HELWAN UNIVERSITY
Faculty of Computers and Artificial Intelligence
Computer Science Department

Stuttering Detection

A graduation project dissertation by:

[**Amira Alaa Eldin Abd Allah Abd Elkhalek (201900182)**]

[**Amaal Khalid Elsayed Khalil (201900171)**]

[**Radwa Mansour Ibrahim Ismail (201900302)**]

[**Omaima Kamal Mostafa El-Sayed Temraz (201900183)**]

[**Omnia Gad Hosny Gad Bayoumy (201900173)**]

[**Enji Mohammed Abdallah Shaban (201900188)**]

Submitted in partial fulfillment of the requirements for the degree of Bachelor of Science in Computers & Artificial Intelligence, at the Computer Science Department, the Faculty of Computers & Artificial Intelligence, Helwan University

Supervised by:

[Dr. Wessam El-Behaidy]

June 2023

جـامـعـة حـلـوان
كـلـيـة الـحـاسـبـات والـذكـاء الإصـطـنـاعـي
قـسـم عـلـوم الـحـاسـب

Stuttering Detection

رسالة مشروع تخرج مقدمة من:

[**اميرة علاء الدين عبدالله عبدالخالق** (201900182)]

[**أمال خالد السيد خليل** (201900171)]

[**رضوي منصور ابراهيم اسماعيل** (201900302)]

[**أميمة كمال مصطفي السيد تمراز** (201900183)]

[**أمنية جاد حسني جاد بيومي** (201900173)]

[**انجي محمد عبدالله شعبان** (201900188)]

رسالة مقدمة ضمن متطلبات الحصول على درجة البكالوريوس في الحاسبات والذكاء الإصطناعي، بقسم كلية الحاسبات والذكاء الإصطناعي، جامعة حلوان علوم الحاسب،

تحت إشراف:

(د/ وسام البهيدي)

2023 يونيو / حزيران

Abstract

Millions of people are affected by stuttering and other speech disfluencies. The vast majority of the world has suffered from stuttering during communication. So, stuttering detection and classification are important issues in speech therap. It can help therapists track speech disorders and this is also important as a way to speech therapy with the help of technology. In this paper, we combined the MFCC, mel-spectrogram and wave2vec to extract features, train the network, detect stuttering, and classify 5 dysfluency types such as blocks, prolongations, sound repetitions, word repetitions, interjections. We evaluate our system on FluencyBank and SEP-28K datasets and show that our system is effective and suitable for real-time applications.

Keywords Speech disorders →Stuttering; Disfluency, Stuttering Detection, deep learning, recurrent neural network, BLSTM, attention.

Acknowledgement

First and foremost, we would express our deepest gratitude and appreciation to our advisor Dr. Wessam El-Behaidy for her support, outstanding guidance, and encouragement throughout our graduation project. As she was always there for us providing more than the needed time and effort from her since the very beginning of our project. We would also like to thank our families, especially our parents, for their encouragement, patience, and assistance over the years. We are forever indebted to our parents, who have always kept us in their prayers and pushed us to Success. Finally, for our faculty for providing the suitable environment that leaded us to represent the best image that computer science graduates of Helwan University are supposed to represent.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

PWS	People who stutter
ALS	Amyotrophic lateral sclerosis
SLP	Speech-Lap Pathologist
DAF	Delayed Auditory Feedback
FAF	Frequency-shifting Auditory Feedback
WMFCC	Weighted Mel Frequency Cepstral Coefficient
MFCC	Mel Frequency Cepstral Coefficient
DNN	deep neural network
ECAPA-TDNN	emphasized channel attention, propagation, and aggregation-time delay
W2V2	wav2vec 2.0
MTL	Multi-task
ADV	Adversarial Learning
FFT	Fast Fourier Transform
DCT	Discrete Cosine Transform
STFT	Short-Time Fourier Transform
SE-ResNet	Squeeze-and-Excitation Residual Network
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network
UCLASS	University College Londons Archive of Stuttered Speech
RMSProp	Root Mean Square Propagation
VGG	for Visual Geometry Group

List of Equations

Chapter 1: Introduction

In this chapter we are going to discuss the overview of the project and know more about its scope and explain some terminologies we will find throughout the document.

1.1 INTRO

The speech disorders problem refers to the difficulties in the production of speech sounds. The various speech disorders include cluttering, lisping, dysarthria, stuttering, etc. Of these speech disorders, stuttering – also known as stammering – is the most predominant one. Stuttering, a neuro developmental speech disorder, caused by the failure of speech sensorimotor, is defined by the disturbance of uncontrolled utterances: interjections, and core behaviors: blocks, repetitions, and prolongations. People who stutter loses not only their confidence but also generate a negative attitude towards their communication skills. Furthermore, it ruins their self-confidence, relationship with others, employment opportunities, and opinions of others about them. Stuttering influence individuals of all ages, culture, and races irrespective of their intelligence and financial status. Many pieces of research have stated that about 70 million people that comprise 1% of the world population suffer from stuttering and it is more common in males as compared to females.

People with the stuttering problem face several difficulties in social and professional interactions. Researchers have categorized the factors that lead to stuttering as of three types, namely, development, neurogenic, and psychogenic.

People who stutter (PWS) may have three sorts of disfluencies: repetition of a sound, syllable, word or phrase, sound prolongation during which a sound is sustained for a markedly more extended period that may be traditional and silent blocks at starting of vocalization or word or within the middle of a word.

The stuttering detection problem is tackled from different angles with many different input features and many different methods.

1.2 Motivation

In this research, we aim to produce a model capable of detecting stuttering disease with high accuracy to provide maximum benefit to affected children and adults. Children may feel some fear of going to doctors or speech centers, so we aim to help children treat stuttering through medical and educational programs that help in detecting stuttering, classifying it into several types, determining the problem, and trying to make suggestions for treatment, which helps in saving the time, effort, and expenses for the parents.

Also there are many adults who suffers from stuttering, it may be difficult for them to go to speech centers due to their age. This application may prevent them from feeling embarrassed regarding this.

1.3 Problem Statement

People and children who have dysarthria always have difficulty communicating with others and learning; however, Private speech therapy is expensive and out of reach for the majority of families in the less affluent areas.

In other words, the problem is caused by several diseases like: stroke, brain tumor, traumatic head injury, cerebral palsy, Bell’s palsy, multiple sclerosis muscular dystrophy, amyotrophic lateral sclerosis (ALS), Guillain-Barre syndrome, Huntington’s disease, myasthenia gravis, Parkinson’s disease, Wilson’s disease, Tongue injury and some infections such as strep throat or tonsillitis. Some medications such as narcotics or tranquilizers can also affect your central nervous system.

Moreover, tragic outcomes emerge from professional adults' lack of knowledge and training regarding the problem of stuttering, including speech therapists, physicians, and educators.

In order to prevent stutters from having to return to the therapy centre repeatedly, speech therapy must be intense for two to three months and include a maintenance phase that lasts for at least a year. The efficiency of the homework assigned to stutters cannot be assessed, but it is crucial because the majority of people stutter in social circumstances.

Additionally, the conclusions reached by one Speech-Lap Pathologist (SLP) and those reached by another SLP may disagree. Thus, the speech therapies are administered at random by the SLPs because there is no good means to alter them by evaluating the effectiveness of the therapies.

1.4 Similar Systems

We found apps that use a similar approach to our technology that we can look at:

1. **Speak Up**

   Sensory Speak Up is a visually stimulating simple game style app that responds to sounds. Speak Up can be used to encourage children to vocalise and make sounds, either through the internal microphone or via and external microphone.

Designed to assist with speech therapy, the louder the voice sound, the bigger the shape or pattern becomes. Different activities either give a gross display of the volume or a short trail of volumes. The colours of the display can be set as well as the sound threshold for a visual response.

2. **DAF Assistant**

DAF Assistant implements Delayed Auditory Feedback (DAF) and Frequency-shifting Auditory Feedback (FAF) techniques that are known to help people with stuttering to speak more fluently.

DAF Assistant can be used by people who stutter to control their speech fluency, slow down speech rate, increase confidence level, and develop good speaking habits.
It is known fact that people with stuttering problem usually speak better when do it together with another speaker. This, so called "chorus effect", causes significant increase in speech fluency. Delayed auditory feedback and Frequency-shifting Auditory Feedback simulate chorus effect by changing your own voice.

DAF delays your voice to your ears a fraction of a second. The application provides delay range from 20 to 320 milliseconds with 10 millisecond increments. FAF shifts the pitch of your voice. The application provides pitch shift in the range from one-half octave down to one-half octave up. FAF enhances the effectiveness of the application when used simultaneously with DAF.

3. **DAF Professional**

DAF Professional has been created to help people who stammer or have a neurological disease such as Parkinson's Disease. It helps people to slow their rate of speech which makes it clearer to others.

DAF Professional is an easy to use professional speech therapy tool which could make a huge difference to you or someone you know.

4. **FluencyCoach**

FluencyCoach is a free software application that uses Altered Auditory Feedback technology to simulate the effects of “Choral Speech” (speaking simultaneously with another person). It has been known for years that choral speech can promote fluency in a person who stutters, and in some people who suffer from Parkinson's Disease.

In recent years, the application of Altered Auditory Feedback has received a lot of attention as the technology underlying the well-known anti-stuttering device SpeechEasy.

5. **Speech4good**

Speech4Good puts speech therapy tools in your pocket. With proven features for people with speech disorders like stuttering or articulation, Speech4Good helps you monitor and record your speech practice from your iPhone or iPad.

Speech4Good is designed for people with speech disorders and speech-language pathologists. Its features are developed based on therapy methods to help students with stuttering, fluency, articulation, aphasia, autism and other speech disorders.

6. **Stammurai**

   A Mobile app to help you learn and practice speech therapy exercises for stuttering.

1.5 Report Organization (Structure)

The Report was organized as follows.

Chapter 1: A brief description of the project, its goal or scope, and the guiding concept that guided the development of the project.

Chapter 2: reviews the work related to automatic detection of speech disorders.

Chapter 3: getting more in details about the background that had been used for implementation.

Chapter4: seeing the entire project and each module's design to clarify project.

Chapter 5 explains The proposed system's framework is detailed. It also contains a brief description of the rule Data used, feature extraction, classification techniques applicable and consists of experimental results and comparative analysis of the model Classification.

Chapter 6 provides a conclusion, future work and references.

Chapter 2: Related work

This section reviews work relating to recognition systems designed to detect or classify stuttering speech disorders; previous research has presented various methods and algorithms that have been applied to recognizing stuttering events from speech signals. The previous works conducted signifies the importance of feature extraction and classification methods in the stuttered events detection.

2.1Analysis of the Related Work

Gupta et al. proposed an automated and efficient method based on the Weighted Mel Frequency Cepstral Coefficient feature extraction algorithm and deep-learning Bidirectional Long-Short Term Memory neural network for classification of stuttered events. The disfluencies such as prolongation and syllable, word, and phrase repetition are accurately detectable using this method. This model can extract static as well as dynamic acoustic features by using WMFCC, which enhances the detection accuracy of stuttered events; and also reduces the computational overhead to the classification stage. The work has utilized the UCLASS stuttering dataset for analysis. The feature vectors are modeled by Bi-LSTM in both forward and backward directions and capable of learning the long dependencies, taking full account of disfluency patterns in speech frames. Experiments show that when the hyper parameters are reconfigured during the training of the model, results in an optimal configuration of parameters and leads to a highly accurate model. The optimally configured model proposed in this study is compared with the unidirectional LSTM model. The test results show that the proposed method reaches the best accuracy of 96.67%, as compared to the LSTM model. The promising recognition accuracy of 97.33%, 98.67%, 97.5%, 97.19%, and 97.67% was achieved for the detection of fluent, prolongation, syllable, word, and phrase repetition, respectively [1].

Bayerl et al [2]. show the applicability of wav2vec 2.0 (W2V2) as a feature extractor for dysfluency detection. All experiments are formulated as binary classification tasks of one specific dysfluency against all other samples, containing speech of the other five stuttering event types and fluent speech. They fine-tuned the Wav2Vec2.0 model on FluencyBank (English) [3] and KSoF (German)[4] datasets. They reported an average F1 score of 53.8% and 46.67% on the KSoF and FluencyBank datasets respectively [5].

Sheikh et al. [6] used podcast ids and proposed multitasking and adversarial-based robust SD via a multi-branched training scheme. The model tries to learn stutter representations from a stutter speech utterance that are podcast invariant but at the same time are stutter discriminative. They found that the blocks are the hardest to detect in the SEP-28k dataset. In addition, including fluent speech of PWS in training greatly affects its recognition performance [7]. The application of advanced DL architectures for SD is constrained by the limited size and high cost of the voice pathology datasets.

Baevski et al[8]. presented wav2vec 2.0, a framework for self-supervised learning of speech representations which masks latent representations of the raw waveform and solves a contrastive task over quantized speech representations. The experiments show the large potential of pre-training on unlabeled data for speech processing. wav2vec 2.0 outperforms the previous state of the art on the 100 hour subset while using 100 times less labeled data. Using just ten minutes of labeled data and pre-training on 53k hours of unlabeled data still achieves 4.8/8.2 WER on test-clean/other of Librispeech. This demonstrates the feasibility of speech recognition with limited amounts of labeled data.

Sheikh et al.[7], where they investigated the impact of speaker embeddings extracted from a pre-trained emphasized channel attention, propagation, and aggregation-timedelay neural network (ECAPA-TDNN) in SD in addition to speech embeddings from Wav2Vce2.0. They showed that the middle layers from Wav2Vec2.0 are good at capturing stuttering-related information. They also showed that fusing information from Wav2Vec2.0 and ECAPA-TDNN pre-trained can further improve the detection accuracy of stuttering types. In addition, they concatenated the information from several layers of Wav2Vec2.0 and reported the state-of-the-results (68.35% overall accuracy) in SD on the SEP-28k dataset.

Kourkounakis et al.[9] design FluentNet, a deep neural network (DNN), to accurately classify these disfluencies. FluentNet is an end-to-end deep neural network designed to accurately classify stuttered speech across six different stutter types: sound, word, and phrase repetitions, as well as revisions, interjections, and prolongations. This model uses a Squeeze-and-Excitation residual network to learn effective spectral frame-level speech representations, followed by recurrent bidirectional long short-term memory layers to learn temporal relationships from stuttered speech. A global attention mechanism was then added to focus on salient parts of speech in order to accurately detect the required influences. Through experiments, they demonstrate that FluentNet achieves state-of-the-art results on disfluency classification with respect to other works in the area as well as a number of benchmark models on the public UCLASS dataset. To tackle the issue of scarce stutter-related speech datasets, they also develop a synthetic dataset based on a non-stuttered speech dataset (LibriSpeech), which they entitle LibriStutter [10].

Sheikh et al. used end-to-end and speech embedding based systems trained in a self-supervised fashion to participate in the ACM Multimedia 2022 ComParE Challenge, specifically the stuttering sub-challenge. In particular, we exploit the embeddings from the pre-trained Wav2Vec2.0 model for stuttering detection (SD)on the KSoF dataset. After embedding extraction, we benchmark with several methods for SD. Our proposed self-supervised based SD system achieves a UAR of 36.9% and 41.0% on validation and test sets respectively, which is 31.32% (validation set) and 1.49% (test set) higher than the best (DeepSpectrum) challenge baseline (CBL). Moreover, we show that concatenating layer embeddings with MFCCs features further improves the UAR of 33.81% and 5.45% on validation and test sets respectively over the CBL. Finally, we demonstrate that the summing information across all the layers of Wav2Vec2.0 surpasses the CBL by a relative margin of 45.91% and 5.69% on validation and test sets respectively [11].

Sheikh et al[12]. Investigate the impact of multi-task (MTL) and adversarial learning (ADV) to learn robust stutter features. They evaluate their system on the SEP-28k stuttering dataset. They explore and provide the first ever deeper analysis by using MTL and ADV in the context of SD. Stuttering Detection models are based on time delay neural networks, The proposed ADV model is based on MTL framework with four components including the encoder E, which generates representations, the FluentNet F classifier branch, the stutter classifier module named as DisfluentNet D, and the speaker classifier S. They achieve up to 10%, 6.78%, and 2% improvement in repetitions, blocks, and interjections respectively over the baseline[13].

Sheikh et al. used a StutterNet to detect and classify several stuttering types. Our method uses a TDNN, which is trained on the MFCC input features. Only the core behaviors and fluent part of the stuttered speech were considered in this study. We evaluate our system on the UCLASS stuttering dataset. Experiments also yielded different results depending on the techniques used, with the basic StutterNet outperforming the latest in science in most cases of non-fluency detection, but showing slightly lower performance in prolongation and block detection with an average accuracy of 17.13%, 42.43% in comparison to 23.17%, 53.33% average accuracies of ResNet+BiLSTM respectively. The StutterNet outperforms ResNet+BiLSTM 4 in correctly detecting fluent speech with a difference of 11.63 points (66.63% for the baseline, and 55.00% for ResNet+BiLSTM)[14].

