**GESTURE CONTROLLED MOUSE POINTER**

#### A PROJECT REPORT

###### Submitted by

### Ansh Vinod Motwani [Reg No: RA1611008010661]

### Ayush Roy [Reg No: RA1611008010641]

### Saumya Awasthi [Reg No: RA1611008010414]

###### Under the Guidance of

### Ms. Deepanjali

(Professor, Department of Information Technology)

###### In partial fulfillment of the Requirements for the Degree of

### BACHELOR OF TECHNOLOGY



### DEPARTMENT OF INFORMATION TECHNOLOGY FACULTY OF ENGINEERING AND TECHNOLOGY SRM INSTITUTE OF SCIENCE AND TECHNOLOGY KATTANKULATHUR – 603203

#### APRIL 2020

## SRM INSTITUTE OF SCIENCE AND TECHNOLOGY KATTANKULATHUR-603203

### BONAFIDE CERTIFICATE

Certified that this project report titled “**Gesture Controlled Mouse Pointer”** is the bonafide work of **“Ansh Vinod Motwani [Reg No: RA1611008010661], Ayush Roy [Reg No: RA1611008010641] and Saumya Awasthi [Reg No: RA1611008010414]** who carried out the project work under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion for this or any other candidate.

Ms. Deepanjali Dr. G. VADIVU

GUIDE HEAD OF THE DEPARTMENT

Assistant Professor Dept. of Information Technology Dept. of Information Technology

Signature of Internal Examiner Signature of External Examiner

#### ABSTRACT

Apna waala copy paste kar dena frand

#### ACKNOWLEDGMENT

The success and the final outcome of this project required guidance and assistance from different sources and we feel extremely fortunate to have got this all along with the completion of our project. We express our sincere thanks to the Head of the Department, Department of Information Technology, **Ms. Deepanjali**, for all the help and infrastructure provided us to complete this project successfully. We owe our profound gratitude to our project guide **Ms. Deepanjali**, who took a keen interest in our project work and guided us all along, till the completion of our project work by providing all the necessary information for developing a good system. We are thankful and fortunate enough to get constant encouragement, support and guidance from all the teaching staff of the Department of Information Technology who helped us in successfully completing our major project. Also, we would like to extend our sincere regards to all the non-teaching staff of the Department of Information Technology for their timely support.

Ansh Vinod Motwani [Reg No: RA1611008010661]

Ayush Roy [Reg No: RA1611008010641]

Saumya Awasthi [Reg No: RA1611008010414]

### TABLE OF CONTENTS

TABLE OF CONTENTS

|  |  |
| --- | --- |
| CHAPTER NO. TITLE | PAGE NO. |
| ABSTRACT | 2 |
| LIST OF TABLES | 6 |
| LIST OF FIGURES | 7 |
| 1. INTRODUCTION | 8 |
| 1.1 OVERVIEW | 8 |
| 1.2 EVOLUTION | 9 |
| 1.3 BACKGROUND AND MOTIVATION | 9 |
| 1.4 CATEGORIES AND TYPE | 10 |
| 1.5 PROBLEM STATEMENT | 11 |
| 2. LITERATURE REVIEW | 12 |
| 2.1 INTELLIGENT PROCESSING OF STUTTERED SPEECH | 12 |
| 2.2 SPEECH RECOGNITION BY MACHINE: A REVIEW | 13 |
| CLASSIFICATION AND RECOGNITION OF STUTTERED  2.3 SPEECH | 14 |
| 2.4 ARS FOR STUTTERING DISABLED PERSON | 16 |
| RECOGNITION AND CLASSIFICATION OF PAUSES IN |  |
| 2.5 STUTTERED SPEECH USING ACOUSTIC FEATURES | 17 |
| 3. PROPOSED WORK | 19 |
| 3.1 DATASET GATHERING AND PREPROCESSING | 19 |
| 3.2 MODEL TRAINING USING NEURAL NETWORK | 20 |
| 3.3 FEATURE EXTRACTION | 22 |
| 3.4 AUDIO CORRECTION | 24 |
| 3.5 SPEECH TO TEXT CONVERSION | 25 |
| 4. IMPLEMENTATION | 27 |
| 5. CONCLUSION | 30 |
| 6. FUTURE ENHANCEMENTS | 33 |
| 7. REFERENCES | 34 |

[APPENDIX 37](#_TOC_250002)

[PAPER PUBLICATION STATUS 61](#_TOC_250001)

[PLAGIARISM REPORT 62](#_TOC_250000)

### LIST OF FIGURES

* + 1. Dataset gathering and preprocessing code. 20
    2. Dataset gathering and preprocessing code. 20
    3. Feature extraction code… 21
    4. Feature extraction code… 22
    5. Model Training Code… 23
    6. Model Training Code… 23
    7. Model Training Code… 23
    8. Audio Correction code… 24
    9. Audio Correction code… 25
    10. Audio Correction code… 25

3.5.1 Speech to text API code 26

* 1. MFCC v/s time graph… 27
  2. Mel Power Spectrograph… 27
  3. Flow chart… 28
  4. Output… 28
  5. Use Case. 29
  6. Front-end Index page… 31
  7. Speech to text… 31
  8. processing test input… 32

### LIST OF ABBREVIATIONS

**ASR** Automatic Speech Recognition

**HMM** Hidden Markov Model

**MFCC** Mel Frequency Cepstral Coefficient

**DNN** Deep Neural Network

**ANN** Artificial Neural Network

**SVM** Support Vector Machines

**UCLASS** University College London's. Archive of Stuttered Speech

**CHAPTER 1**

### INTRODUCTION

* 1. OVERVIEW

Speech is the most rudimentary and natural form of communication for humans. Fluency in speaking measures the effectiveness of the delivery of information and communication with another individual. There are numerous factors for determining whether the speech is fluent or not, most likely of them is continuity. Speech is accompanied by many disruptions like reiteration, hesitance, and pauses. Stuttering is one of these speech impairments which affects 1% of the world’s population which consists generally of males. In this speech impairment, the flow of speech is disrupted in various ways i.e. **Reiterations**, **Blocks**, **Prolongations** of sounds, syllables, phrases, and silent pauses. The symptoms of Stuttering can contrast based on contrasting situations eg. Excitement, Stress, Anger. There are two types of Stuttering viz. Development Stuttering and Neurogenic Stuttering.

Speech Recognition of aka Automatic Speech Recognition (ASR) is the method of conversion of the Speech signal to its attributes. Speech Recognition implemented by various tech companies today has made it possible to convert voice commands to the machine language and understand those commands. The earliest attempt to Speech Recognition technology was made in the 1950s and ever since it has found its application in our day to day life.

The Speech-Recognition system involves stages like speech pre-processing, feature extraction, and feature classification. Feature extraction is known as the process of converting audio signal into its parametric attributes and processing those attributes to gain as much information as possible and discarding unwanted noises. The proposed method for feature extraction is the Mel Frequency Cepstral Coefficient (MFCC) to extract features of a signal.

* 1. EVOLUTION

In the 1920's-1960's machine recognition speech first came into the picture, work on speech technology began in bell labs, and the first ASR system was built in bell labs by investigating spectral resonances.

In the 1960s-1980s to properly analyze vowels scientists started using filter banks. Digit recognizer hardware at NEC Laboratories was developed. Dynamic programming methods were used to develop dynamic time warping.

In the 1980's-1990's Hidden Markov Model (HMM) and the Defence Advanced Research Projects Agency (DARPA) were developed which helped to develop TensorFlow audio recognition. In the 1980s people started using neural networking for ASR models.

After the 1990's Mel-frequency spectral was developed.

Later many more technologies were developed to improvise feature extraction and classification of speech.

