INGENIEURSWETENSCHAPPEN

FACULTY OF ENGINEERING
DEPARTMENT OF APPLIED SCIENCES AND ENGINEERING

Learning the building blocks of speech - Deep Networks

Master thesis submitted in partial fulfillment of the requirements for the degree of
Master of Science in Applied Sciences and Engineering: Applied Computer Science

Yanfang Guo

Promoter: Prof. Bart de Boer

Academic year 2017-2018

Abstract

Speech recognition is a 'speech to text' process. The ASR (Automatic Speech Recognition) system evolved into the deep learning era in 2012 with the emergence of the pretrained deep belief network. Later the ASR deep learning models developed into the CNNs (convolutional neural networks) and RNNs (recurrent neural networks). CNNs have shown a great success in dealing with the variations of speech and our thesis use CNNs as our models.

Variability in speech is a major problem in speech recognition, and there are already a lot of applications using convolutional neural networks which are known to be invariant to frequency shifts. However, there is a lack of understanding why the convolutional neural networks are so successful. Our work constructs a series of convolutional neural networks models with different depths, different sizes of the kernels in convolutional layers pooling layers, as well as different number of hidden units in dense layers. We compare their performances and find the impacts of these convolutional neural networks components.

Our second research topic is about the front-end feature extraction method for the convolutional neural network. We explore the availability of using GFSCs (gammatone filterbank spectral coefficients) method which is inspired by human auditory perceptron, and STFT (short time Fourier transform) method which is fast and efficient. We add different strengths of background noise levels and examine their performances respectively. Our study fills the gap of testing the CNN models using STFT and GFSCs feature extraction method in noisy environments.

Declaration of Originality

I hereby declare that this thesis was entirely my own work and that any additional sources of information have been duly cited. I certify that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis and have included copies of such copyright clearances to my appendix.

I declare that this thesis has not been submitted for a higher degree to any other University or Institution.

Acknowledgements

After discussing several mater thesis topics with different professors in September 2017, I finally decided to choose topic of speech recognition and deep learning as my master thesis topics with Prof. Bart de Boer. At that time, I had already have some machine learning and deep learning knowledges, but speech recognition was completely new to me. And First and foremost thanks to my promoter Prof Bart de Boer for explaining me the knowledge of linguistics, brainstorming weekly discussion, numerous conversations on how this subject could be tackled and suggestions for how to write a master thesis. This is my first thesis, and Prof Bart de Boer really teaches me how to do a literature review, design the experiments in a scientific way and show me how grateful human speech it is.

Then I would like to thank my roommates Xinye Chen, Chenlu Wang and Yangxintong Lyu, for always being good listeners, sharing ideas, and providing the warm home atmosphere. I would also like to thank all of my classmates, for the comprehensive supports and weekend Chinese dinners. Best of the days ever.

Lastly, I must express my infinite gratitude to my parents who support me emotionally and financially, and always try their best to provide the best life for me. I would specially thanks for my dad's humorous moments and my mom's encouragements and understandings.

Contents

List of Figures

List of Tables

1

Introduction

Speech recognition enables computers to understand spoken language. There are many speech-based applications including the mobile phone virtual assistants Siri, automatic translation, automatic subtitle generation etc. Speech recognition history can be dated back to the 1950s, in which the Bell labs invented a system that recognized spoken digits. Later, there appeared using custom special purpose hardware in commercial small vocabulary application over telephone. In 1980s, speech recognition evolved from using templates for sound patterns to HMM based on statistic model (O'Shaughnessy, 2008) and later the GMM-HMM model proved to be a great success in ASR systems. The GMM-HMM model is the most distinguishing model before the emergence of deep learning models.

The emergence of the pretrained deep belief network opened the era of acoustic deep learning models. Later, the deep learning models evolved into the recurrent neural networks which can deal with long term dependencies and convolutional neural networks which have great ability of dealing with variation in speech signals. The RNNs and CNNs have both shown their success in many speech recognition tasks and applications.

Variability in speech is a major obstacle in speech recognition. Each per-

son has a different vocal tract. The variation can be different pitches, loudnesses, speeds, intonations, rhythms, environmental noise etc. The recent trends in acoustic modeling have led to a proliferation of studies that using convolutional neural network to handle the variability in speech and extract features without regard to variation. The convolutional neural networks have already demonstrated remarkable results in dealing with variation. But we still do not well understand how the convolutional neural network deal with variation. How do the convolutional neural networks extract features? What is the optimal CNN network structure for speech recognition? How to set the number and sizes for CNN filter kernels and pooling kernels? What is the impact of order for convolutional layers, maxpooling layers and dropout layers? How many dense layers should be put after the flatten layer? What is the optimal number of hidden units in dense layer? In our thesis, we generate fresh insights into understanding the convolutional neural network. The first topic of our work focuses on what the effects of the components in convolutional neural networks are.

The ASR system includes the front-end feature extraction part and the back-end classifier part. For the back-end classifier, we use the convolutional neural networks. We are also interested in the front-end feature extraction method. The past ASR system history has seen the development of front-end feature extraction method. In 1970s when ASR system focused on small vocabulary tasks over telephone, linear predictive coding (LPC) was the dominant ASR representation. In 1980s when ASR models evolved into HMM statistical model, the Mel-Frequency Cepstral Coefficients (MFCCs) was the leading front-end feature extraction method. Until now, there are still a lot ASR application using MFCCs as their front-end feature extraction method. A possible explanation why MFCC is so popular is that it has many characteristics that are well suited to the HMM-GMM model, which is the dominance in traditional ASR models. However, the MFCC method decorrelates the frequency correlation which is important for speech recognition task, and some evidences had shown that the MFCCs' performance degraded in noisy environment. Paliwal explained the MFCCs' problems in *Decorrelated and Liftered Filterbank Energies for Robust Speech Recognition (Paliwal, 1999)*. We become interested in the filterbank energy feature extraction method. We review filterbank energy methods and we become really interested in Park's paper *Using The Gammachirp Filter for Auditory Analysis of Speech (Park, 2003)*. The gammatone filterbank is inspired by cochlea in human auditory system which works well in noisy environments. We also become interested in the short-time Fourier transform method which preserves all the frequency information and has fast and efficient computations. This fast and efficient

method preserves the temporal and spatial information of the speech, we also want to explore the availability of this feature extraction method, although they are not as popular as MFCCs in ASR systems. In short, our second aim is to explore the availability of some less popular front-end feature extraction methods. One is the human-auditory-perceptron-inspired GFSCs (Gammatone Filterbanks Spectral Coefficients) method and we also design experiments to see its performance in noisy environments. Another one is the fast and efficient STFT (Short Time Fourier Transform) method which might find its use in real-time or limited computation scenarios.

1.1 Research Questions

As stated earlier, our work covers the front-end feature extraction method and back-end classifier in an ASR system.

Understanding the back-end classifier, to be more specifically, the convolutional neural networks, is our first research topic. We control the variables and change one variable each time to examine the impact of CNN's parameters. We can not explore all the combinations and we limit ourselves to three of the factors which we are interested in, the depth of CNN, the size of convolutional layer kernels and pooling layer kernels and the number of hidden units in dense layers.

The front-end extraction part covers the GFSCs (Gammatone Filterbanks Spectral Coefficients) and STFT (Short Time Fourier Transform). We make a thorough inquiry to the robustness of feature extraction methods in noisy environments, and we also want to explore the capability of the efficient STFT method used in deep learning networks. Our approach to achieve that is to add noise to original audio gradually, then extract features using both extraction methods. We use the CNN model with the same structure to evaluate these two feature extraction methods respectively.

In short, we investigate GFSCs and STFT feature extraction methods and their usage in convolutional network. What's more, we design experiments to examine the impact of CNN's parameters.

1.2 Structure of the Thesis

This thesis is composed of seven themed chapters. The first chapter gives an introduction to our work and a statement of our research topics, as well as the structure of the thesis.

The second chapter reviews the front-end acoustic feature extraction methods and deep learning models for automatic speech recognition. The front-end feature extraction method section covers the MFCCs (Mel-Frequency Cepstral Coefficients) , MFSCs (Mel-Frequency Spectral Coefficients), STFT (Short Time Fourier Transform) , GFSCs (GammaTone Filterbank Spectral Coefficients) and we also review their usages in deep learning models. In deep learning model part, we give an overview of the mainstream deep learning models used in speech recognition from 2012. In this section, we first start with the DBN (Deep Belief Networks) which use the RBMs (Restricted Boltzmann Machines), then we introduce CNN (Convolutional Neural Network), RNN (Recurrent Neural Networks) and the combination of CNN and RNN.

The third chapter mainly describes our feature extraction methods. The chapter describes the method of computing spectrograms of Short Fourier Transform and gammatone filterbanks, and also compares their computing efficiency. At last, we give an intuitive comparison of our two methods in varying signal to noise ratio environments. Besides, this chapter also gives a small introduction of Google Command datasets and our method to remove silence in the audio file.

The fourth chapter is concerned with the methodology about the convolutional neural networks. The CNNs are stacked by different kinds of layers and we give the formulas for these layers. We also build a CNN demo and explain functions of these layers in a more intuitive way.

Following by the chapter five which serves as a tutorial for TensorFlow and Keras users. Readers who are familiar with the deep learning framework could skip this chapter. We explain the programming schema for TensorFlow and Keras. We implement the demo CNN in chapter four step by step use these two framework.

Chapter six presents our detailed experimental settings and the results. The experiments are divided into two parts, the experiments of exploring the CNN models and the experiments to compare our STFT and GFSCs front-end feature extraction method.

The final chapter concludes the results and presents the future work.

2

Review of Acoustic Feature Extraction and Deep Learning Models

In this section, we review some front-end feature methods which are the most related to our thesis work and review the deep learning models for automatic speech recognition systems. On the front-end feature extraction domain, we cover the MFCC (Mel-Frequency Cepstral Coefficients), MFSC (Mel-Frequency Spectral Coefficients), STFT (short time Fourier transform), GFSC (GammaTone Filter-bank spectral coefficients) and their roles in the deep learning models as well as relations with our thesis. In deep learning model domain, we give an overview of the mainstream deep learning models used in speech recognition. There are some landmark deep learning models appearing in the most recent 10 years, the DBN (Deep Belief networks) using the RBM(Restricted Boltzmann Machine), the CNN , the RNN and the combinations of CNN and RNN.

2.1 Front-end ASR Feature Extraction Methods

Speech signals are time-varying signals, the time domain waveforms carry all the auditory information. Speech signals are non-stationary but in a sufficiently short time window they could be viewed as stationary. Using original waveforms for ASR system does not yield a good performance in most of the cases, and many ASR models we reviewed do not use the original signals directly, but covert the original signals to time-frequency domain, and this process mimics the human auditory perception.

Converting the original waveforms in time domain to time-frequency domain is usually considered as the front-end feature extraction technique for an ASR system. It decides to keep what kind of information from speech. There are many speech feature extraction techniques in the field of research over the years, we review some papers of deep learning ASR models and present some of the most popular methods as well as their performances in this section.

2.1.1 MFCC/MFSCs: Mel-Frequency Cepstral/Spectral Coefficients

MFCC/MFSCs are the abbreviations of Mel-Frequency Cepstral/Spectral Coefficients. The steps to calculate the MFCCs and MFSCs are almost the same, except that the spectral coefficients (MFSCs) do not need the discrete cosine transform step. The MFCCs is one of the most successful feature extraction techniques and is widely used is ASR systems. We give a simple introduction to MFCCs and MFSCs in this section because of their success in ASR history, but our work are trying to explore the computation-efficient STFT and GFSCs method which mimics human auditory perceptron. We will not use MFSCs and MFCCs in our work.

MFCCs with traditional ASR:

The MFCCs (Mel-Frequency Cepstral Coefficients) is a feature extraction method widely used in ASR system. It has good representation of the perceptually relevant aspects of the short-term speech spectrum. By far the

MFCCs method is still the most widely used feature extraction method in the field of ASR feature extraction. One reason why MFCCs achieving a good performance in ASR is that they have many characteristics that are well suited to the HMM-GMM systems, which is the dominant model among traditional ASR models.

The Figure 2.1 gives the block diagram of computing the MFCCs and MFSCs. The first step is performing the fast Fourier transform in a short time window, then wrap the parameters in Mel-frequency scale and apply log() operation. The MFCCs needs additional discrete Fourier transform step. The Mel-frequency mimics the non-linearity of human cochlea and the log() operation mimics the logarithmic perception of loudness. We will not discuss them in detail since we do not use MFCC/MFSCs in our work. The definition about cepstrum, computation steps of MFCCs, roles of MFCCs and cepstrum could be found in Lawrence R.Rabiner's book(Rabiner, Schafer, et al., 2007): "Introduction to Digital Speech Processig, chapter 5: Homomorphic Speech Analysis".

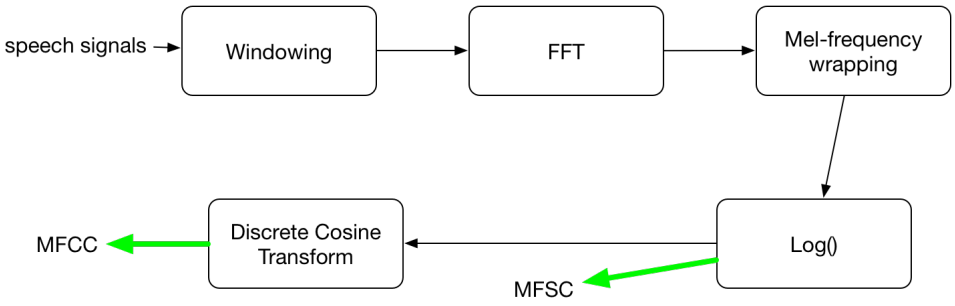


Figure 2.1: The block diagram of MFCC/MFSCs algorithm

Even though MFCCs method has shown a great success in ASR system, especially in clean speech tasks, it still has weakness. Some papers and studies have shown that performances of traditional ASR systems with MFCCs features degrade in the presence of noise. Narayana explored the effect of noise in speech using MFCC features in "Effect of noise-in-speech on MFCC parameters". More details could be found at (Narayana & Kopparapu, 2009).

MFCC/MFSCs with DNN

MFCCs is still the most widely used speech feature extraction method in DNN acoustic models by far. Most of the papers we reviewed about DNN

ASR systems use the MFCC as the feature extraction method. But when using CNN as speech recognition model, the discrete cosine transform of spectral features may make the MFCCs lose locality. To solve this issue, MFSCs feature extraction method which discard the discrete cosine transform is used in some CNN models.

There are some reasons why we do not choose MFCCs. Our model is the convolutional neural network model, which prefers the structural data. The discrete cosine transform decorrelates the frequencies correlation, which makes MFCCs lose the some spatial information which is important for speech recognition task. What's more, the MFCCs performance degrades in noisy environments, we would like to explore performance of noise-robust feature extraction techniques in deep learning models. We hope to find the potential of some feature extraction methods which are not as popular as MFCCs in deep learning models.

2.1.2 STFT: Short Time Fourier Transform

STFT with traditional ASR:

Speech signals could be viewed as quasi-stationary. The STFT feature extraction technique performs FFT in a sufficient short time window. The spectrogram of STFT is given below:

$$X(n,\omega) = \sum_{m=-\infty}^{\infty} x[m]w[n-m]e^{-j\omega n}$$

Where $x(m)$ is the signal and w is the window function. ω denotes the frequency.

By discreting frequency ω, applying the log to the absolute value, we get the discrete STFT :

$$STFT(n,k) = log|X(n,\omega)|_{\omega=\frac{2\pi}{N}k}|$$

In human speech recognition system, speech signals are processed using a temporal window of duration of 20-40ms with 5-15ms overlap.

The STFT method is a simple and fast front end feature extraction method, but it is not as popular as MFCCs in traditional ASR . Shrawankar

explained why STFT is not as popular as MFCC and LPCC in ASR ,*"As compared to methods exploiting knowledge about the human auditory system, the pure FFT spectrum carries comparatively more information about the speech signal. However, much of the extra information is located at the relatively higher frequency bands when using high sampling rates , which are not usually considered to be salient in speech recognition"* (Shrawankar & Thakare, 2013). The STFT contains more redundant information compare to MFCCs and LPCCs (Linear Predictive Cepstral Coefficients), which may lead to the low performance in traditional ASR models.

STFT with DNN:

However, with the development deep neural networks which are able to handle the larger datasets and more redundant information, using STFT in ASR systems becomes possible. DNN has the ability to automatically learn good features to represent the speech signals. There are already some DNN models using STFT as their acoustic feature extraction methods. Examples of using STFT could be found in (Tüske, Golik, Schlüter, & Ney, 2014),(Huang, Kim, Hasegawa-Johnson, & Smaragdis, 2015), and (Liu, Li, & Ma, 2016), (LeCun, Bengio, & Hinton, 2015). Deep learning models in these papers are using STFT features as the input of DNN, which shows the possibility and potential of this fast and simple feature extraction technique.

STFT with my thesis:

In my thesis, the main DNN model is CNN (convolutional neural network). CNN's weight sharing and local connectivity mechanism make it suitable to deal with multi-dimension data with correlations. The STFT is a structural time-frequency data array, which is more suitable for CNN models when compare the most popular MFCCs and LPCCs feature extraction technique. What's more, there are already a lot efficient FFT implementations and programing libraries which require less time and computation power. We would like to explore this fast and efficient technique used in CNN models, and to find the potential of STFT in speech recognition tasks which have limited computing power and require fast response.

There are already some papers such as (Huzaifah, 2017) using CNN with STFT representations, and STFT yields a relatively good performance in CNN models.

2.1.3 GFSCs:Gammatone Frequency Spectral Coefficients

Human Auditory Perception

The human ear is composed of outer ear, middle ear and inner ear. The outer ear includes external auditory canal and tympanic membrane. The middle ear includes malleus and incus and the inner ear is composed of semicircular canals, cochlea, and cochlea nerves. The anatomy of human ear could be found in Figure 2.2. The sound waves travel through the outer ear. The changes of air pressure cause the tympanic membrane to vibrate. Then malleus and incus pass the changes of pressure to cochlea which accounts for translating the vibration of tympanic membrane into into electro chemical impulses in cochlea nerves. The nerves connect to the specific part of brain which performs further processing.

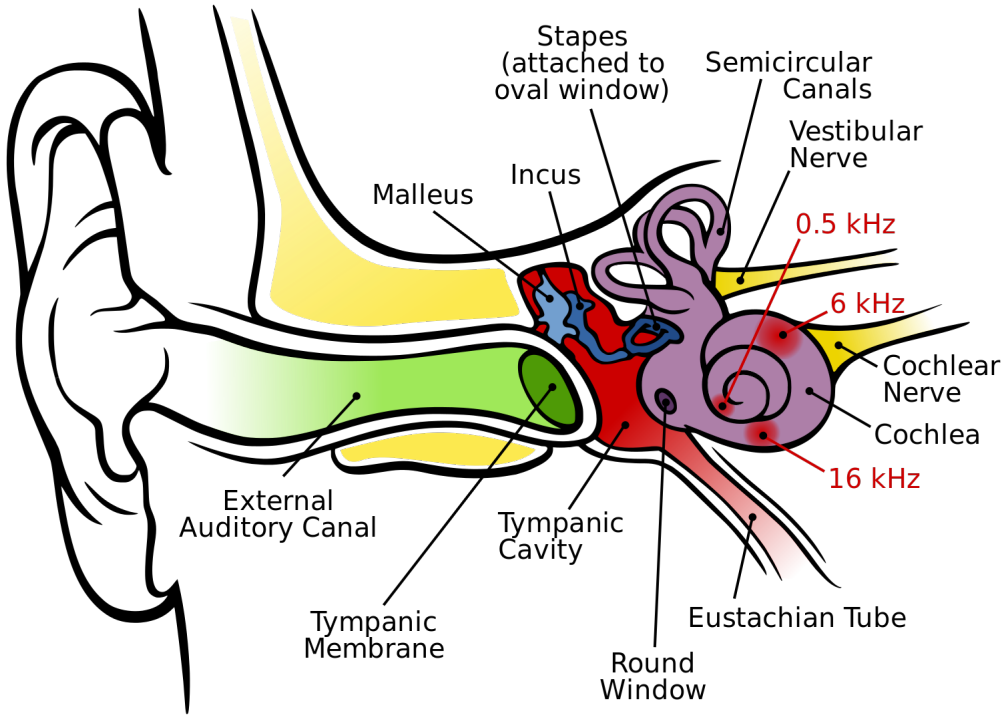


Figure 2.2: Anatomy of Human Ear
(Inductiveload, 2009)

The cochlea in the inner ear is the most important structure in human

auditory perception. A simplified cochlea model can be found in Figure 2.3. The cochlea can be simplified as a rigid, fluid-filled tube. The coiled basilar membrane separates the tube into two chambers and has different thickness and stiffness along its length. The thickness and stiffness at the specific point of the basilar membrane determine the frequencies that the basilar membrane is most sensitive to. The apex is least thick and least stiff while the base of the cochlea is most thick and most stiff. Therefore the apex at the tail of cochlea is more sensitive to lower frequencies while the base is more sensitive to the high frequencies. More detail information about auditory system could be found in (Pickles, 1988).

