Voice Recognition System Using Machine Learning and Deep Learning

Final Senior Project Report Submitted to
The Department of Network Engineering and Security
Faculty of Computer and Information Technology
Jordan University of Science and Technology

In Partial Fulfillment of the Requirements for the Degree of
Bachelors of Science in
Network Engineering and Security

Authors:
Hadeel Sami Rasem Abu Arja, 126237
Baskal Yousef Saleem Maayah, 126230
Masa Ma'moun Na'eem Al Taweel, 124556

Adviser:
Dr. Baha' Alsaify

January 21, 2022
Fall Semester 2021/2022

DEDICATIONS

Praise be to ***Allah*** *the Almighty,*
who gave us strength and enabled us to realize that dream.

"To my beloved parents,
Who made me the strong person I am today,
I dedicate this work and every little success I accomplish in life to you.
Thank you for teaching me how to believe in God, in myself and in my dreams.
To my respected professors,
who draw the road for me and have been true role models to look up to.
And to my friend Dina, who's been a witness of all my fears, tears, and break moments.
Thank you for being there for me in all my downs."

- Baskal Maayah

"To my dear parents,
I wholeheartedly dedicate my success to you.
You have been the source of inspiration and gave me the strength when I thought of giving up, and you have provided me with your continuous moral, emotional, and financial support.
With genuine and warm regard,
I also dedicate this work to my distinguished professors,
who have been the basis of my arrival at this stage, and the strongest upholders with their words, encouragements, compliments, and criticism.
I wouldn't be able to excel if it was not for you.
And lastly, to my sisters, brothers, friends, and classmates.
There are no words that can express my gratitude for having such amazing people like you.
Thanks for your motivating words that push me up."

- Hadeel Abu Arja

"First and foremost, I'd like to express my sincere gratitude to my adviser, Dr. Baha' Al Saify, for his wise counsel, inspiring words, and continuous encouragement during this project.
I am grateful to my parents for giving birth to me, for supporting me throughout my life, and for motivating me to pursue my aspirations.
I appreciate my sister and brother's sleepless evenings in assisting me through difficult circumstances.
Last but not least, I want to express my gratitude to my friends for listening to me, offering me advice, and encouraging me along the way."

- Masa Taweel

ACKNOWLEDGEMENTS

We would like to express our deepest gratitude to our graduation project supervisor **Dr. Baha' AlSaify** for his academic support, commitment, patience and insightful feedback that pushed us to sharpen our thinking and brought our work to a higher level. Dr.Baha' was an ideal teacher, mentor, and thesis supervisor, offering advice and encouragement during our time working on this project.

We are overwhelmed in all humbleness and gratefulness to acknowledge our dearest professors at the Network Engineering and Security Department for their valuable guidance throughout our studies. You provided us with the knowledge needed to choose the right path and successfully complete our five years' journey. We also would like to express our sincere appreciations to our senior graduation unit for their generous discussions and for providing the suitable environment to fulfill the requirements of this project.

We would like to acknowledge our colleagues and friends for their wonderful collaboration and who so generously took time out of their schedules to participate in collecting samples and information needed to build datasets for this work and make this project possible. Furthermore, we would like to send our special thanks to Eng. Omar Al-Hory who helped us find our footing as we started this process.

ABSTRACT

Voice and natural language processing, are at the forefront of any human-machine interaction domain. Speech is an effortless and usable method of communication, based on the voice as the analysed object. It permits the machine to identify and comprehend human spoken language through speech signal processing and pattern recognition. We can use the voice as an identification parameter instead of other biometric-based methods like passwords due to the ease of employing voice authentication mechanism, as it requires less time to verify the user identity, no passwords to remember, no countless attacks issues.

The structure of a speaker recognition system requires cautious attention to various matters like the feature extraction technique, and the voice sample format. To evaluate the performance of speaker recognition system, a dataset that is comprised of voice samples of the speakers to identify needs to be collected. This dataset will be used to train the identification models. The gathered dataset will be analyzed and relevant features will be extracted to reach high identification accuracy. The current state-of-the-art speaker recognition systems are built on deep neural networks and based on feature extraction pipelines to extract mel-filterbanks or cepstral coefficients.

Our work offers a comparison between Machine learning and Deep learning approaches for speaker recognition. Support Vector Machine and Random Forest models are used with statistical features next to Mel Frequency Cepstral Coefficients (MFCC) features to examine the capability of optimized machine learning models to achieve the best performance accuracy. Deep Neural Network (DNN) is used with both statistical features and MFCCs too, Long Short-Term Memory (LSTM) showed better performance when used with MFCCs only, and Convolutional Neural Network (CNN) is used with a generated Spectrogram of each collected sample.

Contents

List of Tables

List of Figures

Chapter 1

INTRODUCTION

Voice recognition systems are designed to authenticate and identify a speaker's voice from a group of others. The goal of the speaker recognition system is to convert the speaker's voice waveform to a parametric representation to be used in further processing pipelines and finally be employed as the input data for many models and approaches that are built to satisfy the system needs.

1.1 Voice Recognition Technology Principle

1.1.1 Voice Verification Versus Voice Identification

Voice Recognition also known as Speaker Recognition, is divided into two categories: speaker identification [1] and speaker verification [2]. Speaker identification is used to determine which one of the people speaks, and it answers the question: "who is speaking?". On the other hand, speaker verification is used to determine whether a specific person speaks and answers the question: "is the person speaking who he claims to be?". Once the speaker is verified, they can be authenticated to the system.

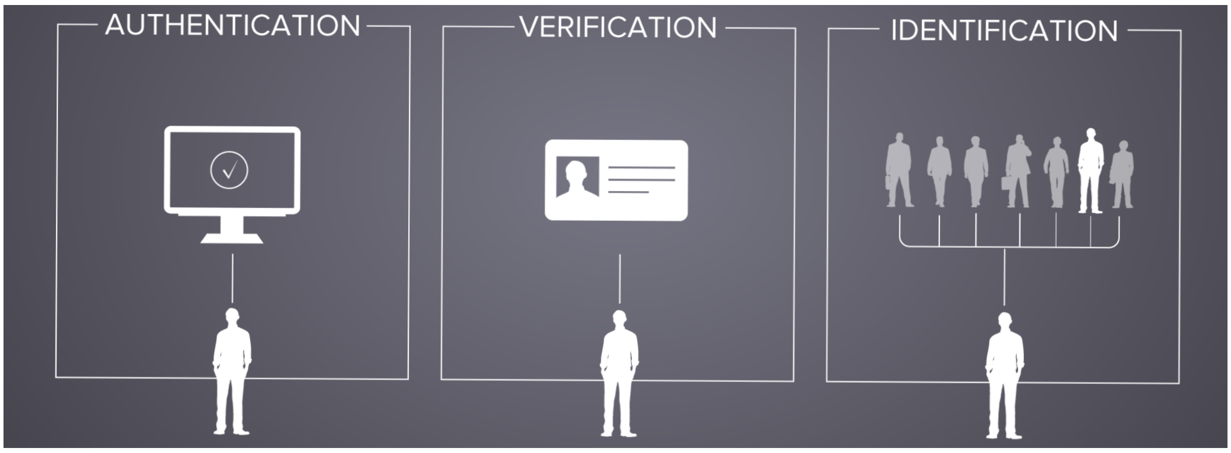

Figure 1.1: Key difference between Authentication, Verification and Identification.

1.1.2 Text-Based Systems

Speaker recognition systems are mainly divided into two types: text-dependent speaker recognition, which requires the speaker to say precisely a given text, and text-independent speaker recognition which doesn't constrain a speaker by a specified speech content. In comparison with Text-Dependent Speaker Recognition, it is more convenient because the speaker can speak freely to the system. However, when it comes to the implementation phase of the Artificial Intelligence system, it requires intensive training and testing utterances to achieve high accuracy and good performance. In this project, we had a combination of text-dependent and independent voice samples and trained our system on both types.

1.2 Voice Recognition in AI

1.2.1 Machine Learning Techniques

Machine Learning is a system of automated data analyzing, pattern identifying, and processing algorithms that enhance the decision-making process automatically through the learning experience, which implies that the algorithm can gain new information by inspecting and training on raw data [3]. These insights and information can change actions and responses, which will enhance the output prediction process and make it more reliable and scalable. Machine learning algorithms, known as Shallow Algorithms [4] due to their "less-complex" mathematical computations than deep learning, are divided into two types: Supervised learning [5], which will be employed in our project, and Unsupervised learning [6].

In a supervised learning model, the algorithms learn using labeled datasets to generate a prediction for an output label. As the dataset used in this project contains a label for each sample and the problem to be solved is to develop models that provide a prediction for an output speaker, then it's convenient to choose supervised learning models that are specialized in classification tasks such as (i,e. Support vector machine, random forest).

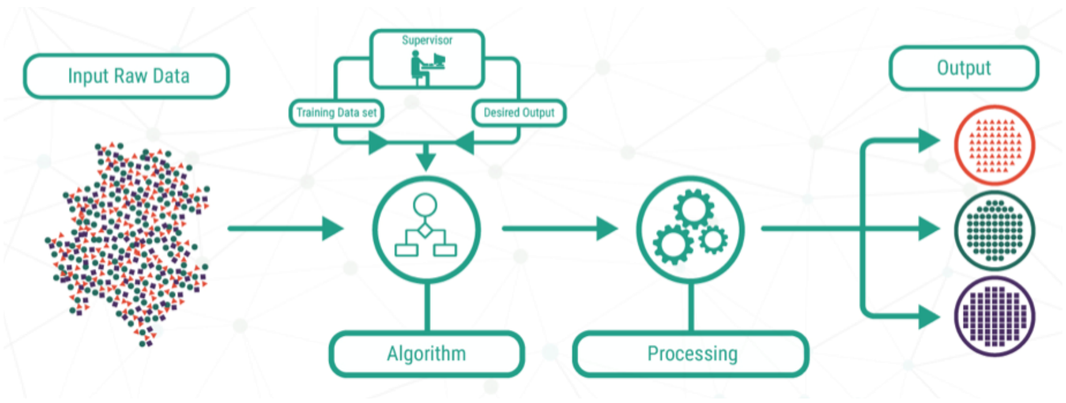

Figure 1.2: Supervised classification machine learning models process .

SVM [7] can be used for Regression [8] problems but mostly used for classification problems. It is based on statistical learning theory. The primary aim of this technique is to project nonlinear separable samples onto another higher dimensional space by using different types of kernel functions. RF [9] is based on ensemble learning techniques and many Decision Trees. All calculations are run in parallel and there is no interaction between the Decision Trees. And just like SVM, RF can be used to solve both Classification and Regression tasks.

1.2.2 Deep Learning Techniques

Deep learning is a type of machine learning that is based on artificial neural networks, neural networks consist of many layers of stacked neurons, these neurons mimic the human brain cells, and the learning process of the neural network model mimics the learning process of a human.

The concept of deep learning is on hype nowadays, DL models provide a very strong processing power through their complex computations in handling giant amounts of data. As in the last 10 years, the number of applications that entered the market and are based on DL increased exponentially.
Through the past 10 years, Many types of researches have been carried out in the field of speaker recognition, but little progress has been made by researchers. As Deep Learning outperformed most of the machine learning techniques, it became used widely in the topics of speech technology and it achieved new state-of-the-art solutions for one of the hardest problems of the decade.

1.2.3 Spectrogram of Voice

The reason a spectrogram [11] plot is an important element in our project is for the fact that it's used as a feature to be fed to our Convolutional Neural Network (CNN) [10] model. As known CNNs take images as a three dimensions input to the first layer, so the typical extracted features used in other models discussed in this project were inappropriate. A spectrogram is a very detailed view of an audio signal, it visualizes the spectrum of frequencies of a signal as it varies with time. A spectrogram general format is a graph of two dimensions where the x-axis represents the time and the y-axis represents the frequencies.

