The University of Nottingham Malaysia Campus, School of Computer Science

Individual Undergraduate Dissertation Final Report

(COMP3024 / 18-19)

Computer-Assisted Pronunciation Validation and Goodness Measurement Through Deep Learning

April 18, 2019

Submitted by:
Anas Nader Almasri
18823789
khcy6ana@nottingham.edu.my

Supervised by:
Dr. Tomas H. Maul
Tomas.Maul@nottingham.edu.my

Contents

Preface

All praise is due to Allah Almighty, the Most Gracious, the Most Merciful, for bestowing upon me the chance of reaching this far. It is with great pleasure that I present the process and outcome of this project. This is just an attempt to put it in words, as experiences cannot be summarised or quantified. Great thanks, with the utmost truthfulness, to Dr. Tomas for his constant guidance and immeasurable patience with me throughout my journey in the University to begin with, and in this project. I could not have imagined to even get near the end of it in such style if it weren't for his care for the project and the enthusiasm he induced in me. Finest gratitude to family and friends who stood beside me this whole time, taking a lot off my shoulders and continuously offering support — even when I least needed it. It is indeed saddening to see the end of this rollercoaster of knowledge, but moving on to *generalising* the knowledge gained (or *trained on*!) is part of life in the big picture.

Anas N. Al-Masri

Abstract

According to linguistics, words are broken into word segments called syllables, which are broken into the smallest parts of spellings — graphemes (i.e. individual letters). Based on the way syllables are uttered, and possibly the intonation involved, syllables can be submerged together or further broken into smaller segments of speech called phonemes. Recognising those segments is a task that Automatic Speech Recognition (ASR) research has shed the light on of decades. ASR has been prospering over the last few years, especially with the field having been introduced to Deep Learning. This project is a facilitation of Deep Learning in Computer-Assisted Language Learning (CALL), to help the learner of the English language fix erroneous parts in their pronunciations by giving them direct feedback about the pronunciation at hand as well as a similarity indicator on the Goodness of Pronunciation they present. This is an attempt to strengthen the link between CALL and Deep Learning, by providing a novel means of live feedback to students in a way that even covers parts of words (i.e. phonemes), not just the words themselves. The project combines a Deep Bidirectional Long Short-Term Memory Recurrent Neural Network with a mobile application that acts as the medium between the model and the user.

List of Figures

List of Abbreviations

ANN	Artificial Neural Network
API	Application Programming Interface
ASR	Automtic Speech Recognition
AWS	Amazon Web Services
BLSTM	Bidirectional LSTM RNN
BRNN	Bidirectional RNN
CALL	Computer-Assisted Language Learning
CNN	Convolutional Neural Network
CTC	Connectionist Temporal Classification
CV	Computer Vision
DCT	Discrete Cosine Transform
DL	Deep Learning
DNN	Deep Neural Network
DTW	Dynamic Time Warping
FFT	Fast Fourier Transform
GCP	Google Cloud Platform
GD	Gradient Descent
GMM	Gaussian Mixture Models
GoP	Goodness of Pronunciation
GPU	Graphics Processing Unit
GUI	Graphical User Interface
HCI	Human-Computer Interaction
HMM	Hidden Markov Models
L2	Second/Foregin Language
LPC	Linear Predictive Coding
LSTM	Long Short-Term Memory
MFCC	Mel-Frequency Cepstral Coefficients
ML	Machine Learning
MRF	Markov Random Field
NN	Neural Network
PCM	Pulse Code Modulation
PER	Phoneme Error Rate
RBM	Restricted Boltzmann Machine
RCNN	Recurrent Convolutional Neural Network
RNN	Recurrent Neural Network
STT	Speech-to-Text
SVM	Support Vector Machines
TTS	Text-to-Speech
UI	User Interface
VGP	Vanishing Gradient Problem

Chapter 1

Introduction

1.1 Aims and Objectives

This project aims to design and implement a mobile application that is capable of listening to the user and validating as well as identifying the erroneous parts in the user's pronunciation, making language acquisition easier and more productive. The desired method includes deploying needed mathematical approaches combined with a Deep Neural Network that produces reasonably accurate results.

One of the key objectives of this project is to investigate engineering and Machine Learning techniques that have previously been used to perform speech signal processing. Although most engineering techniques rely heavily on the mathematical perspective of applicability, it is undeniable that the Machine Learning approach can be superior — for reasons that will be discussed thoroughly in later chapters. This will be arrived to through exploring popular audio signal fitting mechanisms that have been examined by researchers, as a measure of preprocessing speech signals. Since a lot of those techniques involve numerous statistical and algebraic bases, there will be a brief touch on each of the topics involved. Moreover, this project also scrutinises different Automatic Speech Recognition (ASR) techniques, such as Phonemic Transcription. It also alludes to their accuracy rate differences, at least from the perspective of speech signal processing.

The tangible aspect of this project dictated the *attempt* to design and implement an algorithm for finding a similarity indicator between two pronunciations, and find the differences — if any — and present them in the form of phonemes. Even though this was problematic (as will be discussed later), it was an interesting exploratory process. The project also included the experimentation with Deep Learning before going further. The final tangible product includes a mobile application that utilises Deep Learning in making it easier for users to test their pronunciations and find their weaknesses in their language-learning journey.

1.2 Anticipated Outcomes

Language-learning is one of the aspects that are unavoidable in the practical life. This opens up doors for the individual as well as widens the angle of perspectives through which they look at the thing at hand. Learning a new language is not just about the words and the grammar involved, but also about the cultural variables running in accord with it. Different mindsets in people require different approaches of learning. Therefore, expanding the options of learning is a must, at this current stage of technological advancement. It is not

just a matter of learning anymore, but rather a matter of choosing the right medium to learn.

The purpose enriching this project is – in one sentence – *making the learning journey easier on the learner*. One of the major issues that second-language students encounter is the lack of feedback, especially when they are taking a subject as a self-taught one. Feedback is crucial in any form of learning. Researchers have spent substantial amounts of time trying to figure out and test new ways of building programs that present significantly more "intelligence". In such context, when it comes to self-learning, the student-teacher relationship is missing, which is something that is inarguably critical. However, self-learning is usually self-initiated, making students' enthusiasm and dedication likely to be stronger.

This project tries to taper the gap between the self-learning student and the *virtual* teacher, giving students a more personlised language-learning experience through live feedback on their pronunciation. It is built with major hope that this project betters students' relationships with learning materials — interactively.

1.3 Project Outline

The project includes building a Deep Neural Network (DNN) that transcribes speech segments into a list of phonemes (i.e. building blocks of syllables). This DNN is to be used in a mobile application that takes care of the user's input and gives them the feedback accordingly. Users can search for a word or a phrase by its spelling or choose to input by speaking to the application. The phrase is then uttered back to the user so that he/she can know the correct pronunciation of the particular phrase. Users can then pronounce it again and get a visual comparison between the actual pronunciation and their own.

1.4 Structure of The Report

Within the body of this report (Chapters 1 through 7), the most important parts of the system are explained and the theory behind them is discussed. However, aspects that need further explanation were enclosed in the appendices, as well as sections that do not contribute to the functional core of the project. Nevertheless, it is integral to acquiring a thorough understanding to have a look at the appendices. Throughout the report, the importance of some parts of the appendices is emphasised on within sections that directly incorporate details mentioned in those respective appendices.

Chapter 2

Background and Motivation

2.1 Computer-Assisted Language Learning

Computer-Assisted Language Learning (CALL) is a generic term that manifests the deployment of computers and respective technologies associated with them in the language-learning process. This encompasses issues of material design, technologies, pedagogical theories and modes of instruction. Materials for CALL can include those which are purpose-made for language learning and those which adapt existing computer-based materials, video and the such **[1].** CALL is a field of research and development that is constantly changing, making it challenging to follow and keep up with. This is attributed to the fact that CALL relies heavily on the technologies involved, which are real-world technologies in the first place. Therefore, CALL application can be thought of as a direct facilitation of existing tools and practices in language education.

According to Warschauer (1996), the history of CALL can be divided into three phases; structural, communicative and integrative **[2].** *Structural* (i.e. behavioristic) CALL emerged from research back in the 1950s and through the late 1970s. Being a field that is highly associated with kindred fields and practices, the influence of then-prosperous research areas (e.g. behaviorist theories of learning) on CALL is undeniable **[3].** This phase followed a questionnaire-like format that is presented repeatedly to students, for the effectiveness of systematic repetitions in learning **[4].** The introduction of microcomputers allowed a whole new range of possibilities and marked the end of Structural CALL **[3].** This is evident through examining major 1980s' CALL applications, like the then-sophisticated PLATO system **[5].**

Subsequently, the CALL phase that became prominent until the early 1990s was *Communicative* CALL. Thickening the link between CALL and technological advancements, CALL applications boomed at this stage — mainly due to the abundance of personal computers. This phase emerged through the belief that Structural CALL did not involve real communication. Therefore, Communicative CALL focused on improving the student's communication with the learning material rather than the material itself **[6].** In a lot of models, the computer was used as a tool — or a workhorse **[7].** Finally, the third phase (i.e. *Integrative* CALL) came into the light in the 1990s, and was based on multiple technological advances like multimedia (e.g. CD-ROM) and internet applications **[3].** This phase was fundamentally aimed at integrating computer-mediated communication applications for communicative language teaching.

2.1.1 Development of Learning Methods

Computer-Assisted Language Learning — having spanned multiple decades — has gone through a lot of changes when it comes to the way the concept is applied. In terms of learning, changing the means is expected to change the experience. Therefore, the improvements in learning journeys became more effective over time. In fact, major difficulties of CALL were attributed to deficiencies in computers' performance and capabilities **[8].** Meskill (2002) broached the subject by stating what computers cannot do, which includes:

- Judge unexpected input
- Provide individualised feedback beyond a predetermined list of messages
- Engage the learner in rich negotiation of meaning characteristic of face-to-face interaction
- Motivate depth and quality of engagement characteristic of human interaction

Examining the above leads to the realisation that those problems do not exist anymore. This is due to the industry having advanced on so many levels that the issues Meskill stated back in 2002 sound somewhat preposterous. Consequently, CALL has benefited a great deal from technological advances as well as being introduced to new fields (e.g. Machine Learning). The manifestation of such a direct link between fields results in a state of constant development — which is extremely noticeable in CALL.

2.1.2 Computer-Assisted Feedback Systems

CALL is by no means restricted to the students' interaction with the tools at hand (namely, computers, in this context). It has been extended to include teachers directly interacting with computers, mainly seeking computer-assisted assessment. Also called computer-aided marking, this sub-field was always inferior in the industry. However, it has been receiving more attention over the last two decades, to the point where it became a necessity to employ in world-class universities and other forms of education institutes. Denton (2003) **[9]** describes four different types of computer-assisted assessment; objective testing, electronic submission, free text analysis as well as marking assistance **[10].** Denton mentions that objective testing includes multiple-choice questions delivered via the Web. While the rest of the types are self-explanatory, plagiarism testing tools are an example of the free text analysis that Denton mentioned.

2.1.3 Feedback and Disfluency Detection

In Linguistics, feedback is the mechanism of determining whether a set of communication requirements can be met, leading to the continuation of contact between parties **[11].** This is what lets one conceptualise the actuality of progressing through their language-learning journey. Language learning feedback is a complex topic that research has been covering for decades. Arguably, the core of CALL feedback research is "the distinction between 'acquisition' (the gradual and implicit accumulation of L2 competence) and 'learning' (the conscious and explicit learning of L2 knowledge)" **[12]** — where L2

refers to second/foreign language. Due to the importance of corrective, constructive feedback in language learning, CALL research has always kept open ears for any advances in technology that can benefit this major aspect of language learning. In fact, discrete error analysis and feedback were a common feature of traditional CALL, and the more sophisticated programs would attempt to analyse the learner's response, pinpoint errors, and branch to help and remedial activities **[13].**

2.2 Automatic Speech Recognition

2.2.1 Sound Signals

Sound, the stimulus to auditory perception, is defined as "oscillation in pressure, stress, particle displacement, particle velocity, etc., propagated in a medium with internal forces, or the superposition of such propagated oscillation" **[14].** The significance of sound constitutes the role it plays in communication among creatures. This noteworthiness is evident in applications of most industries and fields. While sound signal properties have constantly varying values, especially with their high affectedness by surrounding noise-related oscillations, their nature, from the physical perspective never changes. *Figure2.1* illustrates how air molecule *A*, affected by the vibrations coming from the sound signal source on the left (in black), oscillates between positions *P1* and *P2*, going back to its *equilibrium position* shown in the figure as the initial state (dark gray). The maximum displacement of an air molecule from its equilibrium position, be it caused by desired sound or background noise, is called the *amplitude* (*Figure2.2, A*). The time it takes for an air molecule to fully oscillate back and forth one time is referred to as the *period* of a sound wave (*Figure2.2, B*). The smaller the value of collective periods of sound waves, the higher the note (i.e. pitch) that we perceive. In constant surrounding environmental factors, increasing the volume leads to the oscillations becoming larger. From the representational perspective, every sound wave comprises of *wave cycles* — pairs of a maximum value and a corresponding minimum value. The number of wave cycles that occur in one second is referred to as the *frequency* of the wave, and is measured in *Hertz (Hz)*. For healthy humans, it is usually possible to hear sound frequencies ranging from 20Hz and up to 20kHz. Finally, during oscillations of neighbouring air molecules, causing sound to travel through a medium of air, those molecules will be compressed close together in some

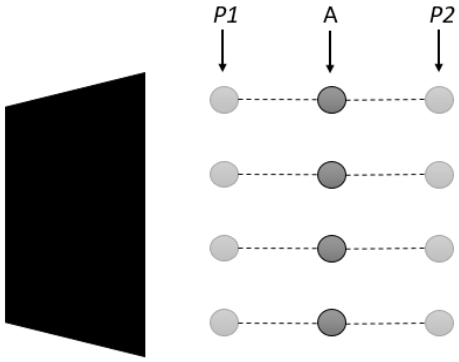


Figure 2.1: Molecule oscillation near a sound source

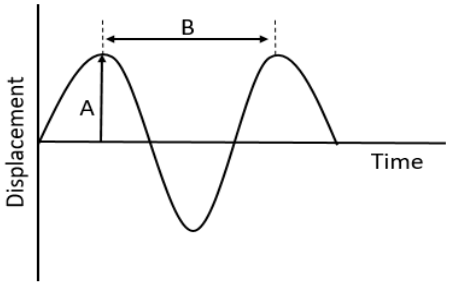


Figure 2.2: Displacement of an air molecule over time

regions and spread far apart from each other in other regions. The distance between two compressed regions is the *wavelength (λ)* of that sound, and is measured in meters.

2.2.2 Speech Signals

Speech is a complex form of sound waves. Its properties are both *periodic* (vowels) and ranging from periodic to *aperiodic* (consonants). In phonetics, there are speech features that accompany vowels and consonants, referred to as *suprasegmental features* (e.g. loudness, pitch, secondary articulation, etc.), which can span multiple syllables or words. Speech signals — being sound signals — can be represented in three main forms; the frequency spectrum, the waveform view, and the spectrogram. The frequency spectrum is used only when temporal information visualisation techniques are needed. In practice, choosing which representational form to use is highly dependent on the particular application. The *waveform view* (*Figure2.3*), for instance, displays the amplitude information over time, providing a general view of the sound wave. As visible in the figure, voicing periods are easily identified through the level of complexity they present. In the waveform view of an utterance (i.e. voicing), small portions of silence signal some sort of closure during articulation, usually occurring before plosive consonants (i.e. p, t, k, b, d, and g). On the other hand, longer portions of silence on a waveform view embody silence between words. Nonetheless, the most famous technique for acoustic analysis visualisation is the *spectrogram*, shown in *Figure2.4*. Spectrograms display the exact form and structure of speech sounds, representing time and frequency changes along the axes and depicting the amplitude with colors. Spectrograms of sound signals are most commonly used in acoustic phonetics, since they enable us to analyse the voicing periods in a more precise way. Interestingly enough, both *Figure2.3* and *Figure2.4* represent the same sentence — "No signal". Regardless of which representation technique is most suitable for the application, spoken utterances are never produced with the same intensity, even by the same person. This makes it difficult to analyse sounds based only on the oscillation representations, especially since other voice characteristics (e.g. pitch) are changing continuously, even through the same speech segment — referred to as a *syllable*.

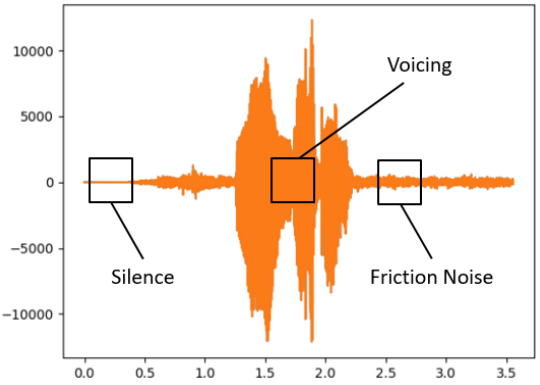


Figure 2.3: Waveform view representation of a speech signal

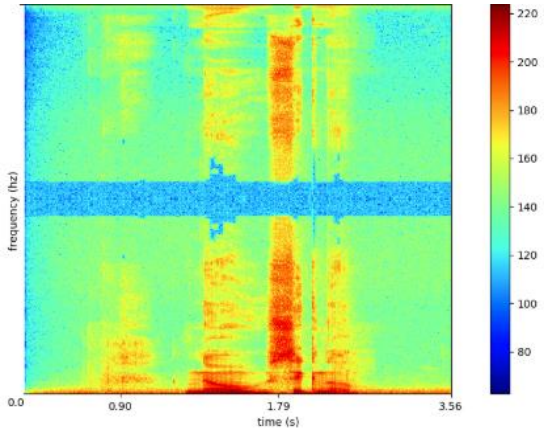


Figure 2.4: Spectrogram representation of a speech signal

2.2.3 Audio Signal Processing

Audio Signal Processing is an engineering field that focuses on the computation of methods for intentional alteration of sounds. Audio signals (i.e. sounds) can be electronically represented in a digital or an analog format (shown in *Figure2.5*). Analog processors operate directly on electrical signals, while digital processors operate mathematically on the binary representation (shown in *Figure2.6*) of signals. No matter which recording process used, analog or digital, both are created by a microphone turning air pressure (sound) into an electrical analog signal. Sound itself is a continuous wave; it is an analog signal. This means that one cannot detect the precise moment the pitch changes. Capturing this continuous wave in its entirety requires an analog recording system; what the microphone receives is exactly what's written onto the vinyl disk or cassette. Analog is believed to be the true representation of the sound at the moment it was recorded.

Figure 2.5: Analog/ real signal

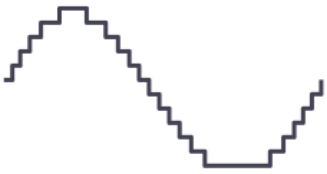


Figure 2.6: Digital signal

Digital sound is not a recording of the actual sound, but rather a combination of binary code, the utmost simplest machine language of zeros and ones, representing the sound's intensity and pitch at precise intervals with relative accuracy. The binary code is arranged in a specific pattern informing the computer how to recreate the sound itself. It is not a single wave the way analog sound is, but rather a composite of multiple segments representing consecutive moments of intensity and pitch. Where an analog recording is similar to the fluency of film, a digital recording is *stop motion* photography.

