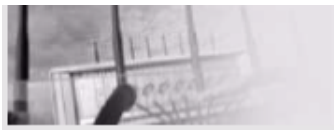


Key Word Detection Using Deep Neural Networks

Master Project

submitted by

Ojo Akinlolu Oluwabusayo

Universität Bremen

Fraunhofer IDMT

Key Word Detection
Using Deep Neural Networks

Faculty 1 - Communication and Information Technology
Derpartment of Communication Engineering
P.O. Box 33 04 40

D-28334 Bremen

Supervisor: Dr.-Ing. Stefan Goete
Responsible Professor: Prof. Dr. Armin Dekorsy

I ensure that this report has been written independently and no sources or aids other than the ones stated herein have been used

.....

October 1, 2019

Abstract

Keyword detection is the task of spotting a keyword of target within continuous speech. Applications of this technology include command controlled devices, customer satisfaction detection, keyword monitoring applications, call center dialogues, amongst others. Despite the influx of researches into this technology, many keyword detection systems still suffer from poor spotting rates, excessive false alarm rates, or slow implementation time, thus reducing the commercial feasibility.

This research investigates and analyzes keyword detection technologies and their applications, as well as how it has influenced the advancement and deployment of speech technology for commercial viability. It makes a number of contributions to the technology of keyword detection. After discussing the introduction to neural networks and the multi-layer perception, it summarizes the methods to regularization and discusses both supervised and unsupervised training of neural networks.

The research summarizes the effects of limited training data for keyword detection technologies. Though, deep neural networks (DNNs) are able to perform excellently with sufficient amount of data. However, since DNNs have huge number of parameters, they can cause overfitting of the model especially if the data is not large enough. This will be avoided by augmenting data and using adequate regularization techniques.

The second major part of this research is the exploitation of mixing noise with input features to improving performance of key word detection devices in a noisy background and ensure good detection rate while working on stream with low signal-to-noise ratio. In addition, the anomaly between validation and training accuracy will be discussed.

Finally, the major part of this work is the development of reliable and optimized convolutional neural network (CNN) and low-latency singular value decomposition (LSVD) models. The CNN model show a better and an improved performance, even with a noisy signal, resulting in a validation accuracy of 89.5%, while the LSVD with fewer number of parameters and reduced complexity shows an accuracy of 80.2%. The research work is carried out with a consideration as to how our contributions can impact the development and commercial deployment of speech technologies for an effective and improved utilization.

Acknowledgements

Foremost, I would like to appreciate my Lord and Saviour Jesus Christ, for He has given me the abundant grace and the opportunity to come this far and accomplish this research work.

I would also like to express my limitless gratitude to my supervisors: Doctor-Ing. Stefan Goetze and Professor-Ing. Armin Dekorsy for believing in my capacity and ideas, and supporting me during this research work. I sincerely appreciate the freedom in the choice of research directions they gave me, it has motivated and improved my research ability through self learning.

I would like to offer a special thanks to Mr. Vinay Nooromkuttil and Mr. Matthias Stennes from Fraunhofer-Institut for Digitale Medientechnologie, without you, the project would have been impossible. Your encouragement, constructive criticism and the series of pre-experiments you made me walk through has helped me a long way. I will also use this medium to appreciate my colleagues; Ekpo Patrick and Ogunlana Femi, you are always there to support and encourage me. My appreciation will be incomplete without mentioning Mr. Emmanouil Gionanidis, a researcher in artificial intelligence and information analysis, University of Thessaloniki. Your support for me anytime I got stuck cannot be over-emphasized.

I would like to thank my loving families, your prayers, words of encouragement and support upheld me during this long venture.

Finally, I would love to appreciate my loving and beautiful fiancee for always giving me a reason to smile, your prayer and support cannot be overemphasized. Thank you for always been there for me.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Keyword detection is an automated task of identifying keywords of target in a stream of speech signal. This technology has been utilized in many applications, ranging from triggering a micro-processor, surveillance applications, phone call routing, to name just a few. This technology is similar to speech to text transcription, but best fit into some certain applications, where only a few words in the speech are transcribed.

