

4.3 Network architectures

4.3.1 VGGVOX architecture

The task at hand was built onto an already trained network, a modified ResNet34 (Thin ResNet) architecture which is detailed in the article “Utterance-level Aggregation For Speaker Recognition in the Wild”. This network can be divided into three separate blocks: frame-level feature extraction, temporal aggregation of frame-level features and the final block, which is actually the optimization of a classification loss. [35]

The first element in the pipeline, which is the feature extraction, is done by using a conventional convolutional DNN which takes as an input the spectrograms computed on the input “.wav” files and outputs a fixed-length feature vector.

layer name	res-34	res-50
conv1	$7 \times 7, 64, \text{stride } 2$	$7 \times 7, 64, \text{stride } 2$
pool1	$3 \times 3, \text{max pool, stride } 2$	$3 \times 3, \text{max pool, stride } 2$
conv2_x	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
conv3_x	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$
conv4_x	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$
conv5_x	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$
fc1	$9 \times 1, 512, \text{stride } 1$	$9 \times 1, 2048, \text{stride } 1$
pool_time	$1 \times N, \text{avg pool, stride } 1$	$1 \times N, \text{avg pool, stride } 1$
fc2	$1 \times 1, 5994$	$1 \times 1, 5994$

Figure 4.3: VggVox architecture

Figure 4.3 shows the architecture of the feature extraction, which was modified for better performance. (ResNet34 - 22 million parameters, Thin ResNet34 - 3 million parameters).

The next step in the pipeline is the temporal aggregation of the frame-level features. This is done by using a concept named NetVlad. NetVlad takes the frame-level descriptors as input and produces a single matrix with the dimensions $K \times D$, where D is the dimension of each cluster and K is the number of clusters chosen. The final output is obtained by l2 normalizing and concatenating. After this, in order to reduce the dimensionality, a fully-connected layer is used to output a dimensionality of 512. Finally, a Softmax classifier is applied at the output in order to make a prediction based on the features learned prior to the classification step.

Chapter 5

Technologies

5.1 Programming languages

5.1.1 Python

Python is the main programming language used in my project. The principal reason of this is because of the existence of a lot of resources, including papers, frameworks and libraries that help in organising and implementing a deep learning project, such as Keras, Tensorflow Pytorch or SKLearn.

Furthermore, Python is a high level language, which is easy to understand and use. Even a person without many programming skills but who wants to work with deep learning frameworks can learn it. Also, Python can be used with programming paradigms such as object-oriented programming and functional programming. [36]

A very important piece in developing a programming project is managing libraries. By using Python, certain applications like "pip" or "Conda" can help managing and keeping track of what libraries you use during the project development. These applications also help tracking the libraries' versions and dependencies.

Python also offers the possibility of using a virtual environment. Each virtual environment has Python at its core and on top the libraries needed to build a project. The usage of virtual environments is useful in many situations, especially if the programmer works on many different projects and wants not to mix libraries' versions and make debugging really hard. Another really good advantage of using a virtual environment is portability, because this entity can be used by other people without manually installing each library used.

In this project, Python was used for writing the automated scripts for creating the audio and image databases, validating them, writing and evaluating the model and pre-processing data.

5.1.2 Java

Java is a general-purpose programming language that is class-based, object-oriented and designed to have as few implementation dependencies as possible. It is intended to let application developers write once, run anywhere, meaning that compiled Java code can run on all platforms that support Java without the need for recompilation.

Java is also a high-level programming language, like Python, which is used in this project only for extracting the information needed to construct the databases from the CDEP website. That information is extracted by using the Java language BeautifulSoup and also using the object-oriented programming paradigm that Java offers.

5.2 BeautifulSoup (JSoup)

JSoup is a Java library for parsing HTML from a URL, file or string. It is used for finding and extracting data, using DOM traversal or CSS selectors, manipulate the HTML elements,

attributes and text by using the object-oriented paradigm. Also, Jsoup is designed to deal with all variety of HTML found in the wild, and build a valid parse tree. [37]

In this project, this library is used in order to parse the CDEP website in order to obtain the transcripts for the videos, timestamps at which each speaker appears in the videos and download videos and audio separately.

5.3 Numpy

Numpy, which stands for Numerical Python, is a library consisting of multidimensional array objects and a collection of routines for processing those arrays. Using NumPy, mathematical and logical operations on arrays can be performed. [38]

This library is the building block of all the data manipulation in our project.

5.4 Pandas

Pandas is a Python package providing fast, flexible and expressive data structures designed to build datasets with "relational" or "labeled" data more easily. It aims to be the fundamental high-level building block for doing practical, real world data analysis in Python. [39]

This library is used in our project for data visualisation, the verification and validation of the image dataset, and debugging.

5.5 Deep learning frameworks

5.5.1 Keras and Tensorflow

TensorFlow is a free to use framework created by Google Brain in order to implement numerical algorithms used in deep learning applications. The main purpose of this framework is to facilitate and implement deep learning models and optimization of the training time and effort. [40]

TensorFlow allows developers to create dataflow graphs - structures that describe how data moves through a graph, or a series of processing nodes. Each node in the graph represents a mathematical operation, and each connection or edge between nodes is a multidimensional data array, or tensor.

TensorFlow provides all of this for the programmer by using Python programming language, because of the fact that it is easy to learn and work with, and also provides convenient ways to express how high-level abstractions can be coupled together. For example, nodes and tensors in TF are Python objects, and the applications are themselves written in Python.

Although data is represented in Python, the actual mathematical operations are not performed in Python. The libraries of transformations are written as high-performance C++ binaries. Python just directs traffic and combines the pieces together, providing high-level programming abstractions.

A great advantage of this framework is the fact that it allows the acceleration of operations by using graphical processors, which have a great parallel calculus power. Also, Tensorflow offers the alternative of training distributively, by using a system based on messages.

Even though TensorFlow is a high-level framework, a layer of abstractization was added called Keras, which is also a framework. By comparison, Tensorflow facilitates the definition of operations in a neural network by using a graph, where in Keras the network can be defined by a graph very easily. Further, Keras defines operations by using a core which can be Tensorflow, CNTK, Theano or even a programmer-defined framework. [41]

With Keras, a neural network can be defined by connecting each layer with each other and also define them the way the programmer wants. Also, Keras has a lot necessary tools for evaluating and debugging a neural network.

The training process is facilitated by some tools provided by Keras such as:

- Displaying a command line interface which contains information about the current state of training, more precisely the performance metric of the training reported to the total number of examples, number of the current epoch, training error and loss, and validation error and loss.
- Callback functions, which automatically run during certain stages of training. These functions can be defined by the programmer, but some are already given with the default framework. The most frequently used callback functions are the ones called "Checkpoint", which saves the model based on a metric chosen by the programmer or the function "Early Stopping" which stops the training at a point chosen by the programmer based on an optimization parameter
- The possibility of training using an object called a generator, which loads data progressively without overloading the memory, and also creating a uniform distribution.

Both Keras and TensorFlow come with a couple of architectures of deep neural networks with pre-trained weights. They are used in our project for transfer learning, by blocking the training for the layers with pre-trained weights.

5.5.2 PyTorch

PyTorch is a Python machine learning package based on Torch, which is an open-source machine learning package based on the programming language Lua. The two main features of Pytorch are the Tensor computation with a strong GPU acceleration and the automatic differentiation for building and training neural networks. [42]

Similar to Keras and Tensorflow, PyTorch also uses the graph representation for deep neural networks. Tensors in PyTorch are similar to numpy's ndarrays, with the addition being that Tensors can also be used with GPUs.

Also, Pytorch uses an imperative / eager paradigm, which means that each line of code required to build a graph defines a component of that graph. The advantage with this paradigm is that we can independently perform computations on these components itself, even before the graph is complete.

5.5.3 SKlearn

SKLearn, also called Scikit-Learn is a library in Python that provides many unsupervised and supervised learning algorithms. This library is built upon other Python libraries such as Numpy, pandas, and Matplotlib. This library was developed by David Cournapeau as a Google summer of code project in 2007. [43]

There are a lot of functions this library provides, such as:

- Regression
- Classification
- Clustering

- Model selection
- Pre-processing of data

The library is mainly built on "SciPy" that must be installed before "SKLearn" can be used. The predominant used libraries are:

- NumPy: Base n-dimensional array package;
- SciPy: Fundamental library for scientific computing;
- Matplotlib: Comprehensive 2D/3D plotting;
- IPython: Enhanced interactive console;
- Sympy: Symbolic mathematics;
- Pandas: Data structures and analysis;

5.6 Ffmpeg

Ffmpeg is a free and open-source library for various programming languages such as Python, Java and C++. It consists of a vast software suite of libraries and programs for handling video, audio, and other multimedia files and streams. The core of "Ffmpeg" is designed for command-line based processing of audio and video files, but can be used automatically by building scripts in Python with the "subprocess" built-in library. [44]

5.7 OpenCV

OpenCV (Open Source Computer Vision Library) is an open source library meant for computer vision and machine learning. It was build to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in commercial products. [45]

The library has more than 2500 optimized algorithms, which includes a complete set of both classic and state of the art computer vision and machine learning algorithms. These algorithms can be used in a lot of applications such as detection and recognition of faces, identifying objects, classify human actions in videos, tracking camera movements, extracting 3D models etc.

For my project, "OpenCV" was used for manipulating videos in order to extract relevant frames, face detection, grayscale conversion and cropping the relevant parts of images.

5.8 LibROSA

LibROSA is a python package for music and audio analysis. It provides the building blocks for creating algorithms for audio signal processing. This library was used in this project for processing the audio, segmenting it and organizing it into a relevant database. [46]

5.9 CUDA

CUDA is a parallel calculus platform developed for the Nvidia GPUs. The purpose of this platform is to allow developers to accelerate the execution of heavy-load computation by using Graphical Processing Units. [47]

The combination of CUDA and Nvidia became very used in various domains which need very high performances, such as training neural networks, data science, numerical analysis, imagistics or bioinformatics.

CUDA is used in neural networks training along with other specialised libraries which offer support, such as Keras or Tensorflow. In this case, a special library was developed named cuDNN which is specialized in performing mathematical operations specific to deep neural networks.

Chapter 6

CDEP Dataset development

CDEP website contains videos of plenary meetings in the Deputy Chamber organized by date. For each video, this website provides IDs of the speakers, names of the speakers and URLs for them. It also provides human readable transcriptions of the speeches along with irrelevant information and timestamps for each speech of each speaker.

6.1 Previous work

The task of dataset development was developed onto an existing project, created by the Speed laboratory. The existing project was created in order to parse the ".html" files on the CDEP website in order to extract a number of pieces of information required in our project. Below, there will be a list of the existing applications and their functions:

- Application 1, DownloadCDepOriginal, which download videos and ".html" files in a certain timeline.
- Application 2, wavExtractor, which extracts ".wav" files from the video files and classifies them correctly.
- Application 3, HTMLProcessor, which processes ".html" files and extract transcriptions, execute sox commands to cut the long audio files into short audio files containing one speech each.
- Application 4, CDEPTranscriptionsCleaner, which process the original html and transcriptions files and create the ASR transcriptions.

6.2 Overview of data processing tasks

The applications described in the above section were used to retrieve the information needed in order to start our recognition project. This information helps by summarizing the data we had available at the beginning of the project:

- Videos and their transcriptions.
- HTML files corresponding to the downloaded videos.
- Audio files, each corresponding to a speech.
- "cdepspk.txt" file which contains IDs for each speaker in the database and information about them on the CDEP website.

Besides the data described above, we needed the timestamps for each speech according to the video and the speaker. In order to do this, we modified the HTMLProcessor application to extract the timestamps only and organize them into a ".txt" file. The application was developed

in Java programming language and the "JSoup" library was used in order to parse the ".html" files and extract the timestamps. [48]

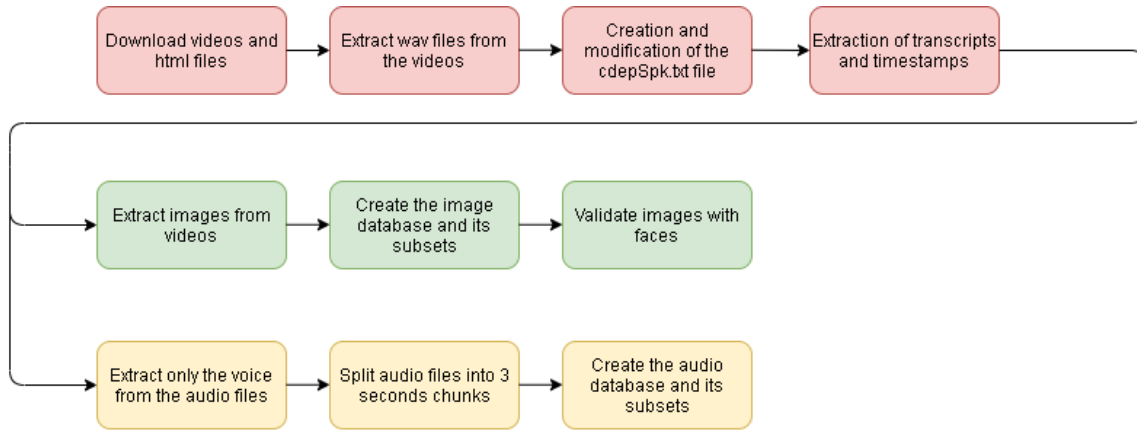


Figure 6.1: Flowchart of the data processing task

Figure 6.1 tries to show an overview of the task of extracting the data and creating the databases required for the multimodal recognition task. The red blocks are actions that are done before the development of our data processing tasks, while the green and yellow blocks are actions done throughout our project. These actions are the building blocks of the creation and validation of the datasets used further in our project, such as the extraction of images with faces from the videos or extracting only the voice from the audio files retrieved before.

6.3 Image dataset development

6.3.1 Image extraction

The first task in the image dataset development is to analyse the videos and identify how to isolate individual speeches in a video which contains many people talking. We have achieved this by parsing the timestamps extracted from the official website and create a database structure where a folder is assigned to each person containing split videos (each video is a speech). We then needed to identify a facial detection method to extract only frames which contained only one face.

After the videos are split, we need to apply an algorithm of face detection in order to identify the frames that contain only a face, and then extract them. For this, we have used OpenCV and a pre-trained Haar-cascade classifier. Figure 6.2 explains how the image extraction algorithm works.

