University of Southern Queensland

School of Engineering

Automated Stuttering Detection to Assist Speech Pathologists

A dissertation submitted by

Lachlan Jackson

in fulfillment of the requirements of

ENP4111 Professional Engineer Research Project

towards the degree of

Bachelor of Engineering Honours (Electrical and Electronic)

Submitted 4th November 2024

ABSTRACT

Stuttering is a complex speech disorder affecting millions of children and adults worldwide. Currently, speech pathologists are the primary health care professionals who provide assessment and treatment to improve the severity of stuttering behaviours. Such behaviours are sporadic throughout the lifespan with no known cure or defined therapeutic agent to date. More specifically, existing techniques for assessing stuttering are manual, inconsistent, and costly for patients. This highlights the necessity for greater innovative solutions to improve the efficacy and efficiency of clinical practice. This dissertation aimed to address such challenges by developing a novel, automated approach to detecting stuttering events.

A rigorous literature review in automated stuttering detection revealed the top performing deep learning architectures. Three of which were developed for this research and evaluated on the SEP-28K dataset. More specifically a Convolutional Neural Network (CNN), Long Short-Term Memory (LSTM), and Convolutional Long Short-Term Memory (Conv-LSTM) models. These architectures were guided by performance metrics for optimisation; a strong focus was placed on class weights to address dataset imbalances. These models were then compared against the innovative pre-trained Audio Spectrogram Transformer (AST) model for the binary classification task; detecting interjection disfluencies.

Findings revealed the hybrid Conv-LSTM outperformed traditional CNN and LSTM models, achieving an 8.72% increase in performance. The AST model made further advancements over the Conv-LSTM model by an additional 4.22%. Subsequently, the AST was then utilised to detect all types of stuttering, achieving an average F1 score of 0.5370 across the five disfluency classes. It is important to note, these results are competitive with

existing literature. However, they are not directly comparable due to the differences in validation protocols used.

The AST model demonstrated capable performance when utilising a shallow MLP network. The visualisation of latent codes from the MLP head further support these observations through the separation and grouping between fluent and disfluent samples. As a result, the pretrained transformer effectively generalised to stuttering specific features. While additional research is necessary to refine the model's efficacy, such results promise a favourable avenue for further research within the stuttering detection field.

University of Southern Queensland

School of Engineering

ENP4111 Dissertation Project

(This is a 2-unit research project in Bachelor of Engineering Honours Program)

Limitations of Use

The Council of the University of Southern Queensland, its Academic Affairs, and the staff of the University of Southern Queensland, do not accept any responsibility for the truth, accuracy or completeness of material contained within or associated with this dissertation.

Persons using all or any part of this material do so at their own risk, and not at the risk of the Council of the University of Southern Queensland, its Faculty of Health, Engineering and Science or the staff of the University of Southern Queensland.

This dissertation reports an educational exercise and has no purpose or validity beyond this exercise. The sole purpose of this dissertation project is to contribute to the overall education within the student's chosen degree program. This document, the associated hardware, software, drawings, and other material set out in the associated appendices should not be used for any other purpose: if they are so used, it is entirely at the risk of the user.

CERTIFICATION

I certify that the ideas, designs and experimental work, results, analyses and conclusions set out in this dissertation are entirely my own effort, except where otherwise indicated and acknowledged.

I further certify that the work is original and has not been previously submitted for assessment in any other course or institution, except where specifically stated.

Student Name: Lachlan Jackson

Student Number: [redacted]

Signature

_____04/11/2024_____

Date

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my supervisor, Professor John Leis, for his support, insightful guidance and encouragement throughout this journey. His expertise and reassurance have been invaluable.

A special thank you to my partner, Carina, whose background as a speech pathologist inspired me to pursue this topic and gave me direction from the very beginning.

TABLE OF CONTENTS

PAGE

LIST OF TABLES

LIST OF FIGURES

GLOSSARY

ANN	=	Artificial Neural Network
AST	=	Audio Spectrogram Transformer
BLSTM	=	Bidirectional Long Short-Term Memory
CBAM	=	Convolutional Block Attention Module
CMVN	=	Cepstral Mean and Variance Normalisation
CNN	=	Convolutional Neural Network
ConvLSTM	=	Convolutional Long Short-Term Memory
DCT	=	Discrete Cosine Transform
DFT	=	Discrete Fourier Transform
DL	=	Deep Learning
EER	=	Equal Error Rate
FFT	=	Fast Fourier Transform
FN	=	False Negatives
FP	=	False Positives
KSoF	=	Kassel State of Fluency

KNN	=	K-Nearest Neighbour
LPCC	=	Linear Predictor Cepstral Coefficient
LSTM	=	Long Short-Term Memory
MFCC	=	Mel Frequency Cepstral Coefficient
MLP	=	Multi-Layer Perceptron
NDIS	=	National Disability Insurance Scheme
%SS	=	Percentage of Stuttered Syllables
ReLU	=	Rectified Linear Unit
RNN	=	Recurrent Neural Network
SD	=	Stuttering Detection
SP	=	Speech Pathologist
STFT	=	Short Time Fourier Transform
SVM	=	Support Vector Machine
TDNN	=	Time Delay Neural Network
TN	=	True Negatives
TP	=	True Positives
T-SNE	=	T-Distributed Stochastic Neighbour Embedding

CHAPTER 1 INTRODUCTION

This chapter introduces the research environment for stuttering detection and its significance in the field of speech pathology. Firstly, underlining the impact stuttering has on a large population of people around the world. The current assessment techniques and costs involved are highlighted, setting the stage for a literature review in automating the detection process and deep learning model development.

1.1 Towards Automation

Artificial intelligence (AI) and deep learning models have revolutionised the world in an ever-growing number of fields by performing complex tasks once reliant on human intelligence. This technology is already making significant waves throughout healthcare industries, yet deep learning (DL) models for Stuttering Detection (SD) are still in their infancy. Automating stuttering therapy is an avenue that could reduce time and costs associated with treatment. This dissertation aims to explore the models capable of delivering these benefits. A literature review will identify the best performing models. The knowledge obtained will help uncover gaps in the field, allowing exploration and design of new models that introduce innovative ideas to advance the field. This dissertation aims to build upon existing work and pave the way for affordable and accessible therapy options.

In the near future, a stuttering recognition system could eliminate the need for manual assessments and enable continuous patient performance tracking, potentially delivering therapy beyond clinical hours. This automation would allow healthcare providers to allocate

resources more effectively, leading to broader service coverage and better overall health outcomes.

1.2 Stuttering Physiology and Psychology

To isolate an exact cause of stuttering is a difficult problem, due to its nuanced and multifaceted aetiology. Smith & Webera (2017) believe stuttering can be defined as a neurodevelopmental disorder that generally manifests in preschool years. The brain's structure dynamically changes during these years from a blend of genetic, environmental, and epigenetic influences. This can lead to varied developmental trajectories across brain regions causing challenges in speech motor planning and execution along with inconsistencies in muscle activation patterns (Smith & Webera, 2017). Stuttering is described as a persisting disturbance in the normal fluency of speech. An example list of the types of stuttering from Table 1 of Sheikh et al. (2022a) work is included in Figure 1 below. These disfluencies can also be accompanied by other motor behaviours such as physical tension, eye blinks, grimacing, changes in pitch, and loudness (Maguire et al., 2020).

The further implication of this disorder is the associated fear in social settings which can exacerbate disfluency. The psychological effects due to lack of effective communication caused by stuttering leads to conditions of anxiety, social phobia, hindered employment opportunities, and overall life quality (Onslow, 2000). For many who have this disorder, these factors are often the most disabling features (Yairi & Ambrose, 2013).

Stutter Type	Definition	Example
Blocks	Involuntary pause before a word	I w **blockage/pause** ant to speak
Prolongations	Prolonged Sounds	**Sssssss**am is kind
Interjection	Insertion of sounds	uh, uhm
Sound Repetition	Phoneme repetition	He **w-w-w**-wants to write
Part-Word Repetition	Repetition of a Syllable	**Go-go-go** back
Word Repetition	Repetition of a Word	**Well, well**, I didn't get you
Phrase Repetition	Repetition of several successive words	**I have, I have** an iphone
Repetition-Prolongation	Repetition and Prolongation	**Ggg**o **b-b-b**back
Multiple	disfluencies occurring at the same time Multiple disfluencies in a word or phrase	**Tt-Tt-Tt**ariq **blockage/pause** is **kkkk**ind
False Start	Revision of a phrase or a word	**I had-** I lost my watch

Figure 1: Various Stutter Types including definitions and examples. Image Source: Sheikh et al., 2022a).

1.3 Prevalence

Notably a higher incidence is seen in younger children, with estimates from 5 – 8% of all children encounter stuttering, of which roughly 80% recover naturally or with therapy (Yairi & Ambrose, 2013). See Figure 2 from Sommer et al. (2021) illustrating the distribution of stuttering by age group in a Germany study, this data is also consistent with international literature figures.

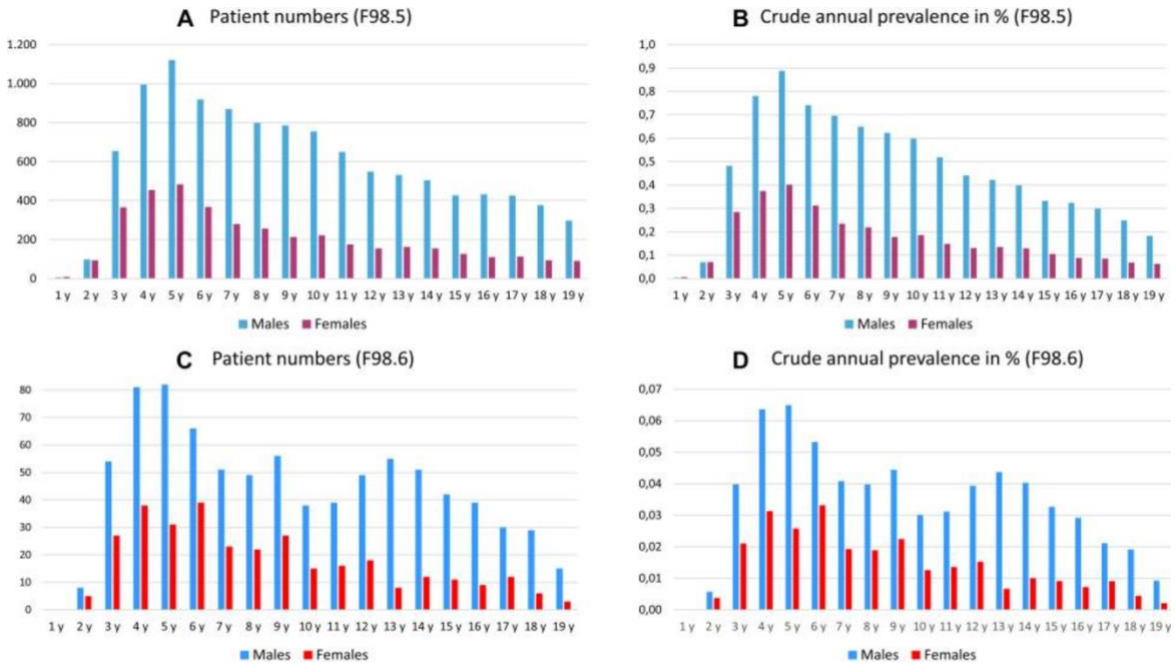


Figure 2: A German study analysed and grouped by age of 27977 insurance patients diagnosed with stuttering (A,B) or cluttering (C,D), only a small fraction was diagnosed with cluttering. Image source: (Sommer, Walterbacher, Schlotmann, & Schröder 2021).

The complex interplay between speech motor control, language development and emotional factors play a crucial role with stuttering persistence. Stuttering is prevalent in the lives of approximately 1% of the general adult population. Late on set of stuttering is rare, and if apparent in teenage years, the chance of recovery drastically reduces, particularly in males (Howell, 2007). While no known cure or defined therapeutic agent exist, the most critical option is speech therapy. Therapy has been shown to significantly decrease severity. If therapy stops however, the reductions in stuttering behaviours are not durable (Blomgren et al., 2005).

1.4 The Problem

The key challenges in the field of speech pathology concerning stuttering are explored below. The variability in stuttering patterns across individuals, languages, and cultures

complicates standardisation; this has led to a reliance on self-reporting and subjective scoring.

1.4.1 Current Stuttering Assessment Techniques

Speech Pathologists (SPs) are trained to treat disfluencies such as stuttering. Despite recent technological advances in speech recognition, SPs still rely on traditional techniques for formal stuttering assessments. Established normative assessment tools such as Test of Childhood Stuttering (TOCS) (Gillam et al. 2009) and the Stuttering Severity Instrument (SSI-4) (Riley 2019) are well regarded and have been created by experts in the field to help diagnose disfluencies. In Australia, SPs will generally simplify these assessments and obtain subjective severity ratings through questionnaires and then implement treatment plans such as the Lidcombe (Onslow et al., 2017) and Camperdown (O'Brian et al., 2018) programs.

Often during assessments, SPs will calculate the percentage of stuttered syllables (%SS). A useful metric to determine the severity of stuttering which supports diagnoses with quantitative data. Finding the %SS requires SPs to listen to a patient read aloud and record each disfluency event based on their own perception. Counting every syllable along the way. This complex and specialised skill is time extensive and prone to inconsistencies in stutter type classifications.

Speech Pathology Australia (2017) acknowledges that stuttering is a communication disorder that necessitates evidence-based treatment following skilled assessment by certified SPs, all of which is expensive.

1.4.2 Speech Therapy Treatment and Costs

Speech pathology can be covered under the National Disability Insurance Scheme (NDIS) with a primary diagnosis relating to developmental delay, e.g. autism. If patients exhibit stuttering associated to this developmental delay, it qualifies for NDIS coverage. The cost of therapy with certified practising SPs is a minimum of $193.99 per hour (National Disability Insurance Agency, 2024). Alternatively, Ryan & Becka (2024) state that private practice costs involved with speech therapy on the Lidcombe program are roughly $240 per hour. Furthermore, Speech Pathology Australia (2024) states speech therapy always incurs an out-of-pocket expense after Medicare rebate. Indicating sessions are a significant financial burden to the stuttering patient.

Onslow et al. (2002) found the median number of clinic visits required on stage 1 of the Lidcombe program was 16, with the reported range from 11-23. And, with the goal of Lidcombe stage 2 being to maintain ‘minimal to no stuttering for a long time’. These follow-up sessions on average consist of eight sessions. The cumulative cost of these sessions reveals the large expense for both NDIS and patients.

1.5 Research Objectives

Deep learning models can improve the accuracy of stuttering classification and detection. By integrating these models into clinical practices, it will streamline the process and empower speech pathologists with more accurate diagnostic tools.

This project can be defined by five key objectives:

1. Establish foundational knowledge through a state-of-the-art review. Explore classical and DL models applicable to stuttering, identify challenges, and highlight areas requiring further investigation.
2. Analyse and develop variations of established DL models with an emphasis on optimising some key hyperparameters. Building custom models and adjusting components will deepen knowledge in the component's role regarding performance.
3. Evaluate the performance of custom models on a dataset focusing on metrics like loss curves, accuracy, and F1 score.
4. Explore novel approaches within the SD field, focusing on underexplored methodologies and investigate their potential.
5. Propose future research suggestions to guide subsequent endeavours.

1.6 Summary

This chapter introduced core background to stuttering and the significant potential DL models possess. The current limitations speech pathologist face regarding assessments and therapy was emphasised. It highlighted the large costs involved to patients and the NDIS. The objectives of the research were outlined, which focused on reviewing the current literature, developing and testing DL models, and investigating novelty as a core objective.

CHAPTER 2 LITERATURE REVIEW

While automated stuttering detection methods have been researched broadly, involuntary pauses and irregular repetitions in stuttered speech make it difficult to implement accurate models compared to fluent speech. This chapter will explore the state-of-the-art advancements in autonomous SD and align findings to the research objectives.

2.1 Available Datasets

Current speech recognition systems such as virtual assistants like Amazon's Alexa or Apple's Siri fail to recognise stuttered speech (Sheikh et al., 2022a). This poses difficulties to people who stutter. A major constraint in this context is the availability of data. Larger datasets improve a model's ability to generalise to new situations as it has increased diversity in samples which reduces overfit risk (Sheikh et al., 2022b). More data when training a model allows for robust speech patterns to be learned. This removes the need for a model to fill gaps with excessive assumptions and in turn build a model that can predict with higher accuracy (Lea, Mitra, Kajarekar, & Smiley, 2021).

Numerous datasets exist in the compass of SD, yet not all are publicly available. FluencyBank, KSoF and VoxCeleb datasets are examples of these, along with various other customised datasets created for specific research purposes. This not only affects accessibility and future collaboration, but the ability to benchmark against other methods. Given these limitations, only public datasets will be explored further.

2.1.1 UCLASS Dataset

The University College London's Archive of Stuttered Speech or UCLASS is a dataset created in 2009 that has been the most adopted in autonomous SD research. Primarily due to the transcribed orthographic versions being available alongside each audio files. This allows for speech to text associations. According to Table 1 in Howell et al. (2009) UCLASS second release contains 318 monologues, conversations, and readings from 160 speakers with a stuttering mean age between 12-13 years old. A predominately male set with 279 samples, versus 39 female samples. The age range and demographic diversity illustrate the limitations in this dataset as model predictions may struggle with mature and diverse populations. It has been used in research such as (Sheikh, Sahidullah, Hirsch, & Ouni, 2021), and (Kourkounakis, Hajavi, and Etemad 2020). But the small set size along with lack of sufficient data annotations is a weakness (Sheikh et al., 2022a).

2.1.2 LibriStutter Dataset

A second publicly available dataset is the LibriStutter set which is a 20-hour long synthetically modified subset from the fluent speech set LibriSpeech. Through Google Cloud Speech-to-Text API (Application Programming Interface), timestamped disfluencies were added at 4 second intervals. Undergoing several processes to simulate natural stuttering behaviour (Kourkounakis, Hajavi, and Etemad 2021). For example, sound repetition was achieved by copying an existing fraction of a random spoken word, repeating it, and then synthesizing. E.g. "She **l-l-l**-likes ice cream". The dataset is more gender balanced with 53% male and 47% female. 15,000 disfluencies are available with each type having 3,000 events (sound, word and phrase repetitions, interjections, and prolongations) (Kourkonakis et al., 2021). This dataset has not been frequently utilized with research endeavours. Possibly because synthesised speech has difficulty capturing all the nuances of naturally occurring

stuttered speech. Thus, affecting generalisability to real-world scenarios. The figure below from (Kourkounakis et al., 2021) shows a visual comgeleparison from real and synthetic spectrograms in the datasets, which do indicate clear similarities.

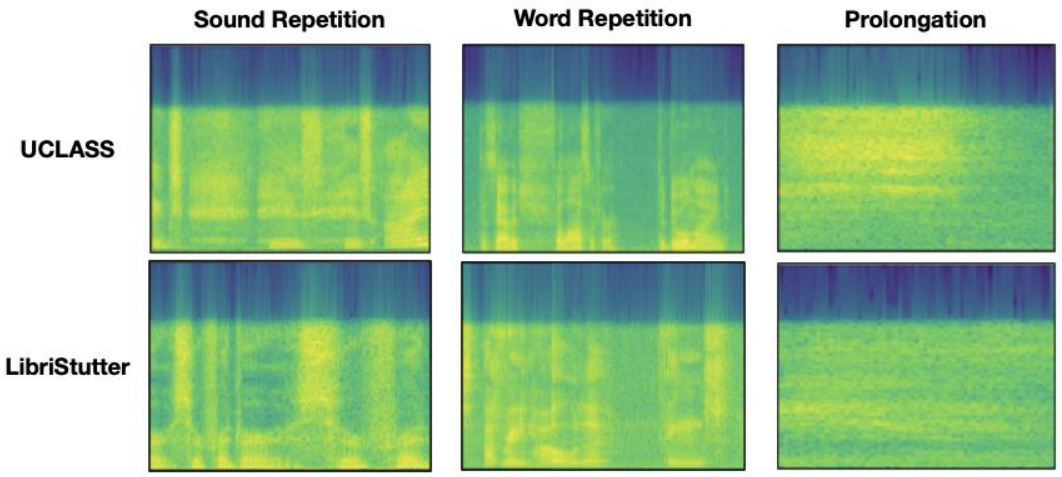


Figure 3: Spectrograms of the same word or sound being spoken, for each stutter type found in UCLASS (natural) dataset and LibriStutter (synthetic) dataset. Image Source: (Kourkounakis et al., 2021).

2.1.3 SEP-28k Dataset

The third publicly available is the SEP-28k dataset. It was released in 2021, with the aim to address the limitations in data and support building generalisable disfluency detection systems that could aid patients and speech pathologists alike (Lea et al., 2021). It contains over 28,000 three second clips from stuttering podcasts which have been annotated and labelled by three trained annotators (not clinicians). Five types of stutters are included from

(i) sound repetitions,

(ii) word repetitions,

(iii) prolongations,

(iv) interjections,

(v) blocks.

It also includes other classifications such as natural pauses, music, no speech, difficult to understand, no stuttered words, unsure, and poor audio quality. The updated version supplies the training and test data breakdown to allow for accurate benchmarking compared to the state-of-the-art SD systems (Lea et al., 2021). The SEP-28k dataset is the largest and latest inclusion to the stuttering domain, and most of the recent detection research incorporates this dataset (Mohapatra et al., 2022), (Ameer et al. 2023), (Al-Banna et al., 2024), (Jouaiti and Dautenha 2022) and (Sheikh et al., 2022b).

In addition, roughly 4,000 clips with stutter event annotations from the FluencyBank dataset were released alongside the SEP-28k dataset (Lea et al., 2021). FluencyBank is a shared database restricted to research, that requires creating an account and agreeing to the terms of use. It has been used extensively in the literature as an evaluation dataset (Bayerl et al., 2022), (Jouaiti and Dautenha 2022) and (Al-Banna et al., 2024).

2.2 Preprocessing Audio Data for Feature Extraction

Once a dataset (or multiple datasets) has been selected, it is essential to thoroughly clean and prepare the raw audio files within. Several methods can be employed to remove redundancies on an audio dataset which can ensure consistency across samples Sheikh et al., 2023). This can enhance the quality of features as models can better capture relevant patterns in a signal.

2.1.1 Reduce Noise in Audio Files

To eliminate unwanted noise from audio files, a traditional technique involves spectral subtraction; estimating the noise spectrum during silent segments of a signal and subtracting this from the original signal can isolate a clean signal. A more modern approach was seen in Ameer et al. (2023) work using the SEP-28k dataset. To enhance stuttered speech

classifications, they used a python library algorithm ‘noisereduce’ to mitigate unwanted noise from the data. This employs spectral gating which is essentially a noise gate that supresses sounds below a frequency threshold and filtering out background disturbances.

2.1.2 Make Audio Files the Same Length

Trimming or padding audio files within a dataset is also a necessity in the pre-processing pipeline. These steps ensure all dataset samples are equal in length; DL models generally require fixed size input lengths. Trimming involves removing the start or end section of an audio file, and padding involves adding 0’s to the start or end of an audio file to reach a desired length.

2.1.3 Check and Modify if Necessary

Further steps to preprocess audio files can be required. It is common practice to pass the audio files through a series of checks to ensure consistency across the dataset. By curating data, researchers ensure their analyses and models are built on quality, error-free data which leads to more accurate results (Lewis, 2024). Such checks may include resampling files to the desired sampling rate if necessary, reducing channel width to mono (single channel), ensuring same data type (floating point number), and removing empty files.

2.1.4 Divide Each Audio File into Segments

Each audio file is often converted to a frequency representation before feature extraction. The frequency domain better represents audio signals compared to the time domain as the human auditory perception is highly dependent on how the ear and brain process frequency components of sound (Moore 2012). By isolating overlapping frequencies in the time domain and transforming them into individual components, complex signals can be analysed.

The Fast Fourier Transform (FFT) extracts frequency components from a signal. However, human speech varies in frequency over time. Applying the FFT across an entire audio signal will capture a signals frequency components but will not show precisely when these changes occur in time. Which is essential for stuttering analysis.

Dividing the signal into short time frames and then applying the FFT yields the assumption that these frequencies are now stationary (Fayek, 2016). This is known as the short-time Fourier Transform (STFT), and it provides a suitable approximation of a signal's frequency contours while maintaining necessary temporal resolution. Often the number of frequency bins in a STFT is equal or slightly larger than the number of frames in the sample. This gives more frequency resolution. Setting the number of bins to a power of two speeds computation due to the nature of DFT calculations, and 512 bins is typically used (Librosa, 2023).

Selecting a suitable frame length is an important step. A good balance between the time and frequency domain is essential. As reducing the frame length improves time resolution but reduces frequency resolution and is more computationally expensive. This segmentation process generally uses a standard framing of 25ms (20-40ms range) (Singh, 2019).

For example, in the case of the SEP-28k dataset, the audio files are sampled at 16kHz and are roughly 3 seconds in length. Therefore, with the standard frame size of 25ms,

$$.025\ seconds * 16000 \frac{samples}{second} = 400\ samples$$

The frame length is 400 samples.

$$3\ seconds * 16000\ \frac{samples}{second} = 48000\ samples$$

Each audio file contains 48000 samples.

$$\frac{48000\ samples}{400\ samples} = 120\ frames$$

Each audio file is broken down into 120 frames.

2.1.5 Hop Length

Another important closely linked parameter to frame size is hop length. This stipulates the sample distance between the start points of two consecutive frames. Overlap between frames is important to ensure no critical information is lost. Common practice is 50% overlap, meaning the hop length is 50% of the frame length (Singh, 2019). The number of frames for each audio file can then be calculated as below in equation (1).

$$Number\ of\ frames = \frac{Total\ samples - Frame\ length}{Hop\ length} + 1 \quad (1)$$

$$Number\ of\ frames = \frac{48000 - 400}{200} + 1$$

$$Number\ of\ frames = 239$$

This almost doubles the density of frames analysed compared to before, thus capturing more detail by improving temporal resolution.

