Neural Voice Activity Detection and its Practical Use

by

Matthew McEachern

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2018

Author . . . .
Department of Electrical Engineering and Computer Science
June 8, 2018

Certified by . . . .
James Glass
Senior Research Scientist
Thesis Supervisor

Certified by . . . .
Hao Tang
Postdoctoral Associate
Thesis Supervisor

Accepted by . . . .
Katrina LaCurts
Undergraduate Officer

Neural Voice Activity Detection and its Practical Use

by

Matthew McEachern

Submitted to the Department of Electrical Engineering and Computer Science
on June 8, 2018, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

The task of producing a Voice Activity Detector (VAD) that is robust in the presence of non-stationary background noise has been an active area of research for several decades. Historically, many of the proposed VAD models have been highly heuristic in nature. More recently, however, statistical models, including Deep Neural Networks (DNNs) have been explored. In this thesis, I explore the use of a lightweight, deep, recurrent neural architecture for VAD. I also explore a variant that is fully end-to-end, learning features directly from raw waveform data. In obtaining data for these models, I introduce a data augmentation methodology that allows for the artificial generation of large amounts of noisy speech data from a clean speech source. I describe how these neural models, once trained, can be deployed in a live environment with a real-time audio stream. I find that while these models perform well in their closed-domain testing environment, the live deployment scenario presents challenges related to generalizability.

Thesis Supervisor: James Glass
Title: Senior Research Scientist

Thesis Supervisor: Hao Tang
Title: Postdoctoral Associate

Acknowledgments

I would like to thank my parents, Nancy and Lee, for bringing out the best in me, and for their love and support throughout my entire journey.

I would also like to thank my best friends and roommates, Karan Kashyap and Ken Leidal. In addition to our camaraderie, Karan and Ken have shown me how to be a better version of myself, and have taught me a countless number of valuable lessons and skills.

I would like to thank fellow lab members Dave Harwath and Hao Tang. Their wealth of knowledge and willingness to spend time helping others learn has been invaluable to the lab.

Lastly but not least, I would like to thank my supervisor, Jim Glass, for his guidance over the last two years. It was always a thrill to share my latest results and milestones with Jim during weekly meetings. He always took the time to brainstorm ideas for both areas of research and methodologies for analysis. He's been incredibly patient and supportive throughout.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

A raw audio signal may contain a variety of audible sounds including human speech, ambient noise (i.e. music, wind, rain), silence, and other non-speech sounds. For Automatic Speech Recognition (ASR) and other speech-processing applications, the underlying models are only meant to process audio that contains human speech. To otherwise process non-speech audio can be costly from a computational or network bandwidth perspective. In addition, the outputs from these models for non-speech audio may not be interpretable or accurate. For example, a particular ASR system may be engineered to make its "best guess" when transcribing raw audio. For noisy environments like a kitchen, this ASR system may transcribe a string of arbitrary noises as a valid English sentence.

To mitigate the problem of processing non-speech audio, it's important to avoid inputting audio that does not contain speech and vice versa. The problem of detecting human speech in a raw audio signal is known as Voice Activity Detection (VAD). VAD models are widely used as a precursor to downstream speech-processing applications. These VAD models may serve as a gating mechanism, effectively filtering out audio segments that do not contain speech. Alternatively, they may provide segmentation information to the downstream application which may be used to drive certain processes.

For many of the new speech-driven interfaces like Apple HomePod, Google Home, or Amazon Echo, upstream filtering of raw audio is crucial. The powerful ASR systems that drive these applications may be too large and too complex to run on one of these computationally-constrained devices. Thus, these ASR systems may need to be run on an external server(s) with high-performance capabilities. To send 100% of the audio recorded from one of these devices would create massive load on the external servers and the network. In addition, this would represent a significant privacy concern. The solution is to run VAD and other audio-filtering techniques on the device, allowing for the exclusive transmission of audio that is intended to be processed.

1.1 Background

VAD can be considered a binary classification problem on time-series data. Finite-length subsegments of audio, or *frames*, are either classified as containing speech or non-speech. Typically, VAD models will accept some feature representation of the raw audio as input. A feature representation, also known as a *feature vector*, is an alternative representation of raw data that accentuates discriminatory information. A mathematical representation of a VAD model is as follows:

$$f: \quad \mathcal{X} \rightarrow \mathcal{Y} \quad \text{where} \quad \mathcal{X} \in \mathbb{R}^m \quad \text{and} \quad \mathcal{Y} \in \{0, 1\}$$

where m is the dimension of the feature vector. The values in the discrete domain for $\mathcal{Y}$ are 0 for non-speech and 1 for speech.

Choosing the right feature vector can sometimes require a lot of thought and creativity. The process of discovering useful features is called Feature Engineering. Many features have been proposed for VAD models over the years, as discussed in Section 1.2.

There are two major considerations when engineering a VAD model. The first is accuracy: how well does the model perform at determining segments of audio that contain speech? The second is computational efficiency: how long does it take for the model to make a decision? The latency of a VAD model is important to consider when the model is intended to be deployed alongside real-time speech-processing

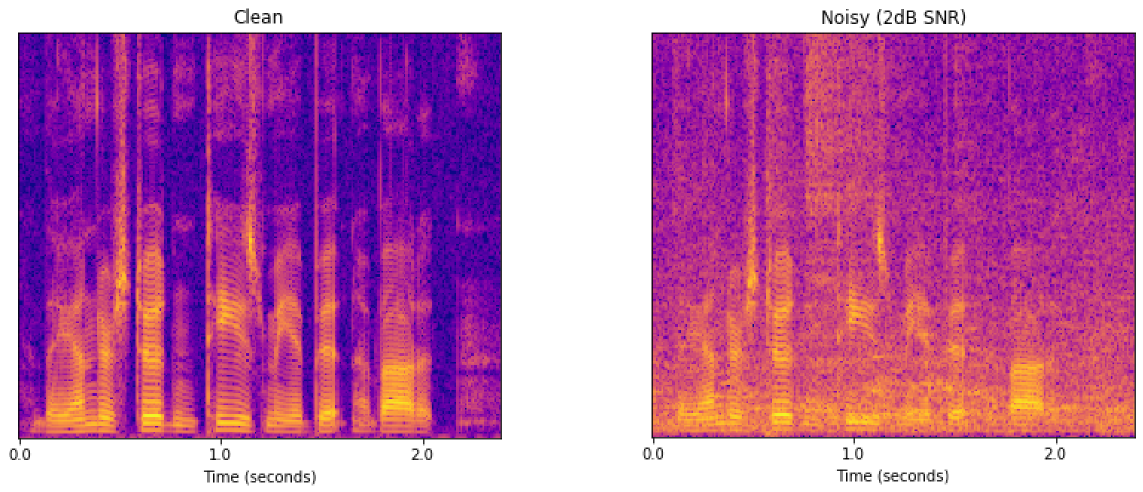


Figure 1-1: Spectrograms for a clean and noisy version of an identical utterance. This example demonstrates how the spectral information of a speech signal is obfuscated in the presence of ambient noise.

systems. An example would be a real-time transcription application that transcribes what you say as you say it. There is a fundamental trade-off between accuracy and computational efficiency. Larger, more complex models are generally more accurate but also require more computation.

An alternative perspective on the VAD problem is based on the following decomposition of raw audio samples into the sum of two components:

$$x_t = s_t + n_t \tag{1.1}$$

where x_t represents the raw audio, s_t represents pure speech, and n_t represents noise. Examples of noise may include wind against the microphone, birds chirping, machines humming, reverberations, loud impulse bursts or bangs, and an infinity of other non-speech sounds. From this perspective, the fundamental goal of VAD is to determine segments of audio in which s_t is non-zero.

One general approach to VAD goes by the following intuition from Equation 1.1: because we know x_t, if we hypothetically also knew n_t, then calculating s_t would be an easy problem. Alternatively, if we knew the energy of the noise, then we could use the energy of the raw audio to determine when the speech signal was non-zero. With this intuition, many early approaches to VAD focused on estimating the energy of noise (Rabiner and Sambur 1975; Lamel et al. 1981; Srinivasan and Gersho 1993; Ramirez et al. 2004). In practice, it's difficult to estimate this value because noise conditions are often highly non-stationary and unpredictable.

While some VAD models focus on estimating the underlying noise conditions, another option is to seek out characteristics in the audio that are unique to speech (Ghaemmaghami et al. 2010; Shen et al. 1998; Chuangsuwanich and Glass 2011; Bach et al. 2010). An example would be a VAD model that's trained to look for harmonics or other spectral characteristics indicative of sound generated from a human vocal tract. This is difficult when background noise is particularly loud relative to the speech signal (i.e. when the audio exhibits a low Signal-to-Noise Ratio (SNR)) because the speech signal is obfuscated by the ambient noise. Figure 1-1 shows an example of how intense background noise blurs a speech signal, especially in the higher frequency bands.

A VAD model that is resilient to a variety of different noise conditions is considered *robust*. The creation of robust VAD models has been an active area of research for several decades.

1.2 Prior Work

Over the years there have been a wide variety of different approaches for VAD, all with varying degrees of complexity. Many of the early methods aimed to differentiate speech from background noise based on measurable audio characteristics. These characteristics include short-time energy, zero-crossing rate, fluctuations in the long-term spectral envelope, harmonicity, spectral entropy, modulation frequency, and many others (Rabiner and Sambur 1975; Tanyer and Ozer 2000; Lamel et al. 1981; Srinivasan and Gersho 1993; Ramirez et al. 2004; Ghaemmaghami et al. 2010; Shen et al. 1998; Kristjansson et al. 2005; Chuangsuwanich and Glass 2011; Bach et al. 2010). Many of the VAD algorithms that exploit these characteristics are not composed of models trained on labeled data. Instead, they consist of hand-tuned heuristic parameters. More recently, statistical models for VAD have begun to take hold (Sohn et al.

1999; Zhang et al. 2016; Bai et al. 2017; Hughes and Mierle 2013; Zazo et al. 2016).

One of the simplest approaches, dating back to 1975, uses discrepancies in short-time energy to detect the boundaries of audio segments containing speech (Rabiner and Sambur 1975). This method assumes that during the first 100ms of recording there will be no speech, and that this interval will be indicative of future background noise. This 100ms interval is measured so that energy thresholds can be set accordingly. In addition to assuming stationary noise conditions a-priori, this method also assumes that any audio segments with short-time energy higher than the energy of the background noise must contain speech. It was shown to work well when the SNR was 30dB or higher (Rabiner and Sambur 1975).

The algorithm used in (Rabiner and Sambur 1975) also employed the use of the audio signal's zero-crossing rate (ZCR). ZCR, defined as the rate at which an audio signal fluctuates across the zero point[1], is generally higher for unvoiced speech than for voiced speech (Rabiner and Sambur 1975). This fact was used in (Rabiner and Sambur 1975) to increase the accuracy of VAD by discovering low-energy fricatives. (Tanyer and Ozer 2000) explored the use of ZCR under the assumption that ZCR is higher for background noise than for regions of audio containing voiced speech.

A slightly more robust VAD technique that uses short-time energy is to dynamically estimate the energy of the signal's background noise. With an accurate estimation, the signal's energy could be normalized by the energy of the background noise to produce an estimate of SNR. If at anytime this SNR estimate exceeds some threshold, the algorithm may be confident that the signal contains speech. Many different algorithms have been developed to produce estimates of the energy of a signal's background noise (Lamel et al. 1981; Srinivasan and Gersho 1993; Ramirez et al. 2004).

One method for estimating the energy of background noise, proposed in (Lamel et al. 1981), is to measure the short-time energies over individual frames, and then assume the background noise energy to be the peak of a smoothed histogram over the values that lie in the lowest 10dB range. This method involves making a-posteriori

[1]Calculating ZCR only makes sense when the audio's DC offset is removed.

estimates due to the fact that it looks at the entire utterance before estimating the background noise. Thus, this method is not meant for real-time VAD, but could nonetheless be adapted for real-time via rolling window updates.

(Srinivasan and Gersho 1993) proposes two techniques for estimating the energy of background noise based on non-speech pauses. One of these techniques, designed to be robust amidst non-stationary "babel-noise," employed the use of two separate VADs. The first VAD was responsible for making final decisions based on adaptive energy thresholds. The second VAD was responsible for recognizing periods of non-speech so that the adaptive energy thresholds could be updated. The second VAD determined periods of non-speech based on the assumptions that regions of speech exhibited higher fluctuations in short-time energy than regions of non-speech.

Similar to the idea updating thresholds during periods of non-speech, Ramirez *et al.* proposed a metric called long-term spectral divergence (LTSD) (Ramirez et al. 2004). LTSD represents the average ratio of the long-term spectral envelope (LTSE) to the noise spectrum across frequency bands:

$$LTSE_N(k,l) = \max_{-N \leq j \leq N} \{X(k, l+j)\}$$

$$LTSD = 10 \log_{10} \left(\frac{1}{NFFT} \sum_{k=0}^{NFFT-1} \frac{LTSE^2(k,l)}{N^2(k)} \right)$$

where $X(k,l)$ represents the amplitude spectrum for frame l and frequency band, k. The LSTE represents the maximum value for a particular frequency band across a window of $2N+1$ frames centered at frame l. The noise spectrum, $N(k)$, is recursively updated during periods of non-speech. This method is different from other energy thresholding methods because it explicitly accounts for changes within individual frequency bands across time.

A unique characteristic of human speech is harmonicity. During segments of voiced speech, especially vowels, periodic vibrations of the vocal cords produce sounds that are harmonically rich. The fundamental frequency (F_0) for voiced speech of adults generally falls in the 50-250Hz range (Rendall et al. 2005). One method of detecting highly periodic structure in a signal is through the autocorrelation function (ACF). The ACF measures the correlation between a signal and a time-delayed copy. For periodic signals, this correlation is maximized when the time delay equals an integer multiple of the fundamental period. Ghaemmaghami *et al.* used the following ACF:

$$R[z] = \frac{\sum_{i=1}^{n-z} x[i]x[i+z]}{\sum_{i=1}^{n} x^2[i]}$$

where $x[i]$ is the audio sample at index i, n is the number of samples, and z is the time delay. This function was used to measure the autocorrelation of noisy speech signals for time delays in the range of 2 to 20ms. For scores in this range, Ghaemmaghami *et al.* used the zero-crossing rate to estimate F_0, performing yet another cross correlation on these values to confirm the periodicity of the autocorrelation. This method was found to be very robust at detecting voiced speech amidst low SNR signals (Ghaemmaghami et al. 2010). To also capture unvoiced speech, Ghaemmaghami *et al.* used heuristic smoothing methods to capture any unvoiced speech that may precede or succeed segments of voiced speech.

Spectral entropy is a feature that quantifies the distribution of spectral density across frequency bands (Shen et al. 1998), denoted by:

$$H(l) = -\sum_{k=0}^{NFFT-1} \hat{\Phi}(k,l) \log \hat{\Phi}(k,l)$$

where $\hat{\Phi}(k,l)$ represents the normalized power spectrum, in which all values for a particular frame, l, sum to 1. The spectral entropy feature is analogous to the entropy of a probabilistic distribution, in which a particular frequency band has a “probability” of being activated by a particular sound. It is assumed that the spectral entropy of speech is lower than the spectral entropy of background noise because “probability density” is spread across fewer frequency bands in speech relative to noise. This method is more robust than traditional energy thresholding in the presence of non-stationary noise, but is susceptible to false alarms from non-speech sounds that equivalently posses localized “probability density” across the frequency space.

Another highly discriminatory feature in the presence of background noise is modulation frequency (Chuangsuwanich and Glass 2011; Bach et al. 2010). Modulation frequency is represented by the temporal changes in energy across frequency bands. Human speech has been shown to possess a fundamental modulation frequency of around 4Hz (Drullman et al. 1994). Chuangsuwanich *et al.* use a harmonicity measure to discover candidate segments of audio, which are then fed into individual SVM classifiers. Each SVM takes as input the modulation spectrum for a sum of energies across neighboring frequency bands. The outputs from each of the SVMs are compressed via sigmoid, and then summed to obtain a final value upon which thresholding can be applied to decide speech or non-speech. While this method works well in low SNR environments, computing modulation frequency requires a large window, so decision latency is high. Chuangsuwanich *et al.* found that computing modulation frequencies on windows of length 500-1000ms gave reasonable results.

With the exception of the use of SVM classifiers in (Chuangsuwanich and Glass 2011), most of the methods described so far have been highly heuristic in nature. Statistical learning models that depend on labeled data have also been explored for VAD.

Sohn *et al.*. proposed a statistical model-based VAD which modeled the DFT coefficients of speech and noise as two separate multivariate Gaussian distributions (Sohn et al. 1999). Sohn *et al.* formulated a likelihood ratio test between the probability of data under the speech distribution and the probability of data under the noise distribution. The geometric mean of the likelihood ratio of each frequency band was used to output a final value upon which thresholding could be applied. Sohn *et al.* layered a Hidden Markov Model (HMM) on top of this model to serve as a hangover scheme, under the assumption that consecutive speech frames are strongly correlated. This HMM hangover scheme significantly improved speech detection with a relatively small increase in the false-alarm rate.

With computing power becoming a commodity, and the recent prominence of Deep Learning, the use of Artificial Neural Networks (ANNs) have become popular for VAD. Su *et al.* had recent success in using a 2D convolutional neural network

(CNN) on spectrogram features to detect candidate speech segments for pitch classification (Zhang et al. 2016). The CNN was used to exploit the highly shift-invariant structure of harmonics in the spectrogram. While this was not exactly a VAD task, it demonstrated that CNNs could be used to accurately find regions of an audio signal containing harmonics, similar to the way periodicity or harmonicity can be used to find candidate regions for detecting speech (Ghaemmaghami et al. 2010; Chuangsuwanich and Glass 2011).

The use of fully connected neural networks (DNNs) have also been explored for VAD. Bai *et al.* explored variants of a 3-hidden layer architecture (Bai et al. 2017). The small variant had 128 nodes per hidden layer, while the big variant had 512 nodes per hidden layer. Similar to (Sohn et al. 1999), Bai *et al.* also layered an HMM on top of the DNN to capture temporal context and smooth fluctuations in decisions. While the big variant was more expressive due to having more non-linearity and trainable parameters, its performance increase over the small variant was negligible. The big variant was over 64 times slower in making decisions, however (Bai et al. 2017).

The underlying models and algorithms discussed so far have made a largely incorrect assumption about a noisy speech signal: the existence of speech within a discrete segment is independent of other segments. The use of HMMs in (Sohn et al. 1999; Bai et al. 2017) attempt to ameliorate this unrealistic assumption by conditioning decisions on not only the output from the underlying model, but also on the decision from the previous frame. This has the effect of penalizing transitions between the speech and non-speech states, and thus forming much smoother decision sequences. In (Hughes and Mierle 2013), Hughes *et al.* point out that while HMMs help to provide contextual information when making decisions, they still possess a few fundamental flaws:

1. Audio frame labels are not conditionally independent. HMMs assume that the label for a particular frame is conditionally independent of all other labels given the labels of its neighboring frames. This is largely a simplification, and is not true in practice.