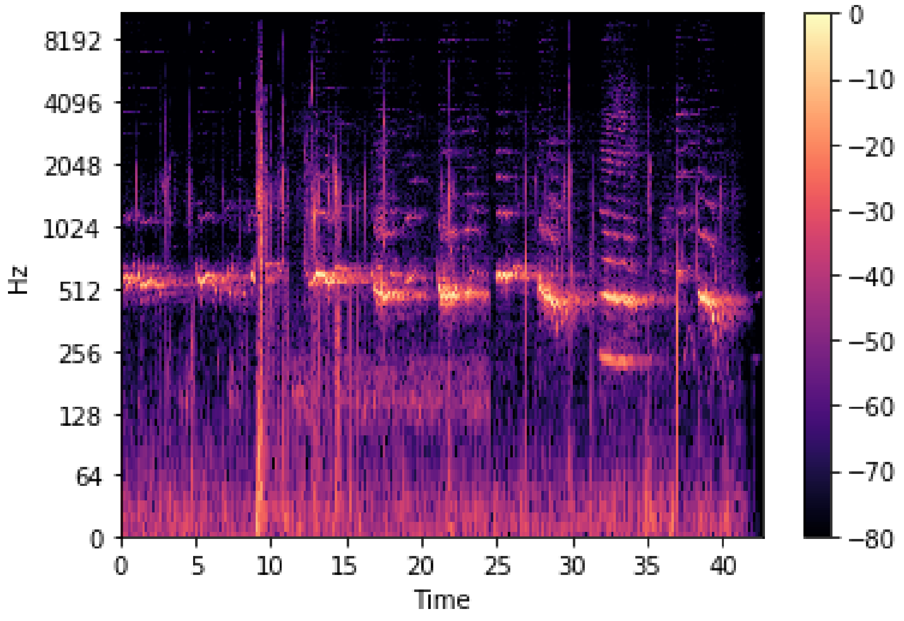

Figure 1.3: A Spectrogram plot of a male speech.

Chapter 2

REVIEW OF RELATED LITERATURE

2.1 History

Research on speaker recognition has a long history. In 1956, speaker recognition began to enter the era of artificial intelligence when several computer scientists conducted a study to forward push the concept of artificial intelligence into the field [12]. But due to poor hardware capabilities and immaturity of related algorithms, the outcomes of the research were unsatisfying. Until the 1980s, the AI field underwent a powerful revolution, and the research on speaker identification has been greatly developed [13]. The process during that period consisted of the following: data preprocessing, feature extraction, model constructing, and model scoring. The above stages made the bedrock for many models and applications that have been developed over the years.
In the past 10 years with the development of deep learning, approaches such as DNN and CNN have been used in speaker identification tasks [14]. As CNN uses Spectrograms, a study was made a long time ago to sort out whether a spectrogram of a humans' voice can identify that human or not [15]. In the process, they concentrated intently on the format construction of phonemes uttered by a speaker and how it changes along with the same speaker aging. Studies showed that one structure of a speaker contains very much variates from another speaker. Yet additionally, one's structure uncovers a vigorous shift in lower frequency components with aging. So, a spectrogram can be used to identify someone's identity with a 90% probability that it will be useless after certain years.

In recent years, speaker recognition technology has achieved tremendous success as it has become more affordable and reliable. Its software has been enriched by the power of the machine and deep learning. Many strategies were designed and deployed for realistic biometric systems. In 2010, Apple launched Siri, while Amazon's Alexa was introduced to the world in 2016. These applications work with a fixed authenticating wake-up word ("Hey Siri" and "Okay Google"), enabling users to control their devices more efficiently than ever before.

2.2 Related work

Many models and applications were developed for voice recognition and automatic speech recognition (ASR) [52] and were deployed in biometric applications and multimedia information systems. This section will discuss three toolkits we used as reference and a starting point in building models presented in this work and to understand some critical points related to the project process.

2.2.1 NeMo

Nemo is an open-source AI toolkit provided by NVIDIA [16], built for researchers to help them in the industrial and academic field to develop new state-of-the-art conversational AI models and reuse previous work, like codes and pre-trained models. Nemo provides some key features:

1. Speech processing that supports Automatic Speech Recognition (ASR), Voice Activity Detection (VAD), and Speaker Recognition.

2. Natural Language Processing, which supports text classification.

3. Speech Synthesis, which supports end-to-end speech generation and spectrogram generation.

2.2.2 Kaldi

Kaldi [17] is an open-source AI toolkit developed for dealing with voice issues and speech data. It's used in voice-related systems and applications such as speech recognition. Kaldi consists of three parts, preprocessing and feature extraction, the model itself, and the training process. Many models that work with audio samples deal with some characteristics extracted from the data, depending on features that will be useful for identifying the speaker and discarding any noise. It works with MFCC features in addition to I-vectors. Kaldi was implemented with an alternative GPU implementation based on CUDA [19]. It can be divided into two main parts, the first part used to be a Gaussian Mixture Model (GMM) [18] , but now it is replaced by deep neural networks. The second part is used for decoding and converting the phonemes to lattices.

2.2.3 SpeechBrain

SpeechBrain [20] is an open-source, all-in-one AI toolkit. It is a simple, flexible, and user-friendly tool that provides state-of-the-art performance in various domains. Speaker recognition, which is deployed in various applications, is worked with X-vectors, PLDA for SpeechBrain. It also provides different methods for speech enhancement, such as spectral masking, spectral mapping, and time-domain enhancement.
The speechBrain toolkit is being used in speech processing, multi-microphone processing, and to speed up the research process for speech technologies.

Chapter 3

ANALYSIS AND DESIGN

Speaker recognition is a valuable bio-feature recognition method. It aims to recognize someone's identity based on their speech signal. These bio-metric recognition technologies have been used in many fields, such as secure access to highly secured areas, machines such as voice dialing, banking, database, and multi-factor authentication.

3.1 Design Requirements

3.1.1 Dataset acquisition

The goal of gathering our dataset is to have a new English speech dataset suitable for training and evaluating speaker recognition systems. Samples were obtained from non-native English speakers from the Arab region over two months. Our dataset provides insights into the accents spoken by English speakers of Middle Eastern descent. It consists of 150 speakers with 3,300 data samples total and about seven hours of speech. Twenty-two voice samples per speaker were collected, each sample is about five to ten seconds long. The dataset was divided into three sub-datasets: The first sub-dataset contains ten samples per speaker repeating the phrase "Machine learning 1, 2, 3, 4, 5, 6, 7, 8, 9, 10". The second sub-dataset also contains ten samples per speaker speaking different phrases randomly. The third sub-dataset contains only two samples per speaker, each speaker speaks random phrases in the Arabic language. It should be mentioned that this sub-dataset is not for training reasons but only for testing and analyzing the stability of the models against different languages.
Table 3.1 shows in detail the division of the dataset.

Subset	**Number of Samples**	**Number of Speakers**		**Data Duration**
		Female	Male	
Same Phrase	1500	96	54	2.9 hours
Different Phrase	1500	96	54	3.1 hours
Arabic Phrase	300	96	54	1 hour
TOTAL	3300	150		7 hours

Table 3.1: Dataset subsets.

The dataset was obtained in a real-world, noisy, and single-speaker environment, which means it is free from audience and people overlapping speech. The voice samples were recorded using different recording hardware (e,g. Microphones, mobile devices, and laptops) then uploaded on cloud applications and shared with us.

3.1.2 Open-Source FrameWork

Practicing and developing AI models requires an efficient, easy-to-use, and friendly-interface a framework that supports related packages and tools. In this project, we chose Google Colab notebook [28] as our working platform. Colab allows programmers to write, debug and run python codes with no previous setup for the language or its packages. Colab is well suited to machine learning and deep learning development and optimization, it provides free access to computing resources such as storage, RAM, and of course GPUs [29].

3.1.3 Graphics Processing Unit (GPU)

GPUs are an essential part of a modern artificial intelligence infrastructure, they dramatically speed up computational processes for deep learning. Fortunately, Colab provides free but limited access to a virtual GPU which has been used during the implementation of this project.

3.2 Engineering Standards

The work in this project depends on the following standards:

3.2.1 Free Lossless Audio Codec (FLAC)

FLAC [30] is an audio coding format categorized as a lossless compression algorithm of digital audio signals. We used it as an extension of the collected samples in our dataset instead of others (i.e, wav and mp3) due to its ability to retain all the quality and information of the original audio. Not to mention that it's a compressing algorithm which mean its files take less space.

3.2.2 Fourier Transform (FT)

Fourier transform is a mathematical standard used to decompose the complex audio signal wave which is composed of multiple single-frequency. After applying FT concept the signal represented by its basic and constituent frequencies with the magnitude of each frequency.
Fast Fourier Transform (FFT) [23] is one of the FT types and an important measurement method in the science of speech and speaker recognition. It converts a signal into individual spectral components and thereby provides frequency information about the signal.

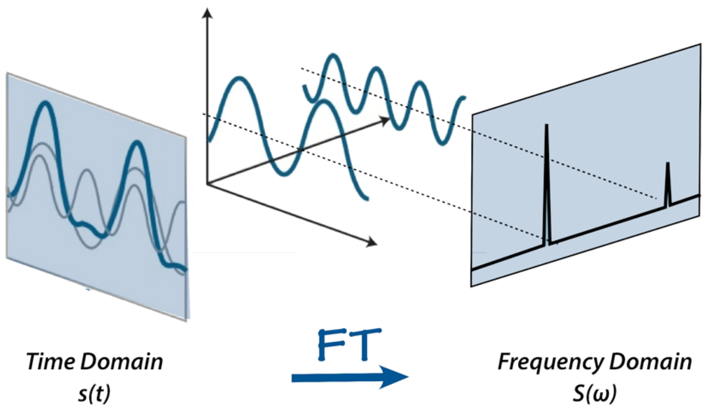

Figure 3.1: Fourier Transform

3.2.3 NumPy

NumPy [31] is a package in the python programming language, providing random number generators, linear algebra routines, and Fourier transforms built-in functions. It also supports comprehensive, multi-dimensional arrays and matrices, along with a great collection of high-level mathematical functions.

3.2.4 Pandas

Pandas [32] is a flexible, fast, and easy-to-use open-source package written in a python programming language. It provides data structures that include labeled axes and procedures for managing statistical tables and time series. It intends to be the fundamental high-level building block for doing practical, real-world data analysis in python.

3.2.5 Keras

Keras [33] is the most used DL framework, it's an open-source and user-friendly software library that acts as an interface for TensorFlow [34] package. It offers compatible APIs, minimizes the number of user actions needed for familiar use cases, and provides cleared actionable error messages.

3.2.6 Tensorflow

TensorFlow is an end-to-end open-source machine learning platform, it contains flexible and comprehensive tools, libraries, and ML resources that enable researchers to build, test, and develop ML/DL models to create new state-of-the-art powered applications.

3.3 Realistic Constraints

The last ten years in Speech Recognition have primarily focused on minimizing mistakes while interpreting voice inputs, that's what made widely known systems like Siri, Alexa, and Google Assistant feasible. After the production of Voice Recognition systems, the are frequently developed to adapt latest updates, but just like other systems they are exposed to many challenges, constraints, limitations, and maybe attacks. This section discuss the most familiar challenges voice recognition faces.

3.3.1 Deep fake and Voice spoofing

Recently, the number of spoofing attacks on voice systems witnessed rapid growth. Voice spoofing [35] presents an attack using unnatural or fake biometrics of a speaker. Fraudsters use progressive speech synthesis, voice modification or fabrication, and recorded replay to mimic systems.
Lately, this issue has been solved by developing an active detection that improves security when authenticating the speaker and provides better identity proofing during enrollment.

3.3.2 Limitations of Machine Learning models

Although machine learning algorithms showed a promising results in terms of classifying speakers, the main problem these models face is that they're not scalable to be applied in huge realistic systems. SVM and RF alone don't support incremental learning, which causes the model to be retrained from scratch every time a speaker register to the system.

3.3.3 Background noise and Environmental Limitations

One of the main obstacles speech models face is dealing with poor samples of data, this is due to the noisy background during data collecting (i.e., rain), and using low-quality recording hardware.

3.3.4 Lack of data

Machine learning algorithms demand enormous amounts of data before they give valid results. An example is neural networks. Neural networks are data-eating machines that require large amount of training data. The more complicated the architecture, the more data is needed to produce viable results.

3.4 Alternative Designs Approaches

The following design were thought about mainly to be employed in realistic systems that requires registering and signing-in such as our JUST student services. The design includes two major features: speaker verification model and ASR model. The proposed approach enable users to sign in to their accounts using their voices with a high security level where the ASR model acts as a defence mechanism against voice spoofing. Figure 3.2 shows the characteristics of the system.