2.1.6 Windowing

When a signal is divided into frames, the abrupt start and ends introduce sharp transitions that are not natural. When a frame signal undergoes a Fourier transform this distortion can lead to spectral leakage (Sahidullah & Saha, 2012), affecting the input features accuracy. To avoid such an issue and make spectral analysis reflect signal energy accurately, windowing is implemented. The most seen in SD literature is a hamming window. A hamming window

diminishes signal amplitude at the boundary edges of a frame, smoothing the transitions, which overall leads to a more accurate spectral representation.

2.1.7 Signal Preparation Overview

Below is an image depicting a raw audio signal being segmented with windowing applied before undergoing STFT.

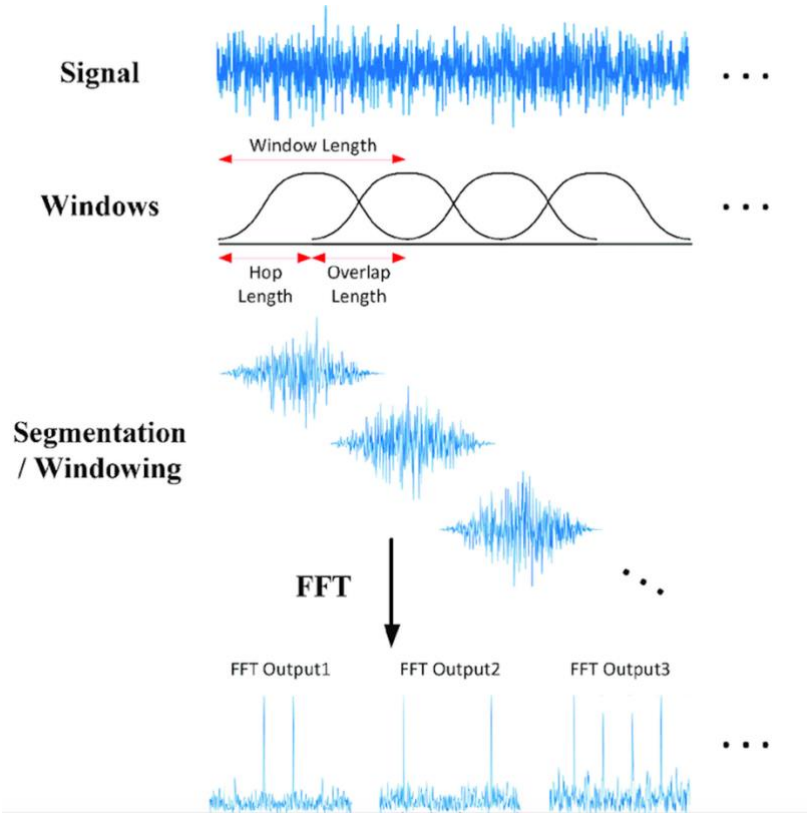


Figure 4: This figure illustrates the signal processing steps to convert a time series signal into its spectral components. Image Source: (Jeon et al., 2020).

2.2 Feature Extraction

Once the data has been cleaned, desired features can be extracted. In audio signal processing this involves transforming the audio data into measurable characteristics. This is essential to reduce the complexity of data. Passing in raw audio signals for model training would consist of thousands of parameters per second, which is costly and increases system complexity (Wang & Paliwal, 2003). By transforming the raw data, dimensionality is reduced,

highlighting key attributes such as frequency, pitch, and energy. Features extraction methods are important for pattern recognition tasks, enabling models to yield better performance (Wang & Paliwal, 2003).

In the SD domain, many features extraction techniques have been researched. Such as phonation features, utterance-based features, acoustic analysis, pitch determining features, Linear Predictor Cepstral Coefficients (LPCCs) to name a few. However, the most implemented in the literature include Mel Frequency Cepstral Coefficients (MFCC) and Spectrograms (Sheikh et al., 2023). MFCC's are the most utilised feature extraction method for stuttering classification. This approach aims to quantify the gross shape of the spectrum in a speech signal, while removing the fine spectral structures that are often irrelevant (Bäckström et al., n.d.).

2.2.1 MFCC Process

MFCC's can be broken down in six steps. Three steps of which were incorporated in the above preprocessing phase. 1. Dividing the data in frames, and 2. Apply a hamming windowing. After which 3. The Short-Time-Fourier Transform (STFT) is performed.

Step 4. is to take the STFT spectra obtained (FFT outputs in Figure 6.) and pass it through a triangular filter bank. The human hearing scale is not represented linearly, and the mel filter bank converts these linear frequencies in the FFT to mel frequencies. This better approximates human hearing. Typically, 26-40 filter coefficients have been used in the speech domain (Sheikh et al., 2022a). The frequencies in Hertz are converted to the mel frequency scale. Mel frequency can be calculated using the formula (2) below.

$$mel\ (f) = 2595.\log\left(1 + \frac{f}{100}\right) \quad (2)$$

A visual representation of the triangular filter bank is seen below, which maps hertz frequency to mel frequency.

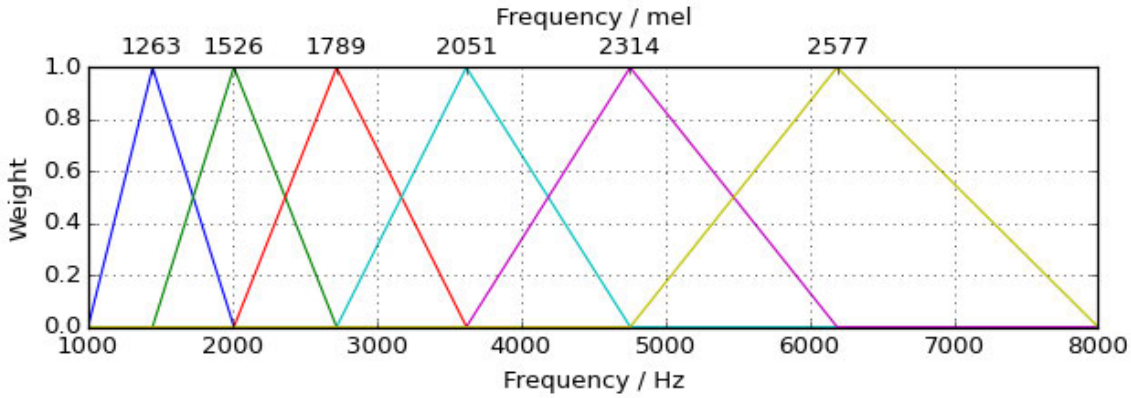


Figure 5: Triangular filter bank converting hertz to mels. This scale better approximates human hearing, Image source: (Gündert, S., 2014).

Step 5. Logarithmic scaling is applied by taking the log of the mel-scaled filter bank energies. This allows for loudness perception as the lower energy bands are emphasised which could represent important quieter nuances in a speech signal (Singh, 2019).

Finally step 6. can be implemented on the log mel spectra. This applies the Discrete Cosine Transform (DCT). The DCT is useful as it helps decorrelate the log mel spectrum coefficients. By transforming these into cepstral coefficients, the resulting features are more linearly separable (Singh, 2019). Which is beneficial for further analysis particularly when training models in a machine learning environment. Thus, MFCC offers advantages over linear frequency cepstra due to its compact representation and emphasis on key acoustic information (Davis & Mermelstein, 1980). Sheikh et al (2021) work on SD employed 20 MFCCs as input features for every 25ms window of speech with a hop length of 12 ms.

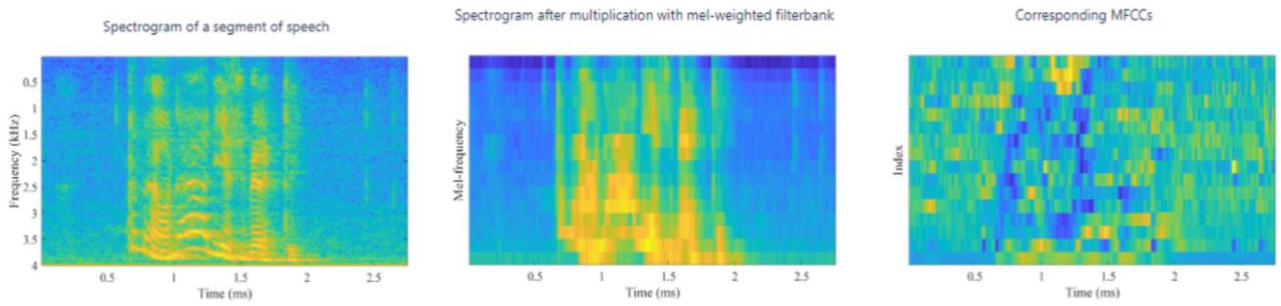


Figure 6: Converting a speech segments spectrogram to the mel log filter bank spectrogram, and finally after applying the DCT, the corresponding MFCC. Image source: (Bäckström et al., n.d.).

Above is an example from Bäckström et al., (n.d.) of how a segmented speech spectrogram converts to its corresponding MFCCs. Numerically speaking, an MFCC pixel with a positive value indicates the spectral energy is in a low frequency region, while a negative coefficient value indicates spectral energy is mostly high frequency in that region. The MFCCs which have undergone the above steps now cleaned, curated, and extracted features. And are ready to be used for training a model. The steps investigated provide beneficial information to the signal analysis aspects of this project as MFCC are the dominant input features in SD models.

It is also common practice to apply normalisation techniques to the features. This can be done with min-max scaling. Rescaling the data range between 0 and 1. By doing so it minimises the impact of external factors such as recording conditions. Normalising the input also helps the learning process by preventing vanishing (too small) or exploding (too large) gradient issues during backpropagation. This leads to stable optimisation and smoother training (Arora, 2024).

2.2.2 Using Raw Data in Pre-Trained Model

The above-mentioned technique is hand engineered and approximate the human auditory system which require manual manipulation. Recent endeavours in alternative approaches have been researched such as representation learning that doesn't require manual annotations. Bayerl et al., (2022) work adapted the self-supervised learning framework Wav2Vec2.0. This is a convolutional feature encoder that leverages large amounts of unlabelled raw audio data and transforms it into to its own latent representations. The pretrained model identifies audio representations such as phoneme recognition, speech emotion recognition, and mispronunciation detection (Baevski et al., 2020). Bayerl et al., (2022) fine-tuned the model using the SEP-28k dataset. All experiments done comprised 12 transformer blocks atop a convolutional feature extractor to extract audio representation features about a vector's relationship to others. 768-dimensional speech representations were derived every 20 ms from raw audio. This trained Support Vector Machines (SVMs) for a single disfluency classification against all other class types (Bayerl et al., 2022) with F1 scores evaluated on the FluencyBank dataset shown below.

System	Mod	Bl	Int	Pro	Snd	Wd
FluencyBank						
W2V2-BASE	-	0.30	0.70	0.51	0.50	0.39
W2V2-STL	-	0.31	0.83	0.52	0.40	0.40
W2V2-MTL	-	**0.33**	**0.84**	**0.60**	**0.60**	**0.43**

Figure 7: Bl (Block), Int (Interjection), Pro (Prolongation), Snd (Sound Repetition), Wd (Word Repetition). W2V2 (Wave2Vec2.0), Base (Base Model), STL (Single Task Learning) MTL (Multi-Task Learning). Image source: (Bayerl et al., 2022).

2.3 Deep Learning Process Thus Far

The diagram below depicts the DL pipeline and its essential stages. The prominent datasets have been identified above, including the necessary data cleaning techniques to prepare these datasets. The most used features in SD have been explored. Before discussing the next components, which are the training and testing splits, learning algorithms (or model architectures) should be explored.

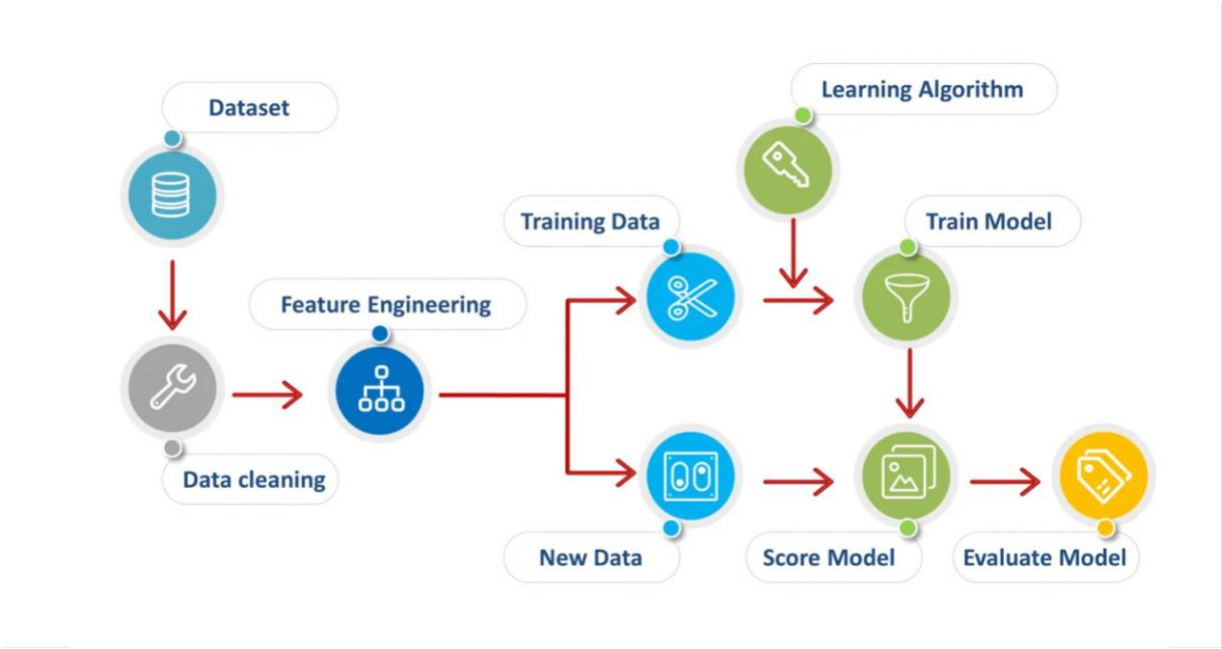


Figure 8: Steps involved using an audio dataset for machine learning purposes. Image source: (GTS AI, 2023).

2.4 Model Architectures Overview

Model Architectures or Learning algorithms are the backbone of machine learning. They can generally be broken down into the following categories: supervised learning, unsupervised learning, semi-supervised, and reinforcement learning. With regards to speech and SD research, emphasis has largely been placed on supervised learning classification models.

Traditional models that have been employed previously in the SD domain include artificial neural networks (ANN), K-nearest neighbour (KNN), and support vector Machines (SVM) (Alnashwan et al., 2023) to name a few. These methods all analyse and learn from historical data and make decisions about new data. ANN is a set of interconnected nodes that learn complex patterns through data. KNN is a proximity classifier that uses those closest training examples in the feature space and votes are cast based on the number of nearest neighbours. SVM utilises a hyperplane that best separates different classes. However, a review on SD by Alnashwan et al. (2023) found a preference for DL architectures such as Long Short-Term Memory (LSTM) and Convolutional Neural Networks (CNN) which have made advancements in stuttering analysis.

DL is made up of layers, and through these layers a model will learn to recognise complex stuttering characteristics. By detecting basic speech edges and texture patterns in the early layers, the deeper layers will look to detect increasingly more complex patterns. The final layer use functions like SoftMax to convert these features into probability-based predictions for various classes (i.e., Prolongation, Block, Interjection, Sound Repetition, Word Repetition, Fluent Speech). To implement these predictions, various DL architectures can be employed. Each with its own strengths when handling speech data.

2.4.1 CNN Architecture

CNNs automatically detect the significant features without any human supervision (Alzubaidi et al., 2021). The first convolutional layers (seen below in Figure 9) learn low-level features by applying filters (kernels) to local regions of the input image. This input image could be an MFCC for example. After each convolutional layer, a ReLU (Rectified Linear Unit) function introduces non-linearity by outputting positive values and zeroing negative values. The pooling layers reduce spatial size, keeping the most important features (Alzubaidi et al., 2021). And after multiple convolutions, the fully connected layers map the extracted features to a final output. Then a function such as SoftMax turns these outputs into a probability distribution of all possible classes.

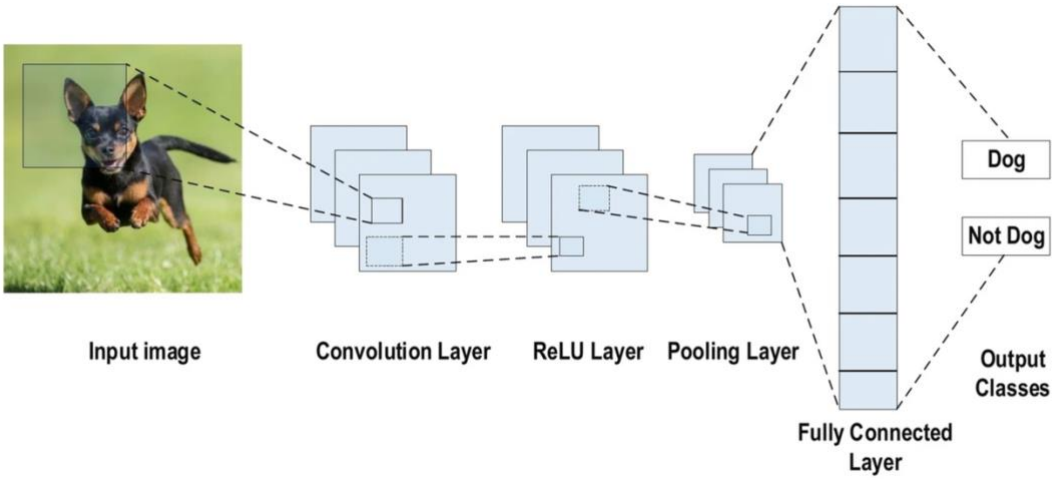


Figure 9: An example of CNN architecture for binary image classification. Image source: (Alzubaidi et al., 2021).

A loss function is applied in a network to measure the difference between predicted output labels with the actual output labels, allowing for the error to be calculated. This error is then propagated backward through the network using backpropagation, which determines the gradients that contribute to the loss (Alzubaidi et al., 2021). As the model iterates through a

training dataset, these gradients update the network parameters (or weights) to minimise the loss and match predicted labels to actual labels.

2.4.2 LSTM Architecture

Bidirectional Long Short-Term Memory (BLSTM) processes data in both the forward and backward direction, see below figure 10 from Gupta et al. (2020) work. This is a type of recurrent neural network (RNN). While LTSM models assume only the previous frame affects the current frame and has been used heavily in SD (Lea et al., 2021). BLTSM assumes the previous and next frame are related to the current frame and can be considered a stronger network for speech recognition applications due to the larger context provided (Graves et al. 2013).

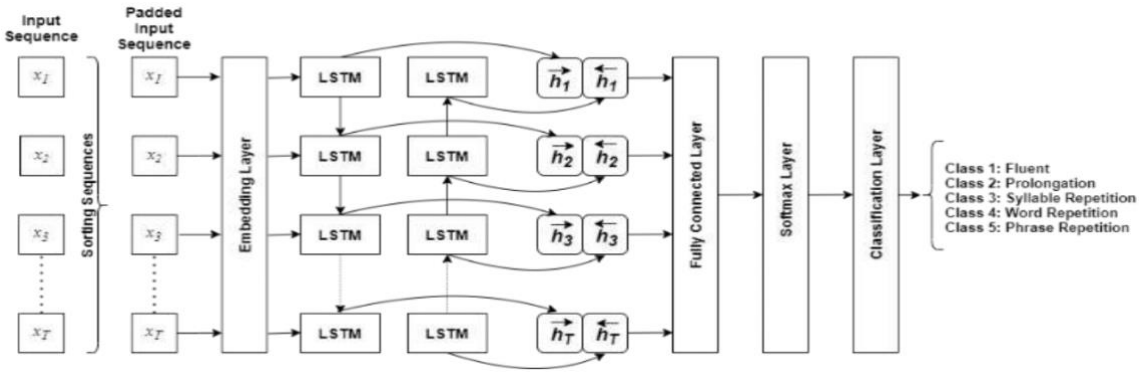


Figure 10: Here is the BLSTM network used in Gupta et al. (2020) with input from weighted MFCCs. Output is the classification of 5 classes (4 disfluencies were classified). Image source: (Gupta et al., 2020)

Both the LSTM and BLSTM architectures incorporate three gates - forget, input, and output, that regulate the flow of information within a network. During training, as the model receives the labelled input features, these gates prioritise what data is important to retain, update, and what can be forgotten (Gupta et al. 2020). These gates adjust the model weights during

backpropagation guided by the loss function. They are responsible for supervising long-term dependencies in the data and avoid vanishing gradient issues (Gupta et al. 2020).

2.4.3 Conv-LSTM Architecture

Such architectures as above can be combined to advance traditional LSTM or CNN models. By combining these networks, Conv-LSTM models process the data through a series of convolutional operations allowing it to learn the spatial dependencies as with CNN, while connecting temporal sequences as seen in LSTMs. Kourkounakis et al. (2021) work along with several others have utilised such architectures for SD.

2.4.4 Residual Connection Architecture

SD models have also implemented models that use residual connections. ResNet18 and ResNetBiLstm have incorporated these connections and have been successful in the field (Filipowicz & Kostek, 2023). Residual networks are favoured as they avoid the vanishing gradient issue. ResNet18 architecture for example, has 72 layers with 18 deep layers that incorporate residual connections that skip over and form shortcuts between layers. Differing from standard feedforward neural networks seen above. See Figure 11 from Wong (2021) below for a diagrammatic depiction of how a residual connection differs.

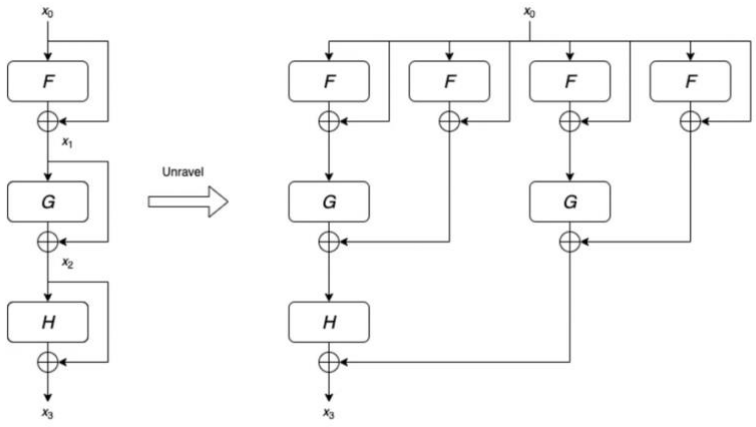


Figure 11: This diagram illustrates the difference between feedforward neural networks and residual connections that consist of many paths and lengths. Image source: (Wong, 2021).

When saturation occurs in a network, it can lead to the vanishing gradient problem during backpropagation, particularly in feedforward neural networks. Residual connections help preserve gradient magnitude through these shortcut paths, ensuring earlier layers update their weights effectively, thus enhancing network performance (Wong 2021).

2.4.5 Pre-Trained Transformers

Transformers are a class of DL model designed for sequential data that leverage an attention mechanism to process not only local dependencies but also global relationships in an audio sample. Pre-trained transformers are initially trained on large-scale datasets to learn meaningful language and audio representations, which can then be applied to fields such as SD. This transferred learning approach enables models to better capture disfluent speech patterns. Several recent studies have used pre-trained models like the Wav2Vec2.0 on the SEP-28K dataset Bayerl et al., (2022) and other datasets Sheikh et al., (2023). Wav2Vec2.0 has been used both as a feature extractor by turning the raw audio signal into latent feature vectors, and through fine-tuning layers for stuttering specific tasks. However, due to the self-attention mechanism, these models are significantly more computationally complex.

2.4.6 Brief Introduction of Key Hyperparameters

After selecting a model architecture, other hyper-parameters of the network must be initialised. The entire training process depends on these hyper-parameters, and time should be taken to investigate optimal values to avoid overfitting a model and meaningless computation (Leonel, 2019). Some of which include number of epochs, batch size and learning rate.

Gupta et al. (2020) investigated variations in these hyper-parameters and firstly found 50 epochs provided the suitable trade-off between accuracy and computation. An epoch is one

complete cycle through the whole training dataset. Secondly, various batch sizes have been evaluated in the literature, from eight which produced the highest accuracy in Gupta et al. (2020) work, while Al-Banna et al. (2024) utilised a batch size of 128 and produced state of the art results. Batch size is the number of training samples in a single iteration. Smaller batch sizes lead to more frequent updates yet are not always desirable due to potentially being more erratic and causing noisier estimates. It also increases training time.

Another parameter to consider is learning rate. It is a decaying step size at each iteration that moves towards the loss function minimum. The minima of the loss function signifies when the model's prediction is closest to the actual values. See the figure extracted from Jordan (2018) below for a representation.

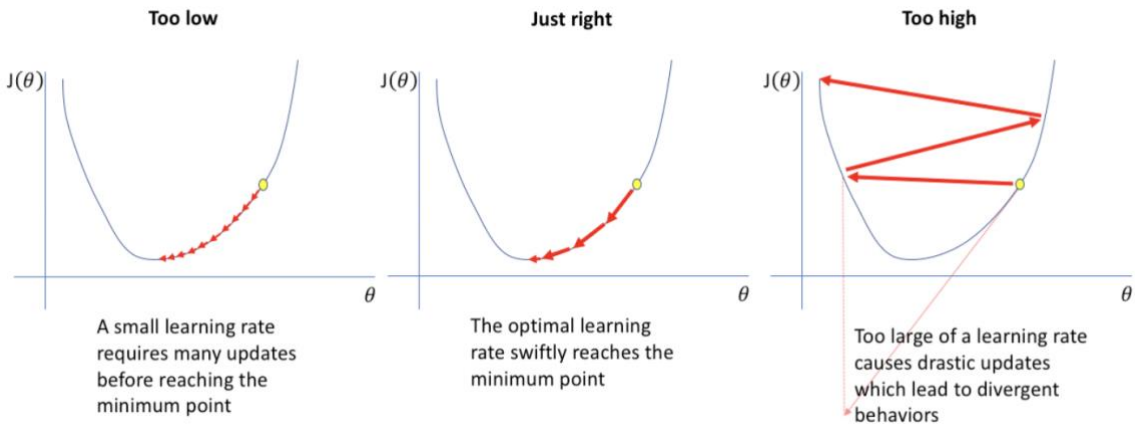


Figure 12: This illustrates how different learning rates affect the loss function in a neural network. The curve represents the difference in the loss values between predicted and actual outcomes, showing the impact of learning rate choices on model convergence. Image Source: (Jordan 2018).

