2016

A Wavelet Transform Module for a Speech Recognition Virtual Machine

Euisung Kim
Minnesota State University Mankato

A Wavelet Transform Module for a Speech Recognition Virtual Machine

by

Euisung Kim

A Thesis Submitted in Partial Fulfillment of the

Requirements for the Degree of

Master of Science

in

Electrical Engineering

Minnesota State University, Mankato

Mankato, Minnesota

May 16, 2016

This thesis paper has been examined and approved.

Examining Committee:

Dr. Rebecca Bates, Chairperson

Dr. Vincent Winstead

Dr. Qun (Vincent) Zhang

Acknowledgment

I must give my special thanks to my thesis adviser, Dr. Bates, for her great care and discipline. I met her just two years ago when I had started grad school. She gave me an opportunity to work as her graduate assistant for her research project and I became interested in her research. Unfortunately, my energy didn't last long and I had a meltdown due to personal problems. I could have easily lost the assistant opportunity and dropped out of school if she hadn't shown her trust and encouraged me. I hadn't even originally planned on writing a thesis because it seemed unrealistic to me. It was her spending a great deal of her time advising me in both research and writing that made me see theres hope. It would have been entirely impossible to complete this thesis work without her care and discipline.

I would like to thank Dr. Winstead. He has been very supportive of me during my grad school and helpful in technical difficulties. I especially want to show thanks for the time he told me not to lose hope when I thought I had came to a dead end and had almost given up on my thesis work.

I would also like to thank Dr. Zhang. Many of his insightful in-class talks during my graduate experience had a very positive influence on my thesis work. He also gave me great feedback during my oral defense and showed me how I could improve my presentation.

I am also very thankful that Dr. Hardwick and Dr. Kelley shared their time to answer my questions throughout my last semester in grad school and showed their support during my defense.

Finally, I must thank my family members, Jaegyu Kim, Jungsoon Kim, and Juhee Kim. I know what they had to give up for me to continue my education in the US and how they tried to keep me from worrying about their sacrifices they made so I can focus only on my work. Over the last six and half years, I learned not only how to study, but also-more importantly-how to give support and show love.

Abstract

This work explores the trade-offs between time and frequency information during the feature extraction process of an automatic speech recognition (ASR) system using wavelet transform (WT) features instead of Mel-frequency cepstral coefficients (MFCCs) and the benefits of combining the WTs and the MFCCs as inputs to an ASR system. A virtual machine from the Speech Recognition Virtual Kitchen resource (www.speechkitchen.org) is used as the context for implementing a wavelet signal processing module in a speech recognition system. Contributions include a comparison of MFCCs and WT features on small and large vocabulary tasks, application of combined MFCC and WT features on a noisy environment task, and the implementation of an expanded signal processing module in an existing recognition system. The updated virtual machine, which allows straightforward comparisons of signal processing approaches, is available for research and education purposes.

Table of Contents

Table of Figures

List of Tables

Chapter 1

Introduction

Humans have developed machines to share their tasks. Although machines cannot outperform humans at many complex tasks, it is known that machines can run simple and repetitive computations very quickly. Unfortunately, the way humans interact with machines is far from ideal because mechanical peripherals are the main means of exchanging data. By using them, it is assumed that users can press buttons and move mouse cursor positions. These are difficult constraints when people with disabilities interact with machines. Using mechanical peripherals is also not natural to humans because humans do not interact with each other by pressing buttons and moving mouse cursor positions. This brings challenges for human-friendly interactions between humans and machines. In order to tackle the challenges, human speech enabled interfaces such as SIRI have been developed [1, 2, 3, 4]. Machines can easily handle digitized signals from the mechanical peripherals (ASCII code from a keyboard, for instance) without additional conversion processes. Human speech, on the other hand, is not so natural to the machines because it is a continuous signal and additional conversion processes are needed to digitize the speech signal and then transform that input to appropriate commands. Feature extraction is a part of the digitization process and extracts characteristics of a *phone*, the smallest unit of speech typically labeled by linguists. Feature coefficients should contain both time and frequency information because they both contain information about the speech signal.

Fourier transforms (FT) allow the representation of the frequency information of the speech signal. One of the most popular feature extraction methods based on the FT for speech processing is Mel-frequency cepstral coefficients (MFCCs) [4, 5]. The MFCC method is based on the cepstrum, the result of taking the inverse Fourier transform of the log magnitude of the FT. It is useful to separate the spectra of vocal excitation and the vocal tract of the speech signal for better phone recognition. However, the speech signal has the spectra of the two multiplied together. By taking the log of the spectra, they have an additive relationship rather than multiplicative. Taking the inverse FT of linearly combined log spectra of the vocal excitation and the vocal tract separates the information from vocal excitation and the vocal tract and returns the information to the time domain. Unfortunately the output of the FT has no time information from the original signal because the FT operates in the frequency domain. The short time Fourier transform (STFT) can be used to partially solve this problem [6]. The STFT is applied to time segments of the signal. The size of the time segments correspond to time resolutions. This means that time-frequency resolutions are constant because the size of the time segments do not change. Since time and frequency information of speech is not evenly concentrated, having constant time-frequency resolutions may not represent speech well.

An alternative approach, wavelet transforms (WTs) [7], can provide both time and frequency information with varying resolutions at different decomposition levels. The WT is based on the idea of representing a signal as shifted (related to time) and scaled (related to frequency) versions of a small wave called a wavelet. The simplest example of a wavelet is the Haar wavelet shown in Figure 1.1. An advantage of the WT method is that it represents local time-frequency content with varying resolutions, which has potential for a higher degree of flexibility and an increase in algorithmic efficiency when desired time-frequency components are not evenly concentrated. Since

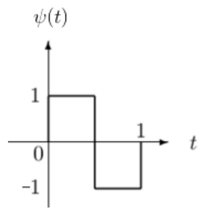


Figure 1.1: The Haar Wavelet [8]

the acoustic information extracted by wavelet transforms is different from MFCCs, the following research questions can be asked in the context of automatic speech recognition (ASR) for both clean and noisy speech environments:

1. Can a wavelet transform signal processing module be added to an existing ASR system?
2. Can WT features match the performance of MFCCs?
3. Are there benefits to combining the MFCC and WT methods?

This thesis begins with some background to describe the process of automatic speech recognition including the two feature extraction methods. The data used in experiments is presented, followed by a description of the experimental approach to answer the research questions. Recognition results for WT coefficients and MFCCs on three data sets are described and discussed in Chapter 5. In addition, two approaches to combining features are presented with a comparison of results.

A virtual machine with both MFCC and wavelet transform modules from the Speech Recognition Virtual Kitchen resource (SRVK) is used to make comparisons and to demonstrate recognition performance improvements when addressing noisy conditions. Along with addressing the research questions, a major contribution of this work is the implementation of a new signal processing module in a publically available, open source virtual machine, which can be used for further research or education applications.

Chapter 2

Background

In this chapter, the major components of an automatic speech recognition (ASR) system shown in Figure 2.1 will be explained. The training speech waveform first goes through signal processing of some sort, whether MFCCs or WTs. Then, the following training block takes the output from the signal processing of the training speech waveform, along with matching transcriptions, to produce acoustic models. Similarly, training text is fed into its training block to produce language models. Finally, the test speech waveform goes through the same signal processing and gets fed into the decoding block. The decoding block then takes these three inputs (acoustic model, language model, and the output from the signal processing of the test speech waveform) and produces hypothesized word sequences for the input waveforms.

2.1 Signal Processing

The signal processing, or feature extraction, portion of an ASR system takes the speech and splits it into many utterances, typically during silence, or in the case of multiple speakers, during speaker changes. Utterances can have different lengths depending on duration, but are analogous to sentences. Utterances contain one or more words, which are built with phones (the smallest labeled unit of sound). An

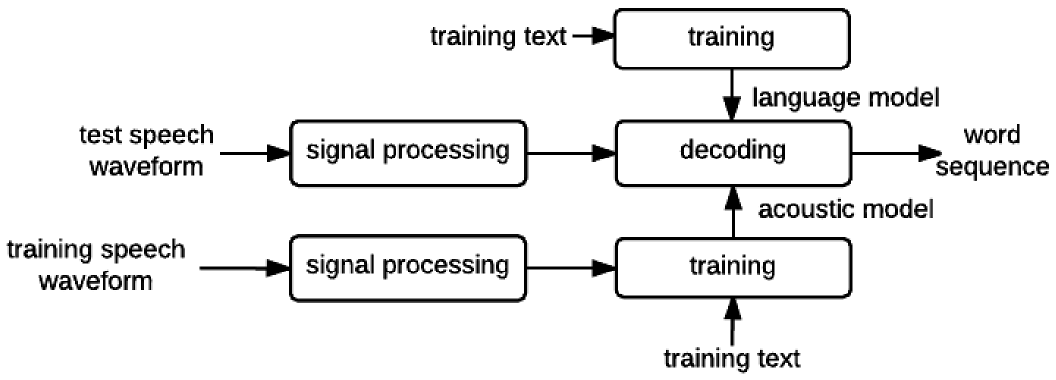


Figure 2.1: A Typical ASR System

example utterance is “Cats are awesome”. A sequence of vectors which represent the acoustic characteristics of phones from each utterance can be generated using feature extraction methods such as Mel-frequency cepstral coefficients (MFCCs) and the wavelet transform (WT). Filter banks are typically used in both MFCC and WT method implementations. A filter bank refers to a series of filters that takes an input and separates it into multiple components based on frequency. A single frequency sub-band of the original signal is carried by each filter. The MFCC method applies fixed-duration time windowing and the Mel filter bank to obtain associated time information from the windowing and frequency information that differentiates phones. The WT method, on the other hand, provides varying time-frequency resolutions at different decomposition levels. Sub-band coding describes the basic behaviors of the decomposition using a pair of filter banks, created from a low pass filter and a high pass filter.

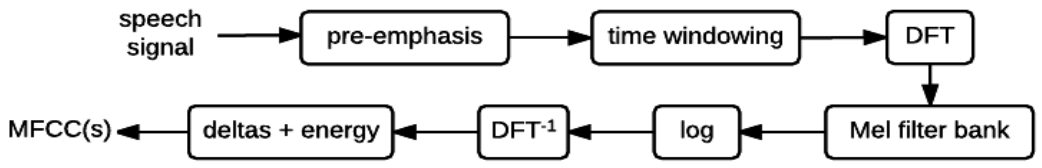


Figure 2.2: The MFCC Method

2.1.1 Mel-frequency Cepstral Coefficients (MFCCs)

The MFCC method consists of several sub-processes shown in Figure 2.2. The first sub-process is called pre-emphasis and is responsible for increasing the energy levels in the high frequencies. Since more energy can be found at the low frequencies than at the higher frequencies, having more detail in the high frequencies can improve speech signal processing. The reason more energy is in the low frequencies is due to the glottal pulse used to generate human speech. This energy drop between high and low frequencies is called spectral tilt [9].

The second sub-process is called windowing. Small windows of approximately 20 to 25ms are applied to an utterance signal with a 10ms frame shift rate to simulate piece-wise stationarity of the utterance signal when it is considered non-stationary because of the changing sounds. Phones are generally a minimum of three windows long so piece-wise stationarity holds for the windows. The feature extraction process is applied to obtain coefficients for each time window. The coefficients will have higher time resolutions if the window size is small and have lower time resolutions if the window size is large. The Hamming window, $w[n]$, is the most common window used in the MFCC method, where

$$w[n] = \begin{cases} 0.53 - 0.46\cos(\frac{2\pi n}{L}), & \text{if } 0 \le n \le L-1 \\ 0, & \text{else.} \end{cases}$$

The third sub-process is called the discrete Fourier transform (DFT), which enables representations of a signal or the windowed utterance signal in the frequency domain. The DFT is defined as

$$\sum_{n=0}^{N-1} x[n]e^{-j\frac{2\pi kn}{N}}.$$

A fast Fourier transform (FFT) algorithm is typically used to implement the DFT. The FFT reduces the complexity of computing the DFT from $O(n^2)$ to $O(n\log(n))$, where n is the data size.

The next sub-processes are the mel filter bank and application of the logarithm. The mel scale separates or compresses frequencies into bands that are perceived as equal by listeners. Since human hearing is more sensitive below roughly 1kHz, incorporating this into the mel scale improves speech signal processing. The mel scale is linear below 1kHz but logarithmic above 1kHz. Implementation of this can be achieved by using a bank of filters collecting each frequency band. One implementation has the first 10 filters spaced linearly below 1kHz and then additional filters are spaced logarithmically above 1kHz. The mel frequency can be obtained by

$$\text{mel}(f) = 1127\ln(1 + \frac{f}{700})$$

where f is the raw acoustic frequency. Taking the log of the output separates the vocal excitation from the vocal tract information and it may help address power variations too.

The next sub-process is the inverse discrete Fourier transform (DFT^{-1}). The inverse DFT enables representations of a signal or the windowed utterance signal in the time domain. So this overall process can be defined as the inverse DFT of the log magnitude of the DFT of the signal:

$$c[n] = \sum_{n=0}^{N-1} \log(|\sum_{n=0}^{N-1} x[n]e^{-j\frac{2\pi kn}{N}}|)e^{j\frac{2\pi kn}{N}}$$

where $c[n]$ is a cepstral vector and $x[n]$ is the windowed signal.

The final sub-process is delta and energy computation. The energy computation is literally obtaining the energy of the framed signal:

$$\text{Energy} = \sum_{t=t_1}^{t_2} x^2[n]$$

where t_1 and t_2 are frame starting and end points. The delta coefficients are computed by taking the difference between nearby frames for each element of the cepstral vector

$$d[n] = \frac{c[n+1] - c[n-1]}{2}$$

where $d[n]$ is the delta value and $c[n]$ is the cepstrum. Typically a delta order of two is used. The first delta computation represents the rate of change and the second delta computation represents a rate of change of the first rate of change.

The results of this process are cepstral coefficients with deltas which can be used as inputs to an ASR system. The cepstral coefficients represent information solely about the vocal tract filter, cleanly separated from information about the glottal source. Signal processing and recognition performance using MFCCs is fairly good [4, 10], primarily because the process has been used and tuned by many people to optimize the features for this application.

2.1.2 Wavelet Transform (WT)

Wavelet transforms were motivated by the shortcomings of the Fourier transform. When the FT represents a signal in the frequency domain, it can not tell where those frequency components are present in time. Cutting the signal at a certain moment in time (windowing) and transforming that into the frequency domain to obtain a relevant time sequence of frequency information is equivalent to convolving the signal and the cutting window, resulting in possible smearing of frequency components along the frequency axis [11, 12]. Wavelet transforms address this by using small waves of integral 1 with a finite duration and represented by

$$\gamma(s,\tau) = \int f(t)\psi^{*}_{s,\tau}(t)dt \tag{2.1.1}$$

where $f(t)$, the signal to be represented, gets decomposed into a set of basis functions $\psi_{s,\tau}(t)$ called the mother wavelet

$$\psi_{s,\tau}(t) = \frac{1}{\sqrt{s}}\psi(\frac{t-\tau}{s})$$

with scale parameter s and shift parameter τ. $\frac{t-\tau}{s}$ from the mother wavelet definition determines different scaled and shifted versions of the wavelet. $\psi_{s,\tau}(t)$ can be selected from a range of wavelet types. Since the mother wavelet can be either real-valued or complex-valued, the conjugate term is added to compensate the inversion of imaginary parts after transforming [13], and this is shown in Equation 2.1.1. Obtaining the correlation between the the signal $f(t)$ and the scaled and shifted versions of the mother wavelet results in wavelet coefficients.

