Duke University

EGR Independent Study

Speech Processing and Recognition

Madeline Briere

supervised by
Dr. Michael Gustafson

April 25, 2017

Contents

1 Abstract **3**

2 Project Goals **4**

3 Concept Review **5**
- 3.1 Speech as a Signal . . . 5
  - 3.1.1 Speech Production . . . 5
  - 3.1.2 Model of Speech Production . . . 6
- 3.2 Speech Perception . . . 8
  - 3.2.1 Auditory Anatomy . . . 8
  - 3.2.2 Mel-Frequency Scale . . . 8
- 3.3 Audio Sampling . . . 10
- 3.4 Feature extraction . . . 10
  - 3.4.1 Pre-emphasis . . . 11
  - 3.4.2 Windowing . . . 12
  - 3.4.3 Fourier Transform . . . 13
  - 3.4.4 Filter-Bank Analysis . . . 13
  - 3.4.5 Discrete Cosine Transform . . . 14
  - 3.4.6 Cepstral Analysis . . . 15
  - 3.4.7 Liftering . . . 17
- 3.5 Speech Recognition . . . 18
  - 3.5.1 Hidden Markov Model . . . 18
  - 3.5.2 Artificial Neural Networks . . . 18
  - 3.5.3 Mapping MFCCs to Phones . . . 20
  - 3.5.4 Matching Phonetic Groupings to Words . . . 21

4 Signal Processing **22**
- 4.1 Relevant Resources . . . 22
  - 4.1.1 Exploring MATLAB's Speech Processing Capabilities . . . 22
  - 4.1.2 External Libraries & Similar Projects . . . 24
- 4.2 Developing an MFCC system . . . 25
  - 4.2.1 Pre-emphasis . . . 25
  - 4.2.2 Windowing . . . 26
  - 4.2.3 Fourier Transform . . . 27
  - 4.2.4 Filter-Bank Analysis . . . 28
  - 4.2.5 Mel-Frequency Cepstral Coefficients . . . 30

5 Speech Recognition **33**
- 5.1 External Libraries & Similar Projects . . . 33
- 5.2 Developing a Speech Recognition System . . . 34
  - 5.2.1 Recognition of Vowels via Correlation . . . 34
  - 5.2.2 Recognition of All Sounds via CNN . . . 37

6 Conclusion **39**
6.1 Remaining Challenges 39
6.2 Future Goals 40

7 Appendix **41**
7.1 Basic Demo, *basicdemo.m* 41
7.2 Frame, *frame.m* 42
7.3 Enframe, *enframe.m* 43
7.4 Frame Demo, *framedemo.m* 44
7.5 RFFT, *rfft.m* 45
7.6 Filterbank, *filterbank.m* 46
7.7 Mel Bank, *melbankm.m* 47
7.8 Melcepst, *melcepst.m* 50
7.9 RDCT, *rdct.m* 51
7.10 Lifter, *lifter.m* 52
7.11 Av Norm Cep, *avnormcep.m* 52
7.12 Plot Ceps, *plotceps.m* 53
7.13 Plot Ave Ceps, *plotaveceps.m* 55
7.14 Predict Vowel, *predictvowel.m* 56

1 Abstract

Speech recognition, the use of machines to translate speech into digitized text, has grown incredibly efficient and accurate in recent years. Current techniques allow us to decompose speech signals into feature vectors representative of windowed frequency content and match these feature vectors (both individual and in combination) to known patterns (e.g., phones, words, phrases). In this paper, we explore the use of *Mel-Frequency Cepstral Analysis* for feature extraction. Using the result Mel-Frequency Cepstral Coefficients (MFCCs), we explore techniques for speech recognition such as basic linear correlation (for basic vowel matching). We demonstrate a vowel prediction accuracy of approximately 87.5 percent using MFCCs and basic linear correlation (attributing a large portion of the remaining error to an extremely limited training set). Finally, we explore modes of speech recognition that could vastly increase accuracy and efficiency (Artificial Neural Networks; Convolutional Neural Networks).

2 Project Goals

The main goal of this project is to develop a unique system for speech processing and recognition, that is both *efficient* and *accurate*. Modern speech recognition engines have attained both these characteristics in the past several years, using complex models such as deep neural networks to attain high accuracy.

The goal for our product is a lightweight and minimal system – we need to maximize accuracy for a small state space of input options (geared towards the client). The user should be able to be able to request something within a small state-space of options and receive feedback accordingly with high accuracy. For instance, if the client were using such a system in a car shop, requests such as "purchase a Chevy Malibu A/C Compressor" should be parsed and carried out (e.g., using Amazon). Hence, we must minimize run time so that the system can be used without inconvenience (for instance, a response time of 10 seconds would be non-viable).

The required steps for this project can be boiled down to the following subcategories:

1. Process a speech signal in small windows, producing feature vectors representative of the content in these frames of time
2. Develop a model to recognize and categorize these feature vectors

The first goal, speech processing, requires several individual steps to be successful. These components can be summarized as follows:

1. Convert from analog (voice input) to digital (a speech signal)
2. Read in a WAV file and process it (derive sampling frequency, data points, etc.)
3. Pre-process the signal (e.g., emphasize higher frequencies that are minimized by human anatomy)
4. Window the signal (cut the signal into small and meaningful units of data)
5. Transform the data in the frequency domain to produce Mel-Cepstral Coefficients (discussed later)
6. Lifter (similar to filtering) the MFCCs in order to cut out individual-based content (e.g., pitch).
7. Normalize the resulting MFCCs (remove volume as a factor)

The second goal, developing and training a neural net, can be sub-categorized as follows:

1. Choose a model for use in this project (either a conventional system like the Hidden Markov Model or a more complex system such as a Convolutional Neural Network)
2. Train/build this neural network using MFCCs from downloaded and/or produced training data
3. Test this network to ensure efficiency and accuracy

Similar to the workflow of this project, this paper will be split into two chunks: one for speech processing and one for speech recognition via neural network.

3 Concept Review

3.1 Speech as a Signal

3.1.1 Speech Production

Speech, in its most basic form, is an audible *signal*, generated via the vocal chords, glottis, epiglottis, lungs, etc. A commonly accepted definition is as follows: "Speech is produced by air-pressure waves emanating from the mouth and the nostrils of a speaker," [9]. These waves oscillate in unique patterns, with each chunk of sound holding a different contextual and linguistic meaning. Speech is informed by the human speech anatomy (Figure 1).

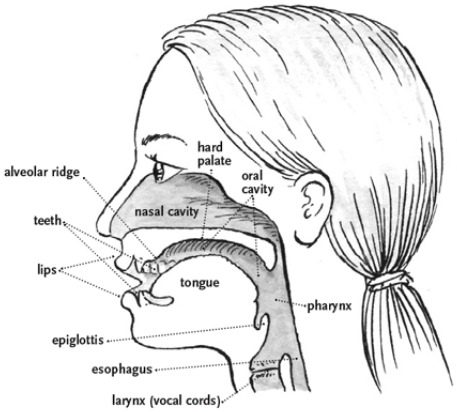

Figure 1: Anatomy of the mouth, [4]

These components are utilized in various ways to produce unique sounds. Simple combinations of tongue placement in the mouth, lip shape, activation of

the vocal chords, etc. produce all of the sounds known to human language

Formants, the main resonant modes of the vocal tract, can be used to classify a speech signal. With most sounds, the first two formants are considered more relevant. Consider, for instance, the formant chart for vowels seen in Figure 2.

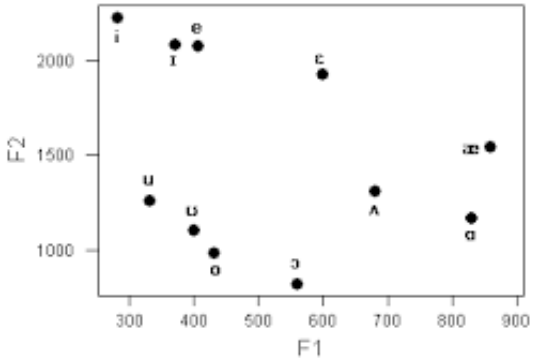

Figure 2: Vowel Formant Chart, [6]

Vowels (produced with an open vocal tract and relatively little audible friction) are characterized on an X and Y axis corresponding to formant 1 (F1) and formant 2 (F2).

3.1.2 Model of Speech Production

As mentioned in the previous section, many different components contribute to a single speech signal. The model for this process, however, is often a simple source-filter model, with an excitation signal passing through a filter (representing all of the components previously discussed) to produce the speech signal (as seen in Figure 3).

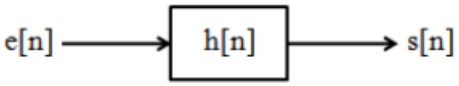

Figure 3: Source-filter model, [9]

One category of speech is *voiced*, meaning that the vocal chords are tensed, vibrating to produce a quasi-periodic waveform. The excitation signal for this type of sound can be viewed as an impulse train convolved with a glottal pulse (Figure 4). The logical "opposite" of a voiced sound is an *unvoiced* sound. An unvoiced sound is one in which the vocal chords of the speaker are not vibrating

and the output is not periodic. Instead, the input for unvoiced sound is random noise.

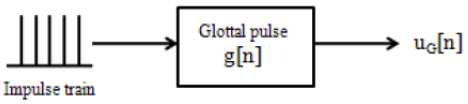

Figure 4: Excitation signal for voiced sound, [9]

The excitation signal in the source-filter model depicted prior incorporates both voiced and unvoiced sound. We can view this duality as a "switch" in the model, allowing for both voiced and unvoiced sounds as input.

Our complete model of speech production (Figure 5) must also include factors introduced by the speech production anatomy (discussed previously). For instance, we must consider the influence of the vocal tract "filter" (represented as V(jω)) and the influence of air pressure at the lips (R(jω)).

We also assume that both voiced and unvoiced sounds will be multiplied by a gain (G) that represents volume (amplitude of signal). This will differentiate between a shout and a whisper.

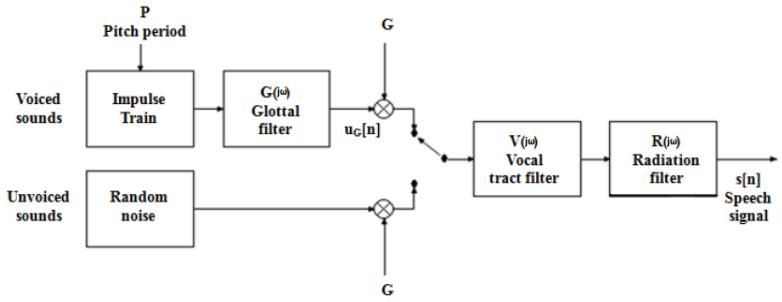

Figure 5: Complete model for speech, [9]

In modeling speech as a signal, we can wrap all of these factors (G(jω), V(hω), R(jω)) into a single filter, H(jω). The resulting source-filter model (Figure 6) will be used for the duration of this paper.

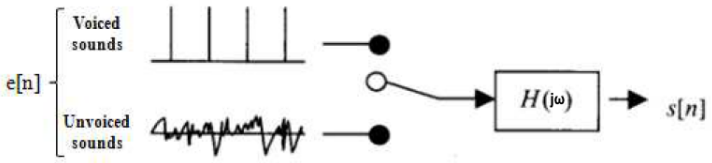

Figure 6: Source-filter model for speech, [9]

3.2 Speech Perception

The next relevant concept in developing a speech processing system is *speech perception*. Because the human speech generation system is specifically built to be "decoded" by another human, models for speech recognition (at least this project model, given its purpose) should be informed by *human speech perception*.

3.2.1 Auditory Anatomy

The human auditory system can be split into two sub-components: the the auditory nervous system (i.e., the brain) and the the peripheral auditory system (i.e., the ears). Sound signals enter the ear and are converted from acoustic pressure signals to neural signals (transmitted via the auditory nerve to the brain). These neural signals are then interpreted by the brain.

The ear is split into outer, middle and inner. Sound travels into the outer ear and is funneled towards the tympanic membrane (i.e., the eardrum). Vibrations of this membrane trigger parallel vibrations in the bones of the middle ear, called ossicles. These tiny bones amplify the noise and pass the sound waves to the inner ear, into the cochlea.

The anatomy of the cochlea is of most relevance in this project. The cochlea, a fluid-filled hearing organ, contains the nerves for hearing. The cochlea accomplishes *frequency-to-place transformation*. In other words, higher frequencies are received by the cochlear base, while lower frequencies are received by the cochlear apex. This unique mapping, which parallels a filter bank (in that is registers frequencies in select bands), has informed several different scales for frequency, such as the *Mel-frequency scale*.

3.2.2 Mel-Frequency Scale

The scale favored in this paper will be the Mel-frequency scale, which is linear below 1kHz and logarithmic above 1kHz. This system "represents the pitch (perceived frequency) of a tone as a function of its acoustic frequency," [9].

The equation for transformation from Hertz to Mels is approximately cited by Equation 1:

$$M(f) = 2595log_{10}(1 + \frac{f}{700}) \tag{1}$$

The conversion from Hertz to Mels can be seen in Figure 7.