As Brownlee (2020) states "Learning rate controls how much to change the model in response to the estimated error each time the model weights are updated". And that choosing a high learning rate, can cause faster convergence, but also may lead to divergence due to

unstable training as seen in the figure. A small value can also create problems, such as longer training time or poor performance due to a vanishing gradient.

The reason why a vanishing gradient is undesired is that it can cause saturation in the network performance, or even degrade performance due to getting stuck in local minima (Chandola et al. 2021).

Number of hidden layers is a model specific parameter that determines the model structure. More hidden layers typically correlate to capturing deeper levels of abstraction in data, but to many layers can lead to overfitting, which as discussed earlier will affect generalisation to real world scenarios. Dropout can be a solution to negate overfitting; by randomly dropping neurons at specific layers in a network can facilitate model generalisation. It also reduces training time and memory usage (Ko et al. 2017). Al-Banna et al. (2024) used a drop of 0.4 at each layer within their CNN.

Designing a DL model involves numerous architectural choices and considerations. This research has explored those commonly used in the stuttering domain backed by strong evaluation results.

2.5 Previous Research in Stuttering Detection and Classification

Table 1: This table illustrates the state-of-the-art literature and the corresponding datasets, features, model architectures, and evaluation protocol used.

Research Study	Dataset	Features	Model	Evaluation
Mohapatra et al. (2022)	SEP-28K FluencyBank	Wav2Vec2.0 Embeddings	CNN	10-Fold Cross Validation
Lea et al. (2021)	SEP-28K FluencyBank	Log Mel FilterBanks	LSTM ConvLSTM	-
Jouaiti and Dautenhahn (2022)	SEP-28K FluencyBank	MFCC + other speech features	BI-LSTM	10-Fold Cross Validation
Sheikh et al. (2021)	UCLASS	MFCC	TDNN	10-Fold Cross Validation
Bayerl et al. (2022)	SEP-28K FluencyBank KSoF	Wav2Vec2.0 Embeddings	Fine-tuned Wav2Vec2.0	10-Fold Cross Validation
Filipowicz and Kostek (2023)	SEP-28K FluencyBank	MFCC + Wav2Vec2.0 Embeddings	ResNet18 ResNet BI-LSTM	10-Fold Cross Validation
Al-Banna et al., (2024)	SEP-28K FluencyBank	MFCC + other speech features	CNN BI-LSTM CBAM	10-Fold Cross Validation

2.6 Class Balancing Methods

Due to the diversity and unique characteristics of stuttering, detecting it remains one of the most challenging tasks, largely due to the significant class imbalance across various types of disfluencies within datasets.

Sheikh et al (2023b) research (Figure 1. from the paper is shown below) illustrated the high class-imbalance across the SEP-28k dataset, which can lead to bias towards the majority class in classification (fluent). Addressing this issue can generally be done three ways. At the data-level, cost-level, and architecture level (Fernandez et al., 2018).

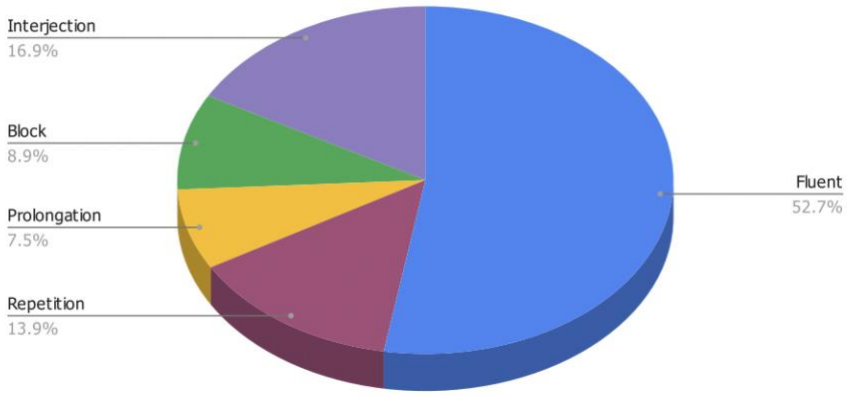


Figure 13: Stuttering Data Distribution in SEP-28k Dataset (removing nonstuttering labels). Image source: (Sheikh et al., 2023b).

Research by Mohapatra et al. (2022) utilising the SEP-28k dataset identified that filtering ambiguous disfluencies (meaning they used only files where all three annotators agreed on the disfluency) in a convolutional neural network (CNN) produced better results than when using samples where one or two annotators marked a disfluency. Although large amounts data is crucial to machine learning, by curating the dataset to a quarter of the original size they illustrated that less data with more 'quality' is more valuable than just the quantity of data alone.

Data-level attempts to re-balance distributions can be done through a combination of under-sampling or over-sampling. In deep neural networks, under-sampling can lead to loss of

crucial training samples or conversely over-sampling can lead to overfitting (Fernandez et al., 2018). When a model is to complex, it cannot generalise well. Meaning high performance on the dataset, but poor performance on new, unseen data.

Sheikh et al (2023b) researched a cost-based approach by assigning class weighting (more weight to minority classes), based inversely on the number of class samples. A cross-entropy loss function was used. This loss function allows errors in minority classes to contribute more heavily toward the total loss, forcing the model to pay more attention to correctly classifying minority classes. This approach did improve minority class detection, yet a trade-off between accuracy on the majority class was seen. Hence, reducing the overall F1 score.

Bader-El-Den, Teitei, & Adda (2016) developed an architecture-based approach to class imbalances by splitting training data into majority and minority classes and relabelling the nearest (data point) neighbours of the minority instances into a subset class. This creates a two-level classification, and Sheikh et al (2023b) utilized such a framework with the first classifier to differentiate fluent vs disfluent speech. And the second classifier focusing on the type of disfluency. This approach looked to optimise the combined losses and train the algorithm to recognise disfluent speech against fluent speech, and subsequently classify the disfluency subtype.

The above-mentioned research has demonstrated the challenges associated with imbalanced datasets, highlighting the need for innovative solutions to improve model performance. In this context, data augmentation offers another strategy to address these limitations.

2.7 Data Augmentation

Data augmentation is another aspect of machine learning that can improve a model's performance and generalisation. It can be beneficial to use such a technique when data size

is small or there is an imbalance in dataset classes. Depending on the need, different approaches can be taken. Due to speech therapy being done in clinics and potentially open spaces, background noise and echoing could affect a model's capability, especially if it hasn't trained on 'noisy' data.

In Sheikh et al., (2023b) study, reverberation and additive noise were uniquely added from the MUSAN dataset which is comprised of speech from different languages, music, and noises. Four additional copies of augmented data were created and the results in cross corpora dataset testing are seen below. Exploiting data augmentation (in orange) illustrated a high accuracy percentage for each class classification (except B: Blocks) and high F1 score. Showing that data augmentation can potentially have a positive impact on the generalisation of models. See Figure (14) below for a graphically presentation of their results.

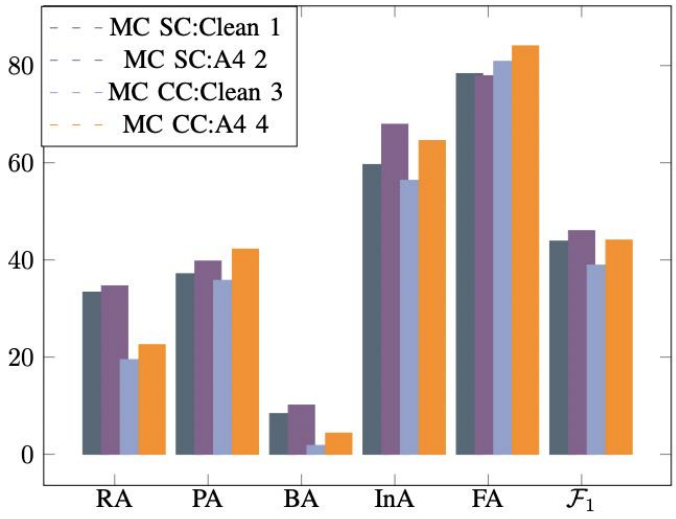


Figure 14: This work compares the Multi Contextual (MC) StutterNet. Same Corpora (SC), cross corpora (CC), Data Augmented (A4).

Repetition (R), Prolongation (P), Block (B), Interjection (In), Fluent (F), Accuracy (A).

Image source: (Sheikh et al., 2023b).

2.8 Validation Protocol and Evaluation Metrics

Having addressed the methods to manage data imbalances, it is also essential to implement a robust validation protocol. Most studies in SD have used 10-fold cross validation, such as Sheikh, Sahidullah, Hirsch, and Ouni (2021) and Al-Banna et al., (2024). Dividing the dataset into subsets reduces bias and variance and measures the generalisation error. If each subset performance is similar, this will be an indication to the model's generalisability. This method is more robust compared to train/test evaluation and reduces the risk of overfitting. An example of 5-fold cross validation is seen below.

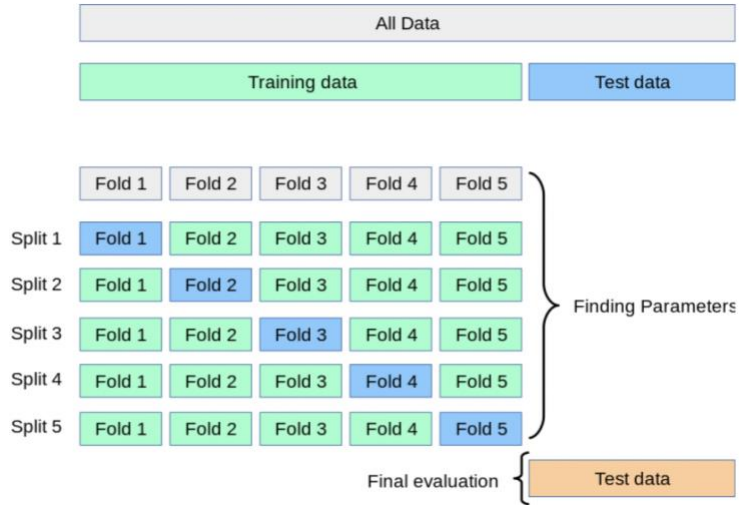


Figure 15: 5-fold cross validation takes the average of results for each split. Image source: (Scikit 2024).

The dataset example above is divided into k equal-sized folds, k=5. And trained for 5 iterations. For each iteration k-1 folds are used for training and the remaining fold used for validation. The validation data changes for every iteration, and the performance is averaged. Although computationally expensive, this approach produces less bias and variance in results.

One crucial evaluation metric is the confusion matrix, which is necessary to understanding a model's strengths and weaknesses. It provides a detailed breakdown of correct and incorrect predictions.

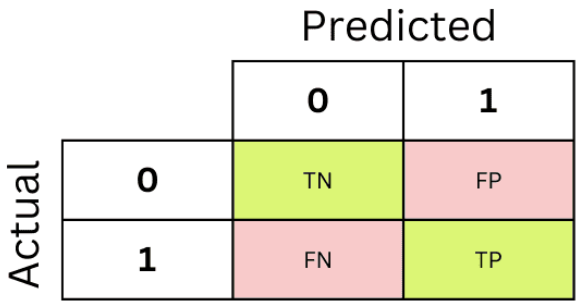


Figure 16: A binary confusion matrix that compares the actual values of a dataset against the models predicted values. Image source: (Singh et al., 2021).

TN = true negatives, TP = true positives, FP = false positives, and FN = false negatives,

By analysing this confusion matrix, the performance of binary classification models can be understood. An actual value = 1, and a predicted value = 0 will lead to a FN, for example. This evaluation tool is employed after model training and again after model validation and the results are used to calculate metrics like accuracy and F1 score.

The F1 score is a single metric described as the harmonic mean of the precision and recall of a classification model. Ranging from 0 to 1. A score closer to 1 indicates better model performance (Sharma, 2023). And is widely used for performance evaluation in the SD domain. A model that achieves high accuracy but a low F1 score indicates the model has poor generalisability and will not perform well on unseen data. F1 score can be calculated using formula (3) below.

$$F1 = 2\,\frac{Precision * Recall}{Precision + Recall} \tag{3}$$

where Precision = $\frac{TP}{TP+FP}$

and Recall = $\frac{TP}{TP+FN}$

EER (equal error rate) is a statistical measure to identify the rate of misclassified predictions with the model. It is calculated as the total inaccurate predictions over the total predictions.

$$EER = \frac{FP+FN}{FP+FN+TN+TP} \quad (4)$$

These evaluation metrics were used across the literature, with emphasis on the F1 score (Al-Banna et al., 2024).

2.9 Summary of the Elements Explored in Automated Stuttering Detection Review

Table 2: Summary of all the elements that were explored in the literature review for autonomous stuttering detection.

Datasets	Preprocessing Techniques	Key Features	Model Architectures	Hyperparameters	Evaluation Metrics
UCLASS LibriStutter SEP-28K	Noise Reduce Resampling Frame Length Hop Length Windowing Padding Trimming Empty File Normalisation	STFT Mel Spectrogram MFCC Pretrained Model Embeddings	Traditional CNN LSTM Conv-LSTM Residual Connections Transformers	Epochs Batch Size Learning Rate No. of Hidden Layers Dropout Rate SoftMax Function Loss Function Class Balancing Augmentation	k-fold Validation Train/Test Validation Confusion Matrix F1 Score EER Accuracy

CHAPTER 3 METHODOLOGY

The above section explored the state-of-the-art literature in SD. The investigated articles moulded the methodology for this research project. As found, DL models can be shaped by a plethora of elements that can all influence performance. This methodology has adapted some compelling elements to accurately perform binary classification of the following disfluencies against all other classes.

(i) Blocks, involuntary or tense pause before a word

(ii) Prolongations, a prolonging of a sound, or stretching a word.

(iii) Interjections, inserting a sound or word during a spoken sentence.

(iv) Sound repetitions, repeating part of a word or syllable multiple times.

(v) Word repetitions, repeating a whole word in a sentence.

This methodology is organised as follows: **3.1 Dataset**, which describes the dataset used and the breakdown for training and evaluation. **3.2 Preprocessing Techniques**, details how the data was prepared; **3.3 Features and Normalisation**, explains the feature extraction and scaling methods; **3.4 Hyperparameters**, lists a brief description of the key hyperparameters for the models and training; **3.5 Models Investigated**, covers the selected model architectures; **3.6 AST Model**, introduces a new approach to SD; and **3.7 Overview of Methodology**, provides a tabular summary of the overall research approach.

3.1 Dataset

The SEP-28K dataset was implemented for this research. It is the largest publicly available annotated dataset. And has been used most frequently since its release in 2021 (Alnashwan

et al., 2023). In studies Lea et al. (2021), Jouaiti and Dautenhahn (2022), Filipowicz and Kostek (2023), Al-Banna et al. (2024) and Mohapatra et al., (2022) used this dataset along with FluencyBank dataset for evaluation. Due to limitations in computational resources using a M1 MacBook Pro, the FluencyBank dataset was not implemented for evaluation.

The SEP-28K dataset contains 28,137 annotated three-second interval speech segments from public stuttering podcasts. Of which the five disfluencies can be sorted below.

Table 3: Breakdown of SEP-28k dataset by disfluency type (Al-Banna et al., 2024).

Dataset	Prolongation	Block	Sound Rep	Word Rep	Interjection	Total of Obs	Total hours
SEP-28K	3480	1679	2517	2295	4322	14293	11.91

On downloading the SEP-28K dataset, a significant number of samples had been removed from their domain pages, making them inaccessible and therefore irrelevant to the dataset's purpose. As a result, 21856 files were downloaded from the initial 28177 audio files, over a 20% reduction.

3.1.1 Training and Validation Protocol

Due to aforementioned limitations, this research opted for a single 80/20 train-validation split (hold-out method), where the data is divided - 80% for training and 20% for validation.

The training/validation split consisted of 17484/4372 audio files. The training set contained 6316 interjection labels with single annotator agreement which indicates imbalance between the minority and majority class (6316 samples versus 11168 samples).

The split was made reproducible, meaning the samples that were divided into training and validation were always the same for each script run, allowing for repeat testing to examine hyperparameters and performance. The validation data was held out and used to check performance of the model after each epoch. This is an inferior evaluation protocol compared to k-fold cross validation as data is considered "wasted" in the sense that 20% of the dataset isn't used at all for training. For this project, the protocol was assumed sufficient as it is computationally inexpensive.

This research involved training models to classify one disfluent class against all other samples. This is known as a binary classification task. The below figure illustrates the basic difference in classification between binary and multi-class.

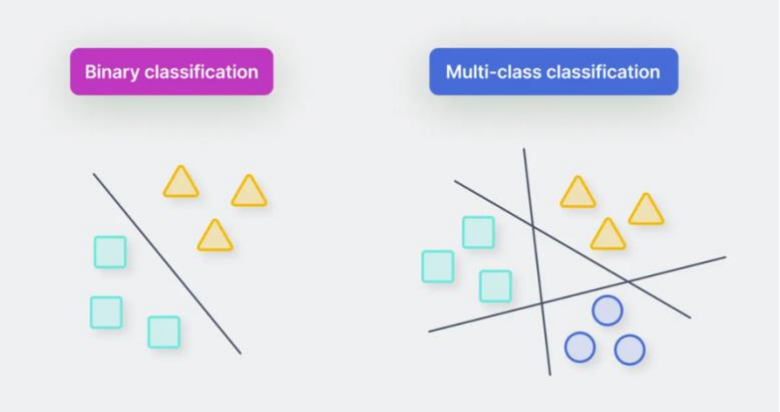


Figure 17: This image shows the difference between binary classification ("Interjection" and "no Interjection") and multi-class classification, i.e., ("no Interjection", "no Sound Rep", "no Word Rep", "no Prolongation", "Block"). Image source: (Shah, 2023).

The 'interjection' class was selected to test and compare model performance. This is because the interjection class had the most samples among disfluencies on the SEP-28K dataset. Subsequent research looked to test the best performing model on all disfluency types.

3.1.2 Annotator Agreement Level Testing

The SEP-28K dataset used three annotators (not SPs) who were trained to classify the disfluencies based on their own perception leading to poor agreement levels for some disfluency types. The Fleiss Kappa score, a statistical measure (from -1 to 1) used to access reliability between annotators found ranges from 0.11 to .62 per disfluency in the dataset (Lea et al., 2021). Indicative of slight agreement to substantial agreement. Mohapatra et al., (2022) addressed this issue using classifications from which all three annotators agreed on disfluency class. Their models showed improved results when using unanimous agreement labels for all disfluency types. This reduced the number of samples for each disfluency but increased sample 'quality', leading to improved model results.

As such, a comparable method of data distillation was evaluated in this research, testing single annotator agreement versus majority versus unanimous. If two annotators agreed on a disfluency, it was classed as a majority. The aim for this was to reduce computational load while maintaining robust data. These agreement levels were tested using the CNN and LSTM model. The remaining class counts in the SEP-28K dataset for each annotator agreement was created in Table 3 below.

Table 4: Breakdown of the available SEP-28K Dataset after download based on annotator agreement levels.

Single Annotator Agreement Classed Disfluency Counts (1 Annotator agreed)					
Dataset	Interjection	Prolongation	Block	Word Rep	Sound Rep
SEP-28K	7862	6694	9343	3922	4824
Majority Annotator Agreement Classed Disfluency Counts (2 Annotators agreed)					
Dataset	Interjection	Prolongation	Block	Word Rep	Sound Rep
SEP-28K	4917	2150	2667	2390	2097
Unanimous Annotator Agreement Classed Disfluency Counts (3 Annotators agreed)					
Dataset	Interjection	Prolongation	Block	Word Rep	Sound Rep
SEP-28K	2810	569	434	1435	785

3.1.3 Class Imbalance Issues with the Dataset

One consistent challenge emerged with the dataset. The uneven spread across disfluencies was evident for each agreement level. A dataset that is unbalanced can lead to model bias favouring the majority class. This can be misleading, as a model that just predicts the majority class, failing to identify the minority class, can still achieve high accuracy. (Chawla, Bowyer, Hall, & Kegelmeyer, 2002).

To address this issue, a weighted loss function was implemented. By assigning a higher weight to the minority class, its contribution to the total loss is increased. This in turn forces

the model to pay more attention to the underrepresented class, thereby improving its ability to detect stuttered speech (Krishna et al., 2021). As seen in equation (4) below class weights can be assigned inversely proportional to how often they appear in a dataset (Kamaldeep, 2023). In this case:

$$Class\ Weight\ = \frac{Total\ number\ of\ training\ samples}{2 * (Number\ of\ training\ samples\ in\ class)} \quad (4)$$

$$Class\ Weight\ 'Interjection' = \frac{17484}{2 * (6316)}$$

$$Class\ Weight\ 'Interjection' = 1.38$$

$$Class\ Weight\ 'No\ Interjection' = \frac{17484}{2 * (17484 - 6316)}$$

$$Class\ Weight\ 'No\ Interjection' = 0.78$$

In this binary classification task, the minority class "Interjection" can be balanced against the majority class "No Interjection" with these class weights. The effect of class weights was investigated in this research.

3.2 Preprocessing Techniques

Before training commenced, the audio files were passed through a series of checks to 'clean' the data. This included

(i) Checking if the file was empty or malformed – skip and ignore,

(ii) Resampling if sampling rate was not 16000 Hz,

(iii) Making all data files the same data type – 32-bit float point files,

(iv) Checking all files are mono – reduce if more than 1 channel existed,

(v) Right padding if an audio file was shorter than 48000 samples and

(vi) Right trimming if an audio file was longer than 48000 samples.

Then the csv file for the SEP-28K dataset containing all classification values had to be modified due to the 20% dataset reduction. A dictionary (labelmap) was implemented to account for all missing audio files (for more details on this and the preprocessing code, see Appendix B). The dictionary key was the filename of the audio clip, and the dictionary value was created as (agreement_map). For each audio clip, the disfluency labels could be assigned based on the annotator's agreement. This coding allowed for annotator agreement level manipulation.

3.3 Features and Normalisation

Once the dataset had been cleaned and curated, the audio data files were converted to both mel spectrograms and MFCCs as these features were the most utilised in the research (Alnashwan et al., 2023) (Shiekh et al., 2022a). The series of images below shows an audio file from SEP-28K dataset and the extracted feature types.

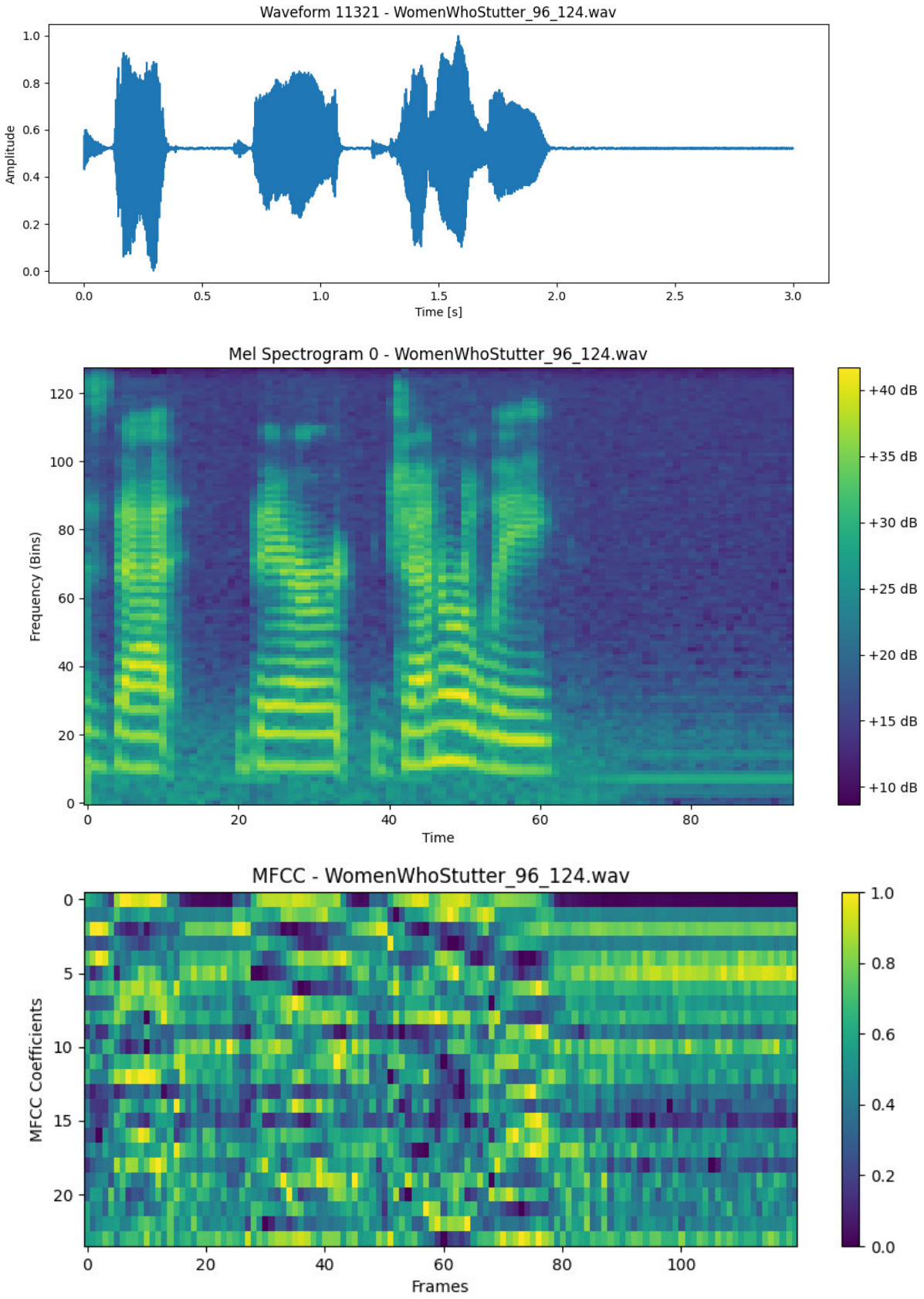


Figure 18: On top is an audio file from the SEP-28k dataset. In the middle shows it transformed into its mel spectrogram. On the bottom shows it transformed into its MFCC. Created in PyCharm.