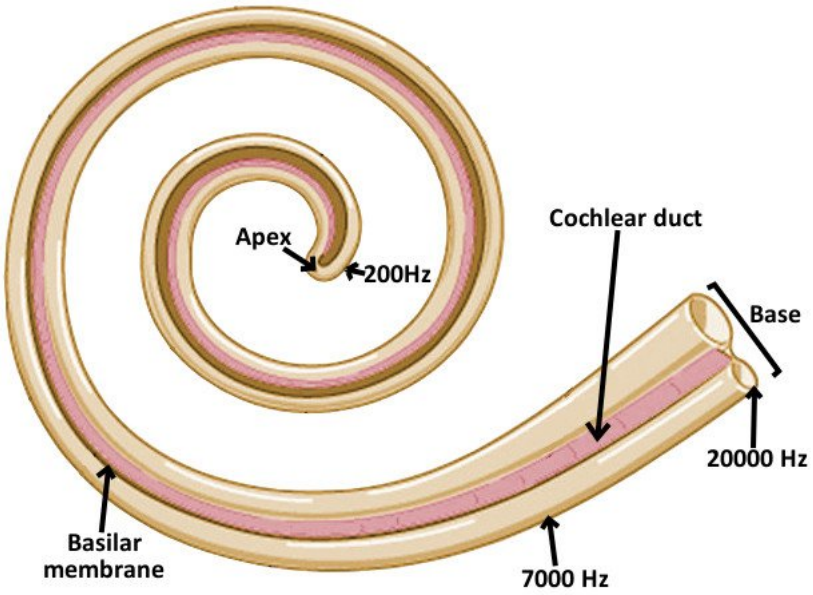


Figure 2.3: Basilar membrane in Cochlea
(Ashish, 2016)

The cochlea is often modeled as a set of filterbanks because it performs frequency analysis along the length of the basilar membrane. Different thickness and stiffness of the basilar membrane cause the different central frequencies f_c in each place. In Alex parks paper (Park, 2003), the author summarizes three most significant characteristics of the cochlea :

1. **Non-uniform filter bandwidths** : The frequency resolution is lower at high frequencies and higher at low frequencies). Which means for a set of filterbank, the lower center frequency filters have narrower bandwidths and the higher center frequencies filters have broader bandwidths.

2. **Asymmetric frequency response of individual filters** : For the specific point of membrane with the center frequency f_c, the response of $f_c + \Delta f$ is lower than the response of $f_c - \Delta f$.

3. **Level-dependent frequency response of individual filters** : The basilar membrane motion is compressive and non-linear. The peak gain of the filter centered at f_c decreases as the level the input stimulus increases.

Gammatone filterbanks

The Gammatone filterbanks are inspired by cochlea in human auditory systems. The cochlea has a compressive non-linear frequency response which depends on input stimulus levels. The human auditory system works well in noisy environments due to the three characteristics of cochlea(mentioned above): non-uniform filter bandwidths, asymmetric frequency response of individual filters and level-dependent frequency response of individual filters(Park, 2003).

The gammatone filters were derived by Irino (Irino & Patterson, 1997) in 1990s. The impulse response at time t is given by

$$g_t(t) = a \cdot t^{(n-1)} \cdot e^{(-2\pi b \cdot ERB(f_c)t)} \cdot cos(2\pi f_c t)$$

where constant a controls the gain, n defines the order of the gammatone filter and f_c is the center frequency and the ERB is the equivalent rectangular bandwidth of the center frequency. The b is associated with the stimulus level.

The ERB(f) is given by

$$ERB(f) = 0.1039f + 24.7$$

.

The impulse response of Gammatone is shown in Figure 3.3. The Re(z) and Im(z) indicate the value in real part and imaginary part.

The GFSCs(Gammatone Filterbank Spectral Coefficients) method is based the gammatone filters. The central frequencies of filterbanks are chosen in mel-frequency scale. The absolute value of gammatone filterbanks outputs are integrated, and the gammatone filterbank spectral coefficients are the

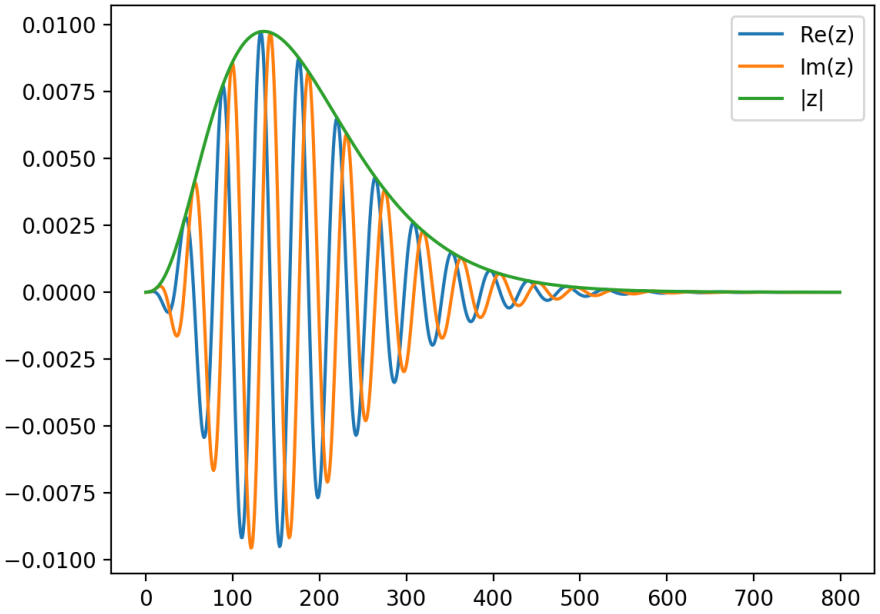


Figure 2.4: The Impulse Response of GammaTone Filter

log() of the integrated filterbank outputs. More details about gammatone filterbank and its implementation could be found in (Irino & Patterson, 1997) and (Hohmann, 2002).

GFSCs with ASR

GFSCs method is not as popular as MFCCs and LPCCs in ASR systems. The GFSCs method is more complicated, needs more computations and lack of attention in ASR feature extraction domain. Most of the papers about ASR system we reviewed do not choose the GFSCs as their feature extraction technique. While the traditional ASR systems performance degraded in noisy environments and we hope this noise-robust GFSCs feature extraction method could solve this issue.

GFSCs with my thesis

The GFSCs method mimics the human ear and human auditory perception which performs good even in noisy condition. My thesis explore the potential of using GFSCs in noisy environment. The GFSCs are two-dimension structural data arrays, which fit our convolutional neural network model.

2.2 ASR Deep Learning Models

The ASR (Automatic Speech Recognition) system allows computers to recognize speech, which is a speech-to-text process. The Figure 2.5 gives the block diagram of general ASR system. The HMM-GMM(Hidden Markov Models and Gaussian Mixture Models) model has proved to be a success in traditional speech recognition system, in which the HMM could handle varying length sequences and the GMM estimates the probabilistic distribution of speech signals. It is the most outstanding model before the emerge of deep learning models. The detail description of HMM-GMM models can be found in Lawrence Rabiner's book (Rabiner et al., 2007), *Introduction to digital speech processing, chapter 9 : Automatic Speech Recognition*, we will not discuss HMM-GMM models here.

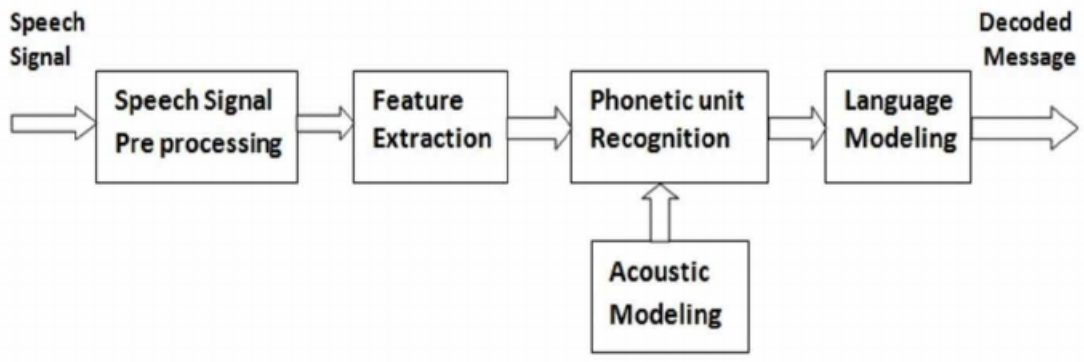


Figure 2.5: The block diagram of ASR system

With the development of machine learning algorithms, feed-forward neural networks have been used in ASR systems. Replacing the GMM with ANN(Artificial Neural Network) has several advantages, ANN does not require the detailed assumptions about the distribution of HMM states and

allows an easy way of combining discrete and continuous features. But large and deep neural networks are hard to train because of the gradient descent vanish/explode problem and local minima. There are already a lot of papers and tutorials discuss these problems, and we recommend Michael Nielsen's blog: *"CHAPTER 5:Why are deep neural networks hard to train*[1]*?"*

Over the recent years, deep learning makes a revolutionary success and now the ASR systems have evolved into the deep learning models era. The usage of RBMs (Restricted Boltzmann Machines) makes training large and deep neural network possible and this was the milestone progress of deep learning models. There is a series of deep learning models including DNN (Deep Neural network), DBN (Deep Belief Network), CNN (Convolution Neural Network), RNN (Recurrent Neural Network). In this section, we review some landmark deep learning papers for speech and we talk about the DBN, RNN, LSTM and CNN models.

2.2.1 DNN+HMM Hybrid System with DBN Pre-training

The DBNs (deep belief networks) are generative models, the structure of DBN is similar as structure in MLPs (Multi-layer Perceptrons), with several hidden units in the several hidden layers. But the training procedure of DBN is different. The basic unit of the hidden unit is RBM (Restricted Boltzmann). The Figure 2.6 illustrates the RBMs . The previous layer(Red) is visible for the blue layer(light blue). This generative model is trained in two directions, forward and reconstruction backward. By training in forward and back word directions, the RBMs do not suffer gradient vanishing or exploding problem. The DBNs are trained two layers at a time, and these two layers are treated as RBMs. The DBNs are trained layer by layer and DBNs are able to find the inherent patterns after training. This is called pre-training. The Pretrained DBNs parameters are used to initialize the DNN model.

We present some papers using DBNs in ASR systems. We go through these papers and evaluate DBNs performance in TIMIT phone recognition task as well as the large vocabulary tasks. The index we focus is the PER (Phone Error Rate and WER (Word Error Rate).

The paper (Mohamed, Dahl, & Hinton, 2009) published in 2009 used the deep belief networks for acoustic modeling. It proposed using DBN to model the spectral variabilities in speech and the task is to recognize the phonemes

[1]Link:`http://neuralnetworksanddeeplearning.com/chap5.html`

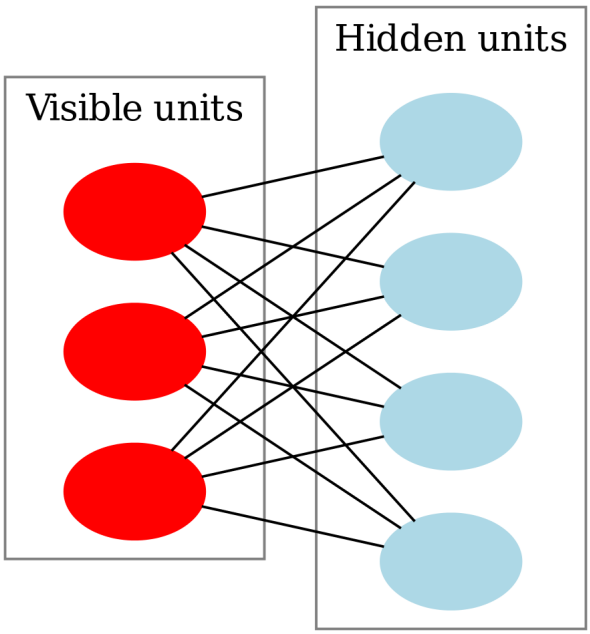


Figure 2.6: Diagram of RBMs with 3 inputs and 4 hidden Units (Quertyus, 2011)

and it reached 23% PER (phone error rate) in TIMIT dataset.

The paper (Dahl, Yu, Deng, & Acero, 2012) proposed a DBN pre-trained DNN-HMM models in large vocabulary speech recognition task in 2012. And it outperforms the GMM-HMM baseline model by 5.8 % PER. The paper (Jaitly, Nguyen, Senior, & Vanhoucke, 2012) trained the DBN-pretrained context-dependent DNN/HMM systems on 5870 hours and 1400 hours dataset (also large vocabulary task). And it also yielded a better result than GMM/HMM. The model structure of this two papers is given by Figure 2.7. The training pipeline of DNN/HMM system is shown in Figure2.7. GMM/HMM models are trained to align the training data, later the DBN are trained to initialize the network, then use the labeled dataset to fine tune the network. The pretrained-DBN-ANN/HMM outperforms the GMM/HMM models by 3.7% and 4.7% absolute WER (word error rate) in these two large datasets in this paper.

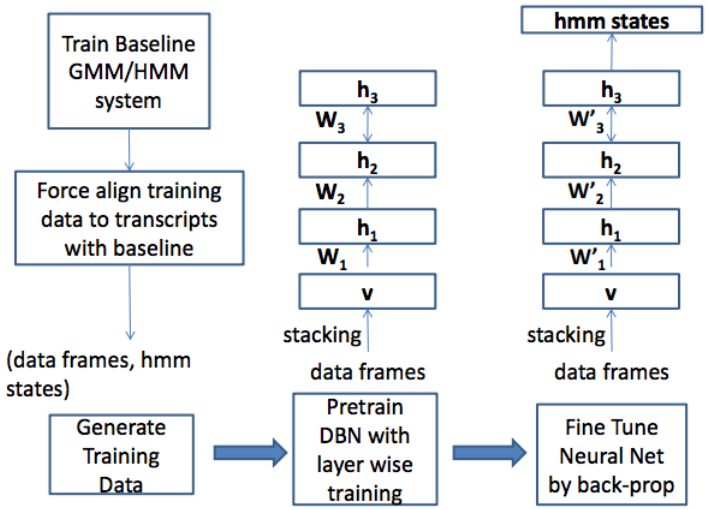


Figure 2.7: Pipeline for training DNN/HMM hybrid systems (Jaitly et al., 2012)

Using DBN to pre-train the network makes training large and deep neural networks possible, but we will not use this model in our work. The first reason is the DBN pretrained networks perform worse than the other deep learning models, and the second reason is that this model structure is still the normal neural network, it does not make good use of the spectral and temporal information of the speech.

2.2.2 Recurrent Neural Networks

Recurrent Neural Network (RNN) is a deep neural network which can deal with the temporal sequence data such as text and speech. Compare to conventional neural networks, the RNNs have memory which store th long-term dependencies information. The Figure2.8 shows the unfold structure of a RNN cell, where the X_t is input and h_t is hidden state and o_t denotes output at the time t. W, U, V are weight matrices.

The RNNs have the memory to store information and learn long term dependencies. The RNNs are not used in large scale until the emerge of advanced RNN Cell. The early RNNs are hard to train because of the gradient

exploding and vanishing problem. There are already a lot of studies working on that. Using GRU (Gated Recurrent Unit) cell and LSTM (Long Short Term Memory) cell with some gate functions are currently the most often used method in RNN networks. The Figure 2.9 shows the inner structure of these two cells. More details about LSTM and GRU cell could be found in two blogs (Olah, 2015) and (Simeon, 2017).

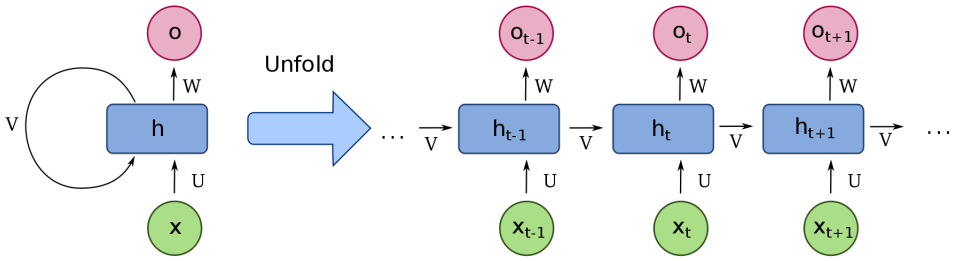


Figure 2.8: Unfold structure of RNN Cell
(Deloche, 2017)

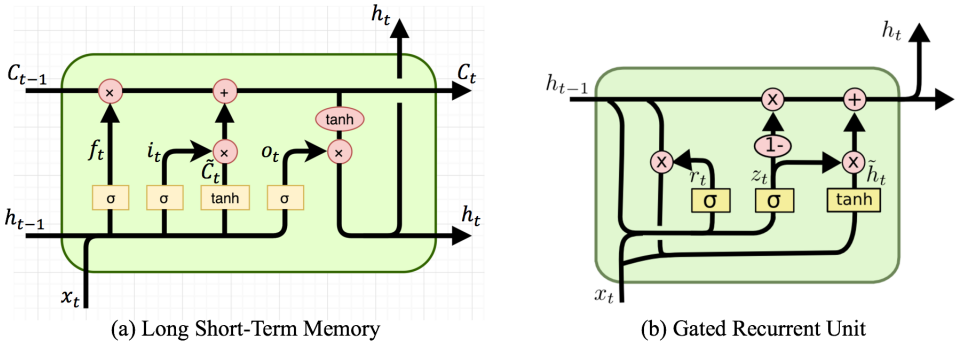


Figure 2.9: LSTM and GRU Cell
(Hao, 2017)

Because of RNN's strong ability of dealing with the long term dependencies, RNNs have shown great success in speech recognition tasks. In 2012, (Vinyals, Ravuri, & Povey, 2012) proposed a hybrid RNN+HMM model, where the role of RNN was to provide the posterior probabilities for HMM systems, and it outperform the DNN and DBN in noisy environments. The RNN's role in this paper is to replace the GMM model.

In 2013, (Graves, Mohamed, & Hinton, 2013) proposed an end-to-end deep short-term Memory RNNs, which combines multiple levels of represen-

tation that have proved effective in deep networks with the flexible use of long range context. This paper achieved 17.7% PER(phone error rate) on the TIMIT dataset. This end-to-end system is completely different from the HMM+GMM models.

The performance of RNNs ASR in large vocabulary speech tasks is also excellent. In 2014, (Hannun et al., 2014) proposed an end-to-end RNN systems which could perform well in noisy environment without hand-designed components to model background noise, or speaker variation. This system learns directly from the large varied data and achieved 16% WER(Word error rate) on Switchboard Hub5'00 datasets.

RNNs have shown impressive ability of catching the long term dependencies which are essential to speech sequences. While my work is focus on the word recognition, more focused on the acoustic model. Our task is to learn the spatial feature of the short speech signal, which do not have a demand of ability to deal with long-term-dependencies.

2.2.3 CNN+NN hybrid system

The convolutional neural networks have shown great success in dealing with structural data. The CNNs first show their success in the well-known image recognition tasks. And for speech recognition tasks, the spectrograms which are integrated in time-frequency domain could also be treated like images in CNN. The CNN has several kinds of layers, convolutional layer, pooling layer, ReLU layers, drop out layer and full connected layer. These layers are stacked together to perform discriminate tasks. More technical details about CNN could be found in *Chapter4, Section 2: convolutional neural network.*

The CNNs have shown significant success in ASR tasks. We list some CNN models performance in the speech recognition tasks. In 2012, (Abdel-Hamid, Mohamed, Jiang, & Penn, 2012) explored the CNN invariance to small shifts along the frequency axis and it reached 20.07% in phone error rates. In 2014, (Abdel-Hamid et al., 2014) proposed a limited weight sharing scheme which leads to a much smaller number of units in the pooling layer, result smaller model size and lower computational complexity than full weight sharing scheme. And it outperform the CNN model using full weight sharing scheme.