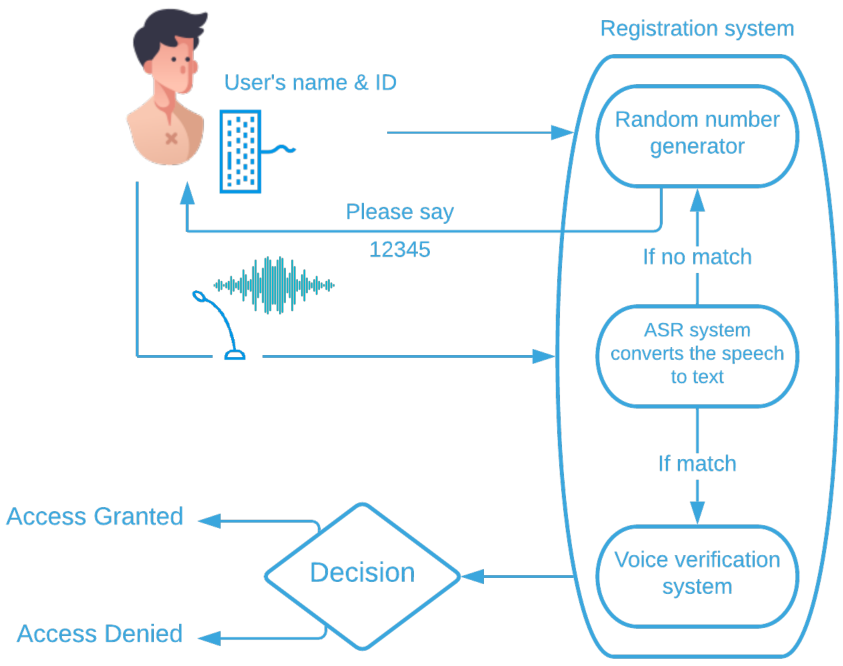

Figure 3.2: Voice Verification with ASR system

While registering to the system, a speaker is given 3-5 phrases to speak out, the system then form a unique numerical representation for the user's voice and save it as a reference model in its database. When signing in to the system the user first enters his username or ID, the system checks whether the username is registered in the system, if not then it asks the user to register first. If it was already registered, the system requests a spoken series of numbers generated randomly. The user should reply saying the specified numbers provided in the text from the

system. Once the user voice is captured, it is sent to the ASR model. The ASR converts the spoken words back into text and then a comparison is applied between the converted text and the original. If the two text don't match then a new number is generated and sent back to the user with an acknowledgment that the previous trial was faulty.
When the required text is finally spoken and converted correctly, the speaker's voice is sent to the Speaker verification model where the model extract the speaker's features again and do a test against the one stored in the database through the registration phase.
The system refers to the model saved in the DB using the ID given by the user as a text. The comparison is done using scoring functions such as cosine similarity [53], the probability obtained from the scoring function will be compared to a predetermined threshold and the result of this comparison will determine the decision logic either accepted or denied.

In the discussed system above, the password is actually a random generated number spoken by the user. This acts as a security feature where the password is unexpected and protected from a fraudster who might for example record the valid user speaking a fixed password.

3.5 Developed Design

In this section we're going to discuss the different designs and approaches applied in our work, giving an overview of the steps taken through the process of this project.

3.5.1 Feature Extraction

Feature extraction requires mindful attention because speech recognition performance depends laboriously on the feature extraction phase. We represent two types of features that were used in our work:

3.5.1.1 Statistical Features

The First Step for voice recognition is to extract features from a given signal. Overall, seventeen statistical features were extracted from each sample in the time and frequency domain. Data were converted from time to frequency domain by applying the FFT on each sample. Figure 3.3 shows a block diagram of how the features were extracted from voice data in both the time and frequency domain and then saved in a CSV [24] file, this file will represent our Pandas Dataframe used in training and evaluating the models.

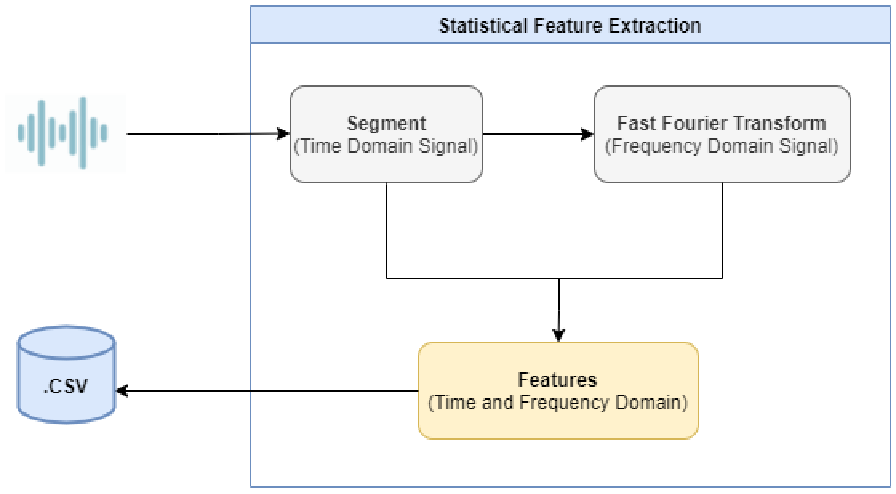
Statistical Feature Extraction
Segment
(Time Domain Signal)
Fast Fourier Transform
(Frequency Domain Signal)
Features
(Time and Frequency Domain)
.CSV

Figure 3.3: An architecture for statistical feature extraction.

3.5.1.2 Mel Frequency Cepstral Coefficients (MFCCs)

Mel Frequency Cepstral Coefficients (MFCC) is a leading approach in speech feature extraction techniques, the main advantage of MFCC is that it is good in error reduction and able to produce a robust feature when the signal is affected by noise. MFCC mimics the human perception for sensitivity at lower frequencies, as the human ear has what is called the cochlea [27] which is an anatomy that has more filters at low frequency and very few filters at a higher frequency. MFCC algorithm converts the conventional frequency to Mel Scale.
The MFCC extraction technique includes windowing the signal, applying the Discrete Fourier transform (DFT) [25], taking the log of the magnitude, and then warping the frequencies on a Mel scale, followed by applying the inverse Discrete Cosine Transform (DCT) [26]. The basic concept of the MFCC method is shown in the block diagram in Figure 3.4, which will be explained in details in the implementation section.

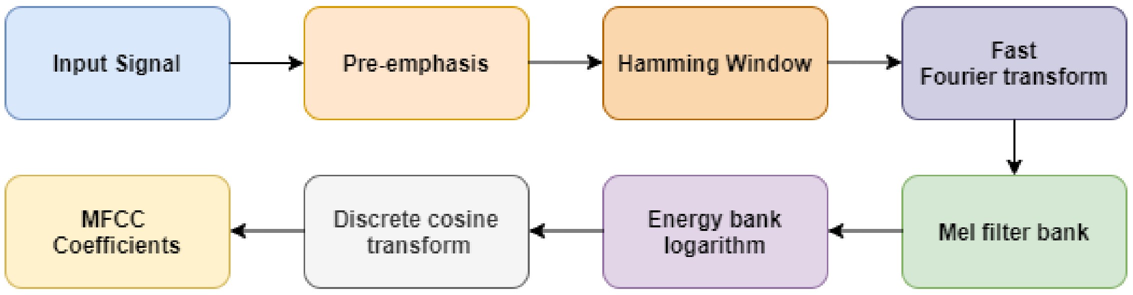

Figure 3.4: Mel Frequency Cepstral Coefficient computation steps.

3.5.2 Machine Learning Designs

3.5.2.1 Support Vector Machine (SVM)

The SVM is a supervised learning method used for regression, classification, and outlier detection. Before the revolution of neural networks, SVM was one of the most powerful classification techniques in machine learning and still to this moment a valuable algorithm used in many fields. The objective of the SVM algorithm is to find a hyperplane in an N-dimensional space that distinctly classifies data points, in our case N is the number of features extracted and each data point represents one speaker label. SVM uses kernel function, which can take many values and provide shortcuts to avoid complex calculations.

3.5.2.2 Random Forest (RF)

The RF algorithm is a supervised classification system composed of several decision trees. They select random data samples, get predictions from each tree, and then choose the best vote as the final result. It prioritizes characteristics by removing the least significant ones and focusing on the most important ones. A Decision Tree is a collection of connected nodes where each node is reachable from the previous node depending on certain criteria. Increasing the number of trees greatly reduces the problem of overfitting and makes an accurate prediction tool in Random Forest Classification. The following block diagram 3.5 shows the process of training and testing for the RF model.

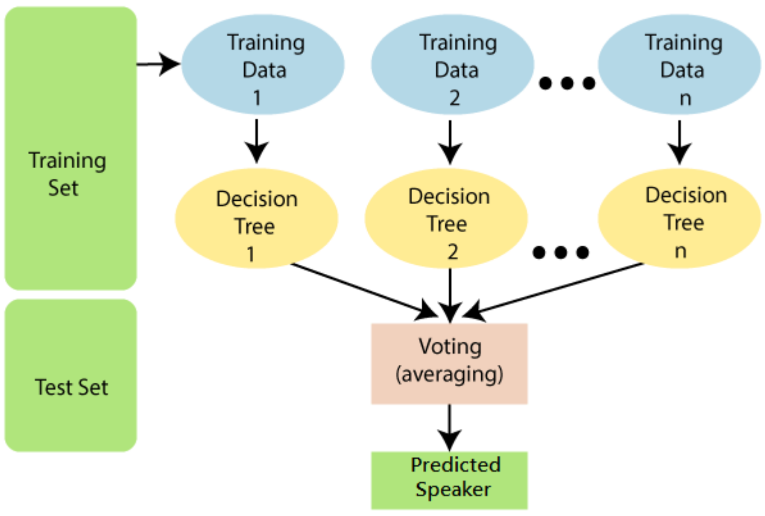

Figure 3.5: Random Forest Algorithm Process.

3.5.3 Deep Learning Approaches

3.5.3.1 Deep Neural Networks (DNN) Design

Deep neural networks (DNNs) are a type of deep learning algorithm that is inspired by the structure of neurons in the human brain. They are also known as Feed Forward Neural Networks (FFNNS) [47]. Deep neural networks are composed of interconnected neurons organized into dense layers, which include input layers, hidden layers, and output layers. In a model, each neuron in the dense layer receives the output (weight) of every neuron from the preceding layer where neurons performs matrix-vector multiplication operations. DNN can be made deeper or wider by adding more hidden layers or by having more hidden nodes in each hidden layer. Figure 3.6 shows the general architecture of a DNN.

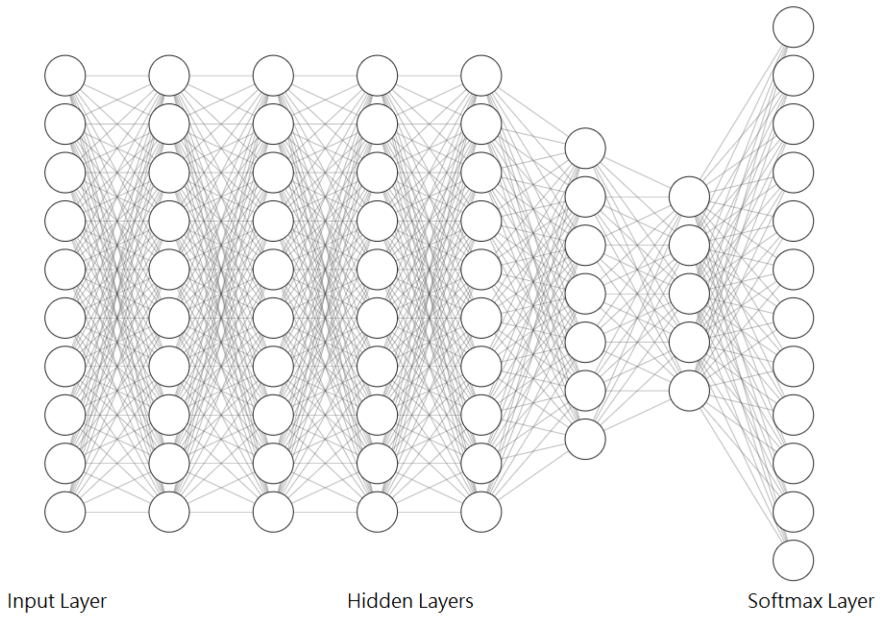

Figure 3.6: DNN model architecture

In this work, we constructed the model to have ten layers: The input layer, eight hidden layers and the output layer as shown below in Table 3.2. We used the *RELU* function as an activation function for all dense layers in addition to two dropout layers which significantly prevents overfitting the model. The last layer which is the softmax layer normalizes the output of the network to a probability distribution over predicted output classes (speakers), the highest probability would be the predicted speaker.

