Project Report titled

A Comparative Study of GMM and SincNet models for Speaker Identification

Submitted
in partial fulfillment of
the requirement of
B.Tech (Electronics)

Reported by

Darshan Kedari (181060021)

Anveshak Rathore (181060012)

Vedant Mankar (181060040)

Sourav Sawhney (181060001)

Dixit Mendon (181060043)

Under the Guidance of
Dr. D.P. RATHOD

DEPARTMENT OF ELECTRICAL ENGINEERING
VEERMATA JIJABAI TECHNOLOGICAL INSTITUTE
MUMBAI - 400019
(2021-22)

DECLARATION

We, hereby declare that the Project titled " **A Comparative Study of GMM and SincNet models for Speaker Identification**" being submitted by us towards the partial fulfillment of the Degree of B.Tech Electronics is a project work carried by us under the guidance of Dr. D. P. Rathod at the Department of Electrical Engineering. Veermata Jijabai Technological Institute (VJTI) and have not been submitted anywhere else.

The Report reflects all the work done during the entire time of dissertation.

Darshan Kedari (181060021)

Anveshak Rathore (181060012)

Vedant Mankar (181060040)

Sourav Sawhney (181060001)

Dixit Mendon (181060043)

ACKNOWLEDGEMENT

We are very glad to expose our sincere and lovable memorial thanks to our management for having on hand the facilities for the triumphant completion of the project. We cordially thank our project guide Dr. D.P. Rathod for giving valuable guidance, steady support and encouragement to inclusive our project lucratively. Also we are vastly obliged to him as our project guide, for his breed and valuable support to make our project a successful one.

We have to express our gratitude to, Head of Department, Electrical Department, for his invaluable motivation and encouragement, which helped us in the whole process of this project.

We are very much thankful to all other teaching and non-Teaching Staff members of the Electrical Department, especially our mentor Dr. Rizwan Ansari for giving unsurpassed suggestions towards successful completion of this new project.

Darshan Kedari (181060021)

Anveshak Rathore (181060012)

Vedant Mankar (181060040)

Sourav Sawhney (181060001)

Dixit Mendon (181060043)

CERTIFICATE

This is to certify that **Darshan Kedari** (181060021) **Anveshak Rathore** (181060012),**Vedant Mankar** (181060040),**Sourav Sawhney** (181060001),**Dixit Mendon** (181060043),students of B.Tech Electronics, Veermata Jijabai Technological Institute, Mumbai has successfully completed the Project titled " **A Comparative Study of GMM and SincNet models for Speaker Identification**" and submitted a report on the same to my satisfaction.

_____ _____

Dr. D.P. Rathore
Project Guide

Head of Electrical Engineering, Dept

_____

Director

VJTI

Date

Place

CERTIFICATE

The Project Titled "**A Comparative Study of GMM and SincNet models for Speaker Identification**" by **Mr**.**Darshan Kedari** (181060021), **Mr.Anveshak Rathore** (181060012), **Mr.Vedant Mankar** (181060040), **Mr.Sourav Sawhney** (181060001), **Mr.Dixit Mendon** (181060043), is found to be satisfactory and is approved for partial fulfillment of the Degree B.Tech Electronics

Dr. D.P. Rathore Project Guide	External Examiner

Date

Place

ABSTRACT

Speaker recognition is a task of identifying people from their voices. It analyzes the acoustic features of speech that have been found to differ among individuals. These usages can be, in general, interpreted as speaker identification or speaker verification tasks. With a long history of research work done in this area, it has been implemented in various biometric identification and forensics applications in the past. While there was a brief decrease in the interest of voice applications over the past few years, today with the advent of eye-garnering products like Alexa, Siri and Cortana, there is no doubt that voice-enabled technologies are experiencing a resurgence. From financial services to healthcare, a wide range of domains are incorporating voice interfaces to better the interaction with their customers. With IOT booming, there is a rapid demand for smart equipment everywhere, including systems which communicate over voice. Speaker recognition will provide user security and authentication in the voice-enabled applications.

Technical Keywords: Speaker Identification, MFCC, GMM, SincNet .

CONTENTS

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

The human ear is a marvelous organ. Beyond our uniquely human ability to receive and decode spoken language, the ear supplies us with the ability to perform many diverse functions. These include, for example, localization of objects, enjoyment of music, and the identification of people by their voices. Currently, along with efforts to develop computer procedures that understand spoken messages, there is also considerable interest in developing procedures that identify people from their voices [1].

Speaker recognition is a generic term which refers to any task which discriminates between people based upon their voice characteristics[1]. Voice being a unique characteristic to humans, it may be used as a biometric identity. Amidst all the authentication alternatives that are applicable today, speaker recognition shall prove to be a more reliable mechanism for personal identification. Security applications such as physical access control (eg: voice-controlled door), computer data access control or telephone transaction control (eg: telephonic payments) are prime targets for developing Automatic Speaker Recognition (ASR) systems[13]. ASR also finds its use in forensics (to identify culprits) and gathering reconnaissance information on either military or home environments.

There are two major applications of speaker recognition technologies and methodologies. If the speaker claims to be of a certain identity and the voice is used to verify this claim, this is called verification or authentication[2]. On the other hand, identification is the task of determining an unknown speaker's identity. In a sense, speaker verification is a 1:1 match where one speaker's voice is matched to a particular template whereas speaker identification is a 1:N match where the voice is compared against multiple templates.

From a security perspective, identification is different from verification. Speaker verification is usually employed as a "gatekeeper" in order to provide access to a secure system. These systems operate with the users' knowledge and typically require their cooperation. Speaker identification systems can also be implemented covertly without the user's knowledge to identify talkers in a discussion, alert automated systems of speaker changes, check if a user is already enrolled in a system, etc[2].

Since the advent of developing technologies that make machines learn to act or think, researchers have toiled to come up with techniques that enable machines to identify sounds or voices.Extensive research work has been done in the past with a wide spectrum of techniques developed, ranging from core statistical approaches to state-of-the-art neural networks machines (deep learning approach)[7]. As part of our project work, we wish to implement these techniques and study their characteristics and extent of accuracy on used datasets.

1.1 Motivation

Speaker identity is correlated with physiological and behavioral characteristics of the speech production system of an individual speaker. These characteristics derive from both the spectral envelope (vocal tract characteristics) and the supra-segmental features (voice source characteristics) of speech. The most commonly used short-term spectral measurements are cepstral coefficients and their regression coefficients. As for the regression coefficients, typically, the first- and second-order coefficients, that is, derivatives of the time functions of cepstral coefficients, are extracted at every frame period to represent spectral dynamics. These regression coefficients are respectively referred to as the delta-cepstral and delta-delta-cepstral coefficients[14].Speaker recognition is the process of automatically recognizing who is speaking by using the speaker-specific information included in speech waves to verify identities being claimed by people accessing systems. Applicable services include voice dialing, banking over a telephone network, telephone shopping, database access services, information and reservation services, voice mail, security control for confidential information, and remote access to computers [1]. Another important application of speaker recognition technology is as a forensics tool.

CHAPTER 2: LITERATURE REVIEW

Speaker recognition refers to recognizing the unknown persons from their voices. With the use of speech as a biometric access system, more and more ordinary persons have benefited from this technology [1]. An example is the automatic speech-based access system. Compared with the conventional password-based system, this system is more suitable for old people whose eyes cannot see clearly and figures are clumsy.

With the development of phone-based service, the speech used for recognition is usually recorded by phone. Moreover, the ambient noise and channel noise cannot be completely removed. Therefore, it is necessary to find a speaker recognition model that is not sensitive to those factors such as noise and low-quality speech.

In a speaker recognition model, the speech is firstly transformed into one or many feature vectors that represent unique information for a particular speaker irrespective of the speech content [2]. The most widely used feature vector is the short vector, because it is easy to compute and yield good performance [4]. Usually, the short vector is extracted by Mel frequency cepstral coefficient (MFCC) method [14]. This method can represent the speech spectrum in compacted form, but the extracted short vector represents only the static information of the speech. To represent the dynamic information, the Fused MFCC (FMFCC) method [14] is proposed. This method calculates not only the cepstral coefficients but also the delta derivatives, so the short vector extracted by this method can represent both the static and dynamic information.

Both of the two methods use discrete Fourier transform (DFT) to obtain the frequency spectrum. DFT decomposes the signal into a global frequency domain. If a part of frequency is destroyed by noise, the whole spectrum will be strongly interfered [12]. In other words, the DFT-based extraction methods, such as MFCC and FMFCC, are insensitive to the noise. Wavelet transform [11] is another type of tool used to obtain the frequency spectrum. Because of those independent bands, the ill effect of noise cannot be transmitted over the whole spectrum. In other words, Wavelet based feature extraction has anti noise ability.

I-vector is another type of feature vector. It is a robust way to represent a speech using a single high-dimension vector and it is generated by the short vectors. I-vector considers both of the speaker-dependent and background information, so it usually leads to good accuracy. References [5-7] have used it to enhance the performance of the speaker recognition model. Specially, [4] uses the i-vector to improve the discrimination of the low-quality speech. Usually, the i-vector is generated from the short vectors extracted by the MFCC or FMFCC methods[14].

Once the speeches are transformed into the feature vectors, a classifier is used to recognize the identity of the speaker based on those feature vectors. The Gaussian mixture model (GMM) is a conventional classifier. Because it is fast and simple, GMM has been widely used for speaker recognition [10].

The domain of speaker recognition has been well explored in past research with the aim to establish real-world ASR systems to garner a variety of applications. In this literature review, we have tried to picture out a bird's view on various approaches and techniques used to establish Automatic Speaker Recognition models[13]. From the applications mentioned earlier, we can classify each into one these tasks[2] :

Speaker Identification:- Determining an anonymous speaker's identity depending on the speaker's spoken utterances. Speaker identification finds the exact speaker from a set of recognised voices of speakers.

Speaker Verification:- It aims at verifying whether an utterance is pronounced by a hypothesized speaker based on his/her pre-recorded utterances.

Speaker Diarization:- It addresses the problem of "who spoke when", which is a process of partitioning a conversation recording into several speech recordings, each of which belongs to a single speaker.

Robust Speaker Recognition:- It deals with the challenges of noise and domain mismatch problems. Thus we have to use the techniques which can eliminate or minimize the above problems.

In addition, we require the following criteria that shall be specified in order to develop an ASR model [1][3]:

Text-dependent or text-independent systems:- In text-dependent systems the test utterance is equivalent to the text employed during the enrollment phase, while in text-independent systems the training and test speech samples are unconstrained. The text-independent systems require longer speech samples for better performance. The text-dependent systems provide better control over the speaker and the environment. This is an important criterion in case of speaker verification.

Open or closed set:- A closed-set system tries to discover a speaker's individuality among a designated number of speakers that are enrolled on the system. An open-set system considers the possibility that a test-speaker may not be included in the training set of the model, and hence assigns it an identity. This is an important factor when dealing with speaker identification.

2.1 Datasets used in Speaker Recognition

The shared datasets that are available for speaker recognition let the researchers demonstrate, evaluate, and compare the model's performance with their existing system. To create such databases, the data can be collected from various sources according to the application, such as speech recorded from acoustic laboratories, speech recorded from mobile devices, telephone calls, forensic data from police, etc. Selecting the proper dataset reduces the pre-processing time and makes the systems more efficient[3]. These speaker recognition datasets can be classified into two categories:

Clean Speech Dataset: This type of dataset contains zero presence of noise in the speech samples.

In the wild Dataset: In this type of dataset, the pure speech samples will be accompanied with noise, vibrations, intra-speaker variability and other means of acoustic disturbances.

2.2 Feature Extraction

2.2.1 Classification of features

The goal of feature extraction is to interpret a speech signal by presetting the signal's number of components. Every speaker will have his/her own set of attribute values that can be used as its representation, and later for evaluation. These features are broadly classified as[3] :-

Behavior-based speech features: These are extracted by measuring the number of parameters like experience, personality, education, familial connections, community, and communication media. High-level features and prosodic & spectro-temporal features are two categories of learned based features. High-level characteristics comprise phonetics, idiolect, semantics, accent and pronunciation, however, the prosodic & spectro-temporal features include pitch, energy, rhythm and temporal features.

Physiological-based speech features: These types of features are affected by the vocal tract's length, dimension and fold size. The short-term spectral characteristics are the types of physiological features that can be measured from small speech utterances.

2.2.2 Feature Extraction Approaches

Some feature extraction approaches often used for speaker recognition are :-

High-level features: PMFC (Phoneme Mean F-ratio Coefficient), PMFFCC (Phoneme Mean F-ratio Frequency Cepstrum Coefficient)

Prosodic: SACC (Sub-band Auto Correlation Classification), PROSACC

Spectrum based feature extraction: MFCC (Mel-Frequency Cepstral Coefficients)[14], LPCC (Linear Predictive Cepstral Coefficients). The Mel-Frequency Cepstral Coefficients (MFCC) feature extraction method is a leading approach for speech feature extraction and current research aims to identify performance enhancements. One of the recent MFCC implementations is the Delta-Delta MFCC, which improves speaker verification. This technique basically includes windowing the signal, applying the DFT (discrete fourier transform) , taking the log of the magnitude, and then warping the frequencies on a Mel scale, following applying inverse DFT. After applying this technique we get a total of 39 features from each audio sample which can be further used as an input for speaker recognition tasks.

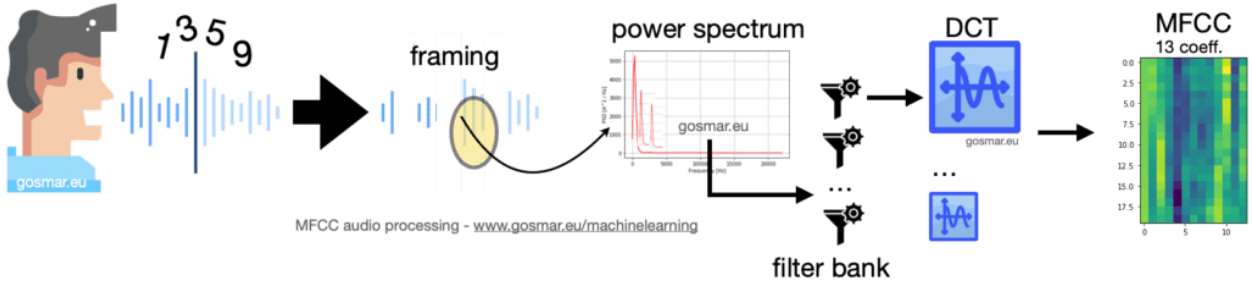


Fig 2.1

Gammatone pulse based feature extraction: Gammatone Feature, GFCC (Gammatone Frequency Cepstral Coefficients) Gender classification technique is a part of the signal processing comprising feature extraction and behavioral gender modeling. Fundamental frequency and pitch are mostly used as features for gender detection due to their unique characteristics in voice source.

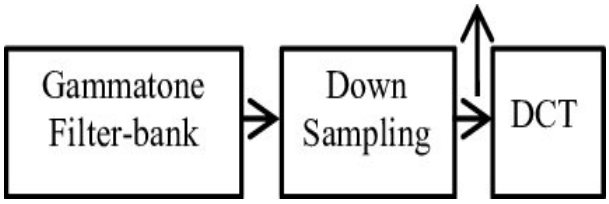


Fig 2.2

Wavelet based feature extraction: The continuous wavelet transform (CWT) is a formal (i.e., non-numerical) tool that provides an overcomplete representation of a signal by letting the translation and scale parameter of the wavelets vary continuously. Since wavelet transform has the ability to decompose complex information and patterns into elementary forms, it is commonly used in acoustics processing and pattern recognition.

2.3 Speaker Recognition Techniques

The task of speaker recognition has been tackled using two prominent architectures namely -- Stage-wise and end-to-end architectures. Stage-wise architecture contains front-end and back-end sections that work independently on extracting features from speech corpuses and classification respectively[2]. End-to-end systems are designed in a way where raw input is fed to the system and instantly returns the similarity rate.

Stage-wise architecture comprises front-end and back-end sections that deal with extracting feature identity vectors from raw speech input and applying classification algorithms to the test vectors respectively.

2.3.1: GMM-UBM/i-vector

A Gaussian mixture model (GMM) is a combination of Gaussian probability density functions (PDFs) generally used to model multivariate data. The GMM clusters the data in an unsupervised way. Evaluating the PDF at different data points (e.g. features obtained from a test utterance) provides a probability score that can be used to compute the similarity between a speaker GMM and an unknown speaker's data[4] .The model is represented by a weighted sum of the individual PDFs. If a random vector x_n can be modeled by M Gaussian components with mean vectors μ_g, covariance matrices Σg, where g = 1, 2...M indicate the component indices, the PDF of x_n is given by

$$f(xn \mid \lambda) = \Sigma (\pi g * N(xn|\mu g, \Sigma g))$$

where πg indicates the weight of the g^{th} mixture component. For a sequence of feature vectors $\chi = \{ x_n \mid n=1....T\}$, the probability of observing these features given the GMM model is computed as [5]

$$p(\chi \mid \lambda) = \Pi\, p(x_n \mid \lambda)$$

For speaker verification, apart from the claimed speaker model, an alternate speaker model (representing speakers other than the target) is needed, i,e, Universal Background Model. UBM serves as an "opposing" model that depicts contrasting feature values to the speaker. In this way, these two models can be compared with the test data and the more likely model can be chosen, leading to an accept or reject decision.

