CHAPTER 1

INTRODUCTION

1.1 SPEECH

Speech is human vocal communication using language. Each language uses phonetic combinations of a limited set of perfectly articulated and individualized vowel and consonant sounds that form the sound of its words. In speaking, speakers perform many different intentional speech acts like informing, declaring, asking, persuading, directing, and can use enunciation, intonation, degrees of loudness, tempo, and other non-representational or paralinguistic aspects of vocalization to convey meaning. In their speech, speakers also unintentionally communicate many aspects of their social position such as sex, age, place of origin, physical states, psychic states, education or experience, and the like. Even though it sounds simple, the acts of speaking and listening have a lot of nuances and comprises of various hidden processes that are occurring to achieve necessary results. Yet the human brain is capable of doing all that behind the scenes with virtually no difficulty and at a very fast pace at that.

1.2 SPEAKER IDENTIFICATION AND VERIFICATION

Technologies relating to speech have become ubiquitous today. This can be understood from the presence and the evolution of various intelligent

personal assistants (IPA) devices/software in the market such as Apple's Siri, Microsoft's Cortana, Amazon's Alexa and the Google Assistant. It is essential that the IPA knows the speaker's identity while communicating with humans. Because recognizing a person's identity is crucial for applications such as financial transactions, forensics, law enforcement, library management systems, etc.

Any machine invented has roots relating to human activities and aims to make jobs easier for people by doing said activities in a quick and an efficient manner. The speech technologies are no exception to this phenomenon and have evolved so much so as they take a small form factor of smart phones nowadays. Interactive Voice Response (IVR) systems are beginning to get replaced by speech recognition based interactive systems. Alongside with speech recognition systems a new technology relating to speech soon piqued the interests of many curious computer scientists, namely speaker recognition system. Speaker recognition aims to analyze and extract information from the speech signal to recognize the speaker's identity. Speaker recognition is generally classified into speaker identification and speaker verification.

Speaker identification is the process of determining the speaker's identity in an utterance from a given set of speakers. A test utterance has to be tested against all the speakers in the set. Since the testing is confined to a set of speakers, this process is also called as closed-set identification. A closed-set identification system can also be made into an open-set identification system by adding the "none of the enrolled speakers" option as a decision. Figure 1.1 shows the process of speaker identification.

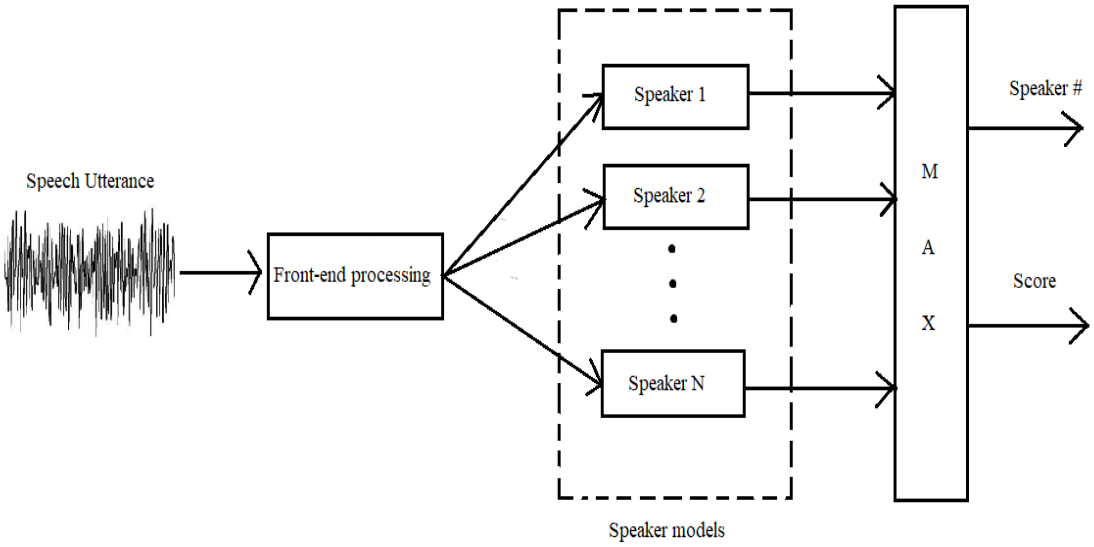

Figure 1.1 Speaker Identification

In the case of Speaker Verification, each test utterance comes with a claim. The task of this system is to identify whether a given speech utterance belongs to the claimed speaker or not. The speaker verification system is usually referred to as the open-set identification system because the test utterance may not even belong to any of the enrolled speakers. Figure 1.2 illustrates the working of a speaker verification system.

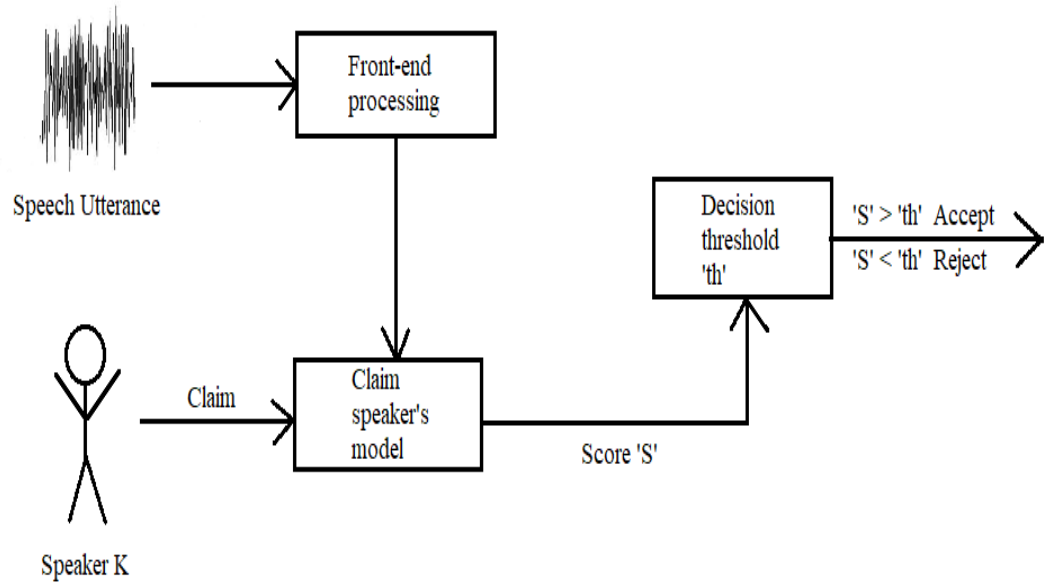

Figure 1.2 Speaker Verification

1.3 SPEECH PROCESSING

Speech processing is the study of speech signals and the processing methods of signals. The signals are usually processed in a digital representation, so speech processing can be regarded as a special case of digital signal processing, applied to speech signals. Aspects of speech processing include the acquisition, manipulation, storage, transfer and output of speech signals. The input is called speech recognition and the output is called speech synthesis.

There are a few techniques that helps process speech and they are Dynamic Time Warping, Hidden Markov models and Artificial Neural Networks. Dynamic time warping (DTW) is an algorithm for measuring similarity between two temporal sequences, which may vary in speed. In general, DTW is a method that calculates an optimal match between two given sequences with certain restriction and rules. The optimal match is denoted by the match that satisfies all the restrictions and the rules and that has the minimal cost, where the cost is computed as the sum of absolute differences, for each matched pair of indices, between their values.

A hidden Markov model can be represented as the simplest dynamic Bayesian network. The goal of the algorithm is to estimate a hidden variable x(t) given a list of observations y(t). By applying the Markov property, the conditional probability distribution of the hidden variable x(t) at time t, given the values of the hidden variable x at all times, depends only on the value of the hidden variable x(t − 1). Similarly, the value of the observed variable y(t) only depends on the value of the hidden variable x(t) (both at time t).

An artificial neural network (ANN) is based on a collection of connected units or nodes called artificial neurons, which loosely model the neurons in a biological brain. Each connection, like the synapses in a biological brain, can transmit a signal from one artificial neuron to another. An artificial neuron that receives a signal can process it and then signal additional artificial neurons connected to it. In common ANN implementation, the signal at a connection between artificial neurons is a real number and the output of each artificial neuron is computed by some non-linear function of the sum of its inputs.

1.4 MACHINE LEARNING

Machine learning is an application of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed. Machine learning focuses on the development of computer programs that can access data and use it to learn for themselves.

The process of learning begins with observations or data, such as examples, direct experience, or instruction, in order to look for patterns in data and make better decisions in the future based on the examples that we provide. The primary aim is to allow the computers to learn automatically without human intervention or assistance and adjust actions accordingly.

Machine learning enables analysis of massive quantities of data. While it generally delivers faster, more accurate results in order to identify profitable opportunities or dangerous risks, it may also require additional time and resources to train it properly. Combining machine learning with AI and

cognitive technologies can make it even more effective in processing large volumes of information.

The thing about traditional Machine Learning algorithms is that as complex as they may seem, they're still machine like. They need lot of domain expertise, human intervention only capable of what they're designed for; nothing more, nothing less. For AI designers and the rest of the world, that's where deep learning holds a bit more promise.

1.5 DEEP LEARNING

Deep Learning (also known as deep structured learning or hierarchical learning) is part of a broader family of machine learning methods based on learning data representations, as opposed to task-specific algorithms. Learning can be supervised, semi-supervised or unsupervised.

Practically, Deep Learning is a subset of Machine Learning that achieves great power and flexibility by learning to represent the world as nested hierarchy of concepts, with each concept defined in relation to simpler concepts, and more abstract representations computed in terms of less abstract ones.

Deep Learning is preferred over Machine Learning nowadays due to the massive amounts of data which is available to us which can be used more efficiently by Deep Learning algorithms as compared to Machine Learning algorithms to achieve better results.

Deep Learning models are vaguely inspired by information processing and communication patterns in biological nervous systems yet have various differences from the structural and functional properties of biological brains (especially human brains), which make them incompatible with neuroscience evidences.

1.5.1 Neural Networks

The original goal of the neural network approach was to solve problems in the same way that a human brain would. Over time, attention focused on matching specific mental abilities, leading to deviations from biology such as back-propagation, or passing information in the reverse direction and adjusting the network to reflect that information.

Artificial neural networks (ANNs) or connectionist systems are computing systems inspired by the biological neural networks that constitute animal brains. Such systems learn (progressively improve their ability) to do tasks by considering examples, generally without task-specific programming. For example, in image recognition, they might learn to identify images that contain cats by analyzing example images that have been manually labeled as "cat" or "no cat" and using the analytic results to identify cats in other images. They have found most use in applications difficult to express with a traditional computer algorithm using rule-based programming.

An ANN is based on a collection of connected units called artificial neurons, (analogous to biological neurons in a biological brain). Each connection (synapse) between neurons can transmit a signal to another neuron. The receiving (postsynaptic) neuron can process the signal(s) and

then signal downstream neurons connected to it. Neurons may have state, generally represented by real numbers, typically between 0 and 1. Neurons and synapses may also have a weight that varies as learning proceeds, which can increase or decrease the strength of the signal that it sends downstream.

A Deep Neural Network (DNN) is an Artificial Neural Network (ANN) with multiple layers between the input and output layers. The DNN finds the correct mathematical manipulation to turn the input into the output, whether it is a linear relationship or a non-linear relationship. The network moves through the layers calculating the probability of each output. The user can review the results and select which probabilities the network should display (above a certain threshold, etc.) and return the proposed label. Each mathematical manipulation as such is considered a layer, and complex DNN have many layers, hence the name "deep" networks. The goal is that eventually, the network will be trained to decompose an image into features, identify trends that exist across all samples and classify new images by their similarities without requiring human input.

DNNs are typically feed-forward networks in which data flows from the input layer to the output layer without looping back. At first, the DNN creates a map of virtual neurons and assigns random numerical values, or "weights", to connections between them. The weights and inputs are multiplied and return an output between 0 and 1. If the network didn’t accurately recognize a particular pattern, an algorithm would adjust the weights. That way the algorithm can make certain parameters more influential, until it determines the correct mathematical manipulation to fully process the data.

1.6 APPROACHING THE TASK OF SPEAKER RECOGNITION

In this section we give a brief overview of approaches to the task of speaker recognition and then explain the workings behind them.

1.6.1 Convolutional Neural Networks

In deep learning, a convolutional neural network (CNN or ConvNet) is a class of deep neural networks, most commonly applied to analyzing visual imagery. CNNs use a variation of multilayer perceptron designed to require minimal preprocessing. They are also known as shift invariant or space invariant artificial neural networks (SIANN), based on their shared-weights architecture and translation invariance characteristics.

Convolutional networks were inspired by biological processes in that the connectivity pattern between neurons resembles the organization of the animal visual cortex. Individual cortical neurons respond to stimuli only in a restricted region of the visual field known as the receptive field. The receptive fields of different neurons partially overlap such that they cover the entire visual field. CNNs use relatively little pre-processing compared to other image classification algorithms. This means that the network learns the filters that in traditional algorithms were hand-engineered. This independence from prior knowledge and human effort in feature design is a major advantage. They have applications in image and video recognition, recommender systems, image classification, medical image analysis, and natural language processing.

1.6.2 Spectrogram

A spectrogram is a visual representation of the spectrum of frequencies of a signal as it varies with time. When applied to an audio signal, spectrograms are sometimes called sonographs, voiceprints, or voicegrams. When the data is represented in a 3D plot they may be called waterfalls. Spectrograms are used extensively in the fields of music, sonar, radar, and speech processing, seismology, and others. Spectrograms of audio can be used to identify spoken words phonetically, and to analyze the various calls of animals.

A spectrogram can be generated by an optical spectrometer, a bank of band-pass filters, by Fourier transform or by a wavelet transform. A spectrogram is usually depicted as a heat map, i.e., as an image with the intensity shown by varying the color or brightness.

1.7 ISSUES AND CHALLENGES

Based on the literature survey, several existing issues and challenges that might hinder the progress have been identified. The findings and potential solutions are listed as follows.

1.7.1 Propagation of Errors

Since using networks which are highly sensitive to hyperparameters, any small change in either network leads to a massive change in the results. This is a major problem for which there is no real solution apart from better computational capabilities.

1.7.2 Training Time

Takes a long time to train even a single epoch. The CNN architecture trained on the VoxCeleb dataset requires about 500 epochs of training to achieve decent results. Low dataset models on the other hand train much faster but have a considerably worse performance. Due to the lack of computational power and time, the use of a decently sized dataset and slight tweaks to the epoch and batch size have been made which helped achieve a significant measure of performance.