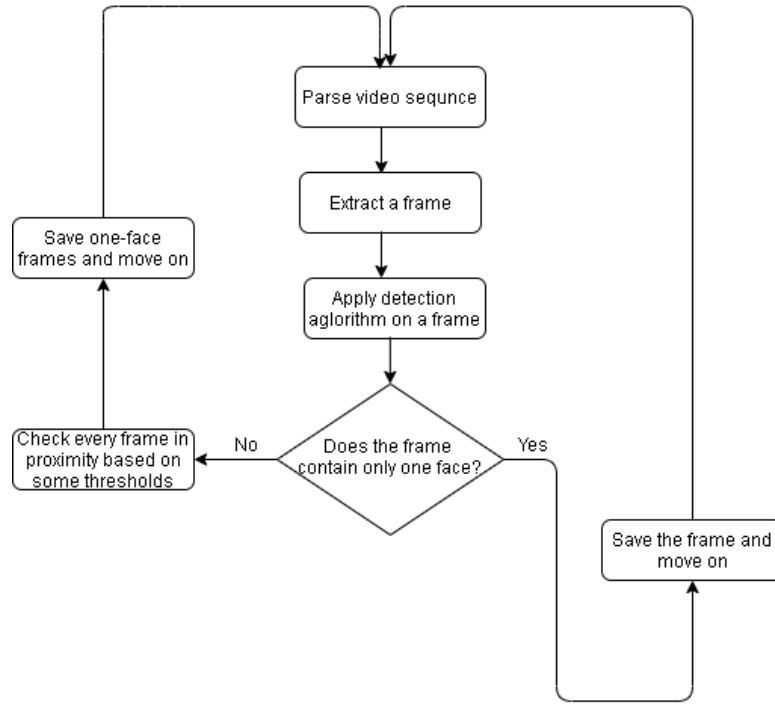


Figure 6.2: Flowchart of the extraction of images algorithm

It can be seen that there are a couple of important steps in this algorithm. Firstly, we need to consider the video as a sequence of frames. Based on the step we configure for the checking of frames, we extract each frame and apply the face detection algorithm. If this algorithm finds that the frame contains only one face, we save the frame and move on with the step we chosen, in milliseconds. Although, if this frame did not contain any faces, or more faces than we needed, the next step consisted in checking the proximity of that frame in order to see if we could find more valid frames, and then move on again. The valid frames were then saved with a suggestive names for each class in a distinct folder with the name being the ID of the corresponding person/class.

6.3.2 Obtaining the similarity matrix

Before diving into the details of obtaining the similarity matrix of a certain class, Figure 6.3 shows the main steps required to be taken in order to obtain a similarity matrix.

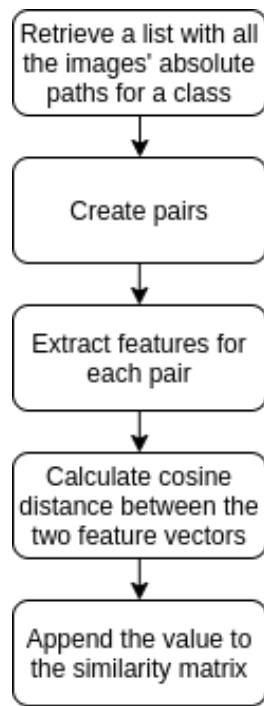


Figure 6.3: Flowchart of the creation of the similarity matrix

Because of the fact that the database contains only cropped images of faces, not random ones, a process called feature extraction using a pre-trained model needs to be used. This process can be described as a dimensionality reduction in order to drastically reduce the amount of variables of the input and to have a greater impact onto the next steps of the analysis(layers that learn different characteristics of a face).

The pre-processing step of this task is to load every image and resize it to match the input layer of the neural network used. After this, the python library “itertools” is used in order to create a list with all the combinations of images, two by two, so that they can be compared further in the algorithm.

Next, in order to extract the features for each picture, the concept that was described in the first paragraph will be applied. A convolutional neural network with the stripped output layers and already pre-trained weights will be used to extract the features of the images. By removing the last fully connected layers of the VGG19 convolutional network, the input will only forward propagate and will stop at a specific layer where the features of that input will be taken as an output.

The Keras model that is used in order to solve this similarity problem is the famous VGG19 architecture which may be described as a linear neural network containing only convolutional and max-pooling layers, aside from the input and output layers. Because of the fact that we use this model just to extract features from the images, the output fully connected layers won’t be needed for this task at hand. Also, the fact that in this case a prediction is not needed at the output, the model will be used with a set of pre-trained weights. This model was trained in order to recognize faces, and it contained approximately 2600 different persons. [32] These are a few examples of the types of images are used in order to train the model, shown in Figure 6.4.



Figure 6.4: Example images from VGGFace dataset

After the process of feature extraction for all the images, these vectors will be mapped onto the names of the targets. The distance between two images will be evaluated by using the cosine distance between two vectors. This distance will be subtracted from 1 in order to get the similarity score. This value will be added into a similarity matrix for the specific class, which will be appended into a number of csv files. Next, a ".csv" file is created with all the mean values of similarity for each specific class, in order to find a threshold from which a folder can be immediately discarded due to the smaller similarity score. After this step, an algorithm is built in order to find and eliminate every faulty image from each class such that the database is clean and the image neural network will be trained on it.

6.3.3 Algorithm development

The first step in creating the image dataset was to split the videos which contained more than one speaker into separate videos, each representing a speech. In order to do that, the file containing the timestamps was parsed and the videos were automatically split and labeled with the ID of the speaker and the date of the speech. The script was designed using Python language, "subprocess" library and "Ffmpeg" software library. After the process of splitting videos into separate ones, each representing a speech, we can move on to the next step. It consists in finding a software for automatically detecting faces in a frame. We found out that the library OpenCV has a built-in function which uses a Haar-Cascade classifier trained on a large dataset of faces in order to detect faces in pictures. [49]

The algorithm we developed takes a video and save a given number of frames which contain only a face, and then crop the faces from the images, such that no unnecessary information is preserved. Also, the frames are saved only and only if the frame contains a single face, such that there is a confirmation that the person in the image corresponds to the person's ID. By inspecting the algorithm in this first state, it was found out that some frames were wrongfully classified as faces. This motivated us to find a second method to detect faces in a frame and found out that MIT had a detection algorithm based also on Haar-Cascade classifiers [50]. By using both methods for safety measurements, the error rate was much smaller.

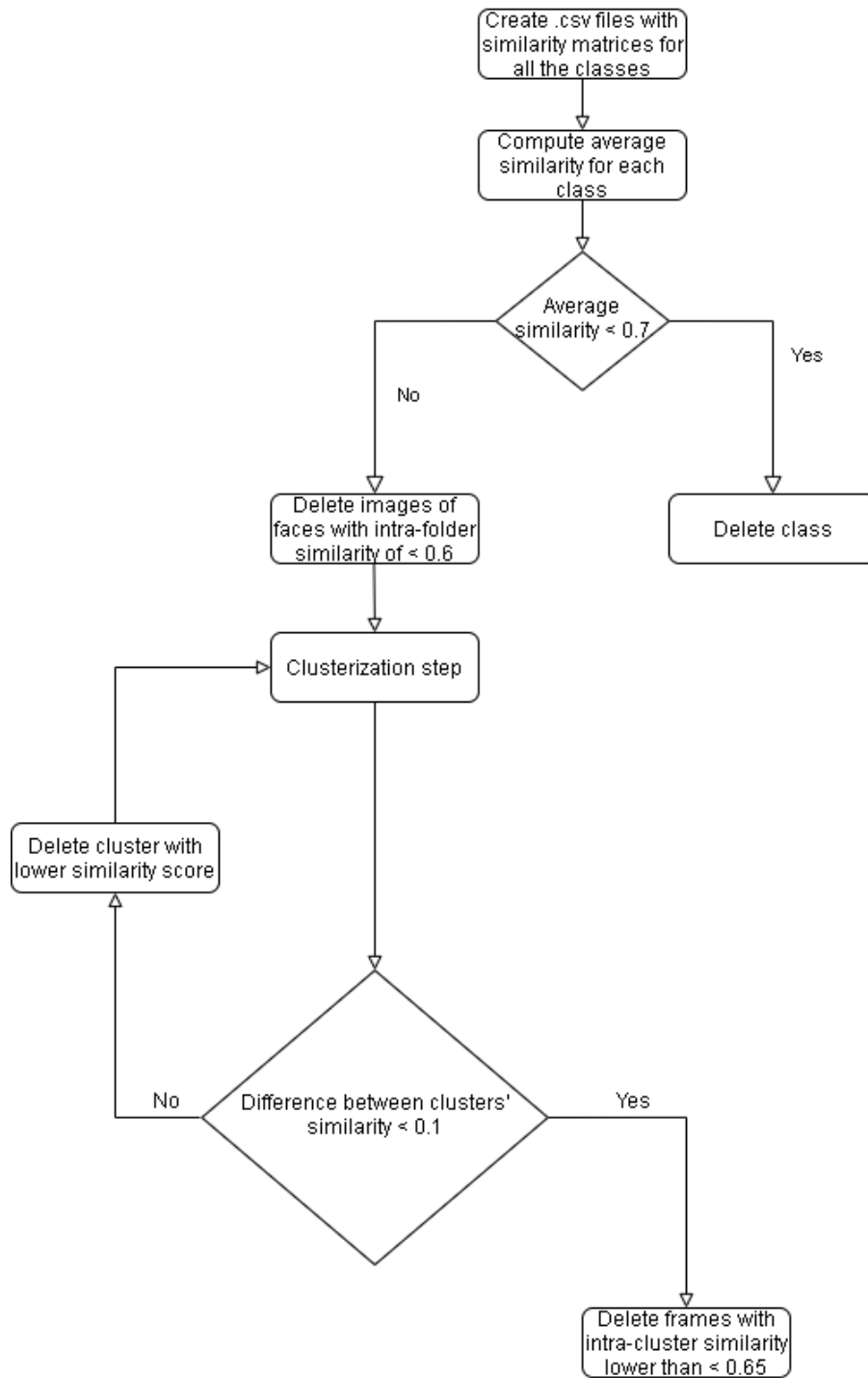


Figure 6.5: Flowchart for the image dataset validation algorithm

After the process of extraction, we noticed by manual verification that still there were some issues with the dataset, such as folders with more than one person, or bad images not containing faces. Figure 6.6 shows examples of bad images taken from the original dataset.



Figure 6.6: Example of bad-labeled images

In order to do this validation steps, we documented on how to find the difference between two pictures, called similarity between these two. We needed to check the similarity between two faces, not two general images, so comparing each pixel by the complementary one in the other images is not a good process of validation. Although it was a hard task to develop, we found that we can find how different these two images are by introducing them as an input in a CNN without the classifications layer, and extract a feature vector from each image. After the extraction of the feature vector for all the images, a good way to calculate the difference between two vectors is the cosine similarity, which is a mathematical algorithm explained by the formula below:

$$similarity(A, B) = \frac{A \times B}{||A|| \times ||B||} \quad (6.1)$$

After calculating this value, it is subtracted from 1 and the similarity is found. Now, we needed to extrapolate for the whole problem: creating a similarity matrix containing the similarities between each two combinations of images in a folder. By generating all the similarity matrices we did a visual inspection on the values and by conditional formatting all the matrix in google spreadsheets we found out that if we checked the values on columns we could find out what photos are bad. Furthermore, we found out that it is a valid verification to check the average values for each image (average on column). This resolve solved only one problem, the problem of having almost all images correct, except a few that don't belong in the folder.

Before passing onto the major issues of the data validation, we had to compute the average similarities of every class, and found out by manual verification of the pictures that the folders with difficult problems had the average similarity of less than 0,77.

Figure 6.7 shows a list of classes with their average similarity and a couple of observations made when manually verified.

Medie scor similitudine	PersonId	observatii
0.644	00454.csv	am verificat manual; sunt poze de la mai multe persoane
0.654	02336.csv	am verificat manual; sunt poze de la mai multe persoane
0.656	02399.csv	am verificat manual; sunt poze de la mai multe persoane
0.713	01704.csv	am verificat manual; sunt poze de la mai multe persoane
0.721	01972.csv	am verificat manual; sunt poze de la mai multe persoane
0.736	02454.csv	am verificat manual, sunt poze de la aceeasi persoana dar 27 poze proaste
0.736	02257.csv	am verificat manual, sunt poze de la aceeasi persoana dar 3 poze proaste
0.745	02300.csv	am verificat manual, sunt poze de la aceeasi persoana dar 7 poze proaste
0.746	02259.csv	am verificat manual, sunt poze de la aceeasi persoana dar 5 poze proaste
0.755	02388.csv	am verificat manual; toate pozele sunt de la aceeasi persoana
0.755	02268.csv	am verificat manual, sunt poze de la aceeasi persoana dar 7 poze proaste
0.756	00293.csv	am verificat manual; e ok; toate pozele sunt de la aceeasi persoana

Figure 6.7: List of classes, their average similarity and observations

Figure 6.8 is an example of the process of extracting the correct value for the similarity of bad images. By inspecting both the matrix which was conditional formatted in Google Spreadsheets and the images themselves, we found out that a red column represents an image that has issues. In our example, columns 3, 6 and 12 represent bad images, because of the fact that their similarity is very low compared to the other images (0.5).