In a similar approach, a speech segment is represented by a low-dimensional "identity vector" (i-vector for short) extracted by Factor Analysis. The approach provides an elegant way of reducing high-dimensional sequential input data to a low-dimensional fixed-length feature vector while retaining most of the relevant information [4].

2.3.2 Deep Learning Methods

This approach includes replacing the i-vector calculation mechanism with a deep learning method for feature extraction. These works train a network on speaker samples using acoustic features (such as MFCCs or spectra) as inputs and speaker IDs as target variables and commonly use the output of an internal hidden layer as an i-vector alternative and apply cosine distance or PLDA as decision making [6].

Below are different variants of feature vectors that can be used in the deep learning process :

d-vector : The core idea of d-vector is to assign the ground-truth speaker identity of a training utterance as the labels of the training frames belonging to the utterance in the training stage, which transforms the model training as a classification problem. The averaged activations of the last hidden layer of a network with multiple fully connected layers are selected as features, called 'd-vector' [7].

x-vector : It is an important evolution of d-vector that evolves speaker recognition from frame-by-frame speaker labels to utterance-level speaker labels with an aggregation process. It first extracts frame-level embeddings of speech frames by time-delay layers, then concatenates the mean and standard deviation of the frame-level embeddings of an utterance as a segment-level (a.k.a., utterance-level) feature by a statistical pooling layer, and finally classifies the segment-level feature to its speaker by a standard feedforward network [7].

2.3.3 Back-end techniques:

LR Test

In speaker verification, the decision if a test sample belongs to a certain speaker is generally given by the likelihood-ratio test [5]. There are two hypotheses for an observation O:

H0: O is from speaker s
H1: O is not from speaker s

PLDA

PLDA, as an extension of LDA, is a probabilistic approach to find orthogonal axes for minimizing within-class variation and maximizing between-class variation. The linear classifier is obtained from the set of feature vectors (i-vector/x-vector), where each feature vector corresponds to a speaker.

Cosine Distance

The cosine distance is simply computing the normalized dot product of target and test i-vectors (w_{target} and w_{test}), which provides a match score [6]. Cosine similarity is a metric, helpful in determining, how similar the data objects are irrespective of their size[9]. In cosine similarity, data objects in a dataset are treated as a vector.

End-to-end architecture incorporates the front-end and back-end solutions into one complete system that takes the raw speech input and gives the output after performing classification.

2.3.4 SincNet

CNN (Convolutional Neural Networks)[8] have been used to extract important speech features and have been performing better than the previously mentioned methods. CNNs are a popular architecture to process raw waveforms. It learns to capture low level speech representations which helps in learning narrow band speaker characteristics such as pitch and formants.

According to [8], the first convolution layer of the current waveform CNN is most important as it gets high dimensional inputs. This leads to the problem of vanishing gradients in neural networks. The filters learned by CNN often take noise when few training samples are available.

To help the CNNs discover more meaningful filters in the input layer, SincNet comes as an efficient technique. SincNet convolves the waveform with a set of parametrized sinc functions

that implements the band-pass filters. The low and high cut-off frequencies are the only parameters of the filter learned from data.

Architecture of SincNet --
We take standard CNN's which performs time-domain convolutions between input waveform and some Finite Impulse Response filters

$$y[n] = x[n] * h[n] = \Sigma(x[l] \cdot h[n - l])$$

where,
x[n] is a part of speech signal,
h[n] is a filter of length L,
y[n] is the output.

In SincNet, convolutions are performed on function 'g'. In digital signal processing, g is defined such that a filter-bank composed of rectangular band-pass filters is employed.

$g[n,f_1,f_2] = 2f_2\text{sinc}(2\Pi f_2 n) - 2f_1\text{sinc}(2\Pi f_1 n)$
Here, f_1 and f_2 are the learned low and high cut-off frequencies resp.

The major advantage of assigning filters this way is that it has the advantage of directly allocating more filters at the lower part of the spectrum which has unique information of speaker voices.The properties of SincNet is that it is designed in such a way that it forces the network to focus on filter parameters impacts its performance, leading to fast convergence rate of the algorithm and also drastically reduces the number of parameters in the first convolutional layer.

CHAPTER 3: PROPOSED METHODOLOGY

3.1 Dataset

For our implementation, we have collected 15 audio samples of 8-10 seconds from each of our team members, with a 66-33% train-test split. This means 50 audio samples for training the model, and 25 for testing it. This speaks for the robustness and accuracy of the model even on a minimalist dataset, and ensures that adding additional speakers to the detection database is a simple process.

The dataset can be seen on the system as:

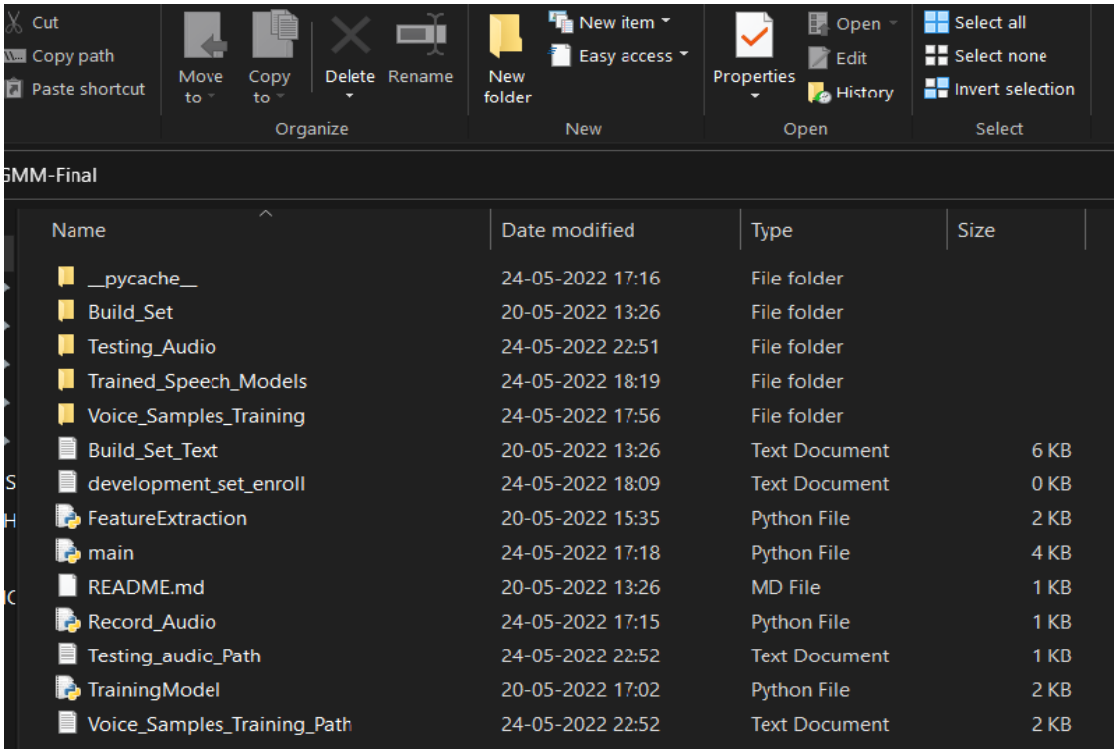


Fig 3.1

The training dataset and testing dataset are contained in the separate folders. The training dataset consists of 10 labeled audio samples in .wav format from each speaker having a duration of 8-10 seconds each.

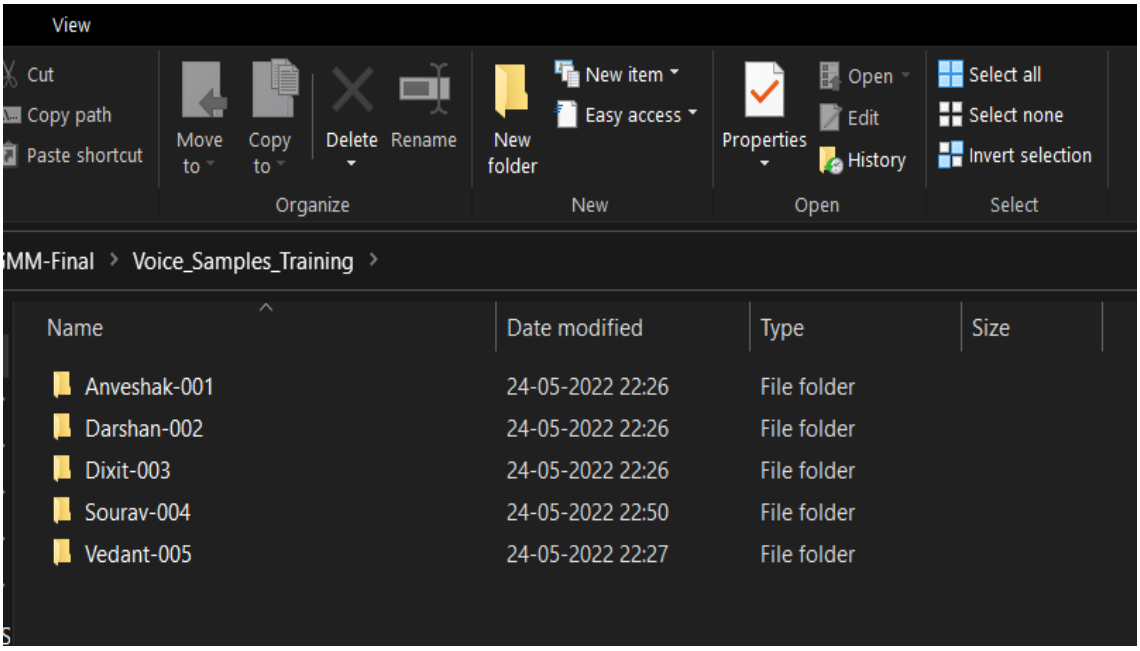


Fig 3.2

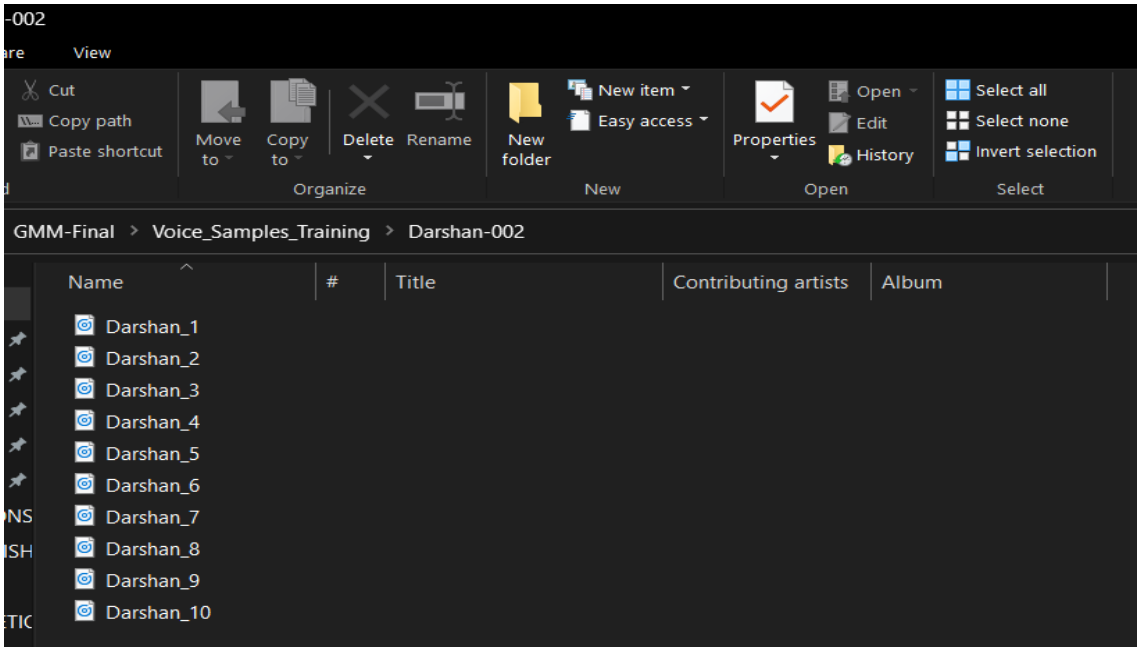


Fig 3.3

The testing dataset consists of 5 audio clips from each of the speakers of the same duration as earlier. The 4 audio samples from each speaker's training set is labeled and one of the 5 training samples is unlabeled.

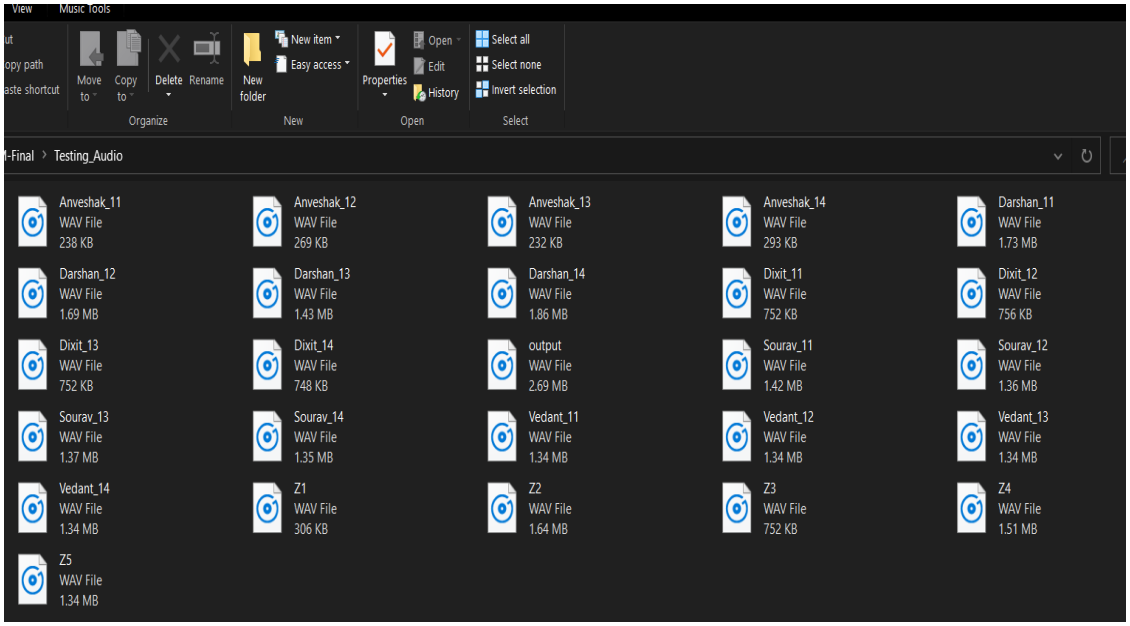


Fig 3.4

3.2 GMM

Gaussian mixture models (GMMs) are a type of machine learning algorithm. They are used to classify data into different categories based on the probability distribution. Gaussian mixture models can be used in many different areas, including finance, marketing and so much more!

3.2.1 What are Gaussian mixture models (GMM)?

Gaussian mixture models (GMM) are a probabilistic concept used to model real-world data sets. GMMs are a generalization of Gaussian distributions and can be used to represent any data set that can be clustered into multiple Gaussian distributions. The Gaussian mixture model is a probabilistic model that assumes all the data points are generated from a mix of Gaussian distributions with unknown parameters. A Gaussian mixture model can be used for clustering, which is the task of grouping a set of data points into clusters. GMMs can be used to find clusters in data sets where the clusters may not be clearly defined. Additionally, GMMs can be used to estimate the probability that a new data point belongs to each cluster. Gaussian mixture models are also relatively robust to outliers, meaning that they can still yield accurate results even if there are some data points that do not fit neatly into any of the clusters. This makes GMMs a flexible and powerful tool for clustering data. It can be understood as a probabilistic model where Gaussian distributions are assumed for each group and they have means and covariances which define their parameters. GMM consists of two parts – mean vectors (μ) & covariance matrices (Σ). A Gaussian distribution is defined as a continuous probability distribution that

takes on a bell-shaped curve. Another name for Gaussian distribution is the normal distribution. Here is a picture of Gaussian mixture models:

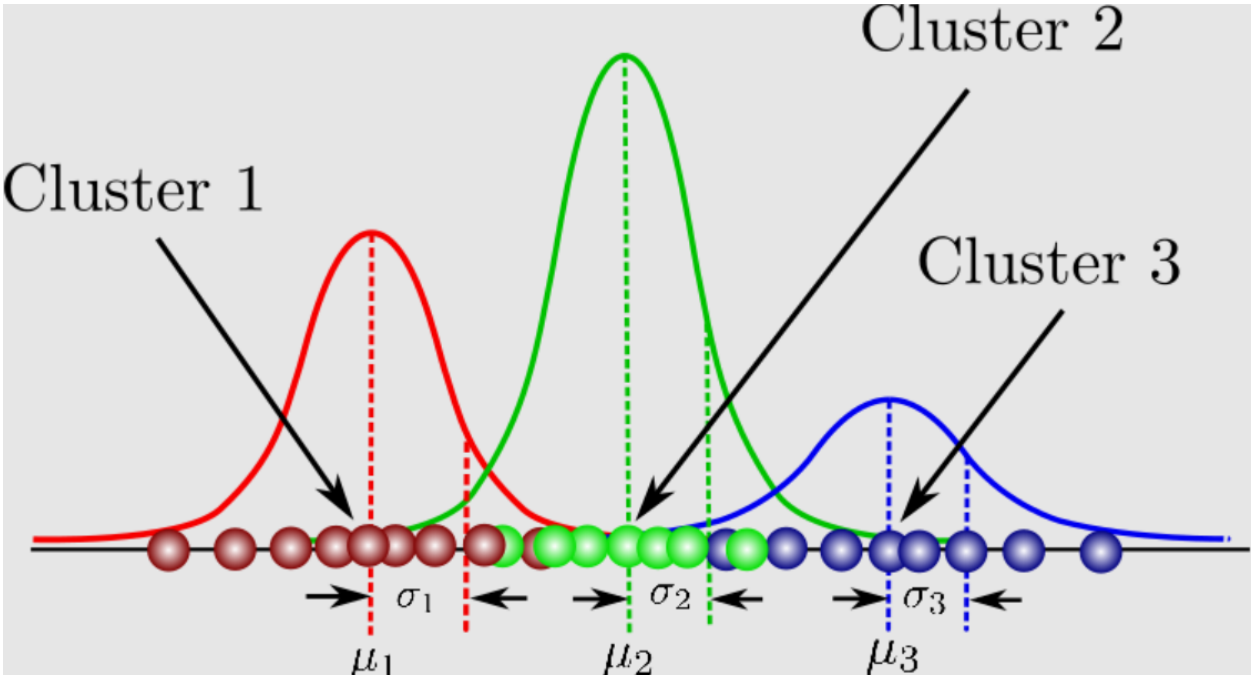


Fig 3.5

GMM has many applications, such as density estimation, clustering, and image segmentation. For density estimation, GMM can be used to estimate the probability density function of a set of data points. For clustering, GMM can be used to group together data points that come from the same Gaussian distribution. And for image segmentation, GMM can be used to partition an image into different regions.