1.7.3 Preprocessing

Even though the dataset from the VoxCeleb dataset is mostly free from noise and silence, they are still not perfect as they are ultimately voice samples of celebrities taken from various events and interviews. This poses as an even bigger obstacle for the custom dataset. Preprocessing techniques are not easy to use. A software is used to achieve a somewhat decently preprocessed set of audio files to work with.

1.7.4 Computational Overheads

The CNN architecture model requires a lot of epochs to produce any decent results as mentioned above in Training Time. The batch size and the epoch values can be changed around to meet optimal requirement but this drastically affects the performance of the model.

1.7.5 Resolution of Spectrogram

The resolution of the generated spectrograms plays a huge role in performance analysis of the model, especially in terms of accuracy. This once again boils down to the Training Time and Computational Overheads problems that are mentioned above indirectly.

CHAPTER 2

LITERATURE SURVEY

2.1 HYBRID METHOD FOR SPEAKER RECOGNITION

Z. Liu, Z. Wu, T. Li, J. Li and C. Shen (2018) state that a critical method, the Gaussian Mixture Model (GMM) makes it possible to achieve the recognition capability that is close to the hearing ability of human in a long speech. However, the GMM is failing to recognize a short utterance speaker with a high accuracy. Aiming at solving this problem, this work proposes a novel model to enhance the recognition accuracy of the short utterance speaker recognition system. Different from traditional models based on the GMM, this work designs a method to train a Convolutional Neural Network (CNN) to process spectrograms, which can describe speakers better. Thus, the recognition system gains the considerable accuracy as well as the reasonable convergence speed. The experiment results show that our model can help to decrease the equal error rate of the recognition from 4.9% to 2.5%.

The current speaker verification system can achieve a fairly good recognition rate only when the testing utterances are long enough. Considering the instability of the acquisition of short utterance features and the possible shortage of training samples, we focus on how to design the

voiceprint model and deep-learning model for the short utterance recognition to overcome the above issue.

This work puts forward speech recognition model based on phono-logical map deep learning. This model can exploit the advantage of deep CNN and extract feature maps from the speech images, which has a higher recognition rate.

2.2 TEXT-INDEPENDENT SPEAKER VERIFICATION

In this work by A. Torfi, J. Dawson and N. M. Nasrabadi (2018), a novel method using 3D-CNN architecture has been proposed for speaker verification in the text-independent setting. Most of the previously-reported approaches create speaker models based on averaging the extracted features from utterances of the speaker, which is known as the d-vector system. In this work, proposes an adaptive feature learning by utilizing the 3D-CNNs for direct speaker model creation in which, for both development and enrollment phases, an identical number of spoken utterances per speaker is fed to the network for representing the speakers' utterances and creation of the speaker model. Demonstrating that the proposed method significantly outperforms the traditional d-vector verification system. Moreover, the proposed system can also be an alternative to the traditional d-vector system which is a one-shot speaker modeling system by utilizing 3D-CNNs.

In this experimental setup investigates the effect of frame-level or utterance-level representation. For the utterance-level, the entire input feature map will be fed to the network, but in the frame-level, the weight update will be performed per frame of input, which can belong to any speaker with the

available class label. Moreover, we compare our results with the traditional i-vector system as well as the state-of-the-art in text-dependent speaker verification. The method presented, trains the system in an end-to-end fashion using LSTM recurrent neural networks in which no enrolment stage is required. In our experiments in the text independent setting, the proposed method outperforms the end-to-end training fashion.

2.3 SPEAKER IDENTIFICATION AND CLUSTERING

For speaker clustering, however, it is still common to use handcrafted processing chains such as MFCC features and GMM-based models. This work by Y. Lukic, C. Vogt, O. Dürr and T. Stadelmann (2016) uses simple spectrograms as input to a CNN and study the optimal design of those networks for speaker identification and clustering. Demonstrating our approach on the well-known TIMIT dataset, achieving results comparable with the state of the art– without the need for handcrafted features.

The TIMIT dataset contains studio quality recordings of 630 speakers (192 female, 438 male), sampled at 16kHz, covering the eight major dialects of American English. Each speaker reads ten phonetically rich sentences, from which we use six for training, two for validation, and two for testing. Training CNNs to perform speaker identification on a subset of 100 speakers as well as on the full dataset.

The work is carried out as three experiments: First, we examine the optimal convolutional filter dimension. Second, we evaluate the speaker identification performance of our network on the whole TIMIT test set using

the optimal convolutional filters. Third, we evaluate the performance on the clustering task using the pre-trained speaker identification network.

2.4 SPEAKER RECOGNITION WITH MINIMAL TRAINING DATA

M. Wang, T. Sirlapu, A. Kwasniewska, M. Szankin, M. Bartscherer and R. Nicolas (2018) state that improved normalization functions and direction feature extraction strategies and will compare their performance with existing techniques. They compare ten normalization functions (seven based on dimensions and three based on moments) and eight feature vectors on three distinct data sources. The normalization functions and feature vectors are combined to produce eighty classification accuracies to each dataset. The comparison of normalization functions shows that moment-based functions outperform the dimension-based ones and the aspect ratio mapping is influential. The comparison of feature vectors shows that the improved feature extraction strategies outperform their baseline counterparts. The gradient feature from grey-scale image mostly yields the best performance and the improved NCFE features also perform well.

The work is concentrated on comparing relatively small CNN and evaluate effectiveness of speaker recognition using these models on edge devices. The preliminary results proved that the chosen model adapts the benefit of computer vision task by using CNN and spectrograms to perform speaker classification with precision and recall ~84% in time less than 60 ms on mobile device with Atom Cherry Trail processor.

The conducted research was focused on evaluating various DNN models used for recognizing speakers using spectrograms generated from

audio files on mobile devices. From preliminary results, we observe that simple and relatively shallow (2 and 3 convolutional layers) CNNs achieve decent accuracy (precision and recall -84%) with minimal training data (300 samples per speaker) and they are feasible to run on edge devices, because of their small memory footprint (3.5 MB) and short inference time.

2.5 VOXCELEB: A LARGE-SCALE SPEAKER IDENTIFICATION DATASET

This work makes two contributions: First, this work proposes a fully automated pipeline based on computer vision techniques to create the dataset from open-source media. Our pipeline involves obtaining videos from YouTube; performing active speaker verification using a two-stream synchronization CNN, and confirming the identity of the speaker using CNN based facial recognition. Further using this pipeline to curate VoxCeleb which contains hundreds of thousands of 'real world' utterances for over 1,000 celebrities.

The second contribution is to apply and compare various state of the art speaker identification techniques on our dataset to establish baseline performance. This is to show that a CNN based architecture obtains the best performance for both identification and verification.

RESULTS: For identification the above method achieves an 80.5% top-1 classification accuracy over 1,251 different classes, almost 20% higher than traditional state of the art baselines. The CNN architecture uses the average pooling layer for variable length test data. Also comparing two variants: 'CNN-fc-3s', this architecture has a fully connected fc6 layer, and divides the

test data into 3s segments and averages the scores. As is evident there is a considerable drop in performance compared to the average pooling original – partly due to the increased number of parameters that must be learnt; 'CNN-fc-3s no var. norm.', this is the CNN-fc-3s architecture without the variance normalization pre-processing of the input (the input is still mean normalized). The difference in performance between the two shows the importance of variance normalization for this data.

Table 2.1 Results for Speaker Identification on VoxCeleb Dataset

Accuracy	Top-1 (%)	Top-5 (%)
I-vectors + SVM	49.0	56.6
I-vectors + PLDA + SVM	60.8	75.6
CNN-fc-3s no var. norm.	63.5	80.3
CNN-fc-3s	72.4	87.4
CNN	**80.5**	**92.1**

2.6 SUPERVISED SPEECH SEPARATION

A more recent approach formulates speech separation as a supervised learning problem, where the discriminative patterns of speech, speakers, and background noise are learned from training data. This work first introduces the background of speech separation and the formulation of supervised separation. Then discusses about three main components of supervised separation: learning machines, training targets, and acoustic features.

This work by D. Wang and J. Chen (2018), has provided a comprehensive overview of DNN based supervised speech separation. This work has also summarized key components of supervised separation, i.e. learning machines, training targets, and acoustic features, explained

representative algorithms, and reviewed a large number of related studies. With the formulation of the separation problem as supervised learning, DNN based separation over a short few years has greatly elevated the state-of-the-art for a wide range of speech separation tasks, including monaural speech enhancement, speech dereverberation, and speaker separation, as well as array speech separation.

2.7 MFCC AND SIMILARITY MEASUREMENTS

This work was carried out by A. Maazouzi, N. Aqili, A. Aamoud, M. Raji and A. Hammouch (2017). Almost all of speaker identification systems are based on distance computation or likelihood. Accuracy of identification process depends on: (i) the number of feature vectors, (ii) their dimensionality, and (iii) the number of speakers. This work aims to develop a system able to identify a person from a sample of his speech. Recognition relies on a text-dependent system using English words as a password. Speech features are extracted using MFCCs. The recognition is based on discrete to continuous algorithm.

In the proposed speaker recognition system, the signal is preprocessed using endpoint detection algorithm and then we used the discrete wavelet transform to generate the approximation coefficients of the speech signal. In the feature extraction step, the MFCC method is applied to these approximation coefficients to determine the speech features. Finally, the input signal is compared with the stored models using similarity measurements and then the most similar model is chosen using the discrete to continuous algorithm.

Based on the used database, the recognition rate exceeded 95%. It is believed that in addition of the accuracy obtained by the system, the time running is also optimized by using the mean value of MFCCs. Also, more training data and good preprocessing methods can increasingly enhance the accuracy of the system.

2.8 EFFICIENT WINDOW IN MFCC

This work is based on an experimental evaluation of the various windowing techniques using MFCC for monolingual and cross lingual speaker identification performed by B. G. Nagaraja and H. S. Jayanna (2013). The set of windows presented here allows a tradeoff between main lobe bandwidth and side lobe ripple decay. The set of windows presented here allows a tradeoff between main lobe bandwidth and side lobe ripple decay. The speaker identification study is conducted using randomly selected 50 speakers from IITG Multi-variability speaker recognition (IITG-MV) database, MFCC feature and GMM-UBM classifier.

FEATURES: Speech recordings were re-sampled to 8 kHz with 16 bits resolution and pre-emphasized (0.97). Frame duration of 20 msec and a 10 msec for overlapping are considered. A Mel-warping is performed using 22 triangular band pass filters followed by DCT. A 13- dimensional MFCC feature vectors are finally obtained (excluding 0th coefficient).

CLASSIFIER: Speech recordings were re-sampled to 8 kHz with 16 bits resolution and pre-emphasized (0.97). Frame duration of 20 msec and a 10 msec for overlapping are considered. A Mel-warping is performed using

22 triangular band pass filters followed by DCT. A 13- dimensional MFCC feature vectors are finally obtained (excluding 0th coefficient).

2.9 SPEECH EMOTION RECOGNITION

This work by A. M. Badshah, J. Ahmad, N. Rahim and S. W. Baik (2017), presents a method for speech emotion recognition using spectrograms and deep convolutional neural network (CNN). Spectrograms generated from the speech signals are input to the deep CNN. The proposed model consisting of three convolutional layers and three fully connected layers extract discriminative features from spectrogram images and outputs predictions for the seven emotions. This study trained the proposed model on spectrograms obtained from Berlin emotions dataset. The proposed framework attempts to utilize feature learning schemes for spectrograms generated from speech.

The proposed CNN model, consist of three convolutional layers, three fully connected layers and a SoftMax layer. The input of the network is a 256 x 256 spectrogram generated from emotional speech signals. The initial convolutional layers extract features from these spectrograms using convolution operations. Layer C1 has 120 (11 x 11) kernels which are applied at a stride setting of 4 pixels. It is followed by rectified linear units (ReLU) and a max pooling layer of size 3 x 3 with stride 2. ReLU act as activation functions instead of the typical sigmoid functions which improves efficiency of the training process. Layer C2 has 256 kernels of size 5 x 5 and they are applied on the input with a stride 1. Similarly, C3 has 384 kernels of size 3 x 3. Each of these conv. layers are followed by ReLU units. Layer C3 is followed by three FC layers having 2048, 2048, and 7 neurons, respectively.

In order to avoid overfitting, the first two FC layers are followed by dropout layers having a dropout ratio of 50%.

Results reveal that the proposed framework is capable of correctly predicting most of the emotions by generating high confidence outputs more than 50% of the time. In case of fear and happy emotions, the performance is acceptable but not very good because some of the spectrograms are confused other emotions. In all the cases mean prediction value for all the emotions was greater than the other emotions. Overall, the proposed method helped us achieve an accuracy of 84.3% for all the speakers on the test set.

CHAPTER 3

PROPOSED WORK

3.1 INTRODUCTION

The whole system is written mainly in python. After breaching through various approaches and different libraries, python_speech_features and LibROSA are chosen. Our main methodology involves training the model using Spectrogram.

3.1.1 Convolutional Neural Network

A convolutional neural network consists of an input and an output layer, as well as multiple hidden layers. The hidden layers of a CNN typically consist of convolutional layers, RELU layer i.e. activation function, pooling layers, fully connected layers and normalization layers. Description of the process as a convolution in neural networks is by convention but mathematically, it is a cross-correlation rather than a convolution. This only has significance for the indices in the matrix, and thus which weights are placed at which index.

The convolutional layer is the core building block of a CNN. The layer's parameters consist of a set of learnable filters (or kernels), which have a small receptive field, but extend through the full depth of the input volume.

During the forward pass, each filter is convolved across the width and height of the input volume, computing the dot product between the entries of the filter and the input and producing a 2-dimensional activation map of that filter. As a result, the network learns filters that activate when it detects some specific type of feature at some spatial position in the input. Stacking the activation maps for all filters along the depth dimension forms the full output volume of the convolution layer. Every entry in the output volume can thus also be interpreted as an output of a neuron that looks at a small region in the input and shares parameters with neurons in the same activation map.

Another important concept of CNNs is pooling, which is a form of non-linear down-sampling. There are several non-linear functions to implement pooling among which max pooling is the most common. It partitions the input image into a set of non-overlapping rectangles and, for each such sub-region, outputs the maximum. Intuitively, the exact location of a feature is less important than its rough location relative to other features. This is the idea behind the use of pooling in convolutional neural networks. The pooling layer serves to progressively reduce the spatial size of the representation, to reduce the number of parameters, memory footprint and amount of computation in the network, and hence to also control over fitting. It is common to periodically insert a pooling layer between successive convolutional layers in CNN architecture. The pooling operation provides another form of translation invariance. The pooling layer operates independently on every depth slice of the input and resizes it spatially. The most common form is a pooling layer with filters of size 2×2 applied with a stride of 2 down samples at every depth slice in the input by 2 along both width and height, discarding 75% of the activations. In this case, every max operation is over 4 numbers. The depth dimension remains unchanged.