Chapter 3 :BACKGROUND

In this chapter, we mention the techniques used in our models and describe briefly each one of them.

3.1 wav2vec 2.0:

Our model is composed of a multi-layer convolutional feature encoder $f: x \rightarrow Z$ which takes as input raw audio X and outputs latent speech representations z1, …… , zT for T time-steps. They are then fed to a Transformer $g: Z \rightarrow C$ to build representations c1, …… , cT capturing information from the entire sequence [15, 16,17]. The output of the feature encoder is discretized to qt with a quantization module $Z \rightarrow Q$ to represent the targets **(Figure 1)** in the self-supervised objective. Compared to vq-wav2vec [18], our model builds context representations over continuous speech representations and self-attention captures dependencies over the entire sequence of latent representations end-to-end.

a. Feature encoder:

The encoder consists of several blocks containing a temporal convolution followed by layer normalization [18] and a GELU activation function [19]. The raw waveform input to the encoder is normalized to zero mean and unit variance. The total stride of the encoder determines the number of time-steps T which are input to the Transformer.

b. Contextualized representations with Transformers:

The output of the feature encoder is fed to a context network which follows the Transformer architecture [20, 15,21]. Instead of fixed positional embedding which encode absolute positional information, we use a convolutional layer similar to [22], 17,23] which acts as relative positional embedding. We add the output of the convolution followed by a GELU to the inputs and then apply layer normalization.

c. Quantization module:

For self-supervised training we discretize the output of the feature encoder z to a finite set of speech representations via product quantization [24]. This choice led to good results in prior work which learned discrete units in a first step followed by learning contextualized representations [16]. Product quantization amounts to choosing quantized representations from multiple codebooks and concatenating them. Given G codebooks, or groups, with V entries e ∈ R$R^{V \times d/G}$, we choose one entry from each codebook and concatenate the resulting vectors $e_1, \ldots, e_G$ and apply a linear transformation $R^d \rightarrow R^f$ to obtain q ∈ R^f.

The Gumbel softmax enables choosing discrete codebook entries in a fully differentiable way [25,26 ,27]. We use the straight-through estimator [28] and setup G hard Gumbel softmax operations [26]. The feature encoder output z is mapped to l ∈ $R^{G \times V}$ logits and the probabilities for choosing the v-th codebook entry for group g are

$$p_{g,v} = \frac{\exp(l_{g,v} + n_v)/\tau}{\sum_{k=1}^{V} \exp(l_{g,k} + n_k)/\tau}, \qquad (1)$$

where T is a non-negative temperature, n = -log(-log(u)) and u are uniform samples from U(0, 1). During the forward pass, codeword i is chosen by i=argmax$_j p_{g,j}$ and in the backward pass, the true gradient of the Gumbel softmax outputs is used.

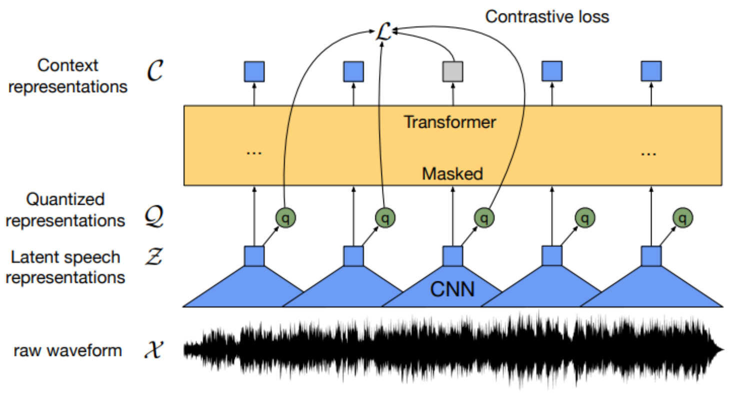


Fig. 3.1: wav2vec framework

d. Training

To pre-train the model we mask a certain proportion of time steps in the latent feature encoder space (a). The training objective requires identifying the correct quantized latent audio representation in a set of distractors for each masked time step (b) and the final model is fine-tuned on the labeled data (c).

a. Masking

We mask a proportion of the feature encoder outputs, or time steps before feeding them to the context network and replace them with a trained feature vector shared between all masked time steps; we do not mask inputs to the quantization module. To mask the latent speech representations output by the encoder, we randomly sample without replacement a certain proportion p of all time steps to be starting indices and then mask the subsequent M consecutive time steps from every sampled index; spans may overlap.

b. Objective

During pre-training, we learn representations of speech audio by solving a contrastive task $\boldsymbol{L_m}$ which requires to identify the true quantized latent speech representation for a masked time step within a set of distractors. This is augmented by a codebook diversity loss $\boldsymbol{L_d}$ to encourage the model to use the

$$\mathcal{L} = \mathcal{L}_m + \alpha \mathcal{L}_d \tag{2}$$

codebook entries equally often.

Where α is a tuned hyper parameter.

1. **Contrastive Loss:**

   Given context network output $\boldsymbol{c_t}$ centered over masked time step t, the model needs to identify the true quantized latent speech representation $\boldsymbol{q_t}$ in a set of K + 1 quantized candidate representations q' ∈ $\boldsymbol{Q_t}$ which includes $\boldsymbol{q_t}$ and K distractors [29,30]. Distractors are uniformly sampled from other

$$\mathcal{L}_m = -\log \frac{\exp(sim(\mathbf{c}_t, \mathbf{q}_t)/\kappa)}{\sum_{\tilde{\mathbf{q}} \sim \mathbf{Q}_t} \exp(sim(\mathbf{c}_t, \tilde{\mathbf{q}})/\kappa)} \tag{3}$$

   masked time steps of the same utterance. The loss is defined as

   where we compute the cosine similarity $sim(a;\ b) = a^t b / \|a\| \|b\|$ between context representations and quantized latent speech representations [31,32].

2. **Diversity Loss:**

   The contrastive task depends on the codebook to represent both positive and negative examples and the diversity loss $\boldsymbol{L_d}$ is designed to increase the use of the quantized codebook representations [33]. We encourage the equal use of the V entries in each of the G codebooks by maximizing the entropy of the averaged softmax distribution l over the codebook entries for each codebook $\boldsymbol{p'_g}$ across a batch of utterances; the

$$\mathcal{L}_d = \frac{1}{GV} \sum_{g=1}^{G} -H(\bar{p}_g) = \frac{1}{GV} \sum_{g=1}^{G} \sum_{v=1}^{V} \bar{p}_{g,v} \log \bar{p}_{g,v} \tag{4}$$

   softmax disribution does not contain the gumbel noise nor a temperature:

c. Fine-tuning

Pre-trained models are fine-tuned for speech recognition by adding a randomly initialized linear projection on top of the context network into C classes representing the vocabulary of the task [17]. For Librispeech, we have 29 tokens for character targets plus a word boundary token. Models are optimized by minimizing a CTC loss [34]and we apply a modified version of SpecAugment [35] by masking to time-steps and channels during training which delays overfitting and significantly improves the final error rates, especially on the Libri-light subsets with few labeled examples.

3.2 Weighted Mel-frequency cepstral coefficients Feature Extraction

The first step of speech recognition process is to extract the features from the speech signal. The purpose of feature extraction is two-fold: first is to compress the speech signal into features, and second is to use features that are insensitive to speech variations, changes of environmental conditions and independent of speaker. The procedural steps of MFCC feature extraction is described as follows:

1. Pre-emphasis

The speech signal spectrum is pre-emphasized by approximately 20 dB per decade to flatten the spectrum of the speech signal [36]. The pre-emphasis filter is used to offset the negative spectral slope of the speech signal to improve the efficiency of the spectral analysis [37]. The filter transfer function is given by $H(z) = 1 - az^{-1}$, where (a) is between 0.9 and 1 Fig. 2. shows the effect of pre-emphasis on the spectrum of a speech signal with the value of 0.97 for a.

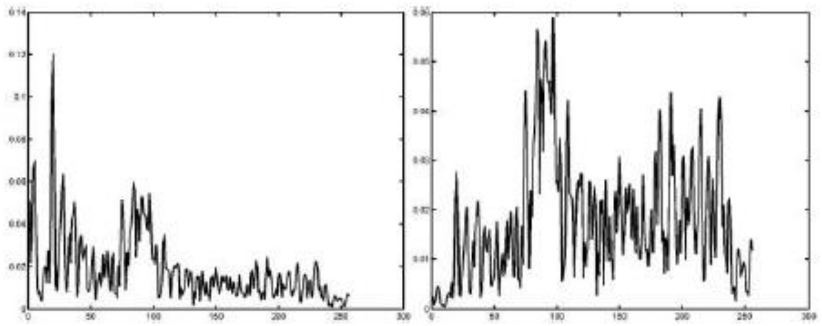


Fig. 3.2: Spectrum before (left) and after Pre-emphasis (right)

2. Framing

Since the human speech signal is slowly time varying, it can be treated as a stationary process when considered under a short time duration [36]. Therefore, the speech signal is usually separated into small duration blocks, called frames, and the spectral and cepstral analysis is performed on these frames. Typically, the frame length is kept as 25 milliseconds and the neighboring frames are overlapped by 10 milliseconds. The frame shift is the frame length minus the frame overlap.

3. Windowing

After being partitioned into frames, each frame is multiplied by a window function prior to the spectral analysis to reduce the discontinuity introduced by the framing process by attenuating the values of the speech samples at the beginning and end of each frame. Typically, Hamming window is used [38].

4. Spectral Estimation

The spectral coefficients of the speech frames are estimated using the Fast Fourier Transform (FFT) algorithm. These coefficients are complex numbers containing both magnitude and phase information. However, for speech recognition, the phase information is usually discarded and only the magnitude of the spectral coefficients is retained [38].

5. Mel Filtering

The spectrum of speech signal is then filtered by a group of triangular bandpass filters as shown in Fig. 3 that simulate the characteristics of human's ear. The purpose of Mel filtering is to model the human auditory system that perceives sound in a nonlinear frequency binning [36].

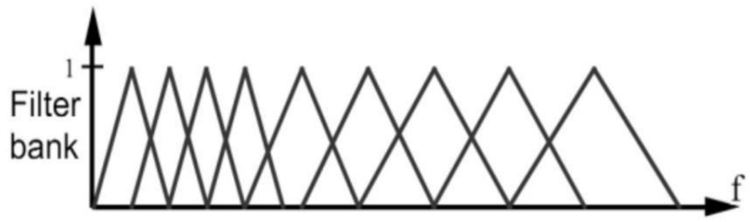


Fig. 3.3: Triangular bandpass Mel filter bank

The ears analyze the spectrum of the sound in groups according to a series of overlapped critical bands. The critical bands are distributed in a way that the frequency resolution is high in low frequency region and low in high frequency region. The bandwidth of the window is narrow in low frequency and gradually increases for high frequency. The edge of the window is arranged so that it coincides with the center of the neighboring window [39] . To decide the location of the Mel frequency of the center of the windows, the Mel frequencies for minimum and maximum linear frequency are first calculated using:

$$f_{Mel} = 2595 \times \log_{10}(1 + f/700) \qquad (5)$$

where f_{Mel} is the Mel frequency corresponding to the linear frequency f. The windows are linearly distributed in the Mel frequency scale, but when converted back to linear frequency, the center frequencies of the windows are logarithmically distributed.

6. Logarithmic Compression

While the Mel filtering approximates the non-linear characteristics of the human auditory system in frequency, the natural logarithm deals with the loudness nonlinearity. It approximates the relationship between the human's perception of loudness and the sound intensity [40]. Besides this, it converts the multiplication relationship between parameters into addition relationship [38]. The convolutional distortions, such as the filtering effect of microphone and channel, and the multiplication in frequency domain, become simple addition after the logarithm. The log Mel filter bank coefficients are computed from the filter outputs as:

$$s(m) = 20\log_{10}\left(\sum_{K=0}^{N-1} X\,|(K)|H(K)\right), \ 0 < m < M \qquad (6)$$

where M is the number of Mel filters (20 to 40), $X(k)$is the N-point FFT of the specific window frame of the input speech signal, and $H(k)$ is the Mel filter transfer function[38] .

7. Log Energy

In addition to the above MFCC features, the energy of the speech frame is also used as a feature [38]. The log energy, denoted as log, is calculated directly from the time-domain signal of a frame as:

$$\log E = \log \sum_{n=1}^{N} x(N)^2 \qquad (7)$$

where $x(n)$ is the speech sample and N is the length of the frame. In this work, the cepstral coefficient (0) is replaced by log E to give a more accurate energy feature.

8. Discrete Cosine Transform (DCT)

The cepstrum is defined as the inverse Fourier transform of the log magnitude of Fourier transform of the signal. Since the log Mel filter bank coefficients are real and symmetric, the inverse Fourier transform operation can be replaced by DCT to generate the cepstral coefficients [41]. This step is crucial in speech recognition as it can separate the vocal tract shape function from the excitation signal of the speech production model. The lower order cepstral coefficients represent the smooth spectral shape or vocal tract shape, while the higher order coefficients represent the excitation information [37].

The cepstral coefficients are the DCT of the M filter outputs obtained as:

$$\boldsymbol{C_{(n)}} = \sum_{m=0}^{M-1} \boldsymbol{S(m)} \cos\left(\frac{\pi n(m-\frac{1}{2})}{M}\right) \qquad \textbf{(8)}$$

Typically, the first 13 cepstral coefficients are used. Another benefit of DCT is that the generated MFCC coefficients (n) are less correlated than the log Mel filter bank coefficients.

3.3 Mel Spectrogram

A spectrogram is a visual depiction of the spectrum of frequencies of an audio signal as it varies with time. Hence it includes both time and frequency aspects of the signal. It is obtained by applying the Short-Time Fourier Transform (STFT) on the signal. In the simplest of terms, the STFT of a signal is calculated by applying the Fast Fourier Transform (FFT) locally on small time segments of the signal.

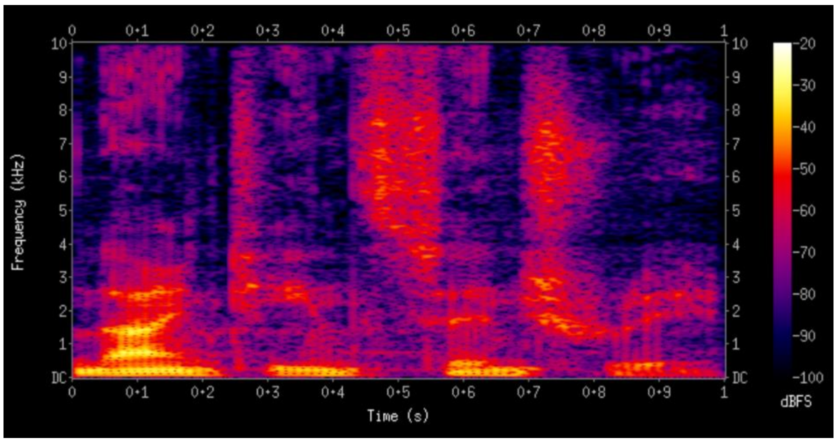


Fig 3.4: Spectrogram plot

Apparently, we humans perceive sound logarithmically. We are better at detecting differences in lower frequencies than higher frequencies. For example, we can easily tell the difference between 500 and 1000 Hz, but we will hardly be able to tell a difference between 10,000 and 10,500 Hz, even though the distance between the two pairs is the same. Hence, the **mel scale** was introduced. It is a logarithmic scale based on the principle that equal distances on the scale have the same perceptual distance.

Conversion from frequency (f) to mel scale (m) is given by

$$\mathbf{M = 2595 \cdot \log\left(1+ \frac{f}{700} \right)} \qquad \mathbf{(9)}$$

A mel-spectrogram is a therefore a spectrogram where the frequencies are converted to the mel scale.