2. For VAD tasks, the HMM hidden state space is often finite and binary: speech and non-speech. This presents a fundamental limitation about the information that can be conveyed by a particular state. Having more states with richer information could improve decision making when conditioning on these states. Engineering more states, however, is largely a heuristic method, and it's not clear how to reconcile these with an underlying model that simply outputs the likelihood that a frame contains speech.

3. HMMs and underlying models cannot be jointly trained. First, the underlying model must be optimized, and then the HMM can be trained.

Hughes *et al.* employs the use of Recurrent Neural Networks (RNNs) for VAD, which solve a lot of the problems that are inherent to the HMM-based architectures. RNNs are advantageous because they can be trained to jointly minimize frame-level errors while also learning a continuous hidden state space that captures useful temporal context which goes back an arbitrary amount of time. The architecture in (Hughes and Mierle 2013) consisted of multiple recurrent cells and feed-forward components, and also made use of an unconventional quadratic activation scheme. It was shown to significantly outperform a GMM-State Machine model, while using one tenth the parameters (Hughes and Mierle 2013).

1.3 Purpose and Overview

The purpose of this thesis to explore the use of deep, recurrent neural models for VAD. This includes exploring the process of learning features directly from raw waveform data, and comparing the pros and cons of this process to using a more traditional feature representation. In addition, this thesis evaluates the efficacy of artificially generating large amounts of noisy speech data from a clean speech source. The models explored in this paper are intended to be computationally efficient so that they may be deployed with real-time responsiveness while taking input from a live audio stream. The ultimate goal is to produce a live VAD model that is robust to a variety of acoustic environments, noises, and voices.

The outline of this work is as follows. In Chapter 2, I discuss the characteristics of a strong dataset, and then present a methodology for artificially inflating a corpus of labeled data. I then discuss the strengths and limitations of this methodology. In Chapter 3, I provide a background on neural networks, and then present the models that are explored in this paper. Chapter 4 describes the the various experiments that are performed using these models, and discusses details about how the models are trained. In Chapter 5, I discuss experimental results and analyze in-depth how features are learned directly from raw waveform data. Lastly, Chapter 6 describes how these models can be deployed and run in real-time with a live audio stream. This chapter also discusses some of the practical concerns of deploying these neural models.

Chapter 2

Dataset Preparation for Statistical VAD Models

A statistical model is a mathematical construction that attempts to approximate reality based on some finite sample of data. The goal of any statistical model is to maximize approximation performance on unseen data (i.e. data that was not used to train the statistical model). A statistical model uses the finite sample of data to make assumptions about reality. For this reason, the dataset is one of the most important considerations when engineering a statistical model.

Not all datasets are created equal. When seeking a good dataset, there are many things to consider. At a high level, it's important that the dataset contains a wide variety of samples that are representative of reality. In many scenarios, the data domain is continuous and infinite, and the distribution of data points may contain many distinct modes. It's important to capture samples from as many of these modes as possible. Unfortunately, this is difficult in practice, and with only a finite dataset, it's impossible to capture a set of samples that is 100% representative of reality. Thus, a "best effort" mindset is needed. One of the most obvious but difficult ways to ameliorate the problem of capturing a representative dataset is to simply obtain *more* data. Again, this is difficult in practice.

In addition to the problem of capturing a representative dataset is the problem of labeling the dataset. For supervised learning models, labeled data is required so that during training, an error function can be minimized. The labels are used to recognize when the model makes a mistake for a particular sample. This information is then used to nudge the model's parameters in the direction such that the degree of error can be reduced. For most corpora, labeling of raw data is performed by humans, which can very costly and time consuming, especially when datasets are large and labels precise.

Due to the costly and difficult process of capturing large amounts of diverse, labeled data, it has become common to generate data artificially. Not only is it possible to generate data artificially, but it's also possible to perturb already acquired data in a way that labels are preserved (Taylor and Nitschke 2017; Dean et al. 2010; Kim et al. 2017). For example, in object recognition tasks, images can be flipped, rotated, warped, or re-colored while the identity of the object in the image remains the same. In this scenario, labeled images can be replicated and perturbed to artificially inflate the amount of data in the original dataset. These perturbations, while artificial, cover a wider surface area of the entire data manifold, and therefore may help a statistical model generalize to unseen data. This technique is a great way to save time, effort, and money, and it can be applied to a variety of other data mediums besides images.

For constructing statistical models that perform VAD, the data domain is represented by the set of all possible audio segments. These audio segments may contain speech, background or ambient noise, impulse noises, etc.; there is an infinite number of possibilities for noise-speech combinations or the lack thereof. It's important for the dataset to contain recordings from a variety of different noise conditions in order for the model to be robust amidst noise. In addition, it's a requirement that each data sample contain high-precision labels that indicate the locations in the signal that bound segments of speech.

To capture a diverse dataset of noisy speech data, one option would be to record speech from many locations: an office, cafe, busy street, amusement park, family room, construction site, etc.—the possibilities are endless. Each recording would then have to be hand-labeled. Fortunately, this costly and arduous option can be bypassed in lieu of an artificial dataset generation technique. It may be appropriate to assume the breakdown from Equation 1.1. That is, a noisy speech signal simply represents the sum of clean speech and noise. With this formula, noisy speech can be generated by simply taking the sum of two distinct and separate recordings: clean speech and non-speech noise. This process assumes, however, that speech and noise are independent sources of sound, which is not true in practice.

2.1 The Augmentation Process

There already exist many clean-speech labeled corpora, as well as corpora that contain various background noises, non-speech sounds etc. In this work, I use clean-speech recordings from the TIMIT[1] corpus, alongside the noise recordings from the QUT-NOISE corpus to generate a large dataset of noisy speech recordings (Dean et al. 2010). In addition, I take non-speech sounds from various sources to serve as negative samples.

The well-known TIMIT corpus is a collection of single-sentence utterances each roughly 3 seconds in length. The corpus includes 6,300 speech files recorded from 192 female and 438 male participants. Each recording is accompanied by phonetic and word-boundary segmentations, suitable for a wide variety of speech-related tasks including ASR, VAD, etc. Each recording was made using a close-talking microphone, and contains a negligible amount of ambient noise. Thus, the recordings are considered very clean.

The QUT-Noise database is a corpus containing over 10 hours of ambient noise from various locations (Dean et al. 2010). The corpus also includes room impulse responses for some of the enclosed locations. Locations include the following: outdoor cafe, food market, home kitchen, home living room, inner-city street, outer-city street, car (windows up), car (windows down), indoor pool, a partially-enclosed carpark. The QUT-Noise corpus was originally assembled to be combined with clean speech corpora like TIMIT for VAD training and evaluation.

In addition to having noisy speech data, its important to have non-speech sam-

[1]https://catalog.ldc.upenn.edu/ldc93s1

ples. During training, negative samples help the model to learn characteristics of sound that are inherent to speech. Without negative samples, it's likely that the model would become "lazy," learning to classify perturbations in short-time energy (or other high-level characteristics) as endpoints for speech. A non-speech sound may have similar high-level characteristics to a segment of speech, so the lack of negative samples during training could result in a model with a high false-alarm rate. I take non-speech samples from the ESC-50 dataset, a collection of 2000 recordings of various environmental sounds, including animals, natural soundscapes, non-speech human sounds, interior domestic sounds, and exterior urban sounds (Piczak 2015). In addition, I also take samples from a collection assembled by Guoning Hu[2].

The data augmentation process I created is as follows:

1. All samples are converted to 32-bit floats if not already. Each sample is scaled to lie in the range of $[-1.0, 1.0]$.

2. For each TIMIT utterance, random padding is concatenated to the beginning and end. This padding consists of complete silence, in the form of zero vectors. The durations of beginning and end padding are chosen independently and uniformly at random in the interval of 0.5 to 2.0 seconds. It's important to add random padding so that the model doesn't learn any artificial offsets inherent to the TIMIT database. For example, each utterance in the out-of-the-box TIMIT corpus begins after roughly the same duration of silence, which a contextual model could inadvertently learn.

3. For each utterance, there is a 20% chance that an additional utterance will be chosen uniformly at random to be concatenated. This additional utterance will receive the same padding as in step 1 before being concatenated. Concatenations are important so that the model, if capturing temporal context, can learn to switch on, off, and then on again. If all utterances consisted of only a single, contiguous region of speech—with little silence—the model may learn that once it turns "on," it should stay on and vice versa. This bias is detrimental.

[2]http://web.cse.ohio-state.edu/pnl/corpus/HuNonspeech/HuCorpus.html

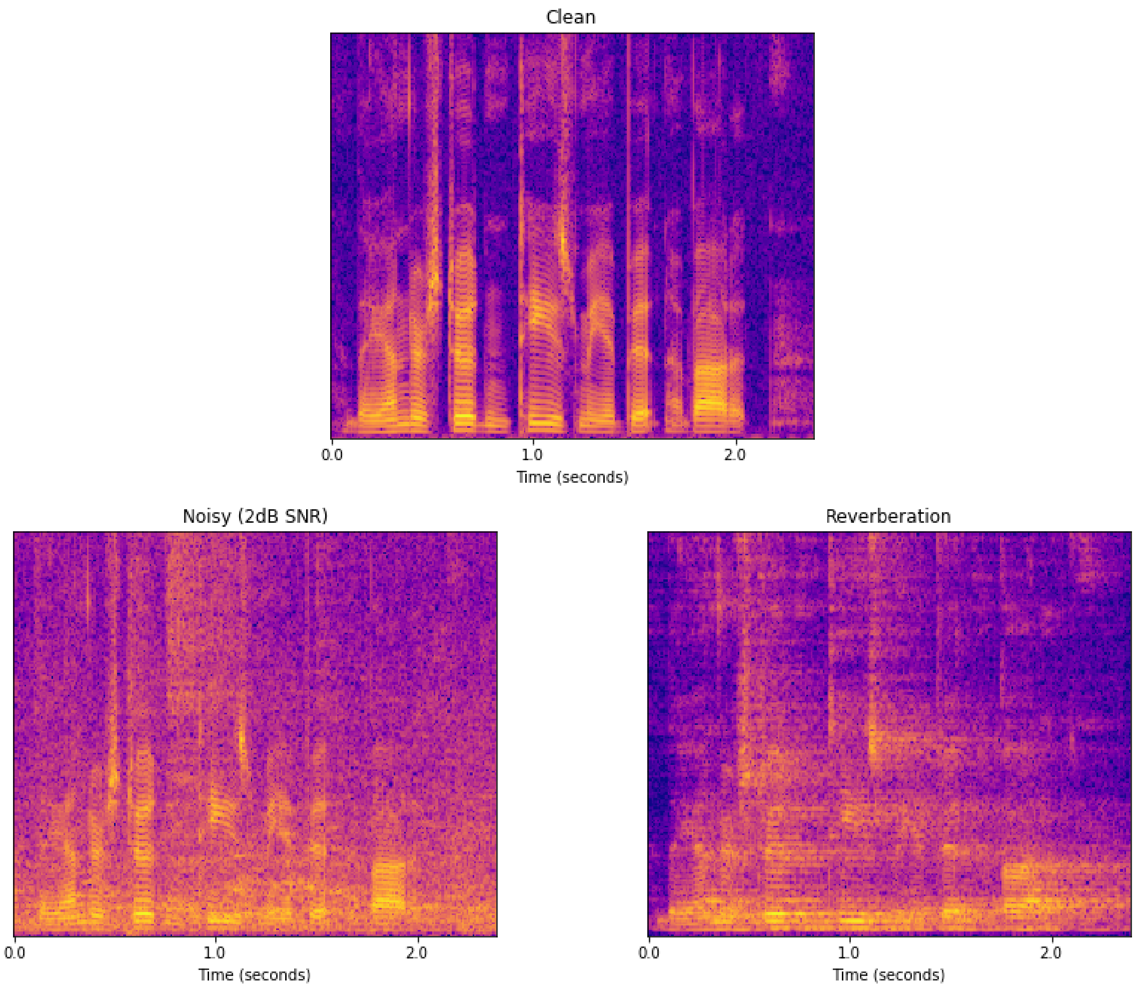


Figure 2-1: The spectrograms for the spoken utterance, "They understood and teased me a bit about it.", in three different simulated noise conditions. In the 2dB SNR condition, the spectrogram appears blurred, especially in the higher frequency bands. In the reverberation condition, the spectrogram appears to be smeared in time.

4. For each utterance, one of the noise environments provided in the QUT-Noise database is chosen uniformly at random. Then, a noise file from this environment is randomly chosen. If available from this noise environment, a room impulse response (RIR) is also gathered.

5. If available, the RIR is convolved with the current utterance, now consisting of any padding and concatenations. This process simulates reverberation, a phenomenon that occurs when sound waves bounce off surfaces in the surrounding environment and reach the listener with a time delay and reduction in energy. Reverberations are an example of noise that cannot be simulated through the simple addition of clean speech and noise.

6. Accounting for padding and any concatenations, the length of the current utterance is calculated. This length is then used to choose a contiguous segment of noise uniformly at random from the randomly chosen noise file.

7. Before the noise is added to the utterance, an SNR is chosen from a uniform categorical distribution with the following candidate values: -5.0, 0.0, 2.0, 4.0, 6.0, 8.0, 10.0, 15.0, 20.0, and 'clean.' The noise file is scaled by a constant factor so that once added to the utterance, the signal exhibits the desired SNR. After adding the noise, the signal is scaled so that the maximum absolute value of a particular sample is not greater than 1.0.

8. A random gain scale factor is chosen uniformly at random from the range of $[0.66, 1.50]$. The noisy speech signal is then multiplied by this factor and clipped to be in the range of $[-1.0, 1.0]$. The factor is used to simulate different signal gains so that the model does not overfit to a particular gain.

9. Throughout the augmentation process, the timing offsets introduced by any paddings or concentrations are maintained, so that new labels can be calculated. For example, if a 2 second padding is prepended to an utterance, the new speech-endpoint labels must also be offset by 2 seconds.

The out-of-the-box TIMIT database comes pre-partitioned with train and test sets. For the train set, 3 passes of augmentation were performed to produce 3 times the number of original utterances. For the test set, only 2 passes of augmentation were performed. Accounting for any padding and concatenations, utterances that were longer than 14 seconds were discarded. I established 14 seconds as the maximum length because so few utterances exceeded this limit, and a limit needed to be set. This limit serves as the length to which all utterances are zero-padded during training, so it was desirable to have a tight bound. The augmented train set was further partitioned into train and dev sets, with the dev set having the same number of samples as the test set. The non-speech sounds were also partitioned with a 4-1-1 ratio of train, dev, and test sets, respectively. Figure 2-1 demonstrates the effects of data augmentation on the spectrum of a particular TIMIT utterance.

The full dataset, including all 3 partitions, contains $19,186$ utterances, accounting for over 34 hours of audio. Figure 2-2 shows the distribution of lengths of utterances. In addition, Figure 2-3 demonstrates heat maps that represent the average label densities for utterances. Table 2.1 provides more in-depth statistics and breakdowns of the dataset.

2.2 Limitations

While this method and similar augmentation methods are great for producing large quantities of labeled, noisy speech samples, there are some limitations:

- All utterances come from the same microphone. In practice, different microphones exhibit subtle differences, so a model that wasn't trained on audio from a variety of different microphones may be easily confused.

- The underlying speech is clean and spoken in close physical proximity to the microphone. In reality, the model may need to classify audio that is spoken at farther distances from the microphone. The energy of a speech signal decays as it travels further distances, a behavior that is not accounted for in the dataset.

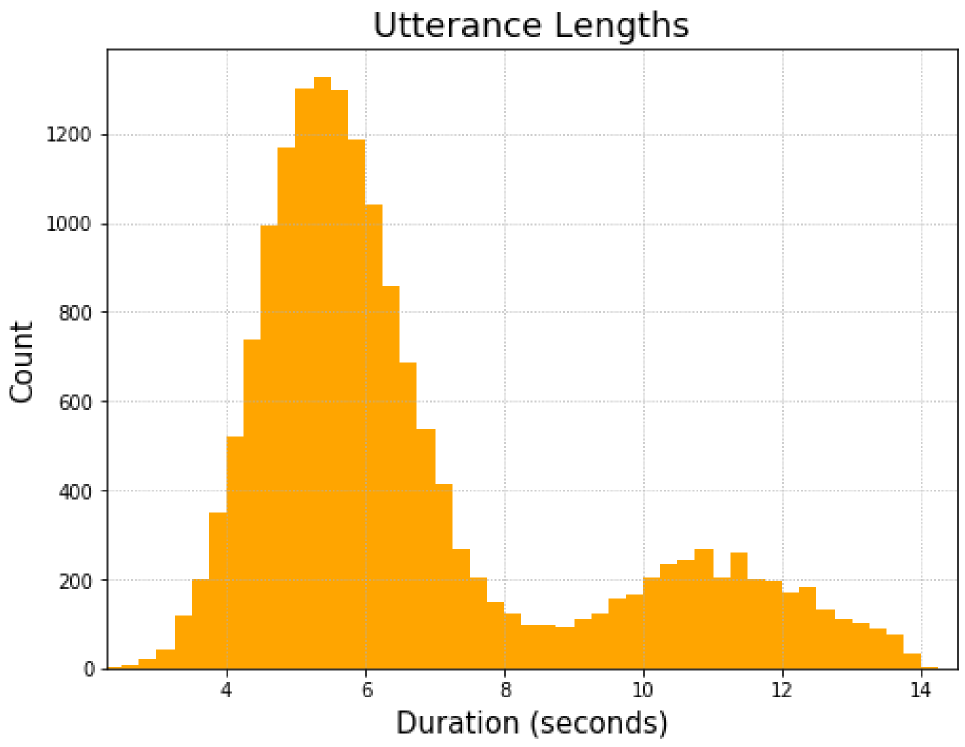


Figure 2-2: A histogram of utterance lengths for samples outputted by the data augmentation process discussed in Section 2.1.

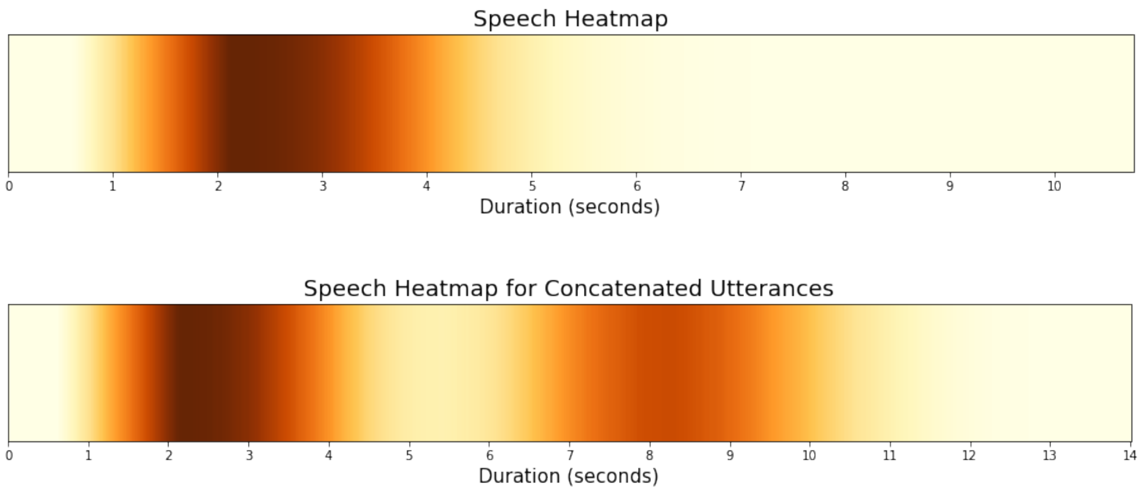


Figure 2-3: Heatmaps which represent the average label densities for utterances outputted by the data augmentation process discussed in Section 2.1. Darker regions indicate a higher proportion of speech at that location in time.

