DISFLUENT SPEECH SEGMENTS DETECTION AND REMEDIATION

by

Pierre Arbajian

A dissertation submitted to the faculty of
The University of North Carolina at Charlotte
in fulfillment of the requirements
for the degree of Doctor of Philosophy in
Computing and Information Systems

Charlotte

2019

Approved by:

Dr. Zbigniew W. Raś

Dr. Wlodek Zadrozny

Dr. Alicja Wieczorkowska

Dr. Jing Yang

Dr. Maciej Noras

ABSTRACT

PIERRE ARBAJIAN. Disfluent Speech Segments Detection and Remediation.
(Under the direction of DR. ZBIGNIEW W. RAŚ)

Speech remediation by identifying those segments which compromise the quality of speech content can be performed by correctly identifying portions of a recording which can be deleted without diminishing from the overall quality of the speech, but rather improving it. Speech remediation is especially important when it is heavily disfluent as in the case of stuttering speakers' speeches.

In our work we focused on two types of disfluency ***blocks*** and ***interjections***. The preparation work and the features required for each type of speech anomaly were different as we used distinct approaches according to the speech disfluency we were detecting and the application method.

In this dissertation, we describe work which consists of (1) developing several methods to extract stutter speech segments, (2) creating the raw digital signal analysis features, (3) performing the feature engineering, (4) labeling the segments, (5) training the classifier, and finally (6) scoring the speech segments to identify the sounds that must be removed from a recording.

The experimentation with statistical aggregation feature types for speech ***blocks*** yielded strong results, but ***interjection*** disfluencies scoring required spectral data analysis. The need for different features per disfluency type led us to three approaches, the first two were suitable for ***blocks*** detection and the third approach was applied to ***interjections***.

Two approaches one with ***Pre-qualification*** sampling and the second with ***Random*** sampling used statistical representations of the segments analysis data as features with minimal neural network classifiers and performed well in two types of application: (1) Sampling of segments with a pre-qualification phase and (2) Sampling of training samples without pre-qualification followed by a sliding window classification scoring method. The second speech disfluency i.e. ***interjections*** required different techniques thus ***Approach Spectral-analysis***. In ***Approach Spectral-analysis*** we used spectral analysis metrics from the sound segments as predictors and trained Neural Network classifiers implemented as multiple ***(CNN)*** models to detect the ***interjections***.

The corpus we used is a well recognized set of stuttered speech recordings $UCLASS$. The speeches we used are not labeled in a matter conducive for our research, therefore we performed extensive experimentation to build and label the training data.

Acknowledgments

I would like to thank my Advisor Dr. Zibgniew Ras for his support and guidance, additionally I express my thanks to Dr Ayman Hajja, my friend for his help and collaboration. Dr Alicia Wieczorkowska's assistance has been instrumental throughout this work.

My committee members have been insightful with their comments and recommendations; I convey my thanks to them for the patience and support they provided.

I would be remiss to leave out my wife Lucy who has encouraged me every step of the way in this endeavor; and my daughter Lexi who helped with the proof reading and was there as my sounding board when I needed to discuss related matters.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

Quality of life is negatively impacted when individuals are chronically unable to express themselves due to lack of speech fluency. Stuttering, also referred to as stammering, is a condition that is exhibited in bad fluency of speech. The onset of stuttering generally occurs during childhood years, and in one fourth of cases this speech impediment persists throughout life [26, 39].

Automatic speech disfluency detection offers many advantages, ranging from time saving, constant speech monitoring, reducing the subjectivity of manual disfluency identification [28, 18] as well as a more effective automatic speech recognition. Therefore, the identification of "episodes" of stutter will offer actionable information [5, 17, 18]. Reliable identification of speech segments, from the beginning to the end of episode, with pauses, ***blocks***, ***interjections*** and hesitations will enable speech cleanup by ridding a speech of the ***blocks***, and ***interjections***, and by smoothing hesitation segments and shortening the prolongations [19, 21, 31].

Our work flow consisted of multiple components: Candidate segment extraction, extracted segment analysis, feature building, segment labeling, classifier training and testing, scoring and remediation.

We used a number of approaches and methods to accomplish the task. Our research relied on ***Praat*** which is a linguistic tool for sound and speech analysis, ***Python*** for

data preparation and ***R*** for classifier training, during ***Approach Spectral-analysis*** we used TensorFlow/Keras for ***CNN*** model training.

With *Praat* used to listen, examine, annotate, and extract our training features, we performed the exploratory work manually followed by scripts to streamline the extraction of speech segments and features.

Our research and development has confirmed that we can deliver much improved speeches as a result of the detection then remediation methods we used. The repaired speeches were superior to the original speeches in terms of intelligibility and quality, afterwards we experimented with multiple methods to detect and repair stuttered speeches. Our work was divided into three approaches, the first two detect and remedy ***block*** speech stutter and the third one detects and repairs speech ***interjections***. The techniques used for each approach are different and we will describe them separately.

1.1 Introduction - Approach Pre-qualification

In the first approach we implemented a pre-qualifier script to create the potential speech **block** segments [27] to label by listening and tagging each one for *Deletion* or *Retention*. After labeling, we extracted raw features: (*Formant*, *Pitch*, *Intensity*) and created the dataset for classifier training. The best classifiers were chosen to predict the remediation segments. This part of the research is distinguished from the other two in the fact that the dataset samples were based on a pre-qualification stage.

With ***Approach Pre-qualification*** we ran a script that identified all audio segments that might consist of ***blocked*** utterances where the speaker was unable to produce intelligible sounds. This pre-qualification was done by identifying segments with two

low intensity thresholds and four sound durations thresholds [37] to identify those that match the pre-qualification criteria. Multi threshold abnormal segment extraction offered an interesting way to look at the same recordings from different perspectives.

We designated speeches with manually labeled segments as the Gold Standard and set out to train various classifiers for high accuracy performance. The results we attained with our segment labeling were encouraging.

We used the linguistic tool *Praat* to identify the segments of speech with the potential to be ***blocked*** speech episodes. The feature engineering work and the segment labeling components were developed with ***Python NumPy***, and, in this approach, the Classifier training and prediction were developed in R.

After iterative adjustment of the **block** segments extraction intensity and duration we identified the optimal intensity and duration ranges of the ***blocks*** candidate segments extraction.

To avoid removing speech snippets by error, we introduced a filter based on a trained classifier to separate the truly *blocked* speech segments marked for deletion from the *falsely* identified candidate segments.

In addition to training our classifier, the labeling results were used to confirm that deleting ***block*** sound segments improves the overall recording quality with no unwanted side effects.

By eliminating the segments that were labeled as *blocks* i.e. *Positive* we reached substantially better speeches which were nearly void of ***block*** stutters.

Recordings remedied according to manual labels (ground truth) were considered the *Gold Standard*; and subsequently repaired speeches would be compared to the

Gold Standard.

To train the classifier for automatic ***block*** detection and then speech remediation, we analyzed the speech segments and created multiple vectors for statistical analysis.

Our decision to select the right sound parameters i.e. formants, pitch and intensity with statistical analysis values was based on visual examination of these parameters for *blocked* and *non-blocked* segments. By extension, we observed that the variations between consecutive analysis vector values would be material to the classifiers' performance and chose to introduce derivative vectors and their statistics.

1.2 Introduction - Approach Random-sampling

Approach Pre-qualification yielded encouraging results but we chose to explore other approaches for two reasons:

(1) As we considered more complex disfluency types such as speech ***interjection*** [12] the difficulty of building the *rules-based* pre-qualifier would be challenging because it is not evident how one would design such rules for other speech disfluencies. Such pre-detection systems can become arbitrarily complex. Without a list of potentially disfluent segments the ***Approach Pre-qualification*** solution would not work, it would have no manual labeling list of segments to work from.

(2) Secondly, we wished to scan a recording with a classifier and score all contained speech segments. ***Approach Pre-qualification*** would proved ineffective for a sliding window application because the process is dependent on a limited pre-qualified samples universe for training.

The ***Approach Pre-qualification*** classifiers are trained on a reduced sample set of

near disfluent segments, whereas ***Approach Random-sampling*** is designed to bypass this limitation through an all inclusive random segment sampling.

Note that the terms "samples" and "sampling" in this document refer to the speech segments selection for model training and rarely to the digital signal processing (DSP) sampling process.

To accomplish the desired outcome for this new ***Approach Random-sampling***, the labeling mechanism was modified to eliminate the need for the preprocessing stage.

The new approach will be able to tackle new types of speech quality detection easily, since research can be undertaken with a single scoring stage without the pre-processing requirements and no multi-component dependency.

Approach Random-sampling, allows the creation of a training dataset samples which do not require the pre-qualification function, they will be based on random segment extraction and manual labeling with a simple training dataset creation process.

Although ***Approach Random-sampling*** simplified the creation of the pool of segments for the label creation process, the labeling process became more complex. The additional labeling effort stemmed from the fact that the labeler has to review a larger selection of segments because many of the randomly selected segments might not be easy to tag as ***normal*** or disfluent - they would be tagged as undetermined therefore dismissed. Unlike ***Approach Pre-qualification*** nearly half the random sample segments were labeled undetermined.

To address this difficulty in the labeling process we developed a new technique which isolated the ***block*** and ***normal*** utterances in separate recordings which contain

only one class or the other.

With ***Approach Random-sampling*** we successfully deployed a sliding window classifier.

1.3 Introduction - Approach Spectral-analysis

As we expanded the scope of our disfluency types detection and remediation from speech ***blocks*** to ***interjections*** our classification requirements became more complex. Detecting speech ***interjections*** could not be performed with the statistical aggregation of *formants*, *pitch* and *intensity* vectors. The ***interjection*** detection challenge with ***Approach Pre-qualification*** and ***Approach Random-sampling*** features necessitated that we explore new data features and classifier algorithms. The new data features we put to the test were spectral analysis results. The new features would be two dimensional matrices with time in the x-dimension and frequency for the y-dimension.

We used three different types of spectral analysis data: Spectrograms, cochleagrams and Melspectrograms. The statistical aggregations were no longer necessary because we used neural network classifiers designed for two dimensional matrix analysis. To analyze the new data we began with Deep Neural Network classifiers and quickly moved to (***CNN***). We experimented with multiple ***CNN*** models.

We trained and tested a number of ***CNN*** models on each of the three types of spectral analysis data; the neural networks we used were inspired by LeNet [29, 24], AlexNet and VGG ***CNN*** models. The performance of the spectrogram types and ***CNN*** models are presented later in this work.

Due to the nature of ***interjections*** which we intended to remove from a recording, which are surrounded with "silent" pauses, it proved practical to segment our recordings into "silent" and "sounding" utterances. To create our data sets we used the technique where two types of recordings are created, one with a series of ***interjections*** with brief pauses in between and the other with the original recording minus the ***interjections*** and pauses in between. This approach allowed us to expand the *positive* and *negative* data pools we used to create our training datasets.

After trying various ***CNN*** models and feature types we were able to identify the best spectral data type/***CNN*** classifier model combinations which were implemented and used for speech remediation on new speaker speeches. The results showed substantial improvements over ***Approach Pre-qualification*** and ***Approach Random-sampling*** features repaired and original recordings. In order to further improve our classifier performance, we manually cleaned the resulting ***interjection*** free and ***interjection*** subsets to supplement and enhance the training datasets.

This iterative training/testing dataset expansion based on *ground truth* data feedback proved useful in improving the performance of the re-trained classifiers.

In the future, additional experimentation should be performed where the remediation process goes beyond simple segment deletion. One option we find particularly interesting is the possibility of speech remediation via spectrogram matrix manipulation.

CHAPTER 2: LITERATURE REVIEW

Disfluency detection is a multidisciplinary exercise which encompasses linguistics, DSP and computer science; more specifically machine learning. In this section, we discuss the general direction of speech disfluency detection evolution with prosodic and textual data, and with regards to the signal analysis techniques and features extraction.

Different classification algorithms and parametrization are used in disfluency detection along with multiple speech analysis techniques for stutter detection and various speech corpora are used by researchers.

Our ability to digitally sample speeches with increasing feature sophistication and the advent of algorithms capable of classifying speech snippets have fueled considerable research in the area of voice recognition and other related exploration.

Speech segments have been analyzed and classified with a wide range of classifier algorithms such as Random Forest, Adaptive boosting, *HMM* (Hidden Markov Model), *ANN* (Artificial Neural Network), SVM (Support Vector Machine), *LDA* (Linear Discriminate Analysis). Since the resurgence of Deep Neural Network (*DNN*), we have seen considerable reliance on *DNN* in speech analysis research especially (**CNN**) [11, 30, 1].

For stutter detection various types off speech analysis such as *LPC*, *MFCC*, Pitch,

Intensity and spectrograms of different types have been used.

2.1 Disfluency Detection Approaches

2.1.1 Prosodic vs Lexical

Disfluencies in speech can be recognized based on a speech's textual content [18, 34, 39] or prosodic features [10, 20, 26].

Lexical features are based on the textual content of the speech. *Lexical* or syntactic research attempts to identify disfluency by analyzing the semantic content of a sentence, word, phoneme repetition and ***interjections*** and breaking down a speech into its textual parts.

Prosodic features which consist of intonation, acoustic, pitch and intensity aspects of a speech tend to be of special interest because they enrich one's ability to convey the intended sentiment, meaning and context, they also mark sentence and paragraph boundaries.

In prosodic detection of disfluency, the acoustic nature of the speech is considered the primary cue to detect the presence of disfluency within an utterance.

In the literature, we find research performed on both data types, *lexical* and *prosodic*. Many papers discuss the advantages of using both simultaneously but few stress the value of the lexical approach without prosodic cues [34].

Prosodic analysis for disfluency classification is performed by extracting and classifying speech segments through acoustic features such as formants, pitch, intensity, spectral components such as *MFCC*, Cochleagram [8, 35] and spectrograms [20] .

From the perspective of applications, the lexical analysis of a speech is best suited

for content recognition and the prosodic analysis of speech is used for the softer qualities of speech such as:

- Structural tagging such as finding sentence boundaries.
- Paralinguistic tagging used for classifying dialog acts and emotions identification
- Speaker recognition

Although disfluencies like ***interjections***, and hesitation are lexically identifiable from a correct word transcription, in the absence of accurate speech recognition, prosodic disfluency detection is necessary [32] for adequate results.

The body of literature tends to consistently lean in favor of analyzing prosodic speech features to detect disfluencies.

The following is a sampling of claims and quotes by a number of authors in support of the prosodic approach:

- Recognizing the lexical aspect of disfluency is critical to speech recognition [27]
- Although disfluencies like ***interjections***, and hesitation are lexically identifiable from a correct word transcription, in the absence of correct word transcription or speech recognition, prosodic disfluency detection is necessary [32]
- The prosodic approach is immune to speech recognition errors when detecting disfluencies [13]
- Prosodic disfluency detection takes durations of pauses and ***interjections*** into consideration, such features enhance our ability to determine whether an ***interjection*** or pause are present in a speech

- Prosodic speech analysis takes into consideration a much larger range of features than semantic analysis and provides a richer understanding of speech.

2.2 Speech Signal Processing

Speech disfluency detection, like all other voice recording analysis, is based on DSP where the speech signal is captured at a sampling rate high enough to maintain good sound quality. A discussion of DSP is beyond the scope of this dissertation but we list a few commonly used DSP features:

MFCC or Mel-frequency cepstral coefficients are values that collectively make up a vector derived from the deconstruction and reconstruction of a sound signal according to frequency bands which approximates the human auditory system's response.

The cepstrum of a signal is defined as the squared magnitude of the inverse Fourier transform of the logarithm of the squared magnitude from the Fourier transform of a signal.

In the area of disfluency detection, the *MFCC* information is utilized in a manner which highlights other aspects of an utterance such as the correlation between *MFCC* contents of time frames and the *MFCC* vectors standard deviation among other extracted features.

In the case of prosodic detection of disfluency, the *MFCC* is frequently used in conjunction with other acoustic non-spectral features. *MFCC* is one of the more commonly encountered features in speech recognition and disfluency detection [2, 5].

Linear predictive coding (LPC) is a method used to compress the spectral information for efficient storage or transmission. The basic idea is the source-filter model

where the source signal produced by the oscillation of the vocal folds is modified by the resonances determined from the morphology of the vocal tract and oral or nasal vocal cavities. This morphology acts as a filter on the signal.

LPC analysis estimates the vocal tract resonances from a signal's waveform; one gets information about the resonance features of the filtering vocal tract and cavities. The *LPC* produces the *formants* which make up an utterance [2, 6, 28].

Pitch perception considers the fundamental frequency that is the lowest divisor on the frequencies contained in a sound [7, 22, 23]. In [43] the authors provide a number of pitch-related/derived feature extraction methods, where pitch is stylized and manipulated to extract information such as pitch regression coefficient (PRC), pitch modulation variance (PMV), relative pitch ratio (RPR), etc. These pitch extractions are then used in the classifier to distinguish fluent from non-fluent speech segments [43].

Intensity/Energy: Sound intensity is the amplitude measurement of sound. The bigger the sound oscillations the higher the intensity. In fact, the intensity corresponds to the absolute integral under the sound wave.

Spectrograms are time-frequency intensity display representations of a sound frequencies spectrum taken at short sound time windows. Spectrograms are also referred to as Sonograms, A spectrogram is computed by dividing the signal into equal overlapping chunks and computing the frequency content for each [36].

Cochleagrams are variations of the spectrogram. It utilizes a gammatone filter and has been shown to better reveal spectral information. It is an auditory spectrogram where the frequencies are represented according to the human hearing sensitivity by

mimicking the components of the inner and middle ear where the sound signal is broken up into different frequencies which are naturally selected by the cochlea and hair cells [4].

Melspectrograms are similar to spectrograms except for the frequency bands which are equally spaced on a scale based on a discreet cosine transform of a log power spectrum on a nonlinear mel scale of frequency which approximates the human auditory system's response more closely than the linearly-spaced frequency bands used in the normal cepstrum. This frequency warping can allow for better representation of sound.

Parameterization: The DSP features used in disfluency detection consist of spectral time frame analysis computed over certain frame lengths; for example, one could compute the frequency domain representation of a 10 millisecond speech time frame or a 20 millisecond time frame. Time-frame windows are generally overlapped when stepping from one time frame to the next. The time frame length and overlap, or hop size, are also parameters which are explored in the literature. Depending on the type of classification the time frame length and overlap between time frames as well as the window shape (Bartlett, Hamming etc) have a direct impact on classification algorithms performance. The optimal parameters tend to vary by disfluency type e.g. ***block***, ***interjection***, *repetition* and *prolongation.*

2.3 Disfluency Classification Algorithms

A number of classification algorithms have been applied to stuttering detection with varying performance results. The following is a non-exhaustive list of the algorithms

in the literature.

Hidden Markov Model or (HMM) is a stochastic approach used to recognize patterns in an input signal [46]. *HMM* is an important algorithm in speech recognition theory where the speech signal pattern is recognized in order to determine phonemes and words. *HMM* is a common classifier used for the reliable detection of speech disfluency [5] with prosodic as well as textual features [31]. In [45] the authors describe the use of the HTK (Hidden Markov Model Toolkit) for phoneme repetition detection. In [32], the authors attempted to qualify sentence boundaries, filler words, and disfluencies with *HMM*. [38] is a seminal paper that provides extensive explanation of *HMM* as it relates to speech recognition.

Artificial Neural Networks or (ANN) are an attempt to simulate the network of neurons that make up a human brain so that the computer will be able to learn things and make decisions in a human like manner, it is a network inspired by the human biological neural networks and used to approximate unknown functions which depend on a large number of inputs.

Support Vector Machine or (SVM) is a machine learning algorithm which is widely used in pattern recognition. In SVM classification optimization attempts to obtain a good separating hyper-plane between two classes in high dimensional spaces [5]. The main concept of the *SVM* classifier design is the notion of a margin which separates classes in hyper planes. In [3] the authors use SVM to classify pathological voice disorders using a number of voice extracted features.

Decision trees construct a flowchart-like structure in which each internal node represents a test on an attribute, each branch represents the outcome of a test and each

leaf node represents a class label. In the area of disfluency detection, decision trees are commonly used with prosodic speech features [32, 33, 22, 23, 34].

Gradient boosting Machine or (GBM) is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models which are combined into a single strong learner in an iterative fashion [9].

Random forests are an ensemble learning method for classification that construct a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random forest classifiers address decision trees tendency to over-fit to the training set. Random forests are a way of averaging multiple deep decision trees, trained on different parts of the same training set, with the goal of reducing the variance typically introduced by single trees.

Linear Discriminant Analysis or (LDA) is a generalization of the Fisher's linear discriminant and is a method used in statistics, pattern recognition and machine learning to find a linear combination of features that better separates two or more classes of objects. *LDA* estimates the relative strength of detection features and is a frequently used algorithm for the detection of speech disfluency [2, 6, 13] .

k-Nearest Neighbors (kNN) is a non-parametric algorithm used for classification. In *kNN* the minimum distance between query instance and each of K training set samples is calculated to determine the proper class based on distance proximity. We encounter *kNN* frequently in stutter detection research [2, 6].

2.4 Speech Corpora

A number of readily available speech corpora are used in the field of disfluency detection:

Switchboard: *Switchboard* is a collection of almost 2,400 two-sided telephone conversations among 543 speakers (302 male, 241 female) from all areas of the United States.

Verbmobil: *Verbmobil* was a long-term interdisciplinary Language Technology, especially in the Machine Translation, research project to develop a system that could recognize, translate and produce natural utterances.

UCLASS: *UCLASS* [21] consists of speech recordings collected at the University College London. These recordings are from children referred to clinics in London for stuttering assessment. The recordings and annotations are available in software formats widely used for language and speech analysis such as CHILDES, *Praat*, and SFS.

CHAPTER 3: APPROACH PRE-QUALIFICATION

3.1 Approach Pre-qualification

To develop a system capable of analyzing stuttered speeches for identifying segments of speech which can be removed and enhance recording quality we used the UCLASS recordings which consist of over 100 stuttered speech sound files by young stuttering speakers.

There exist multiple types of stutter disfluency types: speech ***block***, ***interjection***, *prolongation* and hesitation, we chose to detect and remedy two types of stutter disfluencies: Speech ***blocks*** and speech ***interjections***.

During this approach, a ***Praat*** script extracted our segments, ***Python*** with ***Jupyter Notebook*** was our data manipulation environment with heavy reliance on *Numpy* and *Pandas* libraries and ***R*** provided all machine learning packages and integration.

Figure 1 shows the entire ***Approach Pre-qualification*** Process Flow.

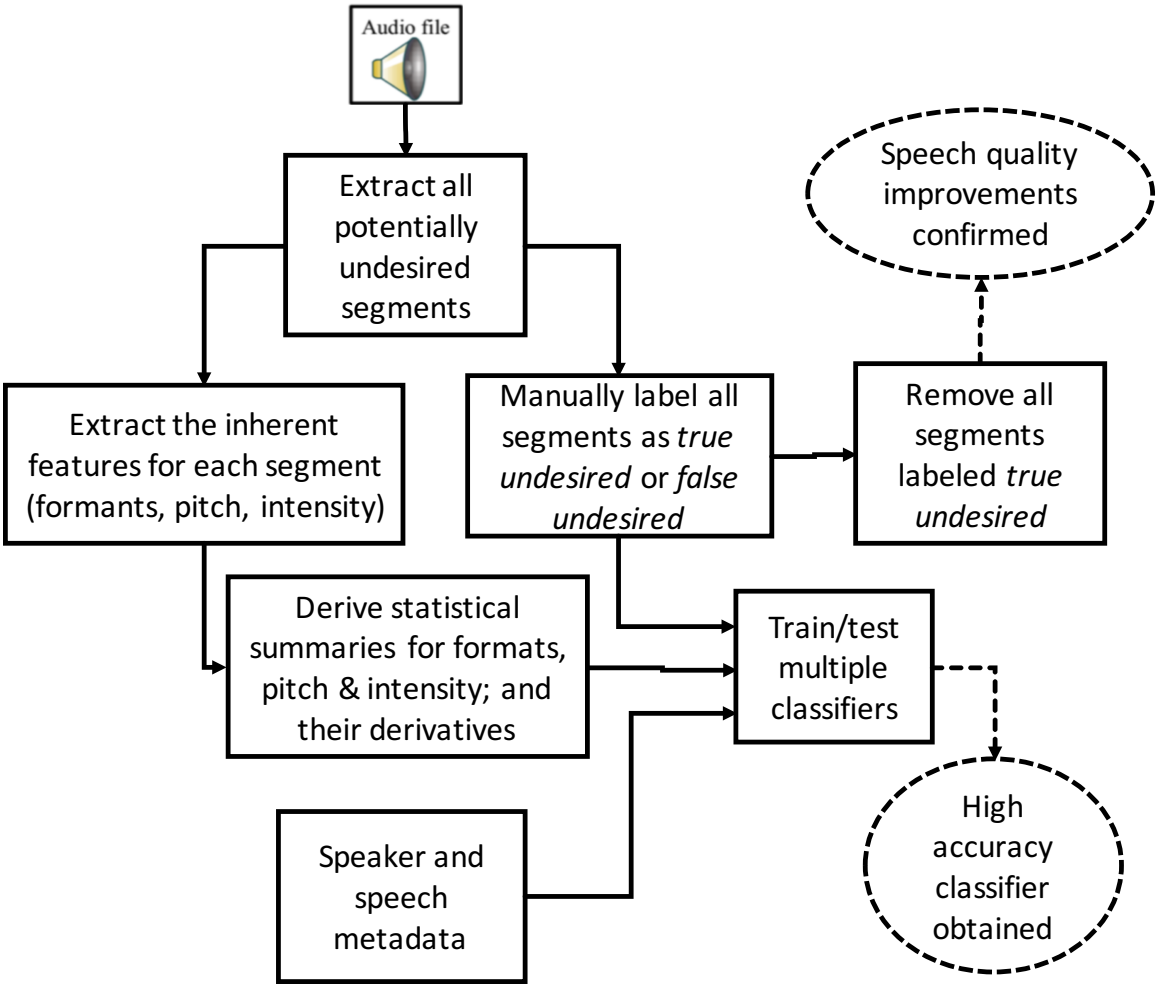


Figure 1: Approach Pre-qualification - Process

3.1.1 Approach Pre-qualification - Segment Extraction

During ***Approach Pre-qualification*** we implemented a *Praat* pre-qualifier script which created the candidate speech ***block*** segments [27] which were selected for their ***blocks*** similarity.

The ***blocked*** speech segment candidates were created by a script designed to detect silent segments that identify weak segments within a certain intensity range.

To identify episodes with medium to high likelihood of *blocked* utterances we reviewed the pre-qualification results of various speech volumes and determined the

sound intensity thresholds above which a speech would not qualify as a *blocked* speech and adjusted the script to extract segments within those threshold boundaries [42]. We used 90% and 95% decibels of the entire speech overall volume as thresholds.

The second parameter of the extraction script was the minimum segment duration threshold because deleting excessively short durations of a natural speech deteriorated overall speech quality. This was confirmed as we listened to remedied speeches following the removal of varying lengths segments and observed the overall effect of segments' removal by duration. Deleting segments of shorter duration than 0.6 *seconds* disrupted speech cadence and proved counter productive thus the minimum threshold of 0.6 *seconds* [41]. Similarly, through trial and error, we determined an upper limit threshold duration of 1.2 *seconds*.

We visually reviewed the graphical characteristics of numerous signal analysis charts to decide which types of analysis detect speech ***blocks*** best. This approach proved adequate to decide which signal analysis we should consider for classification, and segment removal.

The pre-qualification stage provided us the full set of segments from which we created our training and later the testing dataset. Each pre-qualification scan was focused on a *maximum intensity* and *minimum duration* threshold combination. With two intensities and four minimum duration thresholds we scanned the recording eight times and generated 8 candidate segment sets which would be combined for manual labeling and model training.