* 1. BACKGROUND & MOTIVATION

Stammering is usually recognized by a tense struggle to urge words out. This makes it different from the traditional non-fluency we all experience which has hesitations and repetitions. Commonly it involves repeating or prolonging sounds or words, or getting stuck with no sound (silent blocking). Stuttering causes peculiar behavior and social competence in people, with a higher tendency to manifest alterations in this area, in comparison to those who do not stutter. Fear, nervousness/tension, guilt, anxiety, perfectionism, and worry were the most frequent alterations in relation to the

behavior, whereas damages in the social field and in the habitual communication situations characterized the social competence of persons who stutter.

This speech impediment can have significant effects on the performance of speech recognizers, thereby harming the ability of individuals with stuttering to be able to use speech-related tools. Many of the voice interfaces that exist ubiquitously within today’s consumer technology, from smart TVs to car systems, often neglect populations with speech ailments. Inaccuracies of this degree render voice control tools difficult, if not impossible, for affected individuals to use reliably.

Thus, our project seeks to improve the performance of automatic speech recognizers on speech containing stuttering, specifically by trying to develop classifiers that can better detect stuttering in speech signals, as well as to study techniques on applying these classifiers to ASR models so that we can more effectively parse out stuttered speech before processing these speech signals.

* 1. CATEGORIES & TYPES

Stuttering is a speech problem that usually develops in children from the age of 6 – 20 years. The normal flow of speech is disrupted. A child who stutters repeats or prolongs sounds, syllables, or words.

Doctors and researchers all across the world have classified stuttering into 3 main groups. We have Developmental Stuttering, which is the most common type and happens when a child’s speech and language development lags. Then there is Neurogenic Stuttering, which may happen after a stroke or brain injury, lastly, we have Psychogenic Stuttering, which may happen after emotional trauma.

* 1. PROBLEM STATEMENT

More than 70 million people in the world are currently suffering from stammering, which is approximately one person in every hundred. In the past few years, an increase has been seen in the popularity of personal voice assistants, which are able to speed up many day-to-day tasks. However, these systems are limited by the ability of the speaker to use them. If the input to a voice recognition system contains stammering, it fails miserably, with an accuracy as low as 18% and as high as 73% as compared to a baseline of 92% for a normal speaker. This project aims to optimize the speech-to-text conversion process, for speech containing stammers.

**CHAPTER 2**

### LITERATURE STUDY

#### Intelligent Processing of Stuttered Speech

##### Author: Andrzej Czyzewski, Andrzej Kaczmarek & Bozena Kostek (2003)

The paper report on automatic recognition of stuttered audio in normal and frequency altered feedback speech. It showcases several methods of analyzing stuttered speech and describes ways to establish those parameters that represent a stuttering event.

I inferred from the paper is that the process of counting stuttering events could be carried out more objectively through the automatic detection of stop-gaps, syllable repetitions, and vowel prolongations. The alternative would be based on the subjective evaluations of speech fluency and may be dependent on a subjective evaluation method. Meanwhile, the automatic detection of intervocalic intervals, stop-gaps, voice onset time and vowel duration may depend on the speaker and the rules derived for a single speaker might be unreliable when trying to consider them as universal ones.

All these restrictions make the ANN Classifier a bit redundant as it has a lot of dynamic variables and this mismatch in data drastically decreases the accuracy of their ANN model. This configuration resulted in a 71% accuracy for their model.

Hence I can conclude that using a DNN which yields an average accuracy of around 86% is the best choice because the parameters are being automated by the machine

which allows the model to get the best and the optimized way to get the particular output for our stuttered speech.

#### Speech Recognition by Machine: A Review

##### Author: M.A.Anusuya and S.K.Katti (2009)

This paper presents an Automatic Speech Recognition (ASR) model and discusses the benefits of using an Acoustic phonetic approach, which postulates that there exist finite, distinctive phonetic units (phonemes) in spoken language and that these phonetic units are broadly characterized by a set of acoustics properties that are manifested in the speech signal over time. Even though the acoustic properties of phonetic units vary at a high rate, both with the speakers and with the neighboring sounds (known as coarticulation effect), it is assumed in acoustic-phonetic approach that the rules governing the variability are straightforward and can be readily learned by a machine.

The initial step in the acoustic-phonetic approach is a spectral analysis of the speech combined with a feature detection that converts the spectral measurements to a set of features that describe the broad range of acoustic properties of the different phonetic units. The next step is segmentation and labeling phase, in this step the speech signal is segmented into stable acoustic regions, followed by attaching one

or more phonetic labels to each segmented region which results in a phoneme called lattice characterization of the speech. The last step in this approach tries to determine a valid word (or string of words) from the phonetic label series produced by the segmentation to labeling. In the validation process, linguistic constraints on the task (i.e., the vocabulary, the syntax, and other semantic rules) are invoked so to access the lexicon for word decoding based on the phoneme lattice.

The problems that are existing in ASR are that it is very slow and is not able to detect nearly 25% - 30% of the words correctly.

#### Classification and Recognition of Stuttered Speech

##### Author: Manu Chopra, Kevin Khieu and Thomas Liu (2016)

Their approach tackles the problem on two levels: the classifier-level, and the ASR-level. For classifiers, they worked on creating a model that could best identify whether a 1-second clip of audio contained stuttering or not. Classification techniques commonly used in existing literature are Artificial Neural Networks (ANNs), Hidden Markov Model (HMM) and Support Vector Machines (SVM), MFCC and spectral measures. In this paper, neural networks have been used for the

classification of stuttered voice and non-stuttered voice. The second part of their approach is to apply their classifier in various ways to an Automatic

Speech Recognizer to see how best they could apply their algorithms such that the speech we produce is clearer than it was before.

* + - They have also experimented with different ways to extract features. This paper shows studying the effectiveness of neural networks on serving as classifiers for stuttered and non-stuttered speech. They used the Librosa library to extract various features from the WAV file (waveform audio file format). They used MATLAB implementations for feature extraction.

First, they made a baseline classifier which they made on the bases of previously used neural networks, they kept the classifier as simple as possible by putting only two-layered neural network, and analyze the effects of different audio features on the success of the classifier (as well as to verify the effectiveness of

MFCC features on stuttering/non-stuttering classification). The goal of our baseline classifier was to test the simplest possible solution we could create: a two-layer neural network in TensorFlow that captured only MFCC features of the audio files we passed. Their final epoch value was 5000 and learning rate was 0.01and they had 53 audio files (half stuttered and half normal) this

Configuration yielded the best accuracy of 66.0%.

They also discovered that using MFCC for feature extraction gave fairly high accuracy.

#### Automatic Speech Recognition System for Stuttering Disabled Persons

##### Author: Arya A Suryaa and Surekha Mariam Varghese, Published in – 2017

In the present world, Automatic Speech Recognition (ASR) finds its relevance in many applications. But the modern-day Automatic Speech Recognition systems have a lack of understanding for stammered speech. This paper proposes three methods i.e., using the trained model, by removing prolongations/repetitions and by converting to text for recognizing stuttered speech.

In this paper, they have used a supported vector machine (SVM) for model training and MCFF for feature extraction. Three methods are proposed for the recognition of the stuttered speech.

1. Supervised model for stuttered speech recognition

Using MFCC for feature extraction of the audio signals and SVM has two stages training and testing. SVM is a classifier that performs classification. During testing, SVM classifies the stuttered speech input to a correct word. They acquired an accuracy of 76% in classifying the words correctly. The accuracy of this method can be improved by using more training data.

1. Stuttered speech recognition by stuttering pruning

In this, the speech is converted into amplitude/time audio signals and then the max amplitude is given to the neural network to compute the threshold value on the bases of which classification is

done. The speech correction method implemented using neural network acquired less accuracy i.e. of 62% which can be improved further by using more training data as well as incorporating some other features such as energy, frequency, etc of the audio input for training.