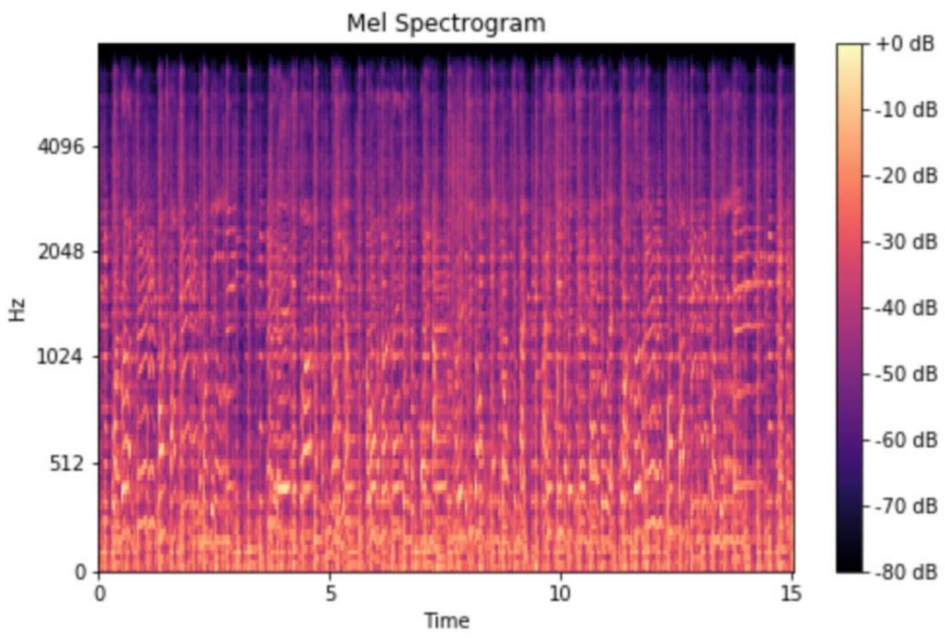


Fig 3.5: Mel Spectrogram plot

3.4Squeeze-and-Excitation Residual Network (SE-ResNet):

Residual networks and squeeze-and-excitation (SE) networks [42] are relatively new in the field of deep learning, both have proven to improve on previous state-of-the-art models in a variety of different application areas [43], [44].

The ResNet architecture has proven a reliable solution to the vanishing or exploding gradient problems, both common issues when back-propagating through a deep neural network. In many cases, as the model depth increases, the gradients of weights in the model become increasing smaller, or inversely, explosively larger with each layer. This may eventually prevent the gradients from actually changing the weights, or from the weights becoming too large, thus preventing improvements in the model. A ResNet, overcomes this by utilizing shortcuts all through its CNN blocks resulting in norm-preserving blocks capable of carrying gradients through very deep models.

Squeeze-and-excitation modules have been recently proposed and have shown to outperform various DNN models using previous CNN architectures, namely VGG and ResNet, as their backbone architectures [42]. SE networks were first proposed for image classification, reducing the relative error compared to previous works on the ImageNet dataset by approximately 25% [42].

Every kernel within a convolution layer of a CNN results in an added channel (depth) for the output feature map. Whereas recent works have focused on expanding on the spectral relationships within these models [45] [46], SE-blocks build stronger focus on channel-wise relationships within a CNN. These blocks consist of two major operations. The squeeze operation aggregates a feature map across both its height and width resulting in a one-dimensional channel descriptor. The excitation operation consists of fully connected layers providing channel-wise weights, which are then applied back to the original feature map.

To exploit the capabilities of both ResNet and SE architectures and learn effective spectral frame-level representations from the input, we use an SE-ResNet model in our end-to end network. The network consists of 8 SE-ResNet blocks, as shown in Figure 6(a). Each SE-ResNet block in FluentNet, illustrated in Figure 6(b), consists of three sets of two dimensional convolution layers, each succeeded by a batch normalization and Rectified Linear Unit (ReLU) activation function. A separate residual connection shares the same input as the block’s non-identity branch, and is added back to the non-identity branch before the final ReLU function, but after the SE unit (described below). Each residual connection contains a convolution layer followed by batch normalization. The Squeeze-and-Excitation unit within each SE-ResNet block begins with a global pooling layer. The output is then passed through two fully connected layers: the first followed by a ReLU activation function, and the second succeeded with a sigmoid gating function. The main convolution branch is scaled with the output of the SE unit using channel-wise multiplication.

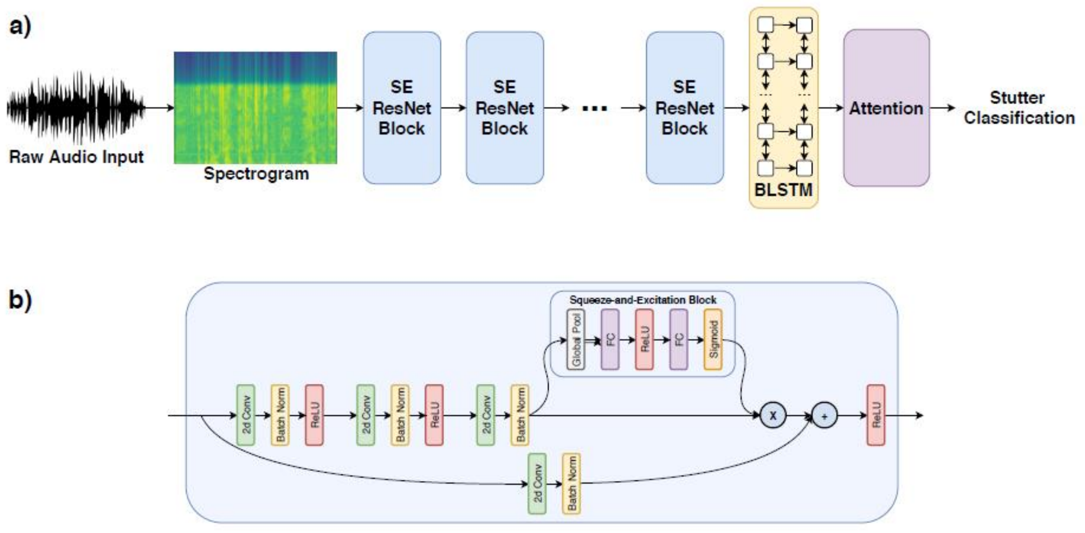


Fig. 3.6: a) A full workflow of FluentNet is presented. This network consists of 8 SE residual blocks, two BLSTM layers, and a global attention mechanism. b) The breakdown of a single SE-ResNet block in FluentNet is presented [10]

3.5 Bi-Directional Long-Short Term Memory

Deep learning Bi-LSTM is applied for the classification of stuttered speech samples. It is composed of LSTM cells (Fig. 7). The set of features vectors discussed in the above section are set as input to the classifier. The model is trained and validated with 80% and 20% of the speech samples of the data store, respectively. The remaining of the samples are used for testing the model.

1. **Long-Short Term Memory:**

LSTM is a specialized Recurrent Neural Network (RNN) architecture, competent in learning long term dependencies [47]. RNN suffers from short-term memory, caused by vanishing gradient problem. To mitigate this problem, LSTM has a hidden layer known as the LSTM cell. LSTM cells are built with various gates and cell state that can regulate the flow of information. Like RNNs, at each time iteration,t, the LSTM cell has the layer input, x_t, and the layer output,h_t . The cell also takes the cell input state, $\tilde{C}_t$, the cell output state, C_t, and the previous cell output state,C_{t-1} . LSTM architecture has three gates, namely, forget, input, and output gate denoted as f_t,i_t , and o_t, respectively.

The cell state act as the network memory, conveying valuable information across the entire sequence. The gates are specific neural networks that determine which information is permitted on the cell state. Throughout the training, the gates will learn which information is essential to retain or forget. The value of gates and cell state can be determined by using the following (10) to (13):

$$\boldsymbol{f_t} = \boldsymbol{\sigma}(\boldsymbol{w_f x_t} + \boldsymbol{U_f h_{t-1}} + \boldsymbol{b_f}) \qquad \textbf{(10)}$$

$$\boldsymbol{i_t} = \boldsymbol{\sigma}(\boldsymbol{w_i x_t} + \boldsymbol{U_i h_{t-1}} + \boldsymbol{b_i}) \qquad \textbf{(11)}$$

$$\boldsymbol{o_t} = \boldsymbol{\sigma}(\boldsymbol{w_o x_t} + \boldsymbol{U_o h_{t-1}} + \boldsymbol{b_o}) \qquad \textbf{(12)}$$

$$\boldsymbol{\widetilde{C}_t} = \mathbf{tanh}(\boldsymbol{w_c x_t} + \boldsymbol{U_c h_{t-1}} + \boldsymbol{b_c}) \qquad \textbf{(13)}$$

where w_f, w_i , w_o, and w_c are the weights connecting the hidden layer input to all the gates and input cell state. The U_f, U_i , U_o , and U_c are the weight matrices mapping previous cell output state to all the gates and input cell state. The b_f, b_i , b_o , and b_c are bias vectors. The σ and are the sigmoid and tanh activation function, respectively. The cell output state, C_t, and the layer output, h_t, at each time iteration t, can be calculated as in (14)-(15):

$$\boldsymbol{C_t} = \boldsymbol{f_t} * \boldsymbol{C_{t-1}} + \boldsymbol{i_t} * \boldsymbol{\widetilde{C}_t} \qquad \textbf{(14)}$$

$$\boldsymbol{h_t} = \boldsymbol{o_t} * \mathbf{tanh}(\boldsymbol{C_t}) \qquad \textbf{(15)}$$

The result of the LSTM layer should be a vector of all the outputs, represented as Y_T = [h_{T-n} , …………… , h_{T-1}].

2. **Bidirectional LSTM:**

The Bi-LSTM are originated from bidirectional RNN [48]. It processes sequential data with two different hidden layers, in both forward and backward directions, and links them to the same output layer. Across certain areas, bidirectional networks are considerably stronger than unidirectional ones, such as speech recognition [49].

Fig. 8 represents an unfolded Bi-LSTM layer structure containing a forward and a backward LSTM layer [50]. The output sequence of the forward layer, $\overrightarrow{h}$, is determined iteratively using inputs in a definite sequence, while the output sequence of backward layer, $\overleftarrow{h}$, is determined using the reversed input. The forward and backward layer outputs are computed using standard LSTM by (10) - (15). The Bi-LSTM layer produces an output vector,Y_T , which defines each element by the following Equation (16).

$$\boldsymbol{y_t} = \boldsymbol{\sigma}(\overrightarrow{\boldsymbol{h}}, \overleftarrow{\boldsymbol{h}}) \qquad \textbf{(16)}$$

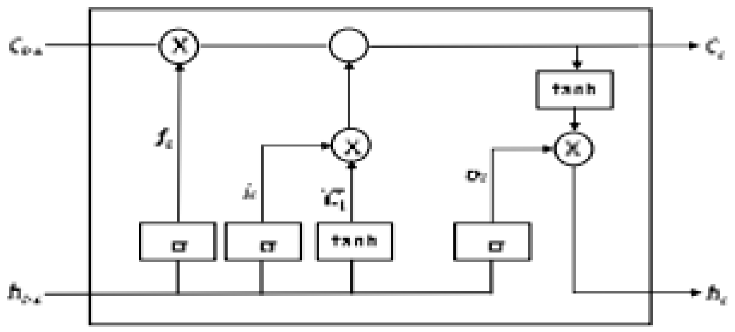


Fig3.7. LSTM Cell

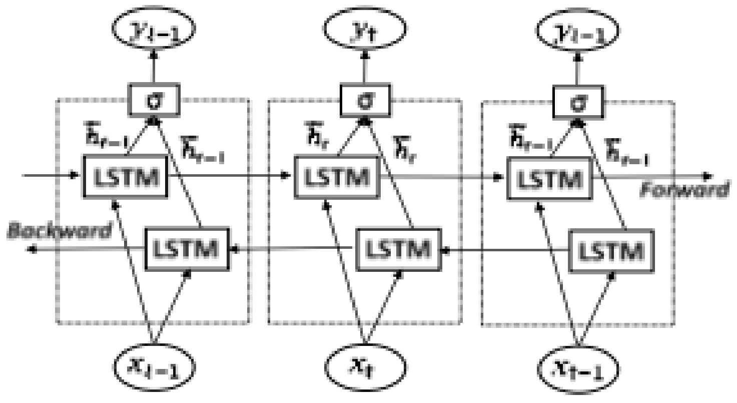


Fig 3.8. Structure of an unfolded Bi-LSTM Layer

where σ function combines the two output sequences. It can be a summation function, a multiplication function, a concatenating function, or an average function. Similar to the LSTM layer, a vector, Y_T = [h_{T-n} , …………… , h_{T-1}] represents the final output of a Bi-LSTM layer.

3.6VGGNet Architecture

Its model used in object detection in image where we used its idea in multilabel classification since we couldn't find any paper that worked in multilabel classification so we used the idea of cnn model then maxpooling than at the end dense layers.

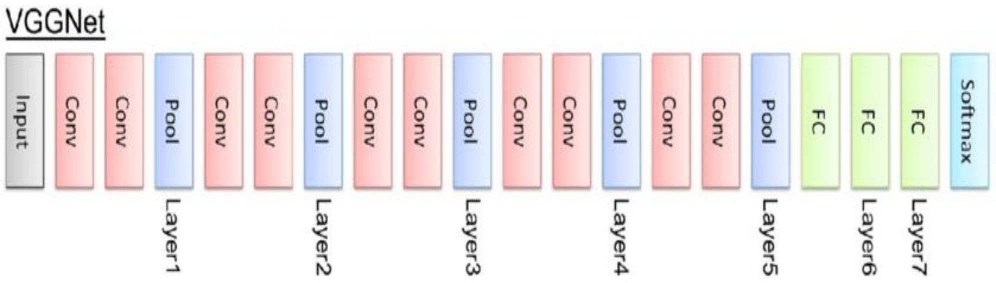


Fig 3.9

Structure of VGGNet model

In VGGNet architecture [64] image is passed through a stack of convolutional (conv.) layers, where we use filters with a very small receptive field: 3 × 3 (which is the smallest size to capture the notion of left/right, up/down, center). In one of the configurations we also utilise 1 × 1 convolution filters, which can be seen as a linear transformation of the input channels (followed by non-linearity). The convolution stride is fixed to 1 pixel; the spatial padding of conv. layer input is such that the spatial resolution is preserved after convolution, i.e. the padding is 1 pixel for 3 × 3 conv. layers. Spatial pooling is carried out by five max-pooling layers, which follow some of the conv. layers (not all the conv. layers are followed by max-pooling). Max-pooling is performed over a 2 × 2 pixel window, with stride 2. A stack of convolutional layers (which has a different depth in different architectures) is followed by three Fully-Connected (FC) layers: the first two have 4096 channels each, the third contains 1000 channels (one for each class). The final layer is the soft-max layer. The configuration of the fully connected layers is the same in all networks. All hidden layers are equipped with the rectification (ReLU [65]) non-linearity.

3.7 InceptionV1 Model:

The InceptionV1 [66] is a deep learning model based on Convolutional Neural Networks, which is used for image classification, we used its idea in the experiments of multilabel classification, we concatenate the output of cnn layers to make input for next one.

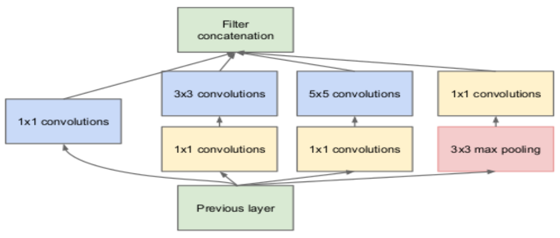


Fig 3.10: part of inceptionv1 model structure

Inception model considered multiple deep layers of convolutions results in the overfitting of the data, so to avoid this from happening it uses the idea of using multiple filters of different sizes on the same level so instead of having deep layers, we have parallel layers making our model wider rather than making it deeper. The computational cost of Inception is also much lower than VGGNet [64]. This has made it feasible to utilize Inception networks in big-data scenarios, where huge amount of data needed to be processed at reasonable cost or scenarios where memory or computational capacity is inherently

3.8 ReLU Function

The Rectified Linear Unit has become very popular in the last few years.

Mathematically it can be represented as:

$$ReLU$$

$$f(x) = max\ (0, x) \qquad \textbf{(17)}$$

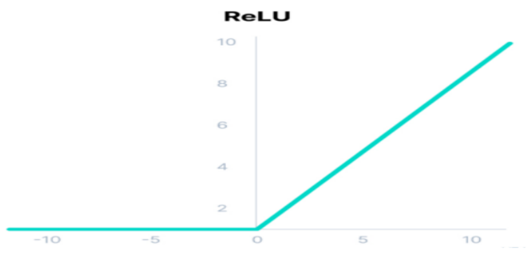


Fig 3.11. Relu Function

In other words, the activation is simply thresholded at zero (see image above). There are several pros and cons to using the ReLUs:

(+) It was found to greatly accelerate the convergence of stochastic gradient descent compared to the sigmoid/tanh functions. It is argued that this is due to its linear, non-saturating form.

(+) Compared to tanh/sigmoid neurons that involve expensive operations (exponentials, etc.), the ReLU can be implemented by simply thresholding a matrix of activations at zero.