1.0	0.9	0.4	0.9	0.8	0.5	0.9	0.9	0.9	0.9	0.9	0.5	0.9
0.9	1.0	0.4	0.9	0.8	0.5	0.9	0.9	0.9	0.9	0.9	0.5	0.9
0.4	0.4	1.0	0.4	0.5	0.6	0.5	0.4	0.4	0.4	0.4	0.6	0.4
0.9	0.9	0.4	1.0	0.8	0.5	0.9	0.9	0.9	0.9	0.9	0.5	0.9
0.8	0.8	0.5	0.8	1.0	0.5	0.8	0.8	0.8	0.8	0.8	0.5	0.8
0.5	0.5	0.6	0.5	0.5	1.0	0.5	0.5	0.4	0.4	0.5	0.5	0.5
0.9	0.9	0.5	0.9	0.8	0.5	1.0	0.9	0.9	0.9	0.9	0.5	0.9
0.9	0.9	0.4	0.9	0.8	0.5	0.9	1.0	0.9	0.9	0.9	0.5	0.9
0.9	0.9	0.4	0.9	0.8	0.4	0.9	0.9	1.0	0.9	0.9	0.5	0.9
0.9	0.9	0.4	0.9	0.8	0.5	0.9	0.9	0.9	1.0	0.9	0.5	0.9
0.5	0.5	0.6	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	1.0	0.5
0.9	0.9	0.4	0.9	0.8	0.5	0.9	0.9	0.9	0.9	0.9	0.5	1.0
average values												
0.8	0.8	0.5	0.8	0.8	0.5	0.8	0.8	0.8	0.8	0.8	0.5	0.8

Figure 6.8: Example of a similarity matrix for a class

The next problem was that there were folders which contained more people. For this problem, we searched how to use clusterization algorithms in order to create two clusters for these folders, one with bad images, and one with good images, and try to remove the one with the bad images. Regarding the clustering, we used the agglomerative clustering algorithm, which recursively merges the pair of clusters that minimally increases a given linkage distance. By running an algorithm we made in order to create the clusters, we found out that there were two types of folders with more people: one in which there was a clear visual of the good person in the folder, such that if the bad cluster was eliminated the problem would have been solved, or if there weren't two clear clusters and a single person could not be observed more in the folder. Figure 6.9 is a perfect example of a folder which has two very separable clusters, and it can be seen which one is the dominant one (purple).

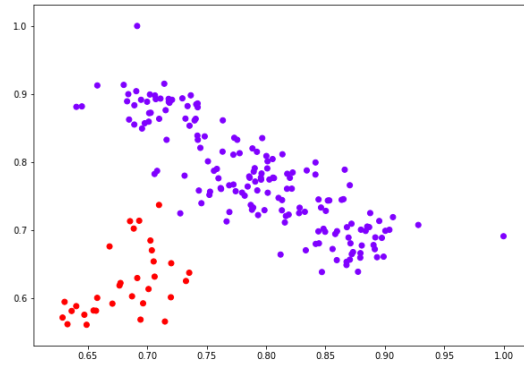


Figure 6.9: Clustering example of a folder with a dominant person

In Figure 6.10, it can be seen that the clusterization method took place optimally, and there can be seen two distinct clusters. In order to find out the good cluster, an algorithm was developed which is described in this chapter.

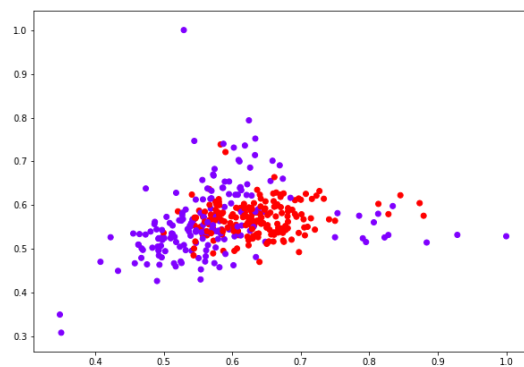


Figure 6.10: Clustering example of a folder with multiple non-dominant persons

In the second case we also found out that the average similarity score for the folder which contained those photos was very small (< 0.70), as seen in Figure 6.7, such that we decided to throw these types of folders away. Next we have done an analysis on how to increase the range of this clustering process, meaning that the first time, we applied a threshold which is a little sensitive if we wanted to apply this algorithm to a bigger dataset. In order to complete this analysis, we raised this threshold in order to see how folders in different scenarios, such as all the pictures are good, or only a couple of them were bad, would react to this threshold appliance, which means to see how many bad photos were retained and how many good photos were thrown away from a specific folder. Of course, we forgot to mention that after each step in which the folder with images changes, we had to recalculate its average similarity and the average similarity of every picture and recreate the ".csv" file for that particular folder.

After verifying a couple of times the data which we figured out that can be used a value which the algorithm uses in order to delete the bad cluster: the difference of similarity between the two clusters. We then took this difference for each folder and created a ".csv" file which we visualized in Google Spreadsheets. After analyzing the data shown in 6.11, we found out that we can also threshold this value in order to preserve a lot of good pictures which were thrown away due to the clusterizations. Because of that reason, we can raise the threshold for

the clusterization process to a higher value, resulting a lower sensitivity to the changes in the dataset.

Furthermore, after realizing these were a lot of split data obtained through manual verification, we merged these results into one google spreadsheet containing: name of the folder, manual observations, difference between the two clusters, the total of good photos, the total of bad photos, how many bad photos remained after the clusterization process, how many good photos were thrown because of the clusterization process.

After looking over this data, we managed to think of a reasonable second threshold for applying the clusterization: before, it was only the average similarity of the folder, now it will be the average similarity and the difference of similarity between these two clusters (approximately ± 0.1). So, the conclusion is that if a folder has an average similarity below 0.8 and a cluster difference of above 0.1, that means it needs the clusterization process. After testing the algorithm with these thresholds, we have found out two bad cases which will be described next.

The first bad case we observed was the fact that even after the clusterization process, a folder with multiple photos still hasn't been cleansed. So, after applying the clusterization, we were curious about the new difference between the two new clusters, and we found out it was still above the threshold. The next idea was that we can modify the algorithm in order to do a "loop clusterization" until the difference will be below 0.1, which got great results.

The second bad case is a little more complicated. By applying the thresholds we mentioned above, in a folder the situation did not stand so good. All the good photos were removed and all the bad pictures were retained, which is a dilemma. We realised that in this cases the only possible explanation is that the cluster of bad pictures has a greater similarity score than the one containing good pictures. We also noticed the fact that in these rare cases the average similarity of the bad photos before any algorithm was applied was below or around the threshold used to delete individual images in the bigger algorithm. So, keeping that in mind, we thought of a solution and realised that we can swap the two parts of the big algorithm of verification (clustering and deletion of individual images) such that those bad pictures will be removed and the folder will not get to clustering, because the similarity score of the folder will be raised and the difference between the two clusters will be very small, regarding the fact that the bad photos were removed. After modifying the algorithm, we performed the same analysis described above in which we verified the folders which initially had the similarity score less than 0.8. After this in depth analysis, we found out that the swapping of the two parts did almost nothing to the good data, and even better, has an almost flawless removal of bad images, better than the last algorithm. Also, after this analysis we found out that first threshold (0.6) might need to be a little higher, such that no bad photos remained. The final results after the validation algorithm is ran on our dataset can be seen in Figure 6.11

Average similarity score	PersonId	clustering difference	# bad kept photos	# good thrown photos	# total no. of bad photos	# total no. of good photos
0.713	01704.csv	0.16	0	0	110	98
0.721	01972.csv	0.20	1	0	37	48
0.736	02454.csv	0.11	0	34	27	158
0.736	02257.csv	0.17	0	0	3	10
0.745	02300.csv	0.07	0	0	7	35
0.746	02259.csv	0.16	0	0	5	45
0.755	02388.csv	0.04	0	1	0	68
0.755	02268.csv	0.05	0	0	7	55
0.756	00293.csv	0.02	0	1	0	198

Figure 6.11: Manual verification results example



Figure 6.12: Examples of data from the validated face dataset

Finally, we have plotted the person histogram, which can be seen in Figure 6.13, for the correct dataset in order to check the balance of it for the task of training a neural network and found out that the image dataset was badly balanced. To use that to our advantage, a resolve for the problem was to create a number of derived datasets with a fixed number of images for each speaker.

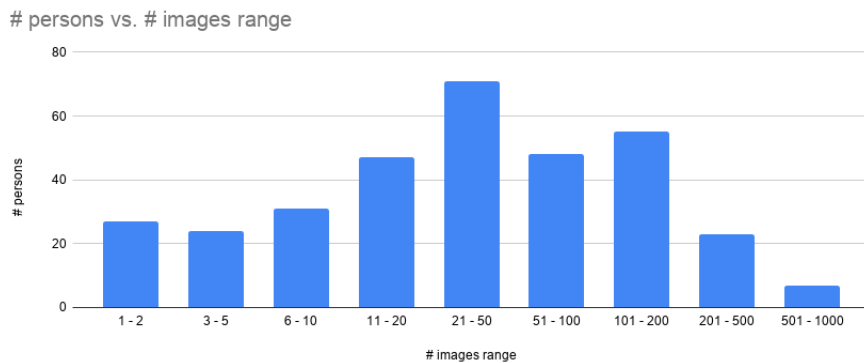


Figure 6.13: Person histogram

Table 6.1 describes the three derived image datasets regarding the number of classes and the number of samples per class. It can be easily observed the fact that the number of classes decreases proportionally to the increase in images per class.

Dataset	No. of images/class	No. of classes
Image10	10	257
Image50	50	132
Image100	100	84

Table 6.1: Image dataset configurations

6.4 Audio dataset development

Once the image dataset was created and validated, we crossed onto the audio problem, since the task is multimodal person recognition. The flow of the algorithm is similar to the image dataset development. Having the timestamps prepared, we had to extract the audio from videos using a Java application "wavExtractor" which also cut the audio according to the speaker and created a folder structure for the audio dataset. The steps required for the development of the audio dataset are depicted in Figure 6.14.

The first problem we encountered was the fact that a speech does not contain voice only, but noise and most importantly silence. In order to trim that silence, we have built a script in Python to automatically detect silent periods in the audio, cut them and remake the audio file such that the structure is not broken. The detection of silence was achieved by using a Voice Activity Detector [17] which is described in the "Technologies" chapter.

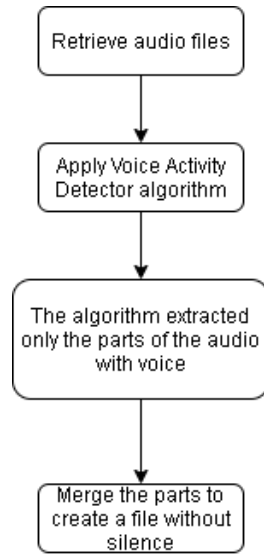


Figure 6.14: Flowchart explaining the audio database creation algorithm

After the dataset was silence-free, we have developed an algorithm to create a derived dataset based on the number of seconds and the number of files per speaker wanted. The final version of the dataset contains over 22000 three second ".wav" files from 331 speakers. By inspecting the Table 6.2, we notice the fact that there is an imbalance in the dataset, 250 persons having between 51 and 100 three seconds audio files and only 79 persons having between 3 and 50 three seconds audio files.

# audio files range	# persons
3 - 5	2
6 - 10	23
11 - 20	15
21 - 50	39
51 - 100	250

Table 6.2: Audio database configuration

Because of the data unbalance, we have decided to build a number of derived datasets which would only contain a fixed number of samples per each class, such that the unbalance would not be a problem in the recognition tasks. The Table 6.3 offers information regarding the number of classes and the total number of samples for each subset.

Dataset name	No. of classes	No. of samples
Audio10_3s	311	3110
Audio50_3s	252	12600

Table 6.3: Derived audio dataset configurations

6.5 Multimodal dataset development

The multimodal dataset is a combination of the image and audio datasets. Because of the fact that the number of classes differ between these two kinds of datasets, a correlation script was built such that both have the same classes. The script checks the two folders and builds a list for each of them with all the classes, and then perform an intersection operation between them, keeping only the common classes. After performing the step, the number of classes were modified only for the audio dataset.

Dataset name	No. of classes	No. of samples
Image10	257	2570
Audio10_3s		
Image50	132	6600
Audio50_3s		

Table 6.4: Multimodal dataset configurations

The Table 6.4 offers information such as the number of classes and the number of samples for the two databases which are going to be used for the multimodal classification of persons. We can observe the fact that as the number of samples per class grows, the number of samples also follows the trend.

Chapter 7

Face recognition

7.1 Experimental setup

In order to complete the experiments, we have used three environments: Google Colab, a machine provided by Google Cloud Compute and my own personal computer. The personal computer was used to write the code and interpret the results. The training was done on Google Colab or the Google Cloud Compute (GCP) machine, depending on the size of the database.

Table 7.1 describe the experimental setup for all the face recognition experiments, comprising of the database on which it was ran, the no. of classes, architecture, optimizer, image size, batch size, epochs of training and steps per epoch.

Furthermore, Table 7.1 shows the split of the derived databases for the training of the models. The split was done by the following rule: 60 percent training set, 20 percent validation set and 20 percent test set.

Dataset	No. of training examples	No. of validation examples	No. of test samples
Image10	1560	52	52
Image50	4050	1350	1350
Image100	5220	1740	1740

Table 7.1: Derived image datasets split numbers of examples

Due to the fact that the concept of transfer learning is the core of this paper, it can be seen that the image sizes are not equal due to differences in architectures. Also, the batch size for the GoogLeNet was chosen so small because of the fact that it was seen an increase of performance for a lower batch size by using PyTorch and CUDA.