1. Automated text-to-speech based stuttered speech recognition

ANN is trained with an intelligent guess to predict the vowels and consonants terms in the speech. This ANN analyses the inputted speech with its training experience and produces equivalent texts. The outputted text is then inputted to a dictionary and picks the most matched word. In this way, all stuttered speeches are eliminated. The above-explained method received an accuracy of 80%.

So according to the paper, the third method was most accurate for the ASR system for stuttered voice, even though the other two can be more efficient if trained more.

#### Recognition and Classification of Pauses in Stuttered Speech using Acoustic Features

##### Author: Fathima Afroz and Shashidhar G Koolagudi, Published in – 2019

The attributes like duration, frequency, position and distribution of pauses during speech tasks are measured and quantified. UCLASS stuttered speech corpus is considered for the analysis. The automatic blind segmentation approach is adopted to segment the speech signal into voice and unvoiced regions using a dynamic threshold set based on energy and zero-crossing rate (ZCR). 4th formant frequencies are analyzed to identify intra-morphic (unfilled) pauses present within voiced regions. The duration of intra-morphic pauses is analyzed for stuttered speech and normal speech. It is observed that the duration of normal endomorphic

pauses ranges from 150 ms-250 ms and inter-morphic pauses are <=250 ms and short pauses have duration ranges from 50 ms-150 ms. Whereas in stuttering short intra-morphic pauses range from 10 ms to 50 ms, long pauses range from 250 ms to

1 or 2 seconds. They have used features like short term energy, spectral flux, zero-crossing rate, pitch and formant frequencies to identify the pauses and stuttering in the speech. They have used blind segmentation so there is no need to redesign model for every speech style, accent or language algorithms need not have to be

Redesigned or modified. After the identification of pauses, all voiced and silent regions were obtained. By setting a threshold value greater than 150 ms as long and less than 150 ms as short, every paused region is classified into long and short categories. They have worked only to identify the pauses in the speech. The threshold they have taken can differ according to the developer but it can cause many errors. According to the paper in previous papers classification for pauses was done manually On average for 108 files the manual segmentation resulted in 15 to 20 % more number of segments than automatic blind segmentation, the reason behind this difference is; in manual method what they see in the waveform as silence or break tried to count (smaller portion of voice) as one segment. This leads to improper segmentation. But in the proposed method they obtained 98% accurate voice segments, they performed manual & perceptual analysis to verify the detected accuracy. This system has wide applications in the medical field especially for the diagnosis of speech disorder.

**CHAPTER 3**

**PROPOSED WORK**

* 1. Dataset Gathering and Pre-processing

The dataset we use is gathered from a repository called UCLASS, which stands for University College London Archive of Stuttered Speech. This specific dataset has been chosen because it has recordings of speeches of various speakers aging from 7 years to 20 years. And another reason that this dataset was selected is that it contains all the files in .wav format. The database has an audio file in 2 releases, release 1 and release 2. Release 1 has 16 files that have time-aligned transcription and release 2 has only 4 files, so we chose release 1 and trained our model on the dataset provided in release 1. The audio files are in .wav format having a sampling rate of 22050Hz and the corresponding time- aligned transcriptions are in CHILDES CHAT format. The time-aligned transcriptions are used to split the data files into stuttered segments and normal segments. The transcriptions had the start time and end time (in milliseconds) for each segment. So there were in all 12,633 segments that were registered after splitting the audio file segments are about 2.5 times longer than the average segment and the longest stutter was around 17 seconds. Training the data on such skewed data will not be useful because seeing a stuttered segment as long as 17 seconds is very unlikely. So, instead of using the segments of variable duration, we segmented the segments of the file further down to less than or equal to 300 ms which is close to the average length of the segments. This segmentation created 17,545 segments that were used for training the models.

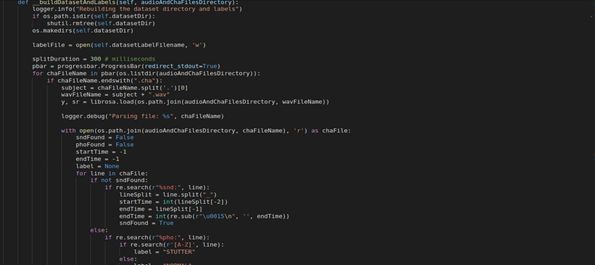


Figure 3.1.1 Dataset gathering and preprocessing code

Figure 3.1.2 Dataset gathering and preprocessing code

* 1. FEATURE EXTRACTION:

We use MFCC to identify the components of audio signals that are useful for the identification of the signal and discard all the other stuff. MFCC is used to represent power spectral density (power versus frequency) of the sound signals on the bases of which all the other 13 features are extracted. As it is known a sound wave is constantly changing with time, on a smaller scale audio signals don’t change very rapidly, so to examine the signals the audio signals are divided into small frames in order to closely examine them and extract accurate features. If the sample fraction is

too short, we don’t have enough samples to compute the characteristics and if the fraction is too large the change will occur very drastically hence, we won’t get accurate values. We use MFCC to identify the components of audio signals that are useful for the identification of the signal and discard all the other stuff. MFCC is used to represent power spectral density (power versus frequency) of the sound signals on the bases of which all the other 13 features are extracted. As it is known a sound wave is constantly changing with time, on a smaller scale audio signals don’t change very rapidly, so to examine the signals the audio signals are divided into small frames in order to closely examine them and extract accurate features. If the sample fraction is too short there are not enough samples to compute the characteristics and if the fraction is too large the change will occur very drastically hence there won’t be accurate values. Now, the Mel-scale spectrogram is computed, then we extract the MFCC sequence of 13, delta matrix (the change in coefficients), and delta-delta matrix (the change in delta values) are extracted. And now on the basis of the features extracted the audio files are classified.

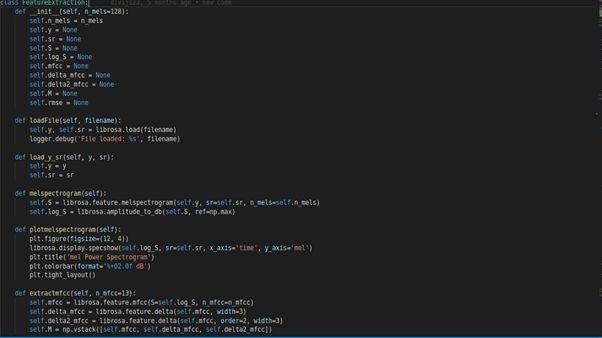


Figure 3.2.1 Feature Extraction Code

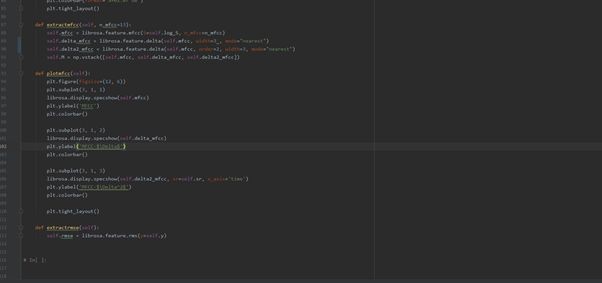


Figure 3.2.2 Feature Extraction Code

* 1. MODEL TRAINING USING NEURAL NETWORK:

In Machine Learning Deep Neural networks (DNN) is a supervised learning algorithm where the machine recurrently performs the same task on every element of the sequence of where the output of each task is dependent on every previous calculation. This method learns in a sequence called feature hierarchy. Where the features on the top of the hierarchy are computed by the help of features at the bottom of the hierarchy. DNN’s are formulated of multiple layers. The computations in each layer are hidden hence the layers are termed as “hidden layers “. There are 3 hidden layers in the proposed neural network and 10 neurons in each layer. The learning rate is set to 0.001, it is set a lower value so that the machine does not skip many values and the value of weights is not updated very frequently. Along with low learning rates, one requires a greater number of training epochs which in this scenario is 1200. Epoch is a parameter that determines the number of times the algorithm will work on the entire training dataset. Epoch is a parameter that determines the number of times the algorithm will work on the entire training dataset.