Layer	# Filters	Activation Function	Output Shape
Dense-1	128	relu	None × 128
Dropout	-	-	None × 128
Dense-2	128	relu	None × 128
Dropout	-	-	None × 128
Dense-3	128	relu	None × 128
Dense-4	128	relu	None × 128
Dense-5	128	relu	None × 128
Dense-6	64	relu	None × 64
Dense-7	32	relu	None × 32
Dense-8	150	softmax	None × 150

Table 3.2: The Architecture Used For DNN model.

3.5.3.2 Convolutional Neural Networks (CNN) Design

Convolutional Neural Network is a type of deep neural networks, it mostly works with visuals and images. As mentioned before, the generated spectrogram of a single voice sample represents the input of the first layer of ConvNet, then each following layer generates several activation functions that are passed on to the next one. Figure 3.7 shows a block diagram of data flow through layers.

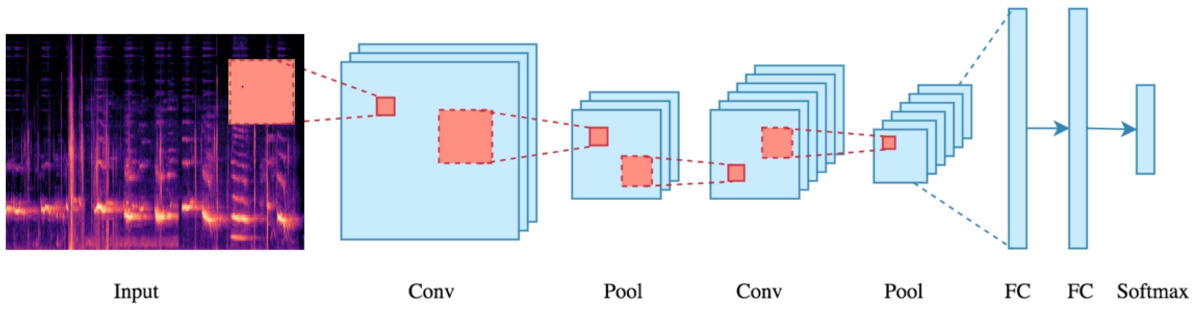

Figure 3.7: Data processed by each layer in CNN model

A pooling layer is added after the convolutional layer to reduce the size of the images while preserving the essential characteristics. The output of the last pooling layer is flattened and passed to a fully connected layer which recieves the flattened input vector, applies a linear combination and an activation function then it produces a new output vector.

In our project we used architecture similar to VGG-M [54], this model is extensively used for image classification and speech-related applications. The architecture details of our own are available in Table 3.3. Some modifications were applied considering the following:(1) input shape should be adapted to our input pipeline, (2) we used more pooling layers in the structure, as we found it to be empirically less prone to overfitting.

CNN looks at an image by its multi-channel concept. A spectrogram image just like any other digital image is Red-Green-Blue (RGB) [48] encoded, CNN mix these colors to construct

Layer	**Kernal**	**#Filters**	**pooling**	**Output Shape**
Conv-1	5 × 5	128	-	383 × 373 × 128
BatchNormalization	-	-	-	383 × 373 × 128
MaxPooling	-	-	2 × 2	191 × 186 × 128
Dropout	-	-	-	191 × 186 × 128
Conv-2	5 × 5	128	-	189 × 184 × 128
BatchNormalization	-	-	-	189 × 184 × 128
MaxPooling	-	-	2 × 2	94 × 92 × 128
Dropout	-	-	-	94 × 92 × 128
Conv-3	3 × 3	64	-	90 × 88 × 64
BatchNormalization	-	-	-	90 × 88 × 64
MaxPooling	-	-	2 × 2	45 × 44 × 64
Flatten	-	-	-	28224
fc-1	-	512	-	512
Dropout	-	-	-	512
fc-2	-	128	-	-
Softmax	-	150	-	-

Table 3.3: The Architecture Used For CNN model.

the spectrum humans perceive. It ingests such images as three separate strata of color stacked on top of each other. Technically the image is fed to the first layer as a 3D array each cell of these arrays contains a pixel of the image.

3.5.3.3 Long Short-Term Memory (LSTM) Design

Long Short-Term Memory networks are kind of Recurrent Neural Networks, competent of learning long-term dependencies. LSTM is explicitly designed to avoid the long-term problem. It is shaped like a chain structure as all Recurrent Neural Networks with a different repeating model structure. The key to LSTMs is the cell state, which can be removed or added carefully controlled by structures called gates. Gates are a path to allow information through. They are formed of a sigmoid neural network layer [49] and a multiplication operation. The sigmoid layer outputs numbers between zero and one, defining how much is each component should be allowed. LSTMs have feedback connections that enable the network to process internal data. As a result, LSTMs are particularly useful in processing data such as text and speech. Figure 3.8 shows a repeating module designed in LSTM network.

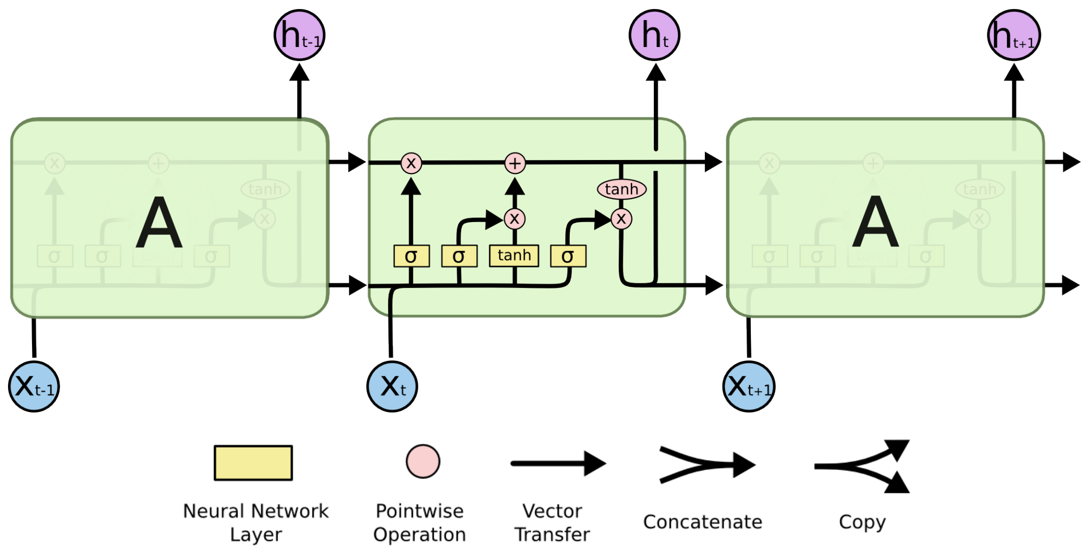

Figure 3.8: The repeating module in an LSTM

Our model is built with many layers in a specific order, represented in the following table 3.4. Each layer has its own characteristics; output shape, number of filters and the used activation function in dense layers. The dropout layer is for preventing overfitting in the network. It sets a node's weight to zero with a given probability, reducing the number of weights required for training at each iteration

Layer	**#Filters**	**Activation Func**	**Output Shape**
LSTM-1	128	-	None × 20 × 128
LSTM-2	128	-	None × 20 × 128
Dropout	-	-	None × 20 × 128
Dense-1	128	relu	None × 20 × 128
Dense-2	128	relu	None × 20 × 128
Dense-3	64	relu	None × 20 × 64
Dense-4	64	relu	None × 20 × 64
Dense-5	64	relu	None × 20 × 64
Dense-6	150	softmax	None × 20 × 64

Table 3.4: The Architecture Used For LSTM-RNN model.

Chapter 4

IMPLEMENTATION

This chapter implements in details the phases of this work. Starting from the processing of the dataset to the implementation of both Machine Learning and Deep Learning models.

4.1 Dataset and Data Processing

In this section, the different stages by which the collected data was checked, cleaned, and unified are described.

Stage1.Collecting Samples and labeling.
In this stage, we gathered all the samples received from volunteering speakers whom used different communication platforms (e.g. Messenger, Whatsapp,Cloud Apps, ...) into one dataset. The collected samples were then divided into three sub-datasets based on the phrase to be included.

Stage2. Dumping data into CSV file.
Speakers were asked to give some personal information about them, these information are: Gender, Age, Height and Weight. So after giving these speakers labels, an entry was created for each one in the CSV file and then their information is added in the same row.

Stage3. Filtering samples.
All samples were checked manually, each voice record were listened to, to make sure that it was free of any overlapping speech noise. For the samePhrase sub-dataset, we made sure that samples did abide by the phrase “Machine Learning 1, 2, 3, 4, 5, 6, 7, 8, 9, 10”. Any speaker sent samples that did not abide by the instructions was removed from the dataset, and its label was dropped from the csv file that contains the speakers information.

Stage4.Unifying the format.
Samples collected were received in various formats. For example, samples collected via Whatsapp have the .ogg [38] extension while Facebook Messenger supports the .mp4 [39] format.All the collected samples were converted to .flac format to have a unified format.

Stage5. Renaming files.
Each file was renamed to start with its speaker label and then a dash then the number of the sample. For example, if a speaker has label 149 then the first sample in the samePhrase sub-dataset will be “149-1.flac” while the last sample in the samaPhrase sub-dataset will have the label “149-10.flac”. The same speaker will have entries in the differentPhrase sub-dataset that will start with label “149-11.flac” and will end with “149-20.flac”. The directories hierarchy for the dataset and the samples numbering scheme are provided in Fig.4.1.

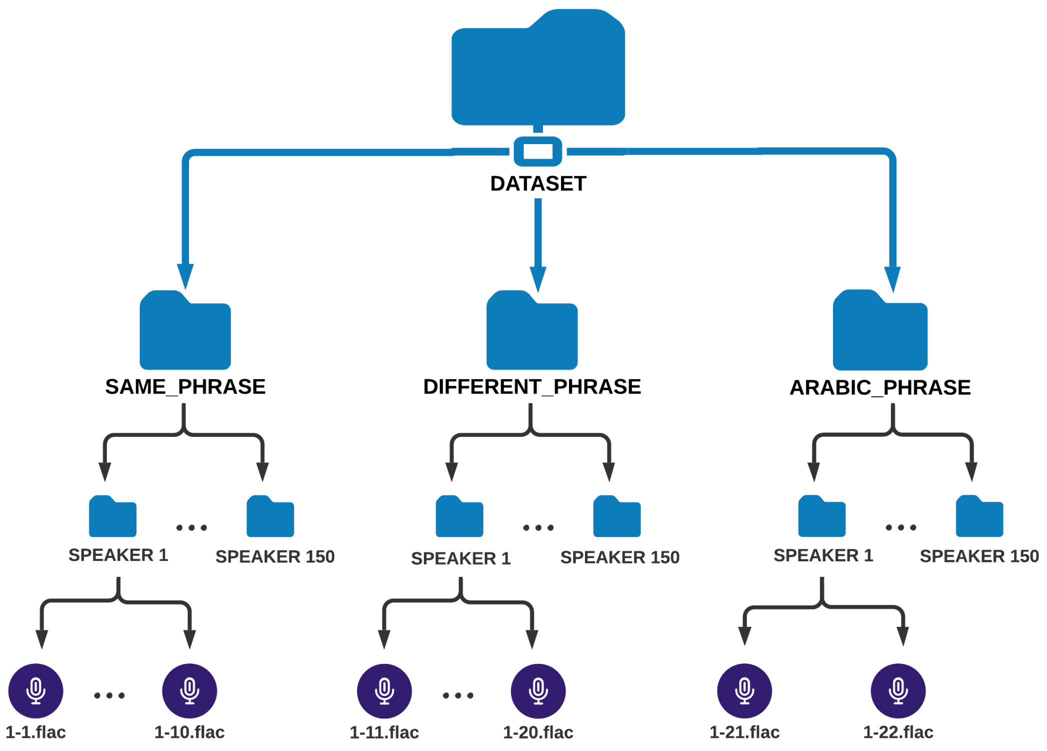

Figure 4.1: Architecture of the dataset.