Database	No. of classes	Architecture	Pretrained on:	Optimizer	Image Size	Batch Size	Epochs	Steps
Image10	257	FaceNet	MS-1-M Celeb	SGD(lr=0.02)	160,160	32	15	56
Image10	257	VGG16	VGGFace2	SGD(lr=0.2)	100,100	64	15	28
Image10	257	GoogLeNet	ImageNet	SGD(lr=0.02)	256,256	4	25	-
Image50	132	FaceNet	MS-1-M Celeb	SGD(lr=0.02)	100,100	64	15	28
Image50	132	VGG16	VGGFace2	SGD(lr=0.02)	100,100	64	15	28
Image50	132	GoogLeNet	ImageNet	SGD(lr=0.02)	256,256	4	25	-
Image100	84	FaceNet	MS-1-M Celeb	SGD(lr=0.02)	100,100	64	15	28
Image100	84	VGG16	VGGFace2	SGD(lr=0.02)	100,100	64	15	28
Image100	84	GoogLeNet	ImageNet	SGD(lr=0.02)	256,256	4	25	-

Figure 7.1: Experimental setup for face recognition

7.2 FaceNet architecture

The FaceNet classifier we chose for transfer learning was trained on the massive MS-Celeb-1M which consists of 10 million images across 100000 different persons. This fact makes it perfect for transfer learning. [33]

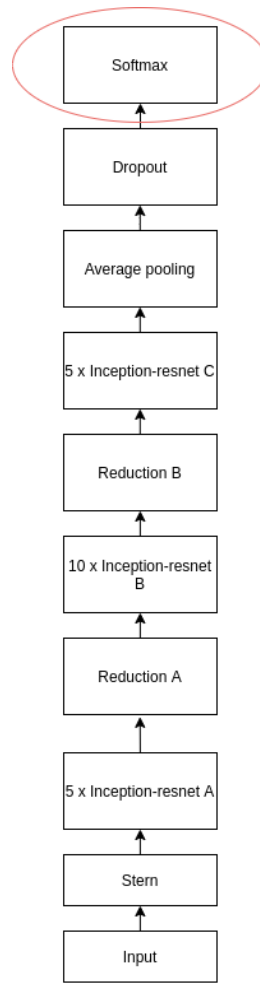
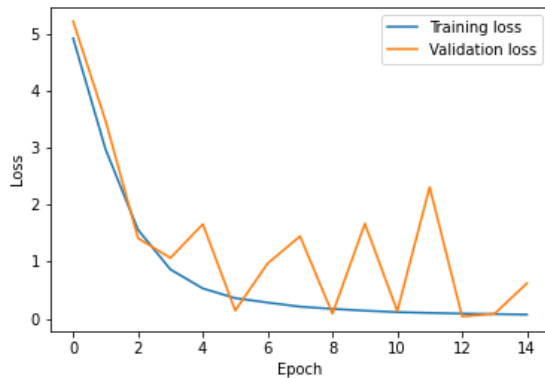
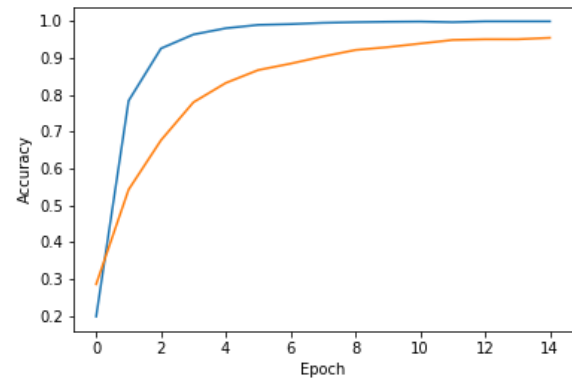


Figure 7.2: FaceNet fine-tuning

Because of the fact that it is trained on a dataset that is similar to ours, we do not need to unfreeze certain layers and train them again, because the features learned previously are from faces. So, what we did is to freeze all the layers in the network and change only the Softmax classification layer on top and start training, with the same hyperparameters used for the other networks, as seen in Figure 7.2.



(a) Training and validation loss curves

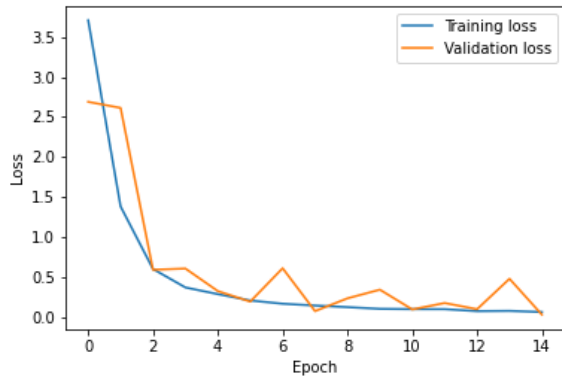


(b) Training and validation accuracy curves

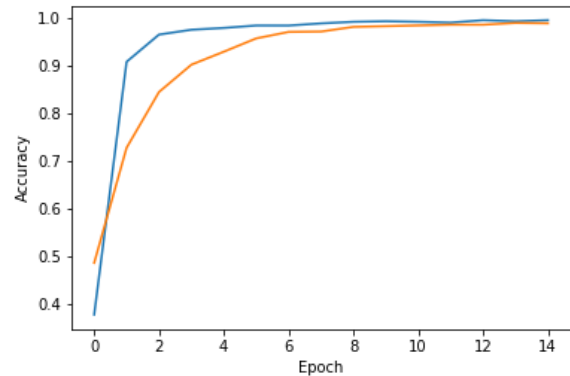
Figure 7.3: FaceNet experiment for 10 images/class dataset

By checking the loss curves and the accuracy curves of the experiment done on the 10

images/class database shown in Figure 7.3a, it can be seen that in the validation step, the algorithm cannot find a global minimum, which can be explained by the fact that there is too less data.



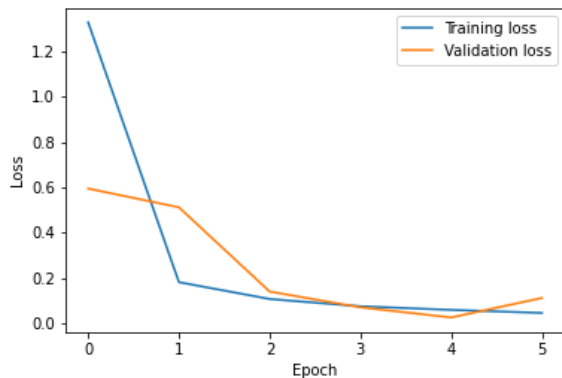
(a) Training and validation loss curves



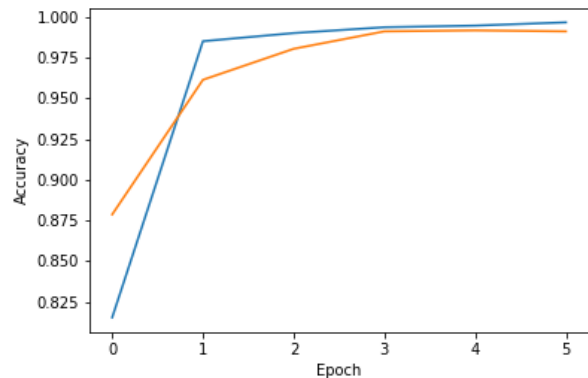
(b) Training and validation accuracy curves

Figure 7.4: FaceNet experiment for 50 images/class dataset

The next experiment is run on the database with 50 images/class. Figure 7.4a and Figure 7.4b show the training and validation loss curves and the training and validation accuracy curves. By observation, the loss curve is much stable now for both training and validation, because of the fact that there is more data and also the number of classes were diminished, from 257 to 132. The validation curve still shows signs of struggles in finding the local minimum, but the jumps' magnitudes are lower. The plateau for both the training and validation accuracy almost overlaps.



(a) Training and validation loss curves



(b) Training and validation accuracy curves

Figure 7.5: FaceNet experiment for 100 images/class dataset

By analysing the loss curves in Figure 7.5a, we can observe the fact that the experiments become more stable and the accuracy raises by increasing the size of the database. It can also be seen that the difference between the experiments with 50 images/class and 100 images/class is less significant than the one between 10 images/class and 50 images/class.

7.3 GoogLeNet architecture

The GoogLeNet model was trained also on a massive dataset called ImageNet. This dataset unfortunately does not contain only faces, so it fits our problem partially. For this, we had to do a lot of experiments with the unfreezing of certain layers and check out the performance of the network in predicting on the test set.

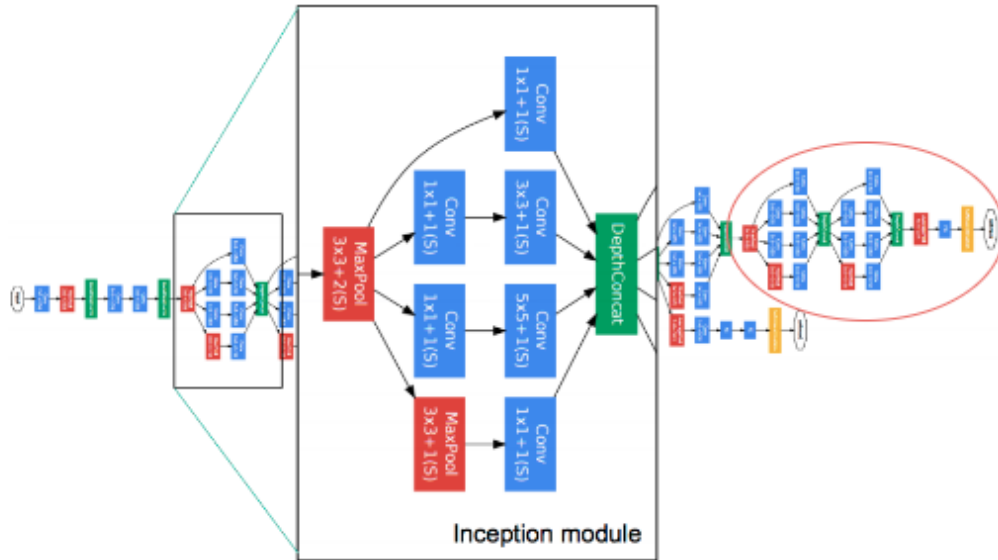
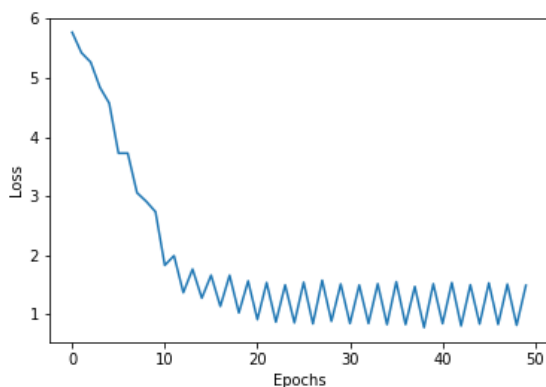
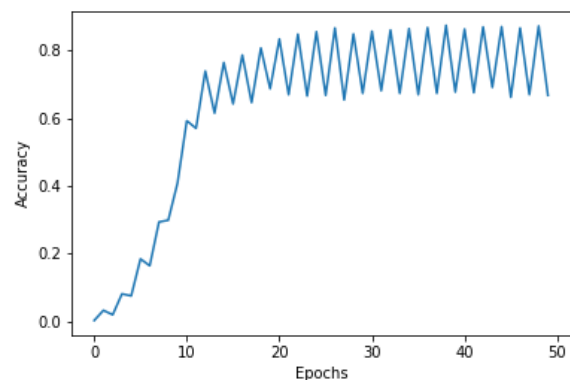


Figure 7.6: GoogLeNet detailed architecture with emphasis on fine-tuning

Firstly, we had to replace the top layers to what our classification corresponds to. After checking the results with only training the top layer (Softmax), the results were not so good, meaning that it had accuracy of about 65 percent on the test set. We figured out the fact that those pre-trained top layers were not trained on faces, so the high-level features of the network didn't correspond to that of a face. We started to unfreeze one inception module and so on, until we found that at about 2 unfrozen inception blocks, the network started predicting very well on the test set (about 90 percent), as seen in the results and in Figure 7.6. Also, this network was trained with the same hyperparameters as the last two described models, such that a comparison can be made.



(a) Training and validation loss curves



(b) Training and validation accuracy curves

Figure 7.7: GoogLeNet experiment for 10 images/class dataset

By analysing the loss graph for the first GoogLeNet architecture experiment in Figure 7.7,

we can observe the fact that the phenomenon of high bias appears due to lack of data in the validation set. Also, this trend is seen in the accuracy graph as well.

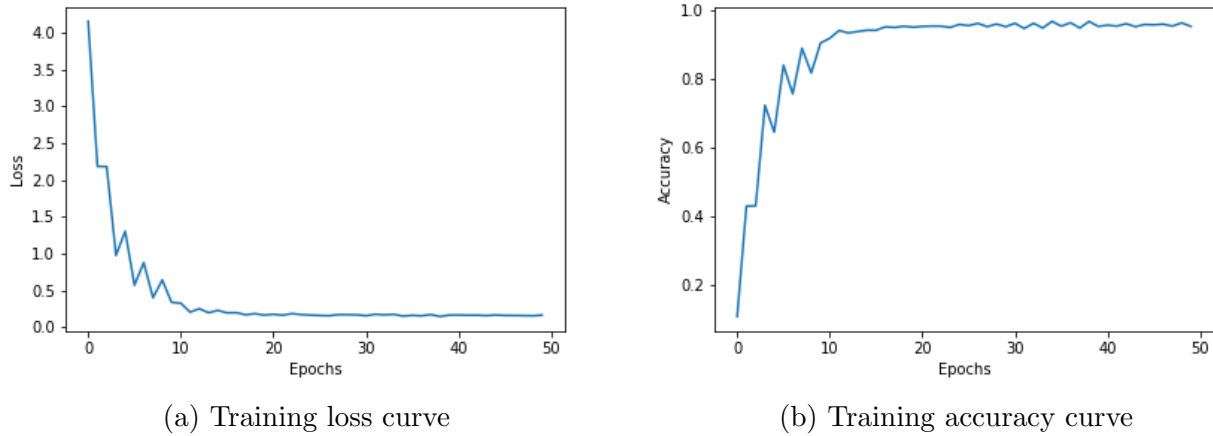


Figure 7.8: GoogLeNet experiment for 50 images/class dataset

After we ran the experiment for the 50 images/class dataset, we noticed a much better loss curve in Figure 7.8. There were still spikes on the loss plateau, but they were almost insignificant in magnitude. The same trend applies to the accuracy of the trained model.

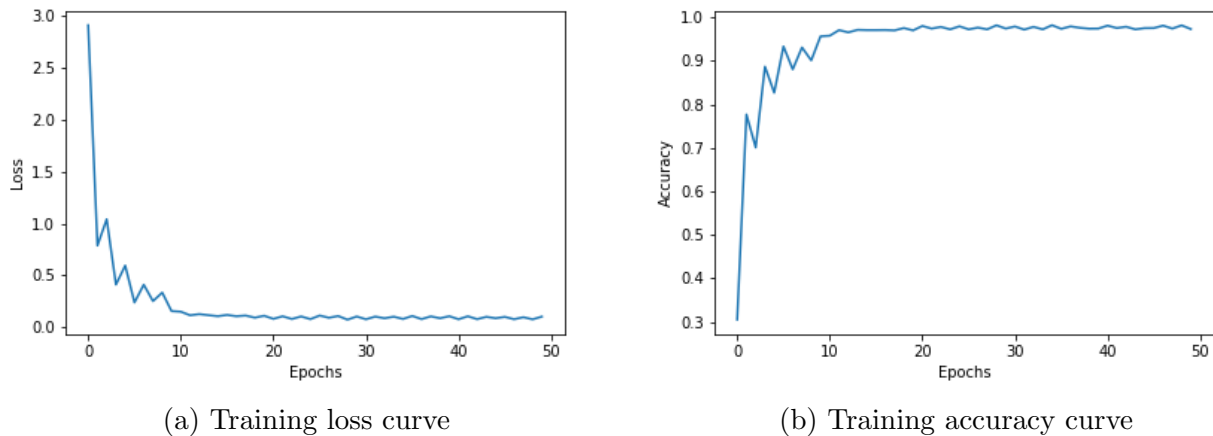


Figure 7.9: GoogLeNet experiment for 100 images/class dataset

The last experiment's for this model was ran using the 100 images/class dataset and the observations are quite clear: there is not a substantial difference in performance between the 100 images/class and the 50 images/class dataset, except the stability of the loss and accuracy plateau. This fact can be seen in Figure 7.9.