However, in practice there are problems with wavelet transforms. First, since calculating the wavelet transform is done by continuously shifting a continuously scalable function over a signal, it produces a lot of redundancy. Second, by definition the WT needs an infinite number of wavelets for covering all frequency bands. Third, the WT generally has no analytical solutions for most functions, but instead can only be solved with numerical computation requiring some sort of fast algorithm. A piecewise continuous version of the function called the discrete mother wavelet therefore should be considered:

$$\psi_{j,k}(t) = \frac{1}{\sqrt{s_0^j}}\psi(\frac{t - ks_0^j}{s_0^j})$$

where j and k are integers and $s_0 > 1$ represents fixed integer scale steps. This discretized wavelet can now be sampled at discrete intervals. In order for the Nyquist rate to hold true for any given frequency of the signal, dyadic sampling is typically chosen as an optimal sampling pattern. A sampling pattern is dyadic if the mother wavelet is shifted by $k2^j$ and scaled by 2^j. Therefore, the value of s_0 is usually 2. The discrete wavelet transform (DWT) can be thought of as a special case of the WT with wavelets scaled and shifted by factors of powers of 2. The DWT coefficients can then be obtained from

$$\gamma(s,\tau) = \int f(t)\frac{1}{\sqrt{2^s}}\psi(\frac{t - \tau 2^s}{2^s})dt$$

where the mother wavelet is

$$\psi_{s,\tau}(t) = \frac{1}{\sqrt{2^s}}\psi(\frac{t - \tau 2^s}{2^s}).$$

Even though the wavelet can be sampled at discrete intervals with the discretized wavelets, an infinite number of scales and shifts are still required. The way to solve this problem is to use a finite number of discrete wavelets with a scaling function,

$\varphi(t)$, introduced by Mallat [7]

$$\varphi(t) = \sum_{j,k} \gamma(j,k)\psi_{j,k}(t),$$

which covers the remaining wavelet spectra of the signal.

In order to use these equations, there are two conditions that should be met: *admissibility* and *regularity*. The *admissibility* condition is defined as

$$\int \frac{|\psi(w)|^2}{|w|} dw < +\infty$$

where $\psi(w)$ is the FT of $\psi(t)$. The equation means that $\psi(t)$ vanishes at the zero frequency producing a band-pass-like spectrum

$$|\psi(w)|^2 \ |_{w=0} = 0 \leftrightarrow \int \psi(t)dt = 0$$

where the average value of the wavelet in the time domain is zero.

The second condition is *regularity* [14]. Notice in Equation 2.1.1 that the WT of a one-dimensional function, $f(t)$, becomes a two-dimensional function, $\gamma(s,\tau)$, a two-dimensional function, $f(t_1,t_2)$, becomes a four-dimensional function, $\gamma(s_1,s_2,\tau_1,\tau_2)$, and so on. The time-bandwidth of the wavelet transform product is the square of the original signal, which causes problems in most WT applications. This undesired property motivates regularity: to decrease the value of wavelet transforms with a factor s.

Implementing the WTs requires using a filter bank. The scaling function behaves like a low pass filter and covers the remaining spectrum of the wavelets by reducing an infinite set of wavelets into a finite set of wavelets. Taking as a given that Fourier theory says compression in time is equivalent to stretching the spectrum in the frequency domain, a time compression of the wavelet by a factor of two will stretch the spectrum of the wavelet by a factor of two. This means the finite spectrum of the signal can be described with the spectra of scaled wavelets. Therefore, a series of scaled wavelets and a scaling function can be seen as a filter bank.

Implementing such a filter bank can be described by sub-band coding. One way to apply this is to build a number of bandpass filters with different ranges of bands. This could take a long time because all individual filters need to be specifically designed. Another way to implement this is to have a low pass filter (LPF) and a high pass filter (HPF). By iteratively applying the pair of filters on the outputs of each low pass filter, a desired number of bands can be obtained. Down-sampling is applied after each filtering in order to keep the output of a discrete WT the same size as the original signal. The advantage of this building method is the simplicity of designing filters as only two filters are needed, a low pass and a high pass.

This method requires a special technique called the lifting scheme [17], where the LPF and HPF pair can be achieved by taking the average and difference between polyphase representation of even and odd components of the original signal. The Z-transforms of the polyphase components are used to take the average and difference between the even and odd polyphase components. Since the LPF and HPF are achieved by taking the average and difference respectively, the outputs are called approximation coefficients when taking the average and detail coefficients when taking the difference. There is another more intuitive way to describe why the coefficients are called the approximation and detail coefficients. The low frequency signal has a longer period than the high frequency signal and thus represents a close approximation of the original signal. However, the high frequency signal represents detailed information about the original signal.

The resulting coefficients from each sub-band represent correlations between the original signal and a wavelet that is shifted and scaled by some factor of τ and s. τ and s change as they go through a series of filters. Convolving the signal and the filters results in wavelet coefficients. Since the τ and s factors are not constant, the coefficients from each band have different (varying) time-frequency resolutions, unlike MFCCs.

The wavelet type (shape) can also further characterize the acoustic features and an optimum type for a given context can be found experimentally. There are four different wavelet shapes with different scales that are likely to be useful for ASR applications. Figures 2.3, 2.4, 2.5, and 2.6 show sample wavelets from the symlet (sym), Coiflet (coif), Daubechies (db), and biorthogonal (bior) families. Different wavelet scales correspond to different time-frequency resolutions. The larger wavelet scales (narrower wavelet width) give better the time resolution. Similarly smaller wavelet scales (wider wavelet width) give better the frequency resolution.

Wavelet types can be described by orthogonality, symmetry, vanishing moment condition, and compact support. Wavelets are orthogonal if the inner products of a mother wavelet and its shifted and scaled version of wavelet are zero, which enables decomposing a signal into non-overlapping sub-frequency bands [19]. In Figure 2.6, wavelet pairs are used for decomposition (left) and reconstruction (right) because biorthogonal wavelets require two scaling functions for a relaxed orthogonal or biorthogonal condition [16]. The symmetry property of a wavelet ensures that the mother wavelet can serve as a linear phase filter; lacking this property can cause phase distortion [19]. The moment vanishing condition is defined by the number of vanishing moments in a wavelet. Consider expanding Equation 2.1.1 into a Taylor

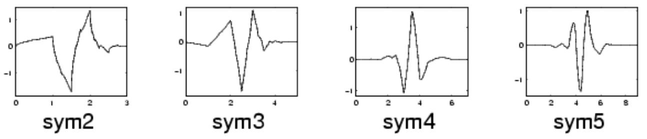


Figure 2.3: Symlet Wavelet Family with Scales of 2, 3, 4, and 5 [18]

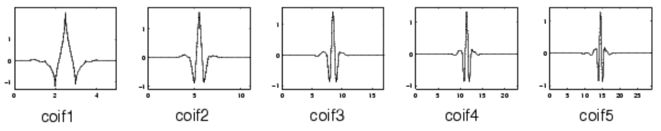


Figure 2.4: Coiflet Wavelet Family with Scales of 1, 2, 3, 4, and 5 [18]

series of order n

$$\gamma(s,\tau) = \frac{1}{\sqrt{s}}[f(\tau)M_0 s + \frac{f^{(1)}(\tau)}{1!}M_1 s^2 + \frac{f^{(2)}(\tau)}{2!}M_2 s^3 + ... + \frac{f^{(n)}(\tau)}{n!}M_n s^{n+1}] \quad (2.1.2)$$

where $f^{(p)}$ is p^{th} derivative of f and M_p is a wavelet moment defined as

$$M_p = \int t^p \psi(t) dt.$$

A wavelet has N vanishing moments if N number of M_p terms go to zero in Equation 2.1.2 [12]. The more vanishing moments a wavelet has, the better the scaling function can represent complex functions. Equivalently, the higher the number of zero moments, the higher the number of zero derivatives and the smoother the signal decays from the center frequency to the zero frequency in the frequency domain [20]. The db wavelet has N vanishing moments, the bior wavelet has $N-1$ vanishing moments, the coif wavelet has $2N$ vanishing moments, and the sym wavelet has N vanishing moments but is more symmetrical than the db wavelet. If wavelets are

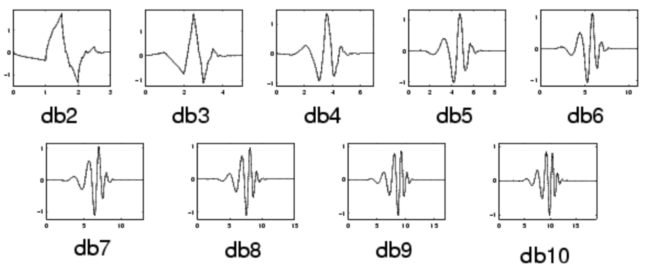


Figure 2.5: Daubechies Wavelet Family with Scales of 2, 3, 4, 5, 6, 7, 8, 9, and 10 [18]

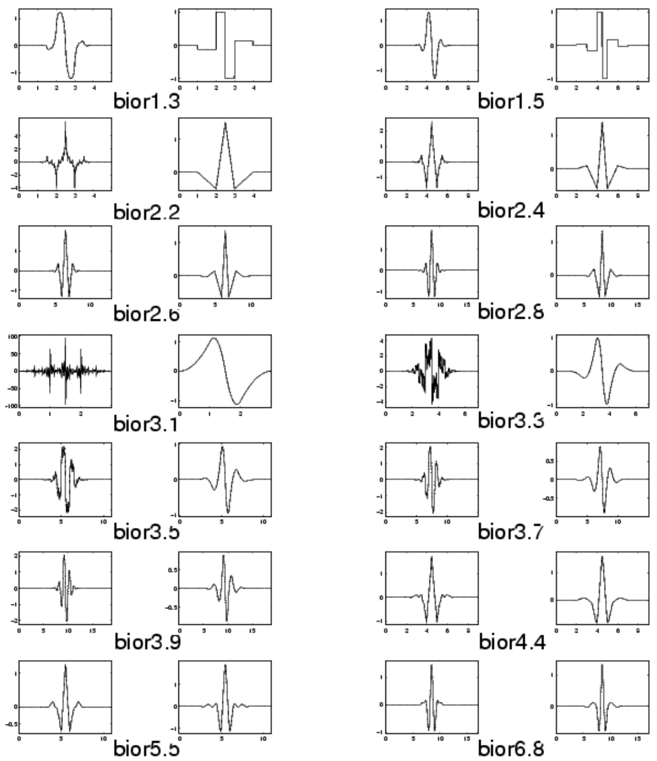


Figure 2.6: Biorthogonal Wavelet Family with Scales of 1.3, 1.5, 2.2, 2.4, 2.6, 2.8, 3.1, 3.3, 3.5, 3.7, 3.9, 4.4, 5.5, and 6.8 [18]. Two wavelets are shown for each bior wavelet. The wavelet on the left is used for decomposition and the wavelet on the right is used for reconstruction.

compactly supported, their duration is finite and non-zero, which provides good localization of time and frequency [21]. These different wavelet shapes can be useful for distinctly localized signals, like spikes or bursts [22] and identifying the symmetry and orientation of the signal [23]. Wavelets that can provide better frequency localization (db and coif wavelets), linear phase characteristic (bior wavelet), and orientation (sym wavelet) could be beneficial for speech processing.

Types of wavelet transforms include the stationary wavelet transform (SWT) and wavelet packet transforms (WPT). The stationary wavelet transform solves the lack of time invariance by not taking down-samples from each decomposition step. However, this requires more memory space and computation time than the discrete wavelet transform. The wavelet packet transform applies decomposition through both a low pass filter that outputs approximation coefficients and a high pass filter that outputs detail coefficients. This can provide more information because of the additional detail coefficients but again requires more memory space and computation time on the order of 2^N where N is the decomposition level.

The results of wavelet transforms are coefficients that can be used as inputs to an ASR system by, for instance, calculating the normalized energy in each sub-band:

$$\text{Wavelet Sub-band Normalized Energy} = \sum_{t=t_1}^{t_2} \frac{x^2[n]}{t_2 - t_1}$$

where t_1 and t_2 are are sub-band start and end points. WT coefficients have been used in pitch detection and formant tracking [24], and phone classification [25], which shows potential for wavelet transform usefulness in the realm of speech processing. Equivalent rectangular bandwidth (ERB) scaling has also been implemented using wavelet packet transforms to test whether ERB scaling can be substituted for MFCC mel scaling in speech processing or not [26, 27]. None of these studies made direct numerical comparisons with MFCCs. Combining the wavelet transform with already existing feature extraction methods such as linear predictive coding [28] and MFCCs [29] showed improvements by 0.6% when using clean data and by 6% when using data with additive white Gaussian noise at different signal to noise ratio (SNR) levels. This suggests that recognition performance could improve when combining the MFCC and WT methods.

2.1.3 Time and Frequency Resolutions of the WT

Fourier transform based feature extraction methods use fixed-duration time windowing and therefore provide constant resolutions over the time-frequency axes unless additional frequency scaling processes are applied. If a signal has time and frequency content evenly spread out along the axes, this is reasonable. However in practice, speech signals have lower frequencies that are usually longer in time and higher frequencies that are usually shorter. This property of unevenly spaced time and frequency content is typical for most non-stationary signals [30]. The MFCC method has separate steps such as windowing and mel-scaling in order to compensate. The WT method, on the other hand, is designed to represent energies at high frequencies with good time resolution and give energies at low frequencies with good frequency resolution. Consider Figure 2.7, which shows the time-frequency resolutions for a short time Fourier transform and a discrete WT. It can be seen that at high frequency, the time window is narrower for the WT, which corresponds to good time resolution and at low frequency, the frequency window is narrower, which corre-

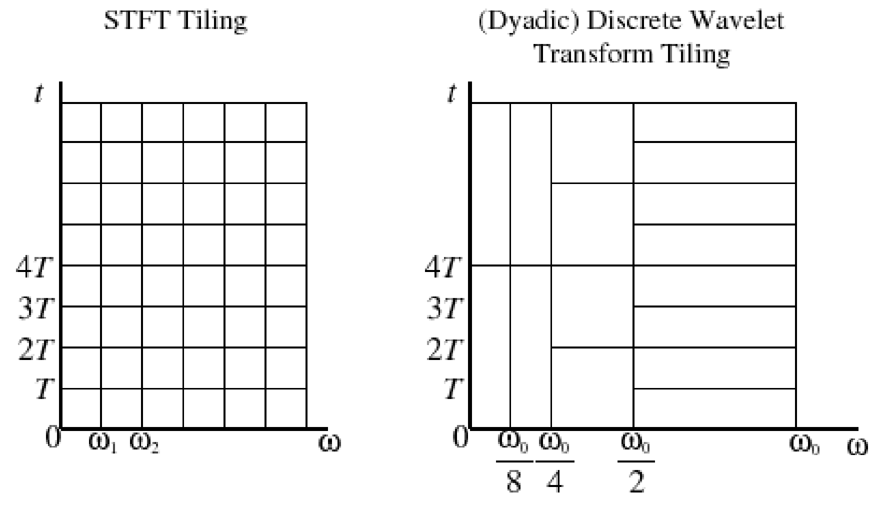


Figure 2.7: Resolution Grids of the Short Time Fourier Transform and the Wavelet Transform [31]

sponds to good frequency resolution. Wavelet transforms have the potential to better describe speech signals. However, MFCCs have been used for ASR for decades and systems have been finely tuned to work with them.

2.1.4 Wavelet Software Libraries

There are multiple wavelet libraries written in different languages such as Blitzwave C++ Wavelet Library [32], the University of Washington Wavelet Library [33], and The Imager Wavelet Library [34]. Some are available as binaries and some as open source tools. The “C++ Wavelet Libraries” package [35] was chosen, because the recognition system used is also written in C++ (see Section 2.5 for more information). This library provides discrete and stationary wavelet transforms with code for the four wavelet families of Daubechies, biorthogonol, Coiflet, and symlet. Other types of wavelet transforms and wavelet types can be added by users. The wavelet transform functions in libraries usually take five input arguments: an input signal vector, decomposition levels, wavelet type and its scale, lengths of respective approximation and detail vectors to the decomposition level, and an output signal vector. The output signal vector contains the wavelet transform coefficients. The coefficients from the output signal vector can be used as acoustic feature vectors in an ASR system.

2.2 Training an Acoustic Model

An acoustic model describes the likelihoods of the observed spectral feature vectors given linguistic units (words, phones, or sub-parts of phones). A commonly used statistical model for this is the hidden Markov model (HMM) [36]. Figure 2.8 shows a 3-state HMM including the special start and the end states, where a_{ij} is the transition probability matrix from state i to j, and b_i is a set of observation likelihoods, which are typically represented by mixtures of Gaussian models.