Stage6. Changing from stereo to mono.
Speakers' samples were recorded on different hardware devices, each device is configured independently. So we noticed while plotting the signals that some have two channels (stereo) and some have one channel (mono) and to have a balanced data we had to unify the number of channels all to mono. The following script 1 was applied.

```python
directories = ["Same","Diff", "Arabic"]
for d in directories:
    path = r'/home/walker/Desktop/2Channels/'+ d + "/"
    for file_name in os.listdir(path):
        path2 = path + file_name + "/"
        for samples in  os.listdir(path2):
            audio = AudioSegment.from_file(path2+samples)
            if audio.channels > 1  :
                channels = audio.split_to_mono()
                audio = channels[0]
            PathToExport = "~/Desktop/1Channel/"+ d + '/' + file_name
            if not os.path.isdir(PathToExport):
                os.mkdir(PathToExport)
            audio.export(PathToExport + "/" + samples , format="flac")
```

Listing 1: Python script for converting to mono type

Stage7. Publishing the dataset.
A journal of the dataset will be published on Data in Brief website along with the dataset as an open source. The attached pdf file with this report is a our journal to be published.

4.2 Implementing Feature Extraction techniques

4.2.1 Statistical Feature Extraction

The following snippet of code 2 shows the methods used to extract the statistical features, the extraction was done using the functions provided by librosa [50] package.

```python
#Extract the statistical features
for flac_path in flac_list:
    speaker_id = flac_path.split('/')[4]
    y, sr = librosa.load(flac_path)
    chroma_stft = librosa.feature.chroma_stft(y=y, sr=sr)
    rms = librosa.feature.rms(y=y)
    spec_cent = librosa.feature.spectral_centroid(y=y, sr=sr)
    spec_bw = librosa.feature.spectral_bandwidth(y=y, sr=sr)
    rolloff = librosa.feature.spectral_rolloff(y=y, sr=sr)
    spectral_contrast=librosa.feature.spectral_contrast(y)
    spectral_flatness=librosa.feature.spectral_flatness(y)
    zcr = librosa.feature.zero_crossing_rate(y)
    mean = np.mean(y)
    std = np.std(y)
    maxv = np.amax(y)
    minv = np.amin(y)
    median = np.median(y)
    skew = scipy.stats.skew(y)
    kurt = scipy.stats.kurtosis(y)
    iqr = scipy.stats.iqr(y)
    mode = np.array(scipy.stats.mode(y))[0]
```

Listing 2: Python script for feature extraction

4.2.2 MFCCs Extraction

The detailed description of various steps involved in the MFCC feature extraction is explained below.

1. **Pre-Emphasis.**
   Pre-emphasis applies filtering that accentuates the higher frequencies in voiced sounds. The goal of this step is to compensate the high-frequency part that was suppressed during the sound production mechanism of humans.

2. **Frame blocking.**
   The input speech signal is segmented into blocks with an optional overlap of the frame size due to its stable characteristics. The purpose of the overlapping analysis is that each speech sound of the input signal should be approximately in the middle of the same frame.

3. **Hamming Window.**
   each frame in our signal is multiplied by a hamming window function this is done to

enhance the harmonics, smooth the edges, and to reduce the edge effect while taking the DFT on the signal.

3. Mel Filter Bank.
The main purpose of the Mel-filter bank is to simulate and give a better resolution of the human ear perception of frequency which is more discriminative at lower frequencies. Each tone with frequency f, measured in Hz, is wrapped and measured on a scale called "mel". In our project, we chose 48khz as a mel value.

5. Discrete Cosine Transform (DCT).
DCT is the last process of MFCC feature extraction, based on correlating value of the mel spectrum and produces a good representation of property spectral local. It is used to generate a two-dimensional time matrix for each pattern to be recognized.

6. MFCC Coefficients.
MFCC coefficients are the resulting features of the steps we went through in the MFCCs extraction process. The average number of features is 12 for each frame. We extracted 20 coefficients in our project.

4.3 Correlation and Feature Selection

Correlation [51] is a statistical technique that determines how one variable is related and changes with the other variable. It gives an idea about the degree of the relationship between the two variables. We've applied this technique on the the extracted features and came back with an interesting results. The figure below 4.2 shows The correlation heat map between features in the time domain.

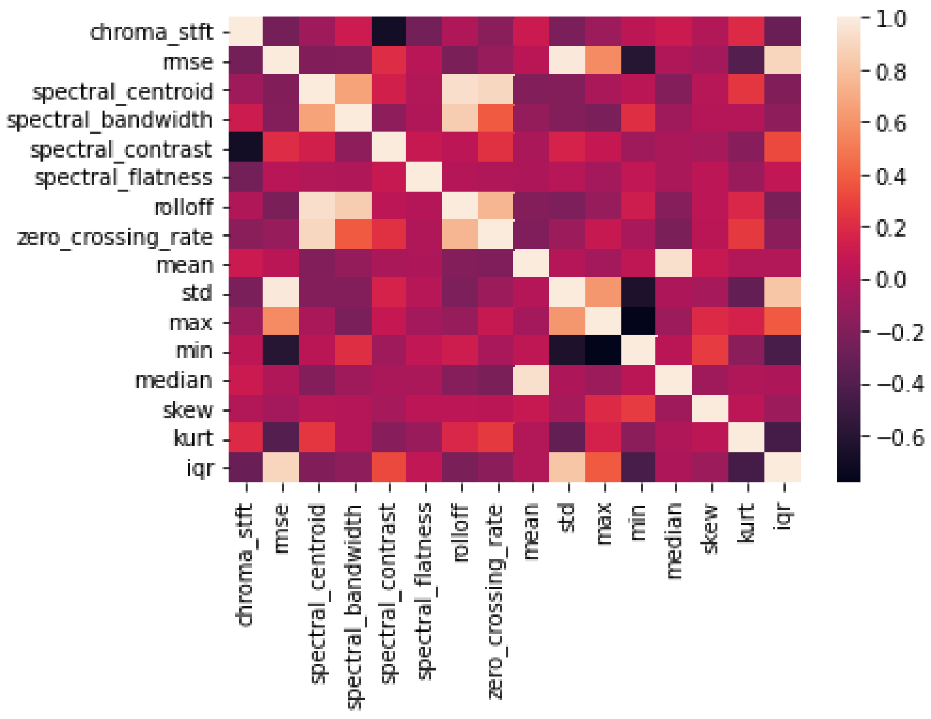

Figure 4.2: Correlation heat map between features in time domain.

One of two features with correlation more than 0.9 were excluded and this left us with 12 features to work with (chroma stft , rmse, spectral centroid, spectral bandwidth, spectral contrast, spectral flatness, mean, max, min, skew, kurt and iqr).
After applying correlation, three feature selection techniques were experimented too to explore which of the features are most important for training the models.

4.3.1 Recursive Feature Elimination (RFE)

RFE [43] is the most popular feature selection algorithm, that works by looking for features in the training dataset, starting with all of them and successfully removing them one by one until the appropriate number of the most relevant features remains. That enables the model to run more efficiently with less space or time complexity and be more successful. RFE provides many hyperparameters, including the estimator argument, which specifies the algorithm to use. The second argument is the number of features, which specifies the number of features to be selected. The last argument is the step, which is optionally used to manage the behavior of computational time that will be very heavy for a large dataset.

4.3.2 Minimum Redundancy Maximum Relevance (MRMR)

Maximum Relevance and Minimum Redundancy [40] is a flexible feature selection algorithm that can effectively reduce the redundant features while holding the relevant features for the model, this makes it fast and easily implementable. MRMR identifies many variants, which define the internal feature selection method. The most usable string values are Mutual Information Difference and Quotient schemes, MID, and MIQ [41]. Both of them result in features with similar accuracy. On the other hand, the MID method is preferred over the MIQ; it results in more stable features, especially in a small number of samples.

4.3.3 Chi-Square

The chi-square test [44] is a simple and applicable algorithm. It is being used to estimate the level of correlation and analyze the dependency between features. In feature selection, the purpose is to select the highly relevant features, so a high Chi-Square value indicates that the assumption of independence is incorrect. As a result, the high value of Chi-Square indicates the more dependent features, which can be selected for training.

4.4 Implementing Machine Learning Models

4.4.1 SVM

Before building the SVM model, methods were applied to estimate the best hyper-parameters in order to improve the model accuracy and achieve the best performance. We used a brute-force approach to determine which kernel performs best based on performance metrics such as precision, recall and f1 score [45]. The following table 4.1 summarizes the performance of each kernel referencing the Classification report.

Kernel	precision	recall	f1-score
Polynomial	0.00	0.01	0.00
RBF	0.03	0.02	0.02
Linear	0.83	0.83	0.82
Sigmoid	0.00	0.00	0.00

Table 4.1: Kernals Performances On The SVM Model.

The Kernel that gave logical results and was used to build our model is the Linear kernel. It was expected given that the provided data is linearly separable data.
After finding the best kernel to be used, *GridSearch* method was applied to estimate the best value for the two other parameters (C and *Gamma*). *Gridsearch* is basically used as an approach for hyper-parameter tuning, it methodically evaluate a model that is built from different combinations of parameters specified in a grid. The parameters passed to the grid were as follow:

```
param_grid = {'C': [0.1,1, 10, 100], 'gamma': [1,0.1,0.01,0.001],'kernel': ['linear']}
```

Listing 3: Parameters Passed To The GridSearch algorithm

The algorithm was Fitting 5 folds for each of 16 candidates, totalling 80 fits. It consumed approximately 9 minutes to fit all combinations then came back with the estimated parameters: C=0.1, *gamma*=1 and *kernel*=linear.

4.4.2 RF

As done previously with the SVM model, a finetunning algorithm was applied to estimate best parameters of the RF model. Different values were generated using a brute-force script then *RandomSearch* algorithm were used to estimate the best value for each parameter. The values of the parameters tested were as follow:

```
{'n_estimators': [200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000],
 'max_features': ['auto', 'sqrt'],
 'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, None],
 'min_samples_split': [2, 5, 10],
 'min_samples_leaf': [1, 2, 4], 'bootstrap': [True, False]
 }
```

Listing 4: Parameters Passed To The RandomSearch algorithm

The algorithm was Fitting 3 folds for each of 100 candidates, totalling 300 fits. It took about 15 minutes to get the following results: *n_estimators*=1000, *min_samples_split*=2, *min_samples_leaf*=1, *max_features*=auto, *max_depth*=50 and *bootstrap*=False.

SVM and RF models were constructed, fit, and trained using the best hyper-parameters found. All 37 features were used, the models showed a very good performance in terms of classifying 150 speaker, results are discussed in chapter 5.