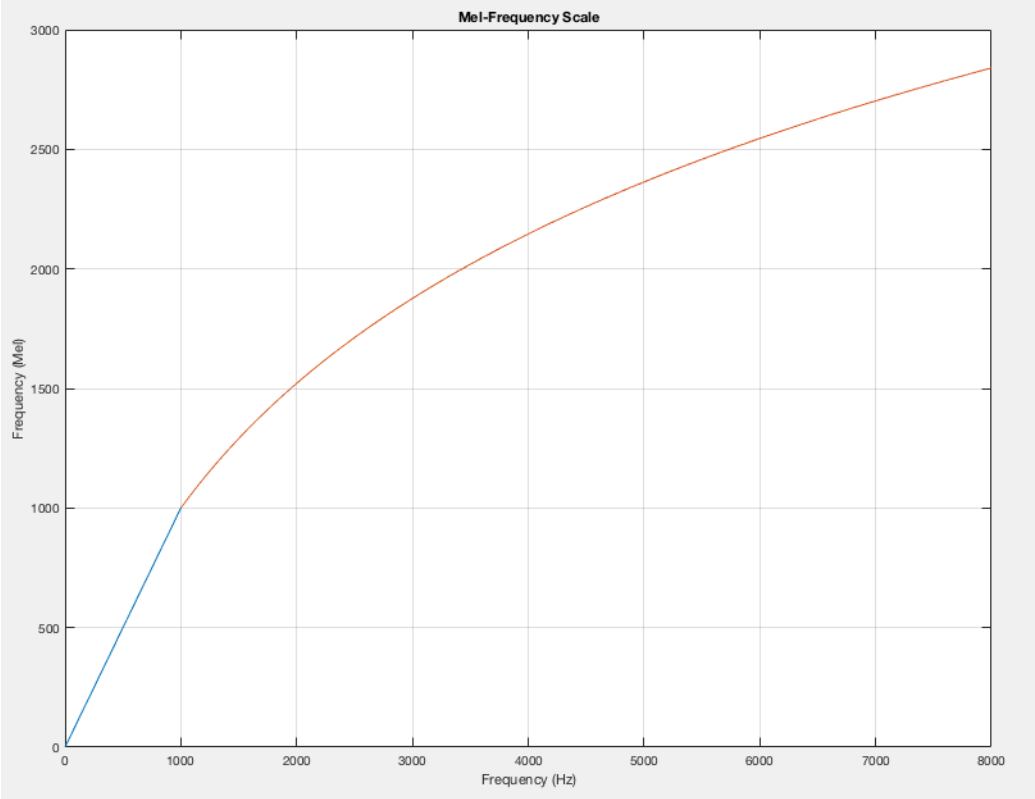

Figure 7: Mel-Frequency Scale

3.3 Audio Sampling

In this project, all speech signals are recorded in the form of a WAV file, or a Waveform Audio File. It must be noted that this process does not capture *all* of the audio information encapsulated in the recorded waveform. Instead, it collects samples at a given sampling frequency to create a discrete representation of the original signal – this signal can be encoded digitally and processed (see Figure 8).

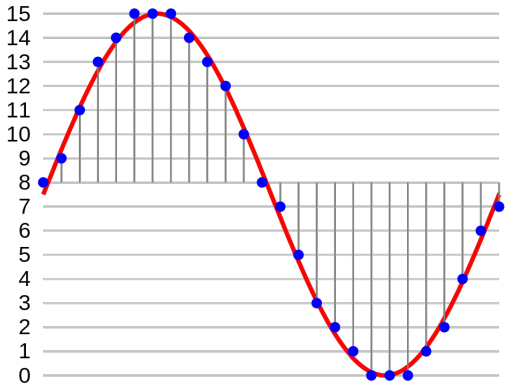

Figure 8: Sampling a waveform

The sampling frequency used throughout this project will typically be 44,100Hz. This is a very common sampling frequency (it is used on audio CDs), as it satisfied the Nyquist criterion for (i.e., allows for accurate sampling of) frequencies up to approximately 20,000Hz, which pushes the limits of human hearing. Lower sampling rates, such as 8,000Hz (the standard telephone sampling rate), are not able to capture the nuances of human speech. For instance, "f" and "s" sounds are blurred at a sampling rate of 8,000Hz. Clearly, for the purposes of this project, a higher sampling rate is necessary for accurate prediction.

3.4 Feature extraction

The type of signal processing relevant to speech recognition is *feature extraction*. Feature extraction allows us to isolate individual frames of sampled sound and represent their content as vectors. These vectors can then be compared and modified for further analysis.

The type of extraction accomplished in this project produces a feature vector of *Mel-Frequency Cepstral Coefficients* (MFCCs) for each audio frame. MFCCs are used in automated telephone systems to record spoken numbers (e.g., pre-

scription numbers, telephone numbers). MFCCs have several benefits (motivating their use in this project). MFCCs:

1. Are relatively simple to calculate in comparison to other signal processing techniques used for speech recognition.
2. Are calculated using the Mel-Frequency scale, meaning that the results are perceptually motivated.
3. Produce a simple representation of a complicated window of audio data (often only 12 coefficients for an entire 30ms chunk).
4. Minimize speaker-related factors such as gender and volume, allowing for higher accuracy.

There are other feature vectors that may have been relevant in this project (e.g., Linear Predictive Coding Coefficients), but MFCCs are the most well-documented and offer the benefits mentioned above.

3.4.1 Pre-emphasis

Pre-emphasis is the first step to audio processing. During speech production, some of the higher frequency content is suppressed (as a result of the human anatomy). In order to restore this content, we must apply a High-Pass Finite Impulse Response (FIR) filter to the signal in order to re-emphasize these higher frequencies (see the transfer function in Equation 2).

$$H(j\omega) = 1 - \frac{a_{preem}}{j\omega} \tag{2}$$

The transfer function for this filter can be seen in Figure 9.

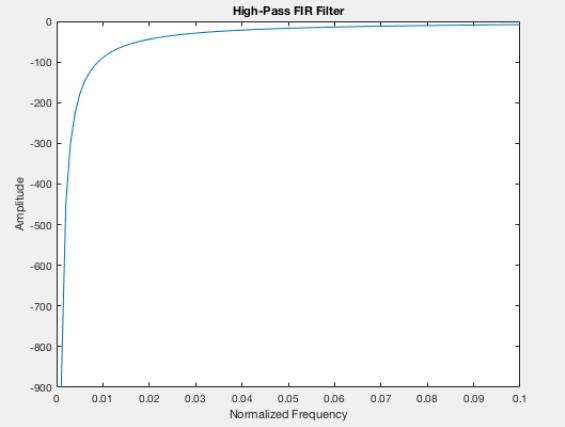

Figure 9: High-Pass FIR with a = .9

3.4.2 Windowing

Follow audio sampling and pre-emphasis, we must *window* the audio information. Speech signals are composed of many different sub-components/sounds, all of which have different "fingerprints" for use in classification. Hence, the signal must be broken up into windows for individual analysis. These windows, along with their feature vectors, can then be analyzed on the whole using probabilistic algorithms and neural networks (discussed later).

There are various forms of windowing, ranging from the simplest (rectangular windowing) to more complex functions.

The typical window length for speech recognition is approximately 20 to 30 milliseconds. Furthermore, we cannot simply cut-off one window and begin another at the same point – so we must choose some overlap value (in order to ensure no loss of information). The common offset is 10ms [9].

The entire process of windowing cuts the original signal into usable data as seen in Figure 10.

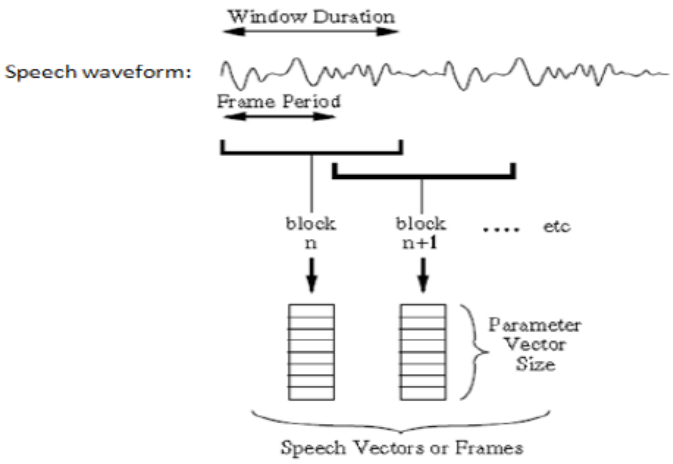

Figure 10: Windowing an audio signal, [23]

The type of window used in this project is the *Hamming window*, which is used in situations requiring higher frequency resolution [9]. The Hamming window tapers off at the edges, allowing for minimal discontinuity. The equation for the Hamming window is found in Equation 3.

$$w(n) = 0.54 - 0.46cos(\frac{2\pi(n-1)}{N-1}) \tag{3}$$

An example Hamming window can be seen in Figure 11.

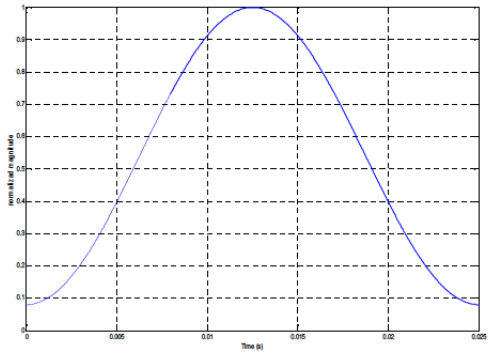

Figure 11: Hamming window

3.4.3 Fourier Transform

After pre-emphasizing the signal and windowing it, we then transfer into the frequency domain, where we can see the frequency content of the signal. A pure sinusoid oscillating at 100Hz, for instance, presents frequency content at 100Hz and -100Hz in the frequency domain. A pseudo-periodic signal (like a vowel) presents peaks at relevant oscillation frequencies.

Because the signal being processed is a discrete signal, the Fourier Transform used must also be discrete. The formula for a discrete Fourier Transform is as follows (Equation 4, 5) [21]:

$$X(e^{j\omega}) = \sum_{n=-\infty}^{\infty} x[n]e^{-j\omega n} \tag{4}$$

$$x[n] = \frac{1}{2\pi}\int_{2\pi} X(e^{J\omega})e^{j\omega n}d\omega \tag{5}$$

Luckily, most coding platforms (including MATLAB and Python) have built in FFT (Fast Fourier Transform) functions, so these formulae need not be input.

3.4.4 Filter-Bank Analysis

Following use of the DFT, the signal is now represented by its frequency content. The next step in processing is to "adapt the frequency resolution to a perceptual frequency scale which satisfies the properties of the human ears," [19]. That is,

we must "re-organize" the frequency content based on a perceptually-motivated scale, such as the Mel-frequency scale.

The strategy for this type of shift is *filter-bank analysis*, which consists of "a set of bandpass filter whose bandwidths and spacings are roughly equal to those of critical bands [here, based on the Mel-frequency scale] and whose range of the center frequencies covers the most important frequencies for speech perception," [13]. These bandpass filters are generally triangular and placed at relevant frequency bands. This set-up can be seen in Figure 12.

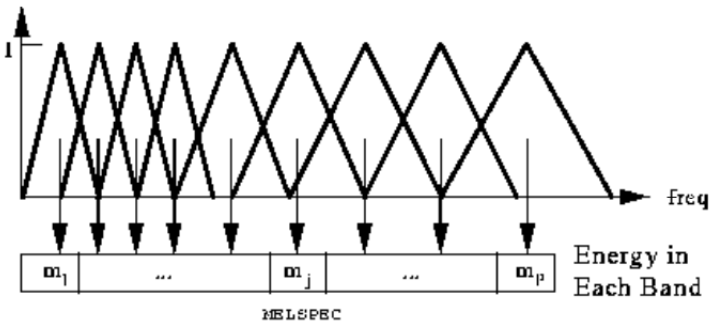

Fig. 5.3 Mel-Scale Filter Bank

Figure 12: Typical Filter-Bank, [5]

In filter-bank analysis, the magnitude of the DFT of a window of speech data is calculated. The resulting *magnitude coefficients are then "binned" by correlating them with each triangular filter* [5]. In other words, the input to this filter-bank is the power spectrum of the given frame. From this input, we are able to produce a *log-spectral-energy vector* as the output. This type of vector is computed via Equation 6. Here, we assume that H_m represents the transfer function of the filter m. The filter-bank is composed of M of these filters (i.e., M filterbank channels), where M can range from 24 to 40 (by convention) [9].

$$E[m] = \sum_{k=1}^{K-1} log[|X[k]^2 H_m[k]] \tag{6}$$

In more simple terms, *the filterbank samples the spectrum of the speech frame at Mel-frequency scale center frequencies* [9].

3.4.5 Discrete Cosine Transform

The Discrete Cosine Transform (DCT), a similar function to the DFT, expresses a discrete sequence of data points as a sum of cosine functions oscillating at

different frequencies. The equation for DCT-II, the most common form of the DCT (and the one used in this project) is shown in Equation 7 (where k = 0, ..., N-1):

$$X_k = \sum_{n=0}^{N-1} x_n cos[\frac{\pi}{N}(n + \frac{1}{2})k] \tag{7}$$

3.4.6 Cepstral Analysis

The major step in speech processing is the application of various functions to the signal in order to cut out irrelevant information and extract relevant features.

The type of analysis used in this project is motivated by our source-filter model of speech, in which the speech signal can be represented by Equation 8.

$$s[n] = e[n] * h[n] \tag{8}$$

As explained prior, the speech signal is assumed to be a convolution of an excitation signal and a vocal tract sequence. As a result, the Fourier transform of the speech signal is dictated by Equation 9 (using the duality principle of the Fourier Transform), where E(j ω) is the transfer function of the excitation signal and H(j ω) is the transfer function of the system:

$$S(j\omega) = E(j\omega)H(j\omega) \tag{9}$$

Once in the frequency domain, we face the problem of separating the excitation and system components. For a voice recognition system, we only want to retain the system component (*not* the excitation component). Why? Phonetic content is dictated by the system (the combination of factors such as vocal tract and radiation components). This is the content of interest in distinguishing sounds. For instance, a variation in the radiation component (the lips) can vary a given sound from an "f" to a "p." Meanwhile, the original source signal will contain information relevant only to pitch and speaker traits. These should be ignored in speech recognition (for the highest accuracy).