In Table 1 we show how the lower threshold (90% of average) resulted in approximately 10% less disfluent candidate segments.

As stated earlier, we used two different thresholds for the sound intensity. The lower threshold (90% of average) resulted in less undesired candidates (Table 1) but most of the undesired candidates extracted using the lower threshold were actually true undesired, there were many instances where stutter segments were not detected due to the low threshold level. On the other hand, the higher threshold (95%) resulted in a higher number of potentially undesired segments, some of which were false undesired; however, most of the stuttered segments were detected in the first phase. Therefore, the value of the second phase (classifier scoring), which identifies true undesired and false undesired from potentially un-desired segments, is most useful when the threshold is high, and most of the actual stutter segments are detected, even if that meant falsely pre-qualifying segments as potentially undesired.

As stated above the classification is most meaningful and effective when the intensity threshold is high because the elevated threshold included more borderline segments and we relegated the task of distinguishing between the disfluent and non-disfluent episodes to the classifier.

The second threshold type we used during pre-qualification was the minimum time duration. The minimum segment length started with 0.6 seconds to 1.2 seconds with a 0.2 second jumps.

We have subsequently added a voting component to this system of remediation which helped enhance the usability of the system.

The varying thresholds approach can be likened to examining speech segment from multiple perspectives in order to acquire a fuller view of the object.

Once the segments are identified, we need classifiers to tag them for deletion or re-

Table 1: Segment count by duration/intensity thresholds

	90% of speech avg sound volume threshold	95% of speech avg sound volume threshold
0.6 seconds	195	211
0.8 seconds	149	159
1.0 seconds	114	130
1.2 seconds	83	93

tention. From a classification training and prediction perspective, every segment will be represented by a tuple to be classified for removal or retention. Every tuple must contain a set of features and our dataset will consist of a sample segment identifier followed by predictors deemed important for the removal or retention decision of a segment. With reliance on prior work in this domain, and, after experimentation we decided to analyze, and base ***Approach Pre-qualification***, on: *Formants*, *Pitch* and *Intensity*.

Fifteen speeches (one hour of stuttered speech) which were chosen for their high ***block*** content, were selected from the UCLASS repository and scanned with 8 threshold combinations each. The minimum duration thresholds were 0.6, 0.8, 1.0 and 1.2 seconds, and the speech average intensity thresholds 95% and 90% of the recording intensity in decibels; i.e. for an average speech intensity of 50.0 decibels, the intensity thresholds would be 47.5 dB SPL and 45 dB SPL respectively (see Algorithm 1)

Algorithm 1 Extract candidate segments

```
procedure Praat Segment Extraction and analysis(segments, vectors)  ▷ Acquire and analyze candidate segments
    fileName ← designatedfilename
    for <all duration thresholds> do                                ▷ Acquire candidate segments
        for <all intensity thresholds> do
            scan entire speech
            extract criteria fitting segments
        end for
    end for
end procedure                                                       ▷ Complete segment extraction
```

The entire speech was pre-processed and the *potentially undesired* candidates were detected according to the threshold combination pairs.

3.1.2 Approach Pre-qualification - Segment Analysis

The classifier training features consisted of *Formant*, *Pitch*, *Intensity* vectors statistical summaries. Figure 2 shows a sound signal with the formants, pitch and derivative values.

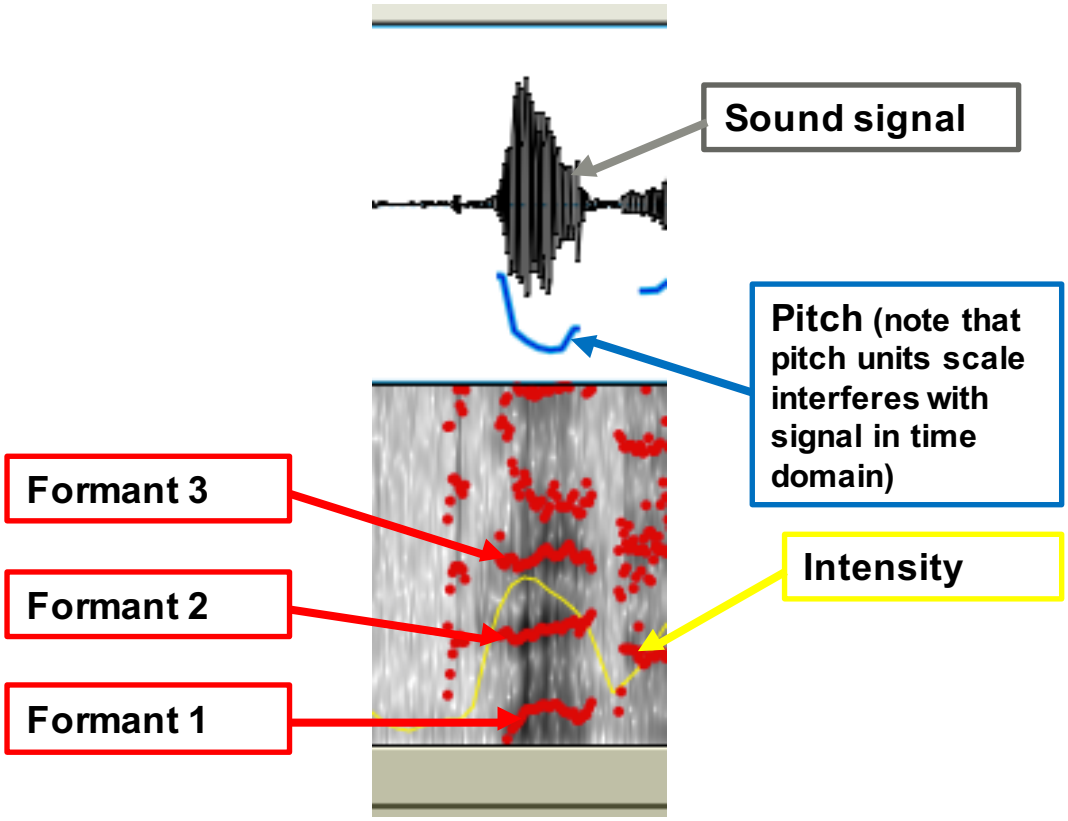


Figure 2: justification=centering

Up to this point our analysis has created *.wav* files and the raw feature vector files and derivative vector files. The *.wav* files represent the samples considered for removal and the statistical data are the model training predictors.

With *Praat*, one could compute up to 6 formants through *Formant* analysis but,

because of the low consistency and availability of formants 4, 5, 6 we limited our research to formants 1, 2, 3.

The formants were computed with a 50 ms Gaussian analysis window and a 12.5 ms offset [14, 17]. We additionally generated *Pitch* and *Intensity* analysis. The *Pitch* values were taken at $10ms$ interval and computed with the *Praat* script which performs an acoustic periodicity detection on the basis of an accurate auto-correlation method [5].

The intensity values were computed by squaring then multiplying the sound values within a Gaussian analysis window of 50 ms length.

Algorithm 2 shows a pseudo code of the Format, Pitch, Intensity extraction

Algorithm 2 Extract Segment Parameters

```
procedure Praat SEGMENT ANALYSIS(segments, vectors)        ▷ Analyze candidate segments
    fileName ← designatedfilename
    Select Formants analysis frame length and hop size
    for <all extracted segments> do                        ▷ Analyze segments
        Perform Formants 1,2,3 analysis
        Save Formants (1,2,3) vectors
    end for
    Select Pitch analysis frame length and hop size
    for <all extracted segments> do                        ▷ Analyze segments
        Perform Pitch analysis
        Save Pitch vector
    end for
    Select Intensity analysis frame length and hop size
    for <all extracted segments> do                        ▷ Analyze segments
        Perform Intensity analysis
        Save Intensity vector
    end for
    return All analysis vectors
end procedure                                              ▷ Complete segments analysis
```

Figure 3 shows the process we used to reach the best segment signal analysis types (Formants, Pitch, Intensity) for each segment.

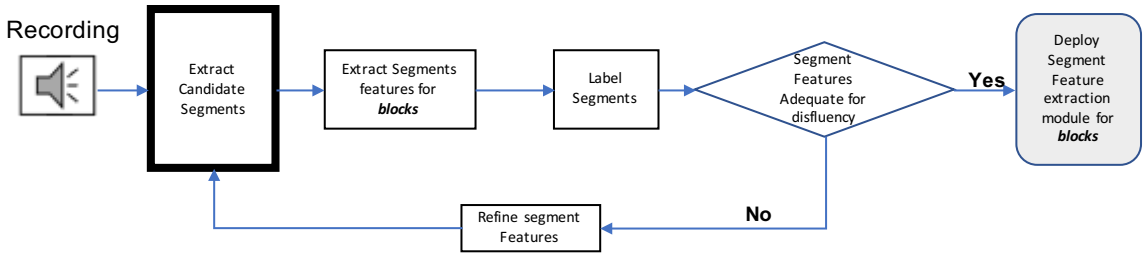


Figure 3: Segment Features Determination Process Flow

3.1.3 Approach Pre-qualification - Segment Statistics

The pitch analysis generated one file per segment; this also was statistically analyzed to create additional features.

Each extracted candidate segment was analyzed by a raw feature analysis *Praat* script designed to extract the *Formants*, *Pitch* and *Intensity* each of which is recognized as a strong factor while determining the presence of speech disfluency.

All three analysis formant (F), pitch (P) and intensity (I) generated separate files to be later fetched and statistically analyzed as our predictor features. We split the formants file into three separate vector files F1, F2, F3.

Up to this point we had five vectors to extract statistical summaries from. The five vectors formant 1 (*F1*), formant 2 (*F2*), formant 3 (*F3*), pitch (*P*), and intensity (*I*) consisted of a sequential list of measurements for each of the five analysis vectors.

As we visually reviewed the raw features the *blocked* speech segments exhibited erratic jumpiness. To capture this signal jitter which is characteristic of ***blocks*** we introduced the derivative vector for each of the five foundational raw analysis features, and ended with a total of ten vectors for statistical analysis: *F1*, *F2*, *F3*, *P*, *I* and derivative vectors *F1d*, *F2d*, *F3d*, *Pd* and *Id*.

We conducted manual graphical reviews of the candidate speech segments to develop a qualitative understanding of important characteristics in five vectors.

Time base plots of formants, pitch and intensity helped determine the best ways to recognize the distinguishing behaviors from the signal analysis vector; we settled on the following: ***Average***, ***median***, ***Standard Deviation***, ***25 Percentile***, ***50 Percentile*** and ***75 Percentile***, ***Vector minimum***, ***Vector maximum***, ***Peak-to-peak Amplitude***, and ***Variance***.

The signal analysis vectors associated with each candidate segment vector were between 50-200 scalars for the formants, 60-240 for the pitch and 70-300 for the intensity. The statistical analysis were performed with the *Numpy* library and shown in Algorithm 3.

Algorithm 3 Compute statistical summaries of all segment vectors

```
procedure Python Feature development(Dataset)                    ▷ Create features dataset
   fileName ← designatedfilename
   for <all formants (1,2,3) vector files> do
      Fetch formant vector files into lists
      Build formant derivative lists(3)
   end for
   for <all pitch vector files> do                               ▷ look at all pitch vector files
      Fetch pitch vector file into list
      From pitch list build pitch derivative list
   end for
   for <all intensity vector files> do                           ▷ look at all pitch vector files
      Fetch intensity vector file into list
      From intensity list build pitch derivative list
   end for
   for <All segments DSP and derivative vectors> do
      build list of averages
      build list of medians
      build list of standard deviations
      build list of percentiles 25, 50, 75
      build list of maximums
      build list of minimums
      build list of peak to peaks
      build list of variances
   end for
   Merge all lists into one list (Features)
   Fetch all UCLASS metadata into one list (Metadata)
   Fetch all Segments sample ids as one list (Support)
   Concatenate Metadata, Features and labels into one training dataset)
   return All analysis vectors
end procedure                                                    ▷ Complete the feature dataset build
```

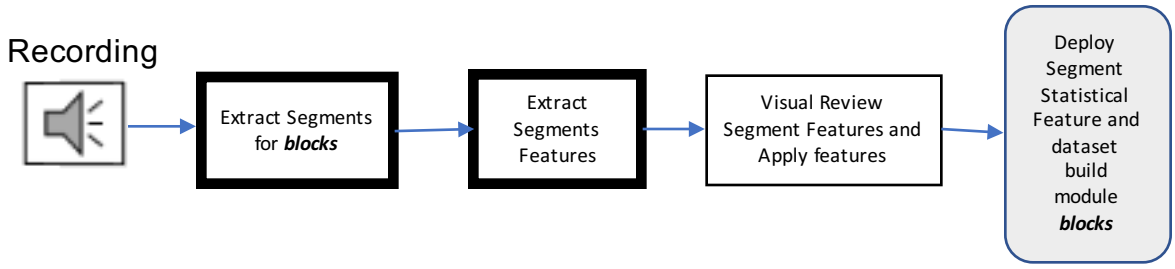


Figure 4: Dataset Features Determination

Figure 4 highlights the process steps we followed to develop the statistical aggregation types.

The *UCLASS* corpus provides speech and speaker information. For Approach Prequalification we supplemented our dataset columns with these UCLASS metadata information as they became additional predictors. The UCLASS speaker and speech information consisted of the following features:

Speaker Information: (1) Gender (M/F), (2) Handedness (L, R, not known), (3) Past history of stuttering in the family, (4) Age of stuttering onset and (5) Age at the time of recording

Speech Information: (1) Location where recording was made (clinic, UCL, or home), (2) Recording conditions (quiet room or sound-treated room), (3) Type of therapy received (family based treatment or holistic treatment), (4) Time between therapy and recording time, (5) Speaker had any history of hearing problems (Y/N), (6) Speaker had a history of language problems (Y/N), and (7) Special educational needs (Y/N).

For the remainder of this document we will refer to speech and speaker data as metadata. Periodically we will discuss their potential role in the current research.

3.1.4 Approach Pre-qualification - Labeling

Up to this point, our research is predicated on the assumption that the candidate segments the *Praat* script extracted would be manually reviewed and labeled to confirm the speech improvement from removing those stuttered ***blocks***. If our hypothesis is proven correct and removing ***block*** segments yields better recording quality, then we have justification to proceed and automate the scoring process followed by selective segment removal accordingly.

Algorithm 4 shows the iterations of removing segments from a speech

Algorithm 4 Remove all bad segments

```
procedure SPEECH MINUS ONE SEGMENT(Speech Id,Start time, End time)
    remove per start, end times
end procedure                                   ▷ Complete speech minus one segment
dataset ← desireddataset
sort bad segments file in decreasing end of segment order
for <all items in dataset, starting at end of speech file> do      ▷ scan from end to start
    if <not contained in previously removed segment> then
        remove segment
        save sound
    end if
end for
```

Throughout our extraction efforts the segments were not labeled. To label the speech segments, we created a program to sequentially playback the audio files.

As the segment files were played back each sound was labeled by the user according to the following criteria:

1) Delete and train (*G*): This label indicates that the *.wav* file is void of semantic meaning and the remediation program must delete it; accordingly we wish to train our classifier to recognize such segments as *positive* and mark them for *removal* (see Algorithm 5).

2) Retain the segment in the speech and use it for training (*B*): This

label is used for audio sounds which where ***block*** candidates for their low intensity and long enough duration, yet they contain semantic meaning and therefore must not be removed from the original speech, and our classifier should learn to label such segments as *negative* e.g. *retain* them in the speech.

3) Delete the sound from the speech but do not use it for training (*D*): This label indicates that a segment should be deleted from the dataset because our trained model classifier must not learn its features as either *positive* or *negative*. An example of a *D* label is a segment of the interviewer's soft voice which is not stuttered. We chose to exclude these segments from the speech during cleanup because they showed no speech disfluency, so we refrained from including them in the training dataset. These segments were not frequent and mainly of limited consequence on the overall process.

4) Ignore the segment, leave it in the speech and exclude it from the training dataset (*I*): This label meant that we chose to retain a segment in the speech but had no intention to analyze it. We elected to bypass the analysis of such episodes because the ***block*** stutter segment disfluency was not pronounced enough to properly train the classifier; they are border line segments. Training with such segments would not improve the classifier, but rather confuse it. As in the case of *D* tagged segments, these snippets of recording are immaterial to the quality of our classifier and we left them in the speech.

Algorithm 5 Listen and label segments

```
procedure ORDER SEGMENTS(OS Folder, list)                          ▷ scan folder, create list
    segList ← Files, Start, End
end procedure                                                      ▷ Complete fetching list of all segment files
sort list
for <Items in sorted list> do                                                          ▷ Listen and label
    if <contained in previous segment> AND <previous segment marked for removal> then
        rename segment file with removal label
    else
        play the segment
        accept user label for segment
        rename segment file with selected label
    end if
end for
```

3.1.4.1 Approach Pre-qualification - Labeling Optimization

Due to the multi-scan nature of our segment extraction approach, the number of candidate segments generated is nearly eight fold the segments count from a single scan because of the eight threshold scans; this makes for a large number of sound snippets one must review and label individually. To avoid redundant labeling and minimize the number of segments which must be reviewed, we sorted the list of segments such that a segment $\boldsymbol{S}$ which is marked for deletion would eliminate the need to review contained segments and they will all be marked as delete. The prevailing logic was that if some segment s is void of semantic meaning, then all other segments contained within s must also be void of semantic meaning.

During the multi-scan pre-qualification, segments identified from various thresholds tended to overlap, see Figure 5 for a visual depiction of the multiple segment extraction layout.

Note that the opposite is not true, given that some segment s contains some semantic meaning, one must not conclude that segments within it carry semantic meaning.

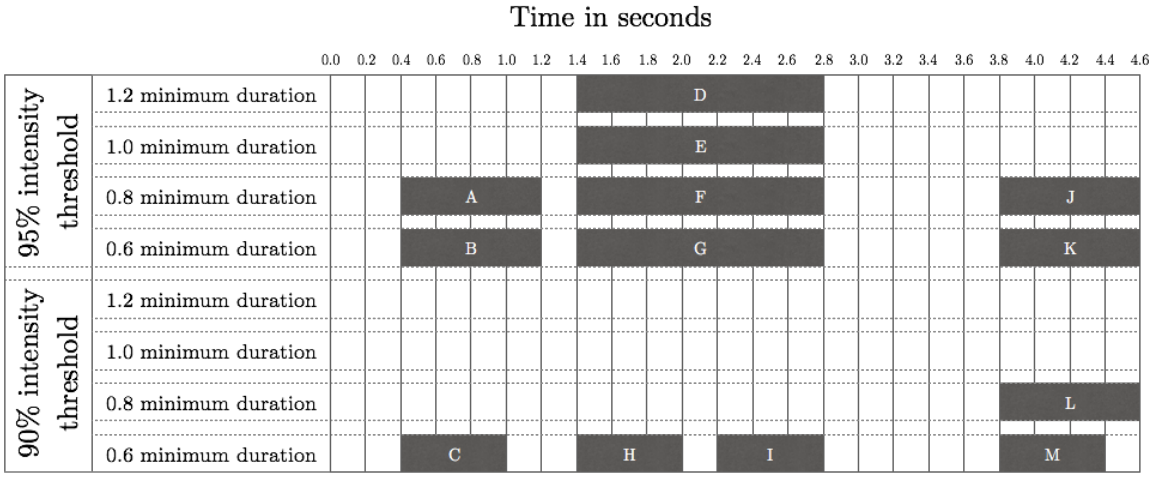


Figure 5: Candidate Segment Overlap Illustration

This streamlined labeling approach allowed us to reduce review time by nearly five fold.

With the labeling options the training and the classification will expect and deliver two class values: G (positive) and B (negative). The positive class indicates a candidate speech segment which must be removed, whereas a negative class means the segment must remain in the speech.

There are three different courses of action for segments removal, each with different implication:

1. **Remove all candidate segments without additional qualification**: This approach does not require manual labeling, and the only action is to remove every pre-qualified candidate segment. This limits our ability to distinguish between truly disfluent candidates and those false candidates i.e. they are not truly disfluent ***blocks***. With no mechanism to avoid removing false positive candidates repaired speeches fell short of the intended remediation and had a

high loss of information due to the semantically valuable segments removal. This option would not be considered a viable one.

2. **Remove only the candidate segments which are manually labeled for deletion**: This approach consisted of deleting all the G and D segments. The resulting speech is essentially based on the best indicators according to manual labeling and would yield the best quality of recordings; we will refer to recordings repaired in this fashion as the *Gold Standards*. Implementing this option offered excellent results; no semantically significant recording snippets were removed and all insignificant portions were deleted. This option requires manual labeling of each speech prior to remediation and cannot be used for automatic repair due to the persistent manual labeling requirement.

3. **Remove candidate segments determined disfluent by a trained classifier**: The third and only plausible approach was to train classifiers with the labeled dataset. After successful training and testing of multiple classifiers we deploy the highest performance model to score segment labels and remove the candidate segments if *true undesired*. An accurate classifier makes this approach the one viable option since it allows the automation of a new speech remediation.

Since Option 2 results helped ensure that the ***block*** removal remediation offers the anticipated improvements we have assurance and motivation to create an automated system with a trained classifier to perform the ***block*** detection and removal process.

As we dismiss Option 1, we use the Option 2 (*Gold Standard*) speech as the success measure benchmark of our subsequent work. During the evaluation of Option 3

the repaired by score speeches are compared to the *Gold Standard* recording for confirmation of successful scoring and remediation.

3.1.5 Approach Pre-qualification - Building the Dataset

The second stage of our effort was the classifier training and testing stage to build a trained model that distinguishes the *true undesired* from the *false undesired* candidates.

Even with the 95% intensity and lowest duration thresholds the *Praat* script thresholds tended to be especially effective and caused a *positive* class imbalance in the dataset. After manual labeling of the data, 85 % of our dataset samples were *positive* and the remaining *negative*. To mitigate the impact of the imbalanced class dataset we considered two class samples balancing techniques:

Sampling based technique: This technique includes two primary methods to address the imbalanced data problem, and both rely on sampling approach modification. The first method uses oversampling minority class tuples thus adding more of the minority class samples and give them a higher effect on the learning algorithm. The second approach consists of under-sampling the majority class samples from the dataset by removing a high proportion of the majority class samples (i.e. eliminate positive class tuples). This leads the majority class samples to exert a lesser influence on the machine learning algorithm. We chose the second approach and excluded many over represented class samples randomly for a near equal distribution of ***block*** and ***normal*** samples.

Cost function based technique is where we consider a false positive to be worse

i.e. more expensive than a false negative and equate a false positive to multiple false negatives. As stated above we did not experiment with this technique and relied on the under sampling method to adjust and compensate for the class imbalance.

Throughout ***Approach Pre-qualification*** we have presented balanced dataset classifier performance results. The data balancing we performed limited the ***block*** labeled samples to equal the negative ones and selected all the negative samples then merged the two class subsets into one balanced dataset.

The training of various classifiers will be based on the dataset described above with metadata and statistical summaries.

The dataset consisted of (1) the derived statistical analysis predictors (2) the metadata and (3) the labels which consisted of the manual review *ground truth* observations. With ten statistical analysis per vector and the 12 metadata features plus the training label, the dataset tuples contained 112 predictors plus the ***Ground Truth*** label (see Algorithm 6).

Algorithm 6 Train a model and predict

```
Fetch dataset
Clean dataset
Create balanced dataset
Select classifiers of interest
for <all classifier algorithms in list of desired classifiers> do      ▷ build and test models
   dataset transformation to classifier requirements
   build and tune a model
   test the model
   log and retain model performance
end for
select best performing model
with selected model predict new dataset
export predicted dataset
```

Figure 6 shows the Approach Pre-qualification data merging process.

With Option 2 *Gold Standard* and Option 3 classifier enhanced recording as the basis of our subsequent work, the repaired recordings are examined and compared to

our *Gold Standard* audio file.

3.1.6 Approach Pre-qualification - Classifier Training

Figure 7 shows the logical steps leading to the classifier training component and we provide the pseudo-code in Algorithm 6.

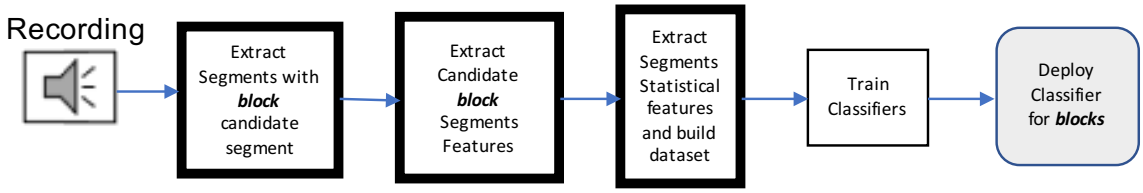


Figure 7: Approach Pre-qualification Classifiers Training Overall Process

With the prepared dataset we trained multiple classifiers in ***R*** where libraries for numerous algorithms are readily available.

In ***R***, classifiers are released by different developers, this leads to wide API variations. To minimize the impact of API inconsistencies on our work we chose to use the *caret* package which offers a common style interface for all supported classifiers.

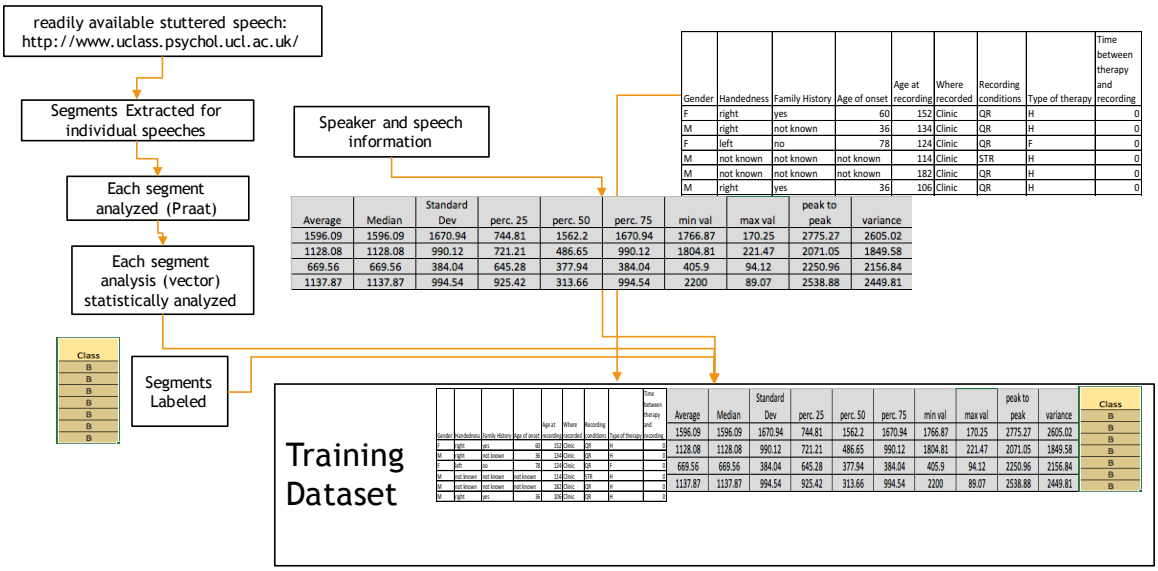


Figure 6: Data Merging

With the *caret* classifiers we were able to train and test multiple algorithms with little account to their underlying API differences because of the common caret interface which gives access to the important training hyper-parameters for each algorithm. With a grid style hyper parameter optimization we created hyper parameter combinations which we iterated through.

Such built-in tuning uniformity simplified experimentation with multiple classifiers and allowed us to test an ever-larger number of algorithms conveniently.