Gaussian mixture models can be used for a variety of use cases, including identifying customer segments, detecting fraudulent activity, and clustering images. In each of these examples, the Gaussian mixture model is able to identify clusters in the data that may not be immediately obvious. As a result, Gaussian mixture models are a powerful tool for data analysis and should be considered for any clustering task.

3.2.2 Relation of expectation-maximization (EM) method with GMM

In Gaussian mixture models, an expectation-maximization method is a powerful tool for estimating the parameters of a Gaussian mixture model (GMM). The expectation is termed E and maximization is termed M. Expectation is used to find the Gaussian parameters which are used to represent each component of gaussian mixture models. Maximization is termed M and it is involved in determining whether new data points can be added or not.

The expectation-maximization method is a two-step iterative algorithm that alternates between performing an expectation step, in which we compute expectations for each data point using current parameter estimates and then maximize these to produce a new gaussian, followed by a maximization step where we update our gaussian means based on the maximum likelihood estimate. The EM method works by first initializing the parameters of the GMM, then iteratively improving these estimates. At each iteration, the expectation step calculates the expectation of the log-likelihood function with respect to the current parameters. This expectation is then used to maximize the likelihood in the maximization step. The process is then repeated until convergence. Here is a picture representing the two-step iterative aspect of the algorithm:

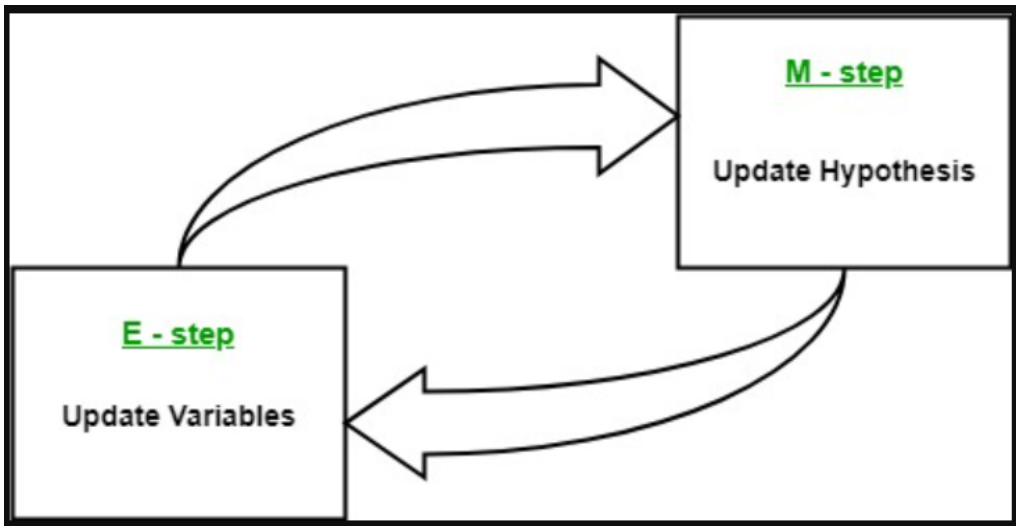


Fig 3.6

3.2.3 Key steps of using Gaussian mixture models for clustering

The following are three different steps to using gaussian mixture models:

1. Determining a covariance matrix that defines how each Gaussian is related to one another. The more similar two Gaussians are, the closer their means will be and vice versa if they are far away from each other in terms of similarity. A gaussian mixture model can have a covariance matrix that is diagonal or symmetric.
2. Determining the number of Gaussians in each group defines how many clusters there are.
3. Selecting the hyperparameters which define how to optimally separate data using gaussian mixture models as well as deciding on whether or not each gaussian's covariance matrix is diagonal or symmetric.

3.3 MFCC-GMM model

The first automatic speaker identification method was based on Gaussian mixture models. GMM is a combination of Gaussian probability density functions (PDFs) that are commonly used to model multivariate data. It does not only cluster data in an unsupervised way, but also gives its PDF. Applying GMM to speaker modeling provides the speaker specific PDF, from which probability score can be obtained. Thus, testing a sample with an unknown label, based on the probability scores of the speaker GMMs, a decision can be made. In this project, we aimed at achieving speaker identification only, hence a UBM was not required.

3.3.1 MFCC Feature Extraction Process

The most popular short-term acoustic features are the Mel-frequency cepstral coefficients (MFCCs). In sound processing, the mel-frequency cepstrum (MFC) 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. MFCCs are coefficients that collectively make up an MFC. They are derived from a type of cepstral representation of the audio clip (a nonlinear "spectrum-of-a-spectrum"). The difference between the cepstrum and the mel-frequency cepstrum is that in the MFC, the frequency bands are equally spaced on the mel scale, which approximates the human auditory system's response more closely than the linearly-spaced frequency bands used in the normal spectrum.

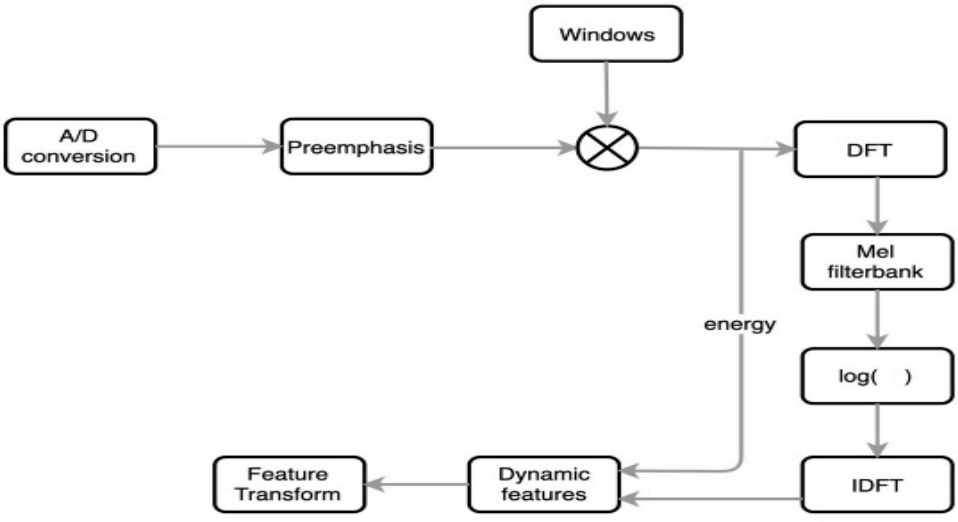


Fig 3.7

The workflow of MFCC extraction is explained as follows :

- To obtain these coefficients from an audio recording, first the audio samples are divided into short overlapping segments (sampling frequency of 16kHz).
- The signal obtained in these segments/frames is then multiplied by a window function (e.g., Hamming and Hanning), and the Fourier power spectrum is obtained.
- In the next step, the logarithm of the spectrum is computed and non linearly spaced Mel-space filter-bank analysis is performed. The logarithm operation expands the scale of the coefficients and also decomposes multiplicative components.
- Finally, MFCCs are obtained by performing DCT on the filter-bank energy parameters and retain a number of leading coefficients.

DCT has two important properties:

- it compresses the energy of a signal to a few coefficients and
- its coefficients are highly decorrelated.

For these reasons, removing some dimensions using DCT improves modeling efficiency and reduces some nuisance components. Also, the decorrelation property of DCT helps the models that assume feature coefficients are uncorrelated.

We have implemented the mfcc function from the library python_speech_features on raw audio data. The specified parameters are ---

- ➢ Sampling frequency -- 16kHz (default)
- ➢ Window length -- 25 milliseconds (default)
- ➢ Window step size -- 0.01 milliseconds (default)
- ➢ FFT size (nfft) -- 1200
- ➢ appendEnergy == True -- zeroth cepstral coefficient is replaced with the log of the total frame energy

It returns an array of feature values, where each row corresponds to a vector (or, mel coefficient).

Deltas are known as differential coefficients. The MFCC feature vector describes only the power spectral envelope of a single frame, but it seems like speech would also have information in the dynamics i.e. what are the trajectories of the MFCC coefficients over time. The delta coefficients are computed using the following formula --

$$d_t = \frac{\sum_{n=1}^{N} n(c_{t+n} - c_{t-n})}{2\sum_{n=1}^{N} n^2}$$

The mel coefficients appended with delta coefficients make our final set of feature values for one audio data file.

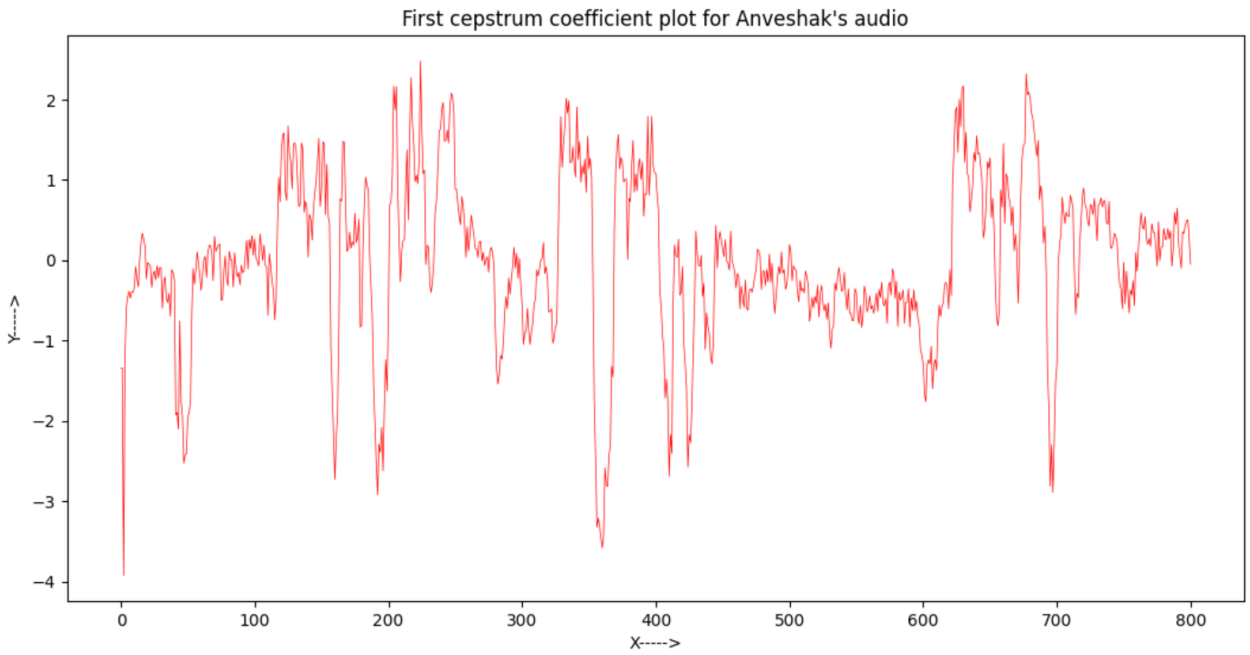

First cepstrum coefficient plot for Anveshak's audio
Y----->
X----->
2
1
0
−1
−2
−3
−4
0
100
200
300
400
500
600
700
800

Fig 3.8

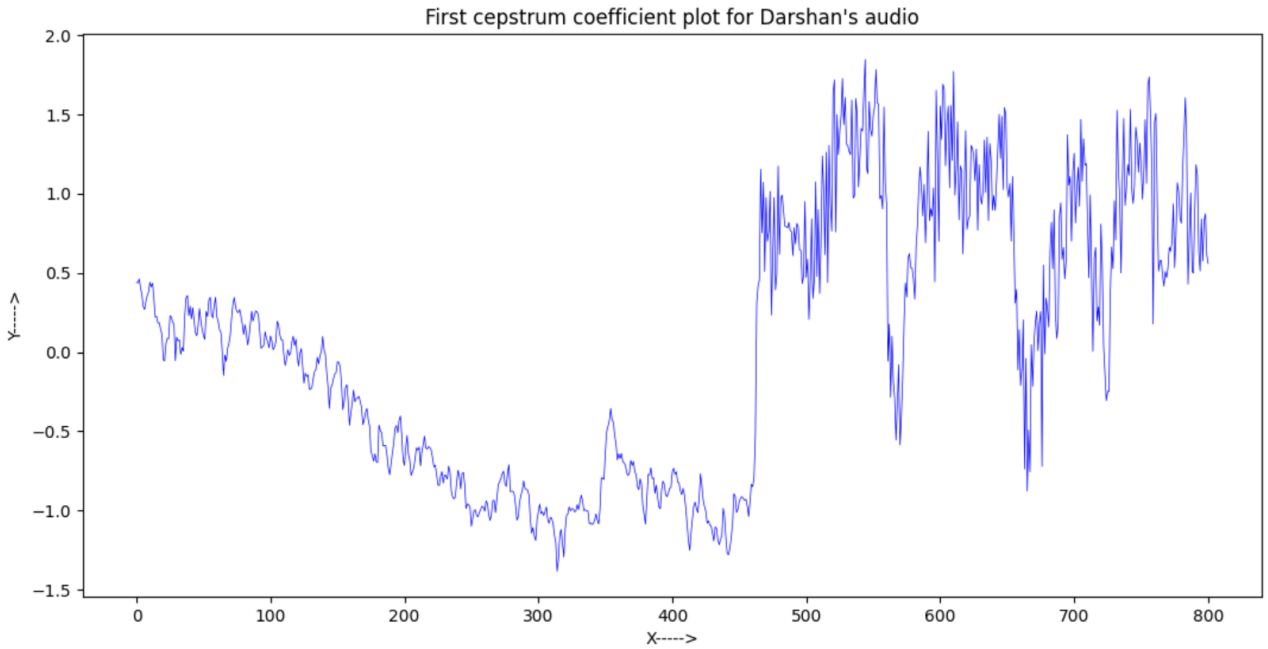

First cepstrum coefficient plot for Darshan's audio
Y----->
X----->
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
0
100
200
300
400
500
600
700
800