2.3 Technical Motivation

Comparison between two audio signals through geometric metrics is a highly exhaustive process. For example, Euclidean distance between two waveform representations would not work globally, simply because it does not account for the time frame associated with the voicing period, making it close to impossible to segment phonemes from a signal. Moreover, and since linear measures of similarity would not work in speech recognition applications (for speech cannot be linearly stretched to align it with another signal), it would be more effective to extract relevant features from both audio signals and compare them, rather than compare the original signals. This is expected to lead to the proficient identification of components of the audio signal that are enough to analyse the linguistic content. This comparison of audio signals usually starts with *Mel-Frequency Cepstral Coefficients (MFCC)* feature extraction, which is a representation of the short-term power spectrum of sound **[7].** Such process makes the result invariant to pitch and amplitude changes. MFCC in its core is based on the linear cosine transform of a log power spectrum on a nonlinear mel scale of frequency.

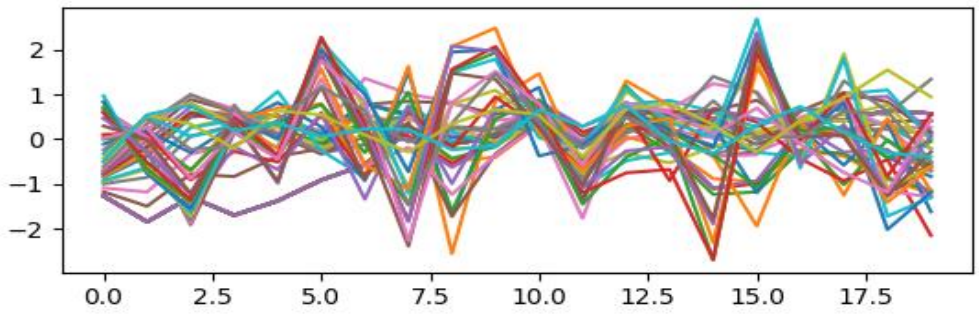


Figure 2.7: 39-dimesional MFCC feature vector of a speech signal

As speech similarity measurement does not stop at extracting MFCC features (shown in *Figure2.7*), the audio signals need to be aligned with one another before making a conclusive statement on similarity. Since some speech sounds can be stretched (e.g. *"caaar"*), extending the duration of — or even shortening — a word (e.g. *"car"*) before comparing it with another is a crucial step. This is where lies the need for dynamic alignment, namely, *Dynamic Time Warping (DTW)*. With DTW, we consider one of the voice recordings to be a template and match the other one with it by aligning the feature vectors of both. Calculating the similarity — or difference — between both would then be a matter of finding the angle between feature vectors of both audio signals. It is worth mentioning that alignment can also be achieved through statistical cross-correlation or convolution, but in an inaccurate manner, especially if the signals are not vastly similar. In such cases, cross-correlation would still give a numerical value, but it would be very difficult to scale.

Since MFCC reduces the frequency information of the speech signal into a small number of coefficients, it could be beneficial to facilitate such technique and combine it with a *Linear Predictive Coding (LPC)* method to extract features from speech. This is done to segment the voice recording into phonemes that can be further analysed. A DNN that is trained with MFCC and LPC features, respectively, would suffice for this stage. It is also worth mentioning that calculating the similarity percentage through a deep learning algorithm would limit the accuracy of comparison depending on the amount of data in the data set. By finding the mismatch between features as well as using the *Artificial Neural Network (ANN)* mentioned, we can determine what phoneme was erroneous in the user's pronunciation. However, mixtures of statistical and probabilistic models, such as *Hidden Markov Models (HMMs)* and *Gaussian Mixture Models (GMMs)* have been proved superior to the hybrid LPC-DNN approach in *phonemic transcription* — extracting phonemes from speech. This is merely because speech signals have both linear and nonlinear characteristics. A hybrid HMM-GMM approach, for example, could simply maximise the performance by handling both types of characteristics.

Over the last few years, other approaches have been closely examined by researchers. These approaches focused mainly on DNNs and their superiority in phonemic transcription. This topic was actually interesting enough for major names in *Machine Learning (ML)* and deep learning to delve into. Geoffrey Hinton et al. in *DNNs for Acoustic*

Modeling in Speech Recognition **[8]** showcased a thorough comparison between the ever-famous HMM-GMM hybrid approach, and applications that involve DNNs. This led major teams of researchers to invest their time into improving current Speech Recognition applications, especially with the rise of *Recurrent Neural Networks (RNNs)*.

Computer-Assisted Language Learning has benefited from recent advances in Machine Learning (discussed in the next chapter), leaving automated language-learning feedback systems more reliable than ever. However, those applications lack the ability to generalise to more than a fixed sets of words (i.e. data set involved). This is typically due to the difficulty of processing speech signals as sound signals and differentiating between utterance signals and background noise.

Chapter 3

Modelling ASR Tasks

In this chapter, the most commonly used ML-based approaches in Automatic Speech Recognition are discussed. This is merely to give an overall idea about how different approaches led to more combinations of ML concepts being utilised in applications.

3.1 Hidden Markov Models

Hybrid HMM-GMM models are most famous in applications when it comes to phoneme extraction in speech recognition research, where HMMs are dealt with as phoneme-level statistical models. The success of which lies in the fact that Hidden Markov Models normalise the temporal variability whereas Gaussian Mixture Models compute the emission probabilities of the HMM states **[19].** Another approach that received a lot of attention was the hybrid HMM-DNN, in which Deep Neural Networks are used to classify speech frames into clustered Context-Dependent states, and whose training still relied on GMMs to obtain initial frame-level labels **[20].** The fact that most HMM-based applications, especially Text-to-Speech (TTS), involve considerable sophistication is attributed to the ineffectiveness of simple, straightforward HMM-base ASR applications in terms of accuracy and sensitivity to changes in operating environment **[21].** Regardless of the method combined with HMMs, and more often that never, those models involved MFCC feature extraction before progressing the speech signal frames to be transcribed into phonemes or syllables.

3.2 Support Vector Machines

Even though kernelised Support Vector Machines (SVMs) (i.e. utilising the Kernel Trick) are non-parametric, they always perform quite well in ASR applications. This is mostly due to the fact that SVMs are very well-adapted to high-dimensional classification problems **[22].** The idea of SVMs as a learning algorithm is to find a hyperplane that best separates classes with a maximum margin. With less hyperparameters than Deep Neural Networks, Margin Maximisation in SVMs allows the SVMs to outperform most nonlinear classifiers in the presence of noise, which is one of the longstanding problems in ASR **[24].** Moreover, SVMs' convergence to the minimum of the associated cost function is guaranteed as a simple problem of quadratic programming **[22].**

3.3 Convolutional Neural Networks

Research has also paid its dues to the fact that Convolutional Neural Networks (CNNs) are essentially most effective in applications where inputs have spatial features that can be convolved or downsampled into a more representational form suitable for classification **[25][26].** Over the years, and despite some positive results, it has long been argued that

CNNs overkill the variation along time-scale by pooling within a temporal window, resulting in deep fully-connected neural network dominance in modeling time variation **[27].** However, in recent years, a newfound interest in ASR using CNNs has emerged through research. The motivation to use CNNs is inspired by the recent successes of the framework in many Computer Vision applications, where the input to the network is typically a two-dimensional matrix with very strong local correlations **[23][28].** Speech Recognition made use of this by utilising information depicted in a spectrogram. Such applications usually involved the spectrogram being fed into the CNN as an image, making the phoneme recognition process invariant to some characteristics like voice pitch and speaker's gender. Although the effectiveness of Convolutional Neural Networks in Image Processing is immeasurable, they did not exhibit significant improvement over other models in Speech Processing tasks **[26].**

3.4 Recurrent Neural Networks

Over the last few years, developments in Deep Learning have made it possible for RNNs to be able to handle recognition and decoding simultaneously **[28].** However, RNNs are generally hard to train because they cannot take full advantage of current highly optimised parallel computing facilities such as GPU **[29].** Nevertheless, RNNs have led to significant advances in speech signal processing. This is merely because speech signals are a form of sequential data in the first place, and because of their nonlinear nature. The recent advances in RNNs are attributed to the fact that they solve the problem of other types of DNNs being time-independent. However, RNNs lack the ability to utilise long time-dependence in sequential data, due to their *Vanishing Gradient Problem (VGP)*, making it difficult to come to the conclusion that a particular prediction was influenced by an early feature in the sequence. Another drawback in conventional *(Unidirectional)* RNNs is that they only take into account information in the sequence prior to the current input. It is worth emphasising on that phonemic transcription from a sequence of frames is asynchronous — phonemes usually span multiple frames whose boundaries are not visible in advance. Therefore, a solution to this asynchronicity of sequential data was formulated in Recurrent Neural Networks, by introducing a *delay* between features and labels in a simplistic manner. This makes the RNN take longer than the fixed time-windows, by spanning a few parts of the input's *future* **[30].**

3.5 Deep Neural Networks and Loss Functions

Although researchers arrived to many ways to improve the performance of DNNs for acoustic modelling, the problem of such models still remained — none of them truly achieved the end-to-end approach. However, this changed after the combination of RNNs with *Connectionist Temporal Classification (CTC)* was introduced, replacing Hidden Markov Models in phonemic transcription tasks. CTC is a training method for RNNs, which decodes the output probability distribution into phoneme sequences without requiring pre-segmented training data **[32].** This combination, which is relatively similar to Dynamic Time Warping (DTW) but very different in applicability, can help the DNN model benefit

from a variety of pronunciations without the need for a comprehensive data set of all possible syllable errors. Further research led to an improvement on the CTC algorithm — *Gram-CTC*, which enables the DNN model to capture longer-term dependency, and solves the problem of having to determine the optimal fixed set of basic units for the output beforehand **[31].**

3.6 Combinations of Deep Neural Network Concepts

Recent advances in research have extensively demonstrated that combining multiple DNN approaches into one model can be highly beneficial. For example, phonemic transcription has been achieved through *Recurrent Convolutional Neural Networks (RCNNs)*, making it possible to deal with temporal and frequency dependence in the spectrogram of the speech signal **[33].** Moreover, *skip-connection*-based approaches, such as *Residual Neural Networks (ResNets)*, have also proved effective in achieving high accuracies when the training data set was large enough. *Long Short-Term Memory RNNs* (shortly, *LSTMs*) have also improved the accuracy of transcription compared to conventional RNNs when trained with the same hyperparameters. This is simply because the effectiveness of LSTMs comes from the learning bias encoded in its architecture, rather than the way it is trained **[30].** However, the type of RNNs that has been receiving much attention in the past few years in Speech Recognition research is *Bidirectional LSTM RNNs (BLSTMs)*. The superiority of which over Unidirectional RNNs comes from the fact that it makes use of previous and future contexts in the sequence — exactly what conventional RNNs lack. The main shortcoming from such approach is the expensiveness of computational power required for training, as well as the need for more memory space for storing multiple gating neural responses at each time-step **[34].**

3.7 Restricted Boltzmann Machines

A Restricted Boltzmann Machine (RBM) is a particular type of Markov Random Field (MRF) that has one layer of stochastic visible units and one layer of stochastic hidden units. There are no visible-visible or hidden-hidden connections but all visible units typically have connections to all hidden units **[35].** RBMs have proved very effective at capturing sequences in acoustic modelling applications. In fact, a lot of the RBM-based research that used the same data set used in this project led to comparable results with Hidden Markov Models for similar applications. Some related previous works in the context of Automatic Speech Recognition have also focused their attention and experiments on considering using a hidden layer as well as prior knowledge in a Prior-Informed RBM system **[36].** This yielded very high accuracies — but only for limited, structured data sets.

Chapter 4

Project Specification

This project is divided into three main segments; a deep learning model, a mobile application and a server-side processing component. These segments are intertwined through a lot of underlying connections. In this chapter, these components are explained in brief, mostly from the project managerial point of view. This is an attempt to clarify the big picture before delving into these system components in real depth (in *Chapter 5*).

4.1 Project Workflow

The project outline can be broken down into two different groups of stages; research and implementation. These two groups were highly convolved together at times, and fairly separated at others. The workflow is demonstrated in *Figure3.1*.

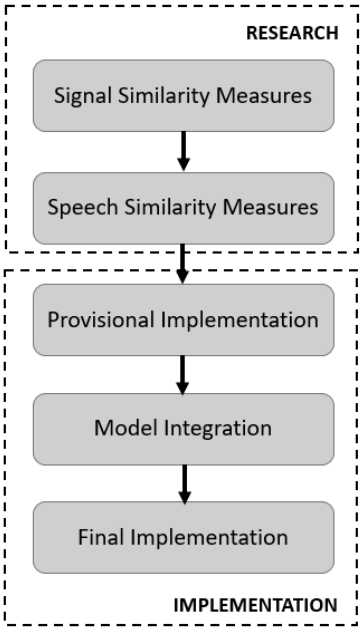


Figure 4.1: Project workflow

4.1.1 Signal Similarity Measures

The first and foremost stage included scrutinising existing signal similarity measures. This was in alignment with the original proposal in which a similarity measure between two sounds was to be found without the need for a data set involvement. However, and due to the lack of tangible results in such a subfield, the plan was tweaked and non-ML-based approaches were neglected.

4.1.2 Speech Similarity Measures

This stage set the beginning of a major step forward in the project outline. It included going through numerous ML approaches (SVMs, HMMs, and the such) trying to figure out the most feasible – and promising – approach to follow. This eventually led to the decision of utilising Deep Learning (as will be disclosed in *Chapter 5*). Although this stage was likely to be highly involved in the implementation, it was expected not to have a clear start and end.

4.1.3 Provisional Implementation

This stage included implementing both the Deep Learning model as well as the mobile application. Although it happened over two different substages, implementation was expected to be temporary, depending on the results obtained along the way.

4.1.4 Model Integration

Having implemented the Deep Learning model as well as the mobile application, this stage is a means to build the link between the two components. Here is where the server-side involvement happens. Although it seems fairly straight-forward, this stage was one of the most problematic ones, as will be explained in *Chapter 5*.

4.1.5 Final Implementation

During this stage, the results that model integration leads to as well as the preset project outline are brought together into becoming the final product. There is a lot of detail involved in the arrival to the results of this stage, which is examined thoroughly in *Chapter 5*.

4.2 Model Expectations

Originally, there were certain expectations for the Deep Learning model regarding inputs, outputs as well as effectiveness. Firstly, the model was to take speech signals as inputs. Although the form of which was still vague, it just made sense to build a model that transcribes a speech signal into the phonemes that the utterance involved comprises of. This means that the output of the model will include the phonemes present in the corresponding data set. Since the English language was the focus here, the output of the model was to be a matrix of P-by-N, where P is the number of phonemes in the language (i.e. English) and N is the number of time frames in the speech recording input. The know-hows of the interiors of the model were still not thoroughly examined, as this was a lengthy process invloving multiple underlying research phases.

4.3 Mobile App Specification

Users are to interact with the mobile application, which utilises the DL model in the back-end. Although the mobile aplication was only fully designed near the beginning of the implementation stage, there were certain functional requirements in mind that the mobile app needed to take care of. Since *Human-Computer Interaction (HCI)* is a major factor in the usability of mobile applications and accessibility of key features, this was to be taken into account as well.

Furthermore, the mobile application was to have a pronunciation assistant (named *Noon*), that appears on every *User Interface (UI)* screen. Noon is the party that does the pronunciation in the mobile application (the pronunciation that users compare theirs against). From the more technical perspective, the application was expected to deal with multiple *Application Programming Interfaces (APIs)* server-side due to the need for searching, *Text-to-Speech (TTS), Speech-to-Text (STT)* functionalities and the such. *Figure4.2* demonstrates the Use Cases whose back-end functional components are expected to be resident in the mobile application.

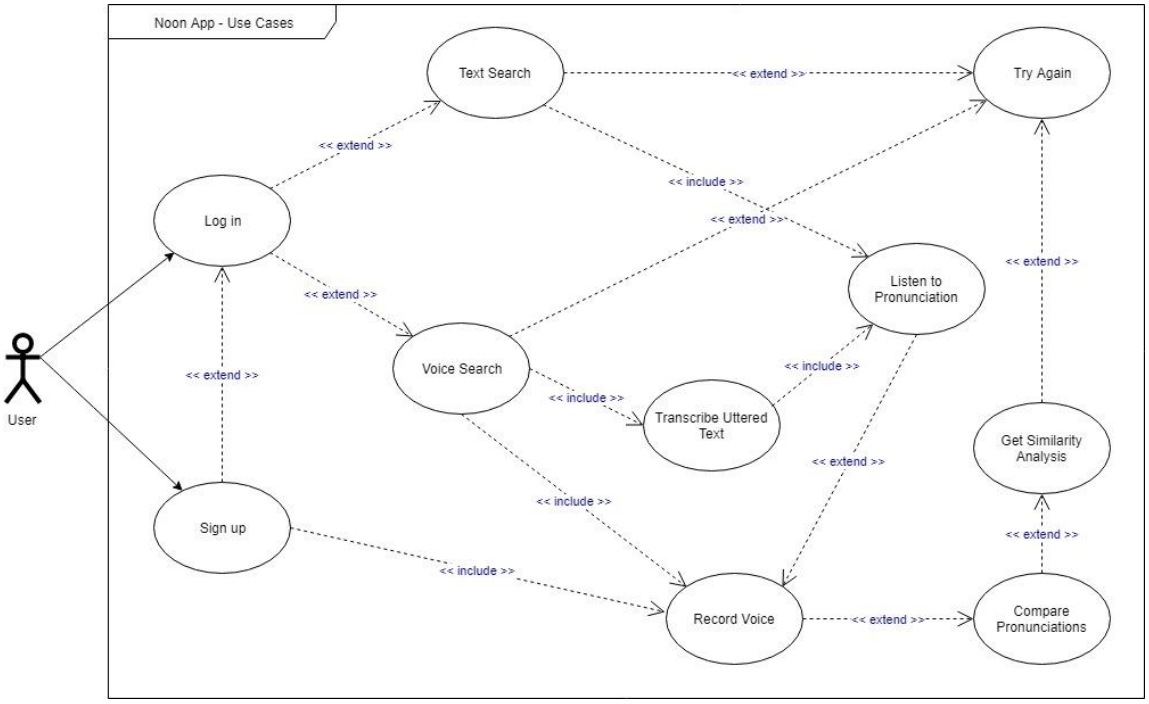


Figure 4.2: Mobile App (Noon) Use Case Diagram

Chapter 5

Methodology and Implementation

In this chapter, the project pipeline is examined thoroughly. The methods behind major components of the system are discussed and the tools used for every stage are disclosed. It is worth mentioning that subsequent to the explanation of parts of every major component is a *Technical Specification* sub-section that is rather an implementation-based clarification in regards to the tools used for the methodologies followed.

5.1 Process Overview

The focus of this section is mostly on the data flow among components rather than in-depth descriptions of the particular parts, as those will be discussed extensively in later sections of the chapter. This general overview (shown in *Figure5.1*) is divided into five main stages of execution, where data flows in three different path sequences depending on the operation; training, validation/testing and inference. These three sequences share nine paths (labelled A, .. I) through which data can flow. The following sub-sections describe those sequences.