The mel spectrogram x axis has been converted, originally from a 3 second audio file to 100 temporal frames. It has 128 mel frequency bins, seen along the y axis which means the mel frequency scale of the audio signal is divided into 128 distinct bins. This technique builds on earlier work with the STFT spectrograms seen in Kourkounakis et al., (2020), changing the frequency from Hertz to mels. This has been shown to improve speech model results, by compressing higher frequencies and expanding lower frequencies to better reflect non-linear sensitivity in human hearing (Kourkounakis et al., 2020). The torch.audio.compliance.kaldi.fbank() function was used to generate the mel spectrograms.

Below that is the MFCC, which has 24 coefficients (y axis) and a window step of 0.025 (120 frames in x axis) which is coherent to the literature (Sheikh et al., 2021). These features were normalised between 0 and 1 using a min max scaler function. The MFCC discards the finer high frequency speech details seen in the mel spectrogram helping concentrate energy, which occurs after applying the Discrete Cosine Transform (DCT). This compression approximates the broader spectral envelope. A reduction in feature dimensionality can also support generalisability. Predominately MFCC's have been used as input features in the literature (Jouaiti and Dautenha 2022), (Sheikh et al., 2021), (Al-Banna et al., 2024).

Several research efforts in the literature have used multi-feature fusion, including Jouaiti and Dautenhahn (2022) work which combined MFCC's and phoneme classes through parallel models. Due to the computational constraints, this research only utilised one feature type as an input. Therefore, it was important to process the large amounts of data efficiently. Features were saved as .npy files prior to run time. This reduced processing time by computing the features once opposed to every epoch. Cutting an epoch's computation time from 10 minutes down to roughly 30 seconds for MFCCs on the CNN architecture. Both MFCC and mel spectrogram will be used in this research.

3.4 Hyperparameters

Hyperparameters control the training process and involve all the elements that are determined before training commences. As this area has complex variability, it is near impossible to test each variation to find a model's best performance on the SEP-28K dataset. As such, some hyperparameters remained fixed. A brief description of key hyperparameters selected are mentioned below:

- The number of epochs was set to 100, this provides sufficient training time to access the stability of the model, while also not being too computationally heavy. The best F1 score over the 100 epochs was used in evaluation.

- The cross-entropy loss function is most suitable for classification tasks, and it was implemented in this research. This loss function helps quantify dissimilarities between discovered probabilities and the predicted disfluency type (Al-Banna et al. 2024).

- The learning rate was set to 1e-3, as this provides a good balance to avoid divergence or getting stuck in local minima (Gupta et al., 2020). In Sheikh et al., (2023) paper using a pre-trained Wav2Vec2.0 model for stuttering, they used a learning grade between 1e-8 and 1e-3 (1e-3 is standard). Similarly, Al-Banna et al., (2024) study with CNN models for SD used a learning grade of 1e-4.

- The batch size was first tested at 8 samples as used in (Gupta et at., 2020), and later extended to 16 to allow more opportunity for minority classes to influence each batch iteration and reduce computation.

- A dropout rate of 0.3 at each layer was used and standardised across all model architectures.

- ReLU activation function was implemented to introduce non-linearity in the hidden layers and ensure the models can learn complex patterns.

- Optimiser Algorithm. Abubakar et al. (2024) used the Adaptive Moment Estimation (Adam) optimiser algorithm in their research, alongside many others in the field. It is well-suited to SD tasks because of its ability to adapt learning rates, handle noisy gradients, and process sparse data. Sheikh et al., (2023) employed an AAM-SoftMax loss function using Adam optimizer. Al-Banna et al., (2024) also used Adam optimiser.

- Kernel size in convolutional layers. Since the input MFCC's are rectangular, 24 in height by 120 in length, the kernel size should also be rectangular to capture local feature patterns (Badshah et al., 2019) such as stuttered speech. A 3x7 kernel size was used initially. This spans 7 pixels horizontally, and 3 pixels vertically.

- Max pooling for CNN was used. For example, a 1x3 kernel slides over MFCC image which takes the max value within the kernel and then reduces the width dimensionality by a factor of 3, while maintaining height. This changes the feature map dimensions. Ultimately reducing model parameters for computation.

- Stride and Padding were set to 1 by default.

- Two hidden layers in the MLP were implemented. Mihalache and Burileanu (2022) work in the voice activity detection field investigated models that could detect deceptive speech and found using MFCC's and a CNN-MLP architecture with two

hidden layers in the MLP had the best results. Similarly in another field, CNN-MLP models with two hidden layers in the MLP have been utilised (Setitra, Fan, Agbley, & Bensalem, 2023).

3.5 Investigated Models

The architectures explored have been tabularised in Table 4 Below. Model architectures that appeared consistently in the literature were CNN and LSTM variations (Mohapatra et al., 2022). Several papers in SD take advantage of fusion architectures such as Conv-LSTM (Lea et al., 2021) (Kourkounakis et al., 2021), or attention models such as CBAM and Wav2Vec2.0 (Woo et al., 2018) (Bayerl et al., 2022).

For this research CNN, LSTM, and Conv-LSTM architectures were designed to be tested and compared against the novel Audio Spectrogram Transformer (AST) model. For the complete diagram and description of the architectures refer to CNN architecture Figure 19, LSTM architecture Figure 20, AST architecture Figure 21 below.

Specifically, baseline architectures were established for CNN and LSTM models. The priority then was to optimise these models through class weight implementation and secondly through testing which annotator agreement level yielded the strongest model performance. Guided by these results, subsequent research involved a hybrid Conv-LSTM model and the novel AST model. All models were constructed for the binary classification task of detecting a single disfluency - Interjection. The best performing model was then evaluated on its ability to classify all disfluency types.

Figure 19: CNN Diagram

A network diagram of the designed Single Task CNN architecture. It takes a MFCC (shape 1,24,120) as an input image and extracts meaningful features using 6 convolutional layers with ReLU activation and 30% dropout. Max pooling between convolutions reduces the feature space. The flatten layer (shape 16,128, where 16 = batch size) is connected to an MLP consisting of 2 hidden layers that map these features to the output layer where a SoftMax function uses class probability to predict “interjection” (1), or “no interjection” (0).

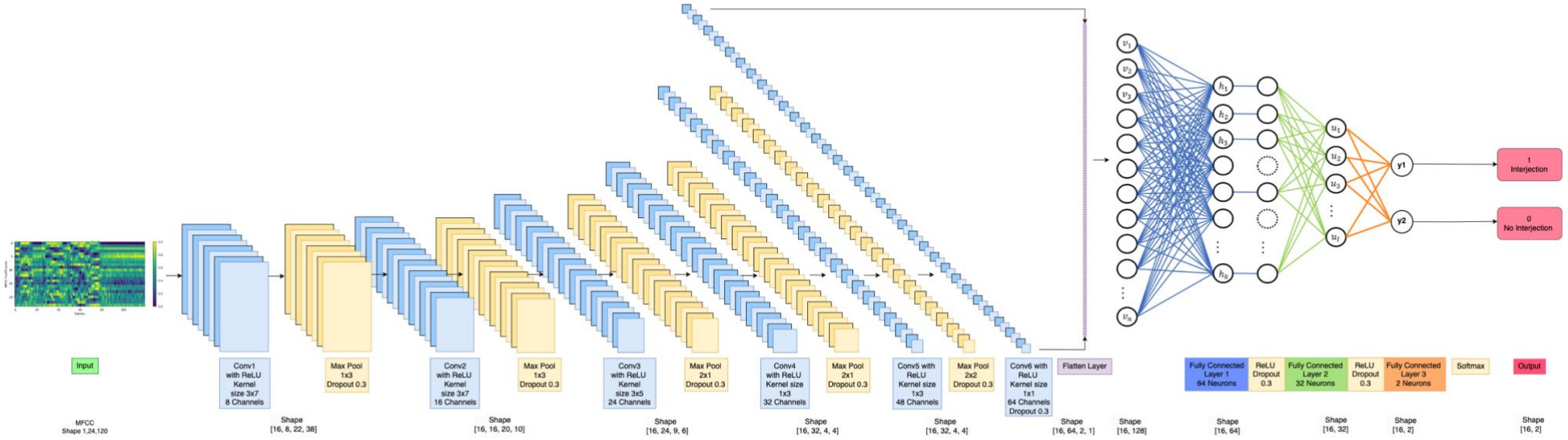
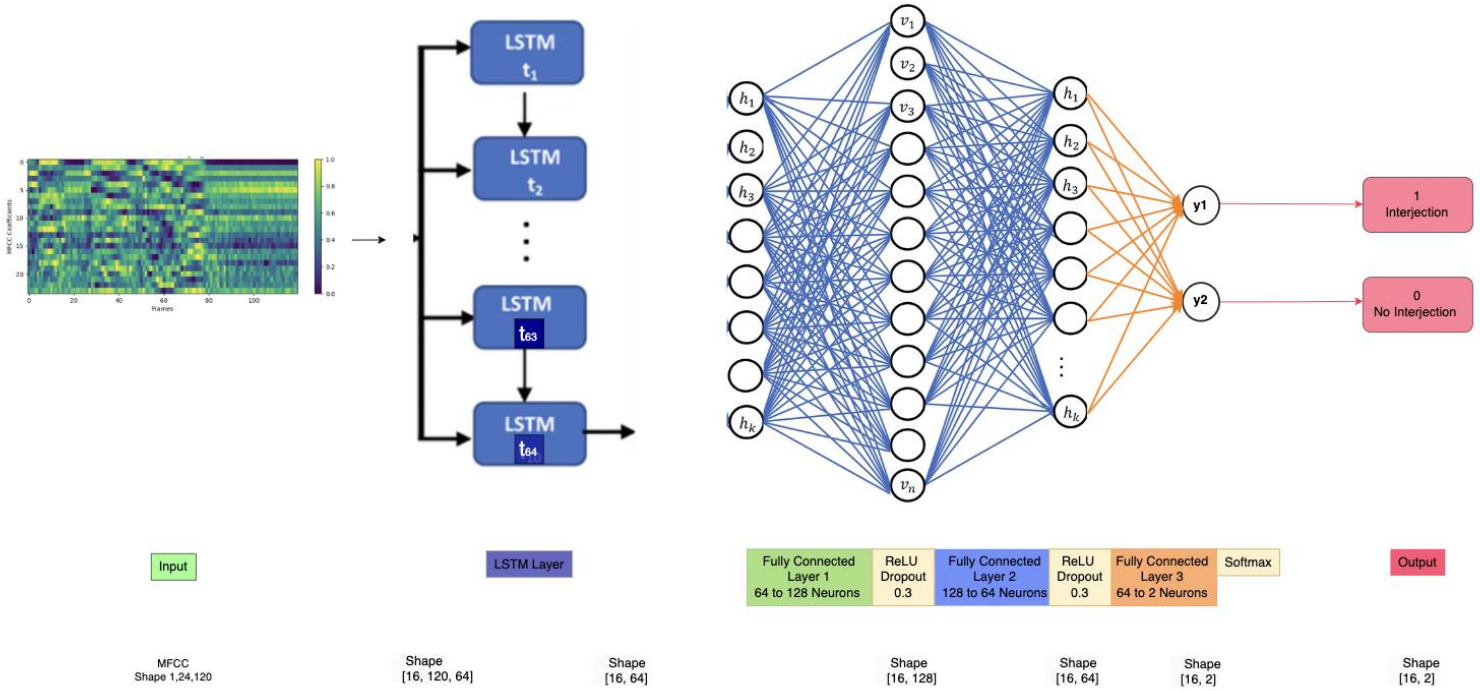


Figure 20: LSTM Diagram

A network diagram of the designed Single Task LSTM architecture. The LSTM section consisting of 64 cells takes 24 MFCC features per time step, capturing the temporal dependencies and patterns. It uses three internal gates to control the flow of information between these time steps. The hidden layer of the LSTM passes this learned information to the MLP. The MLP maps this to a higher dimensional space (128 neurons) where more abstract features are considered. After which it is mapped down to 64 and further down to the 2 output layers. Again, a SoftMax function transforms the raw output to a class probability to predict “interjection” (1), or “no interjection” (0).



3.6 Audio Spectrogram Transformer

Previous research has used pre-trained models, specifically the Wav2Vec2.0 on the SEP-28K dataset Bayerl et al., (2022) and other datasets (Sheikh et al., 2023). This transformer has been used both as a feature extractor by turning the raw audio signal into latent feature vectors (Mohapatra et al., 2022), and secondly through fine-tuning it for SD tasks (Bayerl et al., 2022).

Within in the context of SD research, the Audio Spectrogram Transformer (AST) has never been implemented. The AST uses a vision transformer that was pre-trained on audio data converted into spectrogram images from the AudioSet dataset. Ultimately, presenting this transformer's suitability for audio tasks like SD.

The AST employs 12 transformer layers, with each layer consisting of a multi-head self-attention mechanism. Contrastively, CNNs use layers to capture local patterns in data, such as edges of MFCC images. However, the transformers self-attention mechanism can capture both local and global context by making connections between all patches of the input data, regardless of distance. This approach could be advantageous compared to CNN and LSTM models due to the sporadic nature of stuttering. Though it is important to note limitations; such self-attention architectures significantly increase the number of parameters, approximately 88M weights for the AST.

For this research, the AST's 12 transformer layers are entirely frozen, meaning these weights are not updated during training. Instead, this transferred learning approach uses the AST model's pre-trained knowledge. An MLP head was added on top to learn task-specific features for classification, ultimately reducing the trainable parameters to 230.1k.

The AST model below in Figure 21 uses mel spectrograms of size (128, 1024) from the SEP-28K dataset for the input. It then divides them into fixed sized 1212 patches of 16x16. These patches are flattened and projected into a sequence of feature embeddings. This allows the transformers multiple self-attention layers to refine and capture complex patterns in the mel spectrogram data. Resulting in 768-dimensional output feature vectors which were passed to an MLP. These output feature vectors were pre-saved due to the significant computation time. The MLP then maps these features down through three fully connected layers. This reduced dimensionality from.

- 768 to 256,
- 256 to 128,
- 128 to 2 final output classes.

A SoftMax function then turns these features to a probabilistic output for the binary classification task. In other words, if the audio sample contains “Interjection” or “No Interjection”.

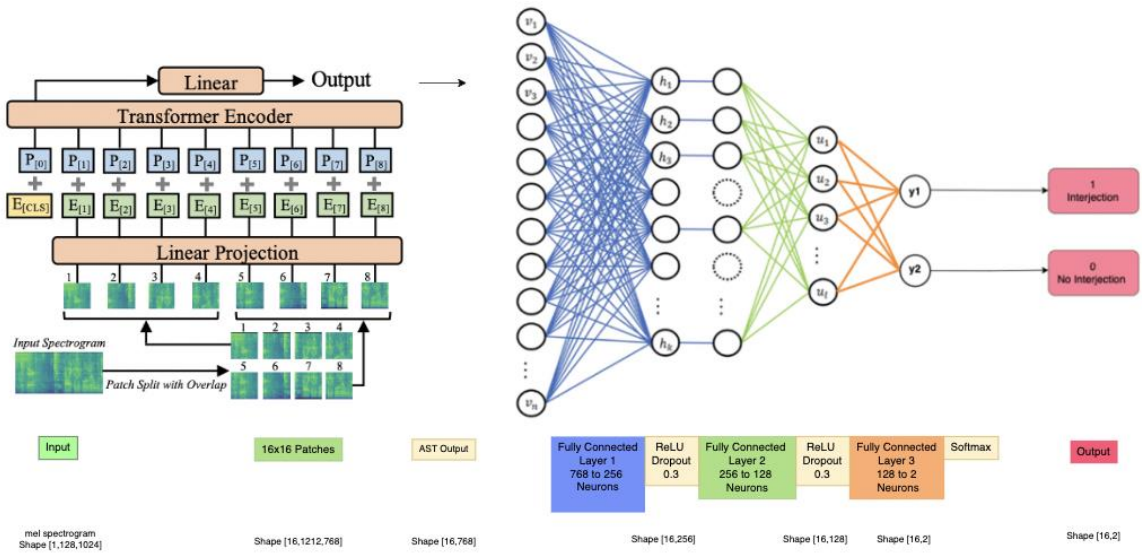


Figure 21: AST Diagram - Audio Spectrogram Transformer architecture from Gong et al., (2021), with MLP head.

3.7 Overview of Methodology

Table 5: This table provides a summary of the investigated models for the research methodology, including model size, the input features used, and the disfluency tested.

Architecture Type	*Trainable Parameters (Weights)*	*Frozen Parameters (Weights)*	*Input Features (SEP-28K Dataset)*	*Disfluency Tested*
CNN	29.1K	-	MFCC	Interjection
LSTM	39.7k	-	MFCC	Interjection
Conv-LSTM	68.7k	-	MFCC	Interjection
Transfer Learning AST-MLP Head (pre-trained) (Gong et al., 2021)	230.1k	88.1M	Mel Spectrogram	Interjection
Best performer above based on F1 score	-	-	-	Interjection Prolongation Block Word Repetition Sound Repetition

The models proposed for this research methodology were developed using PyTorch. The features were generated using NumPy and SciPy signal processing libraries. Training was conducted on a MacBook M1 Pro.

CHAPTER 4 RESULTS AND DISCUSSIONS

This chapter examines and compares the performance of four stuttering detection models: a CNN, a LSTM model, a Conv-LSTM model, and the transfer learning AST model.

4.1 Preliminary Analysis Involving CNN and LSTM Architectures, covers the baseline models performance for interjection; **4.2 Class Weights – CNN and LSTM Architectures**, analyses the effect of class weights based on annotator agreement levels for interjection; **4.3 Combining Models to Create Conv-LSTM Model**, explores the performance benefits of combining the CNN and LSTM architectures; **4.4 AST Model Performance**, analyses the performance of the AST; **4.5 Comparison of Models Investigated**, compares each models performance at classifying the interjection disfluency; **4.6 AST's Performance Across Each Type of Stuttering Disfluency**, Investigates the AST's ability to classify all stuttering disfluency types; and **4.7 T-SNE Plots for AST Parameters** analyses the T-SNE plots before and after training.

The findings provide a broad view to the effectiveness of the AST model in detecting stuttering against common DL architectures like CNN, LSTM and Conv-LSTM.

4.1 Preliminary Analysis Involving CNN and LSTM Architectures for Interjection

This section presents findings from various tests conducted on the SEP-28K dataset which focuses on the performance of two custom designed architectures – a CNN and a LSTM model. Baseline models were developed to guide the optimisation of key hyperparameters. Special attention was given to the impact of class weights. Many tests were conducted to

establish these baselines (see **Appendix C – Key Model Results**), with the objectives here to accumulate DL experience and gain expertise in the critical components of SD.

4.1.1 Baseline CNN

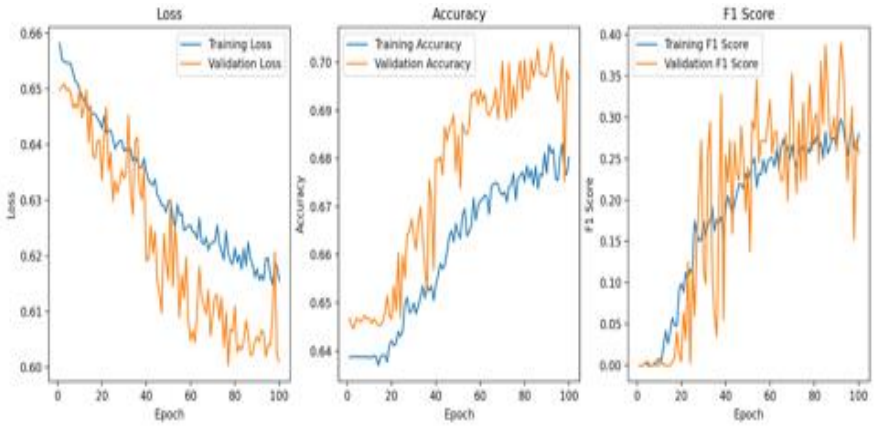


Figure 22: Performance of the baseline CNN model over 100 epochs, trained on the SEP-28K dataset for single-task detection of interjections with extracted MFCC's as input. Single annotator agreement was used here.

The baseline CNN presented a validation accuracy of 0.6962 or 69.62%, which appears promising at first glance. However, accuracy can be misleading when dealing with imbalanced datasets. As displayed in the confusion matrix below (see Figure 23), the model correctly classified 2,772 true negatives, or it predicted ‘no interjection’ correctly 2,772 times. But it also resulted in 1,274 false negatives, where the model failed to identify an ‘interjection’ and instead predicted ‘no interjection’. As the model is largely predicting the majority class, the accuracy becomes artificially inflated.

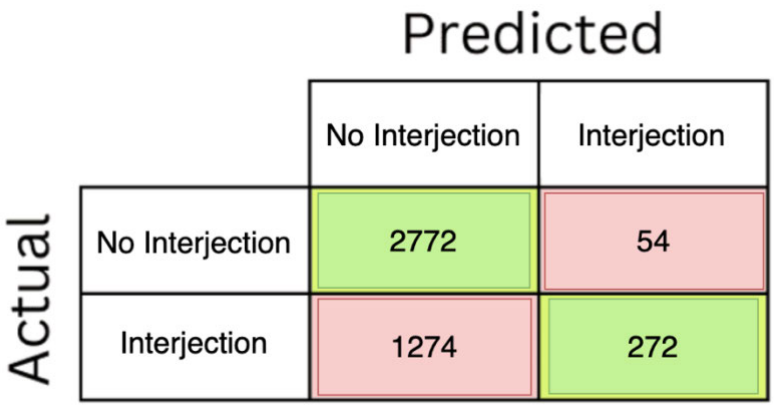


Figure 23: Baseline CNN validation confusion matrix for classifying interjections.

The validation F1 score of 0.2906 reveals the more concerning aspect of the model's performance. Due to the aforementioned, and the fact the SEP-28K dataset is imbalanced, the F1 score is the best measure for overall performance.

A low F1 score in this binary classification task means the CNN model struggles with precision and particularly recall indicated by many false negatives; suggestive of an inability to identify stuttering instances. The EER = 0.4216, which also implies the model is failing to detect stuttering or falsely identifying stuttering 42.16% of the time.

Upon analysing the loss curve, it can be observed the validation loss is lower than the training loss. This discrepancy may be explained by regularisation applied in training but not in validation. The use of a 0.3 dropout at each convolutional layer serves as a form of regularisation, intentionally sacrificing training accuracy to improve the model's generalisation. As a result, the learning process slows leading to a faster decrease in validation loss compared to training loss (Rosebrock, 2019). Ultimately, this CNN will be modified through hyperparameter optimisation with the aim to improve F1 score.

4.1.2 Baseline LSTM

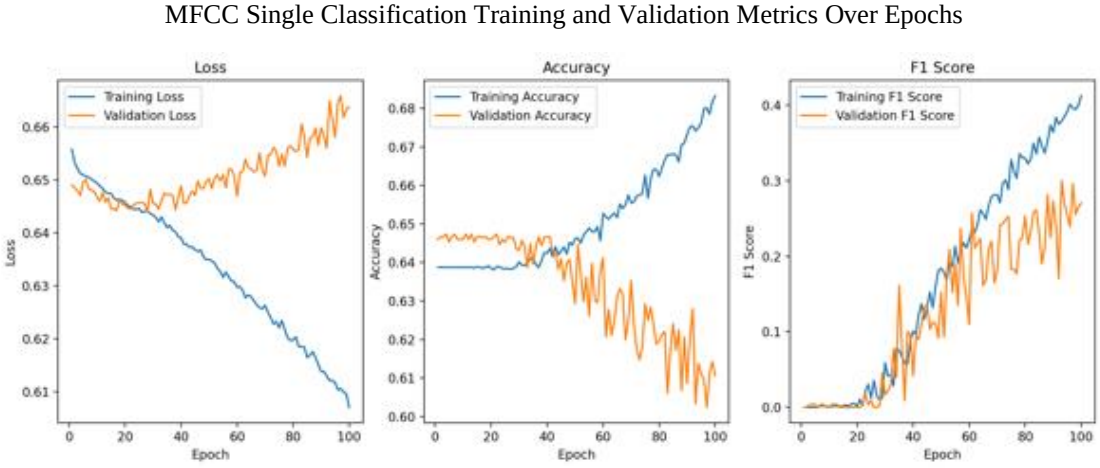


Figure 24: Performance of the designed LSTM model over 100 epochs, this was also trained on the SEP-28K dataset for single-task detection of interjections using extracted MFCC's as input. The results also reflect training based on the single annotator agreement.

The LSTM model's results at 100 epoch's for detecting interjections present accuracy = 0.6102, F1 = 0.2993, and EER = 0.4747. The performance closely matches the CNN model with F1 score slightly better with accuracy and EER slightly worse. The clear concern in observing the graphs is the increasing validation loss. During the early stages of training (first 30 epochs) both the training and validation loss are closely matched, however, after 30 epochs, the model begins too overfit. This continues to perform well on the training set as indicated by the decreasing training loss. However, it is failing to perform on the unseen validation data. As training progresses the model fits further into the training set, yet validation loss increases, ultimately indicating poor model generalisation. This could be caused by insufficient regularisation. Solutions may include penalising large weights using early stopping or increasing dropout rates. The model architecture may also be to complex; learning intricate patterns in the training data that are irrelevant to stuttering. This LSTM model will also be modified through model hyperparameter optimisation.