How can we separate these components? The simplest answer is to take the natural logarithm of the spectrum. This transformation produces Equation 10:

$$|log(S(j\omega))| = |log(E(j\omega))| + |log(H(j\omega))| \tag{10}$$

Now, transformation back into the time domain (via the inverse Fourier Transform) yields two separate signal components: one relating to the initial excitation and the other relating to the system. This yields Equation 11:

$$IDFT(|log(S(j\omega))|) = IDFT(|log(E(j\omega))| + |log(H(j\omega))|) \tag{11}$$

This unique transformation is called *Cepstral Analysis*, a process whose goal is to "separate the speech into its source and system components without any

a priori knowledge about source and/or system," [2]. A *cepstrum* (the name deriving from a combination of "spec" backwards with "trum") is produced by taking the inverse Fourier Transform (translating the signal from the frequency domain into the cepstral domain) of the logarithm of a spectrum. The basic transformation can be seen in Figure 8.

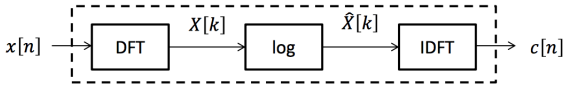

Figure 13: Block Diagram for Basic Cepstrum, [9]

The specific type of Cepstral Analysis used in this project is *Mel-Cepstral Analysis*. The block diagram for this modified process can be seen in Figure 14. It can be noted that this type of anaysis is unique in its use of the Mel filter-bank to generate a log-spectral-energy vector for each audio frame.

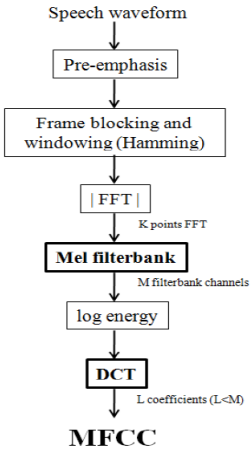

Figure 14: Block Diagram for Mel-Frequency Cepstral Analysis, [9]

The result of Mel-Frequency Cepstral Analysis produces a set of coefficients called *Mel-Frequency Cepstral Coefficients* or MFCCs. These coefficients are *a representation of a speech signal defined as the real cepstrum of a windowed signal derived from the FFT of that signal which is first subjected to a log-based transform of the frequency axis (Mel-frequency scale) and then decorrelated using a Discrete Cosine Transform* [9].

3.4.7 Liftering

In the final steps of MFCC calculation, we entered the cepstral domain. The cepstral domain is very similar to the time domain, but differs by the processes performed in cepstral analysis (making alterations in the frequency domain means that reversal of the initial Fourier Transform will not actually return the signal to the time domain). This domain is hence given an alternate name, the *quefrency* domain (the word "frequency" scrambled).

The process of "filtering" in the quefrency domain is called *liftering*. The process of liftering is informed by the calculations we use to enter the cepstral domain in the first place. In separating the system and excitation components, we produce two very separate components in the frequency domain – one at lower "times" representing the vocal tract component and another a higher "times" representing the original excitation.

In speech recognition, we desire the system component. Hence, to isolate this component, we *low-time lifter*. This can be done by multiplying the whole cepstrum by a rectangular window centered on lower quefrencies, or by de-emphasizing the higher cepstral coefficients. A depiction of low-time liftering can be seen in Figure 15.

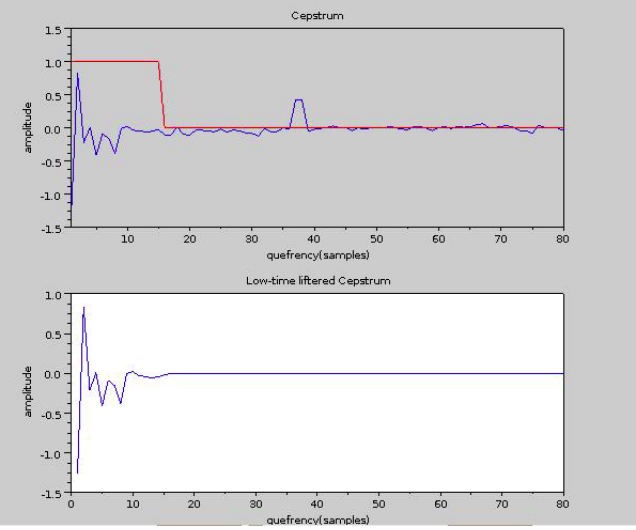

Figure 15: Example of Low-Time Liftering, [2]

Following low-time liftering, only the system component remains in the cepstral representation. This removes the possibility of pitch information interfering with the results.

3.5 Speech Recognition

We have already reviewed the fundamental composition of speech and the steps involved in feature extraction, but what comes next? How can we use these features to make relevant conclusions about the incoming signal? This involved two sub-steps:

1. The MFCCs for each frame must be mapped to a phone (a single sound)
2. These phones must be "strung together" in the most probable arrangement to output the most probable input set of words

3.5.1 Hidden Markov Model

There are various methods for speech recognition that are considered to be "conventional" – one such algorithm is the Hidden Markov Model, a tool for modelling time series data. This type of model represents *probability distributions over sequences of observation,* making it a relevant tool for pattern recognition (and speech recognition) [11]. An HMM is just a *Bayesian Network*, or a graphical model for representing conditional independencies between a set of random variables. The Bayesian Network representation of an HMM can be seen in Figure 16, where S_i is a state and Y_i is its corresponding observation.

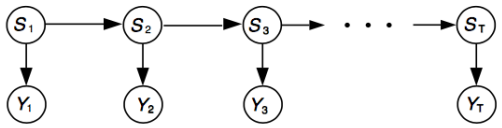

Figure 16: Bayesian Network for Hidden Markov Model, [11]

A Hidden Markov Model is characterized the following qualities:

1. The process used to generate the current state is hidden from the observer
2. The current state S_t is independent of states before t-1
3. The hidden state variable is discrete

This type of model allows us to "string" together various states (in this project, represented by a feature vector) probabilistically in order to produce an observation about the output (a phone, word or phrase).

3.5.2 Artificial Neural Networks

In recent years, more complex systems for speech recognition have rivaled the conventional Hidden Markov Model. Relevant to this type of mapping problem

are *Artificial Neural Networks* (ANNs). As Kamble cites, ANNs "are nothing but the crude electronic models based on neural structure of brain," [16]. In these systems, artificial neurons (or nodes) act as the basic processing unit (see Figure 17). These types of systems are capable of being "trained" using training data, and can developed incredibly accurate processing and predicting abilities if given enough with which to work. These systems can then predict output given input using the information they have "learned" in training.

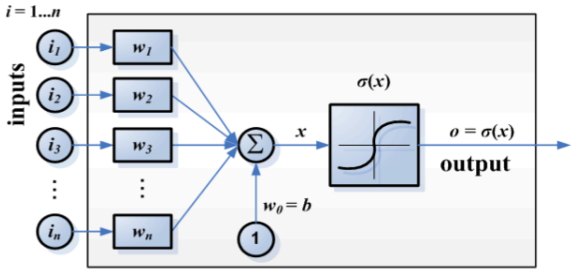

Figure 17: Basic Model for Artificial Neuron, [16]

ANNs are relevant to speech recognition in that the patterns of speech (observed through feature vectors) can be used as training data, in order to create a system capable of identifying feature vectors and categorizing them to specific phones.

The types of ANNs typically implemented in speech recogition are as follows [16]:

1. **Convolutional Neural Network:** A CNN applies a series of filters to a matrix of data to extract and learn features for use in classification. This type of neural network is used for input-output problems, such as image classification (where a matrix of pixels is the input and an image classification is the output).

2. **Recurrent Neural Network:** An RNN is composed of connections that resemble directed cycles, creating a feedback system that facilitates dynamic behavior over time. This "memory" can be used to process independent sequences (such as phones in speech recognition).

3. **Multilayer Perceptron:** As opposed to RNNs, MLPs use a *feedforward* system that allows the mapping of a set of inputs onto a set of outputs.

In this project, we will focus on CNNs because of their relative simplicity in comparison to other ANNs, as well as their feature mapping abilities.

3.5.3 Mapping MFCCs to Phones

Returning now to our initial goals, let us flesh out what must occur in order for audio signals to be classified by entire words and phrases.

The goal of the first sub-step of processing is to map each set of MFCCs for a single frame to a phonetic category. A sample case could be as follows:

1. The letter 'e' is spoken and digitized
2. A set of MFCC vectors are extracted from this signal
3. Example mappings:

   One of the MFCC vectors may map to the following phonetic probabilities:

   a) 70 percent chance: Voiced *e*

   b) 20 percent change: Voiced ə

   c) Other negligible categorizations

   Another the MFCC vectors, perhaps 5ms later, may map to the following phonetic probabilities:

   a) 60 percent chance: Voiced *e*

   b) 30 percent change: Voiced ə

   c) Other negligible categorizations
4. These mappings would be reviewed in order to group phonetically-related frames and identify phones

Because the desired result for this step is simple input output based on a single vector for each frame, there are many different options for processing.

A CNN, which is useful for feature mapping, can be used to address a challenge like this. Alternately, Hosom et. al use a Multilayer Perceptron (MLP) which, as described prior, is a feedforward ANN model that maps sets of input data onto a set of appropriate outputs [14]. This model is essentially a directed map consisting of multiple layers of nodes (processing elements). This type of model utilizes a type of supervised learning called *back-propagation*, which error corrects using a loss function and backwards traversal during each training epoch. The placement of this algorithm in speech recognition can be seen in Figure 18.

This type of analysis certainly fits with the challenge, but as Kamble points out, the speech recognition rates are bested by alternate systems, such as *Recurrent Neural Networks* (RNNs) [16]. New models for faster, more accurate and more efficient speech recognition are constantly being developed.

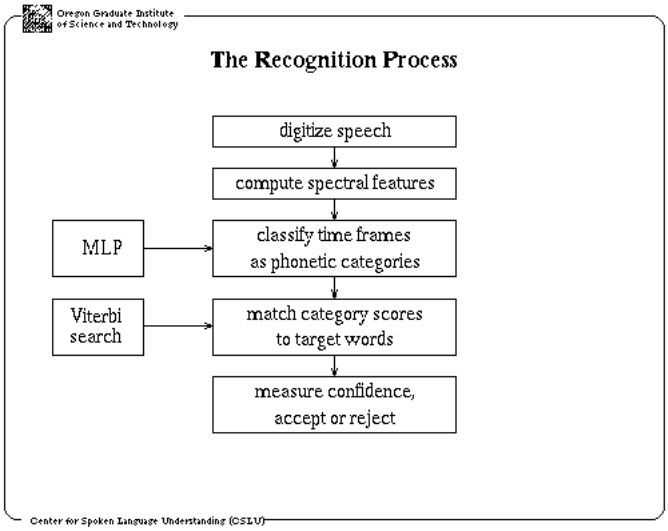

Figure 18: Speech Recognition Process [14]

3.5.4 Matching Phonetic Groupings to Words

The matching of phonetic groupings to words is less emphasized in this paper, but is nevertheless relevant. Such mappings can be approached similarly to the prior problem, but the input is now sets of probable phones used to construct a meaningful output. This type of challenge can be approached using the Hidden Markov Model, as discussed in the concept review, which allows for the *time invariant* combination of states to produce an observed output.

Neural networks could similarly be trained with phonetic combinations to produce reasonable speech output. Neural networks can even be hybridized with the Hidden Markov Model [8].

4 Signal Processing

The first goal of this project involved taking in a signal as a .WAV file, windowing this signal, performing cepstral analysis on these blocks of data and post-processing the resulting feature vectors to return unique representations of each window of time.

4.1 Relevant Resources

4.1.1 Exploring MATLAB's Speech Processing Capabilities

Before diving into hand-coded MATLAB scripts and functions, it is important to note that MATLAB has several useful signal/speech processing libraries/-functions.

Of most relevance will be the following functions:

1. Spectrogram: Performs frequency analysis on the given signal to display the frequency content as a spectrogram (a visual representation of a spectrum)
2. Filter: Used to apply basic filters to a given signal (including LPFs, HPFs, etc.)
3. FFT: Performs the Fast Fourier Transform on the given data set

These functions can be used to confirm non-tangible statements made in the *Concept Review*. For instance, observe the plots produced (using these functions) to represent various unique phonemes in Figures 19, 20, and 21. We can see in these figures the quasi-periodic time domain representation of the voiced phoneme 'e', showing clear bands at select frequencies, compared to the random time domain representation of the unvoiced phoneme 's', with varied bands of frequency over time.