However, it requires less processing power than speech-to-text transcription, i.e automatic speech recognition with large vocabulary, and therefore can perform its task using less computational resources. It is for example, well suited for data mining tasks that process huge amounts of speech data. Live stream speech monitoring is an example scenario for which this could be very important. In this application, some keywords of interest are targeted in a real-time audio stream, such as words peculiar to a specific topic. Hence, a large portion of the stream is not of interest and does not require transcription, and therefore a keyword identification that spot the targeted words will be more efficient and power saving than a full speech to text recognition.

Keyword detection is also a good option for audio search applications [1], where words of interest can be searched within some processing time from hours of audio stream. However, numerous keyword detection models are limited by their poor spotting accuracy and slow search rates. This calls for a vigorous research in managing the trade-off between the speed and accuracy performance of keyword detection models. Unfortunately, many keyword detection models are designed to sacrifice percentage of their accuracy to meet up with the needed speed specification for commercial purposes. Nevertheless, keyword detection technology usually performs far better than a large vocabulary speech to text transcription, considering high accuracy, less false alarm rate and speed.

1.1 Motivation and Goals

Deep learning and neural networks have become state-of-the-art in various machine learning research fields, e.g image and speech recognition. Keyword identification is as a sub-task of automatic speech recognition. This thesis focuses on examining applications of keyword spotting technologies on both comparatively short wave signals and from a continuous audio stream. Detecting words of interest from continuous audio streams at a higher search rate while ensuring high accuracy is an important factor in this research. The research also considers the vast growth in the implementation of audio and multimedia applications as well as the mathematical concept behind keyword detection.

The main objectives of this research are stated as follows:

- To review and evaluate state-of-the-art in keyword detection methods
- To investigate the performance (accuracy) of these techniques
- To summarize the mathematics behind keyword detection
- To develop a model with high accuracy relative to speed in keyword detection

1.2 Major Contributions of this Thesis

This thesis contributes to the field of keyword detection, focusing on the following contributions:

1. A summary of keyword detection technologies and their applications, as well as how they have impacted the development and deployment of speech technologies for commercial purposes.
2. Comparative analysis of different techniques used for keyword detection and summary of deep neural networks (DNNs).
3. The development of great architectures for convolutional neural network models for keyword detection to yield significant improvements particularly for the problematic area of short-word keyword spotting.
4. A detail study of the effect of limited training data set for keyword detection, as well as how this has influenced the deployment of speech technologies for non English languages.
5. The research also puts noise enhancement into consideration, by mixing the signal with white noise and human-made noise during training, in order to improve the performance of keyword detection in a noisy background and enhance accuracy while working with stream with very low signal-to-noise ratio.

1.3 Project Organization

An overview of the remaining section of this research work is shown below:

Chapter 2 - Word Spotting presents a review of keyword detection and various methods used in keyword detection. This is followed by general applications of keyword detection. A literature review of keywords detection detailing techniques used in spotting keywords are discussed, and the challenges being faced by the state-of-the-art are also reviewed. Advancement in deep neural networks, which has actually improved performances on acoustic models are detailed. Insufficient of non-English languages data, available for training neural models has caused set back to the development of many speech technologies, such as large vocabulary audio transcribers. However, keyword detection is less affected by reduced amount of training data. This chapter also discusses how this enhances development in keywords technologies without the need for spending much effort in creating large training databases.

Chapter 3 - Artificial Neural Networks summarizes the introduction to artificial neural networks (ANNs), learning process and layers in neural networks. A summary of convolutional neural network are also presented.

Chapter 4 - Feature Extraction and Performance Measures discusses the necessary steps involved to extracting features from audio signal, which will be fed to the model as input feature. Metrics used to measure the performances of neural models are also discussed herein.

Chapter 5 - Regularization summarizes a number of techniques used for regularization in neural network. Various activation functions, optimizers, and cost functions used in neural networks are discussed.