In addition to max pooling, pooling units can use other functions, such as average pooling or ℓ2-norm pooling. Average pooling was often used historically but has recently fallen out of favor compared to max pooling, which performs better in practice. "Region of Interest" pooling (also known as RoI pooling) is a variant of max pooling, in which output size is fixed and input rectangle is a parameter.

ReLU is the abbreviation of rectified linear unit, which applies the non-saturating activation function f(x) = (0, x). It effectively removes negative values from an activation map by setting them to zero. It increases the nonlinear properties of the decision function and of the overall network without affecting the receptive fields of the convolution layer. Other functions are also used to increase nonlinearity, for example the saturating hyperbolic tangent f(x) = tanh(x), f(x) = |tanh(x)| and the sigmoid function sigma(x) = (1+e^{-x})^{-1}. ReLU is often preferred to other functions because it trains the neural network several times faster without a significant penalty to generalization accuracy.

Finally, after several convolutional and max pooling layers, the high-level reasoning in the neural network is done via fully connected layers. Neurons in a fully connected layer have connections to all activations in the previous layer, as seen in regular (non-convolutional) artificial neural networks. Their activations can thus be computed as an affine transformation, with matrix multiplication followed by a bias offset (vector addition of a learned or fixed bias term).

Figure 3.1 explains how the architecture of CNN and how the various layers or concepts work in order to provide the result.

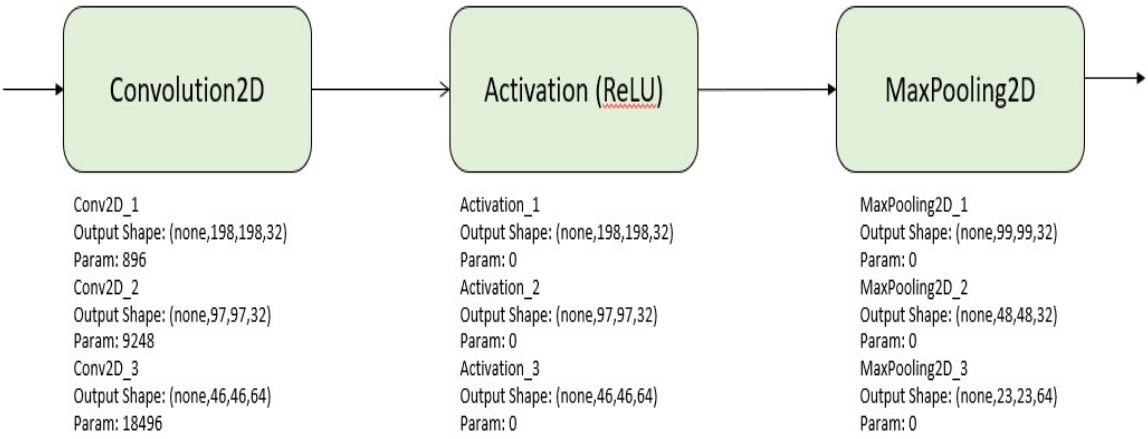

Figure 3.1 CNN Architecture

3.1.2 Mel-Frequency Cepstral Coefficient

Mel-Frequency Cepstral Coefficient is a representation of the short-term power spectrum of a sound, based on a linear cosine transform of a log power spectrum on a nonlinear Mel-scale of frequency. MFCC is the mostly widely used features in Automatic Speech Recognition (ASR), and it can also be applied to Speaker Recognition task. The process to extract MFCC feature is demonstrated in Figure 3.2.

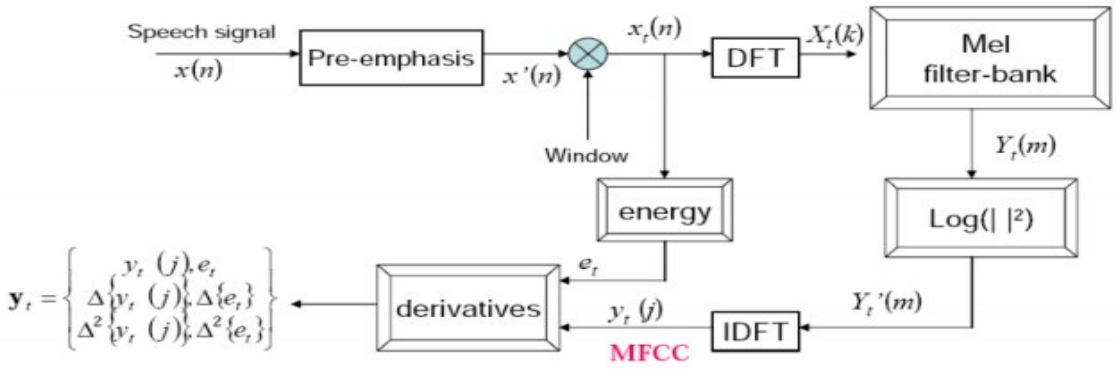

Figure 3.2 MFCC Feature Extraction

First, the input speech should be divided into successive short-time frames of length L, neighboring frames shall have overlap R. Those frames are then windowed by Hamming Window, as shown in Figure 3.3.

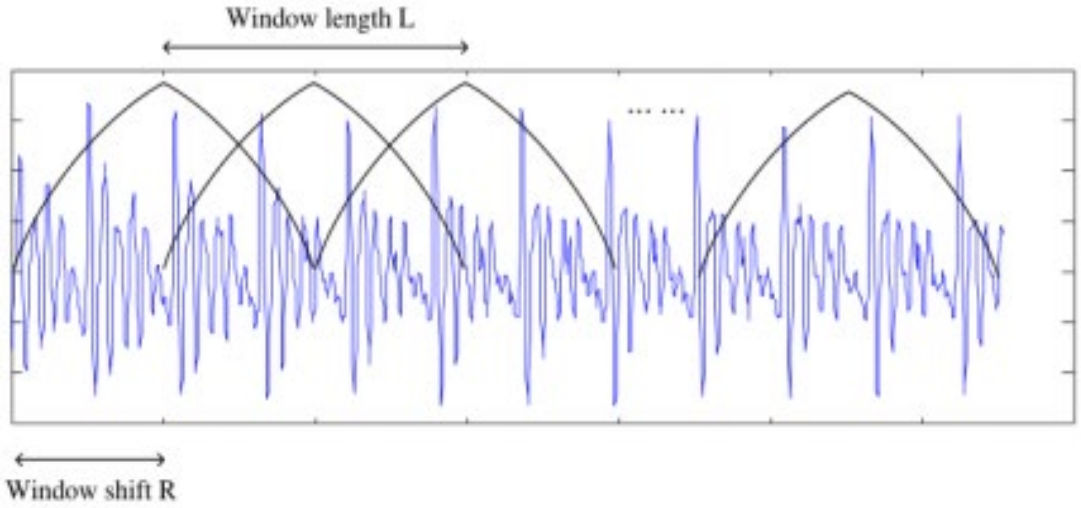

Figure 3.3 Framing and Windowing

Discrete Fourier Transform (DFT) is performed on windowed signals to compute their spectrums. For each of N discrete frequency bands we get a complex number X[k] representing magnitude and phase of that frequency component in the original signal. Considering the fact that human hearing is not equally sensitive to all frequency bands, and especially, it has lower resolution at higher frequencies. Scaling methods like Mel-scale are aimed at scaling the frequency domain to better fit human auditory perception.

They are approximately linear below 1 kHz and logarithmic above 1 kHz, as shown below in Figure 3.4.

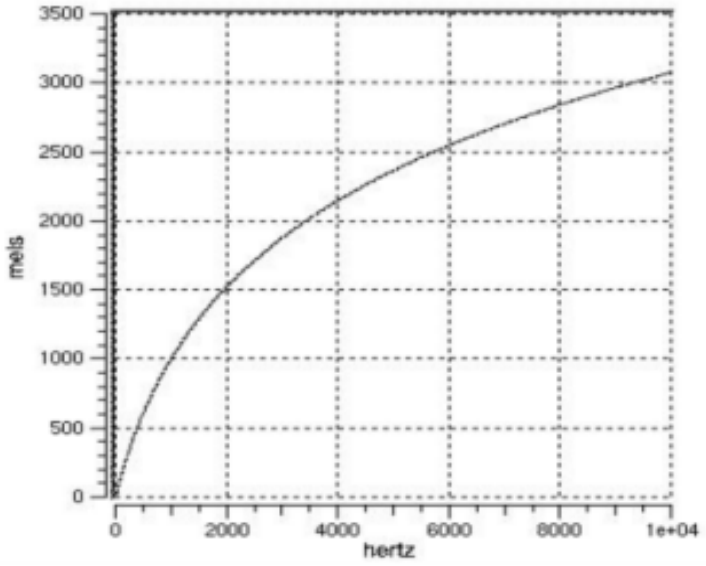

Figure 3.4 Mel-scale plots

In MFCC, Mel-scale is applied on the spectrums of the signals. The expression of Mel-scale warping is as followed:

$$M(f) = 2595 \log_{10}(1 + f/700) \quad \text{(Eqn 3.1)}$$

Then, we apply the bank of filters according to Mel-scale on the spectrum as shown in the Figure 3.5

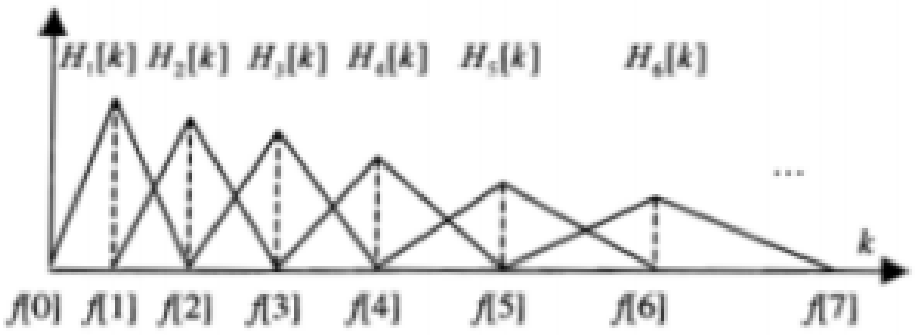

Figure 3.5 Filter Banks (6 filters)

We calculate the logarithm of energy under each bank by

$$E_i[m] = \log({}^{N-1}\textstyle\sum_{k=0} X_i[k]^2 H_m[k]) \quad \text{(Eqn 3.2)}$$

And apply Discrete Cosine Transform (DCT) on Ei[m](m = 1, 2, · · · M) to get an array ci:

$$c_i[n] = {}^{M-1}\sum_{m=0} E_i[m]\cos((\pi n/M)(m - 0.5)) \text{ (Eqn 3.3)}$$

Then, the first k terms in ci can be used as features for future training. The number of k varies in different cases.

3.1.3 Spectrogram

A common format is a graph with two geometric dimensions: one axis represents time or RPM, the other axis is frequency; a third dimension indicating the amplitude of a particular frequency at a particular time is represented by the intensity or color of each point in the image. The following Figure 3.6 shows a spectrogram of a violin waveform, with linear frequency on the vertical axis and time on the horizontal axis.

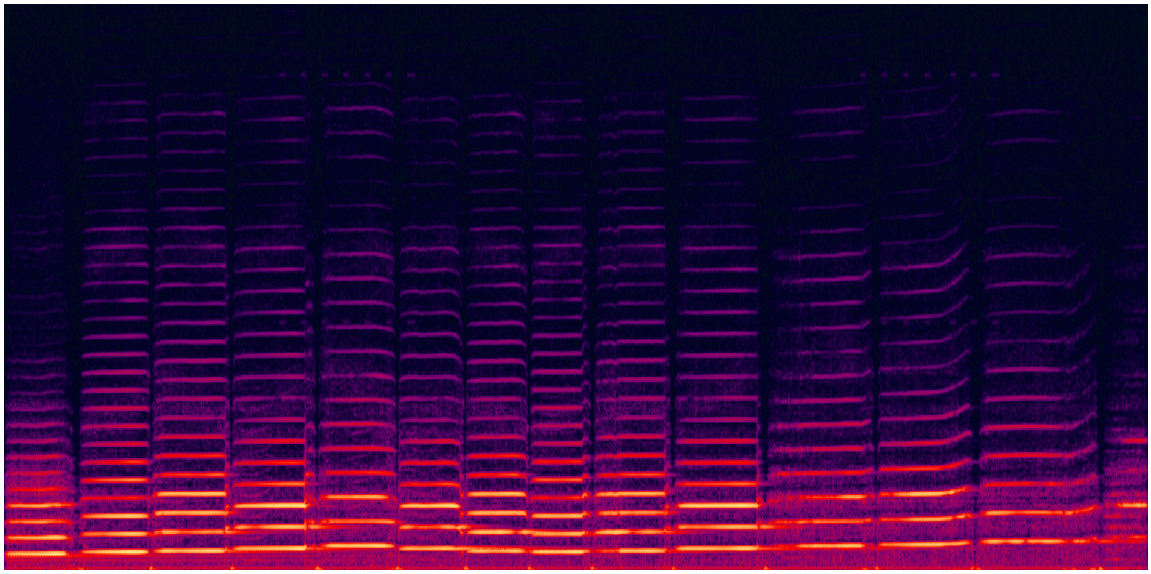

Figure 3.6 Spectrogram of Violin

The bright lines show how the spectral components change over time. The intensity coloring is logarithmic (black is −120 dBFS).

There are many variations of format: sometimes the vertical and horizontal axes are switched, so time runs up and down; sometimes as a waterfall plot where the amplitude is represented by height of a 3D surface instead of color or intensity. The frequency and amplitude axes can be either linear or logarithmic, depending on what the graph is being used for. Audio would usually be represented with a logarithmic amplitude axis (probably in decibels, or dB), and frequency would be linear to emphasize harmonic relationships, or logarithmic to emphasize musical, tonal relationships. Figure 3.7 shows 3D Surface Spectrogram and Figure 3.8, which shows 8 Spectrum and Waterfall Spectrogram.

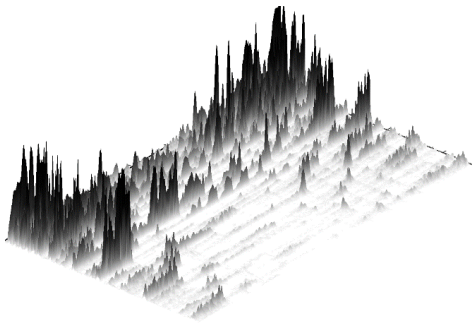

Figure 3.7 3D Surface Spectrogram

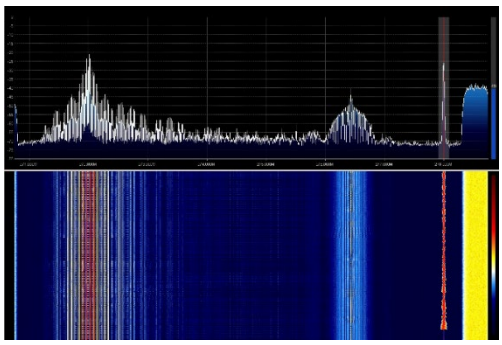

Figure 3.8 Spectrum and Waterfall Spectrogram

Figure 3.9 explains how the model turns the audio files into spectrogram images.