The scope and range of out of the box caret hyper parameters proved effective during our search.

With the large selection of classifiers one could choose from, we tried two or more classifiers from each category: *Decision tree*, *SVM*, *neural network*, and *meta algorithms*.

After training multiple algorithms from each category, we chose the better performing classifiers; those classifiers formed the basis for ***Approach Pre-qualification***. Consequently the classifiers we continued to train and test after a first round of testing were: C5.0, NNet, Recursive Partitioning, Random Forests, Polynomial SVM, and AdaBoost.

In the following remedy improvement discussion, we will use the classifier prediction confusion matrix results as in Table 2 and derived measurements to help frame the proposed solution.

Table 2: Confusion Matrix

		Prediction Outcome	
		p	n
Actual Value	p'	$TruePositive$	$FalseNegative$
	n'	$FalsePositive$	$TrueNegative$

The confusion matrix Table 2 provides general measurements which apply to classification prediction models. We will describe the confusion matrix and measurements as they relate to our proposed remediation environment.

In Table 2 the prediction outcome corresponds to the segment evaluation in our classification, column ***p*** represents a positive scores which means that a sound episode extracted during pre-qualification as candidate for removal is scored as a ***blocked*** utterance segment and must, accordingly be removed from the speech.

The ***n*** column in the Table 2 confusion matrix represents a candidate for deletion which is scored false because of lexical content. The classifier score implies that it must be retained in the recording.

Column ***p*** represents samples a classifier scores as ***blocks*** and must be removed.

Actual Value rows show the results obtained from manual segment labeling. Manual labeling or ***Ground Truth*** are assumed to provide the most accurate results we can obtain.

The ***p'*** row represents an pre-qualified segment manually labeled as ***blocked*** and according to the listener contained no lexical meaning. With a perfect candidate

extraction system, all ***p'*** segments would be predicted as ***p'*** (positive) by our classifier(s) and consequently removed.

The ***n'*** row represents speech candidates presented for removal consideration but when manually reviewed the listener/labeler determined that the segments contain lexically meaningful content and should be retained in the speech.

The confusion matrix is represented by four cells: True Positive, False Positive, False Negative, True Negative at the intersection of actuals and predictions.

The True Positive cell represents the number of candidate segments manually reviewed and determined to be of no value to the speech and scored by the classifier model as "bad", lexically void, segments. This is a case where both the ***Ground Truth*** values and the classifier predictions agree.

True Negative represents the number of segments labeled as lexically meaningful, hence they should be retained, and consistently classified as semantically relevant by the trained model. This also is a prediction outcome that agrees with the ***Ground Truth*** of the segment.

The True Positive and True Negative cells represent accurate classifier outcomes; if all our samples fell in these two categories our classifier would be perfect with an accuracy of 100%.

A False Negative, means that the segment is manually labeled devoid of lexical meaning (e.g. *Positive*) but the trained classifier was not able to detect that and assigned it a class of non-block, to be retained in the recording with no repair to be performed. This is a case of mistaken classification by the model and a required repair is missed.

The False Positive classifications occur when the manual, accurate, label indicates that the candidate segment is a negative segment implying that it contains semantic meaning, yet the score indicates that the segment should be deleted and ***S*** portion of good speech will be deleted. The False Positive cells count represents the total number of samples that fell in that category at time of prediction. These False classifications cause a loss of speech semantic content.

The False Negative and False Positive cells in the confusion matrix represent predictions for we wish to minimize and improve the effectiveness of the remediation..

False Positive conditions lead to speech content compromise due to removal of speech parts which contain semantic information. False Positive mistakes are detrimental to the message and must be avoided if possible, albeit at a price.

Avoiding a False Positive means that we will incur additional False Negatives; it is a trade off because the classifier is generally tuned to minimize the total number of misclassification thus maximizing accuracy. One can assume that further tuning the classifier to minimize the total number of False Positives would result in adding proportionately more False Negatives.

Speech repair quality is not strictly dependent on accuracy, the results must maximize speech quality thus the need for remedy optimization.

Our remedy optimization ensures that little meaning loss is incurred by the process. To this end, we remained cognizant that meaning loss is more "detrimental" to the message conveyed by a recording than the inconvenience of hearing unnecessary ***blocks*** of speech.

Therefore, we explored the available options to help with a remediation process

which causes minimal cognitive meaning compromise. The level of tolerance to collateral lexical loss incurred from our remediation approach and attempts to improve a message prosodic quality were carefully considered.

Deciding our level of tolerance for lexical loss is of central importance to the recording repair. How many bad episodes are we willing to leave in a recording to avoid losing a good segment is an essential question when repairing a speech. Namely, how many false negatives are we be willing incur in order to avoid one False Positive.

To realize our tolerance for False positives, we consider the False Positive Rate (FPR), False Negative Rate (FNR) and Accuracy. False Positive Rate (FPR) represents the percentage of Negative samples that were classified positively as a percentage of the total number of Negative samples.

Although our accuracy was negatively affected by this component, the results of this experimentation proved effective in reducing False Positives.

The lower our FPR, the lesser mistakes our classifier would have made in classifying segments for removal when they should be retained. Therefore, we would like the FPR to be as low as possible; and since we would like to compare our measures to the accuracy which ranges between 0 and 100%, we will consider Specificity SPC instead of FPR, SPC=1-FPR. When working with SPC we would aim to maximize SPC to a high value near 100%.

Also, we consider the False Negative Rate FNR. FNR is the percentage of samples classified as negative when the actual class is Positive; these are segments that must be deleted, however our classifier has classified them for retention. Such mistakes are not as damaging to the speech and will have a lower priority and lower cost. We

would be willing to trade multiple False Negatives for one False Positive. For ease of comparison with Accuracy and Specificity, we will be considering the True Positive Rate TPR = 1 - FNR.

There are multiple ways to perform classifications that tilt the balance in favor of more False Negatives than False Positives (or the opposite).

1. **Cost based training**: This method of tilting the classifier in favor of lower false positives is accomplished by training individual classifiers with higher penalties on False Positives (FP) than False Negatives (FN) thus creating models which are less prone to FPs.

2. **Sampling technique**: This method consists of inflating the number of positive samples in the training sets by repeating positive samples in our dataset, thereby resulting in a higher count of Positive samples and implicitly tilting the classifier towards less FPs. This approach, also, creates classifier models with relatively less FPs and more FNs.

3. **Probability cutoff**: This technique adjusts the probability threshold for classifier models to favor minimizing False Positives as opposed to being optimized to minimize both FNs and FPs combined.

4. **Voting consensus**: Voting based consensus relies on multiple independent classifiers and the use of a percentage of positive classifier scores to negative scores in order to determine a final score.

During Model evaluation we tuned and trained multiple classifiers which performed

well, we chose to use this abundance of trained classifiers, total of 14, in a voting scheme. To implement the fourth technique we experimented with a voting mechanism that utilized all 14 classifier results to reach an optimal FPR.

In this case, we varied the percentage of votes threshold to decide whether a segment should be classified Positive or Negative, i.e. Delete or Retain. In preparation of the results batch data, we averaged the classes from all classifiers and varied the decision cutoff threshold to determine a final class for each sample.

For this experimentation, we used the models that were trained and tuned with the 'caret' package with a balanced dataset described in prior sections. The models' results were combined with the actual label which consists of 15 columns; the one manual label and the 14 classifiers results.

We used the resulting table of 15 columns to evaluate the impact of changing the voting threshold on the Specificity (SPC), True Positive Rate (TPR) and Accuracy. As previously stated, because false positives are especially detrimental to the remediation process, we wanted to maximize the specificity (SPC) while maintaining an acceptable Accuracy and TPR levels. To make FP's less frequent, we intuitively expected that the higher the positive vote threshold we impose on the votes-based classifier, the less FPs (e.g. High SPC) we will get.

As shown in Figure 8 we varied the voting threshold (Θ) to range from 14/14 (100%) down to 1/14 (7.14%), where Θ denotes the number of classifiers needed to classify a segment $\boldsymbol{S}$ as Positive. This means that if Θ is set to 14/14 (leftmost row in Table 3), the audio segment will be classified as Positive if all 14 classifiers vote Positive; if set to 13/14 (92.9%)the audio segment will be classified as Positive when

13 or more classifiers vote Positive etc..

In addition to True Positive, True Negative, False Positive, and False Negative totals, we measured the Accuracy rate, Specificity, and TPR corresponding to each of the 14 voting threshold results. As shown in Table 3 higher thresholds lead to lower false positives.

Table 3: Approach Pre-qualification Results by Voting Thresholds

Vote Θ	**TP**	**TN**	**FP**	**FN**	**Accu**	**SPC (1-FPR)**	**TPR (1-FNR)**
14/14	1601	198	1	121	93.65%	99.50%	92.97%
13/14	**1679**	**193**	**6**	**43**	**97.45%**	**96.98%**	**97.50%**
12/14	1699	180	19	23	97.81%	90.45%	98.66%
11/14	1712	157	42	10	97.29%	78.89%	99.42%
10/14	1715	132	67	7	96.15%	66.33%	99.59%
9/14	1717	99	100	5	94.53%	49.75%	99.71%
8/14	1720	84	115	2	93.91%	42.21%	99.88%
7/14	1720	63	136	2	92.82%	31.66%	99.88%
6/14	1721	50	149	1	92.19%	25.13%	99.94%

Analysis of Table 3 highlights the reverse correlation between FP and FN. A close review of Accuracy, SPC and TPR shows the best Accuracy results (97.81%) to occur at $\Theta = 12/14$ but the nature of our speech remediation is partial to lower FP classes in spite of the relatively high increase in FN. By considering $\Theta = 13/14$ we lower the FP values by 13 classifications while increasing FN by 20. We choose to trade 13 FPs

for 20 FNs, in other terms, we prefer to leave 20 ***block*** segments in the recording to avoid deleting 13 semantically meaningful utterances.

Such a trade-off is reasonable considering the negative impact False Positive classifications have on a recording, varying a voting cut off threshold brings considerable improvement to the system.

The effect of moving the Θ threshold is illustrated in Figure 8 where the line graph helps confirm the validity of our decision for $\Theta = 13/14$ (92.9%)

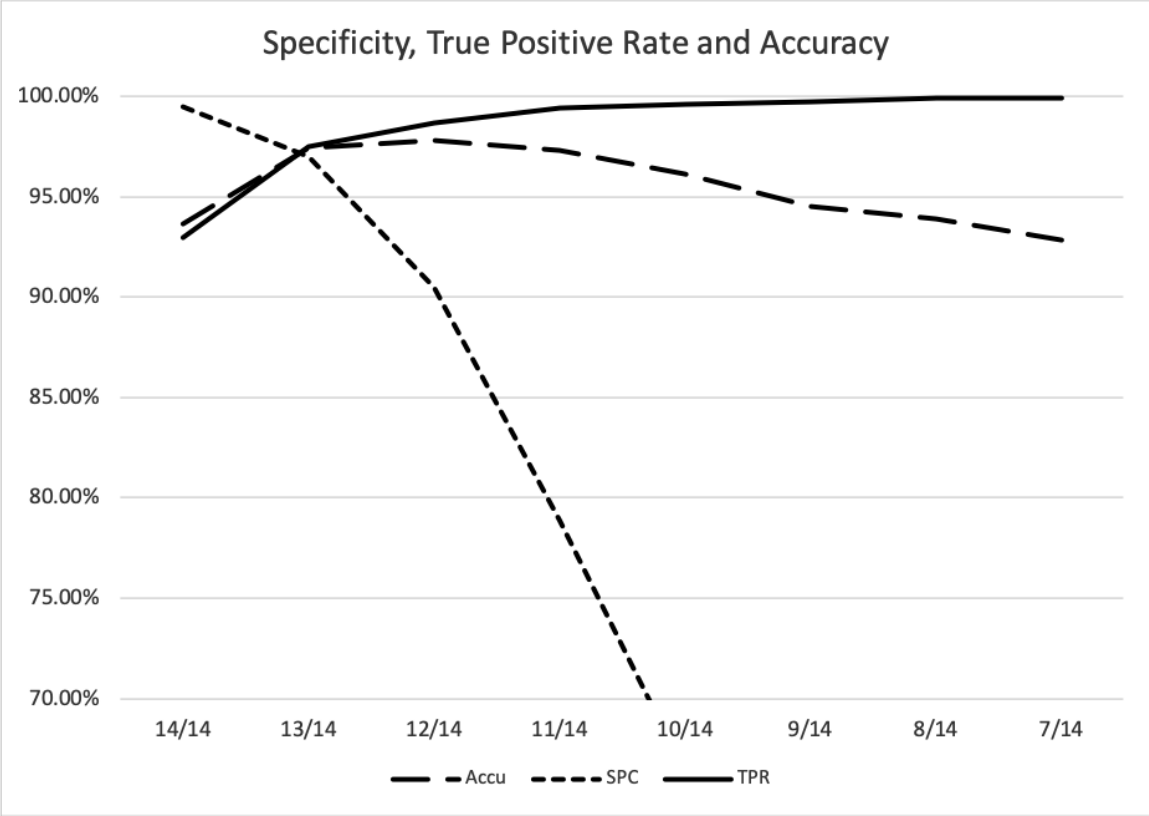


Figure 8: Specificity, TPR, Accuracy Voting Results Graph

The training of various classifiers will be based on our dataset which consists of tuples combining metadata and statistical summaries.

3.1.7 Approach Pre-qualification - Results

For our model training we experimented with a large number of classifiers and selected the best performing ones []. We achieved a high degree of classification accuracy with decision tree models; the two classifiers which performed best were C5 and AdaBoost. The better classifier of the two was C5 and was chosen as the scoring model. We used the C5 classifier to drive the remediation work where segments classified ***block*** are removed from the recording.

Although of slightly lower quality, C5 provided a respectable speech quality and was close to the *Gold Standard* speeches.

The approach we adopted was to train and build a classifier model for disfluent episode detection based on the balanced dataset for all eight threshold combinations i.e. all minimum duration and maximum intensity threshold values; in other words, we combined all the segments extracted from all eight different unique pairs of minimum duration and intensity threshold, and built our classifier models accordingly. The performance results are shown in Table 4. The three best performing classifiers in order of accuracy are C5.0, AdaBoost and Random Forests. Below, we also provide a short summary of the classifiers best performing parameters along with a brief practical explanation of the better performing classifiers below.

As we trained our classifier models, six of them in total, we used 10-fold cross validation. All classifiers were trained with the ***R*** 'caret' package. The ***R*** 'caret' package with the uniform interface to a large number of classifiers and its built-in grid based hyper parameter tuning was a good choice.

Next, we list the parameters chosen for each trained model followed with a brief theoretical summary of the best performing algorithms.

C5 description C5.0 is an enhanced version of C4.5 with speed performance enhancements.

Neural Network (nnet) parameter tuning results: Tuned model parameters when training on the balanced dataset: $size = 5$ and $decay = 0.1$. Tuned model parameters when training on the entire dataset: $size = 1$ and $decay = 0.1$. Size represents the number of units in the hidden layer and can be zero if there are skip-layer units. The decay parameter represents *weight decay* or the weights adjustment with each epoch.

Random Forest (rf) description and parameter tuning results: *Random Forest* was one of the best performers:

For a data universe $((x_1, y_1)..(x_n, y_n)) = D$

A random forest classifier is based on T_i where $i = 1..B$ decision trees based on a random selection of samples S_i and a random selection of predictors M_i.

Random forests are based on two beliefs:

1. Most trees provide a correct prediction of class for most data

2. The mistakes made by the trees are in different places

Observations 1 and 2 imply that to build a classification group of trees with better prediction capability than a single tree one would use multiple random forest trees and use the collection of results to reach a better and more accurate classification.

Random forest provides the *class* based on all trees $T_1..T_n$ where n represents the number of trees.

The large collection of de-correlated trees successfully mitigates the high variance between individual trees, by averaging over an ensemble of trees we can reduce the variance from one tree to the next and improve overall performance.

The rules of thumb for the features used per tree are: $M = \sqrt{K}$ where M is the features per tree and K the number of predictors.

With our $K = 127$ dataset, for a balanced dataset the optimal $M = 2$, whereas for the entire data universe $M = 56$.

SVM Polynomial (svmPoly) parameter tuning results: Tuned model parameters when training on the balanced dataset: *degree* = 3, *scale* = 0.01 and $C = 1$. Tuned model parameters when training on the entire dataset: *degree* = 3, *scale* = 0.01 and $C = 1$. The *degree* parameter represents the polynomial degree of the kernel function. The *scale* is the scaling parameter of the polynomial and tangent kernel. C is the cost regularization parameter.

Adaptive Boosting (AdaBag): Adaptive boosting is an ensemble technique and one of two best performing classifiers. AdaBoost builds a model on sequentially built and combined simple decision tree predictors adapted and improved by the errors from previous models and increased weights on mis-classified samples with subsequent predictors. Adaboost creates a simple predictor and gradually focuses on prior mistakes. The collection of weak predictors are combined into one complex committee based model and the resulting model predicts according to the weighted sum of all predictions.

Adaboost classification performed well during Approach Pre-qualification.

Parameter tuning results:

Tuned model parameters from training on the balanced dataset were: *mfinal* = 100 and *maxdepth* = 3.

Tuned model parameters when training on the entire dataset were: *mfinal* = 150 and *maxdepth* = 3.

The *mfinal* parameters represents the number of trees for which boosting is run.

Maxdepth is the maximum depth of any final tree node.

Recursive Partitioning (rpart): Tuned model parameters when training on the balanced dataset: cp = 0.03797468. Tuned model parameters when training on the entire dataset: cp = 0.05970149. Cp is the complexity parameter; any split that does not decrease the overall lack of fit by a factor of cp is not attempted. The main role of this parameter is to save computing time by pruning off under performing splits.

Figure 9 shows a remediation process.

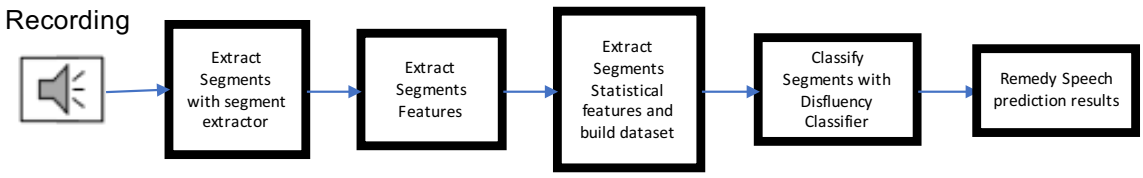


Figure 9: Approach Pre-qualification Summarized Process Flow

As can be seen in Table 4 the trained neural network classifier is weak; this is due to the fact that our dataset is too small for adequate neural network training which requires large datasets [15].

Following the training we analyzed the features for their relative importance and the pitch related predictors were the most important for the Approach Pre-qualification

Table 4: Approach Pre-qualification - TNR, TPR, Accuracy

	True Positive Rate	True Negative Rate	Accuracy
C5.0	100%	94%	97.4%
Neural Networks	75.6%	55.9%	67.7%
Recursive Partitioning	92.6%	94.2%	93.2%
Random Forests	98.4%	91.7%	95.8%
Polynomial SVM	89.9%	75.6%	84.4%
AdaBoost	98.3%	93.8%	96.4%

Table 5: RF, C5.0 Classifier Results by Min Duration & Intensity

	Random Forest	C5.0 Classifier
True Positive Rate (Averaged)	95.3 %	94.8%
True Negative Rate (Averaged)	82.3 %	76.6%
Threshold with Highest True Positive Rate	1.2 minimum duration, 90% intensity threshold	0.8 minimum duration, 90% intensity threshold
Threshold with Highest True Negative Rate	1.2 minimum duration, 90% intensity threshold	1.2 minimum duration, 90% intensity threshold
Threshold with Lowest True Positive Rate	1.2 minimum duration, 95% intensity threshold	1.0 minimum duration, 95% intensity threshold
Threshold with Lowest True Negative Rate	1.0 minimum duration, 95% intensity threshold	1.0 minimum duration, 95% intensity threshold

top classifier (C5.0).

Our approach proved effective in eliminating the vast majority of voice ***blocks*** when applied to a stuttered speech. The results of disfluent recordings remediation repaired difficult to comprehend speeches by eliminating ***block*** segments thus enhancing the fluency of the speech. This also led to a reduction in speech length by large amounts, ranging between 60 % for the most severely stuttered speech to 25 % for the milder ones.

Figure 10 shows the graphical effect of the remediation process.

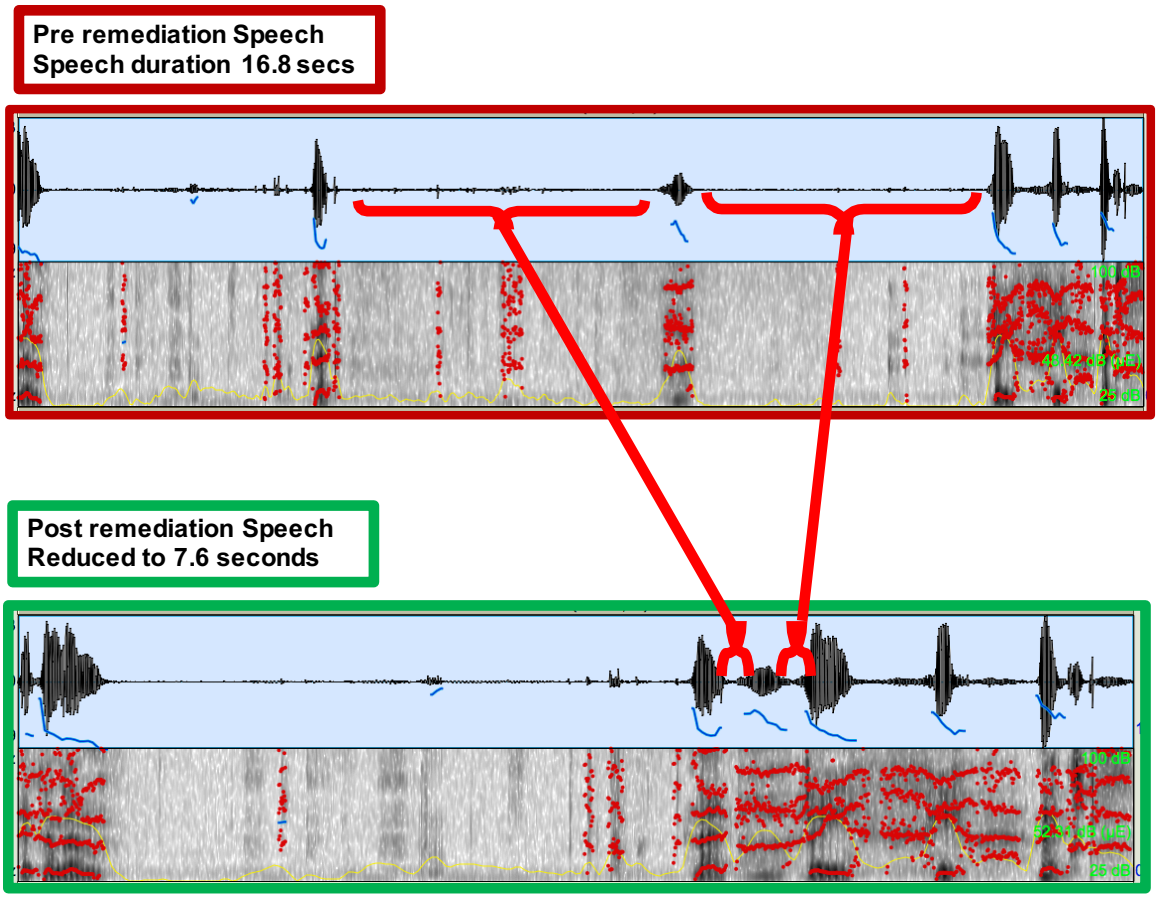


Figure 10: Remediation effect graphical representation

CHAPTER 4: APPROACH RANDOM-SAMPLING

4.1 Approach Random-sampling

4.1.1 Approach Random-sampling - Summary

The second part of the research was also focused on the detection and removal of speech ***blocks*** albeit with different sampling technique of random segment extraction and a sliding window classifier segments scoring.

To avoid a pre-qualification stage, training speeches were separated in two segment pools. Each pool was a merge of adjacently concatenated segments of one long recording. One recording would include the sound episodes for all *positive* dataset tuples and the second for *negative* dataset tuples. The data features engineering from the collection of *positive* and *negative* segments was identical to ***Approach Pre-qualification***.

In terms of application, the trained model would scan over the entire recording submitted for scoring. As we discuss below the results obtained during classifier sliding with the ***Approach Random-sampling*** classifiers were excellent whereas the ***Approach Pre-qualification*** trained models failed this test.

During this approach, ***Praat*** was our raw data extraction tool; ***Python*** with *Jupyter Notebook* was our development environment with heavy reliance on*H2O.ai*, *Numpy* and *Pandas* libraries were used for sample segments extraction, labeling, feature

engineering and classifier training.

4.1.2 Approach Random-sampling - Segment Extraction

Approach Random-sampling was built on the assumption that presenting random short audio segments to users for labeling would yield more consistent overall sampling, labeling results and varied dataset.

4.1.3 Approach Random-sampling - Labeling

In order to create the dataset for training we tried two sampling methods.

1. **Create random segments of speech and manually label them** This technique although simple was difficult to implement because the random samples could not be readily labeled, they invariably contained a mix of *blocked* and *non-Blocked* utterances. This preponderance of segments with mixed content was difficult to manage because mixed sounds were common place and disruptive to the labeling process.

2. **Create two long recordings, one of *block* sounds, the second *non-Block* sounds**. Two recordings were created by listening to numerous speeches and separating the ***block*** disfluent portions from the (*non-Block*) portions. The initial part of the process proved tedious but with completed recordings we extracted an abundance of samples by methodically sampling from each *.wav* file. This is the sampling technique we used during the ***Approach Random-sampling*** data preparation.

4.1.4 Approach Random-sampling - Segment Analysis

Since the ***Approach Pre-qualification*** classifier is intended for the detection of ***block*** sounds, i.e. the same disfluency type as ***Approach Pre-qualification***, we saw no need to re-engineer the features or to consider other feature engineering. Therefore the feature extraction mechanism and engineering rely on the same raw data and statistics and we followed the same technique where the individual segment *.wav* files are individually fetched and their *DSP* vectors (F1, F2, F3, P, I) are created with their respective derivative vectors (dF1, dF2, dF3, dP, dI). The ten vectors are analyzed and individual vector statistics computed.

The computed vector statistics are the same as in ***Approach Pre-qualification***: ***Average***, ***median***, ***Standard Deviation***, ***25 Percentile***, ***50 Percentile*** and ***75 Percentile***, ***Vector minimum***, ***Vector maximum***, ***Peak-to-peak Amplitude***, and ***Variance***.

The results obtained confirmed our intuition that the same features are equally effective in both approaches in spite of the sampling method differences.

During this approach, we chose ***Python*** for the data preparation and classifier training. This decision was motivated by the convenience of using the same development language and environment for data preparation, model training and scoring.

The labels for the training dataset are derived from the segment name embedded in the *.wav* file name.

4.1.5 Approach Random-sampling - Building the Dataset

As stated earlier, the datasets from ***Approach Pre-qualification*** and ***Approach Random-sampling*** had the same list of features. The ***Approach Pre-qualification*** dataset consisted of segments stemming entirely from the set of *potentially undesired* segments, which were obtained with a *Praat* pre-qualification script. The ***Approach Random-sampling*** dataset tuples, contained the statistical aggregations from a diverse list of segments. The dataset obtained from ***Approach Pre-qualification*** was used to build classifiers in two environments and sets of libraries. In ***Approach Random-sampling*** the classifiers were developed with ***H2O.ai*** libraries. Since ***Approach Pre-qualification*** and two classifiers were trained with the same features, the main difference was in the sampling, *potentially disfluent* segments in the pre-qualified universe for one approach but all segments in the second.