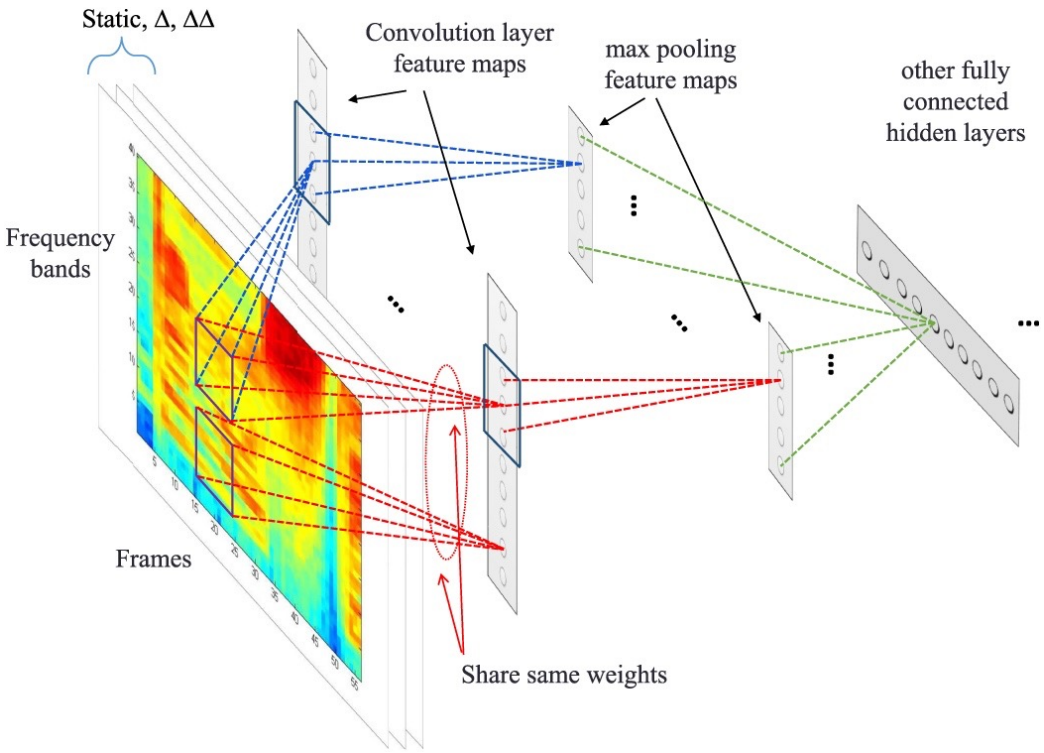


Figure 2.10: An illustration of the regular CNN architecture (Abdel-Hamid et al., 2014)

In 2013, (Sainath, Mohamed, Kingsbury, & Ramabhadran, 2013) applied CNN for large vocabulary speech tasks. It selected 400 hours of English Broadcast News and 300 hours Switchboard hours datasets and the CNNs offer 4-12% error over DNNs and proved that CNNs are better than DNNs also in large vocabulary speech tasks.

Below Figure 2.10 is an illustration of the regular CNN architecture. The input data has three channels, the static, delta and delta-delta feature. The convolution layer and max-pooling layer perform convolution and max-pooling operation. Later the output are connected to full connected layers to perform the classification task.

We explore the CNNs' ability of extracting the spatial information of the speech in our work, learning the building blocks of the speech.

2.2.4 CNN+RNN Hybrid Systems

The CNNs do well in dealing environmental variations and RNNs are good at handling the temporal informations among sequences. The latest research about deep learning models is to combine the CNN and RNN which takes advantage of the characteristics of both models.

In 2015, (Sainath, Vinyals, Senior, & Sak, 2015) proposed a CLDNN (convolutional-LSTM-DNN) architecture, which combines CNN, LSTMRNN and DNN together. The The CNN is use to detect features underlying the spectrogram. In this paper CLDNN performs 4%-6% relative improvement in WER(word error rate) than the LSTM on a variety of large vocabulary tasks.

In (Sainath et al., 2015) model, the output of CNN is fed into the LSTM network, which is a consecutive structure of CNN and LSTM. However (Bae, Choi, & Kim, 2016) proposed a parallel structure where CNN and LSTM accept different inputs separately, where the CNN and LSTM are more like dealing data in parallel. This model could learn the sequential information and spectral information simultaneously. Figure 2.11 gives the parallel CNN-LSTM model architecture. The CNN and LSTM explore the speech spectral data simultaneously and later connect the DNN to do classification.

In 2017, Microsoft AI team described its conversational speech recognition system in (Bae et al., 2016) . The system adds a CNN-BLSTM(CNN - bidirectional LSTM) acoustic model, the BLSTM could explore the long-term dependencies forward and backward and its achieves 5.1% WER(word error rate) on large vocabulary task, the 2000 switch board task.

The CNN+RNN models are by far the most powerful speech recognition models and yield the lowest PER(Phone Error Rate) and WER(Word Error Rate).

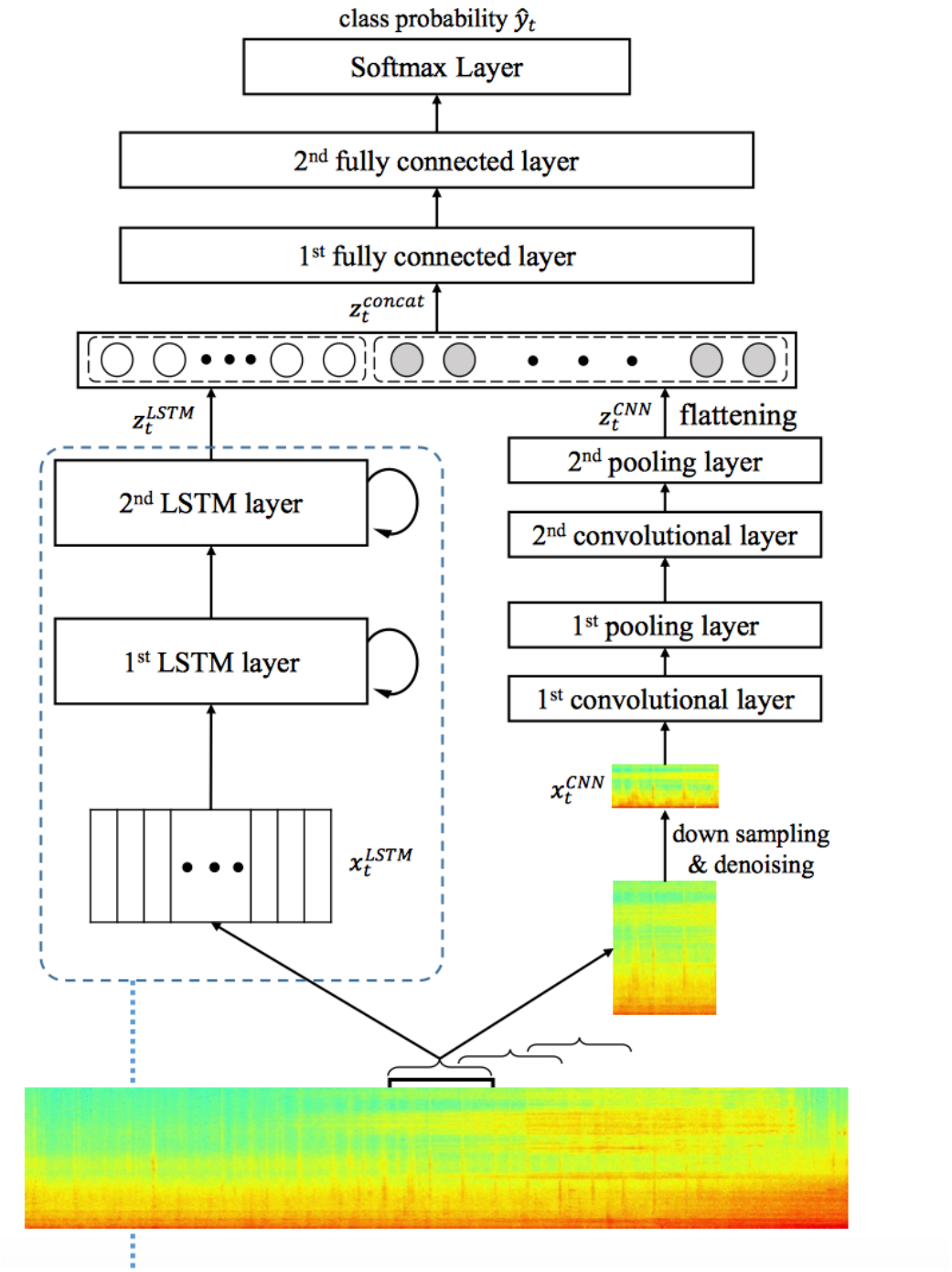


Figure 2.11: An Example of CNN-LSTM architecture (Bae et al., 2016)

2.3 Summary

We selected some milestone deep learning models in ASR systems, list their front-end feature extraction methods and draw their correlations in Figure 2.12.

From the model perspective, we see a evolution of deep learning models. The early model using the deep belief model to pretrained the neural networks which predict the HMM states, and it is a generative model. Later the CNN models are largely used to extract the features, showing ability dealing with speaker variance and feature locality, the feature extracted are always connected to neural networks to do discriminate tasks. While RNN models focus on the temporal information lying on the speech, it try to model the temporal dependencies in speech recognition tasks. The latest deep learning ASR models are trying to combine the CNN and RNN together in order to take the advantage of the CNNs and RNNs.

From the phone error rate perspective, we could notice that the phone error rate keeps decreasing on TIMIT dataset, from the 23% in Deep belief networks pre-trained DNN(Mohamed et al., 2009), then 20% phone error rate in hybrid convolution neural networks and deep learning models (Abdel-Hamid et al., 2012) and 18.07% (Graves, Jaitly, & Mohamed, 2013) in end-to-end RNN system, and 18.2% in end-to-end CNN (Zhang et al., 2017). The word error rate is also keep dropping, and the latest word error rate on large vocabulary task is 5.1% (Bae et al., 2016). All of these models also have shown their capability of large vocabulary tasks.

In the feature domain, we could notice that there are some significant changes. First is that MFCCs are no longer the main feature extraction technique, various technique such as STFT, mel-scale frequency filterbanks are used in deep learning models, the decorrelation of the spectro-temporal feature is no longer needed. Second that the frequency resolution of the input data are becoming higher and higher, from th11e 13 dimensions to 40 dimensions later even 80 dimensions. With the development of computing power and deep learning models, there is a trend to use higher frequency resolution data in case of the loss of informations during feature technique.

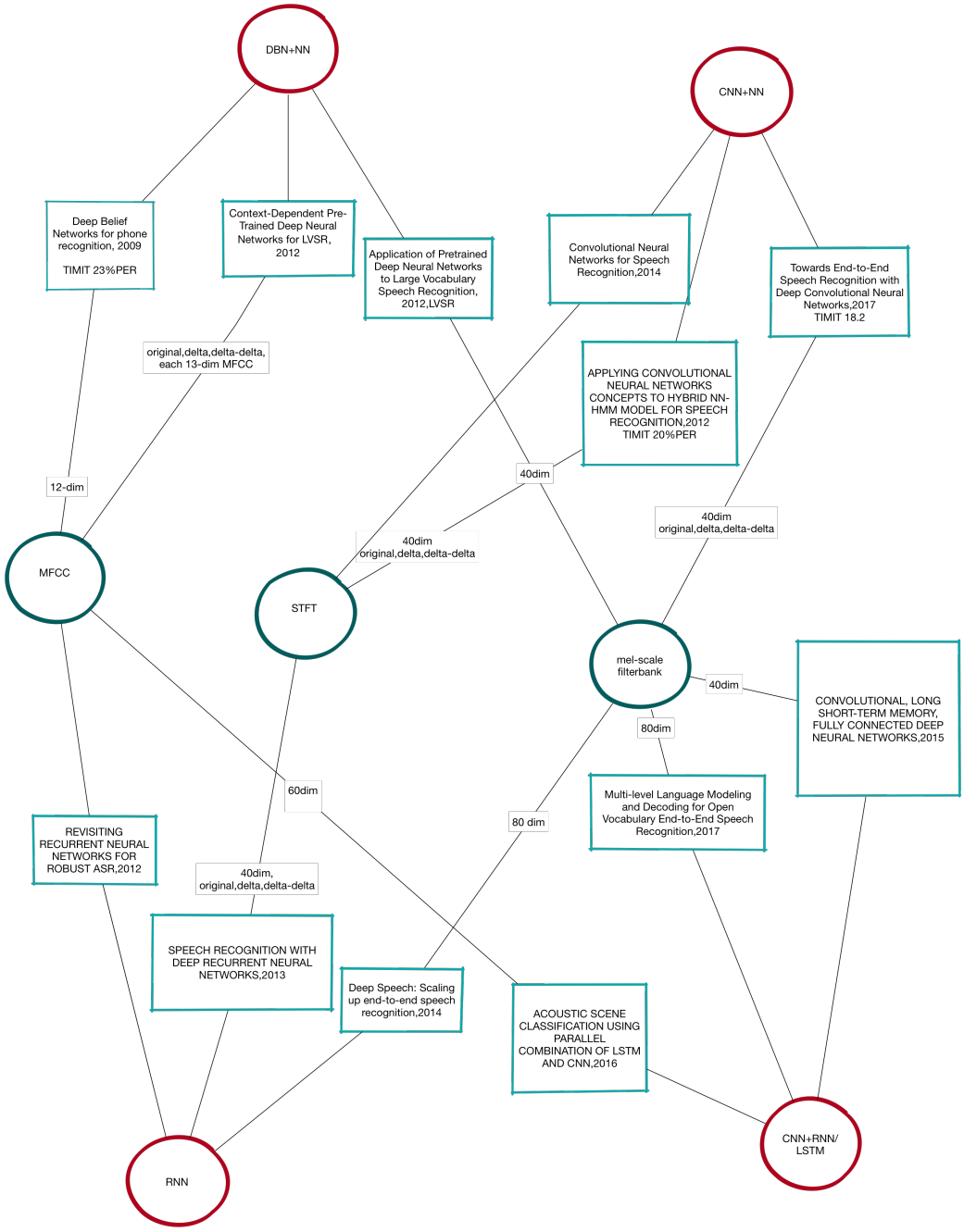


Figure 2.12: A summary of DNN models and their feature extraction methods

3

Front-end Feature Extraction Methods

In this section, we first introduce our dataset–Google Speech Commands Dataset . In our work, we focus on the influences of acoustic features in speech recognitions, and explore the CNN ability of dealing environmental variance, we will not investigate the language model and lexicon. We choose this open-source, word-to-word, contributed by thousands of people in various scenarios dataset for our thesis.

The second section is about the silence removal method. We find there are silences in audio files, and by applying the simple silence removal method, we remove silence and reduce the time dimension.

The third Section introduces the method of calculating the spectrogram of STFT (Short Time Fourier Transform) and the section four talks about gammatone filters and GFSCs (gammatone filterbanks spectral coefficients) which is the output of a set of gammatone filterbanks.

In the last section, we give a comparison between GFSCs and STFT feature extraction methods.

3.1 Google Speech Commands Dataset Introduction

The Google TensorFlow and AIY[1] teams released Google speech commands dataset(Warden, 2017) in August, 2017. It is a free open source speech dataset released under a Creative Commons 4.0 license. The dataset has 64728 one-second long utterances of short words, recorded by thousands of people. This dataset contains background noise and is spoken by a large number of people, which makes it suitable to make a robust acoustic models in the real world. This section briefly presents some details of the dataset.

This dataset contains 64728 .wav format audio files and they are organized into labeling folders. Random ids are assigned to each individual and encoded in each file name as the first part before the underscore. For example, the file path *right/9a8d8d34_nohash_0.wav* indicates that this utterance was spoken by userID *9a8d8d34* and the word spoken word is *right*. We write a python script to count the total utterance and number of speakers for each word. The statistic details are shown in table 3.1. We also summarize general information of the dataset, and it is listed below.

- This dataset is almost balanced with each category contains 1700-2400 utterances, and each word is spoken by more than 1100 unique users which ensure the speaker variability. The details could be found in table 3.1

- The entire speech command dataset is spoken by 1888 people. Over 80% contribute around 25 utterances.

- The original audio files were collected in uncontrolled locations with people from all over the world with variety of genders, ages, accents and signal-noise ratio (Warden, 2017).

- The sample frequency of the audio files is 16kHZ.

[1]Do-it-yourself artificial intelligence `https://aiyprojects.withgoogle.com/`

Table 3.1: Google Speech Command Dataset Details

Word	**total utterance**	**number of speaker**
bed	1713	1177
bird	1731	1194
cat	1733	1180
dog	1746	1209
down	2359	1206
eight	2352	1179
five	2357	1181
four	2372	1194
go	2372	1182
happy	1742	1178
house	1750	1165
left	2353	1180
marvin	1746	1191
nine	2364	1182
no	2375	1203
off	2357	1170
on	2367	1198
one	2370	1179
right	2367	1194
seven	2377	1192
sheila	1734	1178
six	2369	1197
stop	2380	1191
three	2356	1202
tree	1733	1173
two	2373	1175
up	2375	1186
wow	1745	1181
yes	2377	1200
zero	2376	1186

3.2 Silence Removal Method

The original audio file is 1s long and we reduce the time dimensions by removing the preamble and trailing parts which contain silence (and as well as background noise).

The method is based on the root-mean-square energy. By calculating the root-mean-square in a given moving time window, we keep the section with the highest square root energy. The algorithm is described below.

audio is the audio file, the *window* is the size of the audio kept, *step* is the sliding window moving step. This algorithm keeps calculating the sum of square-root-energy $\sqrt{Energy}$ and returns the indexes of the window with largest sum of square-root-energy[2].

function REMOVE_SILENCE($audio, window = 10000, step = 1000$)
  $start \leftarrow 0$
  $end \leftarrow 0 + window$
  while $end < audioLength$ **do**
    $\sqrt{Energy}[] \leftarrow sum(\sqrt{audio[start : end]^2})$
    $start \leftarrow start + step$
    $end \leftarrow end + step$
  $maxWindow \leftarrow [argmax(\sqrt{Energy}) * step, argmax(\sqrt{Energy} * step + window]$
return $audio[maxWindow]$

After several tests, we finally choose a 0.7s (sampling frequency = 16kHZ) window with moving step 0.0625s. The figure 3.1 gives an silence removal demo. The orange line is original signal and blue line is the window we select.

3.3 STFT: Short Time Fourier Transform

The speech signal is quasi-stationary signal and could be seemed as stationary data in a enough short time window. The Short Time Fourier transform is doing Fourier Transform in a short time window.

The Discrete Fourier transform in a small window is given by

$$X(n, \omega) = \sum_{m=-\infty}^{\infty} x[m]w[n-m]e^{-j\omega n}$$

[2]The reason why use square-root-energy instead of energy: In speech, the unvoiced phonemes such as 't','k' have a smaller amplitude than voiced phonemes such as 'a','o'. If we use the sum of energy, the unvoiced part will become even less significant and we may remove the unvoiced phonemes

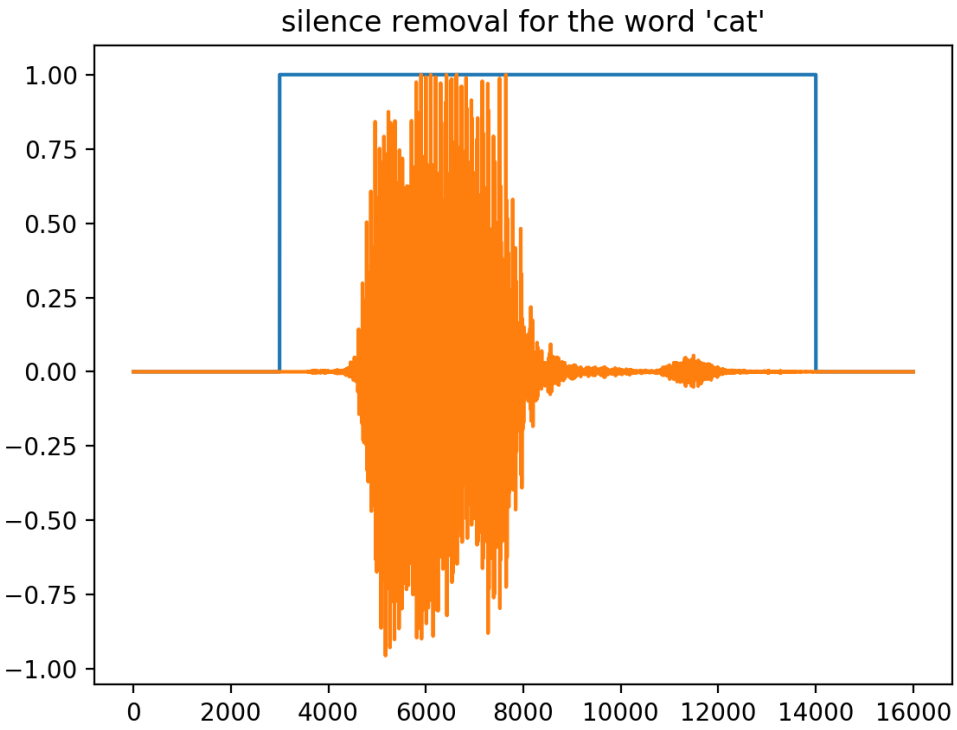


Figure 3.1: A silence remove demo using square-root-energy

Where $x(m)$ is the signal and w is the window function. ω denotes the frequency.

By discreting frequency ω, appy the log to the absolute value, we get the discrete STFT :

$$STFT(n,k) = log|X(n,\omega)|_{\omega=\frac{2\pi}{N}k}|$$

The short-time Fourier transforms of the speech signals are computed using *python scipy*(Jones, Oliphant, & Peterson, 2014) package. The time window is 12.5ms with 3.125ms overlap. In frequency domain, the speech recognition tasks often do not use the frequency above 6000HZ and we discard the information above 6000HZ. So the final dimension of the frequency and time is $76 \cdot 75$. The spectrograms have a high resolution in both time and frequency domain compare to most of the feature extraction methods in related work , which maintains more information.

The figure 3.2 gives a example of STFT transform for the word 'cat'.