Figure 3.3.1 Model Training Code

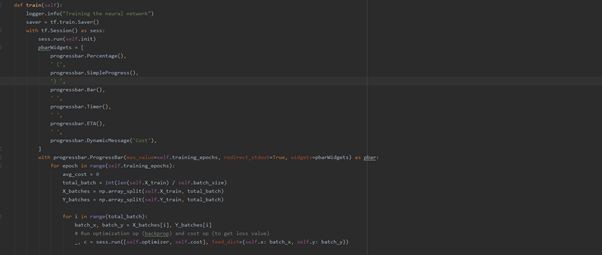


Figure 3.3.2 Model Training Code



Figure 3.3.3 Model Training Code

* 1. AUDIO CORRECTION:

Now we prepare our input audio by overlapping all the segments by 200 ms. With this type of overlapping, we could detect the stuttered and non-stuttered parts with the granularity of 100ms. After the segmentation, these segments were sent to the classifier for classification. The classifier labeled our segments as either STUTTER or NORMAL. So now all we had to do was to remove the segments which were labeled as STUTTER and combine the segments labeled as NORMAL. One way of assembling the segments was to append contiguous chunks together but this will result in sharp interjections at the point of concatenation and will result in a very artificial sounding voice. So instead of naively appending the adjacent chunks, we interpolated the audio samples between the end of the previous chunk and the beginning of the current chunk.

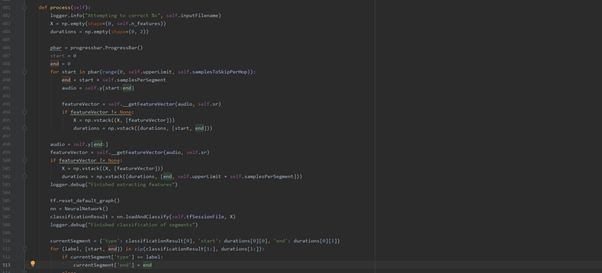


Figure 3.4.1 Audio Correction Code

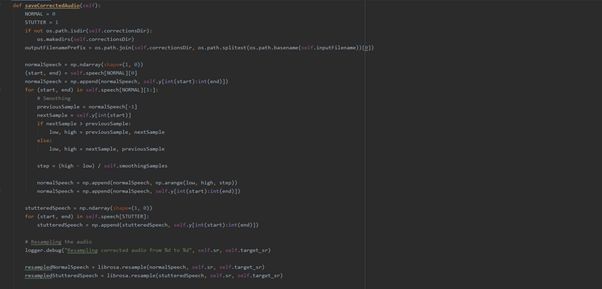


Figure 3.4.2 Audio Correction



Figure 3.4.3 Audio Correction

* 1. SPEECH TO TEXT:

The UCLASS dataset that we are using is in British English. Instead of training our own Speech-to-text framework on British English, we used the Wit's Speech-to-text API that takes an audio file in WAV format and uses its own ML model to determine the correct translation. Right now Wit's Speech-to-text API provides us with an accuracy of 80%.