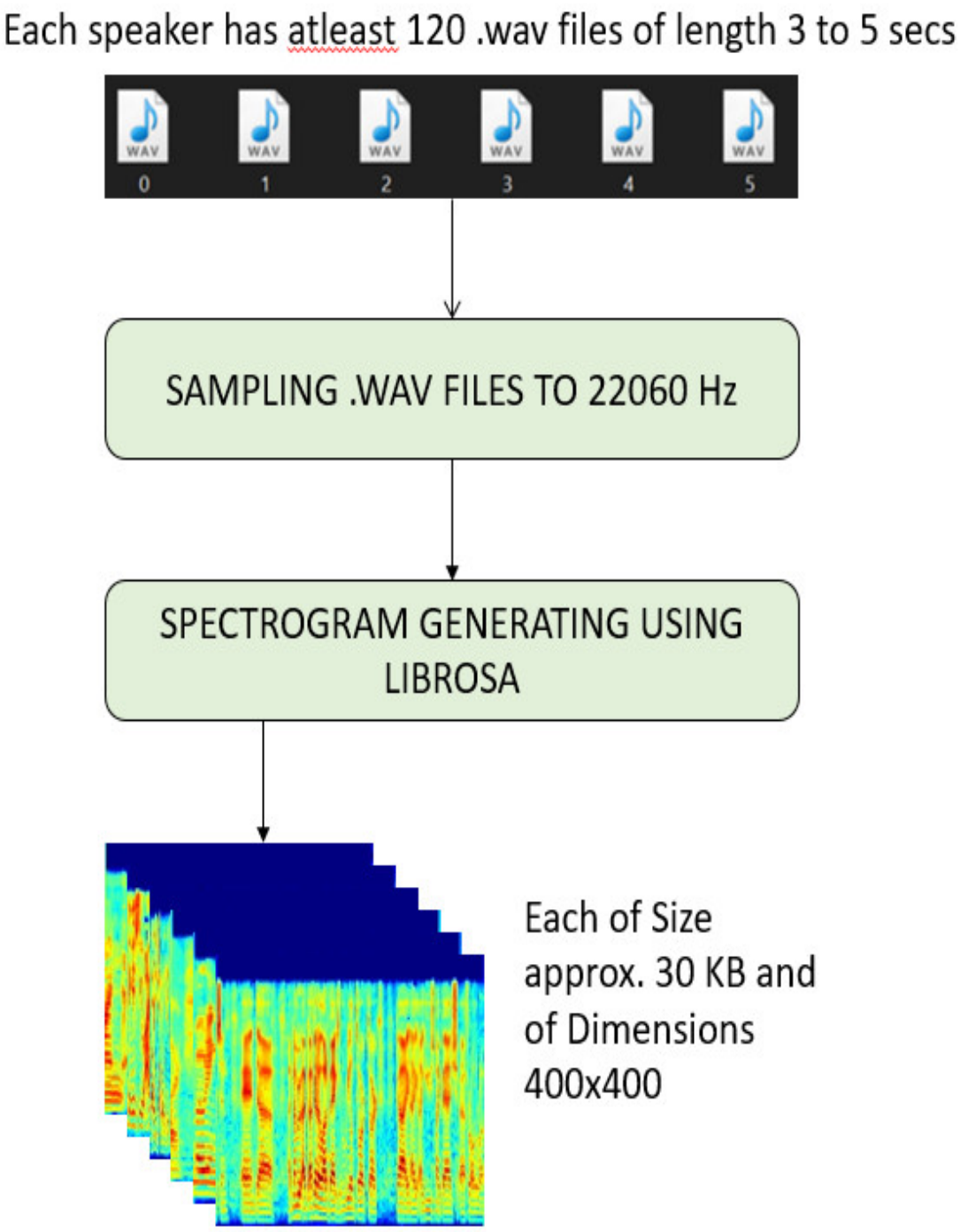

Figure 3.9 Generating Spectrograms

3.2 SYSTEM ARCHITECTURE

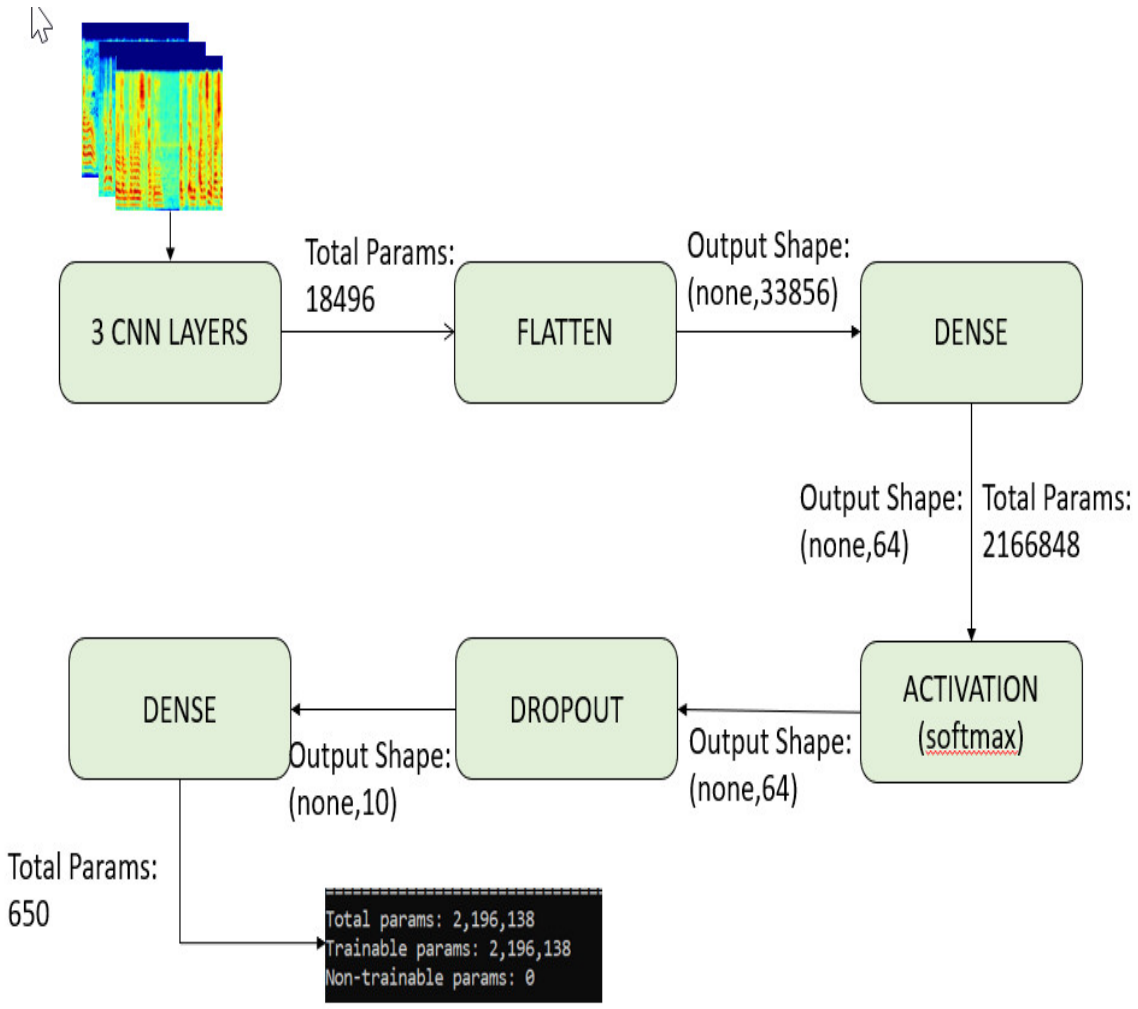

Figure 3.10 Proposed System Architecture

The proposed CNN model, consist of three convolutional layers, three fully connected layers and a SoftMax layer. The input of the network is a 400 x 400 spectrogram generated from audio files. The initial convolutional layers extract features from these spectrograms using convolution operations. Layer C1 has 8 (3x3) kernels which are applied at a stride setting of 4 pixels. It is followed by rectified linear units (ReLU) and a max pooling layer of size 2 x 2 with stride 2. ReLU act as activation functions instead of the typical sigmoid functions which improves efficiency of the training process. Layer

C2 has 256 kernels of size 5 x 5 and they are applied on the input with a stride 1. Similarly, C3 has 384 kernels of size 3 x 3. Each of these conv. layers are followed by ReLU units. Layer C3 is followed by three FC layers having 2048, 2048, and 7 neurons, respectively. In order to avoid overfitting, the last FC layer is followed by dropout layers having a dropout ratio of 50%.
The project can be summarized in 4 phases. They are:

- Dataset
- Data pre-processing
- Feature Extraction
- Model Training

3.2.1 Dataset

The model's working and accuracy was tested on a downloaded dataset, namely the VoxCeleb Dataset. But to check its reliability and correctness, a personal dataset was used which contains the voice recording of our friends and colleagues provided by our project mentor and guide Dr. R.Jayabhaduri. In short, the two datasets that are used:

- VoxCeleb Dataset
- Custom Dataset containing 20 speakers

3.2.2 Data Pre-Processing

The data must be pre-processed in order to achieve better outputs and prediction results. This is to ensure that the model is trained with minimum errors. The data from VoxCeleb was already clean and noise free for the most part.

For the custom dataset, the noise and silence were removed in the audio clips. This task was achieved by using a music/audio amplification software named Audacity. Noise reduction and silence removal was performed manually through a GUI interface of the software. This can be observed from the Figure 3.1 below. All the .mp3 files was converted into .wav file format.

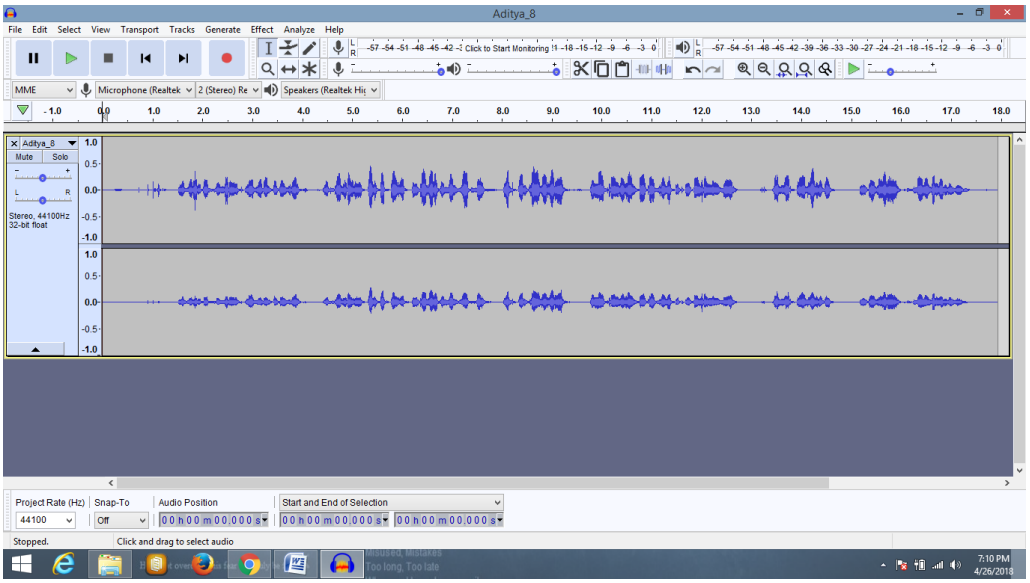

Figure 3.11 Noise Reduction + Silence Removal

3.2.3 Feature Extraction

MFCCs and their Derivatives, say Delta-MFCC, are the two main features that are being focused. 20 MFCCs and 20 Delta-MFCCs are calculated to obtain a total of 40 features in hand. Delta MFCC was calculated by a custom defined function under featureextrction.py module.

MFCC is one of the important feature extraction techniques for speaker identification technique. The goal of feature extraction is to find a set of properties of an utterance that have acoustic correlations to the speech signal

which are parameters that can somehow be computed or estimated through processing of the signal waveform. Such parameters are termed as features.

3.2.3.1 Mel Frequency Cepstral Coefficient (MFCC) estimation

After the process of removing background noises from voice signal has finished, the process of feature extraction will begin. Feature extraction is a process of obtaining different features of voice signal such as amplitude, pitch and the vocal tract. It is a task of finding parameter set obtained from the input voice signal. The extracted features should have some criteria in dealing with the speech signal such as:

- Stable over time
- Should occur frequently and naturally in speech
- Should not be susceptible to mimicry
- Easy to measure extracted speech features
- Shows little fluctuation from one speaking environment to another
- Discriminate between speakers while being tolerant of intra speaker variability's

Mel Frequency Cepstrum Coefficients (MFCC) are used to extract features in the voice signal. MFCC focuses on series of calculation that uses Cepstrum with a nonlinear frequency axis following Mel scale. To obtain Mel Cepstrum, the voice signal is windowed first using analysis window and then Discrete Fourier Transform is computed. The main purpose of MFCC is to mimic the behavior of human ears. MFCC estimation includes following processes, subdivided into six phases or blocks. The following Figure 3.2 explains the working of these phases.

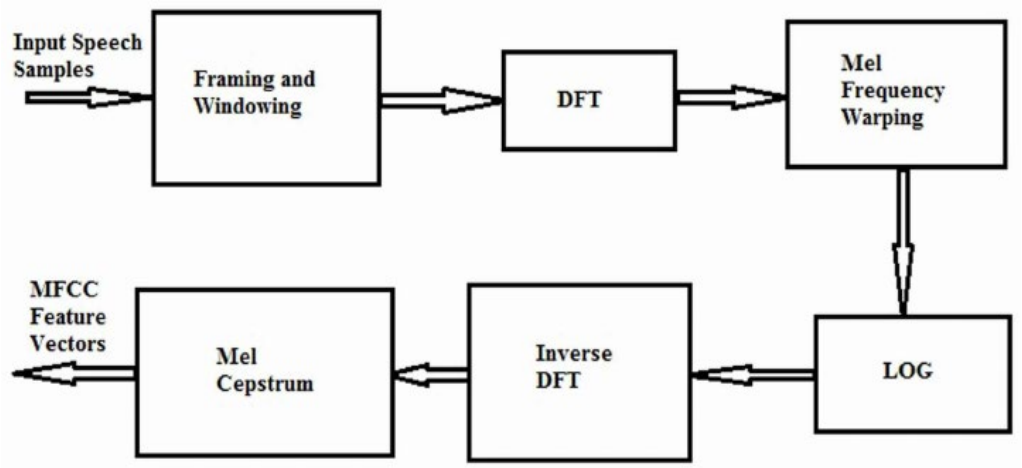

Figure 3.12 Phases of MFCC Estimation

3.2.4 Model Training

We use LibROSA, a python package for analyzing music and audio. We use functions of this library to create spectrogram images by feeding in the audio (.wav) files. To compute a Mel-scaled spectrogram we use the function: melspectrogram([y, sr, S, n_fft, ...]).