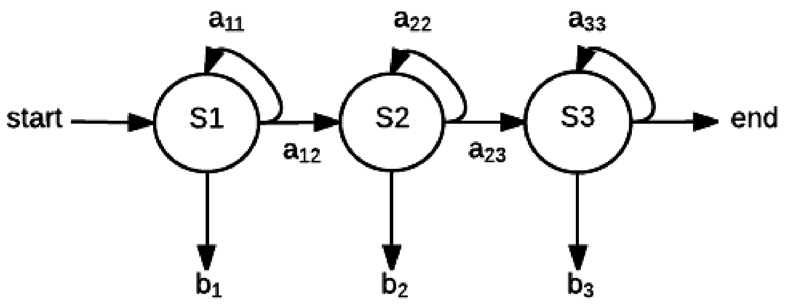


Figure 2.8: 3-State HMM

A word is made of phones, the smallest identifiable units of sound. For example the word “translate” is made of the following eight phones “T R AE N Z L EY T”. There are two ways these phones can be modeled, as monophone or context-independent models, and as triphone or context-dependent models. Monophone models ignore the surrounding context so there would be seven models representing **T**, **R**, **AE**, **N**, **Z**, **L**, and **EY**. Triphone models include the left and right context of the neighboring phones: **T**+R, T-**R**+AE, R-**AE**+N, AE-**N**+Z, N-**Z**+L, Z-**L**+EY, L-**EY**+T, and EY-**T**. Monophone models would have a single model for each phone, typically 41. In the case of triphones, there could be up to 41^3 individual models, but in practice, this is reduced because not all triphones appear in English and some may be acoustically similar to others. Training triphone models requires significantly more training data.

2.2.1 Gaussian Mixture Model

A Gaussian mixture model (GMM) is often used to represent the observation likelihoods ([37] p.425-p.435). The Guassian, or normal, distribution is a function with two parameters, mean and variance. The Gaussian function can be written as

$$f(x|\mu,\sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp(-\frac{(x-\mu)^2}{2\sigma^2}).$$

For a discrete random variable X, the mean, μ, and the variance, σ^2, can be computed using

$$\mu = E(X) = \sum_{i=1}^{N} p(X_i)X_i$$

$$\sigma^2 = E(X_i - E(X))^2 = \sum_{i=1}^{N} p(X_i)(X_i - E(X))^2$$

where $p(X_i)$ is the probability mass function. The mean is the weighted sum over the values of X and the variance is the weighted squared average deviation from the mean.

If the possible values of a dimension of an observation feature vector o_t can be assumed to be normally distributed, the univariate Gaussian probability density function can represent the output probability of an HMM state j determining the value of a single dimension of the feature vector. The observation likelihood function $b_j(o_t)$ can be then expressed with one dimension of the acoustic vector as a Gaussian and written as

$$b_j(o_t) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp(-\frac{(o_t - \mu_j)^2}{2\sigma_j^2}).$$

For each HMM state j, the Gaussian parameters of the mean and the variance can be computed by taking the average of the values for each o_t and taking the sum of the squared difference between each observation and the mean respectively.

Since multiple dimensions are typically used, for instance 13 dimensions for MFCCs (12 + 1 energy coefficient) and at least 9 dimensions for WT coefficients (8 + 1 energy coefficient), multivariate Gaussians are used to estimate the probabilities of the HMM states. The variance of each dimension and the covariance between any two dimensions can be described by the covariance matrix Σ where X and Y are two random variables

$$\Sigma = E((X - E(X))(Y - E(Y))) = \sum_{i=1}^{N} p(X_i Y_i)(X_i - E(X))(Y_i - E(Y)).$$

Then a mean vector with a dimensionality matching the input dimension and the covariance matrix can define a multivariate Gaussian

$$f(\vec{x}|\vec{\mu}, \Sigma) = \frac{1}{(2\pi)^{\frac{D}{2}} |\Sigma|^{\frac{1}{2}}} \exp(-\frac{1}{2}(x - \mu)^T \Sigma^{-1} (x - \mu)), \qquad (2.2.1)$$

where $\vec{x}$ and $\vec{\mu}$ are vector notations and D is the dimensionality of $\vec{\mu}$. When different dimensions of the feature vector do not co-vary, a variance that is distinct for each feature dimension is equivalent to a diagonal covariance matrix, where only the diagonal of the matrix has non-zero elements. A non-diagonal covariance matrix models the correlations between the feature values in multiple dimensions. A Gaussian with a full covariance is therefore a better model of acoustic likelihood. However, diagonal covariances are often used because the full covariance has high implementation complexity and requires more training data with more parameters. The estimation of the observation likelihood of a D-dimensional feature vector o_t given HMM state j using a diagonal covariance matrix can be written as

$$b_j(o_t) = \prod_{d=1}^{D} \frac{1}{\sqrt{2\pi\sigma_{jd}^2}} \exp(-\frac{1}{2}\frac{(o_{td}-\mu_{jd})^2}{\sigma_{jd}^2}),$$

which is simplified from Equation 2.2.1 keeping the mean and the variance for each dimension.

However, each dimension of feature vector may not be a normal distribution. Modeling the observation likelihood with a weighted mixture of multivariate Gaussians should be considered. Based on Equation 2.2.1, this modeling is called a Gaussian mixture model where

$$f(x|\mu,\Sigma) = \sum_{k=1}^{M} c_k f_k(x|\mu,\Sigma)$$

$$f(x|\mu,\Sigma) = \sum_{k=1}^{M} c_k \frac{1}{\sqrt{2\pi|\Sigma_k|}} \exp((x-\mu_k)^T\Sigma^{-1}(x-\mu_k))$$

and where M is the number of Gaussians summed together and c_k is the mixture weight for each Gaussian. The output observation likelihood is

$$b_j(o_t) = \sum_{k=1}^{M} c_{jk} \frac{1}{\sqrt{2\pi|\Sigma_{jk}|}} \exp((x-\mu_{jk})^T\Sigma_{jk}^{-1}(o_t-\mu_{jk})).$$

This likelihood is used for individual output observation in the acoustic model training.

2.2.2 Hidden Markov Model

A classical approach to modeling acoustic information is the hidden Markov model (HMM). The Markov assumption is used to describe the HMM, allowing a particular state to depend on a certain number of previous states instead of all past states. This reduces the number of hidden states of the HMM that are included in the model.

Obtaining the likelihood of a particular observation sequence $P(U|\lambda)$ given an HMM model $\lambda = (A, B)$ and an observation sequence U can be done by using the forward algorithm with the efficiency of $O(N^2T)$ where N is the number of hidden states and T is the number of observations from an observation sequence. The forward algorithm finds the probability of the observation sequence by using a table that saves intermediate values and summing over the probabilities of all possible hidden state paths that get implicitly folded into a single forward trellis [38]:

$$\alpha_t(j) = \sum_{i=0}^{N} \alpha_{t-1}(i) a_{ij} b_j(u_t)$$

where $\alpha_{t-1}(i)$ is the previous $(t-1)$ time step forward path probability, a_{ij} is the transition probability, and $b_j(u_t)$ is the state observation likelihood of the observation symbol o_t given the current state j. This is equivalent to obtaining the probability of being in state j after getting the first t observations given the HMM λ.

Finding the best hidden sequence Q given an observation sequence U and an HMM model $\lambda = (A, B)$ can be done by using the Viterbi algorithm with a dynamic programming trellis [39]:

$$v_t(j) = \max_{i=1}^{N} v_{t-1}(i) a_{ij} b_j(u_t)$$

where $v_{t-1}(i)$ is the previous Viterbi path probability, a_{ij} is the transition probability, and $b_j(u_t)$ is the state observation likelihood. This is equivalent to finding the most probable sequence of states $Q = q_1, q_2 ... q_T$ given an HMM state and a sequence of observations $U = u_1, u_2 ... u_n$.

Training the HMM parameters A and B given an observation sequence U and the set of states in the HMM, where A denotes the transition probability and B denotes the emission probability, can be done by using the Baum-Welch algorithm [40]. This algorithm makes uses of the backward probability

$$\beta_t(i) = P(u_{t+1}, u_{t+2}, ... u_T | q_t = i, \lambda)$$

where it views the partial observations from U_{t+1} to the end given that the HMM is in state i at time t and the HMM model λ. This algorithm is based on two intuitions. The first intuition is to improve the estimate by computing iteratively. The second intuition is to get estimated probabilities by computing the forward probability for an observation and dividing that probability mass among all the different paths that contributed to this forward probability.

In automatic speech recognition, the acoustic model refers to statistical representations for observed feature vector sequences. This is combined with the language model to perform word transcription.

2.3 Training a Language Model

A language model describes the probability of a word sequence. A large amount of transcribed or written sentences may be used to create statistical language models. The probability of a sentence or a sequence of words is

$$P(W) = P(w_1, w_2, w_3 ... w_n)$$

where $P(W)$ is the probability of an entire n word sequence and $P(w_i)$ is the probability of each word, w_i. The joint probability of a sentence or a sequence of words can be computed using the chain rule of probability. The chain rule of probability produces the product rule by rearranging the conditional probability equations

$$P(A, B) = P(A|B)P(B).$$

By applying the chain rule to a word sequence, extending the conditional probabilities on words, the probability of the entire sequence of words can be written as

$$P(w_1^n) = P(w_1)P(w_2|w_1)P(w_3|w_1^2)...P(w_n|w_1^{n-1}) = \prod_{k=1}^{n} P(w_k|w_1^{k-1}) \tag{2.3.1}$$

where $w_1^n = w_1...w_n$ is a sequence of words. Modeling such statistical word sequences results in language models.

In practice, storing the entire word sequence is almost impossible due to memory constraints. In order to solve this, reasonable shortened sequences, N-grams, are modeled [41]. N-grams allow the prediction of a next word from the previous $(N-1)$ words. Bi-grams $(N = 2)$, tri-grams $(N = 3)$, and 4-grams $(N = 4)$ are typically used in the word prediction. The Markov assumption allows depending on only a word's recent history rather than looking at the complete history. An N-gram model is a $(N-1)$ order Markov model. Applying the assumption, the conditional probability of the next word in a sequence can be written as

$$P(w_n|w_1^{n-1}) \approx P(w_n|w_{n-N+1}^{n-1}).$$

Substituting the bi-gram into Equation 2.3.1, the probability of a complete word sequence can be written as

$$P(w_1^n) \approx \prod_{k=1}^{n} P(w_k|w_{k-1}).$$

Estimating the N-gram probabilities can be done by maximum likelihood estimation (MLE) [42]. The generalized N-gram parameter estimation can be calculated using

$$P(w_n|w_{n-N+1}^{n-1}) = \frac{C(w_{n-N+1}^{n-1}, w_n)}{C(w_{n-N+1}^{n-1})},$$

where C is count of the word or word sequence. The numerator represents the count of the observed word given $N-1$ preceding words. The denominator represents the count of the $N-1$ word sequence. This is also called a relative frequency.

There are some special cases where this LM can be replaced. When all the possible word-to-word transitions are known, a finite state grammar can be used. For instance, small corpora such as digits are comprised of sequences of digits from zero to nine. In that case, since the input words are limited to that range, a finite state grammar can replace the LM.

The decoding process takes the acoustic and language models, a dictionary mapping words to phone strings, and an input speech waveform to produce a matching word sequence, which is discussed next.

2.4 Decoding

The goal of decoding is to obtain the string of words with the highest posterior probability given the training data represented by the acoustic and language model and contained by a dictionary representing the vocabulary. The highest value from the product of the two probabilities is used if Bayes rule from noisy-channel modeling can be assumed ([37] p.1). Such value can be obtained from

$$W = \arg\max_{W} P(O|W)P(W)$$

where $P(O|W)$ is the likelihood or the model of the noisy channel producing any random observation, $P(W)$ is the hidden prior term, and W stands for the entire string of words.

Since the acoustic model produces the likelihood of a particular acoustic observation or a frame given a particular state or a sub-phone, multiplying such probabilities together from each frame to get the probability of the whole word introduces a drastic range difference between the two probabilities $P(O|W)$ and $P(W)$. A language model scaling factor (LMSF) is used to better scale the probabilities. When the acoustic model has more influence than the language model, longer words are more likely to be recognized as a sequence of short words, even if that sequence is not likely according to the language model. In order to compensate for this, a word insertion penalty (WIP) is used. Increasing the penalty keeps the decoding processing from including additional words in the output, which can reduce the number of insertion errors. Since calculations are done in logarithms in order to avoid underflow errors in computing, the highest value from the two probabilities can be written as

$$W = \arg\max_{W} (\log P(O|W)P(W) + LMSF \log P(W) + N \log WIP),$$

where N is the number of words in the hypothesized word string.

2.5 Speech Recognition Virtual Kitchen

The Speech Recognition Virtual Kitchen (SRVK), available at www.speechkitchen.org, is a project that "aims to extend the model of lab-internal knowledge transfer to community-wide dissemination via immediate, straightforward access to the tools and techniques used by advanced researchers" [43]. In order to use most speech recognition systems, a lot of programming tasks and experiment setups are required. The SRVK solves this through the use of virtual machines with scripts, data, and pre-compiled code as shown in Figure 2.9.

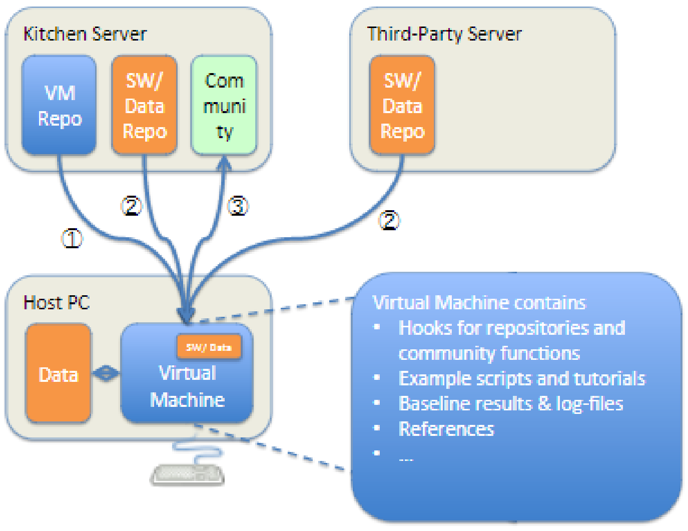


Figure 2.9: Architecture Diagram of the SRVK [44]

A virtual machine (VM) is a tool where users can load an operating system (OS) of their choice on top of their running OS. Open source resources such as Debian Linux derivatives for a base OS and pre-installed open source, C++ speech recognizer Kaldi [45] are included in an ASR VM available through the SRVK. The Kaldi recognizer includes features such as integration with finite state transducers, linear algebra support, open source software (http://kaldi.sourceforge.net), and databases from the Linguistic Data Consortium. Four layers of components shown in Figure 2.10 define the Kaldi recognizer structure. All layers can be modified to expand the recognition system. Other VMs may vary in data sets, test challenges, recogni-

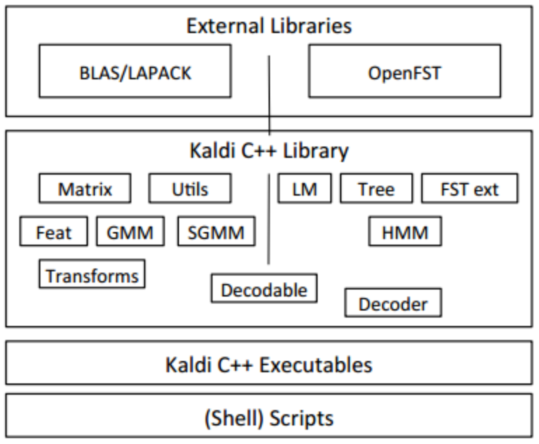


Figure 2.10: The Kaldi Recognizer Overview [45]

tion system, or task. For complex systems, VMs enable infrastructure for meeting numerous challenges in ASR research and education by providing repositories that facilitate exchanges among ASR community members [44].