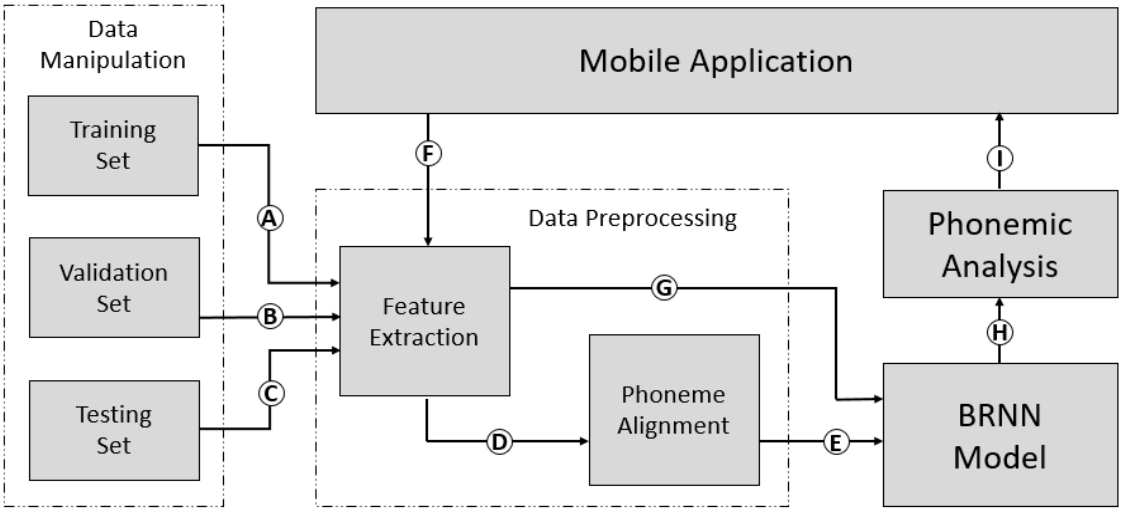


Figure 5.1: Data flow through project components

5.1.1 Training Flow Sequence

The first stage that the system went through was training the model. However, as the case is with most Deep Learning models, preprocessing was performed first. During this stage, Training data flowed through path *A* in the form of voice recordings alongside phoneme lists (playing the role of labels), into the Feature Extraction component. Feature vectors along the labels were passed through path *D*, where those features were aligned with their corresponding phonemes. Finally, both packets went through path *E* and into

the Model using which the *Bidirectional Recurrent Neural Network (BRNN)* was trained. The training data flow through the pipeline stops here.

5.1.2 Validation/Testing Flow Sequence

In order for the model to be trained properly, it is desirable for it to be periodically evaluated, and that is exactly where the need for *validation* data comes into play. Through calculating the error rate the model yields on the validation set at any given point, we can know how accurate it is. This is the essence of training. Subsequently, the model will tune its parameters based on the frequent evaluation results on the validation set. Furthermore, the final evaluation stage (namely, testing) includes *test* data being passed into the Model. For validation and test data, the flow goes through paths *B* and *C*, respectively. After feature vectors are extracted, there is no need for data to pass through the Phoneme Alignment component, since this data is not labelled — as far as the Model is concerned. Therefore, data will flow through path *G*, straight into the Model. Again, flow sequence will stop here, as the analysis component is related to the Mobile Application.

5.1.3 Inference Flow Sequence

Typically, inference data flows from the Mobile Application as a voice recording straight into the Feature Extraction part. Feature vectors are passed thorugh path *G* and into the BRNN to infer the output labels for such a sequence of inputs. The labels are then passed onto the Phonemic Analysis component (following path *H*), which returns – through *I* - the similarity percentage between both sequences of phonemes. It also returns the actual phonemes as infered by the BRNN Model component. Recently acquired data will then be transformed and shown is a presentable form onto the mobile application's interface. It is worth mentioning that there were implementation issues faced that directly affect this typical data flow. These issues are explained in section *5.5*.

5.2 Data Acquisition

Measuring application performance based on how generalisable the model is necessitates the search for a data set whose data is as diverse as possible. This search arrived at the *DARPA TIMIT Acoustic-Phonetic Continuous Speech Corpus (TIMIT),* which comprises of utterances of 10 different sentences by 630 speakers **[40],** resulting in a collection of 6300 sentences in about 5.4 hours of voice recordings, all sampled at 16kHz. This was a joint effort among Defense Advanced Research Projects Agency (DARPA), Massachusetts Institute of Technology (MIT), Stanford Research Institute (SRI), and Texas Instruments (TI). It is worth mentioning that this data set was chosen over data sets like *Kaldi* and *CMUSphinx* due to the wide fame it receives from researchers. On the other hand, said alternatives proved superior when it comes to the vast applicability in production-level systems as well as interoperability support. This project does not require such cross-platform functionality, thus, it follows the steps of related research.

5.2.1 Diversity of Speakers

The TIMIT data set was built for the development and evaluation of of Automatic Speech Recognition systems **[40].** Therefore, collaborators tried to cover as many dialects as possible (8 major dialects in American English). The dialect-gender distribution is as follows:

Table 5.1: TIMIT data set dialect breakdown

Dialect Region	Male	Female	Total
New England	31 (63%)	18 (27%)	49 (8%)
Northern	71 (70%)	31 (30%)	102 (16%)
North Midland	79 (67%)	23 (23%)	102 (16%)
South Midland	69 (69%)	31 (31%)	100 (16%)
Southern	62 (63%)	36 (37%)	98 (16%)
New York City	30 (65%)	16 (35%)	46 (7%)
Western	74 (74%)	26 (26%)	100 (16%)
Military Brat	22 (67%)	11 (33%)	33 (5%)
Total	**438 (70%)**	**192 (30%)**	**630 (100%)**

5.2.2 Distribution of Spoken Material

As the selection criteria attempted to maximise the variety of allophonic contexts found in the texts **[41],** sentences given to speakers were designed to cover three different aspects; dialectal variance, phonetic diversity and phonetic compactness. *Dialect* sentences, which were meant to highlight dialectal variants in speech, were read by all 630 speakers. *Phonetically-diverse* sentences were selected to add diversity in sentence types and phonetic contexts, while *phonetically-compact* sentences were designed to provide a good coverage of pairs of phones with extra occurrences of phonetic contexts thought to be either difficult or of particular interest.

Table 5.2: TIMIT data set sentence types

Sentence Type	Sentences	Speakers	Total	Sentences per speaker
Dialect	2	630	1260	2
Compact	450	7	3150	5
Diverse	1890	1	1890	3
Total	**2342**		**6300**	**10**

5.2.3 Data Format

Figure5.2 shows how parsing through the TIMIT data set works. The data set provides 4 different files for every utterance by any speaker. Each file is more useful for

some applications rather than others. The inclusion of different types was a measure of thoroughness and maximised usefulness **[41].** These data formats are as follows:

.wav SPHERE-headered speech waveform file, downsampled to 16kHz.

.txt Associated orthographic transcription of the words the person said.

.wrd Time-aligned word transcription.

.phn Time-aligned phonetic transcription.

The focus in this project was on the PHN and WAV files because of the data form they provide. It is worth mentioning that because of the SPHERE header in WAV files, an extra preprocessing step was needed.

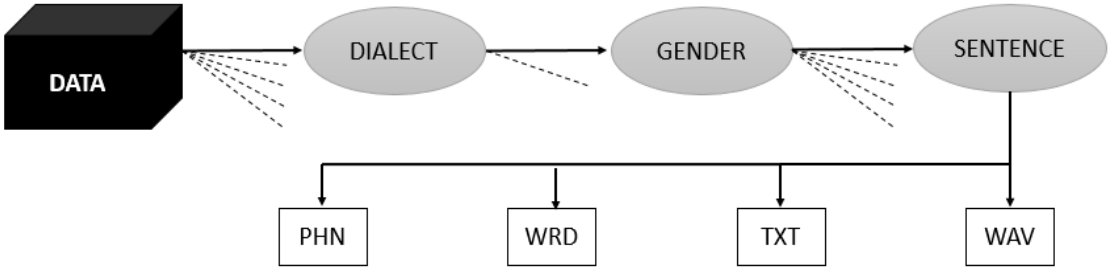


Figure 5.2: TIMIT data set breakdown

5.3 Data Preprocessing

5.3.1 MFCC Feature Extraction

Identifying the linguistic content in a speech signal can only be reliable when certain features are extracted from the signal, discarding those that do not contribute to the generalisability of the application's model. For instance, features that contain coefficients relating to the emotions of the speaker should be excluded for the purpose of this project. *Mel-Frequency Cepstral Coefficient (MFCC)* features are the most commonly used when it comes to Speech Recognition applications. In a form of reverse-engineering sound production, the efficiency of this technique is attributed to the fact that MFCCs represent an envelope of the short time power spectrum which describes the shape of the vocal tract of a human being's — exactly what determines the sound coming out **[37].** This feature extraction method has been extensively examined and improved by researchers over the years, resulting in it being a state-of-the-art approach. Although MFCCs approximate the human system response more than any other descriptors, the robustness of these values is only reliable to an extent, as the effectiveness of which deteriorates with additive noise **[38].** The most important steps of MFCC feature extraction are examined in the following subsections. *Figure5.3* demonstrates the steps that MFCC extraction encompasses.

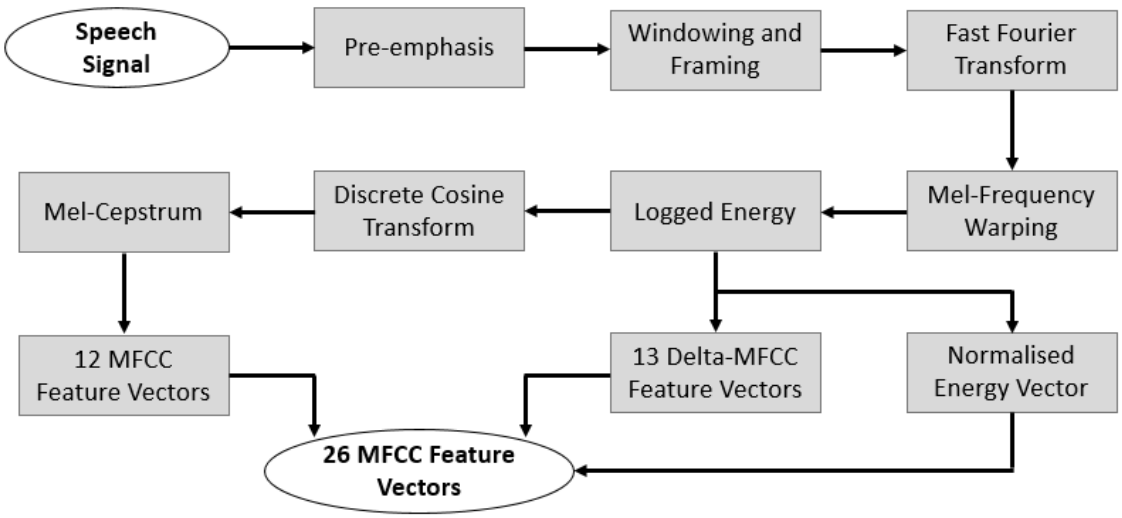


Figure 5.3: MFCC feature extraction pipeline

5.3.1.1 Pre-emphasis

Pre-emphasis, or the amplification of high frequency regions in a signal, might not have an integral effect on performance in modern systems. However, it is still used as part of the feature extraction process, as it was noted that higher frequencies are more important for signal disambiguation than lower frequencies **[53].** This is supported by the fact that feature extraction as a process, in essence, implies that the properties of the system are being viewed as emphasising certain aspects of the signal while reducing or removing others.

5.3.1.2 Framing and Windowing

In its nature, human speech is slowly varying over significantly small periods of time. Therefore, *framing* the speech signal is a key step in the process, forcing spectral analysis to be applied on duration blocks rather than the full signal. This is done through dividing the speech signal into small time-based slices (i.e. frames) — hence the name, framing. By convention, these duration blocks usually span 20 ms each. Pragmatically, framing introduces amplified levels of discontinuity in speech samples. This is remedied through *windowing*, where each frame is multiplied by a window function, attenuating the values of the samples at the beginning and end of each frame. The *Hamming Window* is commonly used here. It decreases the frequency resolution of the spectral analysis while reducing the sidelobe level of the window transfer function **[39].**

5.3.1.3 Fast Fourier Transform

In general, the core of signal processing lies at the extent to which *Fourier* analysis dictates its effectiveness. Such signal processing revolves around the idea that any periodically-repeating waveform can be expressed as a sum of sinusoids, each scaled and shifted in time by appropriate constants. This means that signal *W0* in the *Figure5.4* is composed of multiple signals (namely, *W1* through *W5*) that are summed up into one frequency function. Simplistically, the signal depicted by waveform *W0* in the figure is actually the sum of the air pressure difference of all the waveforms composing *W0* at any

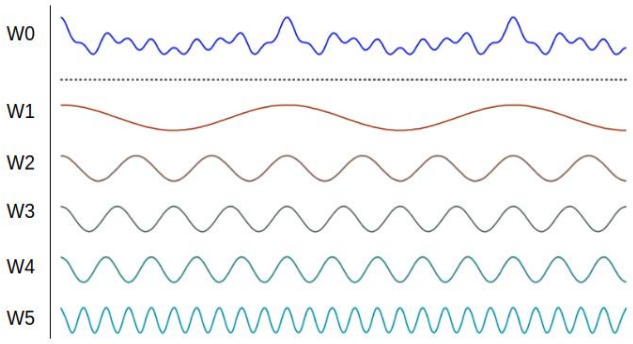


Figure 5.4: Basic representation of a signal decomposition process

given time on the time-intensity graph. Building on this idea, a microphone records sounds by picking up on the air pressure at many points in time. A *Fourier Transform* plays the role of the *link* between a signal in the time domain and the same signal in the frequency domain. In this context, this refers to a form of reverse-engineering the signal picked up by the microphone, and decomposing it into the waveforms whose sum results in the particular signal at hand.

5.3.1.4 Mel Filter Bank Processing

In the *Fast Fourier Transform (FFT)* spectrum, voice signals do not follow a linear scale. This is associated with the presence of various frequencies in such spectrums. Huang et al. explain how this could be remedied by using *Mel Filters* — a set of triangular filters that are used in computing a weighted sum of spectral components, simulating the characteristics of the human ear, and resulting in a non-linear mapping of the audible frequency range **[42].**

5.3.1.5 Discrete Cosine Transform

An *Acoustic Vector* is a set of MFCCs. Each of these coefficients is a descriptor of the log Mel Filter Bank associated with it. This is usually calculated through a *Discrete Cosine Transform (DCT)* that plays the role of the converter of a log Mel spectrum into the time domain. Obtained coefficients in the time domain are of different orders. This is where the resemblance between MFCC extraction and the vocal tract of a human being's. Lower-order coefficients represent the vocal tract shape, while higher-order coefficients represent periodicity in the waveform (i.e. excitation information) **[43].**

5.3.2 Framewise Phoneme Alignment

In order to be able to feed the labels — or phonemes, in this context — into the model, matching every phoneme in the data set with its corresponding frames that it spans is a necessity. This is done while looping through the files present in the data set. The phonemes actually have their own numerations (as explained in section *5.2.3*). Therefore, matching the two sequences is a fairly straightforward task. As illustrated in *Figure5.5*, sequence *S1* is an MFCC coefficient vector *Vi* coming from one of the MFCC feature vectors. Elements in *S1* correspond to frames in the speech signal, where the first element has index 0 and the last element is aligned with the total duration value of the voice recording. It is also shown in the figure how phonemes (i.e. elements in sequence *S2*) can span multiple frames, and are usually separated by a special *blank* phoneme ε which symbolises periodical silences in speech.

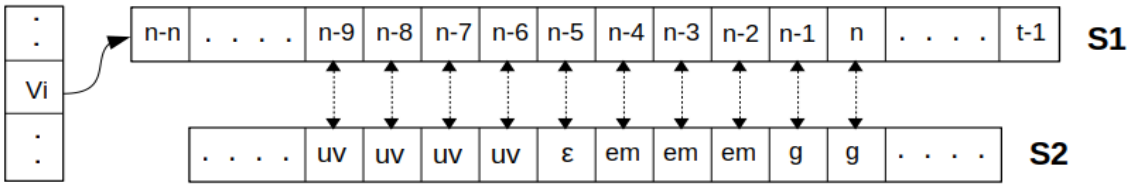


Figure 5.5: Framewise Phoneme Matching

5.3.3 Separating Data Subsets

The available data set needs to be divided into training, validation and testing sets. According to Alex Graves et al. in a very similar application **[30],** the research team predicted to reach presentable accuracies through taking 75% of the data as training data and 25% as test data, given that validation data was to be randomly chosen from the training data but kept constant throughout all relevant experiments.

5.3.4 Technical Specification

The steps and stages discussed in the previous sections were implemented in Python3.7 using multiple related libraries (NumPy, Scikit-learn, Librosa, JSON, Python_Speech_Features, etc). This implementation also included converting the WAV files from the NIST encoding into RIFF, in order to change the headers in those files. This conversion was needed because most available libraries do not support the former. Each file was segmented and appended to a global NumPy array of features along with its associated labels (imported from the corresponding PHN file). This was performed on every sentence in the data set. The data in every feature vector was aligned with its corresponding phoneme code (0-60, plus the blank character). Finally, all the preprocessed data needed for the model was encapsulated in a Python class and serialised into a PKL file in order not to have to preprocess the data every time the model is trained.

5.4 The Model

5.4.1 Network Architecture: LSTM

As discussed in *Chapter 3*, LSTM-based Recurrent Neural Networks have presented significant improvements in speech recognition applications. Therefore, a similar architecture was chosen. ANNs are explained extensively in *Appendix A*. The fact that ANNs learn features at different levels of abstraction, rather than rely on hand-crafted and domain-specific features, is one of the major advantages of using ANNs. This also runs in accordance with the trend of building end-to-end applications, minimising the need for expert knowledge and feature-specific preprocessing **[44].** LSTM architectures incorporate purpose-built memory cells that capture temporal dependencies natively and scale well with the time lag between important events **[46].**

The model of this project has LSTM blocks at its core. Such a block features memory cells that are capable of maintaining their state over time, as well as normal and fully connected units that regualte the information flow into and out of the LSTM block. The "memory" of an LSTM block lies within an internal *cell state* (shown in *Figure5.6*). Moreover, the block contains four "gates" — input, output, forget, and candidate state.