The pre-qualified and random datasets were prepared for training by importing the previous ***.csv*** files and train/test split of 80/20 percent split as shown in Figures 11 and 12:

```
labeled_path = "/Users/pierrearbajian/Library/\
Mobile Documents/com~apple~CloudDocs/1/wavs/labeled_blocks/"
df_new = h2o.import_file(labeled_path+
                         "approach1_dataset_balanced.csv",
                         col_types = {'label':"enum"})
# df_new.describe()
train, test = df_new.split_frame(seed = 1234,
                                 destination_frames=
                                 ["train.hex", "test.hex"])
r = df_new["label"].runif(1234)
train = df_new[r < 0.8]
test = df_new[r >= 0.8]
df_names_x = df_new.names[:]
df_names_x.remove("label")
df_names_x.remove("recording_id")
```

```
Parse progress: |
```

Figure 11: Pre-qualification dataset preparation

```
labeled_path = "/Users/pierrearbajian/Library/\
Mobile Documents/com~apple~CloudDocs/1/wavs/labeled_blocks/"
df_new = h2o.import_file(labeled_path+"recordings2.csv",
                         col_types = {'label':"enum"})
# df_new.describe()
train, test = df_new.split_frame(seed = 1234,
                                 destination_frames=
                                 ["train.hex", "test.hex"])
r = df_new["label"].runif(1234)
train = df_new[r < 0.8]
test = df_new[r >= 0.8]
df_names_x = df_new.names[:]
df_names_x.remove("label")
df_names_x.remove("recording_id")
```

```
Parse progress: |
```

Figure 12: Random samples dataset preparation

4.1.6 Approach Random-sampling - Building the Classifiers

To build the classifiers, we used the open source software (*H2O.ai*) with ***Python***.

The training with ***GBM*** and Neural Networks were performed with the H2O ***GBM*** (10 trees and maximum depth fo 6) and H2O Deep Learning libraries with the default

values, as recommended by H2O.ai. The training script can be found ***Figures*** 13 and 14

```
# gradient boost machine
data_gbm = H2OGradientBoostingEstimator(ntrees =10, max_depth      =6,
                                        distribution ="bernoulli")

data_gbm.train(x=df_names_x, y ="label", training_frame=train,
               validation_frame=test)

# Simple Deep Learning
data_dl = H2ODeepLearningEstimator(variable_importances=True,
                                   loss ="Automatic")

data_dl.train(x =df_names_x, y ="label",
              training_frame  =train,
              validation_frame=test)

train_auc_gbm = data_gbm.model_performance(train).auc()
test_auc_gbm  = data_gbm.model_performance(test) .auc()
train_auc_dl = data_dl.model_performance(train).auc()
test_auc_dl  = data_dl.model_performance(test) .auc()
```

```
gbm Model Build progress: |████████████████████████████████████████████
█| 100%
deeplearning Model Build progress: |████████████████████████████████████
█| 100%
```

Figure 13: Pre-qualification dataset training

```
# gradient boost machine
data_gbm = H2OGradientBoostingEstimator(
    ntrees =10, max_depth      =6,
    distribution ="bernoulli")

data_gbm.train(x=df_names_x, y ="label",
               training_frame=train,
               validation_frame=test)

# Simple Deep Learning
data_dl = H2ODeepLearningEstimator(variable_importances=True,
                                   loss ="Automatic")

data_dl.train(x =df_names_x, y ="label",
              training_frame = train, validation_frame=test)

train_auc_gbm = data_gbm.model_performance(train).auc()
test_auc_gbm  = data_gbm.model_performance(test) .auc()
train_auc_dl = data_dl.model_performance(train).auc()
test_auc_dl  = data_dl.model_performance(test) .auc()
```

```
gbm Model Build progress: |
| 100%
deeplearning Model Build progress: |
| 100%
```

Figure 14: Random samples dataset training

The train and test performance results are in Figures 15 and 16

4.1.7 Approach Random-sampling - Classifier Results

```
header = ["Model", "AUC Train", "AUC Test"]
table  = [ ["GBM", train_auc_gbm, test_auc_gbm],
           ["DL ", train_auc_dl, test_auc_dl]]
h2o.display.H2ODisplay(table, header)

print('\nDeep Learning')
print(data_dl.confusion_matrix())

print('\nGradient Boosting Machine (GBM)')
print(data_gbm.confusion_matrix())


```

Model	AUC Train	AUC Test
GBM	0.9891288	0.8525641
DL	0.9838149	0.8620608

```
Deep Learning
Confusion Matrix (Act/Pred) for max f1 @ threshold = 0.7532926987001989:
```

	Block	Normal	Error	Rate
Block	410.0	14.0	0.033	(14.0/424.0)
Normal	16.0	144.0	0.1	(16.0/160.0)
Total	426.0	158.0	0.0514	(30.0/584.0)

```
Gradient Boosting Machine (GBM)
Confusion Matrix (Act/Pred) for max f1 @ threshold = 0.33137010862867966:
```

	Block	Normal	Error	Rate
Block	418.0	6.0	0.0142	(6.0/424.0)
Normal	15.0	145.0	0.0938	(15.0/160.0)
Total	433.0	151.0	0.036	(21.0/584.0)

Figure 15: GBM & DL models performance with *pre-qualification*

```
header = ["Model", "AUC Train", "AUC Test"]
table  = [ ["GBM", train_auc_gbm, test_auc_gbm],
           ["DL ", train_auc_dl, test_auc_dl]]

h2o.display.H2ODisplay(table, header)

print('\nD E E P   L E A R N I N G')
print(data_dl.confusion_matrix())

print('\nGLOBAL BOOSTING MACHINE (GBM)')
print(data_gbm.confusion_matrix())

```

Model	AUC Train	AUC Test
GBM	0.9956132	0.9538835
DL	0.9986999	0.9640777

```
D E E P   L E A R N I N G
Confusion Matrix (Act/Pred) for max f1 @ threshold = 0.6698413109946612:
```

	Block	Normal	Error	Rate
Block	266.0	6.0	0.0221	(6.0/272.0)
Normal	3.0	424.0	0.007	(3.0/427.0)
Total	269.0	430.0	0.0129	(9.0/699.0)

```
GLOBAL BOOSTING MACHINE (GBM)
Confusion Matrix (Act/Pred) for max f1 @ threshold = 0.4921912055420896:
```

	Block	Normal	Error	Rate
Block	253.0	19.0	0.0699	(19.0/272.0)
Normal	1.0	426.0	0.0023	(1.0/427.0)
Total	254.0	445.0	0.0286	(20.0/699.0)

Figure 16: GBM & DL models performance with Random Samples

From the classification algorithms available on H2O.ai, we decided to focus on Deep Learning (***DL***) and Gradient Boosting Machines (***GBM***). ***GBM*** is a forward learning ensemble method. The guiding heuristic is that good predictive results can be obtained through increasingly refined approximations. H2O's ***GBM*** sequentially builds classification trees on all the features of the dataset in a fully distributed way - each tree is built in parallel [9].

In the case of both, ***Approach Pre-qualification*** and ***Approach Random-sampling***, the ***DL*** model performed marginally better than ***GBM***; the performance of both models are illustrated in Figures 15 and 16

4.1.8 Approach Random-sampling - Application performance

Through manual experimentation, we found that 0.8-second segments of sound are long enough for the labeler to recognize speech ***blocks***, which justifies our selection of 0.8-second lengths for the classifier training segments.

We moved across the same recording with each of the four classifiers: ***Approach Pre-qualification GBM***, ***Approach Pre-qualification DL***, ***Approach Random-sampling GBM***, ***Approach Random-sampling DL*** evaluating 0.8 second sound segments, with 0.1 second offsets. The evaluation performed by the sliding classifier provided disfluency scores between 0.0 and 1.0, representing the ***block*** likelihood at any given time.

To perform our experiment we created the full prediction data file by extracting the formants (formants 1, 2, and 3), pitch and intensity vectors then the statistical summaries from above. We scored a batch of segments representing 132 seconds at a

100 ms step which consisted of (132 - 0.8)/0.1 + 1 = 1313 segments.

After extracting the signal parameters and their statistical analysis for each segment, we had a scoring dataset of 1313 rows, one row per segment. The ***.csv*** file was submitted for scoring and each row was assigned a score.

After completing 1313 classifications, the score for each segment was recorded in a vector. Table 6 is an illustration of the scored segment results during 1 second speech interval.

50 seconds or 500 overlapping segments have been graphically plotted with the sound and other signal analysis for general illustration in Figure 17. Our graph was limited to 50 seconds for graph legibility.

As an illustration of the window scanning results, we provide Table 6 which shows an arbitrary one second recording segment ***block*** scores by the ***Approach Random-sampling GBM*** classifier. The Segment Start and End columns represent the boundaries of each window of 0.8 seconds for which we performed the feature extraction. The Disfluency Score is the prediction result returned by the Classifier. The lower scores indicate weak ***block*** disfluency and higher values a stronger ***block*** disfluency.

Table 6: Approach Random-sampling Sample Results

Segment Start	Segment End	Disfluency Score
39.7	40.5	0.146287819
39.8	40.6	0.146287819
39.9	40.7	0.146287819
40	40.8	0.146287819
40.1	40.9	0.145865812
40.2	41	0.300707975
40.3	41.1	0.300707975
40.4	41.2	0.23527621
40.5	41.3	0.413049702
40.6	41.4	0.413049702
40.7	41.5	0.413049702
40.8	41.6	0.413049702

Approach Random-sampling was developed to eliminate the pre-qualification process from the system which served three purposes:

1. Eliminate the need for a complex rules based program for the detection of various disfluency types

2. Eliminate the dependency on multiple components which will invariably complicate the overall process by creating separate sources for errors

3. A diverse dataset to train an algorithm which enhances the generalization capabilities of a model

The algorithms trained were ***GBM*** and Deep Learning models with limited fine-tuning. After model development all models were used to perform the same *continuous* sliding window scan of one recording. The data was split into training 80% and testing 20%.

Figures 15 and 16 show screenshots of Jupyter ***Python*** notebook code snippet with Area Under the Curve (AUC) and confusion matrices for all trained models.

The dataset for ***Approach Pre-qualification*** had a *class* distribution of 199 ***normal*** samples to 533 ***block*** segment samples. The AUC performance for the training and testing sets gave respectable performance results with a *test* ***GBM*** AUC of 0.853 and a ***DL*** AUC of 0.880. The Confusion matrix results were also acceptable with low error rates and an accuracy of 94%.

The Figure 14 model was trained and tested with the ***Approach Random-sampling*** dataset of 267 ***block*** samples and 400 ***normal*** samples. The AUC figures were considerably better than ***Approach Pre-qualification*** with a test AUC of 0.966 for ***GBM*** and 0.974 for the Deep Learning model.

The implementation of the classifiers was based on a sliding window technique where we detected the ***block*** portions of sound as we moved the classifier one small step at a time. In the sliding window classifier experiment the results were based on shifting the classifier over 0.8 second segments one 0.1 second at a time over the recording entire length of 132 seconds. This test recording was selected for its moderate ***block*** stutter content.

Contrary to the training and testing model statistics in Figures 15 and 16, the

results from the sliding window test did not corroborate the experiment results. Three of our models did not perform as well as expected, only the ***GBM*** model performed well.

By applying the trained classifier in a sliding window fashion over the entire recording, we were able to obtain impressively precise scores from one of the four classifiers.

The sliding window extends our insight into the recording quality by offering continuous evaluation of the speech at small intervals. The continuous nature of the results is depicted in graphical form.

Figure 17 is a graphical representation of the disfluency measure prediction over a 50 seconds time range. The top part of the figure shows the ***Approach Pre-qualification*** disfluency scores with the Approach Random-sampling scores at the bottom. The middle portion of the graph displays the speech signal over the time axis. The speech signal occupies the top part of the *Praat* graph with the Formants, Pitch and Intensity signals in red, blue and green respectively.

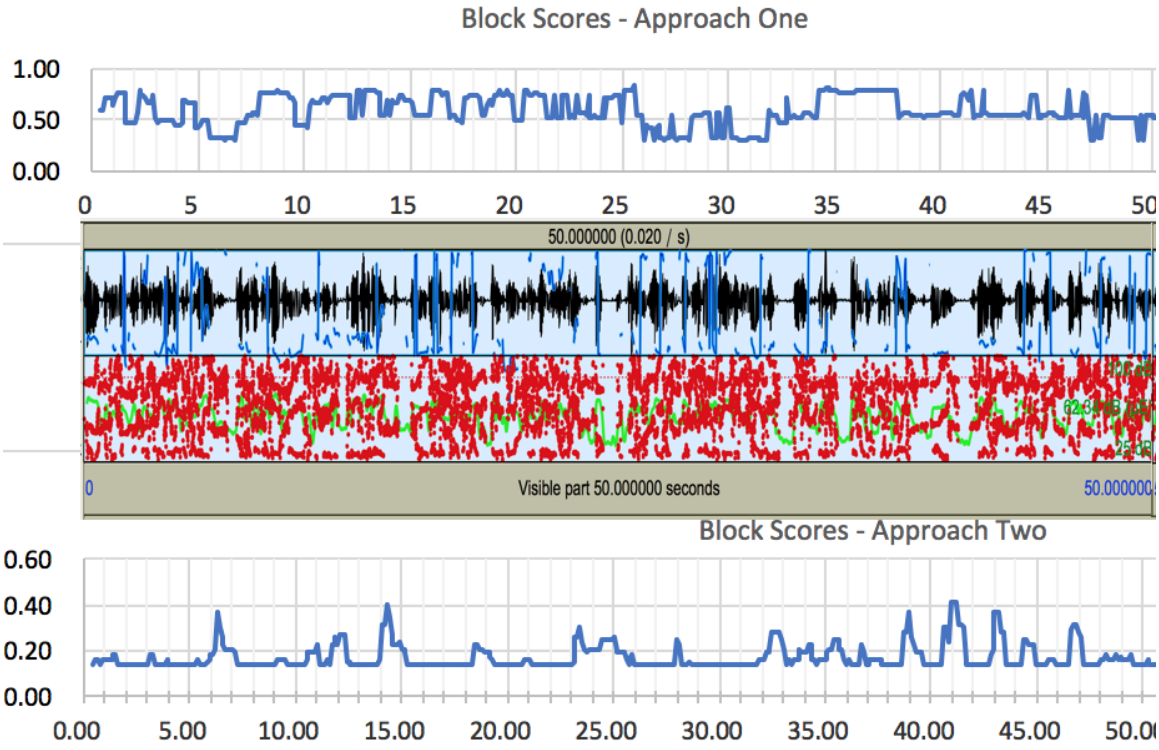


Figure 17: Sliding Window Performance graph

A close review of Figure 17, ***block Scores - Approach Random-sampling*** shows the alignment between classifier model results and Formant, Intensity, Pitch values. To confirm the consistency between the classifier results and the ***block*** stutter occurrences we also performed an audio-visual analysis of the entire recording and graph. Comparison of the Approach Random-sampling graph ***blocks*** timing and the ***block*** occurrences during the speech matched with a near perfect alignment between the ***Ground Truth*** and the sliding classifier results.

The ***Approach Pre-qualification block*** scores were also reviewed for accuracy and we found little to no correlation between speech and scores.

The ***Approach Pre-qualification*** and ***Approach Random-sampling*** sliding window experiment results establish that the pre-qualified samples from ***Approach Pre-***

qualification had a major negative impact of the trained classifier, yet the features we employed are very appropriate for the detection of ***block*** type speech problems.

Our insight as to the underlying reasons for the weak ***Approach Pre-qualification*** performance and the high ***Approach Random-sampling*** accuracy can be summarized as follows. The ***Approach Pre-qualification*** dataset consisted of a speech universe of pre-qualified and weak-fluency segments, the ***Approach Pre-qualification*** classifier was trained on a skewed universe of samples and such a dataset would cause the model to attribute different weights to the features than models trained on diverse samples.

Because the samples were of a certain constituency the classifier placed the emphasis on predictors and values that identify ***blocks*** within that particular, semi monolithic, population. As a result of the skewed training dataset, the scoring scan of a complete speech recording, which requires understanding of a more complex universe with samples from a broad range of disfluency, yielded unreliable results.

Approach Random-sampling, on the other hand, was reliable over the entire recording and over a broad selection of segments. We believe ***Approach Random-sampling*** performed well because, the samples in its training dataset spanned a wide range of disfluency levels which provided the classifier a broad range of good performance throughout the recording.

The ***Approach Random-sampling GBM*** model was the only, one of four, classifier to provide the desired performance results and offer a practical solution to the continuous speech scoring experiment. The ***DL*** model did not perform well during

our sliding classifier experiment. We believe the ***DL*** model generated poor results because the dataset was small and ***DL*** algorithms are known to require large datasets. One other peculiar behavior exhibited by the ***DL*** model and which contributed to its lack of fitness to purpose, was its high swings in the recorded prediction values.

Considering the opaque nature of Deep Learning and its limited interpretability [40], coupled with the strong performance of the ***GBM*** model we chose to focus our analysis on the ***GBM*** models.

In conclusion the ***Approach Pre-qualification*** classifier is only usable with pre-qualified *potentially disfluent* segments.

CHAPTER 5: APPROACH SPECTRAL-ANALYSIS

5.1 Approach Spectral-analysis

5.1.1 Approach Spectral-analysis - Summary

The third approach focused on detecting and repairing speech ***interjections***. We tried the formant, pitch and intensity vector statistics as features but our classifiers performed very poorly. With the poor results we obtained from the prior signal parameters analysis features we chose a different feature development approach. ***Approach Spectral-analysis*** would rely on two dimensional image like features and Deep Neural Network classifier algorithms.

Inspired by the rapid advancements of Deep Neural Networks and especially (***CNN***) we decided to leverage ***CNN*** for which we would use spectrographic data. We tried Spectrogram, Cochleagram and MelSpectrogram data analysis with multiple ***CNN*** models to identify the best data type/model combinations. The results were of high accuracy and our model/data combination yielded a solution that was able to detect ***interjections*** with a high degree of fidelity.

Praat functions were used during ***Approach Spectral-analysis***, both for segment extraction and spectrographic matrices extractions, we used ***Python*** as the scripting language with *Jupyter Notebook* as our development environment, *CNN* as the Deep Neural Network architecture and TensorFlow libraries for matrix manipulation, and

Keras for *Tensorflow* API, *Numpy* and *Pandas* libraries were also used for other dataset preparation and general mathematical computations.

Figure 18 provides a process flow overview for ***Approach Spectral-analysis***

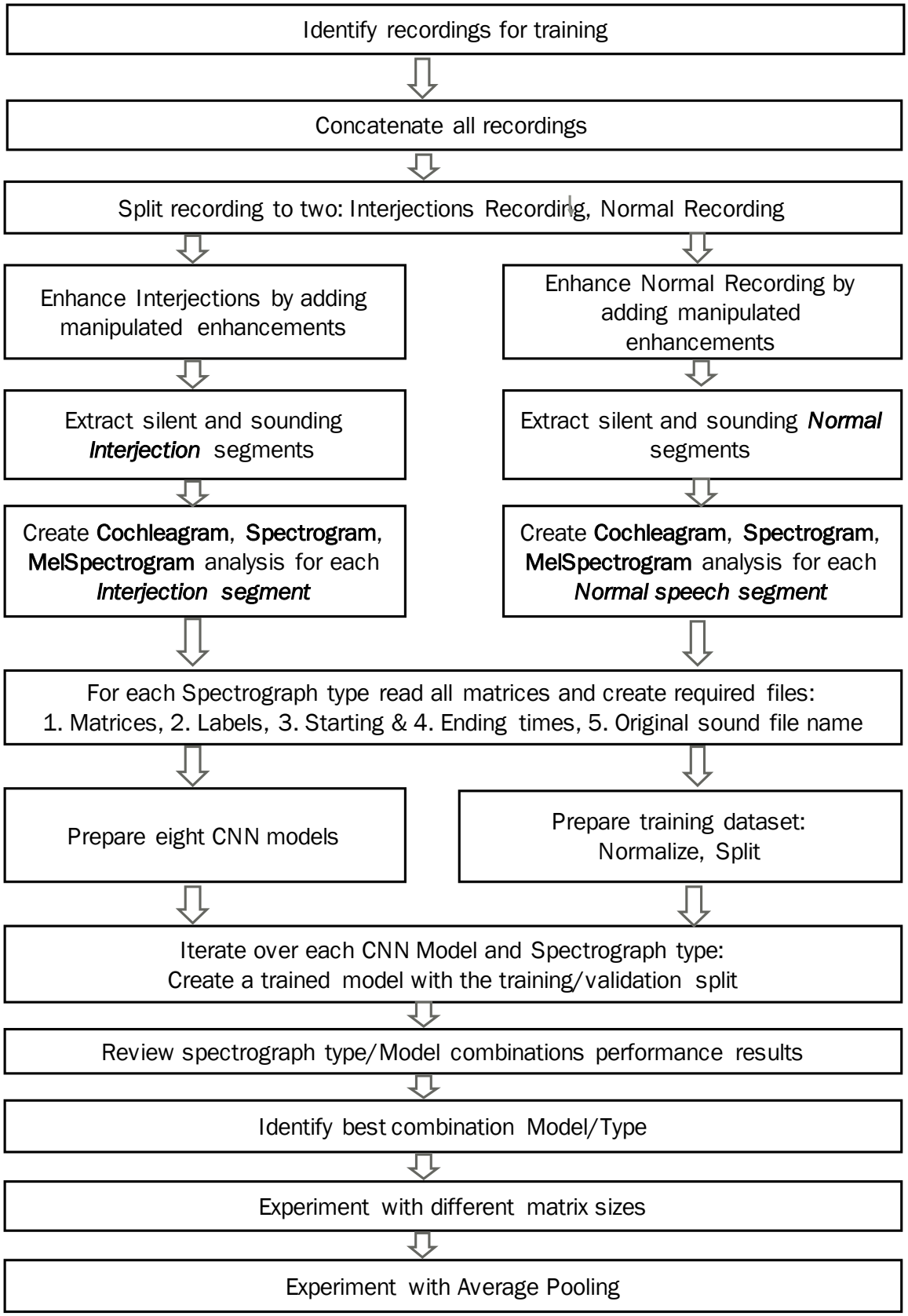


Figure 18: Approach Spectral-analysis overall process

5.1.2 Approach Spectral-analysis - Segment Extraction

During data preparation we used the UCLASS corpus. Of the 140 recordings we chose thirty speakers to maintain an even balance in genders - we only had 15 female speeches in UCLASS.

The UCLASS corpus provides a number of speaker and speech recording condition information, these speaker and recording environment data were not included in our ***Approach Spectral-analysis*** list of predictors. The omission of speech metadata forced the classifiers to better generalize our results so our scoring did not depend on, hard to obtain and document, speaker and environment information, when deployed in a production like situations.

Some of the recordings were abnormally soft and some were loud. Because of this variation in sound intensity it was necessary to ensure that all recordings be scaled to one level and we used 70 dB SPL average sound volume. This intensity scaling consists of raising or lowering the overall volume of an entire recording to an average of 70 dB SPL intensity. Scaling was performed with the ***Praat*** <sound: scale intensity> function and helped us label sound segments easier since we were able to compare sounds of like volume which improved classifier performance quality because of consistent volumes and better labeling.

To process all recordings one might work with individual recording files but we found that combining multiple segments made listening and handling a single recording simpler and more effective thus our decision to concatenate all segments of one class into a single continuous recording. The concatenation process was performed

with the *Praat* function <combine: concatenate>. To concatenate multiple sounds the sampling frequency must be the same and since the UCLASS wav files did not have the same sampling rate, we re-sampled the recordings with the *Praat* function <sound: resample> at a common *Praat* default sampling frequency of 10 KHz and a precision (depth of interpolation) of 50.

Other recording pre-processing options available to us in *Praat* such as *filtering* to attenuate certain frequency ranges as well as *Noise removal*, where acoustic noise is removed from the speech using spectral subtraction, would have also been an option. But because we did not expect our classifiers to benefit from such sound pre-processing we refrained from applying additional sound manipulation techniques.

To better manage the labeling process, we relied on folder names to designate, and later determine, the *ground truth* class of combined sound recordings. During our folder scans the name of the folder where data files resided determined the class of a recording i.e. Disfluent (***interjection***) speech or a fluent (***no-interjection***).

Since the UCLASS data is not labeled for ***interjections*** and manual labeling is a time consuming and tedious process, it was important to find a method to streamline the labeling process. The method of choice was to split our combined speeches into two recordings. One recording would contain all normal (no-***non-interjection***) segments of speech and the other consists of all disfluent (***interjection***) segments.

We used the *Praat* <view and edit> function to observe progress across a speech to highlight and cut portions that must to be eliminated. Once the recording edit and splicing is complete we would save the speeches as one ***interjection*** speech or another Normal (***non-interjection***) speech.

This allowed us to expand our recording files for ever larger "***interjection*** " and "no-***interjection*** " sound files and larger training datasets.

Because ***interjections*** are less frequent that ***normal*** speech and neural networks are known to require large datasets we expanded our ***interjection*** class samples by creating new ***interjection*** sounds which were appended to our ***interjection*** recording. The synthesis of new sounds was performed by (1) adding a lower pitch version of the ***interjections*** to the recording, (2) adding a higher pitch version of the ***interjections*** to the recording, (3) adding a denoised version of the ***interjections*** to the recording. With the variety of synthetic sounds introduced we achieved a balanced class dataset of 5000 ***interjection*** samples to 5000 ***normal*** speech samples.

The ***interjection*** and non-***interjection*** recordings with merged segments are subsequently used to extract the segments to create the classifier training matrices. The majority of ***interjections*** are isolated sounds of 0.6 to 1.0 seconds in duration which prompted us to segment our speech segments. ***Interjection*** are generally sounds which can be split away from a speech and have silent boundaries which justified us segmenting the recordings into episodes of ***normal*** and ***interjection*** sounds. With most ***interjections*** surrounded with silence, the segmentation approach proved useful and the remediation was successful.

5.1.3 Approach Spectral-analysis - Extracting "sounding" segment

The eventual ***interjection*** classification consists of reviewing sound segment images sequentially to determine which spectrographic image resembles an ***interjection***.

We split out original recordings into a group of "sounding" and silent segments, and

because ***interjections*** are not silent, we ignored the silent segments and trained our classifier with the sounding segments.

This was accomplished by listening to each speech and cutting out the ***normal*** speech segments to reach an all ***interjections*** recording. To develop the non-***interjection*** recording we listened to stuttered recordings but this time we cut out the ***interjections*** from the recording.

With two types of recordings we have the audio data which we harvest to create the ***interjection*** and non-***interjection*** segments.

The recordings obtained were used to create segments which can be automatically labeled. Any segment in the ***interjection*** recording is an ***interjection*** segment and all segments in the ***normal*** speech recording are non-***interjection*** segments.

Sounding episodes are extracted with a two step process. In Step 1 we annotate the entire recording and designate its episodes as "sounding" or "silent", Step 2 extracts the "sounding" segments according to the annotation results. Silent speech parts are discarded, Figures 19 and 20 display the ***Praat*** UI dialog parameters for annotation and extraction. Figures 21 displays a list of extracted "sounding" sounds.