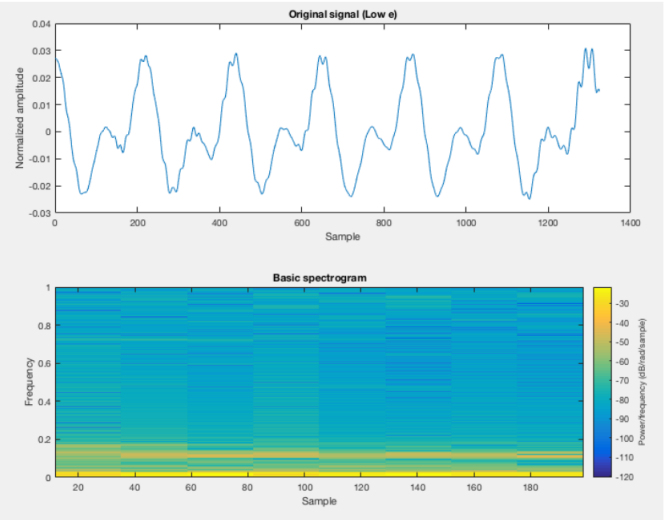

Figure 19: Single frame from the vowel 'e' (time and frequency domain)

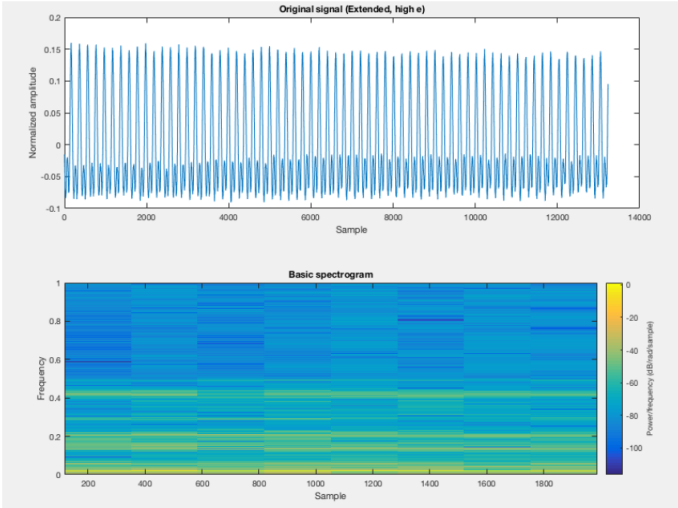

Figure 20: Single frame (extended) from the vowel 'e' (time and frequency domain)

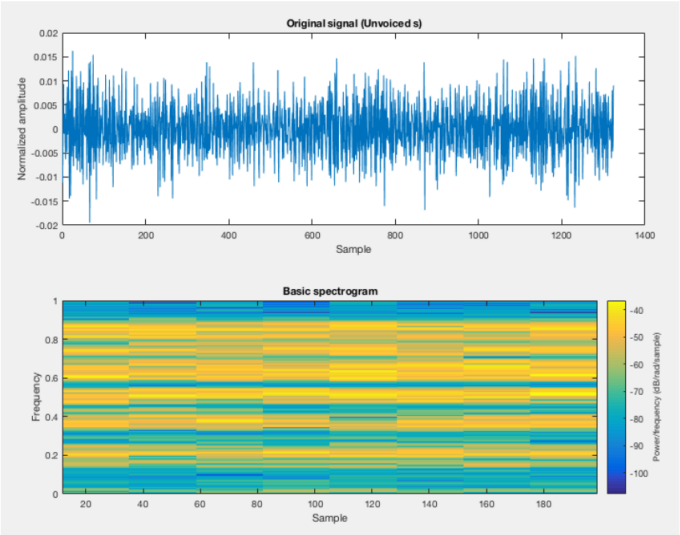

Figure 21: Single frame from an unvoiced 's' (time and frequency domain)

4.1.2 External Libraries & Similar Projects

Also of great use are similar projects and libraries produced in MATLAB and Python, which offer varied approached to sub-functions of the signal processing.

One library of particular use is the VOICEBOX speech processing library, written by Mike Brookes, Department of Electrical & Electronic Engineering, Imperial College (API found on the Imperial College website) [10]. This library offers useful functions such as:

1. Audio I/O functions (e.g., readwav & writewav; read and write WAV files)
2. Frequency conversion functions (e.g., frq2mel; convert from Hz to Mel)
3. Fourier transforms (e.g., rfft; computes real Fourier Transform efficiently)
4. Framing functions (e.g., enframe; frames the given signal)
5. Speech analysis functions (e.g., melbankx & melcepst; compute the Mel-frequency filter bank for a signal, compute the Mel-Cepstral coefficients)

The PLP and RASTA (and MFCC, and inversion) in Matlab project, developed by Mike Shire from Columbia University, also offers another approach to MFCCs (documentation found on the Columbia website) [22]. This library comes equipped with the melfcc and invmelfcc functions for Mel-Cepstral and inverse Mel-Cepstral analysis.

A final project to mention, though not in MATLAB, is the Python speech analysis package written by James Lyons (available on GitHub) [18].This project holds most of its functionality in a single file, *sigproc.py*, which includes functions like framesig (frame signal), *powspec* (power spectrum) and *preemphasis*.

4.2 Developing an MFCC system

4.2.1 Pre-emphasis

The process of pre-emphasis is relatively simple in MATLAB. Using the built-in filter function, one can create a high-pass FIR filter to apply to the signal (see *basicdemo.m* in the Appendix).

Figures 22 and 23 show a simple test signal (the letter e) before and after pre-emphasis. The increased influence of higher frequency content is clear both in the time and frequency domain.

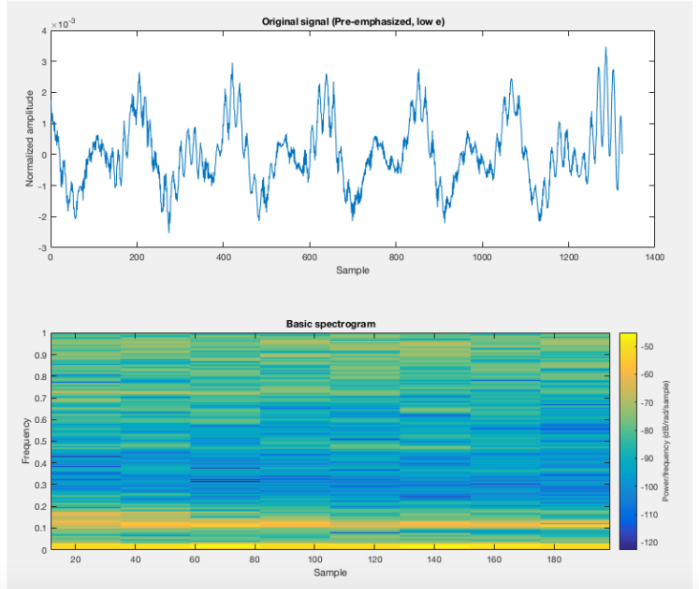

Figure 22: Signal prior to pre-emphasis

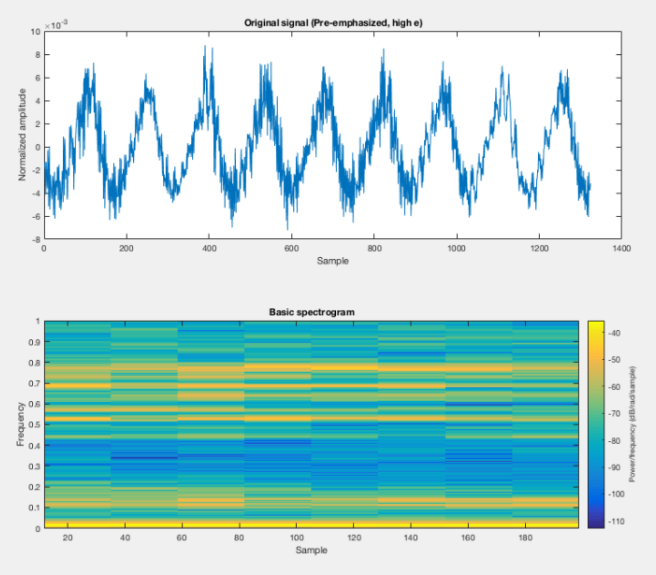

Figure 23: Signal post pre-emphasis

4.2.2 Windowing

A Hamming window was chosen for the windowing process in order to preserve frequency resolution. Windowing was accomplished via a helper function (*frame*, found in the *Appendix*), which multiplies the signal by the Hamming function (Equation 3) at intervals of about 50 percent the window size (by default). These result is a vector of frames, where each frame captures the sound content within an isolated window.

Of great help in this function's development was the enframe function from the VOICEBOX library [10], which can be found in the Appendix. This function also utilizes Hamming windows, and uses similar tactics to window and increment the signal.

The results of the framing function were tested via the *framedemo* (again, found in the Appendix) script, which reads in a small portion of one of the recorded vowels and windows it using the framing function. The result can be seen in Figure 24.

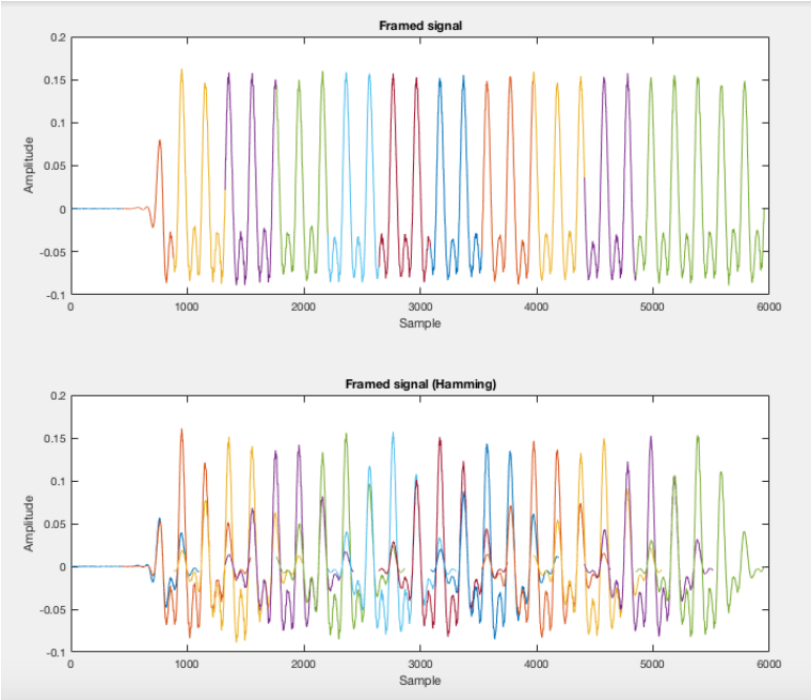

Figure 24: Frame demo results

The influence of the Hamming window is clear – each individual frame (colored differently) takes on the sinusoidal shape of the Hamming window, tapering off towards the edges of the sample.

4.2.3 Fourier Transform

The next step, after windowing the signal, is to move from the time domain and into the frequency domain using the Fourier Transform. Because the signal is a discrete set of data, we must use the Discrete Fourier Transform, which is supported by the *fft* function in MATLAB. It is also necessary to reshape the data following transformation in order to remove mirrored information (resulting from use of the *fft* function. These steps, along with extra "catch" conditions, are encoded in the rfft function found in the *Appendix*. This function is derived from the *rfft* function in the VOICEBOX Library [10].

The *rfftdemo* (also found in the *Appendix*) tests this function by selecting a frame from one of the recorded sample vowels and applying the function to the frame. The original signal and resulting spectrum can be seen in Figure 25.

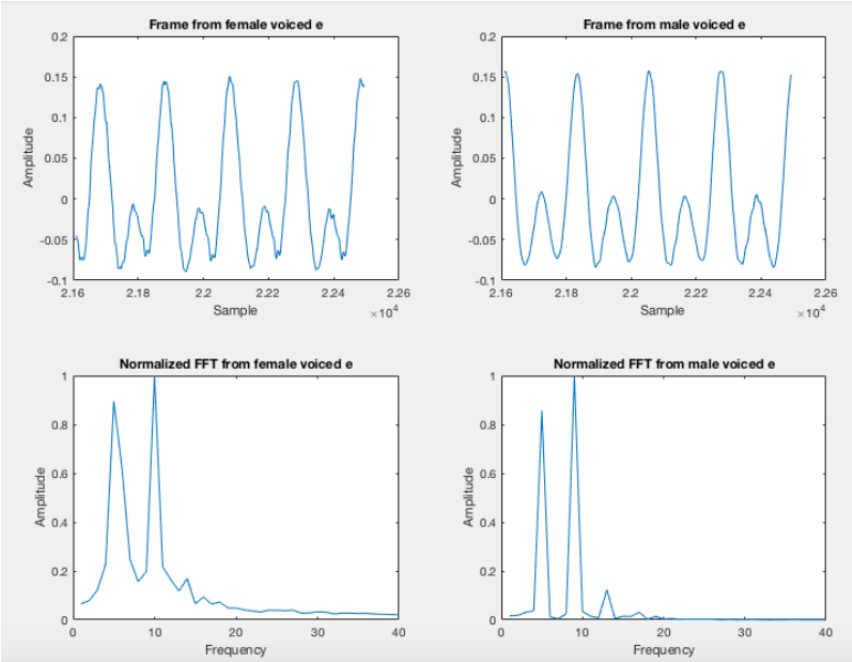

Figure 25: RFFT Demo Results

There is noticeably more frequency variation in the female sample, with frequency content centered at a slightly higher frequency. This is to be expected given the higher frequency of the female voice. We can also observe two major peaks in both spectra, corresponding the frequencies of oscillation present in the pseudo-periodic vowel sound signals. These frequencies can also be observed in the time domain, where the original signal appears to be roughly the summation of two different sinusoids.