7.4 VGG16 architecture

The VGG16 model's weights we used were trained onto the VGGFace2 dataset. This dataset contains over 9000 different classes, with an average of 350 images per class. This dataset's dimensions is one of the reason that this model was used for transfer learning, because it fits the problem at hand perfectly. [?]

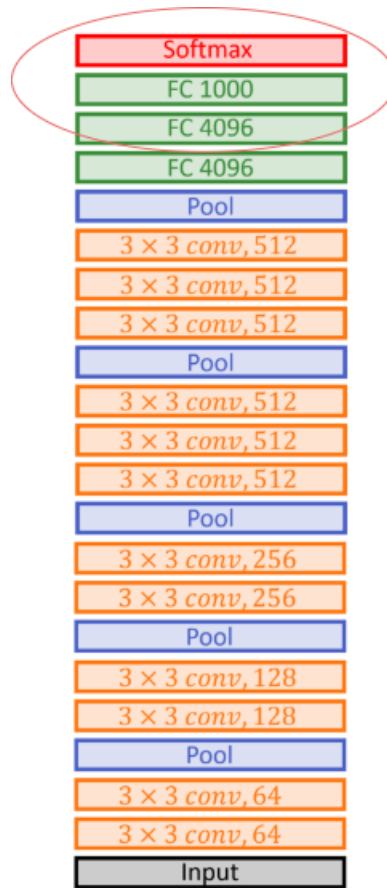
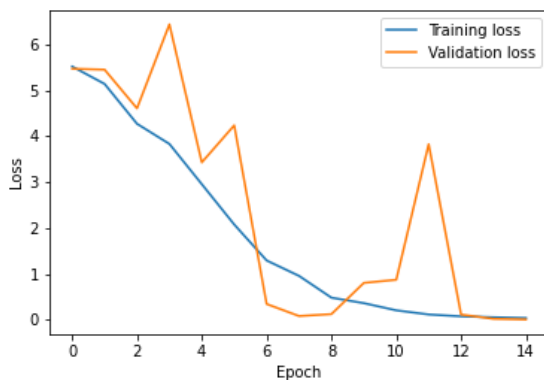
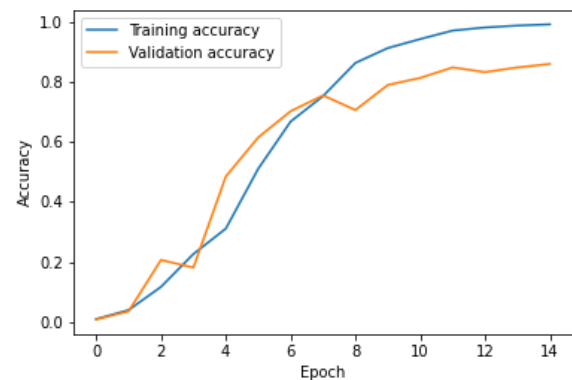


Figure 7.10: VGG16 detailed architecture with emphasis on fine-tuning

Regarding the fine-tuning of this pre-trained model, we took this model, removed all the classification and fully-connected layers on top (3 layers in total), as seen in Figure 7.10, because it didn't fit our classification problem, and added three dense layers on top, one of which being the Softmax layer for classification. These 3 layers were not frozen, such that the network can learn very specific features about our small dataset. Regarding the hyperparameters, we chose a batch size which would accelerate the training process with a high learning rate(0.02). In order for the network not to overfit, we added a learning rate decay which is applied at each epoch.



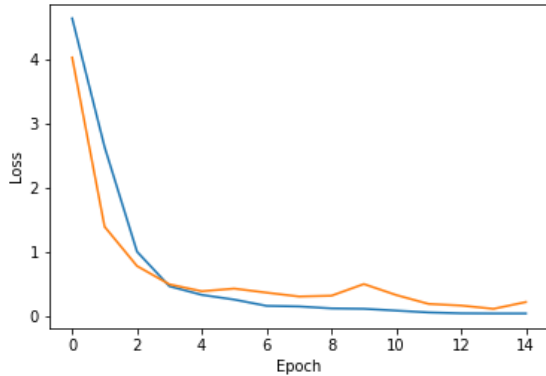
(a) Training and validation loss curves



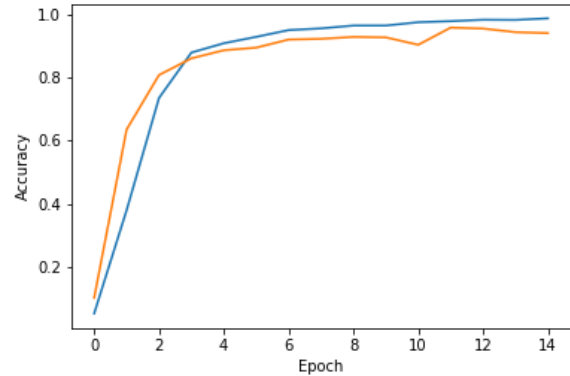
(b) Training and validation accuracy curves

Figure 7.11: VGG16 experiment for 10 images/class dataset

After the analysis of the loss curves in Figure 7.11, it can be noticed the fact that the model is really unstable when the database is small. The reason of this is the model's architecture. VGG16 is a very deep, plain architecture and because of this fact it needs a lot of training data. In the validation step, the global minimum is not reached until the very final epochs. The accuracy graph however looks much stable and does not show signs of overfitting.



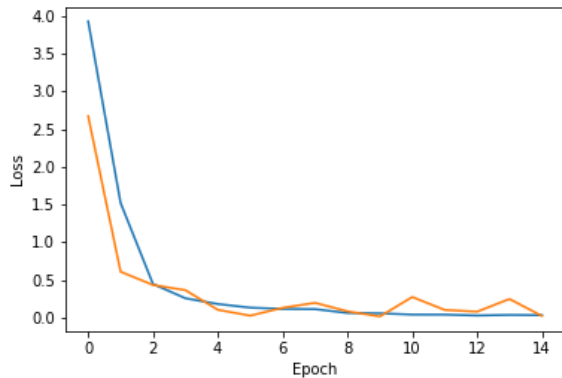
(a) Training and validation loss curves



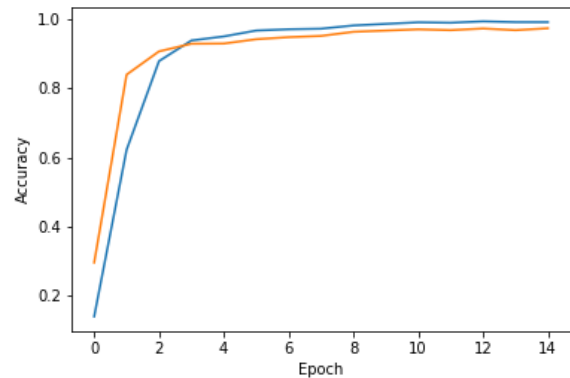
(b) Training and validation accuracy curves

Figure 7.12: VGG16 experiment for 50 images/class dataset

By increasing the dataset size, we observe the fact that the loss and accuracy curves are much better (Figure 7.12), the global minimum being found rapidly. After that, there are little deviations probably due to data diversity or exploding gradients.



(a) Training and validation loss curves



(b) Training and validation accuracy curves

Figure 7.13: VGG16 experiment for 100 images/class dataset

We can observe the fact that the differences between the experiments with 10 and 50 images/class are very low. Both the experiments also show the importance of the abundance of data samples. (Figure 7.13)

7.5 Results and conclusions

All the experiments exposed in this chapter were conducted in order to find out which one of the state-of-the-art face recognition architectures will be used for the task of multimodal person identification. Figure 7.14 shows all the results we extracted by doing the face recognition experiments:

Database	No. of classes	Architecture	Training loss	Training accuracy	Validation loss	Validation accuracy	Test loss	Test accuracy	Precision	Recall	F1 Score
Image10	257	FaceNet	0.076	1.000	0.623	0.955	0.080	0.932	0.936	0.931	0.922
Image10	257	VGG16	0.038	0.992	0.368	0.864	0.175	0.813	0.828	0.813	0.794
Image10	257	GoogLeNet	0.815	0.872	-	-	1.49	0.67	-	-	-
Image50	132	FaceNet	0.068	0.997	0.035	0.990	0.100	0.983	0.980	0.980	0.983
Image50	132	VGG16	0.047	0.989	0.22	0.94	0.01	0.95	0.96	0.95	0.95
Image50	132	GoogLeNet	0.141	0.968	-	-	0.18	0.95	-	-	-
Image100	84	FaceNet	0.047	0.997	0.11	0.991	0.923	0.992	0.993	0.993	0.993
Image100	84	VGG16	0.032	0.991	0.02	0.97	0.08	0.97	0.97	0.97	0.97
Image100	84	GoogLeNet	0.078	0.982	-	-	0.10	0.97	-	-	-

Figure 7.14: Results for the face recognition task

By inspecting Figure 7.14 containing the results for all the architectures on which the experiments were made, we can conclude that FaceNet is the network which we are going to use for the task of multimodal recognition. This architecture had the best accuracy for each of the dataset configurations. Also, by checking the loss curves, we can observe the fact that FaceNet is also the most stable. For all the experiments, this architecture has got 0.932, 0.983 and 0.992 percent accuracy on the test sets.

Database	Network	List of mislabeled classes
Image10	FaceNet	'00293','00620','00855','01040','01272','01320','01724',' '01725','01742','01764','01869','02035','02193','02210',' '02228','02246','02247','02257','02276','02277','02282',' '02308','02316','02325','02329','02339','02372','02381',' '02388','02396','02456','02473','02489','02494','02503'
Image50	FaceNet	'00293','01040','01320','01331','02035','02095','02253', '02305','02326','02371','02374','02414','02473'

Table 7.2: Mislabeled classes from the image experiments

Furthermore, we have used the trained model to predict on the test subset and calculate the confusion matrix in order to reveal which classes had a poor accuracy score. The Table 7.2 shows the classes which did not have 100 percent accuracy score, sorted by the experiment in which they belong. We can notice the fact that the number of mislabeled classes actually drops when the number of samples increases.

Chapter 8

Speaker recognition

8.1 Experimental setup

The audio experiments were done on the same platforms as the image experiments in order to preserve continuity for the multimodal experiments. Depending on the size of the dataset, the experiments were conducted on my personal laptop and on Google Colab. Figure 8.1 shows information about the experimental setup for each speaker recognition experiment, such as the dataset used, the optimizer, the input data shape, the batch size, epochs and steps per epochs.

Database	No. of classes	Pretrained on	Optimizer	Input data shape	Batch size	Epochs	Steps per epoch
Audio_3s_10	257	ResNet34s	SGD(0.01)	257,250	32	10	48
Audio_3s_50	132	ResNet34s	SGD(0.01)	257,250	32	10	123

Figure 8.1: Experimental setup for the speaker recognition tasks

Furthermore, the Table 8.1 shows the split of the derived databases for the training of the models. As an observation, these derived datasets are not the original one. They are the result of the correlation script done in order to have the same classes on both face and audio experiments. The split was done by the following rule: 60 percent training set, 20 percent validation set and 20 percent test set. As we can see, there are two derived datasets, one with 10 audio files per class and one with 50 audio files per class. In the table it can be seen the number of training, validation and test samples for each of these datasets.

Dataset	No. of training examples	No. of validation examples	No. of test samples
Audio10_3s	1542	514	514
Audio50_3s	3960	1320	1320

Table 8.1: Audio derived datasets split number of examples

8.2 VGGVOX

The task at hand was built onto an already trained network, a modified ResNet34 (Thin ResNet) architecture which is detailed in the article “Utterance-level Aggregation For Speaker Recognition in the Wild”. This network can be divided into three separate blocks: frame-level feature extraction, temporal aggregation of frame-level features and the final block, which is actually the optimization of a classification loss. [14] The first element in the pipeline, which is the feature extraction, is done by using a conventional convolutional DNN which takes as an input the spectrograms computed on the input “.wav” files and outputs a fixed-length feature vector.

layer name	res-34	res-50
conv1	$7 \times 7, 64, \text{stride } 2$	$7 \times 7, 64, \text{stride } 2$
pool1	$3 \times 3, \text{max pool, stride } 2$	$3 \times 3, \text{max pool, stride } 2$
conv2_x	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 64 \\ 3 \times 3, 64 \\ 1 \times 1, 256 \end{bmatrix} \times 3$
conv3_x	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1 \times 1, 128 \\ 3 \times 3, 128 \\ 1 \times 1, 512 \end{bmatrix} \times 4$
conv4_x	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1, 256 \\ 3 \times 3, 256 \\ 1 \times 1, 1024 \end{bmatrix} \times 6$
conv5_x	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1 \times 1, 512 \\ 3 \times 3, 512 \\ 1 \times 1, 2048 \end{bmatrix} \times 3$
fc1	$9 \times 1, 512, \text{stride } 1$	$9 \times 1, 2048, \text{stride } 1$
pool_time	$1 \times N, \text{avg pool, stride } 1$	$1 \times N, \text{avg pool, stride } 1$
fc2	$1 \times 1, 5994$	$1 \times 1, 5994$

Figure 8.2: VGGVOX architecture with an emphasis on fine-tuning

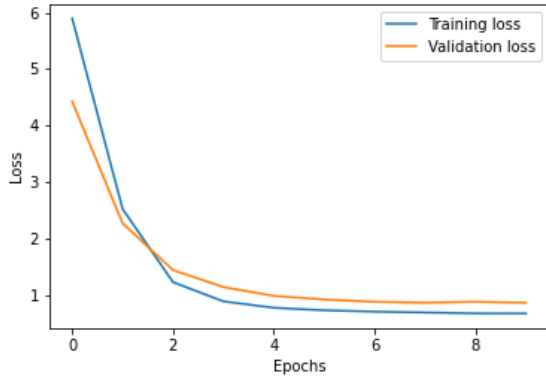
The next step in the pipeline is the temporal aggregation of the frame-level features. This is done by using a concept named NetVlad. The NetVlad layer takes the frame-level descriptors as input and produces a single matrix with the dimensions $K \times D$, where D is the dimension of each cluster and K is the number of clusters chosen. The final output is obtained by l2 normalizing and concatenating. After this, in order to reduce the dimensionality, a fully-connected layer is used to output a dimensionality of 512.