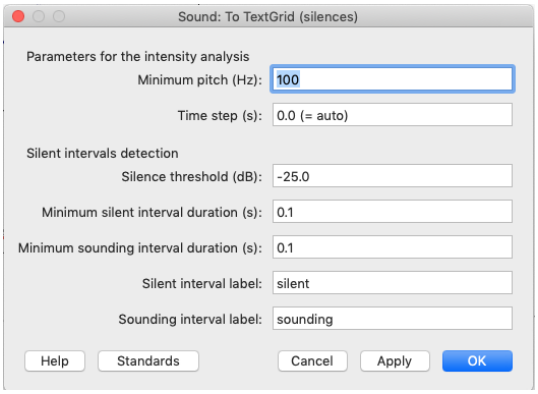


Figure 19: "sounding", silent Recording Annotation

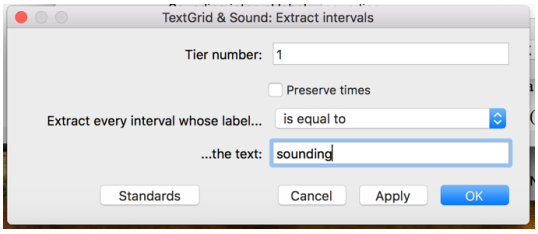


Figure 20: "sounding" Segment Extraction

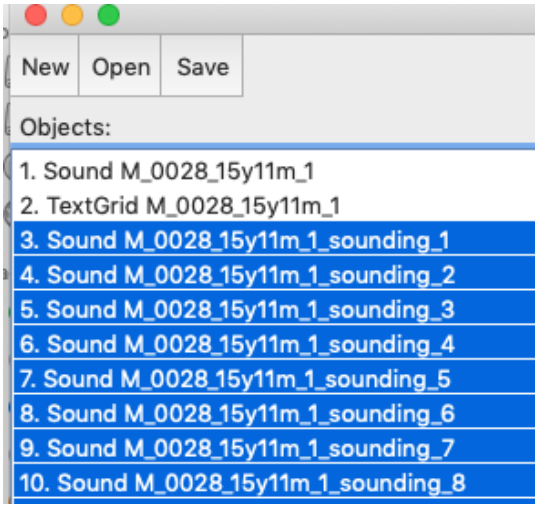


Figure 21: Extracted "sounding" Segments

Each "sounding" segment is converted into a set of predictors for classifier training as one dataset row where the predictor features of the sound segments are the spectrogram matrix scalars.

5.1.4 Approach Spectral-analysis - Feature Extraction

We began our investigation with the same features as ***Approach Pre-qualification*** and ***Approach Random-sampling*** but the formant, pitch and intensity statistical analysis did not capture the essence of the differences between ***interjections*** and ***normal*** speech. The ***interjection*** could not be detected by the old features with the ***GBM*** and Neural Network models therefore both ***Approach pre-qualifier*** and

Approach Random-sampling models and features were discounted from the ***interjection*** detection system.

The observation about the inadequacy of statistical analysis values of the formant, pitch and intensity features can be intuitively confirmed by close visual examination of Figures 22 and 23.

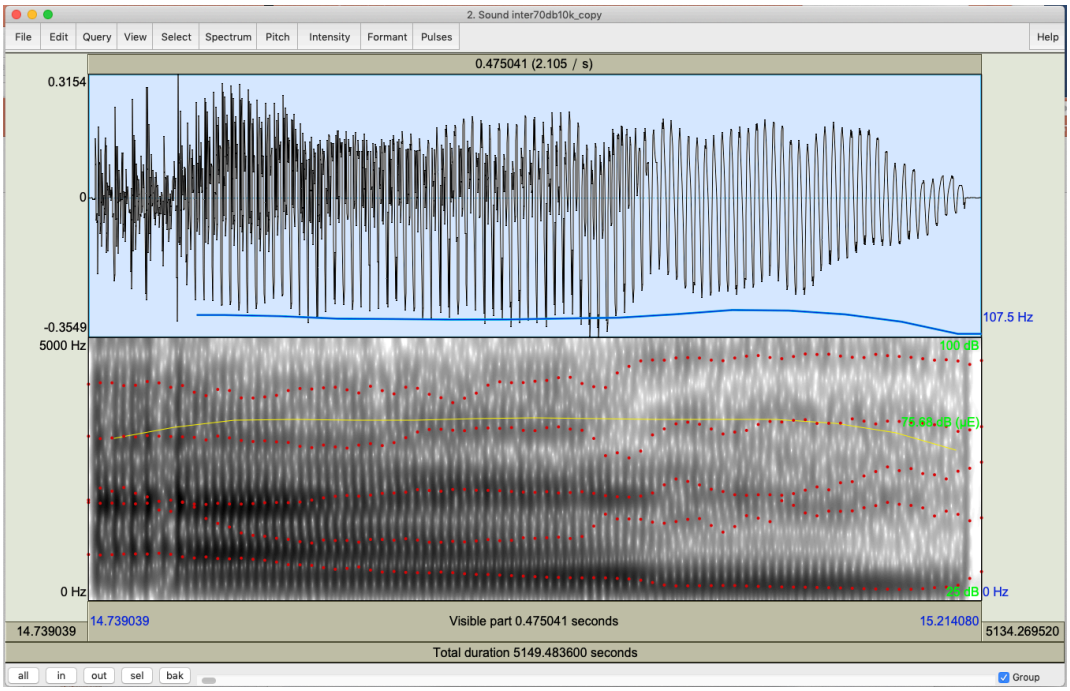


Figure 22: ***interjection*** segment Formant, Pith, Intensity Parameters

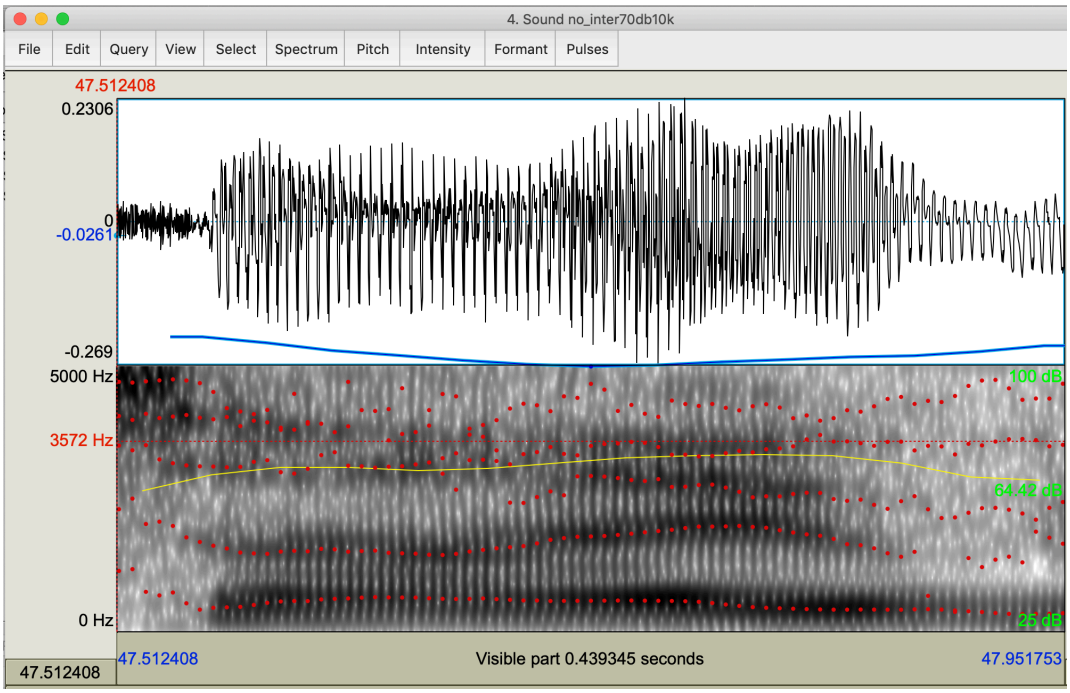


Figure 23: ***normal*** segment Formant, Pith, Intensity Parameters

Review of the formants (1, 2, 3), Pitch and Intensity show that, the ***interjection*** and ***normal*** sounds would be difficult to distinguish with summary statistics. The features which reduce the vectors to 10 statistical values per analysis could not convey the presence of ***interjection*** in the first sound nor detect its absence from the second sound.

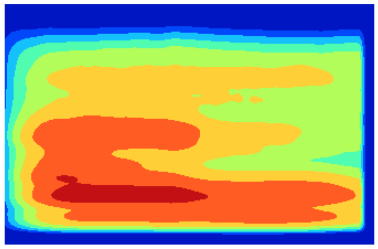
However the same examination of the gray background gradations section at the bottom which represent the spectrogram shows different patterns.

With the poor performance of the ***Approach Pre-qualification*** and ***Approach Random-sampling*** features, coupled with recent Deep Neural Networks accomplishments in the field of speech analysis we chose to experiment with neural networks to

detect the ***interjection*** sounds.

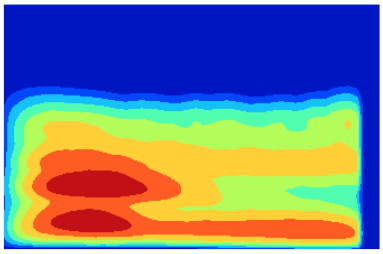
With the advent of (***CNN***) we chose to use spectrum sound analysis data as predictors. Deep Neural Networks, ***CNN*** in particular, have been successfully used in the field of image and sound recognition, where the later relies on image like matrix recognition of spectral frequency analysis.

Spectral sound analysis yields two dimensional image-like matrix representation, we trained ***CNN*** classifiers to recognize whether a spectral image of a sound can detect ***interjections***. The classifiers we trained were designed to detect the ***interjection*** nature of a sound according to its spectral graphical appearance. Figures 24 and 25, offer a contour graphic visual sampling of ***interjection*** and ***normal*** sounds from one of the explored spectral matrix types, the cochleagram:



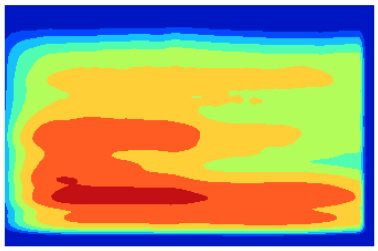
spectros item:5
spectro name:inter70db10k_sounding_6
2.84200000000003 3.3780000000000294
Enter to continue, E to Exclude...

(a) Cochleagram interjection 1



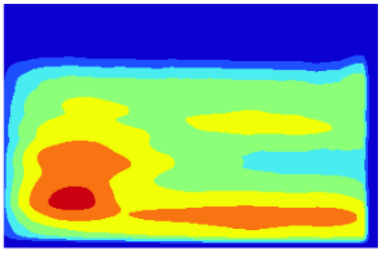
spectros item:10
spectro name:inter70db10k_sounding_11
6.018000000000029 6.226000000000029

(b) Cochleagram interjection 2



spectros item:5
spectro name:inter70db10k_sounding_6
2.84200000000003 3.3780000000000294
Enter to continue, E to Exclude...

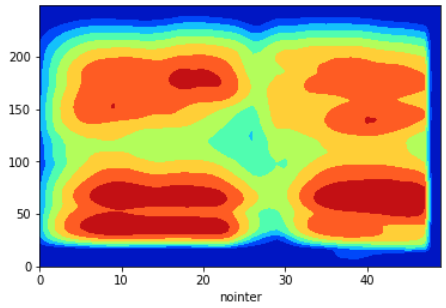
(c) Cochleagram interjection 3



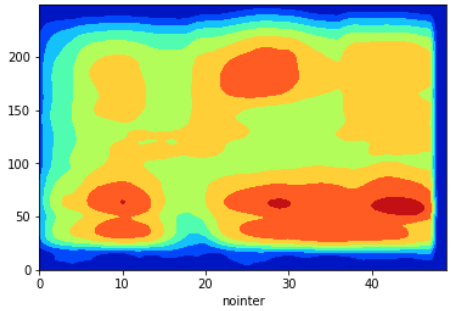
spectros item:6
spectro name:inter70db10k_sounding_7
3.85800000000003 4.226000000000029
Enter to continue, E to Exclude...

(d) Cochleagram interjection 4

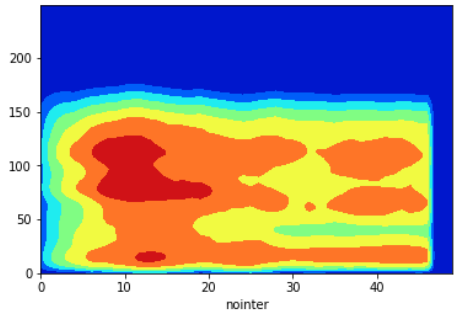
Figure 24: Four Interjection Sounds in Cochleagram Contour Format



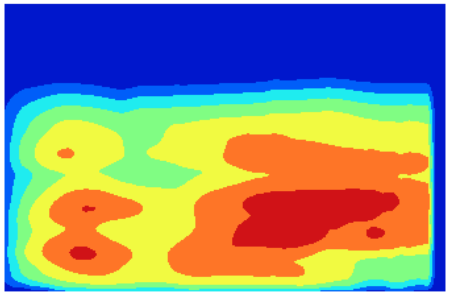
(a) Cochleagram non-interjections 1



(b) Cochleagram non-interjections 2



(c) Cochleagram non-interjections 3



(d) Cochleagram non-interjections 4

Figure 25: Four non-interjection Sounds Cochleagram Contour Format

Visual exploratory examination of the ***interjection*** and no ***interjection*** segments show a clear and distinguishable pattern for each class.

Capturing the visual difference between these two classes will be the object of training new classifiers designed for image identification.

Sounds can be converted to multiple types of spectrograhic images, with the variety of the available representations we wanted to determine the most appropriate spectrographic matrices for ***interjection*** detection.

With reports of varying classifier performances from different types of spectral analysis data [35] we used three spectral analysis methods: Spectrograms, Cochleagrams and Melspectrograms. The examples below represent a one second sound analysis representation of a spectrogram, cochleagram and a melspectrogram.

1. Spectrograms (Figure 27) The spectrogram analysis results were obtained through ***Praat***. Figure 26 presents the analysis criteria during this and subsequent cochleagram analysis.

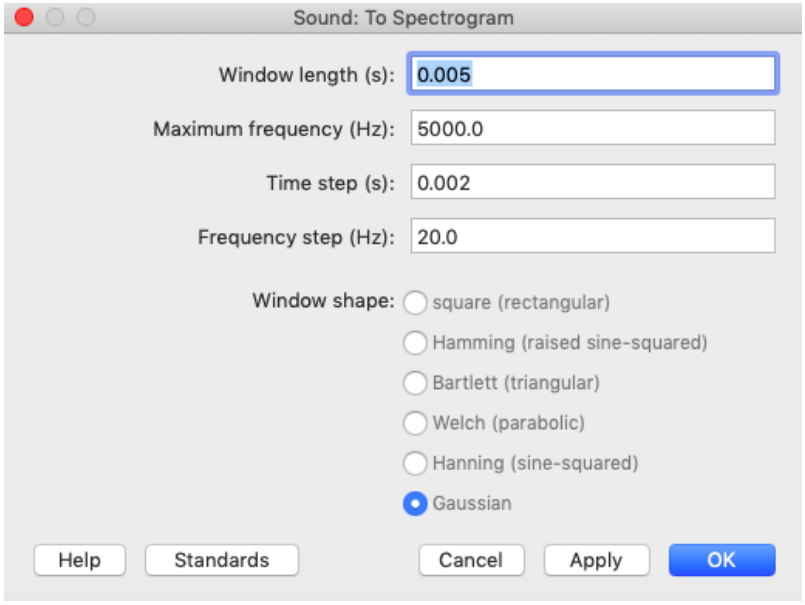


Figure 26: Spectrogram Analysis

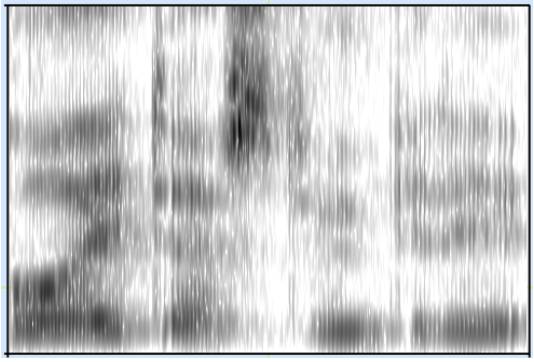


Figure 27: Spectrogram Visualization

2. Cochleagram (Figure 29): The cochleagram analysis results were also obtained with ***Praat***. Figure 28 shows the default cochleagram analysis configuration we used for this and other cochleagrams.

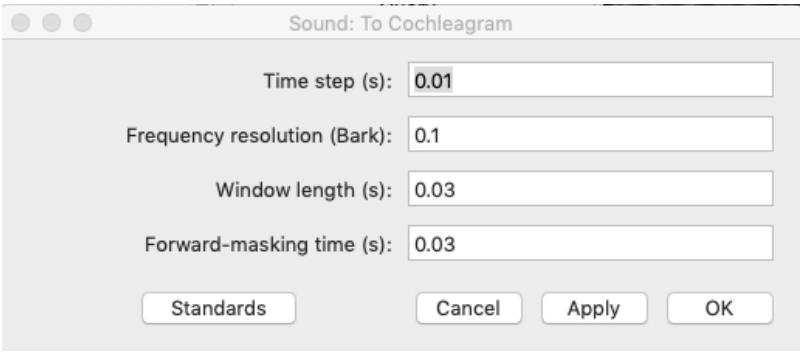


Figure 28: Cochleagram Analysis

Figure 29: Cochleagram Visualization

3. Melspectrograms Figure 31: the MelSpectrogram analysis results shown were obtained through ***Praat***. Figure 30 shows the analysis configuration used in this graph and later in the research.

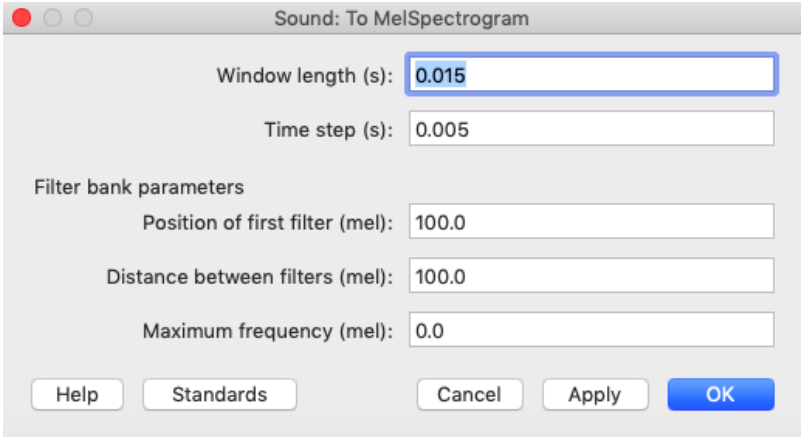


Figure 30: Melspectrogram Analysis

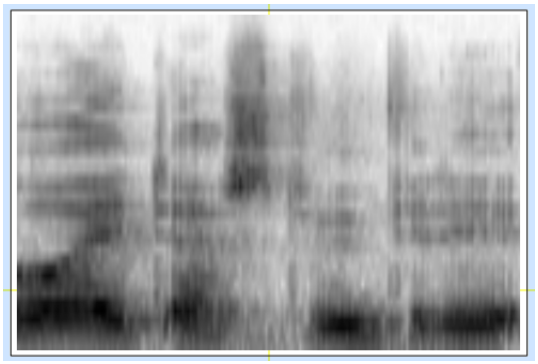


Figure 31: Melspectrogram Visualization

In spite of the basic similarity between the three representations, visually, the cochleagram is especially distinctive. We will work with all three matrix types to decide which of the three will be the matrix type of choice.

5.1.5 Approach Spectral-analysis - Dataset Preparation

CNN models have been successfully trained to distinguish between image classes by processing every pixel of an image as a feature, for instance a 50 by 25 pixels image will consist of 50 x 25 = 1250 feature columns. As the features are input into the classifiers they are scored by a trained model for a likelihood of ***interjection***.

Note that it is possible to feed all pixel values as one vector into a Deep Neural Network and train a decent classifier, but Neural Network techniques known as (***CNN***) have been developed to better detect image characteristics by pattern detectors, which previously depended on manual expert effort. ***CNN*** models have been known to offer excellent image classification performance.

The ***CNN*** classifiers we trained were designed to view images of similar dimensions and process the same number of features from every image/matrix sample i.e. every sound spectrogram. However, the sounds lengths we extracted with ***Praat*** had dif-

ferent durations causing their x-axis dimension to vary from one "sounding" segment to the next.

This variation in sound spectrogram matrix length meant that the pixels on any one image would differ from one sound segment to the next. From the ***interjection*** and ***normal*** sound segment images (***Figures*** 22 and 23) we can intuitively see that the primary task of our classifier is to recognize the overall shape of the image in spite of its dimensions, therefore zooming in and zooming out on a picture should not affect the performance of our classifier since the overall shape will remain intact.

To perform the zooming at spectral analysis extraction time we attempted to force a frequency range and scale the sound duration in ***Praat*** but this resulted in sound frequency modifications, the sound became abnormal, and the models were inferior in performance as they exhibited lower accuracy of classification. The low fidelity of the sound stretched classifier made us dismiss this option and we opted for the matrix zooming approach.

All sound file spectrogram dimensions were therefore adjusted post spectral analysis with *Python* matrix zooming package which casted all matrices to the same dimensions to ensure consistent pixel numbers i.e. the same number of features from every training tuple.

Previously we used ***R*** and ***H2O.ai*** libraries to select and train classifiers. With 2-D spectrographic matrices as training sample features we considered ***H2O.ai*** and other neural network libraries and chose Tensorflow as the computation engine of choice.

The computer we used to run the Tensorflow neural network training was an Apple macbook pro computer with a 2.4 GHz intel core I7, 16 GB RAM and 1600 MHz

DDR3 and no GPU.

5.1.6 Approach Spectral-analysis - Classifier Training

We began our work with the native Tensorflow user interface but soon switched to the Keras API. Keras offers a consistent and simple user interface, with minimal user interaction for most common use cases. Keras was easy to implement with superior TensorFlow integration. With Keras we were able to focus on our exploration and experimented with numerous ***CNN*** models with the full range of tuning parameters such as Dropouts, Decay, Loss function, Epochs, etc.. After extensive exploration we found the high performance ***interjection*** classifiers. Keras was also flexible and integrated with Tensorflow seamlessly.

The training of our ***CNN*** models was performed with the "sounding" .wav files created and saved to be fetched for the creation of sound matrices as in Figures 27, 29 and 31.

Matrix processing and training was performed separately by spectrogram type: 1-Spectrogram 2-Cochleagram and 3-Melspectrogram. All spectrogram training were performed similarly and the performance results compared. This effort was intended to identify the best representation/analysis of a sound to train the ***interjection*** classifiers.

We also compared the performance of several ***CNN*** models. Our first task was to determine the analysis type best suited for ***interjection*** detection and we tried each ***CNN*** model with each of the three spectral analysis types.

A Convolutional Neural Network (ConvNet/***CNN***) is a Deep Learning algorithm

which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower when compared to other classification algorithms. In primitive methods filters are hand-engineered but with enough training, ConvNets have the ability to learn these characteristics and create effective classifier filters. The architecture of a ConvNet is similar to the neuron connectivity pattern in the human brain and was inspired by the organization of the Visual Cortex. Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. A collection of such fields overlap to cover the entire visual area [44].

The specifics of ***CNN*** are beyond the scope of this dissertation and there are excellent resources which describe the foundations of ***CNN***s in detail.

By reviewing well known ***CNN*** networks such as LeNet, AlexNet [16] and VGG we built multiples ***CNN*** models. The details of the model implementation are included in the ***Appendix Section*** both in tabular form and as ***Keras API Python*** code.

Figures 32 through 41 depict the AlexNet style representation of the ten models we employed.

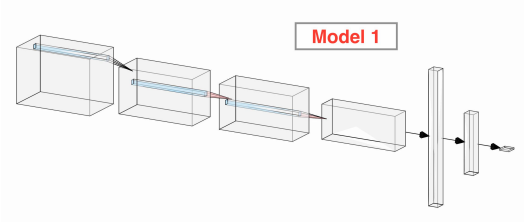


Figure 32: MODEL 1

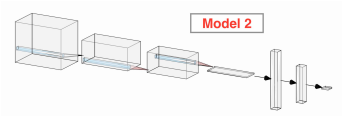


Figure 33: MODEL 2

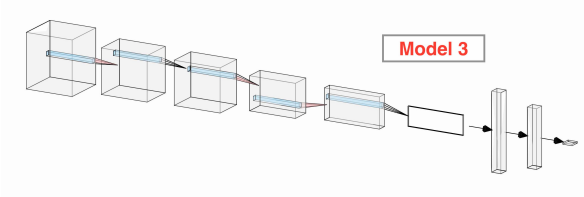


Figure 34: MODEL 3

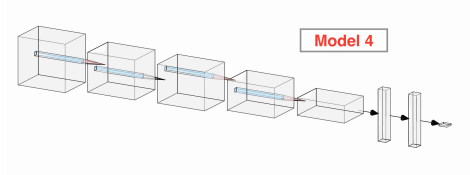


Figure 35: MODEL 4

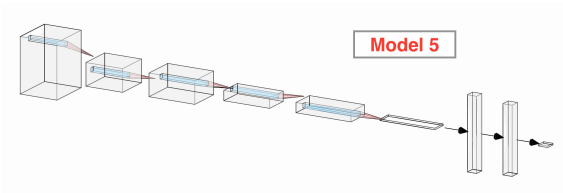


Figure 36: MODEL 5

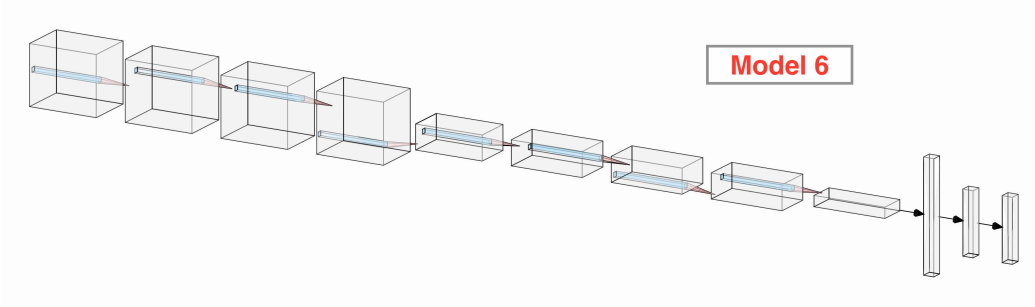


Figure 37: MODEL 6

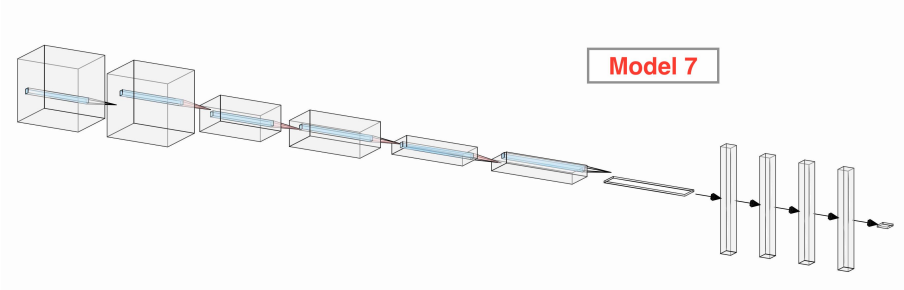


Figure 38: MODEL 7

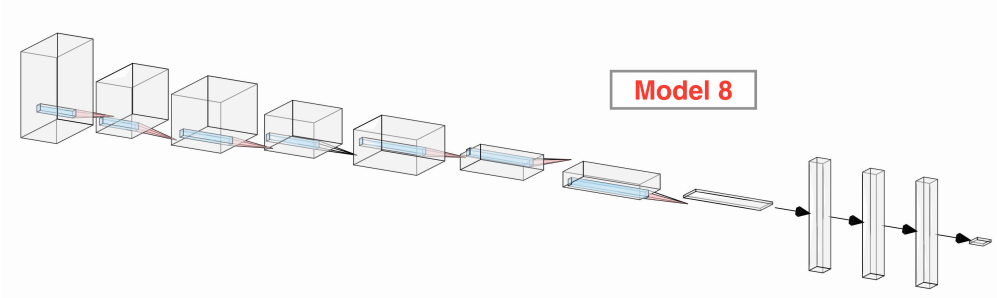


Figure 39: MODEL 8

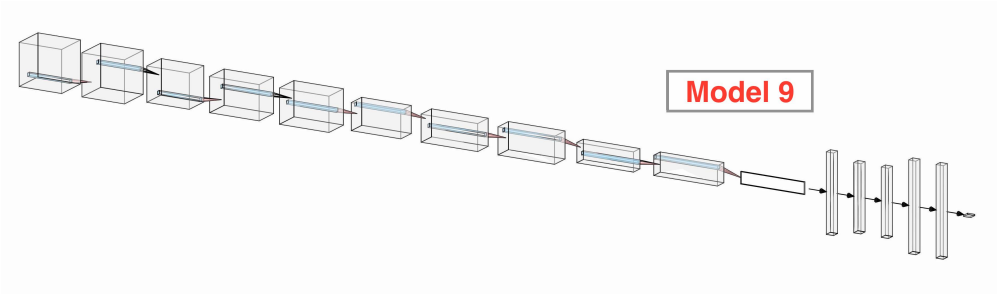


Figure 40: MODEL 9

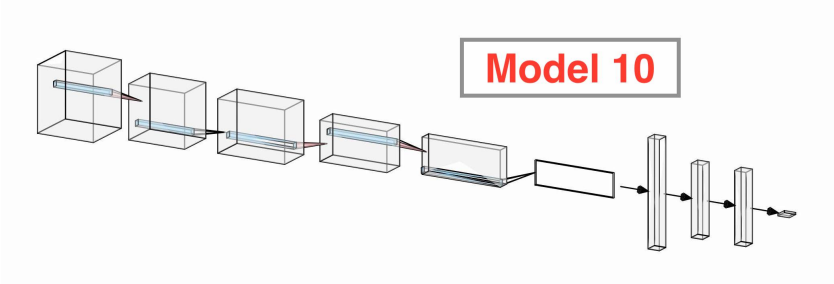


Figure 41: MODEL 10

From a simple viewing perspective, the cochleagram seemed the provide the crispest image but we would not know how well various ***CNN*** networks would train and what the performance results would be until we trained and reviewed the performance numbers.