If a spectrogram input S is provided, then it is mapped directly onto the mel basis mel_f by mel_f.dot(S). If a time-series input y, sr is provided, then its magnitude spectrogram S is first computed, and then mapped onto the mel scale by mel_f.dot(S**power). By default, power=2 operates on a power spectrum.

Parameters:

- y:np.ndarray [shape=(n,)] or None
  Audio time-series
- sr:number > 0 [scalar]
  Sampling rate of y

- S:np.ndarray [shape=(d, t)]

  Spectrogram
- n_fft:int > 0 [scalar]

  Length of the FFT window
- hop_length:int > 0 [scalar]

  Number of samples between successive frames.
- power:float > 0 [scalar]

  Exponent for the magnitude melspectrogram. e.g., 1 for energy, 2 for power, etc.
- kwargs:additional keyword arguments

  Mel filter bank parameters. See librosa.filters.mel for details.

Returns:

- S:np.ndarray [shape=(n_mels, t)]

  Mel spectrogram

The Neural Network model analyzes the spectrogram images generated from the .wav files of each speaker. These spectrogram images are used to identify unique characteristics for every speaker. As these images are pretty big in size and quantity, Graphical Processing Unit (GPU) based training is the best in order to train the model in a feasible time-scale. Training on the CPU is possible but it will take a lot of time.

CHAPTER 4

SYSTEM REQUIREMENTS

4.1 HARDWARE REQUIREMENTS

- Processor: Intel Core i5 2nd Generation/ Intel Core i7 8th Generation
- Graphics card: NVIDIA Graphics card with 4GB VRAM/ MX130
- RAM: 6GB

4.2 SOFTWARE REQUIREMENTS

- Operating System: Ubuntu 18.06/ Windows 7/ Windows 10
- Python 3.6.x
- Libraries: NumPy, LibROSA
- Python Packages: TensorFlow, Keras
- Virtualenv or Anaconda

4.2.1 LibROSA

LibROSA is a python package for music and audio analysis. It provides the building blocks necessary to create music information retrieval systems. It can extract both spectral as well as rhythmic features. Few examples are:

Spectral features:-

- chroma_cens([y, sr, C, hop_length, fmin, ...])

Computes the chroma variant “Chroma Energy Normalized” (CENS).

- melspectrogram([y, sr, S, n_fft, ...])

  Compute a mel-scaled spectrogram.

- mfcc([y, sr, S, n_mfcc])

  Mel-frequency cepstral coefficients

- rmse([y, S, frame_length, hop_length, ...])

  Compute root-mean-square (RMS) energy for each frame, either from the audio samples y or from a spectrogram S.

- spectral_centroid([y, sr, S, n_fft, ...])

  Compute the spectral centroid.

Rhythm Features:-

- tempogram([y, sr, onset_envelope, ...])

  Compute the tempogram: local autocorrelation of the onset strength envelope.

LibROSA is the easiest python library to use and understand but falls short when it comes to efficiency, computationally speaking.

4.2.2 Python Speech Features

This library provides common speech features for ASR including MFCCs and filterbank energies. NumPy and SciPy are needed to run these files.

Supported features:

- python_speech_features.mfcc() - Mel Frequency Cepstral Coefficients

- python_speech_features.fbank() - Filterbank Energies
- python_speech_features.logfbank() - Log Filterbank Energies
- python_speech_features.ssc() - Spectral Subband Centroids

4.2.3 NumPy

NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more.

At the core of the NumPy package, is the ndarray object. This encapsulates n-dimensional arrays of homogeneous data types, with many operations being performed in compiled code for performance.

4.2.4 TensorFlow

TensorFlow is an open-source software library developed by the Google Brain team for internal use. It enables differentiable and dataflow programming across various tasks. Being a symbolic math library, it is used for various machine learning applications such as neural networks etc. It is utilized at both research level, as well as production purposes. Prior experience with TensorFlow has become a prerequisite to work in the machine learning industry.

4.2.5 Keras

Keras is an open-source library written in Python designed for rapid experimentation with deep neural networks. It provides numerous implementations of commonly used neural network building blocks such as layers, activation functions and optimizers. In support of Keras' core library, it acts as an interface to various computational backends such as Google's TensorFlow, Theano and Microsoft's CNTK.

4.2.6 Anaconda

Anaconda is a free and open-source distribution of the Python and R programming languages for scientific computing, that aims to simplify package management and deployment. Package versions are managed by the package management system conda. The Anaconda distribution is used by over 12 million users and includes more than 1400 popular data-science packages suitable for Windows, Linux, and MacOS as well as the Conda package and virtual environment manager, called Anaconda Navigator, so it eliminates the need to learn to install each library independently.

The open source packages can be individually installed from the Anaconda repository with the conda install command or using the pip install command that is installed with Anaconda. Pip packages provide many of the features of conda packages and in most cases, they can work together. Custom packages can be made using the conda build command, and can be shared with others by uploading them to Anaconda Cloud, PyPI or other repositories.

CHAPTER 5

IMPLEMENTATION MODULES

5.1 DATASET

The dataset used to train the proposed model contains audio files in the .wav format, each of which has duration of 3-5 seconds on an average. Each speaker has approximately 120 audio files which are extracted from various sources such as online interviews and movies.

5.1.1 VoxCeleb Dataset

The dataset used for training the model is downloaded from VoxCeleb official site by gaining authorized access to its database. VoxCeleb is an audio-visual dataset consisting of short clips of human speech, extracted from interview videos uploaded to YouTube.

- VoxCeleb1 contains over 100,000 utterances for 1,251 celebrities, extracted from videos uploaded to YouTube.
- VoxCeleb2 contains over 1 million utterances for 6,112 celebrities, extracted from videos uploaded to YouTube. The development set of VoxCeleb2 has no overlap with the identities in the VoxCeleb1 or SITW datasets.

Table 5.1 VoxCeleb1

Verification split

	dev	test
# of speakers	1,211	40
# of videos	21,819	677
# of utterances	148,642	4,874

Identification split

	dev	test
# of speakers	1,251	1,251
# of videos	21,245	1,251
# of utterances	145,265	8,251

Table 5.2 VoxCeleb2

	dev	Test
# of speakers	5,994	118
# of videos	145,569	4,911
# of utterances	1,092,009	36,237

5.1.2 Labeling Speakers

The audio files for a particular speaker are contained within a directory. The name of the directory is represented in the code as the label under which the audio files map the respective speaker. It is important that each directory doesn't have further sub-directories to speed up the process. The dataset already labels the speakers starting from '0' and each folder contains approximately 75-100 audio (.wav) files.

An XSL file is maintained to map the speaker with their audio data which is used to verify the final output of the neural network. This XSL file is also used to analyze the performance (accuracy) of the underlying Convolutional Neural Network.

5.2 PREPROCESSING MODULE

The preprocessing module consists of functions and libraries which help convert the audio files to a standard format. If there are any audio files with very low frequency data, this pre-processing module enhances the speech by reducing the noise frequency.

This conversion has many advantages. One of the advantages is that the audio files have a fixed sampling rate which makes it easier to generate spectrogram through the LibROSA function. Converting to a standard sampling rate of 22,050Hz helps to improve consistency and reduce complexity in the code for training the model while also reducing the computation resources used by the project.

As with all unstructured data formats, audio data has a couple of preprocessing steps which have to be followed before it is presented for analysis. The first step is to actually load the data into a machine understandable format. For this, the values after every specific time steps are taken. For example; in a 2 second audio file, values at half a second can be extracted. This is called sampling of audio data, and the rate at which it is sampled is called the sampling rate.

Another way of representing audio data is by converting it into a different domain of data representation, namely the frequency domain. When

sampling an audio data, more data points are required to represent the whole data. The sampling rate should be as high as possible.

On the other hand, if the audio data is represented in frequency domain, much less computational space is required. Figure 5.1 explains this.

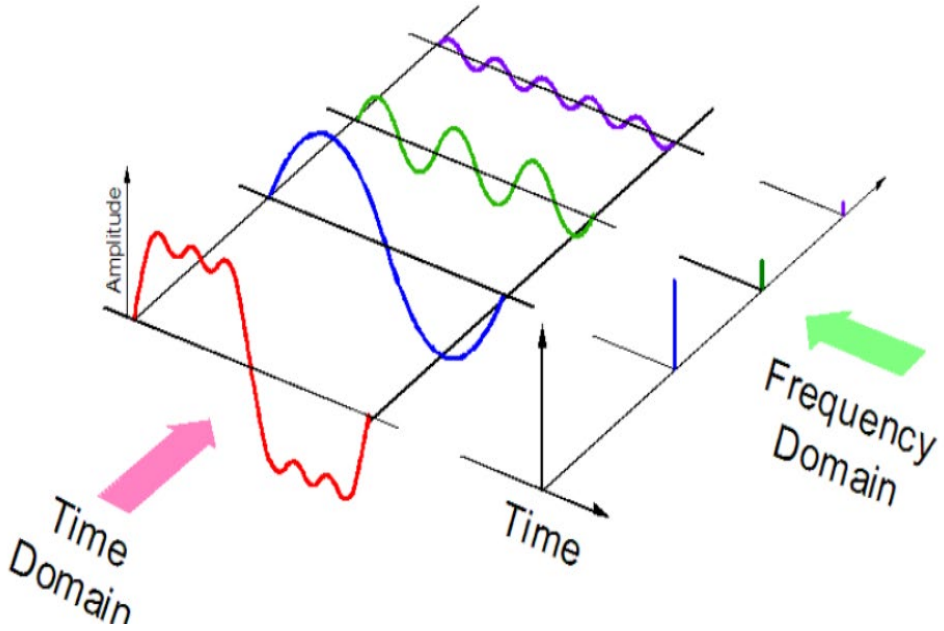

Figure 5.1 Representing Audio into 3 Pure Signals

Here, one audio signal is separated into 3 different pure signals, which can now be represented as three unique values in frequency domain. There are a few more ways in which audio data can be represented, for example using MFCs (Mel Frequency Cepstrum).

5.3 FEATURE EXTRACTION AND SPECTROGRAM GENERATION

The next step is to extract features from these audio representations, so that the algorithm can work on these features and perform the task it is

designed for. Here's a visual representation of the categories of audio features that can be extracted as shown in the Figure 5.2.

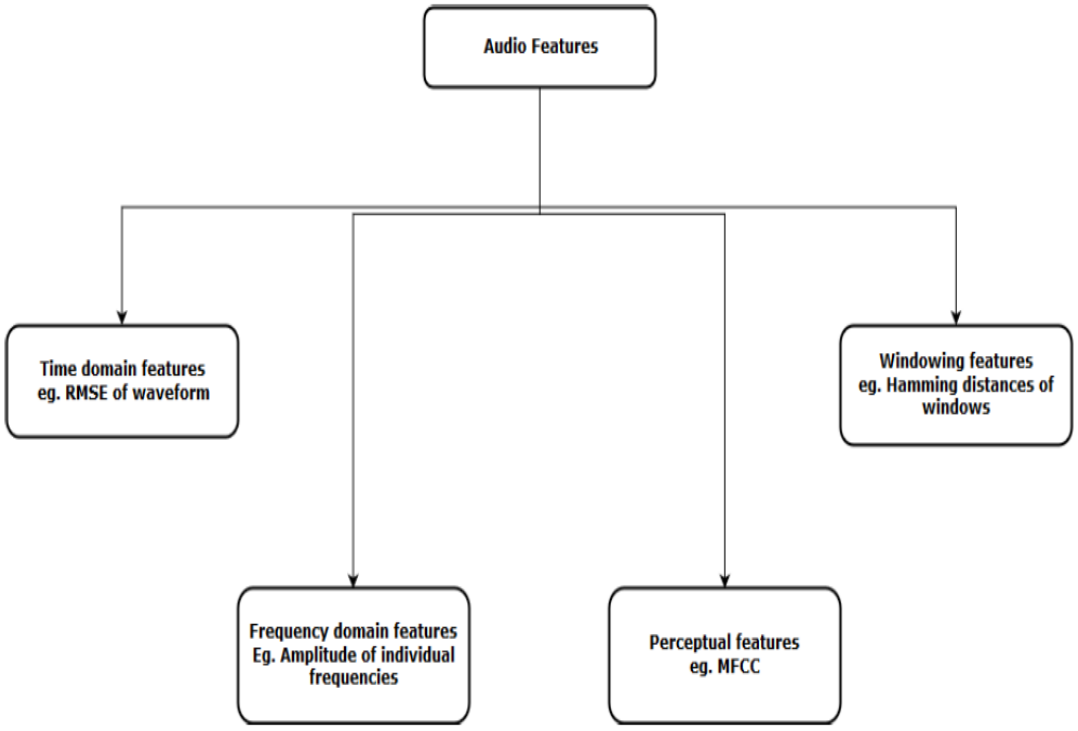

Figure 5.2 Feature Extraction

After extracting these features, it is then sent to the machine learning model for further analysis. Audio feature extraction underpins a massive proportion of audio processing, music information retrieval, audio effect design and audio synthesis. Design, analysis, synthesis and evaluation often rely on audio features, but there is a large and diverse range of feature extraction tools presented to the community. An evaluation of existing audio feature extraction libraries was undertaken. Almost 7+ libraries were tried to perform the feature extraction task for retrieval of MFCC and delta MFCC. A summary of official documentation about those libraries with their features and application areas is briefly given in Chapter 4.

5.4 NEURAL NETWORK MODULE

In this module, the various Convolutional Neural Network layers and the Keras pre-processing image function is implemented.

There is a total of 3 CNN layers followed by ReLU and max pooling layers. The Convolutional Neural Network layers consist of 3 max pooling layers and 3 core layers (Flatten, Dense, and Dropout) as well.