4.2 Class Weights – CNN And LSTM Models for Interjection

Cost sensitive learning applied to this training set required class weights of 0.783 for the majority class and 1.384 for the minority class. This was calculated based on the formulas described in methodology section 3.1.3. Such tuning aimed to make each class contribute equally to the loss by penalising misclassifications more heavily, thus reducing majority class bias. In theory this leads to better model performance. The CNN and LSTM models were tested for each annotator level utilising class weights.

However, due to the severe class imbalance within the SEP-28K dataset, such tuning was still unfavourable. These class weights did not put enough emphasis on the minority class. To account for this, the model's sensitivity to the minority class had to be further increased. By doing so, the model's become forced to pay more attention to the underrepresented samples. Various class weights were tested, and the best performing models exploited class weights corresponding to the inverse frequency of interjections within the entire dataset.

4.2.1 CNN Performance

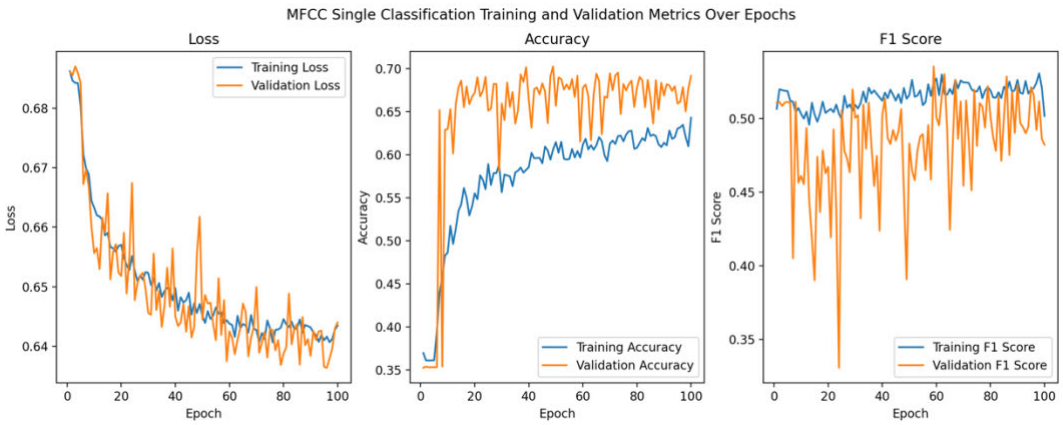


Figure 25: This figure displays the best F1 score CNN model for classifying interjections with extracted MFCC's from the SEP-28K dataset. This used single annotator agreement.

Figure 25. above shows validation accuracy over training accuracy. It is plausible to state this may be caused by an unbalanced training process. Specifically, if the training set was exposed to more challenging samples compared to the validation set, the model will struggle to learn this training data. A robust validation protocol would eliminate this possibility. However, the model utilising class weights of [0.7 majority, 1.73 minority] did see a significant performance improvement. Such class weighting achieved an F1 score of 0.5153, showcasing an increase of 77.32% above baseline F1.

Below are the results for the annotator agreement levels, and contrary to Mohapatra et al., (2022) work, the single annotator agreement achieved the highest F1 score.

Table 6: The CNN classification results on the SEP-28K dataset. Each annotator agreement level used different class weights, based on the inverse frequency with respect to the entire dataset.

Interjection Results with CNN				
	Class Weight Balance [majority/minority]	**Accuracy**	**F1 Score**	**EER**
Baseline CNN	-	0.6962	0.2906	0.4216
Single Agreement	[0.7, 1.73]	0.6775	0.5153	0.366
Majority Agreement	[0.61, 2.77]	0.7738	0.5092	0.31405
Unanimous Agreement	[0.56, 4.86]	0.7717	0.3707	0.3332

4.2.2 LSTM Performance

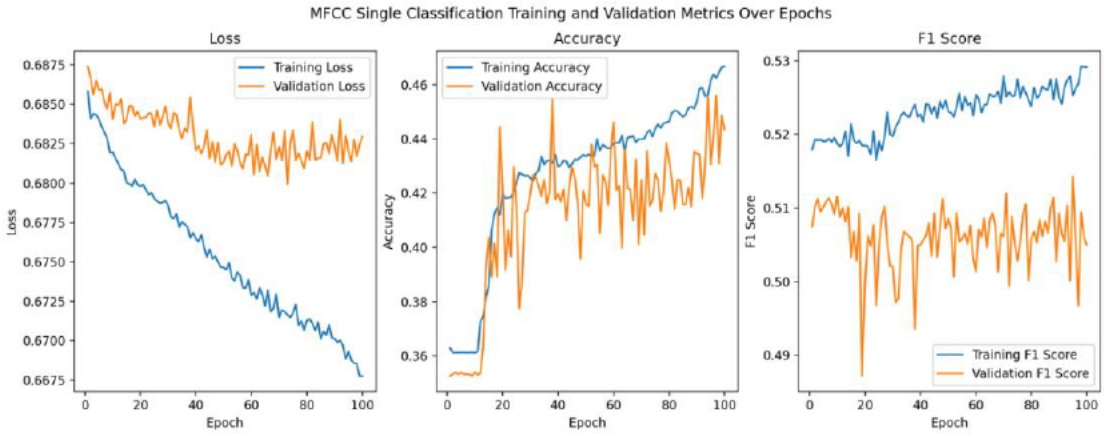


Figure 26: The best performing LSTM model based on F1 score is shown, this was also achieved using single annotator agreement.

A similar result was noticed for the LSTM model, with the single annotator agreement level using class weights of [0.7 majority, 1.73 minority] achieved the highest F1 score of 0.5253. Resulting in an increase of 75.51% above the baseline result.

Table 7: The LSTM model classification results on the SEP-28K dataset. Each annotator agreement level used different class weights, based on the inverse frequency with respect to the entire dataset.

Interjection Results with LSTM				
	Class Weight Balance [majority/minority]	**Accuracy**	**F1 Score**	**EER**
Baseline LSTM	-	0.6102	0.2993	0.4747
Single Agreement	[0.7, 1.73]	0.4296	0.5253	0.4556
Majority Agreement	[0.61, 2.77]	0.4195	0.373	0.4529

Unanimous Agreement	[0.56, 4.86]	0.6109	0.2208	0.4658

4.2.3 CNN and LSTM Comparison

Upon comparing single annotator agreement levels, the LSTM achieved a 1.9% improvement over the CNN. The CNN model produced better than baseline F1 scores for all three annotator levels. While the LSTM produced stronger results than the baseline F1 score for two annotator levels, and lower F1 score for unanimous agreement.

LSTMs are inherently more complex due to their ability to maintain cell states across time steps. The lower F1 score could be attributed to their sensitivity to severe class imbalances requiring more class samples to extract meaningful temporal patterns. In contrast, the CNN can still extract well-defined spatial features despite having limited samples.

4.3 Combining Models to Create Conv-LSTM Architecture

As the CNN and LSTM both obtained the highest F1 score on single annotator agreement. The model architectures were combined to evaluate the effectiveness of the Conv-LSTM for SD. This common architecture aims to combine CNN's effective spatial pattern detection with LSTM's ability to remember relevant temporal information.

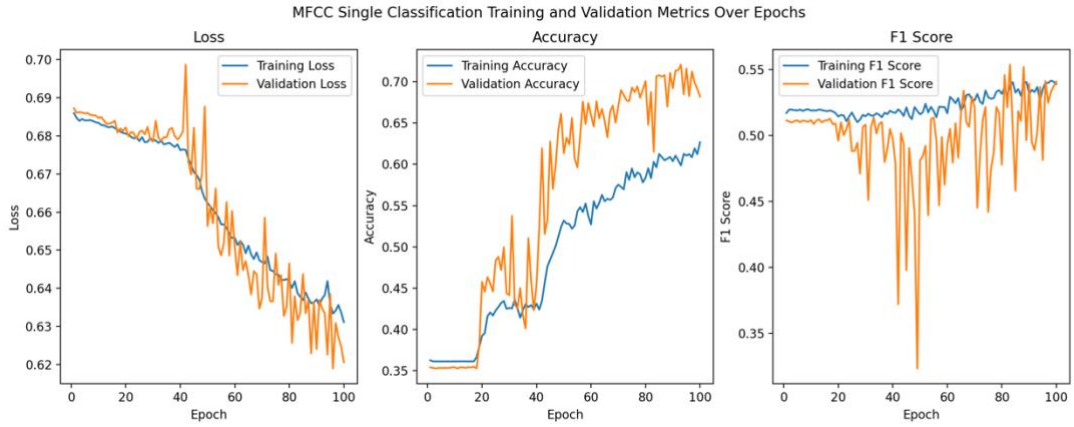


Figure 27: The performance of the hybrid architecture Conv-LSTM model using single annotator agreement with class weights [0.7, 1.73] on the SEP-28K dataset.

This model did increase the trainable parameters to 68.78k which caused a significant increase in computation time. However, the Conv-LSTM model achieved further advancements in classifying interjections as seen in Figure 27. The best epoch returned an accuracy of 66.95%, an EER = 34.12% and importantly an F1 score of 0.5711. A further advancement of 8.72% on the LSTM model, and 96.52% above baseline results.

By adjusting and analysing the performance of these common model architectures within the SD field, a foundation was laid. Such knowledge was then transferred to test a novel approach. In the next section, the pre-trained AST is evaluated to see its potential for SD.

4.4 Pre-Trained AST Model Results

This section presents the outcomes from the novel AST model results for classifying interjection disfluencies on the SEP-28K dataset. For the AST-MLP training, single annotator agreement was used along with the same class weights mentioned above. Even though the 12 transformer layers were frozen due to the output feature embeddings having high dimensionality, the MLP head was larger than previous architectures. This caused a substantial increase in trainable parameters to 230.1k, resulting in significantly more computations.

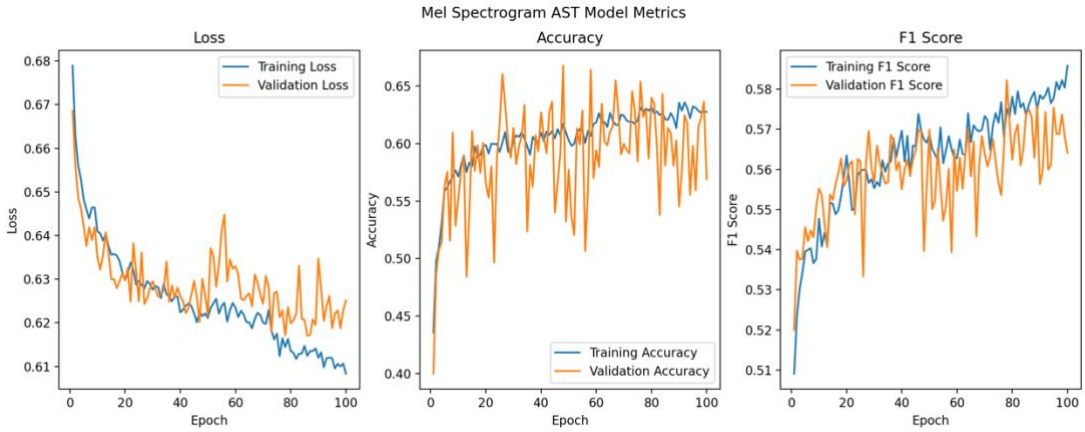


Figure 28: The AST-MLP model results for interjection classification. Single annotator agreement with class weights [0.7, 1.73] was used. Mel spectrograms from the SEP-28K dataset was the input.

The AST model achieved an accuracy of 62.88%, an EER of 33.89% and an F1 score of 0.5952. Therefore, this achieved a further 4.22% advancement on the Conv-LSTM model and 104.82% increase over baseline CNN F1 score.

By harnessing a large pre-trained transformer like the AST, which has in-built knowledge from the diverse AudioSet dataset – including music, environmental sounds, and a wide demographic of human speech (Gemmeke et al., 2017). The AST parameters can extract

high-level features that enable the MLP to learn nuanced characteristics of disfluent speech. This approach has resulted in impressive performance for classifying interjections.

4.5 Comparison of Model Performance for Interjection

Table 8: The performance of each model architecture using class weights [0.7, 1.73] with single annotator agreement.

Interjection Binary Classification Performance			
	Accuracy	**F1 Score**	**EER**
CNN	0.6755	0.5153	0.366
LSTM	0.4296	0.5253	0.4556
Conv-LSTM	0.6695	0.5711	0.3412
AST	0.6288	0.5952	0.3389

Table 8. highlights that although the CNN attained the highest accuracy largely due to consistently predicting the majority class, it achieved the lowest F1 score of 0.5153. The LSTM model saw an advancement of 1.9% in F1 score to 0.5253. Additionally, the Conv-LSTM model saw an increase of 10.83% in F1 score over the CNN to 0.5711. The AST achieved the highest F1 score for interjection classification, and the lowest equal error rate. It saw a 15.53% advancement in F1 score over the CNN to 0.5952.

For this given validation protocol, these results reveal the AST is superior for stuttering detection compared to CNN, LSTM, and Conv-LSTM models. It is observed that the pretrained model generalised to stuttering specific features. A large performance increase was observed utilising a shallow MLP network. Due to the AST presenting with the highest F1 score, the model was used to evaluate all disfluency types. In conclusion, pretrained transformers such as AST offer more robust features that can generalise to novel tasks.

4.6 AST's Performance Across Each Type of Stuttering Disfluency

Table 9: This table shows the performance of the AST-MLP model using mel spectrograms as input from the SEP-28K dataset. Each test was a binary classification task for each disfluency, using single annotator agreement and class weights. The same 80/20 training/validation split was used.

AST Binary Classification Performance			
	Accuracy	**F1 Score**	**EER**
Interjection	0.6288	0.5952	0.3389
Prolongation	0.6500	0.5721	0.3245
Block	0.5492	0.6304	0.4167
Word Repetition	0.6340	0.3789	0.3436
Sound Repetition	0.7354	0.5091	0.3160
Average	0.6395	0.5370	0.3479

As the class weights used for interjection were based on the inverse frequency within the entire dataset, the same method was employed to determine class weights for each disfluency. It can be seen from the results; ‘Block’ disfluency achieved the highest F1 score of 0.6304, while ‘Word Repetition’ achieved the lowest F1 score of 0.3789.

This could be attributed to several reasons. Firstly, for the training validation split, Block had the most samples equal to 7417, while Word Repetition had the lowest with 3164. This means that the AST-MLP model had more than twice the instances to learn stuttering specific features. Blocks have traditionally been the most difficult to detect in the literature due to involuntary, silent stoppages that can mimic natural pauses (Lea et al., 2021). Such results highlight the potential benefit AST can bring to SD.

Sound Repetition had only 3803 samples and achieved an F1 score of 0.5091. This suggests that the AST model struggled to extract high-level features specific to word repetitions. Given the conversational nature of the SEP-28K dataset, fluent speech typically involves natural word repetitions rather than sound repetitions. This overlap could add difficulty to distinguishing stuttering-related word repetitions, ultimately contributing to the low F1 score.

This section has illustrated the promising results of the AST model for SD. Through achieving an average F1 score of 0.5370 across disfluencies results can be broadly compared to the state-of-the-art literature.

4.7 Performance of Models from The Literature

The results collated below are not directly comparable yet provide a rough overview to the AST’s capability to advance SD. There are broad comparisons to similar constructions,

however it is difficult to draw accurate contrasts due to dataset reliability, and the training/validation split used.

Results from Filipowicz & Kostek (2023) which used solely MFCC's from the SEP-28K dataset in an 80%/10%/10% (train/validation/test) split obtained an average F1 score of 0.3402 with a ResNetBiLstm architecture, and 0.2542 with ResNet18 architecture across the five disfluencies. It is important to note, researchers used 10-fold cross validation which is known to be more robust. Subsquently, Filipowicz & Kostek implemented multiple input features and achieved an average F1 score of roughly 0.4500 (from estimating graph values). The pre-trained Wav2Vec2.0 transformer as a feature extractor was then tested using raw audio data as the input and achieved an average F1 score of 0.3300 (from estimating graph values).

StutterNet's optimised model by Sheikh et al., (2021) used MFCC's through a time delay neural network which was tested across only three disfluency types on the UCLASS dataset. They achieved an average F1 score of 0.2967. Later in the Sheikh et al., (2023) paper, a model called MC StutterNet trained and evaluated on the SEP-28K dataset, achieving an average F1 score of .4386. Additionally, 10-fold cross validation was used here.

Lea et al., (2021) used an 89.3%/7.1%/3.6% train/validation/test split on the SEP-28K dataset. Their LSTM model used mel-filterbank energy features as the input, achieving an average F1 score of .6062. Their Conv-LSTM model with multiple input features achieved .6386. Al-Banna et al., (2024) single feature MFCC model achieved an F1 score 0.6641 on the SEP-28K dataset, and the improved hybrid model achieved an impressive average F1 score of 0.7444.

Some other notable complex architectures include the transformer 'Whisper' model by Ameer et al., (2023). This used a transformer-based encoder model using log-mel

spectrograms after being processed through convolutional layers. This in-depth model was trained on a dominant speaker subset of the SEP-28K which was then evaluated on FluencyBank, achieving an average F1 score of 0.7520.

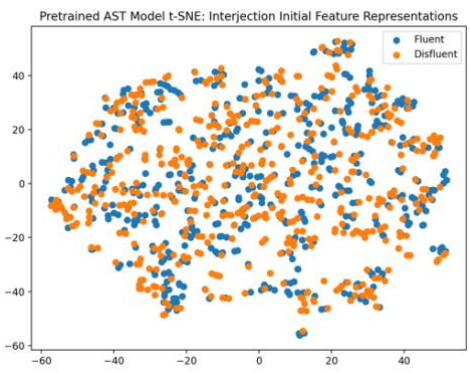
Mohapatra et al., (2022) DisfluencyNet, a more complex model achieved the highest average F1 score seen of 0.8640 across the five disfluencies. This utilised Wav2Vec2.0 raw audio embeddings from the SEP-28K dataset which were passed through convolutional layers, and an MLP head thereafter.

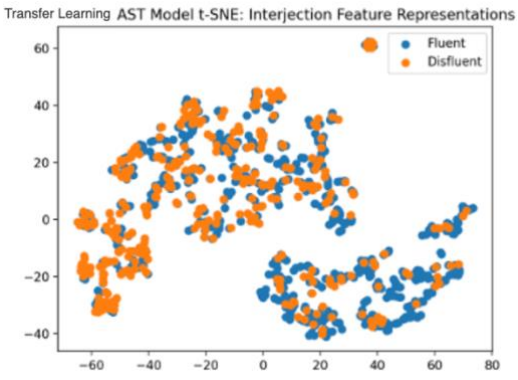
Analysing the results in the existing literature reveals the significant potential of the AST for SD.

4.8 T-SNE Plots to Visualise AST Parameters

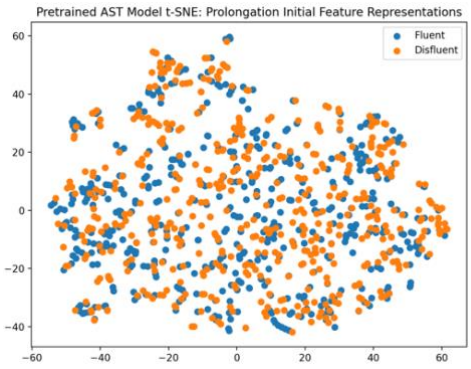
T-Distributed Stochastic Neighbour Embedding plots or T-SNE plots is a visualisation technique that gives high-dimensional data a location in a 2-dimensional space. This is widely used across multiple fields in machine learning, and was utilised in Sheikh et al., (2021) work in SD. These plots reveal the AST's capability to classify disfluencies; visualising the latent codes highlights the model's effectiveness in distinguishing disfluent audio samples from fluent ones.

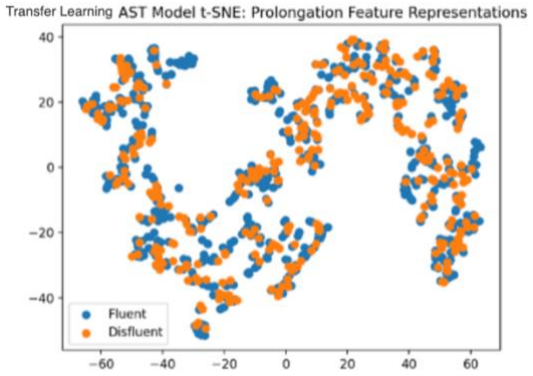
a)



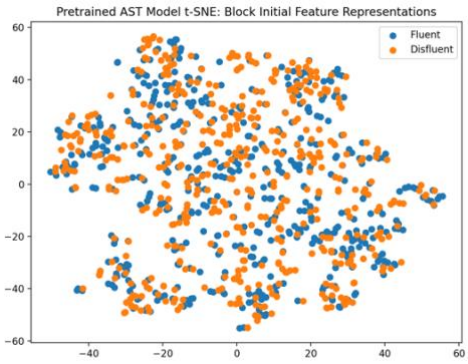


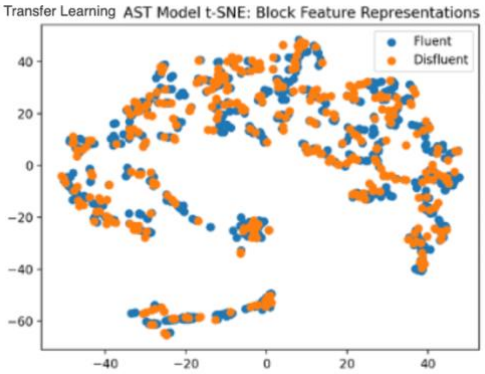
b)


Pretrained AST Model t-SNE: Prolongation Initial Feature Representations
Fluent
Disfluent

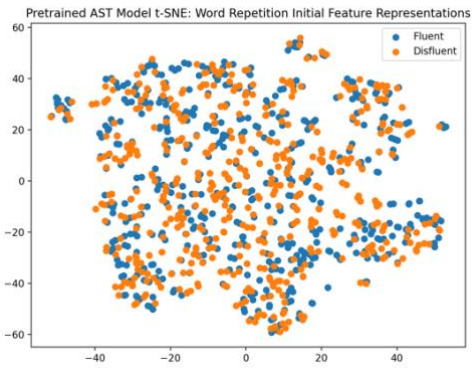

Transfer Learning AST Model t-SNE: Prolongation Feature Representations
Fluent
Disfluent

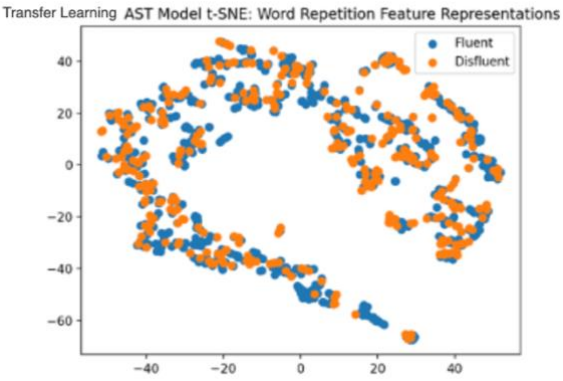
c)


Pretrained AST Model t-SNE: Block Initial Feature Representations
Fluent
Disfluent


Transfer Learning AST Model t-SNE: Block Feature Representations
Fluent
Disfluent

d)


Pretrained AST Model t-SNE: Word Repetition Initial Feature Representations
Fluent
Disfluent


Transfer Learning AST Model t-SNE: Word Repetition Feature Representations
Fluent
Disfluent

e)

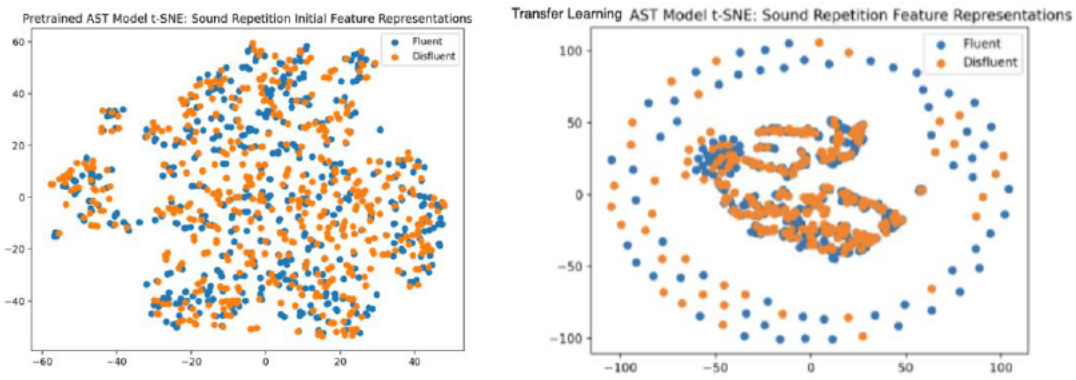


Figure 29: T-SNE plots for each disfluent class against the fluent class. Features are taken from the last MLP layer before and after training. a) Interjection. b) Prolongation. c) Block. d) Word Repetition. e) Sound Repetition.

The initial features representations which is extracted from the final MLP layer before training appear random, with no grouping or patterns differentiating disfluent and fluent samples. After training, Prolongation, Block and Word Repetition parameters show signs of separation, although the groupings remain weak and visually dispersed. In contrast, Sound Repetition exhibits obvious clustering, though the reliability of these clustering is questionable, potentially caused by incorrect T-SNE implementation.

However, the interjection parameters show strong and well-separated clusters between fluent and disfluent samples. This suggests the AST model effectively learns patterns and can distinctly separate interjection and non-interjection features.