The labeling approach we used is similar to ***Approach Random-sampling*** where we took 30 speeches which contain stuttered speech segments and separated them into ***interjection*** and non-***interjection*** recordings.

5.1.7 Approach Spectral-analysis - Results

We trained the first eight of the models above with the three spectrogram types.

With ***MaxPooling*** we experimented with different activation techniques and discovered that ***relu*** activations are the best choice for early layers of a network and ***sigmoids*** are the better activation method with dense connections for the later layers.

The training optimizer we used consistently was the ***adam*** optimizer [25] and the loss function we used was the ***binary crossentropy*** with class balanced $10,000$ samples, $5,000$ ***interjections*** and $5,000$ ***no-interjection*** segments, in our dataset we consistently split our training and validation data by 80%, 20%. After numerous exploratory iterations we settled on what we considered the optimal hyper-parameters. We used early stopping during training with the Validation accuracy as the stopping trigger with a minimum delta of 0.0001.

The early stopping ***patience*** we used was 10 which means we would stop executing new epochs if the early stopping trigger i.e. Validation Accuracy does not improve

after 10 epochs.

The ***batch size*** which represents the number of training examples used between weights adjustments in each iteration was 16 and until ***epoch*** completion.

The maximum number of epochs was 200 but with ***early stopping*** and a ***patience*** of 10 we at no point reached the maximum number of epochs and always stopped early.

Figure 42 represents a graph of the training data and validation data model accuracy value change and loss change by ***epoch*** during training progress. The ***trigger*** we selected was the ***validation data accuracy***. In the shown graph the number of epochs executed was 80 which means that there was no improvement in the performance of the classifier on the ***early stopping trigger*** (validation test accuracy) for ten (***patience***) consecutive epochs after execution of 80 back propagation epochs.

The training and testing results will be further analyzed to better assert the best spectrogram types and models.

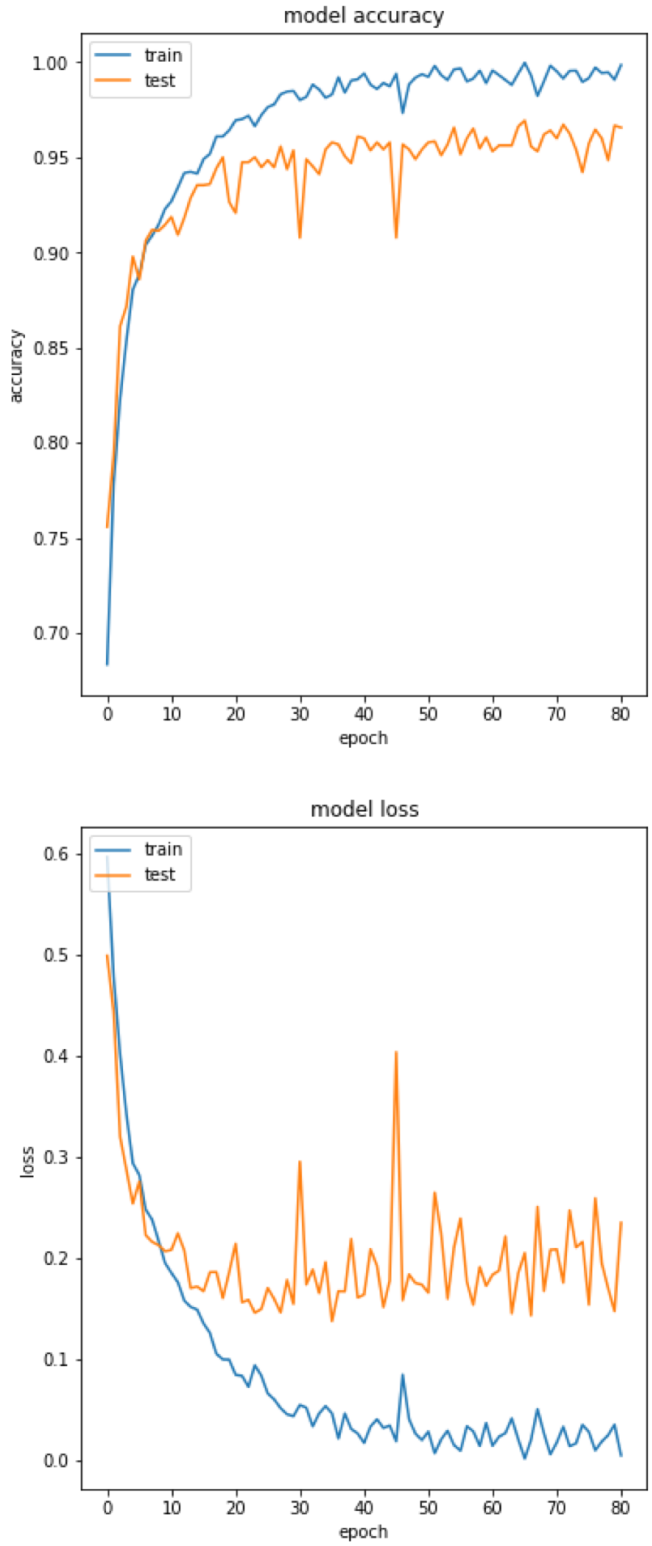


Figure 42: Model Accuracy & Loss by Training Epoch

With the eight models (models 1 - 8) trained and tested with MaxPooling on the

three spectrogram types the cochleagram data trained the best classifiers. Tables 7, 8, 9 show the training and testing data with loss and accuracy for each of the three spectrogram types.

	Cochleagram			
Model	Train Loss	Train Accu	Test Loss	Test Accu
1	**0.0498**	**0.9822**	**0.3478**	**0.9098**
2	**0.0824**	**0.9669**	**0.3145**	**0.9077**
3	**0.1671**	**0.9321**	**0.2514**	**0.9077**
4	**0.0681**	**0.9804**	**0.1483**	**0.9414**
5	**0.0047**	**0.9992**	**0.2985**	**0.9440**
6	**0.0059**	**0.9981**	**0.3188**	**0.9398**
7	**0.1459**	**0.9356**	**0.2244**	**0.9082**
8	**0.0842**	**0.9688**	**0.1900**	**0.9331**

Table 7: Models Training and Testing Performance for Cochleagram

	Spectrogram			
Model	Train Loss	Train Accu	Test Loss	Test Accu
1	**0.0004**	**1.0000**	**0.8759**	**0.8481**
2	**0.0006**	**1.0000**	**0.9067**	**0.8514**
3	**0.0280**	**0.9950**	**0.4915**	**0.8431**
4	**0.0157**	**0.9983**	**0.4029**	**0.8648**
5	**0.0011**	**1.0000**	**0.6502**	**0.8765**
6	**0.0349**	**0.9887**	**0.7701**	**0.8264**
7	**0.1644**	**0.9387**	**0.3634**	**0.8331**
8	**0.0420**	**0.9883**	**0.4294**	**0.8564**

Table 8: Models Training and Testing Performance for Spectrogram

	MelSpectrogram			
Model	Train Loss	Train Accu	Test Loss	Test Accu
1	**0.0544**	**0.9766**	**1.0029**	**0.8739**
2	**0.0101**	**0.9965**	**0.6968**	**0.9063**
3	**0.0678**	**0.9858**	**0.2428**	**0.9135**
4	**0.0220**	**0.9934**	**0.4554**	**0.9045**
5	**0.0162**	**0.9960**	**0.4630**	**0.8973**
6	**0.0181**	**0.9934**	**0.8562**	**0.8937**
7	**0.0874**	**0.9743**	**0.2646**	**0.8865**
8	**0.0884**	**0.9788**	**0.2728**	**0.8793**

Table 9: Models Training and Testing Performance for MelSpectrogram

The training results in Figures 43 and 44 provide an interesting view of the results with models 4, 5 and 6 performing very well with all three data types. We could say that the performance of the classifiers from model 4, 5 and 6 in relation to the data types requires further scrutiny. The testing data split results will be reviewed next to better determine the best classifiers by model/data combinations.

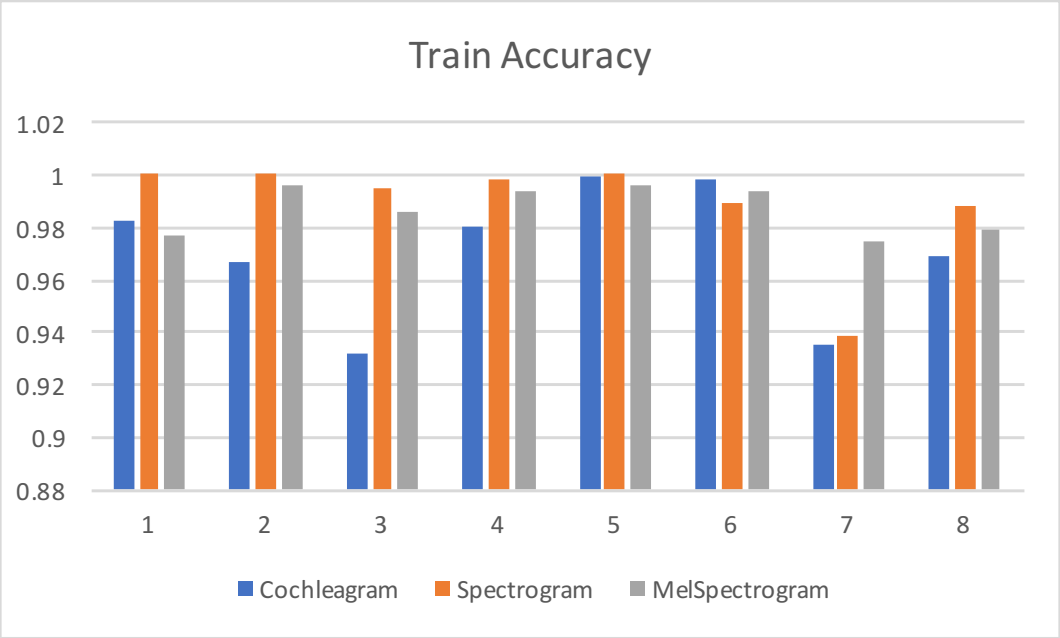


Figure 43: Models Train Accuracy for Cochleagram, Spectrogram & MelSpectrogram

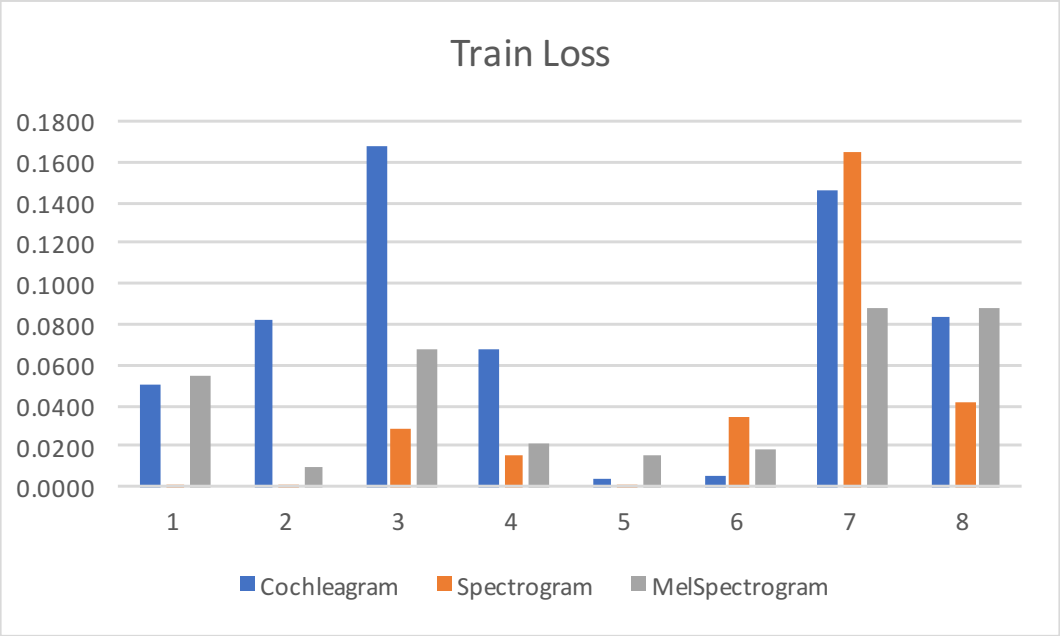


Figure 44: Models Train loss for Cochleagram, Spectrogram & MelSpectrogram

The test samples results in Figures 45 and 46 show the performance of the cochleagram data classifiers provide the results in loss and accuracy. The spectrogram and melspectrogram classifiers performed well during training but the difference between the training and testing results were considerably higher that the cochleagram, a sign of weak generalization and high degree of overfitting.

Figures 47 and 48 provide a classifier generalization measure by showing the classifier comparative loss and accuracy on test and train data. We can see that the cochleagram training results track each other better for all our models and both Loss and Accuracy.

This cochleagram data performance in comparison to spectrograms has been previously reported in the literature [35].

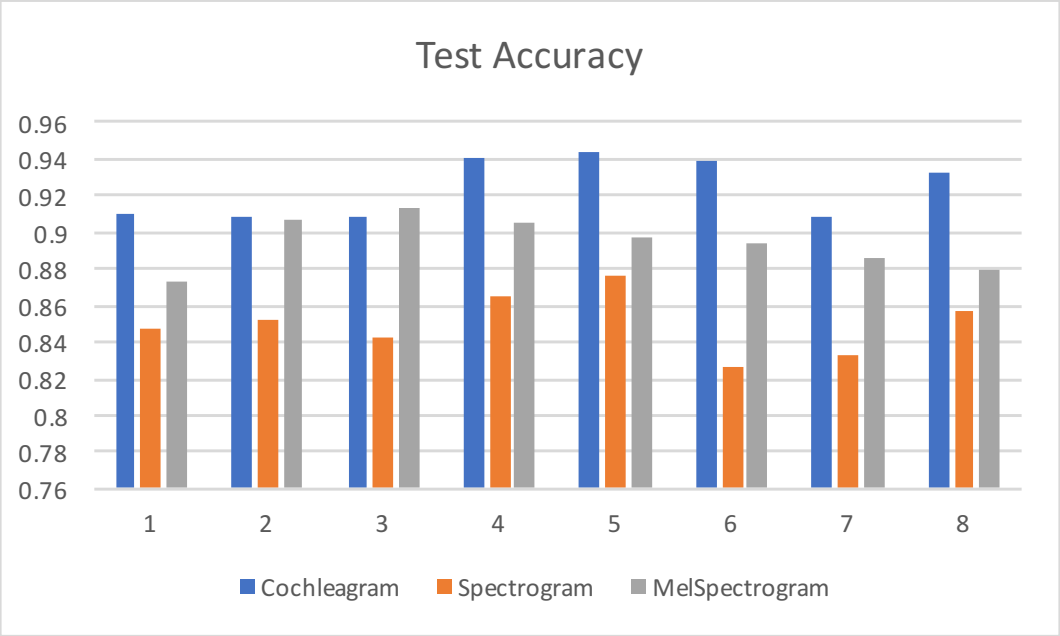


Figure 45: Models Test Accuracy for Cochleagram, Spectrogram & MelSpectrogram

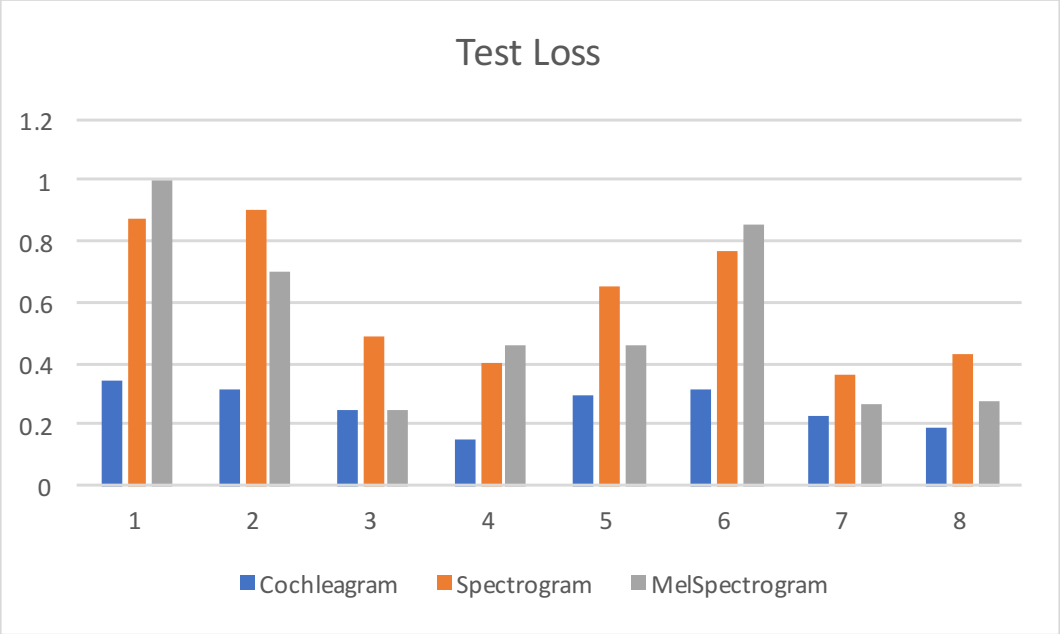


Figure 46: Models Test loss for Cochleagram, Spectrogram & MelSpectrogram

accuracy variation.pdf

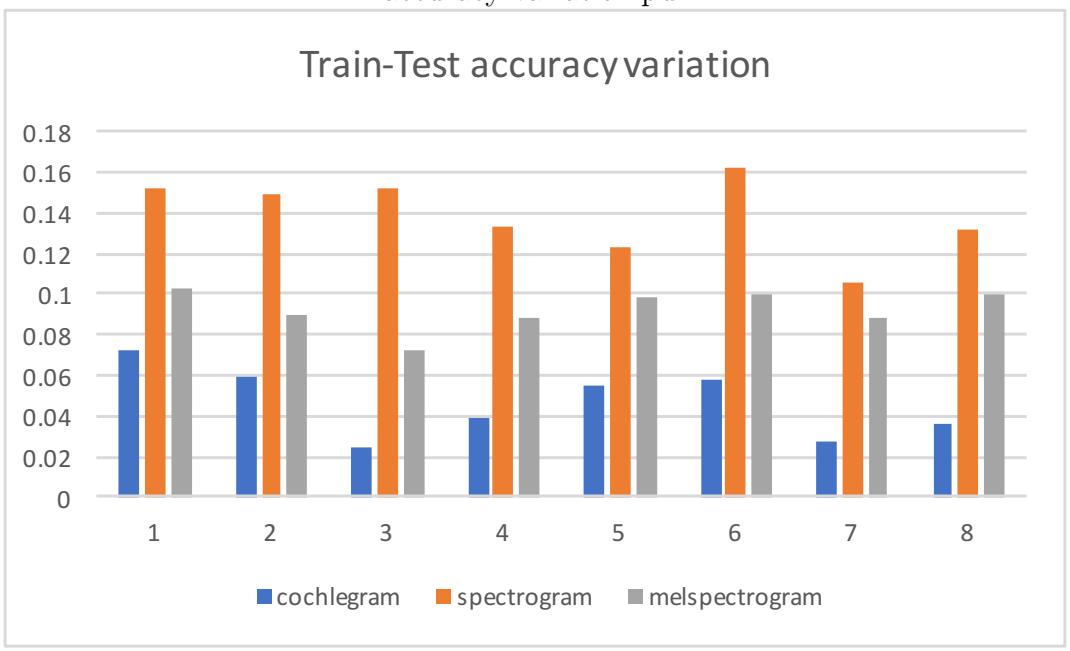


Figure 47: Models Train-Test Accuracy Variance

loss variation.pdf

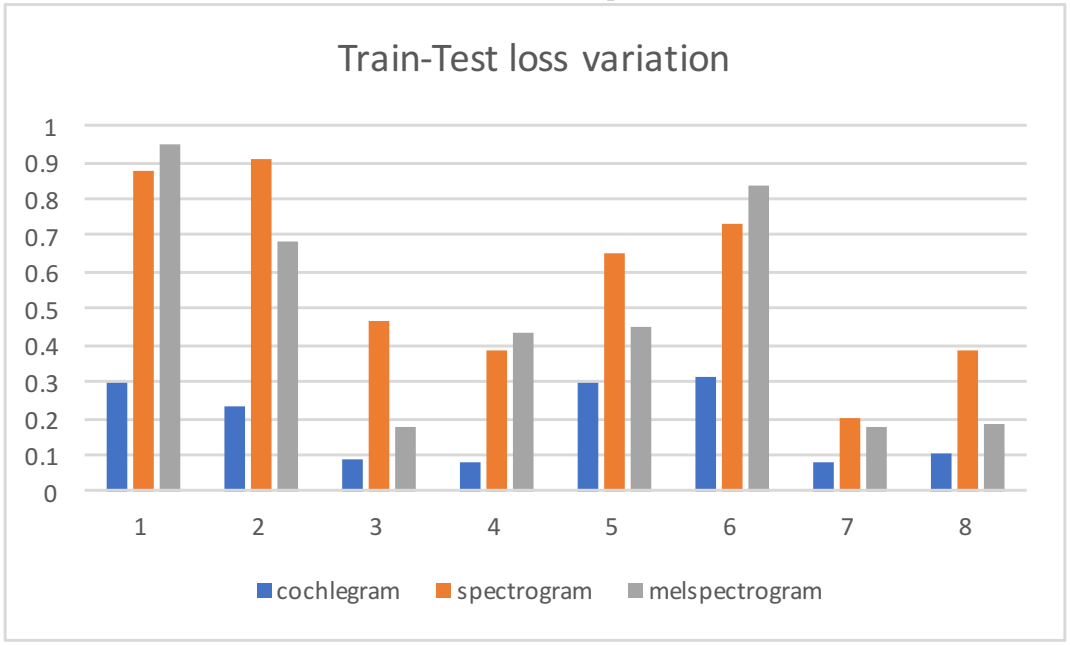


Figure 48: Models Train-Test Loss Variance

With the cochleagram's superior performance on all models we selected this data type for additional experimentation. We went on to train the three models 4, 5 and

6 under different conditions and determined that Model 6 (Figure 37) provides the best overall results.

The disadvantages of large matrices consist of long computation time requirements, the high data collection requirement and storage demands. *Python* has a limit of 2Gb files which we reached during spectrogram data extraction and analysis. One could split the data into multiple files but large data files management could prove cumbersome and expensive. Also, the computational time is considerably different for various file. We tried different matrix sizes to determine the impact of matrix size on the model accuracy with our leader models. The three cochleagram matrix sizes we experimented with were 100x50 pixels, 50x25, 40x20. The times per epoch and the accuracies we achieved are displayed in Table 10.

The only data type we experimented with was the cochleagram. The results were interesting, the best performance was achieved with the 50x25 matrices. The three matrix sizes we compared with training and accuracy results are shown in Table 10 .

Table 10: Cochleagram Matrix Training Performance

Matrix Size	Training duration per epoch	Acuracy
100x50	324 sec.	93.1 %
50x25	67 sec.	96.4 %
40x20	10 sec.	94.9 %

Figures 49, 50, 51 display the ROC curves for the three sizes we explored.