Chapter 6 - Implementation and Results presents the pre-experiments, evaluates and analyses the data set used for this work. A number of experiments are reported to measure the performance of deep neural networks, both on short audio waves and long audio stream. The results demonstrate that a very good accuracy can be achieved with convolutional neural networks (CNNs). In this chapter, some hyper parameters are tuned to evaluate the implemented methods. In addition, results of the

different models of these methods are evaluated.

Chapter 7 - Conclusion and discussion summarizes the research and presents the conclusion as well as a discussion for possible future research work for improvement.

Chapter 2

Word Spotting

2.1 Introduction

This chapter reviews keyword detection methods, the respective challenges, methods used in keyword detection, followed by applications and a description of how keyword detection performance is measured.

2.2 Keywords Detection (KD)

Keyword detection can be seen as a special case of speech-to-text transcription [2, 3], in which the transcription word-bank is limited to the keywords of interest and a non-keyword symbol that represents all other words in the transcription vocabulary.

Let O be an estimated sequence of audio signal, V be the vocabulary of the interest application domain, Q represents the set of keywords of interest and Ω be the none-keyword symbol. If speech-to-text transcription is denoted by:

$$W = \text{ASR}(O, V), \quad \text{with} \quad W = \{w_1, w_2, w_3.....w_n\} \tag{2.1}$$

W is the resulting hypotheses of word sequence, we can define keyword detection (KD) as:

$$KD(O, V, Q) = f(Transcribe(O, V), Q) \tag{2.2}$$

$f(W, Q)$ is the transformation applied to speech-to-text transcription, given as:

$$f(W,Q) = \begin{cases} W & |W| = 1, w_1 \in Q \\ \Omega & |W| = 1, w_1 \notin Q \\ \{w_1, f(Tail(W), Q)\} & |W| > 1, w_1 \in Q \\ \{\Omega, f(Tail(W), Q)\} & |W| > 1, w_1 \notin Q, w_2 \in Q \end{cases} \tag{2.3}$$

$|W|$ is the length of word sequence w, w_1 denotes the first word in sequence w. $f(W, Q)$ will replace all the sequences of non-keywords with a non-keywords symbol Ω in the output by the transcriber. The operation f picks the keywords Q from w. For example, given a vocabulary of application domain V = [yes, no, up, off, down, Ω], with Ω being the non-keyword, and let Q=[yes, no], which denotes the target words. Function f spots the word whose transcription corresponds to any vocabulary in the domain templates. However, this approach of keyword detection could be considered inefficient because it requires full transcription using a vocabulary of size $|V|$. Specifically, keyword detection is interested in occurrences of some target words, defined by Q. In this regard, a more simplified

formulation of keyword detection can be given as:

$$(O, V, Q)\ \text{Transcribe}(O, g(Q)),\ \text{Where} g(Q) = Q \cup \Omega\ \text{(set of target words and the garbage combined).} \tag{2.4}$$

This approach reduces the complexity in calculation by using smaller vocabulary of size $|Q| + 1$. Consequently, the presence of non-keywords remains a burden in this approach and is much of interest among researchers, and further discussed in subsequent section of this chapter.

2.3 Applications of Keyword Detection

The importance of keyword detection in many area of life cannot be over-emphasized, as it is well suited to applications where huge amount of audio speech needs to be processed [4]. Its possible significant speed benefit compared to large vocabulary automatic speech recognition makes it a fit for several scenarios. Some major applications of this technologies are discussed below.

2.3.1 Command Controlled Devices

Command controlled devices monitor the terrain audio and trigger an action whenever they spot any of the set of interested words. A very good example are speech-enabled mobile applications, command controlled factory machines, or voice controlled remotes ($VCRs$).

Although, generic keyword monitoring technologies can as well be used for command controlled devices, they need large memory to be efficient, which might be a problem especially for the case of embedded systems [5]. Hence, significant amount of modifications and enhancement have been made on existing keywords detection approaches to provide maximum performance for the intended application, e.g. needless for large memory because only target words are captured.