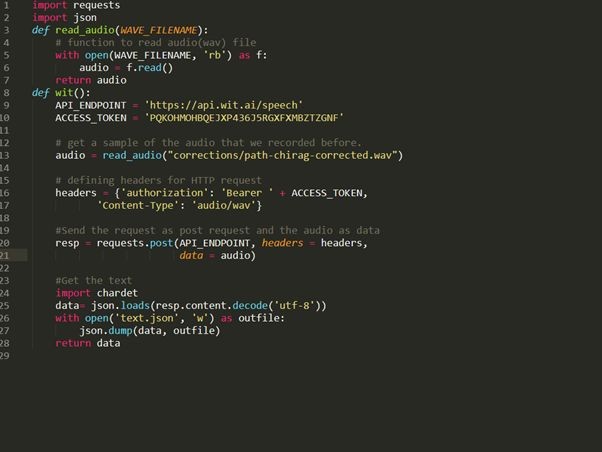


Figure 3.5.1 Speech to text API Code

**CHAPTER 4**

**IMPLEMENTATION**

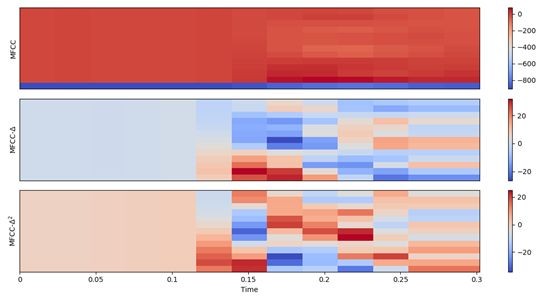


Figure 4.1 MFCC v/s time

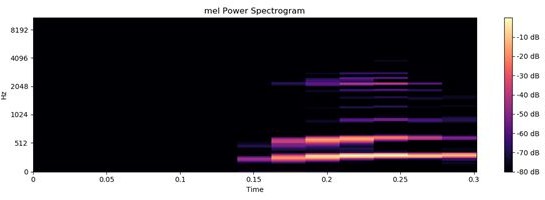


Figure 4.2 Mel power Spectrogram

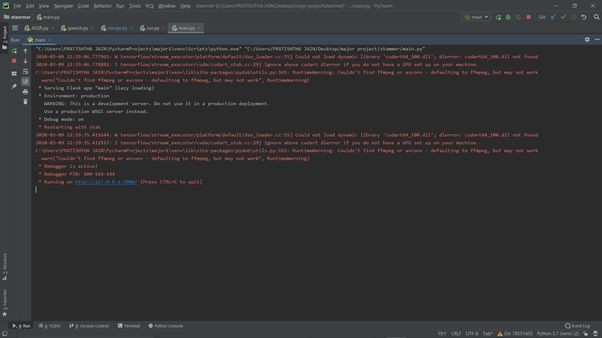


Figure 4.3 Output

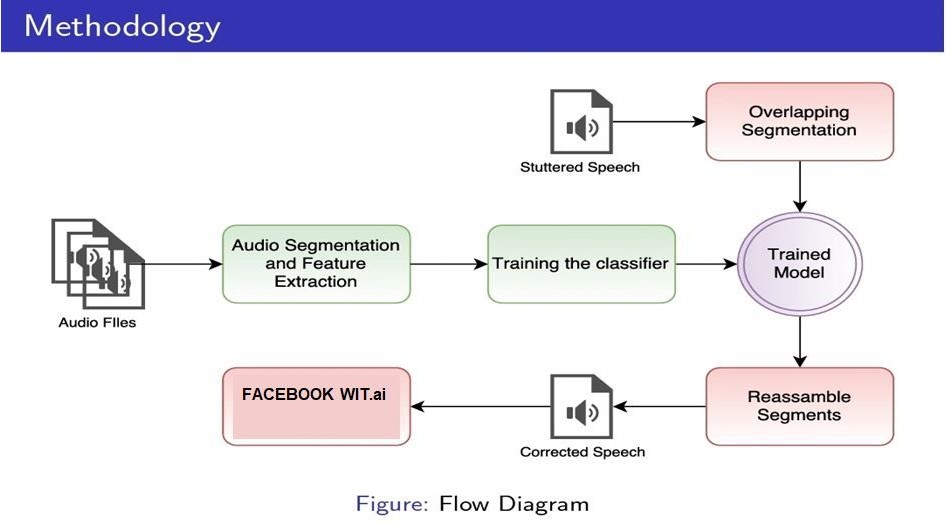


Figure 4.4 Flow Diagram

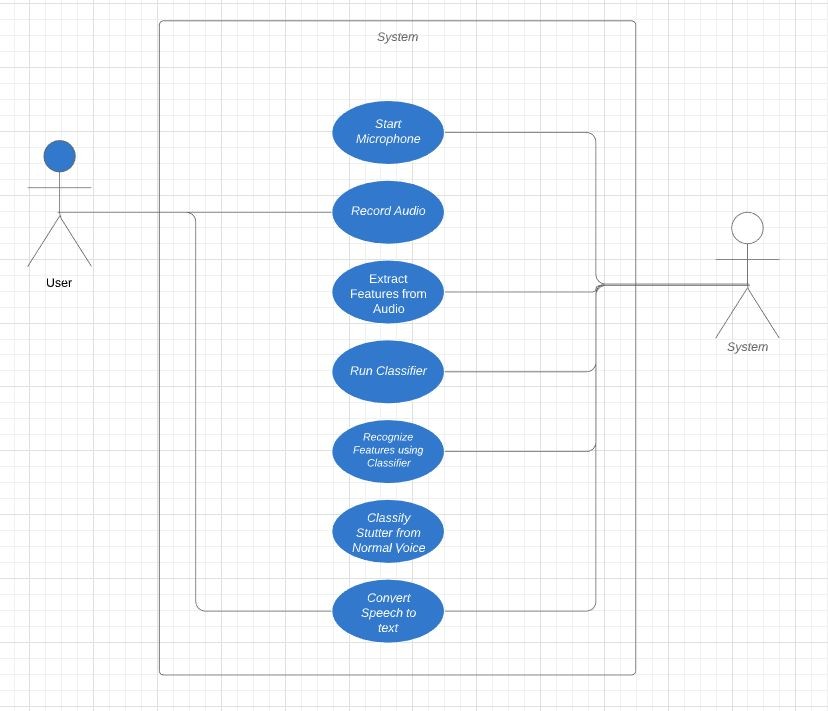


Figure 4.5 Use Case Diagram

**CHAPTER 5**

**CONCLUSION**

Data Samples for this system were obtained from University College London Archive of Stuttered Speech (UCLASS) and it consists of recordings of speakers who stutter and provides us with background details about the conditions in which the recordings were made.

The first method implemented as data processing that follows the breaking of audio input into multiple segments of less than or equal to 300ms, this allowed us to extract MFCC features from each segment.

Before sending the extracted MFCC features to the classifier, the segments obtained had a lot of error margin so to reduce that we overlapped the segments, which in turn will help us in distinguishing between stuttered and non-stuttered parts with a granularity of 100ms.

These overlapped segments were passed onto the classifier and by taking a set difference we were able to obtain all the segments, which were labeled as NORMAL and we interpolated the audio segments between the end of the previous chunk and the beginning of the current chunk. Overall our output is corrected with an accuracy of around 85% and takes about 40secs to process the audio of 2 minutes. Finally, the corrected speech is being passed through wit’s speech to text API and text output is displayed to the end-user.



Figure 5.1 Front-end Index page



Figure 5.2 Speech to text

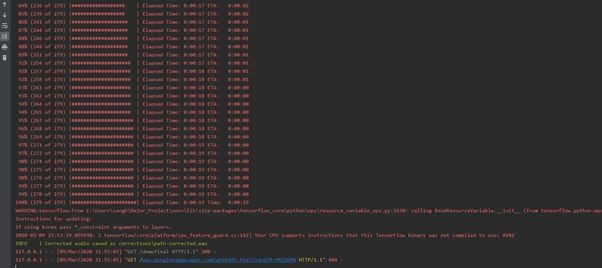


Figure 5.3 processing test input

**CHAPTER 6**

**FUTURE ENHANCEMENTS**

In the near future, users are going to be able to integrate this technology seamlessly into their lives without being visually or physically required to interact with a screen. Progress will likely be made with large-vocabulary recognition (upto 10,000 words or more), speaker-independent continuous speech recognition (upto 1000 words or more), and naturally spoken language understanding.

Robust Speech Recognition devices for stuttered speech will need the AI that has the capability to handle challenges such as accents and background noise more efficiently and smoothly.

Future research needs to achieve an even better accuracy than can be achieved through existing systems are as follows: The first is to develop more precise models of human spoken language production and perception. The second is to develop sophisticated models for computer learning and recognition. Third, we need to develop a knowledge base of language and domains. Systems that support inputs in multiple languages as well as multilingual translation may be developed.

Probably in the future, virtual assistants will dominate our lives as voice will help us communicate with our home appliances like alarm systems, lights, sound systems and even kitchen appliances. We'll also experience an outsized growth of voice-controlled devices to rule our workplaces, being hands-free will play a key role in hospitals, laboratories and producing units. Additionally, we'll have intelligent voice-driven cars, entertainment and location-based searches and also the users are often completely hands-free. All of those systems must cater to the wants of their users having stuttered speech.

**CHAPTER 7**

**REFERENCES**

1. Fathima Afroz and Shashidhar G Koolagudi, "Recognition and Classification of Pauses in Stuttered Speech using Acoustic Features", in 6th International Conference on Signal k Processing and Integrated Networks (SPIN), 2019.
2. Vikhyath Narayan K N and S P Meharunnisa, "Detection and Analysis of Stuttered Speech", in International Journal of Advanced Research in Electronics and Communication Engineering (IJARECE), Volume 5, Issue 4, April 2016
3. Lim Sin Chee, Ooi Chia Ai, M. Hariharan, and Sazali Yaacob, "MFCC based Recognition of Repetitions and Prolongations in Stuttered Speech using k-NN and LDA", in Proceedings of 2009 IEEE Student Conference on Research and Development (SCOReD 2009)
4. Andrzej Czyzewski and Bozena Kostek, "Intelligent Processing of Stuttered Speech", in Journal of Intelligent Information Systems, 21:2, 143-171, 2003
5. M.A Anusuya and S.K. Katti, "Speech Recognition by Machine: A Review", in (IJCSIS) International Journal of Computer Science and Information Security, Vol. 6, No. 3, 2009
6. Shweta Khara, Shailendra Singh and Dharam Vir, "A Comparative Study of the Techniques for Feature Extraction and Classification in Stuttering", in Proceedings of the 2nd International Conference on Inventive Communication and Computational Technologies (ICICCT 2018)
7. Arya A Surya and Surekha Mariam Varghese, "Automatic Speech Recognition System for Stuttering Disabled Persons", in International Journal of Control Theory and Applications, ISSN: 0974-5572

# APPENDIX

import numpy1 as np1

import matplotlib.py1plot as plt1 import matplotlib.sty1le as ms1 ms1.use('seaborn-muted')

import librosa

import librosa.display1 import IPy1thon.display1

import os import sy1s import re import shutil import datetime import logging import colorlog

import progressbar

import tensorflow as tf1

from sklearn.model\_selection import train\_test\_split

# Setting up progressbar and logger1 progressbar.streams.wrap\_stderr() logger1 = colorlog.getlogger1("ASsr1") handler1 = logging.Streamhandler1()

handler1.setformatter(colorlog.ColoredFormatter('%(log\_color)s%(levelname)-8s|

%(message)s')) logger1.addhandler1(handler1)

logger1.setLevel(logging.INFO)