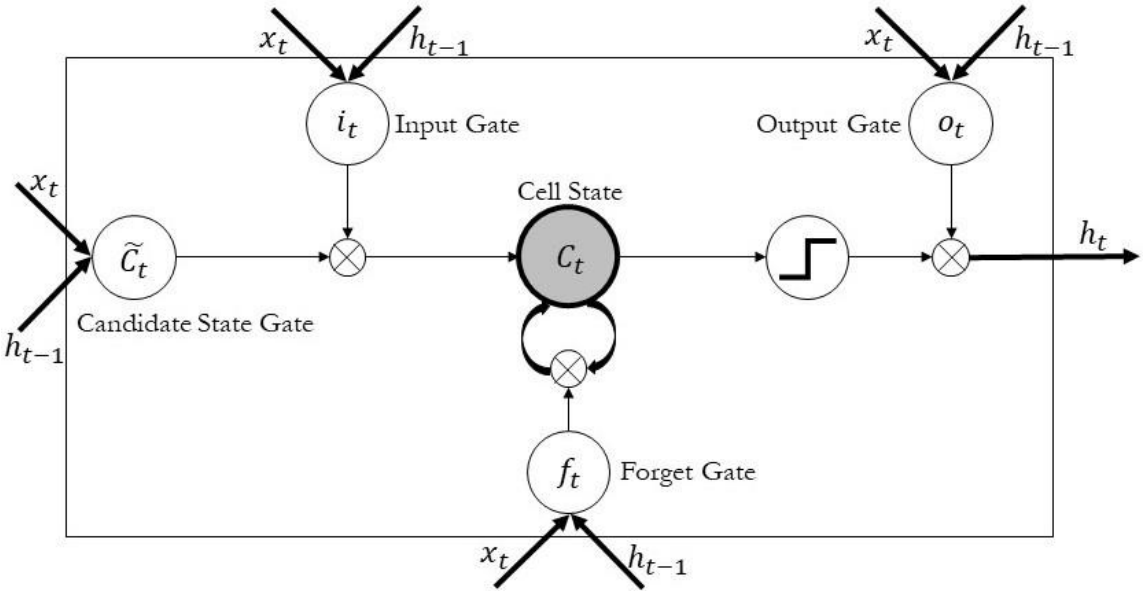


Figure 5.6: LSTM cell architecture

Each of the gates is a small RNN with dimensions $d \times h$ where d is the dimension of the input vector x_t and h is the dimension of the cell state C_t and LSTM output h_t, also called the hidden size. Each of these gates interact with the cell state, controlling how new information entering the LSTM will alter it. The equations that describe the forward pass of an LSTM block are stated and explained below.

The forget gate controls how much the current input will influence the removal of information from the previous cell state.

$$f_t = \sigma(W_f \cdot x_t + R_f \cdot h_{t-1} + b_f)$$

The input gate controls how much the current input will influence the addition of new information to the current cell state.

$$i_t = \sigma(W_i \cdot x_t + R_i \cdot h_{t-1} + b_i)$$

The output gate controls how much the current input will directly influence the current output of the network.

$$o_t = \sigma(W_o \cdot x_t + R_o \cdot h_{t-1} + b_o)$$

The candidate stage gate is the representation of new information created by the current input.

$$\tilde{C}_t = \tanh(W_C \cdot x_t + R_o \cdot h_{t-1} + b_C)$$

The current cell state is a combination of things to be forgotten and things to be assimilated.

$$C_t = f_f \odot C_{t-1} + i_t \odot \widetilde{C_t}$$

In a nutshell, with an LSTM architecture like ours, the MFCC inputs x_t are combined with the previous outputs h_{t-1} to alter the cell state C_t and to output a new prediction h_t. Context from all previous inputs is incorporated into the new cell state C_t, and therefore, the LSTM can take into account temporal dependencies that are very far apart. This is what the strength of LSTM-based speech recognition applications is attributed to — capturing long dependencies in sequences. In such applications, and in accordance with our notation, the network output is a combination of the input and the current cell state:

$$h_t = o_t \odot \tanh(C_t)$$

where,

x_t	= the current input vector
h_t	= the current LSTM output
h_{t-1}	= the previous LSTM output
f_t, i_t, o_t	= the output of the forget, input and output gates
$\widetilde{C_t}$	= the output of the candidate state gate
C_t	= the current cell state
C_{t-1}	= the previous cell state
W_f, W_i, W_o, W_C	= the input weights for gates f_t, i_t, o_t, and $\widetilde{C_t}$
R_f, R_i, R_o, R_C	= the recurrent weights for gates f_t, i_t, o_t, and $\widetilde{C_t}$
b_f, b_i, b_o, b_C	= the bias weights for gates f_t, i_t, o_t, and $\widetilde{C_t}$

Using these mechanisms during training, the LSTM can learn to discriminate between relevant and irrelevant context by adjusting the weights of its gates **[45].** During the inference process, the output of the LSTM block h_t is a combination of both the current input and the cell state, as it has been "sculpted" from the previous inputs, via their passage through the four gates. To sum up, given an input sequence $X = (x_1, \dots, x_T)$, an LSTM Neural Network computes its output squence $H = (h_1, \dots, h_T)$ by evaluating the equations mentioned earlier. The following algorithm demonstrates the procedure:

Algorithm 5.3: Forward LSTM Pass

Input: X
Output: H
$C_0 \leftarrow 0$
$h_0 \leftarrow 0$
for every sequence frame x_t, 1 ≤ t ≤ T **do**
$f_t = \sigma(W_f \cdot x_t + R_f \cdot h_{t-1} + b_f)$
$i_t = \sigma(W_i \cdot x_t + R_i \cdot h_{t-1} + b_i)$
$o_t = \sigma(W_o \cdot x_t + R_o \cdot h_{t-1} + b_o)$
$\widetilde{C_t} = \tanh(W_C \cdot x_t + R_o \cdot h_{t-1} + b_C)$
$C_t = f_f \odot C_{t-1} + i_t \odot \widetilde{C_t}$
$h_t = o_t \odot \tanh(C_t)$

5.4.2 Network Architecture: Bidirectional LSTM

Due to the asynchronous nature of speech, the model needs to take into account both previous and future information in the sequence when predicting a label (in a classification problem like ours). Therefore, a Bidirectional LSTM RNN was chosen, in which two identically-sized LSTM blocks are used in parallel, where one processes the input sequence in a forward manner (1st through T-th frame), while the other processes the input sequence in a backward manner (T-th through 1st). *Figure5.7* demonstrates the generic form of the BRNN. The project included an initial model design, which was altered through experimentation — as will be discussed in *Chapter 6*. This initial model design includes an input layer with 26 input neurons (one for each MFCC feature vector. The model also contains two bidirectional hidden layers, each of which utilises two LSTM layers (one backward and one forward) that have 100 units each, a fully connected layer **[30],** and an output layer (this descibes model *EXP04* in *Appendix D*). The output layer has 62 units (61 phonemes + a blank character). In other words, the first hidden layer consists of an LSTM network with two parallel LSTM blocks (a Bidirectional LSTM). The second layer is identical to the first. The suceeding layer is a fully connected layer that connects the outputs of both the parallel blocks to the output of the ANN as a whole.

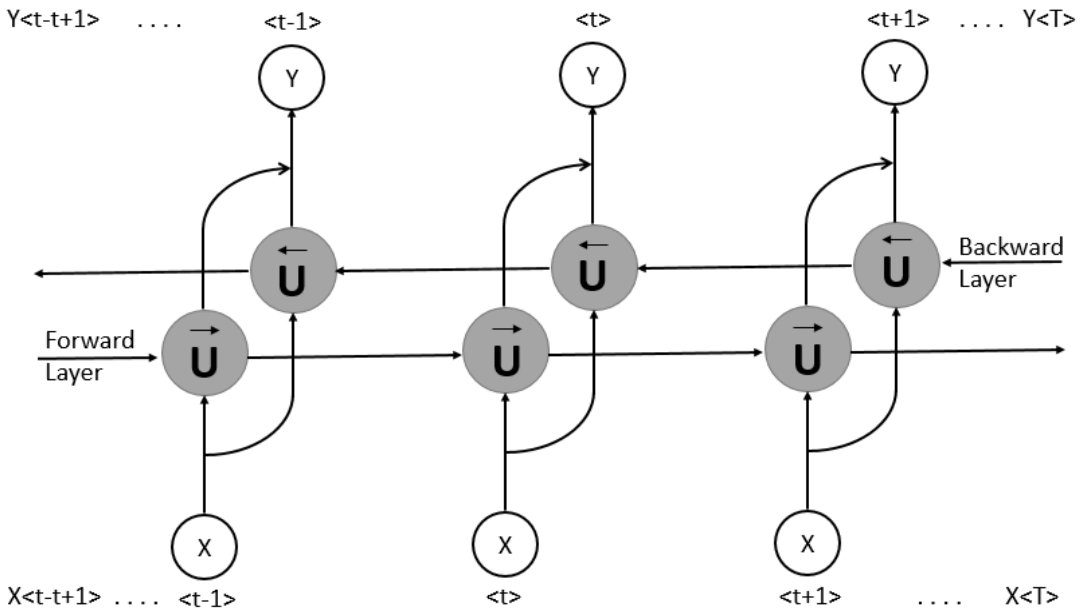


Figure 5.7: Bidirectional LSTM RNN architecture

It is common practice to reduce the number of phonemes — or classes — down to 44 by merging those with very similar sounds. Therefore, it could be beneficial to try both sets of classes and rely on the accuracy of the model for the given architecture to determine which set of classes should be the standard one in the project. Moreover, a logistic sigmoid loss function will be used in the hidden layers. For the output layer, the CTC loss function will be used, merely because of its invulnerability to continuous occurrences of phonemes (e.g. "*heeeellllllooooo"* would be transcribed as "*hello"*).

5.4.3 Connectionist Temporal Classification Loss Function

In RNNs, the fact that input and output sequnces are not necessarily synchronised raises questions regarding the way they are intertwined within the ANN. In reality, this asynchronicity does not affect the order of input components (i.e. signal frames) and output components (i.e. phonemes). In other words, the output will follow the same order as the input, even if there is no time-based synchronicity between them, thus preserving the temporal correspondence between the input and output tensors. For instance, if the input signal describes the sentence "Hello, world", then the output is — typically — expected to represent the phonemes associated with "hello" before the ones uttered in "world".

The need for a decoding-based loss function arises from the fact that every phoneme depends on the preceeding phonemes classified. Therefore, the "choice" of a phoneme will affect the succeeding ones to come. This is usually apparent in words that contain duplicate letters. For example, if the time-aligned output of the model is "bbooott", then we can remove all the duplicates, leaving us with "bot". This means that if the transcription was supposed to arrive to the word "boot" instead, the model would not have given that result. In other words, the issue of *exactly* which duplicate to remove is integral to the generalisation of the model. *Figure5.8* demonstrates how CTC attempts to fix this in a generic manner.

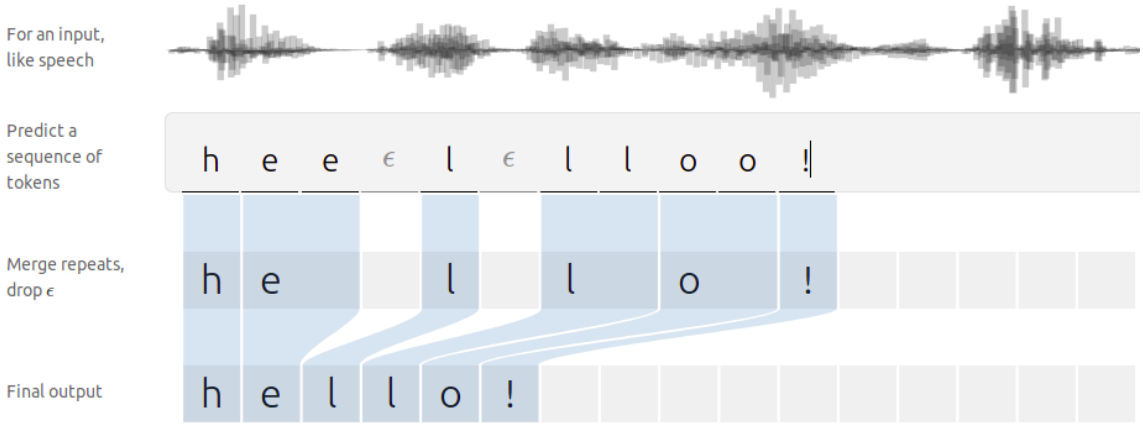


Figure 5.8: Connectionist Temporal Classification Decoding

CTC is a generic loss function to train systems on sequence problems where the alignment between the input and output sequences is unkown. Given an input sequence X of length$T$, CTC assumes the probability of a length Tcharacter sequence $C$is given by:

$$p(C|X) = \prod_{t=1}^{T} p(c_t|X)$$

This assumes that character outputs at each timestep are conditionally independent given the input **[17].** The distribution $p(c_t|X)$ is the output of some predictive model. CTC also assumes that our ground truth transcript is a character sequence W with length τ where $\tau \leq T$. As a result, we need a way to construct possibly shorter output sequences from our

length T sequence of character probabilities. The CTC collapsing function achieves this by introducing a special blank symbol, denoted by the blank phoneme class in the BLSTM (i.e. ɛ), and collapsing any repeating characters in the original length T output. This output symbol contains the notion of "junk" or other so as to not produce a character in the final output hypothesis.

The strength of CTC comes from its ability to allow true end-to-end training without requiring any frame-level alignment between inputs and outputs. By summing up over all sets of label locations that yield the same label sequence, CTC determines a probability distribution over possible labellings, conditioned on the input sequence. A CTC layer has as many output units as there are distinct labels for a task, plus an extra blank character unit for "no label". The activations of the outputs at each timestep are mormalised and interpreted as the probability of observing the corresponding label (or no label) at that point in the sequence.

5.4.4 Phonemic Similarity

From another perspective, the project as a whole requires a phonemic similarity metric. As discussed in *Appendix B*, sequence matching algorithms like Levenshtein Error Distance, Phonolgical Edit Distance and Trigram Comparison suffer from length dependency. However, and since phoneme transcription generally yields more phonemes than there are characters in corresponding words, such dependency is likely to lead to insufficient accuracy in mesurement. Therefore, said algorithms were excluded. This bias towards some variable (i.e. length, in this context) also caused the Jaro-Winkler algorithm to be put aside, as it gives higher weights to characters in the beginning of the sequence. The phonemic similarity measure chosen is the Khorsi Similarity Metric. This is merely because it gives more weights to characters that are of greater importance to the word as a whole, regardless of where they fall in the sequence.

5.4.5 Technical Specification

The model was implemented in Python3.7 using Google's TensorFlow (TF) Deep Learning library. This is due to the wide support as well as reliable documentation provided by the company on Bidirectional RNNs, compared to Theano, Keras or PyTorch. Though PyTorch is superior when it comes to DL data visualisation, TF still has the higher foot in being well-documented. Moreover, this choice looked more promising for integrating the model into the mobile application, as Google released a TF-like mobile device tool (i.e. TensorFlow Lite) which supports model integration through frozen checkpoints. Finally, the Khorsi Similarity Metric was utilised using the Python library Phonological CorpusTools.

5.5 The Mobile Application

5.5.1 Development Parameters

The mobile application was developed to operate on AndroidOS devices of API version 22 and above. The front-end code is written in XML, and is divided into Fragments and Activities. The back-end code is divided into three major sections; *Speech-to-Text (STT)* conversion, *Text-to-Speech (TTS)* synthesis, and the DL model utilities — all written in Java.

5.5.2 Speech-to-Text Functionality

The mobile app includes a voice search functionality through which the user can utter a word or a sentence that they would like to find out the correct pronunciation of. This means that the user does not have to pronounce the phrase correctly, but rather say something as close to it as possible. Regardless of the STT tools used, a crucial part of the process was to record the user pronouncing the search term or phrase. This is merely for uses by other components in the application. For example, the user should be able to hear the app's pronunciation as well as their own before they choose to move onto the similarity analysis section, so that they can hear the difference for themselves. Therefore, there needed to be a part of the app that records *while* recognising (i.e. transcribing) the speech signal. Although Android has a built-in speech recognition tool (called SpeechRecognizer), it does not support the simultaneous execution we are looking for. Android's SpeechRecognizer performs live speech recognition, but it does not support saving the voice recording. Moreover, Android does not allow multiple built-in services to be run simultaneously. Meaning, as a security measure, Android does not allow the VoiceRecorder function to operate alongside SpeechRecognizer. This is a major issue in the subtlty of execution flow.

Since the STT transcription functionality is of great importance to the app, other means of speech recognition were examined — *Google Cloud Platform (GCP), Amazon Web Services (AWS)* and *Microsoft Azure* Cloud Computing platforms. STT was eventually implemented using *Google Cloud Speech (GCS) API*. Multiple cloud platform APIs were tested, specifically *Amazon Transcribe* and *Microsoft Azure Cognitive Speech Services*. However, GCS proved superior in the variety of languages and accents that it supports (over 120). The final implementation of STT includes the app using the built-in VoiceRecorder functionality to record the user uttering the phrase in mind. This recording is passed as a *Pulse Code Modulation (PCM)* onto the Cloud Speech API, which returns the transcription of the speech uttered in the speech recording. It is worth mentioning that audio file formats like MP3 were not used in this project, as their nature dictates the loss of a lot of details — trimming the audio file into only the frequency domain that we, humans, can hear, even though there could be useful acoustic data beyond our hearing capacity (20kHz).

5.5.3 Text-to-Speech Functionality

From another perspective, Text-to-Speech (TTS) is the mobile app's way of determining the actual pronunciation the user is looking for. This also recquired the search for an API, as Android's built-in tool (namely, Android Speech API) does not support saving the synthesised speech voice recording. Meaning, the same problem faced with Android SpeechRecognizer was faced with Android Speech API. This introduced the need for an alternative, also involving Cloud Computing platforms. Although GCP has a very reliable TTS API, AWS was chosen over it. GCP, just like Microsoft Azure, does not support saving the utterance in lossless WAV with the same header. It uses a different encoding scheme for such file formats. Therefore, AWS' TTS API (namely, *Amazon Polly*) was chosen here. The API takes a string as an input and returns the pronunciation of it as a PCM file (which is dealt with the same way as the STT file).

5.5.4 Model Deployment

The model is at the core of the project as a whole. The form in which it was saved expects two WAV files — in RIFF encoding — to be input into the model, following the input tensor specification. The model then performs the phonemic transciption on both, consequtively, and passes the phoneme vectors onto the phonemic similarity measurement code, which returns the final transcriptions as well as the similarity percentage. Therefore, there is some preprocessing and postprocessing involved at every inference.

Even though Google's TensorFlow Lite is getting more famous, especially due to the wide cross-functional interoperability with TensorFlow, it led to the biggest implementation issue faced in this project. Since inference does not only rely on the model itself (as a frozen checkpoint), but also on the preprocessing and postprocessing involved, TensorFlow Lite was unusable. The TFLITE file that checkpoint freezing yields can only be so useful. It only includes the model's body — as an ANN. Consequently, there arose a need for breaking the source code into code that can be frozen into a TFLITE file and code that needs to be run as is — in Python. Android does not natively run Python code. Rather, a Python interpreter would need to be installed on the same phone. This means installing two mobile apps only to use one. Therefore, the option was overthown.

Cloud Computing platforms, again, seemed more promising. GCP's Compute Engine was utilised in the mobile application, as the mobile application was already authenticated with the platform (using Cloud Speech API). However, it turned out that the authentication process for the two APIs differs. Subsequently, authenticating to Compute Engine was initiated, but it eventually failed. Another security measure was that GCP could not let an Android application authenticate two instances for the same use at the same time. Breaking the authentication into two parts was out of the question (for reasons that will be expalined in *5.5.5*). Therefore, AWS' *Elastic Compute Cloud (EC2)* API was implemented instead.

Migrating the model to the Cloud includes uploading all the Python code needed for inference and any processing related to it on the Cloud, and accessing it through a *Secure*

Shell (SSH) request to execute the code and receive the results from it — much like executing the project through the Terminal. Therefore, a virtual CPU instance was created on EC2 and accessed through an SSH request from an Ubuntu-running computer. This was successful, but led to a different kind of a problem. The mobile application needed to do the same (i.e. request an SSH channel and authenticate it using valid AWS credentials). Android apps can easily start an SSH channel, but attaching a file to it can be tricky, if the permissions are an issue. As a security measure, AWS requires the credentials to be in an encrypted PEM file whose access permissions are set to: *"Owner → Read-Only"* and *"Other → None"*. On Ubuntu and other operating systems, this is simply a matter of changing permissions to the required ones so that the SSH channel can authenticate to EC2. However, Android does not support files with such permissions unless they are set as an *Application Signature*. Such a case is only applicable in apps that have already been verified and are currently on the App Store.