There are 3 layers of Convolutional Neural Network and after each Convolutional Neural Network layer there is a Rectified Linear Unit (ReLU) function which is the activation function of the model. Each subsequent layer uses the output of the previous layer as its input. The function performed by each Convolutional Neural Network layer is as follows:

- First Layer: Convolution (3x3) + Pool (2x2) with 32 filters
  Input size: 400x400
  Output Shape – 198, 198, 32
- Second Layer: Convolution (3x3) + Pool (2x2) with 32 filters
  Output Shape – 97, 97, 32
- Third Layer: Convolution (3x3) + Pool (2x2) + Core Layers with 64 filters
  Output Shape – 46, 46, 64

After the final max pooling is done at the third layer the output is fed to the core optimization layers of the Neural Network which is the Flatten, Dense and Dropout. In the end of 1 epoch of the neural network, a total of approximately 2,000,000 params can be trained.

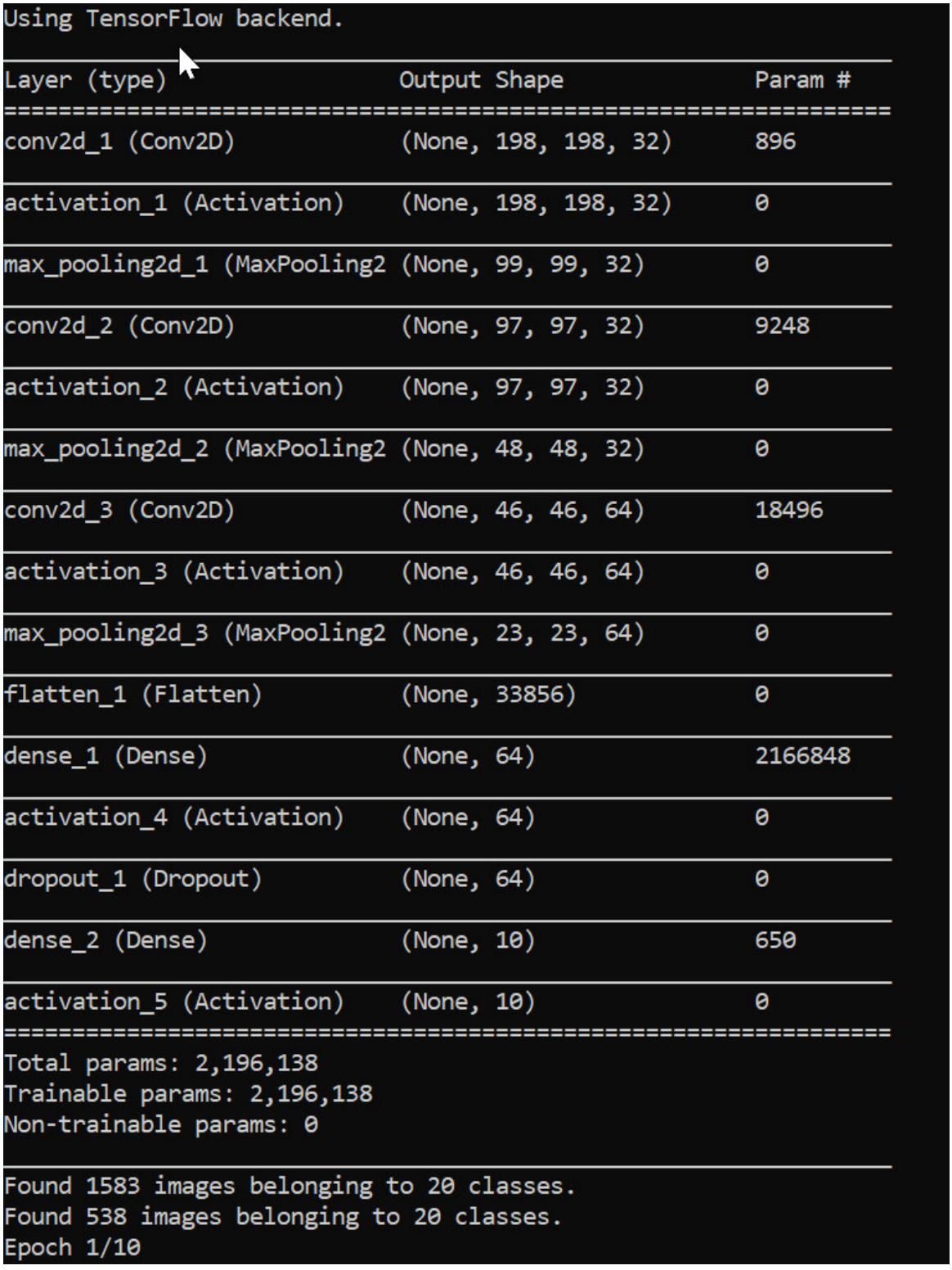

Figure 5.3 Neural Network Configuration

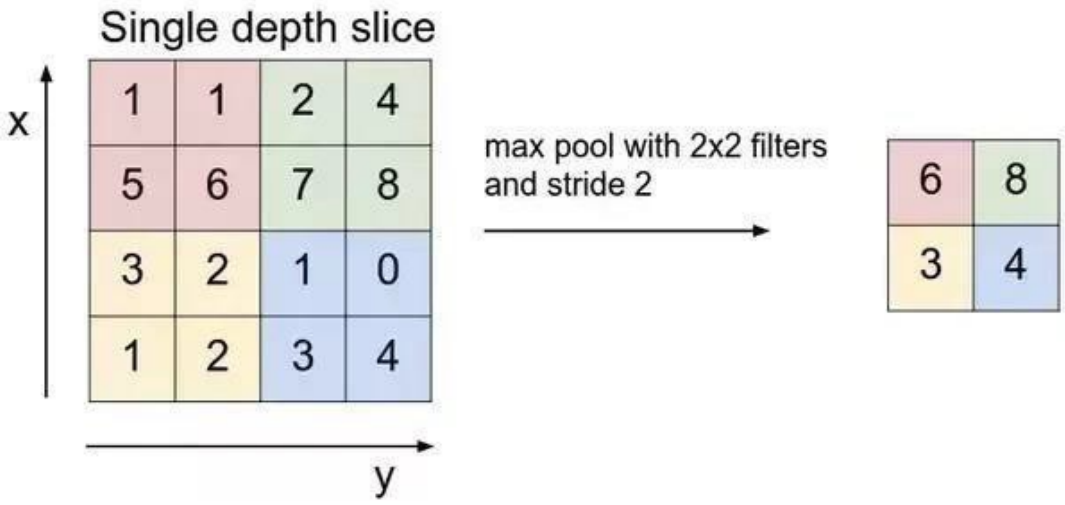

Figure 5.4 Max Pooling with Stride 2

5.5 DATALOADER MODULE

This module is responsible for loading the images for training the model. It contains the path to the character list file, accuracy file and input image file. It reads the “labels.jpeg” file which contains the location of the dataset and loads the images according to the parameters such as batch size and image size that we have set.

The module also splits the entire dataset into images used for validation and testing. The project uses 90% of the images for training and the remaining 10% of the images for validation. The number of images used per epoch is also set in this module. The project has 80,000 images per epoch. The batch size is set to 16 images per batch which means that there are 500 batches for validation and 4500 batches for training the Neural Network during each epoch.

The images in the epoch are randomly selected from the full dataset during the start of each epoch. So, each epoch will not have the same set of

images as they are chosen randomly from the dataset during each epoch iterations.

5.6 TRAINING MODULE

This module contains the train function which is responsible for the training of the model. The initial parameters are set as following:

- Variable epoch is set to 0. This keeps track of current epoch number.
- Variable train_data_dir is set to the directory from which the spectrogram images should be obtained to train the model.
- Variable target_size=(img_width, img_height) determines the size of the final output from the training generator.
- Variable batch_size is set to a pre-defined value to direct the number of batches it must train.
- Variable batchNum is set to 0. This keeps track of the current batch number.

Each epoch is then sent to the neural network module for training. If the parameters such as precision, recall and accuracy of the model have improved at the end of the epoch then the model is saved at current point.

5.7 VALIDATION MODULE

The validation module takes the remaining 10% of the dataset to validate the model that has been trained. It has the following parameters:

- Confusion Matrix
- Precision

- Recall
- F1score

The above parameters are used to evaluate the performance of the trained model. Each batch is then tested to see if the spectrogram features fall into the trained speaker module. The module then prints these parameters at the end of validation along with the accuracy and loss. This happens at the end of every epoch after the training process takes place.

This module can be run alone as well after a Neural Network model has been saved by using “testingcnn.py” as an argument when running the project. It uses the last set of validation batches to run a validation of the Neural Network model.

5.8 INFERENCE MODULE

This module takes a single image as input and identifies the speaker label it belongs to. Initially, the audio file is sent to the pre-processor module to standardize the sampling rate and produce the spectrogram images. The image is then rescaled to make the process of feature extraction fast and efficient. This batch of images are then recognized and labeled corresponding to the specific speakers. After validation takes place, we compare the results of the actual value with the predicted value to analyze the performance of the trained model. This analyze of the model is recorded in an XSL file which can later on be compared with the original values given in the dataset.

Existing results from VoxCeleb shows at most 80% accuracy for the given dataset using 6 CNN layers. Refer Table 2.1.

The model proposed produces result having accuracy of 82% with only 3 CNN Layers. The neural network module is tuned by changing the batch size and epoch values. It is found that the CNN reaches 82% accuracy when batch size is 32 and epoch is 20. It is believed that increasing batch size further will improve the accuracy and F1 score.

Table 5.3 Results for Speaker Identification

1st iteration	16	5	1.640949912	0.521739141	0.082760033	0.048627646	0.043811275
2nd iteration	16	10	1.091904809	0.671428586	0.080601343	0.068247354	0.06859052
3rd iteration	16	20	1.789673019	0.721428583	0.080538716	0.081134259	0.077976672
4th iteration	32	5	1.717259792	0.470588244	0.038922588	0.060079365	0.043223546
5th iteration	32	10	1.160873682	0.642857156	0.110161367	0.061779592	0.063653085
6th iteration	32	20	0.737835331	0.828571437	0.09952381	0.095902778	0.089858586
7th iteration	100	3	UNABLE TO TRAIN DUE TO LARGE BATCH SIZE. NOT ENOUGH VIRTUAL RAM AND SLOW PROCESSING DUE TO OLDER GPU CONFIGURATION				
8th iteration	80	3					
9th iteration	50	3					
10th iteration	5	5	1.435737557	0.591224032	0.121332738	0.068692534	0.071064382
11th iteration	5	20	1.836458789	0.739030034	0.036553125	0.054955484	0.042451278
RESULT			0.737835331	0.828571437			0.089858586

CHAPTER 6

SNAPSHOTS OF MODULES

6.1 DATASET SNAPSHOTS

Figure 6.1 shows the directory containing the VoxCeleb Dataset. Each speaker has approximately 150 audio files of 5 seconds each. The proposed model is trained using this dataset having 1211 distinct speakers.

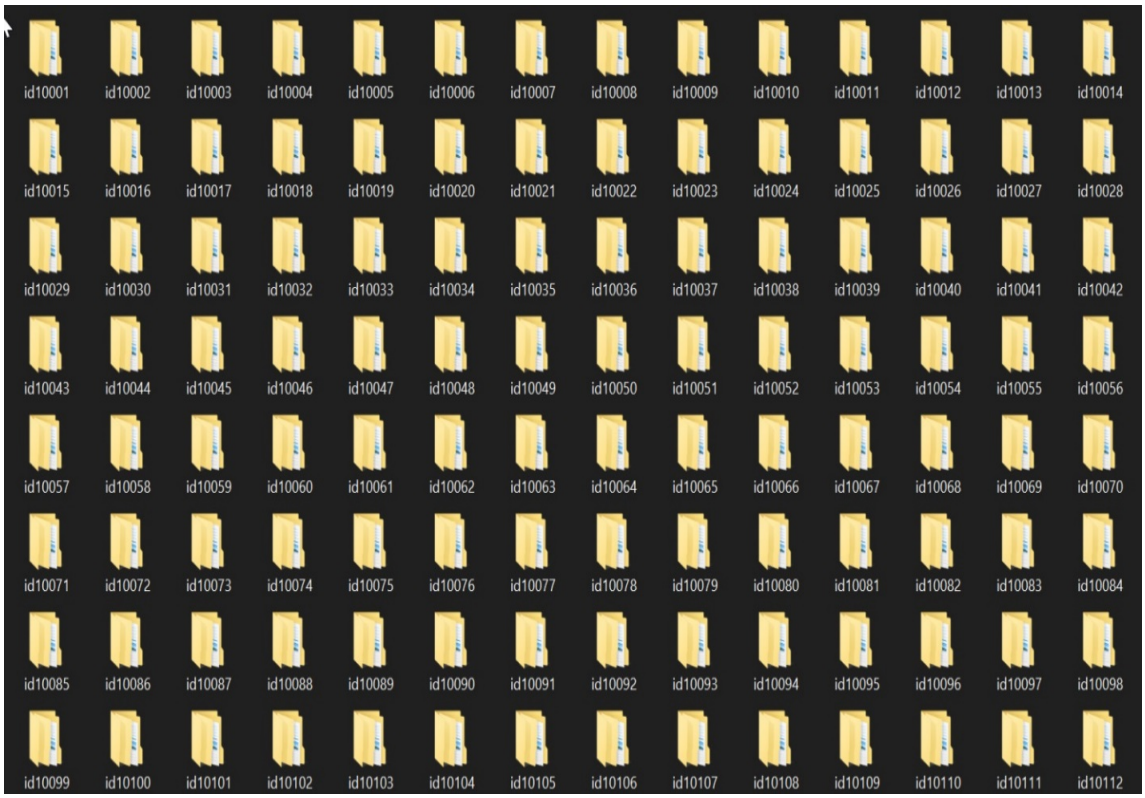

Figure 6.1 VoxCeleb Dataset Folders

Figure 6.2 gives us the meta data of each audio file under a particular speaker. Each speaker has one such file describing all the .wav files respective to that speaker.

```
Identity  :     id10002
Reference :     0_1aIeN-Q44
Offset    :     -6
FV Conf   :     21.731
ASD Conf  :     5.036

FRAME    X          Y          W          H
007333   208        80         130        130
007334   208        80         130        130
007335   208        80         130        130
007336   208        80         130        130
007337   234        95         90         91
007338   234        95         90         91
007339   208        80         130        130
007340   208        80         130        130
007341   233        78         109        109
007342   233        78         109        109
007343   223        80         130        130
007344   223        80         130        130
007345   245        78         109        109
007346   245        78         109        109
007347   245        78         109        109
007348   245        90         109        109
007349   245        90         109        109
007350   245        90         109        109
007351   245        90         109        109
007352   245        90         109        109
007353   245        90         109        109
007354   245        90         109        109
007355   245        90         109        109
007356   245        90         109        109
007357   223        80         130        130
007358   245        90         109        109
007359   245        90         109        109
007360   245        90         109        109
007361   245        90         109        109
007362   245        90         109        109
007363   245        90         109        109
007364   223        80         130        130
007365   233        90         109        109
007366   223        80         130        130
007367   233        90         109        109
007368   233        90         109        109
007369   233        90         109        109
007370   233        90         109        109
007371   221        90         109        109
007372   221        90         109        109
007373   221        90         109        109
```

Figure 6.2 Audio File Meta-Data

Figure 6.3 helps us map speaker ID with the voice of the Celebrity, which is later used in the validation part of the proposed model. The proposed model validates using this information and calculates precision, recall and F1 Score.