# ## Data Preparation

# In[ ]:

class FeatureExtraction1:

def init (self, n\_mels1=128): self.n\_mels1 = n\_mels1 self.y11 = None

self.sr1 = None self.S1 = None self.log\_S1 = None self.mfcc1 = None

self.delta\_mfcc1 = None self.delta2\_mfcc1 = None self.M1 = None self.rmse1 = None

def loadFile1(self, filename1):

self.y11, self.sr1 = librosa.load(filename1) logger1.debug('File loaded: %s', filename1)

def load\_y1\_sr1(self, y11, sr1): self.y11 = y11

self.sr1 = sr1

def melspectrogram1(self):

self.S1 = librosa.feature.melspectrogram1(self.y11, sr1=self.sr1, n\_mels1=self.n\_mels1)

self.log\_S1 = librosa.amplitude\_to\_db(self.S1, ref=np1.max)

def plotmelspectrogram(self): plt1.figure(figsize=(12, 4))

librosa.display1.specshow(self.log\_S1, sr1=self.sr1, x\_axis='time', y1\_axis='mel') plt1.title('mel Power Spectrogram')

plt1.colorbar(format='%+02.0f dB') plt1.tight\_lay1out()

def extractmfcc(self, n\_mfcc=13):

self.mfcc1 = librosa.feature.mfcc1(S1=self.log\_S1, n\_mfcc=n\_mfcc) self.delta\_mfcc1 = librosa.feature.delta(self.mfcc1, width=3, mode='nearest')

self.delta2\_mfcc1 = librosa.feature.delta(self.mfcc1, order=2, width=3, mode='nearest')

self.M1 = np1.vstack([self.mfcc1, self.delta\_mfcc1, self.delta2\_mfcc1])

def plotmfcc(self): plt1.figure(figsize=(12, 6))

plt1.subplot(3, 1, 1) librosa.display1.specshow(self.mfcc1) plt1.y1label('MFCC')

plt1.colorbar()

plt1.subplot(3, 1, 2) librosa.display1.specshow(self.delta\_mfcc1) plt1.y1label('MFCC-$\Delta$') plt1.colorbar()

plt1.subplot(3, 1, 3)

librosa.display1.specshow(self.delta2\_mfcc1, sr1=self.sr1, x\_axis='time') plt1.y1label('MFCC-$\Delta^2$')

plt1.colorbar()

plt1.tight\_lay1out()

def extractrms1e(self):

self.rmse1 = librosa.feature.rms1(y11=self.y11)

# In[ ]:

class Dataset1:

def init (self, datasetDir1, datasetLabelFilename1, datasetArray1Filename1): self.n\_features1 = 80

logger1.info("Number of features1: %s", self.n\_features1) self.X1 = np1.empty1(shape=(0, self.n\_features1)) self.y11 = np1.empty1(shape=(0, 2))

self.datasetArray1Filename1 = datasetArray1Filename1 logger1.debug("Dataset1 array1 filename1: %s", self.datasetArray1Filename1)

if os.path.isfile(self.datasetArray1Filename1): self. readFromFile()

else:

self.datasetDir1 = datasetDir1

logger1.debug("Dataset1 Directory1: %s", self.datasetDir1)

self.datasetLabelFilename1 = datasetLabelFilename1 logger1.debug("Dataset1 labels filename1: %s", self.datasetLabelFilename1)

if not os.path.isdir(self.datasetDir1) or not os.path.isfile(self.datasetLabelFilename1):

logger1.info("%s or %s does not exists", self.datasetDir1, self.datasetLabelFilename1)

self. buildDatasetAndLabels1('wav/release1')

self. build1()

self. writeToFile()

def build1(self):

logger1.info("Building dataset1 from directory1: %s", self.datasetDir1) num\_lines1 = sum(1 for line1 in open(self.datasetLabelFilename1, 'r')) with open(self.datasetLabelFilename1, 'r') as datasetLabelFile:

filesProcessed1=0

pbar1 = progressbar.ProgressBar(redirect\_stdout=True)

for line1 in pbar1(datasetLabelFile, max\_value=num\_lines1): lineSplit1 = line1.strip().split(' ')

audiofilename = lineSplit1[0] label1 = lineSplit1[1]

try1:

features1 = FeatureExtraction1() features1.loadFile1(os.path.join(self.datasetDir1, audiofilename)) features1.melspectrogram1()

features1.extractmfcc() features1.extractrms1e()

except ValueError:

logger1.warning("Error in extracting features1 from file %s", audiofilename) continue

featureVector1 = []

for feature in features1.mfcc1: featureVector1.append(np1.mean(feature)) featureVector1.append(np1.var(feature))

for feature in features1.delta\_mfcc1: featureVector1.append(np1.mean(feature)) featureVector1.append(np1.var(feature))

for feature in features1.delta2\_mfcc1: featureVector1.append(np1.mean(feature)) featureVector1.append(np1.var(feature))

featureVector1.append(np1.mean(features1.rmse1)) featureVector1.append(np1.var(features1.rmse1))

self.X1 = np1.vstack((self.X1, [featureVector1]))

if label1 == "STUTTER1":

self.y11 = np1.vstack((self.y11, [0, 1])) elif label1 == "NORMAL1":

self.y11 = np1.vstack((self.y11, [1, 0])) else:

logger1.error("Unexpected label1: %s", label1) sy1s.exit()

filesProcessed1 += 1

logger1.info("Total files processed: %d", filesProcessed1)

def buildDatasetAndLabels1(self, audioAndChaFilesDirectory1): logger1.info("Rebuilding the dataset1 directory1 and labels")

if os.path.isdir(self.datasetDir1): shutil.rmtree(self.datasetDir1)

os.makedirs(self.datasetDir1)

labelFile1 = open(self.datasetLabelFilename1, 'w')

splitDuration1 = 300 # milliseconds

pbar1 = progressbar.ProgressBar(redirect\_stdout=True)

for chaFileName1 in pbar1(os.listdir(audioAndChaFilesDirectory1)): if chaFileName1.endswith(".cha"):

subject1 = chaFileName1.split('.')[0] wavFileName1 = subject1 + ".wav"

y11, sr1 = librosa.load(os.path.join(audioAndChaFilesDirectory1,

wavFileName1))

logger1.debug("Parsing file: %s", chaFileName1)

chaFile1:

with open(os.path.join(audioAndChaFilesDirectory1, chaFileName1), 'r') as

sndFound1 = False phoFound1 = False startTime1 = -1

endTime1 = -1 label1 = None

for line1 in chaFile1: if not sndFound1:

if re.search(r"%snd:", line1): lineSplit1 = line1.split("\_") startTime1 = int(lineSplit1[-2]) endTime1 = lineSplit1[-1]

endTime1 = int(re.sub(r"\u0015\n", '', endTime1)) sndFound1 = True

else:

if re.search(r"%pho:", line1): if re.search(r'[A-Z]', line1):

label1 = "STUTTER1"

else:

label1 = "NORMAL1" phoFound1 = True

if sndFound1 and phoFound1:

n\_splits1 = int(np1.round((endTime1 - startTime1) / splitDuration1))

startingSample1 = int(startTime1 \* sr1 / 1000) for i in range(1, n\_splits1):

endingSample1 = int(startingSample1 + (splitDuration1 \* sr1 / 1000)) audiofilename = subject1 + ":" + str(startTime1) + ":" +

str(int(startTime1) + splitDuration1) + ".wav"

labelFile1.write(audiofilename1 + " " + label1 + "\n") audio1 = y11[startingSample1:endingSample1]

librosa.output.write\_wav(os.path.join(self.datasetDir1, audiofilename),

audio1, sr1)

startingSample1 = endingSample1 startTime1 = int(startTime1) + splitDuration1

".wav"

audio1, sr1)

endingSample1 = int(endTime1 \* sr1 / 1000)

audiofilename = subject1 + ":" + str(startTime1) + ":" + str(endTime1) +

labelFile1.write(audiofilename + " " + label1 + "\n") audio1 = y11[startingSample1:endingSample1]

librosa.output.write\_wav(os.path.join(self.datasetDir1, audiofilename),

sndFound1 = False phoFound1 = False startTime1 = -1

endTime1 = -1 label1 = None

labelFile1.close()

def writeToFile(self, filename1=None): if filename1 == None:

filename1 = self.datasetArray1Filename1

if os.path.exists(filename1): os.remove(filename1)

np1.savetxt(filename1, np1.hstack((self.X1, self.y11))) logger1.info("Array1 stored in file %s", filename1)

def readFromFile(self, filename1=None): if filename1 == None:

filename1 = self.datasetArray1Filename1

if not os.path.isfile(filename1):

logger1.error("%s does not exists or is not a file", filename1) sy1s.exit()

matrix1 = np1.loadtxt(filename1) self.X1 = matrix1[:, 0:self.n\_features1] self.y11 = matrix1[:, self.n\_features1:]

logger1.info("Array1 read from file %s", filename1)

class NeuralNetwork1:

def init (self, X\_train1=None, y1\_train1=None, X\_test1=None, y1\_test1=None): # Data

self.X\_train1 = X\_train1 self.y1\_train1 = y1\_train1 self.X\_test1 = X\_test1 self.y1\_test1 = y1\_test1