4.2.4 Filter-Bank Analysis

Filter-bank analysis was implemented using the *melbankm.m* function in the *Appendix*, allowing for the concentration of frequency content at perceptually relevant frequencies. This step represented a useful mid-point in that the output log-spectral-energy vectors can be plotted in order to observe where the highest concentration of energy resides for each sound vector. The *filterbank* function in the *Appendix* was used to generate such plots for observation. These plots can be seen in Figures 26, 27, and 28.

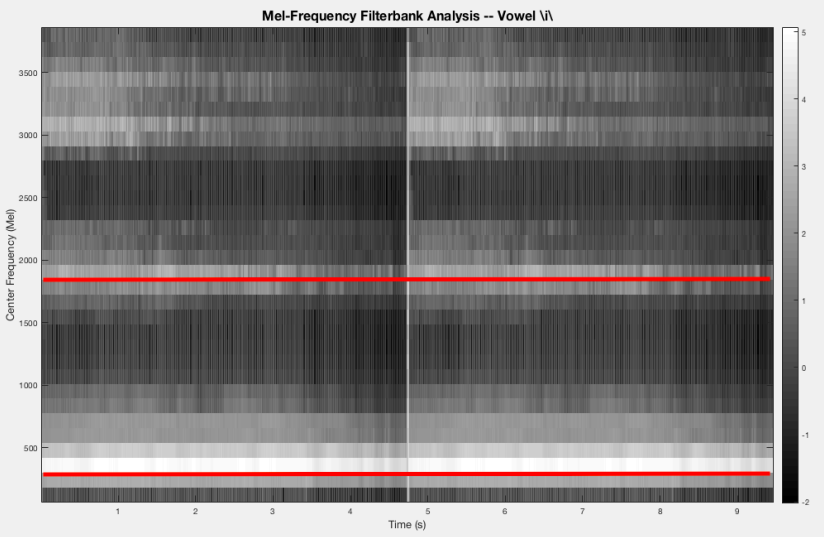

Figure 26: Filterbank Results for Vowel $\backslash i \backslash$

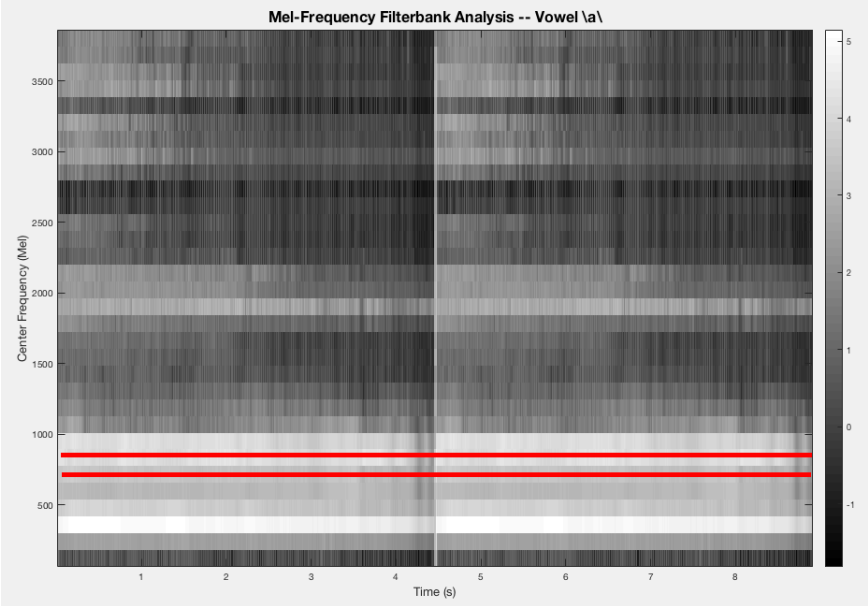

Figure 27: Filterbank Results for Vowel $\backslash a \backslash$

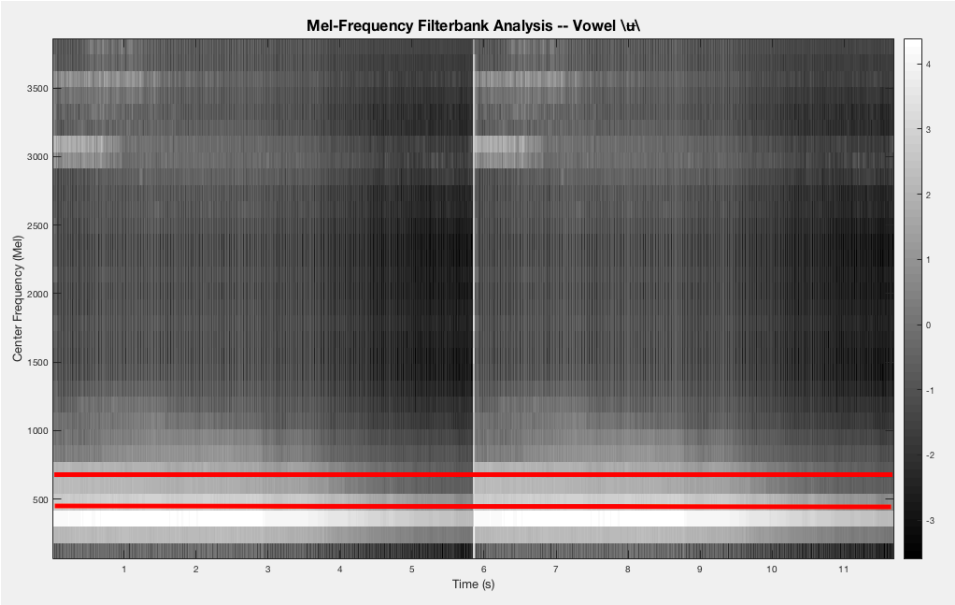

Figure 28: Filterbank Results for Vowel \ʉ\

Each of these figures is marked with two red lines, representing the expected frequency for high energy content (the first two formants of that particular vowel). For the most part, these match up correctly, confirming the calculations up to this point. The actual vs. expected high-energy center frequencies can be viewed in Table 1 (noting that all frequencies have been converted to Hz; [3]).

Table 1: Filter-Bank Energy Concentration Results for Three Vowels

	Expected F1 (Hz)	Expected F2 (Hz)	Actual F1 (Hz)	Actual F2 (Hz)
\i\	350	2400	200	2600
\a\	570	870	500	900
\ʉ\	320	950	400	700

For the most part, these pairs seem to match up reasonably (attributing error to speaker-related factors, pitch, etc.).

4.2.5 Mel-Frequency Cepstral Coefficients

All of the previous demos culminate in the development of the Mel-Frequency Cepstral Coefficient function.

The first step in calculating the MFCCs is generating all of the parameters necessary for calculations (see the MATLAB code below). This includes:

1. The number of coefficients is set to 12 (coefficents above this number typically have little to no relevance to the frequency content)
2. The number of filters in the filterbank is calculated from the sampling frequency (where a higher sampling frequency will trigger a larger number of filters)
3. The length of the window is also derived from the sampling frequency (where a higher sampling rate triggers a window holding more samples)
4. The filter bank limits (high and low) are set (for the Mel-scale filter bank)
5. The increment for framing is set to approximately half the length of a window

Note that many of these default parameters are informed by values generated in the VOICEBOX library [10].

```
%% Set the initial variables
if nargin<2 fs=44100; end
nc=12; %% number of coefficients
%number of filters in filterbank
p=floor(3*log(fs));
%length of window
n=pow2(floor(log2(0.03*fs)));
%Filter bank limits
fh=0.5;
fl=0;
%Frame increment by sample
inc= floor(n/2); %hamming
```

Next the sample is framed using a hamming window (of the calculated window length), incremented by the calculated increment of approximately 50 percent of the window length. These frames are pre-emphasized (emphasizing higher frequencies). The *rfft* function is applied to the resulting frames.

```
%% Frame the sound
[z,tc]=enframe(s,hamming(n),inc);
%% Pre-emphasis
z = filter([ 1 -.95], 1, z);
%% Take the fourier transform
f=rfft(z.');
```

With the signal windowed and translated into the frequency domain, the Mel-scale filter-bank (see this code in the *Appendix*) is applied to each frame. The output of the filter-bank is used to calculate the log-spectral-energy vectors for each frame.

```
%% Create mel Filter bank (centered on frequencies)
[m,a,b]=melbankm(p,n,fs,fl,fh);
% m = filterbank magnitudes
% a = filterbank center frequencies
% b = lowest FFT bin with non-zero coefficient

%% Calculate the log of the energies in each filter bank
pw=f(a:b,:).*conj(f(a:b,:));
pth=max(pw(:))*1E-20;
ath=sqrt(pth);
y=log(max(m*abs(f(a:b,:)),ath));
```

Finally, the Discrete Cosine Transform (see this code in the *Appendix*) is applied to the result in order to produce the Mel-Frequency Cepstral Coefficients. The results are fitted (zero-padded, if necessary), and low-liftered to isolate formant-relevant information. Finally, the first coefficient is excluded (as it represents the summation of energy across all of the coefficients and is not necessary in following analysis).

```
%% Discrete Cosine Transform (last step)
% When plotted, look the same.
c = rdct(y).';

%% Fit the coefficients
nf=size(c,1);
nc=nc+1; %(extra added on at 0)
if p>nc
   c(:,nc+1:end)=[];
elseif p<nc
   c=[c zeros(nf,nc-p)];
end
c = (lifter(c', -.6))';

%% Exclude first coefficient
c(:,1)=[];
```

The result of this function is a matrix of cepstral coefficients, representing a list of MFCC vectors (12 coefficients long) matching to each frame in the signal. These MFCCs have been low-liftered using the *lifter* function (which can be found in the *Appendix*) so that speaker-related factors are minimized. Note that the liftering accomplished in this project is done by multiplying each coefficient by i^{lift} – hence, the later coefficients (e.g, i = 6, 7, 8), are de-emphasized while the earlier coefficients are retained.

The resulting MFCCs can be viewed using the spectrogram function in MATLAB. The result for a short audio clip (an \i\) can be seen in Figure 29. Each vertical band is a single MFCC vector, representing a single frame in the signal. The lighter colors correspond to larger coefficients while the darker colors correspond to smaller (and less relevant) coefficients. The earlier coefficients (e.g., 1-3) are consistently larger than the rest of the coefficients.

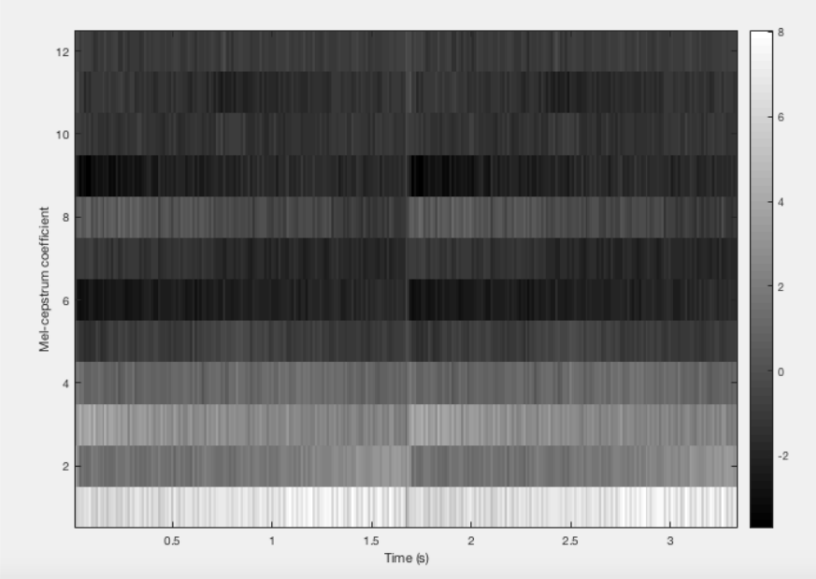

Figure 29: MFCCs for short clip, \i\

5 Speech Recognition

5.1 External Libraries & Similar Projects

Of all of the possible methods for speech recognition using MFCCs, the Hidden Markov Model is perhaps the most documented and tested. One such fleshed out version is the *HMM Speech Recognition in Matlab* project, which is available at the following site. This project offers viterbi path function, MFCC conversion function and training/testing methods [15].

A similar project can be found in the MIT HMM Toolkit, authored by Kevin Murphy. This project has all the basic HMM functions (e.g., viterbi path), as well as multiple demos [20].

For the work with Neural Networks, we utilize *TensorFlow*, an open source library for machine learning. Of particular use is the *tflearn* package, which utilizes TensorFlow to implement a variety of neural nets with varied training data. This package offers a fleshed-out API and examples package, including examples for CNNs.

5.2 Developing a Speech Recognition System

5.2.1 Recognition of Vowels via Correlation

The first step to developing a system to recognize phones from MFCCs is establishing the accuracy of basic MFCC-based prediction. As a demonstration of the capabilities of MFCCs, let us observe the variation between the vowels \i\ (the letter e), \u\ (an "oo" sound), and \a\ (an "ah" sound). Each of these sounds has a distinct "fingerprint" in its MFCCs.

To observe these unique profiles, we observe 8 recordings of each of these three vowels, representing varied pitch, volume and tone. The process for extraction across an entire audio clip included the following steps:

1. Read in the audio and sampling rate using audioread
2. Pass this information to the *melcepst* function to produce a list of MFCC vectors (each 12 coefficents long)
3. Pass these vectors to the *avnormcep* function which takes the average of all of these coefficents and normalizes the result (see avnormcep in the Appendix)

The resulting average MFCC vector for these audio clips can be seen in Figure 30 (MATLAB code found in *Appendix: Plot Ceps*). The varied maxima and minima presenting (relatively) consistently for each vowel represent identifiers. When all of these values are averaged and normalized across vowels, we can see a clear distinction between each phone (Figure 31; code found in *Appendix: Plot Ave Ceps*).