The work of this thesis is based on SRVK VMs. The processes of learning an ASR system, implementing sub-processes of the system, and conducting experiments using the VM are all done using and modifying VMs from the SRVK. Because this is an open source project, the extensions developed in this work will be made available as new VMs through the SRVK.

2.6 Summary

A standard ASR system was introduced in this chapter. Since this work focuses on the signal processing components, two feature extraction methods, MFCCs and WTs, were presented, with MFCCs being the baseline standard and WTs providing a different approach to time-frequency resolution. Finally, the SRVK virtual machine framework used in experiments was discussed. The first research question is addressed by successfully implementing a wavelet module based on an open source library into an SRVK VM. Conducting different experiments with MFCCs and WTs will address the second and the third research questions.

Chapter 3

Data

ASR systems require a large amount of data for training and testing a system. The process of creating a data corpus includes recording and transcribing speech. Further processing can be done to cut waveforms into smaller segments, label data at the sub-word, or phone level, as well as sentence level, and add different types of noise to the data. Three datasets, or corpora, are used in this work: TI-Digits (small vocabulary) [46], TED-LIUM (large vocabulary) [47], and Noizeus (noisy data) [48].

3.1 TI-Digits

The TI-Digits corpus contains speech collected by Texas Instruments to design and evaluate algorithms for speaker-independent recognition of digit sequences [46]. Because it was collected in 1982, it has been used in many systems (e.g., [49, 50]). Digit sequences were read into a high quality, close-talking microphone. 326 speakers are included in the data (111 men, 114 women, 50 boys, and 51 girls). Each speaker pronounced 77 digit sequences, with a total of more than 25,000 digit sequences. There is 4 hours of data, sampled at 20kHz. The data has been partitioned into a test set of 12,551 words and a training set of 12,551 words. The vocabulary is small, comprised of the ten digits, with two pronunciations of 0, “oh” and “zero”. Example strings are

“7 3 8 7 oh” and “3 8 zero zero”. Because of the small vocabulary size, 21 phones are sufficient to describe the sounds used in TI-Digits.

3.2 TED-LIUM

The TED-LIUM corpus [47] contains speech collected from TED talks [51]. TED-LIUM speech is planned and delivered in a formal presentation to a large audience. 685 speakers are included in the 122 hours of speech (85 hours of male speech and 37 hours of female speech). The data has been partitioned into a test set of 28,000 words and a training set of 2,560,000 words. This data is sampled at 16kHz. The vocabulary is large, comprised of 156,700 number of words. The corpus includes labels for show names, channels, speaker IDs, start and end times, genre identifications, silences, fillers, and pronunciation variants. An example sentence is “What I’m going to tell you about in my eighteen minutes is how we’re about to switch from reading the genetic code to the first stages of beginning to write the code ourselves.” The TED-LIUM dictionary used in this work describes word pronunciations in terms of 41 unique phones.

3.3 Noizeus

The Noizeus corpus contains speech collected by the University of Texas at Dallas [48]. Six speakers (3 adult male and 3 adult female) each recorded 5 sentences. The data consists of a test set of 241 words. The vocabulary size is small, comprised of 167 words. Sentences in the Noizeus corpus are not very long and the speakers frequently hyper-articulate. Example sentences are “A good book informs of what we ought to know.” and “The lazy cow lay in the cool grass.” Eight different noisy conditions, airport, babble, car, exhibition, restaurant, street, and train noises, were added to each utterance at 0dB, 5dB, 10dB, and 15dB SNRs. Sentences were originally sampled at 25kHz but were down-sampled at 8kHz by the corpus creators. Because the Noizeus corpus does not have enough data to train either acoustic or language models, it is used only as a test set.

Chapter 4

Methodology

Three research questions were posed in Chapter 1:

1. Can a wavelet transform signal processing module be added to an existing ASR system?
2. Can WT features match the performance of MFCCs?
3. Are there benefits to combining the MFCC and WT methods?

This chapter shows how the questions are answered given the background from Chapter 2 and using the data described in Chapter 3. First, the ASR virtual machine context is presented. Then the WT features and the combined features from the two methods are shown. Finally, the performance metrics used to compare approaches are presented.

4.1 The ASR Context

In a typical ASR system, the test speech waveform and the training speech waveform go through signal processing that produces the MFCCs. In this work, the signal processing extends to include WTs and combinations of WTs and MFCCs as shown in Figure 4.1. In order to implement this, "C++ Wavelet Libraries" (version 0.4.0.0)

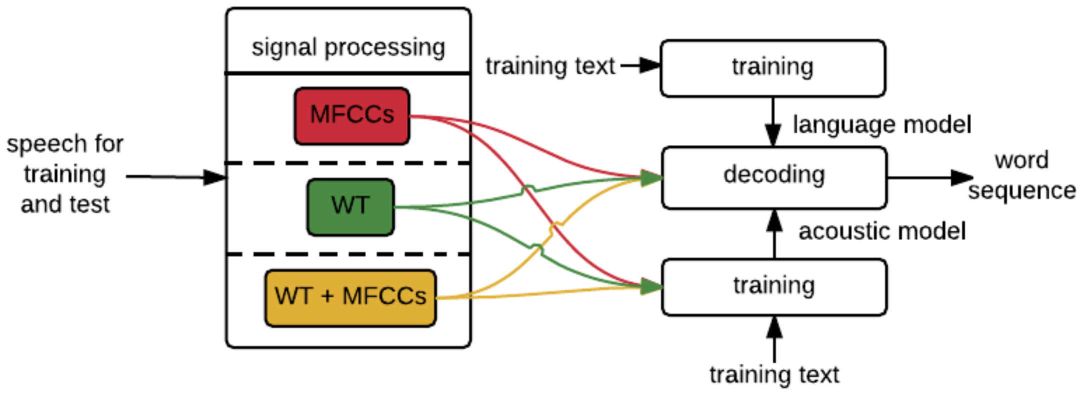


Figure 4.1: An ASR System with Multiple Options for Signal Processing

[35] was added to the Kaldi recognizer (version 1.0) [45] on an SRVK VM. Figure 4.2 is an updated version of Figure 2.10 showing where the C++ Wavelet Libraries and new functions were added to the system. All layers needed modification except for the external library component in experiments presented in this work. Since the Kaldi

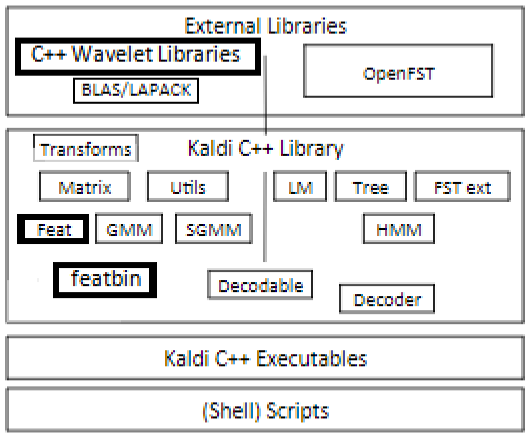


Figure 4.2: The Kaldi Recognizer Overview [45]

recognizer declares its own data types such as VectorBase and BaseFloat, these types have to be converted into the base C++ variable types when interfacing with the wavelet libraries in “featbin”. Options for feature extraction such as the number of features and time window duration were changed in header files from “feat” in order to match the baseline experiments of this work. In shell scripts, to run experiments, options such as whether the input will go through cepstral mean variance normalization or not were changed since MFCCs use this option by default but the WT features do not. These options can be changed in different header files. Appendix A.1 describes where these header files are located and how these options can be changed.

4.2 Wavelet Transform Features

Three different types of wavelet transforms were used to extract acoustic features and compare to MFCCs. For each WT, maximum energy values from each decomposition level are taken as features with varying time window durations. Varying time-frequency resolution is provided for by the varying window whereas the MFCC method typically uses a fixed 25ms window. There was a limitation with this approach, because the Kaldi recognizer checks intermediate values in ways that they are finely tuned for MFCCs. This made it difficult to adjust the process to use appropriate values for the wavelet features. For instance, the duration of the frame window can only go up to 50ms, otherwise the process was terminated for having bad values. This negates the benefit of varying resolutions, one of the advantages the WT has over MFCCs.

4.2.1 Discrete Wavelet Transform

The discrete wavelet transform (DWT) decomposes a signal using a set of low and high pass filters followed by down-sampling that outputs approximation and detail coefficients respectively. By definition, the decomposition takes place on only low pass filters. Figure 4.3 shows an example three level decomposition where g[n] is the output of the high pass filter and h[n] is the output of the low pass filter. If a 16kHz sampled signal with 10,000 sample points is assumed, the first detail coefficients have the frequency band range from 8kHz to 16kHz with 5,000 sample points, the second detail coefficients have the frequency band range from 4kHz to 8kHz with 2,500 sample points, the third detail coefficients have the frequency band range 2kHz to 4kHz with 1,250 sample points, and the third approximation coefficients have the frequency band range from 0Hz to 2kHz with 1,250 sample points. Changing the duration of the window has trade-offs: the longer the duration of window, the better the frequency resolutions and the shorter the duration of window, the better the time resolutions. The maximum energies from each decomposition level with a 50ms window and a 25ms frame rate are taken as input features to the ASR system using the DWT function. For details, see the DWT source code in Appendix B.1.

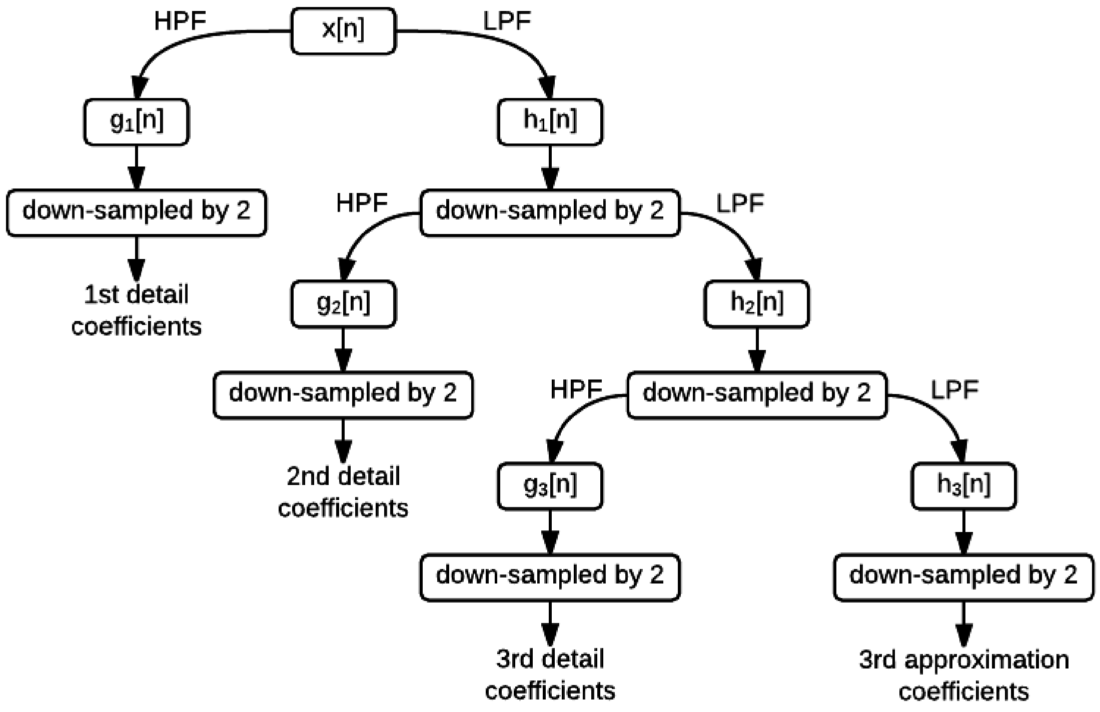


Figure 4.3: Three Level Discrete Wavelet Transform Decomposition

4.2.2 Stationary Wavelet Transform

The stationary wavelet transform (SWT) also decomposes a signal into a set of low and high pass filters but without down-sampling. Figure 4.4 shows a three level decomposition as an example. Omitting the down-sampling processes ensures that the output size from each decomposition is the same as the original signal. This solves the lack of time-invariance with respect to the DWT but brings redundancy. This trade-off needs to be considered when the stationary WT is used, as the output size from each decomposition level is increased. If a 16kHz sampled signal with

10,000 sample points is assumed, the frequency band range will be the same as the previous example, but the number of sample points is different as it keeps the same 10,000 sample points at all decomposition levels. The maximum energies from each decomposition level with a 50ms window and a 25ms frame rate are taken as input features to the ASR system using the SWT function. For details, see the SWT source code in Appendix B.2.

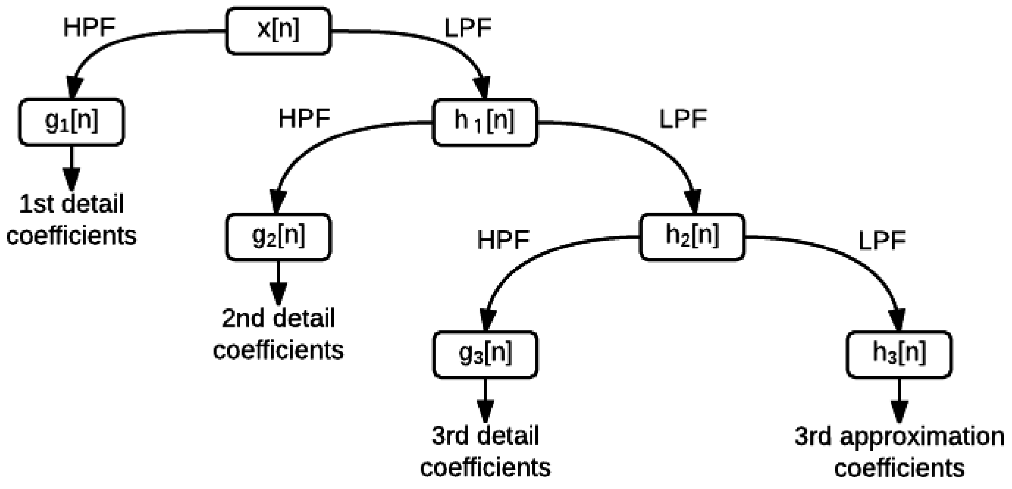


Figure 4.4: Three Level Stationary Wavelet Transform Decomposition

4.2.3 Wavelet Packet Transform

The wavelet packet transform (WPT) is very similar to the DWT but differs in that it applies decomposition at both the high pass filter and low pass filter. Figure 4.5 shows an example of two level decomposition. The WPT uses more memory space and computation time than DWT or SWT, but provides more information at all levels because the decomposition is also taken at the HPF. If a 16kHz sampled signal with

10,000 sample points is assumed, the second detail coefficients from the first detail coefficients have the frequency band range from 12kHz to 16kHz with 2,500 sample points, the second approximation coefficients from the first detail coefficients have the frequency band range from 8kHz to 12kHz with 2,500 sample points, the second detail coefficients from the first approximation coefficients have the frequency band range from 4kHz to 8kHz with 2,500 sample points, and the second approximation coefficients from the first approximation coefficients have the frequency band range from 0Hz to 4kHz with 2,500 sample points.

The WPT was not originally implemented in the selected wavelet library. Since the only different between the WPT and the DWT is whether to decompose detail coefficients from the HPF or not, the WPT was implemented by decomposing the coefficients from the HPF. Decomposing the coefficients from the HPF required additional temporal values, which explains why the WPT uses more memory space and computation time. Another option for WPT was to extend the SWT in a similar manner. Extending the SWT to implement the WPT required even more temporal values, so was not done. The maximum energies from each decomposition level with a 50ms window and a 25ms frame rate are taken as input features to the ASR system using the WPT function. Window energies are obtained in the same way for all WT approaches. For more detail of the WPT source code, see in Appendix B.3.