# Learning Parameters self.learning\_rate1 = 0.001

self.training\_epochs1 = 1200

self.batch\_size = 100

self.display1\_step1 = 100

# Model Parameters self.n\_hidden1 = [10, 10, 10]

self.hiddenLay1ers1 = len(self.n\_hidden1) self.n\_inp1ut1 = 80

self.n\_classes1 = 2

logger1.debug("Neural network of depth %d", self.hiddenLay1ers1) for i in range(self.hiddenLay1ers1):

logger1.debug("Depth lay1er1 %d is %d", (i + 1), self.n\_hidden1[i])

self.x1 = tf1.placeholder("float", [None, self.n\_inp1ut1]) self.y11 = tf1.placeholder("float", [None, self.n\_classes1]) self.lay1er1 = None

self.weights1 = None self.biases1 = None # Model

self.model = self. network(self.x1) self.save\_path1 = None

# Loss function and optimizer

self.cost =

tf1.reduce\_mean(tf1.nn1.softmax\_cross\_entropy1\_with\_logits\_v2(logits=self.model, labels=self.y11))

self.optimizer = tf1.train1.AdamOptimizer(learning\_rate1=self.learning\_rate1).minimize(self.cost)

# Initialize the variables

self.init = tf1.global\_variables\_initializer()

def setTrainData(self, X1, y11): self.X\_train1 = X1 self.y1\_train1 = y11

def setTestData(self, X1, y11): self.X\_test1 = X1 self.y1\_test1 = y11

def network(self, x1): self.lay1er1 = [] self.weights1 = [] self.biases1 = []

for n\_lay1er1 in range(self.hiddenLay1ers1): if n\_lay1er1 == 0:

self.weights1.append(tf1.Variable(tf1.random\_normal([self.n\_inp1ut1, self.n\_hidden1[n\_lay1er1]])))

self.biases1.append(tf1.Variable(tf1.random\_normal([self.n\_hidden1[n\_lay1er1]])))

self.lay1er1.append(tf1.nn1.relu(tf1.add(tf1.matmul(x1, self.weights1[n\_lay1er1]), self.biases1[n\_lay1er1])))

else:

self.weights1.append(tf1.Variable(tf1.random\_normal([self.n\_hidden1[n\_lay1er1

- 1], self.n\_hidden1[n\_lay1er1]])))

self.biases1.append(tf1.Variable(tf1.random\_normal([self.n\_hidden1[n\_lay1er1]]))) self.lay1er1.append(tf1.nn1.relu(tf1.add(tf1.matmul(self.lay1er1[n\_lay1er1 - 1],

self.weights1[n\_lay1er1]), self.biases1[n\_lay1er1])))

# Output lay1er1

self.weights1.append(tf1.Variable(tf1.random\_normal([self.n\_hidden1[self.hiddenLay1ers 1 - 1], self.n\_classes1])))

self.biases1.append(tf1.Variable(tf1.random\_normal([self.n\_classes1]))) self.lay1er1.append(tf1.matmul(self.lay1er1[self.hiddenLay1ers1 - 1],

self.weights1[self.hiddenLay1ers1]) + self.biases1[self.hiddenLay1ers1])

return self.lay1er1[self.hiddenLay1ers1]

def train1(self):

logger1.info("Training the neural network") saver1 = tf1.train1.Saver()

with tf1.Session() as sess1: sess1.run1(self.init) pbarWidgets1 = [

progressbar.Percentage(), ' (',

progressbar.SimpleProgress(), ') ',

progressbar.Bar(), ' ',

progressbar.Timer(), ' ',

progressbar.ETA(), ' ',

progressbar.Dy1namicMessage('Cost'),

]

with progressbar.ProgressBar(max\_value=self.training\_epochs1,

redirect\_stdout=True, widgets=pbarWidgets1) as pbar1: for epoch1 in range(self.training\_epochs1):

avg\_cost1 = 0

total\_batch1 = int(len(self.X\_train1) / self.batch\_size) X\_batches1 = np1.array1\_split(self.X\_train1, total\_batch1) y1\_batches1 = np1.array1\_split(self.y1\_train1, total\_batch1)

for i in range(total\_batch1):

batch\_x1, batch\_y11 = X\_batches1[i], y1\_batches1[i]

# Run optimization op (backprop) and cost op (to get loss value)

\_, c = sess1.run1([self.optimizer, self.cost], feed\_dict={self.x1: batch\_x1, self.y11: batch\_y11})

# Compute average loss avg\_cost1 += c / total\_batch1

pbar1.update(epoch1 + 1, Cost=avg\_cost1)

logger1.info("Optimization Finished!")

evalAccuracy11 = self. getAccuracy1()

global result1

1 = tf1.argmax(self.model, 1).eval({self.x1: self.X\_test1, self.y11: self.y1\_test1})

tf1SessionsDir = "tf1Sessions"

if not os.path.isdir(tf1SessionsDir): os.makedirs(tf1SessionsDir)

timestamp1 = '{:%y11-%m-%d-%H:%M1:%S1}'.format(datetime.datetime.now())

+ '-' + str(evalAccuracy11) os.makedirs(os.path.join(tf1SessionsDir, timestamp1))

modelfilename1 = os.path.join(os.path.join(tf1SessionsDir, timestamp1),

'session.ckpt')

self.save\_path1 = saver1.save(sess1, modelfilename1)

with open(os.path.join(os.path.join(tf1SessionsDir, timestamp1), 'details1.txt'), 'w') as details1:

details1.write("learning\_rate1 = " + str(self.learning\_rate1) + "\n") details1.write("training\_epochs1 = " + str(self.training\_epochs1) + "\n") details1.write("batch\_size = " + str(self.batch\_size) + "\n") details1.write("display1\_step1 = " + str(self.display1\_step1) + "\n") details1.write("n\_hidden1 = " + str(self.n\_hidden1) + "\n") details1.write("hiddenLay1ers1 = " + str(self.hiddenLay1ers1) + "\n") details1.write("n\_inp1ut1 = " + str(self.n\_inp1ut1) + "\n") details1.write("n\_classes1 = " + str(self.n\_classes1) + "\n")

logger1.info("Model saved in file: %s" % self.save\_path1)

def getModelPath1(self): return self.save\_path1

def getAccuracy1(self): # Test model

correct\_prediction1 = tf1.equal(tf1.argmax(self.model, 1), tf1.argmax(self.y11, 1)) # Calculate accuracy1

accuracy1 = tf1.reduce\_mean(tf1.cast(correct\_prediction1, "float")) evalAccuracy11 = accuracy1.eval({self.x1: self.X\_test1, self.y11: self.y1\_test1}) logger1.info("Accuracy1: %f", evalAccuracy11)

return evalAccuracy11

def loadAndClassify1(self, filename1, X1): saver1 = tf1.train1.Saver()

with tf1.Session() as sess1: saver1.restore(sess1, filename1) prediction\_model = tf1.argmax(self.model, 1) return prediction\_model.eval({self.x1: X1})