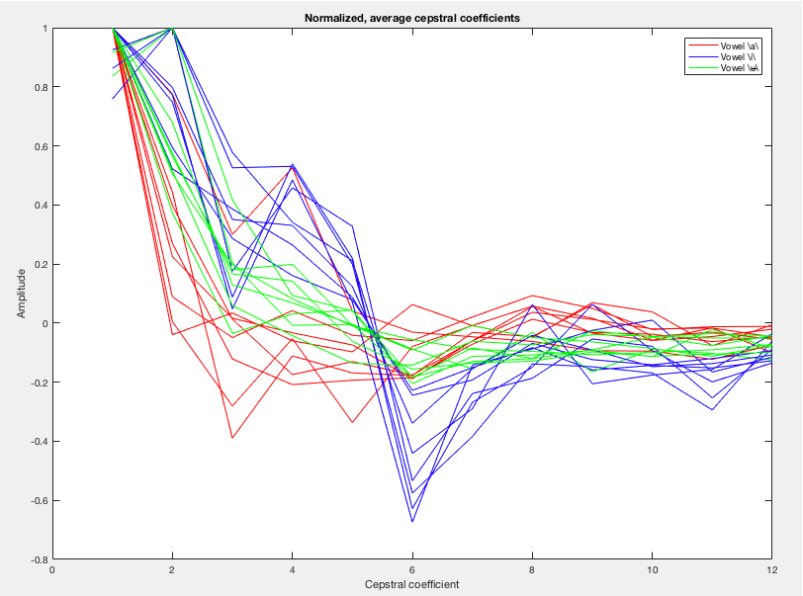

Figure 30: MFCCs for 8 separate recordings of the sounds \i\, \ʉ\ and \a\

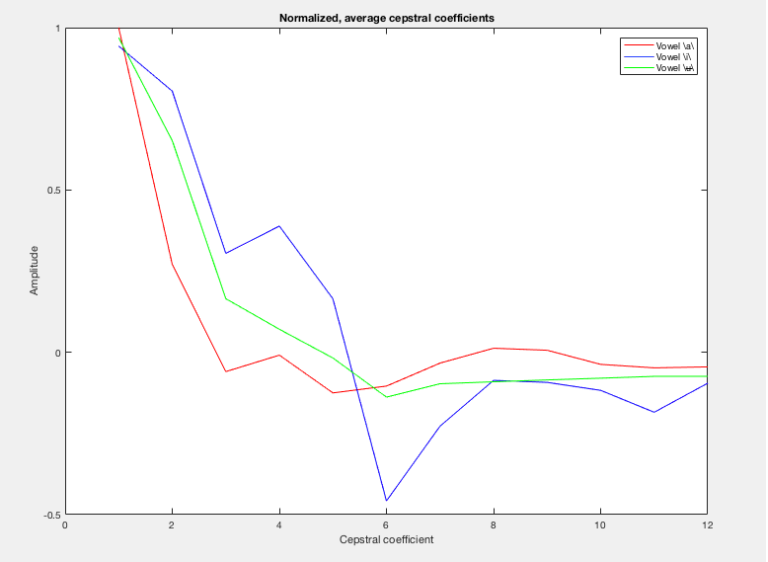

Figure 31: Averaged MFCCs for phones \i\, \ʉ\ and \a\

Visually, one can easily distinguish one vowel from another. However, this is not the goal of this project – no human processing should be necessary. Instead, we can utilize *correlation* in order to determine the most likely match.

The correlation function used for this particular analysis is the MATLAB *corr* function, which performs linear correlation analysis and returns a value dictating the similarity of the two datasets (where 1 represents a perfect match and -1 represents no relationship). The primary purpose of this type of analysis is "to determine whether there is a relationship between two sets of variables," [1]. The equation for correlation is shown in Equation 12.

$$r = \frac{n\sum xy - \sum x \sum y}{\sqrt{(n\sum x^2 - (\sum x)^2)(n\sum y^2 - (\sum y)^2)}} \quad (12)$$

Returning the tag (e.g., "e", "a") for the highest correlation can consistently map a set of MFCCs to the correct vowel (N.B. the phoneme $\backslash i \backslash$ is mapped to the sound "e", the phoneme $\backslash \textstrike{u} \backslash$ is mapped to the sound "o" and the phoneme $\backslash a \backslash$ is mapped to the sound "a"). This is accomplished via the *predictvowel* method (found in the *Appendix*). The results, using the "training data" as input, are promising. The returns of this function for each audio clip can be seen in Table 2.

Table 2: Predict Vowel Test

Audio Clip	$\backslash i \backslash$	$\backslash \textstrike{u} \backslash$	$\backslash a \backslash$
1	"o"	"o"	"a"
2	"e"	"o"	"a"
3	"e"	"o"	"a"
4	"e"	"o"	"a"
5	"e"	"o"	"e"
6	"e"	"a"	"a"
7	"e"	"o"	"a"
8	"e"	"o"	"a"

The results of this testing show a prediction success rate of approximately 87.5 percent, with some misfires for each vowel. The mistakes in matching can likely be attributed to the rudimentary nature of this prediction system – it uses basic correlation (without contextual/temporal cues) to make its decision. Likewise, each vowel is represented by eight 5-10 second clips, varied but spoken consistently by the same people. Even attributing these mistakes to the MFCC-matching system itself, it is still clear that MFCCs present a unique enough "fingerprint" for different phones to build a speech recognition system.

5.2.2 Recognition of All Sounds via CNN

Because the focus of this paper is not ANNs, we will not delve too deeply into the inner mechanics of such machine learning techniques. Instead, we will examine a sample case relevant to the current task of pattern matching.

As mentioned in the concept review, Convolutional Neural Nets (CNNs) represent a reasonable option for pattern matching, as they represent an "input-output" system-type, allowing for mapping between features vectors and speech patterns.

We will review a CNN as a case study, utilizing MNIST training data (The MNIST database is a large collection of labels matched to hand-written letters, representing useful training data for different neural net models [17]). This (and other) CNN is *composed of a stack of convolutional modules that perform feature extraction* [7].

A huge part of neural nets is simply formatting the dataset. In the following excerpt, we read in this labelled data and reshape it accordingly (here, we have a 28 by 28 pixel, monochrome image). This translates to -1 for the batch-size (for dynamic computation), 28x28 for the width and height of the pixel matrix and 1 for the number of channels.

```
1 X, Y, testX, testY = mnist.load_data(one_hot=True)
2 X = X.reshape([-1, 28, 28, 1])
3 testX = testX.reshape([-1, 28, 28, 1])
```

Following reshaping, we must actuall built the neural model. We set-up the following layers:

1. Layer to receive input
2. Convolutional Layer #1 (ReLu): Applies a saturating activation function
3. Pooling Layer 1: Reduce spatial size of representative and reduce overfitting of data
4. Convolutional Layer #2 (ReLu)
5. Pooling Layer #2
6. Fully Connected Layer #1: Connects previous layers to the current layer (with a drop-out function to randomly remove neurons and "force" different paths in order to reduce over-fitting)
7. Fully Connected Layer #2 (Also with dropout)
8. Fully Connected Layer #3
9. Regression Layer using Adam algorithm for optimization

```
network = input_data(shape=[None, 28, 28, 1], name='input')

network = conv_2d(network, 32, 3, activation='relu', ...
    regularizer="L2")

network = max_pool_2d(network, 2)
network = local_response_normalization(network)

network = conv_2d(network, 64, 3, activation='relu', ...
    regularizer="L2")

network = max_pool_2d(network, 2)
network = local_response_normalization(network)

network = fully_connected(network, 128, activation='tanh') ...
    #connect layers
network = dropout(network, 0.8) #remove neurons and force new paths

network = fully_connected(network, 256, activation='tanh')
network = dropout(network, 0.8)

network = fully_connected(network, 10, activation='softmax')

network = regression(network, optimizer='adam', learning_rate=0.01,
loss='categorical_crossentropy', name='target')
```

Finally, we must fit and train this model. Using the *tflearn DNN* function, we can construct the model from the architecture laid out. This can then be fit to the data (the MNIST dataset) to create a functional CNN model.

```
# Deep neural net --> build model
model = tflearn.DNN(network, tensorboard_verbose=0)

#Train the model, 1 epoch
model.fit({'input': X}, {'target': Y}, n_epoch=1,
           validation_set=({'input': testX}, {'target': testY}),
           snapshot_step=100, show_metric=True, ...
               run_id='convnet_mnist')
```

The training steps of this function output model stats as follows, showing the batch-size, accuracy, etc.:

```
---------------------------------
Run id: convnet_mnist
Log directory: /tmp/tflearn_logs/
---------------------------------
Training samples: 55000
Validation samples: 10000
--
Training Step: 100  | total loss: 0.22624 | time: 21.781s
| Adam | epoch: 001 | loss: 0.22624 - acc: 0.9222 | val_loss: ...
    0.17081 - val_acc: 0.9487 -- iter: 06400/55000
--
Training Step: 200  | total loss: 0.24586 | time: 49.407s
```

```
12  | Adam | epoch: 001 | loss: 0.24586 - acc: 0.9285 | val_loss: ...
        0.18801 - val_acc: 0.9505 -- iter: 12800/55000
13  --
```

Clearly, this is by no means a comprehensive tutorial. A similar (if not more-in-depth) walk-through can be found on *TensorFlow's site*.

Noting the output of the training system, we can see the the training model reaches a value accuracy of 95% by the second training batch. This early accuracy is ideal in comparison to the 85% accuracy offered by the previously proposed basic correlation model (noting, of course, that this model is using a different type of training data). This is likely partially related to the larger training size (which can be accommodated easily via this type of system). This is also likely a result of the many layers of the neural architecture, which allow for high-level feature extraction.

This same system for matching can be used with MFCCs. MFCCs matched to expected phone labels can be used as a dataset similarly to the MNIST dataset. Not only is this system functional, but it is arguably better than other approaches. Han et. al, using the DCASE challenge dataset, show that *ConvNet outperforms both of the baseline system with hand-crafted features and a deep neural network approach by around 7%* [12].

This type of neural net hybridized with a Hidden Markov Model has also been explored by several groups. Abdel-Hamid et. al show that such a hybrid *can achieve over 10% relative error reduction in the core TIMIT test sets when comparing with a regular NN using the same number of hidden layers and weights* [8].

6 Conclusion

Clearly, all of speech processing and recognition was not addressed in this paper. There are other ways to extract features from speech signals (such as LPCs), and other ways to predict speech input (such as HMM-Gaussian hybrids). Still, using the chosen tactics, we have demonstrated the general process of rudimentary speech recognition, exploring various avenues for matching speech patterns to phones, words and phrases.

6.1 Remaining Challenges

Moving forward, there remain several challenges with the techniques used in this project. These include:

1. **Interference with MFCCs:** MFCCs have been known to be prone to

interference. They can be expanded using add-on functions to filter out bacgkground noise, but such methods have not been explored in this paper.

2. **Variable length words:** The temporal quality of speech (the difference between a brief "Hello" and an extended "Hellooooo?") has also not been thoroughly considered. Ideally, variation in length should not vary the output of the system.

3. **Speed:** For the most part, speed has been ignored in this paper. Still, as mentioned in the introduction, this product requires speed in order to be useful. Hence, this is a point of future testing and development.

4. **Accuracy:** The basic accuracy (using a small training set) of this system has been demonstrated, but its accuracy with a large training set has only been assumed to reach the accuracy standards. This would also require the collection of additional samples and extensive testing.

6.2 Future Goals

In future extensions of this project, we hope to address the remaining challenges presented above and develop high speed and accuracy from our system. This will include application of the Convolutional Neural Network architecture present prior. This will also require testing units and additional sample collection (as the current training data is minimal). The end goal is a fast, accurate system capable of identifying entire words and phrases (instead of just phones).