(-) Unfortunately, ReLU units can be fragile during training and can “die”. For example, a large gradient flowing through a ReLU neuron could cause the weights to update in such a way that the neuron will never activate on any datapoint again. If this happens, then the gradient flowing through the unit will forever be zero from that point on. That is, the ReLU units can irreversibly die during training since they can get knocked off the data manifold. For example, you may find that as much as 40% of your network can be “dead” (i.e. neurons that never activate across the entire training dataset) if the learning rate is set too high. With a proper setting of the learning rate this is less frequently an issue. [51]

3.9 Parametric ReLU Function

Parametric ReLU is another variant of ReLU that aims to solve the problem of gradient's becoming zero for the left half of the axis.

This function provides the slope of the negative part of the function as an argument *a*. By performing backpropagation, the most appropriate value of *a* is learnt.

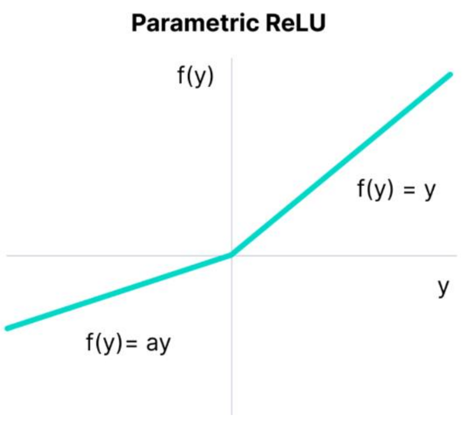


Fig 3.12. PRelu Function

Mathematically it can be represented as:

$$\text{Parametric ReLU}$$
$$f(x) = max\ (ax, x) \qquad \textbf{(18)}$$

Where *"a"* is the slope parameter for negative values.

The parameterized ReLU function is used when the leaky ReLU function still fails at solving the problem of dead neurons, and the relevant information is not successfully passed to the next layer.

This function's limitation is that it may perform differently for different problems depending upon the value of slope parameter ***a*** .[52]

3.10 Leaky ReLU function

Leaky ReLU function is an improved version of the ReLU activation function. As for the ReLU activation function, the gradient is 0 for all the values of inputs that are less than zero, which would deactivate the neurons in that region and may cause dying ReLU problem.

Leaky ReLU is defined to address this problem. Instead of defining the ReLU activation function as 0 for negative values of inputs(x), we define it as an extremely small linear component of x. Mathematically it can be represented as:

$$\text{Leaky ReLU}$$
$$f(x) = max\,(0.1x, x) \qquad \textbf{(19)}$$

This function returns x if it receives any positive input, but for any negative value of x, it returns a really small value which is 0.01 times x. Thus it gives an output for negative values as well. By making this small modification, the gradient of the left side of the graph comes out to be a non-zero value. Hence, we would no longer encounter dead neurons in that region. [53]

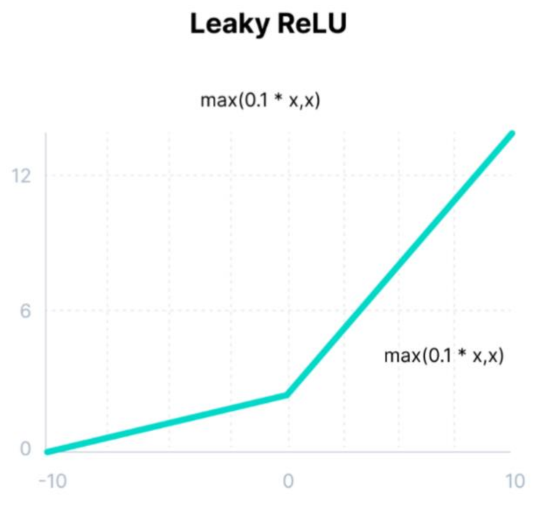


Fig 3.13. Leaky ReLU function

3.11 Swish function

The Swish function**,** or the self-gated function, is just another activation function proposed by **Google**. It has been proposed as a possible improvisation to the already existing sigmoid function.

We define the Swish activation function as follows:

$$\boldsymbol{Swish(x) = x * \text{sigmoid}(x) = \frac{x}{1+e^{-x}}} \qquad \textbf{(20)}$$

Swish activation is used in the Long Short-Term Memory (LSTM) neural networks, which are used extensively in sequence prediction and likelihood problems it is better than **ReLU** in the case of deeper neural networks that are trained on datasets with interlinked and complex patterns. [54]

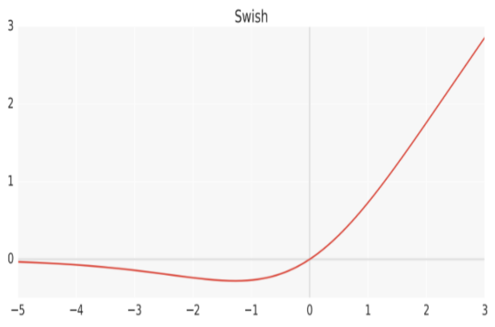


Fig 3.14. Swish function

The difference between ReLU and Swish:

Most activation functions are described to be monotonic. In the comparison between Swish and ReLU, it is notable that in a particular region of increasing, negative x-coordinates, the corresponding activations decrease and form a slight concavity in the graph pattern.This bypasses the monotonic property for certain trends, and highlights a unique feature of Swish.

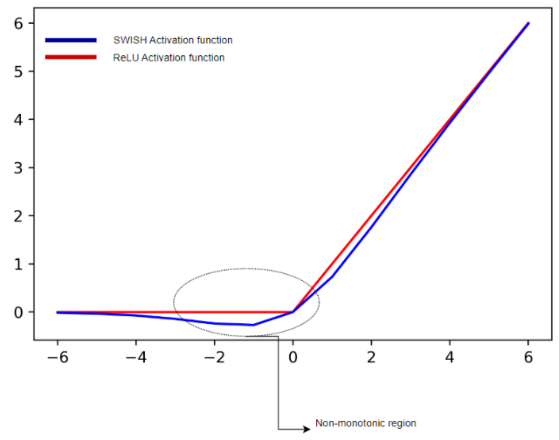


Fig 3.15. ReLU VS Swish

Advantages of the Swish function:

- The function allows data normalization and leads to quicker convergence and learning of the neural network.
- It works better in deep neural networks that require LSTM, compared to ReLU.

- With deeper neural networks requiring minor updates to the gradient during backpropagation, the update is not enough. This leads to the **vanishing gradient problem** in the case of the sigmoid and ReLU activation functions. Swish can work around and prevent the vanishing gradient program, and hence allow training for small gradient updates.

Limitations of the Swish function:

- The function is time-intensive to compute for deeper layers with large parameter dimensions.
- Lower sums of weighted inputs which would want to increase their output would not work because the function has a non-monotonic region for negative values closer to zero. [55]

3.12 Adamax optimizer

It’s a variant of Adam based on the infinity norm, it is a first-order gradient-based optimization method. [63]

The update rule for parameters:

$gt = \nabla\theta ft(\theta t-1)$ (Get gradients w.r.t. stochastic objective at timestep t) **(21)**

$mt = \beta 1 \cdot mt-1 + (1-\beta 1) \cdot gt$ (Update biased first moment estimate) **(22)**

$ut = \max(\beta 2 \cdot ut-1, |gt|)$ (Update the exponentially weighted infinity norm) **(23)**

$\theta t = \theta t-1 - (\alpha/(1-\beta\ t\ 1)) \cdot mt/ut$ (Update parameters) **(24)**

$bt \leftarrow vt/(1-\beta t2)$ (Compute bias-corrected second raw moment estimate) **(25)**

$\theta t \leftarrow \theta t-1 - \alpha \cdot mb\ t/(\sqrt{vbt} + \varepsilon)$ (Update parameters) **(26)**

$\beta 1, \beta 2 \in [0, 1)$: Exponential decay rates Require

$f(\theta)$: Stochastic objective function with parameters θ

θ: Initial parameter vector

$m \leftarrow 0$ (Initialize 1st moment vector)

$u0 \leftarrow 0$ (Initialize the exponentially weighted infinity norm)

$t \leftarrow 0$ (Initialize time-step)

And the $(\alpha/(1-\beta\ t\ 1))$ is the learning rate with the bias-correction term for the first moment, all operations on vectors are element-wise.

Advantages of the Adamax optimizer:

Its capable of adjusting the learning rate based on data characteristics, it is suited to learn time-variant process, e.g., speech data with dynamically changed noise conditions.

3.13 Binary Focal Crossentropy

It used in multilabel model to help in increasing the accuracy as the dataset isn't balanced.

The Focal Loss is designed to address the one-stage object detection scenario in which there is an extreme imbalance between foreground and background classes during training (e.g., 1:1000). It introduce the focal loss starting from the cross entropy (CE) loss for binary classification:

$$CE(p, y) = \begin{Bmatrix} -\log(p) & \text{if } y = 1 \\ \log(1 - p) & \text{otherwise} \end{Bmatrix} \quad (25)$$

It helps when the large class imbalance encounter during the training and easily classify negatives comprise the majority of the loss and dominate the gradient. While α balances the importance of positive/negative examples, it does not differentiate between easy/hard examples. Instead, it propose reshape to the loss function to down-weight easy examples and focus the training on hard negatives by adding a modulating factor (1 − pt) γ to the cross entropy loss, with tunable focusing parameter $\gamma \geq 0$.

We define the α-balanced variant of the focal loss:

$$FL(pt) = -\alpha(1 - pt)^{\gamma} \log(pt) \quad (26)$$

When an example is misclassified and pt is small, the modulating factor is near 1 and the loss is unaffected. As $pt \rightarrow 1$, the factor goes to 0 and the loss for well-classified examples is down-weighted. The focusing parameter γ smoothly adjusts the rate at which easy examples are downweighted. When $\gamma = 0$, FL is equivalent to CE, and as γ is increased the effect of the modulating factor is likewise increased [67].

Chapter4:System Implementation and Design

In this chapter, we present all the experiments we conducted during the working period on the stutter detection and classification project. We have applied more than one model to get the best results through different classification methods.

4.1 Dataset

- UCLASS [62]: Speech samples were collected from the University College Londons Archive of Stuttered Speech. The dataset contains samples of monologues from 139 participants, ranging between 8 and 18 years of age, with known stuttered speech impediments of different severity. Of these recordings, 25 unique participants were used due to the availability of their orthographic transcriptions of the monologues.
- FluencyBank [56] This dataset includes recordings from 32 persons who stutter. We used the dataset annotated by [57] (4 144 clips), who discovered errors in the original annotations' temporal alignments. With the exception of the underrepresented prolongations, this dataset is largely balanced.
- SEP-28K: [57] manually selected and curated this dataset, which includes 28 177 snippets taken from freely accessible podcasts.

4.2 System Design

4.2.1 High Level System Diagram (binary classification):

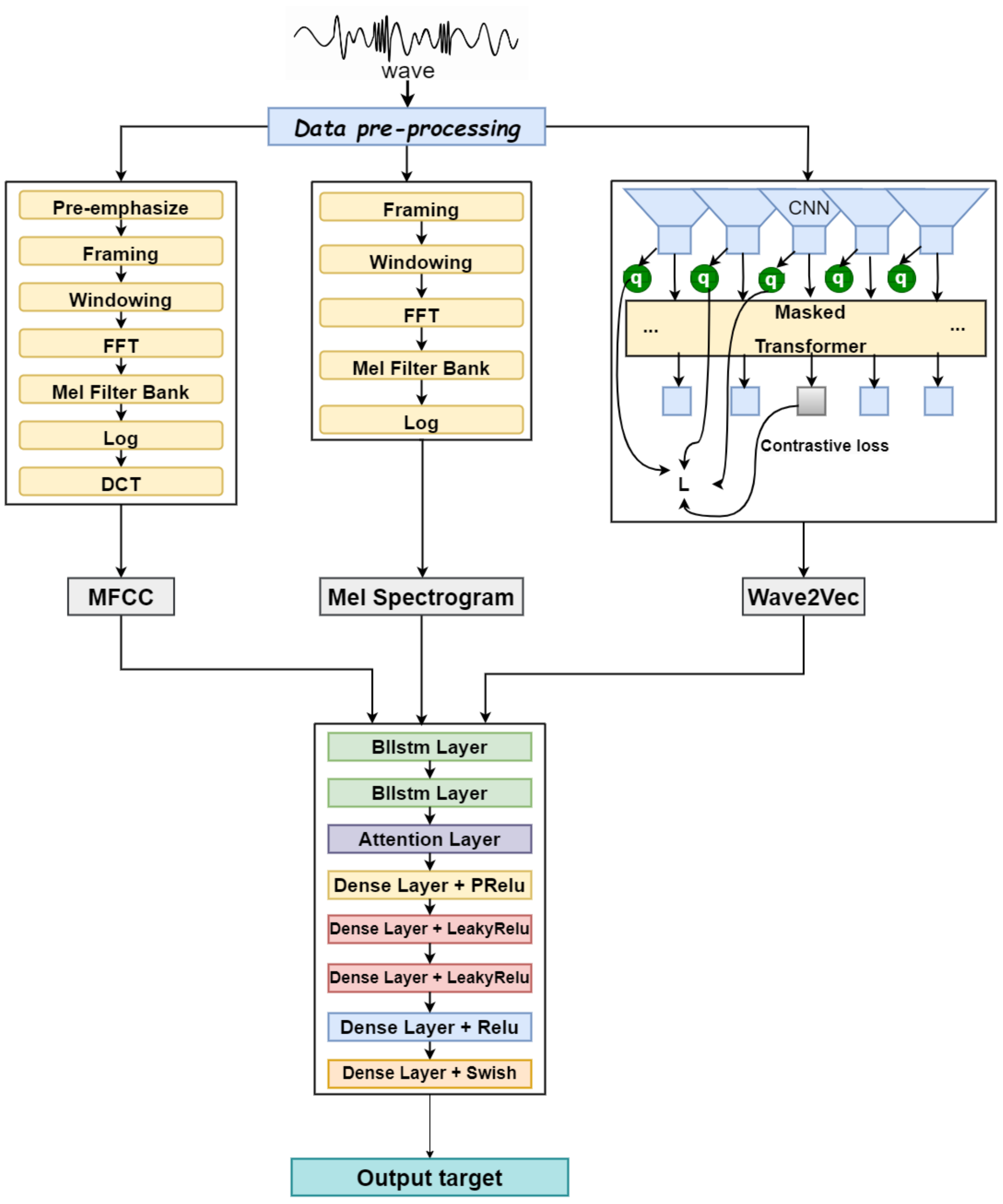


Fig 4.1. High Level System Diagram (binary classification)

4.2.2 Stuttering Detection Diagram (Multilabel classification):

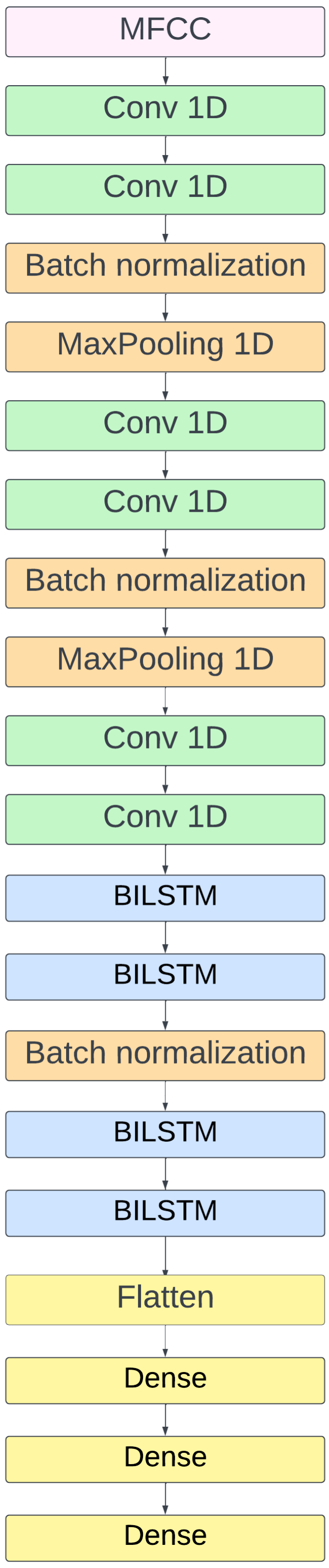


Fig 4.2. Stuttering Detection Diagram (Multilabel classification)

4.3 Data preparing

For UCLASS, data was already pre-processed.

For SEP_28k and Fluencybank datasets, we loaded, combined, and performed some pre-processing on the SEP_28k and fluencybank datasets. We disregarded any records that were empty and deleted any files that included explanatory comments that read "unsure," "DifficultToUnderstand" "PoorAudioQuality," "Music," or "NoSpeech." Then, after extracting features from the records. we combined the extracted features with the labels and entered them into our neural network

4.4 Feature extraction

For UCLASS, input audio clips recorded with a sampling rate of 16 khz are converted to spectrograms using STFT with 256 filters (frequency bins) to be fed to our end-to-end model. Following the common practice in audio-signal processing, a 25 ms frame has been used with an overlap of 10 ms.