Finally, a Softmax classifier is applied at the output in order to make a prediction based on the feature learned prior.

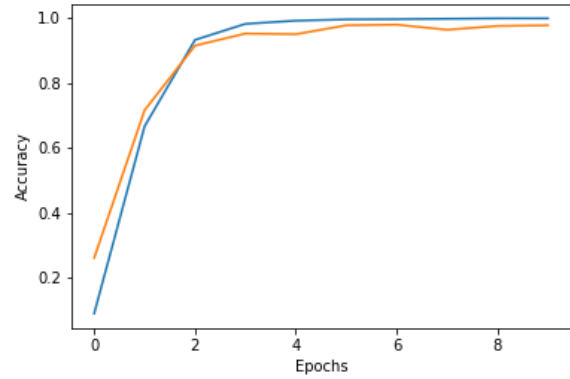
Initially, the model implemented according to the article in the Keras framework was not designed for fine-tuning, only for training and evaluation. Because of this fact, we have taken the code provided with the paper mentioned above [14] and created a new fine-tuning profile designed for our task. The model described above was pre-trained on the VoxCeleb2, which is a dataset containing 2000 hours of speaking from over 7000 speakers.

The input spectrograms are generated as follows: sliding windows fashion, using a hamming window of 25 ms and a 10 ms step. After this, a 512 points FFT is applied and the output is normalized by subtracting the mean and dividing by the standard deviation of all frequency components in a single time step. Regarding the modification of the initial model, we have frozen all of its layers excepting the Batch Normalization layers and added a Softmax classification layer at the top of the model, as seen in Figure 8.2. We have trained this model for 10 epochs, 68 steps/epoch. For the optimizer, we have used the stochastic gradient descent, with an initial learning rate of 0.01, which decrease with a decay of 0.01 every two epochs.

8.3 Results and conclusions



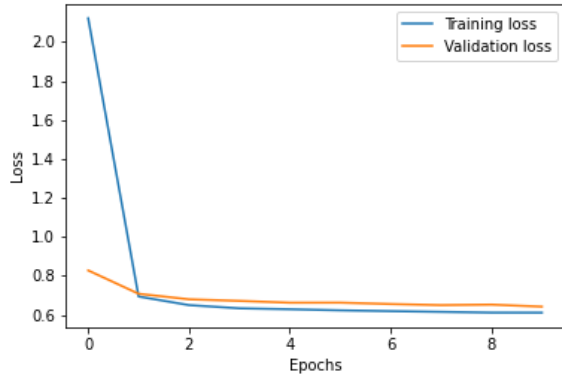
(a) Training and validation loss curves



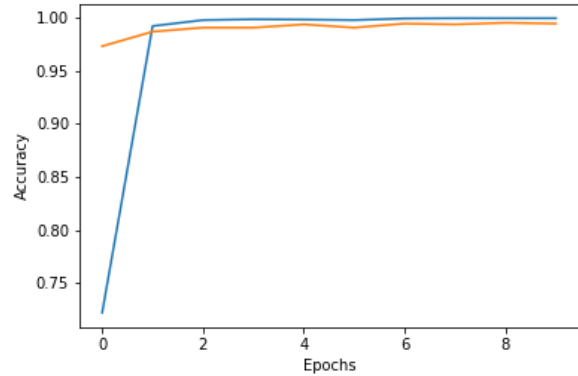
(b) Training and validation accuracy curves

Figure 8.3: VGGVOX experiment for 10 images/class dataset

It can be seen that from the first experiment which is ran on the 10 three seconds audio files/class dataset the results are very good, as seen in Figure 8.3. By looking at the loss and accuracy curves, and also on the test set results, we observe the fact that there is no sign of overfitting or underfitting.



(a) Training and validation loss curves



(b) Training and validation accuracy curves

Figure 8.4: VGGVOX experiment for 50 images/class dataset

By increasing the size of the dataset, we observe the fact that the wanted results, which is a high accuracy on the validation set, is reached more rapidly, only in 3 epochs of training. There are also, as in the previous case, no signs of overfitting due to the high test set accuracy.

Database	No. of classes	Training loss	Training accuracy	Validation loss	Validation accuracy	Test loss	Test accuracy
Audio_3s_10	257	0.6703	0.9987	0.8607	0.9805	0.8162	0.9824
Audio_3s_50	132	0.6117	0.9997	0.6426	0.9947	0.6491	0.9923

Figure 8.5: Results of the speaker recognition experiments

Figure 8.5 concludes the experiments regarding the task of checking if the VGGVOX architecture is suitable for the multimodal experiments. By checking the results of all the experiments we have validated the decision of using the VGGVOX neural network architecture for the multimodal task, together with the FaceNet architecture for image recognition.

Furthermore, we did the same final experiments as in the face recognition experiments. By the help of the confusion matrices, we extracted the classes that did not have 100 percent accuracy scores for further analysis in Chapter 9. These results are shown in Table 8.2.

Database	Network	List of mislabeled classes
Audio_3s_10	VGGVOX	'01724', '02496', '01232', '00326', '02353', '02343', '02294', '02237', '02370', '02201', '00293', '02384', '02359'
Audio_3s_50	VGGVOX	'02328', '02203', '02351', '02393', '02371', '02218', '00849'

Table 8.2: Mislabeled classes for the audio experiments

It is notable the fact that the audio network dealt better with the recognition, as seen by the accuracy scores and also the fact that the number of mislabeled classes is smaller.

Chapter 9

Multimodal person identification

9.1 Previous work

Multimodal tasks have been widely studied in recent years because of the fact that combining individual modalities under the premise that these complementary networks may help each other in our advantage [51]. One overlooked multimodal domain is the recognition of people in videos, which consist of image frames and audio signals [52]. Furthermore, video was not the only domain in which the interest in multimodal tasks is growing. Human activity recognition systems using different types of sensors such as motion systems, fingerprint scanners and microphones.

Regarding multimodal experiments, relatively high accuracy score have been obtained by fusing computer vision techniques and audio recognition and identification techniques. Researchers have proposed a number of different techniques for neural networks' fusion, such as early (data) and late (classification) fusion.

Multimodal video processing has been tackled differently by the two scientific communities of image processing and speech processing. As instance, the speech processing community focused on topic segmentation or speaker role recognition [53], using only audio and transcriptions. In an almost same way, the image processing community focused on exploring scene analysis and action recognition [54] based only on images. In addition to these studies, there have been attempts in using video segmentation [55]. However, these approaches limit the cooperation between the networks when talking about the fusion. [56]

Although image recognition is a vast and very popular domain, multimodal recognition using audio and image recognition techniques does not have such a massive community

9.2 Experimental setup

The multimodal experiments were ran only on Google Colab, due to a higher number of data samples, thus a need of greater computational power. In Figure 9.5 we can also see the experimental setup for both the experiments consisting of the batch size, optimizer, number of training epochs and the number of steps per epoch.

9.3 Multimodal architecture

A multimodal system can operate in different modes: serial mode, parallel mode and hierarchical mode. In serial mode, it can be said that one of the networks is "helping" the other one by firstly predicting the outcome of one modality and the providing the second network with the result such that it narrows the possibilities of classification. Secondly, the parallel mode classifies a pair of images and audio and does that in the same time. This also ensures a more rapid computation. Hierarchical mode is used for systems with a great number of classifiers, so it does not apply to our problem. [51]

For our project, it is relevant to use a parallel multimodal system because of the fact that we have two neural networks which are trained on the datasets and can be used in parallel for feature extraction.

Fusion techniques can be divided into three categories: early integration, intermediate integration and late integration. The input data of the multimodal system in our case is not dependentant (time-correlated) such that we can apply an intermediate or a late integration which could work well with the provided data. By using the two pre-trained models, we can extract a feature vector for each pair of data, meaning a 3 seconds audio and an image, and then concatenate them and train a Softmax classifier on the concatenated feature vectors. Furthermore, in order to have a compatible feature vector at the output of the concatenation layer, the input data for both neural networks is normalized between 0 and 1. [57]

Figure 9.1 shows how the fusion process works. There are two branches in parallel, one for image processing and one for audio processing. Each one extracts features that are then concatenated and introduced into a classifier.

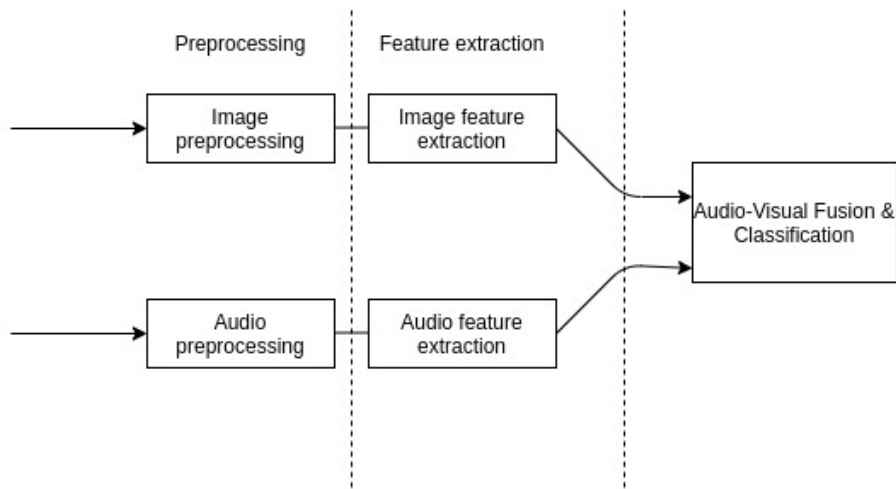


Figure 9.1: Fusion process

In Figure 9.1, there can be observed three types of blocks: preprocessing, extraction and classification. The preprocessing blocks are referring to the actions taken before training the networks, such as rescaling, mel-frequency spectrogram extraction etc. The feature extraction blocks refer to the actual pre-trained neural networks which have at the output two feature vectors. The last block, the classification block is placed after the concatenation of the feature vectors.

9.4 Experiments

For the multimodal experiments, the architecture from Figure 9.2 has been used:

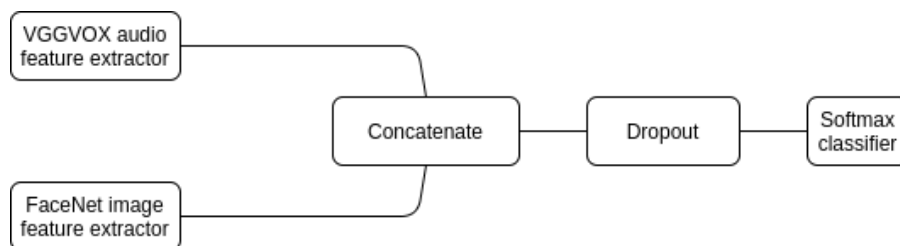
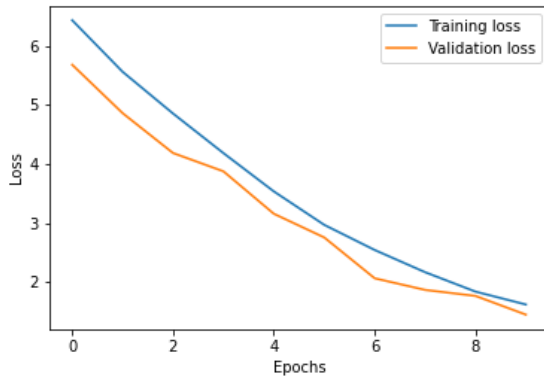


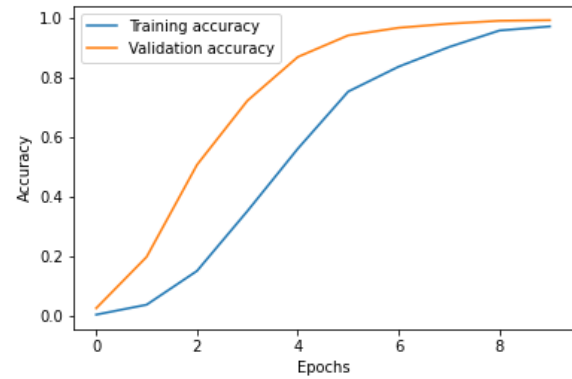
Figure 9.2: multimodal neural network architecture

The experiments in this section were conducted on the Google Colab platform, due to a high demand of memory. This flaw of my own generator was discovered by trying to run on my personal laptop, because the Python kernel would shut down in the middle of the training process due to lack of memory.

The experiments were ran on the two multimodal datasets, which are a combination of the 10 image and 10 three seconds audio files per class, and the 50 images and 50 three seconds audio files per class. Because of the fact that Keras does not provide us with a data generator for the multimodal dataset, we had to build one from scratch. This data generator takes as input pairs of absolute paths of audio and image data and asynchronously reads them and inputs them into the network in order to train it.