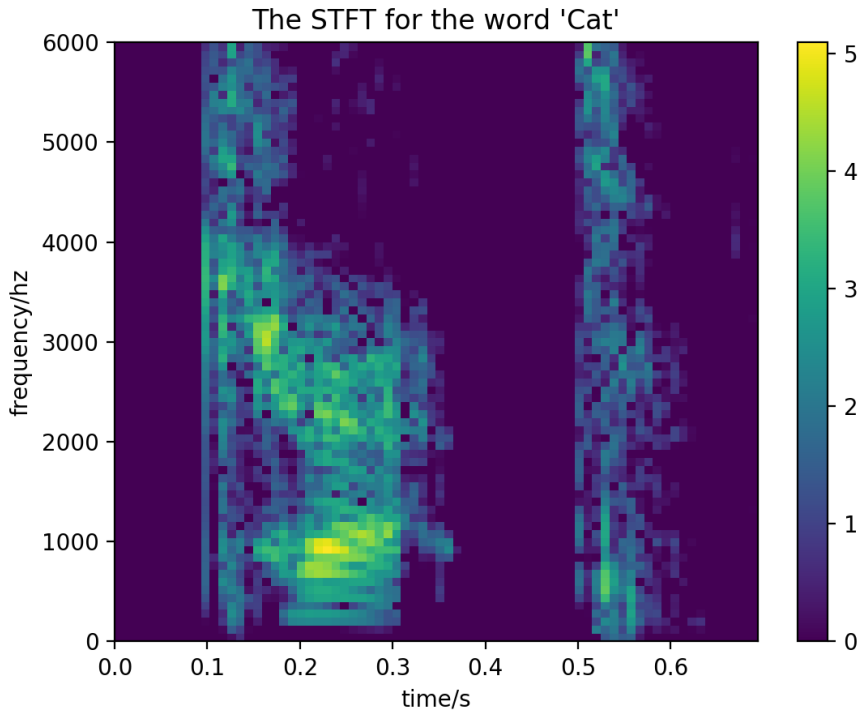


Figure 3.2: STFT representation for word 'cat'

3.4 GFSC: GammaTone Filterbank coefficients

GFSCs

The GammaTone filterbank coefficients are inspired by human auditory perception. The human auditory system shows an impressive ability in recognizing speech, where the cochlea plays an important role. There are three characteristics of cochlea : non-uniform filter bandwidths, asymmetric frequency response of individual filters and level-dependent frequency response of individual filters (Park, 2003). More introduction about cochlea could be found in *Chapter 2, Section 2.1.3.*

The impulse of a gammatone filter with a center frequency f_c at timestamp t is given by :

$$g_t(f_c, t) = a \cdot t^{(n-1)} \cdot e^{(-2\pi b \cdot ERB(f_c)t)} \cdot cos(2\pi f_c t)$$

where constant a controls the gain, n defines the order(usually we use 4-th order) of the gammatone filter and f_c is the center frequency and the ERB is the equivalent rectangular bandwidth of the center frequency. The b is associated with the stimulus level.

The ERB(f) is given by

$$ERB(f) = 0.1039f + 24.7$$

.

The impulse response of GammaTone is shown in Figure 3.3. The Re(z) and Im(z) indicate the value in real part and imaginary part.

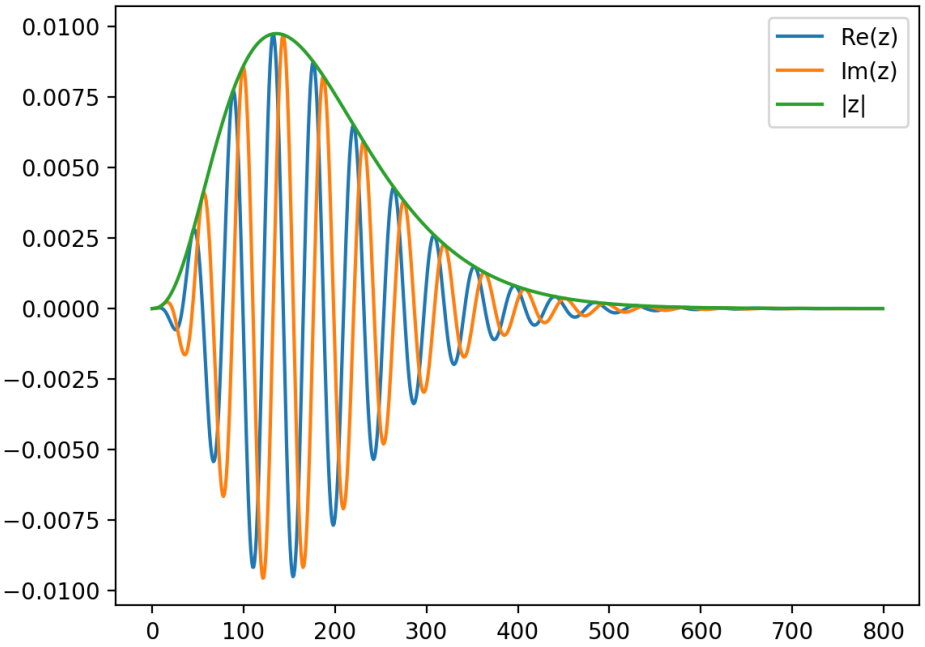


Figure 3.3: The Impulse Response of GammaTone Filter

The center frequencies f_c are chosen along the mel frequency scale. The mel-scale is a perceptual scale, which was developed by experimenting with the human ears interpretation of a pitch by Steven and Volkmann (Stevens, Volkmann, & Newman, 1937). The pitch is almost linear perceived in the frequency range 0-1000HZ, above the 1000hz, the scale becomes logarithmic.

The relation of Mel frequency scale and frequency could be approximated by

$$F_{Mel} = q \cdot log(1 + \frac{F_{HZ}}{q \cdot l})$$

$$l = 24.7, q = 9.265(Hohmann,2002)$$

The magnitude frequency responses of GammaTone filter bank are shown in 3.4.

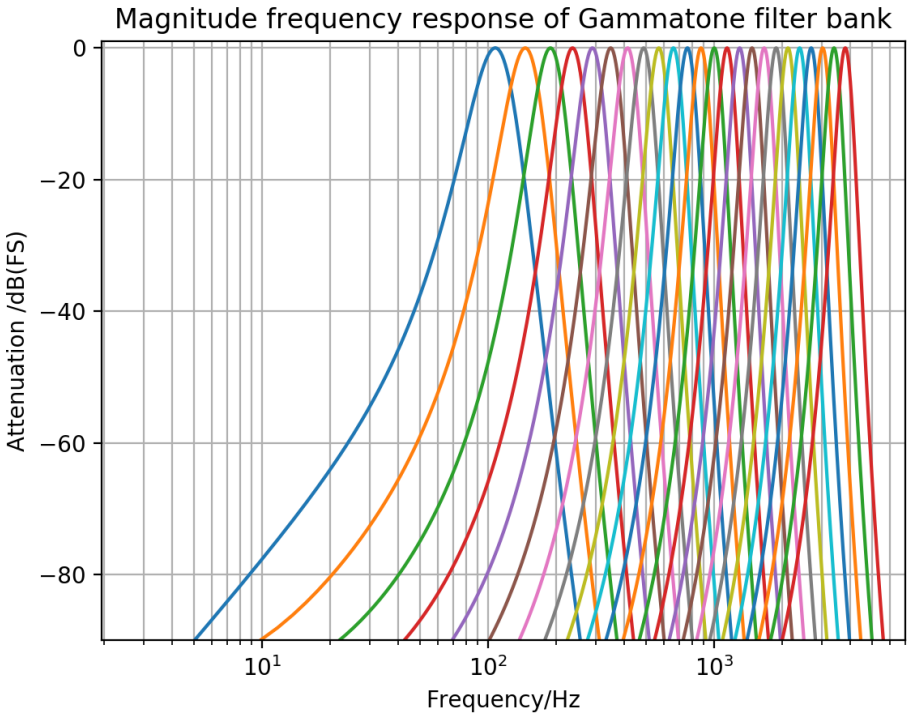


Figure 3.4: The magnitude frequency responses of GammaTone filter bank

The response $X(c, t)$ of the gammatone filter for input signal $x(t)$ is given by

$$X(c, t) = x(t) * g(f_c, t)$$

where "$*$" denotes the convolution computation in time domain (Gao, Woo, & Dlay, 2013). The dimension of $X(c, t)$ is $length\_x(t) \cdot num\_channles$, we choose 75 channels and apply a 12.5ms time window with 3.125ms overlap,

so the final dimension for the frequency and time is $76 \cdot 15$, which is the same as that in STFT in the previous chapter. And the GFSCs are computed using the python filterbank library(iegfried GÃijndert, 2014).

The figure 3.5 gives an GFSC example for the utterance 'cat'.

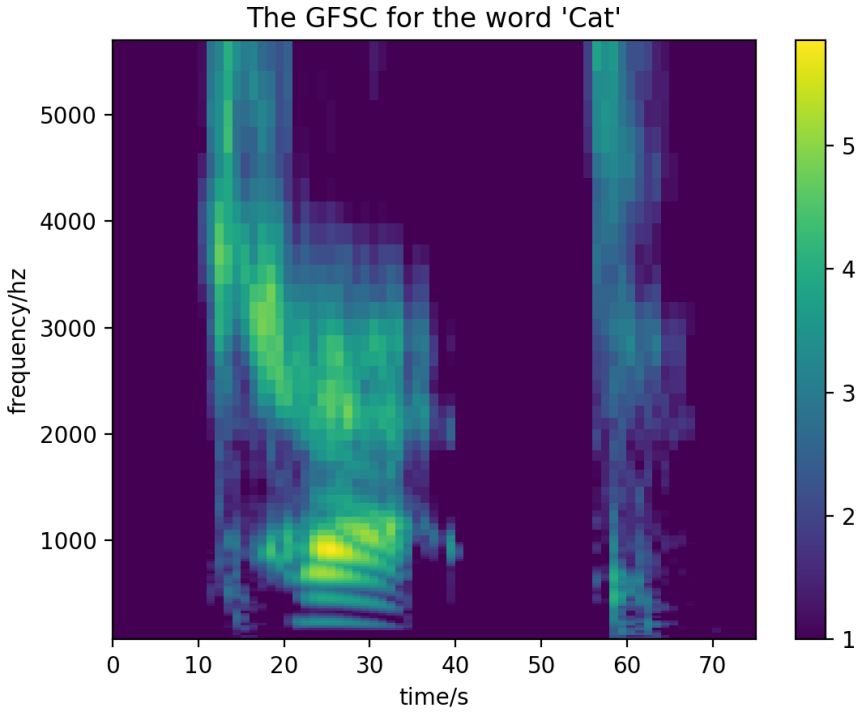


Figure 3.5: GFSC representation for word 'cat'

3.5 Comparison of STFT and GFSC

Time Comparison

The Fourier Transform is used in a lot of areas and there are already a lot of library support fast and efficient computation of Fourier transform, while the GFSCs computation do not have library that support fast and efficient computation, and the GFSCs method needs to calculate the output for each filterbank and then apply the window function to reduct the dimension. Therefore the GFSCs needs more computations.

We test the speed of calculations for the STFT and GFSCs for our whole dataset (64728 0.7s long utterances) under the same condition (VUB-ULB hydra sever with 4 cores). The computation time comparison could be found in table 3.2. From the table, we could found that the GFSCs method is slower than the STFT, but it is still tolerable for computing 64728 samples within 1 hour.

Methods	**Exact Time**	**Speed**
STFT	96s	x1
GFSCs	3196.8s	x33.3

Table 3.2: Computation Time Comparison for STFT and GFSCs

STFTs and GFSCs in different SNR scenarios

The Figure 3.6 shows the STFT spectrogram and gammatone filterbank for the utterance 'cat' in different SNR(Signal Noise Ratio) scenarios. The noise we add is white noise. The first difference is the frequency resolution, the GFSCs has higher resolution in lower frequency, and we could spot the formants more easily in GFSCs representation, the gap between the first formants and the second formants are wider in GFSCs. The second difference is that the GFSCs suffers less than STFT in noisy environments, there are less interferences in GFSCs compare that in STFT.

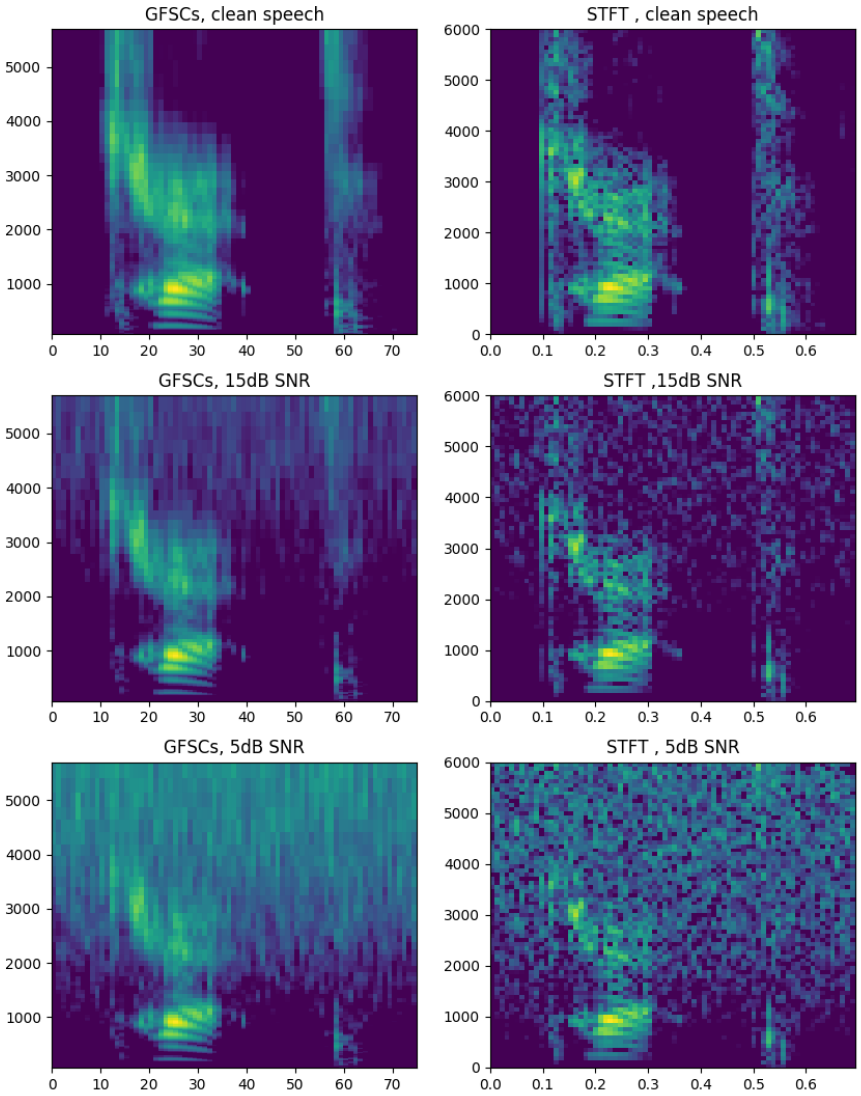


Figure 3.6: GFSCs and STFT representation for word 'cat' in different SNR(signal-noise ratio)

4

Deep Learning Neural Networks

Machine learning is a subfield of AI that learns patterns from data. Machine leaning techniques construct models from observed data using statistical techniques and allow computers to learn without being explicitly programed.

Deep learning is a subfield of Machine learning and it has caused revolutionary advances in machine learning and AI. Large datasets and strong computing power make it possible to train large neural networks. Compared to traditional machine learning algorithms, deep learning neural networks present a better performance with more features, bigger datasets and stronger computing power.

Convolutional Neural networks are biologically-inspired variants of MLPs (Multi layer perceptrons). They are made of neurons that have learnable weights and biases. Convolutional neural networks introduce a spatial network structure, and their **local connectivity, weight sharing and pooling** make them suitable to dealing with high dimension data and this section also briefly introduces the characteristics of CNNs.

4.1 Convolutional Neural Network overview

Convolutional neural network (CNN or ConvNet) is a deep, feed-forward neural network. CNNs were inspired by the visual cortex. The cortical neurons respond to stimuli only in a restricted region of the visual field known as receptive field. The restricted regions are tiled to cover the entire visual field, and the cells in restricted region are called filters. Over the last years, CNNs have obtained a revolutionary success in image recognition, recommender systems and NLP(Natural Language Processing). Our work is inspired by the CNNs' success in image recognition. The scalogram, which is a time-frequency representation of speech signals, could also be treated like images in CNNs.

Convolutional neural networks are based on layers, and the convolutional layer, ReLU layer and pooling layer are the most important layers in CNNs. The layers are stacked together, performing convolutional operations and pooling operations to extract the features. Later the extracted features connect to full connected layers, doing the classification task.

Convolutional neural networks have the ability to be invariant to irrelevant variation of the input and preserve the relevant information(LeCun, Kavukcuoglu, & Farabet, 2010). A lot of CNNs applications in various kind of areas have proved that (Oyedotun & Dimililer, 2016). Despite of the success of CNNs, we still do not well understand the structure of CNNs. We do not know how to choose the depth of the layers, the order of convolutional layers and pooling layers, how do the CNNs extract the features and deal with the redundancies and how to set the parameters such as the quantities of convolutional layers' filters , the shapes of filters as well as the the shapes of pooling kernels. This is one of our research questions. We design experiments to explore these questions.

In this section, we discuss the CNNs especially the convolutional layers which perform the convolutional operations and the pooling layers which reduce the size of output. We give a simple CNN (2 convolutional layers and 1 pooling layer) example which does our speech recognition task, and explain the mathematical formulas. The task is to classify the spectrograms (shape $1 \cdot 76 \cdot 75$, see Chapter 3: Front-end Feature Extraction Method) into 30 categories. The structure of the CNNs can be found in Figure 4.1. In this Figure, the sizes of input and output for each layer are labeled.

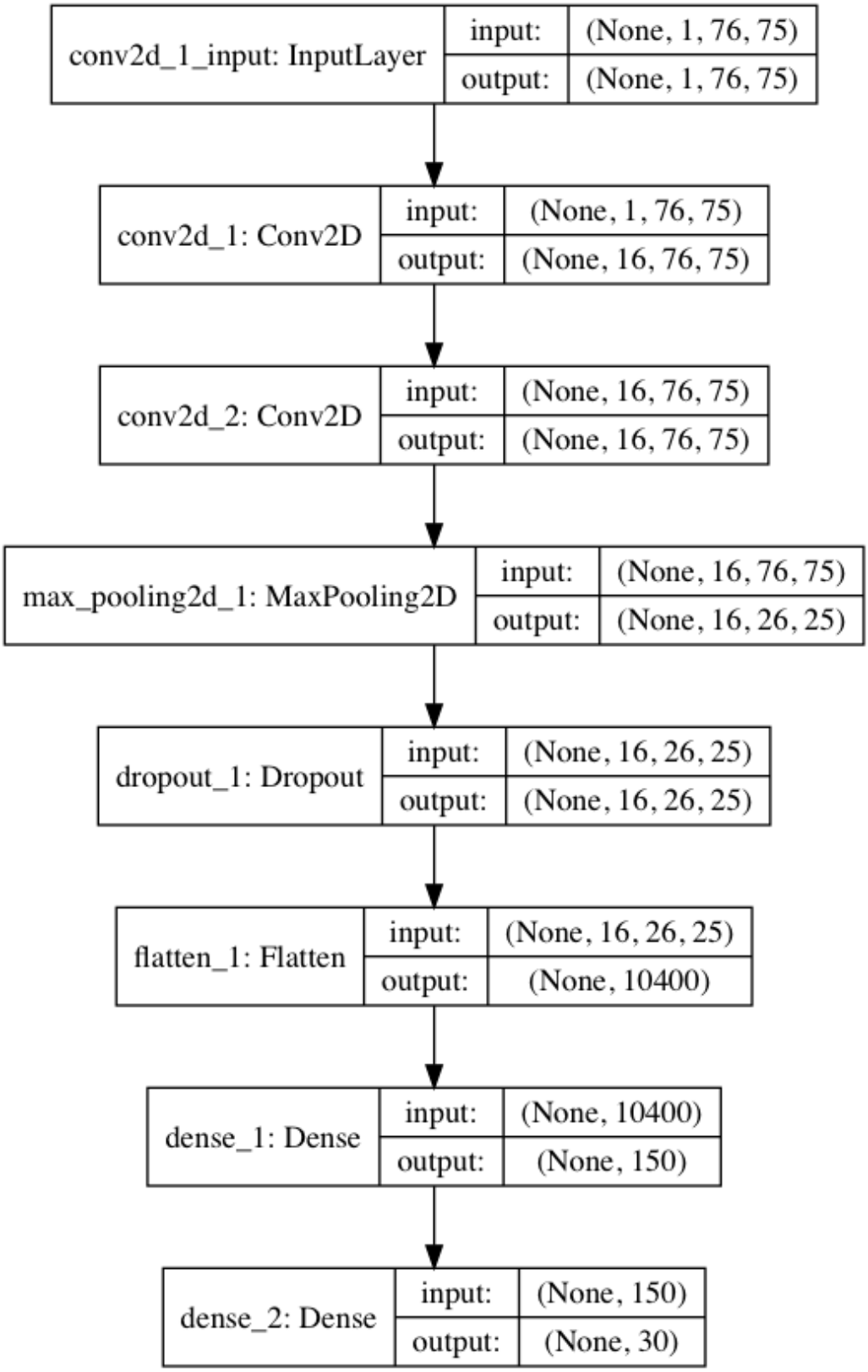


Figure 4.1: The structure of convolutional neural network used in this chapter

4.2 Convolutional layer

Convolution layers are the core layers which account for most of the computations in CNN. The CNNs are designed to deal with the 2D or higher dimension data with correlations among the features. The local filtering and weight sharing in convolution layer could deal with the spectral variations and spatial connections in speech signals. We discuss convolution layer's local filtering , weight sharing and activation function in this section.