Android applications can create a *File* instance – in code – and set the permissions of it to read-only. However, this does not include the party with Owner privilages (namely, the app itself). Adding the PEM file to the application is one of the worst practices out there if the application is production-level. But, since the application is for academic purposes, adding the PEM file to be compressed alongside the rest of the files into an executable APK file was also tested. However, the same permissions problem was faced. In short, an app cannot create a file, save it on the Android device and change its Owner's permissions to read-only, as the Owner is the app itself and it needs to be able to modify it all the time. Other parties' access permissions can be set to None or Read-only. Just not the Owner's

5.5.5 Multithreaded Components

An long-lasting issue that was faced was the limitation caused by Android's security measure of "No network operations should be on the main thread". Android forces apps to use their respective main threads only for UI-related operations. Any network access, download, upload or other operations that can run simultaneously alongside the UI should be running on different threads. Therefore, evey API authentication needed to be performed on a separate thread. This introduced multthreading-related issues (e.g. when one thread is waiting for a result from another thread that has not finished executing). This "forcable" multithreading was implemented successfully.

Chapter 6

Evaluation and Results

In this chapter, the evaluation criteria used in determining the effectiveness and efficiency of the model are laid over. This also includes the evaluation measures taken into account. The chapter also discusses the overall results obtained from experiments (included in *Appendix D: Experimental Trials*) as well as justification of the best results obtained.

6.1 Evaluation Criteria and Reliability Measures

In Automatic Speech Recognition systems, evaluation approaches are usually categorised into subjective and objective approaches [REF]. The former includes direct human involvement in the measurement process, while the latter does not require human involvement (testing is done through voice recordings). This clear line between both categories is attributed to the fact that approaches involved in each are very different. Although they yield reproducible results, objective methods are usually hard to build, as replicating or simulating testing environments that resemble real-life situations is not a straightforward task. On the other hand, subjective evaluation measures are usually more implementable, but less reliable **[54].** This is mostly due to the fact that, with semantic content, humans are not expected to perform high-quality performance measurements on standardised scales across applications.

6.1.1 Bias-Variance Tradeoff

It is undeniable how strong the effect of model architecture is on the model's generalisation **[55].** This is usually arrived to through experimentation, as measuring generalisation is specific to the model and the data set on which it was trained. With Machine Learning in general, reliability of models is determined by the presence (or absence) of overfitting and underfitting. *Overfitting* (i.e. high-variance) in models refers to the result of model training when the model learns too much. In other words, the model can learn the desired features as well as the scattered noise around those features. This can usually be attributed to *over-training*, in which the model is left to train for longer than needed to the point where it starts learning "unnecessary" patterns. On the other hand, *underfitting* (i.e. high-bias) refers to the situation where the model does not learn enough or capture useful patterns from the training set. This can usually be "blamed" on the data set or the data preprocessing involved prior to training. In Machine Learning, there is a tradeoff faced in every application between the two (bias and variance) for determining the best generalisation performance that minimises both sources of error. It is worth mentioning that, in speech recognition systems, models are more likely to suffer from overfitting rather than underfitting. This is due to the fact that the recording environment can include too much noise — and so can the voice recordings. Therefore, a fixed

sampling rate usually suffices when it comes to regulating the amount of noise we want the voice recording to capture or present.

6.1.2 Dimensionality Reduction

ASR systems still rely, to a great extent, on the feature set extracted from the input speech. One of the problems that are usually faced with such applications is the high dimensionality of the feature space corresponding to the large number of temporal features extracted. Physical system parameters can also worsen this dimensionality, depending on the physical components involved in the recording and frequency transmission of sound. Although MFCC feature extraction compresses the speech signal, it does not overcome the *Curse of Dimensionality* — the amount of computations and data required for training a model grows exponentially with the increase of the dimensionality of feature vectors.

Aiming for higher recognition accuracy dictates the need for feature dimensionality reduction, ideally while preserving discriminability between phonetically different sounds **[47].** High-dimensional acoustic representations of fetures do not imply better results, but rather does the nature of extracted features. Wang and Pilawal mention: "[…] with the increase of the dimensionality of feature vectors, more training data are needed to train the system models. […] When the number of features increases faster than the increase of the number of training data, the system models obtained may lose the generalisation properties because of insufficient training data" **[48].** Notwithstandingly, given enough dimensions, a group of patterns may be split up in any arbitrary way, likely leading to better representations for discrimination on the training set, making higher-dimensional feature spaces more prone to overfitting. Interestingly, the best representations for generalisation to the test set are usually succinct **[56].** Too detailed a representation may also represent characteristics of the particular training set that will not be present for independent data sets — forcing the model to learn set-specific noise and likely resulting in overfitting. Finding a *good* set of generalisable features in a perceivably moderate-size feature dimension space is a challenging process that relies heavily on experimentation.

6.1.3 Accuracy Metrics

Model performance evaluation metrics are determined by the nature of the problem that the model is trying to solve. When it comes to classification, the *Confusion Matrix* is one of the most frequently used accuracy measures. Confusion Matrices help identify the effectiveness of a classifier in a class-based manner, breaking classifications into the structure shown in *Table6.1*. It is worth mentioning that the performance metrics to come rely heavily on the values extracted and presented within the confusion matrix. Since the outputs simply would not be visuallly appealing, the confusion matrix values were calculated but the matrix itself was not presented. Rather, other confusion matrix-based metrics (shown in *Table6.2*) were presented at training, as those are also of great importance in performance evaluation.

Table 6.1: Confusion Matrix classification labels

Label	Observtion	Prediction
True Postitive (TP)	Positive	Positive
False Positive (FP)	Negative	Positive
True Negative (TN)	Negative	Negative
False Negative (FN)	Positive	Negative

Through the breakdown above, confusion matrix-based metrics help pinpoint the strengths and weaknesses of the classifier when it comes to the types of mistakes it makes with certain classes. It is worth mentioning that metrics derived from the confusion matrix are used in different approaches to generate a polymorphic performance measure. For instance, comparing *percision* and *recall* is usually integral. A lot of models are deliberately built in such a way that skews the precision-recall relationship to one side rather than the other. For example, when the stakes are high, as is the case with disease detection models, the recall value is far more important than that of precision's, as false negatives could be detrimental. On the other hand, in content recommendation systems, precision is likely to be of more importance than recall, as false negatives are less of a concern. Finally, *F1 Score* is calculated as a harmonic mean (i.e. reciprocal of the arithmetic mean of reciprocals) of both precision and recall, taking both metrics into account.

Table 6.2: Accuracy Metrics derived from Confusion Matrices

Accuracy	Precision	Recall	F1 Score
$acc = \frac{TP + TN}{Total}$	$pre = \frac{TP}{TP + FP}$	$rec = \frac{TP}{TP + FN}$	$f1 = 2 \times \frac{pre \times rec}{pre + rec}$

Other involved measures include Mean Error, where error rates come from the CTC loss function. Furthermore, an ASR-specific performance metric used is the Mean Label Error Rate. This works the same way as Word Error Rates, which are used very frequently when evaluating speech recognition systems.

6.2 Results and Discussion

Proving the validity of the initial expectations, different architectures led to fairly different results, even when all the other hyperparameters were set constant. This is shown in the experiments log (*Appendix D*). The experimentation stage started off with a couple of experiments (namely, *EXP01* and *EXP02*) that yielded highly insufficient results of 8-10%. This can be attributed to the topology of the ANN, as those two ANNs in particular have narrow architectures with relatively high depth and low number of hidden neurons, compared to state-of-the-art applications that usually include 2-4 hidden layers. Another experiment (*EXP14*) was later on added to the list of highly inaccurate experiments, but for a different reason. The accuracy *EXP14* yielded (i.e. 7%) can be explained by looking at the topology, epoch count as well as batch size. Such a DNN with a fairly complex architecture, which includes 1200 hidden neurons in total (600 forward LSTMs + 600

backward LSTMs), needs to be given the chance to learn properly. A batch size of 20 and epoch count of 10 just would not be efficient enough in regards to the training duration for the ANN to capture useful patterns.

Interestingly, even though they had different architectures, the rest of the experiments enclosed within *Appendix D* had similar testing accuracies, falling between 57 and 67 percent. This is where the need for other evaluation metrics arises. In terms of F1 Scores, those experiments fall in three different ranges. While *EXP05*, *EXP08* and *EXP09* simply had insufficient F1 Scores of 0.4 on average, *EXP07* and *EXP10* just proved superior with their F1 Scores exceeding 0.7. The rest of the experiments did not present any outlying promise, with F1 Scores of almost 5.5 on average.

It could be argued that the error function used (i.e. CTC) did not need to stay constant throughout the experimentation stage. However, the function proved efficient and extremely useful in resolving sequence loss measurement problems in applications that have varying time frames. Moreover, the Label Error Rate might just not be the optimal error rate metric. This could also be argued, as Word Error Rates (the basis of LER) is still highly famous in ASR research.

The signals themselves could also have affected the overall accuracy. Even though the TIMIT group made it clear that all samples were saved at 16kHz, this does not disprove the inclusion of a downsampling stage during data collection. Downsampling could affect the quality of recordings, resulting in more noise that can steer the training phase away from optimality. Furthermore, there is a clear imbalance between the male speaker count and the female one. This can be a major factor in determining the generalisaiton of the model as a whole, and the training of the model to begin with.

Other variables that could have yielded different results include the number of phonemes and the number of MFCC feature vectors. It is still unclear the difference between results arrived to with the full English phoneme list (i.e. 61) against those with the reduced phoneme dictionary (44). The consistency of using 61 phonemes in all experiments might have dissuaded the model from reaching a high accuracy. From a different perspective, feature vectors were also kept constant throughout experimentation. Even though research has shown that the 26 39-dimentional MFCCs are likely to reach better results than the original 13 39-dimensional ones, it still does not make the impact of which negligible in terms of overall accuracy.

It is probably worth mentioning that the experimentation stage was, in fact, not completely satisfactory. This is mostly due to the time limitation that was directly affected by the research stage as well as implementation before arriving to the experimentation phase. With different objectives and variables, the results obtained would have definitely been different.

Chapter 7

Reflections and Future Work

As the project was a lengthy one, certain modifications would have probably made a difference in the implementability and efficiency of project components. These "technical regrets" can be broken down into three categories; reflections in regards to the research stage and approach, reflections on the implementation as well as personal reflections on the project as a whole. This chapter discusses the strengths and weaknesses of the project, tackling each major component separately. It is worth mentioning that future work refers to the approaches expected to fix the drawbacks that the project suffers from.

7.1 Data Preprocessing

Although MFCC feature extraction usually suffices in related applications, as discussed in *Chapter 2*, experimenting with more preprocessing sub-components would benefit the research a great deal. Experimentation is a major part on the path to proving (or disproving) a scientifc hypothesis. Abiding by that statement could be accomplished through manipulating the speech signals themselves before extracting the features. For instance, reducing background noise is likely to yield better results, or a model that is less prone to overfitting (with the right architecture) to say the least. Furthermore, normalising volume in such a way that manoeuvres peaks in the speech signal as well as loudness of articulation is worth testing, as it could eventually result in better performance in regards to variance in amplitude levels.

When dealing with audio signals, production-level systems usually incorporate signal filtering. Dealing differently with frequencies that do not fall within a desired range leads to more-easily managed variables in the system as a whole. For example, a low-pass filter is likely to attenuate friction noise in audio recordings. On the other hand, a high-pass filter introduces the idea of dealing differently with frequencies above a particular limit, which could be useful when including other variables of articulation, such as emotion.

7.2 Data Set Involved

Even though the TIMIT data set used is probably the most famous one in phonemic transcription applications, it could be beneficial to explore other data sets or even integrate them together if their formats and structures allow data integration. This could require other data mining concepts to be utilised, such as data cleaning. A simpler modification on the project would be manipulating the TIMIT data set split. The TIMIT group already split the data set into training (75%) and test (25%), making it not a straightforward task to adjust the split, especially with all the files and categories involved in parsing through the set. This was actually implemented, but it was not tested on splits other than the one mentioned in *Appendix D* (i.e. 75-15-10), mostly due to time limitations.

7.3 ANN Hyperparameters

ANN training vastly depends on the loss function involved. The need for a good loss metric (i.e. function) arises from the way backpropagation works — penalising nodes that are responsible for a larger portion of the error. The project used the CTC loss function due to its spread in similar tasks and ASR in general. Recently, there was an updated version of the function (mentioned in section *3.5*) called *Gram-CTC*, that follows the same concept, but with the additional functionality of yielding multiple output characters for every time frame, improving performance with long-term dependency. Even though the function is worth testing, it was not implemented in this project due to lack of experimentation time.

Gradient Descent, as an optimisation algorithm, relies heavily on the learning rate. Since it was used in tuning the weights during backpropagation to get the performance nearer to a minimum-cost, evaluating the performance with different learning rates would have benefitted the overall accuracy of the model. The learning rate can sometimes be responsible for whether or not the model converges to a minimum cost. Therefore, it is recommended in research to tune hyperparameters of training and make observations regarding the accuracy rates associated with each.

7.4 Evaluation Methods

The TIMIT data set is nothing but rich with a plethora of accents and pronunciations. However, improving the generalisation of the model could be a matter of validating on the right data. For example, facilitating *k-Fold Cross-Validation* would be a better measure of inference performance. Moreover, class-based accuracy measures like the Confusion Matrix are a good way of deciding on the optimal training by being able to identify the model's performance in regards to particular classes. Since this project includes classes that have relatively unuseful labels to us in terms of readability (e.g. *"ix", "hh", "ax-h", "uw",* etc), it was excluded from this project. However, visualising the patterns in classification (i.e. the diagonal of the confusion matrix) could be of great help — given the right tools. For example, visualising the confusion matrix on a monitor rather than on paper. Therefore inclusion of such visualisation technique is worth testing out during the training process.

7.5 The Mobile Application

It is worth emphasising on that the model deployment (or absence of) into the application is the biggest issue faced with the project. This led the project to not be usable or tested by other people, due to the missing communication between the model and the applciation. Had there been more time to deal with the issue, there would have been changes regarding different causes of the problem. For instance, unifying the Cloud Computing platform APIs into those that belong to the same platform is a must. This bad practice of involving multiple cloud computing platforms (i.e. GCP and AWS) is a major stigma —unfortunately — in the performance of the application. It is worth mentioing that the application also suffers from a security-related problem, as it has the GCP and AWS authentication files

compressed into it. This is a major security breach in production-level systems. However, the purpose of the application was imagined to be a "proof of point" rather than a reproducable product.

Fixing the model deployment problem that was still unresolved, in real production-level systems, usually takes the approach of having a server instance that deals with user and app authentication, and another instance for actually running the code on the Cloud. This includes two steps of SSH requests. One request is initiated by the application to authenticate to the Cloud, while the other is done from the authentication server instance to the actual Cloud to execute the code and get results — much like using the Terminal.

Although the project in ubiquity does not suffer from absence of results, it does lack a way of visualising those results. This is where the original objectives and the final outcomes of the project face a relative mismatch. The core functionality as well as the components proposed were attempted, but some of them were unsuccessful. For instance, developing user profiles through which users can visualise their progress in the learning journey was not implemented. These profiles were to be a tool of letting users see their highest and average pronunciation accuracies over the time they have been using the application. Moreover, speaker recognition as well as voice activation were a proposed feature that was not implemented either. It was actually tested out using Microsoft Azure, since it provides an API with neat documentation for that purpose. However, including yet another Cloud Computing platform — with its own authentication scripts — did not look like such a bright idea. The application was dealing with enough threads running simultaneously as is.

The mobile application could also have introduced more flexibility in performance to the user. For example, choices regarding personal preference in terms of accent and gender of the speaker (i.e. the API) would have made the app more usable, enriching user experience. Furthermore, multi-language functionality is an aspect that can have its own research and implementation stages, for the benefit of the user. With such functionality, language detection could also make the difference in increasing the usability of the app.

Finally, components involving complex multithreading need to be minimalised and have efficient time-based and process-based communication. Threads that last for as long as the execution can be merged into a thread that regulates access and functionality based on the process at hand. It is important to mention that multithreading cannot be neglected, as Android forces authentication scripts to run in parallel to the UI thread. A final thought would be to better the manner in which the code executes, both in the application and on the Cloud, in such a way that imporves time complexity. In the same context, improving space complexity of the model is not of high priority, as the frozen training checkpoint is usually not large in terms of memory.

7.6 Personal Reflections

The most intellectually-exhaustive stage of the project was the research, incorporating multiple challenges and taking up much of the time spent on the project, especially in the first semester. A problem faced by junior researchers is the excitement of seeing their knowledge actually be implemented. Unfortunately, this led to "clumsy" intermediate implementations, that were not included in the final product but took up proportionally a substanial amount of time during research. For instance, learning how MFCC feature extraction works did not do the trick, but rather did implementing the feature extractor itself. Moreover, algorithms like Dynamic Time Warping, signal autocorrelation, and acoustic fingerprinting were also implemented, even though they were not included in the final product. It is worth mentioning that this helped me facilitate my knowledge through tangible manifestations. But, it did indeed take up much of the time.

The project was one of the biggest challenges I faced in my academic life. However, with all honesty, it was probably the most exciting. Running in circles, not knowing exactly what to do next, but knowing enough to believe that the answer is in a book or a research paper that I have not got to yet was what kept me going throughout the research stage. In fact, I actually took pride in and enjoyed writing the report. Everything mentioned in this report was going around in my head for months, waiting for a chance to be expressed in words. Within the same context, the amount of research papers read throughout the process is too large for me to even be confident talking about the mobile application. I was always focused on the research and never really enjoyed implementing the mobile application — contrary to intermediate implementations. I guess now I know exactly where I lie on the Computer Science spectrum.

Conclusion

Computer-Assisted Language Learning is growing bigger than ever, with the frequent advances in Deep Learning and its applications. This project goes over an application that utilises two major aspects of Automatic Speech Recognition through a Deep Learning approach; Phonemic Transcription and Goodness of Pronunciation (GoP) scoring. The former being limited to the speakers' dialect when the data was collected, and the latter being often challenging to researchers, resulting in applications that only work on fixed sets of words. The project aims to help students learning English fix their pronunciations. This is done through a DNN that transcribes speech into phonemes and matches the learner's pronunciation to that of the API's through sequence and string comparison algorithms, giving the user a similarity indicator between the two.

Bibliography

[1] Beatty, K. (2003). *Teaching and researching computer-assisted language learning*. New York: Longman.

[2] Barson J. & Debski R. (1996) "Calling back CALL: technology in the service of foreign language learning based on creativity, contingency, and goal-oriented activity". In Warschauer M. (ed.) Telecollaboration in foreign language learning, Honolulu: University of Hawaii, Second Language Teaching and Curriculum Center: 49-68.

[3] Warschauer M. (1996) "Computer Assisted Language Learning: an Introduction".
In Fotos S. (ed.) *Multimedia language teaching*, Tokyo: Logos International: 3-20.

[4] Taylor R. (1980) *The computer in the school: tutor, tool, tutee*, New York: Teachers College Press.