CHAPTER 5 CONCLUSIONS

In this study, the AST has been introduced as a potential pretrained deep learning model for solving SD tasks. It was compared with CNN, LSTM, Conv-LSTM architectures. A comprehensive literature review in datasets, preprocessing techniques, features extraction, methodologies, and key hyperparameters opened the door to DL. By exploring these model components through coding, this formed stronger understandings and the ability to experiment. Thus, helping find novel approaches to SD.

The SEP-28K dataset, despite being comprehensive, posed significant challenges due to 20% of files missing from domain pages when downloaded. Secondly, class imbalances and annotator agreement inconsistencies required additional considerations. Effort was made to address such issues with hyperparameter optimisation and class weight adjustments. Single annotator agreement was found to yield the strongest F1 scores, along with class weights. This was based on the inverse frequency of the disfluent class compared to the entire dataset.

The introduction of the AST model with a trainable MLP head presented notable advancements on traditional architectures such as CNN, LSTM and Conv-LSTM. Specifically, the LSTM model outperformed the CNN, achieving a 1.9% increase in F1 score. Additionally, the combined Conv-LSTM further improved performance with an 8.72% higher F1 score. Finally, the AST model demonstrated an additional 4.22% advancement in F1 score, ultimately establishing the highest F1 score for interjection at 0.5952.

The AST was then used to evaluate all five disfluencies, achieving an average F1 score of 0.5370. It is important to note, while this performance is competitive with existing literature, the results are not directly comparable due to the differences in train/validation protocols.

In additional to the quantitative analysis, the T-SNE visualisations of latent codes from the MLP head indicated clear separation, particularly for the interjection class. Thus, these results support the ability of pretrained audio transformers like the AST to offer robust representations for audio tasks. Consequently, the AST has been identified as a promising avenue, showcasing its potential for SD applications.

Additional investigations leveraging the AST within a hybrid architecture should be examined. Specifically, by taking the output feature vectors and passing them to a CNN could combine the strengths of both architectures.

In this research a frozen encoder was utilised, by adapting the initial or later layers through transfer learning. This technique could be a suitable avenue to fully explore the AST model's capabilities. Fine-tuning these layers could enable the model to capture unique disfluency patterns to which generic pre-trained representations might overlook, thus ultimately enhancing performance.

Moreover, adopting more dependable evaluation techniques, like k-fold cross-validation would provide a comprehensive assessment of model generalisability.

Further advancements are still required in the field of SD to ensure that SP's can reliably adopt these models in real-time applications. Future systems leveraging this technology have the potential to streamline stuttering assessments and provide digital therapy avenues, benefiting both patients and SPs.

REFERENCES

Abubakar, M., Mujahid, M., Kanwal, K., Iqbal, S., Asghar, N., & Alaulamie, A. (2024). StutterNet: Stuttering Disfluencies Detection in Synthetic Speech Signals via Mel Frequency Cepstral Coefficients Features using Deep Learning. *IEEE Access*.

Alnashwan, R., Alhakbani, N., Al-Nafjan, A., Almudhi, A., & Al-Nuwaiser, W. (2023). Computational intelligence-based stuttering detection: A systematic review. *Diagnostics (Basel)*, *13*(23), 3537. https://doi.org/10.3390/diagnostics13233537

Alzubaidi, L., Zhang, J., Humaidi, A. J., Al-Dujaili, A., Duan, Y., Al-Shamma, O., ... & Farhan, L. (2021). Review of deep learning: Concepts, CNN architectures, challenges, applications, future directions. *Journal of Big Data*, *8*(1), 53. https://doi.org/10.1186/s40537-021-00444-8

Ameer, H., Latif, S., Latif, R., & Mukhtar, S. (2023). Whisper in Focus: Enhancing Stuttered Speech Classification with Encoder Layer Optimization. arXiv. https://arxiv.org/abs/2311.05203

Arora, K. (2024, May 10). Everything You Must Know About Data Normalization in Machine Learning. *Markov ML*. Retrieved September 15, 2024, from https://www.markovml.com/blog/normalization-in-machine-learning

Author unknown. (n.d.). NDIS Review Submission [PDF]. Retrieved from https://www.ndisreview.gov.au/sites/default/files/submissions/SUB-W0H8-000601%20-%201_clean.pdf

Bäckström, T., Räsänen, O., Zewoudie, A., Pérez Zarazaga, P., Koivusalo, L., Das, S., Gómez Mellado, E., Bouafif Mansali, M., & Ramos, D. (n.d.). 3.8. The cepstrum, mel-

cepstrum and mel-frequency cepstral coefficients (MFCCs). In *Introduction to Speech Processing*. Retrieved April 9, 2024, from https://speechprocessingbook.aalto.fi/Representations/Melcepstrum.html?highlight=mfcc

Bader-El-Den, M., Teitei, E., & Adda, M. (2016). Hierarchical classification for dealing with the class imbalance problem. In *Proceedings of the 2016 International Joint Conference on Neural Networks (IJCNN)* (pp. 3584–3591).

Badshah, A. M., Rahim, N., Ullah, N., Ahmad, J., Muhammad, K., Lee, M. Y., ... & Baik, S. W. (2019). Deep features-based speech emotion recognition for smart affective services. *Multimedia Tools and Applications, 78*(5), 5571-5589. https://doi.org/10.1007/s11042-018-6465-1

Baevski, A., Zhou, Y., Mohamed, A., & Auli, M. (2020). wav2vec 2.0: A framework for self-supervised learning of speech representations. *Advances in neural information processing systems, 33*, 12449-12460.

Bayerl, S. P., Roccabruna, G., Chowdhury, S. A., Ciulli, T., Danieli, M., Riedhammer, K., & Riccardi, G. (2022, September). What can Speech and Language Tell us About the Working Alliance in Psychotherapy. In *Interspeech 2022* (Vol. 2022, No. 347). ISCA. https://doi.org/10.21437/interspeech.2022-347

Bayerl, S. P., Wagner, D., Nöth, E., & Riedhammer, K. (2022). Detecting dysfluencies in stuttering therapy using wav2vec 2.0. *arXiv preprint arXiv:2204.03417*.

Blomgren, M., Roy, N., Callister, T., & Merrill, R.M. (2005). Intensive Stuttering Modification Therapy. *Journal of Speech, Language and Hearing Research, 48* (3), 509-523. https://doi.org/10.1044/1092-4388(2005/035)

Brignell, A., Krahe, M., Downes, M., Kefalianos, E., Reilly, S., & Morgan, A. T. (2020). A systematic review of interventions for adults who stutter. *Journal of Fluency Disorders, 64*, 105766. https://doi.org/10.1016/j.jfludis.2020.105766

Brownlee, J. (2020, September 12). Understand the Impact of Learning Rate on Neural Network Performance. Machine Learning Mastery. Retrieved April 9, 2024, from https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/

Chandola, Y., Virmani, J., Bhadauria, H.S., & Kumar, P. (2021). End-to-end pre-trained CNN-based computer-aided classification system design for chest radiographs. In *Deep Learning for Chest Radiographs* (pp. 117-140). Academic Press. https://doi.org/10.1016/B978-0-323-90184-0.00011-4

Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research, 16*, 321–357. https://doi.org/10.1613/jair.953

Davis, S. B., & Mermelstein, P. (1980). Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE Transactions on Acoustics, Speech, and Signal Processing, 28*(4), 357-366. https://doi.org/10.1109/TASSP.1980.1163420

Fayek, H. (2016). Speech processing for machine learning: Filter banks, Mel-frequency cepstral coefficients (MFCCs) and what's in-between. Retrieved from https://haythamfayek.com/2016/04/21/speech-processing-for-machine-learning.html

Fernandez, A., Garcia, S., Galar, M., Prati, R. C., Krawczyk, B., & Herrera, F. (2018). Learning from Imbalanced Data Sets (Vol. 10). Springer.

Filipowicz, P., & Kostek, B. (2023). Rediscovering automatic detection of stuttering and its subclasses through machine learning—The impact of changing deep model architecture and amount of data in the training set. *Applied Sciences, 13*(10), 6192. https://doi.org/10.3390/app13106192

Gemmeke, J. F., Ellis, D. P. W., Freedman, D., Jansen, A., Lawrence, W., Moore, R. C., & Plakal, M. (2017). Audio set: An ontology and human-labeled dataset for audio events. *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 776–780. https://doi.org/10.1109/ICASSP.2017.7952261

Gong, Y., Chung, Y. A., & Glass, J. (2021). Ast: Audio spectrogram transformer. *arXiv preprint arXiv:2104.01778*.

Graves, A., Jaitly, N., & Mohamed, A. (2013). Hybrid speech recognition with deep bidirectional LSTM. Proceedings of the IEEE Workshop on Automatic Speech Recognition and Understanding, 273-278

Gts.AI. (2023, March 23). From Noise to Knowledge: Curating an Audio Dataset for Machine Learning. *Medium*. Retrieved September 15, 2024, from https://medium.com/@Gts.AI/from-noise-to-knowledge-curating-an-audio-dataset-for-machine-learning-56a017a3393d

Gupta, S., Shukla, R.S., Shukla, R.K., & Verma, R. (2020). Deep Learning Bidirectional LSTM based Detection of Prolongation and Repetition in Stuttered Speech using Weighted MFCC. International Journal of Advanced Computer Science and Applications, 11. https://doi.org/10.14569/IJACSA.2020.0110941

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778)

Howell, P. (2007). Signs of developmental stuttering up to age eight and at 12 plus. *Clinical Psychology Review, 27*(3), 287-306. https://doi.org/10.1016/j.cpr.2006.08.005

Howell, P., Davis, S., & Bartrip, J. (2009). The UCLASS Archive of Stuttered Speech. *Journal of Speech, Language, and Hearing Research*, 52, 556-569. https://doi.org/10.1044/1092-4388(07-0129)

Jeon, H., Jung, Y., Lee, S., & Jung, Y. (2020). Area-Efficient Short-Time Fourier Transform Processor for Time–Frequency Analysis of Non-Stationary Signals. *Applied Sciences*, 10(20), 7208. https://doi.org/10.3390/app10207208

Jordan, J. (2018, March 1). Setting the learning rate of your neural network. Jeremy Jordan. https://www.jeremyjordan.me/nn-learning-rate/

Jouaiti, M., & Dautenhahn, K. (2022). Dysfluency classification in stuttered speech using deep learning for real-time applications. In *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (pp. 6482-6486). IEEE. https://doi.org/10.1109/ICASSP43922.2022.9746638

Kamaldeep. (2023, July 6). How to Improve Class Imbalance using Class Weights in Machine Learning. *Analytics Vidhya.* Retrieved from https://www.analyticsvidhya.com/blog/2020/10/improve-class-imbalance-class-weights/

Ko, B., Kim, H.-G., Oh, K.-J., & Choi, H.-J. (2017). Controlled dropout: A different approach to using dropout on deep neural network. *2017 IEEE International Conference on*

Big Data and Smart Computing (BigComp), 358-362. https://doi.org/10.1109/BIGCOMP.2017.7881693

Kourkounakis, T., Hajavi, A., & Etemad, A. (2021). FluentNet: End-to-End Detection of Stuttered Speech Disfluencies With Deep Learning. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 29, 2986–2999.

Kourkounakis, T.; Hajavi, A.; Etemad, A. Detecting Multiple Speech Disfluencies Using a Deep Residual Network with Bidirectional Long Short-Term Memory; IEEE: Barcelona, Spain, 2020; p. 6093

Krishna, R., Patel, R., & Sahu, V. (2021). A comparative study on stuttering detection using deep learning approaches. *International Journal of Speech and Language Processing*, *23*(4), 234-240.

Latif, S., Zaidi, A., Cuayahuitl, H., Shamshad, F., Shoukat, M., & Qadir, J. (2023). Transformers in speech processing: A survey. *arXiv preprint arXiv:2303.11607*.

Lea, C., Mitra, V., Joshi, A., Kajarekar, S., & Bigham, J.P. (2021). SEP-28k: A Dataset for Stuttering Event Detection from Podcasts with People Who Stutter. In *Proceedings of the ICASSP 2021—2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (pp. 6798-6802). Toronto, ON, Canada

LeCun, Y.A., Bottou, L., Orr, G.B., & M¨uller, K.R. (2012). Efficient backprop. In Neural networks: Tricks of the trade (pp. 9–48). Springer

Leonel, J. (2019, April). Hyperparameters in Machine/Deep Learning. Medium. Retrieved from https://medium.com/@jorgesleonel/hyperparameters-in-machine-deep-learning-ca69ad10b981

Lewis, C. (2024, June). Cleaning and organizing data. *Data Management in Educational Research*. Retrieved September 15, from https://datamgmtinedresearch.com/clean

Maguire, G. A., Nguyen, D. L., Simonson, K. C., & Kurz, T. L. (2020). The pharmacologic treatment of stuttering and its neuropharmacologic basis. Frontiers in Neuroscience, 14, 158. https://doi.org/10.3389/fnins.2020.00158

Mihalache, S., & Burileanu, D. (2022). Using voice activity detection and deep neural networks with hybrid speech feature extraction for deceptive speech detection. *Sensors*, *22*(3), 1228.

Mohapatra, P., Pandey, A., Islam, B., & Zhu, Q. (2022). Speech disfluency detection with contextual representation and data distillation. In *Proceedings of the 1st ACM International Workshop on Intelligent Acoustic Systems and Applications* (pp. 19–24). Association for Computing Machinery. https://doi.org/10.1145/3539490.3539601

Moore, B. C. (2012). *An introduction to the psychology of hearing*. Brill.

National Disability Insurance Agency. (2024, March 26). Pricing arrangements. NDIS. https://www.ndis.gov.au/providers/pricing-arrangements

Nguyen, T. S., Sperber, M., Stüker, S., & Waibel, A. (Year). Building Real-time Speech Recognition without CMVN. Institute for Anthropomatics and Robotics, Karlsruhe Institute of Technology. Retrieved April 10, 2024, from https://isl.anthropomatik.kit.edu/pdf/Nguyen2018a.pdf

O'Brian, S., Carey, B., Lowe, R., Onslow, M., Packman, A., & Cream, A. (2018). *The Camperdown Program Stuttering Treatment Guide.*

https://www.uts.edu.au/sites/default/files/2018-10/Camperdown%20Program%20Treatment%20Guide%20June%202018.pdf

Ong, K. L., Lee, C. P., Lim, H. S., Lim, K. M., & Alqahtani, A. (2023). Mel-MViTv2: Enhanced Speech Emotion Recognition With Mel Spectrogram and Improved Multiscale Vision Transformers. *IEEE Access*.

Onslow, M., (2000) *Stuttering Treatment for Adults*. South Western Sydney Local Health District.https://www.swslhd.health.nsw.gov.au/bankstown/stuttering/pdf/ResourcesStutteringAdults.pdf

Onslow, M., Harrison, E., Jones, M., & Packman, A. (2002). Beyond-clinic speech measures during the Lidcombe Program of early stuttering intervention. ACQuiring Knowledge in Speech, Language and Hearing, 4, 82–85.

Onslow, M., Packman, A., & Harrison, E. (Eds.). (2017). *The Lidcombe Program of early stuttering intervention: A clinician's guide*. Pro-Ed. https://www.uts.edu.au/sites/default/files/2018-10/Lidcombe%20Program%20Treatment%20Guide%20December%202017.pdf

Rosebrock, A. (2019, October 14). Why is my validation loss lower than my training loss? *PyImageSearch*. https://pyimagesearch.com/2019/10/14/why-is-my-validation-loss-lower-than-my-training-loss/

Ryan, B., & Becka, A. (2024, January 25). Speech therapists answer questions about the Lidcombe Program. SpeechFit. https://speechfit.io/resources/overview-lidcombe-program-stuttering

Sahidullah, M., & Saha, G. (2012). A novel windowing technique for efficient computation of MFCC for speaker recognition. *IEEE Signal Processing Letters*, 20. https://doi.org/10.1109/LSP.2012.2235067

scikit-learn. (2007-2024). Cross-validation: evaluating estimator performance. Retrieved May 23, 2024, from https://scikit-learn.org/stable/modules/cross_validation.html

Setitra, M. A., Fan, M., Agbley, B. L. Y., & Bensalem, Z. E. A. (2023). Optimized MLP-CNN model to enhance detecting DDoS attacks in SDN environment. *Network, 3*(4), 538-562.

Shaip. (2024). What Is Data Annotation 2024? (Best Tools, Types, Challenges, Trends). Retrieved April 3, 2024, from https://www.shaip.com/blog/the-a-to-z-of-data-annotation/#:~:text=Importance%20of%20data%20annotation%20in%20machine%20learning&text=Data%20annotation%2C%20or%20labeling%2C%20is,digital%20neurons%20organized%20in%20layers.

Shah, D. (2023). *Cross Entropy Loss: Intro, Applications, Code.* Retrieved May 24, 2024, from https://www.v7labs.com/blog/cross-entropy-loss-guide

Sharma, N. (2023, June 06). *Understanding and Applying F1 Score: AI Evaluation Essentials with Hands-On Coding Example*. Arize. https://arize.com/blog-course/f1-score/

Sheikh, S. A., Sahidullah, M., Hirsch, F., & Ouni, S. (2021, August). Stutternet: Stuttering detection using time delay neural network. In *2021 29th European Signal Processing Conference (EUSIPCO)* (pp. 426-430). IEEE.

Sheikh, S. A., Sahidullah, M., Hirsch, F., & Ouni, S. (2022a). Machine learning for stuttering identification: Review, challenges and future directions. *Neurocomputing*, *514*, 385-402. https://doi.org/10.1016/j.neucom.2022.10.015

Sheikh, S. A., Sahidullah, M., Hirsch, F., & Ouni, S. (2022b, August). Robust stuttering detection via multi-task and adversarial learning. In *2022 30th European Signal Processing Conference (EUSIPCO)* (pp. 190-194). IEEE.

Sheikh, S. A., Sahidullah, M., Hirsch, F., & Ouni, S. (2022c). Introducing ECAPA-TDNN and Wav2Vec2. 0 embeddings to stuttering detection. *arXiv preprint arXiv:2204.01564*.

Sheikh, S. A., Sahidullah, M., Hirsch, F., & Ouni, S. (2023a). Stuttering detection using speaker representations and self-supervised contextual embeddings. *International Journal of Speech Technology*, *26*(2), 521-530.

Sheikh, S. A., Sahidullah, M., Hirsch, F., & Ouni, S. (2023b). Advancing Stuttering Detection via Data Augmentation, Class-Balanced Loss and Multi-Contextual Deep Learning. *IEEE Journal of Biomedical and Health Informatics*, *27*, 2553-2564. Available at https://api.semanticscholar.org/CorpusID:257079207

Singh, P., Singh, N., Singh, K. K., & Singh, A. (2021). Diagnosing of disease using machine learning. In *Machine learning and the internet of medical things in healthcare* (pp. 89-111). Academic Press.

Singh, T. (2019, June 15). *MFCCs Made Easy*. Medium. https://medium.com/@tanveer9812/mfccs-made-easy-7ef383006040

Smith, A., & Webera, C. (2017). How stuttering develops: The multifactorial dynamic pathways theory. *Journal of Speech, Language, and Hearing Research*, 60(9), [2483-2505]. https://pubs.asha.org/doi/10.1044/2017_JSLHR-S-16-0343

Snyder, D., Chen, G., & Povey, D. (2015). MUSAN: A Music, Speech, and Noise Corpus. *arXiv*. https://arxiv.org/abs/1510.08484

Sommer, M., Waltersbacher, A., Schlotmann, A., & Schröder, H. (2021). Prevalence and Therapy Rates for Stuttering, Cluttering, and Developmental Disorders of Speech and Language: Evaluation of German Health Insurance Data. Frontiers in Human Neuroscience, 15, 645292. https://doi.org/10.3389/fnhum.2021.645292

Wang, X., & Paliwal, K. K. (2003). Feature extraction and dimensionality reduction algorithms and their applications in vowel recognition. *Pattern recognition*, *36*(10), 2429-2439.

Wong, W. (2021, December 19). What Is Residual Connection? Towards Data Science. https://towardsdatascience.com/what-is-residual-connection-efb07cab0d55

Yairi, E., & Ambrose, N. (2013). Epidemiology of stuttering: 21st century advances. *Journal of Fluency Disorders, 38* (2), 66-87. https://doi.org/10.1016/j.jfludis.2012.11.002

APPENDIX A – PROJECT SPECIFICATION AND WORK PLAN

Introduction and Background:

While the exact cause of stuttering is still unknown, and there is no known cure, therapy has been shown to significantly decrease severity. If therapy stops however, the reductions in stuttering behaviours are not durable (Blomgren et al., 2005). Stuttering is prevalent in the lives of approximately 1% of the general population. Notably a higher incidence is seen in younger children of which roughly 80% recover naturally or with therapy (Yairi & Ambrose, 2013). If stuttering is apparent in teenage years, the chance of recovery drastically reduces, particularly in males (Howell, 2007). Lack of effective communication caused by stuttering leads to conditions of anxiety, social phobia, hindered employment opportunities, and overall life quality (Onslow, 2000).

Speech Pathologists (SP) are the ones trained to treat such disfluencies. Despite recent technological advances in speech recognition, SPs still rely on traditional techniques for formal stuttering assessments. Established normative assessment tools such as Test of Childhood Stuttering (TOCS) (Gillam et al. 2009) and the Stuttering Severity Instrument (SSI-4) (Riley 2019) are well regarded and have been created by experts in the field to help diagnose disfluencies. In Australia, SPs simplify these assessments and obtain subjective severity ratings through questionnaires and then implement treatment plans such as the Lidcombe (Onslow et al., 2017) and Camperdown (O'Brian et al., 2018) programs. Often, SPs will calculate the percent of stuttered syllables (%SS). A particularly useful metric to determine the severity of stuttering. Finding the %SS requires SPs to listen and record each disfluency event based on their own perception. This complex and specialised skill can lead to different interpretations of the same sample. Speech Pathology Australia (2017) acknowledges that stuttering is a communication disorder that necessitates evidence-based treatment 'following' skilled assessment by certified SPs, all of which is time intensive and expensive.

Automated stuttering identification methods have been researched for decades, however due to involuntary pauses and irregular repetitions in speech, it is difficult to implement accurately compared to fluent speech. The signal processing and machine learning areas of this multidisciplined field have been explored comparing various extraction methods and

many have delivered high levels of accuracy. Yet no real time stutter identification systems are available (Sheikh et al., 2022). As such, implementing an accurate automated solution to aids SPs with their assessments could reduce time, allow progress tracking, mitigate practitioner differences, and improve treatment plans.

Objectives and Aims:

Upon research it was found that speech pathologists use a manual approach in practice to determine the severity of stuttering. This is a time intensive process that opens room for varying interpretations.

The objectives of this research project are to

- Extract meaningful spectral information from large stuttering audio datasets online.
- Compare the machine learning, neural network, and deep learning methods success rates on the available datasets.
- Implement a successful method and emulate their high success rates in MATLAB.
- Create a useful prototype that will listen to real time speech, then output a display indicating any stuttering event heard with its classification. Along with the % of stuttered syllables.
- Aid SP assessment.

Expected Outcomes:

- Understand what feature extraction information will be beneficial to stuttering detection.
- Understand the limitations and shortcomings of machine learning methods with large audio datasets.
- Produce a real time solution that will be beneficial to SP assessments in Australia and the ongoing management of stuttering patients. Creating a better experience for the SP and patient.

Work Plan:

Timeline of Research Project

Month 1:	Project Proposal: - Gain approval from supervisor and examiner for topic. - Develop rough objectives of research project along with expected outcomes
Month 2-3:	Literature Review: - Investigate exact practices currently employed in SP field to treat stuttering. - In-depth review of spectral components necessary in signal processing large data sets. - Review and compare success rates of previous machine learning methods. - Finalise research objectives.
Month 4:	MATLAB: - Read in and Organise Large datasets in MATLAB. - Learn and Research Machine Learning Tutorials from MATLAB.
Month 5-7:	MATLAB and Speech Pathologist Dialog: - Employ autocorrelation functions for pitch analysis. - Extract spectral information from audio files. Including FFTs - Investigate best features to implement for example, Linear Predictive Cepstral Coefficients. - Communicate with practising speech pathologist to identify what features would improve stuttering assessments.
Month 8-9:	MATLAB: - Employ appropriate methods based off literature review. Test outcomes. - Create an output for stuttering analysis. Potentially using another software platform.
Month 10:	Finalise Dissertation: - Review and finalise report. - Prepare presentation of findings and recommendations for future work.

Resources Required:

Software and Data

- MATLAB and may include additional toolboxes but not limited to Audio, Signal Processing, Statistics and Machine Learning, Deep Learning, DSP System Toolboxes.
- SEP-28k Stuttering Events in Podcasts Dataset
- Fluency Bank Dataset
- Potentially an additional software platform for display.

Access

- Communication with Practicing Speech Pathologist
- University library resources and additional journal article databases online

APPENDIX B – PYCHARM CODE SNIPPETS

This appendix includes the custom Python script snippets developed for the implementation and evaluation of the SD models. These scripts were designed to preprocess the audio data, extract relevant features, and train DL models to classify disfluencies. Due to word formatting inconsistencies when pasting snippets, the code has been manually adjusted in attempts to maintain readability. Therefore, these snippets present a rough guide to the process and should be used with caution when pasted into programming software.