	A	B	C	D	E
1	VoxCeleb1	IDVGGFace1	IDGender	Nationality	Set
2	id10001	A.J._Buckley	m	Ireland	dev
3	id10002	A.R._Rahman	m	Indiа	dev
4	id10003	Aamir_Khan	m	Indiа	dev
5	id10004	Aaron_Tveit	m	USA	dev
6	id10005	Aaron_Yoom		USA	dev
7	id10006	Abbie_Cornish	f	Australia	dev
8	id10007	Abigail_Breslin	f	USA	dev
9	id10008	Abigail_Spencer	f	USA	dev
10	id10009	Adam_Beach	m	Canada	dev
11	id10010	Adam_Brody	m	USA	dev
12	id10011	Adam_Copeland	m	Canada	dev
13	id10012	Adam_Driver	m	USA	dev
14	id10013	Adrianne_Curry	f	USA	dev
15	id10014	Adrianne_Palicki	f	USA	dev
16	id10015	Agyness_Deyn	f	UK	dev
17	id10016	Aidan_Turner	m	Ireland	dev
18	id10017	Ajay_Devgn	m	Indiа	dev
19	id10018	Akshay_Kumar	m	Indiа	dev
20	id10019	Alain_Delon	m	Norway	dev

Figure 6.3 Speaker Information for Training Data

Figure 6.4 shows the custom dataset information. The custom dataset is used to check the reliability and correctness of the trained module.

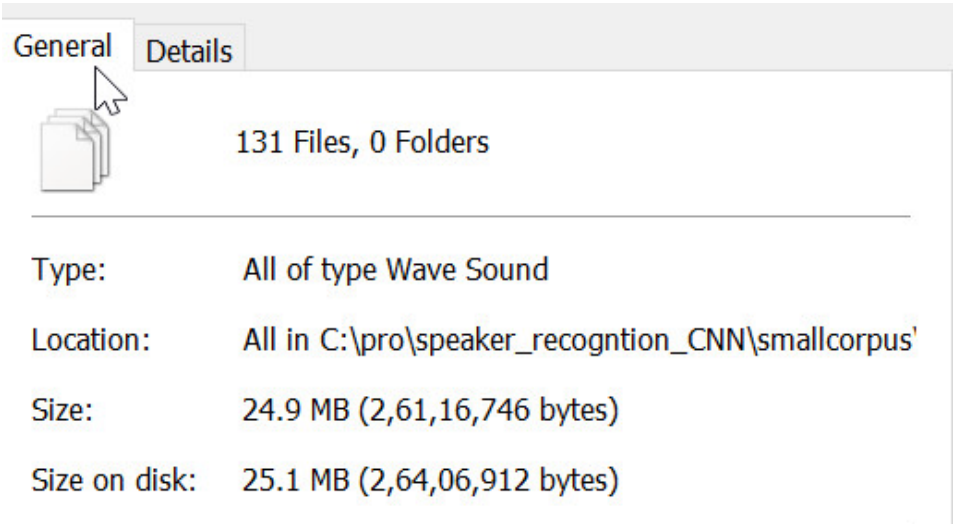

Figure 6.4 Custom Dataset

Figure 6.5 shows the total volume of the entire VoxCeleb Dataset.

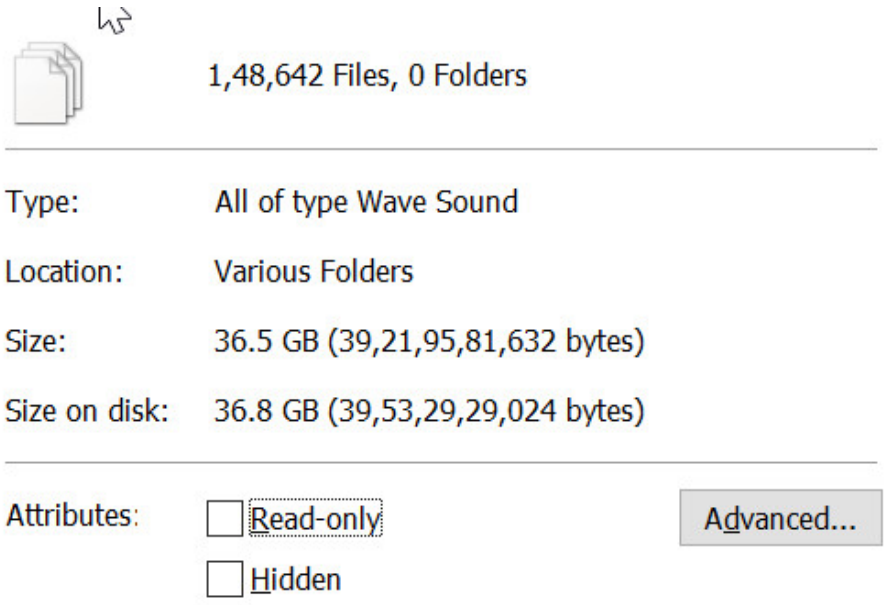

Figure 6.5 VoxCeleb Dataset Information

6.2 PREPROCESSING SNAPSHOTS

Figure 6.6 shows the source code for the preprocessing of audio files using the LibROSA library. The audio files are sampled to 22050Hz to maintain consistency and isolate low frequency noise from the speaker audio.

```
for root, dirs, files in os.walk("./smallcorpus", topdown=False):
      if root ==["./smallcorpus":
       for name in dirs:
           #X =y = os.listdir(root+"/"+name)
           #wav_file=os.listdir(root+"/"+name)
           WAV_DIR="./smallcorpus/"+name+"/"
           wav_file=os.listdir(WAV_DIR)
           IMG_DIR="./spectrogram_images/"+name+"/"
           for f in (wav_file):
               try:
                    # Read wav-file
                   y, sr = librosa.load(WAV_DIR+f)
                   # Use the default sampling rate of 22,050 Hz
                   print(sr);
                   # Pre-emphasis filter
                   pre_emphasis = 0.97
                   y = np.append(y[0], y[1:] - pre_emphasis * y[:-1])
```

Figure 6.6 Preprocessing of Audio Files

6.3 FEATURE EXTRACTION AND SPECTROGRAM CREATION

Figure 6.7 shows the source code for the generation of spectrograms using pyLab and LibROSA. The melspectrogram function of Librosa package is carried out.

```
    # Compute spectrogram
M = librosa.feature.melspectrogram(y, sr,
                                   fmax = sr/2, # Maximum frequency to b
                                   n_fft=2048,
                                       hop_length=256, # changed from 51
                                   n_mels = 96, # As per the Google Larg
                                   power = 2) # Power = 2 refers to squa

# Power in DB
log_power = librosa.power_to_db(M, ref=np.max)# Covert to dB (log) sca
      # Plotting the spectrogram and save as JPG without axes (just th
pylab.figure(figsize=(2,2))
pylab.axis('off')
pylab.axes([0., 0., 1., 1.], frameon=False, xticks=[], yticks=[]) # Re
librosa.display.specshow(log_power, cmap=cm.jet)
pylab.savefig(IMG_DIR + f[:-4]+'.jpg', bbox_inches=None, pad_inches=0)
pylab.close()
```

Figure 6.7 Spectrogram Generation

Figure 6.8 shows the directory where the spectrogram images are saved from the source code from Figure 6.7.

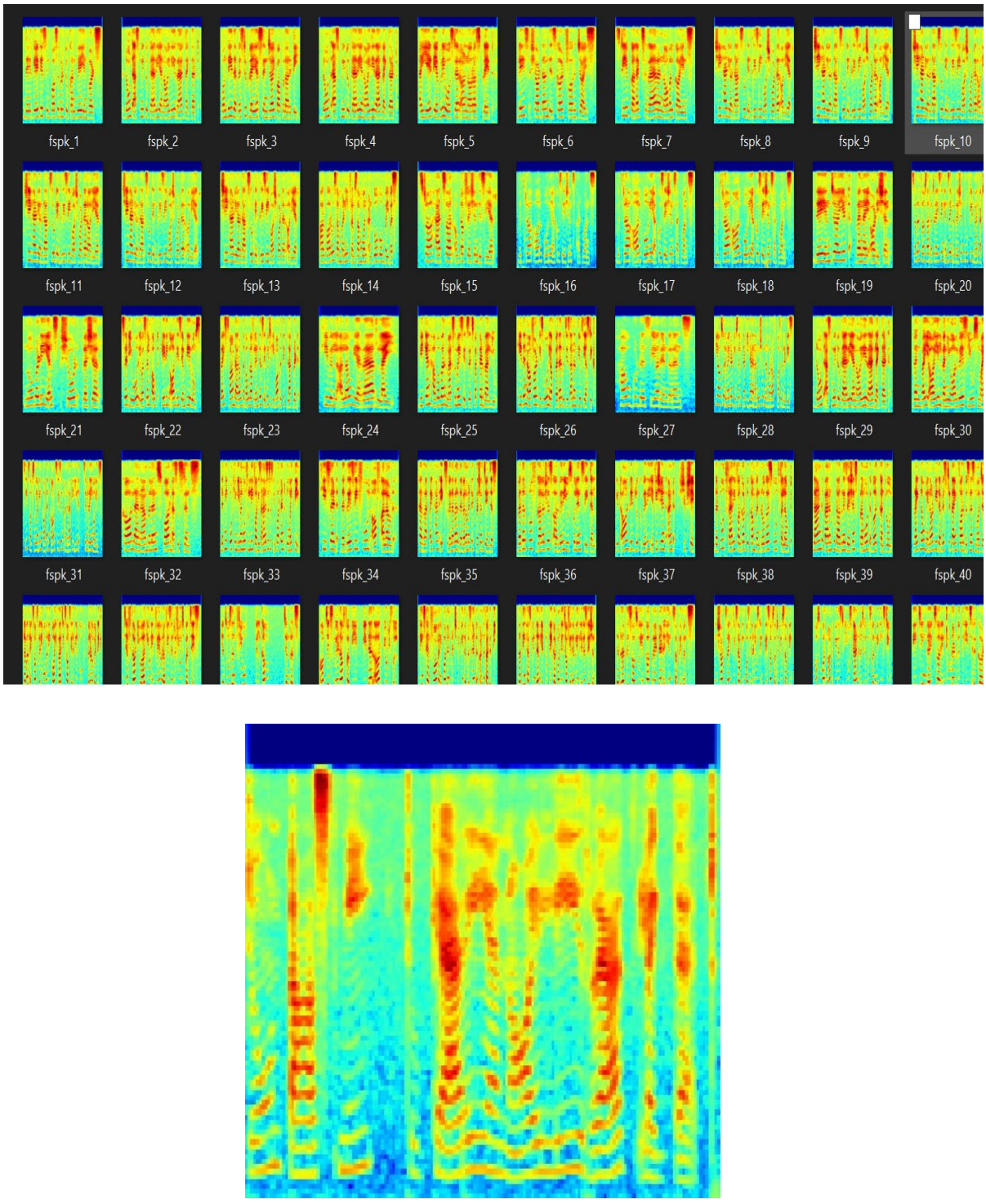

Figure 6.8 Generated Spectrograms

6.4 NEURAL NETWORK MODEL

Figure 6.9, source code for the train generated. The tunable parameters are predefined in the program “cnn_train.py”.

```
model.fit_generator(
    train_generator,
    steps_per_epoch=nb_train_samples // batch_size,
    epochs=epochs,
    validation_data=test_generator,
    validation_steps=nb_test_samples // batch_size)
```

Figure 6.9 Train Generator

Figure 6.10 illustrates the source code of the 3 Layers of CNN in the proposed model. The first two layers have 32 filters each and the last layer has 64 filters.

```
model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=input_shape))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(10))
model.add(Activation('softmax'))
```

Figure 6.10 CNN Layers

Figure 6.11 illustrates the configuration of the underlying CNN Layers. There are 3 layers of CNN and after each Convolutional Neural Network layer there is a ReLU function which is the activation function of the model. Each subsequent layer uses the output of the previous layer as its input.

```
Using TensorFlow backend.
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 198, 198, 32)      896
_________________________________________________________________
activation_1 (Activation)    (None, 198, 198, 32)      0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 99, 99, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 97, 97, 32)        9248
_________________________________________________________________
activation_2 (Activation)    (None, 97, 97, 32)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 48, 48, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 46, 46, 64)        18496
_________________________________________________________________
activation_3 (Activation)    (None, 46, 46, 64)        0
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 23, 23, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 33856)             0
_________________________________________________________________
dense_1 (Dense)              (None, 64)                2166848
_________________________________________________________________
activation_4 (Activation)    (None, 64)                0
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                650
_________________________________________________________________
activation_5 (Activation)    (None, 10)                0
=================================================================
Total params: 2,196,138
Trainable params: 2,196,138
Non-trainable params: 0
```

Figure 6.11 Output Configurations of CNN

6.5 CROSS VALIDATION SNAPSHOTS

```
from sklearn.model_selection import train_test_split
if not os.path.exists('training_data'):
    os.makedirs("./training_data")
if not os.path.exists('testing_data'):
    os.makedirs("./testing_data")
for root, dirs, files in os.walk("./spectrogram_images", topdown=False):
    if root == "./spectrogram_images":
        for name in dirs:
            X =y = os.listdir(root+"/"+name)
            if not os.path.exists("./training_data/"+name):
                os.makedirs("./training_data/"+name)
            if not os.path.exists("./testing_data/"+name):
                os.makedirs("./testing_data/"+name)
            X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=0)
            for x in X_train:
                print(root+"/"+name+"/"+x,"./training_data/"+name+"/"+x)
                os.rename(root+"/"+name+"/"+x,   "./training_data/"+name+"/"+x)
            for x in X_test:
                print(root+"/"+name+"/"+x,"./testing_data/"+name+"/"+x)
                os.rename(root+"/"+name+"/"+x, "./testing_data/"+name+"/"+x)
```

Figure 6.12 Split Train-Test

Figure 6.12 shows the source code for the data-loader module is responsible for determining which images should be loaded for training and testing of the model.

We should make a note that this module also performs cross validation. It splits the entire dataset into images used for validation and testing. Our project uses 75% of the images for training and the remaining 25% of the images for validation.