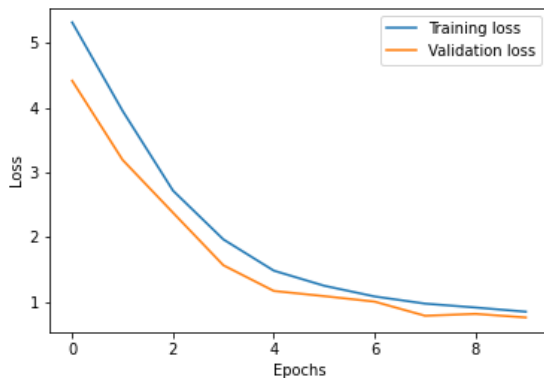
(a) Training and validation loss curves



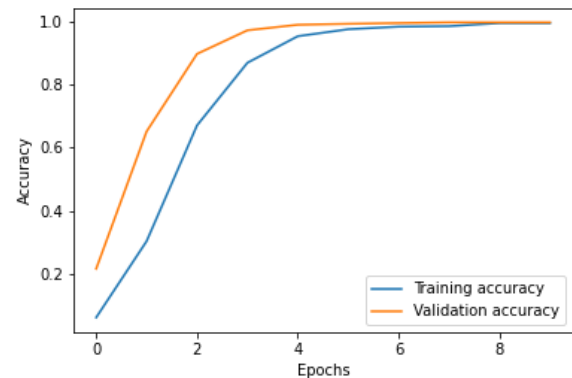
(b) Training and validation accuracy curves

Figure 9.3: multimodal experiment for 10 images/class dataset

By analysing the training loss in the first multimodal experiment, depicted in Figure 9.3, we can observe the fact that the loss is rapidly decreasing in the first two epochs, from approximately 3.5 to approximately 0.8 in just one epoch. This can be explained by the fact that we are training a single layer, the classification layer (Softmax), the rest of the networks being pre-trained on huge datasets. Also, it can be seen that the training accuracy has the same trend, from 0.1 to 0.9 in just one epoch.



(a) Training and validation loss curves



(b) Training and validation accuracy curves

Figure 9.4: Multimodal experiment for 50 images/class dataset

The difference between the two multimodal experiments is not great, by looking at the curves. It can be only seen the fact that in the second experiment, where the number of

samples was almost doubled for each class that the plateau of the validation loss curve is slightly more stable than of the first experiment.

9.5 Conclusions

Figure 9.5 shows the results we have acquired by running the multimodal experiments, which are described in the chapter above. It can be observed that the results are very good, the neural network trained on the 50 samples/class almost reaching 100 percent accuracy.

Database	Number classes	Batch Size	Optimizer	Learning rate	Train Loss	Train Accuracy	Validation loss	Validation accuracy	Test loss	Test accuracy
ImageDatabase10	257	32	SGD	0.01	1.6167	0.9714	1.445	0.9902	1.201	0.9982
AudioDatabase_3s_10										
ImageDatabase50	132	32	SGD	0.01	0.6741	0.999	0.6453	0.997	0.7	0.99924
AudioDatabase_3s_50										

Figure 9.5: multimodal experimental results

There were some concerns regarding the almost perfect accuracy of the multimodal model. These doubts were cleared by looking at the previous tasks and finding motives for which the accuracy is so high, which are:

- Balanced and correlated datasets.
- The pre-trained models were trained on massive datasets.
- The accuracy of the standalone audio and image recognition models is high 0.95 percent.
- The only trainable part in the multimodal model is the end positioned Softmax classifier.

Finally, in order to validate our experimental results and emphasise the improvement of creating a multimodal neural network, we analysed the confusion matrices for each experiment and extracted the classes which did not have a perfect accuracy.[58]

A confusion matrix is a table that is often used to describe the performance of a classification model on a set of test data, for which the true values are known. It consists of the values of accuracy for each class in our classification problem. In our project, this concept helps us discover if the monomodal networks actually help each other in the multimodal classification.

The Table 9.1 shows the mislabeled classes found by computing the confusion matrix for the multimodal experiments. It can be observed the fact that on the 50 samples/class experiment, the test accuracy score is approximately 100 percent, so no classes are mislabeled. The next paragraph will analyse this table in comparison to Table 8.2 and Table 7.2.

Database	Network	List of mislabeled classes
Audio_3s_10 Image10	VGGVOX+ FaceNet	'01232', '02381', '01724', '02359', '01331'
Audio_3s_50 Image50	VGGVOX + FaceNet	

Table 9.1: Mislabeled classes from the multimodal experiments

By analysing these three tables in parallel, the contribution made by each network can be observed. This quantity is measured by how many mislabeled classes from the monomodal networks were correctly labeled by the multimodal networks. We apply the union operator on

the two lists for each experiment, and then compare them to the final experiments' lists. It can be seen that for the 50 samples/class experiments, the monomodal networks completed each other and so the test accuracy score is approximately 100 percent. On the other hand, on the 10 samples/class, there were still some classes that didn't have a perfect accuracy score. Out of the 5 mislabeled classes in the final experiment, only one was not predicted correctly by all the networks, the class with the identification number "01724". This class had the image recognition accuracy of 50 percent, meaning that one image was incorrectly classified, and also the same accuracy score on speaker recognition.

Figure 9.6 shows the waveforms of the two test audio files of the class "01724". By analysing them, we saw the fact that the first one's amplitude is much lower than the second one, with the speech windows being much smaller than in the second waveform, which could be a factor in the mislabeling of the audio file.

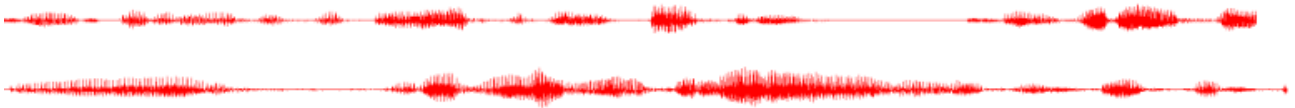


Figure 9.6: Test waveforms of the "01724" class

Also, Figure 9.7 shows the two test images corresponding to the class "01724".



Figure 9.7: Test images of the "01724" class

It can be seen that the pictures are quite different, because of the fact that they were took in different time moments. Furthermore, in the right picture, the person wears glasses and has a different posture than in the first picture.

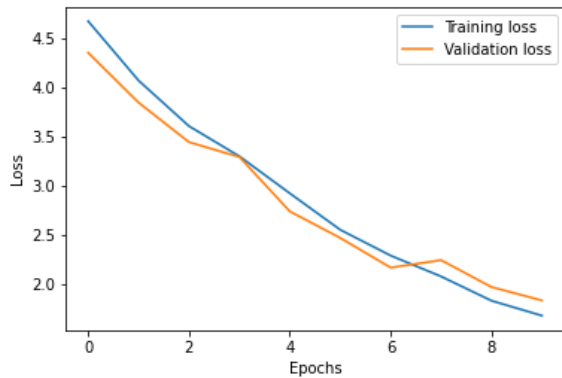
9.6 Multimodal network validation

The next task we accomplished was to validate the multimodal model, meaning that we wanted to see if this architecture would have a high accuracy on another public audio-video dataset. The dataset on which the experiments were ran was "VidTimit". This database [59] is an audio-visual database comprised of audio-visual recordings of 43 persons reciting different sentences of the TIMIT corpus. This database was developed in 3 sessions with a delay of 7 days between sessions 1 and 2 and 6 days between session 2 and 3. These delays were introduced in order to bring some real-life aspects to the dataset [60]. Figure 9.8 shows examples of images from the VidTIMIT database.

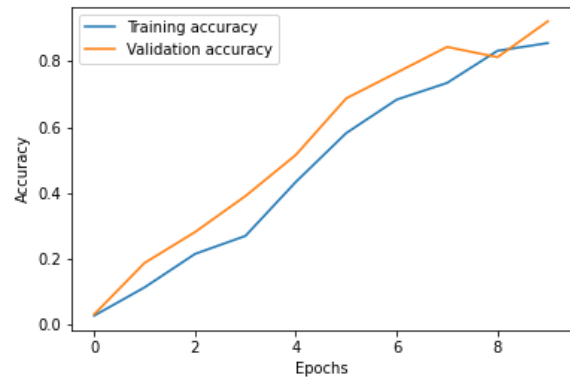


Figure 9.8: Examples of images from the VidTIMIT database

We have trained our multimodal classifier on this dataset and extracted the results. As an observation, no fine-tuning was done prior to this training. The hyperparameters used were the same as the ones from the experiments ran on our dataset and the number of steps per epoch was found by dividing the number of training examples to the size of a batch.



(a) Training and validation loss curves



(b) Training and validation accuracy curves

Figure 9.9: VidTIMIT experiment accuracy and loss curves

After training and evaluation, we have achieved a validation accuracy of 92.19 percent and a test accuracy of 76.74 percent. By inspecting the training and validation loss and accuracy curves in Figure 9.9, we can observe the fact that the model is not overfitting and that by further fine-tuning this model in order to fit this dataset perfectly we can achieve a much greater test accuracy score.

Chapter 10

Conclusions and further development

10.1 General conclusions

The main objective of our paper was to create a multimodal system that can recognize people based on audio and video using Deep Neural Networks. Also, a second objective was to acknowledge the fact that by combining two neural networks which have some accuracy scores, the multimodal architecture would perform better than any of those two neural networks individually. For these objectives to be achieved, we have built the two databases that are used to train these neural networks by parsing and manipulating public data from the internet.

Although the results were almost perfect, there are certain limitations of our project due to constrained audio and video, all the videos being made with almost the same setup in the same position. Also, a few other constraints are the resolution of the video and audio, bad labeling of the data and audio noise and overlapping of certain voices.

All in all, this thesis presented a solution for the creation and development of a multimodal recognition system using an in-house database. The experiments and the results reflected the fact that such a system can be created by taking the state-of-the-art systems in both domains, audio and image recognition, and through the concept of transfer learning to make this project have very good results.

10.2 Personal contributions

By developing this project, the main objective was achieved with the following personal contributions:

- Developing an algorithm for in-video frame extraction with constraints.
- Developing an algorithm for face database creation and validation using classical Python scripting and ML approaches.
- Creating an algorithm for audio database creation and validation using VAD and Python classical scripting.
- Fine-tuning three state-of-the-art networks for face recognition and choosing the best option.
- Fine-tuning one state-of-the-art network for speaker recognition.
- Creating a multimodal architecture, performing training, and evaluating the performance metrics of this network.
- Results analysis and conclusions.

10.3 Further development

In our times, the artificial intelligence subject is a very subject and is rapidly increasing in popularity. Because of this fact, the increase in technology which is happening day by day, a project like ours needs to be kept at the state-of-the-art level.

In my opinion, the next step in the development of this project is to adapt the neural networks and create scripts such that a live demo can be obtained. By modifying the networks to take into considerations batches of images and audio according to the order in the live video, we can build this system to recognize distinct persons in a video.

Bibliography

- [1] Sri Harsha Dumpala. Spectrogram of a speech signal. https://www.researchgate.net/figure/Spectrogram-of-a-speech-signal-with-breath-sound-marked-as-Breath-whose-bounds-are_fig1319081627.
- [2] Hierarchical clustering. <http://www.sthda.com/english/articles/28-hierarchical-clustering->.
- [3] Akshat Maheshwari. Face detection. <https://iq.opengenus.org/face-detection-using-viola-jones-algorithm/>.
- [4] Nadia Berchane. Artificial intelligence, machine learning, and deep learning: Same context, different concepts. <https://master-iesc-angers.com/artificial-intelligence-machine-learning-and-deep-learning-same-context-different-concepts/>.
- [5] Cs231n: Convolutional neural networks for visual recognition. <http://cs231n.stanford.edu/>.
- [6] Understand local receptive fields in convolutional neural networks. <https://towardsdatascience.com/understand-local-receptive-fields-in-convolutional-neural-networks-f26d700be16c>.
- [7] Dropout in (deep) machine learning. <https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-machine-learning-74334da4bfc5>.
- [8] Stephen Balaban. Deep learning and face recognition: the state of the art. 2019.
- [9] Vgg16 - convolutional neural network for classification and detection. <https://neurohive.io/en/popular-networks/vgg16/>.
- [10] Facenet inception v3. <http://www.kaimalo.com/img/336702e6f69647075636e696.html>.
- [11] Schematic diagram of triplet loss. https://www.researchgate.net/figure/Schematic-diagram-of-Triplet-Loss-The-input-of-TL-is-a-triplet-containing-one-pair-of_fig334780722.
- [12] Cnn architectures. <https://mc.ai/cnn-architectures-lenet-alexnet-vgg-googlenet-and-resnet/2>.
- [13] Joon Son Chung, Arsha Nagrani, and Andrew Zisserman. Voxceleb2: Deep speaker recognition, 2018.
- [14] Jesús Villalba, Nanxin Chen, David Snyder, Daniel Garcia-Romero, Alan McCree, Gregory Sell, Jonas Borgstrom, Leibny Paola García-Perera, Fred Richardson, Réda Dehak, Pedro A. Torres-Carrasquillo, and Najim Dehak. State-of-the-art speaker recognition with neural network embeddings in nist sre18 and speakers in the wild evaluations. *Computer Speech Language*, 60:101026, 2020.
- [15] Pacific Northwest Seismic Network. What is a spectrogram? <https://pnsn.org/spectrograms/what-is-a-spectrogram>.

- [16] Björn Lindblom, Johan Sundberg, Peter Branderud, Hassan Djamshidpey, and Svante Granqvist. The Gunnar Fant Legacy in the Study of Vocal Acoustics. In Société Française d'Acoustique SFA, editor, *10ème Congrès Français d'Acoustique*, pages –, Lyon, France, April 2010.
- [17] Thomas Drugman, Yannis Stylianou, Yusuke Kida, and Masami Akamine. Voice activity detection: Merging source and filter-based information. 2019.
- [18] Alboukadel Kassambara. Agglomerative hierarchical clustering. <https://www.datanovia.com/en/lessons/agglomerative-hierarchical-clustering/>.
- [19] Jupyter notebook. <https://jupyter.org/>.
- [20] Alexander Mordvintsev. Python face detection. https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_objdetect/py_face_detection/py_face_detection.html.
- [21] Mu Li Aston Zhang, Zachary C. Lipton and Alexander J. Smola. *Dive into Deep Learning*. 8th edition, 2020.
- [22] C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, June 2015.
- [23] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.
- [24] Kamil Krzyk. Cost functions. <https://towardsdatascience.com/coding-deep-learning-for-beginners-linear-regression-part-2-cost-function-49545303d29f>.
- [25] Sebastian Ruder. An overview of gradient descent optimization algorithms, 2016.
- [26] Guodong Zhang, Chaoqi Wang, Bowen Xu, and Roger Grosse. Three mechanisms of weight decay regularization, 2018.
- [27] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. [U+FFFD] dropout: a simple way to prevent neural networks from overfitting, [U+FFFD] The Journal of Machine Learning Research. 15(1):1929 [U+FFFD] 1958, 2014.
- [28] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning, 2019.
- [29] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.
- [30] Q. Cao, L. Shen, W. Xie, O. M. Parkhi, and A. Zisserman. Vggface2: A dataset for recognising faces across pose and age. In *International Conference on Automatic Face and Gesture Recognition*, 2018.
- [31] Yandong Guo, Lei Zhang, Yuxiao Hu, Xiaodong He, and Jianfeng Gao. Ms-celeb-1m: A dataset and benchmark for large-scale face recognition, 2016.
- [32] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2014.