[5] Ahmad K., Corbett G., Rogers M., & Sussex R. (1985) *Computers, language learning and language teaching*, Cambridge: Cambridge University Press.

[6] Underwood J. (1984) *Linguistics, computers and the language teacher: a communicative approach*, Rowley, MA: Newbury House.

[7] Taylor M.B. & Perez L.M. (1989) *Something to do on a Monday*, La Jolla, CA: Athelstan.

[8] Meskill, C. (2002). *Teaching and learning in real time: Media, technologies, and language acquisition*. Houston, TX.

[9] Denton, P. (2003). Enhancing Student Learning using Electronic Feedback 9.
Exchange, 4(Spring), 24.

[10] Denton, P. (2003). Returning Feedback to Students via Email Using Electronic
Feedback 9, Learning and Teaching in Action 2(1), 1-2.

[11] Allwood, Jens. (1999). Feedback in Second Language Acquisition.

[12] Cornillie, F., Clarebout, G., & Desmet, P. (2012). The role of feedback in foreign language learning through digital role playing games. Procedia - Social and Behavioral Sciences, 34, 49–53.

[13] Davies G., Walker R., Rendall H. & Hewer S. (2012) Introduction to Computer Assisted Language Learning (CALL). *Information and Communications Technology for Language Teachers (ICT4LT)*, Slough, Thames Valley University [Online].

[14] Acoustical Society of America. ANSI/ASA Standard S1.1-2013, 2013.

[15] Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review, 65*(6), 386-408.

[16] John E. Moody. 1991. The effective number of parameters: an analysis of generalization and regularization in nonlinear learning systems.
In Proceedings of the NIPS'91.

[17] K. Zhang, W. Zuo, Y. Chen, D. Meng and L. Zhang, "Beyond a Gaussian Denoiser: Residual Learning of Deep CNN for Image Denoising," in IEEE Transactions on Image Processing, vol. 26, no. 7, pp. 3142-3155, July 2017.

[18] Riedmiller, M., & Braun, H. (1993, March). A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *Proceedings of the IEEE international conference on neural networks* (Vol. 1993, pp. 586-591).

[19] Maas, A. L., Jurafsky, D., & Ng, A. Y. (2014). Lexicon-Free Conversational Speech Recognition with Neural Networks.

[20] Miao, Y., Gowayyed, M., & Metze, F. (n.d.). EESEN: End-to-End Speech Recognition Using Deep RNN Models and WFST-based Decoding. Language Technologies Institute, School of Computer Science, Carnegie Mellon University. (n.d.).

[21] Gales, M., & Young, S. (2008). The Application of Hidden Markov Models in Speech Recognition, 1(3), 195–304.

[22] J. M. García-Cabellos, C. Peláez-Moreno, A. Gallardo-Antolín, F. Pérez-Cruz and F. Díaz-de-María, "SVM classifiers for ASR: A discussion about parameterization," *2004 12th European Signal Processing Conference*, Vienna, 2004, pp. 2067-2070.

[23] Krizhevsky, et al., "ImageNet Classification with Deep Convolutional Neural Networks." NIPS, 2012.

[24] Solera-Ureña, R., Martín-Iglesias, D., Gallardo-Antolín, A., Peláez-Moreno, C., & Díaz-de-María, F. (2007). Robust ASR using Support Vector Machines. Speech Communication, 49(4), 253–267.

[25] Wu, J. (2017). Introduction to Convolutional Neural Networks.

[26] Scherer, D., Müller, A., & Behnke, S. (n.d.). Evaluation of Pooling Operations in Convolutional Architectures for Object Recognition.

[27] K.-F. Lee and H.-W. Hon. Speaker-independent phone recognition using Hidden Markov Models. IEEE Transactions on Acoustics, Speech and Signal Processing, 37(11):1641–1648, 1989.

[28] Song, W., & Cai, J. (2015). End-to-End Deep Neural Network for Automatic Speech Recognition. CS224N Projects, 1–8.

[29] Vaziri, N. D., Yuan, J., Rahimi, A., Ni, Z., Said, H., & Subramanian, V. S. (2012). Disintegration of colonic epithelial tight junction in uremia: A likely cause of CKD-associated inflammation. Nephrology Dialysis Transplantation, 27(7), 2686–2693.

[30] Graves, A., & Schmidhuber, J. (2005). Framewise phoneme classification with bidirectional LSTM networks. Proceedings of the International Joint Conference on Neural Networks, 4, 2047–2052.

[31] Liu, H., Zhu, Z., Li, X., & Satheesh, S. (2017). Gram-CTC: Automatic Unit Selection and Target Decomposition for Sequence Labelling.

[32] Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. 2006. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd international conference on Machine learning* (ICML '06). ACM, New York, NY, USA, 369-376.

[33] Zhao, Y., Jin, X., & Hu, X. (2017). Recurrent convolutional neural network for speech processing. In 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) (pp. 5300–5304).

[34] Zhang, Z., Sun, Z., Liu, J., Chen, J., Huo, Z., & Zhang, X. (2016). Deep Recurrent Convolutional Neural Network: Improving Performance For Speech Recognition.

[35] Mohamed, Abdel-rahman & Hinton, Geoffrey. (2010). Phone recognition using Restricted Boltzmann Machines. ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings. 4354 - 4357. 10.1109/ ICASSP.2010.5495651

[36] Ma, Yukun & Cambria, Erik & Bigot, Benjamin. (2017). ASR Hypothesis Reranking using Prior-informed Restricted Boltzmann Machine.

[37] Majeed, S. A., Husain, H., Samad, S. A., & Idbeaa, T. F. (2015). Mel frequency cepstral coefficients (MFCC) feature extraction enhancement in the application of speech recognition: A comparison study. Journal of Theoretical and Applied Information Technology.

[38] Kumar, Kshitiz & Kim, Chanwoo & Stern, Richard. (2011). Delta-spectral cepstral coefficients for robust speech recognition. ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings. 4784-4787. 10.1109/ ICASSP.2011.5947425.

[39] X. Xiong, "Robust speech features and acoustic models for speech recognition," PhD. Thesis, 194 p., Nanyang Technological University, Singapore, 2009.

[40] Lopes, C., & Perdigao, F. (2011). Phoneme Recognition on the TIMIT Database. Speech Technologies.

[41] John S. Garofolo, L F. Lamel, W M. Fisher, Jonathan G. Fiscus, D S. Pallett, Nancy L. Dahlgren (1993). Darpa Timit Acoustic-Phonetic Continuous Speech Corpus CD-ROM. NIST Interagency/Internal Report (NISTIR) – 4930.

[42] X. Huang, A. Acero, and H.-W. Hon, Spoken Language Processing vol. 15: Prentice Hall PTR New Jersey, 2001.

[43] J. W. Picone, "Signal Modeling Techniques in Speech Recognition," Proceedings of the IEEE, vol. 81, pp. 1215-1247, 1993.

[44] A. Graves and N. Jaitly. Towards End-to-End Speech Recognition with Recurrent Neural Networks. JMLR Workshop and Conference Proceedings, 32(1):1764-1772, 2014.

[45] G. Evangelopuolos, Efficient Hardware Mapping of Long Short-Term Memory Neural Networks for Automatic Speech Recognition, 2016.

[46] A. Graves, A. Mohamed, and G. Hinton. Speech Recognition with Deep Recurrent Neural Networks. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), (3):6645-6649, 2013.

[47] A., S., & Hu, H. (2011). Nonlinear Dimensionality Reduction Methods for Use with Automatic Speech Recognition. Speech Technologies. doi:10.5772/16863

[48] X. Wang, K.K. Paliwal. Feature extraction and dimensionality reduction algorithms and their applications in vowel recognition Pattern Recognition, 36 (2003), pp. 2429-2439

[49] Sanders, N. C., & Chin, S. B. (2009). Phonological Distance Measures. Journal of quantitative linguistics, 16(1), 96–114.

[50] Levenshtein VI. Binary codes capable of correcting deletions, insertions, and reversals. Doklady Akademii. 1965;163(4):845–848.

[51] Khorsi, A. (2013). On morphological relatedness. Natural Language Engineering, 19(4), 537-555.

[52] Winkler, W. E. (1990). "String Comparator Metrics and Enhanced Decision Rules in the Fellegi-Sunter Model of Record Linkage" (PDF). Proceedings of the Section on Survey Research Methods. American Statistical Association: 354–359.

[53] Muda, L., Begam, M., & Elamvazuthi, I. (2010). Voice Recognition Algorithms using Mel Frequency Cepstral Coefficient (MFCC) and Dynamic Time Warping (DTW) Techniques. CoRR, abs/1003.4083.

[54] Kate S. Hone and Robert Graham. 2000. Towards a tool for the Subjective Assessment of Speech System Interfaces (SASSI). Nat. Lang. Eng. 6, 3-4 (September 2000), 287-303.

[55] Neyshabur, B., Bhojanapalli, S., McAllester, D.A., & Srebro, N. (2017). Exploring Generalization in Deep Learning. NIPS.

[56] Gold, B., Morgan, N., and Ellis, D. (2011). Speech and Audio Signal Processing (2nd Edition). Wiley Publications.

Appendix A

Deep Learning Foundations

A1 Biological Background

It makes all the difference to think that the one thing that defines our effectiveness as human beings is the thing that we understand the least — the human brain. A set of billions of small, heavily interconnected components that exchange electrical signals whose values determine what the surrounding neurons should exchange with their neighbors. This master synchronicity depicts what thought and memory in the brain are. The building block of which (i.e. the neuron) is what set the start of a long-lasting facination with the topic among researchers.

A2 The Single-layer Perceptron

The early theoretical foundations of the field of Deep Learning were set on the idea of having a hardware-algorithm that is able to learn. Funded by the United States Army, Frank Rosenblatt introduced the world to the first utilisation of the Perceptron concept back in 1958, which he defined as a "hypothetical nervous system" **[15].** A single-layer perceptron (i.e. basic neuron) is in fact a linear classifier that separates an input into two categories with a straight line. This input is dealt with as a linear function that follows:

$$y = wx + b$$

Where w is the weight value of the link between the input x and the neuron. b is a bias term that allows more flexibility for the Activation Function. The then-newfound manifestation of the perceptron concept had several shortcomings. From the applicability perspective, it was very limited, mostly due to the fact that it only supports linear activations.

A3 Artificial Neural Networks

Numerous neuroscientists and computer scientists who understood the power of the human brain realised that introducing the world to an *artificial* form of this biological masterpiece could be anything but short of benefit — resulting in *Artificial Neural Networks* (ANNs). This artificial depiction actually comprises of interconnected perceptrons (namely, neurons). These neurons are separated into layers that feed into each other in a scheme that satisfies the network architecture followed. An artificial neuron is shown in *Figure A1*. Technically, any ANN that has more than one hidden layer follows the concept of Deep Learning. It is worth mentioning that a lot of people argue that the resemblance between biological and artificial neural networks stops at this point, emphasising on the fact that an

ANN only might look like any very small part of the brain, but doesn't necessarily act like it, attributing that to the fact that we still do not fully understand how the brain works in a thorough manner. Although it came at substantial computational and memory-related complexities, this *simple* utilisation of the idea of a perceptron has revolutionised the world of computing.

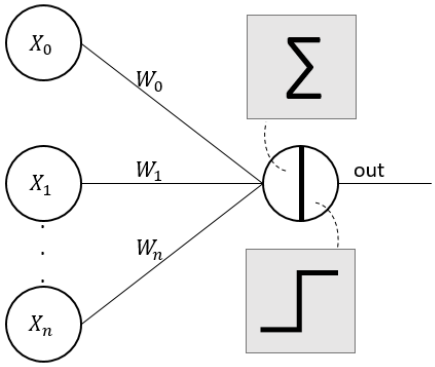


Figure A1: Single-layer perceptron

In Machine Learning, the objective is always to build something that generalises well **[16]** (i.e. making it able to make a *hypothesis* about new inputs that it has never seen before). The interconnectedness of neurons fixed the shortcoming mentioned in *A2*, by making the ANN able to solve nonlinear classification problems, broadening the plethora of applications that can utilise this concept. The idea lies behind that we can introduce some non-linear complexity to the algorithm and try perfecting —to an extent— the newly formed combination. Therefore, the formula we introduced earlier can be written as:

$$a_j = \sum_{i=0}^{D} w_{ji} \cdot x_i$$

Therefore, in *Figure A1*, the objective neuron takes input values that can be features or outputs from previous neurons. It calculates the objective function value (∑) and inputs the result into the activation function. The figure demonstrates that with the Threshold function. The output of the activation function determines the output of the neuron into the succeeding network components.

A4 Types of ANNs

ANNs hace different types and formations that depend on two things; network architecture and activations. Network architecture (i.e. topology) is how the neural network is designed. This includes the number of neurons/nodes, the number of layers and how those neurons are interconnected. Activations include the functions that neurons use to determine what output to feed into their neighboring neurons. Different types of ANNs are conceptually similar, as they follow the same theoretical background and require mostly the same knowledge. However, they are different in the way they are applied. For example, most image classification applications include a form of a Convolutional Neural Network (CNN) due to convolution's effectiveness in Image Processing as a field **[17].** On the other hand, Speech Recognition applications rely heavily on Recurrent Neural Networks (RNNs) as such applications require an architecture that represents sequential data patterns in a fruitful manner.

A5 Backpropagation

Since the essence of Machine Learning is to build a hypothesis function whose generalisation is sufficient, Deep Learning is no different. This generalisability of the model at hand comes from good training practice to start with. ANNs follow the same principle. However, training an ANN usually includes more steps than simple Machine Learning models. Since ANNs introduce nonlinearity in applications, they need a more complex learning algorithm: *Backpropagation*. The basic idea of backpropagation as a learning algorithm is the repeated application of the chain rule to compute the influence of each weight in the network with respect to an arbitrary error function **[18].** The way backpropagation operates is that it makes the ANN *learn* by "penalising" the neurons whose values relatively caused the largest portion of the total network error. This functionality, provided by the learning algorithm, returns (i.e. propagates, backwards) — in a sense — the error of the network to its corresponding neurons. Hence, the name — backpropagation. Simply put, the goal is always to achieve proper tuning of the weights to ensure lower error rates, making the model reliable by increasing its generalisation to new inputs.

A6 Activation Functions

Activations functions are what truly introduces nonlinearity into ANNs. In Deep Learning, the hypothesis function is extended to include an activation function that takes as input the sum of features X, weighted by the link weights in the network, and returns a value that gets passed onto the next network component. In principle, activation functions are chosen based on the application. Types of activation functions include Threshold, Sigmoid, Rectifier, Hyperbolic Tangent, etc. It is important to note that, whatever its type is, an activation function should always be differentiable, in order for training (backpropagation, more specifically) to work.

$$z_j = h(a_j)$$
$$= h(\sum_{i=0}^{D} w_{ji} \cdot x_i)$$

A7 Error Functions

An error rate (i.e. loss) is a measure of *how off* the model prediction is. In most applications, it is calculated as the difference between the model prediction and the actual output present in the data set. This follows the generic formula:

$$J(W) = y - \hat{y}$$

Where *J(W)* is the loss resulting from inference with weight vector *W*. y is the model's output and $\hat{y}$ is the expected output. Loss/error functions have different types, each of

which could give a different error for the same prediction. Choice of error functions highly depends on the problem at hand. For instance, Mean Squared Error function is frequently used in regression problems, and Cross Entropy is conventionally used in classification.

A8 Optimisation Algorithms

The loss of an inference is backpropagated through the model in order to tweak the weights associated with nodes, and the biases associated with layers. This requires an optimisation algorithm, such as *Gradient Descent (GD),* which includes calculating the gradient of the data set and updating the weights in the direction opposite to the gradient. In gradient descent, we can imagine the quality of our network's predictions (as a function of the weight/parameter values) as a landscape. The hills represent locations (parameter values or weights) that give a lot of prediction error; valleys represent locations with less error. We choose one point on that landscape at which to place our initial weight. We then can select the initial weight based on domain knowledge (if we're training a network to classify a flower species we know petal length is important, but color isn't). Or, if we're letting the network do all the work, we might choose the initial weights randomly.

The purpose is to move that weight downhill, to areas of lower error, as quickly as possible. An optimization algorithm like gradient descent can sense the actual slope of the hills with regard to each weight; that is, it knows which direction is down. Gradient descent measures the slope (the change in error caused by a change in the weight) and takes the weight one step toward the bottom of the valley. It does so by taking a derivative of the loss function to produce the gradient. The gradient gives the algorithm the direction for the next step in the optimization algorithm, as depicted in:

Algorithm A1: Gradient Descent

```
Input: Y, Θ, X, α, tolerance, max iterations
Output: Θ
  for i = 0; i < max iterations; i++ do
    current cost = Cost(Y,X, Θ)
    if current cost < tolerance then
      break
    else
      gradient = Gradient(Y,X, Θ)
      θ_j ← θ_j − α · gradient
```

Appendix B

Phonemic Similarity Measures

In Automatic Speech Recognition, an essential step that a lot of applications incorporate is *Grapheme-to-Phoneme (G2P)* or *Phoneme-to-Grapheme (P2G)* conversion. The former is usually directly involved in the building of ASR-prepared data sets. On the other hand, the latter is usually made use of in ASR applications themselves. When it comes to evaluating the similarity between two sequences of phonemes, a common measure is the *Phoneme Error Rate (PER).* ASR applications usually need such a metric in order to make an informed determination on the accuracy of the speech perceived by the application compared to the expected one residing in the training set. PER suffers from the inadueqacy of substitution cost representation, as it treats every difference between a pair of phonemes equally.

In this chapter, the most common phoneme matching approaches are scrutinised, to arrive to an understanding of the effectiveness of popular PER calculation methods. The need for deciding on an approach of such type arises from the fact that there needs to be a measure of determining the similarity between speech representations. This is where the actual similarity rate or percentage is calculated and presented, making the choice of a related algorithm crucial to the generalisation of the application of this project as a whole.

B1 Levenshtein Error Distance

Phoneme matching tasks are fairly similar to those of string matching. Both types of tasks include finding the error between two sequences of speech segments whose order is of great importance. One of the most commonly used algorithms in the context of string matching is the *Levenshtein Error Distance (LD)* **[50],** which takes a fairly different apporach. LD computation tries to match one label sequence with another by calculating the minimum number of one-symbol operations that lead to the two sequences being equal. Operations include additions, deletions and substitutions. LD yields a positive integer that is highly dependent on the string length. This means that matching the sequence [b, a, r] with the sequence [f, o, o] will actually result in the same LD value as matching [b, e, a, u, t, i, e, s] with [b, e, a, u, t, i, f, u, l]. However, getting the percentage of similarity would then be as simple as getting the inverse (1/LD) of the result.

B2 Phonological Edit Distance

Originally proposed by Sander et al. **[49],** *Phonological Edit Distance (PED)* follows the steps of LD in one aspect, but it takes a different approach in another. Similarity between the two is posed through the fact that PED depends on how many one-symbol changes it takes for one sequence to match the other. Notwithstandingly, the changes it takes are not

of equal importance, thus, PED uses a system of weighted changes that depend on featural similarity of sequence segments.