Implementing these three WTs provides a means to addressing the second research question, whether using WTs can provide comparable performance to MFCCs in an ASR system. The DWT provides varying time-frequency resolutions with the use of a pair of filters and a down-sampler. The SWT also provides varying time-frequency resolutions without time-invariance by omitting the down-sampling, but requires more

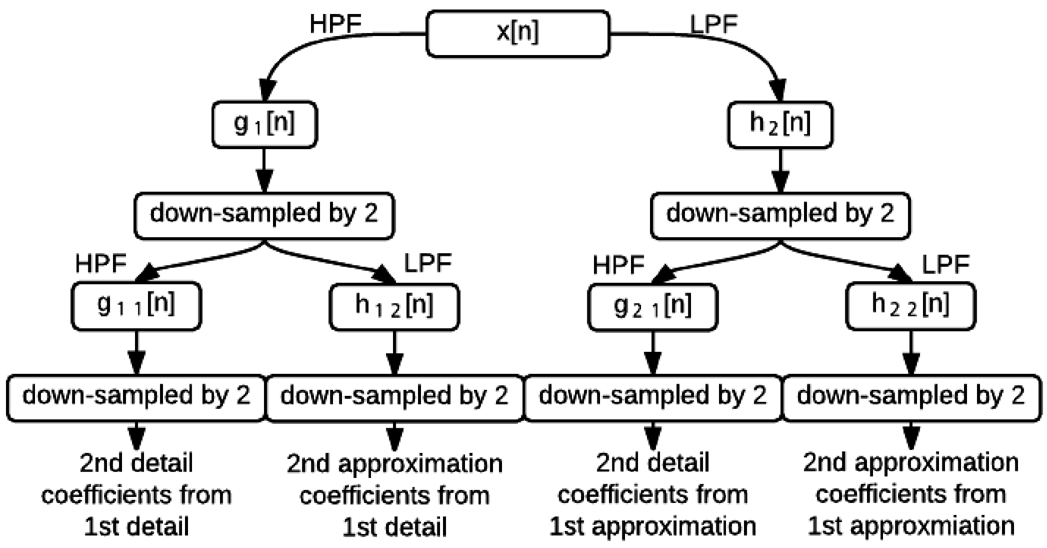

HPF
x[n]
LPF
g 1 [n]
h 2 [n]
down-sampled by 2
down-sampled by 2
HPF
LPF
HPF
LPF
g 1 1 [n]
h 1 2 [n]
g 2 1 [n]
h 2 2 [n]
down-sampled by 2
down-sampled by 2
down-sampled by 2
down-sampled by 2
2nd detail coefficients from 1st detail
2nd approximation coefficients from 1st detail
2nd detail coefficients from 1st approximation
2nd approximation coefficients from 1st approxmiation

Figure 4.5: Two Level Wavelet Packet Transform Decomposition

memory space and computation time than the DWT. The WPT provides varying time-frequency resolutions by decomposing the output of the HPF and providing more information in the detail coefficients, which also requires more memory space and computation time than the DWT.

Different wavelet families, Daubechies, biorthogonal, Coiflet, and symlet, were used to simulate qualities such as frequency localization, linear phase characteristic, and orientation of speech signals. Orthogonality, symmetry, vanishing moment conditions, and compact support of a wavelet determines which wavelet families better describe the speech signals. The best family shape for the ASR tasks examined here is chosen experimentally.

4.3 Combined Features from the Two Methods

Two ways of combining the two feature extraction methods are considered in order to address the third research question. The first is adding wavelet features to the MFCC feature vector. Since the wavelet feature vector contains different information about the speech signal, it can be appended to the MFCC vector. However, the same limitation discussed in the previous section should be considered: how the duration of the window is limited by the ASR system to 50ms. The second approach is using the WT as a pre-processing step to address noise before MFCC processing. For each experiment, different wavelet families and scales can be applied as desired. Since that produces so many possible combinations, three wavelets with different scales from different wavelet families were used to address noise before the MFCC processing.

4.3.1 Wavelet Sub-band Energy

The first way to combine features is to add wavelet features to the MFCC feature vectors as shown in Figure 4.6. Since the wavelet coefficients have different time-frequency resolutions than MFCC's mel-scaled resolutions, the wavelet coefficient energies contain different information about the speech signal. Wavelet shapes can also make a difference in the energy level from different time-frequency resolutions provided by wavelet shapes. Another option to this approach is to replace MFCC feature with WT features. The number of feature added or replaced can vary as desired. For details about appending and replacing WT features, see the source code in Appendix B.4.

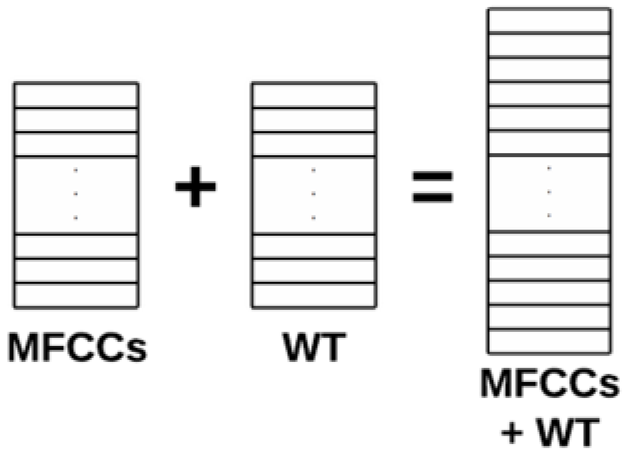


Figure 4.6: Combining Wavelet Features with the MFCC Features

4.3.2 Wavelet Denoising

The second way to combine features is to pre-process the acoustic signal using a wavelet transform then process the MFCCs as shown in Figure 4.7. The stationary WT is known to give good denoising performance [52, 53]. Different wavelet shapes are used to see what types give better denoising performance under different noise conditions. The speech signal is decomposed into a desired number of sub frequency

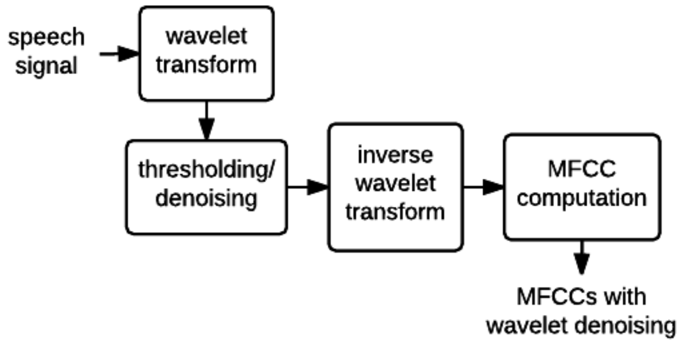


Figure 4.7: The Wavelet Denoising Process

bands by taking the stationary WT. The next step is to define an absolute value that limits or modifies the signal value. This absolute value is called the threshold value. Applying the threshold value to the noisy signal, d, provides an updated signal $\delta_\lambda(d)$ where

$$\delta_\lambda(d) = \begin{cases} 0 & \text{if } |d| \leq \lambda, \\ d - \lambda & \text{if } d > \lambda\,, \\ d + \lambda & \text{if } d < -\lambda\,, \end{cases}$$

and λ is the threshold value, typically the variance of a signal. For every value of d, the threshold value is compared and either forces the signal value to zero, subtracts the threshold value from the signal value, or adds the threshold value to the signal. Since threshold value parameters such as mean and variance were implemented in the source code, other thresholding methods, such as hard thresholding, could be used. Figure 4.8 shows hard and soft thresholding methods. For details about thresholding, see the source code in Appendix B.5. The last step is taking the inverse stationary

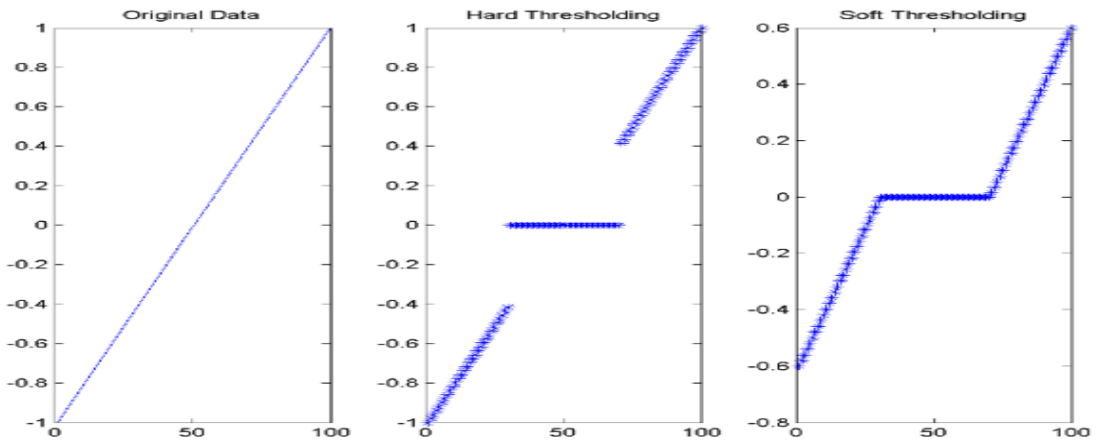


Figure 4.8: Hard Thresholding and Soft Thresholding [54]

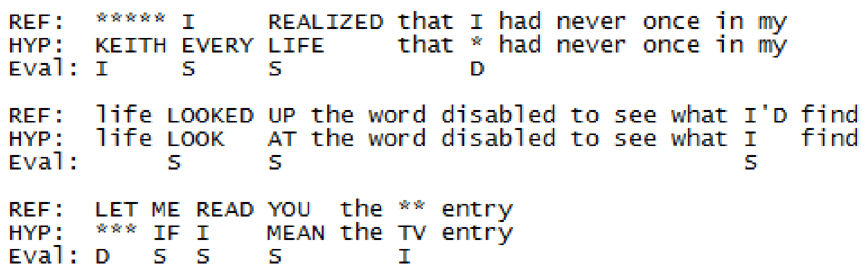
WT to the thresheld signals and recover the signal to compute MFCCs. To test the denoising performance, recognition experiments are done using the Noizeus corpus.

4.4 Performance Evaluation

Word error rate (WER) is commonly used to evaluate speech recognition systems. It is based on how much the hypothesized word string differs from the reference transcription. The WER is the number of words substituted, inserted, and deleted in the hypothesized string divided by the number of words in the reference transcription string. The reference and hypothesized strings are dynamically aligned so that substitutions, insertions, and deletions can be identified. An example alignment is shown in Figure 4.9. Then using the WER equation

$$\%\text{WER} = (\frac{S + D + I}{N}) \times 100$$

where S is the number of substituted words, D is the number of deleted words, I is the number of inserted words, and N is the number of words in the reference string, the example results in a WER of (8+2+2)/(26) $\times$100% = 46.1%. Other measures for



```
REF:  ***** I     REALIZED that I had never once in my
HYP:  KEITH EVERY LIFE     that * had never once in my
Eval: I     S     S             D

REF:  life LOOKED UP the word disabled to see what I'D find
HYP:  life LOOK   AT the word disabled to see what I   find
Eval:      S      S                                S

REF:  LET ME READ YOU  the ** entry
HYP:  *** IF I    MEAN the TV entry
Eval: D   S  S    S        I
```

Figure 4.9: An Example of Aligned Reference and Hypothesized Word Strings from TED-LIUM

performance evaluation are computation time and memory usage. A shell script was used to keep track of the execution time in seconds and the memory usage was seen using the Linux "top" command and kept in a separate text file.

4.5 Summary

In order to address the research questions, three different types of wavelet transforms were introduced: the discrete wavelet transform, the stationary wavelet transform, and the wavelet packet transform. The DWT and SWT provide varying time-frequency resolutions, although the SWT avoids the problem of time invariance. The WPT provides more information through additional detail coefficients. Two different combining methods were introduced: using wavelet sub-band energy in the MFCC feature vector and wavelet denoising as a pre-process. Results for these methods are discussed in Chapter 5.

Chapter 5

Results

The first research question has been addressed by creating the signal processing module, allowing for three different types of wavelet transforms, DWT, SWT, and WPT with four different wavelet families: Daubechies, biorthogonal, Coiflet, and symlet. This chapter presents results that answer the second and third research questions posed in Chapter 1. The baseline MFCC results with 13 features (+ deltas and delta deltas features) with the window size of 25ms and frame rate of 10ms are shown in Table 5.1 for the TI-Digits and TED-LIUM systems. The systems were trained using matching acoustic data.

Table 5.1: Baseline MFCC Results

	WER
TI-Digits	0.48%
TED-LIUM	21.4%

5.1 Wavelet Transform Features

Results with 15 wavelet transform features (+ delta and delta delta features) for the TI-Digits corpus and also 15 features (+ delta and delta delta features) for the TED-LIUM corpus from different WTs are shown in Table 5.2. Experiments with different window sizes and frame rates were run. Originally, a 20ms frame rate was used but smaller rates were tried to represent finer time windows. The window size and frame rate used for different WT features are also presented inside parentheses in Table 5.2. Although the 15ms frame rate worked better for DWT on TI-Digits, the smaller frame rate did not improve the SWT in TI-Digits, and showed only a slight improvement for WPT. Therefore, the shorter frame rate was only tried for the DWT TED-LIUM case.

Different wavelet families, Daubechies (db), biorthogonol (bior), Coiflet (coif), and symlet (sym), and scales provided in the wavelet software library were tested on the TI-Digits corpora. The wavelet packet transform features showed better recognition performance than discrete and stationary WTs because WPT decomposes the information in detail coefficients, which increases the information about high frequency components of the speech signal. All provided wavelet types from the library were used for WPT on the TI-Digit system. The biorthogonal wavelet with a scale of 2.2 gave the best recognition performance in preliminary results so it was used for the other two WTs on TI-Digits and for all WTs on TED-LIUM. MFCC results (Table 5.1) show better recognition performance for both TI-Digits and TED-LIUM systems (Table 5.2).

Computation time and memory usage of the WTs and MFCCs with 45 features,

Table 5.2: Recognition WER Results Using 2.2 Scale Biorthogonal Wavelet Transforms with 15 Features for TI-Digits and TED-LIUM. Two numbers in parentheses are the window size and frame rate respectively. All experiments used delta and delta delta features.

	TI-Digits	TI-Digits	TED-LIUM	TED-LIUM
DWT	6.76% (50ms/15ms)	7.61% (50ms/20ms)	82.7% (50ms/15ms)	84.5% (50ms/20ms)
SWT	10.96% (50ms/13ms)	13.2% (50ms/20ms)	NA	80.4% (50ms/20ms)
WPT	3.13% (50ms/16ms)	3.5% (50ms/20ms)	NA	51.3% (50ms/20ms)

Table 5.3: Computation Time and Memory Usage of the Wavelet Transforms

	Computation Time	Memory Usage
MFCCs	577 seconds	10,000 MB
DWT	3212 seconds	16,400 MB
SWT	9084 seconds	16,800 MB
WPT	6705 seconds	17,200 MB

including deltas and delta-deltas for the TI-Digits corpus are shown in Table 5.3. The window size and frame rate used for MFCCs are 25ms and 10ms respectively. For WTs, the window size and frame rate used are 50ms and 20ms respectively. Having smaller frame rates gave longer computation time in general. The results show that the WTs take more time and memory to compute the same number of features. However since the libraries do not take the advantage of faster wavelet algorithms [55], the wavelet feature computation time result could improve.

While the theoretical contribution of wavelets to automatic speech recognition could better address the time-frequency variation in speech, the limitations of simply adding a wavelet transform module to a speech recognition system tuned for MFCCs mean that using wavelet transforms on their own do not improve recognition performance. Therefore, at this point, the second research question is answered negatively in this work.

5.2 Combined MFCC and WT Features

Results from the two approaches for combining feature extraction methods are presented in the following sections. For the first method, both the TI-Digits and TED-LIUM systems were used but the results from the TED-LIUM test data are not presented because the recognition performance was the same as the baseline in Table 5.1. For the second method, the Noizeus data was used as the test set in an ASR system trained with TED-LIUM data.