- The TIMIT dataset contains many utterances that represent the same spoken phrase, but spoken by different people (out of the 4,620 utterances in the TIMIT train partition, the phrase "She had your dark suit in greasy wash water all year" is repeated 462 times by different speakers. The phrase "Don't ask me to carry an oily rag like that" is also repeated 462 times by different speakers). In addition, the augmentation process creates copies of identical utterances, but with different noise conditions. While these utterances are varied by speaker and noise conditions, the inherent similarities may be characteristics of the dataset that the model falsely attributes to being inherent to speech, thus hindering its ability to generalize.

Many other methods have been explored for artificially creating labeled data. One of these methods is to re-record audio by audibly playing the samples from a speaker in a variety of different rooms, noise conditions, and distances from the microphone. Another method is to take large corpora of transcribed audio, and to obtain segmentation labels via a process known as forced alignment. This process uses an ASR model to align textual transcriptions to their temporal locations in the audio. Another process is to sample noise configurations from virtual rooms, to be used when augmenting clean speech (Kim et al. 2017).

Table 2.1: A breakdown of the dataset generated from the process described in Section2.1.

Augmented TIMIT Dataset Breakdown				
	train	dev	test	total
-5 SNR				
utterances	1042	332	353	1727
duration (h)	1.92	0.62	0.63	3.18
0 SNR				
utterances	1015	321	346	1682
duration (h)	1.88	0.58	0.65	3.11
2 SNR				
utterances	1044	354	331	1729
duration (h)	1.93	0.65	0.64	3.21
4 SNR				
utterances	1040	321	346	1707
duration (h)	1.91	0.60	0.62	3.13
6 SNR				
utterances	1058	352	318	1728
duration (h)	1.99	0.64	0.60	3.23
8 SNR				
utterances	1010	327	326	1663
duration (h)	1.83	0.60	0.61	3.04
10 SNR				
utterances	1021	342	323	1686
duration (h)	1.89	0.63	0.61	3.13
15 SNR				
utterances	1068	327	317	1712
duration (h)	1.98	0.60	0.56	3.14
20 SNR				
utterances	1051	335	327	1713
duration (h)	1.98	0.61	0.62	3.21
clean				
utterances	1068	326	345	1739
duration (h)	2.04	0.63	0.63	3.30
sounds				
utterances	1386	357	357	2100
duration (h)	1.88	0.49	0.49	2.86
Totals				
utterances	11803	3694	3689	19186
duration (h)	21.23	6.64	6.66	34.54
% speech	43.33	44.07	44.24	43.65

Chapter 3

Neural Models for VAD

Inspired by the recent use of deep, recurrent neural models for VAD (Hughes and Mierle 2013; Zazo et al. 2016), I explore my own variant of a relatively lightweight deep recurrent architecture. In this context, lightweight means that the computation required by this architecture is relatively small. One of the overarching goals of this model is to make it fast enough to be deployed for real time operation on a live audio stream. Despite being relatively lightweight, it's still important that the model performs well.

3.1 Artificial Neural Networks

Artificial Neural Networks (ANNs) are a class of biologically-inspired computation models. ANNs consist of a connected network of units, also called neurons. Traditionally, neural networks are arranged in layers, consisting of neurons which each compute a non-linear activation of a linear combination of outputs from neurons in the previous layer. The multiplicative weights for these linear combinations correspond to the weights of the connections between neurons. Each neuron also has its own scalar bias term which contributes to this linear combination. These weights and bias terms can be learned such that the neural network computes a useful vector mapping $\mathcal{X} \to \mathcal{Y}$ of its inputs, $\mathcal{X}$. Figure 3-1 shows a traditional single hidden-layer neural network.

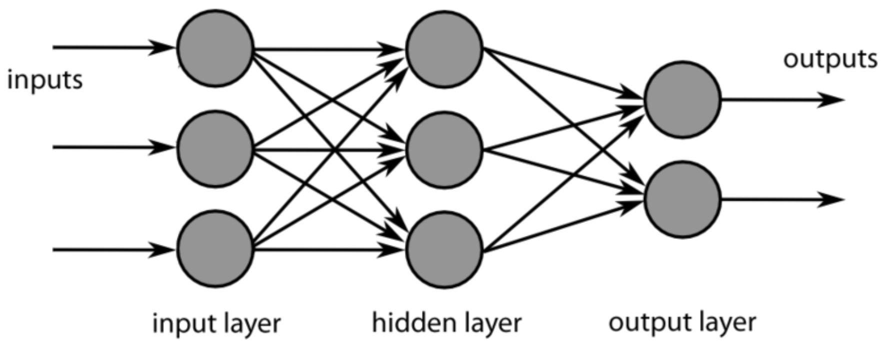


Figure 3-1: An artificial neural network (ANN), in one of its simplest forms.

During training, a neural network's weights and biases are iteratively nudged in a direction that minimizes some loss function. A loss function gives a quantitative description of how well the network is performing. For supervised learning tasks, typically a loss function will accept as input pairs of training samples and labels, as well as the current values of the weights and biases. Assuming the loss function is differentiable, gradients can be computed for each weight and bias vector in the network. If you imagine the loss function as a multi-dimensional surface, the gradients represent a vector that points in the direction of greatest increase for any point on this surface. Thus, if the learnable parameters are nudged in the opposite direction of the gradient, the loss function is being effectively minimized. In many cases, the multi-dimensional loss function is non-convex. There often exist many local minima, some with lower loss values than others. Gradient Descent optimization methods do not guarantee that the best local minimum is found. Nonetheless, if there do exist local minima, gradient descent methods are capable of finding one. Despite this flaw in deep neural networks, they are still extremely powerful learning models, especially for high-dimensional input spaces.

3.1.1 Recurrent Neural Networks

A Recurrent Neural Network (RNN) is a type of ANN that is useful for computing transformations on sequential data. Sequential data often exhibit the property that individual samples are not independent of each other. An RNN is able to make use of these dependencies by storing intermediate state between successive computations. Effectively, an RNN is able to "remember" useful information. Non-recurrent networks make the assumption that each sample is independent, and thus are not designed to remember any information.

RNN's are generally implemented like traditional feed-forward neural networks with one main exception: some intermediate computation vector (i.e. the values outputted by the hidden layer in Figure 3-1) is fed as input to the next computation. Thus, an RNN accepts not only the data sample as input, but also a vector that represents some intermediate state from the previous computation. During the very first computation of a sequence, such an intermediate state does not exist. In this case, a zero-vector or Gaussian noise is often used.

RNNs are extremely powerful because in addition to being optimized to learn a desired vector-mapping, they can be optimized to learn a hidden state space that stores useful information. In fact, it has been proven that finitely-sized RNNs are capable of mimicking a universal Turing Machine (Siegelmann and Sontag 1991). While very powerful in theory, RNNs are difficult to train in practice. Many have hypothesized reasons for the difficulties in training RNNs, and a common argument is that RNNs have trouble modeling long-term dependencies due to the vanishing gradient problem[1]. Specialized recurrent cell architectures, like LSTM and GRU, help to ameliorate some of the problems associated with vanishing gradients, and therefore make training easier (Hochreiter and Schmidhuber 1997; Cho et al. 2014).

3.1.2 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are another type of neural network which have been applied extremely successively to the field of Computer Vision, but are also useful in speech and natural language processing. CNNs are similar to traditional feed-forward neural networks like the one depicted in Figure 3-1, but are unique because of their ability to learn shift-invariant structure. Shift-invariant structure may be defined as the existence of a localized pattern at any location in a larger input space.

[1]Similarly, RNNs may suffer from an exploding gradient problem, which makes learning unstable.

For example, in an image, a cat may appear in the middle, bottom, top, left, right, etc.—the pattern still represents a cat regardless of where it may be located in the image. CNNs work by convolving a window, or *kernel*, across the input space, such that upon each discrete shift, a nonlinear activation of a linear combination of inputs encompassed by the window is computed (Nielsen 2015). A unique feature of CNNs is that the multiplicative weights associated with this window are shared for all discrete shifts. This process can be repeated with more windows, such that each window gets its own set of multiplicative weights. After convolving all windows across the input space, the output is a set of transformations called *feature maps*, equal in number to the number of windows that were convolved across the input space (Nielsen 2015). When trained, each feature map learns to seek out a shift-invariant pattern(s) from the input space.

3.2 Feature Computation: Mel Filter Banks

As briefly mentioned in Section 1.1, a feature vector is an alternative representation of raw data that accentuates discriminatory information. For one of my model variants, I use log Mel filter banks as input features.

Log Mel filter bank features are computed by mapping the energy spectrum for a particular frame of audio to the Mel scale, and then taking the log of result. The Mel scale is a biologically-inspired, non-linear scale of frequencies based empirically on pitches that humans perceive to be equidistant from each other (Stevens et al. 1937). This scale places greater resolution in the lower frequencies than the higher ones.

The steps to compute log Mel filter bank features are as follows:

1. Apply a pre-emphasis filter to the raw audio. A pre-emphasis filter has the effect of boosting the presence of higher frequencies, which is useful because higher frequency signals tend to decay faster than lower frequency ones. Higher frequency bands can provide useful discriminatory information when deciding on speech or no-speech. I use a pre-emphasis coefficient of 0.97, which is commonly used.

2. Frame the audio. This process involves splitting raw audio into a series of overlapping frames. A common convention is to have a frame length of 25ms and frame hop of 10ms. This means that any given frame shares 15ms with its neighboring frames. For audio sampled at 16kHz, the frame length is 400 samples while the frame hop is 160 samples.

3. Window each frame. This step is an important preprocessing step before step 4, which involves taking the discrete Fourier transform (DFT). When taking the DFT over a finite segment of audio, the resulting spectrum is susceptible to a phenomenon called spectral leakage. Spectral leakage occurs when the density of one frequency band "leaks" into neighboring frequency bands. This leakage may have have the effect of obfuscating the presence of weaker frequency components. Different window functions, when element-wise multiplied with the audio signal, result in different leakage patterns, as characterized by observing the DFT of the window function itself. Because multiplication in the time domain corresponds to convolution in the frequency domain, window functions with relatively dominant main lobes are preferred because leakage from neighboring frequencies is less prevalent, relatively speaking. The Hamming Window is a commonly used window that helps to emphasize a sharper frequency response.

4. Compute the Energy Spectral Density of the audio frame. This step involves taking an N-point DFT of the windowed audio frame. After performing the DFT, only the first half of the coefficients are relevant due to symmetry about the Nyquist frequency. The magnitude of the DFT coefficients are then squared to get the Energy Spectral Density of the audio frame. When the Energy Spectral Density of each frame is lined up side-by-side, the result is a spectrogram (Figure 3-2).

5. Map the Energy Spectral Density to the Mel scale. This is done by applying a series of triangular filters who's peaks equally spaced on the Mel scale. These filters may be collectively represented as a matrix, in which each row represents

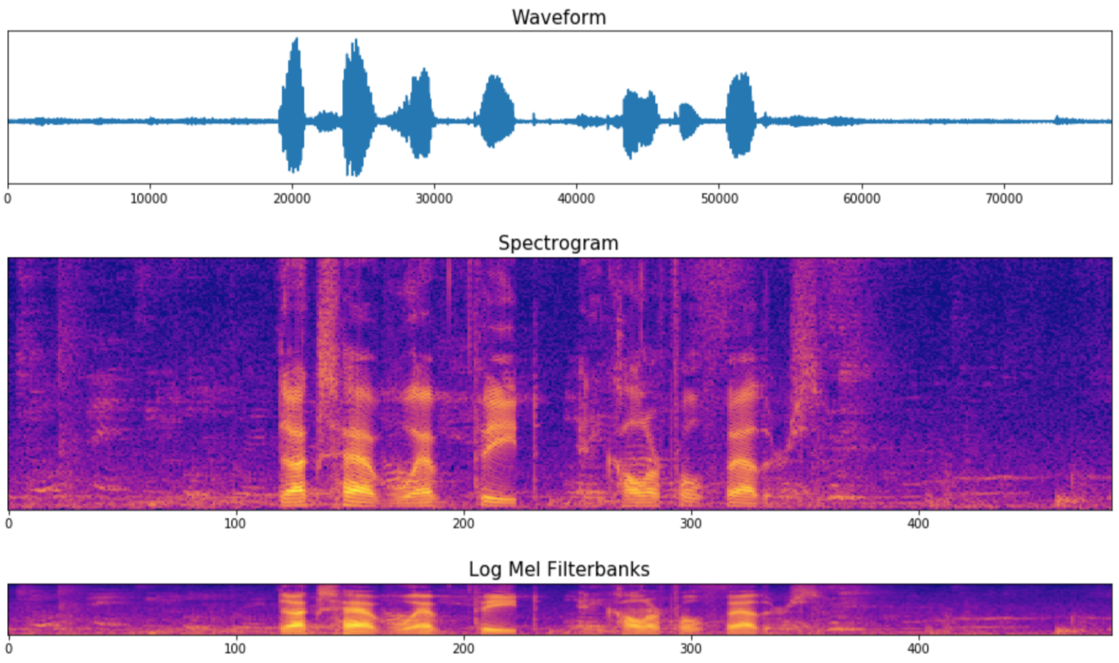


Figure 3-2: A side-by-side comparison of the raw waveform, spectrogram, and log Mel filter bank features for the utterance "Ducks have webbed feet and colorful feathers." at 15dB SNR.

a filter. This matrix is multiplied by the Energy Spectral Density to produce Mel filter bank coefficients. The number of Mel filter bank coefficients can be chosen depending on the desired resolution. 40 is a commonly used number.

6. The Mel filter bank coefficients are then scaled by their maximum value and converted to dB units by taking the base-10 logarithm and then multiplying by 10.

Log Mel filter bank features are particularly useful for speech tasks due to their biological grounding and relatively concise representation. A one second audio recording sampled at 16kHz may be represented by 16,000 floating point values, while the corresponding log Mel filter bank representation consists of just 4,000 floating point values. Figure 3-2 demonstrates a comparison of a raw waveform, spectrogram, and log Mel filter banks side-by-side.

3.3 Architecture

The model variant that I explore for VAD is a deep, recurrent neural architecture. Dubbed CNN-GRU, the network consists of a 1D convolutional layer, followed by 3 GRU cells, and then a fully-connected layer for computing logits. The logits are used in a 2-class softmax, which outputs the probability of speech. CNN-GRU takes as input 40-dimensional log Mel filter bank features. The 1D convolutional layer convolves across the entire frequency dimension. In its traditional form, it only takes in one frame at a time[2]. A depiction of the CNN-GRU architecture is in Figure 3-3.

3.3.1 GRU

Due to the practical difficulties in training traditional RNNs—a problem briefly discussed in Section 3.1.1—it was necessary that the network use a recurrent cell that was more resilient to vanishing and exploding gradients. Two of the most widely used recurrent cells are LSTM and GRU (Hochreiter and Schmidhuber 1997; Cho et al. 2014). Both LSTM and GRU use learned gating functions to control the flow of stored information between forward passes in the network. The primary difference between LSTM and GRU is that LSTM has a dedicated memory unit that is separate from the hidden state. This memory unit is updated via learned gating functions and is used when computing the hidden state for a particular forward pass. GRU cells are different in that they update the hidden state directly, having no concept of a dedicated memory unit. GRU cells have 2 learned gating functions while LSTMs have 3. GRU cells, therefore, have fewer parameters than LSTM when controlling for the number of units. GRU has been compared to LSTM, and it was inconclusive which performed better for modeling sequential data (Chung et al. 2014). The experiments in (Chung et al. 2014) controlled for the number of learnable parameters, not the number of units. While inconclusive which cell is better, I chose GRU cells due to

[2]A slight variation of CNN-GRU will accept a contiguous slice of frames, such that consecutive slices poses overlap with each other. If any of these frames contain speech, the whole slice is labeled as containing speech. A large slice may allow a network to learn speech characteristics that span multiple frames of time. While this functionality is supported, I do not explore it in this work.

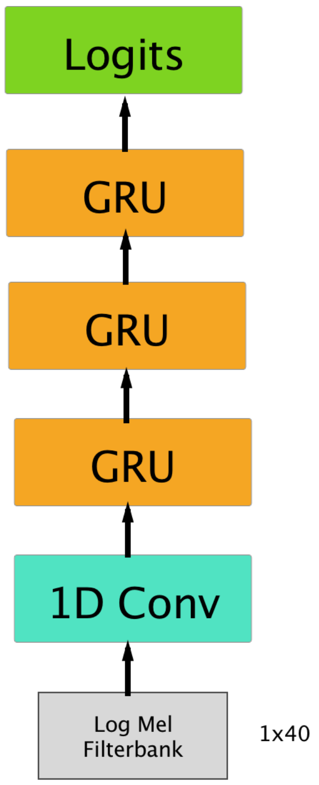


Figure 3-3: The CNN-GRU architecture. The network accepts as input a single frame of 40-dimensional log Mel filter bank features. The network consists of a 1D convolutional layer, followed by several GRU cells and then a fully-connected layer for computing logits.

their relative simplicity over LSTM. This decision was motivated by a having a computationally efficient neural architecture, but a careful comparison of the performance differences between LSTM and GRU is needed to confirm or deny that GRU is more computationally efficient, controlling for performance.

3.3.2 Raw Waveform Feature Learning

When praising the inherent power of deep neural networks, researchers often point to their ability to jointly learn features while optimizing some objective function. A deep neural network can be considered a hierarchy of feature computation. Each layer in the network can be considered responsible for computing a useful representation of the input data. Each layer computes new features based on the representation outputted by the layer before it, and after many such layers, an abstract representation is achieved upon which a classifier can be employed.

Due to the ability of deep neural networks to learn features, fully end-to-end neural models have gained popularity (Zazo et al. 2016; Sainath et al. 2015; Bojarski et al. 2016; Graves and Jaitly 2014). A fully end-to-end neural model takes raw data as input, and is not composed of sub-components that are independently trained. Instead, the entire network is simultaneously trained under the same loss function. The CNN-GRU model presented in Section 3.3 is not fully end-to-end because it accepts log Mel filter bank features as input. Computing log Mel filter banks may be considered a heuristic method because the process is not learned from data. In addition, computing log Mel filter bank features is a lossy process in which useful information may be destroyed. One may wonder if log Mel filter banks are the best representation. What if a neural network could learn its own transformation directly from a raw waveform? What if this representation is superior to log Mel filter bank features? These questions provide motivation for the exploration of fully end-to-end networks. With inspiration from (Zazo et al. 2016), I explore a variant of CNN-GRU dubbed CNN-GRU-RAW. CNN-GRU-RAW is identical to CNN-GRU with the exception of an extension prepended to the beginning of the network that extracts features directly from a raw waveform. The extension is designed to extract features with the same dimensions as 40-dimensional log Mel filter banks. A depiction of CNN-GRU-RAW is in Figure 3-4.