For SEP_28k and Fluencybank , Speech Recognition is a supervised learning task. In the speech recognition problem input will be the audio signal and we have to predict the text from the audio signal. We can’t take the raw audio signal as input to our model because there will be a lot of noise in the audio signal. It is observed that extracting features from the audio signal and using it as input to the base model will produce much better performance than directly considering raw audio signal as input.

- **MFCC** extracted with librosa [58]

  MFCC is a widely used technology to extract features from audio signal. So, we used it to extract the 20 features with sample rate = 8000 Hz using librosa.

- **Mel-spectrogram** extracted with librosa

  Mel Spectrograms are commonly used features in speech analysis in different applications ranging from speech recognition to noise cancellation. We use Mel spectrograms to extract 20 features with sample rate = 8000 hz using librosa.

- **Wave2Vec**

The W2V2 model takes the raw wave-form audio as its inputs and yields 768-dimensional speech representations for roughly every 0.02s of audio, yielding 149 vectors for every 3s long clip in the dataset. For our experiments, we used frozen W2V2 models with a classification head equivalent to the implementation from the Transformers library [59].

The classification head consists of a mean-pooling operation, pooling model outputs over the time dimension, yielding a single 768-dimensional vector for every audio clip. The pooling operation is followed by a 256-dimensional dense projection layer and a classification layer. We extract W2V2 features for every three-second-long clip and, similar to the mean-pooling operation in the classification head described previously, take the mean over the time dimension, yielding one 768-dimensional vector for every three-second-long audio clip [60].

- We also attempted to use spectrogram and WMFCC, however, no significant improvements so we will not report these results here.

4.5 Implementation Details.

For UCLASS, FluentNet [10] was implemented using Keras with a Tensorflow backend. The model was trained with a learning rate of 0.0001 yielded the strongest results. A root mean square propagation (RMSProp) optimizer, and a binary cross-entropy loss function were used. All experiments were trained using an Nvidia GeForce GTX 1080 Ti GPU. Python's Librosa library [58] was used for audio importing and manipulation towards creating our synthetic dataset as described later. Each STFT spectrogram was generated using four-second audio clips. This length of time can encapsulate any stutter apparent in the dataset, with no stutters lasting longer than four seconds.

For SEP_28k and Fluencybank, The Keras API of TensorFlow was used to create the model [61]. It was trained using a learning rate of 0.0001 over the course of 45 epochs, with subsequent epochs showing just a little gain in performance. Both the (Adam) optimizer and the sigmoid loss function were applied. The training was carried out using an NVIDIA Quadro P500 (GP108GL) GPU.

4.6 Experimental / Simulations Setup

Now we list each model that has been applied in detail with its advantages, disadvantages and results.

4.6.1 UCLASS Experiments

4.6.1.1 The first experiment: 22 November 2022

We used 25 records from the UCLASS dataset that had data annotations. We pre-processed these records and extracted the spectrogram feature. Then we entered the extracted spectrogram into the SE-ResNet model, Where the model was compiled and about 17 new features were extracted. then we pre-processed it so that it would fit as an input to the BLSTM model, we divided the data into 80% training and 20% testing. After entering the data on the model, we obtained an accuracy of 84.77% for training and 61.07% for testing. We printed the confusion matrix and the Classification Report.

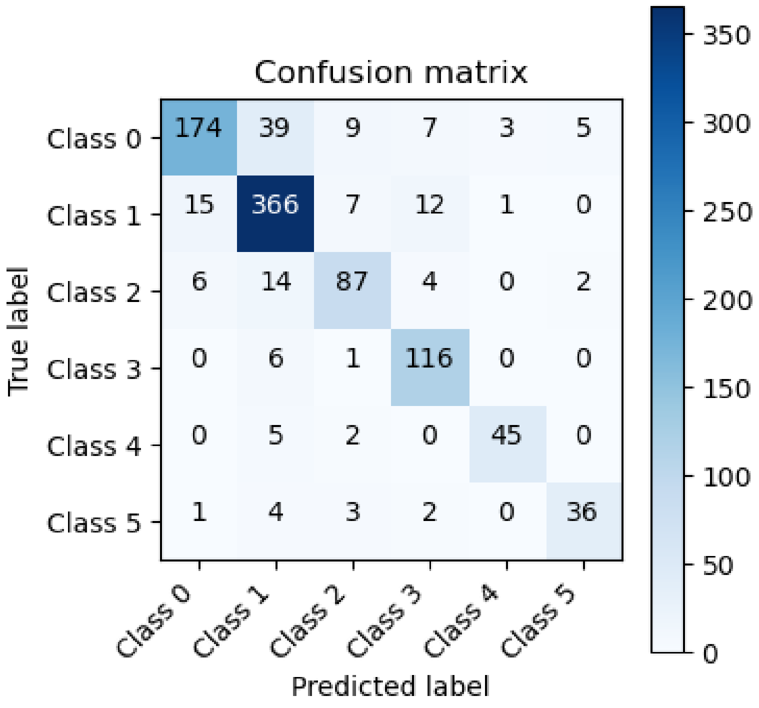


Fig 4.3. Confusion matrix (first experiment)

Classification Report

```
           precision    recall    f1-score   support


class 0   0.887755   0.734177   0.803695       237
class 1   0.843318   0.912718   0.876647       401
class 2   0.798165   0.769912   0.783784       113
class 3   0.822695   0.943089   0.878788       123
```

```
     class 4   0.918367   0.865385   0.891089        52
     class 5   0.837209   0.782609   0.808989        46
     accuracy                        0.847737       972
  macro avg   0.851252   0.834648   0.840499       972
weighted avg   0.850020   0.847737   0.845905       972
```

Table 4.1. accuracy (Acc) of the six stutter types trained on the UCLASS dataset

Label	Class name	Accuracy
0	Interjection	91.25%
1	sound repetition	89.4%
2	word repetition	95.06%
3	phrase repetition	96.7%
4	Revision	98.86%
5	Prolongation	98.25%

Finally, we took the output is from BLSTM model, and we did some pre-processing on it and used it as input to the attention model, but a low precision of 0.1586 was obtained.

4.6.1.2 The second experiment: 6 December 2022

In the first experiment we used 25 records from the UCLASS dataset that contained data annotati ons. but this quantity was not enough to obtain a satisfactory result, so in this experiment we produce d about 5 other records manually and prepared their data annotations, then this time we pre-processed about 30 records and extracted the spectrogram feature. Then we inserted the extracted spectrogram i nto the SE-ResNet model The model was compiled and about 17 new features were extracted. We the n pre-processed it to fit the BLSTM model, dividing the data into 80% training and 20% testing. After entering the data on the model, we obtained an accuracy of 82.79% for training and 59.71% for testin g. We printed the confusion matrix and classification report.

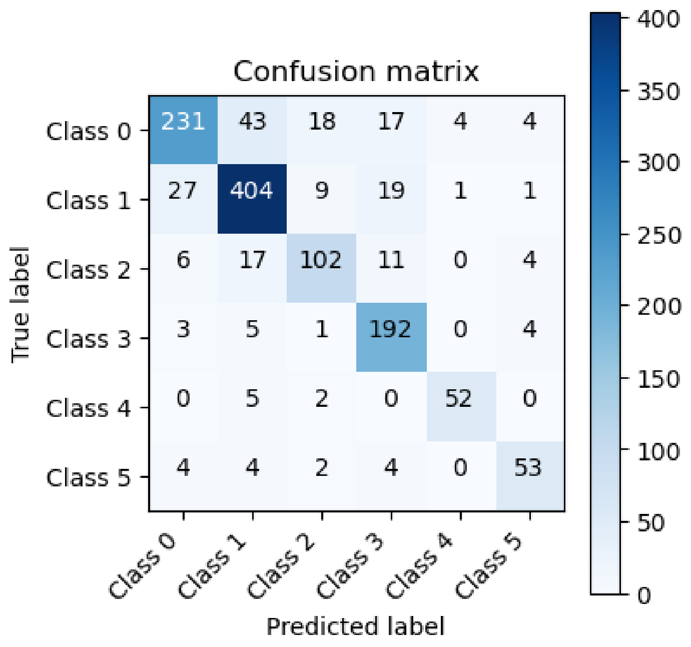


Fig 4.4. Confusion matrix (second experiment)

Classification Report

```
              precision    recall   f1-score   support

    class 0    0.852399   0.728707   0.785714       317
    class 1    0.845188   0.876356   0.860490       461
    class 2    0.761194   0.728571   0.744526       140
    class 3    0.790123   0.936585   0.857143       205
    class 4    0.912281   0.881356   0.896552        59
    class 5    0.803030   0.791045   0.796992        67

   accuracy                          0.827862      1249
  macro avg    0.827369   0.823770   0.823569      1249
weighted avg    0.829473   0.827862   0.826261      1249
```

Table 4.2. accuracy (Acc) of the six stutter types trained on the UCLASS dataset

Label	Class name	Accuracy
0	Interjection	89.9%

1	sound repetition	89.5%
2	word repetition	94.4%
3	phrase repetition	94.9%
4	Revision	99%
5	Prolongation	97.8%

Finally, we took the output is from BLSTM model, and we did some pre-processing on it and used it as input to the attention model, but a slightly higher accuracy was obtained from the first e xperiment, but it is still lower than desired, with a ratio of 0.1790.

Problems:

Overfitting occurs due to a lack of trained data. The model could not reach the required accuracy.

Since the dataset (UCLASS) has lots of problems (e.g., Data Imbalance, Data Annotation Issue an d insufficient number of labelled samples), we decided to work on different dataset(s).

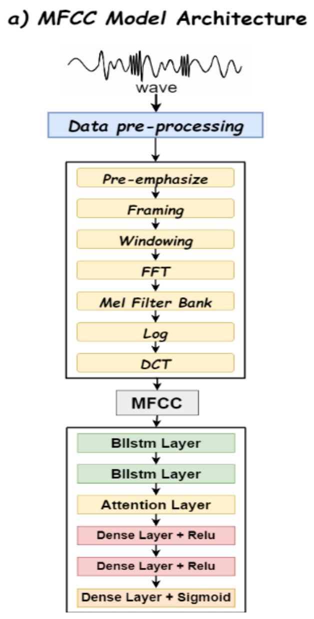
We applied binary classification, multilabel and multiclass to the SEP_28k+fluencybank dataset a nd also used a paper entitled (Classification of stuttering speech imbalance using deep learning for rea l-time applications) As a reference including models and layers knowing that the paper uses multilabe l but we used it in the binary classification also with some changes in the model to fit the binary classi fication and these experiments were conducted on mfcc and we also applied them to other features ext racted such as mel-spectrogram wave2vec.

Our technique takes speech data from 3-second audio samples, applies a temporal model, and pro duces a dysfluency label and a stuttering detection score for each clip.

4.6.2 SEP_28k + Fluencybank Experiments

4.6.2.1 Binary classification

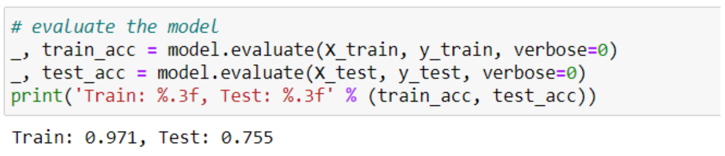
4.6.2.1.1 Model (a)



We applied the MFCC, Mel-Spectogram and Wav2Vec features we had extracted to our model. In our experiment, networks were trained on 80% of the data and tested on the remaining 20%. The models were trained with a batch size of 64 using Adam Optimizer with learning rates of 1e-3.

MFCC:

The accuracy of these experiments was 97% train and 75% test, which was quite excellent (see figure a).



```
# evaluate the model
_, train_acc = model.evaluate(X_train, y_train, verbose=0)
_, test_acc = model.evaluate(X_test, y_test, verbose=0)
print('Train: %.3f, Test: %.3f' % (train_acc, test_acc))
```

```
Train: 0.971, Test: 0.755
```

Fig 4.5. Result of model (a)

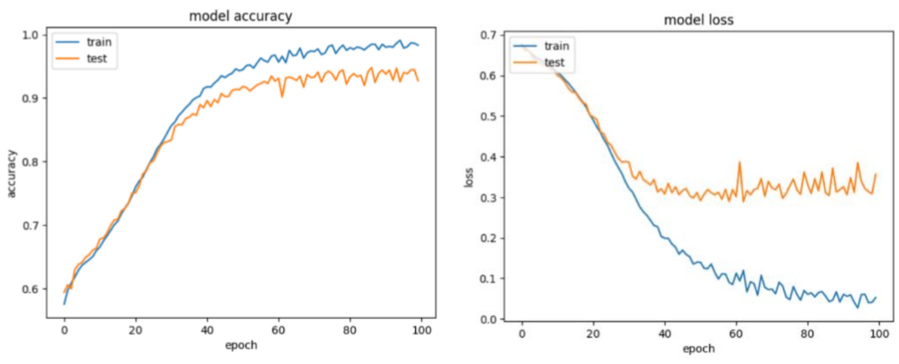
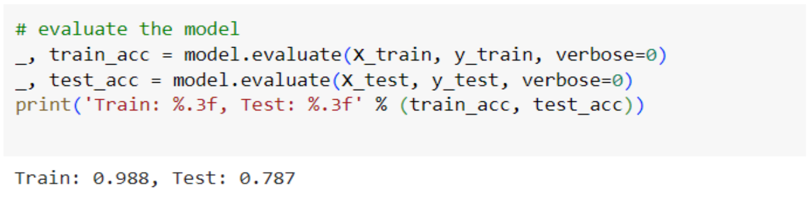


Fig 4.6. Model (a) loss and accuracy curve

Mel Spectrogram Result:

The accuracy of these experiments was 98.8% train and 78.7% test.



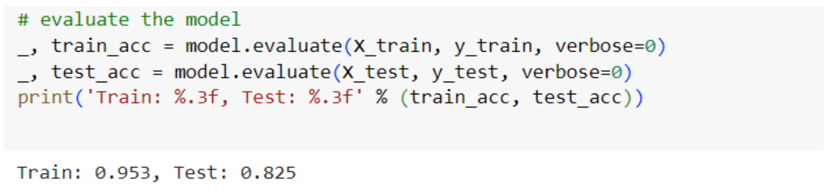
```
# evaluate the model
_, train_acc = model.evaluate(X_train, y_train, verbose=0)
_, test_acc = model.evaluate(X_test, y_test, verbose=0)
print('Train: %.3f, Test: %.3f' % (train_acc, test_acc))
```

```
Train: 0.988, Test: 0.787
```

Wav2Vec:

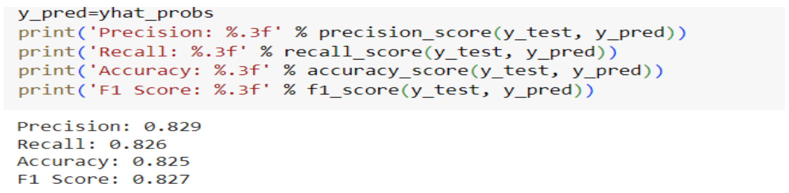
The accuracy of these experiments was 95% train and 82% test, which was quite excellent.

a)



```
# evaluate the model
_, train_acc = model.evaluate(X_train, y_train, verbose=0)
_, test_acc = model.evaluate(X_test, y_test, verbose=0)
print('Train: %.3f, Test: %.3f' % (train_acc, test_acc))

Train: 0.953, Test: 0.825
```

b)



```
y_pred=yhat_probs
print('Precision: %.3f' % precision_score(y_test, y_pred))
print('Recall: %.3f' % recall_score(y_test, y_pred))
print('Accuracy: %.3f' % accuracy_score(y_test, y_pred))
print('F1 Score: %.3f' % f1_score(y_test, y_pred))

Precision: 0.829
Recall: 0.826
Accuracy: 0.825
F1 Score: 0.827
```

Fig 4.7: a) Accuracy of model (a). b) precision, recall, acc and F1-score of W2V of model (a)

Multiclass Classification:

MFCC Result:

Table 4.3 Accuracy and F1-Score for each stutter type of model (a).

Classes Name	***Prolongation***	***SoundRep***	***WordRep***	***Interjection***	***Block***	***Repetition***
Test accuracy	0.710	0.795	0.77	0.677	0.64	0.77

Mel Spectrogram Result:

Table 4.4 Accuracy and F1-Score for each stutter type of model (a).