Fig 3.9

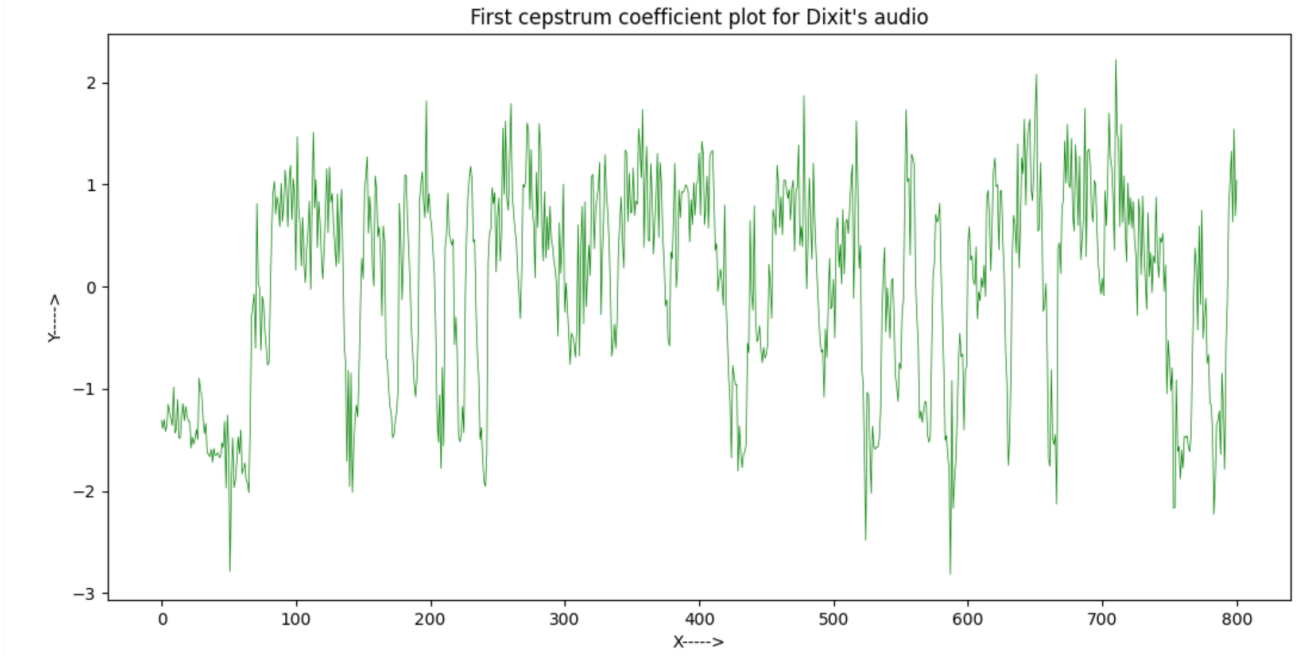


Fig 3.10

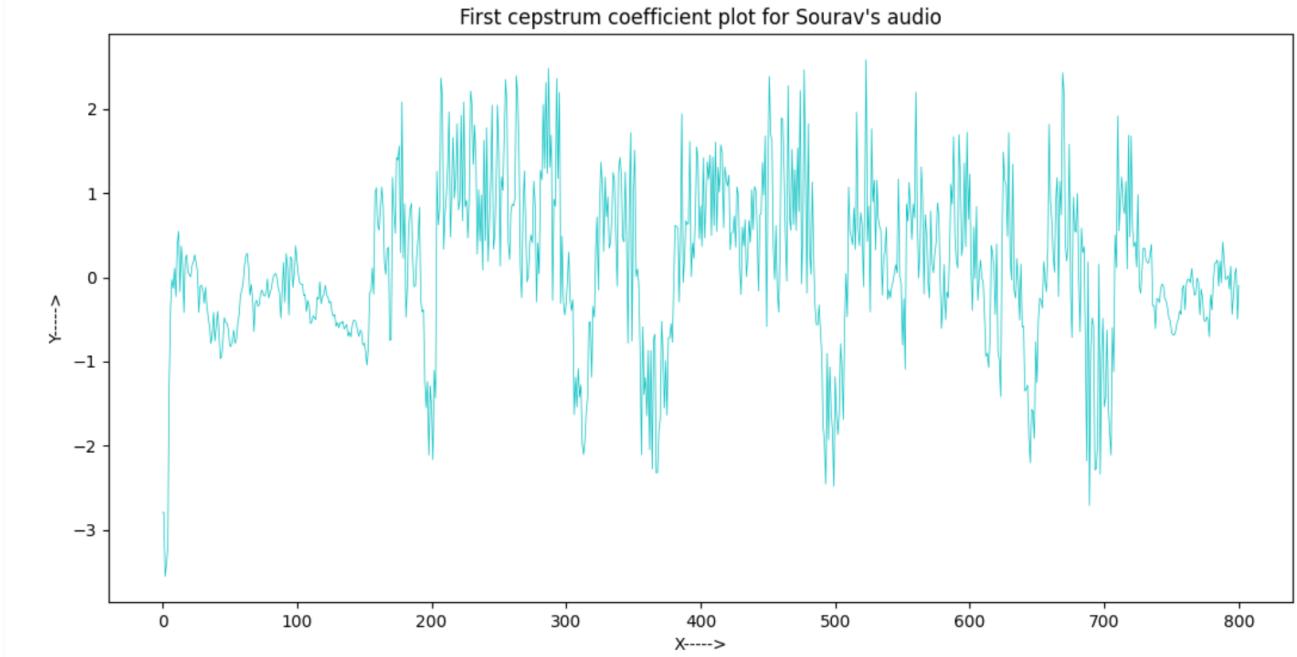


Fig 3.11

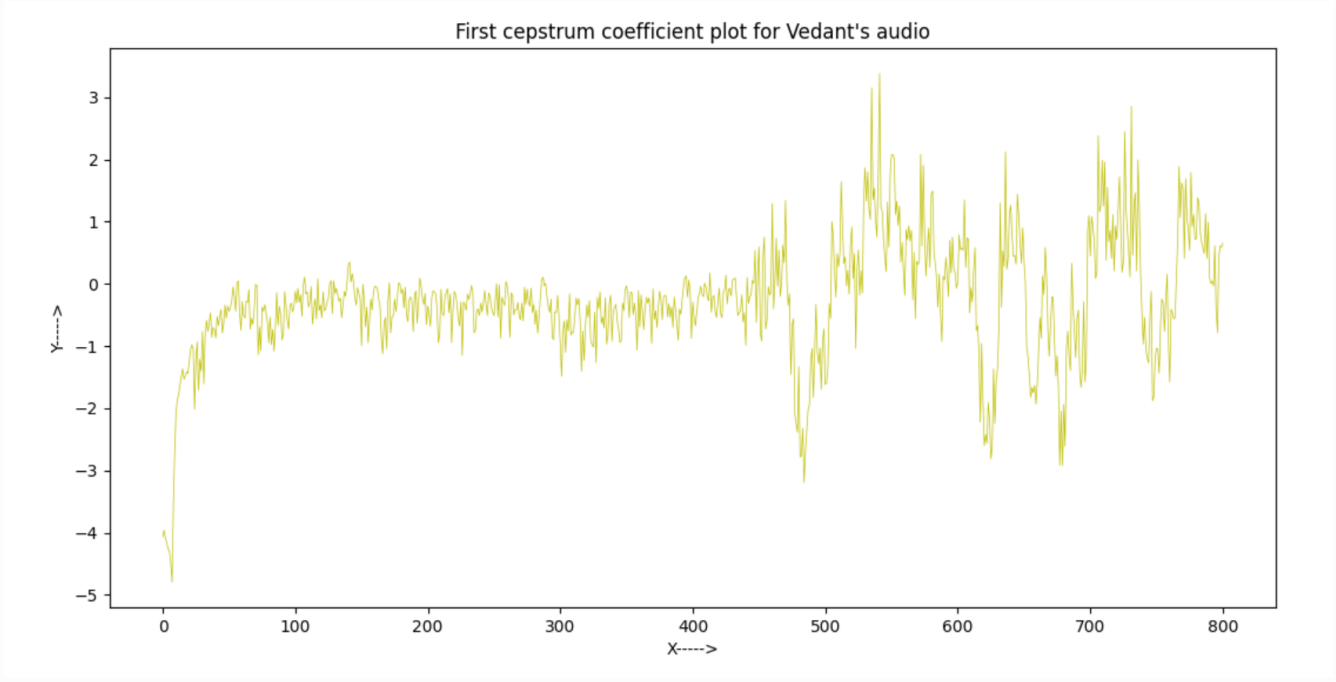


Fig 3.12

3.2.2 Model Building

From the training data set, we fit a Gaussian model to every distinct speaker. A total of 10 audio samples of every speaker is used for training the model. The feature arrays extracted in the previous step are feeded to the model framework to obtain a unique model for every speaker in the database.

PLDA technique is implemented as the back-end framework to the GMM model for providing classification. It involves computing a log-likelihood function between the input audio data and known speaker's audio data. The speaker with maximum log-likelihood value for the given audio input is thus recognized by the system.

3.4 CWT-GMM model

3.4.1 Feature Extraction Process

Usually, Fourier Transform is used to extract frequencies from a signal. The Fourier Transform uses a series of sine waves with different frequencies to analyze a signal. But it has a big drawback. It's picking the right window size[11]. According to Heisenberg's uncertainty principle:

- ➢ A narrow window will localize the signal in time but there will be significant uncertainty in frequency.

- ➢ If the window is wide enough, then the time uncertainty increases.

This is the tradeoff between time and frequency resolution.One way to avoid this problem is Multiresolution Analysis (MRA). An example of MRA is Wavelet Transform[11]. In MRA, a signal is analyzed at different resolution levels.

In layman's terms: A Fourier transform (FT) will tell you what frequencies are present in your signal. A wavelet transform (WT) will tell you what frequencies are present and where (or at what scale). If you had a signal that was changing in time, the FT wouldn't tell you when (time) this has occurred. You can also think of replacing the time variable with a space variable, with a similar analogy.

Like the Fourier transform, the continuous wavelet transform (CWT) uses inner products to measure the similarity between a signal and an analyzing function. In the Fourier transform, the analyzing functions are complex exponentials, $e^{j\omega t}$. The resulting transform is a function of a single variable, ω. In the short-time Fourier transform, the analyzing functions are windowed complex exponentials, $w(t)e^{j\omega t}$, and the result is a function of two variables. The STFT coefficients, $F(\omega,\tau)$, represent the match between the signal and a sinusoid with angular frequency ω in an interval of a specified length centered at τ.

In the CWT, the analyzing function is a wavelet, ψ. The CWT compares the signal to shifted and compressed or stretched versions of a wavelet. Stretching or compressing a function is collectively referred to as dilation or scaling and corresponds to the physical notion of scale. By comparing the signal to the wavelet at various scales and positions, you obtain a function of two variables. The 2-D representation of a 1-D signal is redundant. If the wavelet is complex-valued, the CWT is a complex-valued function of scale and position. If the signal is real-valued, the CWT is a real-valued function of scale and position. For a scale parameter, $a>0$, and position, b, the CWT is:

Continuous Wavelet Transform (CWT)

$$T(a,b) = \frac{1}{\sqrt{a}} \int_{-\infty}^{\infty} x(t)\, \psi^* \frac{(t-b)}{a} dt$$

a: Scale (or dilation) parameter
b: Location of wavelet
Ψ: Wavelet function
x: Signal

Fig 3.13

where ∗ denotes the complex conjugate. Not only do the values of scale and position affect the CWT coefficients, but the choice of wavelet also affects the values of the coefficients. Wavelet transforms can change the scale parameter(a) to find different frequencies in the signal along with their location. So, now we know which frequencies exist in the time signal and where they exist. Smaller-scale means a wavelet is squished. So, it can capture higher frequencies. On the other hand, a larger scale can capture lower frequencies. You can see in the image below an example of a squashed and stretched wavelet.

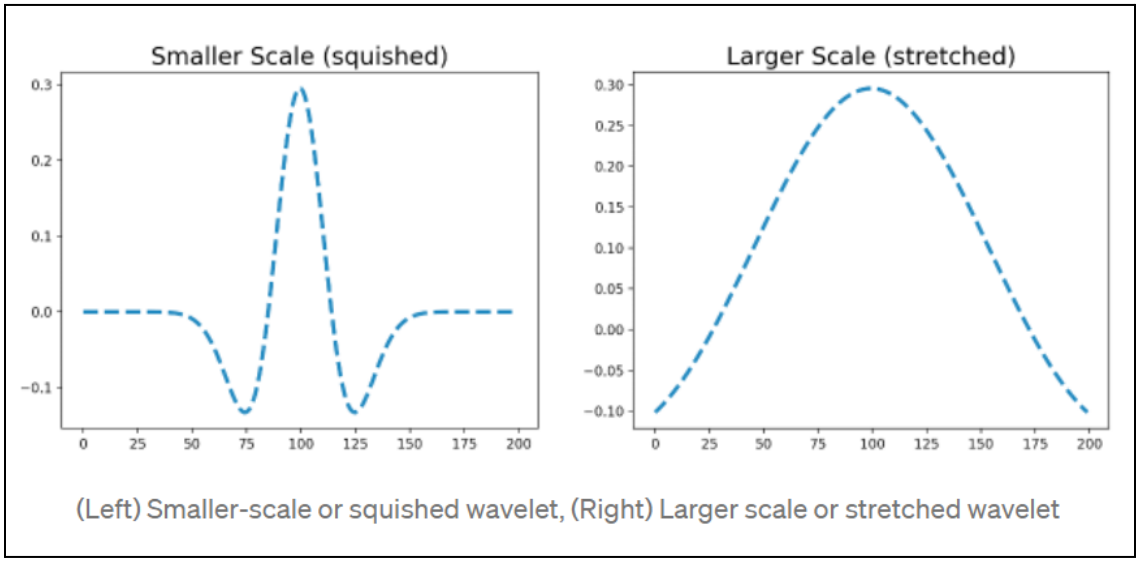


Fig 3.14

Wavelet analysis represents a windowing technique with variable-sized regions. Wavelet analysis allows the use of long time intervals where you want more precise low-frequency information, and shorter regions where you want high-frequency information.

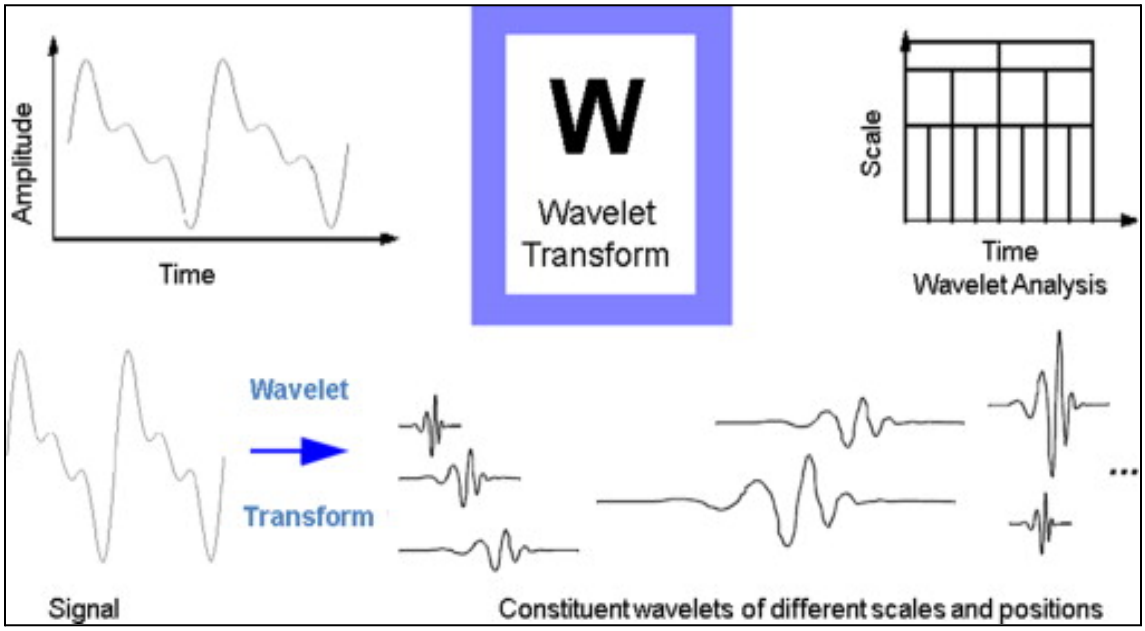


Fig 3.15

After computing wavelet features, we divide the time series into frames of length 400. The sampling rate is 8 kHz. So, 400 frames are equivalent to 50 milliseconds. We have also created a band-pass filter. We will only take frequencies between 80Hz and 1000 kHz.

The human ear can hear between 20 and 20,000 Hz (20 kHz) but it is most sensitive to everything that happens between 250 and 5,000 Hz. The voiced speech of a typical adult male will have a fundamental frequency from 85 to 180 Hz, and that of a typical adult female from 165 to 255 Hz. For a child's voice, the average fundamental frequency is 300 Hz. Consonants take up space between 2kHz and 5kHz. Vowel Sounds are prominent between 500Hz and 2kHz.

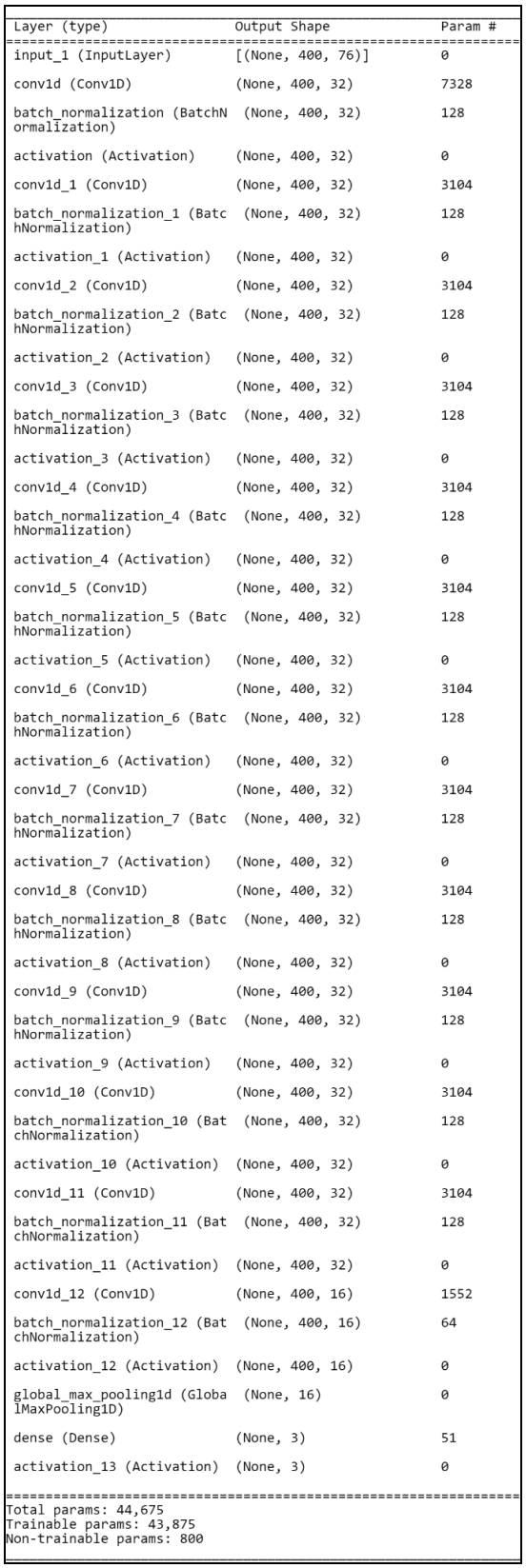
We will write all features in a .npz file so that we can easily load the features from there later. The shape of the features is (76 x 400). It's in the format: (Features x timesteps).