The extension, based on the architecture from (Zazo et al. 2016), consists of a 1D convolutional layer, max pooling, and stabilized log. The 1D convolutional layer convolves 40 separate filters across the waveform, with no non-linear activation. These convolution vectors are then max-pooled across the entire time dimension, to produce a 1D vector with 40 values. Then, this vector is passed through a ReLU activation and stabilized logarithm. The intuition is that these filters may be considered impulse responses which, when convolved across a window of raw audio, produce spectral coefficients. The maximum values within the vectors resulting from these convolutions are large when there is a high overlap between the spectrum of the raw audio and that of the impulse response. Effectively, the CNN-GRU-RAW has the theoretical ability to learn a combination of filters that seek out the optimal spectral information from the raw waveform.

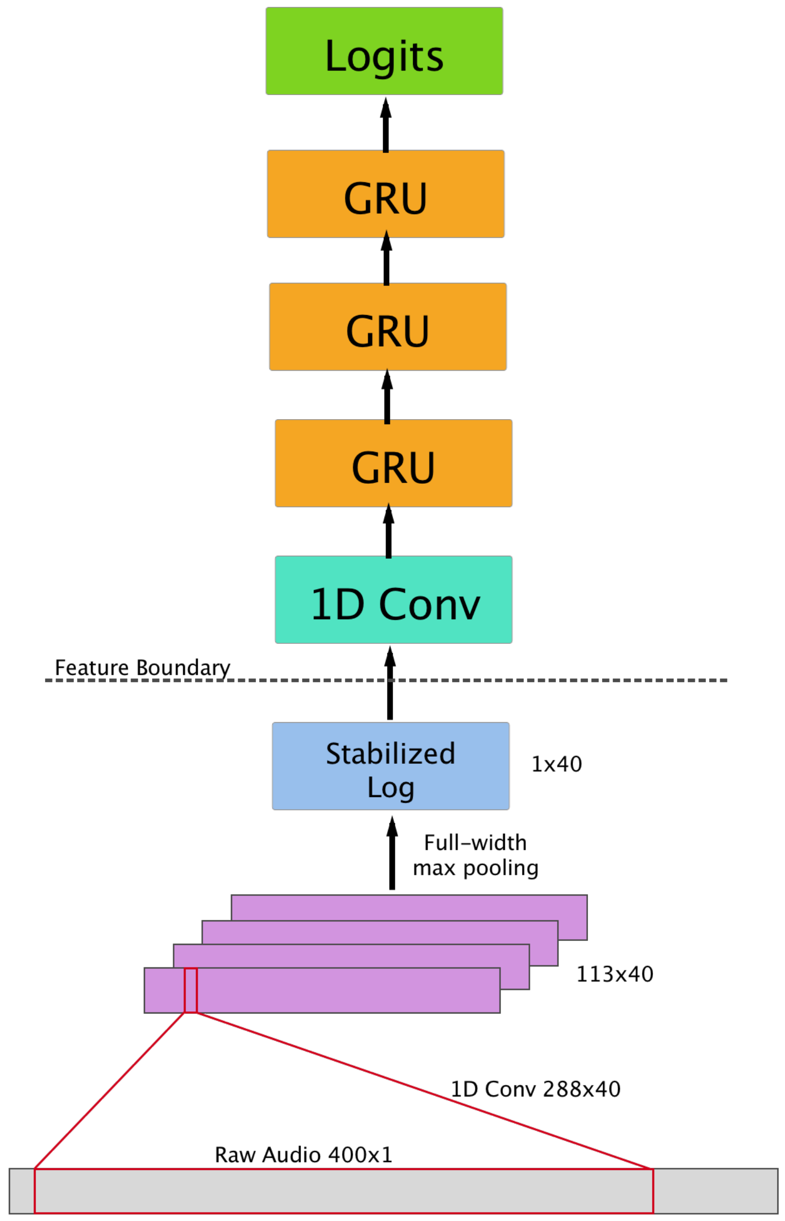


Figure 3-4: The CNN-GRU-RAW architecture. This network is identical to CNN-GRU (Figure 3-3), with the exception of an extension prepended to the beginning of the network that extracts 40-dimensional features directly from the raw waveform.

Chapter 4

Experimental Framework

In this chapter I describe a set of experiments using the dataset described in Section 2.1 and models outlined in Section 3.3 to achieve the following goals:

1. Explore the effect that model size has on VAD performance. Exploring small, medium, and large variants of the CNN-GRU architecture will provide insights into the performance-speed trade-off.

2. Analyze the effect of training the CNN-GRU model on clean-speech and non-speech sounds only. How would a model trained in these conditions generalize to much higher SNRs?

3. Analyze and compare the performance of the CNN-GRU and CNN-GRU-RAW models. Does raw-waveform feature learning improve performance? Is the raw waveform feature extraction too slow?

4. Analyze what the CNN-GRU-RAW model is learning. This involves an analysis of the learned features, and exploring how individual layers are activated when the model is inputted with various types of audio.

To compare the performance of models, both frame-level accuracy and receiver operating characteristic curves (ROCs) will be used. The frame-level accuracy represents the percentage of frames that are correctly classified as speech or non-speech. An ROC curve is a graphical representation of the trade-off between false-alarm rate and true-positive rate. A false alarm is when the model incorrectly classifies a non-speech frame as containing speech. A true-positive is when the model correctly classifies a speech frame as containing speech. Because statistical binary classifiers, including the VAD models presented here, output the probability of a particular class, a confidence threshold is needed to make a decision. For example, we may only classify a frame as containing speech if the model is over 70% confident. If this threshold is high, the false-alarm rate will likely be low, and equivalently the true-positive rate will be low; one benefit is traded off for the other.

To compare computational speed, the average per frame computation time will be benchmarked on a designated machine in a single-threaded computation environment. Benchmarks will not make use of a GPU, instead performing all computations on a 3.9 GHz CPU. While not a perfect means for comparing computational efficiency, it will be useful for rough, relative comparisons of speed.

4.1 Model Size

To explore the performance-speed trade-off of the CNN-GRU architecture, I train and evaluate 3 separate variants of the CNN-GRU model. The small model consists of only 2 GRU layers, while the medium and large variants have 3 and 4, respectively. In addition to varying the number of GRU layers, the number of units in each layer is also varied accordingly. Table 4.1 provides a breakdown of the number of layers, units, and trainable parameters for each size variant. These models are explored both in terms of performance and computational speed.

4.2 Single-Condition Training

For these experiments, I explore the ability of the CNN-GRU model to generalize to noisy conditions by training it only on the non-speech sounds and clean speech data. I analyze only the medium model variant described in Section 4.1. By comparing results to the multi-condition experiments, the helpfulness of the noisy data in training

Table 4.1: A breakdown of the number of units and parameters for each of the small, medium, and large variants of CNN-GRU.

CNN-GRU Size Variants			
	small	medium	large
1D Conv			
filters	32	48	64
parameters	1,312	1,968	2,624
BatchNorm			
dimensions	32	48	64
parameters	64	96	128
GRU			
layers	2	3	4
units per layer	24	32	40
total parameters	11,160	20,256	32,040
Logits			
in_dim	24	32	40
parameters	50	66	82
total parameters	12,586	22,386	34,874

a robust VAD model can be gauged. This will provide insight into whether or not clean speech and non-speech sounds are sufficient for the CNN-GRU model to learn characteristics that are unique to speech. Both in terms of frame-level accuracy and ROC curve, the model will be evaluated on data from each simulated SNR condition: -5dB, 0dB, 2dB, 4dB, 6dB, 8dB, 10dB, 15dB, 20dB, and 'clean.'

4.3 Raw Waveform Experiments

The CNN-GRU-RAW model explored in these experiments is equivalent to the medium variant of CNN-GRU described in Section 4.1, with the addition of the extension that extracts features directly from the raw waveform, as described in Section 3.3.2. The focus of these experiments is not only to compare the CNN-GRU and CNN-GRU-RAW models in terms of performance and computational speed, but also to analyze what the CNN-GRU-RAW network is learning. Deep neural networks are sometimes considered to be black-boxed, obscure, and monolithic due to the fact that their ability to learn is still not completely understood. Despite the highly obscure nature of neural networks, I explore how individual layers in the CNN-GRU-RAW network respond to different inputs, and look for clues that indicate what the network is learning.

4.4 Training

In this section I discuss specific details about how these models are trained. When training any RNN model, one of the most important considerations is the unrolling length. The unrolling length indicates the number of times the network is unrolled during training. When an RNN is unrolled, it can be thought of as a non-recurrent, deep neural network in which weights are shared across layers. The unrolling length specifies the limit on the number of time steps upon which a particular data sample may influence parameter gradients during training. In theory, every data sample going infinitely far back in time will influence how a network's trainable weights should be updated. In practice, however, accounting for inputs that go infinitely far back in time is not feasible. Thus, RNNs are trained with an algorithm called Truncated Back-propagation Through Time (TBPTT). TBPTT is analogous to unrolling an RNN for some finite number of time steps and then performing the standard Back-Propagation algorithm to efficiently compute parameter gradients. A benefit to TBPTT is that it makes the training of RNNs feasible and fast depending on the unrolling length, n. A drawback is that the network won't be able to model temporal dependencies that are greater in length than the unrolling length[1]. For these experiments, I constrain the unrolling length to 20 frames[2]. 20 frames with a 10ms frame hop will capture temporal dependencies going up to 200ms. 200ms is a lower bound for the length of most spoken syllables. In addition to being large enough to almost capture full

[1] This does not suggest that only the last n samples are considered when an RNN makes a decision. In fact, the model is trained to make decisions based on the current hidden state, which assimilates information from timestamps going further back. In essence, the presence of a particular feature as indicated by the hidden state may aid the model in making decisions, but there is no explicit dependence between samples that are farther away than the unrolling length being learned.

[2] This decision was partially inspired by (Zazo et al. 2016).

syllables, an unrolling length of 20 allows the network to train in a reasonable amount of time.

Another important consideration to training any neural model is regularization. Regularization is the process of introducing bias into a model such that it does not overfit to idiosyncrasies in the training data. Its desirable to have a statistical model learn the “big picture” rather than to assume every detail of the training set is indicative of reality. One regularization method, known as dropout, is the process of randomly null-ing out connections in the network during training (Hinton et al. 2012). Dropout has the effect of mitigating co-dependencies which may be highly idiosyncratic and unique only to the training data. I employ dropout layers leading into each GRU cell. I found a dropout rate of 0.5 to be sufficient for each architecture. I also explored weight decay, but was unable to find a non-zero coefficient that didn't deteriorate performance on unseen data.

When deciding upon these various architecture parameters, the performance on the dev partition was used to benchmark what worked and what did not.

All data getting inputted to the network is normalized to have zero mean and unit variance. This normalization is done on the batch level. In addition, all labels are offset so that the network can “see into the future” before making a decision. While not literally seeing into the future, the offset has the effect of letting the network gather more evidence before making a decision. Models trained with a label offset exhibited improved performance over ones that did not. For both CNN-GRU and CNN-GRU-RAW, I use a label offset of 80ms.

The Adam optimization algorithm was used for making gradient-based parameter updates during training (Kingma and Ba 2014). For all experiments, $\alpha = 0.0001$ was used and otherwise the default parameters suggested in (Kingma and Ba 2014). Cross-entropy loss was used for the objective function.

A batch size of 64 was chosen for all experiments. With this value, training was relatively stable, but not too slow.

4.4.1 DataManager

In this section, I describe the DataManager that I use for batch allocation during training and evaluation. The VADBatchManager is a class I implemented in Python designed to manage batch allocation of sequential data, including any meta information that's pertinent to training. Meta-information may include the id of an utterance or its frame-length. The VADBatchManager may be configured with any data path, unrolling length, label offset, group size, group hop[3]. It's used during the training of all CNN-GRU and CNN-GRU-RAW variants.

When initialized, the VADBatchManager accepts a `data_train` path and `data_test` path as input. Each of these arguments represent a path to a `.txt` file containing lines of utterance information. In these files, there is a line for each utterance, with the following format:

`<utt_id> <num_frames> <data_path>`

Here, `<data_path>` represents the path to a numpy[4] pickle containing both data and frame-level labels. In the case of CNN-GRU, the data is 40-dimensional log Mel filter bank features. In the case of CNN-GRU-RAW, the data is 16kHz raw waveform samples[5]. All utterances are zero-padded to the same length so that they can be appropriately stacked into a batch matrix. The `<num_frames>` parameter provides length information which is important to provide to the network so that it knows not to to train upon zero-padded frames.

One of the challenges of batch allocation to RNNs is elegantly handling the case when the utterance lengths are significantly larger than the unrolling length. During each training iteration, the network sees only a "slice" of the batch equal in length to the unrolling length. The network's recurrent state must be stored and re-introduced between each training iteration. Only after the entire batch of utterances (i.e. all

[3]Group size and group hop are related to the fact that the models are designed to accept as input slices containing multiple frames at a time, such that successive slices exhibit overlap.

[4]http://www.numpy.org/

[5]The raw waveform data does not have a one-to-one frame-level mapping to the labels like the log Mel filter banks do. However, the CNN-GRU-RAW model is designed to extract features with the same dimensions as log Mel filter banks.

slices) is trained may the hidden state be reset to the zero-vector[6]. There is an additional technicality which is that each utterance within the batch is of different length. Because of this, shorter utterances will "run out" of data faster than longer ones. For the batch rows dedicated to these shorter utterances, the slice will consist of zero-padding after the utterance data is surpassed. It's important that the network does not train on this zero-padding.

At the beginning of each epoch, the VADBatchManager shuffles the pool of available training utterances and arranges them into a stack. For each batch, the appropriate number of utterances (i.e. the batch size) is popped off the stack and the corresponding data is loaded into memory. The VADBatchManager then iteratively pops off slices of this batch in chronological order. The VADBatchManager is careful to maintain the time signature of each slice, so that the appropriate lengths may be computed. For example, if a slice corresponds to frames 580-599 and the utterance at index 7 has a frame-length of 591, then the length for that index of this slice is 11. These lengths would be computed for each utterance in the slice, and passed to the network along with the data and labels. The network would then know not to train the zero padding that exists beyond frame 11 for that index in the slice. After all slices for a batch are trained, the network's hidden state gets reset, a new batch popped off the stack, and the process repeats. After an entire epoch, the entire training pool gets reshuffled, and then training continues.

[6]I experimented with both zero-vector initialization and initialization with random Gaussian noise. I did not notice a difference in performance between the two.

Chapter 5

Experimental Results and Analysis

After training each model for 15 epochs, I cherry-picked training checkpoints that exhibited the lowest loss on the dev partition. For the small, medium, and large variants of the CNN-GRU, I chose checkpoints from epoch 14, 13, and 14, respectively. For the single-condition CNN-GRU, convergence occurred much earlier, so I chose a checkpoint after just 2 epochs of training. For the CNN-GRU-RAW, I chose a checkpoint at epoch 14.

As can be seen in Table 5.1, the CNN-GRU exhibited diminished performance returns as model size increased. While larger models yielded better performance, all three variants performed quite similarly. While the large variant performed the best, the medium variant had a slight edge in the 0dB SNR category. In addition, the CNN-GRU-RAW model was best at classifying non-speech sounds, with a near perfect 99.23% frame-level accuracy in that category. Figure 5-1 shows the plots for probability of speech across time, for both the medium CNN-GRU and CNN-GRU-RAW.

The single-condition model, despite being trained only on clean speech and non-speech sounds, performed well for noisy speech with 15dB SNR and higher. Below 10dB SNR, performance deteriorated at a rate of roughly 5% for every 2dB reduction in SNR. The CNN-GRU-RAW model performed very similarly to the medium variant of the CNN-GRU trained on all noise conditions.

A full breakdown of the results of all 5 experiments is in Table 5.1. ROC curves

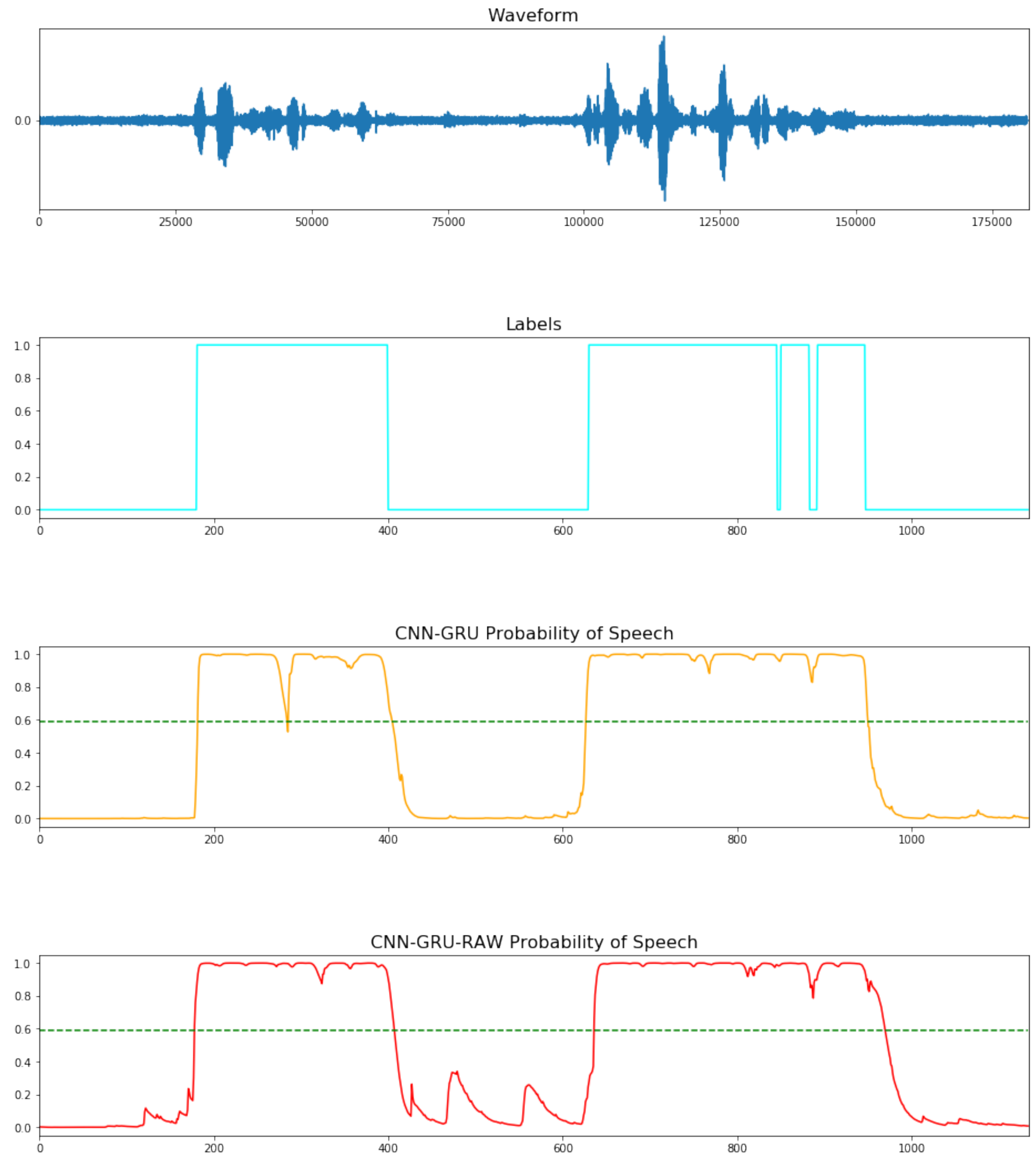


Figure 5-1: Graphs of the probability of speech over time outputted by the medium CNN-GRU and CNN-GRU-RAW models. Above these graphs are the waveform and truth labels.