Classes Name	***Prolongation***	***SoundRep***	***WordRep***	***Interjection***	***Block***	***Repetition***
Test accuracy	0.706	0.794	0.802	0.652	0.610	0.772

Wav2Vec Result:

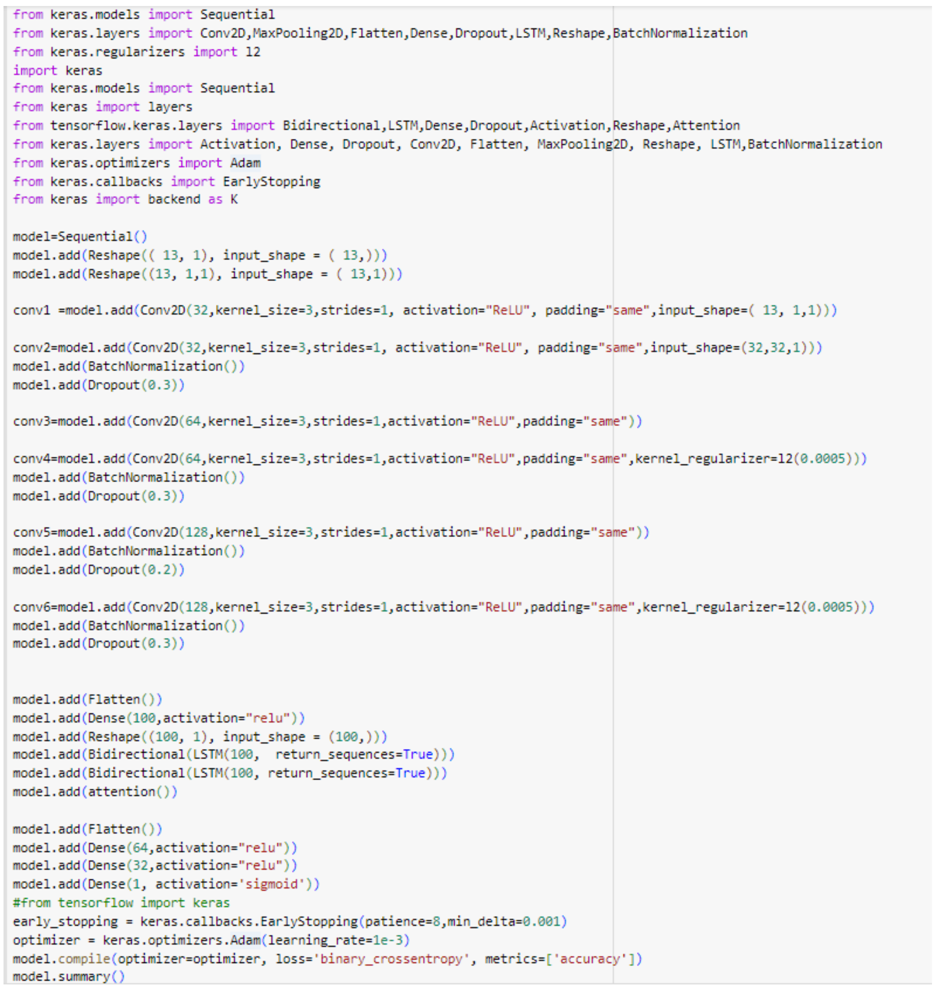
Table 4.5 Accuracy and F1-Score for each stutter type of model (a).

Classes Name	*Prolongation*	*SoundRep*	*WordRep*	*Interjection*	*Block*	*Repetition*
Train accuracy	0.992	0.983	0.961	0.828	0.932	0.929
Test accuracy	0.746	0.852	0.817	0.741	0.613	0.744
F1-Score	0.752	0.862	0.815	0.741	0.619	0.751

Table 4.6. Accuracy for combined features for stuttering detection of model (a).

Feature Name	*MFCC + Mel Spectrogram*	*MFCC + Mel Spectrogram + Wav2Vec*	*Mel Spectrogram + Wav2Vec*	*MFCC + Wav2Vec*
Train accuracy	0.99	0.989	0.992	0.995
Test accuracy	0.81	0.837	0.843	0.831

However, because this high accuracy caused overfitting, we tried to handle the overfitting problem by adding dropout layers, batch normalization, and Regularization. The accuracy of these experiment was 69.1% train and 65.6% test, which was quite better.



```python
from keras.models import Sequential
from keras.layers import Conv2D,MaxPooling2D,Flatten,Dense,Dropout,LSTM,Reshape,BatchNormalization
from keras.regularizers import l2
import keras
from keras.models import Sequential
from keras import layers
from tensorflow.keras.layers import Bidirectional,LSTM,Dense,Dropout,Activation,Reshape,Attention
from keras.layers import Activation, Dense, Dropout, Conv2D, Flatten, MaxPooling2D, Reshape, LSTM,BatchNormalization
from keras.optimizers import Adam
from keras.callbacks import EarlyStopping
from keras import backend as K

model=Sequential()
model.add(Reshape(( 13, 1), input_shape = ( 13,)))
model.add(Reshape((13, 1,1), input_shape = ( 13,1)))

conv1 =model.add(Conv2D(32,kernel_size=3,strides=1, activation="ReLU", padding="same",input_shape=( 13, 1,1)))

conv2=model.add(Conv2D(32,kernel_size=3,strides=1, activation="ReLU", padding="same",input_shape=(32,32,1)))
model.add(BatchNormalization())
model.add(Dropout(0.3))

conv3=model.add(Conv2D(64,kernel_size=3,strides=1,activation="ReLU",padding="same"))

conv4=model.add(Conv2D(64,kernel_size=3,strides=1,activation="ReLU",padding="same",kernel_regularizer=l2(0.0005)))
model.add(BatchNormalization())
model.add(Dropout(0.3))

conv5=model.add(Conv2D(128,kernel_size=3,strides=1,activation="ReLU",padding="same"))
model.add(BatchNormalization())
model.add(Dropout(0.2))

conv6=model.add(Conv2D(128,kernel_size=3,strides=1,activation="ReLU",padding="same",kernel_regularizer=l2(0.0005)))
model.add(BatchNormalization())
model.add(Dropout(0.3))


model.add(Flatten())
model.add(Dense(100,activation="relu"))
model.add(Reshape((100, 1), input_shape = (100,)))
model.add(Bidirectional(LSTM(100,  return_sequences=True)))
model.add(Bidirectional(LSTM(100, return_sequences=True)))
model.add(attention())

model.add(Flatten())
model.add(Dense(64,activation="relu"))
model.add(Dense(32,activation="relu"))
model.add(Dense(1, activation='sigmoid'))
#from tensorflow import keras
early_stopping = keras.callbacks.EarlyStopping(patience=8,min_delta=0.001)
optimizer = keras.optimizers.Adam(learning_rate=1e-3)
model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy'])
model.summary()
```

Fig 4.8. Model (a.2) implementation

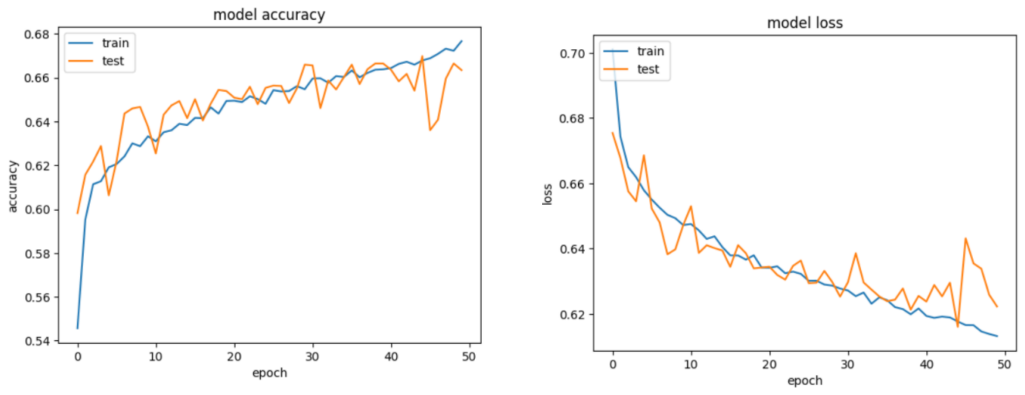
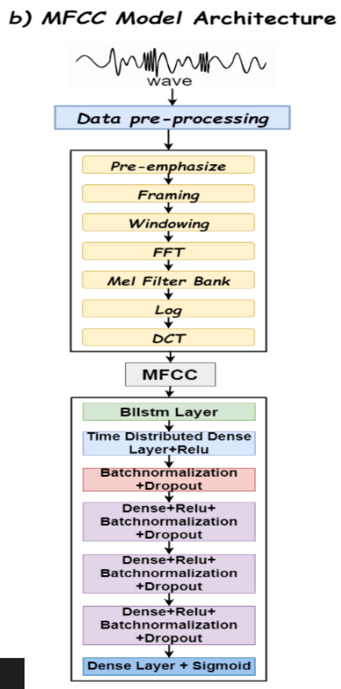


Fig 4.9. Model (a.2) loss and accuracy curve

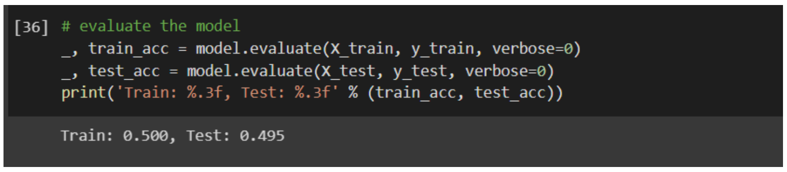
we chose to change this model in order to increase the accuracy and overcome the problem of overfitting.

4.6.2.1.2 Model (b)

In this experiment, we applied the MFCC feature we extracted to our model (see picture 2).


b) MFCC Model Architecture
wave
Data pre-processing
Pre-emphasize
Framing
Windowing
FFT
Mel Filter Bank
Log
DCT
MFCC
Bilstm Layer
Time Distributed Dense Layer+Relu
Batchnormalization +Dropout
Dense+Relu+ Batchnormalization +Dropout
Dense+Relu+ Batchnormalization +Dropout
Dense+Relu+ Batchnormalization +Dropout
Dense Layer + Sigmoid

In our experiment, we configured a similar model to the one in the paper called "Classification of stuttering speech imbalance using deep learning for real-time applications," and the networks were trained on 90% of the data and tested on the remaining 10%. We trained the 32-batch model using Adam Optimizer at 0.0001 learning rates, as mentioned in the paper. The accuracy of this experiment was very poor and ranged from 50% in the train test to 49% (see figure b).



```
[36] # evaluate the model
     _, train_acc = model.evaluate(X_train, y_train, verbose=0)
     _, test_acc = model.evaluate(X_test, y_test, verbose=0)
     print('Train: %.3f, Test: %.3f' % (train_acc, test_acc))

     Train: 0.500, Test: 0.495
```

Fig 4.10. Result of model (b)

Since these results are not usable because they may increase the rate of false expectations, we have decided to replace the model used in this paper because it does not fit our classification type even after adjusting to fit the binary classification. So, we configured and tested the C model.

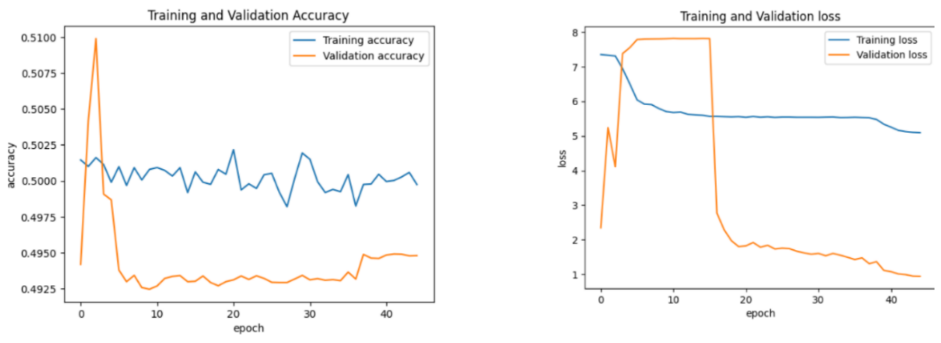
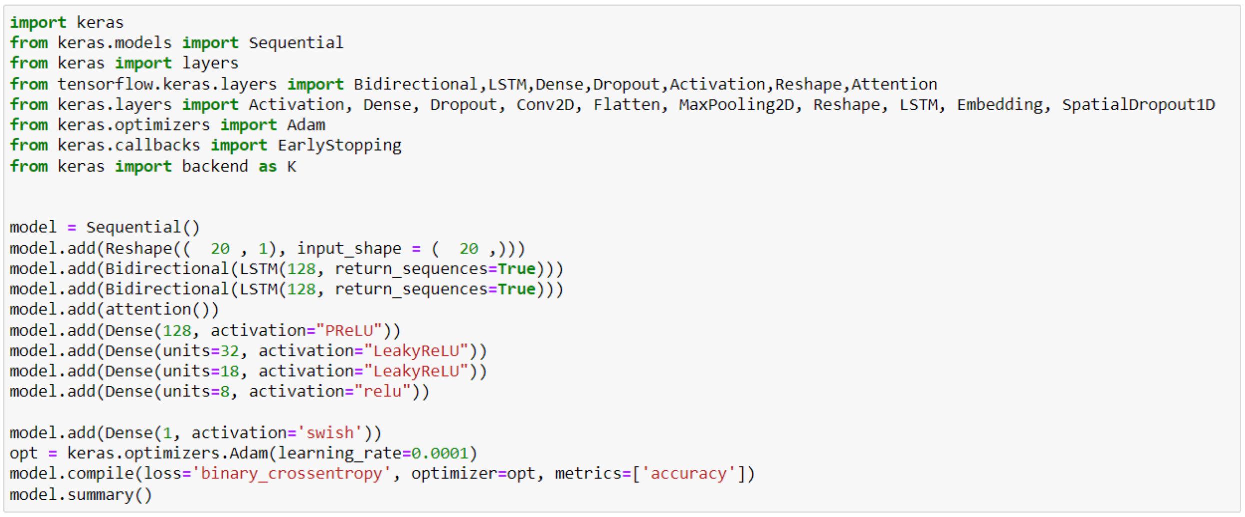


Fig 4.11. Model (b) loss and accuracy curve

4.6.2.1.3 Model (c)

It is the latest experience we have applied to all features extracted from MFCC, Mel Spectrogram, and Wave2Vec.

In this experiment, networks were trained on 80% of the data and tested on the



```python
import keras
from keras.models import Sequential
from keras import layers
from tensorflow.keras.layers import Bidirectional,LSTM,Dense,Dropout,Activation,Reshape,Attention
from keras.layers import Activation, Dense, Dropout, Conv2D, Flatten, MaxPooling2D, Reshape, LSTM, Embedding, SpatialDropout1D
from keras.optimizers import Adam
from keras.callbacks import EarlyStopping
from keras import backend as K


model = Sequential()
model.add(Reshape((  20 , 1), input_shape = (  20 ,)))
model.add(Bidirectional(LSTM(128, return_sequences=True)))
model.add(Bidirectional(LSTM(128, return_sequences=True)))
model.add(attention())
model.add(Dense(128, activation="PReLU"))
model.add(Dense(units=32, activation="LeakyReLU"))
model.add(Dense(units=18, activation="LeakyReLU"))
model.add(Dense(units=8, activation="relu"))

model.add(Dense(1, activation='swish'))
opt = keras.optimizers.Adam(learning_rate=0.0001)
model.compile(loss='binary_crossentropy', optimizer=opt, metrics=['accuracy'])
model.summary()
```

remaining 20%. The 64-batch model was trained using Adam Optimizer at learning rates of 0.0001. (See Figure 4.1)

Fig 4.12. Model (c) implementation

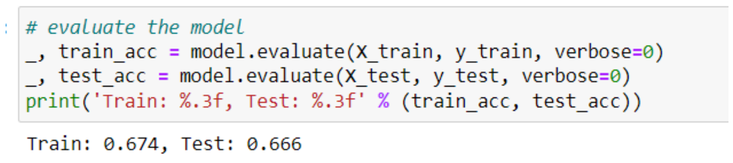
This experiment successfully raised the model's accuracy by 20% relative to Model B. The overfitting problem mentioned in Model A was also overcome.

Results

In this part, we explain the test results for each feature that was extracted and used as a model input. These are the results from each experiment.