- [33] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. 2015.
- [34] C. Zhang, K. Koishida, and J. H. L. Hansen. Text-independent speaker verification based on triplet convolutional neural network embeddings. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 26(9):1633–1644, 2018.
- [35] Weidi Xie, Arsha Nagrani, Joon Son Chung, and Andrew Zisserman. Utterance-level aggregation for speaker recognition in the wild, 2019.
- [36] Python programming language. <https://www.python.org/>.
- [37] BeautifulSoup java library. <https://www.crummy.com/software/BeautifulSoup/bs4/doc/>.
- [38] Numpy python library. <https://numpy.org/doc/>.
- [39] Pandas python library. <https://pandas.pydata.org/docs/>.
- [40] Tensorflow framework. https://www.tensorflow.org/api_docs.
- [41] Keras framework. <https://keras.io/documentation/>.
- [42] Pytorch framework. <https://pytorch.org/docs/stable/index.html>.
- [43] Scikit-learn library. <https://scikit-learn.org/stable/>.
- [44] Ffmpeg library. <https://ffmpeg.org/ffmpeg.html>.
- [45] Opencv library. <https://opencv-python-tutroals.readthedocs.io/en/latest/>.
- [46] Librosa library. <https://librosa.github.io/librosa/>.
- [47] Jayshree Ghorpade, Jitendra Parande, Madhura Kulkarni, and Amit Bawaskar. Gpgpu processing in cuda architecture. 2012.
- [48] Cdep website. www.cdep.ro.
- [49] Rafael Padilla, Cicero Filho, and Marly Costa. Evaluation of haar cascade classifiers for face detection. 04 2012.
- [50] Cvlib library. <https://www.cvlib.net/>.
- [51] Marcos Faundez-Zanuy, Julian Fierrez, Javier Ortega-Garcia, and Joaquin Gonzalez-Rodriguez. Multimodal biometric databases: An overview. *Aerospace and Electronic Systems Magazine, IEEE*, 21:29 – 37, 09 2006.
- [52] Youssef Mroueh, E. Marcheret, and Vaibhava Goel. Deep multimodal learning for audio-visual speech recognition. 01 2015.
- [53] Camille Guinaudeau, Guillaume Gravier, and Pascale Sébillot. Enhancing lexical cohesion measure with confidence measures, semantic relations and language model interpolation for multimedia spoken content topic segmentation. *Computer Speech Language*, 26(2):90 – 104, 2012.
- [54] Adrien Gaidon, Zaid Harchaoui, and Cordelia Schmid. Temporal Localization of Actions with Actoms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(11):2782–2795, November 2013.

- [55] Émilie Dumont and Georges Quénot. Automatic story segmentation for tv news video using multiple modalities. *International Journal of Digital Multimedia Broadcasting*, 2012, 04 2012.
- [56] Frédéric Béchet, Meriem Bendris, Delphine Charlet, Géraldine Damnati, Benoit Favre, Mickael Rouvier, Rémi Auguste, Benjamin Bigot, Richard Dufour, Corinne Fredouille, Georges Linarès, Jean Martinet, Grégory Senay, and Pierre Tirilly. Multimodal understanding for person recognition in video broadcasts. 09 2014.
- [57] Dana Lahat, Tülay Adalı, and Christian Jutten. Multimodal Data Fusion: An Overview of Methods, Challenges and Prospects. *Proceedings of the IEEE*, 103(9):1449–1477, August 2015.
- [58] Cvlib library. <https://www.geeksforgeeks.org/confusion-matrix-machine-learning>.
- [59] Conrad Sanderson. *Biometric Person Recognition: Face, Speech and Fusion*. 2008.
- [60] DHAVAL SHAH, KYU J. HAN, and SHRIKANTH S. NARAYANAN. Robust multimodal person recognition using low-complexity audio-visual feature fusion approaches. *International Journal of Semantic Computing*, 04(02):155–179, 2010.

Annex A

Similarity computation

1

```

1  import csv
2  import itertools
3  import os
4  from os import listdir, walk
5  from os.path import isfile, join
6  import sys
7  import numpy as np
8  import pandas as pd
9  from keras.applications.resnet50 import (ResNet50, decode_predictions, preprocess_input)
10 from keras.models import Model
11 from keras.preprocessing import image
12 from keras_vggface.vggface import VGGFace
13 from scipy import spatial
14
15 from helper_functions import getAllFilesInDirectory, getNumberOfFiles, predict,
16 findDifference
17
18 def compute_similarity(directory, model):
19     feature_vectors = dict()
20     similarity_array = []
21     temp = []
22     # vgg_features = VGGFace(include_top=False, input_shape=(224, 224, 3), pooling='avg')
23
24     for img_path in getAllFilesInDirectory(directory):
25         try:
26             feature_vectors[img_path] = predict(img_path, model)[0]
27         except Exception:
28             print("Error! ", sys.exc_info()[0])
29
30     keys = [k for k,v in feature_vectors.items()]
31     possible_combinations = list(itertools.product(keys, repeat=2))
32
33     for i in range(0, len(possible_combinations), getNumberOfFiles(directory)):
34         for k,v in possible_combinations[i:i + getNumberOfFiles(directory)]:
35             try:
36                 temp.append(findDifference(feature_vectors[k], feature_vectors[v]))
37             except Exception:
38                 print("Error! ", sys.exc_info()[0])
39
40         similarity_array.append(temp)
41         temp=[]
42
43     return similarity_array
44
45

```

¹Full source code can be found on: <https://git.speed.pub.ro/diploma/multimodal-person-identification>

Annex B

Image dataset validation algorithm

1

```

1  import pandas as pd
2  import os
3  import csv
4  from sklearn.cluster import AgglomerativeClustering
5  from remove_folder import remove_folder
6  from remove_image import remove_image
7  from config import *
8  from clustering import clustering
9  from create_csv import create_csv
10 from all_similarities import all_similarities
11 from cluster_difference import cluster_difference
12 from keras_vggface.vggface import VGGFace
13 import logging
14
15 # Initialize model for feature extraction
16 vgg_features = VGGFace(include_top=False, input_shape=(224, 224, 3), pooling='avg')
17
18 # Initialize logging
19 logging.basicConfig(filename='validation.log', filemode='w', format='%(name)s - %(
levelname)s - %(message)s', level = logging.DEBUG)
20 logging.warning('This will get logged to a file')
21
22 #Delete every folder with similarity score less than threshold4
23 data = all_similarities(all_sim_path)
24 sims = data.values
25 for array in sims:
26     if array[0] < threshold4:
27         remove_folder("{0}{1}{2}".format(image_folder, os.path.sep, array[1].split(".")
[0]))
28         os.remove("{0}{1}{2}".format(csv_folder, os.path.sep, array[1]))
29
30 # Check every folder for images with a similarity score less than threshold1
31 for fl in os.listdir(csv_folder):
32     df = pd.read_csv("{0}{1}{2}".format(csv_folder, os.path.sep, fl))
33     logging.debug("CSV with the name {0} was read".format(fl))
34     data = df.mean()
35     logging.debug("Deleting images with similarity less than 0.6")
36     for index, value in data.items():
37         if value <= threshold1:
38             temp = fl.split(".")
39             index = index.split("/")
40             remove_image("{0}{1}{2}{3}{4}".format(image_folder, os.path.sep, str(temp[0]),
os.path.sep, str(index[-1])))
41             logging.debug("Deleted images succesfully")
42
43 # Remove all the csv files and replace them with new ones because there were changes
44 logging.info("Recreating CSV files")
45 for csv_name in os.listdir(csv_folder):
46     logging.debug("Removing and updating CSV with the name {0}".format(csv_name))
47     os.remove("{0}{1}{2}".format(csv_folder, os.path.sep, csv_name))
48     create_csv("{0}{1}{2}".format(image_folder, os.path.sep, csv_name.split(".")[0]),
vgg_features)
49 logging.debug("Recreating CSV files succesfull")
50
51 # Load data in a dataframe and initialize clustering method
52 var = False
53 data = all_similarities(all_sim_path)

```

¹Full source code can be found on: <https://git.speed.pub.ro/diploma/multimodal-person-identification>

```

54     sims = data.values
55     cluster = AgglomerativeClustering(n_clusters = 2, affinity="euclidean", linkage="ward")
56     low_sim = []
57
58     # Apply clustering on folders with similarity less than threshold2 and delete the wrong
59     # cluster
60     for array in sims:
61         count = 0
62         logging.debug("Applying clustering loop for the folder {}".format(array[1]))
63         if array[0] < threshold2:
64             logging.debug("Value for average similarity: {}".format(array[0]))
65             while(cluster_difference("{}{}{}{}".format(csv_folder, os.path.sep, array[1]),
66                 cluster) > 0.1):
67                 count += 1
68                 difference = clustering("{}{}{}{}".format(csv_folder, os.path.sep, array[1]),
69                     cluster)
70                 logging.debug("Applying clustering for the {} time".format(count))
71                 logging.debug("Difference between clusters: {}".format(str(difference)))
72                 os.remove("{}{}{}{}".format(csv_folder, os.path.sep, array[1]))
73                 create_csv("{}{}{}{}".format(image_folder, os.path.sep, array[1].split(".")
74                     [0]), vgg_features)
75                 low_sim.append(array[1])
76
77     # Check every folder for images with a similarity score less than threshold3
78     for fl in os.listdir(csv_folder):
79         df = pd.read_csv("{}{}{}{}".format(csv_folder, os.path.sep, fl))
80         logging.debug("CSV with the name {} was read".format(fl))
81         data = df.mean()
82         logging.debug("Deleting images with similarity less than 0.6")
83         for index, value in data.items():
84             if value <= threshold3:
85                 temp = fl.split(".")
86                 index = index.split("/")
87                 remove_image("{}{}{}{}{}{}".format(image_folder, os.path.sep, str(temp[0]),
88                     os.path.sep, str(index[-1])))
89                 logging.info("Deleted images succesfully")
90
91     logging.info("Recreating CSV files")
92     for csv_name in os.listdir(csv_folder):
93         logging.debug("Removing and updating CSV with the name {}".format(csv_name))
94         os.remove("{}{}{}{}".format(csv_folder, os.path.sep, csv_name))
95         create_csv("{}{}{}{}".format(image_folder, os.path.sep, csv_name.split(".")[0]),
96             vgg_features)
97         logging.debug("Recreating CSV files succesfull")
98
99

```

Annex C

Multimodal data generator

1

```

1  from tensorflow.keras.preprocessing.image import load_img
2  from keras.preprocessing.image import img_to_array
3  import keras
4  import numpy as np
5  import utils as ut
6
7  class MultimodalDataGenerator(keras.utils.Sequence):
8      'Generates data for Keras'
9      def __init__(self, list_IDs, labels, dim, mp_pooler, augmentation=True, batch_size=32,
10         nfft=512, spec_len=250,
11         win_length=400, sampling_rate=16000, hop_length=160, n_classes=5994,
12         shuffle=True, normalize=True):
13         'Initialization'
14         self.dim = dim
15         self.nfft = nfft
16         self.sr = sampling_rate
17         self.spec_len = spec_len
18         self.normalize = normalize
19         self.mp_pooler = mp_pooler
20         self.win_length = win_length
21         self.hop_length = hop_length
22
23         self.labels = labels
24         self.shuffle = shuffle
25         self.list_IDs = list_IDs
26         self.n_classes = n_classes
27         self.batch_size = batch_size
28         self.augmentation = augmentation
29         self.on_epoch_end()
30
31     def __len__(self):
32         'Denotes the number of batches per epoch'
33         return int(np.floor(len(self.list_IDs) / self.batch_size))
34
35     def __getitem__(self, index):
36         'Generate one batch of data'
37         # Generate indexes of the batch
38         indexes = self.indexes[index*self.batch_size:(index+1)*self.batch_size]
39
40         # Find list of IDs
41         list_IDs_temp = [self.list_IDs[k] for k in indexes]
42
43         # Generate data
44         X, y = self.__data_generation_mp(list_IDs_temp, indexes)
45
46         return X, y
47
48     def on_epoch_end(self):
49         'Updates indexes after each epoch'
50         self.indexes = np.arange(len(self.list_IDs))
51         if self.shuffle:
52             np.random.shuffle(self.indexes)
53
54     def __data_generation_mp(self, list_IDs_temp, indexes):

```

¹Full source code can be found on: <https://git.speed.pub.ro/diploma/multimodal-person-identification>

```

56         X1 = [self.mp_pooler.apply_async(ut.load_data,
57                                     args=(ID[1], self.win_length, self.sr, self.
hop_length,
58                                     self.nfft, self.spec_len)) for ID in list_IDs_temp
]
59         X2 = [self.mp_pooler.apply_async(load_img, args=(ID[0], False, 'rgb', (160,160)))
for ID in list_IDs_temp]
60         X2 = np.array([img_to_array(p.get()) for p in X2])
61         X2 = np.divide(X2, 255)
62         X1 = np.expand_dims(np.array([p.get() for p in X1]), -1)
63         y = self.labels[indexes]
64         return [X2, X1], keras.utils.to_categorical(y, num_classes=self.n_classes)
65
66
67     def __data_generation(self, list_IDs_temp, indexes):
68         'Generates data containing batch_size samples' # X : (n_samples, *dim, n_channels)
69         # Initialization
70         X = np.empty((self.batch_size,) + self.dim)
71         y = np.empty((self.batch_size), dtype=int)
72
73         # Generate data
74         for i, ID in enumerate(list_IDs_temp):
75             # Store sample
76             X1[i, :, :, 0] = ut.load_data(ID, win_length=self.win_length, sr=self.sr,
hop_length=self.hop_length,
77                                     n_fft=self.nfft, spec_len=self.spec_len)
78             X2[i, :, :, 0] = load_img(ID[0], target_size = (160,160))
79             # Store class
80             y[i] = self.labels[indexes[i]]
81
82         return [X2, X1], keras.utils.to_categorical(y, num_classes=self.n_classes)
83

```