7 Appendix

7.1 Basic Demo, *basicdemo.m*

```
function [signal] = basicdemo(name, conditions, window, shift)
% Basic demo which reads in a file, applies the given conditions
% and windows using the window and shift size given
%   param: name File name
%   param: conditions Array of conditions (e.g., 'p' for ...
    pre-emphasis)
%   param: window Window size
%   param: shift Shift size (how far into the signal to shift ...
    for display)
% Returns single channel from original signal
%
% Author: Maddie Briere, 2017

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


% Fill in missing parameters
if nargin<3
    window = .030;
end

if (nargin<4)
    shift = 1000;
end

if shift≤0
    shift = 0;
end

preemph = [1 1];
if any(conditions) == 'p'
    preemph = [ 1 -.95];
end


% Read in file
[signal, fs] = audioread(name);

% Isolate first channel (fine for demo purposes)
signal = signal(:, 1);

% Apply pre-emphasis filter
emp = filter(preemph, 1, signal);

% Define sample size using window length
samplesize = fs*window;

% Isolate single frames (using simple rectangular windowing)
empframe = emp(shift:samplesize+shift);


```

```
figure 1
subplot(2, 1, 1)
plot(empframe)
title('Original Signal')
xlabel('Sample')
ylabel('Normalized amplitude')

subplot(2,1,2)
spectrogram(empframe, 'yaxis')
title('Basic spectrogram')
xlabel('Sample')
ylabel('Frequency')
```

7.2 Frame, *frame.m*

```
function [ frames, times ] = frame(signal, fs, window, inc, none)
% Frames signal using Hamming window.
% Split signal up into (overlapping) frames: one per row.
% Input:
%   signal: Vector of audio data
%   fs: Sampling frequency
%   window: Length of frame
%   inc: Separation of frames
%    none: Do not perform framing
% Output:
%   frames: mxn matrix of frames
%
% Author: Maddie Briere, 2017
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if nargin<3
    window = .020*fs; %standard
end

if nargin<4
    inc = .010*fs; %standard
end

limit = size(signal, 1)-window;

frames = [];
times = [];

samples = 1:1+window;
len = size(samples, 2);
 w = .54 - .46.*cos((2.*pi.*(samples-1))./(len-1));


for i = 1:inc:limit
    frame = signal(samples+i-1)';
    if nargin≤4
        frame = (frame.*w);
    end
    frames = [ frames frame' ];
```

```
40      times = [times (samples+i-1)'];
41  end
42
43  frames = frames';
44  times = times'; % row = frame
45
46  end
```

7.3 Enframe, *enframe.m*

```
1   function [f,t,w]=enframe(x,win,inc,m)
2   %ENFRAME split signal up into (overlapping) frames: one per row. ...
        [F,T]=(X,WIN,INC)
3   %
4   % See full usage in VOICEBOX package
5   %       Copyright (C) Mike Brookes 1997-2014
6   %       Version: $Id: enframe.m 6490 2015-08-05 12:47:13Z dmb $
7   %
8   %   VOICEBOX is a MATLAB toolbox for speech processing.
9   %   Home page: ...
        http://www.ee.ic.ac.uk/hp/staff/dmb/voicebox/voicebox.html
10  %
11  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
12  %   This program is free software; you can redistribute it ...
        and/or modify
13  %   it under the terms of the GNU General Public License as ...
        published by
14  %   the Free Software Foundation; either version 2 of the ...
        License, or
15  %   (at your option) any later version.
16  %
17  %   This program is distributed in the hope that it will be useful,
18  %   but WITHOUT ANY WARRANTY; without even the implied warranty of
19  %   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
20  %   GNU General Public License for more details.
21  %
22  %   You can obtain a copy of the GNU General Public License from
23  %   http://www.gnu.org/copyleft/gpl.html or by writing to
24  %   Free Software Foundation, Inc.,675 Mass Ave, Cambridge, MA ...
        02139, USA.
25  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
26
27  nx=length(x(:));
28  if nargin<2 || isempty(win)
29      win=nx;
30  end
31  if nargin<4 || isempty(m)
32      m='';
33  end
34  nwin=length(win);
35  if nwin == 1
36      lw = win;
37      w = ones(1,lw);
38  else
```

```
39      lw = nwin;
40      w = win(:).';
41  end
42  if (nargin < 3) || isempty(inc)
43      inc = lw;
44  end
45  nli=nx-lw+inc;
46  nf = max(fix(nli/inc),0);   % number of full frames
47  na=nli-inc*nf+(nf==0)*(lw-inc);       % number of samples left over
48  fx=nargin>3 && (any(m=='z') || any(m=='r')) && na>0; % need an ...
        extra row
49  f=zeros(nf+fx,lw);
50  indf= inc*(0:(nf-1)).';
51  inds = (1:lw);
52  if fx
53      f(1:nf,:) = x(indf(:,ones(1,lw))+inds(ones(nf,1),:));
54      if any(m=='r')
55          ix=1+mod(nf*inc:nf*inc+lw-1,2*nx);
56          f(nf+1,:)=x(ix+(ix>nx).*(2*nx+1-2*ix));
57      else
58          f(nf+1,1:nx-nf*inc)=x(1+nf*inc:nx);
59      end
60      nf=size(f,1);
61  else
62      f(:) = x(indf(:,ones(1,lw))+inds(ones(nf,1),:));
63  end
64  if (nwin > 1)   % if we have a non-unity window
65      f = f .* w(ones(nf,1),:);
66  end
67  if nargout>1
68      if any(m=='E')
69          t0=sum((1:lw).*w.^2)/sum(w.^2);
70      elseif any(m=='A')
71          t0=sum((1:lw).*w)/sum(w);
72      else
73          t0=(1+lw)/2;
74      end
75      t=t0+inc*(0:(nf-1)).';
76  end
```

7.4 Frame Demo, *framedemo.m*

```
1  function [frames, hamframes] = framedemo(name)
2  % Show original, framed, and hamming framed
3  %
4  % Author: Maddie Briere, 2017
5  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
6
7  [signal, fs] = audioread(name);
8  signal = signal(:, 1); %take first channel
9  signal = signal (1:6000);
10
11 window = .025*fs;
12 inc = .010*fs;
```

```
13 [frames, times] = frame(signal, fs, window, inc, 1);
14 [hamframes, hamtimes] = frame(signal, fs, window, inc);
15
16 subplot(2, 1, 1);
17 plot(times', frames');
18 title('Framed signal')
19 xlabel('Sample')
20 ylabel('Amplitude')
21
22 subplot(2, 1, 2);
23 plot(hamtimes', hamframes');
24 title('Framed signal (Hamming)')
25 xlabel('Sample')
26 ylabel('Amplitude')
27
28 end
```

7.5 RFFT, *rfft.m*

```
1  function y=rfft(x)
2  % Edited by: Maddie Briere (2017)
3  % -> Alteration = hard-coded N (length) and D (dimension)
4  % -> Removed extra computing options
5  %
6  % RFFT     Calculate the DFT of real data Y=(X,N,D)
7  % Data is truncated/padded to length N if specified.
8  %   N even: (N+2)/2 points are returned with
9  %           the first and last being real
10 %   N odd:  (N+1)/2 points are returned with the
11 %           first being real
12 % In all cases fix(1+N/2) points are returned
13 % D is the dimension along which to do the DFT
14 %
15 %       Copyright (C) Mike Brookes 1998
16 %      Version: $Id: rfft.m 713 2011-10-16 14:45:43Z dmb $
17 %
18 %   VOICEBOX is a MATLAB toolbox for speech processing.
19 %   Home page: ...
       http://www.ee.ic.ac.uk/hp/staff/dmb/voicebox/voicebox.html
20
21 s=size(x);
22 if prod(s)==1
23     y=x
24 else
25     %Set dimension for transform
26     d=find(s>1,1);
27
28     %Set length to size of sample
29     n=s(d);
30
31     %Fourier transform
32     y=fft(x,n,d);
33
34     %Reshape
```

```
35      dimsize1 = prod(s(1:d-1));
36      dimsize2 = prod(s(d+1:end));
37      y=reshape(y, dimsize1, n, dimsize2);
38      s(d)=1+fix(n/2);
39      y(:,s(d)+1:end,:)=[];
40      y=reshape(y,s);
41  end
```

7.6 Filterbank, *filterbank.m*

```
1   function [c, tc]=filterbank(s,fs)
2   % FILTERBANK: Display results of filterbank analysis
3   % Authored by: Maddie Briere
4   %   Resource: VOICEBOX is a MATLAB toolbox for speech processing.
5   %   Home page: ...
        http://www.ee.ic.ac.uk/hp/staff/dmb/voicebox/voicebox.html
6
7
8   %% Set the initial variables
9   if nargin<2 fs=44100; end
10  nc=12; %% number of coefficients
11
12  %number of filters in filterbank
13  p=floor(3*log(fs));
14
15  %length of fft
16  n=pow2(floor(log2(0.03*fs)));
17
18  %Filter bank limits
19  fh=0.5;
20  fl=0;
21
22  %Frame increment by sample
23  inc= floor(n/2); %hamming
24
25  %% Frame the sound
26  [z,tc]=enframe(s,hamming(n),inc);
27
28  %% Take the fourier transform
29  f=rfft(z.');
30
31  %% Create mel Filter bank (centered on frequencies)
32  [m,a,b,freqs]=melbankmx(p,n,fs,fl,fh, 'M')
33  % m = filterbank magnitudes
34  % a = filterbank center frequencies
35  % b = lowest FFT bin with non-zero coefficient
36
37  %% Calculate the log of the energies in each filter bank
38  pw=f(a:b,:).*conj(f(a:b,:));
39  pth=max(pw(:))*1E-20;
40  ath=sqrt(pth);
41  y=log(max(m*abs(f(a:b,:)),ath));
42
43  %% Create output or display
```

```
if nargout<1
    [nf,ny]=size(y);
    imagesc(tc/fs,freqs, y);
    axis('xy');
    xlabel('Time (s)');
    ylabel('Center Frequency (Mel)');
    title('Mel-Frequency Filterbank Analysis -- Vowel e');
    map = (0:63)'/63;
    colormap([map map map]);
    colorbar;
end
```

7.7 Mel Bank, *melbankm.m*

```
function [x,mc,mn,coefs, mx]=melbankmx(p,n,fs,fl,fh)
%MELBANKM determine matrix for a mel/erb/bark-spaced filterbank ...
    [X,MN,MX]=(P,N,FS,FL,FH,W)
% Edited by: Maddie Briere, 2017
%
% Inputs:
%       p   number of filters in filterbank or the filter ...
    spacing in k-mel/bark/erb [ceil(4.6*log10(fs))]
%       n   length of fft
%       fs  sample rate in Hz
%       fl  low end of the lowest filter as a fraction of fs ...
    [default = 0]
%       fh  high end of highest filter as a fraction of fs ...
    [default = 0.5]
%
%      Copyright (C) Mike Brookes 1997-2009
%      Version: $Id: melbankm.m 713 2011-10-16 14:45:43Z dmb $
%
%   VOICEBOX is a MATLAB toolbox for speech processing.
%   Home page: ...
    http://www.ee.ic.ac.uk/hp/staff/dmb/voicebox/voicebox.html
%
% Note "FFT bin_0" assumes DC = bin 0 whereas "FFT bin_1" means ...
    DC = bin 1


if nargin < 5
     fh=0.5; % max freq is the nyquist
     if nargin < 4
        fl=0; % min freq is DC
     end
end
sfact=2;


mflh=[fl fh]*fs;
mflh=frq2mel(mflh);       % convert frequency limits into mel
melrng=mflh*(-1:2:1)';          % mel range
fn2=floor(n/2);     % bin index of highest positive frequency ...
    (Nyquist if n is even)
```

```
34
35 if isempty(p)
36     p=ceil(4.6*log10(fs));         % default number of filters
37 end
38
39
40 if p<1
41    p=round(melrng/(p*1000))-1;
42 end
43 melinc=melrng/(p+1);
44
45 %
46 % Calculate the FFT bins corresponding to [filter#1-low ...
       filter#1-mid filter#p-mid filter#p-high]
47 %
48
49 blim=mel2frq(mflh(1)+[0 1 p p+1]*melinc)*n/fs;
50 mc=mflh(1)+(1:p)*melinc;    % mel centre frequencies
51 b1=floor(blim(1))+1;            % lowest FFT bin_0 required ...
       might be negative)
52 b4=min(fn2,ceil(blim(4))-1);    % highest FFT bin_0 required
53 %
54 % now map all the useful FFT bins_0 to filter1 centres
55 %
56 pf=(frq2mel((b1:b4)*fs/n)-mflh(1))/melinc;
57
58 %
59 %  remove any incorrect entries in pf due to rounding errors
60 %
61 if pf(1)<0
62     pf(1)=[];
63     b1=b1+1;
64 end
65
66 if pf(end)≥p+1
67     pf(end)=[];
68     b4=b4-1;
69 end
70
71 fp=floor(pf);                  % FFT bin_0 i contributes to ...
       filters_1 fp(1+i-b1)+[0 1]
72 pm=pf-fp;                       % multiplier for upper filter
73 k2=find(fp>0,1);   % FFT bin_1 k2+b1 is the first to contribute ...
       to both upper and lower filters
74 k3=find(fp<p,1,'last');  % FFT bin_1 k3+b1 is the last to ...
       contribute to both upper and lower filters
75 k4=numel(fp); % FFT bin_1 k4+b1 is the last to contribute to any ...
       filters
76
77 if isempty(k2)
78     k2=k4+1;
79 end
80
81 if isempty(k3)
82     k3=0;
83 end
84
```

```
r=[1+fp(1:k3) fp(k2:k4)]; % filter number_1
c=[1:k3 k2:k4]; % FFT bin_1 - b1
v=[pm(1:k3) 1-pm(k2:k4)];
mn=b1+1; % lowest fft bin_1
mx=b4+1;  % highest fft bin_1

if b1<0
    c=abs(c+b1-1)-b1+1;     % convert negative frequencies into ...
        positive
end

% end
v=0.5-0.46/1.08*cos(v*pi);  % convert triangles to Hamming

% double all except the DC and Nyquist (if any) terms
msk=(c+mn>2) & (c+mn<n-fn2+2);  % there is no Nyquist term if n ...
    is odd
v(msk)=2*v(msk);

%
% sort out the output argument options
%
if nargout > 2
    x=sparse(r,c,v);
    if nargout == 4     % if exactly four output arguments, then
        coefs = mc;
        mc=mn;          % delete mc output for legacy code ...
            compatibility
        mn=mx;
    end
else
    x=sparse(r,c+mn-1,v,p,1+fn2);
end

%
% plot results if no output arguments or g option given
%
if ¬nargout % plot idealized filters
    ng=201;     % 201 points
    me=mflh(1)+(0:p+1)'*melinc;
    fe=mel2frq(me); % defining frequencies
    xg=mel2frq(repmat(linspace(0,1,ng),p,1).*repmat(me(3:end)-
    me(1:end-2),1,ng)+repmat(me(1:end-2),1,ng));

    v=1-abs(linspace(-1,1,ng));
    v=0.5-0.46/1.08*cos(v*pi);  % convert triangles to Hamming

    v=v*sfact;  % multiply by 2 if double sided
    v=repmat(v,p,1);

    plot(xg',v','b');
    set(gca,'xlim',[fe(1) fe(end)]);
    xlabel(['Frequency (' xticksi 'Hz)']);
end
```