MFCC Result:

The results are displayed after we inserted the MFCC (See Figure)



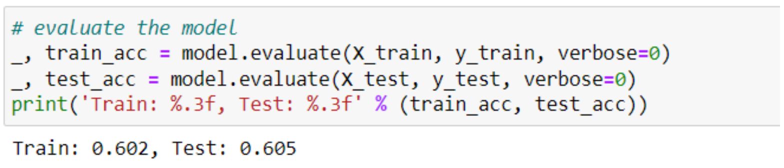
```python
# evaluate the model
_, train_acc = model.evaluate(X_train, y_train, verbose=0)
_, test_acc = model.evaluate(X_test, y_test, verbose=0)
print('Train: %.3f, Test: %.3f' % (train_acc, test_acc))
```

```
Train: 0.674, Test: 0.666
```

Fig 4.13. MFCC feature result at model (c)

Mel Spectrogram Result:

These were the results after inputting the Mel Spectrogram on the model (See Figure)



```
# evaluate the model
_, train_acc = model.evaluate(X_train, y_train, verbose=0)
_, test_acc = model.evaluate(X_test, y_test, verbose=0)
print('Train: %.3f, Test: %.3f' % (train_acc, test_acc))
```

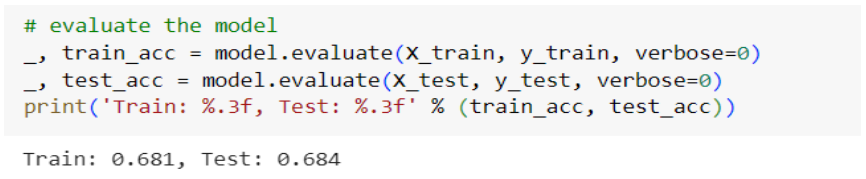
```
Train: 0.602, Test: 0.605
```

Fig 4.14. Mel Spectrogram feature result at model (c)

Wav2Vec Result:

The results are displayed after we inserted the Wav2Vec (See Figure)

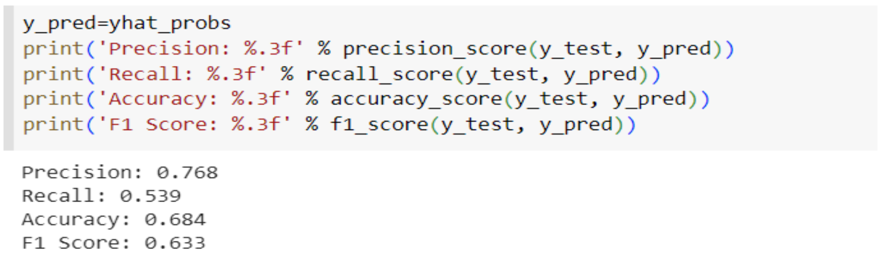
a)



```
# evaluate the model
_, train_acc = model.evaluate(X_train, y_train, verbose=0)
_, test_acc = model.evaluate(X_test, y_test, verbose=0)
print('Train: %.3f, Test: %.3f' % (train_acc, test_acc))
```

```
Train: 0.681, Test: 0.684
```

b)



```
y_pred=yhat_probs
print('Precision: %.3f' % precision_score(y_test, y_pred))
print('Recall: %.3f' % recall_score(y_test, y_pred))
print('Accuracy: %.3f' % accuracy_score(y_test, y_pred))
print('F1 Score: %.3f' % f1_score(y_test, y_pred))
```

```
Precision: 0.768
Recall: 0.539
Accuracy: 0.684
F1 Score: 0.633
```

Fig 4.15: a) Wav2Vec feature accuracy results at model (c). b) Wav2Vec feature precision, recall, acc and f1-score results at model (c)

In an ongoing attempt to improve the accuracy of the model, we also conducted several experiments to integrate interchangeably extracted features, and the results were as follows:

Table 4.7. Accuracy for combined features for stuttering detection of model (c).

Feature Name	*MFCC + Mel Spectrogram*	*MFCC + Mel Spectrogram + Wav 2Vec*	*Mel Spectrogram + Wav 2Vec*	*MFCC + Wav 2Vec*
Train accuracy	0.661	0.757	0.661	0.744
Test accuracy	0.646	0.732	0.659	0.736

4.6.2.2 Multiclass Classification

In this type of classification, we did two experiments.

The first experiment, is that we introduced all the features extracted on Model C and applied them to each class.

MFCC Results:

Table 4.8. Accuracy of MFCC feature for each stutter type of model (c).

Classes Name	*Prolongation*	*SoundRep*	*WordRep*	*Interjection*	*Block*
Train accuracy	0.580	0.661	0.707	0.605	0.571
Test accuracy	0.571	0.682	0.701	0.591	0.562

Mel-Spectogram Results

Table 4.9. Accuracy of Mel-Spectogram feature for each stutter type of model (c).

Classes Name	*Prolongation*	*SoundRep*	*WordRep*	*Interjection*	*Block*
Train accuracy	0.555	0.570	0.601	0.543	0.533

Test accuracy	0.518	0.578	0.583	0.548	0.530

Wave2Vec Results

Table 4.10. Accuracy and F1-Score of W2V feature for each stutter type of model (c).

Classes Name	*Prolongation*	*SoundRep*	*WordRep*	*Interjection*	*Block*	*Repetition*
Train accuracy	0.714	0.721	0.689	0.713	0.602	0.666
Test accuracy	0.670	0.727	0.697	0.722	0.597	0.658
F1-Score	0.681	0.739	0.662	0.727	0.299	0.700

We compared the results with other research papers.

Table 4.11. F1-Score of W2V feature compared to other papers for each stutter type of model (c).

Models	*Dataset*	*Prolongation*	*SoundRep*	*WordRep*	*Interjection*	*Block*	*Repetition*
Bayerl [68]	***Sep-28k***	0.44	0.42	0.45	0.70	0.33	-
Bayerl [69]	***Sep-28k + FluencyBank***	0.55	0.52	0.55	**0.79**	**0.31**	-
Bayerl [68]	***Sep-28k***	0.51	0.50	0.39	0.70	0.30	-
Model (c)	***Sep-28k + FluencyBank***	**0.681**	**0.739**	**0.662**	0.727	0.299	0.700

In the second experiment, is that we used a set of classifiers and applied them to each class and entered all the features extracted.

MFCC Results:

Table 4.12. Accuracy of MFCC feature for each stutter type using classifiers.

Classifiers Name	***Prolongation***	***SoundRep***	***WordRep***	***Interjection***	***Block***
DecisionTreeClassifier	62.264	87.800	86.224	62.5165	58.1081
MLPClassifier	72.776	91.580	93.197	61.72185	55.4054
RandomForestClassifier	74.9326	92.7835	93.367	68.7417	70.1474

Mel-Spectogram Results

Table 4.13. Accuracy of Mel-Spectogram feature for each stutter type using classifiers.

Classifiers Name	***Prolongation***	***SoundRep***	***WordRep***	***Interjection***	***Block***
DecisionTreeClassifier	62.129380	86.769759	87.074829	61.589403	58.108108
MLPClassifier	72.371967	92.783505	91.156462	66.357615	66.093366
RandomForestClassifier	75.067385	93.127147	93.367346	68.476821	69.410319

Wave2Vec Results

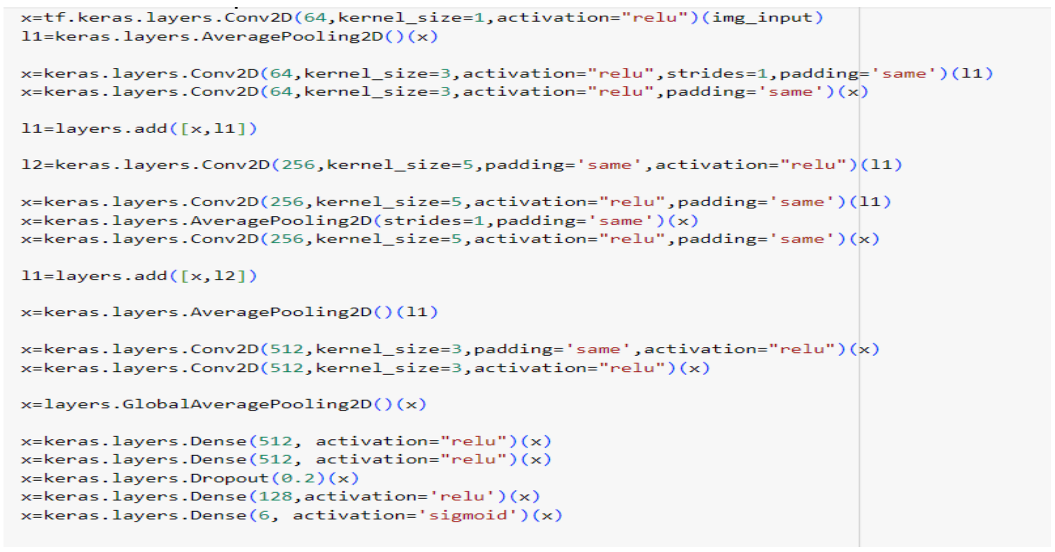
Table 4.14. Accuracy of Wave2Vec feature for each stutter type using classifiers.

Classifiers Name	***Prolongation***	***SoundRep***	***WordRep***	***Interjection***	***Block***
DecisionTreeClassifier	61.4555256	91.065292	84.863945	68.741721	58.230958
MLPClassifier	75.6064690	91.580756	92.857142	80.529801	63.636363
RandomForestClassifier	75.6064690	93.127147	93.367346	80.529801	68.918918

5.5.2.3 Multilabel Model.

We applied the MFCC we had extracted to our model (See Figure 4.2).

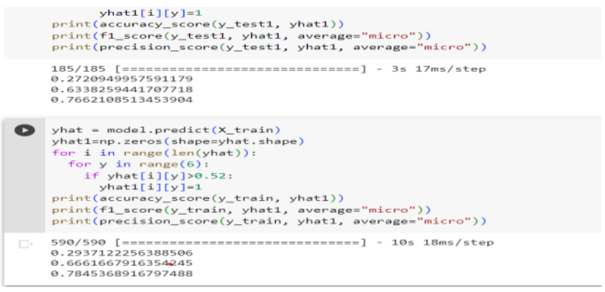
5.5.2.3.1 Model (A):



```
x=tf.keras.layers.Conv2D(64,kernel_size=1,activation="relu")(img_input)
l1=keras.layers.AveragePooling2D()(x)

x=keras.layers.Conv2D(64,kernel_size=3,activation="relu",strides=1,padding='same')(l1)
x=keras.layers.Conv2D(64,kernel_size=3,activation="relu",padding='same')(x)

l1=layers.add([x,l1])

l2=keras.layers.Conv2D(256,kernel_size=5,padding='same',activation="relu")(l1)

x=keras.layers.Conv2D(256,kernel_size=5,activation="relu",padding='same')(l1)
x=keras.layers.AveragePooling2D(strides=1,padding='same')(x)
x=keras.layers.Conv2D(256,kernel_size=5,activation="relu",padding='same')(x)

l1=layers.add([x,l2])

x=keras.layers.AveragePooling2D()(l1)

x=keras.layers.Conv2D(512,kernel_size=3,padding='same',activation="relu")(x)
x=keras.layers.Conv2D(512,kernel_size=3,activation="relu")(x)

x=layers.GlobalAveragePooling2D()(x)

x=keras.layers.Dense(512, activation="relu")(x)
x=keras.layers.Dense(512, activation="relu")(x)
x=keras.layers.Dropout(0.2)(x)
x=keras.layers.Dense(128,activation='relu')(x)
x=keras.layers.Dense(6, activation='sigmoid')(x)
```

Fig 4.16. code of model A

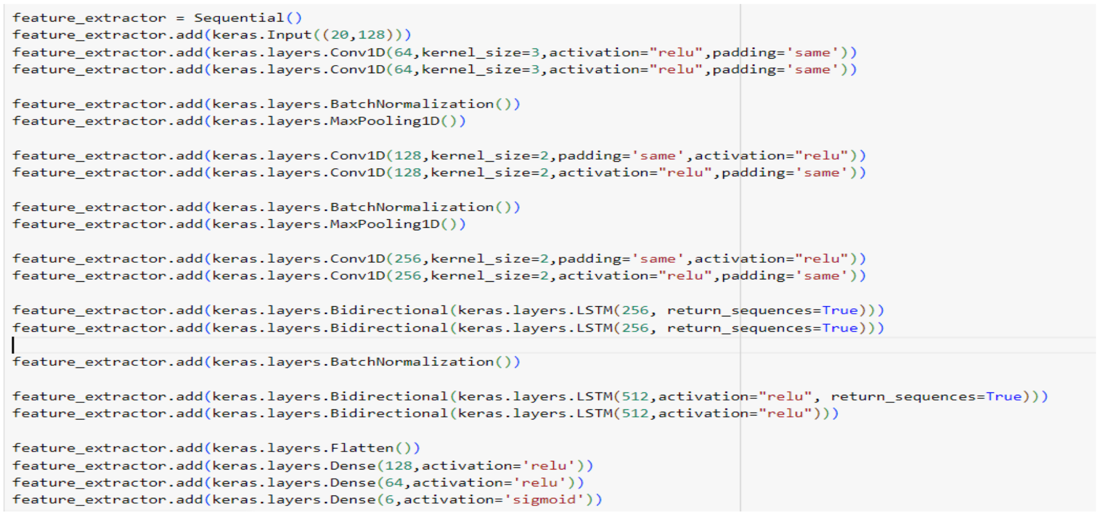
In our experiment, networks were trained on 18862 audios and tested on 4716 audios. The model was trained with Adamax Optimizer and binary focal crossentropy loss with learning rate of 0.00005. The accuracy of this experiment was train 29% and 27% test.



```
        yhat1[i][y]=1
print(accuracy_score(y_test1, yhat1))
print(f1_score(y_test1, yhat1, average="micro"))
print(precision_score(y_test1, yhat1, average="micro"))

185/185 [==============================] - 3s 17ms/step
0.2720949957591179
0.6338259441707718
0.7662108513453904

yhat = model.predict(X_train)
yhat1=np.zeros(shape=yhat.shape)
for i in range(len(yhat)):
  for y in range(6):
    if yhat[i][y]>0.52:
      yhat1[i][y]=1
print(accuracy_score(y_train, yhat1))
print(f1_score(y_train, yhat1, average="micro"))
print(precision_score(y_train, yhat1, average="micro"))

590/590 [==============================] - 10s 18ms/step
0.2937122256388506
0.6661667916354245
0.7845368916797488
```

Fig 4.17. accuracy of test and train for model A

5.5.2.3.2 Model (B):



```
feature_extractor = Sequential()
feature_extractor.add(keras.Input((20,128)))
feature_extractor.add(keras.layers.Conv1D(64,kernel_size=3,activation="relu",padding='same'))
feature_extractor.add(keras.layers.Conv1D(64,kernel_size=3,activation="relu",padding='same'))

feature_extractor.add(keras.layers.BatchNormalization())
feature_extractor.add(keras.layers.MaxPooling1D())

feature_extractor.add(keras.layers.Conv1D(128,kernel_size=2,padding='same',activation="relu"))
feature_extractor.add(keras.layers.Conv1D(128,kernel_size=2,activation="relu",padding='same'))

feature_extractor.add(keras.layers.BatchNormalization())
feature_extractor.add(keras.layers.MaxPooling1D())

feature_extractor.add(keras.layers.Conv1D(256,kernel_size=2,padding='same',activation="relu"))
feature_extractor.add(keras.layers.Conv1D(256,kernel_size=2,activation="relu",padding='same'))

feature_extractor.add(keras.layers.Bidirectional(keras.layers.LSTM(256, return_sequences=True)))
feature_extractor.add(keras.layers.Bidirectional(keras.layers.LSTM(256, return_sequences=True)))

feature_extractor.add(keras.layers.BatchNormalization())

feature_extractor.add(keras.layers.Bidirectional(keras.layers.LSTM(512,activation="relu", return_sequences=True)))
feature_extractor.add(keras.layers.Bidirectional(keras.layers.LSTM(512,activation="relu")))

feature_extractor.add(keras.layers.Flatten())
feature_extractor.add(keras.layers.Dense(128,activation='relu'))
feature_extractor.add(keras.layers.Dense(64,activation='relu'))
feature_extractor.add(keras.layers.Dense(6,activation='sigmoid'))
```

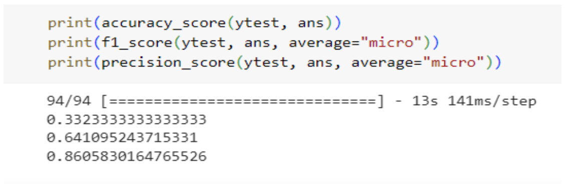
Fig 4.18. code for model B

We divided to divide the dataset into 2 parts:

1. Balanced dataset contained 8161 audios.
2. Dataset contained 5000 audios.

In our experiment, networks were trained 2 times each time on one of the datasets and tested on 3000 audios. The models were trained with a batch size 32 using Adamax Optimizer with learning rates of 0.000009. The accuracy of this experiment was 73% train and 33% test. It considered overfit but that the best we could get.