2D Convolution

In signal processing and analysis, convolution is an important concept. The output of a system is constructed by convolving the input with impulse response (songhoAhn, 2008). In convolutional layer, the concept is very similar, we get the feature map by convolving input with its kernels. First we define the 2D convolution :

Input is $X \in \Re^{t \times f}$ where t, f are the shape of input. The kernel is $W \in \Re^{m \times n}$ where m, n are the shape of the kernel. The $*$ denotes the convolution operation and $\cdot$ denotes the multiply. The output Y is given by :

$$Y_{p,q} = X_{p,q} * W_{p,q} = \sum_{n1=0}^{t} \sum_{n2=0}^{f} X_{n1,n2} \cdot W_{p-n1,q-n2}$$

Convolutional Layer

Let layer l be a convolutional layer(Stutz, 2014), and layer l contain one or more filters. The input is the output of the front layer, and it may contain $m^{(l-1)}$ channels. We define our feature map as Y, and i^{th} feature map in layer l is denoted as $Y_i^{(l)}$. The $*$ denotes the convolution operation. So the feature map is given by :

$$Y_i^{(l)} = B_i^{(l)} + \sum_{j=1}^{m^{l-1}} K_{i,j}^{(l)} * Y_j^{(l-1)}$$

where $B_i^{(l)}$ is the bias matrix for i^{th} channel and $K_{i,j}^{(l)}$ is weight matrix of filter for the j^{th} channel.

The size of the feature map

The output feature maps' size are decided by the size of the kernel, the stride and the padding. The output size is given by:

$$Y_i^{(l)} \in \Re^{\frac{t-m+1+p}{s1} \times \frac{f-r+1+p}{s2}}$$

where t and f are the size in each dimension, $s1$ and $s2$ are the strides in each dimension, p is the padding size added to the input signal, m and r describe the shape of filter. In Figure 4.1, our simple convolutional neural network example, the first convolutional layer "Conv2D" has 16 filters, the strides is 1 and the padding is the same as the filter size. The input tensor shape is (None[1], 1, 76, 75), the output is (None, 16, 76, 75).

Weight Sharing

Each filter is replicated across the entire input. This replicated units share the same weight , bias and form a feature map. The convolution operation is spread across the time and domain. In order to simply this concept, we illustrate it with 1D vector. The Figure4.2 explains the weight sharing in 1 dimension. The input I has the size of 7×1 and the filter F size is 3×1. Different colors represent three different weights in the filter. Weight sharing is using the same weight vector to do convolution across the dimensions.

4.3 ReLU layer

ReLU (Rectified Linear Units) layer is activation layer. There are several commonly used activation functions such sigmoid $\sigma(x) = \frac{1}{1+e^{(-x)}}$and tanh $tanh(x) = 2\sigma(2x) - 1$. However, the sigmoid and tanh activation function suffer the gradients vanishing or exploding problems, not zero centered problems and also require a lot power of computing $e^{(x)}$. Glorot discussed these problems in (Glorot & Bengio, 2010). In 2010, Nair and Hinton(Nair & Hinton, 2010) first applied the ReLU units in training the RBMs (Restricted

[1]None means accepting any number in this dimension

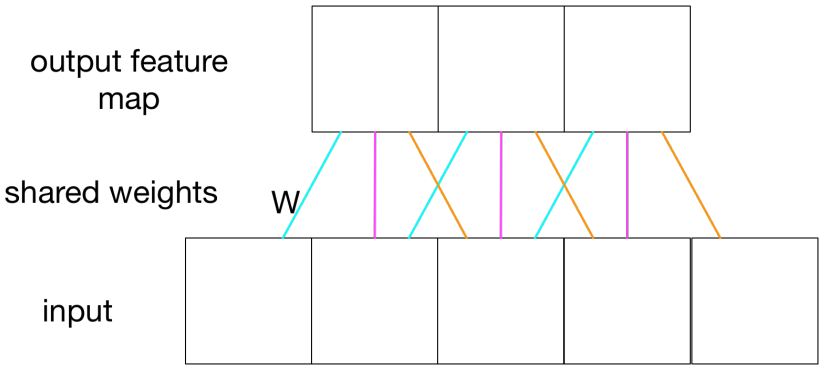


Figure 4.2: The weight sharing

Boltzmann Machines) and ReLU improved the result greatly. The ReLU has become the state-of-art activation function in CNNs. The ReLU function is simple, and it is given by:

$$f(x) = max(0, x)$$

The ReLU simply sets the threshold to 0. Using this simple activation has several advantages:

1. Computationally efficient.

2. Its derivation is constant when $x > 0$, so it does not saturate the gradient.

The Figure 4.3 shows the ReLU function and its derivation.

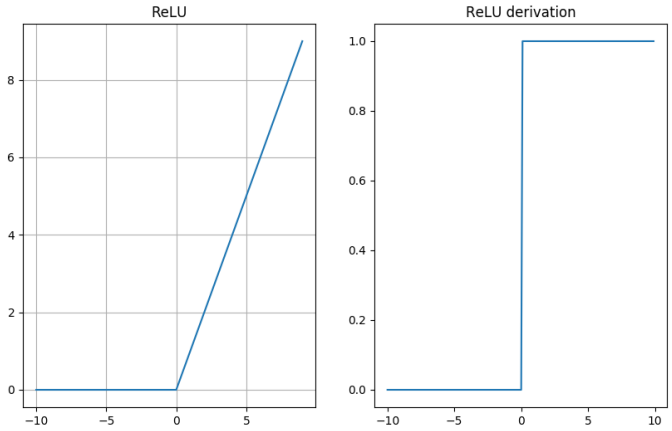


Figure 4.3: ReLU activation function and its derivation

4.4 Pooling layer

2D pooling

The pooling layer is a down sample layer which tries to extract the most significant features to represent spatial information. This technique reduces the computation complexity and make the feature map smaller. The pooling operations only take place on width and height and keep the depth the same. In general, the pooling layers place windows along the width and height and keep the averaged value or max value to represent this field(Stutz, 2014).

The pooling has many different kind of versions, such as maxpooling used most often, average pooling. Let l layer be the pooling layer, the maxpooling is given :

$$Y^l_{i,j,k} = max_{p,q \in P_{i,j}} z_{p,q,k}$$

where $P_{i,j}$ denotes the value matrices of Y^{l-1} in the window, and the average-pooling is very similar, except it uses the average value for all values inside the window. The average-pooling is given:

$$Y^l_{i,j,k} = \frac{\sum_{p,q \in P_{i,j}} z_{p,q,k}}{p1 \times p2}$$

where $p1$ and $p2$ are the shape of pooling window. The Figure 4.4(*CS231nCNN*, 2018) gives an pooling operation with 2×2 filter size with a stride of 2 on a single slice.

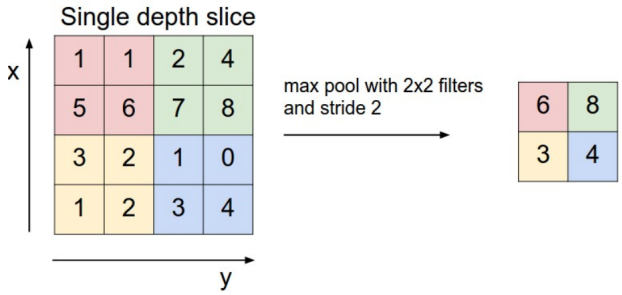


Figure 4.4: The pooling operation demo on a single slice

Output Size

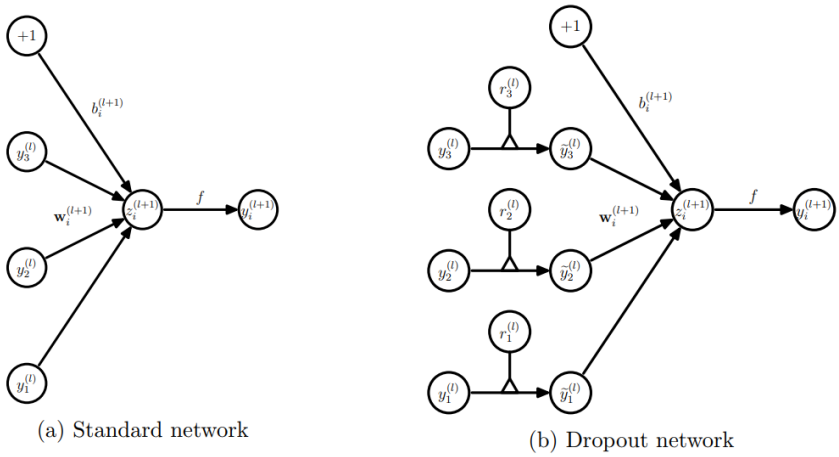
The pooling layer reduces the resolution along the height and width, and it does not change the depth of the input. The output size is very similar to the output of convolutional layer, and it is given by:

$$Y_i^{(l)} \in \Re^{\frac{t-m+1+p1}{s1} \times \frac{f-r+1+p2}{s2}}$$

where all symbols are the same with that in convolutional layer except the m and r denote the shape of pooling window. In Figure 4.1, our simple convolutional neural network example, the maxpooling layer pooling window is (3,3) and the stride is (3, 3), the input dimension of the pooling layer is (None, 16, 76, 75), and the output dimension is (None, 16, 26, 25).

4.5 Dropout layer

Large deep neural networks saturate from the overfitting problem. Dropout is a widely used technique to prevent overfitting. Srivastava and Hinton reviewed this simple technique to prevent overfitting in 2014(Srivastava, Hinton, Krizhevsky, Sutskever, & Salakhutdinov, 2014). Dropout prevents units co-adapting too much by randomly dropping units from neural networks during training. The Figure 4.5 (Srivastava et al., 2014) compares the standard network with the dropout network where $r_j^{(l)}$ is a selection function.



(a) Standard network (b) Dropout network

Figure 4.5: Comparisons of the standard network and dropout network.

Let l be the index of layers, let $Z^{(l)}$ be the input vector for the layer l, $y^{(l)}$ denote the outputs from the layer l. $W^{(l)}$ and $b^{(l)}$ denote the weights and biases at the layer l, the feed-forward operation is described by the equations below:

$$
\begin{aligned}
r_j^{(l)} &\sim Bernoulli(1-p), \\
\tilde{y} &= y^{(l)} * r^{(l)}, \\
z_i^{(l+1)} &= w_i^{(l+1)} \tilde{y}^{(l)} + b_i^{(}l+1), \\
y_i^{(l+1)} &= f(z_i^{(l+1)})
\end{aligned}
$$

where p is the dropout rate, $r^{(l)}$ follows the Bernoulli distribution and has probability $1-p$ of being 1. $\tilde{y}^{(l)}$ is the an element-wise product of $r_j^{(l)}$ and $y^{(l)}$. f is the activation function. Through the dropout function, we drop out some weights.

4.6 Flatten layer

The feature map extracted is a multi-dimension array, and the extracted features need to be connected to the dense layers which perform the discriminate task. The flatten layer is the intermediate layer which convert the high dimension feature map into 1-D vector. It is describes by:

$$Y_{m=i\cdot j\cdot k} = Z_{i,j,k}$$

where Y is the 1D output and Z is the multi-dimension input.

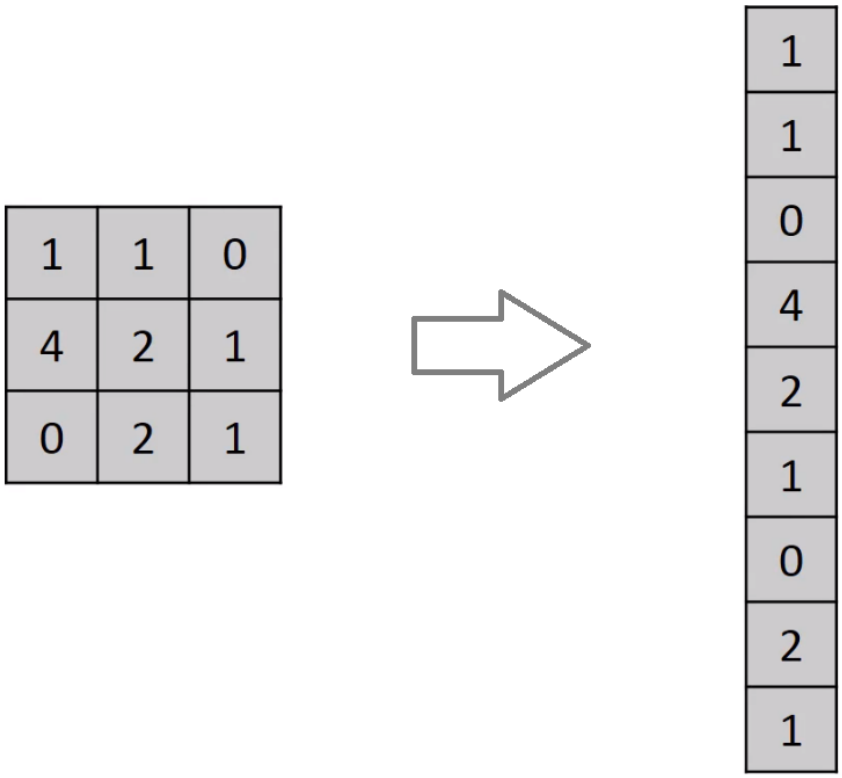


Figure 4.6: Flatten layer
(code project, 2018)

The Figure 4.6 (code project, 2018) shows the function of flatten layer.

4.7 Training Methods

This section introduces the training methods for convolutional neural networks. Here we only talk about the loss function, optimization function and evaluation metrics we used in our thesis.

Loss function

The training process is to find the smallest loss value and loss functions are used to guide the training process. Our task is a classification problem, and the most common used classification problem is MSE (Mean Square Error) and cross entropy. The MSE is given by:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - y_i')^2$$

which is the sum of square errors between the predictive value y_i and the ground truth label y_i'.

The cross entropy is given by:

$$H_{y'}(y) = -\sum_{i=1}^{n} y_i' log(y_i)$$

where y_i denotes the predictive value and y_i' denotes the ground truth label. In our training, we use the cross entropy as our loss function.

Optimization function

Optimization function is used to minimize the loss function. There are three kinds of the gradient descent, batch gradient descent, stochastic gradient descent and mini match gradient descent. Here we denote the loss function as $J(\theta)$. The batch gradient descent computes the loss function to the parameter θ for the whole dataset(Ruder, 2016), and it is described by:

$$\theta = \theta - \eta\nabla_\theta J(\theta)$$

The stochastic gradient descent updates the parameters for every training example, an it is given by:

$$\theta = \theta - \eta\nabla_\theta J(\theta; x^{(i)}; y^{(i)})$$

While the mini-batch gradient descent updates the parameter for every n examples

$$\theta = \theta - \eta\nabla_\theta J(\theta; x^{(i:i+n)}; y^{(i:i+n)})$$

When $n = 1$, the mini-batch gradient descent becomes stochastic gradient descent and when $n = num_samples$, it becomes the batch gradient descent.

Our dataset contains around 65000 examples. In our work, we choose the mini-batch gradient descent optimization with $n = 400$.

There are several commonly used optimization functions including, momentum, nesterov accelerated gradient, adagrad, adaDelta and adam. More details about the optimization function could be found in *"An overview of gradient descent optimization algorithms"* (Ruder, 2016). In our work, we use the adam (Adaptive Moment Estimation) optimization function.

m_t and v_t are estimates of the mean and uncentered variance of gradients g_t respectively.

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t,$$
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

Biases are counteracted by computing bias-corrected mean and variance.

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

The update rule for adam is described by:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t + \epsilon}} \hat{m}_t$$

5

TensorFlow and Keras

There are quite a few deep learning frameworks, TensorFlow developed by Google Brain team, Caffe developed by Berkeley AI research, MxNet supported by Amazon and Microsoft and etc. Our work mainly uses the TensorFlow and Keras. This chapter serves a simple tutorial for TensorFlow and Keras new users. We give an simple demo of constructing a CNN model from scratch using both TensorFlow and Keras.

This chapter is just a simple tutorial, and we can not cover all the characteristics of TensorFlow and Keras. For the TensorFlow section, we give an introduction to TensorFlow graph and also codes for the demo model in chapter 4 from scratch. The Keras is build on the top of TensorFlow, but it is more compact than the TensorFlow. We also provide the Keras code to implement the demo model in chapter 4.

TensorFlow(Abadi et al., 2018) is an open source software framework for numerical computation using **dataflow graphs(also called computation graph**. **Nodes** represent mathematical operations, while **graph edges** are multidimensional data arrays (tensors) which represent the communications between the Nodes. The TensorFlow kernel is "TensorFlow Distributed Execution Engine" which enables parallel computations to all computation powers such CPUs and GPUs in a simple API , without writing additional multithreading, multi-processing and CUDA code for deployment and parallelism. TensorFlow is a cross platform framework which supports several programming languages including Python, Java, C++, Go. More technical details about TensorFlow Programming stack, general architecture, the parallelism and other technical details could be found `https://www.tensorflow.org/extend/architecture` and `https://www.tensorflow.org/deploy/`. We will not discuss the technical background because we could still construct the machine learning models through simple APIs and without knowing these complicated mechanism. We will focus on the art of constructing models —-representing computations in a dataflow graph .

Keras(*Keras*, 2018) is a high-level neural networks API running on top of TensorFlow, CNTK or Theano. It is written in Python and allows for fast implementation of Deep Neural Networks through its user-friendliness, modularity and extensibility. This framework

These two deep learning frameworks are used in my master thesis work. The following section will give a small introduction for this two libraries and more tutorials can be found on the TensorFlow and Keras official website (*TensorFlow*, 2018) and (*Keras*, 2018).

5.1 TensorFlow Brief Introduction

5.1.1 Explanation of Terms

In TensorFlow(*TensorFlow*, 2018), computations are represented in a computation graph. This graph is a directed graph, where **nodes** represent operations and **edges** represent data flow between operations. **Tensors** are data arrays. Tensors flow between operations and that's why this framework called TensorFlow. The computation graph's component tensors and operations will be introduced in this section, and also we will introduce how to construct the dataflow graph and initialize the session briefly. We include

some python codes to explain dataflow graph components and to show how to start building models following the TensorFlow programming art.

A. Tensor

Tensors are generalizations of vectors and data arrays which have higher dimensions. A tensor has a data type(for example,float32, int32, or string) and a shape and each element in the Tensor has the same data type. They are the primary data structure and we can declare these tensors as variables or feed them in as placeholders. Two types of tensors are key tools in TensorFlow Programming, and they are listed below

- tf[1].Variable: the value of Variable is changeable, and they are always used to declare the weights in networks. We use the tf.Variable() to declare a tensor, later we need to initialize it. Initializing is puts the variable with the corresponding methods on the computational graph.

- tf.placeholder:the value of placeholder is immutable, it is always used to feed data with a specific data type and shape. A placeholder must be fed first before Use.

B. Operation

Operation takes tensors as input and do some operations on tensors then output an tensor. The table5.1 (Buduma & Locascio, 2017) below summarize basic operations.

C. Dataflow Graph

TensorFlow uses a dataflow graph to represent computations in terms of the dependencies between individual operations. The figure**??** gives an example of dataflow graph, each node is an operation. Dataflow graph is a common programming model for parallel computing.

[1]tf represents TensorFlow

Table 5.1: TensorFlow Operation Summary

Category	Examples
Basic mathematical Ops	add, sub, mul, div, exp, log, greater, less, equal ...
Array Ops	concat, split, slice, constant, rank, shape, shuffle...
Matrix Ops	matmul, matInverse, matTranspose...
Stateful Ops	Variable, Assign, assignAdd ...
Neural Network Building Blocks	SoftMax, ReLu, MaxPool, Convolution2D ...
Checkpointting Ops	save, restore ...
Control flow Ops	Merge, Switch, Enter, Leave, NextIteration ...

D: Session

Session is a management tool for TensorFlow. A graph defines the computation, and it does not computing anything. After creating a dataflow graph, create a TensorFlow Session to execute the parts of the graphs. A session allocates the resources and does the computations. Below is a piece of code that running computation in a Session.