3.4.2 Model Building

We will use dilated 1D Convolutions along with Batch Normalization layers. We have used Time Distributed 1D Convolutions because this is a multivariate time series. Spectrograms and wavelet transforms are not standard images. In a standard image, both the x-axis and y-axis carry similar pixel content. But, in the case of spectrograms and wavelet transforms, the x-axis is time and the y-axis is frequency. dilated convolutions will help increase the reception field while keeping the number of parameters the same. This will be helpful because we have a multivariate time series of length 400. The Batch Normalization layer normalizes the mini-batches during training and solves the problem of internal covariate shift.

We will train the network using 5-fold cross-validation. We need to change the data format before starting the training process. For the deep learning model, we need the data in the format: (Num_samples x Timesteps x Features).

Firstly, we need to standardize the data using a Standard scaler. We will save the mean and standard deviation of the training data. During testing, we will use this mean and standard deviation to standardize the data and then make predictions.



Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 400, 76)]	0
conv1d (Conv1D)	(None, 400, 32)	7328
batch_normalization (BatchNormalization)	(None, 400, 32)	128
activation (Activation)	(None, 400, 32)	0
conv1d_1 (Conv1D)	(None, 400, 32)	3104
batch_normalization_1 (BatchNormalization)	(None, 400, 32)	128
activation_1 (Activation)	(None, 400, 32)	0
conv1d_2 (Conv1D)	(None, 400, 32)	3104
batch_normalization_2 (BatchNormalization)	(None, 400, 32)	128
activation_2 (Activation)	(None, 400, 32)	0
conv1d_3 (Conv1D)	(None, 400, 32)	3104
batch_normalization_3 (BatchNormalization)	(None, 400, 32)	128
activation_3 (Activation)	(None, 400, 32)	0
conv1d_4 (Conv1D)	(None, 400, 32)	3104
batch_normalization_4 (BatchNormalization)	(None, 400, 32)	128
activation_4 (Activation)	(None, 400, 32)	0
conv1d_5 (Conv1D)	(None, 400, 32)	3104
batch_normalization_5 (BatchNormalization)	(None, 400, 32)	128
activation_5 (Activation)	(None, 400, 32)	0
conv1d_6 (Conv1D)	(None, 400, 32)	3104
batch_normalization_6 (BatchNormalization)	(None, 400, 32)	128
activation_6 (Activation)	(None, 400, 32)	0
conv1d_7 (Conv1D)	(None, 400, 32)	3104
batch_normalization_7 (BatchNormalization)	(None, 400, 32)	128
activation_7 (Activation)	(None, 400, 32)	0
conv1d_8 (Conv1D)	(None, 400, 32)	3104
batch_normalization_8 (BatchNormalization)	(None, 400, 32)	128
activation_8 (Activation)	(None, 400, 32)	0
conv1d_9 (Conv1D)	(None, 400, 32)	3104
batch_normalization_9 (BatchNormalization)	(None, 400, 32)	128
activation_9 (Activation)	(None, 400, 32)	0
conv1d_10 (Conv1D)	(None, 400, 32)	3104
batch_normalization_10 (BatchNormalization)	(None, 400, 32)	128
activation_10 (Activation)	(None, 400, 32)	0
conv1d_11 (Conv1D)	(None, 400, 32)	3104
batch_normalization_11 (BatchNormalization)	(None, 400, 32)	128
activation_11 (Activation)	(None, 400, 32)	0
conv1d_12 (Conv1D)	(None, 400, 16)	1552
batch_normalization_12 (BatchNormalization)	(None, 400, 16)	64
activation_12 (Activation)	(None, 400, 16)	0
global_max_pooling1d (GlobalMaxPooling1D)	(None, 16)	0
dense (Dense)	(None, 3)	51
activation_13 (Activation)	(None, 3)	0

Total params: 44,675
Trainable params: 43,875
Non-trainable params: 800

Figure 3.16

3.4.3 Model Flowchart

1) We have used a morlet waveform and set the sampling frequency to 8kHz.
2) We divide the time series into frames of length 400.
3) Then we have obtained scales for the morlet waveform in the range 1 to 64 and obtained the frequencies associated with it.
4) The sampling rate is 8 kHz. So, 400 frames are equivalent to 50 milliseconds. We have also created a band-pass filter. We will only take frequencies between 80Hz and 1000 kHz.
5) We will write all features in a .npz file so that we can easily load the features from there later. The shape of the features is (76 x 400). It's in the format: (Features x timesteps).
6) We have also saved the unique id of each sample. It is important because we are dividing each sample into multiple frames and we need to know which frame belongs to which sample. All frames from one sample will have the same id.
7) Then we compute the continuous wavelet transform using an audio numpy array with the wavelet set as morlet and the sampling period of 8kHz.
8) We took random samples from training data and some random frames from each sample in order to decrease the training data. The data is in the format: (Num_samples x Features x timesteps)
9) Then we pass this to the Deep Learning model.

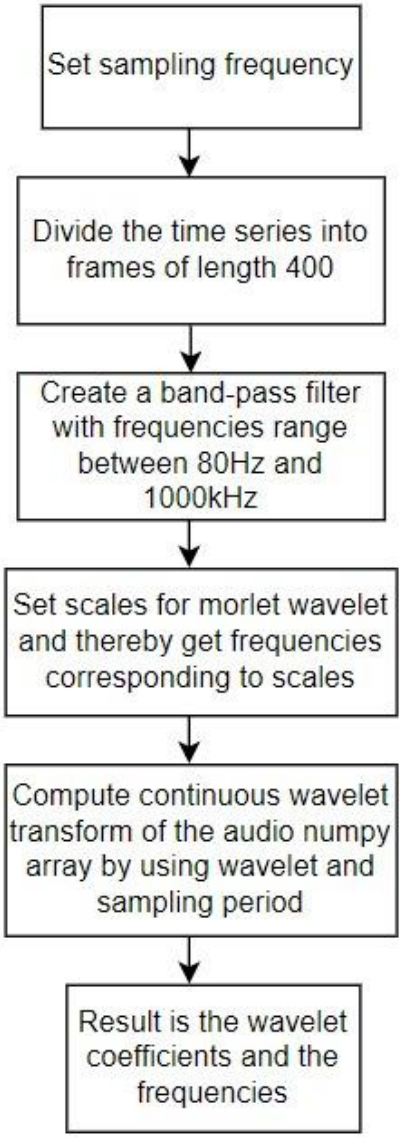


Fig 3.16

3.5 SincNet model

Speaker identification is a prominent research area with a variety of applications including forensics and biometric authentication. Many speaker identification systems depend on precomputed features such as i-vectors or MFCCs, which are then fed into machine learning or deep learning networks for classification. Other deep learning speech systems bypass the feature extraction stage and feed the audio signal directly to the network. In such end-to-end systems, the network directly learns low-level audio signal characteristics.

In our SincNet implementation, we first implement a traditional end-to-end speaker identification CNN. The filters learned tend to have random shapes that do not correspond to perceptual evidence or knowledge of how the human ear works, correlating to our scenario where the amount of training data is limited. We then replace the first convolutional layer in the network with a custom sinc filterbank layer that introduces structure and constraints based on perceptual evidence.

This sinc layer convolves the input frames with a bank of fixed bandpass filters. The bandpass filters are a linear combination of two sinc filters in the time domain. The frequencies of the bandpass filters are spaced linearly on the mel scale.

Finally, we train this SincNet architecture, which adds learnability to the sinc filter bank parameters, which implies it tries to learn better parameters for these bandpass filters within the neural network framework.

3.5.1 SincNet Feature Extraction Process

CNN (Convolutional Neural Networks) has also come into the mix to capture important speech-features, exhibiting better performance than the aforementioned hand-crafted features.
When we feed in the raw speech inputs in CNN, it learns to capture low-level speech representations, which helps in learning important narrow-band speaker characteristics such as pitch and formants. Speaker related information is captured at a lower frequency region. Thus, CNN should be designed in a way that it can capture meaningful features.

According to [8], the first Convolution layer of the current waveform CNN is the most important as it gets high dimensional inputs. It also gets affected by the Vanishing Gradient.To overcome this in SincNet, we have parameterized sinc functions to implement band-pass filters.
The low and high cutoff frequencies are the only parameters of the filter learned from data. This solution still offers considerable flexibility, but forces the network to focus on high-level tunable parameters with a broad impact on the shape and bandwidth of the resulting filter.

3.5.2 SincNet Architecture

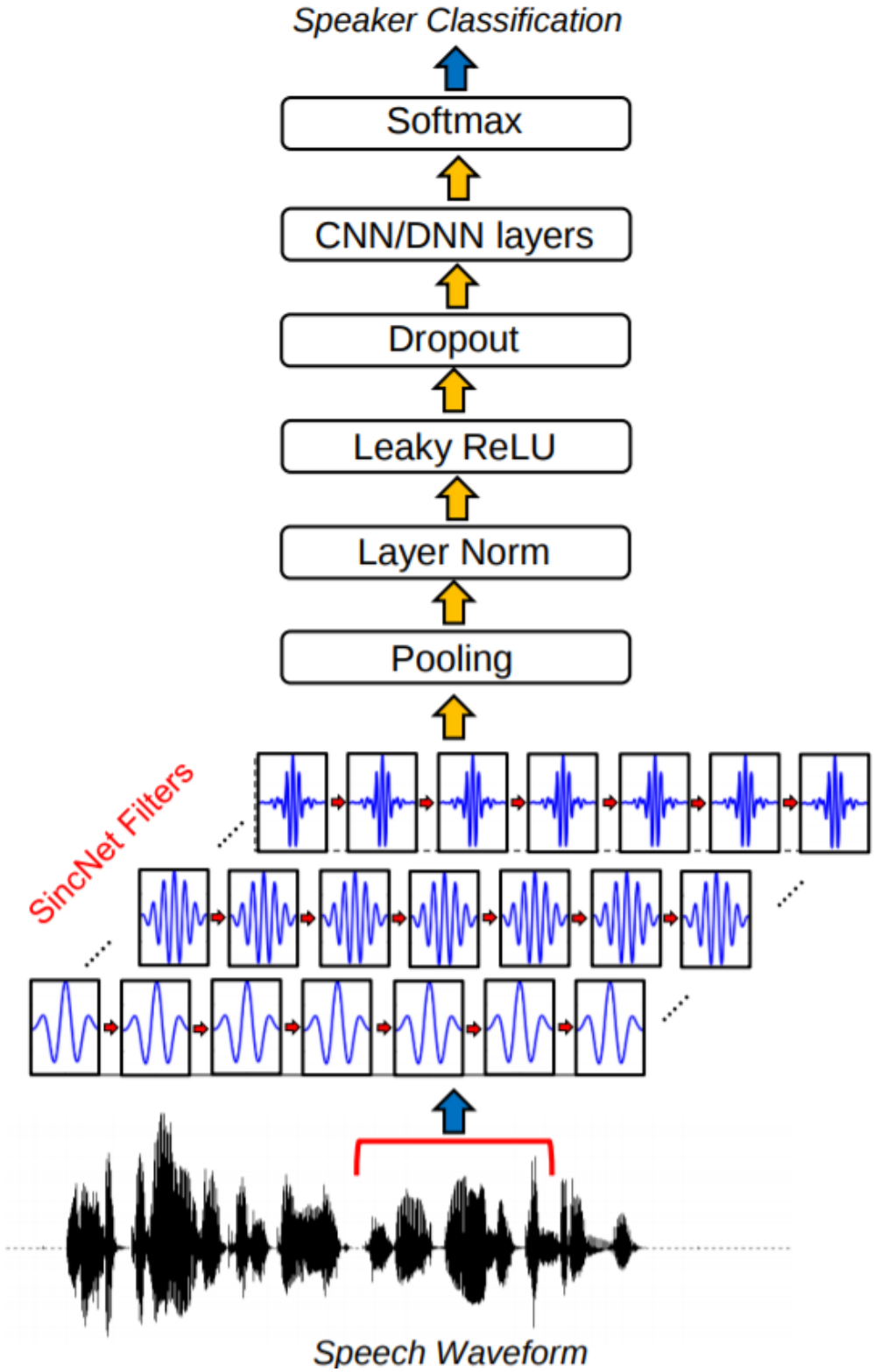


Fig 3.17

We take a standard CNN which performs time-domain convolutions(Convolution in time domain equals multiplication in the frequency domain) between input waveform and some _Finite Impulse Response(FIR) filters as given below.

$$y[n] = x[n] * h[n] = \sum_{l=0}^{L-1} x[l] \cdot h[n-l] \qquad (1)$$

Here,

x[n] = chunk of speech signal,

h[n] is a filter of length **L,**

y[n] is the output.

We can keep this chunk based on experiments with the dataset and all **L** elements of each filter are learned from the data.

SincNet performs its convolutions on function **g** which depends on **Theta**.

$$y[n] = x[n] * g[n, \theta] \qquad (2)$$

In digital signal processing, **g** is defined such that a **filter-bank** composed of rectangular bandpass filters is employed. In the frequency domain, the magnitude of a generic bandpass filter can be written as the difference between two low-pass filters.

Here **f1** and **f2** are the learned low and high cutoff frequencies, and **rect(·)** is the rectangular function in the magnitude frequency domain.

$$G[f, f_1, f_2] = rect\left(\frac{f}{2f_2}\right) - rect\left(\frac{f}{2f_1}\right), \qquad (3)$$

Sinc function is defined as **sinc(x) = sin(x)/x**.

After returning to the time domain(using the inverse Fourier Transform), the reference function g becomes:

$$g[n, f_1, f_2] = 2f_2 sinc(2\pi f_2 n) - 2f_1 sinc(2\pi f_1 n), \quad (4)$$

The major features which determine a unique speaker are captured in the lower frequency range. In the above equation, **fs** is equal to the sampling frequency of the input signal and cut-off frequency is initialized randomly in the range [0, fs/2].

The sampling frequency may vary with the type of data you are experimenting with. An IVR system has a sampling frequency of 8Khz, whereas a stereo system has a sampling frequency of 44khz.

We can initialize the filters based on the cut-off frequencies of the mel-scale filter-bank. The major advantage of assigning filters this way is that it has the advantage of directly allocating more filters at the lower part of the spectrum which has unique information of speaker voices.

To ensure f1 ≥ 0 and f2 ≥ f1, the previous equation is fed by the following parameters:

$$f_1^{abs} = |f_1| \quad (5)$$

$$f_2^{abs} = f_1 + |f_2 - f_1| \quad (6)$$

Here we keep no bounds on **f2,** i.e., no force is imposed on f2 to be smaller than the Nyquist frequency(the minimum rate at which a signal can be sampled without introducing errors, which is twice the highest frequency present in the signal) as the model learns this while training. Different subsequent layers decide to give more or less importance to each filter output.

An infinite number of elements **L** are required by an ideal bandpass filter. An ideal bandpass filter is where the passband is perfectly flat, and the attenuation in the stopband is infinite. Any truncation of ***g*** thus inevitably leads to an approximation of the ideal filter, characterized by ripples in the passband and limited attenuation in the stopband.

So, windowing is performed to solve this issue. It is performed just by multiplying the truncated function **g** with a window function **w**, which aims to smooth out the abrupt discontinuities at the ends of g:

$$g_w[n, f_1, f_2] = g[n, f_1, f_2] \cdot w[n]. \qquad (7)$$

This paper uses the popular Hamming window, defined as follows:

$$w[n] = 0.54 - 0.46 \cdot cos\left(\frac{2\pi n}{L}\right). \qquad (8)$$

We can get high-frequency selectivity with the use of the Hamming window. We can use other windows too. One important note here is that due to the symmetry, the filters can be computed efficiently by considering one half of the filter and inheriting the results for the other half.

3.5.3 Model Building

We built a dataset consisting of 10 training samples of each speaker and 5 testing samples of each speaker. We are taking the paths of all speakers and converting them into arrays. We also created a dictionary of speaker Ids where each ID number is assigned to a speaker .All the speaker arrays are combined along with their labels and are further passed for training the model. The Model consists of a Sinclayer which has the following parameters:

- No. of Filters = 64
- Sampling rate = 16KHz

- Filter dimension = 129

The output of the Sinclayer is then passed as an input for the Normalization layer. Then an activation function (LeakyRelu) is applied on the output which is then passed to the 1 dimensional Max Pooling layer. The Max pooling layer has a pool size of 2. The output from this set of layers is then passed on to 4 1-dimensional convolution layers which have different numbers of filters. The output from these layers is flattened i.e the array is converted into 1 dimension. This output is then passed to a set of 2 Dense layers. The final dense layer gives us the result. The activation function used here is 'softmax' which gives us the probability of all predicted speakers. The label with highest probability is stored as a result. The speaker with the respective stored label is the one detected by our model. The 3D representation of the model is shown below.