2.3.2 Customer Satisfaction

Monitoring the satisfaction of customers is one of the major secrets in growing a business or company. Keyword detection technology could be incorporated in products in such way that it could trigger an automatic email or a text message e.g., to the manager, alerting them of the customer's evaluation. Upset or offensive words could be tagged as stemming from an upset customer or score the call with very low star, otherwise tag it with happy customer or rate it with high star.

2.3.3 Keyword Monitoring Application

Another application of keyword detection is application to continuous real time stream of audio speech and spot the occurrence of some targeted words. Specific keyword monitoring applications include telephone tapping [6], listening devices and broadcast monitoring.

Criminality and malicious activities can be detected by security organizations through telephone tapping and listening device monitoring technologies. Keyword detection provides a fast and profound solution to this task with a degree of accuracy when compared to human monitoring service, especially when there are large number of audio streams needed to be monitored. However, one of the biggest challenges facing this keyword detection technology is poor performance while trying to spot interesting words in an audio stream with noisy background, cross-talk with different languages, multiple speakers, and other interferences [7, 8]. Spotting devices could suffer from very low signal-to-noise ratio, which affects the performance accuracy of speech processing application.

Another interesting application of keyword detection is broadcast monitoring. Commercial broadcasting companies may want to use keywords detection to spot broadcasting sections that maybe of

particular interest to the customer. For instance a famous politician or celebrity maybe interested in news segments where his or her name is being mentioned from a comprehensive set of broadcast sources such as television, commercial radio and community radio which will result in a vast amount of audio streams. Keyword detection would be an excellent solution to process such a huge task in a more reliable and faster way than a possible human detection ability.

2.3.4 Audio Document Indexing

Locating some targeted words or topics of interest in an audio document database is an important task in the multimedia field, and this could be done with keywords detection. This technology works like traditional text indexing systems such as Google Internet search engine, but works on audio documents rather than text documents. The necessity for effective and accurate way of audio indexing is important in a community where audio and multimedia documents play a vital role in everyday life.

Speech-to-text is a traditional way of providing solution to audio document indexing problem, where audio is first transcribed to text, and the equivalent text is then searched during query time [9]. However, many audio documents indexing tasks require names, places, foreign words to be transcribed, which are in many cases not a part of the trained model's vocabulary.

The involvement of Deep Neural Network in keyword detection technology has provide support for unrestricted vocabulary queries in a trade-off to reduction in query speeed [10]. In a more improved system, a keyword detection system can be used to augment a speech-to-text based system to improve the speed of queries while still supporting unrestricted words which are out of vocabulary queries.

2.3.5 Dialogue System

The use of automated dialogue system is fast growing in commercial environment as an effective substitute for human-operated call centers. A dialogue system tries to mimic a human call-center operator by playing voice prompt to a caller and trying to detect some keywords in the customer audio speech to indicate the response of customers. Since the volume of calls processed by call centers can be very huge, speech-to-text transcriber fails in this regards and this prompt the use of keyword detection to interpret the callers' responses. It offers flexibility in the response of the speaker by accommodating out-of-vocabulary words and understand the intention of a caller. Therefore proving more appropriate for many applications.

2.4 Advancement of Keyword Detection

In a similar thread to improvement in speech recognition technology, keyword detection has passed through a number of stages of developments. Traditional approaches were limited by inadequate computing resources. Deep learning has been exploited to solve more advanced tasks in an effective and reliable ways. The following are the advancement in the methods and approaches of detecting keywords.

2.4.1 Slinding Window Method

Primitive method was based on using sliding window method such as dynamic time wrapping method proposed by Sakoe, Chiba and Bridle [11], or the sliding window based on neural network method proposed by Zeppenfeld [12]. The algorithm calculates word unit in each frame during training, and uses a local minimum dynamic time warping algorithm to detect the words in speech [13]. These two methods produced good results in isolated keyword detection but suffer a big degradation in performance when detecting keywords in a stream of long audio speech.