5.1.2 Coding for the Model in Chapter 4

Here we draw a figure to show the steps to build a CNN model and train the model. We implement model in Chapter 4, the model structure is shown in 4.1. The basic steps to coding with TensorFlow is building the graph, the prepare for the training and train the model. The basic steps are shown in 5.1. We code for this model from scratch. We modified the code from `https://github.com/aymericdamien/TensorFlow-Examples/blob/master/notebooks/3_NeuralNetworks/convolutional_network_raw.ipynb`

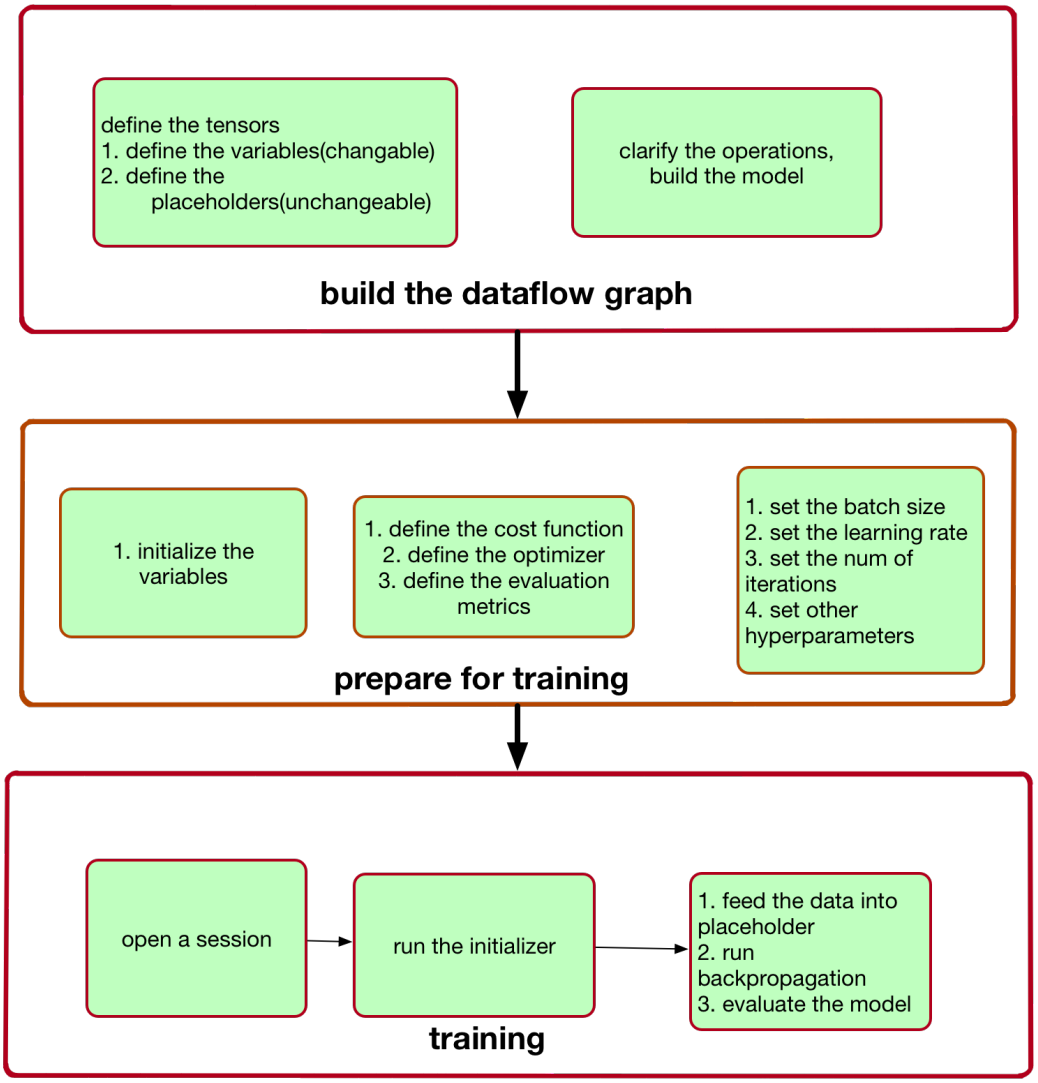


Figure 5.1: Build a model with TensorFlow

Build the dataflow graph

The piece of code below is showing how to build the dataflow graph. We first define the weights and biases as variables, which is changeable. And we define the placeholders for our dataset. We need to specify the size for variables and placeholders.

Then we build our models with layers, clarify the operations.

```
import tensorflow as tf
```

```
weights = {
    # 3x3 conv, 1 input, 16 outputs
    'wc1': tf.Variable(tf.random_normal([3, 3, 1, 16])),
    # 3x3 conv, 16 inputs, 16 outputs
    'wc2': tf.Variable(tf.random_normal([3, 3, 16, 16])),

    # fully connected,
    'wd1': tf.Variable(tf.random_normal([16*26*25, 150])),
    # 150 inputs, 30 outputs (class prediction)
    'out': tf.Variable(tf.random_normal([150, num_classes]))
}

biases = {
    'bc1': tf.Variable(tf.random_normal([16])),
    'bc2': tf.Variable(tf.random_normal([16])),
    'bd1': tf.Variable(tf.random_normal([150])),
    'out': tf.Variable(tf.random_normal([num_classes]))
}

X = tf.placeholder(tf.float32, [None,1,76,75])
Y = tf.placeholder(tf.float32, [None, num_classes])

def conv2d(x, W, b, strides=1):
    # Conv2D wrapper, with bias and relu activation
    x = tf.nn.conv2d(x, W, strides=[1, strides, strides, 1],
                                    padding='SAME')
    x = tf.nn.bias_add(x, b)
    return tf.nn.relu(x)

def maxpool2d(x, k=2):
    # MaxPool2D wrapper
    return tf.nn.max_pool(x, ksize=[1, k, k, 1], strides=[1,
                                    k, k, 1],
                          padding='SAME')


def buildGraph(x, weights, biases, dropout):

    # Convolution Layer
    conv1 = conv2d(x, weights['wc1'], biases['bc1'])
    conv2 = conv2d(conv1, weights['wc2'], biases['bc2'])
    # Max Pooling (down-sampling)
    maxpool1 = maxpool2d(conv2, k=3)
    maxpool1 = tf.nn.dropout(maxpool1, dropout)
    # Fully connected layer
    # Flatten feature map  to fit fully connected layer input
    fc1 = tf.reshape(conv2, [-1, weights['wd1'].get_shape().
                                    as_list()[0]])
    fc1 = tf.add(tf.matmul(fc1, weights['wd1']), biases['bd1'
```

```
                                    ])
    fc1 = tf.nn.relu(fc1)
    # Output, class prediction
    out = tf.add(tf.matmul(fc1, weights['out']), biases['out'
                                    ])
    return out
```

Prepare for training

As shown in graph, we first initialize the variables, then define the cost function, optimizer and evaluation metrics for our model, we also specify the hyperparameters include the learning_rate, number of steps, batch size and dropout rate.

```
# Initialize the variables (i.e. assign their default value)
init = tf.global_variables_initializer()

logits = conv_net(X, weights, biases, keep_prob)
prediction = tf.nn.softmax(logits)

# Define loss and optimizer
loss_op = tf.reduce_mean(tf.nn.
                                softmax_cross_entropy_with_logits
                                (
    logits=logits, labels=Y))
optimizer = tf.train.AdamOptimizer(learning_rate=
                                learning_rate)
train_op = optimizer.minimize(loss_op)


# Evaluate model
correct_pred = tf.equal(tf.argmax(prediction, 1), tf.argmax(Y
                                , 1))
accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))

learning_rate = 0.001
num_steps = 500
batch_size = 128
display_step = 10
dropout = 0.25
```

Training

The training process starts with a session. We feed the data into out model. We use a for loop to train the model iteratively.

```
with tf.Session() as sess:

    # Run the initializer
    sess.run(init)

    for step in range(1, num_steps+1):
        batch_x, batch_y = X.train.next_batch(batch_size)
        # Run optimization op (backprop)
        sess.run(train_op, feed_dict={X: batch_x, Y: batch_y,
                                      keep_prob: dropout})
        if step % display_step == 0 or step == 1:
            # Calculate batch loss and accuracy
            loss, acc = sess.run([loss_op, accuracy],
            feed_dict={X: batch_x,Y: batch_y,keep_prob: 1.0})

     sess.run(accuracy, feed_dict={X:X['gfsc'][test],
                                   Y:X['label'][test] ,
                                   keep_prob: 1.0})
```

5.2 Keras Brief Introduction

Keras is a python library that run on top of Theano, CNTK or TensorFlow. It minimizes the number of user actions required for common use cases. It makes implementing deep learning models as fast and easy as possible for research and development.

The steps towards implementing the Keras are more concise, and they are also divided into the building model, preparing for training and training part.

Build the model and prepare for training

Compare the building model and comparing model to TensorFlow, Keras provides a more simple and concise way to implement it. Keras do not need to define the variables and placeholders for weights and dataset. Defining

the optimizer, loss function and evaluation metrics are also integrated into one function.

```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import Flatten
from keras.layers.convolutional import Conv2D
from keras.layers.convolutional import MaxPooling2D

def buildGraph(shape,num_classes=30,cnnfsize=(3, 3), poolsize
                                =(3, 3),dense=150):
    model = Sequential()
    model.add(Conv2D(16, cnnfsize, padding='same', activation
                                    ='relu', input_shape=shape
                                    ,
                      data_format="channels_first"))
    model.add(Conv2D(16, cnnfsize, activation='relu',
                                    data_format="
                                    channels_first", padding='
                                    same'))
    model.add(MaxPooling2D(poolsize, data_format="
                                    channels_first", padding='
                                    same'))
    model.add(Dropout(0.25))
    model.add(Flatten())
    model.add(Dense(dense, activation='relu'))
    model.add(Dense(num_classes, activation='softmax'))
    # Compile model
    model.compile(loss='categorical_crossentropy', optimizer=
                                    'adam', metrics=['accuracy
                                    '])
    return model
```

Training

The training process for Keras are simpler. In Keras, there is no concept of sessions. Just build the model, and then start the training process by using fit function.

```
model = buildGraph(shape=(1,76,75))
X_train, X_test, y_train, y_test, path_train, path_test =
                                train_test_split(Dataset.X,
                                Dataset.y, Dataset.path,
                                test_size=0.3, random_state=
                                seed)
```

```
model.fit(X_train, y_train, validation_data=(X_test, y_test),
                                  epochs=250, batch_size=200,
                                 verbose=2)
scores = model.evaluate(X_test, y_test, verbose=2)
```

5.3 Recommended books and tutorials

The best materials of learning deep learning frameworks are official released docs, and we also found some interesting YouTube videos for beginners.

Websites

- TensorFlow official docs: `https://www.tensorflow.org/api_docs/python/`
- Keras official docs: `https://keras.io/`
- Siraj Raval YouTube videos: `https://youtu.be/2FmcHiLCwTU`, I really recommend this YouTube tutorials about TensorFlow. It is a really interesting tutorial that build a model from scratch.

Books

- Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems(Géron, 2017).
- Deep Learning with Keras (Gulli & Pal, 2017)
- Python Deep Learning Cookbook (den Bakker, 2017)

6

Experiments: Settings, Results and Discussions

This chapter describes the detailed experimental settings and results. Our experiments are divided into two parts, the experiments towards understanding CNN models and the experiments of comparing our STFT and GFSCs feature extraction methods.

The first section of this chapter describes experiments about the CNN models. Here we define a concept called block. A block refers to 2 convolutional layers, 1 pooling layer and 1 dropout layer connected sequentially. The structure of our model is several blocks which extract the features, following by 2 dense layers which do the classification task. Each pooling layer followed by a dropout layer, and each dense layer in our model also followed by a dropout layer. We focus on three of the CNN parameters which we think are interesting: the depth of the CNN, the sizes of convolutional layers kernels and pooling layers kernel, and the number of hidden layer units in the first dense layer. We organize all these experiments in 1 block , 2 blocks, 3 blocks and 4 blocks model (the structure of the model can be found in Appendix A.2).

We are also interested in the front feature extraction method in ASR systems. We add different strengths of white noise to the audio and then extract the features using the STFT and GFSCs methods. We use a 3-block CNN model and evaluate the performance of feature extraction method using 8-fold cross-validation in this section.

6.1 CNN Model Building Experiments

Convolutional neural networks have the ability to be invariant to irrelevant variations of the input and preserve the relevant information(LeCun et al., 2010), and many applications of CNNs have proved that. But we still do not well understand underlying key mechanisms the of CNNs. There are some key factors that we should consider when designing the CNNs models.

- The number of convolutional layers and the number of pooling layers.
- The arrangement of convolutional layers and pooling layers.
- The number and the size of filters in convolutional layers.
- The number of dense layers and the hidden units in each dense layer.
- The position of dropout layers and the dropout rate for each dropout layer.
- The selection of cost function and the optimization method.
- ...

The performances of CNN models are contributed by many factors, and there are a lot possible models with various combinations of these factors. There are so many combinations and why it is hard to understand the CNN models. Understanding the CNN network is one of our research goals. We control variables and design experiments to explore the influences of these factors which we think are important. We can not explore all the combination of factors and we limit ourselves to the four factors which we think are essential in CNN models:

1. The depth of the CNN.

2. The size of kernels in pooling layers and convolutional layers.
3. The number of hidden units in dense layer.

And other variables are set to the same , and we list the settings below.

- feature extraction method: The GFSCs (GammaTone Filterbank Spectral Coefficients), the detail of the method could be found in chapter 3, section 4: GammaTone Filterbank Spectral Coefficients.
- cost functions: categorical cross entropy, the detail of this cost function could be found in chapter 4.
- optimization method: Adam gradient descent method.
- training epochs: 250
- size of the batch: 400
- number of the dense layers doing classification task: 2 dense layers after the flatten layer.
- ReLU layer: A ReLU layer is added to each convolutional layer.
- dropout layer: Each pooling layer and dense layer are followed by a dropout layer.

6.1.1 The depth of CNN

We analyze here the impact of the CNNs' depth. The CNNs are stacked by convolutional layers, pooling layers, ReLU layers and dropout layers. We construct different models with different number of layers and evaluate our model using the GFSCs features.

Experimental settings

- Block : We define 2 convolutional layers following a pooling layer and a dropout layer as a block.
- We stack the blocks sequentially, then connected to two dense layers to do the classification task.

- We explored a large set of CNN kernel size and pool size, we first start with the small kernel size (2,2) and pool size (2,2) and then increase the size gradually. We continue increasing the pool size and filter size until the error rate climbs or keeps steady.

- We test one Block, two blocks, three blocks and four blocks networks. We use the best result in each group to evaluate the impact of depth in CNN.

- Training a model needs a lot time, though we use the HPC to train our models, it is still unrealistic for us to do cross validation for all of our models. Here we divide our dataset into 70% training set and 30% test set. We use the same training set and test set to train our models and same test set to evaluate our models.

Results and Discussion

The Table 6.1, Table 6.2, Table 6.3 and Table 6.4 give the error rates for models with different depths and different kernel sizes. We only put the most significant results in these tables. For the one block model, the best result is around 6.7%, while the best result for two blocks model is around 4.1%, and the lowest error rate for three blocks model is around 3.9%, and for four blocks model, the best result is 4.0%. The overall performance of CNN models are first increased then slightly reduced with the increase of depth.

We can observe that both the lowest error rate and average error rate drop from one block to three blocks models, but the four blocks models get slightly worse results than the three blocks model.

The 1B6, 2B6, 3B4 and 4B4 yield the best results in corresponding depth, and among them the 3B4 is by far the best model in our experiments. Another interesting thing we found is that although the 1B6 and 2B6 have more parameters than the 3B4 models, they have worse performance. More parameters does not mean more powerful model.

Table 6.1: Error rates for one block models with different kernel size

oneBlock	**1B1**	**1B2**	**1B3**	**1B4**	**1B5**	**1B6**	**1B7**
Conv2D, 32	3x3	3x3	4x4	4x4	6x6	5x5	6x6
Conv2D,32	3x3	3x3	4x4	4x4	6x6	5x5	6x6
MaxPool	3x3	4x4	4x4	5x5	5x5	6x6	6x6
Dropout	0,3	0,3	0,3	0,3	0,3	0,3	0,3
Dense	150	150	150	150	150	150	150
Dropout	0,3	0,3	0,3	0,3	0,3	0,3	0,3
Dense	30	30	30	30	30	30	30
params	3.134.248	1.747.048	1.754.440	1.173.640	1.194.760	842.344	853.960
error	9,38%	8,48%	7,62%	7,35%	6,82%	6,64%	6,73%

Table 6.2: Error rates for one blocks models with different kernel size

twoBlocks	**2B1**	**2B2**	**2B3**	**2B4**	**2B5**	**2B6**	**2B7**
Conv2D,32	2x2	3x3	3x3	4x4	5x5	6x6	5x5
Conv2D,32	2x2	3x3	3x3	4x4	5x5	6x6	5x5
MaxPool	2x2	2x2	3x3	4x4	5x5	5x5	6x6
Dropout	0,25	0,25	0,25	0,25	0,25	0,25	0,25
Conv2D,64	2x2	3x3	3x3	4x4	5x5	6x6	5x5
Conv2D,64	2x2	3x3	3x3	4x4	5x5	6x6	5x5
MaxPool	2x2	2x2	3x3	4x4	5x5	5x5	6x6
Dropout	0,25	0,25	0,25	0,25	0,25	0,25	0,25
Dense	300	300	300	300	300	300	300
Dropout	0,2	0,2	0,2	0,2	0,2	0,2	0,2
Dense	30	30	30	30	30	30	30
params	6.969.522	7.005.522	1.629.522	604,722	419,922	362,322	499,122
error	8,19%	6,77%	5,30%	4,49%	4.14%	4.02%	4.29%

Table 6.3: Error rates for three blocks models with different kernel sizes

threeBlocks	**3B1**	**3B2**	**3B3**	**3B4**	**3B5**	**3B6**
Conv2D,32	2x2	3x3	2x2	3x3	4x4	5x5
Conv2D,32	2x2	3x3	2x2	3x3	4x4	5x5
MaxPool	2x2	2x2	3x3	3x3	3x3	3x3
Dropout	0,25	0,25	0,25	0,25	0,25	0,25
Conv2D,32	2x2	3x3	2x2	3x3	4x4	5x5
Conv2D,32	2x2	3x3	2x2	3x3	4x4	5x5
MaxPool	2x2	2x2	3x3	3x3	3x3	3x3
Dropout	0,25	0,25	0,25	0,25	0,25	0,25
Conv2D 64	2x2	3x3	2x2	3x3	4x4	5x5
Conv2D,64	2x2	3x3	2x2	3x3	4x4	5x5
MaxPool	2x2	2x2	3x3	3x3	3x3	3x3
Dropout	0,3	0,3	0,3	0,3	0,3	0,3
Dense	300	300	300	300	300	300
Dropout	0,3	0,3	0,3	0,3	0,3	0,3
Dense	30	30	30	30	30	30
params	1.966.578	2.012.818	219.378	265.618	330.354	413.586
error	5,88%	4,86%	4,85%	3,89%	3,91%	4,18%

Table 6.4: Error rates for four blocks models with different kernel sizes

fourBlocks	**4B1**	**4B2**	**4B3**	**4B4**	**4B5**	**4B6**
Conv2D,16	2x2	3x3	2x2	3x3	4x4	4x4
Conv2D,16	2x2	3x3	2x2	3x3	4x4	4x4
MaxPool	2x2	2x2	3x3	3x3	4x4	3x3
Dropout	0,2	0,2	0,2	0,2	0,2	0,2
Conv2D,32	2x2	3x3	2x2	3x3	4x4	4x4
Conv2D,32	2x2	3x3	2x2	3x3	4x4	4x4
MaxPool	2x2	2x2	3x3	3x3	4x4	3x3
Dropout	0,2	0,2	0,2	0,2	0,2	0,2
Conv2D,64	2x2	3x3	2x2	3x3	2x2	4x4
Conv2D,64	2x2	3x3	2x2	3x3	2x2	4x4
MaxPool	2x2	2x2	3x3	3x3	2x2	3x3
Dropout	0,2	0,2	0,2	0,2	0,2	0,2
Conv2D,64	2x2	3x3	2x2	3x3	2x2	4x4
Conv2D,64	2x2	3x3	2x2	3x3	2x2	4x4
MaxPool	2x2	2x2	3x3	3x3	2x2	3x3
Dropout	0,1	0,1	0,1	0,1	0,1	0,1
Dense	300	300	300	300	300	300
Dropout	0,1	0,1	0,1	0,1	0,1	0,1
Dense	30	30	30	30	30	30
params	554.258	594.018	247.058	286.818	172.754	268.754
error	6,09%	4,77%	5,42%	3,97%	4,49%	4,70%

6.1.2 The size of pooling layers and convolutional layers

Experimental Settings

The experiment settings are the same as the previous experiment, we have different filter sizes and pooling kernel sizes for models with different depths. We see the influences of the pooling kernel sizes and filter sizes in both 'shallow' networks and 'deep' networks.

Results and Discussion

The result are also given in Table 6.1, Table 6.2, Table 6.3 and Table 6.4, we analyze the results from the size of pooling layers and filters kernels perspec-

tive in this section.

From the Table 6.1, we find that the error rate first drops when increase the filter size and pool size, later the error rates slightly climb or keep relatively stable. We examine the combinations of pooling and filter kernel size from (2,2) to (6,6) and we see a downward trend from 9.38% to 6.73%. The best result is the 1B5, whose CNN kernel size and pooling kernel size are (5x5) and (6x6) respectively. And the two blocks CNN gets the best result with CNN filter size (5,5) and pooling size (4,4), while three blocks CNNs get the best result with CNN filter size CNN filter size (3,3) and pooling kernel size (3,3). When the CNNs increased their depths to 4 blocks, the 4B4 with CNN kernel size (3x3) and pooling kernel size (3x3) yield the best result. We notice that the deeper model get its best result in a smaller CNN filter size and pool size. Later error rates for one block, two blocks and three blocks model keep relatively stable, but for the four blocks model, the error rate climbs.