5.2.1 Adding Wavelet Sub-band Energy to MFCC Feature Vectors

Simply adding from one to four additional wavelet sub-band energy features to the MFCC vector did not show improvement in recognition performance. Adding additional features to the MFCC vector degraded recognition performance. Replacing the frame energy in the MFCC vector with the wavelet sub-band energy showed slight improvements, depending on different wavelet shapes and scales. Results with 13 MFCC features (+ delta and delta delta features) for TI-Digits data are shown in

Tables 5.4, 5.5, 5.6, and 5.6 with Daubechies, biorthogonol, symlet, and Coiflet sub-band energies with varying scales, each replacing the MFCC frame energy. Features were calculated on 25ms windows and had a 10ms frame rate. Not all results are better than the baseline results (Table 5.1), so only results with better recognition performance are shown. At most improvements of 10.4% were shown in Table 5.4 and Table 5.5, however these changes were not significant given the size of the test set.

Table 5.4: Combining Daubechies Sub-band Energy Results with MFCCs

Wavelet	MFCCs	db1	db3	db6	db8	db9	db14
WER	0.48%	0.47%	0.45%	0.47%	0.43%	0.43%	0.47%

Table 5.5: Combining Biorthogonol Sub-band Energy Results with MFCCs

Wavelet	MFCCs	bior1.1	bior2.4	bior3.3
WER	0.48%	0.47%	0.45%	0.47%

Table 5.6: Combining Symlet and Coiflet Sub-band Energy Results with MFCCs

Wavelet	MFCCs	sym3	sym6	sym8	sym9	coif2
WER	0.48%	0.45%	0.47%	0.46%	0.47%	0.46%

5.2.2 Wavelet Denoising

The stationary WT is used to perform the denoising pre-processing. Soft thresholding was used in this work. Recognition results using 13 MFCC features (+ delta and delta delta features) for the Noizeus corpus with the ASR system trained with TED-LIUM data are shown in Table 5.7. Baseline results for Noizeus data under different noise levels and conditions using 13 MFCC features (+ delta and delta delta features) are shown in Table 5.8. The Noizeus data with 15dB added train noise was not available. As expected, results degrade with higher levels of noise.

Table 5.7: Baseline Clean Noizeus Result using MFCCs

	WER
Noizeus	67.22%

Table 5.8: Baseline Noisy Noizeus Results using MFCCs

	Airport	Babble	Car	Exhibition	Restaurant	Station	Street	Train
15dB	80.91%	81.74%	85.06%	85.06%	80.08%	80.08%	81.74%	NA
10dB	87.97%	86.31%	90.87%	91.29%	88.80%	86.31%	88.80%	88.80%
5dB	92.53%	95.44%	92.12%	92.12%	89.63%	93.36%	90.04%	94.19%
0dB	94.19%	93.78%	93.78%	94.61%	91.70%	93.78%	91.70%	93.78%

The Daubechies (db) wavelet family is first used to denoise the signal with scales of 4 in Table 5.9, 8 in Table 5.10, and 12 in Table 5.11. The db wavelet with scale 8 on average gave the best result for the Daubechies family. On average, there was a 1.98% improvement over the baseline.

Table 5.9: Results for Wavelet Denoising with Scale 4 Daubechies Wavelet. The bold numbers indicate better recognition performance than the baseline

	Airport	Babble	Car	Exhibition	Restaurant	Station	Street	Train
15dB	82.16%	**79.67%**	**81.74%**	**83.82%**	80.50%	**79.25%**	**80.08%**	NA
10dB	**83.82%**	**84.23%**	**87.14%**	**86.72%**	**87.14%**	**85.89%**	**88.38%**	89.63%
5dB	**92.12%**	**92.95%**	**90.87%**	92.12%	**89.21%**	**92.53%**	**89.21%**	93.78%
0dB	94.61%	95.44%	93.78%	**92.12%**	92.53%	**93.36%**	94.19%	94.61%

Table 5.10: Results for Wavelet Denoising with Scale 8 Daubechies Wavelet. The bold numbers indicate better recognition performance than the baseline

	Airport	Babble	Car	Exhibition	Restaurant	Station	Street	Train
15dB	81.33%	**77.59%**	**81.74%**	**83.82%**	**79.25%**	80.50%	**80.50%**	NA
10dB	**84.23%**	**82.57%**	**86.72%**	**86.72%**	**85.06%**	**84.23%**	87.55%	90.04%
5dB	**90.87%**	**92.95%**	92.95%	**91.70%**	**87.97%**	**90.04%**	**86.72%**	92.95%
0dB	94.61%	**92.02%**	**92.95%**	**93.78%**	**83.36%**	**92.12%**	94.61%	94.19%

Table 5.11: Results for Wavelet Denoising with Scale 12 Daubechies Wavelet. The bold numbers indicate better recognition performance than the baseline

	Airport	Babble	Car	Exhibition	Restaurant	Station	Street	Train
15dB	**79.67%**	**80.91%**	**80.08%**	85.89%	81.74%	81.74%	**79.67%**	NA
10dB	**84.65%**	**85.06%**	**87.97%**	**85.48%**	**87.55%**	**84.65%**	**88.38%**	88.80%
5dB	92.53%	**92.12%**	**90.46%**	**90.87%**	**89.21%**	**92.12%**	**89.63%**	**90.04%**
0dB	94.61%	94.19%	**92.95%**	95.02%	93.78%	**92.12%**	95.02%	94.19%

The biorthogonal (bior) wavelet family is used to denoise the signal with scales of 2.2 in Table 5.12, 3.7 in Table 5.13, and 5.5 in Table 5.14. The bior wavelet with the scale of 3.7 on average gave the best result. On average, there was a 1.88% improvement over the baseline.

Table 5.12: Results for Wavelet Denoising with Scale 2.2 biorthogonal Wavelet. The bold numbers indicate better recognition performance than the baseline

	Airport	Babble	Car	Exhibition	Restaurant	Station	Street	Train
15dB	83.82%	**81.33%**	85.06%	85.06%	80.91%	81.74%	82.16%	NA
10dB	**84.23%**	**84.65%**	**90.04%**	**89.63%**	**87.55%**	87.14%	**88.38%**	90.46%
5dB	**91.70%**	**94.19%**	92.53%	92.12%	**88.80%**	93.78%	**89.21%**	**92.95%**
0dB	95.02%	94.61%	95.02%	94.61%	92.12%	**92.95%**	95.02%	94.19%

Table 5.13: Results for Wavelet Denoising with Scale 3.7 biorthogonal Wavelet. The bold numbers indicate better recognition performance than the baseline

	Airport	Babble	Car	Exhibition	Restaurant	Station	Street	Train
15dB	**80.50%**	**77.18%**	**79.67%**	**83.82%**	**76.76%**	**78.42%**	**80.50%**	NA
10dB	**83.82%**	**82.99%**	**87.97%**	**87.14%**	**86.76%**	**83.82%**	**87.14%**	89.63%
5dB	**92.12%**	**91.29%**	**90.87%**	92.53%	**85.89%**	**91.70%**	**86.72%**	**92.12%**
0dB	94.19%	95.02%	**92.53%**	94.61%	92.95%	**91.29%**	93.78%	94.61%

Table 5.14: Results for Wavelet Denoising with Scale 5.5 biorthogonal Wavelet. The bold numbers indicate better recognition performance than the baseline

	Airport	Babble	Car	Exhibition	Restaurant	Station	Street	Train
15dB	83.40%	**77.59%**	**81.33%**	85.06%	**78.42%**	**80.50%**	81.74%	NA
10dB	**84.65%**	**82.57%**	**87.55%**	**87.55%**	**85.06%**	**85.06%**	**87.97%**	90.04%
5dB	92.53%	**94.19%**	**91.29%**	92.12%	93.36%	93.36%	**89.63%**	**92.53%**
0dB	**93.78%**	94.61%	**92.53%**	**93.78%**	92.53%	**92.53%**	94.19%	93.78%

The Coiflet (coif) wavelet family is used to denoise the signal with scales of 1 in Table 5.15, 3 in Table 5.16, and 5 in Table 5.17. The coif wavelet with the scale of 3 on average gave the best result. On average, there was a 1.23% improvement over the baseline.

Table 5.15: Results for Wavelet Denoising with Scale 1 Coiflet Wavelet. The bold numbers indicate better recognition performance than the baseline

	Airport	Babble	Car	Exhibition	Restaurant	Station	Street	Train
15dB	84.23%	**80.91%**	**83.40%**	85.48%	82.57%	82.57%	82.16%	NA
10dB	**84.23%**	**85.89%**	**88.80%**	89.63%	88.38%	88.38%	**88.38%**	90.46%
5dB	92.95%	**94.19%**	92.53%	92.95%	93.36%	93.36%	90.04%	**92.53%**
0dB	95.02%	94.19%	94.61%	**94.19%**	92.12%	**92.12%**	95.44%	94.19%

Table 5.16: Results for Wavelet Denoising with Scale 3 Coiflet Wavelet. The bold numbers indicate better recognition performance than the baseline

	Airport	Babble	Car	Exhibition	Restaurant	Station	Street	Train
15dB	81.74%	**78.84%**	**81.74%**	**82.57%**	80.08%	**78.84%**	**80.50%**	NA
10dB	**84.65%**	**83.52%**	**87.97%**	**86.31%**	**86.31%**	**84.65%**	**87.55%**	90.04%
5dB	93.36%	**94.19%**	**91.70%**	**91.70%**	**87.97%**	**92.95%**	**87.14%**	**92.12%**
0dB	95.02%	94.61%	**92.95%**	**93.78%**	92.95%	**92.12%**	94.61%	93.78%

Table 5.17: Results for Wavelet Denoising with Scale 5 Coiflet Wavelet. The bold numbers indicate better recognition performance than the baseline

	Airport	Babble	Car	Exhibition	Restaurant	Station	Street	Train
15dB	**80.50%**	**79.67%**	**81.33%**	**84.65%**	80.91%	81.74%	**80.91%**	NA
10dB	**85.06%**	**84.65%**	**87.97%**	**86.31%**	**88.38%**	**84.23%**	**87.55%**	90.87%
5dB	**92.12%**	**91.29%**	**90.04%**	**90.87%**	90.49%	**92.53%**	**87.55%**	**90.87%**
0dB	**93.78%**	93.78%	93.78%	**94.19%**	94.61%	**92.53%**	95.02%	93.78%

The symlet (sym) wavelet family is used to denoise the signal with scales of 3 in Table 5.18, 6 in Table 5.19, and 9 in Table 5.20. The sym wavelet with the scale of 9 on average gave the best result. On average, there was a 1.42% improvement over the baseline.

Table 5.18: Results for Wavelet Denoising with Scale 3 symlet Wavelet. The bold numbers indicate better recognition performance than the baseline

	Airport	Babble	Car	Exhibition	Restaurant	Station	Street	Train
15dB	82.57%	**79.67%**	**81.74%**	85.48%	80.50%	80.50%	**78.42%**	NA
10dB	**83.82%**	**85.06%**	**87.97%**	**87.14%**	86.31%	87.97%	**87.97%**	90.04%
5dB	92.53%	**92.95%**	92.12%	92.95%	89.63%	**91.70%**	**89.21%**	**92.95%**
0dB	**93.78%**	95.02%	94.19%	94.19%	92.53%	**92.95%**	94.19%	93.78%

Table 5.19: Results for Wavelet Denoising with Scale 6 symlet Wavelet. The bold numbers indicate better recognition performance than the baseline

	Airport	Babble	Car	Exhibition	Restaurant	Station	Street	Train
15dB	81.74%	**79.25%**	**81.33%**	**82.57%**	80.08%	**78.84%**	**80.91%**	NA
10dB	**85.06%**	**83.40%**	**87.55%**	**86.31%**	**86.72%**	**84.23%**	**87.97%**	90.04%
5dB	92.95%	**93.36%**	**91.29%**	92.12%	**87.14%**	93.78%	**87.97%**	**91.70%**
0dB	95.02%	94.61%	**92.95%**	**93.78%**	93.36%	**92.53%**	94.61%	**92.12%**

Table 5.20: Results for Wavelet Denoising with Scale 9 symlet Wavelet. The bold numbers indicate better recognition performance than the baseline

	Airport	Babble	Car	Exhibition	Restaurant	Station	Street	Train
15dB	**80.50%**	**78.84%**	81.33%	85.06%	80.50%	**78.84%**	**79.25%**	NA
10dB	**82.99%**	**83.40%**	**87.55%**	**87.97%**	**86.31%**	**81.74%**	88.80%	90.04%
5dB	93.36%	**93.78%**	**89.63%**	**90.46%**	90.04%	**91.70%**	**87.97%**	**91.29%**
0dB	94.61%	95.02%	**92.95%**	94.61%	92.12%	**92.12%**	94.61%	93.78%

Table 5.21 shows the best possible WER improvement under different noise levels and environments by the provided wavelets of different shapes and scales. Each number in the table represents the percentage improvement followed by the wavelet used.

Table 5.21: WER Improvement by Wavelets in Different Noisy Environments

-	Airport	Babble	Car	Exhibition
15dB	1.24%(db12)	4.56%(bior3.7)	5.39%(bior3.7)	2.49%(coif3)
10dB	4.98%(sym9)	3.74%(bior5.5)	4.15%(db8)	3.74%(db12)
5dB	1.66%(db8)	4.15%(coif5)	2.49%(sym9)	1.66%(sym9)
0dB	0.41%(bior5.5)	1.76%(db8)	1.25%(bior3.7)	2.49%(db4)
-	Restaurant	Station	Street	Train
15dB	3.32%(bior3.7)	1.66%(bior3.7)	3.32%(sym3)	-
10dB	3.74%(bior5.5)	4.57%(sym9)	8.30%(bior3.7)	-
5dB	3.74%(bior3.7)	3.32%(db8)	2.9%(bior3.7)	4.15%(db12)
0dB	-	2.49%(bior3.7)	-	1.66%(sym6)

Average WER values from the different wavelets are shown in Table 5.22. The best denoising performance was given by bior3.7 wavelet and the worst was given by coif1 wavelet degrading the performance.

Table 5.22: Average WER for all Noisy Experiments

Type	WER	% Improvement
MFCC	89.37%	-
db4	88.50%	0.97%
db8	87.60%	1.98%
db12	88.42%	1.06%
bior2.2	89.38%	0.00%
bior3.7	87.69%	1.88%
bior5.5	88.55%	0.92%
coif1	89.65%	-0.31%
coif3	88.27%	1.23%
coif5	88.45%	1.03%
sym3	88.70%	0.75%
sym6	88.23%	1.27%
sym9	88.10%	1.42%

While the theoretical contribution of wavelet denoising has the potential for better denoising performance, the third research question is answered inconclusively in this work since the improvement is not significant.

5.3 Summary

The first research question is answered positively by creating the wavelet transform module that supports three different types of wavelet transforms: DWT, SWT, and WPT with four different wavelet types. The second research question is answered negatively using the three different types of wavelet transforms since MFCC performacne was not matched. The third research question is answered inconclusively, because although improvements were seen with pre-processing, the denoising results were not significantly different.

The maximum energy feature is one of the simplest ways to extract features from wavelet coefficients. This might have not fully taken advantage of useful wavelet properties. The Noizeus corpus used for wavelet denoising was very small and mismatched training data was used. Using other noisy corpora with matched recording and language condition would constrain these variables and allow for better exploration of denoising to deal with added noise.

Chapter 6

Conclusion

This thesis has contributed a wavelet transform module implemented on a speech recognition virtual machine and explored two different feature extraction methods, made comparisons, and looked for benefits of combining the two methods. The three research questions posed in Chapter 1 are reviewed and discussed, followed by suggestions for future work.

6.1 Summary

- Can a wavelet transform signal processing module be added to an existing ASR system?

The answer is yes. An open source wavelet library was used to implement a new module with static compiling on two or more VMs and showed successful functionality. Since the wavelet library had to be installed on top of the Kaldi recognizer, simply adding the static library to sub-programs in the recognizer rather than setting up all the necessary dynamic library paths was preferred for simpler implementation.

- Can WT features match the performance of MFCCs?

The answer is no. Using the approaches discussed in Chapter 4, it was shown that the WT features did not give compatible recognition performance compared to the

MFCCs shown in Chapter 5. Although there were constraints and limitations of the experiment setup that may have caused the non-matching performance results, no further tuning for better recognition performance was made in this work.