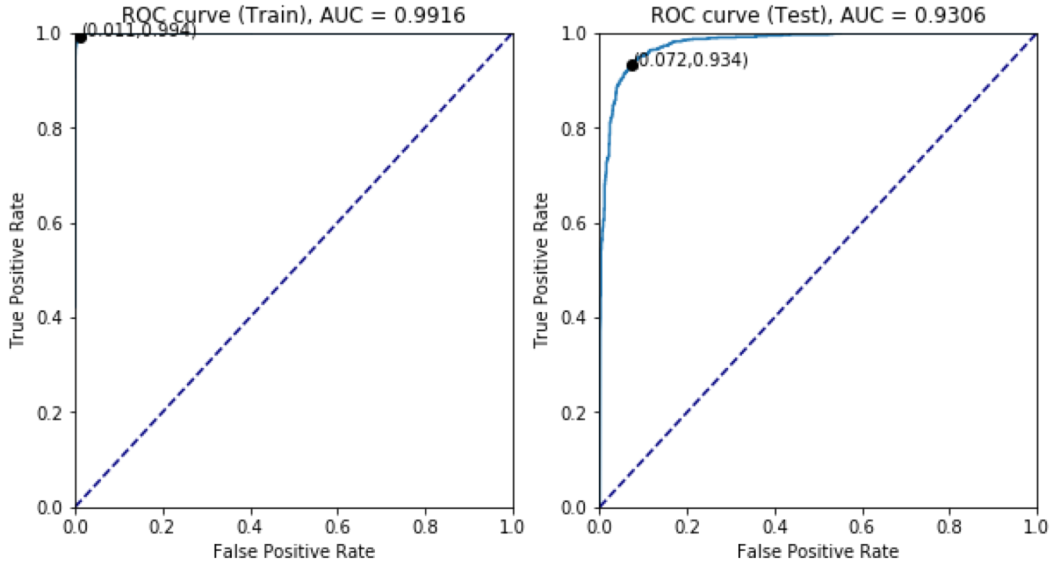


Figure 49: Cochleagram 100x50 Model 6 performance
Accuracy: 93.1%, Duration:324 seconds per epoch

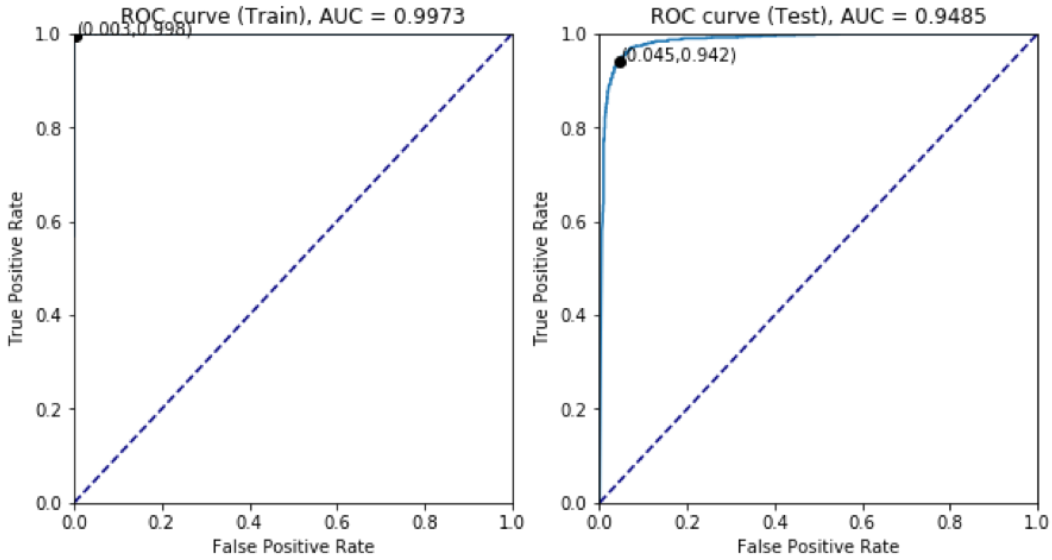


Figure 50: Cochleagram 40x20 Model 6 performance
Accuracy: 94.9%, Duration:10 seconds per epoch

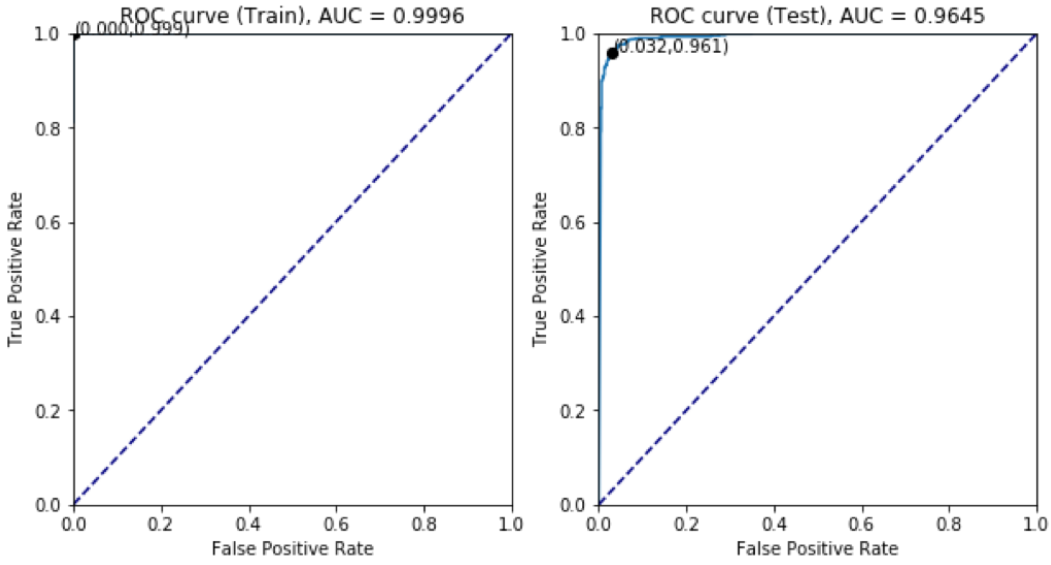


Figure 51: Cochleagram 50x25 Model 6 performance
Accuracy: 96.4%, Duration:67 seconds per epoch

On the Average the training for the cochlegrams took 100 epochs. The amount of time consumed was therefore not trivial when considering a 100x50 matrix training duration coming at 32,400 seconds or 9 hours, the 50x25 matrix came in at 67 seconds or 111 minutes for 100 epochs, and 40x20 matrix took 10 seconds per epoch or 17 minutes for 100 epochs.

We found Model 6 with cochleagram spectral analysis and a matrix of 50x25 pixel the optimal combination of meta parameters. We analyzed and remedied random UCLASS speeches and were able to eliminate 80% of all ***interjections***.

Note that we arrived to the results described above with ***CNN*** networks that utilize Max Pooling. The section below considers an alternative pooling technique.

5.1.8 Approach Spectral-analysis - Average vs Max Pooling

The general consensus in the literature with respect to pooling techniques is that Max Pooling models perform better that Average Pooling. But we chose to experiment with two Pooling techniques: MaxPooling and AveragePooling.

After systematic trials, we reached results which were not consistent with the general belief that Max Pooling is better, at least not for our cochleagram data and interjection detection. These tests were conducted with the original eight plus two additional models for a total of ten ***CNN***s. We trained each model twice, one time with ***Max-Pooling*** and again with ***Average-Pooling*** for a maximum of 200 epochs and early stopping. The accuracy of the results for ***Max-Pooling*** and ***Average-Pooling*** pooling are graphically depicted in Figure 52. The majority of models performed considerably better with Average pooling, those models which showed better Max-Pooling results were only marginally better. In summary, Average pooled ***CNN*** models proved superior to the ***Max-pooled*** ones for cochleagram spectrograms in our case.

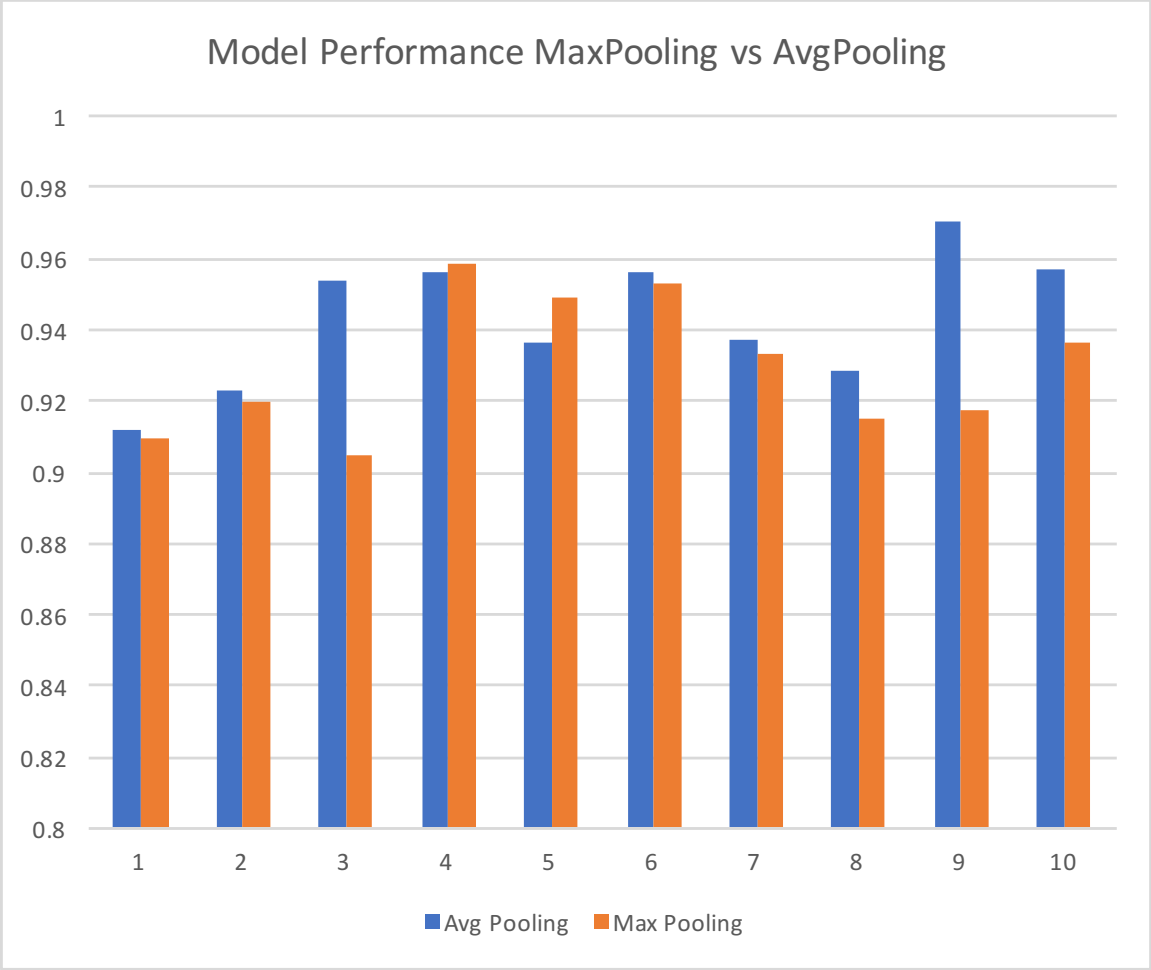


Figure 52: Pooling Impact on Classifier Accuracy

The numeric table with the exact accuracy results is shown in Table 11.

Table 11: Avg vs Max Pooling Performance

Model	Avg Pooling	Max Pooling
1	91.21%	90.95%
2	92.26%	92.00%
3	95.40%	90.48%
4	95.61%	95.87%
5	93.62%	94.93%
6	95.61%	95.29%
7	93.72%	93.31%
8	92.83%	91.53%
9	97.07%	91.74%
10	95.66%	93.62%

CHAPTER 6: CONCLUSION

6.1 Conclusion - Approach Pre-qualification

In ***Approach Pre-qualification*** we implemented *Blocked* speech remediation designed to perform one remediation function i.e. *blocked* speech segment removal. The approach was based on a *pre-qualification* stage. We were successful in classifying various segments of speech detected under different extraction constrains. Our ability to create segments derived with different perspectives generated overlapping segments of speech, thus a large number of ***block*** candidate segments. The tedium of manually labeling a large population of speech segments was addressed by devising a process which simplified the labeling effort.

With the labeling completed we confirmed our hypothesis that eliminating *Ground Truth blocked* segments improves speech quality. Recordings repaired by cutting out entire ***block*** segments showed a marked improvement in the speech. To automate the process of labeling candidate speech segments, frequency and time domain signal analysis was performed on the candidate segments followed by statistical analysis to create the features for classifier training. The trained models were tested and their performances tabulated.

The accuracy of the top performing classifier models was high; removing segments based on its predictions was performed and the remedied speech was compared to the

Gold Standard speech. The top classifier repaired recording was at par with the *Gold Standard* remedied recording.

This approach proved effective in eliminating the vast majority of voice ***blocks*** when applied to a ***blocks*** prone stuttered speech.

The effectiveness of our remediation process was highly dependent on the speech intensity threshold during the pre-qualification stage.

As an added measure we implemented a voting based classification system to reduce speech meaning loss by tilting the model in favor of high specificity at the expense of retaining a relatively larger number of ***blocked*** segments.

6.2 Conclusion - Approach Random-sampling

In order to streamline the detection process and eliminate the pre-qualification stage, we introduced an enhancement which would expand the types of disfluency types to include arbitrary speech anomalies by removing the dependency on a complex pre-qualification step. The data samples for training were selected from entire speeches and not only the pre-qualified segments. The results of ***Approach Random-sampling*** were of improved accuracy thanks to the unbiased samples pool.

The training and testing of the classifier followed a similar methodology to ***Approach Pre-qualification*** where multiple classifier algorithms were trained and their performance evaluated.

The main accomplishment of ***Approach Random-sampling*** was the implementation of a sliding window classifier. The results of scoring every segment of a ***block*** prone speech were graphed and manually evaluated. The comparison revealed com-

plete consistency between the listener's audio stutter identification and the ***GBM*** classifier's graph.

Although ***Approach Pre-qualification*** performed adequately with a set of pre-qualified samples and application, its shortcomings during the sliding classifier application were evidence of the importance of training models with data which resembles production data.

We find ***Approach Random-sampling***'s implications in terms of simplicity, flexibility and performance a big advancement over ***Approach Pre-qualification*** and confirmation of the positive implications of adequate sampling. This approach also proved to be research friendly by making crowd sourcing readily accessible [47].

6.3 Conclusion - Approach Spectral-analysis

Approach Spectral-analysis was designed to detect another type of stutter: ***interjections***. We tried the sampling, feature engineering and model algorithms used in prior approaches but the performance of our classifiers was inadequate. The features used previously failed to properly train our models.

We would not use the statistical features or the model algorithms from the prior experiments, instead we used spectral analysis results as predictors. We elected to use three different types of spectral analysis data, Cochleagram, Spectrogram, Mel-Spectrogram with multiple *CNN* models.

As we labeled the training speech segments we created a new technique of compiling speech data by separating all samples of a class ***interjection*** from the class normal and creating two long duration recordings which we would harvest for training dataset

creation.

Although the other two spectral analysis matrices performed well on training data, the cochleagram analysis provided the best results on test data. The differences in model performance between training and testing was consistently better with cochleagram which became our spectral analysis data type of choice.

Approach Spectral-analysis made use of a matrix zooming technique which casted the matrices to a consistent dimensions for proper image comparison. This zooming technique was further explored to evaluate the impact of varying image sizes on the ***CNN*** classifier performance where we found that the optimal image size is not always the largest one. Of the three different sizes we tried, the medium size matrix was consistently the best performing one.

Our analysis of image sizes was not exhaustive but we find that the image size 50x25 was close to optimal since bigger and smaller images yielded worse classifiers. The smaller representation did not only provide smaller storage requirements benefit, it proved considerably faster in training and scoring. Based on these findings, subsequent experimentation was performed on 50x25 cochleagram representations.

In ***Approach Spectral-analysis*** we experimented with the two pooling techniques, ***Max-Pooling*** and ***Average-Pooling*** with the previously developed 8 models in addition to two newly designed models. This was a valuable experiment given the superior results we achieved with the majority of our ***CNN***s and especially the two new models when using Average pooling instead of Max pooling.

REFERENCES

[1] Abdel-Hamid, O., Mohamed, A. R., Jiang, H., Deng, L., Penn, G., and Yu, D. Convolutional neural networks for speech recognition. IEEE/ACM Transactions on audio, speech, and language processing 22.10 (2014): 1533-1545.

[2] Ai, O. C., Hariharan, M., Yaacob, S., Chee, L. S. (2012). Classification of speech dysfluencies with MFCC and LPCC features. Expert Systems with Applications, 39(2), 2157-2165.

[3] Boey, R. A., Wuyts, F. L., Van de Heyning, P. H., De Bodt, M. S., Heylen, L. (2007). Characteristics of stuttering-like disfluencies in Dutch-speaking children. Journal of fluency disorders, 32(4), 310-329.

[4] Brown, G.J. and Cooke, M.P. (1994) Computational auditory scene analysis. Computer Speech and Language, 8: 297-336.

[5] Chee, L. S., Ai, O. C., Yaacob, S. (2009, October). Overview of automatic stuttering recognition system. In Proc. International Conference on Man-Machine Systems, no. October, Batu Ferringhi, Penang Malaysia (pp. 1-6).

[6] Chee, L. S., Ai, O. C., Hariharan, M., Yaacob, S. (2009, December). Automatic detection of prolongations and repetitions using LPCC. InTechnical Postgraduates (TECHPOS), 2009 International Conference for (pp. 1-4). IEEE.

[7] Chen, L. (2008). Incorporating Nonverbal Features Into Multimodel Models (Doctoral dissertation, Purdue University West Lafayette).

[8] Chen, J., Wang, Y., & Wang, D. A feature study for classification-based speech separation at low signal-to-noise ratios. IEEE/ACM Transactions on Audio, Speech, and Language Processing 22.12 (2014): 1993-2002.

[9] Click, C., Lanford, J., Malohlava, M., Parmar, V. and Roark, H. Gradient Boosted Models, http://h2o.ai/resources, August 2015

[10] Czyzewski, A., Kaczmarek, A., Kostek, B. Intelligent processing of stuttered speech. Journal of Intelligent Information Systems 21.2 (2003): 143-171.

[11] Deng, L., Hinton, G., and Kingsbury, B. New types of deep neural network learning for speech recognition and related applications: An overview. Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on. IEEE, 2013.

[12] Esmaili, I., Dabanloo, N. J., Vali, M. "Automatic classification of speech dysfluencies in continuous speech based on similarity measures and morphological image processing tools." Biomedical Signal Processing and Control 23 (2016): 104-114.

[13] Ferguson, J., Durrett, G., Klein, D. (2015, June). Disfluency Detection with a Semi-Markov Model and Prosodic Features. In Proc. NAACL HLT.

[14] Fook, C. Y., Muthusamy, H., Chee, L. S., Yaacob, S. B., Adom, A.H. (2013). Comparison of speech parameterization techniques for the classification of speech

disfluencies. Turkish Journal of Electrical Engineering Computer Sciences, 21(Sup. 1), 1983-1994.

[15] Geetha, Y. V., Pratibha, K., Ashok, R., Ravindra, S. K. (2000). Classification of childhood disfluencies using neural networks. Journal of fluency disorders, 25(2), 99-117.

[16] Graves, A., Abdel-rahman, M., and Hinton, G. Speech recognition with deep recurrent neural networks. Acoustics, speech and signal processing (icassp), 2013 ieee international conference on. IEEE, 2013.

[17] Hariharan, M., Chee, L. S., Ai, O. C., Yaacob, S. (2012). Classification of speech dysfluencies using LPC based parameterization techniques. Journal of medical systems, 36(3), 1821-1830.

[18] Honal, M., Schultz, T. (2005, March). Automatic Disfluency Removal on Recognized Spontaneous Speech-Rapid Adaptation to Speaker Dependent Disfluencies. In ICASSP (1) (pp. 969-972).

[19] Honal, M., Schultz, T. Correction of disfluencies in spontaneous speech using a noisy-channel approach. Eighth European Conference on Speech Communication and Technology. 2003.

[20] Howell, P., Sackin, S. (1995, August). Automatic recognition of repetitions and prolongations in stuttered speech. In Proceedings of the first World Congress on fluency disorders (Vol. 2, pp. 372-374).

[21] Howell, P., Davis, S., Bartrip, J. (2009). The University College London archive of stuttered speech (UCLASS). Journal of Speech, Language, and Hearing Research, 52(2), 556-569.

[22] Huang, Z., Chen, L., Harper, M. (2006, May). An open source prosodic feature extraction tool. In Proceedings of the Language Resources and Evaluation Conference (LREC).

[23] Huang, Z., Chen, L., Harper, M. P. (2006). Purdue Prosodic Feature Extraction Tool on Praat.

[24] Jaitly, N., Nguyen, P., Senior, A., and Vanhoucke, V. Application of pretrained deep neural networks to large vocabulary speech recognition. Thirteenth Annual Conference of the International Speech Communication Association. 2012.

[25] Kingma, D. P., Ba, J. "Adam: A method for stochastic optimization." arXiv preprint arXiv:1412.6980 (2014).

[26] Km, R. K., Ganesan, S. (2011). Comparison of multidimensional MFCC feature vectors for objective assessment of stuttered disfluencies. age, 2(05), 854-860.

[27] Kons, Z., Satt, A., Hoory, R., Uloza, V., Vaiciukynas, E., Gelzinis, A., Bacauskiene, M. "On feature extraction for voice pathology detection from speech signals." Proceedings of the 1st Annual Afeka-AVIOS Speech Processing Conference, Tel Aviv Academic College of Engineering, Tel Aviv, Israel. 2011.

[28] Lease, M., Johnson, M., Charniak, E.: Recognizing disfluencies in conversational speech. In: IEEE Transactions on Audio, Speech, and Language Processing, 14(5), 1566-1573 (2006).

[29] LeCun, Y., and Yoshua B. Convolutional networks for images, speech, and time series The handbook of brain theory and neural networks 3361.10 (1995): 1995.

[30] Lei, Y., Scheffer, N., Ferrer, L., and McLaren, M. A novel scheme for speaker recognition using a phonetically-aware deep neural network. Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on. IEEE, 2014.

[31] Liu, Y., Shriberg, E., Stolcke, A., Harper, M. P. (2005, September). Comparing HMM, maximum entropy, and conditional random fields for disfluency detection. In INTERSPEECH (pp. 3313-3316).

[32] Liu, Y., Shriberg, E., Stolcke, A., Hillard, D., Ostendorf, M., Harper, M. (2006). Enriching speech recognition with automatic detection of sentence boundaries and disfluencies. IEEE Transactions on audio, speech, and language processing, 14(5), 1526-1540.

[33] Mareüil, P. B. D., Habert, B., Bénard, F., Adda-Decker, M., Barras, C., Adda, G., and Paroubek, P. A quantitative study of disfluencies in French broadcast interviews. Disfluency in Spontaneous Speech. 2005.

[34] Medeiros, H., Moniz, H., Batista, F., Trancoso, I., Nunes, L. (2013, July). Disfluency detection based on prosodic features for university lectures. In INTERSPEECH (pp. 2629-2633).

[35] Muthusamy, Y. K., Ronald A. C., & Slaney, M. Speaker-independent vowel recognition: Spectrograms versus cochleagrams. Acoustics, Speech, and Signal Processing, 1990. ICASSP-90., 1990 International Conference on. IEEE, 1990.

[36] Oppenheim, A. V. "Speech spectrograms using the fast Fourier transform." IEEE spectrum 7.8 (1970): 57-62.

[37] Palfy, J. (2014). Analysis of dysfluencies by computational intelligence.Information Sciences and Technologies, 6(2), 45.

[38] Rabiner, L., Juang, B. (1986). An Introduction to Hidden Markov Models. IEEE ASSP Magazine, 3(1), 4-16.

[39] Ravikumar, K. M., R. Rajagopal, and H. C. Nagaraj. "An approach for objective assessment of stuttered speech using MFCC." The International Congress for Global Science and Technology. 2009.

[40] Ribeiro, M. T., Singh, S., & Guestrin, C. (2016, August). Why should i trust you?: Explaining the predictions of any classifier. In Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining (pp. 1135-1144). ACM.

[41] Shriberg, E. (2001). Toerrrr'is human: ecology and acoustics of speech disfluencies. Journal of the International Phonetic Association, 31(1), 153-164.

[42] Shriberg, E., Stolcke, A., Hakkani-TÅr, D., TÅr, G. (2000). Prosody-based automatic segmentation of speech into sentences and topics. Speech communication, 32(1), 127-154.

[43] Stouten, F., Duchateau, J., Martens, J. P., Wambacq, P. (2006). Coping with disfluencies in spontaneous speech recognition: Acoustic detection and linguistic context manipulation. Speech Communication, 48(11), 1590-1606.

[44] Sumit, S. A Comprehensive Guide to Convolutional Neural Networks the ELI5 way. Retrieved from https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[45] Wiśniewski, M., Kuniszyk-Jóźkowiak, W. (2011). Automatic detection and classification of phoneme repetitions using HTK toolkit. Journal of Medical Informatics Technologies, 17, 141-147.

[46] Wiśniewski, M., Kuniszyk-Jóźkowiak, W., Smołka, E., Suszyński, W. (2007). Automatic detection of prolonged fricative phonemes with the hidden Markov models approach. Journal of Medical Informatics Technologies, 11, 2007.

[47] Winkelmann,R.,Raess,G.(2014,May).Introducing a web application for labeling, visualizing speech and correcting derived speech signals. In LREC (pp. 4129-4133).