6.1.3 The number of Hidden Units in The Hidden Layer

Experimental Settings

The front-end CNN layers and pooling layers are used to extract the features, and they do not perform the classification task (Abdel-Hamid et al., 2014). The dense layers which the CNN layers and pooling layers are connected to are the layers to do the classification task. In this work, we connect the extracted feature to 2 dense layers to do classification task, and the number of hidden units in the final dense are set equal to the number of categories, while the optimal number of hidden units in the intermediate layer is our research question.

We design our experiments in different depths. For each depths, we have the fixed parameters for kernel sizes and dropout rates, we only change the number of hidden units in the dense layers.

Results and Discussion

The Table 6.5, Table 6.6, Table 6.7 and Table 6.8 present errors for models with different number of hidden units in the dense layer for one block model, two blocks, three blocks and four blocks models.

For one block model, there is a drop trend in the number of error rate with the increase of hidden units in the dense layer. Then the error stays around 6.2%. While for other models, the error rates have small fluctuations. The Figure 6.1 shows the trend for these four groups of models. A possible explanation for this might be that the one block model has less number of pooling layer which results larger feature map before the dense layer. The mismatch between the larger feature map and less hidden units leads to the under fitting of the model, so when increase the number of hidden units, the error rate for the one block model decreases. For other groups of model, around 100 hidden units are sufficient for the model, and the error rates do not change much with the increase of hidden units.

Table 6.5: Error rates for one block models with different hidden units

oneBlock	**1D1**	**1D2**	**1D3**	**1D4**	**1D5**	**1D6**
Conv2D,32	5x5	5x5	5x5	5x5	5x5	5x5
Conv2D,32	5x5	5x5	5x5	5x5	5x5	5x5
MaxPool	6x6	6x6	6x6	6x6	6x6	6x6
Dropout	0,3	0,3	0,3	0,3	0,3	0,3
Dense	50	100	150	200	250	300
Dropout	0,3	0,3	0,3	0,3	0,3	0,3
Dense	30	30	30	30	30	30
params	298.444	570.394	842.344	1.114.294	1.386.244	1.658.194
error	7,80%	6,84%	6,64%	6,16%	6,24%	6,11%

Table 6.6: Error rates for two blocks models with different hidden units

twoBlocks	**2D1**	**2D2**	**2D3**	**2D4**	**2D5**	**2D6**	**2D7**
Conv2D,32	4x4	4x4	4x4	4x4	4x4	4x4	4x4
Conv2D,32	4x4	4x4	4x4	4x4	4x4	4x4	4x4
MaxPool	4x4	4x4	4x4	4x4	4x4	4x4	4x4
Dropout	0,25	0,25	0,25	0,25	0,25	0,25	0,25
Conv2D,64	4x4	4x4	4x4	4x4	4x4	4x4	4x4
Conv2D,64	4x4	4x4	4x4	4x4	4x4	4x4	4x4
MaxPool	4x4	4x4	4x4	4x4	4x4	4x4	4x4
Dropout	0,25	0,25	0,25	0,25	0,25	0,25	0,25
Dense	50	100	150	200	300	400	500
Dropout	0,2	0,2	0,2	0,2	0,2	0,2	0,2
Dense	30	30	30	30	30	30	30
params	196,972	278.522	360,072	441.622	604,722	767,822	930.922
error	4,39%	4,21%	4,30%	4,27%	4,49%	4,55%	4,49%

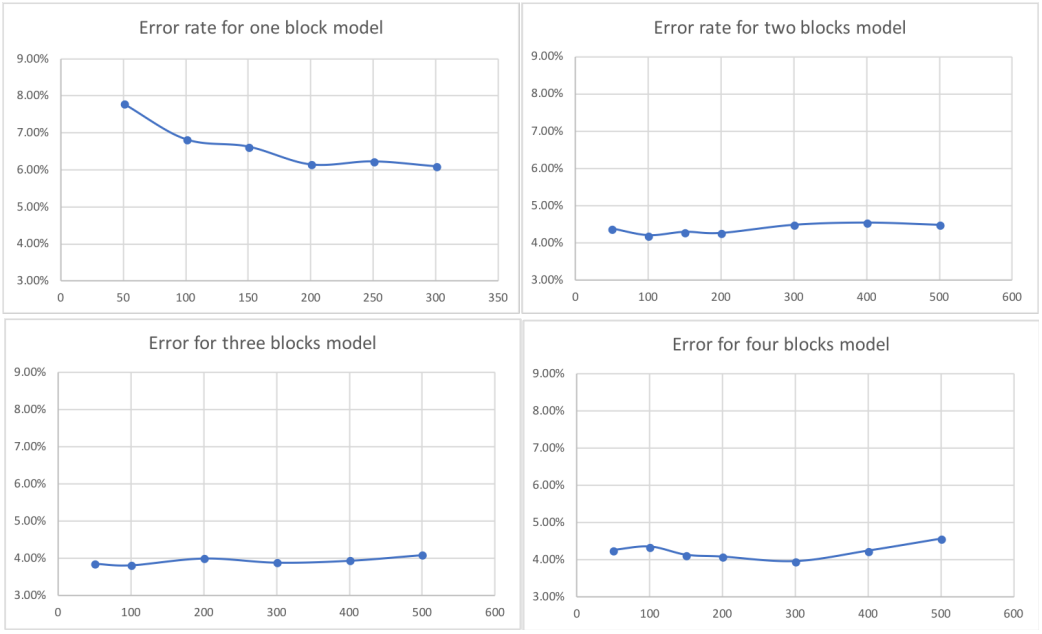
Table 6.7: Error rates for three blocks models with different hidden units

threeBlocks	**3D1**	**3D2**	**3D3**	**3D4**	**3D5**	**3D6**
Conv2D,32	3x3	3x3	3x3	3x3	3x3	3x3
Conv2D,32	3x3	3x3	3x3	3x3	3x3	3x3
MaxPool	3x3	3x3	3x3	3x3	3x3	3x3
Dropout	0,25	0,25	0,25	0,25	0,25	0,25
Conv2D,32	3x3	3x3	3x3	3x3	3x3	3x3
Conv2D,32	3x3	3x3	3x3	3x3	3x3	3x3
MaxPool	3x3	3x3	3x3	3x3	3x3	3x3
Dropout	0,25	0,25	0,25	0,25	0,25	0,25
Conv2D,64	3x3	3x3	3x3	3x3	3x3	3x3
Conv2D,64	3x3	3x3	3x3	3x3	3x3	3x3
MaxPool	3x3	3x3	3x3	3x3	3x3	3x3
Dropout	0,3	0,3	0,3	0,3	0,3	0,3
Dense	50	100	200	300	400	500
Dropout	0,3	0,3	0,3	0,3	0,3	0,3
Dense	30	30	30	30	30	30
params	113.868	144.218	204.918	265.618	326.318	387.018
error	3,86%	3,82%	4,00%	3,89%	3,94%	4,09%

Table 6.8: Error rates for four blocks models with different hidden units

fourBlocks	**4D1**	**4D2**	**4D3**	**4D4**	**4D5**	**4D6**	**4D7**
Conv2D,16	3x3	3x3	3x3	3x3	3x3	3x3	3x3
Conv2D,16	3x3	3x3	3x3	3x3	3x3	3x3	3x3
MaxPool	3x3	3x3	3x3	3x3	3x3	3x3	3x3
Dropout	0,2	0,2	0,2	0,2	0,2	0,2	0,2
Conv2D,32	3x3	3x3	3x3	3x3	3x3	3x3	3x3
Conv2D,32	3x3	3x3	3x3	3x3	3x3	3x3	3x3
MaxPool	3x3	3x3	3x3	3x3	3x3	3x3	3x3
Dropout	0,2	0,2	0,2	0,2	0,2	0,2	0,2
Conv2D,64	3x3	3x3	3x3	3x3	3x3	3x3	3x3
Conv2D,64	3x3	3x3	3x3	3x3	3x3	3x3	3x3
MaxPool	3x3	3x3	3x3	3x3	3x3	3x3	3x3
Dropout	0,2	0,2	0,2	0,2	0,2	0,2	0,2
Conv2D,64	3x3	3x3	3x3	3x3	3x3	3x3	3x3
Conv2D,64	3x3	3x3	3x3	3x3	3x3	3x3	3x3
MaxPool	3x3	3x3	3x3	3x3	3x3	3x3	3x3
Dropout	0,1	0,1	0,1	0,1	0,1	0,1	0,1
Dense	50	100	150	200	300	400	500
Dropout	0,1	0,1	0,1	0,1	0,1	0,1	0,1
Dense	30	30	30	30	30	30	30
params	135.068	165.418	195.768	226.118	286.818	347.518	408.218
error	4,27%	4,36%	4,14%	4,09%	3,97%	4,25%	4,57%

Figure 6.1: Error rate for model with different hidden units in dense layer



6.1.4 Summary

We investigate the impact of convolutional network components in this section. We focus on the depth of the CNN, the sizes of filters in convolutional layers and pooling layers, as well as the number of hidden units in the dense layers.

The first experiment about the depth of CNN shows that when increase the depth, the model performance first increases then drops. The optimal depth for our model is three block.

The second experiment about the sizes of kernels in convolutional layer and pooling layer shows that models with different depths have different optimal kernel size. The optimal sizes for kernels is between (2,2) and (7,7) for models in our experiment. The deeper model has smaller optimal size. For three blocks model, the best result is CNN kernel size (3,3) and pool kernel size (3,3). The three blocks model with convolutional kernel size (4,4) and pool kernel size (3,3) yields very similar result. So we choose the convolutional kernel size (3,3) and pool kernel size (3,3) for our three block model used in the next section.

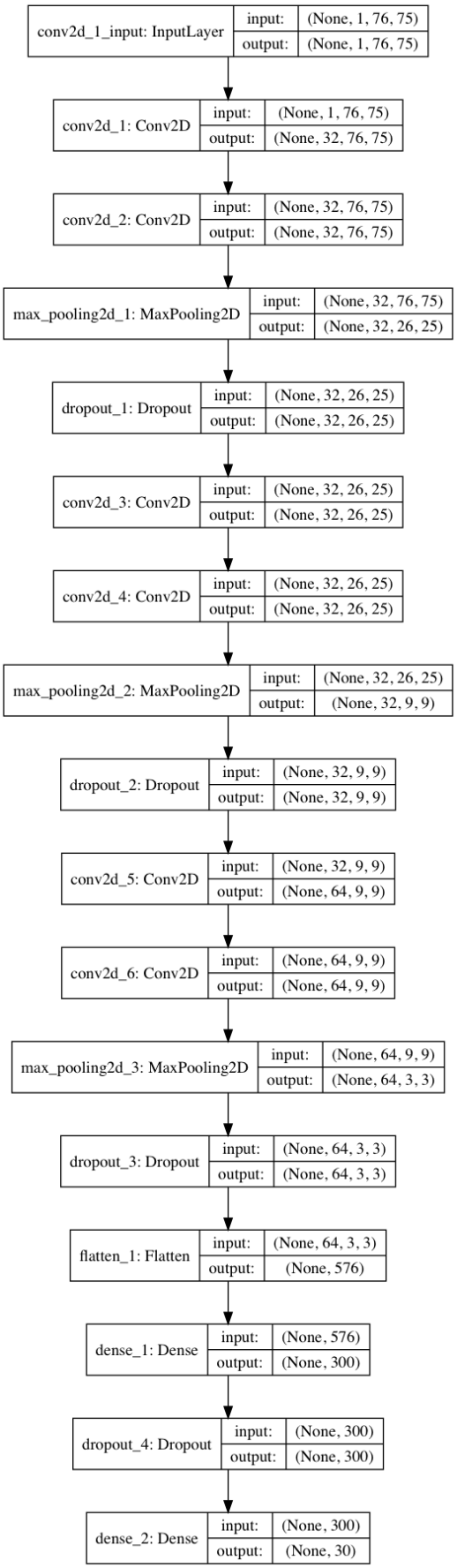


Figure 6.2: Model structure for section 2

The third experiment explores the hidden units in dense layers. The one block model experiment suggests that number of hidden units should match the number of units in the flatten layer. An unbalance between the number units in flatten layer and dense layer leads to the underfitting of the model. Experiments of two blocks, three blocks and four blocks model show no significant difference when changing the number of hidden units in the dense layer. So we simply choose 300 hidden units in the dense layer for the three blocks model used in the next section.

Finally, we choose the three blocks model with convolutional kernel size (3,3), pool kernel size (3,3) and 300 hidden units in the intermediate dense layer. The model structure is shown in Figure 6.2.

6.2 The feature extraction method's experiment

As we discussed in Chapter 2, the feature extraction method is essential in acoustic modeling. Here we design experiments to compare two front-end acoustic feature extraction methods—-GFSCs based on gammatone filterbanks and STFT based on FFT.

6.2.1 Experimental Settings

The dataset *Google speech commands dataset* is recorded by thousands of people under various uncontrolled environments. In spite of this dataset already contains noise, we add extra noise to the speech signals, and evaluate their robustness to different noise levels.

The function of adding noise to the speech sound is presented below, we choose the white noise as our noise source, and we select the factors following exponential distribution. The final factors we choose are $0.000001 \times [4, 8, 16, 32, 64, 128, 256, 512]$.

```
function ADD_NOISE(audio, noise, factor)
    audio ← min_max_normalise(audio, range = (−2000, 2000))
    noise ← min_max_normalise(noise, range = (−2000, 2000))
     return audio + factor ∗ noise
```

Later GFSCs and STFT extract the features from the audios with extra noise.

For the deep learning model, we simply select our best three blocks models

Table 6.9: Error rates for GFSCs in various noise levels

Noise	gfsc0	gfsc4	gfsc8	gfsc16	gfsc32	gfsc64	gfsc128	gfsc256	gfsc512
valid1	3,75%	4,37%	5,06%	6,32%	7,72%	9,71%	11,70%	16,15%	20,58%
valid2	3,85%	4,62%	5,00%	6,52%	7,63%	9,92%	11,72%	15,09%	19,83%
valid3	3,87%	4,34%	5,21%	6,09%	7,30%	9,86%	12,71%	15,95%	***96,33%***
valid4	3,90%	4,42%	4,63%	5,76%	7,62%	9,22%	12,94%	14,65%	21,05%
valid5	3,78%	4,25%	4,87%	5,54%	7,18%	10,03%	11,45%	15,35%	19,62%
valid6	3,97%	4,43%	5,10%	6,37%	7,47%	9,69%	11,73%	14,92%	21,72%
valid7	3,75%	4,48%	4,93%	4,51%	7,36%	9,17%	11,26%	15,08%	***96,32%***
valid8	4,11%	4,36%	4,88%	6,30%	8,04%	10,15%	13,03%	17,22%	24,46%
mean	3,87%	4,41%	4,96%	6,15%	7,54%	9,72%	12,07%	15,55%	21,21%
std	0,11%	0,10%	0,16%	0,32%	0,25%	0,33%	0,66%	0,79%	1,62%

with 3x3 filter size and 3x3 pooling kernel size. And we do a 8-fold cross validation in this section.

6.2.2 Results and discussion

Table 6.10: Error rates for STFT in various noise levels

Noise	stft0	stft4	stft8	stft16	stft32	stft64	stft128	stft256	stft512
valid1	4,81%	5,53%	5,54%	6,98%	8,34%	9,60%	11,76%	15,99%	***96,34%***
valid2	4,52%	4,73%	5,37%	6,68%	7,85%	8,82%	11,45%	***96,33%***	***96,33%***
valid3	4,58%	4,69%	5,72%	6,30%	8,14%	9,45%	11,70%	14,59%	***96,33%***
valid4	4,86%	5,29%	5,74%	6,97%	8,09%	9,59%	12,27%	***96,33%***	***96,33%***
valid5	4,94%	5,24%	5,98%	7,13%	8,36%	9,38%	10,76%	14,87%	***96,33%***
valid6	4,55%	5,16%	5,46%	6,56%	7,63%	9,03%	***96,33%***	20,83%	***96,33%***
valid7	4,58%	5,11%	5,77%	6,84%	7,78%	9,37%	12,73%	18,27%	27,45%
valid8	4,48%	4,87%	5,82%	5,96%	8,31%	9,35%	12,58%	18,12%	30,44%
mean	4,67%	5,08%	5,68%	6,68%	8,06%	9,32%	11,89%	17,11%	28,95%
std	0,16%	0,27%	0,19%	0,37%	0,26%	0,25%	0,64%	2,19%	0,75%

The Table[1] 6.9 and 6.10 provide the detailed results for each cross validation as well as the mean value and standard derivation for 8-fold cross validation for our two feature extraction methods.

[1]The naming rules for these tables are following the rule: *extraction_method+noise_factor*. Such as gfsc8 means GFSCs method for $audio + 0.000001 \times 8 \times noise$.

Table 6.10 compares the STFT feature extraction method's performance in environments with increasing noise levels. What stands out in the table is these error rates in (stft128, valid6), (stft256, valid2), (stft256, valid2), (stft256, valid4), (stf512, valid1), (stf512, valid2), (stf512, valid3), (stf512, valid4), (stf512, valid5), (stf512, valid6) are all close to 96.3%(30 categories classification game), which means that these models fail to learn because of the presence of the strong noise. We calculate the mean error rates and standard derivations without these failed models. The mean error rates are climbing when the noise level is increasing.

Table 6.9 presents the experimental data of GFSCs feature extraction method's performance in environments with increasing noise. And there are also some models that fail in classification task, they are (gfsc512,valid3), (gfsc512, valid7). These failed models are not included in the mean error rates and standard derivations. There is a clear trend of increasing error rates with increasing noise levels.

Compare the differences between Table 6.10 and 6.9 from the error rates perspective, the GFSCs method performs slightly better than the STFT when the noise factor is smaller than 32×0.000001, while STFT overcomes the GFSCs a little bit in noise factor 64×0.000001 and 128×0.000001. What can be clear seen in two tables is the error rates when noise factors are 256×0.000001 and 512×0.000001, the STFT fails 2 times and 6 times respectively, while the GFSCs only fails 2 times when noise factor is 512×0.000001.

We do a t-test to determine two sets of data in the same noise level are significantly different from each other. The Table 6.11 shows the p-value. We observe the differences are very significant for **Noise0, Noise4, Noise8, Noise16, Noise32, Noise64, Noise512**. While for group **Noise128, Noise256**, the significances are less significant.

Table 6.11: T-Test for STFT and GFSCs for different noise levels

Level	**Noise0**	**Noise4**	**Noise8**	**Noise16**	**Noise32**
t-test	1.57049E-07	0.000194535	3.07393E-06	0.017841825	0.00200968
Level	**Noise64**	**Noise128**	**Noise256**	**Noise512**	
t-test	0.026801441	0.358341139	0.143936896	0.031660657	

We remove the failed models, then draw the box-plot for GFSCs and STFT in Figure 6.3. The error rates drops with the increase of noise levels. The variance also increases.

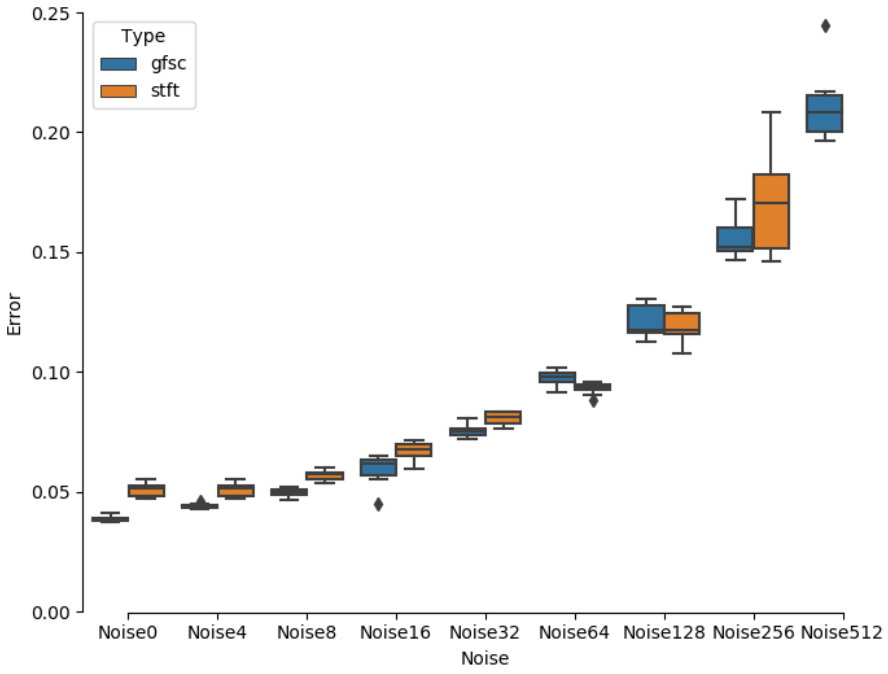


Figure 6.3: Error rate for GFSCs and STFT in various noise levels

6.2.3 Summary

We compare the performance of STFT and GFSCs in this section. Our first finding is the GFSCs performances are better than those of STFT, especially in extremely conditions. The second finding is that although the STFT performance worse than the GFSCs, but it is still acceptable in low noise ratio scenarios which require fast response.

7

Conclusions

Our thesis aims to investigate both the front-end feature extraction methods and back-end convolutional neural network classifier.