4.5 Implementing Deep Learning Model

4.5.1 DNN

4.5.1.1 Data preprocessing

The the dataset is shuffled randomly and split into a 90% training subset which is used to train the model and a 10% testing subset to test the model performance. A 10% is divided for the validation subset from the 90% of the training data to evaluate the model during training. All numerical features are scaled prior to modeling using *StandardScaler()* method. This technique standardizes the input variables directly by subtracting the mean to center them on 0.0 and dividing by the standard deviation to give the standard deviation of 1.0. Then the scaled dataset is sent to *fit_ transform()* function to create a transformed version of our dataset. Now our dataset is processed and ready to be fed to the neural network. Script 5 shows how the above process was done by code.

```
    dataset=pd.read_csv('/content/OurData.csv')
    array = dataset.values
    X = array[0:,1:]
    Y = array[0:,0]
    lb = LabelEncoder()
    Y = to_categorical(lb.fit_transform(Y))
    X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.1,
                     random_state=150)
    X_train, X_val, y_train, y_val = train_test_split(X_train, y_train,
    test_size=0.1, random_state=12)
    ss = StandardScaler()
    X_train = ss.fit_transform(X_train)
    X_val = ss.transform(X_val)
    X_test = ss.transform(X_test)
    np.unique(y_test).shape
```

Listing 5: preprocessing the data

4.5.1.2 Defining the Keras model

The DNN model is constructed with dense and dropout layers. Each added dense layer has a set of hyperparameters. Units parameter determines the size of the dense layer's output. The kernel-initializer parameter initializes the kernel weights matrix to extract relevant feature kernels. The activation function parameter transforms the values of neurons' inputs, and the most commonly used function with DNN is *RELU*. The last activation function that is used in all discussed models is the softmax function, number of neurons in this layer is the number of classes in our dataset. Script 6 below shows the construction of the model.

```python
model = Sequential()
model.add(Dense(128, activation='relu', kernel_initializer='he_normal',
input_shape=(36,)))
model.add(Dropout(0.2))  #visible layer
model.add(Dense(128, activation='relu', kernel_initializer='he_normal'))
model.add(Dropout(0.2))
model.add(Dense(128, activation='relu', kernel_initializer='he_normal'))
model.add(Dense(128, activation='relu', kernel_initializer='he_normal'))
model.add(Dense(128, activation='relu', kernel_initializer='he_normal'))
model.add(Dense(64, activation='relu', kernel_initializer='he_normal'))
model.add(Dense(32, activation='relu', kernel_initializer='he_normal'))
model.add(Dense(150, activation='softmax'))
```

Listing 6: DNN Model Construction

4.5.2 CNN

4.5.2.1 Spectrogram Extraction

After loading the dataset, it's organised in a dataframe in three columns: the path to the sample, the sample name and the label to the owner of that sample. Then the dataframe is iterated row by row to defining the path of the voice sample and loading the it with no sample rate to use the original. The size of the image is set to (387,377) and a couple of operations were done to get rid of the axes to only keep the image itself. Figure 4.3 shows a generated spectrogram with axes and on the right is the same spectogram after eliminating the axes.

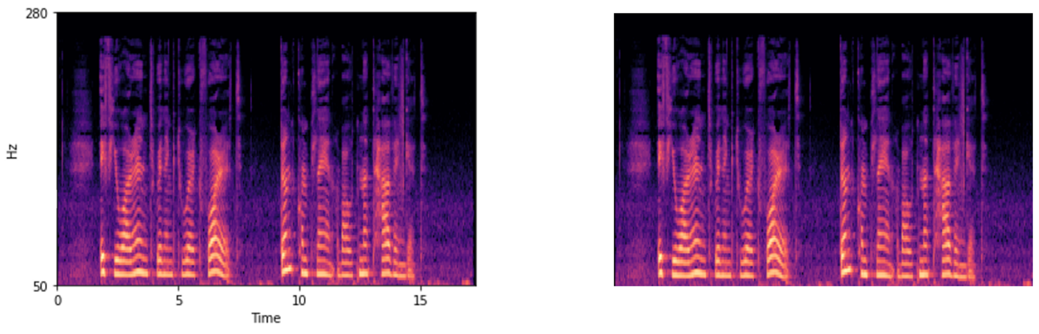

Figure 4.3: Eleminating the axes from a generated Spectogram

A high resolution is chosen and the images were saved in different folders to be easily used later with the generators. All generated images were re-scaled to (1/255) as 255 is the maximum pixel value.
The data is randomly split into 90% training and 10% for testing and validation, meaning that same data were used for testing and validating the model. The images were fetched for each subset of the data using datagen.flow_from_dataframe(), this is due the limited resources we have on colab that don't allow us to fetch three thousand images from memory all at once. The following code 7 shows how the images were fetched for the training subset using the package explained before.

```
    train_generator=datagen.flow_from_dataframe( dataframe=train,
    directory="images",
    x_col="file",    y_col='label',
    batch_size=40,   shuffle=False,
    class_mode="categorical",
    target_size=(387,377))
```

Listing 7: A code that fetch the images of the training subset

4.5.2.2 Defining the keras model

Once the data is ready, it is fed to the model as a 3D array each array holds the shape of the image generated and the 3 arrays represent the encoding of the image (RGB). This fullfill the input of our model which is (387,377,3) and is passed to the first layer of the model. The model is constructed using the follwing code 8.

```
model = Sequential()
model.add(Conv2D(filters=128, kernel_size=(5, 5),
                 activation='relu', input_shape=(387,377,3)))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.5))

model.add(Conv2D(filters=128, kernel_size=(5, 5), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.2))

model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(128, activation='relu'))
model.add(Dense(150, activation='softmax'))
```

Listing 8: CNN Model Construction

4.5.3 LSTM-RNN

4.5.3.1 Preparing the data

After extracting the features and saving them in a CSV file, the data was divided into subsets. The train-test split is a method for estimating the performance of machine learning and deep learning algorithms. The training and testing subsets are respectively split into 90% and 10% from the original full dataset. The training set is used to fit the model, while the testing set is used to evaluate the model by comparing the predictions to the expected values from the training process.

The last subset is the validation set, which is split into 10% from the training subset. This subset is held back from training the model and is used to estimate the model skill of the final tuned model when comparing or selecting between final models. The train-test split process is done using the following script 9.

```
data = pd.read_csv('/content/OurData.csv')
#Y taking the values of the label
y = data['id']
#X taking the values of features and here takes the MFCCs only
x = data.iloc[:,17:].values
#Encoding the labels
lb = LabelEncoder()
y = to_categorical(lb.fit_transform(y))
#Split twice to get the validation set
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.1,
random_state=123, stratify=y)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train,
test_size=0.1, random_state=123)
```

Listing 9: Preparing the data of the model

LSTM is fed with the extracted MFCCs only, which provided the model with better performance results.

4.5.3.2 Defining the keras model

The LSTM-RNN model is constructed with many layers. We need to stack multiple layers on top of each other, so the model is defined using Sequential class from Keras. There is no definite rule on how many layers the model must-have. In general, two LSTM layers are enough. Importantly, the LSTM layer requires three dimensions input. This step is done by setting the *return_ sequences* value to True, which will output all the hidden states of each time step, meaning the output will be a sequence of three dimensions array of real numbers instead of a single value. Each LSTM layer is accompanied by a Dropout layer, which helps control overfitting by skipping selected neurons –takes a group of weighted inputs, applies an activation function, and returns an output– during training [46]. After LSTM layers did their work, some activation layers are added. Activation functions that act as linear functions are needed to train deep multi-layered neural networks. They can be added through an activation layer or a dense layer as an argument. The used activation function in the LSTM model is, Rectified Linear Activation Function ($RELU$), which allows the neural network to learn nonlinear dependencies.

RELU returns input directly if the value is greater than 0 and returns 0.0 if the value is less than 0. Script 10 below shows the details of the LSTM model construction.

```
input_shape=(20,1)
model = keras.Sequential()
model.add(LSTM(128, return_sequences=True, input_shape=input_shape))
model.add(LSTM(128, return_sequences=True))
model.add(Dropout(0.2))
model.add(Dense(128, activation='relu'))
model.add(Dense(128, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(150, activation='softmax'))
```

Listing 10: LSTM model construction

4.5.4 Compiling and fitting the models

The processes defined previously initialize the structure of the models, now the models are ready to be compiled. The compilation step allows us to define the loss function and the optimizer to use as presented in snippet 11 below .

```
model.compile(optimizer='adam',loss='CategoricalCrossentropy',metrics=['acc'])
```

Listing 11: Models compilation

By reading the literature in the field we've found that categorical cross-entropy works very well for a classification task. The optimizer *Adam* [55] provides an efficient, less memory-consuming, and the best adaptive optimization for training deep learning models. It is precisely used in problems that hold terms of large data or parameters. Early stopping is a necessary step before fitting the model, which accommodates an essential rule in preventing the overfitting of the data. The substantive work of early stopping lies in intercepting the training once the model performance stops improving on a hold-out validation dataset. *EarlyStopping()* is a callback function, called in *fit()* and takes many options, monitor option can take two values, *val_ loss* or *val_ acc*, and the patience option specifies how long to wait for the non-improvement epoch. For example, if its value is set to five, the learning ends when consecutive five times no improvement happens. The following line of code 12 shows how *early_ stop* is used in our models, and how the models are fitted.

```
early_stop = EarlyStopping(monitor='val_loss', min_delta=0, patience=100,
                                           verbose=1, mode='auto')
history = model.fit(X_train, y_train, epochs=50, batch_size=30,
validation_data=(X_val, y_val), shuffle=False, callbacks=[early_stop])
```

Listing 12: EarlyStopping Function

Chapter 5

TESTING AND EVALUATION

In this chapter, evaluations for our implemented algorithms we proposed in Chapter 4 are presented. First, we present the results of each implemented approach. Then, we discuss the results and compare models to highlight the best achieved performance.

5.1 Feature Selection Algorithms Results

After applying feature selection techniques and fitting the selected features on the ML models, It was approved that the models tend to give higher performance without excluding any feature, this is due to the small number of features that we have, every information extracted was important enough to be fed into the models for training and evaluation.
The following table 5.1 shows the selected features of each algorithm applied and how much on average the accuracy of the ML models has dropped.

Algorithm	**Output Features**	**Models Accuracy Drop**
RFE	spectral bandwidth, Standard Deviation , skew,median, MFCC 5	20%
MRMR	spectral centroid, spectral bandwidth, mean, rolloff, kurt, skew	13%
Chi-2	spectral centroid, spectral bandwidth, spectral flatness, rolloff , skew, kurt	32%

Table 5.1: Features Selection Techniques Result.

In our implementation of all approaches, no features were eliminated or dropped, and training was done on all extracted features.

5.2 Machine Learning Approaches Results

5.2.1 SVM Model

As mentioned previously, we found that the linear kernel is the best to be used with our SVM model. By Applying *GridSearch* and hyper-parameter tuning we were able to fit the model on the best combination of parameters which resulted in the most precise predictions the model was able to give. The model was tested on three hundred samples and achieved 83% accuracy.

5.2.2 RF Model

By comparing the performance of both SVM and RF models, we found that RF offers better results in speaker classification. The best accuracy the model achieved was 94%.

5.3 Deep Learning Approaches Results

5.3.1 DNN results

Our DNN model was trained on 300 epochs, the two figures below shows the learning curves during training process. Figure 5.5 witnesses a very smooth loss process. The plot of both training loss and validation loss decreases to a point of stability. The generalization gap is minimal, so we can say that our model achieved an optimal fit. Figure 5.6 shows the accuracy of the model in each epoch, it's seen that no accuracy drop was encountered which reinforce the above conclusion that the model performance was optimal. The model was then tested on

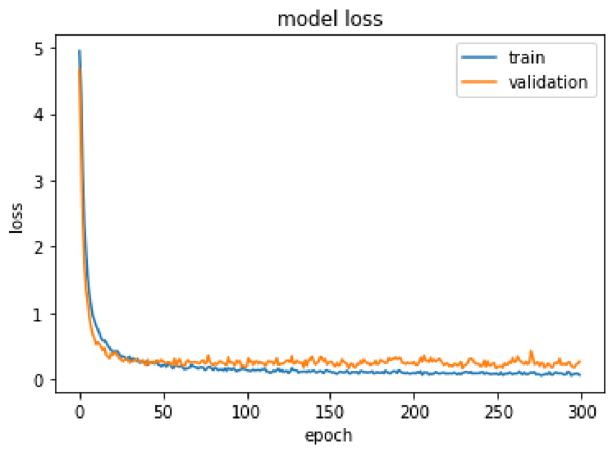

Figure 5.1: Loss curve.

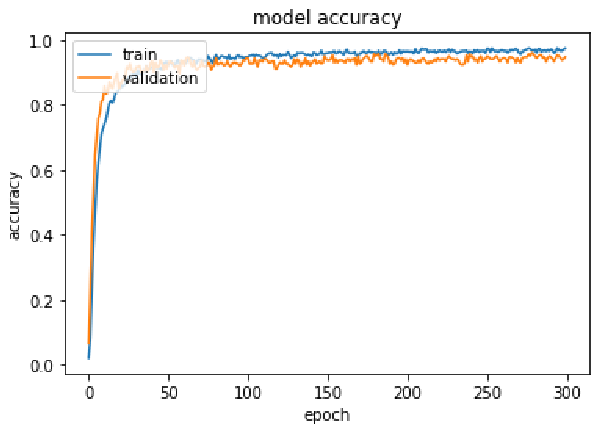

Figure 5.2: Accuracy curve.