(i) This script cleaned and saved audio files from the downloaded SEP-28K dataset as MFCC features to a file on hard drive as described in **3.3 Features and Normalisation**. Similar steps were used for the mel spectrograms. This saved significant time when training the models.

```
import os
import numpy as np
import glob
import pandas as pd
import matplotlib.pyplot as plt
from scipy.io import wavfile
from python_speech_features import mfcc
from sklearn.preprocessing import MinMaxScaler
import torchaudio


def preprocess_mfccs(list_file_paths, target_sample_rate, output_dir,
                     plot_dir):
    if not os.path.exists(output_dir):
        os.makedirs(output_dir)
    if not os.path.exists(plot_dir):
        os.makedirs(plot_dir)

    plot_count = 0
    max_plots = 10

    for audio_file_path in list_file_paths:
        try:
            print(f"Processing {audio_file_path}")
            samplerate, data = wavfile.read(audio_file_path)
            data = data.astype(np.float32)

            # Ensure data is not empty or malformed
            if data.size == 0:
                print(f"Skipping {audio_file_path}, data is empty")
                continue
```

```
            # Check sampling rate
            if samplerate != target_sample_rate:
                resampler = torchaudio.transforms.Resample(samplerate,
                                                           target_sample_rate)

                data = resampler(torch.from_numpy(data)).numpy()

            # Ensure data length is correct
            if data.shape[0] < 48000:
                data = np.concatenate([data, np.zeros(48000 -
                                                      data.shape[0])])

            if data.shape[0] > 48000:
                data = data[:48000]

            # Compute MFCC
            mfcc_feat = mfcc(data, target_sample_rate, numcep=24,
                             winstep=.025)
            print(f"MFCC shape: {mfcc_feat.shape}")

            # Scale MFCC
            scaler = MinMaxScaler()
            mfcc_feat = scaler.fit_transform(mfcc_feat)
            mfcc_feat = mfcc_feat.T
            print(f"Transformed MFCC shape: {mfcc_feat.shape}")

            filename = os.path.basename(audio_file_path)
            np.save(os.path.join(output_dir, filename.replace('.wav',
                                                              '.npy')),
                                                              mfcc_feat)

            # Show plot for the first 10 MFCCs before saving
            if plot_count < max_plots:
                plot_mfcc(mfcc_feat, filename)
                plot_count += 1
            else:
                # Save MFCC
                save_mfcc_plot(mfcc_feat, filename, plot_dir)

        except Exception as e:
            print(f"Error processing {audio_file_path}: {e}")


# Plot
def plot_mfcc(mfcc_feat, filename):
    print(f"Plotting {filename}")
    plt.figure(figsize=(10, 4))
    plt.imshow(mfcc_feat, interpolation='nearest', cmap='viridis',
               aspect='auto')
    plt.title(f'MFCC - {filename}')
    plt.ylabel('MFCC Coefficients')
    plt.xlabel('Frames')
    plt.colorbar()
    plt.show()


def save_mfcc_plot(mfcc_feat, filename, plot_dir):
    plt.figure(figsize=(10, 4))
    plt.imshow(mfcc_feat, interpolation='nearest', cmap='viridis',
               aspect='auto')
```

```
    plt.title(f'MFCC - {filename}')
    plt.ylabel('MFCC Coefficients')
    plt.xlabel('Frames')
    plt.colorbar()
    plt.savefig(os.path.join(plot_dir, filename.replace('.wav', '.png')))
    plt.close()


if __name__ == "__main__":
    ANNOTATIONS_FILE = "./SEP-28k_labels.csv"
    AUDIO_DIR = "/Users/lachlanjackson/Downloads/clips/*"
    SAMPLE_RATE = 16000
    OUTPUT_DIR = "./preprocessed_mfccs"
    PLOT_DIR = "./mfcc_plots"

    annotations_file = pd.read_csv(ANNOTATIONS_FILE)

    dataset_file_paths = []
    for subfolder in glob.glob(AUDIO_DIR):
        for subsubfolder in glob.glob("{}/*".format(subfolder)):
            for wav in glob.glob("{}/*.wav".format(subsubfolder)):
                dataset_file_paths.append(wav)

    labelmap = {}
    for show, ep_id, clip_id, label1, label2, label3, label4, label5 in
            (zip(
            annotations_file.Show,
            annotations_file.EpId,
            annotations_file.ClipId,
            annotations_file.Interjection,
            annotations_file.Prolongation,
            annotations_file.Block,
            annotations_file.WordRep,
            annotations_file.SoundRep)):

        majority_agreement_map = dict()
        majority_agreement_map['Interjection'] = 0 if label1 < 2 else 1
        majority_agreement_map['Prolongation'] = 0 if label2 < 2 else 1
        majority_agreement_map['Block'] = 0 if label3 < 2 else 1
        majority_agreement_map['WordRep'] = 0 if label4 < 2 else 1
        majority_agreement_map['SoundRep'] = 0 if label5 < 2 else 1

        filename = "{}_{}_{}.wav".format(show, ep_id, clip_id)
        labelmap[filename] = majority_agreement_map

    filtered_audio_data = []
    for file in dataset_file_paths:
        if file.split('/')[-1] in labelmap:
            filtered_audio_data.append(file)

    preprocess_mfccs(filtered_audio_data, SAMPLE_RATE, OUTPUT_DIR,
                     PLOT_DIR)
```

(ii) This is the model training script that used the saved MFCC's as the input to train CNN, LSTM and Conv-LSTM architectures which produced results seen in **4.2 Class Weights – CNN and LSTM Models for Interjection.**

```
import pandas as pd
import torch
import glob
from torch.utils.data import random_split
import numpy as np
from sklearn.metrics import accuracy_score, confusion_matrix, f1_score
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import os
from sklearn.utils.class_weight import compute_class_weight


# THIS SCRIPT USES A MODEL WITH BINARY CROSS ENTROPY LOSS TO TRAIN AND
CLASSIFY ONE STUTTERING DISFLUENCY TYPE

from MODELS import CNNNetwork4, LSTMBaseLineSingle, CNNNetworkMulti,
                   LSTMReduceDim, LSTMCNN, ConvLSTMSingle


class SEPDataset(torch.utils.data.Dataset):
    def __init__(self, list_file_paths, labels, class_type,
                 target_sample_rate, device):
        super().__init__()
        # store provided parameters as class attributes
        self.labels = labels
        self.list_file_paths = list_file_paths
        self.class_type = class_type
        self.target_sample_rate = target_sample_rate
        self.device = device
        self.mfcc_dir = "./preprocessed_mfccs"  # Directory where
                                                  preprocessed MFCCs
                                                  are stored

    def __len__(self):
        return len(self.list_file_paths)

    # This loads the .npy MFCCs files from a location on computer
    def __getitem__(self, index):
        audio_file_path = self.list_file_paths[index]
        filename = os.path.basename(audio_file_path)
        mfcc_path = os.path.join(self.mfcc_dir, filename.replace('.wav',
                                                                 '.npy'))

        if os.path.exists(mfcc_path):
            mfcc_feat = np.load(mfcc_path)
            mfcc_feat = torch.from_numpy(mfcc_feat).float()
        else:
            raise FileNotFoundError(f"MFCC file not found for
                  {audio_file_path}")

        disfluency_class_detected =
        self.labels[audio_file_path.split('/')[-1]][self.class_type]

        return mfcc_feat.unsqueeze(0), disfluency_class_detected


if __name__ == "__main__":
    # Check if GPU or CPU
    device = 'cuda' if torch.cuda.is_available() else 'cpu'
    print("-----------------------------------")
    print(f'Using Device: {device}')
```

```
print("-------------------------------------")

# CONSTANTS
ANNOTATIONS_FILE = "./SEP-28k_labels.csv"
AUDIO_DIR = "/Users/lachlanjackson/Downloads/clips/*"
SAMPLE_RATE = 16000
LEARNING_RATE = 0.0001
CLASS_OF_INTEREST = "Interjection"
EPOCH = 100
MFCC_DIR = "./preprocessed_mfccs"  # Ensure this matches the
                                   directory in SEPDataset
EPOCH_TRAIN_BATCHES = np.floor_divide(17484, 16)
EPOCH_VAL_BATCHES = np.floor_divide(4372, 16)

# Load the dataset csv file
annotations_file = pd.read_csv(ANNOTATIONS_FILE)

# Load all audio file paths in dataset
dataset_file_paths = []
for subfolder in glob.glob(AUDIO_DIR):
    for subsubfolder in glob.glob("{}/*".format(subfolder)):
        for wav in glob.glob("{}/*.wav".format(subsubfolder)):
            dataset_file_paths.append(wav)

# Annotator agreement level loop
labelmap = {}
for show, ep_id, clip_id, label1, label2, label3, label4, label5 in
    (zip(
    annotations_file.Show,
    annotations_file.EpId,
    annotations_file.ClipId,
    annotations_file.Interjection,
    annotations_file.Prolongation,
    annotations_file.Block,
    annotations_file.WordRep,
    annotations_file.SoundRep)):

    # Select annotator agreement level
    majority_agreement_map = dict()
    majority_agreement_map['Interjection'] = 0 if label1 < 1 else 1
    majority_agreement_map['Prolongation'] = 0 if label2 < 3 else 1
    majority_agreement_map['Block'] = 0 if label3 < 3 else 1
    majority_agreement_map['WordRep'] = 0 if label4 < 3 else 1
    majority_agreement_map['SoundRep'] = 0 if label5 < 3 else 1

    filename = "{}_{}_{}.wav".format(show, ep_id, clip_id)
    labelmap[filename] = majority_agreement_map

# Filter to only the audio files that were downloaded
filtered_audio_data = []
for file in dataset_file_paths:
    if file.split('/')[-1] in labelmap:
        filtered_audio_data.append(file)


# Display audio file lengths
print("Audio Files listed in SEP28K Dataset: ", len(labelmap))
print("Total Available after Download: ", len(filtered_audio_data))

# Divide data into training and validation, set random state to 1 for
# fixed split
```

```
train_data, validation_data = train_test_split(filtered_audio_data,
test_size=.2,random_state=1)
print("Length of Training Dataset: ", len(train_data))
print("Length of Validation Dataset: ", len(validation_data))


# Parameters
params = {'batch_size': 16,  # taking 8 samples each for loop
          'shuffle': True,  # split the data randomly
          'num_workers': 0}  # only main process will load batches
                               due to MACBOOK and no GPU

# Training and Validation Generators
sep_dataset_train = SEPDataset(train_data,
                               labelmap,
                               class_type='Interjection',
                               target_sample_rate=SAMPLE_RATE,
                               device=device)

training_generator = torch.utils.data.DataLoader(sep_dataset_train,
                                                 **params)

sep_dataset_validation = SEPDataset(validation_data,
                                    labelmap,
                                    class_type='Interjection',
                                    target_sample_rate=SAMPLE_RATE,
                                    device=device)

validation_generator =
torch.utils.data.DataLoader(sep_dataset_validation, **params)

# Initialise the model
model_name = "best_f1_ConvLSTM"
model = ConvLSTMSingle()

# Create class weights for minority and majority class
weights = torch.tensor([0.7, 1.73])

# Initialise optimiser and loss function
optimiser = torch.optim.Adam(model.parameters(), lr=LEARNING_RATE)
#weight_decay=1e-6
loss_function = torch.nn.CrossEntropyLoss(weights)

# Print number of parameters in model
print(sum(p.numel() for p in model.parameters() if p.requires_grad))

# Initialize lists to store metrics for each epoch
train_losses = []
train_accuracies = []
train_f1_scores = []
val_losses = []
val_accuracies = []
val_f1_scores = []

# Set up live plot
plt.ion()
fig, axs = plt.subplots(1, 3, figsize=(15, 5))
plt.suptitle('MFCC Single Classification Training and Validation
              Metrics Over Epochs')

max_f1 = -np.inf # initialise F1 value for saving
```

```
# Begin
for epoch in range(EPOCH):
    print(f"\n Epoch {epoch + 1}/{EPOCH}")

    # SET TO TRAINING MODE
    model.train()
    print("\n Model Training")
    training_mats = []
    training_labels = []

    for iteration, (x, y) in enumerate(training_generator):
        if iteration == -1:
            break

        # Calc loss and back propogate, then reset gradients
        optimiser.zero_grad()
        output = model(x.float())
        loss = loss_function(output, y.long())
        loss.backward()
        optimiser.step()

        # Calc acc and f1 for batch
        output_rounded = output.argmax(dim=-1).detach().numpy()
        accuracy = accuracy_score(y.numpy(), output_rounded)
        f1 = f1_score(y.numpy(), output_rounded, zero_division=1)
        # Collect metrics for current batch
        training_mats.append([loss.item(), accuracy, f1])
        print("'\r[{:04d}]/[{}] Batches Trained,
        [{:.2f}],{:d}".format(iteration, EPOCH_TRAIN_BATCHES,
        loss.item(), (y==0.).sum(dim=0)), end="")

        # Collect true and predicted labels
        training_labels.append(np.concatenate([y.unsqueeze(dim=-
        1).numpy(), np.expand_dims(output_rounded,axis=-1)], axis=-
        1))

    # Combine labels
    training_labels = np.concatenate(training_labels, axis=0)
    # compute mean loss, f1, and acc across batches, then store
    training_mean = np.array(training_mats).mean(axis=0)
    train_losses.append(training_mean[0])
    train_accuracies.append(training_mean[1])
    train_f1_scores.append(training_mean[2])
    print('\n')
    print(confusion_matrix(training_labels[:, 0],training_labels[:,
                                                                 1]))


    # SET TO EVALUATION MODE
    print("\n Model Evaluation")
    model.eval()
    validation_mats = []
    validation_labels = []

    for iteration, (x, y) in enumerate(validation_generator):
        if iteration == -1:
            break
        # forward pass and calc loss
        output = model(x.float())
```

```
        loss = loss_function(output, y.long())

        # convert to highest class prediction to numpy
        output_rounded = output.argmax(dim=-1).detach().numpy()
        # calc acc and f1 score
        accuracy = accuracy_score(y.numpy(), output_rounded)
        f1 = f1_score(y.numpy(), output_rounded, zero_division=1)
        print("'\r[{:04d}]/[{}] Batches Evaluated".format(iteration,
                                            EPOCH_VAL_BATCHES), end="")

        # collect current batch loss, acc, f1 score
        validation_mats.append([loss.item(), accuracy, f1])

        # collect true and predicted labels
        validation_labels.append(np.concatenate([y.unsqueeze(dim=-
        1).numpy(), np.expand_dims(output_rounded,axis=-1)], axis=-
        1))

    # combine all labels
    validation_labels = np.concatenate(validation_labels, axis=0)
    # compute the mean loss, acc, and f1 score and store
    validation_mean = np.array(validation_mats).mean(axis=0)
    val_losses.append(validation_mean[0])
    val_accuracies.append(validation_mean[1])
    val_f1_scores.append(validation_mean[2])

    # Save best model based on f1 score
    if validation_mean[2] > max_f1:
        torch.save(model.state_dict(),
        "best_f1_score_{}.pt".format(model_name))
        print("models performance increased from {:.2f} to
        {:.2f}".format(max_f1,validation_mean[2]))
        max_f1 = validation_mean[2]

    print('\n')
    print(confusion_matrix(validation_labels[:, 0],
    validation_labels[:, 1]))

    # Update live plot
    axs[0].clear()
    axs[0].plot(range(1, epoch + 2), train_losses, label='Training
    Loss')
    axs[0].plot(range(1, epoch + 2), val_losses, label='Validation
    Loss')
    axs[0].set_xlabel('Epoch')
    axs[0].set_ylabel('Loss')
    axs[0].legend()
    axs[0].set_title('Loss')

    axs[1].clear()
    axs[1].plot(range(1, epoch + 2), train_accuracies,
    label='Training Accuracy')
    axs[1].plot(range(1, epoch + 2), val_accuracies,
    label='Validation Accuracy')
    axs[1].set_xlabel('Epoch')
    axs[1].set_ylabel('Accuracy')
    axs[1].legend()
    axs[1].set_title('Accuracy')

    axs[2].clear()
    axs[2].plot(range(1, epoch + 2), train_f1_scores, label='Training
```

```
                                                                F1 Score')

        axs[2].plot(range(1, epoch + 2), val_f1_scores, label='Validation
                                                                F1 Score')
        axs[2].set_xlabel('Epoch')
        axs[2].set_ylabel('F1 Score')
        axs[2].legend()
        axs[2].set_title('F1 Score')

        plt.pause(0.01)  # Pause to update the plot

    plt.ioff()  # Turn off interactive mode
    plt.show()  # Show the final plot
```

(iii) The script shows the designed model architectures for CNN, LSTM and Conv-LSTM as described in **Figure 19** and **Figure 20** from section **4.5 Investigated Models.**

```
import pandas as pd
import torch
import glob
import numpy as np
from sklearn.metrics import accuracy_score, confusion_matrix, f1_score
from sklearn.model_selection import train_test_split
from torch import nn
import matplotlib.pyplot as plt
import os

# THIS FILE CONTAINS ALL THE MODELS THAT HAVE BEEN CREATED FOR STUTTERING
DETECTION


class CNNNetwork4(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Sequential(
            nn.Conv2d(in_channels=1,out_channels=8,kernel_size=(3,7)),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=(1,3)),
            nn.Dropout(0.3)

        )
        self.conv2 = nn.Sequential(
            nn.Conv2d(in_channels=8,out_channels=16,kernel_size=(3,7)),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=(1,3)),
            nn.Dropout(0.3)
        )
        self.conv3 = nn.Sequential(
            nn.Conv2d(in_channels=16,out_channels=24,kernel_size=(3,5)),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=(2, 1)),
            nn.Dropout(0.3)
        )
```

```
        self.conv4 = nn.Sequential(
            nn.Conv2d(in_channels=24, out_channels=32,
                      kernel_size=(1,3)),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=(2,1)),
            nn.Dropout(0.3)

        )
        self.conv5 = nn.Sequential(
            nn.Conv2d(in_channels=32, out_channels=48,
                      kernel_size=(1,3)),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2),
            nn.Dropout(0.3)
        )
        self.conv6 = nn.Sequential(
            nn.Conv2d(in_channels=48, out_channels=64, kernel_size=1),
            nn.ReLU(),
            nn.Dropout(0.3)
        )
        self.flatten = nn.Flatten()

        # Calculate the size of the input to the first fully connected
          layer
        self._calculate_conv_output_size()

        self.mlp = nn.Sequential(
            nn.Linear(self.conv_output_size, 64),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(64, 32),
            nn.ReLU(),
            nn.Linear(32, 2),
            nn.Softmax(dim=-1)
        )
    def _calculate_conv_output_size(self):
        with torch.no_grad():
            input_data = torch.zeros(1, 1, 24, 120)  # MFCC input example

            output = self.conv1(input_data)
            print("After conv1: ", output.shape)
            output = self.conv2(output)
            print("After conv2: ", output.shape)
            output = self.conv3(output)
            print("After conv3: ", output.shape)
            output = self.conv4(output)
            print("After conv4: ", output.shape)
            output = self.conv5(output)
            print("After conv5: ", output.shape)
            output = self.conv6(output)
            print("After conv6: ", output.shape)
            self.conv_output_size = output.numel()


    def forward(self, input_data):
        x = self.conv1(input_data)
        x = self.conv2(x)
        x = self.conv3(x)
        x = self.conv4(x)
        x = self.conv5(x)
        x = self.conv6(x)
```

```
        x = self.flatten(x)
        return self.mlp(x)


class LSTMBaseLineSingle(nn.Module):

    def __init__(self):

        super().__init__()
        self.lstm = nn.LSTM(input_size=24, hidden_size=64,
num_layers=1,dropout=0.3)
        self.flatten = nn.Flatten()
        self.mlp1 = nn.Sequential(nn.Linear(64, 128),

                                  nn.ReLU(),
                                  nn.Dropout(0.3),
                                  nn.Linear(128, 64),
                                  nn.ReLU(),
                                  nn.Dropout(0.3),
                                  nn.Linear(64, 2),
                                  nn.Softmax(dim= 1)
                                  )

    def forward(self, input_data):
        output, _ = self.lstm(input_data.squeeze(dim=1).permute(0, 2, 1))
        x = output[:, -1, :]
        return self.mlp1(x)


class ConvLSTMSingle(nn.Module):
    def __init__(self):
        super().__init__()

        # CNN layers
        self.conv1 = nn.Sequential(
            nn.Conv2d(in_channels=1, out_channels=8, kernel_size=(3, 7)),
            nn.ReLU(),
            nn.Dropout(0.3)
        )
        self.conv2 = nn.Sequential(
            nn.Conv2d(in_channels=8, out_channels=16, kernel_size=(3,
                                                                   7)),
            nn.ReLU(),
            nn.Dropout(0.3)
        )
        self.conv3 = nn.Sequential(
            nn.Conv2d(in_channels=16, out_channels=24, kernel_size=(3,
                                                                    5)),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=(3, 1)),
            nn.Dropout(0.3)
        )
        self.conv4 = nn.Sequential(
            nn.Conv2d(in_channels=24, out_channels=32, kernel_size=(1,
                                                                    3)),
            nn.ReLU(),
            nn.Dropout(0.3)
        )
        self.conv5 = nn.Sequential(
            nn.Conv2d(in_channels=32, out_channels=48, kernel_size=(1,
```

```
                                                                    3)),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=(3,2)),
            nn.Dropout(0.3)
        )
        self.conv6 = nn.Sequential(
            nn.Conv2d(in_channels=48, out_channels=64, kernel_size=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=(2,1)),
            nn.Dropout(0.3)
        )

        # LSTM layers
        self.lstm = nn.LSTM(input_size=64, hidden_size=64,
                            num_layers=1,dropout=0.3,batch_first=True)

        # Fully connected layers (MLP)
        self.mlp = nn.Sequential(
            nn.Linear(64, 128),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(128, 64),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(64, 2),
            nn.Softmax(dim=1)
        )

    def forward(self, input_data):
        # CNN forward pass
        x = self.conv1(input_data)
        x = self.conv2(x)
        x = self.conv3(x)
        x = self.conv4(x)
        x = self.conv5(x)
        x = self.conv6(x)

        # reduce dimension from [16,64,1,50] to [16,64,50]
        x = x.squeeze(2)

        # LSTM forward pass, change shape to [16,50,64]
        lstm_out, _ = self.lstm(x.permute(0, 2, 1))
        lstm_out = lstm_out[:, -1, :]  # Take the last time step output

        # MLP forward pass
        return self.mlp(lstm_out)
```

(iv) This is the script used to access the mel spectrogram files, pass them through the AST and save output feature vectors prior to training, this was done to reduce computation in training time (see **3.6 Audio Spectrogram Transformer**).

```
from ast_master.src.models.ast_models import ASTModel
import torch
import glob
import numpy as np
```

```
# directory containing mel spectrogram files
dir =
"/Users/lachlanjackson/PycharmProjects/pythonProject/dataset/preprocessed
_spec/"

# Get list of all mel spectrogram files directory
stft_files = glob.glob(dir+"*")

input_tdim = 1024

# path to the pretrained AST model
pretrained_mdl_path =
'/Users/lachlanjackson/PycharmProjects/AST_MODEL/ast_master/pretrained_mo
dels/audioset_10_10_0.4593.pth'

# get the frequency and time stride of the pretrained model from its name
fstride, tstride = int(pretrained_mdl_path.split('/')[-1].split('_')[1]),
                   int(pretrained_mdl_path.split('/')[-
                                          1].split('_')[2].split('.')[0])

# initialize AST model and load
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
sd = torch.load(pretrained_mdl_path, map_location=device)
audio_model = ASTModel(input_tdim=input_tdim, fstride=fstride,
tstride=tstride)
audio_model = torch.nn.DataParallel(audio_model)
audio_model.load_state_dict(sd, strict=False)

# directory to save AST features
ast_features_dir =
"/Users/lachlanjackson/PycharmProjects/pythonProject/dataset/AST_features
/"

# iterate each mel spectrogram file
for filename in stft_files:
    spec = np.load(filename)
    # change dimension 128 to 1024
    x = torch.from_numpy(spec).unsqueeze(0).permute(0, 2, 1).float()
    features = audio_model(x)
    torch.save(features, ast_features_dir + filename.split("/")[-
                                                1].replace("npy","tm"))
    print("saving",ast_features_dir + filename.split("/")[-
                                                1].replace("npy","tm"),
                                                    features.shape)
```

(v) This script snippet was implemented to train the AST model. This is not the full script as it was largely the same lines of code in the aforementioned model training. Below illustrate the main differences, including the MLP head used (see network diagram in **3.6 Audio Spectrogram Transformer)**. Also accessing the AST features file opposed to the MFCC features file.

```
    # INITIALISE MODEL, OPTIMISER AND LOSS FUNCTION
    pretrained_mdl_path =
'/Users/lachlanjackson/PycharmProjects/AST_MODEL/ast_master/pretrained_mo
dels/audioset_10_10_0.4593.pth'

    # get the frequency and time stride of the pretrained model from its
      name
    fstride, tstride = int(pretrained_mdl_path.split('/')[-
                                                          1].split('_')[1]),
                       int(pretrained_mdl_path.split('/')[-
                                                  1].split('_')[2].split('.')[0])

    model_name = "ASTModel_class_weighted_soundrep"
    sd = torch.load(pretrained_mdl_path, map_location=device)
    audio_model = ASTModel(input_tdim=input_tdim, fstride=fstride,
                                                    tstride=tstride)
    audio_model.load_state_dict(sd, strict=False)

    #weights = torch.tensor([0.7, 1.73])  # interjection
    #weights = torch.tensor([0.66, 2.06]) # prolongation
    #weights = torch.tensor([0.71,1.67]) # block
    #weights = torch.tensor([0.59,3.29]) # word rep
    weights = torch.tensor([0.61,2.87]) # sound rep

    # MLP head
    model =  nn.Sequential(nn.Linear(768, 256),
             nn.ReLU(),
             nn.Dropout(0.3),
             nn.Linear(256, 128),
             nn.ReLU(),
             nn.Dropout(0.3),
             nn.Linear(128, 2),
             nn.Softmax(dim=-1)
             )

    optimiser = torch.optim.Adam(model.parameters(), lr=LEARNING_RATE )
    loss_function = torch.nn.CrossEntropyLoss(weights)
```