Our thesis set out to investigate convolutional neural network for spoken word recognition and we focus on the depth of the CNN, the size of the convolutional layer and pooling layer kernels, the number of hidden units in dense layer. We first find the best design of CNN structure and then use this optimal structure to compare our front-end feature extraction methods.

We define two convolutional layers, one pooling layer and one drop out layer connected sequentially as a block. We stack several blocks and add two dense layers at the end. We select difference size of kernels and hidden units for these different depth models. We use the GFSCs feature extraction method as the front-end feature extraction method when evaluating CNN.

The result of the depth of the CNN shows that increasing the depth of the CNN leads to an increase to the performance first, but the performance can not keep on increasing. When the model goes deeper, the performance drops. Here the best depth in our work is three blocks model.

The result of exploring convolutional layer and pooling layer kernel sizes suggests that the optimal kernel sizes for the CNN are intermediate number between (2,2) and (7,7) in all of our models. When increasing the size of the kernel, the performance first increases then keeps steady or drops. And another finding is that for deeper model, the optimal kernel size is smaller. This experiment has identified that when designing a shallower network we should choose larger kernel size compare to a deeper network. We can not give the exact size for the convolutional layer kernel and pooling layer kernel. The optimal size for CNN structure depends on the depth of the model, the size of the input and other design of the network.

The experiment of exploring the hidden units gives some suggestions for how to choose the number of hidden units in dense layers. The one block model experiment suggests that the number of hidden units should match the number of units in the flatten layer. An imbalance between the number units in flatten layer and dense layer leads to the underfitting of the model. Experiments of two blocks, three blocks and four blocks models show no significant difference when changing the number of hidden units in the dense layer. Two or more blocks of the model have fewer units in the flatten layer, so even a small number of hidden units in dense layer is sufficient for these models.

In all, it is not an easy problem to design an optimal CNN, most published studies of CNN used for speech recognition just use, do not mention why they choose their model structure and how they set the parameters. And the depth of the CNN and these parameters really matter to the result of speech recognition. Our exploration about the CNNs gives suggestions on the impact of the depth, sizes of convolutional kernels and pooling kernels, as well as the the number of hidden units. These experiments contribute to designing CNN models by providing an insight into how to set the size of kernel size and hidden units, as well as the depth of the CNN.

The second part of our experiment is about the front-end feature extraction methods and their capability in noisy environments. We increase the noise gradually and test the performance of our two feature extraction methods. The gammatone filterbanks method presents a slightly better performance in noise, especially in the extremely noisy conditions: the GFSCs method fails to learn less often. While considering the time and complexity of computing the STFT and GFSCs, the performance of STFT is still acceptable in less noisy environments. We recommend using STFT in scenarios which require real time or have limited computing power.

The second part of our experiment is the first comprehensive investigation of gammatone filterbanks spectral coefficients used in CNN models. While for STFT feature extraction method, there are already some studies about using the STFT in CNN models, but our work fills the gap of testing STFT in noisy environments.

Some differences in the results are minor, being limited to the computing power, our study lacks of the cross-validation for the first part of our experiments. Another limitation of this study is that we only explore the STFT feature extraction and GFSCs feature extraction methods, we do not have the time to explore other kind of interesting feature extraction methods for CNN networks such as mel-frequency spectral coefficients, other kind of filterbanks and etc.

Another contribution for our work is that we check other research papers which also use the *Google Speech command Dataset*, such as Raphael Tang's (Tang & Lin, 2017) which yield the best results among all the papers we reviewed, the best result is 95.2% , while in our thesis, the best result 96.12% which is given by the three blocks model. Our work has reached the state of art for this dataset.

7.1 Future Work

There are many extensions that can be applied to our work. We present here a few ideas or directions one can take in.

The problem raised by this study is computing the gammatone filterbanks spectral coefficients needs a lot computing resource and time. Further research might explore the fast and efficient way to calculate not only the gammatone filterbanks response but all the other filterbanks feature extraction method. The development of the feature extraction methods helps us to explore other feature extraction methods in deep learning other than the MFCCs.

In our work, the deepest model is our four block model which has eight convolutional neural networks and 4 pooling layers, although we use the ReLU activation function which suffer less from the gradient vanishing problem, and the best result for our work appears in three blocks, we can not go deeper just by simply stack more layers. We noticed a CNN structure called ResNet, which introduce a so-called "identity shortcut connection" that skips

one or more layers (Fung, 2017). Readers who are interested in this direction could read this review (Fung, 2017) first.

Lastly, the speech recognition is the word level. The further studies need to carried out in sentence level. Our CNN model could used as the front-end classifier and provide the probabilities of the words for the back-end classifier. There are some CNN-RNN models which performs in the sentence level, such as these two papers (Sainath et al., 2015) and (Bae et al., 2016), which combine the advantages of CNN and RNN. But the CNN-RNN model still lacks of exploration, and it still has a long way to go. For example, further study might explore using GFSCs the in CNN-RNN model, or explore the impacts of CNN-RNN parameters etc.

In a word, future study can not only work on the feature extraction domain, to find a faster implementation of the filterbanks or explore other feature extraction other than MFCCs, but also work on the deep learning domain, to explore other network structure such as ResNet or CNN-RNN model.

A

Appendix

A.1 List of Acronyms

2D	Two-dimensional
ANN	Artificial Neural Network
ASR	Automatic Speech Recognition
DBN	Deep Belied Network
HMM	Hidden Markov Model
GMM	Gaussian Mixture Model
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
LSTM	Long Short-Term Memory
DNN	Deep Neural Network
MFCCs	Mel-Frequency Cepstral Coefficients
MFSCs	Mel-Frequency Spectral Coefficients
STFT	Short Time Fourier Transform
GFSCs	Gammmatone Filterbank Spectral Coefficients

A.2 Model Structure

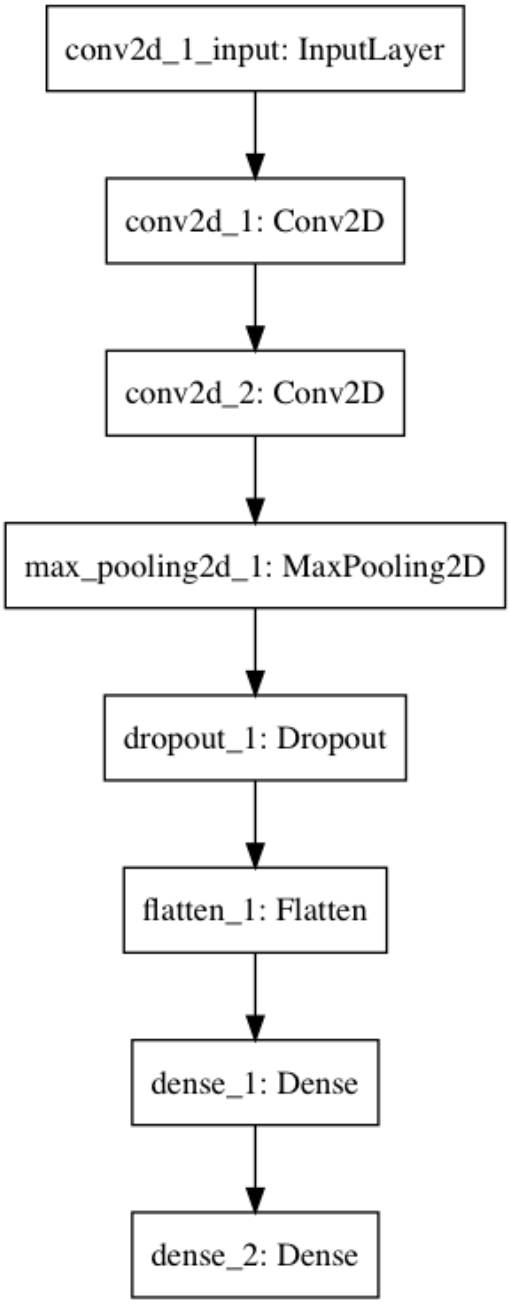


Figure A.1: One Block Model

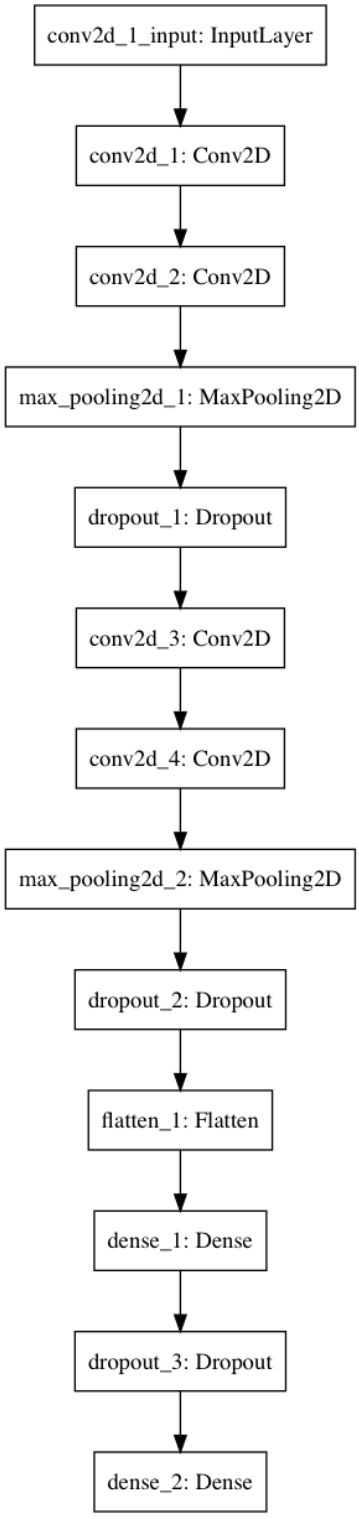


Figure A.2: Two Blocks Model

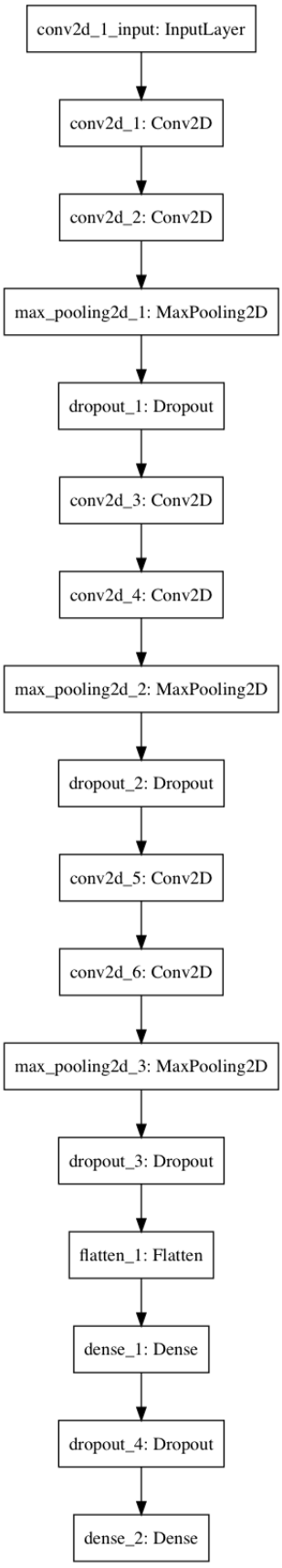


Figure A.3: Three Blocks Model

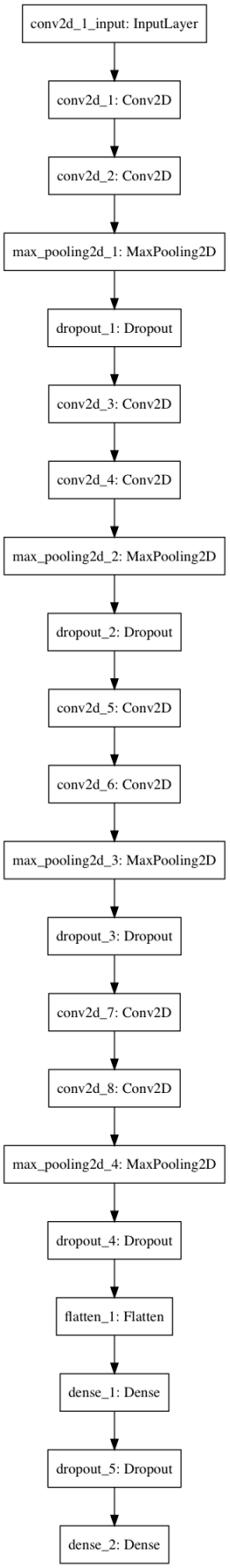


Figure A.4: Four Blocks Model

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., ... others (2018). Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*.

Abdel-Hamid, O., Mohamed, A.-r., Jiang, H., Deng, L., Penn, G., & Yu, D. (2014). Convolutional neural networks for speech recognition. *IEEE/ACM Transactions on audio, speech, and language processing*, *22*(10), 1533–1545.

Abdel-Hamid, O., Mohamed, A.-r., Jiang, H., & Penn, G. (2012). Applying convolutional neural networks concepts to hybrid nn-hmm model for speech recognition. In *Acoustics, speech and signal processing (icassp), 2012 ieee international conference on* (pp. 4277–4280).

Ashish. (2016). *basilar membrane.* `https://www.scienceabc.com/wp-content/uploads/2017/08/Basilar-membrane-in-Cochlea.webp`.

Bae, S. H., Choi, I., & Kim, N. S. (2016). Acoustic scene classification using parallel combination of lstm and cnn. In *Proceedings of the detection and classification of acoustic scenes and events 2016 workshop (dcase2016)* (pp. 11–15).

Buduma, N., & Locascio, N. (2017). *Fundamentals of deep learning: Designing next-generation machine intelligence algorithms* (1st ed.). O'Reilly Media, Inc.

code project. (2018). *Flatten layer.* `https://i.imgur.com/dDYphPB.png`.

Cs231ncnn. (2018). `http://cs231n.github.io/convolutional-networks/`.

Dahl, G. E., Yu, D., Deng, L., & Acero, A. (2012). Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *IEEE Transactions on audio, speech, and language processing*, *20*(1), 30–42.

Deloche, F. (2017). *Unfold structure of rnn.* `{https://commons.wikimedia.org/wiki/File:Recurrent\_neural\_network\_unfold.svg}`.

den Bakker, I. (2017). *Python deep learning cookbook.* Packt Publishing Ltd.

Fung, V. (2017). *An overview of resnet and its variants.* `https://towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035`.

Gao, B., Woo, W. L., & Dlay, S. S. (2013). Unsupervised single-channel separation of nonstationary signals using gammatone filterbank and itakura–saito nonnegative matrix two-dimensional factorizations. *IEEE Transactions on Circuits and Systems I: Regular Papers*, *60*(3), 662–675.

Géron, A. (2017). *Hands-on machine learning with scikit-learn and tensorflow: concepts, tools, and techniques to build intelligent systems.* " O'Reilly Media, Inc.".

Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics* (pp. 249–256).

Graves, A., Jaitly, N., & Mohamed, A.-r. (2013). Hybrid speech recognition with deep bidirectional lstm. In *Automatic speech recognition and understanding (asru), 2013 ieee workshop on* (pp. 273–278).

Graves, A., Mohamed, A.-r., & Hinton, G. (2013). Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on* (pp. 6645–6649).

Gulli, A., & Pal, S. (2017). *Deep learning with keras.* Packt Publishing Ltd.

Hannun, A., Case, C., Casper, J., Catanzaro, B., Diamos, G., Elsen, E., ... others (2014). Deep speech: Scaling up end-to-end speech recognition. *arXiv preprint arXiv:1412.5567*.

Hao, Z. (2017). *Lstm and gru.* {`https://isaacchanghau.github.io/img/deeplearning/lstmgru/lstmandgru.png`}.

Hohmann, V. (2002). Frequency analysis and synthesis using a gammatone filterbank. *Acta Acustica united with Acustica*, *88*(3), 433–442.

Huang, P.-S., Kim, M., Hasegawa-Johnson, M., & Smaragdis, P. (2015). Joint optimization of masks and deep recurrent neural networks for monaural source separation. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, *23*(12), 2136–2147.

Huzaifah, M. (2017). Comparison of time-frequency representations for environmental sound classification using convolutional neural networks. *arXiv preprint arXiv:1706.07156*.

iegfried GÃijndert. (2014). *Filter banks and acoustics tools package for python.* `http://siggigue.github.io/pyfilterbank/`.

Inductiveload. (2009). *Anatomy of human ear with cochlear frequency mapping.* `https://commons.wikimedia.org/wiki/File:Anatomy_of_Human_Ear_with_Cochlear_Frequency_Mapping.svg`.

Irino, T., & Patterson, R. D. (1997). A time-domain, level-dependent auditory filter: The gammachirp. *The Journal of the Acoustical Society of America*, *101*(1), 412–419.

Jaitly, N., Nguyen, P., Senior, A., & Vanhoucke, V. (2012). Application of pretrained deep neural networks to large vocabulary speech recognition. In *Thirteenth annual conference of the international speech communication association.*

Jones, E., Oliphant, T., & Peterson, P. (2014). {SciPy}: open source scien-

tific tools for {Python}.

Keras. (2018). `https://keras.io/`.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *nature*, *521*(7553), 436.

LeCun, Y., Kavukcuoglu, K., & Farabet, C. (2010). Convolutional networks and applications in vision. In *Circuits and systems (iscas), proceedings of 2010 ieee international symposium on* (pp. 253–256).

Liu, H., Li, L., & Ma, J. (2016). Rolling bearing fault diagnosis based on stft-deep learning and sound signals. *Shock and Vibration*, *2016*.

Mohamed, A.-r., Dahl, G., & Hinton, G. (2009). Deep belief networks for phone recognition. In *Nips workshop on deep learning for speech recognition and related applications* (Vol. 1, p. 39).

Nair, V., & Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (icml-10)* (pp. 807–814).

Narayana, M. L., & Kopparapu, S. K. (2009). Effect of noise-in-speech on mfcc parameters. In *Proceedings of the 9th wseas international conference on signal, speech and image processing, and 9th wseas international conference on multimedia, internet & video technologies* (pp. 39–43).

Olah, C. (2015). *Understanding lstm networks.* `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`.

O'Shaughnessy, D. (2008). Automatic speech recognition: History, methods and challenges. *Pattern Recognition*, *41*(10), 2965–2979.

Oyedotun, O. K., & Dimililer, K. (2016). Pattern recognition: invariance learning in convolutional auto encoder network. *International Journal of Image, Graphics and Signal Processing*, *8*(3), 19.

Park, A. (2003). Using the gammachirp filter for auditory analysis of speech. *May*, *14*, 18–327.

Pickles, J. O. (1988). *An introduction to the physiology of hearing* (Vol. 2). Academic press London.

Quertyus. (2011). *Restricted boltzmann machine.* `https://upload.wikimedia.org/wikipedia/commons/e/e8/Restricted\_Boltzmann\_machine.svg`.

Rabiner, L. R., Schafer, R. W., et al. (2007). Introduction to digital speech processing. *Foundations and Trends® in Signal Processing*, *1*(1–2), 1–194.

Ruder, S. (2016). An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*.

Sainath, T. N., Mohamed, A.-r., Kingsbury, B., & Ramabhadran, B. (2013).

Deep convolutional neural networks for lvcsr. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on* (pp. 8614–8618).

Sainath, T. N., Vinyals, O., Senior, A., & Sak, H. (2015). Convolutional, long short-term memory, fully connected deep neural networks. In *Acoustics, speech and signal processing (icassp), 2015 ieee international conference on* (pp. 4580–4584).

Shrawankar, U., & Thakare, V. M. (2013). Techniques for feature extraction in speech recognition system: A comparative study. *arXiv preprint arXiv:1305.1145*.

Simeon, K. (2017). *Understanding gru networks.* `https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be`.

songhoAhn. (2008). *Convolution and c++ algorithms for convolution.* `http://www.songho.ca/dsp/convolution/convolution.html`.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, *15*(1), 1929–1958.

Stevens, S. S., Volkmann, J., & Newman, E. B. (1937). A scale for the measurement of the psychological magnitude pitch. *The Journal of the Acoustical Society of America*, *8*(3), 185–190.

Stutz, D. (2014). Understanding convolutional neural networks. *InSeminar Report, Fakultät für Mathematik, Informatik und Naturwissenschaften Lehr-und Forschungsgebiet Informatik VIII Computer Vision*.

Tang, R., & Lin, J. (2017). Deep residual learning for small-footprint keyword spotting. *arXiv preprint arXiv:1710.10361*.

Tensorflow. (2018). `https://www.tensorflow.org`.

Tüske, Z., Golik, P., Schlüter, R., & Ney, H. (2014). Acoustic modeling with deep neural networks using raw time signal for lvcsr. In *Fifteenth annual conference of the international speech communication association.*

Vinyals, O., Ravuri, S. V., & Povey, D. (2012). Revisiting recurrent neural networks for robust asr. In *Acoustics, speech and signal processing (icassp), 2012 ieee international conference on* (pp. 4085–4088).

Warden, P. (2017). Speech commands: A public dataset for single-word speech recognition. *Dataset available from http://download.tensorflow.org/data/speech_commands_v0.01.tar.gz*.

Zhang, Y., Pezeshki, M., Brakel, P., Zhang, S., Bengio, C. L. Y., & Courville, A. (2017). Towards end-to-end speech recognition with deep convolutional neural networks. *arXiv preprint arXiv:1701.02720*.