7.8 Melcepst, *melcepst.m*

```
function [c, tc]=melcepst(s,fs)
% MELCEPST Calculate the mel cepstrum of a signal C=(S,FS)
%   param: s Signal data
%   param: fs Sampling rate
%  Resources: VOICEBOX  ...
    (http://www.ee.ic.ac.uk/hp/staff/dmb/voicebox/voicebox.html)
%
% Author: Maddie Briere, 2017
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%% Set the initial variables
if nargin<2 fs=44100; end
nc=12; %% number of coefficients

%number of filters in filterbank
p=floor(3*log(fs));

%length of window
n=pow2(floor(log2(0.03*fs)));

%Filter bank limits
fh=0.5;
fl=0;

%Frame increment by sample
inc= floor(n/2); %hamming

%% Pre-emphasis
%s = filter([ 1 -.95], 1, s);

%% Frame the sound
[z,tc]=enframe(s,hamming(n),inc);
%% Pre-emphasis
z = filter([ 1 -.95], 1, z);
%% Take the fourier transform
f=rfft(z.');

%% Create mel Filter bank (centered on frequencies)
[m,a,b]=melbankm(p,n,fs,fl,fh);
% m = filterbank magnitudes
% a = filterbank center frequencies
% b = lowest FFT bin with non-zero coefficient

%% Calculate the log of the energies in each filter bank
pw=f(a:b,:).*conj(f(a:b,:));
pth=max(pw(:))*1E-20;
ath=sqrt(pth);
y=log(max(m*abs(f(a:b,:)),ath));

%% Discrete Cosine Transform (last step)
% When plotted, look the same.
c = rdct(y).';

```

```
54 %% Fit the coefficients
55 nf=size(c,1);
56 nc=nc+1; %(extra added on at 0)
57 if p>nc
58    c(:,nc+1:end)=[];
59 elseif p<nc
60    c=[c zeros(nf,nc-p)];
61 end
62 c = (lifter(c', -.6))';
63
64 %% Exclude first coefficient
65 c(:,1)=[];
66
67 %% Create output or display
68 if nargout<1
69    [nf,nc]=size(c);
70    ci=(1:nc);
71    imagesc(tc/fs,ci,c.');
72    axis('xy');
73    xlabel('Time (s)');
74    ylabel('Mel-cepstrum coefficient');
75    map = (0:63)'/63;
76    colormap([map map map]);
77    colorbar;
78 end
```

7.9 RDCT, *rdct.m*

```
1  function y=rdct(x,n,a,b)
2  %RDCT     Discrete cosine transform of real data Y=(X,N,A,B)
3  % Data is truncated/padded to length N.
4  %
5  % This routine is equivalent to multiplying by the matrix
6  %
7  %      Copyright (C) Mike Brookes 1998
8  %      Version: $Id: rdct.m 713 2011-10-16 14:45:43Z dmb $
9  %
10 %   VOICEBOX is a MATLAB toolbox for speech processing.
11 %   Home page: ...
       http://www.ee.ic.ac.uk/hp/staff/dmb/voicebox/voicebox.html
12 %
13 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
14 %   This program is free software; you can redistribute it ...
       and/or modify
15 %   it under the terms of the GNU General Public License as ...
       published by
16 %   the Free Software Foundation; either version 2 of the ...
       License, or
17 %   (at your option) any later version.
18 %
19 %   This program is distributed in the hope that it will be useful,
20 %   but WITHOUT ANY WARRANTY; without even the implied warranty of
21 %   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
22 %   GNU General Public License for more details.
```

```
%
%   You can obtain a copy of the GNU General Public License from
%   http://www.gnu.org/copyleft/gpl.html or by writing to
%   Free Software Foundation, Inc.,675 Mass Ave, Cambridge, MA ...
    02139, USA.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

fl=size(x,1)==1;
if fl x=x(:); end
[m,k]=size(x);
if nargin<2 n=m;
end
if nargin<4 b=1;
    if nargin<3 a=sqrt(2*n);
    end
    end
if n>m x=[x; zeros(n-m,k)];
elseif n<m x(n+1:m,:)=[];
end

x=[x(1:2:n,:); x(2*fix(n/2):-2:2,:)];
z=[sqrt(2) 2*exp((-0.5i*pi/n)*(1:n-1))].';
y=real(fft(x).*z(:,ones(1,k)))/a;
y(1,:)=y(1,:)*b;
if fl y=y.'; end
```

7.10 Lifter, *lifter.m*

```
function y = lifter(x, lift)
% y = lifter(x, lift)
%   Author: Maddie Briere, 2017
%
%   Apply lifter to matrix of cepstra (one per column)
%   lift = exponent of x i^n liftering
%
%   Resource: Columbia Speech Analysis (dpwe@ee.columbia.edu)

if nargin < 2;   lift = 0.6; end   % liftering exponent
if nargin < 3;   invs = 0; end      % flag to undo liftering

[ncep, nfrm] = size(x);

if lift == 0
  y = x;
else
  liftwts = [1, ([1:(ncep-1)].^lift)];
  y = diag(liftwts)*x;

end
```

7.11 Av Norm Cep, *avnormcep.m*

```
function [average] = avnormcep( ceps )
% Take average cepstral coefficients for
% short excerpt of sound

[s1, s2] = size(ceps);
sum = zeros(1,s2);

for x = 1:s1
    sum = sum + ceps(s1, :);
end

average = (sum/s1)';
average = average./max(average);

end
```

7.12 Plot Ceps, *plotceps.m*

```
ah1 = melcepst(audioread('ah1.wav'));
ah2 = melcepst(audioread('ah2.wav'));
ah3 = melcepst(audioread('ah3.wav'));
ah4 = melcepst(audioread('ah4.wav'));
ah5 = melcepst(audioread('ah5.wav'));
ah6 = melcepst(audioread('ah6.wav'));
ah7 = melcepst(audioread('ah7.wav'));
ah8 = melcepst(audioread('ah8.wav'));

o1 = melcepst(audioread('o1.wav'));
o2 = melcepst(audioread('o2.wav'));
o3 = melcepst(audioread('o3.wav'));
o4 = melcepst(audioread('o4.wav'));
o5 = melcepst(audioread('o5.wav'));
o6 = melcepst(audioread('o6.wav'));
o7 = melcepst(audioread('o7.wav'));
o8 = melcepst(audioread('o8.wav'));

e1 = melcepst(audioread('e1.wav'));
e2 = melcepst(audioread('e2.wav'));
e3 = melcepst(audioread('e3.wav'));
e4 = melcepst(audioread('e4.wav'));
e5 = melcepst(audioread('e5.wav'));
e6 = melcepst(audioread('e6.wav'));
e7 = melcepst(audioread('e7.wav'));
e8 = melcepst(audioread('e8.wav'));

ah1 = avnormcep(ah1);
ah2 = avnormcep(ah2);
ah3 = avnormcep(ah3);
ah4 = avnormcep(ah4);
ah5 = avnormcep(ah5);
ah6 = avnormcep(ah6);
ah7 = avnormcep(ah7);
ah8 = avnormcep(ah8);
```

```
36
37  o1 = avnormcep(o1);
38  o2 = avnormcep(o2);
39  o3 = avnormcep(o3);
40  o4 = avnormcep(o4);
41  o5 = avnormcep(o5);
42  o6 = avnormcep(o6);
43  o7 = avnormcep(o7);
44  o8 = avnormcep(o8);
45
46  e1 = avnormcep(e1);
47  e2 = avnormcep(e2);
48  e3 = avnormcep(e3);
49  e4 = avnormcep(e4);
50  e5 = avnormcep(e5);
51  e6 = avnormcep(e6);
52  e7 = avnormcep(e7);
53  e8 = avnormcep(e8);
54
55  a = plot(ah1, 'r-');
56  hold on
57
58  b = plot(e1, 'b-');
59  hold on
60
61  c = plot(o1, 'g-');
62  hold on
63
64  legend('Vowel \a\', 'Vowel \i\', 'Vowel \u\')
65
66  plot(ah2, 'r-');
67  hold on
68
69  plot(ah3, 'r-');
70  hold on
71
72  plot(ah4, 'r-');
73  hold on
74
75  plot(ah5, 'r-');
76  hold on
77
78  plot(ah6, 'r-');
79  hold on
80
81  plot(ah7, 'r-');
82  hold on
83
84  plot(ah8, 'r-');
85  hold on
86
87  plot(e2, 'b-');
88  hold on
89
90  plot(e3, 'b-');
91  hold on
92
```

```
plot(e4, 'b-');
hold on

plot(e5, 'b-');
hold on

plot(e6, 'b-');
hold on

plot(e7, 'b-');
hold on

plot(e8, 'b-');
hold on

plot(o2, 'g-');
hold on

plot(o3, 'g-');
hold on

plot(o4, 'g-');
hold on

plot(o5, 'g-');
hold on

plot(o6, 'g-');
hold on

plot(o7, 'g-');
hold on

plot(o8, 'g-');
hold on

title('Normalized, average cepstral coefficients');
xlabel('Cepstral coefficient');
ylabel('Amplitude');
```

7.13 Plot Ave Ceps, *plotaveceps.m*

```
ahave = zeros(1,12);
oave = zeros(1,12);
eave = zeros(1,12);
for x=1:8
    ahave = ahave + ...
        avnormcep(melcepst(audioread(sprintf('ah$\%$d.wav', x))));
    oave = oave + ...
        avnormcep(melcepst(audioread(sprintf('o$\%$d.wav', x))));
    eave = eave + ...
        avnormcep(melcepst(audioread(sprintf('e$\%$d.wav', x))));
end
ahave = ahave./8;
```

```
oave = oave./8;
eave = eave./8;
ceps = 1:12;

hold on
a= plot(ceps, ahave, 'r-');
b= plot(ceps, eave, 'b-');
c= plot(ceps, oave, 'g-');
hold off

legend('Vowel \a\','Vowel \i\', 'Vowel \u\');
title('Normalized, average cepstral coefficients');
xlabel('Cepstral coefficient');
ylabel('Amplitude');
```

7.14 Predict Vowel, *predictvowel.m*

```
function [guess] = predictvowel(s, fs)
% Predict which vowel the input sound is.
ahave = zeros(12,1);
oave = zeros(12,1);
eave = zeros(12,1);
for x=1:8
    a = avnormcep(melcepst(audioread(sprintf('ah%d.wav', x))));
    ahave = ahave + a;

    o = avnormcep(melcepst(audioread(sprintf('o%d.wav', x))));
    oave = oave + o;

    e = avnormcep(melcepst(audioread(sprintf('e%d.wav', x))));
    eave = eave + e;
end
ahave = ahave./8;
oave = oave./8;
eave = eave./8;

save = avnormcep(melcepst(s, fs));

aguess = corr(ahave, save);
oguess = corr(oave, save);
eguess = corr(eave, save);

if aguess>oguess & aguess>eguess
    guess = 'a';
end

if oguess>aguess & oguess>eguess
    guess = 'o';
end

if eguess>aguess & eguess>oguess
    guess = 'e';
end

```

```
38 end
```

References

[1] Radford university statistics: *Linear Correlation Explanation*, 2002.

[2] Speech signal processing: Cepstal analysis of speech, 2011.

[3] Acoustic analysis of vowels (chapter 2), 2017.

[4] Anatomy of the mouth, 2017.

[5] Filterbank analysis, 2017.

[6] General phonetics: Formants, 2017.

[7] A guide to tf layers: Building a convolutional neural network, 2017.

[8] Mohamed A. Jiang H. Penn G. Abdel-Hamid, O. Applying convolutional neural networks concepts to hybrid nn-hmm model for speech recognition.

[9] Noelia Alcaraz Meseguer. Speech analysis for automatic speech recognition, 2009.

[10] M. Brooks. Voicebox: Speech processing toolbox for matlab, 2017.

[11] Z. Ghahramani. An introduction to hidden markov models and bayesian networks, 2001.

[12] Lee H. Han, Y. Acoustic scene classification using convolutional neural network and multiple-width frequency-delta data augmentation, Augusut 2015.

[13] W. Holmes, J. Holmes. *Speech Synthesis and Recognition*. Tailor Francis, 2001.

[14] Cole R. & Fanty M. Hosom, J. Speech recognition using neural networks at the center for spoken language understanding, 1999.

[15] Imlee. Hmm speech recognition in matlab, 2016.

[16] B Kamble. Speech recognition using artificial neural network – a review, 2016.

[17] Cortes C. Burges C. LeCun, Y. The mnist database of handwritten digits, 2010.

[18] J. Lyons. jameslyons/python.speech.features.

[19] Pitz M. Schlüter R. Ney H. Molau, S. Computing mel-frequency cepstral coefficients on the power spectrum, 2001.

[20] Kevin Murphy. Hmm toolkit, 2005.

[21] Hamid S. Oppenheim A., Willsky A. *Signals and Systems*. 2nd edition, 1997.

[22] M. Shire. Plp and rasta (and mfcc, and inversion) in matlab, 2012.

[23] Evermann G. Gales M. Hain T. Kershaw D. Liu X. Moore G. Odell J. Ollason D. Povey D. Valtchev V. Woodland P. Young, S. The htk book version 3.4, 2006.