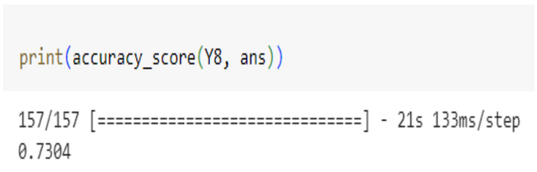
a)



```
print(accuracy_score(ytest, ans))
print(f1_score(ytest, ans, average="micro"))
print(precision_score(ytest, ans, average="micro"))
```

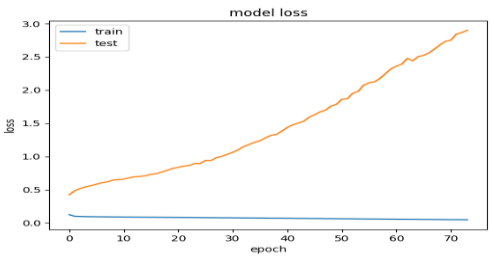
```
94/94 [==============================] - 13s 141ms/step
0.3323333333333333
0.641095243715331
0.8605830164765526
```

b)



```
print(accuracy_score(Y8, ans))
```

```
157/157 [==============================] - 21s 133ms/step
0.7304
```



c)

Fig 4.19: a) accuracy of test b) accuracy of train c) loss curve

Chapter 5: Discussion, Conclusions, and Future Work

5.1. Discussion

As we explained previously each model we have tried, what is the structure of each model and how we tried to optimize our models' parameter and use different features and different datasets to get the best results. Now, we can discuss the performance of each model. Our first Model is BiLSTM on UCLASS dataset, we made two experiments on UCLASS dataset. In the First Experiment, we obtained an accuracy of 84.77% for training and 61.07% for testing, so we were trying to get a better performance by adding records to the dataset as the 25 records from the UCLASS dataset that contained data annotations were not enough to obtain a satisfactory result. So, in the second experiment, we produced about 5 other records manually and prepared their data annotations, The model was compiled and about 17 new features were extracted. We then pre-processed it to fit the BLSTM model, dividing the data into 80% training and 20% testing. After entering the data on the model, we obtained an accuracy of 82.79% for training and 59.71% for testing. Since the dataset (UCLASS) has lots of problems (e.g., Data Imbalance, Data Annotation Issue and insufficient number of labelled samples), we decided to work on different dataset(s). Our second Model (Model (a)) is BiLSTM on the SEP_28k+fluencybank datasets. We applied binary classification, multilabel and multiclass to the datasets. For Binary Classification, the accuracy of these experiments was 97% train and 75% test, which was quite excellent. However, because this high accuracy caused overfitting, we chose to modify this model to a B model in order to overcome the problem of overfitting. For model(b), we configured a similar model to the one in the paper called "Classification of stuttering speech imbalance using deep learning for real-time applications". The accuracy of this experiment was very poor and ranged from 50% in the train test to 49%. Since these results are not usable because they may increase the rate of false expectations, we have decided to replace the model used in this paper because it does not fit our classification type even after adjusting to fit the binary classification. So, we configured and tested the C model. For model(c), The accuracy of these experiments was 67.4% train and 66.6% test, which successfully raised the model's accuracy by 20% relative to Model B. The overfitting problem that was mentioned in Model A was also overcome. we also conducted several experiments to integrate interchangeably extracted features, which successfully raised the model's accuracy For Multiclassification, we did two experiments. The first experiment, is that we introduced all the features extracted on Model C and applied them to each class. In the second experiment, we used a set of classifiers and applied them to each class and entered all the features extracted. Random Forest Classifier has better performance than Decision Tree and MLP.

5.2. Summary & Conclusion

We Present a method for detection and classification of different types of stutter disfluencies. Our model utilizes a bidirectional LSTM units trained using input MFCC, Mel-Spectrogram, Wav2Vec and combination of the three features calculated from labeled audio segments of stuttered speech. Six classes of stutter were considered in this paper: sound repetition, word repetition, repetition (sound repetition + word repetition), block, interjection and prolongation. We also Present a method for multi-class and multilabel learning of different stutter disfluencies. As multiple stutter types may occur at once. Most limitations are due to the small size and the imbalance of the datasets SEP-28k and FluencyBank.

5.3. Future Work

Due to the diversity and uniqueness of stuttering, it continues to be the most demanding and challenging to detect due to its inherent nature of huge class imbalance across different types of disfluencies. Training a model on such a type of dataset will bias the majority class. From here, more work will be put into overcoming our data limitations so that we can more effectively train and perfect classifier models for stuttering speech. Upon developing a better classification tool, we would then focus on optimizing how we apply our classification model to actual speech, the goal being that we can refactor any stuttered speech fed to us into non-stuttered speech. We may also explore data augmentation and contrastive training, as data quantity and diversity play a role in creating more robust dysfluency detection systems. Future work should also explore alternative approaches, e.g., using language models, which may improve performance for some dysfluency types that are more difficult to detect. Lastly, while dysfluencies are most common in those who stutter, future work should address how they can be detected from people with other speech disorders, such as dysarthria, which may be characterized differently.

References (or Bibliography)

[1] Deep Learning Bidirectional LSTM based Detection of Prolongation and Repetition in Stuttered Speech using Weighted MFCC Sakshi Gupta1, Ravi S. Shukla2, Rajesh K. Shukla3, Rajesh Verma4 , 9 November 2020.
[2] S. P. Bayerl, D. Wagner, E. N□oth, K. Riedhammer, Detecting dysuencies in stuttering therapy using wav2vec 2.0, arXiv preprint arXiv:2204.03417.
[3] C. Lea, V. Mitra, A. Joshi, S. Kajarekar, J. P. Bigham, Sep-28k: A dataset for stuttering event detection from podcasts with people who stutter, in: Proc. ICASSP 2021 - 2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2021, pp. 6798{6802. doi:10.1109/ICASSP39728.2021.9413520
[4] S. P. Bayerl, A. W. von Gudenberg, F. H□onig, E. N□oth, K. Riedhammer, KSoF: The Kassel state of uency dataset{A therapy centered dataset of stuttering, arXiv preprint arXiv:2203.05383
[5] Machine Learning for Stuttering Identi_cation: Review,Challenges and Future Directions Shakeel A. Sheikha,_, Md Sahidullaha, Fabrice Hirschb, Slim Ounia 28 Jul 2022.
[6] S. A. Sheikh, et al., Robust stuttering detection via multi-task and adversarial learning, in: Proc. 30th EUSIPCO, 2022.
[7] S. A. Sheikh, M. Sahidullah, F. Hirsch, S. Ouni, Introducing ECAPA-TDNN and Wav2Vec2.0 embeddings to stuttering detection, arXiv preprint arXiv:2204.01564.
[8] A. Baevski et al., "wav2vec 2.0: A framework for self-supervised learning of speech representations," in Advances in NIPS, vol. 33.Curran Associates, Inc., 2020, pp. 12 449–12 460.
[9] T. Kourkounakis, A. Hajavi, A. Etemad, Detecting multiple speech disuencies using a deep residual network with bidirectional long short-term memory, in: Proc. ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), IEEE, 2020, pp.6089{6093.
[10] T. Kourkounakis, A. Hajavi, A. Etemad, Fluentnet: End-to-end detection of speech disuency with deep learning, arXiv preprint arXiv:2009.11394
[11] Introducing ECAPA-TDNN and Wav2Vec2.0 Embeddings to Stuttering Detection Shakeel A. Sheikh1, Md Sahidullah1, Fabrice Hirsch2, Slim Ouni1, 4 Apr 2022.
[12]] S. A. Sheikh et al., "Machine learning for stuttering identification: Review, challenges & future directions," arXiv preprint arXiv:2107.04057,2021.
[13] S. A. Sheikh, et al., Robust stuttering detection via multi-task and adversarial learning, in: Proc.30th EUSIPCO, 2022.
[14] Shakeel A. Sheikh , Md Sahidullah , Fabrice Hirsch , Slim Ouni., "StutterNet: Stuttering Detection Using Time Delay Neural Network".8 Jun 2021.
[15] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv, abs/1810.04805, 2018.
[16] A. Baevski, S. Schneider, and M. Auli. vq-wav2vec: Self-supervised learning of discrete speech representations. In Proc. of ICLR, 2020.
[17]] A. Baevski, M. Auli, and A. Mohamed. Effectiveness of self-supervised pre-training for speech recognition. arXiv, abs/1911.03912, 2019
[18] J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization. arXiv, 2016]
[19] D. Hendrycks and K. Gimpel. Gaussian error linear units (gelus). arXiv, 2016.
[20] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In Proc. of NIPS, 2017.
[21]] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. Roberta: A robustly optimized bert pretraining approach. arXiv preprint arXiv:1907.11692, 2019.
[22] A. Mohamed, D. Okhonko, and L. Zettlemoyer. Transformers with convolutional context for ASR. arXiv, abs/1904.11660, 2019.
[23] F. Wu, A. Fan, A. Baevski, Y. N. Dauphin, and M. Auli. Pay less attention with lightweight and dynamic convolutions. In Proc. of ICLR, 2019.
[24] H. Jegou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. IEEE Trans. Pattern Anal. Mach. Intell., 33(1):117–128, Jan. 2011.
[25]] E. J. Gumbel. Statistical theory of extreme values and some practical applications: a series of lectures, volume 33. US Government Printing Office, 1954.
[26] E. Jang, S. Gu, and B. Poole. Categorical reparameterization with gumbel-softmax. arXiv, abs/1611.01144, 2016.
[27] C. J. Maddison, D. Tarlow, and T. Minka. A* sampling. In Advances in Neural Information Processing Systems, pages 3086–3094, 2014

[28] D. Jiang, X. Lei, W. Li, N. Luo, Y. Hu, W. Zou, and X. Li. Improving transformer-based speech recognition using unsupervised pre-training. arXiv, abs/1910.09932, 2019.
[29]] M. G. A. Hyvärinen. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In Proc. of AISTATS, 2010.
[30]] A. van den Oord, Y. Li, and O. Vinyals. Representation learning with contrastive predictive coding. arXiv, abs/1807.03748, 2018.
[31]] K. He, H. Fan, Y. Wu, S. Xie, and R. Girshick. Momentum contrast for unsupervised visual representation learning. arXiv, abs/1911.05722, 2019.
[32] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton. A simple framework for contrastive learning of visual representations. arXiv, abs/2002.05709, 2020.
[33] S. Dieleman, A. van den Oord, and K. Simonyan. The challenge of realistic music generation: modelling raw audio at scale. arXiv, 2018.
[34] A. Graves, S. Fernández, and F. Gomez. Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks. In Proc. of ICML, 2006.
[35] D. S. Park, W. Chan, Y. Zhang, C.-C. Chiu, B. Zoph, E. D. Cubuk, and Q. V. Le. Specaugment: A simple data augmentation method for automatic speech recognition. In Proc. of Interspeech,2019
[36] L. Rabiner and B. H. Juang, Fundamentals of Speech Recognition, Prentice Hall, 1993.
[37] J. Picone, "Signal modeling techniques in speech recognition", Proc. of the IEEE, vol. 81, no. 9, Sep 1993.
[38] J. Deller, J. Proakis, and J. Hansen, Discrete Time Processing of Speech Signals, Prentice Hall, NJ, USA, 1993.
[39] S. Kopparapu, and M. Laxminarayana, "Choice of Mel filter bank in computing MFCC of a resampled speech", Proc. IEEE Intl. Conf. Information Sciences Signal Processing and their Applications, pp. 121-124, May 2010
[40] G. Bekesy, Experiments in Hearing, Mc-Graw Hill, New York, 1960
[41] H. Hassanein, and M. Rudko, "On the use of Discrete Cosine Transform in cepstral analysis", IEEE Trans. Acoustics, Speech and Signal Processing, vol. 32, no. 4, pp. 922-925, 1984.
[42]] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," CoRR, vol. abs/1709.01507, 2017. [Online]. Available: http://arxiv.org/abs/ 1709.01507.
[43] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning," arXiv preprint arXiv:1602.07261, 2016.
[44] A. G. Roy, N. Navab, and C. Wachinger, "Concurrent spatial and channel squeeze & excitationin fully convolutional networks," in International conference on medical image computing and computer-assisted intervention. Springer, 2018, pp. 421–429.
[45] S. Bell, C. Lawrence Zitnick, K. Bala, and R. Girshick, "Inside-outside net: Detecting objects in context with skip pooling and recurrent neural networks," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2016, pp. 2874–2883
[46]] human pose estimation," in European conference on computer vision. Springer, 2016, pp. 483–499.
[47] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," Neural Comput., vol. 9, no. 8, pp. 1735–1780, 1997
[48] . Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," IEEE Trans. Signal Process., vol. 45, no. 11, pp. 2673–2681,1997.
[49] A. Graves, N. Jaitly, and A. Mohamed, "Hybrid speech recognition with deep bidirectional LSTM Alex Graves , Navdeep Jaitly and Abdelrahman Mohamed University of Toronto Department of Computer Science 6 King ' s College Rd . Toronto , M5S 3G4 , Canada," pp. 273–278, 2013.
[50] Z. Cui, R. Ke, Z. Pu, and Y. Wang, "Stacked bidirectional and unidirectional LSTM recurrent neural network for forecasting networkwide traffic state with missing values," Transp. Res. Part C Emerg. Technol., vol. 118, Sep. 2020.
[51] https://cs231n.github.io/neural-networks-1/
[52] https://www.v7labs.com/blog/neural-networks-activation-functions
[53] https://www.mygreatlearning.com/blog/relu-activation-function/
[54] SWISH: A SELF-GATED ACTIVATION FUNCTION 16 Oct 2017
[55] https://www.educative.io/answers/what-is-the-swish-activation-function
[56] Nan Bernstein Ratner and Brian MacWhinney, "Fluencybank: A new resource for fluency research and practice,"Journal of fluency disorders, vol. 56, pp. 69–80, 2018.
[57] Colin Lea, Vikramjit Mitra, Aparna Joshi, Sachin Kajarekar, and Jeffrey P Bigham, "Sep-28k: A dataset for stuttering event detection from podcasts with people who stutter," in ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, 2021, pp. 6798–6802.
[58] Brian McFee, Alexandros Metsai, Matt McVicar, Stefan Balke, Carl Thom´e, Colin Raffel, Frank Zalkow, Ayoub Malek, Dana, Kyungyun Lee, Oriol Nieto, Dan Ellis, Jack Mason, Eric Battenberg, Scott Seyfarth, Ryuichi Yamamoto, viktorandreevichmorozov, Keunwoo Choi, Josh Moore, Rachel Bittner, Shunsuke Hidaka, Ziyao Wei, nullmightybofo,

Dar´ıo Here˜n´u, Fabian-Robert St¨oter, Pius Friesch, Adam Weiss, Matt Vollrath, Taewoon Kim, and Thassilo, “librosa/librosa: 0.8.1rc2,”May 2021.
[59] Wolf, T., Debut, L., Sanh, V., et al.: Transformers: State-of-the-Art Natural Language Processing. In: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations. pp. 38–45. Association for Computational Linguistics, Online (Oct 2020).
[60] Sebastian P. Bayerl1, "The Influence of Dataset Partitioning on Dysfluency Detection Systems," p. 4, 7 Jun 2022.
[61] Franc¸ois Chollet et al., “Keras,” 2015.
[62] P. Howell, S. Davis, and J. Bartrip, “The university college London archive of stuttered speech (uclass),” Journal of Speech, Language, and Hearing Research, vol. 52, pp. 556–569, 2009.
[63] Diederik P. Kingma. and Jimmy Lei Ba, “Adam: A method for stochastic optimization”, 2015.
[64] Karen Simonyan and Andrew Zisserman, “VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION”
[65] Krizhevsky, A., Sutskever, I., and Hinton, G. E. ImageNet classification with deep convolutional neural networks. In NIPS, pp. 1106–1114, 2012.
[66] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, Zbigniew Wojna “Rethinking the Inception Architecture for Computer Vision” 2015
[67] Tsung-Yi Lin,Priya Goyal, Ross Girshick, Kaiming He,Piotr Dollar “Focal Loss for Dense Object Detection”
[68] Sebastian P. Bayerl, Dominik Wagner, Elmar N¨oth, Tobias Bocklet, and Korbinian Riedhammer “The Influence of Dataset Partitioning on Dysfluency Detection Systems” 7 Jun 2022.
[69] Sebastian P. Bayerl, Dominik Wagner, Elmar N¨oth, Tobias Bocklet, and Korbinian Riedhammer “DYSFLUENCIES SELDOM COME ALONE – DETECTION AS A MULTI-LABEL PROBLEM” 28 Oct 2022.
[70] Sebastian P. Bayerl, Dominik Wagner, Elmar N¨oth, Tobias Bocklet, and Korbinian Riedhammer “Detecting Dysfluencies in Stuttering Therapy Using wav2vec 2.0” 16 Jun 2022.