Table 5.1: VAD frame-level accuracy results for various experiments, reported as percentages. The accuracies listed here are on the test partition, while the thresholds were chosen to optimize accuracy on the dev partition. For any given category, the highest accuracy is shown in bold.

VAD Frame-level Accuracy					
	small	medium	large	raw	single-cond
threshold	37.00	53.00	50.00	61.00	1.00
-5dB SNR	89.43	90.86	**91.44**	89.34	56.89
0dB SNR	93.45	**94.73**	94.62	93.99	64.71
2dB SNR	94.63	95.23	**95.83**	95.16	70.78
4dB SNR	95.72	96.02	**96.51**	96.08	75.37
6dB SNR	96.37	96.64	**96.93**	96.50	80.74
8dB SNR	96.80	96.94	**97.28**	97.00	85.25
10dB SNR	97.18	97.25	**97.72**	97.02	89.49
15dB SNR	97.13	97.28	**97.93**	97.21	94.33
20dB SNR	97.71	97.61	**98.13**	97.18	96.10
clean	97.46	97.50	**98.04**	97.56	96.66
sounds	98.36	98.15	98.11	**99.23**	98.54
total	95.74	96.12	**96.52**	95.92	82.06

for each experiment can be seen in Figures 5-2, 5-3, 5-4, 5-5, and 5-6. In addition to comparing performance across SNRs, these figures also include ROC curves that compare the performance on audio exhibiting reverberation to audio without reverberation. Figure 5-7 shows ROC curves that directly compare the performance of the small, medium, large, and raw model variants.

Table 5.2 lists the per-frame computation times for each of the small, medium, and large variants of CNN-GRU, as well as the CNN-GRU-RAW. The per-frame computation time for log Mel filter bank features is also included. These metrics were recorded on a 3.9GHz CPU machine in a completely single-threaded environment. The large CNN-GRU variant is about as fast as the CNN-GRU-RAW when accounting for log Mel filter bank feature computation. Accounting for the fact that the medium CNN-GRU and CNN-GRU-RAW represent the same network above the feature boundary (depicted in Figure 3-4), the per-frame computation time for raw waveform feature extraction can be inferred by subtracting the CNN-GRU-RAW computation time

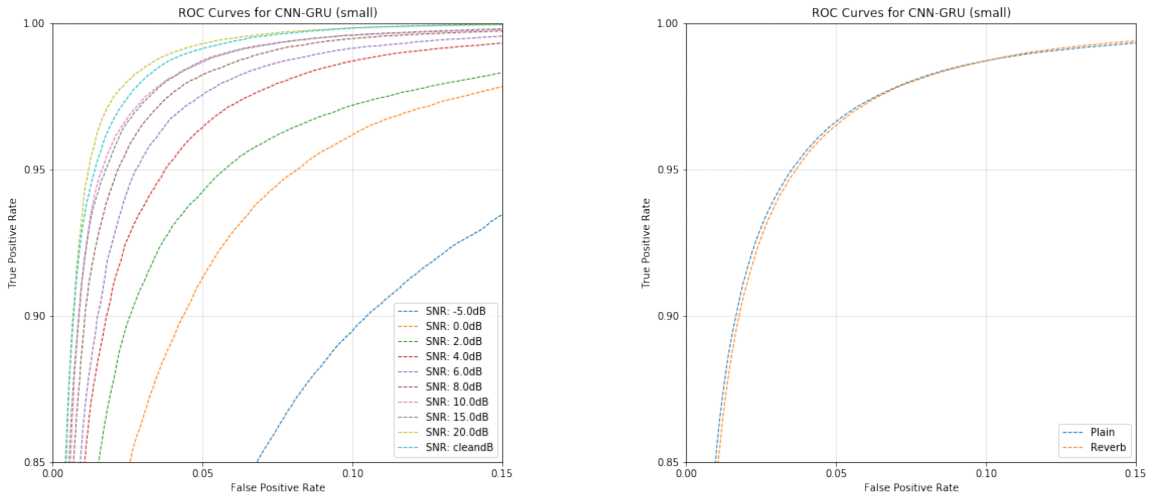


Figure 5-2: ROC curves on test data for the small CNN-GRU variant. On the left is a breakdown of performance by each SNR category, and on the right is a comparison of performance on utterances that contain reverberation and ones that do not.

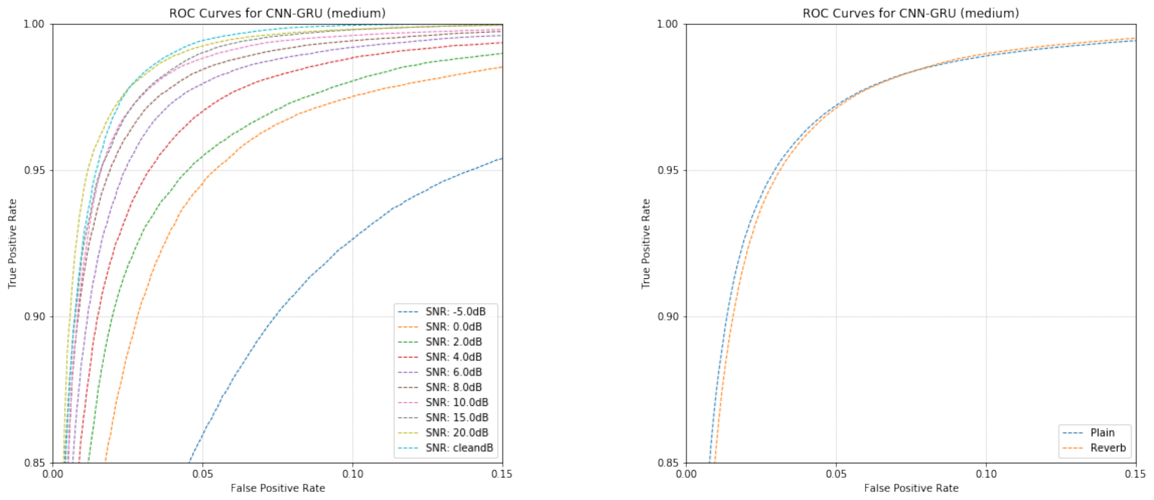


Figure 5-3: ROC curves on test data for the medium CNN-GRU variant. On the left is a breakdown of performance by each SNR category, and on the right is a comparison of performance on utterances that contain reverberation and ones that do not.

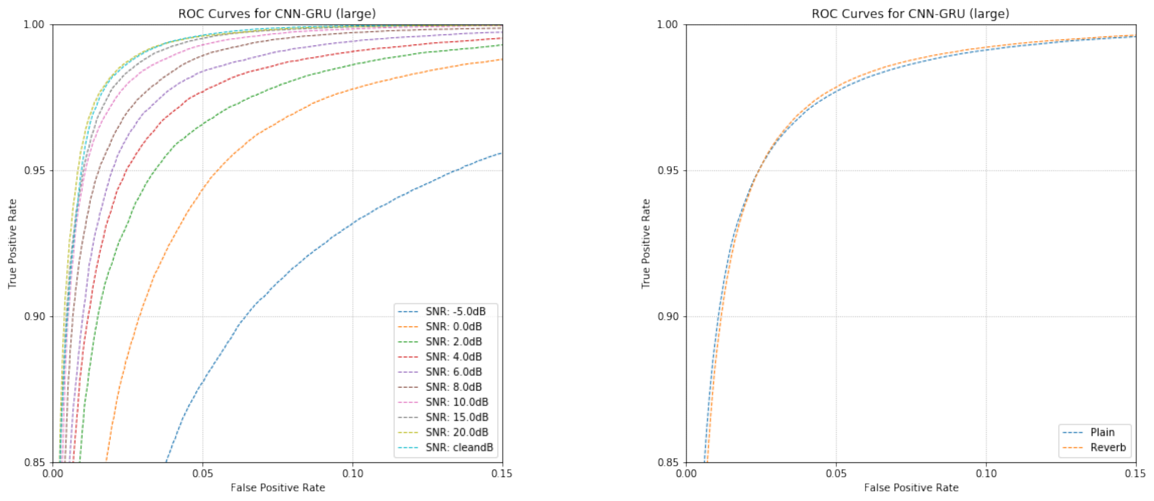


Figure 5-4: ROC curves on test data for the large CNN-GRU variant. On the left is a breakdown of performance by each SNR category, and on the right is a comparison of performance on utterances that contain reverberation and ones that do not.

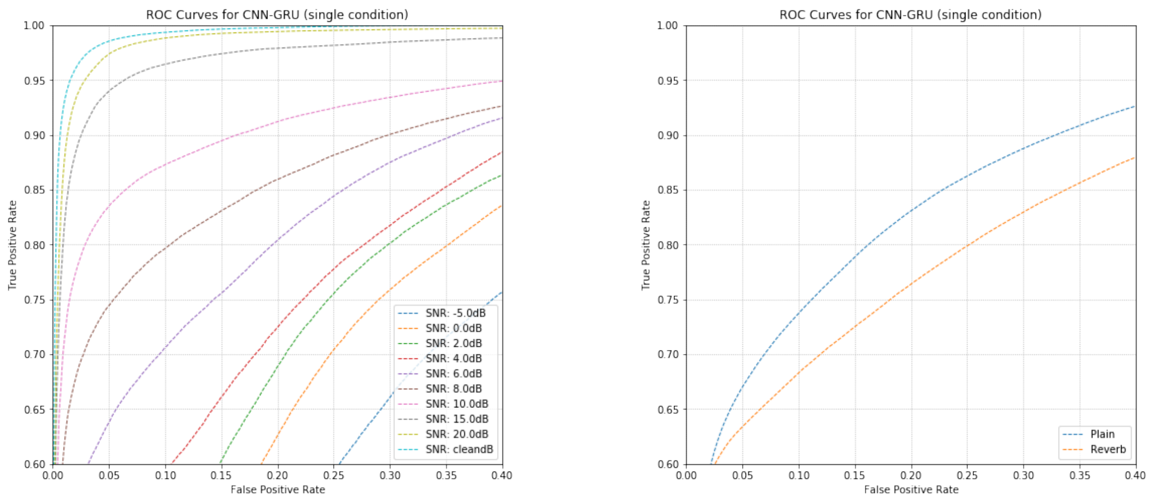


Figure 5-5: ROC curves on test data for the single-condition CNN-GRU variant. On the left is a breakdown of performance by each SNR category, and on the right is a comparison of performance on utterances that contain reverberation and ones that do not.

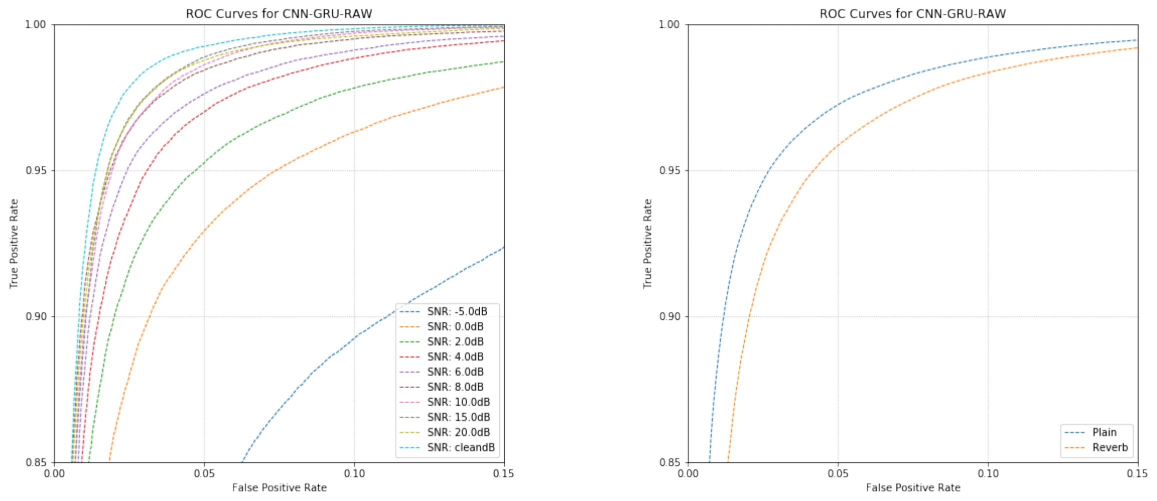


Figure 5-6: ROC Curves on test data for the CNN-GRU-RAW. On the left is a breakdown of performance by each SNR category, and on the right is a comparison of performance on utterances that contain reverberation and ones that do not.

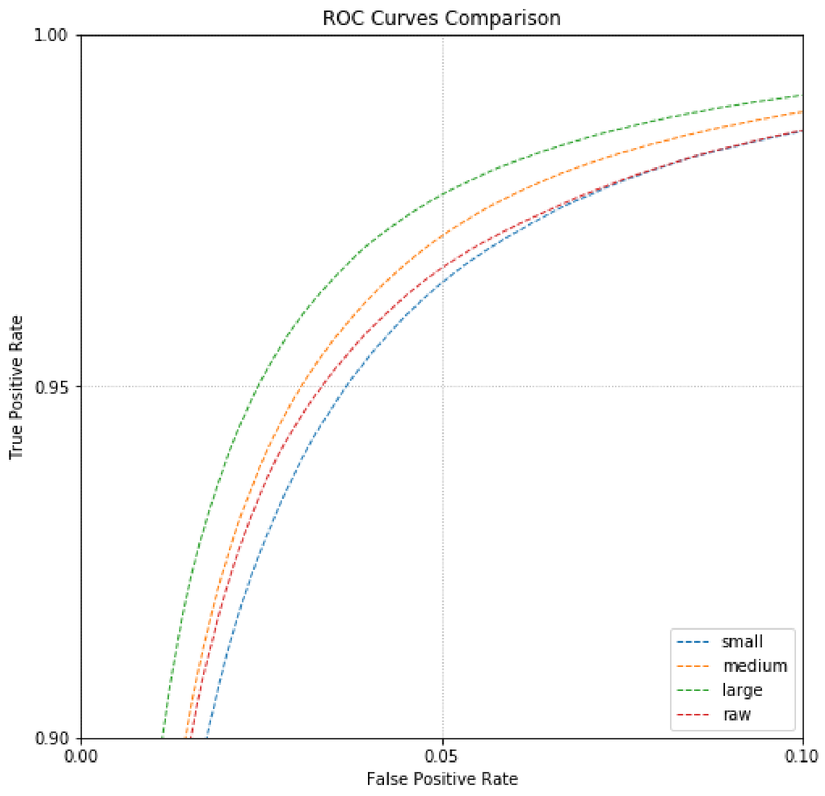


Figure 5-7: ROC curves on test data that compare the relative performance of the small, medium, large variants of the CNN-GRU, as well as the CNN-GRU-RAW.

Table 5.2: Per-frame computation times for each of the different size variants of CNN-GRU, as well as the CNN-GRU-RAW. For reference, the per-frame computation time of log Mel filter bank features is also included. These times were recorded on a 3.9GHz CPU machine in a completely single-threaded environment.

Per-frame Computation Time					
	small	medium	large	raw	log Mels
time	0.396ms	0.545ms	0.700ms	0.905ms	0.186ms
real-time factor	25.2	18.3	14.3	11.0	53.7

from the medium CNN-GRU computation time. In doing so, it's determined that the per-frame raw waveform feature extraction computation time is 0.360ms, roughly double the time it takes to compute a frame of log Mel filter bank features.

5.1 CNN-GRU-RAW Analysis

The extension that differentiates the CNN-GRU from the CNN-GRU-RAW is designed to extract spectral features directly from raw waveform data. These extracted features are designed to have the same dimensions as 40-dimensional log Mel filter bank features. We can observe what these learned features look like by passing audio through a trained CNN-GRU-RAW, and logging the activations that are outputted at the feature boundary depicted in Figure 3-4. It's important to note that while these features are learned and optimized for maximum discriminatory performance on the VAD task, to us humans, their ability to facilitate the discerning of speech versus non-speech may not be immediately obvious. The lack of structure is probably due to the fact that the convolutional layer immediately beyond the feature boundary convolves across the entire frequency dimension[1]. Because of this, it's unlikely that shift-invariant structure is learned due to the fact that the network has no incentive to enforce such structure. Figures 5-8, 5-9, and 5-10 show the learned features for a

[1]A variant of CNN-GRU-RAW not explored in this paper convolves shorter windows across the frequency dimension, such that shift-invariant structure may be learned. This variant required many more parameters to work well, and was not explored more thoroughly due to the emphasis being placed on computational efficiency.

couple of noisy utterances and also a non-speech sound, alongside their corresponding Mel filter bank features.

5.1.1 Spectral Analysis

Despite the lack of structure in these learned features, we can gain a better understanding of what the network is learning by observing the filters (i.e. the impulse responses) that get convolved across the raw waveform. As mentioned in Section 3.3.2, the maximum values of the vectors resulting from the convolution of these filters are high with the spectrum of the filter has high overlap with the spectrum of the raw audio.

Figure 5-11 shows a few learned filters, along with their spectra. We see that these filters have been engineered by the CNN-GRU-WAV to discover specific frequencies in the audio. These filters are in fact capable of discovering multiple frequencies, sometimes in non-neighboring regions, as we see in the top-most filter in Figure 5-11 around the 200Hz and 2500Hz regions.