B3 Khorsi Similarity Metric

String matching through modification counting does not usually do justice to the importance of certain characters in some languages. For example, in languages that rely heavily on words that are a result of modifications performed on other words (i.e. *Inflectional Lnguages*), some letters in the word could have a greater importance than others. *Khorsi's* **[51]** delve into the topic resulted in a lightweight, unsupervised similarity metric that deals with all characters of a word — with different weights. The algorithm follows the formula:

$$\sum_{i=1}^{||LCS(w_1,w_2)||} log\left(\frac{1}{freq(LCS(w_1,w_2)[i])}\right) - \sum_{i=1}^{||LCS(w_1,w_2)||} log\left(\frac{1}{freq(\overline{LCS(w_1,w_2)}[i])}\right)$$

where,

w_1, w_2 are two words/strings

$LCS(w_1, w_2)$ is the Longest Common Shared Sequence of symbols between the two words.

As shown above, the method takes the inverse of the frequency of occurrence, and sums up the log corresponding to it, for every letter in the LCS between two strings. The method then takes the non-shared letters and subtracts the sum of the log of the inverse of the frequency of them from the preceeding value.

B4 Trigram Comparison

Phonemic similarity measurement approaches could include those that are dictionary-based. For instance, *N-gram* mathods include dependency on the different combinations of adjacent words or letters of length N. Trigram Comparison is a case of N-gram — a contiguous sequence of N items from a given sample. Even though computing a value for the similarity indicator using Trigram Comparison is as easy as dividing the number of matching trigrams in both strings by the number of unique trigrams, it suffers from the same issue usually faced with LD — strong dependency on string length. This limitation leads to lower similarity rates in short strings than those in long ones, when the strings have one or two different trigrams.

B5 Jaro-Winkler Distance

While methods like LD attempt to match one string to the other, the *Jaro-Winkler Distance* metric attempts to merely compare them, giving more weight to set prefixes **[52].** It follows the observation that differences between two strings in the early characters in the sequences are of more significance to the similarity-indicative value than those differences near the end of the sequences of characters. This bias towards the beginning might not work very well in phoneme recognition tasks (at least without being combined with another algorithm). This could be attributed to the fact that phonemes, being originally based on sounds rather than word segments — as is the case with syllables — might not provide direct transcriptions for the speech. In other words, sounds in the beginning of words do not necessarily provide a lot of information about the utterance as a whole. Therefore, always penalising errors in early characters could lead to an imbalance in results and in similarity measurement as a process.

Appendix C

Progress and Project Management

C1 Time Management

Although the progress was fluctuating throughout the first semester due to unexpected obstacles in the research stage regarding the actual implementability of speech comparison through only signal analysis, most of the tasks included in the original project proposal were executed successfully. However, this was at the expense of a couple of postponed tasks (namely, *B02* and *B03* in *Table C1*). On the other hand, the project was run fairly consistently with the plan in the second semester. *Table C1* showcases the task list breakdown throughout the academic year. Since the beginning of the project, the tasks were labelled based on their respective category of work. Tasks were labelled C for university *Commitment*, R for *Research*-based task, B for *Build* (i.e. implementation-related) and E for *Evaluate*. These tasks were kept track of using a note-taking program that also played the role of a brainstorming wall that is revised and updated regularly. *Figure C1* illustrates this stormboard.

Table C1: Project tasks list

#	Task Title	Projected End Date	Actual End Date	Status
C01	Complete Project Proposal	14-Oct-18	13-Oct-2018	Complete
R01	Review popular speech similarity measures	21-Oct-18	10-Nov-18	Complete
B01	Implement speech feature extraction	28-Oct-18	15-Nov-18	Complete
B02	Implement speech comparison	04-Nov-18	Postponed till the second semester	
B03	Design and implement mobile app	11-Nov-18		
R02	Research speech signal segmentation	25-Nov-18	25-Nov-18	Complete
B05	Implement speech signal segmentation	02-Dec-18	09-Dec-18	Complete
C02	Write Interim Report	15-Dec-18	10-Dec-18	Complete
C03	Study break	20-Jan-19	18-Jan-19	Complete
E01	Evaluate model performance	27-Jan-19	18-Apr-19	Complete
B02	Implement speech comparison	11-Feb-19	12-Dec-18	Complete
B03	Design and implement mobile app	20-Feb-19	09-Feb-19	Complete
E02	Test app	27-Mar-19	17-Apr-19	Complete
B05	Fix leftover bugs	27-Mar-19	15-Apr-19	Complete
E03	Final implementation touches	31-Mar-19	17-Apr-19	Complete
C04	Write Final Report	18-Apr-19	17-Apr-19	Complete

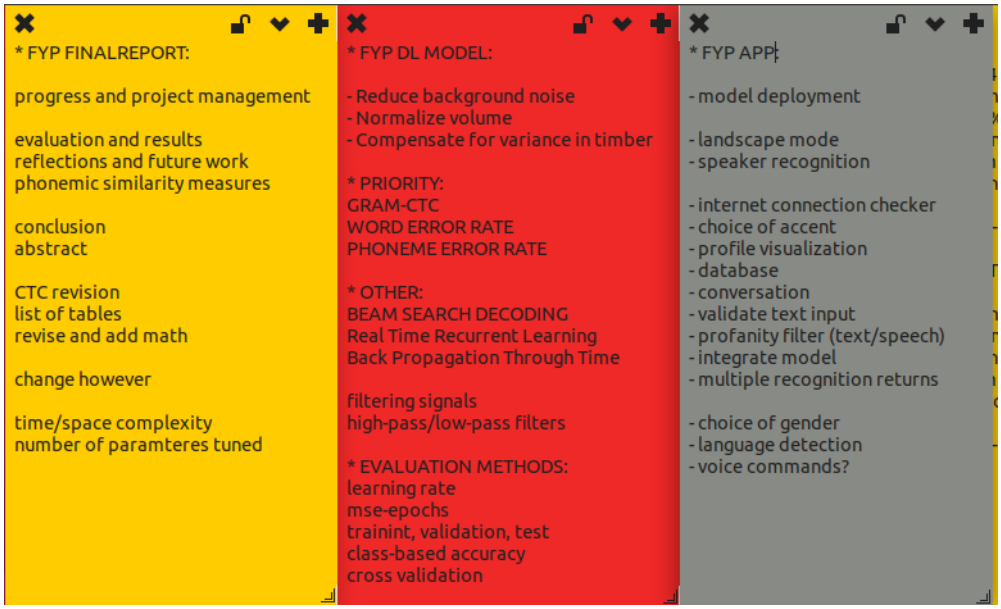


Figure C1: Project Stormboard

The chart in *Figure C2* demonstrates the overall work timeline compared to expected durations. It is visible how the project was split into equal segments (7-weeks each) for most tasks. This made the project easier to track, as most weeks had only one major objective.

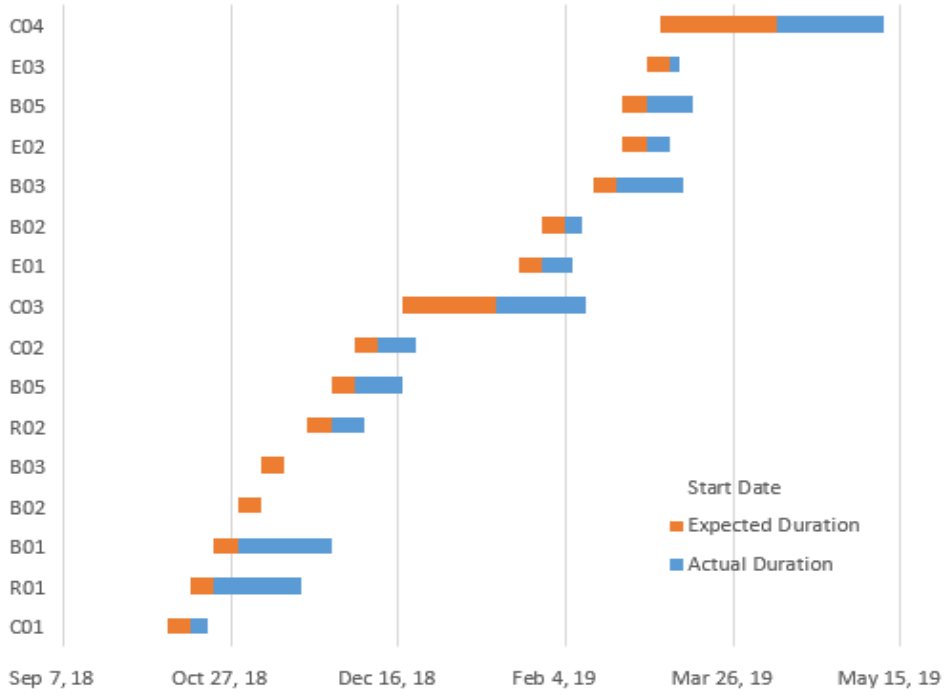


Figure C2: Work Timeline Gantt Chart

As mentioned earlier, most performance fluctuations took place in the first semester rather than the second. This is apparent in *Figure C3*. Although the actual remaining days' work was higher than the expected one at most times, it was always close to the optimal timeline

set from the beginning. It is worth mentioning that the project took about 30% more time than expected, mostly due to unexpected issues such as the model deployment matter discussed in *Chapter 5*, section *5*.

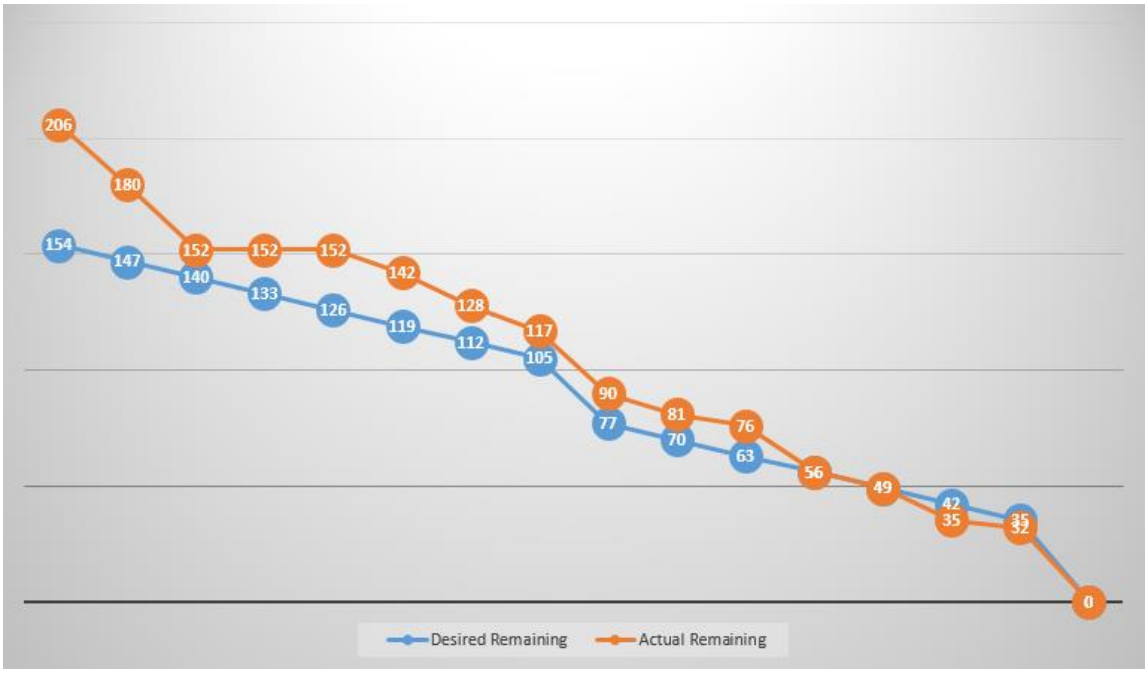


Figure C3: Workload Burndown Chart (in days)

C2 Resource Management

The research aspect of the project started early. In fact, references were being bookmarked and managed since July 2018 using the *Mendeley* software (shown in *Figure C4*), which provides an inter-operable environment that can be used through Web browsers as well as Desktop applications. The need for a research management platform arose further along the way, where there needed to be made links among some key research papers in the field. Having an all-platform research repository definitely made the difference here.

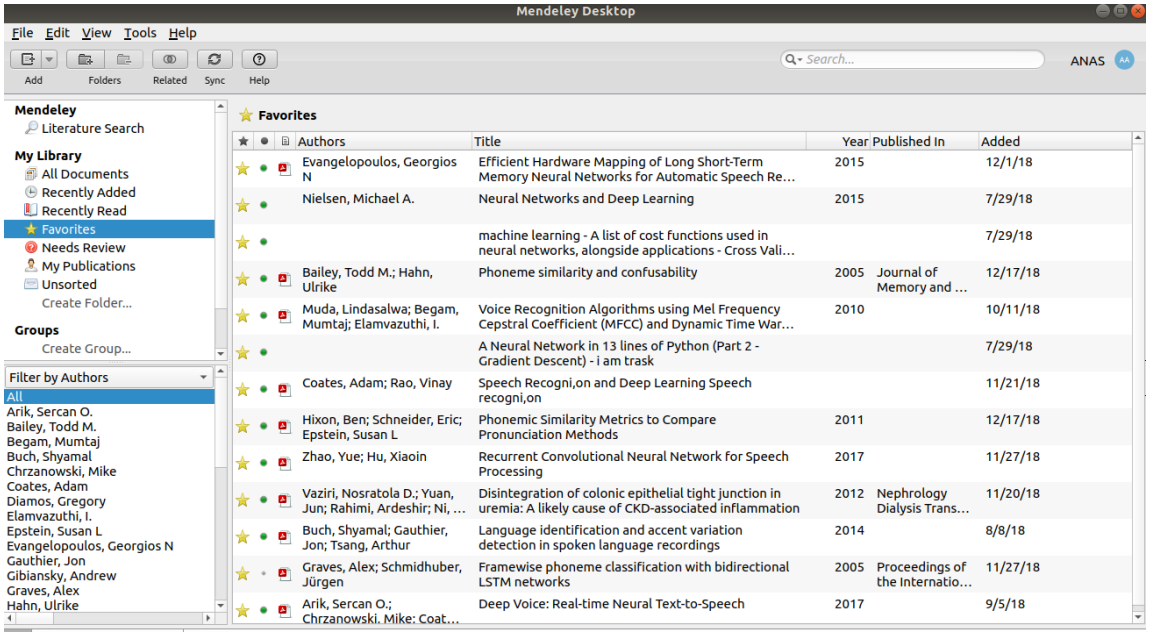


Figure C4: Research management platform (Mendeley)

From another perspective, source code also needed to be kept safe at all times. Throughout the development of this project, the source code was pushed onto two private *GitHub* repositories — one for the model code and one for the mobile app (shown in *Figure C5*). The model repository was linked straight to Visual Studio Code, while the mobile app's repository was linked to Android Studio, as it was the main Integrated Development Environment used.

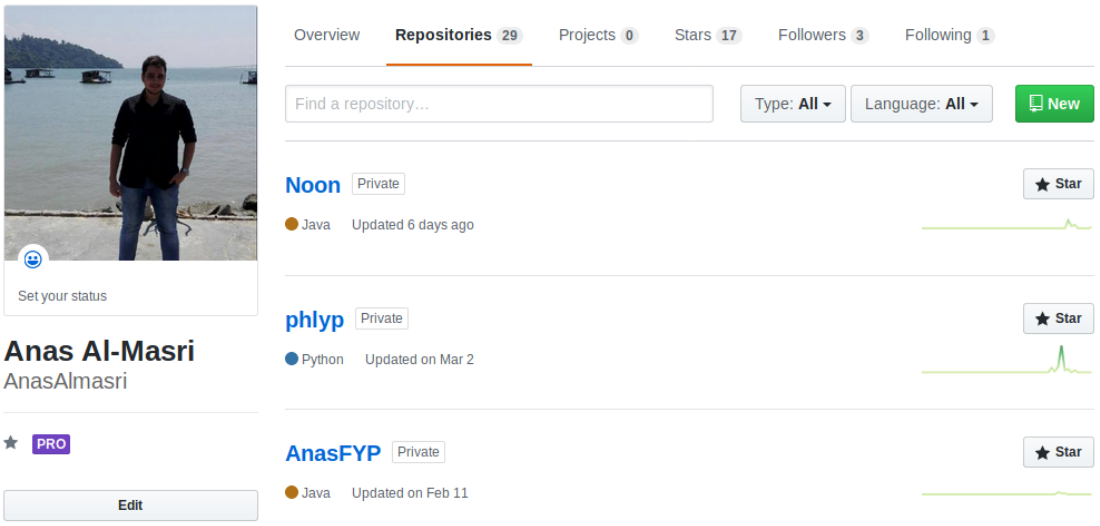
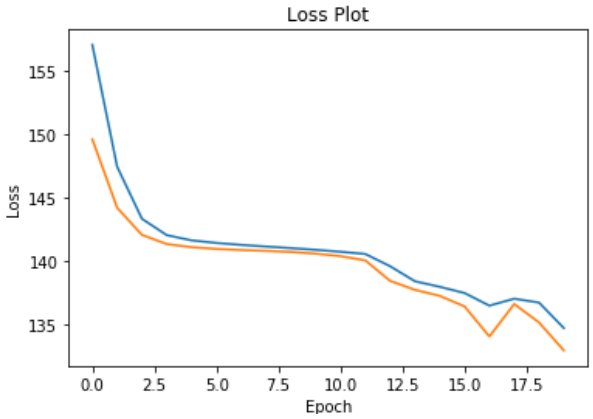


Figure C5: Involved GitHub Repositories

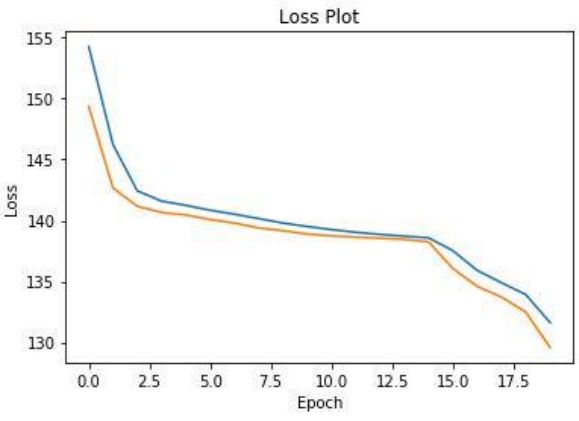
Appendix D

Experimental Trials

EXP01

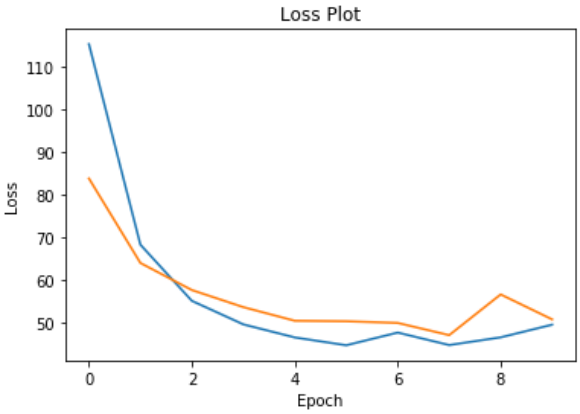
Hidden layers	5	Training accuracy	8%
Nodes per hidden layer	20	Validation accuracy	8%
MFCC feature vectors	26	Testing accuracy	8%
logEnergy	True	Average Precision	0.112
Loss Function	CTC	Average Recall	0.073
Epochs	20	F1 Score	0.088
Batch size	30	*Training — blue. Validation — orange.*	
Batches per epoch	739		
Data split (Train. - Val. - Test.)	(75 - 15 - 10)		
Machine specs	CPU / 3.4GHz / 8GB RAM		
Training duration	3 hours		
Mean training loss	140.8		
Mean training Label Error Rate	0.92		
Mean testing loss	132.9		
Mean testing Label Error Rate	0.92		