10% of the samples that has never been heard by the model before. DNN gave 97% accuracy which is a promising accuracy given the lack of data we encountered.

5.3.2 LSTM results

LSTM model was trained on 400 epochs. The model started overfitting at $val\_acc = 77\%$. The problem with overfitting, is that the more specialized the model becomes to training data, the less well it is able to generalize to new data, resulting in an increase in generalization error. Again, this is due to the small number of samples and an unrepresentative validation samples.

Figures 5.3 and 5.4 shows the performance of the model during training and how it started overfitting with time. It can be seen that at some points in the first 50 epochs, the validation dataset was easier to predict compared to the validation dataset, this might have happened due to information leakage and the fact that raining samples contains features with less variance than the validation samples.

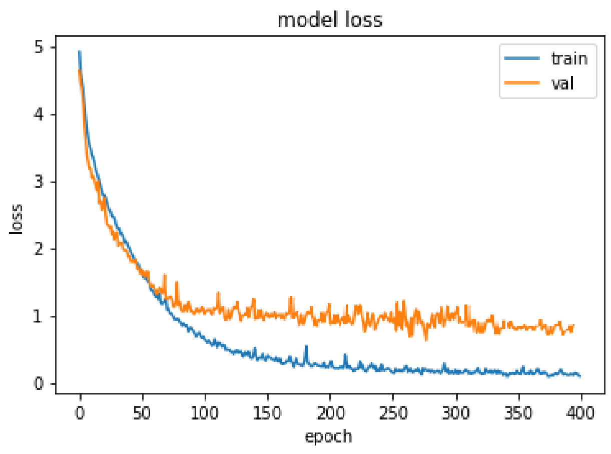

Figure 5.3: Loss curve.

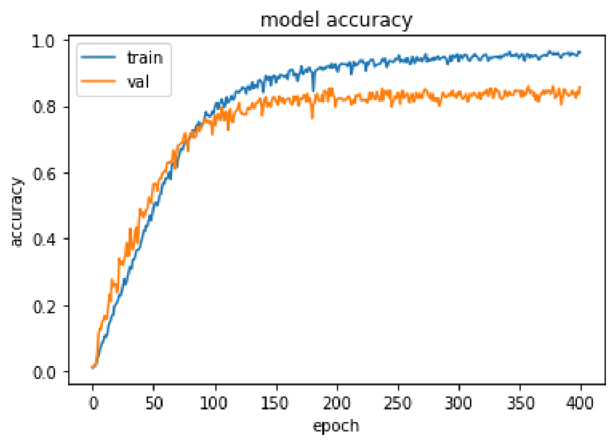

Figure 5.4: Accuracy curve.

Our model with two LSTM layers gave accuracy of 92% when predicting the test subset, which comes second in performance between the DL models.

5.3.3 CNN results

The CNN model was trained on 500 epochs. Looking at figure 5.5 which represents the loss curve of the model during training, it is noticeable that the training loss goes down over time, achieving low error values while the validation loss goes down until a stabilizing point is found, this indicates that the model started overfitting too, if more epochs where iterated then the gap would grow and so is overfitting.

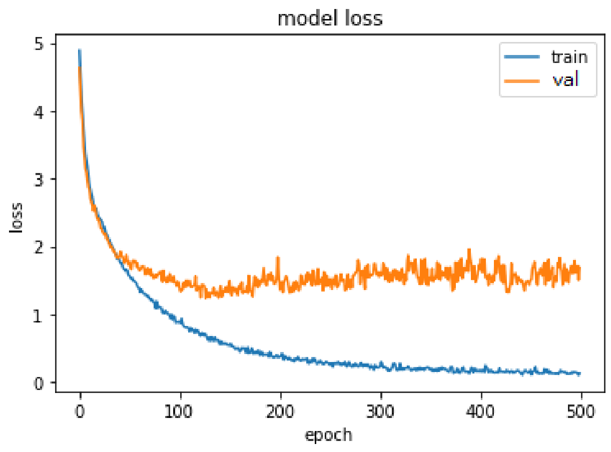

Figure 5.5: Loss curve.

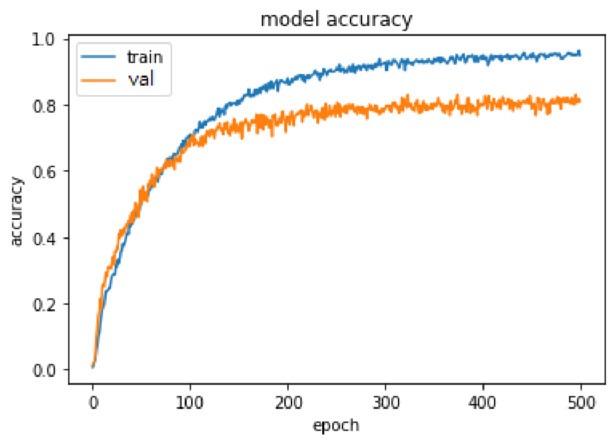

Figure 5.6: Accuracy curve.

CNN was one of the most complex models we had to work with regarding the input shape of the model and the overall hierarchy, yet it showed the lowest performance of all the other DL models. Testing and evaluating the model on the test dataset, it gave 83% accuracy which comes last in performance.

5.4 Comparison Of Different Approaches

As expected Deep Learning models gave higher accuracy and is always a more reliable approaches than Machine learning. It is to be mentioned that the lack of samples per speaker in our collected dataset dramatically affected the performance of these models, yet models like

DNN and LSTM came back with an interesting results. Even the RF model showed great performance for a ML model dealing with a very complex problem like voice recognition. The figure 5.7 summarises the performance of all models proposed in this project.

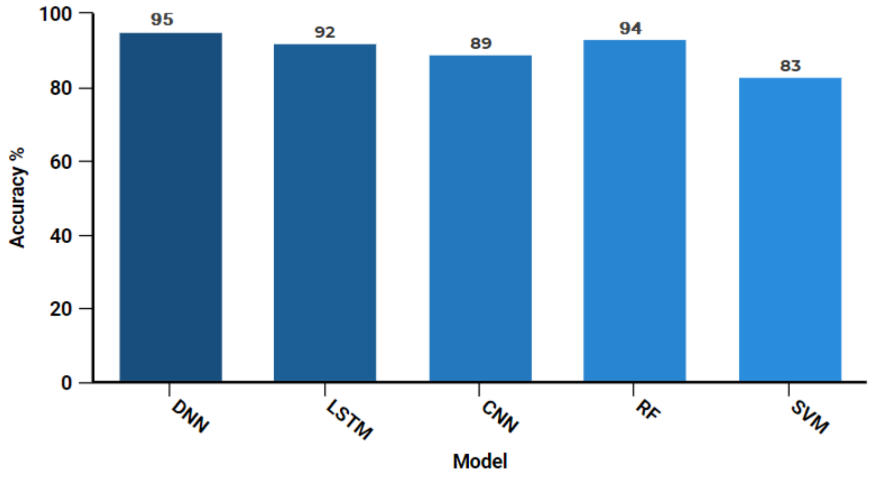

Figure 5.7: Accuracy of Proposed Models.

Chapter 6

CONCLUSION

Voice recognition has attracted scientists as an important domain and has created a technological impact on society that is expected to grow in this area of human-machine interaction. In this research, we tried to summarize the applied deep learning and machine learning practices in the field of voice recognition. We described the basic principles, methods, and analysis of issues that voice recognition faces. We've collected a large number of speaker samples, which were organized into sub-datasets and used to train, evaluate, and test the models on our own data. Different models were proposed in Machine Learning and Deep Learning, a comparison was constructed of the performance of each of these models, DNN model has achieved the best performance in terms of recognition accuracy and stability, then the LSTM-RNN model followed by the CNN model. Some of the lessons learned from this project are the importance of good designing and planning to provide reliability and to achieve that long hours of training and testing were required.

6.1 Future Work

One of the effecting limitations we encountered while training was the lack of samples collected, which resulted in overfitting the models. So, one of the main steps to take in the future is to collect more data from speakers, by increasing the number of samples in each sub-dataset, especially the Arabic sub-dataset to increase the stability of the models. For speech recognition to be deployed in real-world systems, maximum accuracy should be obtained and architectures of the discussed models should be developed and enhanced to reach the most satisfactory performance. Voice recognition systems have the potential to replace a lot of existing biometric authentication methods. One of the goals we aimed for while working on this project, is deploying our work in the JUST student services website, where student can access their accounts via their voices. Voice recognition systems have become increasingly complex and advanced over time. It won't be long until we can control, access, and work out every day's habits with only our voices.

Bibliography

References

[1] Homayoon Beigi (2011), "Fundamentals of Speaker Recognition", Springer-Verlag, Berlin, 2011, ISBN 978-0-387-77591-3.

[2] Vielhauer, C., et al. "Multimodal speaker authentication–evaluation of recognition performance of watermarked references." Proceedings of the 2nd workshop on multimodal user authentication (MMUA), Toulouse, France. 2006.

[3] https://www.sas.com/en_us/insights/analytics/machine-learning.html

[4] Chauhan, S., Vig, L., De Filippo De Grazia, M., Corbetta, M., Ahmad, S., & Zorzi, M. (2019). A Comparison of Shallow and Deep Learning Methods for Predicting Cognitive Performance of Stroke Patients From MRI Lesion Images. Frontiers in neuroinformatics, 13, 53. https://doi.org/10.3389/fninf.2019.00053

[5] A. Maity (2016). "Supervised Classification of RADARSAT-2 Polarimetric Data for Different Land Features". arXiv:1608.00501

[6] Hinton, Geoffrey; Sejnowski, Terrence (1999). Unsupervised Learning: Foundations of Neural Computation. MIT Press. ISBN 978-0262581684.

[7] Cortes, Corinna; Vapnik, Vladimir N. (1995). "Support-vector networks" (PDF). Machine Learning. 20 (3): 273–297. CiteSeerX 10.1.1.15.9362. doi:10.1007/BF00994018. S2CID 206787478.

[8] David A. Freedman (27 April 2009). Statistical Models: Theory and Practice. Cambridge University Press. ISBN 978-1-139-47731-4.

[9] Ho, Tin Kam (1995). Random Decision Forests (PDF). Proceedings of the 3rd International Conference on Document Analysis and Recognition, Montreal, QC, 14–16 August 1995. pp. 278–282. Archived from the original (PDF) on 17 April 2016. Retrieved 5 June 2016.

[10] Valueva, M.V.; Nagornov, N.N.; Lyakhov, P.A.; Valuev, G.V.; Chervyakov, N.I. (2020). "Application of the residue number system to reduce hardware costs of the convolutional neural network implementation"

[11] Flanagan, Speech Analysis, Synthesis and Perception, Springer- Verlag, New York, 1972

[12] Minsky, M. Steps toward Artificial Intelligence. Proc. IRE 1961, 46, 8–30.

[13] Jordan, M.I.; Mitchell, T.M. Machine learning: Trends, perspectives, and prospects. Science 2015, 349, 255–260.

[14] Khan, M.A.; Kim, Y. Deep Learning-Based Hybrid Intelligent Intrusion Detection System. Comput. Mater. Contin. 2021, 68, 671–687.

[15] Taylor S, Dromey C, Nissen SL, Tanner K, Eggett D, Corbin-Lewis K. Age-Related Changes in Speech and Voice: Spectral and Cepstral Measures. J Speech Lang Hear Res. 2020;63(3):647-660. doi:10.1044/2019_JSLHR-19-00028

[16] Kuchaiev, Oleksii and Li, Jason and Nguyen, Huyen and Hrinchuk, Oleksii and Leary, Ryan and Ginsburg, Boris and Kriman, Samuel and Beliaev, Stanislav and Lavrukhin, Vitaly and Cook, Jack and others in Nemo: a toolkit for building ai applications using neural modules, arXiv preprint arXiv:1909.09577, 2019.