Despite the filters learning multiple frequencies in disjoint regions, if we took the most prominent frequency in each filter and then sorted them, we could get an idea of distribution of frequencies that the CNN-GRU-RAW is focusing on the most—this was similarly done in (Zazo et al. 2016). This analysis revealed a tendency for filters to respond to high frequency audio near 8kHz, as can be seen on the left in Figure 5-12. A potential explanation for the bias towards high-frequency audio is the fact that TIMIT was artificially and intentionally aliased when originally being down-sampled from 20kHz to 16kHz, resulting in unusually high energy density in the frequency bands at and near 8kHz (Fisher et al. 1986). The CNN-GRU-RAW may have overfit to this idiosyncrasy during training. In sorting the most prominent frequencies, if we only select from the 0-7920Hz range, we get the plot on the right in Figure 5-12. Remarkably, this graph lines up very similarly with the Mel scale! Perhaps this is not surprising given that the Mel scale is based empirically on the human perception of frequencies, and VAD is the task of discerning human speech. Despite these coinciding factors, it's amazing that the CNN-GRU-RAW effectively

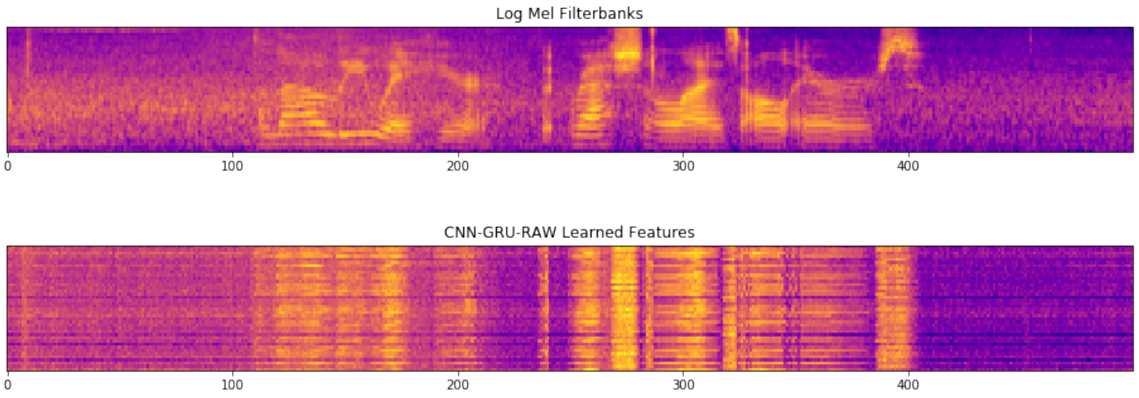


Figure 5-8: Learned features alongside log Mel filter bank features for the the utterance, “Allow leeway here but rationalize all errors,” at 4dB SNR.

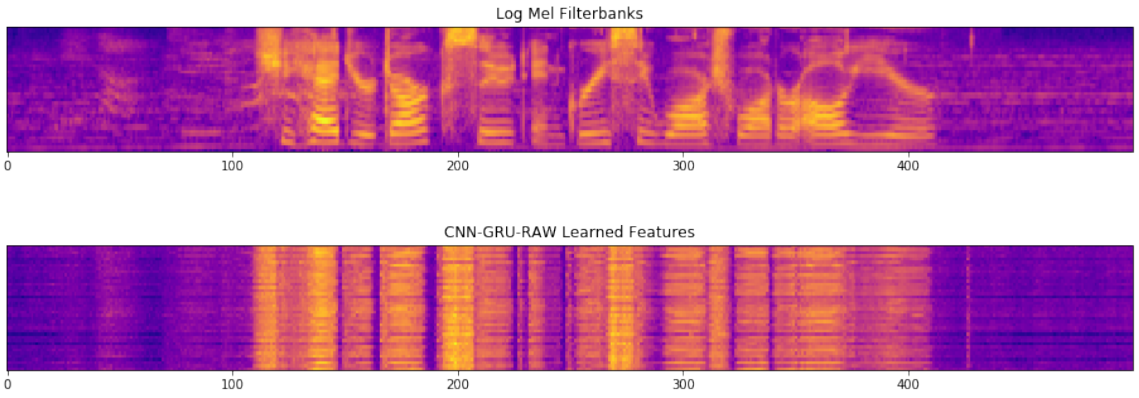


Figure 5-9: Learned features alongside log Mel filter bank features for the the utterance, “She had your dark suit in greasy wash water all year,” at 20dB SNR.

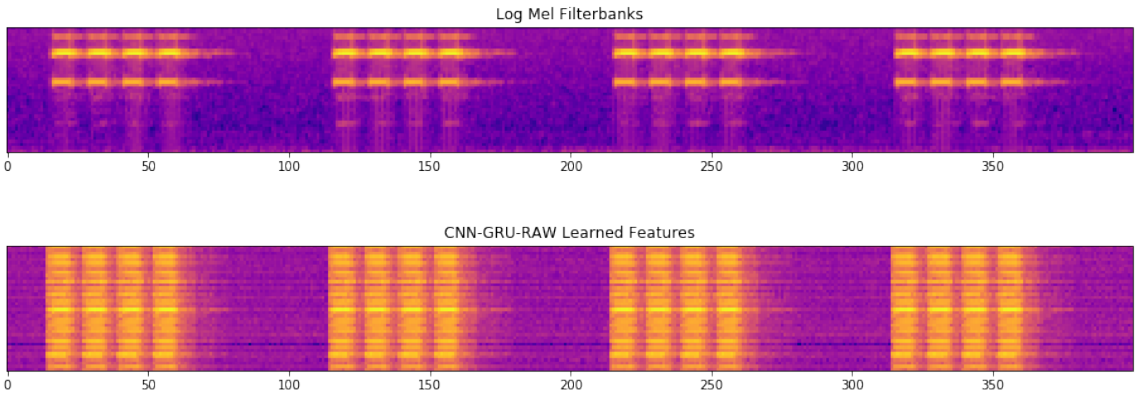


Figure 5-10: Learned features alongside log Mel filter bank features for an alarm clock sound.

reverse-engineered this nonlinear scale of pitches. Even in selectively ignoring the upper 80Hz, the curve does not perfectly mimic the Mel scale. Subtle differences may indicate that its favorable for the model to have more discriminatory power in certain frequency bands when compared to the Mel scale. This fact, while confirming the inherent power of the Mel scale, also supports the efficacy of raw waveform feature learning.

5.1.2 Chirp Analysis

Another method to gain insight into the CNN-GRU-RAW is to evaluate the model on a segment of audio called a chirp. A chirp is a short sound characterized by a steadily rising pitch. Chirps can be synthesized artificially, for example by the following formula:

$$x_i = \sin\left(\frac{\pi i^2}{2\ell}\right) \tag{5.1}$$

where ℓ is the length of the chirp, in samples. Figure 5-13 show the log Mel filter bank features of a 3-second chirp generated from this formula. Because the frequency is increasing linearly, the depiction in Figure 5-13 is indicative of the Mel scale.

When we pass the chirp into a trained CNN-GRU-RAW, it produces the learned feature representation in Figure 5-14. As we might expect, these features do not exhibit much structure. If we sort the frequency rows by order of increasing prominent frequency as explored Section 5.1.1, some structure is observable. In fact, you can faintly pick out the non-linear, increasing curved line, especially in the lower frequencies, as displayed in Figure 5-15.

While each filter is seeking out a prominent frequency, they are also—to some degree—responding to all other frequencies. It's likely that these filters are assuming many different responsibilities with respect to the frequencies that activate them, creating a dynamic interplay that only the CNN-GRU-RAW is capable of deciphering fully.

We can further understand which roles these filters are assuming by observing the

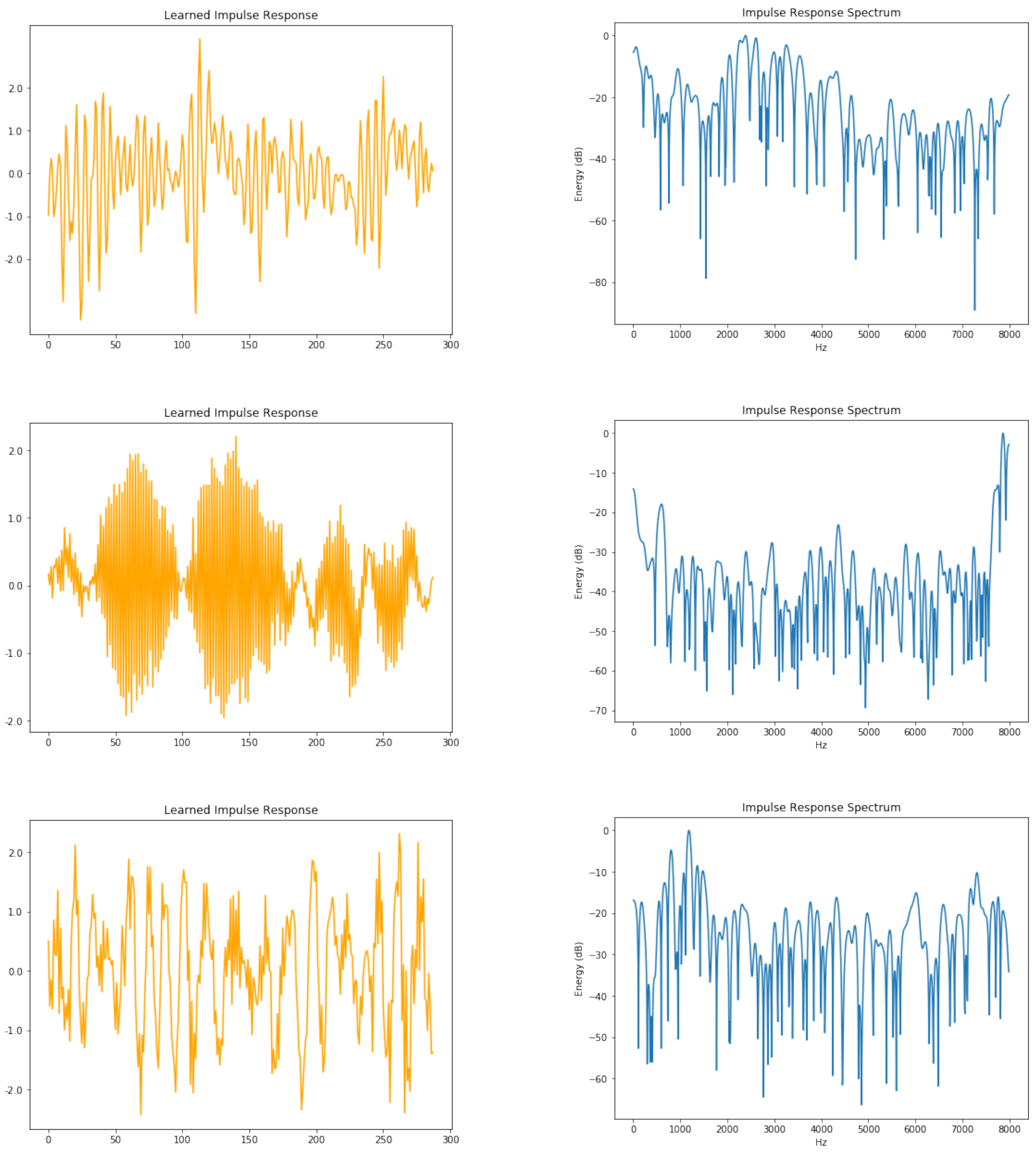
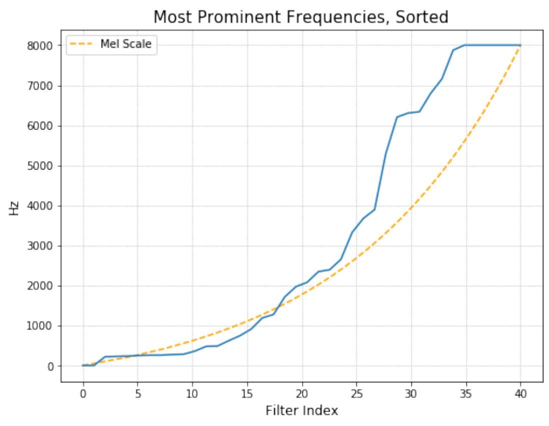


Figure 5-11: Learned impulse responses from the CNN-GRU-RAW feature extraction process. These impulse responses represent the convolutional filters that get convolved across the raw waveform. Next to each impulse response is its spectrum.



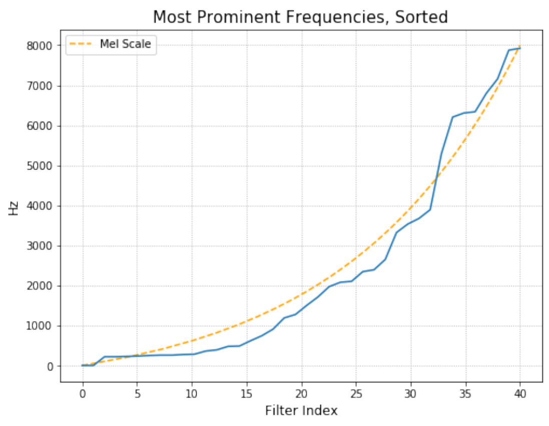


Figure 5-12: Plots of most prominent frequencies for each raw waveform filter of the trained CNN-GRU-RAW, sorted. These plots are super-imposed with the O'Shaugnessy Mel scale for comparison (O'Shaughnessy 1987). On the right, the most prominent frequencies were taken from between 0-7920Hz, selectively ignoring the CNN-GRU-RAW's tendency to respond to high frequency audio.

convolution vectors produced when they are convolved across the raw waveform of the chirp, before max pooling. Figures 5-16 and 5-17 show the filter convolution for the middle and bottom filters depicted in Figure 5-11, alongside a graph that tracks the maximum value for each frame. We see that for Figure 5-17, there are multiple peaks at disjoint locations in the chirp. This analysis relates to the discovery of multiple disjoint peaks in the spectrum of the learned impulse responses, displayed in Figure 5-11. The filter convolution in Figure 5-16, however, shows that this filter is highly specialized for high frequency audio.

5.2 Summary

In this chapter, I analyzed the relative performances of variants of the CNN-GRU. This included 3 size variants: small medium and large, as well as a variant trained only on clean-speech and non-speech sounds. I found that as model size increased, there were diminished performance returns. I also trained a CNN-GRU-RAW model, designed to learn and extract features directly from raw waveform data. While the CNN-GRU models performed better and were more computationally efficient, the success of the CNN-GRU-RAW provided some interesting insights into raw waveform

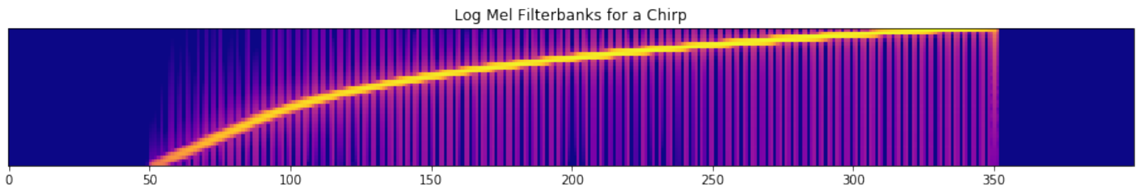


Figure 5-13: Log Mel filter bank features for a 3 second chirp produced by Equation 5.1.

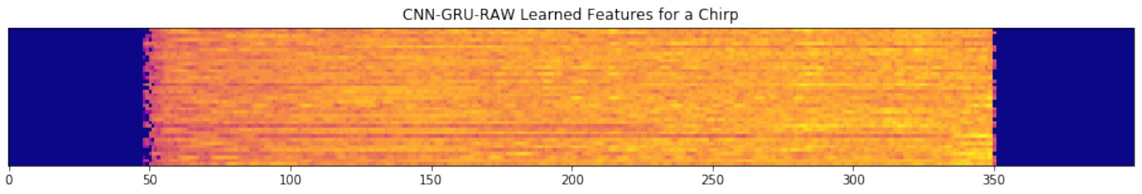


Figure 5-14: Learned features from the CNN-GRU-RAW for a 3 second chirp produced by Equation 5.1.

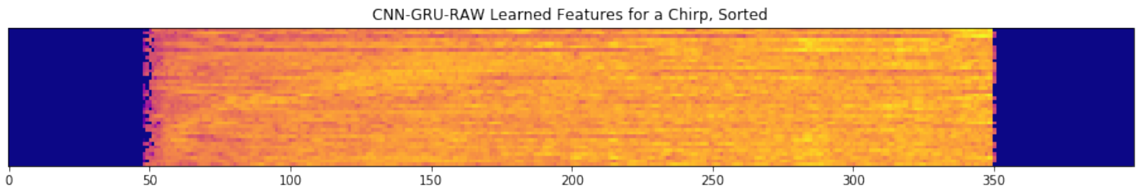


Figure 5-15: Learned features from the CNN-GRU-RAW for a 3 second chirp produced by Equation 5.1, where the rows are sorted according to their filter's most prominent frequency, as displayed in Figure 5-12. In the lower frequencies, you can make out a faint curve indicative of the log Mel filter banks in Figure 5-13.

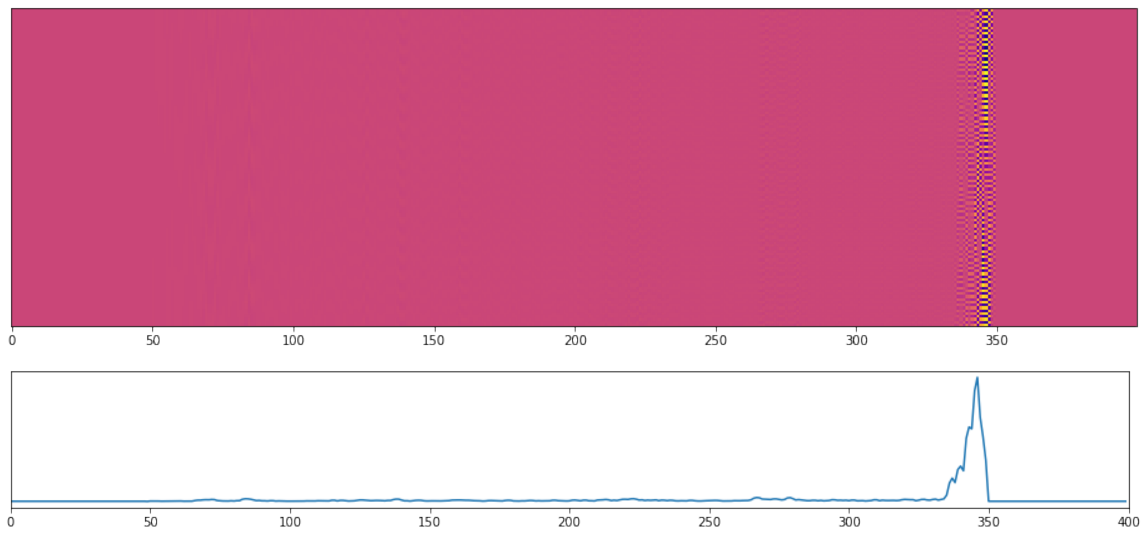


Figure 5-16: The convolution vectors produced by convolving the filter depicted in the middle row of Figure 5-11 across a 3 second chirp produced from Equation 5.1. The graph tracks the max value of each frame across time, to simulate max-pooling across the vertical dimension.