EXP02

Hidden layers	7	Training accuracy	9%
Nodes per hidden layer	20	Validation accuracy	10%
MFCC feature vectors	26	Testing accuracy	8%
logEnergy	True	Average Precision	0.094
Loss Function	CTC	Average Recall	0.107
Epochs	20	F1 Score	0.101
Batch size	20	*Training — blue. Validation — orange.*	
Batches per epoch	1108		
Data split (Train. - Val. - Test.)	(75 - 15 - 10)		
Machine specs	CPU / 3.4GHz / 8GB RAM		
Training duration	4 hours		
Mean training loss	132.5		
Mean training Label Error Rate	0.91		
Mean testing loss	129.6		
Mean testing Label Error Rate	0.92		

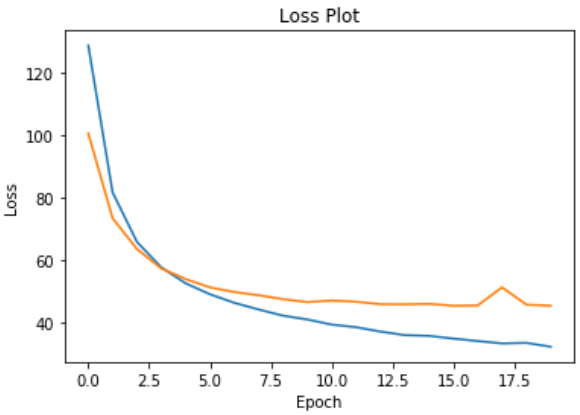
EXP03

Hidden layers	4	Training accuracy	65%
Nodes per hidden layer	100	Validation accuracy	62%
MFCC feature vectors	26	Testing accuracy	61%
logEnergy	True	Average Precision	0.689
Loss Function	CTC	Average Recall	0.562
Epochs	10	F1 Score	0.619
Batch size	10	*Training — blue. Validation — orange.*	
Batches per epoch	2217		
Data split (Train. - Val. - Test.)	(75 - 15 - 10)		
Machine specs	CPU / 3.4GHz / 8GB RAM		
Training duration	9 hours		
Mean training loss	42.3		
Mean training Label Error Rate	0.35		
Mean testing loss	50.8		
Mean testing Label Error Rate	0.39		



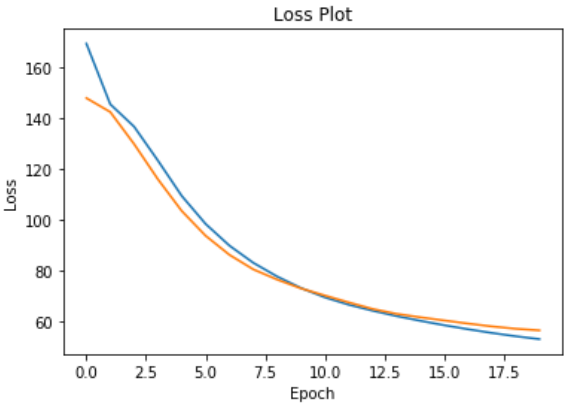
EXP04

Hidden layers	2	Training accuracy	78%
Nodes per hidden layer	100	Validation accuracy	74%
MFCC feature vectors	26	Testing accuracy	66%
logEnergy	True	Average Precision	0.803
Loss Function	CTC	Average Recall	0.551
Epochs	20	F1 Score	0.654
Batch size	20	*Training — blue. Validation — orange.*	
Batches per epoch	1108		
Data split (Train. - Val. - Test.)	(75 - 15 - 10)		
Machine specs	CPU / 3.4GHz / 8GB RAM		
Training duration	10 hours		
Mean training loss	23.1		
Mean training Label Error Rate	0.22		
Mean testing loss	45.3		
Mean testing Label Error Rate	0.34		



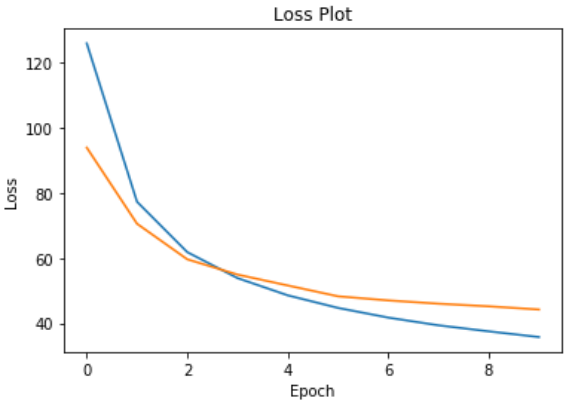
EXP05

Hidden layers	2	Training accuracy	61%
Nodes per hidden layer	75	Validation accuracy	56%
MFCC feature vectors	26	Testing accuracy	57%
logEnergy	True	Average Precision	0.414
Loss Function	CTC	Average Recall	0.509
Epochs	20	F1 Score	0.457
Batch size	128	*Training — blue. Validation — orange.*	
Batches per epoch	173		
Data split (Train. - Val. - Test.)	(75 - 15 - 10)		
Machine specs	CPU / 3.4GHz / 8GB RAM		
Training duration	10 hours		
Mean training loss	51.68		
Mean training Label Error Rate	0.39		
Mean testing loss	56.39		
Mean testing Label Error Rate	0.43		



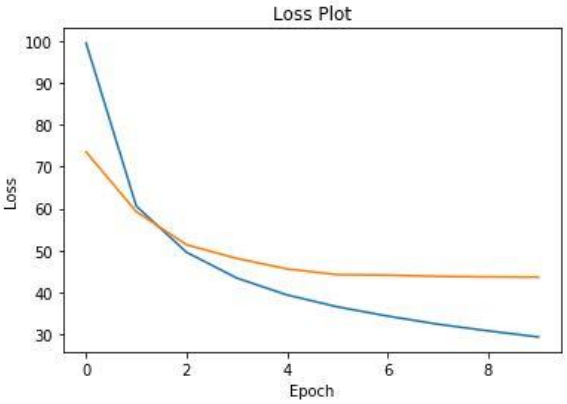
EXP06

Hidden layers	2	Training accuracy	67%
Nodes per hidden layer	150	Validation accuracy	71%
MFCC feature vectors	26	Testing accuracy	66%
logEnergy	True	Average Precision	0.643
Loss Function	CTC	Average Recall	0.589
Epochs	10	F1 Score	0.615
Batch size	20	*Training — blue. Validation — orange.*	
Batches per epoch	1108		
Data split (Train. - Val. - Test.)	(75 - 15 - 10)		
Machine specs	CPU / 3.4GHz / 8GB RAM		
Training duration	5 hours		
Mean training loss	43.5		
Mean training Label Error Rate	0.33		
Mean testing loss	44.23		
Mean testing Label Error Rate	0.34		



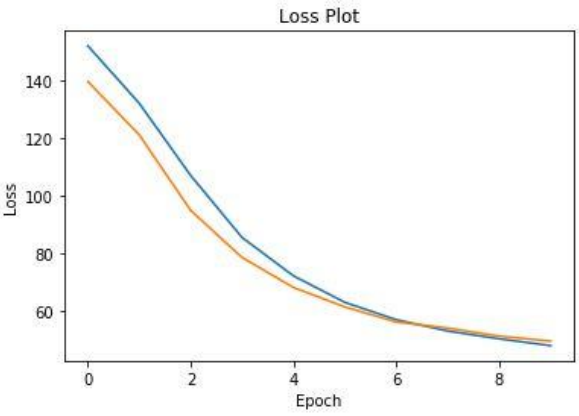
EXP07

Hidden layers	2	Training accuracy	69%
Nodes per hidden layer	150	Validation accuracy	76%
MFCC feature vectors	26	Testing accuracy	67%
logEnergy	True	Average Precision	0.695
Loss Function	CTC	Average Recall	0.721
Epochs	10	F1 Score	0.708
Batch size	10	*Training — blue. Validation — orange.*	
Batches per epoch	2217		
Data split (Train. - Val. - Test.)	(75 - 15 - 10)		
Machine specs	CPU / 3.4GHz / 8GB RAM		
Training duration	9 hours		
Mean training loss	38.32		
Mean training Label Error Rate	0.31		
Mean testing loss	43.61		
Mean testing Label Error Rate	0.33		

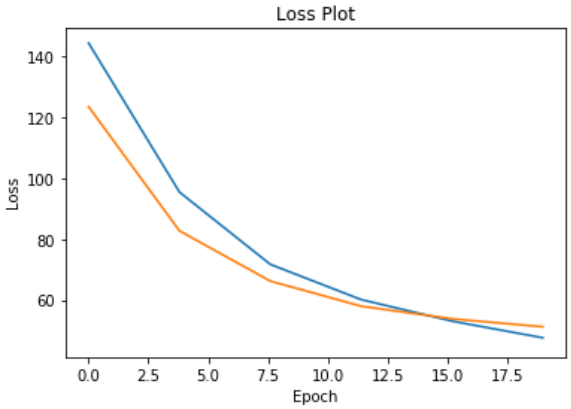


EXP08

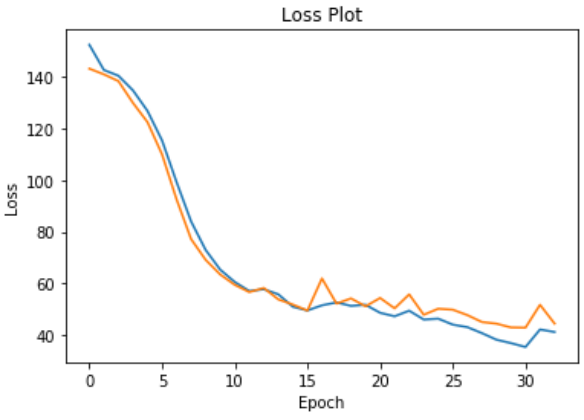
Hidden layers	3	Training accuracy	64%
Nodes per hidden layer	75	Validation accuracy	62%
MFCC feature vectors	26	Testing accuracy	61%
logEnergy	True	Average Precision	0.536
Loss Function	CTC	Average Recall	0.339
Epochs	10	F1 Score	0.415
Batch size	32	*Training — blue. Validation — orange.*	
Batches per epoch	692		
Data split (Train. - Val. - Test.)	(75 - 15 - 10)		
Machine specs	CPU / 3.4GHz / 8GB RAM		
Training duration	2 hours		
Mean training loss	43.89		
Mean training Label Error Rate	0.36		
Mean testing loss	49.61		
Mean testing Label Error Rate	0.39		



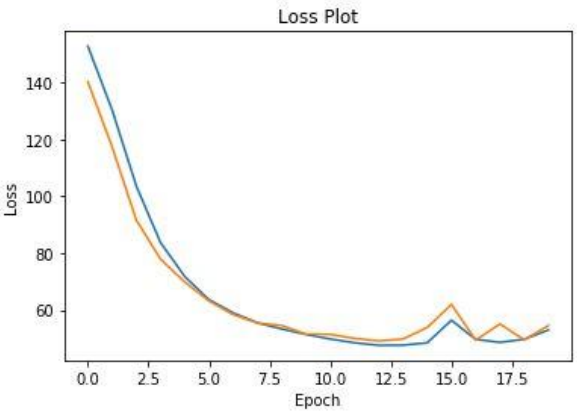
EXP09

Hidden layers	2	Training accuracy	61%
Nodes per hidden layer	200	Validation accuracy	58%
MFCC feature vectors	26	Testing accuracy	60%
logEnergy	True	Average Precision	0.376
Loss Function	CTC	Average Recall	0.449
Epochs	20	F1 Score	0.386
Batch size	25	*Training — blue. Validation — orange.*	
Batches per epoch	887		
Data split (Train. - Val. - Test.)	(75 - 15 - 10)		
Machine specs	CPU / 3.4GHz / 8GB RAM		
Training duration	23 hours		
Mean training loss	47.63		
Mean training Label Error Rate	0.39		
Mean testing loss	51.25		
Mean testing Label Error Rate	0.40		

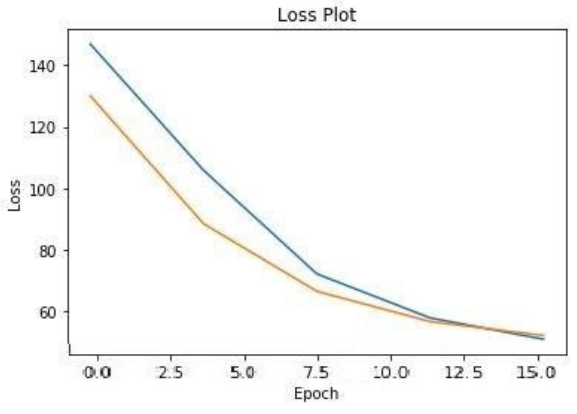
EXP10

Hidden layers	4	Training accuracy	67%
Nodes per hidden layer	100	Validation accuracy	63%
MFCC feature vectors	26	Testing accuracy	66%
logEnergy	True	Average Precision	0.866
Loss Function	CTC	Average Recall	0.716
Epochs	35	F1 Score	0.784
Batch size	32	*Training — blue. Validation — orange.*	
Batches per epoch	692		
Data split (Train. - Val. - Test.)	(75 - 15 - 10)		
Machine specs	CPU / 3.4GHz / 8GB RAM		
Training duration	26 hours		
Mean training loss	41.19		
Mean training Label Error Rate	0.33		
Mean testing loss	43.06		
Mean testing Label Error Rate	0.34		

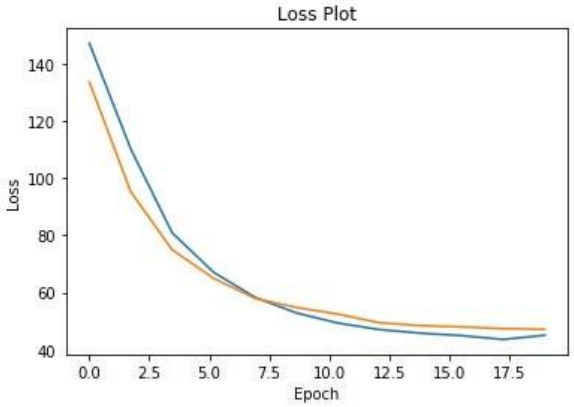
EXP11

Hidden layers	3	Training accuracy	64%
Nodes per hidden layer	75	Validation accuracy	59%
MFCC feature vectors	26	Testing accuracy	58%
logEnergy	True	Average Precision	0.644
Loss Function	CTC	Average Recall	0.492
Epochs	20	F1 Score	0.557
Batch size	32	*Training — blue. Validation — orange.*	
Batches per epoch	692		
Data split (Train. - Val. - Test.)	(75 - 15 - 10)		
Machine specs	CPU / 3.4GHz / 8GB RAM		
Training duration	4.5 hours		
Mean training loss	45.09		
Mean training Label Error Rate	0.36		
Mean testing loss	54.46		
Mean testing Label Error Rate	0.42		

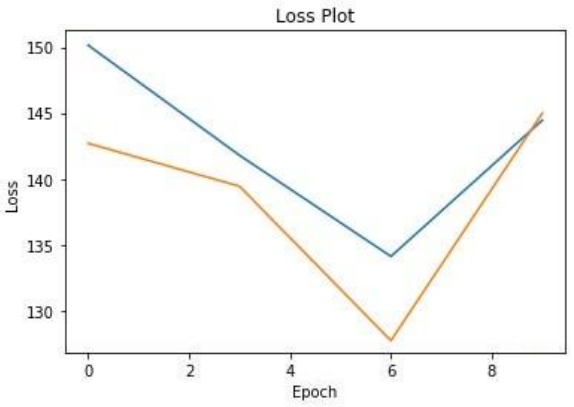
EXP12

Hidden layers	3	Training accuracy	60%
Nodes per hidden layer	150	Validation accuracy	56%
MFCC feature vectors	26	Testing accuracy	59%
logEnergy	True	Average Precision	0.719
Loss Function	CTC	Average Recall	0.536
Epochs	15	F1 Score	0.614
Batch size	25	*Training — blue. Validation — orange.*	
Batches per epoch	1108		
Data split (Train. - Val. - Test.)	(75 - 15 - 10)		
Machine specs	CPU / 3.4GHz / 8GB RAM		
Training duration	9 hours		
Mean training loss	50.78		
Mean training Label Error Rate	0.40		
Mean testing loss	52.09		
Mean testing Label Error Rate	0.41		

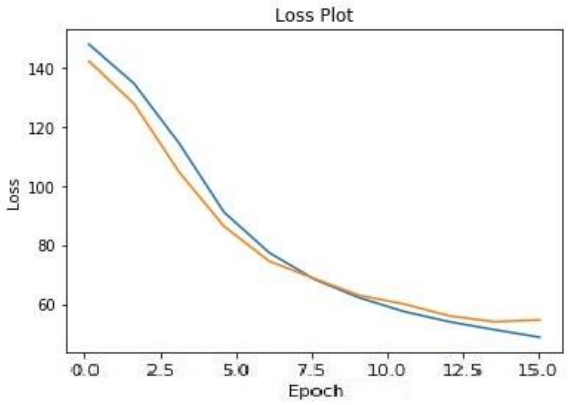
EXP13

Hidden layers	3	Training accuracy	65%
Nodes per hidden layer	100	Validation accuracy	64%
MFCC feature vectors	26	Testing accuracy	61%
logEnergy	True	Average Precision	0.633
Loss Function	CTC	Average Recall	0.681
Epochs	20	F1 Score	0.656
Batch size	20	*Training — blue. Validation — orange.*	
Batches per epoch	1108		
Data split (Train. - Val. - Test.)	(75 - 15 - 10)		
Machine specs	CPU / 3.4GHz / 8GB RAM		
Training duration	6.5 hours		
Mean training loss	49.96		
Mean training Label Error Rate	0.35		
Mean testing loss	53.77		
Mean testing Label Error Rate	0.39		

EXP14

Hidden layers	4	Training accuracy	5%
Nodes per hidden layer	150	Validation accuracy	5%
MFCC feature vectors	26	Testing accuracy	7%
logEnergy	True	Average Precision	0.031
Loss Function	CTC	Average Recall	0.007
Epochs	10	F1 Score	0.011
Batch size	20	*Training — blue. Validation — orange.*	
Batches per epoch	1108		
Data split (Train. - Val. - Test.)	(75 - 15 - 10)		
Machine specs	CPU / 3.4GHz / 8GB RAM		
Training duration	21 hours		
Mean training loss	144.45		
Mean training Label Error Rate	0.95		
Mean testing loss	144.99		
Mean testing Label Error Rate	0.93		

EXP15

Hidden layers	4	Training accuracy	62%
Nodes per hidden layer	75	Validation accuracy	57%
MFCC feature vectors	26	Testing accuracy	57%
logEnergy	True	Average Precision	0.525
Loss Function	CTC	Average Recall	0.599
Epochs	15	F1 Score	0.559
Batch size	15	*Training — blue. Validation — orange.*	
Batches per epoch	1478		
Data split (Train. - Val. - Test.)	(75 - 15 - 10)		
Machine specs	CPU / 3.4GHz / 8GB RAM		
Training duration	13 hours		
Mean training loss	48.80		
Mean training Label Error Rate	0.38		
Mean testing loss	54.66		
Mean testing Label Error Rate	0.43		