Appendix A - Model Definitions

MODEL: <<<< 1 >>>>>

```
Layer (type)                 Output Shape              Param #
=================================================================
=
conv2d (Conv2D)              (None, 48, 23, 256)       2560
activation (Activation)      (None, 48, 23, 256)       0
max_pooling2d (MaxPooling2D) (None, 24, 11, 256)       0
conv2d_1 (Conv2D)            (None, 22, 9, 256)        590080
activation_1 (Activation)    (None, 22, 9, 256)        0
max_pooling2d_1 (MaxPooling2 (None, 11, 4, 256)        0
flatten (Flatten)            (None, 11264)             0
dense (Dense)                (None, 64)                720960
dense_1 (Dense)              (None, 1)                 65
activation_2 (Activation)    (None, 1)                 0
=================================================================
=
Total params: 1,313,665
Trainable params: 1,313,665
Non-trainable params: 0
```

MODEL: <<<< 2 >>>>>

```
Layer (type)                 Output Shape              Param #
=================================================================
=
conv2d_2 (Conv2D)            (None, 48, 23, 128)       1280
activation_3 (Activation)    (None, 48, 23, 128)       0
max_pooling2d_2 (MaxPooling2 (None, 9, 11, 128)        0
conv2d_3 (Conv2D)            (None, 7, 9, 64)          73792
activation_4 (Activation)    (None, 7, 9, 64)          0
max_pooling2d_3 (MaxPooling2 (None, 1, 4, 64)          0
flatten_1 (Flatten)          (None, 256)               0
dense_2 (Dense)              (None, 32)                8224
dense_3 (Dense)              (None, 1)                 33
activation_5 (Activation)    (None, 1)                 0
=================================================================
=
Total params: 83,329
```

```
Trainable params: 83,329
Non-trainable params: 0
```

MODEL: <<<< 3 >>>>>

```
Layer (type)                 Output Shape              Param #
=================================================================
=
conv2d_4 (Conv2D)            (None, 48, 23, 32)        320
dropout (Dropout)            (None, 48, 23, 32)        0
activation_6 (Activation)    (None, 48, 23, 32)        0
max_pooling2d_4 (MaxPooling2 (None, 24, 11, 32)        0
conv2d_5 (Conv2D)            (None, 22, 9, 32)         9248
dropout_1 (Dropout)          (None, 22, 9, 32)         0
activation_7 (Activation)    (None, 22, 9, 32)         0
max_pooling2d_5 (MaxPooling2 (None, 11, 4, 32)         0
conv2d_6 (Conv2D)            (None, 9, 2, 64)          18496
dropout_2 (Dropout)          (None, 9, 2, 64)          0
activation_8 (Activation)    (None, 9, 2, 64)          0
max_pooling2d_6 (MaxPooling2 (None, 4, 1, 64)          0
flatten_2 (Flatten)          (None, 256)               0
dense_4 (Dense)              (None, 64)                16448
dropout_3 (Dropout)          (None, 64)                0
activation_9 (Activation)    (None, 64)                0
dense_5 (Dense)              (None, 1)                 65
activation_10 (Activation)   (None, 1)                 0
=================================================================
=
Total params: 44,577
Trainable params: 44,577
Non-trainable params: 0
```

MODEL: <<<< 4 >>>>>

```
Layer (type)                 Output Shape              Param #
=================================================================
=
conv2d_7 (Conv2D)            (None, 48, 23, 64)        640
activation_11 (Activation)   (None, 48, 23, 64)        0
average_pooling2d (AveragePo (None, 24, 23, 64)        0
```

```
conv2d_8 (Conv2D)            (None, 22, 21, 64)        36928
activation_12 (Activation)   (None, 22, 21, 64)        0
average_pooling2d_1 (Average (None, 11, 21, 64)        0
conv2d_9 (Conv2D)            (None, 9, 19, 64)         36928
activation_13 (Activation)   (None, 9, 19, 64)         0
average_pooling2d_2 (Average (None, 4, 19, 64)         0
flatten_3 (Flatten)          (None, 4864)              0
dense_6 (Dense)              (None, 64)                311360
dropout_4 (Dropout)          (None, 64)                0
activation_14 (Activation)   (None, 64)                0
dense_7 (Dense)              (None, 64)                4160
dropout_5 (Dropout)          (None, 64)                0
activation_15 (Activation)   (None, 64)                0
dense_8 (Dense)              (None, 1)                 65
activation_16 (Activation)   (None, 1)                 0
=================================================================
=
Total params: 390,081
Trainable params: 390,081
Non-trainable params: 0
```

MODEL: <<<< 5 >>>>>

```
Layer (type)                 Output Shape              Param #
=================================================================
=
conv2d_10 (Conv2D)           (None, 50, 25, 16)        160
activation_17 (Activation)   (None, 50, 25, 16)        0
average_pooling2d_3 (Average (None, 5, 12, 16)         0
conv2d_11 (Conv2D)           (None, 5, 12, 32)         4640
activation_18 (Activation)   (None, 5, 12, 32)         0
average_pooling2d_4 (Average (None, 2, 6, 32)          0
conv2d_12 (Conv2D)           (None, 2, 6, 64)          18496
activation_19 (Activation)   (None, 2, 6, 64)          0
average_pooling2d_5 (Average (None, 1, 3, 64)          0
flatten_4 (Flatten)          (None, 192)               0
dense_9 (Dense)              (None, 128)               24704
dropout_6 (Dropout)          (None, 128)               0
activation_20 (Activation)   (None, 128)               0
```

```
dense_10 (Dense)             (None, 1)                 129
activation_21 (Activation)   (None, 1)                 0
=================================================================
=
Total params: 48,129
Trainable params: 48,129
Non-trainable params: 0
```

MODEL: <<<< 6 >>>>>

```
Layer (type)                 Output Shape              Param #
=================================================================
=
conv2d_13 (Conv2D)           (None, 50, 25, 96)        2496
activation_22 (Activation)   (None, 50, 25, 96)        0
conv2d_14 (Conv2D)           (None, 50, 25, 96)        83040
activation_23 (Activation)   (None, 50, 25, 96)        0
conv2d_15 (Conv2D)           (None, 50, 25, 96)        83040
activation_24 (Activation)   (None, 50, 25, 96)        0
conv2d_16 (Conv2D)           (None, 50, 25, 96)        83040
activation_25 (Activation)   (None, 50, 25, 96)        0
average_pooling2d_6 (Average (None, 5, 12, 96)         0
conv2d_17 (Conv2D)           (None, 5, 12, 128)        110720
activation_26 (Activation)   (None, 5, 12, 128)        0
conv2d_18 (Conv2D)           (None, 5, 12, 128)        147584
activation_27 (Activation)   (None, 5, 12, 128)        0
conv2d_19 (Conv2D)           (None, 5, 12, 128)        147584
activation_28 (Activation)   (None, 5, 12, 128)        0
average_pooling2d_7 (Average (None, 2, 6, 128)         0
flatten_5 (Flatten)          (None, 1536)              0
dense_11 (Dense)             (None, 64)                98368
dropout_7 (Dropout)          (None, 64)                0
activation_29 (Activation)   (None, 64)                0
dense_12 (Dense)             (None, 64)                4160
dropout_8 (Dropout)          (None, 64)                0
activation_30 (Activation)   (None, 64)                0
dense_13 (Dense)             (None, 1)                 65
activation_31 (Activation)   (None, 1)                 0
=================================================================
```

```
=
Total params: 760,097
Trainable params: 760,097
Non-trainable params: 0
```

MODEL: <<<< 7 >>>>>

```
Layer (type)                 Output Shape              Param #
=================================================================
=
conv2d_20 (Conv2D)           (None, 50, 25, 64)        640
dropout_9 (Dropout)          (None, 50, 25, 64)        0
activation_32 (Activation)   (None, 50, 25, 64)        0
conv2d_21 (Conv2D)           (None, 50, 25, 64)        36928
dropout_10 (Dropout)         (None, 50, 25, 64)        0
activation_33 (Activation)   (None, 50, 25, 64)        0
max_pooling2d_7 (MaxPooling2 (None, 5, 12, 64)         0
conv2d_22 (Conv2D)           (None, 5, 12, 128)        73856
dropout_11 (Dropout)         (None, 5, 12, 128)        0
activation_34 (Activation)   (None, 5, 12, 128)        0
max_pooling2d_8 (MaxPooling2 (None, 2, 6, 128)         0
conv2d_23 (Conv2D)           (None, 2, 6, 256)         295168
dropout_12 (Dropout)         (None, 2, 6, 256)         0
activation_35 (Activation)   (None, 2, 6, 256)         0
max_pooling2d_9 (MaxPooling2 (None, 1, 3, 256)         0
flatten_6 (Flatten)          (None, 768)               0
dense_14 (Dense)             (None, 512)               393728
activation_36 (Activation)   (None, 512)               0
flatten_7 (Flatten)          (None, 512)               0
dense_15 (Dense)             (None, 512)               262656
activation_37 (Activation)   (None, 512)               0
dense_16 (Dense)             (None, 512)               262656
activation_38 (Activation)   (None, 512)               0
dense_17 (Dense)             (None, 1)                 513
activation_39 (Activation)   (None, 1)                 0
=================================================================
=
Total params: 1,326,145
Trainable params: 1,326,145
```

```
Non-trainable params: 0
```

MODEL: <<<< 8 >>>>>

```
Layer (type)                 Output Shape              Param #
=================================================================
=
conv2d_24 (Conv2D)           (None, 50, 25, 6)         60
dropout_13 (Dropout)         (None, 50, 25, 6)         0
activation_40 (Activation)   (None, 50, 25, 6)         0
max_pooling2d_10 (MaxPooling (None, 10, 25, 6)         0
conv2d_25 (Conv2D)           (None, 10, 25, 12)        660
dropout_14 (Dropout)         (None, 10, 25, 12)        0
activation_41 (Activation)   (None, 10, 25, 12)        0
max_pooling2d_11 (MaxPooling (None, 5, 12, 12)         0
conv2d_26 (Conv2D)           (None, 5, 12, 24)         2616
dropout_15 (Dropout)         (None, 5, 12, 24)         0
activation_42 (Activation)   (None, 5, 12, 24)         0
max_pooling2d_12 (MaxPooling (None, 2, 6, 24)          0
conv2d_27 (Conv2D)           (None, 2, 6, 48)          10416
dropout_16 (Dropout)         (None, 2, 6, 48)          0
activation_43 (Activation)   (None, 2, 6, 48)          0
max_pooling2d_13 (MaxPooling (None, 1, 3, 48)          0
flatten_8 (Flatten)          (None, 144)               0
dense_18 (Dense)             (None, 120)               17400
activation_44 (Activation)   (None, 120)               0
flatten_9 (Flatten)          (None, 120)               0
dense_19 (Dense)             (None, 120)               14520
activation_45 (Activation)   (None, 120)               0
dense_20 (Dense)             (None, 1)                 121
activation_46 (Activation)   (None, 1)                 0
=================================================================
=
Total params: 45,793
Trainable params: 45,793
Non-trainable params: 0
```

MODEL: <<<< 9 >>>>>

```
Layer (type)                 Output Shape              Param #
=================================================================
=
conv2d_28 (Conv2D)           (None, 50, 25, 64)        640
dropout_17 (Dropout)         (None, 50, 25, 64)        0
activation_47 (Activation)   (None, 50, 25, 64)        0
conv2d_29 (Conv2D)           (None, 50, 25, 64)        36928
dropout_18 (Dropout)         (None, 50, 25, 64)        0
activation_48 (Activation)   (None, 50, 25, 64)        0
max_pooling2d_14 (MaxPooling (None, 25, 12, 64)        0
conv2d_30 (Conv2D)           (None, 25, 12, 128)       73856
dropout_19 (Dropout)         (None, 25, 12, 128)       0
activation_49 (Activation)   (None, 25, 12, 128)       0
conv2d_31 (Conv2D)           (None, 25, 12, 128)       147584
dropout_20 (Dropout)         (None, 25, 12, 128)       0
activation_50 (Activation)   (None, 25, 12, 128)       0
max_pooling2d_15 (MaxPooling (None, 12, 6, 128)        0
conv2d_32 (Conv2D)           (None, 12, 6, 256)        295168
dropout_21 (Dropout)         (None, 12, 6, 256)        0
activation_51 (Activation)   (None, 12, 6, 256)        0
conv2d_33 (Conv2D)           (None, 12, 6, 256)        590080
dropout_22 (Dropout)         (None, 12, 6, 256)        0
activation_52 (Activation)   (None, 12, 6, 256)        0
max_pooling2d_16 (MaxPooling (None, 6, 3, 256)         0
conv2d_34 (Conv2D)           (None, 6, 3, 512)         1180160
dropout_23 (Dropout)         (None, 6, 3, 512)         0
activation_53 (Activation)   (None, 6, 3, 512)         0
conv2d_35 (Conv2D)           (None, 6, 3, 512)         2359808
dropout_24 (Dropout)         (None, 6, 3, 512)         0
activation_54 (Activation)   (None, 6, 3, 512)         0
max_pooling2d_17 (MaxPooling (None, 3, 1, 512)         0
flatten_10 (Flatten)         (None, 1536)              0
dense_21 (Dense)             (None, 512)               786944
activation_55 (Activation)   (None, 512)               0
flatten_11 (Flatten)         (None, 512)               0
dense_22 (Dense)             (None, 4096)              2101248
activation_56 (Activation)   (None, 4096)              0
```

```
dense_23 (Dense)             (None, 4096)              16781312
activation_57 (Activation)   (None, 4096)              0
dense_24 (Dense)             (None, 1)                 4097
activation_58 (Activation)   (None, 1)                 0
=================================================================
=
Total params: 24,357,825
Trainable params: 24,357,825
Non-trainable params: 0
```

MODEL: <<<< 10 >>>>>

```
Layer (type)                 Output Shape              Param #
=================================================================
=
conv2d_36 (Conv2D)           (None, 46, 21, 32)        832
dropout_25 (Dropout)         (None, 46, 21, 32)        0
activation_59 (Activation)   (None, 46, 21, 32)        0
average_pooling2d_8 (Average (None, 23, 10, 32)        0
conv2d_37 (Conv2D)           (None, 21, 8, 64)         18496
dropout_26 (Dropout)         (None, 21, 8, 64)         0
activation_60 (Activation)   (None, 21, 8, 64)         0
average_pooling2d_9 (Average (None, 10, 4, 64)         0
conv2d_38 (Conv2D)           (None, 8, 2, 128)         73856
dropout_27 (Dropout)         (None, 8, 2, 128)         0
activation_61 (Activation)   (None, 8, 2, 128)         0
average_pooling2d_10 (Averag (None, 4, 1, 128)         0
flatten_12 (Flatten)         (None, 512)               0
dense_25 (Dense)             (None, 64)                32832
dropout_28 (Dropout)         (None, 64)                0
activation_62 (Activation)   (None, 64)                0
dense_26 (Dense)             (None, 64)                4160
dropout_29 (Dropout)         (None, 64)                0
activation_63 (Activation)   (None, 64)                0
dense_27 (Dense)             (None, 1)                 65
activation_64 (Activation)   (None, 1)                 0
=================================================================
=
Total params: 130,241
```

```
Trainable params: 130,241
Non-trainable params: 0
```

MODEL: <<<< 11 >>>>>

```
Layer (type)                 Output Shape              Param #
=================================================================
=
conv2d_39 (Conv2D)           (None, 48, 23, 96)        960
activation_65 (Activation)   (None, 48, 23, 96)        0
conv2d_40 (Conv2D)           (None, 46, 21, 96)        83040
activation_66 (Activation)   (None, 46, 21, 96)        0
average_pooling2d_11 (Averag (None, 23, 21, 96)        0
conv2d_41 (Conv2D)           (None, 21, 19, 96)        83040
activation_67 (Activation)   (None, 21, 19, 96)        0
conv2d_42 (Conv2D)           (None, 19, 17, 96)        83040
activation_68 (Activation)   (None, 19, 17, 96)        0
conv2d_43 (Conv2D)           (None, 17, 15, 96)        83040
activation_69 (Activation)   (None, 17, 15, 96)        0
average_pooling2d_12 (Averag (None, 8, 15, 96)         0
conv2d_44 (Conv2D)           (None, 6, 13, 128)        110720
activation_70 (Activation)   (None, 6, 13, 128)        0
conv2d_45 (Conv2D)           (None, 4, 11, 128)        147584
activation_71 (Activation)   (None, 4, 11, 128)        0
conv2d_46 (Conv2D)           (None, 2, 9, 128)         147584
activation_72 (Activation)   (None, 2, 9, 128)         0
average_pooling2d_13 (Averag (None, 1, 9, 128)         0
flatten_13 (Flatten)         (None, 1152)              0
dense_28 (Dense)             (None, 64)                73792
dropout_30 (Dropout)         (None, 64)                0
activation_73 (Activation)   (None, 64)                0
dense_29 (Dense)             (None, 64)                4160
dropout_31 (Dropout)         (None, 64)                0
activation_74 (Activation)   (None, 64)                0
dense_30 (Dense)             (None, 64)                4160
dropout_32 (Dropout)         (None, 64)                0
activation_75 (Activation)   (None, 64)                0
dense_31 (Dense)             (None, 1)                 65
```

```
activation_76 (Activation)   (None, 1)                 0
=================================================================
=
Total params: 821,185
Trainable params: 821,185
Non-trainable params: 0
```

Appendix B - Model Definitions with Keras API

All Models Definition Keras

February 26, 2019

1 Model 1

In [ ]:
```
# model 1

        #-- layer 1
        model = Sequential()
        model.add(Conv2D(256, (3, 3),
                         input_shape=input_shape))
        model.add(Activation('relu'))
        model.add(MaxPooling2D(pool_size=(2, 2)))

        #-- layer 2
        model.add(Conv2D(256, (3, 3)))
        model.add(Activation('relu'))
        model.add(MaxPooling2D(pool_size=(2, 2)))

        #-- layer 3
        model.add(Flatten())
        model.add(Dense(64))

        model.add(Dense(1))
        model.add(Activation('sigmoid'))

        model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])

        # define early stopping callback
        earlystop = EarlyStopping(monitor='val_acc', min_delta=0.0001,
                                  patience = patience_val,
                                  verbose = verbosity, mode='auto')
        callbacks_list = [earlystop]
```

2 Model 2

In [ ]:
```
#-- layer 1
        model = Sequential()
        model.add(Conv2D(128, (3, 3), input_shape=X_train_norm_conv.shape[1:]))
        model.add(Activation('relu'))
```

```
model.add(MaxPooling2D(pool_size=(5, 2)))  # changed from (2,2) to (5,2)

#-- layer 2
model.add(Conv2D(64, (3, 3)))     # went from 256 to 64
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(4, 2)))

#-- layer 3
model.add(Flatten())  # this converts our 3D feature maps to 1D feature vectors
model.add(Dense(32))    # went from 64 to 32
model.add(Dense(1))
model.add(Activation('sigmoid'))

model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])

# define early stopping callback
earlystop = EarlyStopping(monitor='val_acc', min_delta=0.0001, patience = patience_val,
                          verbose = verbosity, mode='auto')

callbacks_list = [earlystop]
```

3 Model 3

In [ ]:
```
# model 3

#-- layer 1
model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=input_shape))
model.add(Dropout(0.3))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

#-- layer 2
model.add(Conv2D(32, (3, 3)))
model.add(Dropout(0.3))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

#-- layer 3
model.add(Conv2D(64, (3, 3)))
model.add(Dropout(0.3))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

#-- layer 4
model.add(Flatten())
model.add(Dense(64))
```

```
model.add(Dropout(0.5))
model.add(Activation('relu'))
model.add(Dense(1))
model.add(Activation('sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer='adam',metrics=['accuracy'])

earlystop = EarlyStopping(monitor='val_acc', min_delta=0.0001,
                          patience = patience_val,
                          verbose = verbosity, mode='auto')

callbacks_list = [earlystop]

# ##########################
```

4 Model 4

In [ ]:
```
# model 4
epochs = min(global_epochs, 200)
batch_size = 16

#-- layer 1
model = Sequential()
model.add(Conv2D(64, (3, 3), input_shape=input_shape))
model.add(Activation('relu'))
model.add(AveragePooling2D(pool_size=(2, 1)))

#-- layer 2
model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(AveragePooling2D(pool_size=(2, 1)))

#-- layer 3
model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(AveragePooling2D(pool_size=(2, 1)))

#-- layer 4
model.add(Flatten())
model.add(Dense(64))
model.add(Dropout(0.2))
model.add(Activation('relu'))

#-- layer 5
model.add(Dense(64))
model.add(Dropout(0.2))
```

```
model.add(Activation('relu'))

model.add(Dense(1))
model.add(Activation('sigmoid'))


model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])

earlystop = EarlyStopping(monitor='val_acc', min_delta=0.0001, patience = patience_val,
                          verbose = verbosity, mode='auto')
callbacks_list = [earlystop]
```

5 Model 5

In [ ]:
```
# model 5
epochs = min(global_epochs, 200)
batch_size = 16

#-- layer 1
model = Sequential()
model.add(Conv2D(16, (3, 3), input_shape=input_shape, padding='same'))
model.add(Activation('relu'))
model.add(AveragePooling2D(pool_size=(10, 2)))

#-- layer 2
model.add(Conv2D(32, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(AveragePooling2D(pool_size=(2, 2)))

#-- layer 3
model.add(Conv2D(64, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(AveragePooling2D(pool_size=(2, 2)))

#-- layer 4
model.add(Flatten())
model.add(Dense(128))
model.add(Dropout(0.5))
model.add(Activation('relu'))
model.add(Dense(1))
model.add(Activation('sigmoid'))


model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])

# define early stopping callback
earlystop = EarlyStopping(monitor='val_acc', min_delta=0.0001,
```

```
                                patience = patience_val,
                                verbose = verbosity, mode='auto')
        callbacks_list = [earlystop]
```

6 Model 6

In [ ]:
```
        # model 6
        epochs = min(global_epochs, 200)
        batch_size = 16

        ''' define the CNN model '''

        #-- layer 1
        model = Sequential()
        model.add(Conv2D(96, (5, 5), input_shape=input_shape, padding='same'))
        model.add(Activation('relu'))

        #-- layer 2
        model.add(Conv2D(96, (3, 3), input_shape=input_shape, padding='same'))
        model.add(Activation('relu'))

        #-- layer 3
        model.add(Conv2D(96, (3, 3), input_shape=input_shape, padding='same'))
        model.add(Activation('relu'))

        #-- layer 4
        model.add(Conv2D(96, (3, 3), input_shape=input_shape, padding='same'))
        model.add(Activation('relu'))
        model.add(AveragePooling2D(pool_size=(10, 2)))

        #-- layer 4
        model.add(Conv2D(128, (3, 3), input_shape=input_shape, padding='same'))
        model.add(Activation('relu'))

        #-- layer 5
        model.add(Conv2D(128, (3, 3), input_shape=input_shape, padding='same'))
        model.add(Activation('relu'))

        #-- layer 6
        model.add(Conv2D(128, (3, 3), input_shape=input_shape, padding='same'))
        model.add(Activation('relu'))
        model.add(AveragePooling2D(pool_size=(2, 2)))

        #-- layer 7
        model.add(Flatten())
        model.add(Dense(64))
        model.add(Dropout(0.5))
```

```
model.add(Activation('relu'))

#-- layer 8
model.add(Dense(64))
model.add(Dropout(0.5))
model.add(Activation('relu'))
model.add(Dense(1))
model.add(Activation('sigmoid'))


model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])

# define early stopping callback
earlystop = EarlyStopping(monitor='val_acc', min_delta=0.0001,
                          patience = patience_val, verbose = verbosity, mode='auto')
callbacks_list = [earlystop]
```

7 Model 7

In [ ]:
```
# model 7
epochs = min(global_epochs, 200)
batch_size = 16

''' define the CNN model '''
dropout = 0.2
model = Sequential()

#-- layer 1
model.add(Conv2D(64, (3, 3), input_shape=input_shape, padding='same'))
model.add(Dropout(dropout))
model.add(Activation('relu'))

#-- layer 2
model.add(Conv2D(64, (3, 3),  padding='same'))
model.add(Dropout(dropout))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(10, 2)))

##--layer 3
model.add(Conv2D(128, (3, 3),  padding='same'))
model.add(Dropout(dropout))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

##--layer 4
model.add(Conv2D(256, (3, 3),  padding='same'))
model.add(Dropout(dropout))
```

```
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

##-- layer 5
model.add(Flatten())
model.add(Dense(512))
model.add(Activation('relu'))

#-- layer 6
model.add(Flatten())
model.add(Dense(512))
model.add(Activation('relu'))

#-- layer 7
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dense(1))
model.add(Activation('sigmoid'))


sgd = SGD(lr=learningRate, decay = lr_weight_decay)
model.compile(loss='binary_crossentropy',optimizer='sgd',metrics=['accuracy'])

earlystop = EarlyStopping(monitor='val_acc', min_delta=0.0001,
                          patience = patience_val, verbose = verbosity, mode='auto')
callbacks_list = [earlystop]
```

8 Model 8

In [ ]:
```
# model 8
epochs = min(global_epochs, 200)
batch_size = 16

''' define the CNN model '''
dropout = 0.2
model = Sequential()


#-- layer 1
model.add(Conv2D(6, (3, 3), input_shape=input_shape, padding='same'))
model.add(Dropout(dropout))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(5, 1)))

##--layer 2
model.add(Conv2D(12, (3, 3),  padding='same'))
model.add(Dropout(dropout))
```

```
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

##--layer 3
model.add(Conv2D(24, (3, 3),  padding='same'))
model.add(Dropout(dropout))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

##--layer 4
model.add(Conv2D(48, (3, 3),  padding='same'))
model.add(Dropout(dropout))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

##-- layer 4
model.add(Flatten())
model.add(Dense(120))
model.add(Activation('relu'))

#-- layer 5
model.add(Flatten())
model.add(Dense(120))
model.add(Activation('relu'))

#-- layer 6
model.add(Dense(1))
model.add(Activation('sigmoid'))


model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])


# define early stopping callback
earlystop = EarlyStopping(monitor='val_acc', min_delta=0.0001, patience = patience_val,
                          verbose = verbosity, mode='auto')
callbacks_list = [earlystop]
```

9 Model 9

In [ ]:
```
# model 9
epochs = min(global_epochs, 200)
batch_size = 16

''' define the CNN model '''
```

```
dropout = 0.2
model = Sequential()

#-- layer 1
model.add(Conv2D(64, (3, 3), input_shape=input_shape, padding='same'))
model.add(Dropout(dropout))
model.add(Activation('relu'))

#-- layer 2
model.add(Conv2D(64, (3, 3),  padding='same'))
model.add(Dropout(dropout))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

##--layer 3
model.add(Conv2D(128, (3, 3),  padding='same'))
model.add(Dropout(dropout))
model.add(Activation('relu'))

#-- layer 4
model.add(Conv2D(128, (3, 3),  padding='same'))
model.add(Dropout(dropout))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))


##--layer 5
model.add(Conv2D(256, (3, 3),  padding='same'))
model.add(Dropout(dropout))
model.add(Activation('relu'))

#-- layer 6
model.add(Conv2D(256, (3, 3),  padding='same'))
model.add(Dropout(dropout))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

##--layer 7
model.add(Conv2D(512, (3, 3),  padding='same'))
model.add(Dropout(dropout))
model.add(Activation('relu'))

#-- layer 8
model.add(Conv2D(512, (3, 3),  padding='same'))
model.add(Dropout(dropout))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
```

```
##-- layer 6
model.add(Flatten())
model.add(Dense(512))
model.add(Activation('relu'))

#-- layer 7
model.add(Flatten())
model.add(Dense(4096))
model.add(Activation('relu'))

#-- layer 8
model.add(Dense(4096))
model.add(Activation('relu'))
model.add(Dense(1))
model.add(Activation('sigmoid'))


sgd = SGD(lr=learningRate, decay = lr_weight_decay)
model.compile(loss='binary_crossentropy',optimizer='sgd',metrics=['accuracy'])

earlystop = EarlyStopping(monitor='val_acc', min_delta=0.0001,
                          patience = patience_val, verbose = verbosity, mode='auto')
callbacks_list = [earlystop]

# train the model
with tf.Session() as sess:
    keras.backend.get_session().run(tf.global_variables_initializer())

    history = model.fit(X_train_norm_conv, Y_train, batch_size=32,
                        nb_epoch=epochs, callbacks=callbacks_list,
                        validation_split=0.2,
                        validation_data=(X_test_norm_conv, Y_test))

    plot_history(history)

    save_nn_model(style_todo,model_number)
```

10 Model 10

In [ ]:
```
# Model 10
epochs = min(global_epochs, 200)
batch_size = 16

#-- layer 1
model = Sequential()
model.add(Conv2D(32, (5, 5), input_shape=input_shape))
```

```
model.add(Dropout(0.2))
model.add(Activation('relu'))
model.add(AveragePooling2D(pool_size=(2, 2)))

#-- layer 2
model.add(Conv2D(64, (3, 3)))
model.add(Dropout(0.2))
model.add(Activation('relu'))
model.add(AveragePooling2D(pool_size=(2, 2)))

#-- layer 3
model.add(Conv2D(128, (3, 3)))
model.add(Dropout(0.2))
model.add(Activation('relu'))
model.add(AveragePooling2D(pool_size=(2, 2)))

#-- layer 4
model.add(Flatten())
model.add(Dense(64))
model.add(Dropout(0.5))
model.add(Activation('relu'))

#-- layer 5
model.add(Dense(64))
model.add(Dropout(0.5))
model.add(Activation('relu'))
model.add(Dense(1))
model.add(Activation('sigmoid'))

model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])

# define early stopping callback
earlystop = EarlyStopping(monitor='val_acc', min_delta=0.0001, patience = patience_val,
                          verbose = verbosity, mode='auto')
callbacks_list = [earlystop]
```

11 Model 11

In [ ]:
```
# Model 11
epochs = min(global_epochs, 200)
batch_size = 16

#-- layer 1
model = Sequential()
model.add(Conv2D(96, (3, 3), input_shape=input_shape))
model.add(Activation('relu'))
```

```
#-- layer 2
model.add(Conv2D(96, (3, 3)))
model.add(Activation('relu'))
model.add(AveragePooling2D(pool_size=(2, 1)))

#-- layer 3
model.add(Conv2D(96, (3, 3)))
model.add(Activation('relu'))

#-- layer 4
model.add(Conv2D(96, (3, 3)))
model.add(Activation('relu'))

#-- layer 5
model.add(Conv2D(96, (3, 3)))
model.add(Activation('relu'))
model.add(AveragePooling2D(pool_size=(2, 1)))

#-- layer 6
model.add(Conv2D(128, (3, 3)))
model.add(Activation('relu'))

#-- layer 7
model.add(Conv2D(128, (3, 3)))
model.add(Activation('relu'))

#-- layer 8
model.add(Conv2D(128, (3, 3)))
model.add(Activation('relu'))
model.add(AveragePooling2D(pool_size=(2, 1)))

#-- layer 9
model.add(Flatten())
model.add(Dense(64))
model.add(Dropout(0.1))
model.add(Activation('relu'))

#-- layer 10
model.add(Dense(64))
model.add(Dropout(0.1))
model.add(Activation('relu'))

#-- layer 11
model.add(Dense(64))
model.add(Dropout(0.1))
model.add(Activation('relu'))

model.add(Dense(1))
```

```
model.add(Activation('sigmoid'))


model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])

# define early stopping callback
earlystop = EarlyStopping(monitor='val_acc', min_delta=0.0001, patience = patience_val,
                          verbose = verbosity, mode='auto')
callbacks_list = [earlystop]
```