6.6 TRAINING AND VALIDATING SNAPSHOTS

In Figure 6.13, we write the label values of the testing dataset into an .csv file, which is used to compare the actual values with the predicted values.

```
model=load_model('speakermodel.h5')
print("model loaded")


f1 = open("speakerresult.csv",'w')


for root, dirs, files in os.walk("./testing_data", topdown=False):
    if root == "./testing_data":
        for name in dirs:
            TEST_DIR="./testing_data/"+name+"/"
            img_file=os.listdir(TEST_DIR)
            for f in (img_file):
                img = Image.open(TEST_DIR+f)
                x = image.img_to_array(img)
                x = np.expand_dims(x, axis=0)
                preds = model.predict_classes(x)
                label = { 0:"speaker0", 1:"speaker1", 2:"speaker2", 3:"s
                5:"speaker5", 6:"speaker6", 7:"speaker7", 8:"speaker8",
                11:"speaker11", 12:"speaker12", 13:"speaker13", 14:"spea
                 17:"speaker17", 18:"speaker18", 19:"speaker19"}

                #label = { 0:"speaker1", 1:"speaker2", 2:"speaker3", 3:'
                #how to print releveant info is needed
                f1.write(name+"\t"+ label[preds[0]] +"\n")

f1.close()
```

Figure 6.13 Testing Dataset

Figure 6.14 shows the code for validating a single file and determining the class of the spectrogram image.

```
model=load_model('speakermodel.h5')
print("model loaded")

#filename = sys.argv[1]
filename = "testing_data\\3\\16.jpg"
print(filename)
img = Image.open(filename)
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
preds = model.predict_classes(x)
label = { 0:"speaker0", 1:"speaker1", 2:"speaker2", 3:"speaker3",
4:"speaker4", 5:"speaker5", 6:"speaker6", 7:"speaker7", 8:"speaker8", 9:"speaker
 10:"speaker10", 11:"speaker11", 12:"speaker12", 13:"speaker13", 14:"speaker14",
 15:"speaker15", 16:"speaker16", 17:"speaker17", 18:"speaker18", 19:"speaker19"}
print(label[preds[0]])
```

Figure 6.14 Validating a Single File

Figure 6.15 shows the runtime snapshot of the training of the proposed model. The current training module has 10 epochs.

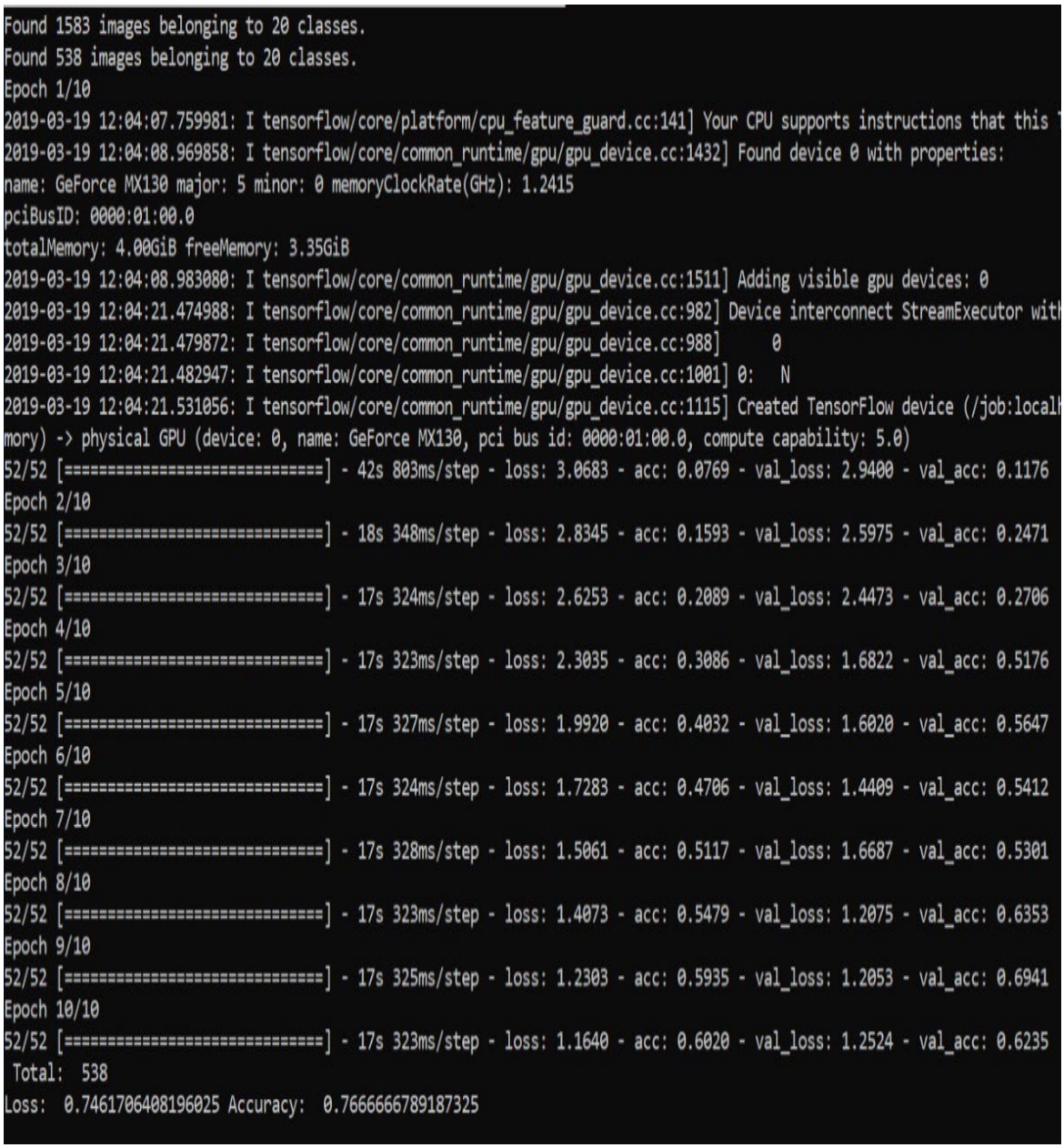

Figure 6.15 Training Runtime

6.7 PERFORMANCE ANALYSIS

Figure 6.16 illustrates the code for the analyzing the performance of the trained model. We calculate the confusion matrix, precision, recall and F1 score. We compare the values under different epochs and batch size and infer the best fort the proposed CNN.

```
from sklearn.metrics import confusion_matrix
import numpy
from sklearn import metrics

CSVFILE='./speakerresult.csv'
test_df=pd.read_csv(CSVFILE)

actualValue=test_df['actual']
predictedValue=test_df['predicted']

actualValue=actualValue.values
predictedValue=predictedValue.values

#Confusion matrix
cmt=confusion_matrix(actualValue,predictedValue)
print (cmt)

#Precision
precision=metrics.precision_score(actualValue,predictedValue,average='macro')
print('Precision',precision)

#Recall
recall=metrics.recall_score(actualValue,predictedValue,average='macro')
print('Recall',recall)

#F1-score
F1score=metrics.f1_score(actualValue,predictedValue,average='macro')
print('Recall',F1score)
```

Figure 6.16 Performance Analysis

Figure 6.17 shows the runtime snapshot of the "performanceanalysis.py", where the precision, recall, F1 score and confusion matrix is calculated.

```
(env) C:\pro\speaker_recogntion_CNN>python performanceanalysis.py
[[ 3  0  0  0  0  3  0  0  5  0  3  0  0  2  0  0  0  0  0  0]
 [ 0 14  0  0  0  0  0  0  0  0 10  0  3  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  3  0  0  4  0  9  0 18  0  0  0  0  1  0  0]
 [ 0  0  0  0  0  1  0  0  4  0  5  0  1 11  0  0  0  0  0  0]
 [ 0  0  1  0  0  0  0  0  1  0  6  0  1  3  0  0  0  0  0  0]
 [ 0  0  0  0  0  2  0  0  0  0  1  0  0  1  0 21  0  1  0  0]
 [ 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  4 12  1  0  0]
 [ 0  0  0  0  0  2  0  0  0  0  0  0  5  0  0  1  0 14  0  0]
 [ 0  0  0  0  0  6  0  0  2  0  0  0  0  4  0  0  0  0  0  0]
 [ 0  0  0  0  0  3  0  0  1  1  4  0  0  2  0  0  0  0  0  0]
 [ 0  0 26  0  0  0  0  0  0  0  2  0  1  1  0  1  0  1  0  0]
 [ 0  1  0  2  0  1  0  0  1  0  6  0  3  1  0  0  0  0  0  0]
 [ 0  0  0  0  0  2  0  1  0  0  2  0  1  0  0  0  0  4  0  0]
 [ 0  0  1  0  0 28  0  0  7  1  5  0  1  5  0  0  0  0  0  0]
 [ 0  0  0  0  0  3  1  0  1  1  3  0  0  2  0  0  0  0  0  0]
 [ 0  2  2  0  0  2  0  3  2  0  1  0  1  1  0  1  0  0  0  0]
 [ 0  0  0  0  0  0  0  0 32  0  2  0  2  1  0  0  0  0  0  0]
 [ 0  0  0  0  0  5  0  0  6  0  0  0  0  3  0  0  0  0  0  0]
 [ 0  0  0  0  0  1  0  0  0  0 36  0  1  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  3  0  0  5  0  2  0  0  2  0  0  0  1  0  0]]
C:\pro\env\lib\site-packages\sklearn\metrics\classification.py:114
es.
  'precision', 'predicted', average, warn_for)
Precision 0.1213327375022953
Recall 0.0686925343175343
C:\pro\env\lib\site-packages\sklearn\metrics\classification.py:114
.
  'precision', 'predicted', average, warn_for)
F1score 0.07106438215963155
```

Figure 6.17 Runtime Calculation

CHAPTER 7

CONCLUSION AND FUTURE WORK

7.1 CONCLUSION

The performance of speaker identification system using various feature extraction and matching techniques is studied. Mel-Spectrogram algorithm is used in our system because it has least false acceptance ratio. The speakers were trained and tested by using mel-spectrogram algorithm and CNN model. The results from tuning the parameter of the neural network module illustrates that the accuracy of the speaker identification system can be improved further. The results from VoxCeleb which has 6 CNN layers show a maximum accuracy of 80%. The proposed model gives an accuracy of 82% while using 3 CNN Layers. We have gained improvement in performance while having a smaller number of CNN layers. It is significant that the proposed model has better identification rate for speaker features.

7.2 FUTURE WORK

This work can be further extended with a GUI interface for ease in user experience. If we talk about its applicability, it has enormous future scope. The procedure followed and outputs from my project would be helpful for other researchers. They can find and suggest new methods to improve accuracy and consistency of the model.

In the future,

- The model can be improved by enabling training with single audio file of a standard length.
- The accuracy can be improved by adding more Neural Network layers and the concepts of RNN and LSTM.
- The optimizer can be replaced with Adam optimizer to check for further performance increase.

This particular work can be used as a voice biometric system to identify and catch criminals. Vision of extending this project further by adding Speech Recognition into this project. Speech Recognition + Speaker Recognition would serve perfect for practical implementation in society. The extended work is to make an intelligent system that can start tapping phone calls when getting any vulnerable keywords, mostly used by terrorists; then the speaker recognition system would recognize that terrorist or other criminal. This would help a lot in reducing crime and terror attacks in our country.

REFERENCES

1. Arjun. P. H (2005) 'Speaker Recognition in Indian Languages: A Feature Based Approach', Ph.D. dissertation, Indian Institute of Technology Kharagpur, India.

2. Badshah. A.M, J. Ahmad, N. Rahim and S. W. Baik (2017) 'Speech Emotion Recognition from Spectrograms with Deep Convolutional Neural Network, International Conference on Platform Technology and Service, pp. 1-5.

3. Jayanna. H. S (2009), Limited data Speaker Recognition', Ph.D. dissertation, Indian Institute of Technology, Guwahati, India.

4. Liu. Z, Z. Wu, T. Li, J. Li and C. Shen (July 2018) 'GMM and CNN Hybrid Method for Short Utterance Speaker Recognition', IEEE Transactions on Industrial Informatics, Vol. 14, No. 7, pp. 3244-3252.

5. Lukic. Y, C. Vogt, O. Dürr and T. Stadelmann (2016) 'Speaker identification and clustering using convolutional neural networks', IEEE 26th International Workshop on Machine Learning for Signal Processing (MLSP), pp. 1-6.

6. Maazouzi. A, N. Aqili, A. Aamoud, M. Raji and A. Hammouch (2017) 'MFCC and similarity measurements for speaker identification systems', International Conference on Electrical and Information Technologies (ICEIT), pp. 1-4.

7. Nagaraja. G. S and H. S. Jayanna (2013) 'Efficient window for monolingual and crosslingual speaker identification using MFCC', International Conference on Advanced Computing and Communication Systems, pp. 1-4.

8. Nagrani. A, J. S. Chung, A. Zisserman (2017) 'VoxCeleb: a large-scale speaker identification dataset' , Interspeech Association.

9. Schuster. M and K.K. Paliwal (Nov. 1997) 'Bidirectional Recurrent Neural Networks' IEEE Trans. Signal Processing Vol. 45, pp. 2673-2681.

10. Szegedy. C, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V.Vanhoucke, and A. Rabinovich. (2014) 'Going deeper with convolutions' arXiv preprint: 1409.4842.

11. Torfi. A, J. Dawson and N. M. Nasrabadi (2018) 'Text-Independent Speaker Verification Using 3D Convolutional Neural Networks' IEEE International Conference on Multimedia and Expo (ICME), CA, Pp. 1-6.

12. Wang. D and J. Chen (Oct. 2018) 'Supervised Speech Separation Based on Deep Learning: An Overview', IEEE/ACM Transactions on Audio, Speech, and Language Processing, Vol. 26, No. 10, pp. 1702-1726.

13. Wang. M, T. Sirlapu, A. Kwasniewska, M. Szankin, M. Bartscherer and R. Nicolas (2018) 'Speaker Recognition Using Convolutional Neural Network with Minimal Training Data for Smart Home Solutions', 11th International Conference on Human System Interaction, pp. 139-145.

14. https://towardsdatascience.com/gradient-descent-in-a-nutshell-eaf8c18212f0

15. https://www.tensorflow.org/api_docs

16. https://keras.io

17. https://towardsdatascience.com/tagged/speech-recognition

18. https://librosa.github.io/librosa/

19. http://practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/#deltas-and-delta-deltas

20. http://www.fon.hum.uva.nl/praat/manual/MelSpectrogram.html

21. https://www.tensorflow.org/tutorials/sequences/audio_recognition

22. https://medium.com/@ageitgey/machine-learning-is-fun-part-6-how-to-do-speech-recognition-with-deep-learning-28293c162f7a

23. https://medium.com/datadriveninvestor/audio-and-image-features-used-for-cnn-4f307defcc2f