[17] https://github.com/kaldi-asr/kaldi

[18] Bishop, Christopher (2006). Pattern recognition and machine learning. New York: Springer. ISBN 978-0-387-31073-2.

[19] NVIDIA, Vingelmann, P., Fitzek, F. H. P. (2020). CUDA, release: 10.2.89. Retrieved from https://developer.nvidia.com/cuda-toolkit

[20] Mirco Ravanelli, Titouan Parcollet, Peter Plantinga, Aku Rouhe, Samuele Cornell, Loren Lugosch in SpeechBrain: A General-Purpose Speech Toolkit. arXiv:2106.04624. 2021.

[21] V. Panayotov, G. Chen, D. Povey and S. Khudanpur, "Librispeech: An ASR corpus based on public domain audio books," 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)

[22] Nagrani, Arsha et al. “VoxCeleb: A Large-Scale Speaker Identification Dataset.” INTERSPEECH (2017).

[23] Fast Fourier Transforms, Connexions online book edited by Charles Sidney Burrus, with chapters by Charles Sidney Burrus, Ivan Selesnick, Markus Pueschel, Matteo Frigo, and Steven G. Johnson (2008)

[24] Shafranovich, Y. (October 2005). Common Format and MIME Type for CSV Files. IETF. p. 1. doi:10.17487/RFC4180. RFC 4180.

[25] Smith, Steven W. (1999). "Chapter 8: The Discrete Fourier Transform". The Scientist and Engineer’s Guide to Digital Signal Processing (Second ed.). San Diego, Calif.: California Technical Publishing

[26] Stanković, Radomir S.; Astola, Jaakko T. (2012). "Reminiscences of the Early Work in DCT: Interview with K.R. Rao"

[27] Anne M. Gilroy; Brian R. MacPherson; Lawrence M. Ross (2008). Atlas of anatomy. Thieme. p. 536. ISBN 978-1-60406-151-2.

[28] hhttps://research.google.com

[29] Denny Atkin. "Computer Shopper: The Right GPU for You". Archived from the original on 2007-05-06. Retrieved 2007-05-15.

[30] Coalson, Josh. "FLAC - format". Retrieved 4 April 2013. "fLaC", the FLAC stream marker in ASCII, meaning byte 0 of the stream is 0x66, followed by 0x4C 0x61 0x43"

[31] "Releases – numpy/numpy". Retrieved 8 February 2021 – via GitHub.

[32] "License – Package overview – pandas 1.0.0 documentation". pandas. 28 January 2020. Retrieved 30 January 2020.

[33] "Keras backends". keras.io. Retrieved 2018-02-23.

[34] Metz, Cade (November 9, 2015). "Google Just Open Sourced TensorFlow, Its Artificial Intelligence Engine". Wired. Retrieved

[35] F. Hassan and A. Javed, "Voice Spoofing Countermeasure for Synthetic Speech Detection," 2021 International Conference on Artificial Intelligence (ICAI), 2021, pp. 209-212, doi: 10.1109/ICAI52203.2021.9445238.

[36] Everitt B.S., Skrondal A. (2010), Cambridge Dictionary of Statistics, Cambridge University Press.

[37] "Claesen, Marc, and Bart De Moor. "Hyperparameter Search in Machine Learning." arXiv preprint arXiv:1502.02127

[38] [3] I. Goncalves; S. Pfeiffer; C. Montgomery (2008). Ogg Media Types. sec. 10. doi:10.17487/RFC5334. RFC 5334.

[39] 3GPP2 (2007). 3GPP2 File Formats for Multimedia Services (1st ed., Vol. 3). 3GPP2 C.S0050-B.

[40] T. M. Cover, "The best two independent measurements are not the two best," IEEE Transactions on Systems, Man, and Cybernetics, no. 1, pp.116–117, 1974.

[41] https://towardsdatascience.com/mrmr-explained-exactly-how-you-wished-someone-explained-to-you-9cf4ed27458b

[42] Bahzad T. Jijo ;Adnan M. Abdulazeez; Lawrence M. Ross,Vol. 02, no. 01, pp.20–28(2021)

[43] Yishi Zhang; Shujuan Li; Teng Wang; Zigang Zhang (2013). "Divergence-based feature selection for separate classes". Neurocomputing. 101 (4): 32–42. doi:10.1016/j.neucom.2012.06.036.

[44] Pearson, Karl (1895). "Contributions to the mathematical theory of evolution, II: Skew variation in homogeneous material". Philosophical Transactions of the Royal Society. 186: 343–414.

[45] https://medium.com/analytics-vidhya/confusion-matrix-accuracy-precision-recall-f1-score-ade299cf63cd

[46] https://ml-cheatsheet.readthedocs.io/en/latest/

[47] Zell, Andreas (1994). Simulation Neuronaler Netze [Simulation of Neural Networks] (in German) (1st ed.). Addison-Wesley. p. 73. ISBN 3-89319-554-8.

[48] Robert Hirsch (2004). Exploring Colour Photography: A Complete Guide. Laurence King Publishing. ISBN 1-85669-420-8.

[49] Han, Jun; Morag, Claudio (1995). "The influence of the sigmoid function parameters on the speed of backpropagation learning". In Mira, José; Sandoval, Francisco (eds.). From Natural to Artificial Neural Computation. Lecture Notes in Computer Science. 930. pp

[50] McFee, Brian, Colin Raffel, Dawen Liang, Daniel PW Ellis, Matt McVicar, Eric Battenberg, and Oriol Nieto. "librosa: Audio and music signal analysis in python." In Proceedings of the 14th python in science conference, pp. 18-25. 2015.

[51] Croxton, Frederick Emory; Cowden, Dudley Johnstone; Klein, Sidney (1968) Applied General Statistics, Pitman. ISBN 9780273403159

[52] "Speaker Independent Connected Speech Recognition- Fifth Generation Computer Corporation". Fifthgen.com. Archived from the original on 11 November 2013. Retrieved 15 June 2013.

[53] [15] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.cosine_similarity.htm

[54] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman, "Return of the devil in the details: Delving deep into convolutional nets," arXiv preprint arXiv:1405.3531, 2014.

[55] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization", arXiv:1412.6980, 2014, [online] Available: https://arxiv.org/abs/1412.6980.

APPENDICES

Appendix A

Additional Information and definitions about the constructed models

Referring to the implementation chapter, we present the developed design for each model in detail and explain the algorithms, techniques, and script used for each step. The scripts that have not been discussed yet will be presented in Appendices.

A.0.1 Features Correlation

The strength of the relationship between features is found using the following script 13, which presents how the correlation is done.

```python
    corr = feature.corr()
    columns = np.full((corr.shape[0],), True, dtype=bool)
    for i in range(corr.shape[0]):
        for j in range(i+1, corr.shape[0]):
            if corr.iloc[i,j] >= 0.9:
                if columns[j]:
                  columns[j] = False
    selected_columns = features.columns[columns]
    Features_After_Corr = features[selected_columns]
    data_after_Corr = pd.DataFrame()
    data_after_Corr = pd.DataFrame(data = Features_After_Corr.values ,
                                   columns = selected_columns)
    data_after_Corr['id'] = IDs
```

Listing 13: Features correlation

A.0.2 Minimum Redundancy Maximum Relevance

This snippet presents how the MRMR algorithm is employed and shows the chosen parameters discussed previously.

```
!pip install pymrmr
import pandas as pd
import pymrmr
selected_columns=pymrmr.mRMR(df, 'MIQ',6)
```

Listing 14: MRMR

A.0.3 Chi-2 algorithm

```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2

X = data.values[:,:-1]
Y = data.values[:,-1:]
select_model = SelectKBest(score_func=chi2, k=10)
fit = select_model.fit(X, Y)
chi2_features_values = fit.transform(X)
```

Listing 15: Chi-2

A.0.4 Support Vector Machine

We found the best kernel to be used with SVM by the following code.

```
kernels = ['Polynomial', 'RBF', 'Sigmoid','Linear']
#A function which returns the corresponding SVC model
def getClassifier(ktype):
    if ktype == 0:
        # Polynomial kernal
        return SVC(kernel='poly', degree=8, gamma="auto")
    elif ktype == 1:
        # Radial Basis Function kernal
        return SVC(kernel='rbf', gamma="auto")
    elif ktype == 2:
        # Sigmoid kernal
        return SVC(kernel='sigmoid', gamma="auto")
    elif ktype == 3:
        # Linear kernal
        return SVC(kernel='linear', gamma="auto"


with open("Kernels_Report.txt", "w") as f:
  for i in range(4):
      # Separate data into test and training sets
      X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.05)
      # Train a SVC model using different kernal
      svclassifier = getClassifier(i)
      svclassifier.fit(X_train, y_train)# Make prediction
      y_pred = svclassifier.predict(X_test)# Evaluate our model

      kernal_name = "Evaluation:", kernels[i], "kernel"
      report = classification_report(y_test,y_pred)
      print(kernal_name, file=f)
      print(report, file=f)
  f.close()
```

Listing 16: SVM

A.0.5 Random Forest

The script 17 below shows how Rf model is worked, fitted, and evaluated.

```python
from sklearn.ensemble import RandomForestClassifier
clf=RandomForestClassifier(n_estimators= 2000)
#Train the model using the training sets y_pred=clf.predict(X_test)
clf.fit(X_train,y_train)
# prediction on test set
y_pred=clf.predict(X_test)
n_estimators=[int(x) for x in np.linspace(start= 200,stop= 2000,num= 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]
random_grid = {'n_estimators': n_estimators,
           'max_features': max_features,
           'max_depth': max_depth,
           'min_samples_split':   min_samples_split,
           'min_samples_leaf': min_samples_leaf,
           'bootstrap': bootstrap}
# First create the base model to tune
rf = RandomForestClassifier()
rf_random = RandomizedSearchCV(estimator = rf, param_distributions =
random_grid, n_iter= 100, cv= 3, verbose=2, random_state=42, n_jobs = -1)
# Fit the random search model
rf_random.fit(X_train, y_train)
```

Listing 17: RF model

A.0.6 Defining the model imports

This section presents the first step in defining and building any model. This script 19 shows some dependencies and libraries required to construct all examined models.

```python
#LSTM and DNN imports
import keras
import librosa
import pandas as pd
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import normalize
import warnings
from tensorflow.keras.layers import LSTM, Dense, Dropout, Activation
from keras.models import Sequential
from keras.layers import Activation
from sklearn.preprocessing import LabelEncoder
from keras.utils.np_utils import to_categorical
from sklearn.preprocessing import StandardScaler
from keras.callbacks import EarlyStopping
from numpy import sqrt


#CNN imports
from keras.layers import Conv1D, GlobalAveragePooling1D, MaxPooling1D
from keras_preprocessing.image import ImageDataGenerator
from keras.layers import  BatchNormalization
from keras.layers import Conv2D, MaxPooling2D
from keras import regularizers, optimizers
from keras import regularizers
```

Listing 18: Model imports

A.0.7 CNN spectrogram

```python
def images(files):

    # We define the audiofile from the rows of the dataframe when we iterate through
    # every row of our dataframe for train, val and test
    audiofile = str(files.path)
    # Loading the image with no sample rate to use the original sample rate
    X, sample_rate = librosa.load(audiofile, sr=None, res_type='kaiser_fast')
    # Setting the size of the image
    fig = plt.figure(figsize=[1,1])
    # This is to get rid of the axes and only get the picture
    ax = fig.add_subplot(111)
    ax.axes.get_xaxis().set_visible(False)
    ax.axes.get_yaxis().set_visible(False)
    ax.set_frame_on(False)
    # This is the melspectrogram from the decibels with a linear relationship
    # Setting min and max frequency to account for human voice frequency
    S = librosa.feature.melspectrogram(y=X, sr=sample_rate)
    librosa.display.specshow(librosa.power_to_db(S, ref=np.max),
    x_axis='time',y_axis='mel', fmin=50, fmax=280)

    name = files.file
    file  = '/content/images/' + str(name) + '.jpg'
    # Here we finally save the image file choosing the resolution
    plt.savefig(file, dpi=500, bbox_inches='tight',pad_inches=0)

    # Here we close the image because otherwise we get a
    #warning saying that the image stays
    # open and consumes memory
    plt.close()
```

Listing 19: Function to generate a spectrogram