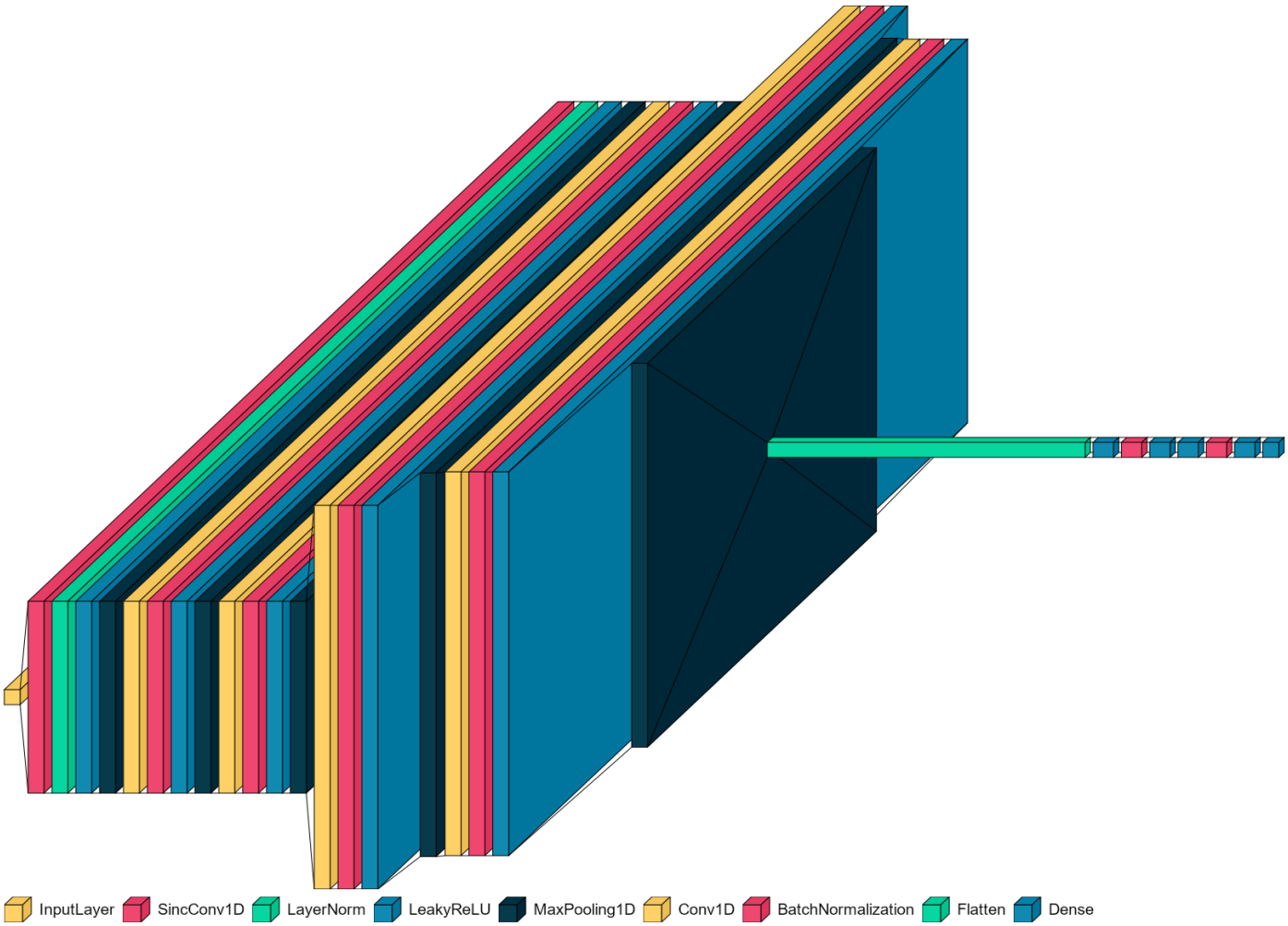


Fig 3.18

3.5.4 Model Properties

Fast Convergence: SincNet is designed in such a way that it forces the network to focus on filter parameters which impacts its performance. This style of filtering technique helps to adapt to data while capturing knowledge just like feature extraction techniques on audio data. This prior knowledge makes learning the filter characteristics much easier, helping SincNet to converge significantly faster to a better solution. We get fast convergence within the first 10–15 epochs.

Less Number of parameters for training:SincNet drastically reduces the number of parameters in the first convolutional layer. For instance, if we consider a layer composed of F filters of length L, a standard CNN employs $F \cdot L$ parameters, against the 2F considered by SincNet. If $F = 80$ and $L = 100$, we employ 8k parameters for the CNN and only 160 for SincNet. Moreover, if we double the filter length L, a standard CNN doubles its parameter count (e.g., we go from 8k to 16k), while SincNet has an unchanged parameter count (only two parameters are employed for each filter, regardless of its length L). This offers the possibility to derive very selective filters with many taps, without actually adding parameters to the optimization problem. Moreover, the compactness of the SincNet architecture makes it suitable in the few sample regimes.

Interpretability: The SincNet feature maps obtained in the first convolutional layer are definitely more interpretable and human-readable than other approaches. The filter bank, in fact, only depends on parameters with a clear physical meaning.

3.5.5 Model Flowchart

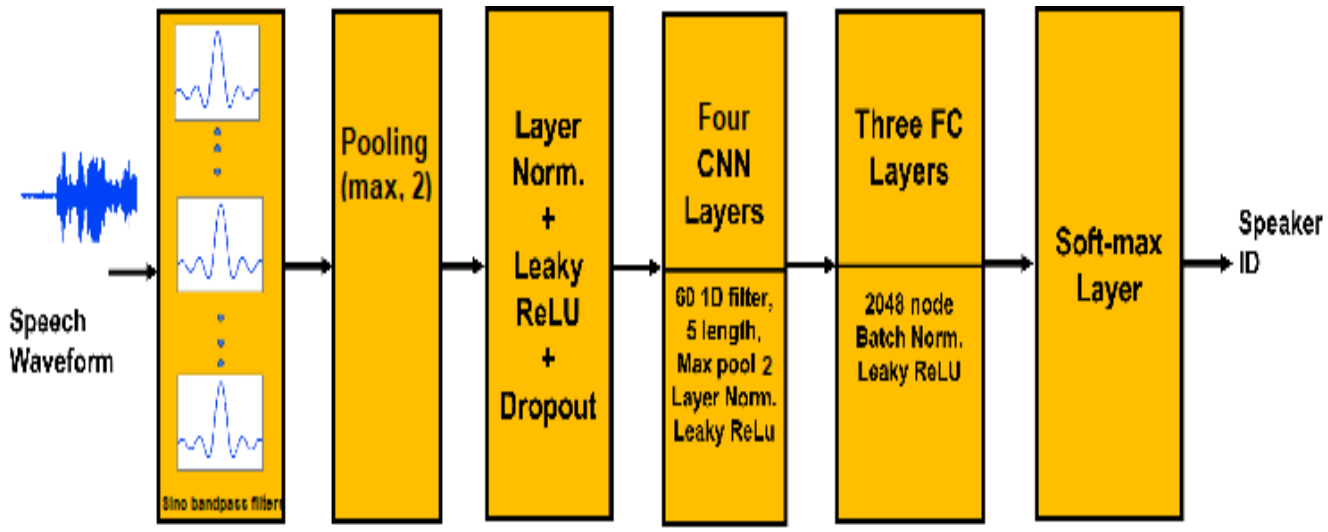


Fig 3.19

A simplified workflow of the algorithm is as follows:

- The data is loaded in the given format and preprocessed according to our requirement.
- A standard 4-layer CNN model is implemented, with 1-D convolutional and max-pooling layers, and with Leaky ReLu as the activation function.
- We then replace the first convolutional layer in the standard CNN with a convolutional sinc filterbank layer.
- We then train the neural network for 10 epochs using adam optimization. Shuffle the training data before every epoch. We also use the test data as the validation data to observe how the network performance improves as training progresses.

CHAPTER 4: RESULT AND ANALYSIS

4.1 Results

4.1.1 MFCC-GMM

Five audio files of every speaker have been taken for testing the GMM model. We have implemented labeled testing using four audio files, and unlabeled testing with the remnant audio file. We have also incorporated testing on a live-audio sample for speaker identification among the speakers in our dataset. An 8-sec audio file is recorded, and is evaluated against the trained models of every speaker. The speaker model which fits closest to the feature structure of the input file (i.e. which produces lowest value of loss function), is thus assigned as the identified speaker.

A GUI framework is built to interact with the system, opt for a suitable testing environment (i.e. labeled or unlabeled data file) and to obtain a vivid display of the results.

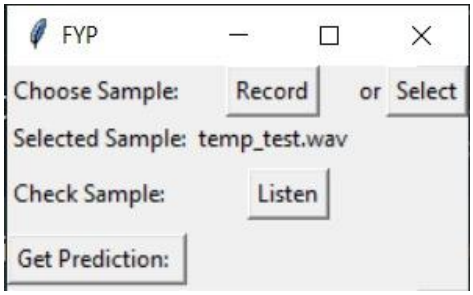


Fig 4.1

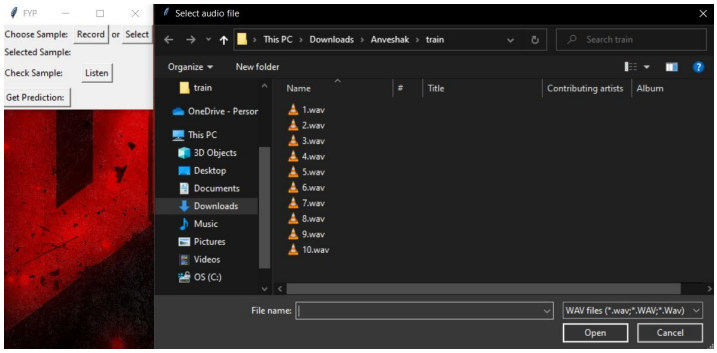


Fig 4.2

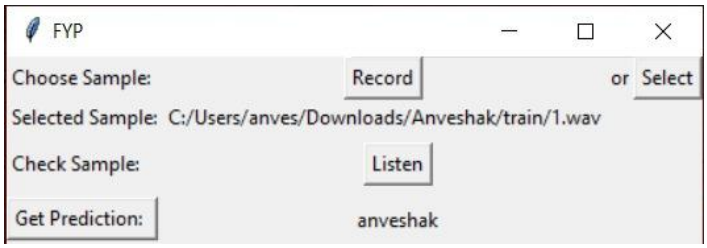


Fig 4.3

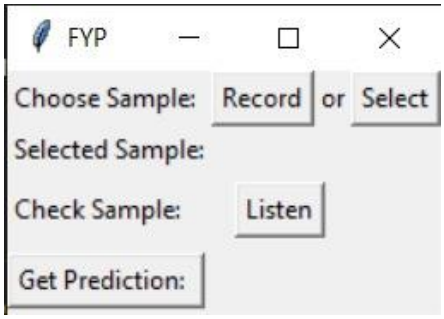


Fig 4.4

4.1.2 CWT-GMM

The performance metrics of the model are as follows:

	precision	recall	f1-score	support
speaker1	0.33	1.00	0.50	3
speaker2	0.00	0.00	0.00	3
speaker3	0.00	0.00	0.00	3
accuracy			0.33	9
macro avg	0.11	0.33	0.17	9
weighted avg	0.11	0.33	0.17	9

Fig 4.5

4.1.3 SincNet

The accuracy of the Trained Model obtained on the testing dataset was 0.9615 i.e 96.15%.

The validation loss was 0.205

The GUI implementation of SincNet model is shown below:

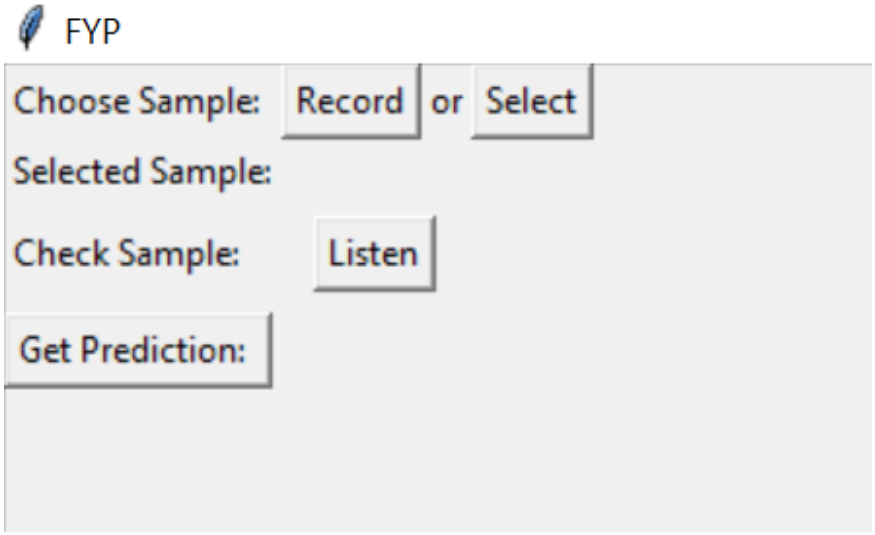


Fig 4.6

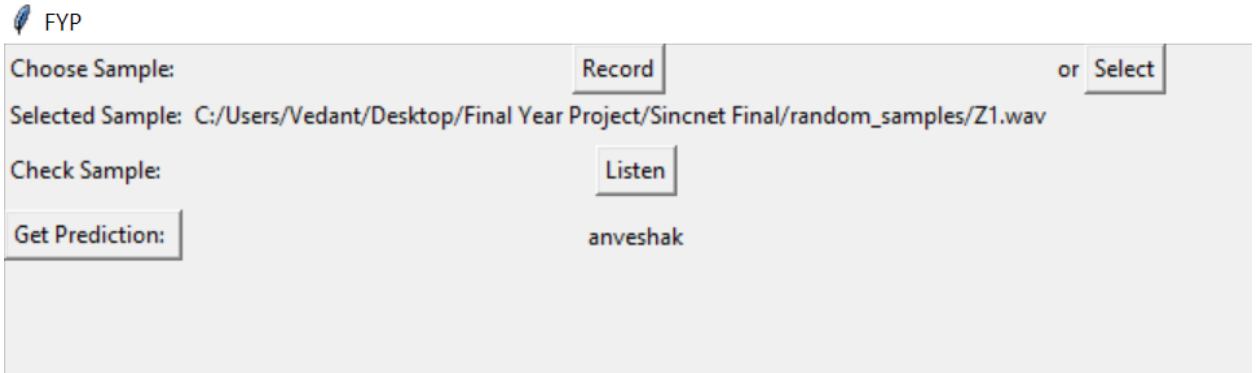


Fig 4.7

4.2 Comparative Analysis

In the preceding chapters, we have taken into consideration two models - **SincNet** and **Gaussian Mixture Models (GMMs)** - for our use-case. We have considered a real-life scenario, where the training dataset size is minimalistic and the training samples are short enough for a user to realistically generate. Keeping this in mind, we can go ahead and compare the metrics of the three models i.e. training, testing and prediction.

4.2.1 SincNet

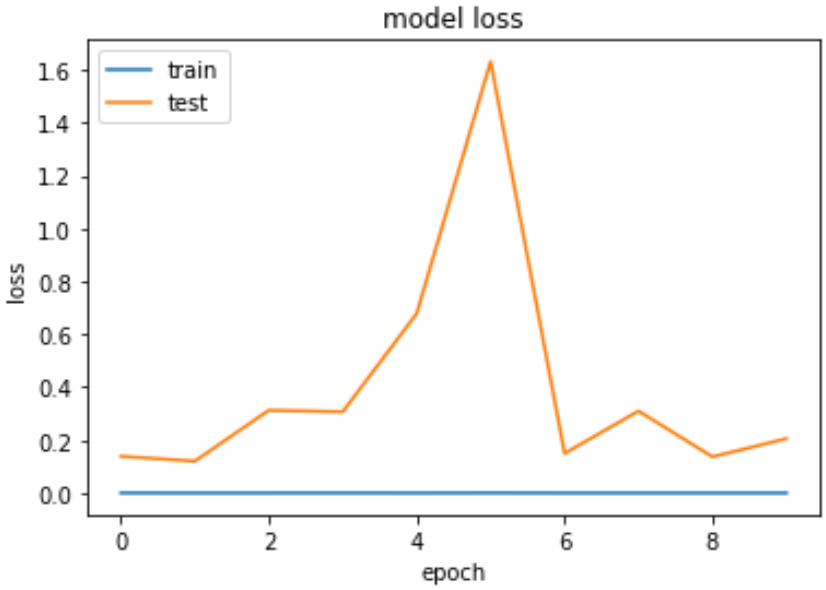


Fig 4.8

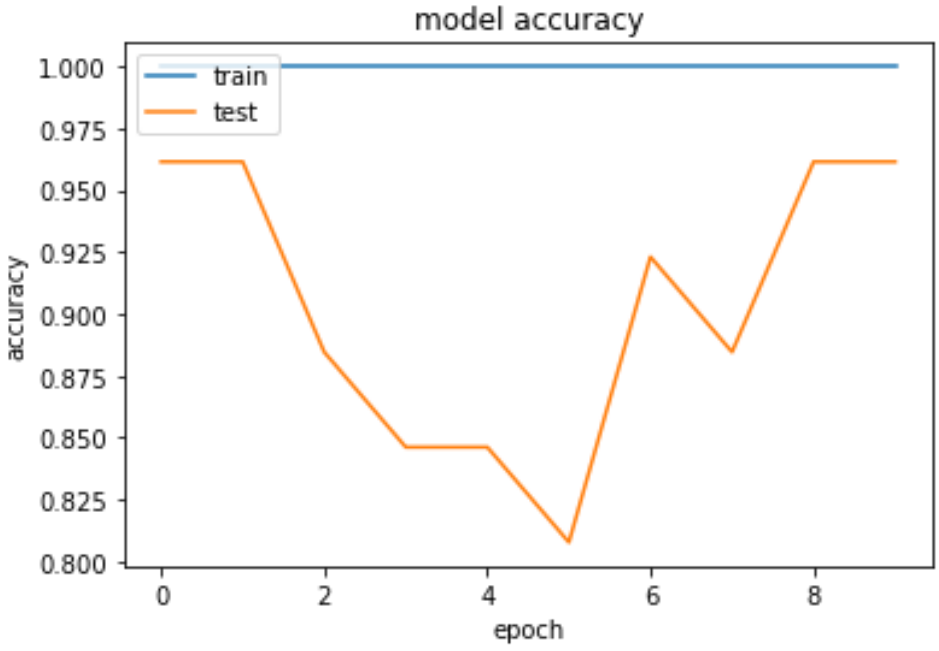


Fig 4.9

Model Evaluation

```
In [57]: model.evaluate(test_data,test_labels)

         1/1 [==============================] - 0s 284ms/step - loss: 0.2055 - accuracy: 0.9615

Out[57]: [0.20547880232334137, 0.9615384340286255]
```

Fig 4.10

In the above figures we can see that the accuracy and loss stabilize at great values in their respective charts, with no indication of overfitting or underfitting. This demonstrates the most attractive quality of SincNet - its ability to perform adequately on minimal data. The accuracy varies from a minimum of ~80% to a maximum of 96.15%, and the loss dips to a maximum of 0.2.