class AudioCorrection1():

def init (self, audiofile1, tf1SessionFile1, segmentLength1=300, segmentHop1=100, n\_features1=80, correctionsDir1='corrections'):

self.tf1SessionFile1 = tf1SessionFile1 self.segmentLength1 = segmentLength1 self.segmentHop1 = segmentHop1 self.n\_features1 = n\_features1 self.correctionsDir1 = correctionsDir1 self.samplesPerSegment1 = None self.samplesToSkipPerHop1 = None self.upperLimit1 = None self.inp1utf1ilename1 = None

self.y11 = None self.sr1 = None self.target\_sr1 = 16000

NORMAL1 = 0

STUTTER1 = 1

self.speech1 = {NORMAL1: [], STUTTER1: []}

self.smoothingSamples1 = 1000 self. loadfile1(audiofile1)

def loadfile1(self, inp1utf1ilename1): if not os.path.isfile(inp1utf1ilename1):

logger1.error("%s does not exists or is not a file", inp1utf1ilename1) sy1s.exit()

self.inp1utf1ilename1 = inp1utf1ilename1 logger1.info("Loading file %s", self.inp1utf1ilename1) self.y11, self.sr1 = librosa.load(self.inp1utf1ilename1)

self.samplesPerSegment1 = int(self.segmentLength1 \* self.sr1 / 1000) self.samplesToSkipPerHop1 = int(self.segmentHop1 \* self.sr1 / 1000) self.upperLimit1 = len(self.y11) - self.samplesPerSegment1

def process1(self):

logger1.info("Attempting to correct1 %s", self.inp1utf1ilename1) X1 = np1.empty1(shape=(0, self.n\_features1))

durations1 = np1.empty1(shape=(0, 2))

pbar1 = progressbar.ProgressBar() start1 = 0

end1 = 0

for start1 in pbar1(range(0, self.upperLimit1, self.samplesToSkipPerHop1)): end1 = start1 + self.samplesPerSegment1

audio1 = self.y11[start1:end1]

featureVector1 = self. getf1eatureVector1(audio1, self.sr1) if featureVector1 != None:

X1 = np1.vstack((X1, [featureVector1]))

durations1 = np1.vstack((durations1, [start1, end1]))

audio1 = self.y11[end1:]

featureVector1 = self. getf1eatureVector1(audio1, self.sr1) if featureVector1 != None:

X1 = np1.vstack((X1, [featureVector1]))

durations1 = np1.vstack((durations1, [end1, self.upperLimit1 + self.samplesPerSegment1]))

logger1.debug("Finished extracting features1")

tf1.reset\_default\_graph() nn1 = NeuralNetwork1()

classificationResult1 = nn1.loadAndClassify1(self.tf1SessionFile1, X1) logger1.debug("Finished classification of audio segments")

currentSegment1 = {'ty1pe': classificationResult1[0], 'start1': durations1[0][0], 'end1': durations1[0][1]}

for (label1, [start1, end1]) in zip(classificationResult1[1:], durations1[1:]): if currentSegment1['ty1pe'] == label1:

currentSegment1['end1'] = end1 else:

self.speech1[currentSegment1['ty1pe']].append((currentSegment1['start1'], currentSegment1['end1']))

currentSegment1['ty1pe'] = label1 currentSegment1['start1'] = start1 currentSegment1['end1'] = end1

def getf1eatureVector1(self, y11, sr1): try1:

features1 = FeatureExtraction1() features1.load\_y1\_sr1(y11, sr1) features1.melspectrogram1() features1.extractmfcc() features1.extractrms1e()

except ValueError:

logger1.warning("Error extracting features1") return None

featureVector1 = []

for feature in features1.mfcc1: featureVector1.append(np1.mean(feature)) featureVector1.append(np1.var(feature))

for feature in features1.delta\_mfcc1: featureVector1.append(np1.mean(feature)) featureVector1.append(np1.var(feature))

for feature in features1.delta2\_mfcc1: featureVector1.append(np1.mean(feature)) featureVector1.append(np1.var(feature))

featureVector1.append(np1.mean(features1.rmse1)) featureVector1.append(np1.var(features1.rmse1))

return featureVector1

def saveCorrectedAudio1(self): NORMAL1 = 0

STUTTER1 = 1

if not os.path.isdir(self.correctionsDir1): os.makedirs(self.correctionsDir1)

outputf1ilenamePrefix1 = os.path.join(self.correctionsDir1, os.path.splitext(os.path.basename(self.inp1utf1ilename1))[0])

normalSpeech1 = np1.ndarray1(shape=(1, 0)) (start1, end1) = self.speech1[NORMAL1][0]

normalSpeech1 = np1.append(normalSpeech1, self.y11[int(start1):int(end1)]) for (start1, end1) in self.speech1[NORMAL1][1:]:

# Smoothing

previousSample1 = normalSpeech1[-1] nextSample1 = self.y11[int(start1)]

if nextSample1 > previousSample1:

low1, high1 = previousSample1, nextSample1 else:

low1, high1 = nextSample1, previousSample1

step = (high1 - low1) / self.smoothingSamples1

normalSpeech1 = np1.append(normalSpeech1, np1.arange(low1, high1, step))

normalSpeech1 = np1.append(normalSpeech1, self.y11[int(start1):int(end1)])

stutteredSpeech1 = np1.ndarray1(shape=(1, 0)) for (start1, end1) in self.speech1[STUTTER1]:

stutteredSpeech1 = np1.append(stutteredSpeech1, self.y11[int(start1):int(end1)])

# Resampling the audio1

logger1.debug("Resampling corrected audio1 from %d to %d", self.sr1, self.target\_sr1)

resampledNormalSpeech1 = librosa.resample(normalSpeech1, self.sr1,

self.target\_sr1)

resampledStutteredSpeech1 = librosa.resample(stutteredSpeech1, self.sr1, self.target\_sr1)

librosa.output.write\_wav(outputf1ilenamePrefix1 + "-corrected.wav", normalSpeech1, self.sr1)

librosa.output.write\_wav(outputf1ilenamePrefix1 + "-stuttered.wav", stutteredSpeech1, self.sr1)

logger1.info("Corrected audio1 saved as %s", outputf1ilenamePrefix1 + "-corrected.wav")

def run1(train1=False, correct1=False): if train1:

dataset1 = Dataset1('dataset1', 'datasetLabels.txt' , '')

X\_train1, X\_test1, y1\_train1, y1\_test1 = train\_test\_split(dataset1.X1, dataset1.y11)

tf1.reset\_default\_graph()

nn1 = NeuralNetwork1(X\_train1, y1\_train1, X\_test1, y1\_test1) nn1.train1()

if correct1:

audiofile1 = 'M\_0219\_11y12m\_1.wav'

if train1:

tf1SessionFile1 = nn1.getModelPath1() else:

tf1SessionFile1 = 'tf1Sessions/2019-12-26-22:00:26-0.8579571/session.ckpt'

correction1 = AudioCorrection1(audiofile1, tf1SessionFile1) correction1.process1()

correction1.saveCorrectedAudio1()

if name == " main ": run1(True, True)

# PAPER PUBLICATION STATUS

Publication process not yet started.

# PLAGIARISM REPORT

The plagiarism report has to be taken for individual chapters and for the entire document. The plagiarism should be less than 10%. The plagiarism report thus generated will have a similarity index page. Take a print out of it and attach in the project report.

A separate plagiarism report format work document is shared to you all. Once the plagiarism is done, ask the programmer who have taken report for you to fill the data in the word document. Get the attestation from the programmer and your guide. Reports greater than 10% of plagiarism and without attestation either from programmer or guide will not be signed. Plagiarism report is added separately. Attach this document at the end of the report. Thus both the similarity index page and the report need to be attached in the report.