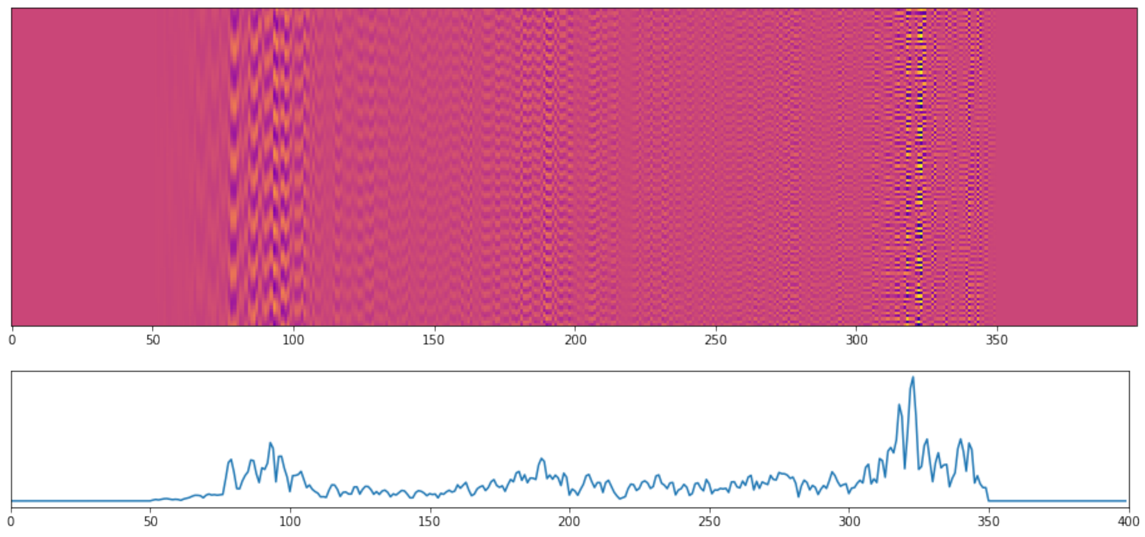


Figure 5-17: The convolution vectors produced by convolving the filter depicted in the bottom row of Figure 5-11 across a 3 second chirp produced from Equation 5.1. The graph tracks the max value of each frame across time, to simulate max-pooling across the vertical dimension.

feature learning, including the inherent power of the Mel scale for discerning human speech.

Chapter 6

Real-Time VAD Deployment

Until now, the models presented in this paper haven't been discussed from the perspective of real-time deployment. Real-time deployment is a fundamentally different use case than evaluating on predesignated audio recordings. In the real-time deployment scenario, there is no prior knowledge of the acoustic environment, and no constraints on how it may change. In addition, the specifications of the hardware upon which the model is deployed may be unknown. Details including microphone idiosyncrasies, network bandwidth, and computational power come into play. Most importantly, the underlying hardware needs to be capable of continuously running the model at a real-time rate.

On the bright side, the real-time deployment scenario may present some relaxed constraints on accuracy. Depending on the use case, it may be desirable and affordable to be liberal in classifying audio segments as containing speech. For example, a particular application that depends on VAD might require an extremely high true positive rate, but also may be tolerant of false alarms.

The performance of a deployed model cannot as easily be quantified as it can when evaluating on predesignated recordings. While computational metrics like decision latency may serve as an empirical indicator of performance, other metrics like frame-level accuracy, are less informative. As mentioned previously, accuracy may not reflect how well an application functions. Instead, performance metrics may be highly application-specific. In addition, accuracy metrics do not carry as much weight for a model that is deployed in a completely unseen and non-stationary acoustic environment. Unless model accuracy is evaluated on a set of recordings that perfectly model the acoustic environment, accuracy only provides a rough guess of how a model may perform. This is a point that I discuss in Section 6.4.

In this chapter, I present an architecture for an FST that can be layered on top of the CNN-GRU or CNN-GRU-RAW models, effectively serving as an audio gating mechanism for a downstream speech processing application. Then, I provide background on the SLS Streamer framework, and describe how it may be used to implement a real-time VAD. I then introduce a computationally-constrained Android Iot Smart Speaker, and how VAD can be applied to it. Then I provide a brief analysis, highlighting some of the difficulties associated with the deployment of neural models.

6.1 VAD FST

While the CNN-GRU and CNN-GRU-RAW models perform well at classifying individual frames as containing speech or non-speech, these decisions possess relatively high granularity. This granularity may not always be favorable. During natural speech, many pauses may occur when speakers are attempting to add clarity, emphasis, or merely trying to think of the right word. These pauses, while not containing speech, may provide contextual information to speech processing applications that aid in understanding. An ASR system, for example, may use natural speech pauses to hypothesize the beginning of a new sentence.

If a VAD model is serving as an audio gating function for a downstream ASR application, it may be detrimental to transmit highly granular segments of audio containing speech. While the VAD could potentially chunk an audio segment into individual words, this would render the language model component of the ASR system useless. Instead, it may be desirable to send contiguous segments of audio which include intermittent pauses, so long as entire utterances are captured.

For CNN-GRU and CNN-GRU-RAW, an additional layer of decision making is needed to achieve this behavior.

To facilitate the capturing of whole utterances, including brief pauses, I designed a finite state transducer (FST), which can be layered on top of the CNN-GRU or CNN-GRU-RAW. This FST, dubbed VADFST, takes as input the mean probability of speech across 200ms segments of audio, as outputted by CNN-GRU or CNN-GRU-RAW. VADFST consists of the following states and transition behavior:

- Steady State (S_0): This is the initial state of VADFST, representing a lack of speech activity. If the mean probability for a 200ms segment of audio exceeds some threshold, the VADFST immediately enters the Active State, S_1. Upon transitioning to this state, the last 1s of audio is immediately sent to the downstream application. The last 1s of audio is sent so that we can be sure the boundary encapsulates the beginning of the spoken utterance. While this is prone to capturing non-speech, it's assumed that these false alarms can be tolerated by the downstream application.

- Active State (S_1): This state is represented by the immediate presence of speech activity. So long as the mean probability for 200ms audio segments exceeds the threshold, VADFST remains in this state. While in S_1, every incoming 200ms segment of audio gets transmitted to the downstream application. If the mean probability drops below the threshold, the VADFST transitions to the Cool Down State, S_2.

- Cool Down State (S_2): Upon entering this state, the VADFST will continue to transmit the next 1s of audio to the downstream application. If during this 1s interval the mean probability exceeds the threshold, the VADFST immediately transitions back to the Active State, S_1. Otherwise, after the 1s cool down, the VADFST transitions to the Steady State, S_0. Upon moving back to the Steady State, an end-of-sentence (EOS) message is transmitted to the downstream application, indicating the end of an utterance. The purpose of the 1s cool down is to tolerate intermittent speech pauses, as well as to conservatively capture the endpoint of an utterance.

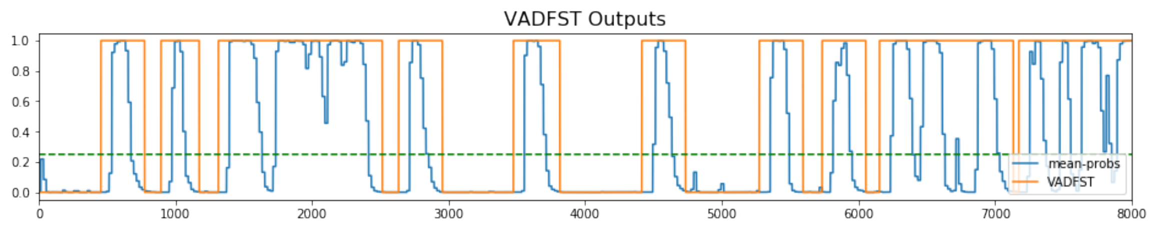


Figure 6-1: Outputs from the VADFST, superimposed with mean probability scores outputted by the CNN-GRU across 200ms segments. The VADFST captures entire segments of speech, including short pauses. The green dotted line represents the mean probability threshold.

A schematic of VADFST is in Figure 6-2. A visualization of the interplay between CNN-GRU and VADFST can be seen in Figure 6-1, which demonstrates the ability of VADFST to capture entire utterances while tolerating intermittent pauses.

6.2 SLS Streamer

The SLSStreamer framework[1], designed by fellow MEng candidate Kenneth Leidal, is a Python framework that allows for the free-form creation of component graphs for handling asynchronous binary data streams. Connections between components indicate a path upon which data may flow. Each component computes transformations of incoming payloads, and may store intermediate data. The SLSStreamer allows for recursive substructure, such that an individual component may in fact be a wrapper for many interconnected subcomponents. SLSStreamer provides many pre-built components, including ones that allow for the inflow and outflow of data via Websockets, as well as a component that wraps a kaldi-gstreamer ASR server[2].

The SLSStreamer framework can be used to implement a real-time VAD-ASR application. Because the framework is written in Python, it's relatively simple to write components that wrap powerful neural models built using frameworks like PyTorch or Tensorflow. Using the SLSStreamer framework, I implemented the following linear component graph to perform real-time VAD on an audio stream. The architecture

[1]http://groups.csail.mit.edu/sls/slstreamer/
[2]https://github.com/alumae/kaldi-gstreamer-server

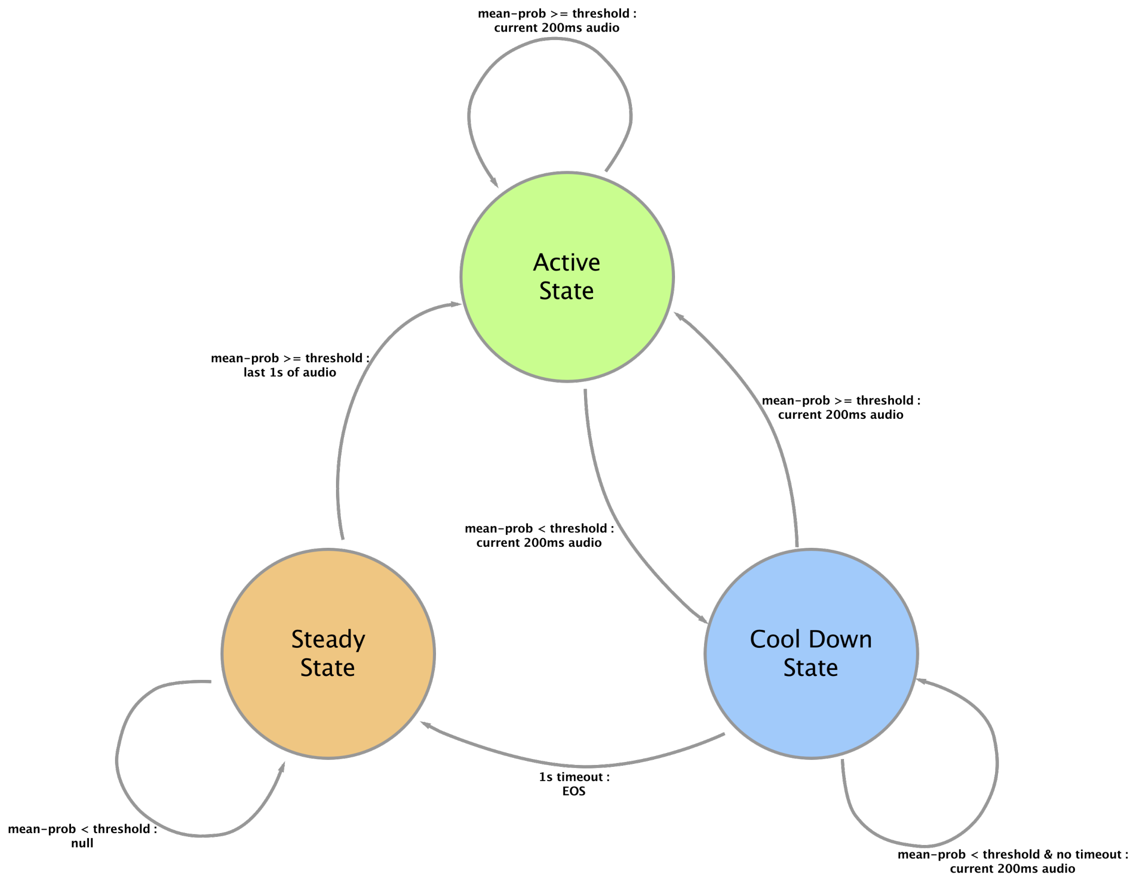

mean-prob >= threshold : current 200ms audio
Active State
mean-prob >= threshold : last 1s of audio
mean-prob >= threshold : current 200ms audio
mean-prob < threshold : current 200ms audio
Steady State
Cool Down State
mean-prob < threshold : null
1s timeout : EOS
mean-prob < threshold & no timeout : current 200ms audio

Figure 6-2: A schematic of the VADFST. Each transition is labeled with the trigger and output separated by a colon, **:**.

consists of the following components:

- Downsample: Downsamples incoming 44.1kHz 16-bit PCM audio to 16kHz. This component may be configured to accept any incoming sampling rate. It's not needed if the incoming audio is already 16kHz.
- Chunkify: This component serves as a queue for incoming audio buffers. As this component appends incoming buffers to a master queue, it outputs buffers with a specified width by popping samples off of this master queue in LIFO[3] order. This component also supports backtracking so that consecutive outputted buffers possess overlap. Backtracking is used so that any downstream feature computation that involves the framing of audio with overlap (i.e. log Mel filter bank features), can be done correctly in a streamlined fashion.
- VAD Gate: This component serves as a gating mechanism for incoming audio, and consists of the following subcomponents
  - Pre-emphasis: This component applies pre-emphasis to incoming audio buffers in a streamlined fashion.
  - Log Mels (optional): This component computes log Mel filter bank features on incoming audio buffers. This component is optional because the downstream VAD may accept raw audio as input (i.e. CNN-GRU) instead of log Mel filter bank features.
  - VAD: This component encapsulates a trained CNN-GRU or CNN-GRU-RAW model, implemented in Tensorflow. This component accepts 200ms buffers of either raw audio or log Mel filter banks, and outputs the mean frame-level probability of speech over this segment.
  - VADFST: This component encapsulates the FST architecture described in Section 6.1. It takes the mean probability scores as input and outputs raw audio and EOS messages.

[3]Last in, first out.

- ASR: This component takes raw audio and EOS tags as input and outputs hypothesized textual transcriptions. These hypotheses are generated in real-time and may change dynamically as the component receives more audio.

This VAD-ASR architecture can be wrapped in a websocket component which allows raw audio to be input from an external source over the Internet. This websocket component can send the textual transcriptions outputted by the ASR back to the client in real time. The final system is a VAD-ASR server that can accommodate a variety of interfaces, including web browsers and smart speakers.

This VAD-ASR architecture has been shown to work quite well on certain audio sources. One way to qualitatively visualize the performance of the VAD is to remove the VADFST and ASR components, and simply output the probability scores. The result is a server that accepts raw audio from the client, and sends back probability scores, which can be plotted in real time. Assuming the network is sufficiently fast, the probability plot will have a latency of approximately 250ms. A depiction of a web browser interface that plots the probability of speech is in Figure 6-3.

6.3 Smart Speaker Deployment

I also explore the use of VAD on an experimental smart speaker that runs Android operating system. The speaker is capable of running Android applications, and is equipped with a dual-channel microphone array and Internet connectivity over WiFi. Its programmable operation and Internet connectivity make it a perfect candidate for the practical application of VAD.

For computationally constrained smart speakers, it may be a requirement that ASR models be run on external servers. In this scenario, the speaker must stream incoming audio over the network for processing. While ASR systems in and of themselves benefit from VAD, there is an additional benefit of limiting network traffic by running VAD locally so that outgoing audio can be filtered.

Despite being both computationally constrained and running Android OS, it's still possible to run the CNN-GRU or CNN-GRU-RAW implemented in Tensorflow on the

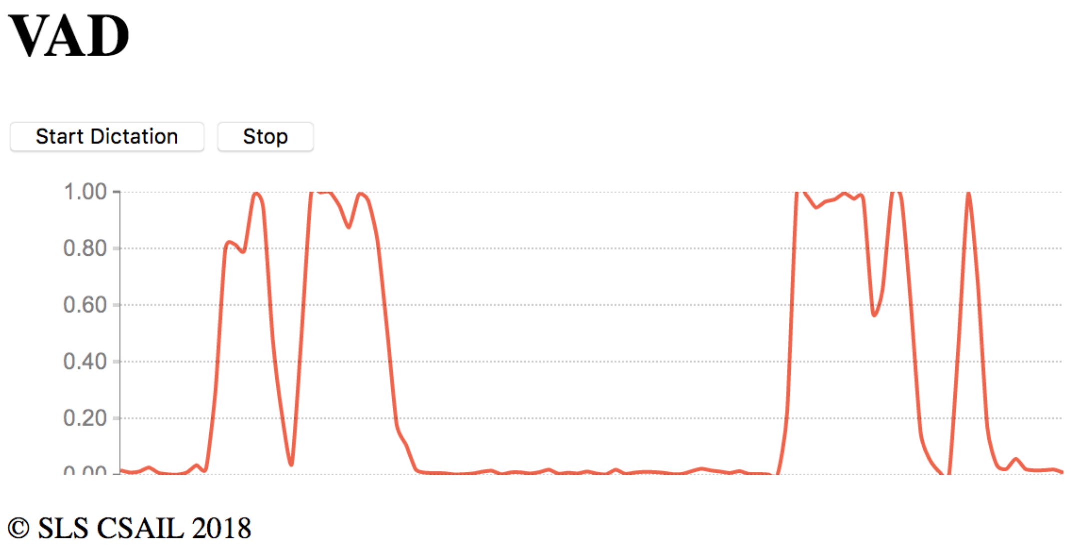


Figure 6-3: A simple browser interface for plotting the probability of speech in real time. This browser interface takes in raw audio from the host machine's microphone, and sends it to a server running the VAD architecture implemented with SLS Streamer described in Section 6.2.

device. The CNN-GRU and CNN-GRU-RAW models are implemented in Python, but trained version can be exported and run via the open source TensorFlowInferenceInterface library[4]. The TensorFlowInferenceInterface library facilities the execution of exported Tensorflow models within an Android application.

The TensorFlowInferenceInterface library is part of a binary that is compiled directly from Tensorflow source code on Tensorflow's master repo on Github[5]. Tensorflow also provides nightly builds on their Jenkins Continuous Integration server, accessible to the public. Unfortunately, to keep binaries small in size, the out-of-the-box nightly build for the TensorFlowInferenceInterface library does not support all Tensorflow ops. In fact, it only supports a small subset of the most common ops.

I found it to be the case that the out-of-the-box nightly build did not support all the ops used by CNN-GRU and CNN-GRU-RAW. After browsing many blog

[4]https://github.com/tensorflow/tensorflow/tree/master/tensorflow/contrib/android
[5]https://github.com/tensorflow/tensorflow

posts, StackOverflow[6] posts, and Github issues, I learned that I could get around this by compiling the TensorFlowInferenceInterface binary myself while specifying the exact ops that I needed. This process involves using a script[7] to audit the exported Tensorflow graph and generate an `ops_to_register.h` file. This header file, when copied into the tensorflow/core/framework directory, directs the compiler to compile only the specified ops. Working inside of a Docker Container that replicated the exact environment of the official Tensorflow nightly builds, I was able to compile— from scratch— a TensorFlowInferenceInterface binary that supported all of the ops needed by the CNN-GRU and CNN-GRU-RAW models[8].

With the proper TensorFlowInferenceInterface binary, I implemented a component graph architecture almost identical to the one described in Section 6.2 in Java. This architecture supported real-time VAD running directly on the smart speaker. I did not attempt to run the CNN-GRU, but instead opted for the CNN-GRU-RAW model. The reason for this decision was to avoid implementing a feature computation layer for converting raw audio to log Mel filter bank features in Java. Due to the potential for subtle differences when implementing a relatively complex feature computation pipeline across different programming languages and environments, I did not want to risk the effort only for the CNN-GRU to perform poorly amidst such differences[9]. Despite choosing to run the CNN-GRU-RAW model, it was still capable of running in 1.5 times real-time on the smart speaker.