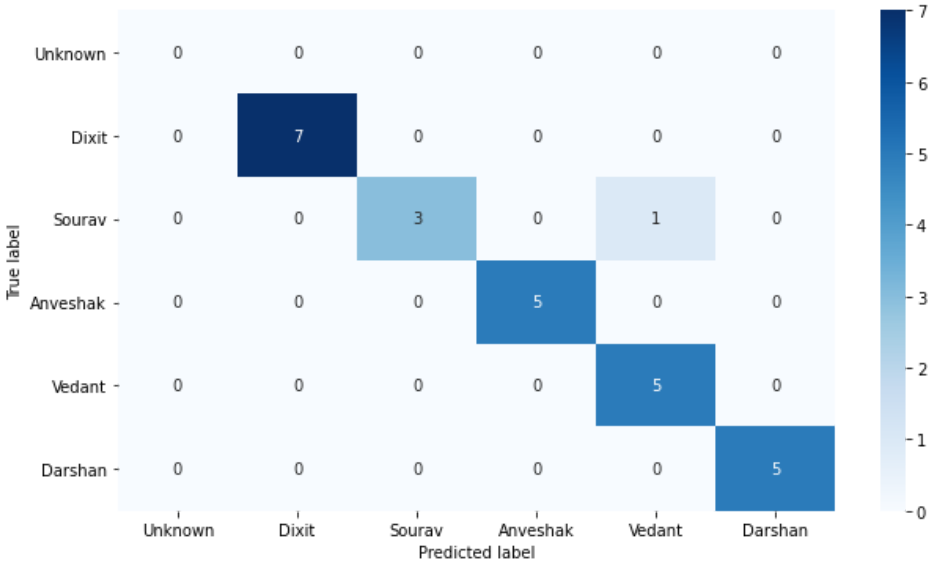


Fig 4.11

On examining the above confusion matrix, we can see that SincNet performs equally well during prediction with the predicted labels matching the test labels for almost every single prediction class that is defined.

	precision	recall	f1-score	support
1	1.00	1.00	1.00	7
2	1.00	0.75	0.86	4
3	1.00	1.00	1.00	5
4	0.83	1.00	0.91	5
5	1.00	1.00	1.00	5
accuracy			0.96	26
macro avg	0.97	0.95	0.95	26
weighted avg	0.97	0.96	0.96	26

Fig 4.12

From the above classification report, we can see the accuracy reports. It displays the Precision, Recall, F1-scores and Support value metrics for each classification class. Precision is the ratio of correctly predicted positive observations to the total predicted positive observations. Recall is the ratio of correctly predicted positive observations to the all observations in actual class. F1 Score is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account. Support is the number of actual occurrences of the class in the specified dataset.

The precision for the SincNet model shows that the positive observations were almost always correct, with a score of almost a perfect 1. The recall is similarly impressive, with almost all the positives being correctly guessed with a score of 1 again.

4.2.2 MFCC-GMM

We have implemented GMMs with two different feature extraction techniques. For this section, we will take into consideration the Mel-Frequency Cepstral Coefficients (MFCCs).

```
medee@LAPTOP-V2KCAVA2 MINGW64 ~/Documents/COC/Speaker-Recognition-FYP/GMM-Final
$ python TrainingModel.py
Anveshak-001/Anveshak_1.wav
Anveshak-001/Anveshak_2.wav
Anveshak-001/Anveshak_3.wav
Anveshak-001/Anveshak_4.wav
Anveshak-001/Anveshak_5.wav
Anveshak-001/Anveshak_6.wav
Anveshak-001/Anveshak_7.wav
Anveshak-001/Anveshak_8.wav
Anveshak-001/Anveshak_9.wav
Anveshak-001/Anveshak_10.wav
+ modeling completed for speaker: Anveshak.gmm  with data point =  (8157, 40)
Darshan-002/Darshan_1.wav
Darshan-002/Darshan_2.wav
Darshan-002/Darshan_3.wav
Darshan-002/Darshan_4.wav
Darshan-002/Darshan_5.wav
Darshan-002/Darshan_6.wav
Darshan-002/Darshan_7.wav
Darshan-002/Darshan_8.wav
Darshan-002/Darshan_9.wav
Darshan-002/Darshan_10.wav
+ modeling completed for speaker: Darshan.gmm  with data point =  (19599, 40)
Dixit-003/Dixit_1.wav
Dixit-003/Dixit_2.wav
Dixit-003/Dixit_3.wav
Dixit-003/Dixit_4.wav
Dixit-003/Dixit_5.wav
Dixit-003/Dixit_6.wav
Dixit-003/Dixit_7.wav
Dixit-003/Dixit_8.wav
Dixit-003/Dixit_9.wav
Dixit-003/Dixit_10.wav
+ modeling completed for speaker: Dixit.gmm  with data point =  (7990, 40)
Sourav-004/Sourav_1.wav
Sourav-004/Sourav_2.wav
Sourav-004/Sourav_3.wav
Sourav-004/Sourav_4.wav
Sourav-004/Sourav_5.wav
Sourav-004/Sourav_6.wav
Sourav-004/Sourav_7.wav
Sourav-004/Sourav_8.wav
Sourav-004/Sourav_9.wav
Sourav-004/Sourav_10.wav
+ modeling completed for speaker: Sourav.gmm  with data point =  (16598, 40)
Vedant-005/Vedant_1.wav
Vedant-005/Vedant_2.wav
Vedant-005/Vedant_3.wav
Vedant-005/Vedant_4.wav
Vedant-005/Vedant_5.wav
Vedant-005/Vedant_6.wav
Vedant-005/Vedant_7.wav
Vedant-005/Vedant_8.wav
Vedant-005/Vedant_9.wav
Vedant-005/Vedant_10.wav
+ modeling completed for speaker: Vedant.gmm  with data point =  (16367, 40)
```

Fig 4.13

In the above picture, we can see the modeling characteristics for each user. The output for each user is shown as (number of data points, number of coefficients). As we can see, the number of data points for different users varies by a lot, with it ranging from 8157 for Anveshak upto 19599 for Darshan. This is a consequence of the quality of audio files submitted to the model for training. The audio files for Darshan may contain more ambient noise than that of Anveshak, who may have recorded his audio in a quiet environment. This leads to data that has less data points, but more focus on the actual data for Anveshak, and conversely more data points for Darshan but with the actual data being blurred by the noise. This shows the dependence of the GMM model on the quality of the audio samples.

```
error-0         total-sample-20.0
The Accuracy Percentage for the current testing Performance with MFCC + GMM is :  100.0 %
Speaker Identified Successfully
```

Fig 4.14

Taking into consideration now the accuracy metrics for GMM, we can straightaway notice an issue with the accuracy value. The model returns a 100% accuracy, which is a clear sign of overfitting taking place. Overfitting occurs when a statistical model fits exactly against its training data. When this happens, the algorithm unfortunately cannot perform accurately against unseen data, defeating its purpose.

```
$ python main.py
Press '0' for testing a complete set of audio with Accuracy or Press '1' for checking a single Audio or Press '2' for testing live audio (microphone required):
0
Testing Audio :  Anveshak/Anveshak_11.wav
        detected as -  Anveshak
Testing Audio :  Anveshak/Anveshak_12.wav
        detected as -  Anveshak
Testing Audio :  Anveshak/Anveshak_13.wav
        detected as -  Anveshak
Testing Audio :  Anveshak/Anveshak_14.wav
        detected as -  Anveshak
Testing Audio :  Darshan/Darshan_11.wav
        detected as -  Darshan
Testing Audio :  Darshan/Darshan_12.wav
        detected as -  Darshan
Testing Audio :  Darshan/Darshan_13.wav
        detected as -  Darshan
Testing Audio :  Darshan/Darshan_14.wav
        detected as -  Darshan
Testing Audio :  Dixit/Dixit_11.wav
        detected as -  Dixit
Testing Audio :  Dixit/Dixit_12.wav
        detected as -  Dixit
Testing Audio :  Dixit/Dixit_13.wav
        detected as -  Dixit
Testing Audio :  Dixit/Dixit_14.wav
        detected as -  Dixit
Testing Audio :  Sourav/Sourav_11.wav
        detected as -  Sourav
Testing Audio :  Sourav/Sourav_12.wav
        detected as -  Sourav
Testing Audio :  Sourav/Sourav_13.wav
        detected as -  Sourav
Testing Audio :  Sourav/Sourav_14.wav
        detected as -  Sourav
Testing Audio :  Vedant/Vedant_11.wav
        detected as -  Vedant
Testing Audio :  Vedant/Vedant_12.wav
        detected as -  Vedant
Testing Audio :  Vedant/Vedant_13.wav
        detected as -  Vedant
Testing Audio :  Vedant/Vedant_14.wav
        detected as -  Vedant
```

Fig 4.15

In the above image, we can see that the model predicts the testing data with 100% accuracy, but when we use the model to predict on audio samples that are recorded live using the GUI, then it is observed that the prediction accuracy is terrible, with the model giving the same output for every audio sample given to it.

```
$ python main.py
Press '0' for testing a complete set of audio with Accuracy or Press '1' for checking a single Audio or Press '2' for te
sting live audio (microphone required):
2
Please speak......
output.wav succesfully saved in Testing_Audio folder
('Testing Audio : ', 'output.wav')
        The live audio sample of person is detected as -  Anveshak
```

Fig 4.16

The audio sample for this was submitted by Dixit, but the predicted result was Anveshak

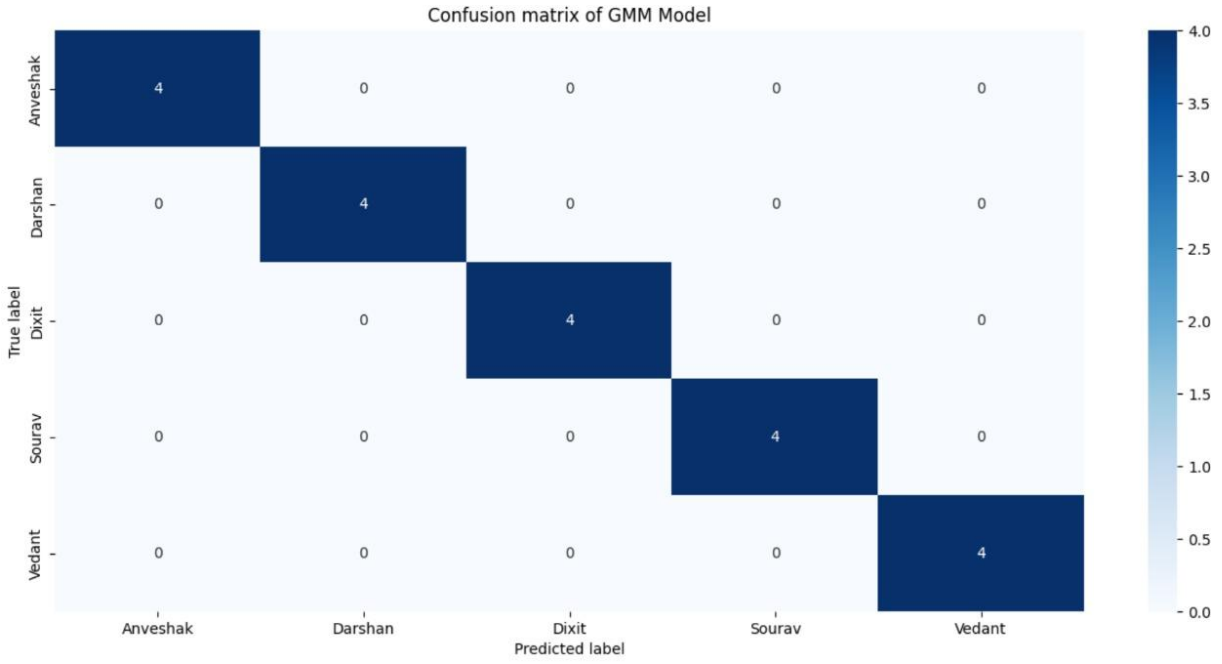


Fig 4.17

	precision	recall	f1-score	support
Anveshak	1.00	1.00	1.00	4
Darshan	1.00	1.00	1.00	4
Dixit	1.00	1.00	1.00	4
Sourav	1.00	1.00	1.00	4
Vedant	1.00	1.00	1.00	4
accuracy			1.00	20
macro avg	1.00	1.00	1.00	20
weighted avg	1.00	1.00	1.00	20

Fig 4.18

The confusion matrix and accuracy metrics of the MFCC-GMM model also reflects the same results. The confusion matrix is an identity matrix, which shows that every single test data is perfectly predicted by the model with no error, and all the accuracy metrics are perfect scores as well. All of this evidence points to clear overfitting on the dataset.

We can see that for GMM, the dataset size has to be considerable for it to be viable as a speaker identification model. Also the audio quality has to be taken into consideration as well while training the model or asking for predictions, as the same speaker on different audio input devices gives different outputs.

4.2.3 CWT-GMM

The CWT-GMM model implementation that we have used takes the audio samples of three users and trains each one of them separately. The training accuracy for the same is shown in the image below:

```
Epoch 1/20
2022-05-26 03:41:20.056121: I tensorflow/stream_executor/cuda/cuda_dnn.cc:368] Loaded cuDNN version 8100
6/6 - 6s - loss: 1.4871 - val_loss: 1.4633 - 6s/epoch - 957ms/step
Epoch 2/20
6/6 - 0s - loss: 1.4473 - val_loss: 1.4255 - 188ms/epoch - 31ms/step
Epoch 3/20
6/6 - 0s - loss: 1.4109 - val_loss: 1.3909 - 189ms/epoch - 32ms/step
Epoch 4/20
6/6 - 0s - loss: 1.3772 - val_loss: 1.3594 - 182ms/epoch - 30ms/step
Epoch 5/20
6/6 - 0s - loss: 1.3469 - val_loss: 1.3308 - 183ms/epoch - 30ms/step
Epoch 6/20
6/6 - 0s - loss: 1.3194 - val_loss: 1.3051 - 185ms/epoch - 31ms/step
Epoch 7/20
6/6 - 0s - loss: 1.2947 - val_loss: 1.2820 - 188ms/epoch - 31ms/step
Epoch 8/20
6/6 - 0s - loss: 1.2725 - val_loss: 1.2613 - 179ms/epoch - 30ms/step
Epoch 9/20
6/6 - 0s - loss: 1.2526 - val_loss: 1.2428 - 187ms/epoch - 31ms/step
Epoch 10/20
6/6 - 0s - loss: 1.2348 - val_loss: 1.2262 - 184ms/epoch - 31ms/step
Epoch 11/20
6/6 - 0s - loss: 1.2189 - val_loss: 1.2114 - 185ms/epoch - 31ms/step
Epoch 12/20
6/6 - 0s - loss: 1.2046 - val_loss: 1.1982 - 207ms/epoch - 35ms/step
Epoch 13/20
6/6 - 0s - loss: 1.1919 - val_loss: 1.1865 - 211ms/epoch - 35ms/step
Epoch 14/20
6/6 - 0s - loss: 1.1806 - val_loss: 1.1760 - 205ms/epoch - 34ms/step
Epoch 15/20
6/6 - 0s - loss: 1.1707 - val_loss: 1.1668 - 213ms/epoch - 36ms/step
Epoch 16/20
6/6 - 0s - loss: 1.1617 - val_loss: 1.1585 - 211ms/epoch - 35ms/step
Epoch 17/20
6/6 - 0s - loss: 1.1538 - val_loss: 1.1513 - 208ms/epoch - 35ms/step
Epoch 18/20
6/6 - 0s - loss: 1.1467 - val_loss: 1.1448 - 212ms/epoch - 35ms/step
Epoch 19/20
6/6 - 0s - loss: 1.1406 - val_loss: 1.1391 - 203ms/epoch - 34ms/step
Epoch 20/20
6/6 - 0s - loss: 1.1350 - val_loss: 1.1341 - 198ms/epoch - 33ms/step
Metrics :  1.1406753063201904
```