- Are there benefits to combining the MFCC and WT methods?

The answer is inconclusive. Two different ways to combine the two feature extraction methods introduced in Chapter 4 showed slight improvement in recognition performance presented in Chapter 5. The first method showed a slight improvement when different wavelet sub-band energy was used in the MFCC vector and the second method also showed slight improvements when wavelet denoising was applied before processing MFCCs on a noisy data set.

6.2 Future Work

Implementing the wavelet library using dynamic compiling that only keeps functions to be used should be considered in order to more positively answer the first research question by reducing memory usage. Further tuning the wavelet features and modifying the Kaldi recognizer experiment setup should be considered in order to positively answer the second research question. One approach would be to modify the Kaldi recognizer to loosen frame rate conditions and allow more dynamic frame rate ranges. Implementing different thresholding methods and exploring different wavelets under all decomposition levels for denoising could better address the third research question.

Bibliography

[1] Apple Inc, "iOS-Siri-Apple," http://www.apple.com/ios/siri, 2016. [Online]. Available: http://www.apple.com/ios/siri/. [Accessed: 24- Apr- 2016].

[2] P. Vilar, *Designing the User Interface: Strategies for Effective Human-computer Interaction, 5th Edition*. Pearson Addison-Wesley, 2009.

[3] A. Dix, J. Finlay, G. Abowd and R. Beale, *Human-Computer Interaction, 3rd Edition*. Prentice Hall, 2003.

[4] D. Jurafsky and J. Martin, *Speech and Language Processing, 2nd Edition.* Prentice Hall, 2009.

[5] P. Mermelstein, "Distance Measures for Speech Recognition, Psychological and Instrumental," in *Proc. Joint Workshop on Pattern Recognition and Artificial Intelligence*, Hyannis, MA, 1976, pp. 374388.

[6] A. Mertins, *Signal Analysis: Wavelets, Filter Banks, Time-Frequency Transforms and Applications.* Wiley, 1999.

[7] S. Mallat, *A Wavelet Tour of Signal Processing, 3rd Edition*, Academic Press, 2009.

[8] P. Kechichia, “On the Partial Haar Dual Adaptive Filter for Sparse Echo Cancellation,” M.E thesis, McGill University, 2006.

[9] J. Hillenbrand and R. Houde, “Acoustic Correlates of Breathy Vocal Quality,” *Journal of Speech Language and Hearing Research*, vol. 39, no. 2, pp. 311, 1996.

[10] S. Bharali and S. Kalita, “A Comparative Study of Different Features for Isolated Spoken Word Recognition using HMM with Reference to Assamese Language,” *International Journal of Speech Technology*, vol. 18, no. 4, pp. 673-684, 2015.

[11] G. Kaiser, *A Friendly Guide to Wavelets*, Boston: Birkhauser, 1995.

[12] C. Valens, “A Really Friendly Guide To Wavelets,” Polyvalens.pagesperso-orange.fr, 2016. [Online]. Available: http://polyvalens.pagesperso-orange.fr/clemens/wavelets/wavelets.html. [Accessed: 25- Jan- 2016].

[13] P. Addison, *The Illustrated Wavelet Transform Handbook*. Bristol, UK: Institute of Physics Pub., 2002, pp. 38-39.

[14] S. Atluri, G. Yagawa and T. Cruse, *Computational Mechanics’95*. Berlin: Springer, 1995, pp. 672-677.

[15] S. Mallat, “A Theory for Multiresolution Signal Decomposition: the Wavelet Representation,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 11, no. 7, pp. 674-693, 1989.

[16] N. Selvarasu, A. Nachiappan and N. Nandhitha, “Statistical Description of Wavelet Coefficients of Thermographs for Quantitative Characterization of Breast

Cancer," *Journal of Computer Science and Engineering Research and Development*, vol. 1, no. 1, pp. 1-10, 2011.

[17] W. Sweldens, "The Lifting Scheme: A Construction of Second Generation Wavelets," *SIAM Journal on Mathematical Analysis*, vol. 29, no. 2, pp. 511-546, 1998.

[18] The MathWorks Inc, "Wavelet Toolbox - MATLAB," Mathworks.com, 2016. [Online]. Available: http://www.mathworks.com/products/wavelet/. [Accessed: 25- Jan- 2016].

[19] R. Gao and R. Yan, *Wavelets*. Springer, 2011, pp. 61-64.

[20] V. Vaishnav, R. Kateeyare, J.S Chouhan and B. Dehriya, "Compression of Natural Images using Wavelet Transform," *International Journal of Engineering Sciences and Management*, vol. 5, no. 2, 2015.

[21] C. Heil, P. Jorgensen and D. Larson, *Wavelets, Frames and Operator Theory*. Providence: American Mathematical Society, 2004, p. 169.

[22] N. Smith, M. Gils and P. Prior, *Neurophysiological Monitoring during Intensive Care and Surgery*. Edinburgh: Elsevier, 2006, pp. 341.

[23] Z. Yuan, F. Li, P. Zhang and B. Chen, "Description of Shape Characteristics through Fourier and Wavelet Analysis," *Chinese Journal of Aeronautics*, vol. 27, no. 1, pp. 160-168, 2014.

[24] O. Farooq and S. Datta, “Speech Recognition with Emphasis on Wavelet based Feature Extraction,” *The Institution of Electronics and Telecommunication Engineers Journal of Research*, vol. 48, no. 1, pp. 3-13, 2002.

[25] O. Farooq, “Phoneme Recognition using Wavelet based Features,” *Information Sciences*, vol. 150, no. 1-2, pp. 5-15, 2003.

[26] A. Biswas, P. Sahu, A. Bhowmick and M. Chandra, “Feature Extraction Technique using ERB like Wavelet Sub-band Periodic and Aperiodic Decomposition for TIMIT Phoneme Recognition,” *International Journal of Speech Technology*, vol. 17, no. 4, pp. 389-399, 2014.

[27] A. Biswas, P. Sahu, A. Bhowmick and M. Chandra, “Articulation based Admissible Wavelet Packet Feature based on Human Cochlear Frequency Response for TIMIT Speech Recognition,” *Ain Shams Engineering Journal*, vol. 5, no. 4, pp. 1189-1198, 2014.

[28] N. Nehe and R. Holambe, “DWT and LPC based Feature Extraction Methods for Isolated Word Recognition,” *The European Association for Signal Processing Journal on Audio, Speech, and Music Processing*, vol. 2012, no. 1, pp. 7, 2012.

[29] M. Abdalla and H. Ali, “Wavelet-Based Mel-Frequency Cepstral Coefficients for Speaker Identification using Hidden Markov Models,” *Journal of Telecommunications*, vol. 1, no. 2, pp. 16-21, 2010.

[30] M. Lo, P. Tsai, P. Lin, C. Lin and Y. Hsin, “The Nonlinear and Nonstationary Property in EEG Signals: Probing the Complex Fluctuations by Hilbert-Huang Transform,” *Advances in Adaptive Data Analysis*, vol. 01, no. 03, pp. 465, 2009.

[31] Stack Exchange Inc, “Which Time Frequency Coefficients Does the Wavelet Transform Compute?” Dsp.stackexchange.com, 2016. [Online]. Available: http://dsp.stackexchange.com/questions/651/which-time-frequency-coefficients-does-the-wavelet-transform-compute. [Accessed: 25- Apr- 2016].

[32] O. Schulz, “Blitzwave C++ Wavelet Library,” Oschulz.github.io, 2016. [Online]. Available: http://oschulz.github.io/blitzwave/. [Accessed: 25- Apr- 2016].

[33] S. Neph, M. Kuehn and J. Stamatoyonnapoulos, “Wavelets:library,” Staff.washington.edu, 2016. [Online]. Available: http://staff.washington.edu/dbp/WMTSA/NEPH/wavelets.html. [Accessed: 25- Apr- 2016].

[34] B. Lewis, “Wavelets at Imager,” Cs.ubc.ca, 2016. [Online]. Available: http://www.cs.ubc.ca/nest/imager/contributions/bobl/wvlt/top.html. [Accessed: 25- Apr- 2016].

[35] R. Hussain, “C++ Wavelet Libraries,” Wavelet2d.sourceforge.net, 2016. [Online]. Available: http://wavelet2d.sourceforge.net/. [Accessed: 25- Jan- 2016].

[36] L. Rabiner, “A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition,” *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257-286, 1989.

[37] V. Matousek, *Text, Speech, and Dialogue*. Berlin: Springer, 2001.

[38] Y. Lifshits, S. Mozes, O. Weimann and M. Ziv-Ukelson, “Speeding up HMM Decoding and Training by Exploiting Sequence Repetitions,” *Algorithmica*, vol. 54, no. 3, pp. 379-399, 2007.

[39] H. Hickersberger and W. Zagler, "A Voice Command System for Autonomy using a Novel Speech Alignment Algorithm," *International Journal of Speech Technology*, vol. 16, no. 4, pp. 461-469, 2013.

[40] J. Sousa and R. Rossi, *Computer-based Modelling and Optimization in Transportation.* Berlin: Springer, 2012. pp. 361-363.

[41] R. Solomonoff, "An Inductive Inference Machine," *IRE Convention Record, Section on Information Theory*, Vol. 2. 1957.

[42] K. Ponting *Computational Models of Speech Pattern Processing.* Berlin: Springer, 1999, pp. 259-279.

[43] F.Metze, E. Fosler-Lussier, B. Bates, E. Riebling and A. Plummer, "Speech Kitchen," Speechkitchen.org, 2016. [Online]. Available: http://speechkitchen.org/. [Accessed: 27- Apr- 2016].

[44] F. Metze, E. Fosler-Lussier and R. Bates, "The Speech Recognition Virtual Kitchen: Launch Party," in *Proc. Interspeech.* Singapore, ISCA, 2014.

[45] D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlcek, Y. Qian, P. Schwarz, J. Silovsky, G. Stemmer, and K. Vesel y, "The Kaldi Speech Recognition toolkit," in *Proc. Automatic Speech Recognition and Understanding Workshop*, Big Island, HI; USA: IEEE, Dec. 2011.

[46] R. Gary Leonard, and George Doddington. TIDIGITS LDC93S10. https://catalog.ldc.upenn.edu/LDC93S10. Philadelphia: Linguistic Data Consortium, 1993.

[47] A. Rousseau, P. Deleglise and Y. Esteve, "TED-LIUM: an Automatic Speech Recognition Dedicated Corpus," *Proc Language Resources and Evaluation Conference.* Istanbul, May. 2012.

[48] Y. Hu and P. Loizou, "Subjective Comparison and Evaluation of Speech Enhancement Algorithms," *Speech Communication*, vol. 49, no. 7-8, pp. 588-601, 2007.

[49] D. Yu, L. Deng, X. He and A. Acero, "Large-Margin Minimum Classification Error Training: A Theoretical Risk Minimization Perspective," *Speech Communication*, vol. 49, no. 7-8, pp. 588-601, 2007.

[50] A. Gorin and D. Jouvet, "Component Structuring and Trajectory Modeling for Speech Recognition" in *Proc. Interspeech.* Singapore, ISCA, May. 2014.

[51] TED, "TED Talks — TED.com," Ted.com, 2016. [Online]. Available: https://www.ted.com/talks. [Accessed: 22- Apr- 2016].

[52] J. Fowler, "The Redundant Discrete Wavelet Transform and Additive Noise," *IEEE Signal Processing Letters*, vol. 12, no. 9, pp. 629-632, 2005.

[53] R. Naga, S. Chandralingam, T. Anjaneyulu and K. Satyanarayana, "Denoising EOG Signal using Stationary Wavelet Transform," *Measurement Science Review*, vol. 12, no. 2, 2012.

[54] The MathWorks Inc, "Wavelet Denoising and Nonparametric Function Estimation - MATLAB & Simulink," Mathworks.com, 2016. [Online].

Available: http://www.mathworks.com/help/wavelet/ug/wavelet-denoising-and-nonparametric-function-estimation.html#f8-22146. [Accessed: 27- Apr- 2016].

[55] G. Beylkin, R. Coifman and V. Rokhlin, "Fast Wavelet Transforms and Numerical Algorithms I," *Communications on Pure and Applied Mathematics*, vol. 44, no. 2, pp. 141-183, 1991.

Appendix A

Experiment Preparation

This Appendix shows how the primarily accessed directories are structured and where the key files are located. It also presents installation and use instruction for "C++ Wavelet Libraries" along with compiling commands for use in the VM.

A.1 Directory Block

Figure A.1 shows the directory block of the primary locations on the virtual machine used in this work. There are three directories frequently accessed in this work. Directory 1 contains the wavelet library related files: the source code, the object file from the source code, and the library file that corresponds to the object file. Directory 2 contains header files used by code in Directory 3. The window types, the frame length, the frame shift rate, the number of features and other things can be configured in this directory. Directory 3 contains the source code and its executable files. The wavelet header file is placed in directory 3 as well because static compiling was done in this work. Directory 4 contains the main script that uses the programs in directory 3 and other programs in the Kaldi recognizer. The data used is placed in the data sub-directory and the results are stored in the exp sub-directory in directory 4.

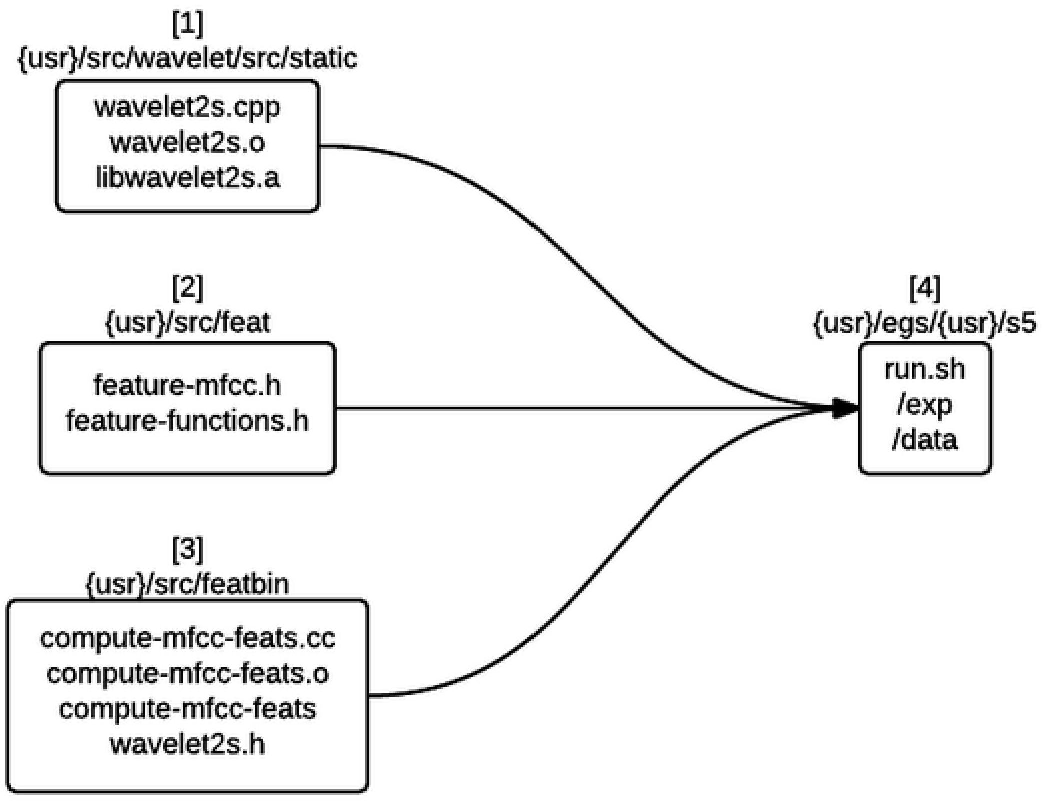


Figure A.1: Directory Block

A.2 Installing the Wavelet Library

The "C++ Wavelet Libraries" is external to the Kaldi recognizer, therefore it needs to be installed where it can be used. Note the wavelet library is placed in Directory 1 in Figure A.1. The wavelet library should be downloaded and installed in that directory using this link: http://sourceforge.net/projects/wavelet2d/files/ wavelib-0.4.0.0/wavelib.tar.gz. This library relies on another library called fftw which can be downloaded link: ftp://ftp.fftw.org/pub/fftw/fftw-3.3.4.tar.gz. The fftw library should be installed in the same directory before installing the wavelet library.