(vi) This script was used to extract disfluent and fluent features for the purpose of T-SNE visualisation seen in **4.8 T-SNE Plots to Visualise AST Parameters**.

```
import pandas as pd
import torch
import glob
from torch.utils.data import random_split
import numpy as np
from sklearn.metrics import accuracy_score, confusion_matrix, f1_score
from sklearn.model_selection import train_test_split
from torch import nn
import matplotlib.pyplot as plt
import os
from ast_master.src.models.ast_models import ASTModel
import torch
import sys
```

```
if __name__ == "__main__":

    # CHECK IF GPU OR CPU
    if torch.cuda.is_available():
        device = 'cuda'
    else:
        device = 'cpu'
    print("-------------------------------------")
    print(f'Using Device: {device}')
    print("-------------------------------------")

    # CONSTANTS
    ANNOTATIONS_FILE = "./SEP-28k_labels.csv"
    AUDIO_DIR = "/Users/lachlanjackson/Downloads/clips/*"
    SAMPLE_RATE = 16000
    LEARNING_RATE = 0.001
    EPOCH = 50
    EPOCH_TRAIN_BATCHES = np.floor_divide(14571,16)
    EPOCH_VAL_BATCHES = np.floor_divide(7285,16)
    CLASS_OF_INTEREST = "WordRep"

    # LOAD CSV
    annotations_file = pd.read_csv(ANNOTATIONS_FILE)

    # LOAD ALL AUDIO FILE PATHS IN DATASET
    dataset_file_paths = []
    for subfolder in glob.glob(AUDIO_DIR):
        for subsubfolder in glob.glob("{}/*".format(subfolder)):
            for wav in glob.glob("{}/*.wav".format(subsubfolder)):
                dataset_file_paths.append(wav)

# TAKE ONLY UNANIMOUS AGREEMENT FILES FOR THE 5 CLASSES
    labelmap = {}
    for show, ep_id, clip_id, label1, label2, label3, label4, label5 in
zip(
        annotations_file.Show,
        annotations_file.EpId,
        annotations_file.ClipId,
        annotations_file.Interjection,
        annotations_file.Prolongation,
        annotations_file.Block,
        annotations_file.WordRep,
        annotations_file.SoundRep):

        majority_agreement_map = dict()
        majority_agreement_map['Interjection'] = 0 if label1 < 3 else 1
        majority_agreement_map['Prolongation'] = 0 if label2 < 3 else 1
        majority_agreement_map['Block'] = 0 if label3 < 3 else 1
        majority_agreement_map['WordRep'] = 0 if label4 < 3 else 1
        majority_agreement_map['SoundRep'] = 0 if label5 < 3 else 1

        filename = "{}_{}_{}.wav".format(show, ep_id, clip_id)
        labelmap[filename] = majority_agreement_map

    # FILTER TO ONLY AUDIO FILES THAT WERE AVAILABLE FOR DOWNLOAD
    filtered_audio_data = []
    for file in dataset_file_paths:
        if file.split('/')[-1] in labelmap:
            filtered_audio_data.append(file)

    print("Audio Files listed in SEP28K Dataset: ", len(labelmap))
```

```
    print("Total Available after Download: ", len(filtered_audio_data))

    train_data, validation_data = train_test_split(filtered_audio_data,
                                                   test_size=.2,
                                                   random_state=1)

    parentdir = str(os.path.abspath(os.path.join(__file__, "../../"))) +
                                                                    '/src'
    print(parentdir)
    sys.path.append(parentdir)


    pretrained_mdl_path =
'/Users/lachlanjackson/PycharmProjects/AST_MODEL/ast_master/pretrained_mo
dels/audioset_10_10_0.4593.pth'

    # get frequency and time stride of pretrained model
    fstride, tstride = int(pretrained_mdl_path.split('/')[-
                                                    1].split('_')[1]),
                       int(pretrained_mdl_path.split('/')[-
                                              1].split('_')[2].split('.')[0])

    # The input of audioset pretrained model is 1024 frames.
    input_tdim = 1024

    # initialize an AST model
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    sd = torch.load(pretrained_mdl_path, map_location=device)
    audio_model = ASTModel(input_tdim=input_tdim, fstride=fstride,
                                                        tstride=tstride)
    audio_model = torch.nn.DataParallel(audio_model)
    audio_model.load_state_dict(sd, strict=False)



    # Iterate through training set, extract fluent and disfluent samples
    fluent = []
    disfluent = []
    for file in train_data:
        disfluency_class_detected = labelmap[file.split('/')[-
                                                1]][CLASS_OF_INTEREST]
        if disfluency_class_detected == 0 and len(fluent)<1000:
            fluent.append(file)
        elif len(disfluent)<1000:
            disfluent.append(file)
        else:
            pass


# mel spectrogram directory
    dir =
"/Users/lachlanjackson/PycharmProjects/pythonProject/dataset/preprocessed
_spec/"

# directory to save extracted features
    ast_features_dir =
"/Users/lachlanjackson/PycharmProjects/AST_MODEL/ast_features/"

# process and extract features for fluent samples
    for i,file in enumerate(fluent):
        filename = dir+file.split('/')[-1].replace("wav","npy")
```

```
        try:
            spec = np.load(filename)
             # change dimension 128 to 1024
            x = torch.from_numpy(spec).unsqueeze(0).permute(0,2,1)
            features = audio_model(x)



            # save fluent samples
            torch.save(features,ast_features_dir+
            "ast_fluent_{}_features_{}.tm".format(CLASS_OF_INTEREST,i))
            print(i)
        except:
            pass

# Do same for disfluent features
    for i,file in enumerate(disfluent):
        filename = dir+file.split('/')[-1].replace("wav","npy")
        try:
            spec = np.load(filename)
             # change dimension 128 to 1024
            x = torch.from_numpy(spec).unsqueeze(0).permute(0,2,1)
            features = audio_model(x)
            torch.save(features,ast_features_dir+
           "ast_disfluent_{}_features_{}.tm".format(CLASS_OF_INTEREST,i))
            print(i)
        except:
            pass
```

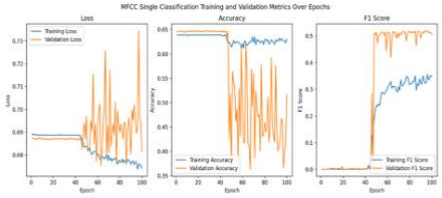
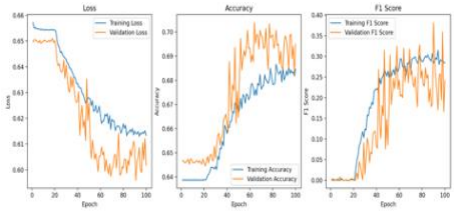
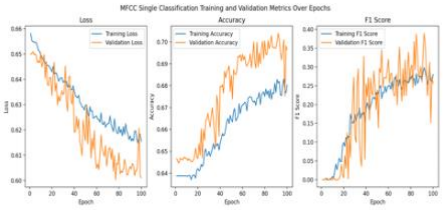
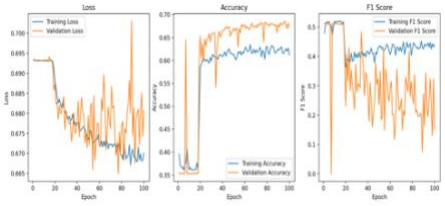
(vii) This script was created to plot a T-SNE comparing AST extracted features of fluent and disfluent samples from either before or after training. Here shows word repetition initial features which are displayed in **Figure 29.d)** of section **4.8 T-SNE plots to Visualise AST Parameters.**

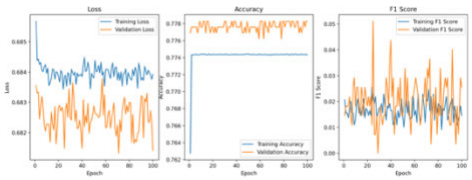
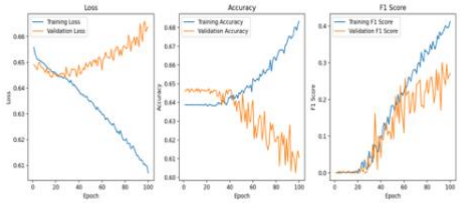
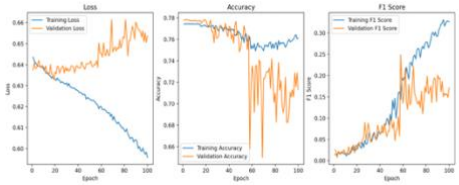
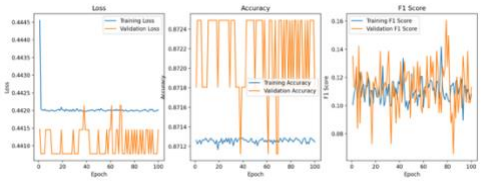
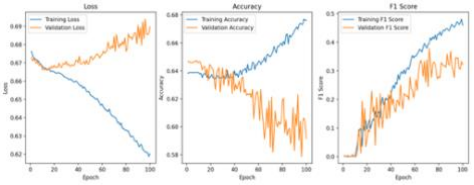
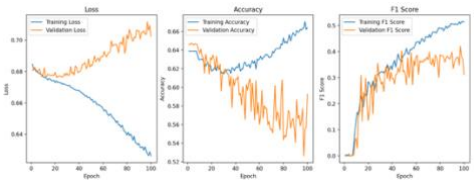
```
import glob
import torch
import numpy as np
import matplotlib.pyplot as plt
from sklearn.manifold import TSNE

features = []
labels = []  # To store the labels (0 for fluent, 1 for disfluent)

# Get all the files
files =
glob.glob("/Users/lachlanjackson/PycharmProjects/AST_MODEL/ast_features/W
                                                          ordRep/ast*")
print(f"Number of files: {len(files)}")  # Should return 990 files

# Iterate through files and load features and labels
for file in files:
    feature = torch.load(file)
    features.append(feature)
```

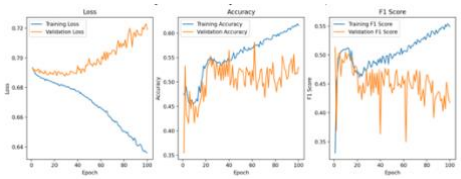
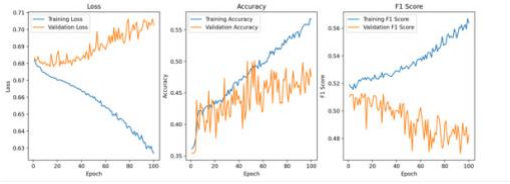
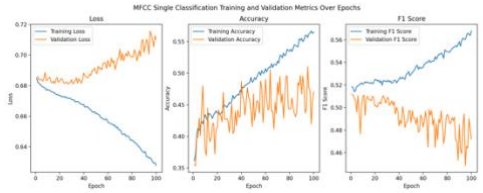
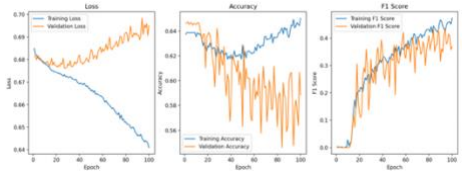
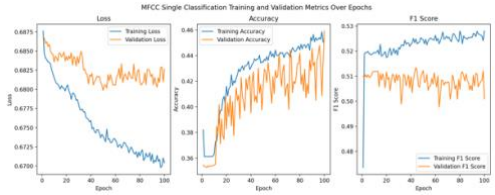
```
    # Assign label based on filename
    if "disfluent" in file:
        labels.append(1)  # Disfluent = 1
    elif "fluent" in file:
        labels.append(0)  # Fluent = 0

# Convert lists to tensors
features = torch.cat(features, dim=0)
labels = np.array(labels)  # Labels in a numpy array
# Print shapes to ensure correctness
print("Features shape:", features.shape)
print("Labels shape:", labels.shape)

# Apply t-SNE on features
X_embedded = TSNE(n_components=2, learning_rate='auto', init='random',
                  perplexity=12, max_iter=1000).fit_transform(
                                                features.detach().numpy())

# Plotting t-SNE results
plt.figure(figsize=(8, 6))
plt.scatter(X_embedded[labels == 0, 0], X_embedded[labels == 0, 1],
label='Fluent')
plt.scatter(X_embedded[labels == 1, 0], X_embedded[labels == 1, 1],
label='Disfluent')
plt.legend()
plt.title("Pretrained AST Model t-SNE: Word Repetition Initial Feature
Representations")

plt.show()
```

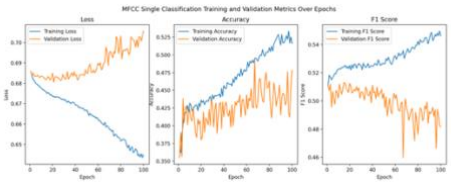
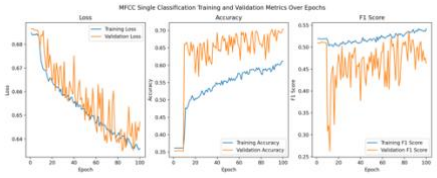
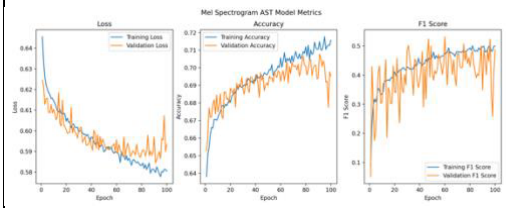
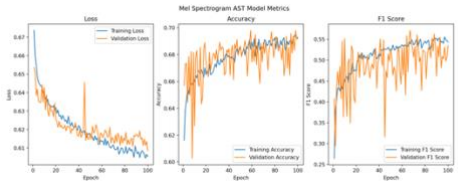
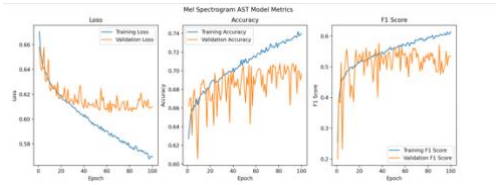
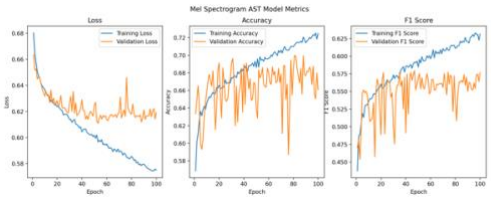
APPENDIX C – KEY MODEL RESULTS

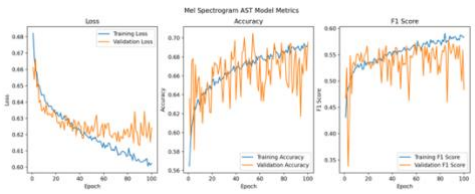
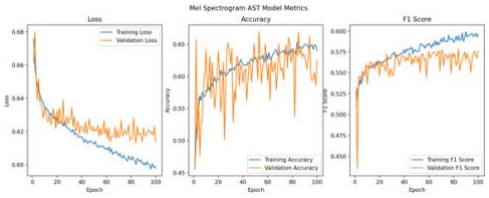
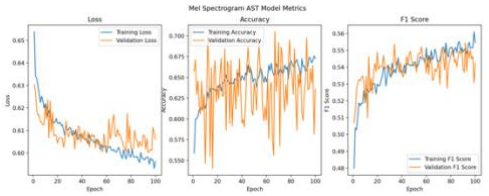
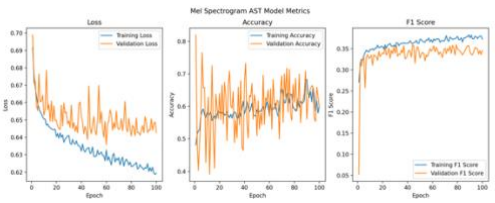
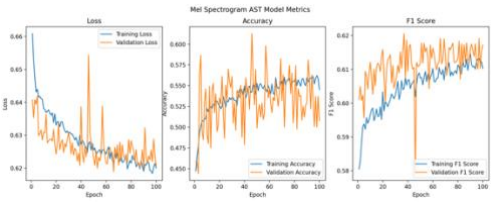
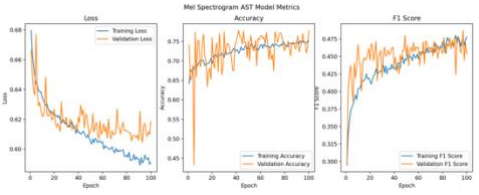
This appendix shows some key results from the tests conducted. It displays the model architecture, the disfluency tested, which annotator agreement level, along with other key hyperparameters. The plots give a general snapshot of training performance, and the confusion matrix can be used to calculate important metrics such as accuracy, F1 score and EER. It should be noted that potentially 200 training runs were conducted with many not obtaining meaningful results.

Setting	**Loss Function and Hyperparameters**	**Plot Results**	**Confusion Matrix**
1: CNN-v0/MFCC Interjection CNNNetwork4 arch 1 annotator agreement Batch size = 16	CE Weights = [0.4, 0.6] LR = 0.001 Dropout = 0.3		Epoch 100 Model Training [[8996 2172] [4292 2024]] Model Eval [[1089 1737] [382 1164]]
2: CNN-v0/MFCC Interjection CNNNetwork4 arch 1 annotator agreement Batch size = 16	CE No Weights LR = 0.001 Dropout = .25		Epoch 100 Model Training [[10683 485] [5029 1287]] Model Eval [[2794 32] [1305 241]]
3; CNN-v0/MFCC Interjection CNNNetwork4 arch 1 annotator agreement Batch size = 16	CE No Weights LR = 0.001 Dropout =0.3		Epoch 100 Model Training [[10604 564] [5027 1289]] Model Eval [[2772 54] [1274 272]]
4: CNN-v0/MFCC Interjection CNNNetwork4 arch 1 annotator agreement Batch size = 16	CE Weights = [0.45, 0.85] LR= 0.001 Dropout = 0.3		Epoch 99 Model Training [[8133 3035] [3490 2826]] Model Eval [[2597 229] [1166 380]]

5: CNN-v0/MFCC Interjection CNNNetwork4 arch 2 annotator agreement Batch size = 16	CE Weights = [0.3, 0.85] LR = 0.001 Dropout = 0.3		Epoch 100 Model Training [[13539 0] [3945 0]] Model Eval [[3400 0] [972 0]]
6: LSTM-/MFCC Interjection LSTMBaseline arch 1 annotator agreement Batch size = 16	CE No Weights LR = 0.001 Dropout = 0.3		Epoch 100 Model Training [[9762 1406] [4132 2184]] Model Eval [[2304 522] [1182 364]]
7: LSTM-/MFCC Interjection LSTMBaseline arch 2 annotator agreement Batch size = 16	CE Weights = [0.35, 0.65] LR = 0.001 Dropout = 0.3		Epoch 100 Model Training [[12083 1456] [2723 1222]] Model Eval [[2971 429] [829 143]]
8: LSTM-/MFCC Interjection LSTMBaseline arch 3 annotator agreement Batch size = 16	CE No Weights LR = 0.001 Dropout = 0.3		Epoch 100 Model Training [[15233 0] [2251 0]] Model Eval [[3813 0] [559 0]]
9: LSTM-/MFCC Interjection LSTMBaseline arch 1 annotator agreement Batch size = 16	CE Weights = [0.45, 0.55] LR = 0.001 Dropout = 0.3		Epoch 99 Model Training [[8881 2287] [3376 2940]] Model Eval [[2159 667] [1069 477]]
10: LSTM-/MFCC Interjection LSTMBaseline arch 1 annotator agreement Batch size = 16	CE Weights = [0.42, 0.58] LR = 0.001 Dropout = 0.3		Model Training [[8207 2961] [2915 3401]] Model Evaluation [[2130 696] [1090 456]]

11: LSTM-/MFCC Interjection LSTMBaseline arch 1 annotator agreement Batch size = 16	CE Weights = [0.35, 0.65] LR = 0.001 Dropout = 0.3		Epoch 97 Model Training [[6351 4817] [1897 4419]] Model Eval [[1312 1514] [604 942]]
12: LSTM-/MFCC Interjection LSTMBaseline arch 1 annotator agreement Batch size = 16	CE Weights = [0.28, 0.72] LR = 0.001 Dropout = 0.3		Epoch 99 Model Training [[4652 6516] [1045 5271]] Model Eval [[1057 1769] [467 1079]]
13: LSTM-/MFCC Interjection LSTMBaseline arch 1 annotator agreement Batch size = 16	EXTRA DROPOUT CE Weights = [0.7, 1.73] LR = 0.001 Dropout = 0.3		Epoch 100 Model Training [[4598 6570] [1037 5279]] Model Eval [[948 1878] [438 1108]]
14: LSTM-/MFCC Interjection LSTMBaseline arch 1 annotator agreement Batch size = 16	CE Weights = [0.42, 0.58] LR = 0.001 Dropout = 0.3 Weight decay = 1e-6		Epoch 99 Model Training [[8422 2746] [3471 2845]] Model Eval [[2142 684] [1037 509]]
15: LSTM-/MFCC Interjection LSTMBaseline arch 1 annotator agreement Batch size = 16	 Weights = [0.7, 1.73] LR = 0.0001 Dropout = 0.3 Weight decay = 1e-5		Epoch 100/100 Model Training [[2304 8864] [698 5618]] Model Eval [[736 2090] [278 1268]]

16: Reduced LSTM-/MFCC Interjection LSTMBaseline arch 1 annotator agreement Batch size = 16	CE No weights LR = 0.0001 Dropout = 0.3		Epoch 100 Model Training [[3626 7542] [917 5399]] Model Eval [[972 1854] [427 1119]]
17: CONV LSTM-/MFCC Interjection LSTMBaseline arch 1 annotator agreement Batch size = 16	CE Weights = [0.7,1.73] LR = 0.0001 Dropout = 0.3		Epoch 84 Model Training [[5891 5277] [1892 4424]] Model Eval [[1655 1171] [508 1038]]
18: Mel spectrogram to AST Model features MLP head	CE No Weights LR = 0.001 Dropout = 0.3		Epoch 100 Model Training [[9716 1452] [3519 2797]] Model Eval [[2316 510] [826 720]]
19: Mel spectrogram to AST Model features MLP head	CE Weights = [0.42,0.58] LR = 0.001 Dropout = 0.3		Epoch 100 Model Training [[8595 2573] [2798 3518]] Model Eval [[2184 642] [702 844]]
20: Mel spectrogram to AST Model features MLP head	CE Weights = [0.42,0.58] LR = 0.0001 Dropout = 0.3		Model Training [[8989 2179] [2371 3945]] Model Eval [[2188 638] [688 858]]
21: Mel spectrogram to AST Model features MLP head	CE Weights = [1.56,2.78] LR = 0.0001 Dropout = 0.3		Epoch 100 Model Training [[8138 3030] [1785 4531]] Model Eval [[1809 1017] [464 1082]]

22: Mel spectrogram to AST Model features MLP head	CE Weights = [1.56,2.78] LR = 0.001 Dropout = 0.3		Epoch 91 Model Training [[7921 3247] [2187 4129]] Model Eval [[1676 1150] [434 1112]]
23: Mel spectrogram to AST Model features MLP head	CE Weights = [0.7,1.73] LR = 0.001 Dropout = 0.3		Epoch 100 Model Training [[3626 7542] [917 5399]] Model Eval [[972 1854] [427 1119]]
24: Prolongation Mel spectrogram to AST Model features MLP head	CE Weights = [0.66, 2.06] LR = 0.001 Dropout = 0.3		Model Training [[1819 1178] Model Eval [352 1023]]
25: Word Rep Mel spectrogram to AST Model features MLP head	CE Weights = [0.59, 3.21] LR = 0.001 Dropout = 0.3		Model Training [[2284 1330] Model Eval [270 488]]
26: Block Mel spectrogram to AST Model features MLP head	CE Weights = [0.71, 1.67] LR = 0.001 Dropout = 0.3		Model Training [[720 1726] Model Eval [245 1681]]
27: Sound Rep Mel spectrogram to AST Model features MLP head	CE Weights = [0.61, 2.87] LR = 0.001 Dropout = 0.3		Model Training [[2615 736] Model Eval [421 600]]

APPENDIX D – ETHICAL CONSIDERATIONS

The research presented in this dissertation utilised the SEP-28K dataset, a publicly available resource commonly used for research and development studies in stuttering detection. Since the dataset annotations used podcast source details without including personal identifiers, ethical concerns related to data privacy were minimal.

However, it is acknowledged that the original copyright of the audio files remains with the podcast owners. This requires careful ethical considerations when downloading and processing these audio files. The research sort transparency in the use of this dataset, aiming to contribute advancements to the stuttering detection field.

No new data collection was conducted, thereby eliminating the need for ethics approval relating to participants and data handling.

APPENDIX E – RISK MANAGEMENT PLAN

NUMBER	RISK DESCRIPTION	TREND	CURRENT	RESIDUAL
6096	Lachlan_Jackson_Dissertation_Risk Assessment		Medium	Very Low
DOCUMENTS REFERENCED				

RISK OWNER	RISK IDENTIFIED ON	LAST REVIEWED ON	NEXT SCHEDULED REVIEW
Lachlan Jackson	20/05/2024	31/10/2024	30/04/2025

RISK FACTOR(S)	EXISTING CONTROL(S)	PROPOSED CONTROL(S) OWNER DUE DATE
Data reliability and accessibility. dataset is downloadable and is usable.	**Control:** check to see if dataset can be downloaded prior to literature review.	**Control:** find a different dataset in the stuttering detection field. 04/11/2024
data preprocessing risks, ensuring data is cleaned and doesn't introduce errors in results	**Control:** use a series of checks within my code to clean data.	**Control:** perform by hand calculations as well to see if code aligns. 04/11/2024
Ethical considerations, even though it is a public dataset, should be cautious about using audio samples from people with stuttering disorder	**Control:** Only choose from publicly available datasets	**Control:** Ensure no sensitive data is present, and save dataset only to hard drive, and delete afterwards.
Ensuring the computational power is sufficient for the task	**Control:** use smaller model architectures to reduce complexity and parameters.	**Control:** Find effective ways to reduce computational load, such as pre saving features before training models.
Access to a usable computer and power source is essential.	**Control:** Macbook M1 Pro being used, and it should last one year as it is new equipment.	**Control:** buy new computer, or use public one. or borrow friends. Only need to download PyCharm and Dataset to new device. 04/11/2024