Fig 4.19

```
Epoch 1/20
6/6 - 0s - loss: 1.1301 - val_loss: 1.1297 - 218ms/epoch - 36ms/step
Epoch 2/20
6/6 - 0s - loss: 1.1258 - val_loss: 1.1258 - 187ms/epoch - 31ms/step
Epoch 3/20
6/6 - 0s - loss: 1.1221 - val_loss: 1.1223 - 185ms/epoch - 31ms/step
Epoch 4/20
6/6 - 0s - loss: 1.1188 - val_loss: 1.1193 - 186ms/epoch - 31ms/step
Epoch 5/20
6/6 - 0s - loss: 1.1158 - val_loss: 1.1167 - 179ms/epoch - 30ms/step
Epoch 6/20
6/6 - 0s - loss: 1.1132 - val_loss: 1.1144 - 188ms/epoch - 31ms/step
Epoch 7/20
6/6 - 0s - loss: 1.1109 - val_loss: 1.1124 - 181ms/epoch - 30ms/step
Epoch 8/20
6/6 - 0s - loss: 1.1090 - val_loss: 1.1107 - 182ms/epoch - 30ms/step
Epoch 9/20
6/6 - 0s - loss: 1.1073 - val_loss: 1.1092 - 181ms/epoch - 30ms/step
Epoch 10/20
6/6 - 0s - loss: 1.1057 - val_loss: 1.1078 - 183ms/epoch - 30ms/step
Epoch 11/20
6/6 - 0s - loss: 1.1044 - val_loss: 1.1067 - 206ms/epoch - 34ms/step
Epoch 12/20
6/6 - 0s - loss: 1.1033 - val_loss: 1.1057 - 201ms/epoch - 34ms/step
Epoch 13/20
6/6 - 0s - loss: 1.1023 - val_loss: 1.1049 - 204ms/epoch - 34ms/step
Epoch 14/20
6/6 - 0s - loss: 1.1013 - val_loss: 1.1042 - 205ms/epoch - 34ms/step
Epoch 15/20
6/6 - 0s - loss: 1.1006 - val_loss: 1.1035 - 208ms/epoch - 35ms/step
Epoch 16/20
6/6 - 0s - loss: 1.0998 - val_loss: 1.1030 - 211ms/epoch - 35ms/step
Epoch 17/20
6/6 - 0s - loss: 1.0993 - val_loss: 1.1025 - 207ms/epoch - 35ms/step
Epoch 18/20
6/6 - 0s - loss: 1.0988 - val_loss: 1.1021 - 203ms/epoch - 34ms/step
Epoch 19/20
6/6 - 0s - loss: 1.0983 - val_loss: 1.1018 - 202ms/epoch - 34ms/step
Epoch 20/20
6/6 - 0s - loss: 1.0979 - val_loss: 1.1015 - 197ms/epoch - 33ms/step
Metrics :  1.1136882305145264
```

Fig 4.20

```
Epoch 1/20
6/6 - 0s - loss: 1.0976 - val_loss: 1.0898 - 246ms/epoch - 41ms/step
Epoch 2/20
6/6 - 0s - loss: 1.0976 - val_loss: 1.0898 - 213ms/epoch - 36ms/step
Epoch 3/20
6/6 - 0s - loss: 1.0976 - val_loss: 1.0898 - 203ms/epoch - 34ms/step
Epoch 4/20
6/6 - 0s - loss: 1.0977 - val_loss: 1.0896 - 218ms/epoch - 36ms/step
Epoch 5/20
6/6 - 0s - loss: 1.0976 - val_loss: 1.0896 - 200ms/epoch - 33ms/step
Epoch 6/20
6/6 - 0s - loss: 1.0976 - val_loss: 1.0896 - 200ms/epoch - 33ms/step
Epoch 7/20
6/6 - 0s - loss: 1.0976 - val_loss: 1.0896 - 210ms/epoch - 35ms/step
Epoch 8/20
6/6 - 0s - loss: 1.0976 - val_loss: 1.0897 - 199ms/epoch - 33ms/step
Epoch 9/20
6/6 - 0s - loss: 1.0976 - val_loss: 1.0896 - 203ms/epoch - 34ms/step
Epoch 10/20
6/6 - 0s - loss: 1.0976 - val_loss: 1.0896 - 213ms/epoch - 36ms/step
Epoch 11/20
6/6 - 0s - loss: 1.0975 - val_loss: 1.0896 - 199ms/epoch - 33ms/step
Epoch 12/20
6/6 - 0s - loss: 1.0976 - val_loss: 1.0896 - 205ms/epoch - 34ms/step
Epoch 13/20
6/6 - 0s - loss: 1.0975 - val_loss: 1.0896 - 194ms/epoch - 32ms/step
Epoch 14/20
6/6 - 0s - loss: 1.0975 - val_loss: 1.0896 - 205ms/epoch - 34ms/step
Epoch 15/20
6/6 - 0s - loss: 1.0975 - val_loss: 1.0896 - 201ms/epoch - 34ms/step
Epoch 16/20
6/6 - 0s - loss: 1.0975 - val_loss: 1.0896 - 192ms/epoch - 32ms/step
Epoch 17/20
6/6 - 0s - loss: 1.0975 - val_loss: 1.0895 - 192ms/epoch - 32ms/step
Epoch 18/20
6/6 - 0s - loss: 1.0975 - val_loss: 1.0895 - 195ms/epoch - 33ms/step
Epoch 19/20
6/6 - 0s - loss: 1.0975 - val_loss: 1.0896 - 198ms/epoch - 33ms/step
Epoch 20/20
6/6 - 0s - loss: 1.0975 - val_loss: 1.0896 - 198ms/epoch - 33ms/step
Metrics :  1.1168482303619385
```

Figure 4.21

```
K Fold Step   4
Epoch 1/20
6/6 - 0s - loss: 1.0974 - val_loss: 1.0896 - 236ms/epoch - 39ms/step
Epoch 2/20
6/6 - 0s - loss: 1.0974 - val_loss: 1.0895 - 196ms/epoch - 33ms/step
Epoch 3/20
6/6 - 0s - loss: 1.0974 - val_loss: 1.0895 - 197ms/epoch - 33ms/step
Epoch 4/20
6/6 - 0s - loss: 1.0975 - val_loss: 1.0895 - 200ms/epoch - 33ms/step
Epoch 5/20
6/6 - 0s - loss: 1.0974 - val_loss: 1.0895 - 195ms/epoch - 33ms/step
Epoch 6/20
6/6 - 0s - loss: 1.0975 - val_loss: 1.0894 - 207ms/epoch - 35ms/step
Epoch 7/20
6/6 - 0s - loss: 1.0974 - val_loss: 1.0895 - 205ms/epoch - 34ms/step
Epoch 8/20
6/6 - 0s - loss: 1.0974 - val_loss: 1.0895 - 201ms/epoch - 34ms/step
Epoch 9/20
6/6 - 0s - loss: 1.0975 - val_loss: 1.0895 - 194ms/epoch - 32ms/step
Epoch 10/20
6/6 - 0s - loss: 1.0974 - val_loss: 1.0895 - 204ms/epoch - 34ms/step
Epoch 11/20
6/6 - 0s - loss: 1.0975 - val_loss: 1.0894 - 198ms/epoch - 33ms/step
Epoch 12/20
6/6 - 0s - loss: 1.0974 - val_loss: 1.0894 - 208ms/epoch - 35ms/step
Epoch 13/20
6/6 - 0s - loss: 1.0974 - val_loss: 1.0895 - 195ms/epoch - 33ms/step
Epoch 14/20
6/6 - 0s - loss: 1.0974 - val_loss: 1.0895 - 206ms/epoch - 34ms/step
Epoch 15/20
6/6 - 0s - loss: 1.0974 - val_loss: 1.0895 - 207ms/epoch - 35ms/step
Epoch 16/20
6/6 - 0s - loss: 1.0974 - val_loss: 1.0895 - 194ms/epoch - 32ms/step
Epoch 17/20
6/6 - 0s - loss: 1.0974 - val_loss: 1.0895 - 202ms/epoch - 34ms/step
Epoch 18/20
6/6 - 0s - loss: 1.0974 - val_loss: 1.0895 - 200ms/epoch - 33ms/step
Epoch 19/20
6/6 - 0s - loss: 1.0974 - val_loss: 1.0895 - 200ms/epoch - 33ms/step
Epoch 20/20
6/6 - 0s - loss: 1.0974 - val_loss: 1.0895 - 204ms/epoch - 34ms/step
Metrics :  1.1166386604309082
```

Fig 4.22

The above images show the training accuracies for each model relating to each of the speakers. As we can see, the losses show no deviation throughout the 20 epochs.

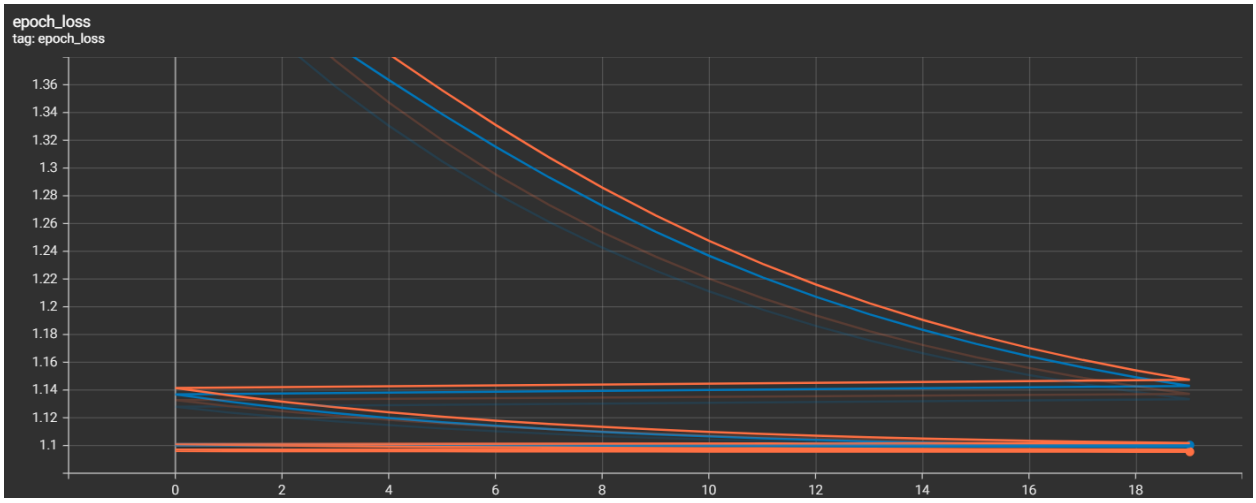


Fig 4.23

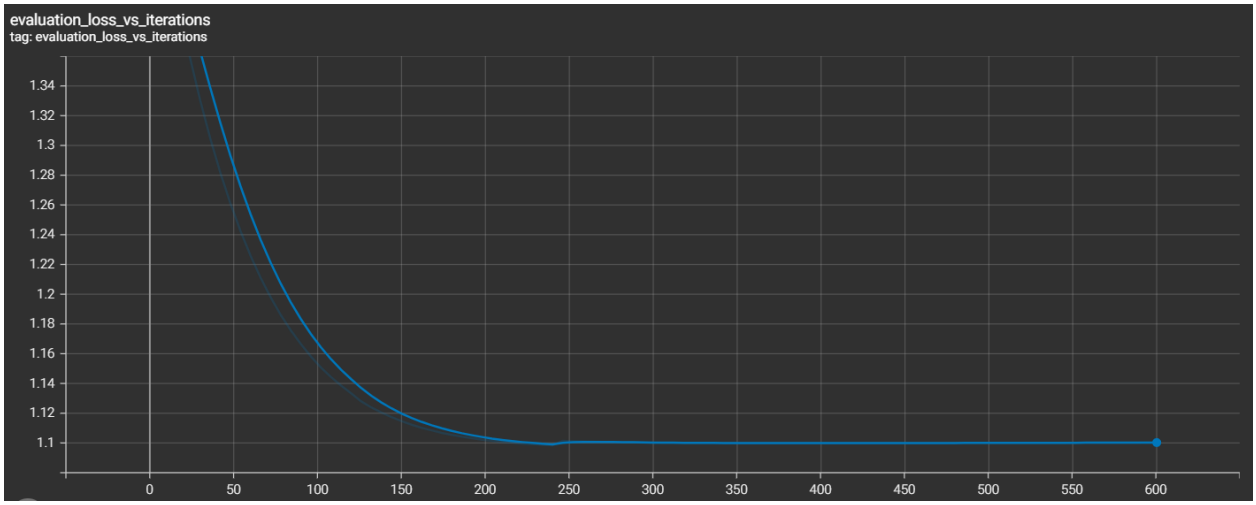


Fig 4.24

	precision	recall	f1-score	support
speaker1	0.33	1.00	0.50	3
speaker2	0.00	0.00	0.00	3
speaker3	0.00	0.00	0.00	3
accuracy			0.33	9
macro avg	0.11	0.33	0.17	9
weighted avg	0.11	0.33	0.17	9

Fig 4.25

The image above shows the accuracy metrics of the model. As we can see, the results are not very good, with only speaker1 giving any True Positives/True Negatives.

On the other hand, on training this same dataset on a much larger dataset, like the Free Spoken Digit Dataset, the accuracy metrics are much better. This could be attributed to the increase in the amount of data and timesteps.

CHAPTER 5: CONCLUSION AND FUTURE SCOPE

5.1 Conclusion

The proposed comparative analysis signifies the merits and de-merits of using a probability-based learning model (GMM) over a deep learning based SincNet layer model for the purpose of speaker identification. The models were trained over a short audio dataset, containing audio samples of the five members of this project group. The individual performances of the two trained models suggest that for a small dataset the GMM model overfits and even though it generates better accuracy, it fails in live-audio testing. On the other hand, the SincNet model gives better results for a live recorded audio sample, thus proving to be a more robust model.

5.2 Future Scope

Nowadays with the overwhelming boom in fields of IOT and voice-enabled technologies, we shall see an uprising concern with developments in speaker recognition, as it provides the necessary user authentication for one to access and use the voice-mechanized products with utmost safety and uncompromised integrity of information. Hence the next possible step to take will be to embed the software into a hardware system, that suffices as a handy voice-biometric tool. It can also be incorporated into various products as an authenticator interface that grants appropriate users access into the system.

REFERENCES

[1] George R Doddington, Member, IEEE, "Speaker Recognition- Identifying People by their Voices", proceedings of the IEEE, Vol. 73, no. 11, pp. 1651-1664, November 1985

[2] Varun Sharma and Dr. P. K. Bansal. "A review on speaker recognition approaches and challenges." International Journal of Engineering Research and Technology (IJERT) 2.5 (2013): 1581-1588.

[3] Kabir, Muhammad Mohsin, M. F. Mridha, Jungpil Shin, Israt Jahan, and Abu Quwsar Ohi. "A Survey of Speaker Recognition: Fundamental Theories, Recognition Methods and Opportunities." IEEE Access (2021)

[4] Ibrahim, Noor Salwani, and Dzati Athiar Ramli. "I-vector extraction for speaker recognition based on dimensionality reduction." Procedia Computer Science 126 (2018): 1534-1540.

[5] J. H. Hansen, T. Hasan, "Speaker recognition by machines and humans: A tutorial review", IEEE Signal processing magazine 32 (6) (2015)

[6] Sztahó, Dávid, György Szaszák, and András Beke. "Deep learning methods in speaker recognition: a review." arXiv preprint arXiv:1911.06615 (2019).

[7] Bai, Zhongxin, and Xiao-Lei Zhang. "Speaker recognition based on deep learning: An overview." Neural Networks (2021).

[8]Ravanelli, Mirco, and Yoshua Bengio. "Speaker recognition from raw waveform with SincNet." 2018 IEEE Spoken Language Technology Workshop (SLT). IEEE, 2018

[9] Lei Lei, She Kun, "Speaker Recognition Using Wavelet Packet Entropy, I-Vector, and Cosine Distance Scoring", Journal of Electrical and Computer Engineering, vol. 2017.

[10] K. Sajeer, Paul Rodrigues, "Speaker Recognition System Based on Wavelet Features and Gaussian Mixture Models" , International Journal of Engineering and Advanced Technology (IJEAT) ISSN: 2249 – 8958, Volume-9 Issue-1, October 2019

[11] Z.Tufekci, J.N. Gowdy, "Feature extraction using discrete wavelet transform for speech recognition", Conference: Southeastcon 2000. Proceedings of the IEEE

[12] Loknath Debnath, Jean-Peirre Antonio , "Wavelet Transforms and Their Application", Journal - Physics Today - 2002

[13] S. Malik, Fayyaz ul Amir Afsar Minhas , "Wavelet transform based automatic speaker recognition" , Conference: Multitopic Conference, 2009. INMIC 2009. IEEE 13th International.

[14] Prabakaran Durairaj, S. Sri Uppili, "Speech Processing: MFCC Based Feature Extraction Techniques- An Investigation", Journal of Physics Conference Series 1717(1):012009