A.3 Compiling Commands

The file wavelet2s.cpp contains all wavelet functions. When modifications are made in this file, the following compiling commands are used to update the wavelet library. A Kaldi executable such as “compute-mfcc-feats” in directory 3 then can be recompiled using the updated wavelet library.

```
g++ -static -c wavelet2s.cpp

ar rcs libwavelet2s.a wavelet2s.o
```

Appendix B

Source Code

In this appendix, code for implementing three different wavelet transform features (discrete, stationary, and wavelet packet transform), wavelet sub-band energy, and wavelet denoising is provided. This is implemented into the Kaldi recognizer, which has source code available at http://kaldi.sourceforge.net/. The code provided here was written for the thesis work, but uses library variables and functions such as dwt_sym() and swt(). Kaldi functions are also used in this code such as NumFrames(), ExtractWindow(), and CopyFromVec().

The wavelet library functions take six input arguments for the discrete wavelet transform (DWT) and five input arguments for the stationary wavelet transform (SWT). Since the wavelet packet transform (WPT) was not implemented in the library, the WPT function was extended from the DWT. The six input arguments for the DWT are input signal vector "sig", decomposition level "J", wavelet name "nm", vector that stores lengths of respective approximation and detail in each decomposition step "length", output vector "dwt_output", and a housekeeping vector that contains an indicator of whether the signal is even or odd and a current decomposition level "flag". Since every length in all decomposition steps is the same as the original signal for SWTs, the "flag" vector is not needed. Therefore five input arguments are needed for the SWT. The output vector "dwt_output" is mainly used to extract features. The size of the output vector depends on the decomposition level "J" because the coefficients from all decomposition levels are stored in the output vector, through "J" level coefficients, "J-1" level coefficients, etc. The library provides the following wavelets: Daubechies (with scales of 1, 2, 3, 4, 5, 6, 8, 9, 10, 11, 12, 13, 14, and 15), biorthogonol (with scales of 1.1, 1.3, 1.5, 2.2, 2.4, 2.6, 2.8, 3.1, 3.3, 3.5, 3.7, 3.9, 4.4, 5.5, and 6.8), Coiflet (with scales of 1, 2, 3, 4, and 5), and symlet (with scales of 1, 2, 3, 4, 5, 6, 7, 8, and 9). More scales are theoretically possible but they were not implemented in the library, which can be considered a limitation of this library.

B.1 Discrete Wavelet Transform Features

```
Matrix<BaseFloat> wavelet_features;
Vector<BaseFloat> feature_vector, avg_sig, level_J, window;
vector<double> windowed_wave, dwt_output, flag;
vector<int> length;
string nm = "db8";
int J = 6, max_det = 0, max_appx = 0;
int32 rows_out = NumFrames(waveform.Dim(), mfcc.opts_.frame_opts);
int32 cols_out = mfcc.opts_.num_ceps;
features.Resize(rows_out, cols_out); // pre-declare feature size
for (int32 r = 0; r < rows_out; r++) {
// this for loop goes through one utterance
//by iterating windows with some frame rate
```

```
BaseFloat log_energy;
ExtractWindow(waveform, r, mfcc.opts_.frame_opts,
mfcc.feature_window_function_, &window,
(mfcc.opts_.use_energy && mfcc.opts_.raw_energy ?
&log_energy : NULL)); // pull out a window
int windowed_sig_energy = 0;
for(int i = 0; i < window.Dim(); i++){ // energy calculation
    windowed_wave.push_back(window(i));
    windowed_sig_energy += (window(i)*window(i));
}
dwt_sym(windowed_wave, J, nm, dwt_output, flag, length);
// wavelet transform applied to the windowed signal
windowed_wave.clear();
feature_vector.Resize(9);
feature_vector(8) = windowed_sig_energy/window.Dim();
windowed_sig_energy = 0;
int previous_index = 0, current_index = 0;
for(int temp = 0; temp < 4; temp++){
//getting maximum energy features in each sub-band
    previous_index = current_index;
    current_index += length.at(3 - temp);
    for(int i = previous_index; i < current_index; i++){
        if(max_det < dwt_output[i])
            max_det = dwt_output[i];
            if(max_appx < dwt_output[i + current_index])
                max_appx = dwt_output[i + current_index];
        }
            feature_vector(temp * 2) = max_det;
            feature_vector((temp * 2) + 1) = max_appx;
            max_det = 0;
            max_appx = 0;
    }
    features.Row(r).CopyFromVec(feature_vector);
    dwt_output.clear();
    length.clear();
    flag.clear();
    avg_sig.Resize(0);
}
```

B.2 Stationary Wavelet Transform Features

```
Matrix<BaseFloat> wavelet_features;
Vector<BaseFloat> feature_vector, avg_sig, level_J, window;
vector<double> windowed_wave, swt_output, flag; // ss
int length;
string nm = "db8";
int J = 6, max_det = 0, max_appx = 0;
int32 rows_out = NumFrames(waveform.Dim(), mfcc.opts_.frame_opts);
```

```
int32 cols_out = mfcc.opts_.num_ceps;
features.Resize(rows_out, cols_out); // pre-declare feature size
for (int32 r = 0; r < rows_out; r++) {  // r is frame index..
// this for loop goes through one utterance
//by iterating windows with some frame rate
    BaseFloat log_energy;
    ExtractWindow(waveform, r, mfcc.opts_.frame_opts,
    mfcc.feature_window_function_, &window,
    (mfcc.opts_.use_energy && mfcc.opts_.raw_energy ?
    &log_energy : NULL)); // pull out a window
    int windowed_sig_energy = 0;
    for(int i = 0; i < window.Dim(); i++){ // energy calculation
        windowed_wave.push_back(window(i));
        windowed_sig_energy += (window(i)*window(i));
    }
    swt(windowed_wave, J, nm, swt_output, length);
    // wavelet transform applied to the windowed signal
    windowed_wave.clear();
    feature_vector.Resize(9);
    feature_vector(8) = windowed_sig_energy/window.Dim();
    windowed_sig_energy = 0;
    int previous_index = 0, current_index = 0;
    for(int temp = 0; temp < 4; temp++){
    //getting maximum energy features in each sub-band
```

```
        previous_index = current_index;

        current_index += length;

        for(int i = previous_index; i < current_index; i++){

            if(max_det < swt_output[i])

                max_det = swt_output[i];

            if(max_appx < swt_output[i + current_index])

                max_appx = swt_output[i + current_index];

        }

        feature_vector(temp * 2) = max_det;

        feature_vector((temp * 2) + 1) = max_appx;

        max_det = 0;

        max_appx = 0;

    }

    features.Row(r).CopyFromVec(feature_vector);

    swt_output.clear();

    length = 0;

    avg_sig.Resize(0);

}
```

B.3 Wavelet Packet Transform Features

```
Matrix<BaseFloat> wavelet_features; // ss

Vector<BaseFloat> feature_vector, avg_sig, level_J, window;

vector<double> windowed_wave, swt_output, flag; // ss
```

```
int length;
string nm = "db8";
int J = 3, max_det = 0, max_appx = 0;
int32 rows_out = NumFrames(waveform.Dim(), mfcc.opts_.frame_opts);
int32 cols_out = mfcc.opts_.num_ceps;
features.Resize(rows_out, cols_out); // pre-declare feature size
for (int32 r = 0; r < rows_out; r++) {  // r is frame index..
// this for loop goes through one utterance
//by iterating windows with some frame rate
    BaseFloat log_energy;
    ExtractWindow(waveform, r, mfcc.opts_.frame_opts,
    mfcc.feature_window_function_, &window,
    mfcc.opts_.use_energy && mfcc.opts_.raw_energy ?
    &log_energy : NULL)); // pull out a window
    int windowed_sig_energy = 0;
    for(int i = 0; i < window.Dim(); i++){ // energy calculation
        windowed_wave.push_back(window(i));
        windowed_sig_energy += (window(i)*window(i));
    }
    swt(windowed_wave, J, nm, swt_output, length);
// wavelet transform applied to the windowed signal
    windowed_wave.clear();
    feature_vector.Resize(15);
    feature_vector(14) = windowed_sig_energy/window.Dim();
    windowed_sig_energy = 0;

    int previous_index = 0, current_index = 0;

    for(int temp = 0; temp < 7; temp++){

    //getting maximum energy features in each sub-band

        previous_index = current_index;

        current_index += length;

        for(int i = previous_index; i < current_index; i++){

            if(max_det < swt_output[i])

                max_det = swt_output[i];

            if(max_appx < swt_output[i + current_index])

                max_appx = swt_output[i + current_index];

        }

        feature_vector(temp * 2) = max_det;

        feature_vector((temp * 2) + 1) = max_appx;

        max_det = 0;

        max_appx = 0;

    }

    features.Row(r).CopyFromVec(feature_vector);

    swt_output.clear();

    length = 0;

    avg_sig.Resize(0);

}
```

B.4 Wavelet Sub-band Energy

```
SubVector<BaseFloat> waveform(wave_data.Data(), this_chan);
Matrix<BaseFloat> features;
try {
// calculate MFCC features first
    mfcc.Compute(waveform, vtln_warp_local, &features, NULL);
} catch (...) {
    KALDI_WARN << "Failed to compute features for utterance " << utt;
    continue;
}
size_t dim = features.NumCols();
MatrixIndexT index = 0;
Vector<BaseFloat> features_col;
features_col.Resize(dim);
Vector<BaseFloat> feature_vector, avg_sig, level_J, window;
vector<double> windowed_wave, dwt_output, swt_output, flag; // ss
vector<int> length;
string nm = "sym4";
int J = 3;
float max_det = 0, max_appx = 0;
int32 rows_out = NumFrames(waveform.Dim(), mfcc.opts_.frame_opts);
int32 cols_out = mfcc.opts_.num_ceps;
for (int32 r = 0; r < rows_out; r++) { // r is frame index..
```

```
BaseFloat log_energy;
ExtractWindow(waveform, r, mfcc.opts_.frame_opts,
mfcc.feature_window_function_, &window,
(mfcc.opts_.use_energy && mfcc.opts_.raw_energy ?
&log_energy : NULL)); // pull out a window
dwt_sym(windowed_wave, J, nm, dwt_output, flag, length);
// wavelet transform applied to the windowed signal
windowed_wave.clear();
features_col.CopyFromVec(features.Row(r));
int previous_index = 0, current_index = 0;
for(int temp = 0; temp < 1; temp++){
// getting wavelet sub-band energy
    previous_index = current_index;
    current_index +=length.at(0 - temp);
    for(int i = previous_index; i < current_index; i++){
        max_det += dwt_output[i];
        max_appx += dwt_output[i + current_index];
    }
    features_col(12 + (temp * 2))
    = (max_det + max_appx)/length.at(0 - temp);
    max_det = 0;
    max_appx = 0;
}
features.Row(r).CopyFromVec(features_col);
```

```
    // replacing the MFCC frame energy with
    // the wavelet sub-band energy
    dwt_output.clear();
    length.clear();
    flag.clear();
    avg_sig.Resize(0);
}
```

B.5 Wavelet Denoising

```
vector<double> wf_waveform, swt_output, iswt_output;
int J = 3, length = 0;
string nm = "bior2.2";
for(int i = 0; i < waveform.Dim(); i++){
// converting variable tpyes between Kaldi and
// default C++ variables
    wf_waveform.push_back(waveform(i));
}
swt(wf_waveform, J, nm, swt_output, length);
// apply SWT to the signal
double total = 0, mean = 0;
for(int i = 0; i < length; i++){
// total value
    total +=swt_output.at(i);
```

```
}
mean = total/length; // mean
double standard_deviation = 0;
total = 0;
for(int i = 0; i < length; i++){
// variance
    total += (swt_output.at(i) - mean) * (swt_output.at(i) - mean);
}
standard_deviation = sqrt(total/(length));
for(int i = 0; i < length; i++){
// soft thresholding
    if(swt_output.at(i) <= standard_deviation){
        swt_output.at(i) = 0;
    }else if(swt_output.at(i) > standard_deviation){
        swt_output.at(i) -= standard_deviation;
    }else if(swt_output.at(i) < -standard_deviation){
        swt_output.at(i) += standard_deviation;
    }
}
iswt(swt_output, J, nm, iswt_output);
// signal reconstruction
Vector<BaseFloat> denoised_waveform(length);
for(int i = 0; i < length; i++){
// converting variable tpyes between Kaldi and
// default C++ variables

    denoised_waveform(i) =iswt_output.at(i);

}
```

B.6 Compiling Commands

All provided wavelet transform code can be compiled using the following commands. There are two sets of compiling commands separated by dash lines. The first compiling command is used on the TI-Digits VM, and the other one is used on the TED-LIUM VM. The outputs are C++ executable sub-programs that running scripts can call and work with.

```
g++ -msse -msse2 -Wall -I.. -pthread -DKALDI_DOUBLEPRECISION=0

-DHAVE_POSIX_MEMALIGN -Wno-sign-compare -Wno-unused-local-typedefs

-Winit-self -DHAVE_EXECINFO_H=1 -rdynamic -DHAVE_CXXABI_H -DHAVE_ATLAS

-I/home/mario/src/kaldi/kaldi-trunk/tools/ATLAS/include

-I/home/mario/src/kaldi/kaldi-trunk/tools/openfst/include

-I/home/mario/src/kaldi/kaldi-turnk/src/wavelib/src/static

-Wno-sign-compare -g -DKALDI_NO_EXPF -rdynamic -Wl,

-rpath=/home/mario/src/kaldi/kaldi-trunk/tools/openfst/lib

compute-mfcc-feats.cc ../feat/kaldi-feat.a

../wavelib/src/static/libwavelet2s.a

../transform/kaldi-transform.a ../gmm/kaldi-gmm.a

../thread/kaldi-thread.a ../tree/kaldi-tree.a

../matrix/kaldi-matrix.a ../util/kaldi-util.a
```

```
../base/kaldi-base.a
-L/home/mario/src/kaldi/kaldi-trunk/tools/openfst/lib
-lfst -L/home/mario/src/kaldi/kaldi-trunk/src/wavelib/src/static
-lwavelet2s -lfftw3 /usr/lib/libatlas.so.3 /usr/lib/libf77blas.so.3
/usr/lib/libcblas.so.3 /usr/lib/liblapack_atlas.so.3
-lm -lpthread -ldl -o compute-mfcc-feats
/////////////////////////////////////////////////////////////////
g++ -msse -msse2 -Wall -I.. -pthread -DKALDI_DOUBLEPRECISION=0
-DHAVE_POSIX_MEMALIGN -Wno-sign-compare -Wno-unused-local-typedefs
-Winit-self -DHAVE_EXECINFO_H=1 -rdynamic -DHAVE_CXXABI_H -DHAVE_ATLAS
-I/kaldi-trunk/tools/ATLAS/include -I/kaldi-trunk/tools/openfst/include
-I/home/mario/src/kaldi/kaldi-turnk/src/wavelib/src/static
-Wno-sign-compare -g -DKALDI_NO_EXPF
-c -o compute-mfcc-feats.o compute-mfcc-feats.cc
g++ -rdynamic -Wl,-rpath=/kaldi-trunk/tools/openfst/lib
compute-mfcc-feats.o ../feat/kaldi-feat.a
../wavelib/src/static/libwavelet2s.a
../transform/kaldi-transform.a ../gmm/kaldi-gmm.a
../thread/kaldi-thread.a ../tree/kaldi-tree.a
../matrix/kaldi-matrix.a ../util/kaldi-util.a
../base/kaldi-base.a   -L/kaldi-trunk/tools/openfst/lib
-lfst -L/kaldi-trunk/src/wavelib/src/static -lwavelet2s -lfftw3
-L/usr/lib -llapack -lcblas -latlas -lf77blas -lm
-lpthread -ldl -o compute-mfcc-feats
```