6.4 Towards Generalizability

With both the smart speaker VAD implementation, and the browser-based application shown in Figure 6-3, I did a lot of qualitative evaluation. Almost immediately, it was apparent that the CNN-GRU-RAW performed much better on audio from a Macbook microphone than from the smart speaker. To rule out factors that may

[6]https://stackoverflow.com/

[7]https://github.com/tensorflow/tensorflow/blob/master/tensorflow/python/tools/-print_selective_registration_header.py

[8]There were *many* snags along this journey.

[9]Because the model was trained on features computed in the Python environment.

have been related to the Java implementation, I configured the smart speaker to send all of its audio straight to the SLSStreamer VAD architecture. I then configured the SLSStreamer VAD architecture to support broadcasting. That is, one client connection inputs audio, while the VAD results are broadcasted to all client connections. This allowed for the real-time plotting of VAD results in the browser, despite the audio stream coming from the smart speaker. With these factors controlled, I confirmed that the worse performance on the smart speaker was attributed directly to differences in the audio sources. The VAD with smart speaker audio was much more "trigger-happy." While the CNN-GRU qualitatively performed better than CNN-GRU-RAW on both sources of audio, the prevalence of false alarms on smart speaker audio was still present.

The lack of generalizability to the smart speaker audio was disappointing, and it may have turned out to be a coincidence that the VAD performed so well on audio from a Macbook. This lack of generalizability may have been related to some of the limitations of the dataset mentioned in Section 2.2, including the fact the TIMIT corpus represents a very specific acoustic environment. While the additive noise and artificial reverberations helped to expand the acoustic environment, it may not have been sufficient. These thoughts motivated me to gather more diverse data.

In gathering more data, I combined the dataset with two additional sources. The first source was a collection of non-speech sounds assembled for the DCASE 2018 acoustic event detection competition[10]. I added roughly 9000 non-speech segments, and spread them across the train, dev, and test partitions with a 4-1-1 ratio.

The second source, motivated by the poor performance on smart speaker audio, was a collection of TIMIT re-recordings. These re-recordings were recorded with the Android smart speaker in a single room with an absorbent ceiling. Various settings were simulated including having the smart speaker placed at different distances and positions with respect to the audio source. 200 TIMIT utterances were recorded at each of 0.2, 1m, 2m, and 3m distances. Another setting, designed to simulate reverberation, had the smart speaker placed slightly behind the audio source, with

[10]http://dcase.community/challenge2018/index

both devices facing the opposite side of the room. In total, there were 1000 re-recorded TIMIT utterances spanning roughly 106 minutes. These re-recordings were spread across the train, dev, and test partitions with a 2-1-1 ratio.

In addition to having more data, I needed a better way to empirically gauge how well the model generalized to other environments. Instead of selecting a model that performed well on the dev partition—a collection of data with very similar characteristics to the train partition—it seemed better to evaluate on data from a completely separate distribution. For closed-domain tasks with well-defined test partitions, evaluating on the dev partition would have been sufficient. For real-time deployment, a more robust methodology was needed to select a generalizable model. With this motivation I collected 36 minutes of labeled data from the Librispeech corpus, a collection of audio books with transcriptions (Panayotov et al. 2015). I used English forced alignments from (Kocabiyikoglu et al. 2018) to generate labels. This small collection of data was then used to evaluate the CNN-GRU periodically during training, so a high-performing checkpoint could be selected . This process assumed that if CNN-GRU generalized well to LibriSpeech, then it may have a propensity to generalize to other environments as well[11].

Figure 6-4 show the probability outputs for the medium CNN-GRU trained on the original dataset described in section 2.1. Figure 6-5 show the probability outputs for the medium CNN-GRU trained with the original dataset plus the additionally gathered data. While gathering more data, including a wider variety of non-speech sounds, had the effect of making the CNN-GRU slightly less trigger-happy on smart speaker audio, it also appeared to have lowered the true positive rate on Macbook audio. It's not immediately clear which dataset yielded better results. This experience highlights some of the difficulties related to the tendency of neural models to overfit to their training data. Training a robust CNN-GRU that generalizes to a wide variety of acoustic environments is an area for further exploration.

[11]More-so than if model selection was based on dev partition performance.

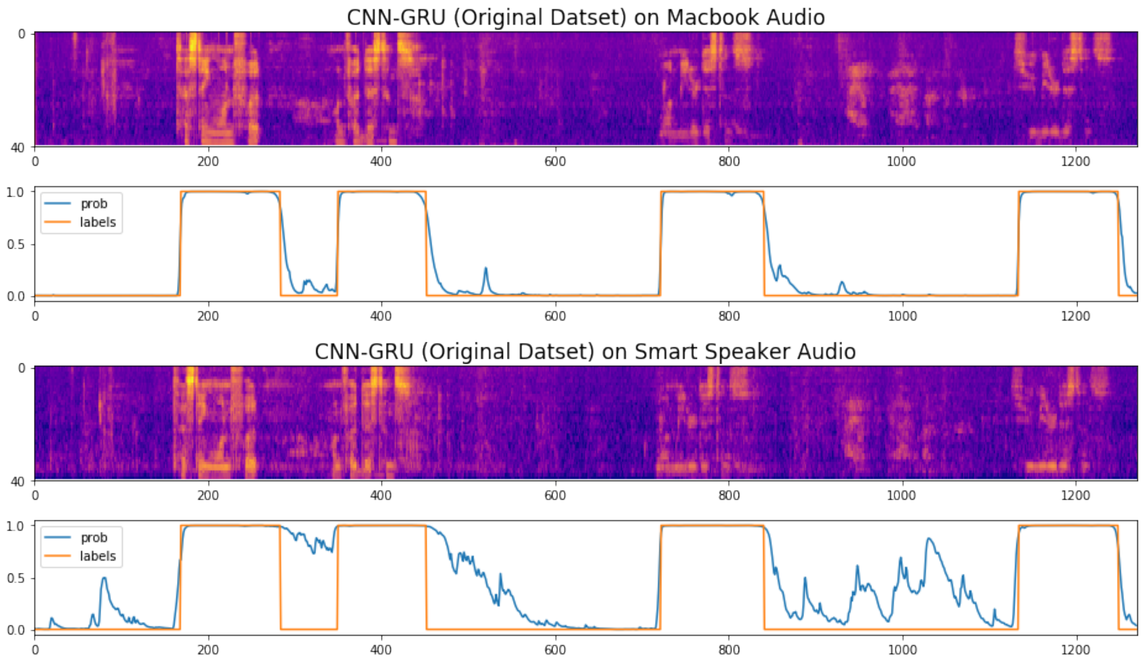


Figure 6-4: A comparison of the CNN-GRU model trained on the original dataset presented in Section 2.1. Audio was recorded from both a Macbook and smart speaker simultaneously, with words spoken at the same distance from each microphone. The CNN-GRU had a greater tendency to give probability density to non-speech frames for the smart speaker audio.

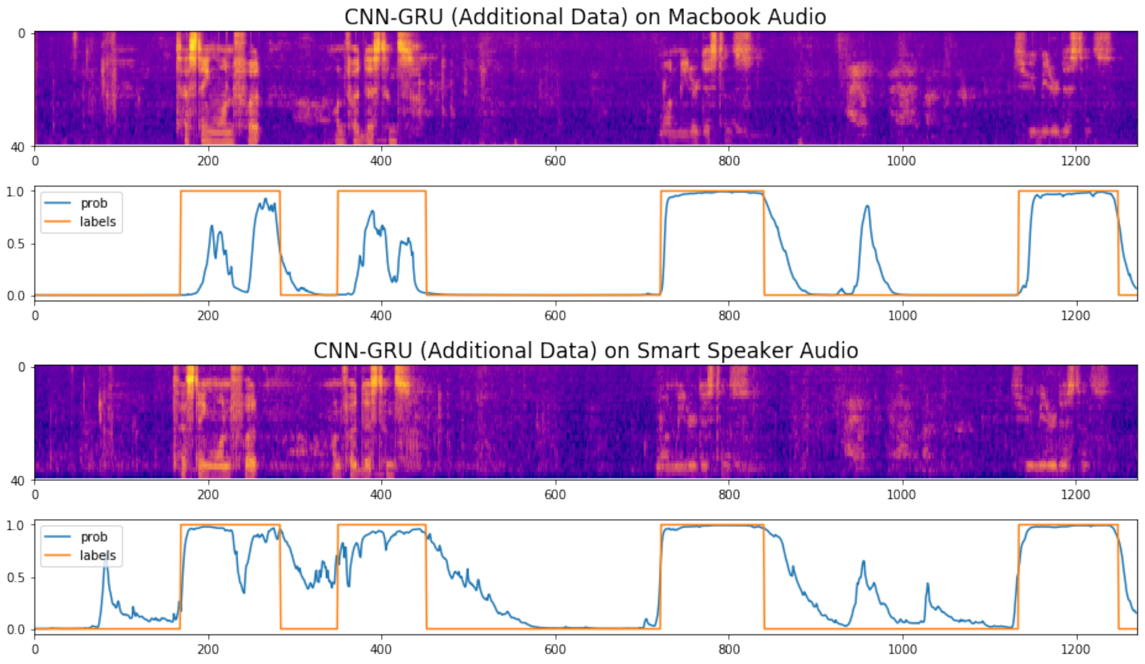


Figure 6-5: A comparison of the CNN-GRU model trained on the original dataset plus additional data, as discussed in Section 6.4. While the CNN-GRU is less likely to give probability density to non-speech frames in the smart speaker audio than in Figure 6-4, performance is degraded in other respects, especially on Macbook audio.

Chapter 7

Conclusion

In this thesis, I've introduced the problem of Voice Activity Detection (VAD), including some of the challenges related to producing a VAD model that is robust amidst a variety of acoustic environments. This task of producing a robust VAD has been explored for several decades. Historically, many of the proposed VAD models have been highly heuristic in nature—it was not until recent that statistic models have gained popularity, including the use of Deep Neural Networks (DNNs).

In exploring a lightweight, deep, recurrent neural architecture called CNN-GRU, I first introduced a methodology for artificially generating large amounts of noisy speech data from a clean speech source. While this methodology was able to produce realistic samples, it still possesses a few fundamental limitations.

In analyzing the performance of the CNN-GRU architecture, I explored 3 size variants, testing for both accuracy and computational efficiency. I found that while larger variants were more accurate, they were also less computationally efficient. While computational cost was roughly linear in the size of the network, there were diminished returns in accuracy. I also explored the performance of a CNN-GRU variant trained only on clean speech and non-speech sounds. This model performed well on noisy speech at 15dB SNR and above.

I explored a fully end-to-end variant of the CNN-GRU, called CNN-GRU-RAW, which learned features directly from raw waveform data as opposed to accepting a feature representation like log Mel filter banks as input. This model, while computationally less efficient than CNN-GRU, learned to extract spectral information from the raw waveform with a distribution similar to the Mel scale. The propensity of CNN-GRU to respond particularly strongly to high frequencies may have shed light on idiosyncrasies of the TIMIT recordings used during training.

Lastly, I demonstrated how both the CNN-GRU and CNN-GRU-RAW models could be deployed live with a real-time audio stream. In deploying, I described how a finite state transducer (FST) could be layered on top of these models to effectively capture whole spoken utterances instead of highly granular segments of speech. In deploying amidst unseen environments, including an Android IoT smart speaker, I found that the models had trouble generalizing to the subtle differences. I briefly explored how to combat this problem with the introduction of more diverse training data. The problem of making the CNN-GRU more robust to unseen environments is an area for further research.

Bibliography

J. H. Bach, B. Kollmeier, and J. AnemÃijller. Modulation-based detection of speech in real background noise: Generalization to novel background classes. In *2010 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 41–44, March 2010. doi: 10.1109/ICASSP.2010.5496244.

Liang Bai, Zhen Zhang, and Jun Hu. Voice activity detection based on deep neural networks and viterbi. *IOP Conference Series: Materials Science and Engineering*, 231(1):012042, 2017. URL `http://stacks.iop.org/1757-899X/231/i=1/a=012042`.

Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars. *CoRR*, abs/1604.07316, 2016. URL `http://arxiv.org/abs/1604.07316`.

Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014. URL `http://arxiv.org/abs/1406.1078`.

Ekapol Chuangsuwanich and James Glass. Robust voice activity detector for real world applications using harmonicity and modulation frequency, 2011.

Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014. URL `http://arxiv.org/abs/1412.3555`.

David B. Dean, Sridha Sridharan, Robert J. Vogt, and Michael W. Mason. The qut-noise-timit corpus for the evaluation of voice activity detection algorithms. In *Interspeech 2010*, Makuhari Messe International Convention Complex, Makuhari, Japan, September 2010. URL `https://eprints.qut.edu.au/38144/`. Click on the Related data link below to download the QUT-NOISE-TIMIT corpus dataset (listed under the databases tab) of QUT's SAVIT Research Program.

Rob Drullman, Joost M. Festen, and Reinier Plomp. Effect of temporal envelope smearing on speech reception. *The Journal of the Acoustical Society of America*, 95(2):1053–1064, 1994. doi: 10.1121/1.408467. URL `https://doi.org/10.1121/1.408467`.

William M. Fisher, George R. Doddington, and Kathleen M. Goudie-Marshall. The darpa speech recognition research database: Specifications and status. In *DARPA Workshop on Speech Recognition*, pages 93–99, 1986.

Houman Ghaemmaghami, Brendan J. Baker, Robert J. Vogt, and Sridha Sridharan. Noise robust voice activity detection using features extracted from the time-domain autocorrelation function. In *Interspeech 2010*, Makuhari Messe International Convention Complex, Makuhari, Japan, 2010. URL `https://eprints.qut.edu.au/40656/`.

Alex Graves and Navdeep Jaitly. Towards end-to-end speech recognition with recurrent neural networks. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, pages II–1764–II–1772. JMLR.org, 2014. URL `http://dl.acm.org/citation.cfm?id=3044805.3045089`.

Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012. URL `http://arxiv.org/abs/1207.0580`.

Sepp Hochreiter and JÃijrgen Schmidhuber. Long short-term memory. 9:1735–80, 12 1997.

T. Hughes and K. Mierle. Recurrent neural networks for voice activity detection. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 7378–7382, May 2013. doi: 10.1109/ICASSP.2013.6639096.

Chanwoo Kim, Ananya Misra, Kean Chin, Thad Hughes, Arun Narayanan, Tara Sainath, and Michiel Bacchiani. Generation of large-scale simulated utterances in virtual rooms to train deep-neural networks for far-field speech recognition in google home. pages 379–383, 2017. URL `http://www.isca-speech.org/archive/Interspeech_2017/pdfs/1510.PDF`.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL `http://arxiv.org/abs/1412.6980`.

Ali Can Kocabiyikoglu, Laurent Besacier, and Olivier Kraif. Augmenting librispeech with french translations: A multimodal corpus for direct speech translation evaluation. *CoRR*, abs/1802.03142, 2018. URL `http://arxiv.org/abs/1802.03142`.

Trausti Kristjansson, Sabine Deligne, and Peder A. Olsen. Voicing features for robust speech detection, 01 2005.

L. Lamel, L. Rabiner, A. Rosenberg, and J. Wilpon. An improved endpoint detector for isolated word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 29(4):777–785, Aug 1981. ISSN 0096-3518. doi: 10.1109/TASSP.1981.1163642.

Michael A. Nielsen. *Neural Networks and Deep Learning*. 2015.

D. O'Shaughnessy. *Speech communication: human and machine*. Addison-Wesley series in electrical engineering. Addison-Wesley Pub. Co., 1987. ISBN 9780201165203. URL `https://books.google.com/books?id=mHFQAAAAMAAJ`.

V. Panayotov, G. Chen, D. Povey, and S. Khudanpur. Librispeech: An asr corpus based on public domain audio books. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5206–5210, April 2015. doi: 10.1109/ICASSP.2015.7178964.

Karol J. Piczak. Esc: Dataset for environmental sound classification, 2015. URL `https://doi.org/10.7910/DVN/YDEPUT`.

L. R. Rabiner and M. R. Sambur. An algorithm for determining the endpoints of isolated utterances. *The Bell System Technical Journal*, 54(2):297–315, Feb 1975. ISSN 0005-8580. doi: 10.1002/j.1538-7305.1975.tb02840.x.

Javier Ramirez, Jose C Segura, Carmen Benitez, Angel de la Torre, and Antonio Rubio. Efficient voice activity detection algorithms using long-term speech information. *Speech Communication*, 42(3):271 – 287, 2004. ISSN 0167-6393. doi: https://doi.org/10.1016/j.specom.2003.10.002. URL `http://www.sciencedirect.com/science/article/pii/S0167639303001201`.

Drew Rendall, Sophie Kollias, Christina Ney, and Peter Lloyd. Pitch (f0) and formant profiles of human vowels and vowel-like baboon grunts: The role of vocalizer body size and voice-acoustic allometry. *The Journal of the Acoustical Society of America*, 117(2):944–955, 2005. doi: 10.1121/1.1848011. URL `https://doi.org/10.1121/1.1848011`.

Tara N. Sainath, Ron J. Weiss, Andrew W. Senior, Kevin W. Wilson, and Oriol Vinyals. Learning the speech front-end with raw waveform cldnns. In *INTERSPEECH*, 2015.

Jia-Lin Shen, Jeih-Weih Hung, and Qin Fen. "robust entropy-based endpoint detection for speech recognition in noisy environments", 01 1998.

Hava T. Siegelmann and Eduardo D. Sontag. Turing computability with neural nets. *Applied Mathematics Letters*, 4(6):77 – 80, 1991. ISSN 0893-9659. doi: https://doi.org/10.1016/0893-9659(91)90080-F. URL `http://www.sciencedirect.com/science/article/pii/089396599190080F`.

Jongseo Sohn, Nam Soo Kim, and Wonyong Sung. A statistical model-based voice activity detection. *IEEE Signal Processing Letters*, 6(1):1–3, Jan 1999. ISSN 1070-9908. doi: 10.1109/97.736233.

K. Srinivasan and A. Gersho. Voice activity detection for cellular networks. In *Proceedings., IEEE Workshop on Speech Coding for Telecommunications,*, pages 85–86, 1993. doi: 10.1109/SCFT.1993.762351.

S. S. Stevens, J. Volkmann, and E. B. Newman. A scale for the measurement of the psychological magnitude pitch. *The Journal of the Acoustical Society of America*, 8(3):185–190, 1937. doi: 10.1121/1.1915893. URL `https://doi.org/10.1121/1.1915893`.

S. G. Tanyer and H. Ozer. Voice activity detection in nonstationary noise. *IEEE Transactions on Speech and Audio Processing*, 8(4):478–482, Jul 2000. ISSN 1063-6676. doi: 10.1109/89.848229.

Luke Taylor and Geoff Nitschke. Improving deep learning using generic data augmentation. *CoRR*, abs/1708.06020, 2017. URL `http://arxiv.org/abs/1708.06020`.

Ruben Zazo, Tara N. Sainath, Gabor Simko, and Carolina Parada, editors. *Feature Learning with Raw-Waveform CLDNNs for Voice Activity Detection*, 2016.

Hui Zhang, Hong Su, Xue-Liang Zhang, and Guang-Lai Gao. Convolutional neural network for robust pitch determination. *Acta Automatica Sinica*, 42(6):959, 2016. doi: 10.16383/j.aas.2016.c150672. URL `http://www.aas.net.cn/EN/abstract/article_18887.shtml`.