A significant reason for this poor performance was the non-consideration of non-keywords either implicitly or explicitly. Detecting keywords in a continuous stream of audio speech is essentially a 2-class discrimination task, that classifies a segment as either a keyword or non-keyword. Since the initial method of sliding window did not put non-keywords into consideration, it is synonymous to making measurements without a point of reference, which implies that all observations are purely relative and therefore provide little confidence for making correct decisions.

2.4.2 Non-Keyword Model Method

This model is also known as keyword/filler method, it is introduced to address the lack of knowledge of the non-target class. This acoustic keyword detection models keyword and non-keyword explicitly in parallel using sub-word units. At detection phase, interested utterances are aligned with both the target words model and the filler model, and decision will be made based on the alignment cost. The non-keyword model attempted to model all the speech utterances that are not part of the targeted words. Using a non-keyword model gives a better performance when compared to sliding window method.

The introduction of non-keyword models into keyword spotting brought about the incorporation of keyword detection with continuous speech recognition research. More importantly, keyword detection could simply be seen as a special case of continuous speech recognition, where all non-keyword speech are labeled with a single non-keyword tag.

2.4.3 Query-by-Example

Query-by-Example method is one of the earliest attempts for keyword detection [14]. It typically consists of two steps: a template representation step where audio examples of the targeted words are represented as templates in a certain format and a template matching step where templates are compared with interested speech words which have been processed in the same way. It uses dynamic time warping to locate query term only at the syllable boundaries in a long audio.

2.4.4 Hidden Markov Model Method (HMM)

The most widely used recognition algorithm in the early 2000s was Hidden Markov Model (HMM) before the successful evolution of neural networks [15].

HMM is a finite set of states, each of which has a probability distribution. Transition probabilities are also assigned to transitions from one state to another. Consider a particular state, an outcome can be generated according to the associated probability distribution. The external observer can only see the outcome, not the state. Hence, the states are hidden to the outside.
Each spoken word w is split into a sequence of K_w basic sounds known as phones. The sequence is known as its pronunciation $q_1^{(w)} : K_w = q_1, ...qk_w$.

Each base phone q is represented by a continuous density HMM of the form illustrated in Fig 2.1 with transition probability parameter $\{a_{ij}\}$ and output observation distributions $\{b_j()\}$. It makes a transition from it current state to one of its connected states every time step. The probability of making a particular transition from state s_i to state s_j is given by the transition probability $\{a_{ij}\}$ using the standard conditional independence assumptions for HMM:

- states are conditionally independent of all other states given the previous state;
- observation are conditionally independent of all other observations given the state that generate it.

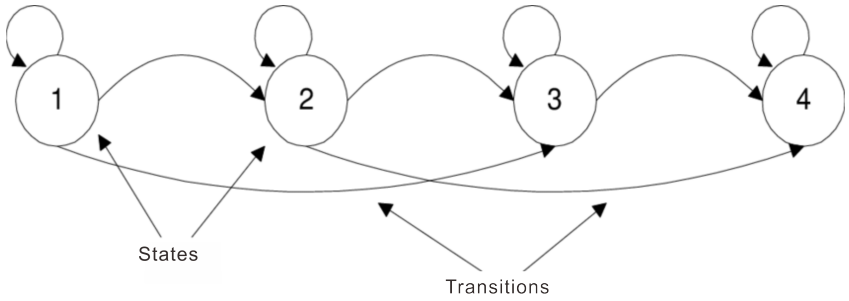


Figure 2.1: Hidden Markov Model

Each word in the vocabulary has distinct HMM. During testing, the unknown word is scored against all HMM model, and HMM with maximum score is detected as recognized word. Following is the summary of different types of HMM:

- Context-Independent Phoneme HMM
  - Number of State: d-state HMM for each phoneme (d is usually chosen to be 3).
  - Accuracy: not accurate in continuous speech.
  - Compact: d-state HMM requires less parameter calculation.
  - General: yes, HMM for new word can be built using existing phoneme HMM
- Context-Dependent Triphone HMM
  - Number of State: d-state for each phoneme.
  - Accuracy: accurate as it has left-right phoneme relation.
  - Compact: each phoneme has immediate left-right relation, more parameters need to be calculated.
  - General: yes
- Whole-Word HMM
  - Number of State: no phoneme generation, assign number of state to model a word as a whole
  - Accuracy: yes, uses large training data
  - Compact: no, it needs too many state as vocabulary increases.
  - General: No, it cannot build new word using this representation

The advent of HMM speech recognition model leads to evolution of HMM-based keyword detection techniques. This leads to the development of better performing keyword detection systems, paving ways to more advanced keyword detection applications.

2.4.5 Gaussian Mixture Model(GMM)

Gaussian mixture model is a parametric probability density function, which is depicted as a weighted sum of Gaussian component densities. GMM density is a weighted sum of its M component densities. It is used as a parametric model of probability distribution of measuring features. GMM is used as a classifier to distinguish the features extracted from the MFCC with the stored templates.

GMM model assumes that a M component mixture model with component weights $p(w_m)$ and parameters θ_m can represent the spectral shape. A probability density function must be formed from the spectrum. The continuous probability density function is formed as the summation of functions based on the FFT of N-points magnitude representation of the speech. In other words, to form a continuous probability function, each point in the FFT is associated with a bin function.
There are several methods available for estimating the GMM parameters. The most popular method is maximum likelihood (ML) estimation, which finds the model parameters and maximizes the likelihood of the GMM given the training data, more details of GMM can be found here [16].

2.4.6 Futher Development

The geometrically increased importance of keyword detection technologies in academic, industrial, security and commercial purposes, as well as in audio and multimedia has introduced the requirements for fast unrestricted vocabulary keyword detection in large audio database. This led to the introduction of two-stage algorithms proposed by Young and Brown [17], which improves the query speed than previously existing keyword detection methods. In recent years, neural networks have been deployed for keyword detection for acoustic modeling and feature extraction. Summary of neural networks will be discussed in chapter three of this research.

Chapter 3

Artificial Neural Network

3.1 Introduction

Artificial neural networks (ANNs) and other deep models have evolved from a long way. Early generation of ANNs were based on simple neural layers perception. Their major limitations are complex computations and poor performance on huge data set. The second generation introduced backpropagation to adjust weights of neurons, based on the error computed. Support vector machine (SVM) emerged thereafter and performed better than ANNs for a while. Then other techniques including feedforward neural networks (FFNs), convolutional neural networks (CNNs) and recurrent neural networks (RNNs) evolved, which improved ANNs in various ways and for different purposes. Detail of evolution and history of deep neural network can be found in: "Recent Advances in Deep Learning: An Overview [18]."

3.2 General Pitfalls of Neural Networks

Some setbacks in neural networks, which are under research to provide profound solutions include:

- Sampling biases: Using dataset that underestimates or misrepresent the real cases, with which results are distorted.
- Irrelevant feature selection: Choosing the right and relevant features can be challenging in learning process
- Inaccurate scaling and normalization
- Neglecting outliers
- Miscalculated features
- Ignoring multi-collinear input
- Missing data
- Data leakage

3.3 Deep Neural Network

Deep learning consists of multiple layers to model and analyze data representation by using backpropagation to recognize structure in large data. Deep learning has performed greatly with supervised, unsupervised and hybrid learning.

3.3.1 Supervised Learning

A supervised learning means having each of the subset in a data fully labeled while training an algorithm or model. In other words, each sample in the set is tagged with a label which is expected to be predicted by the algorithm. During testing, the model compares the testing sample to the training examples to predict the correct right label.

Classification problems and Regression problems are the major fields where supervised learning is useful. Classification tasks request the algorithm to predict discrete value, signifying that the input data belongs to a particular class or group. In a training of keyword detection model, each epoch will be pre-labeled. The model is then evaluated by how accurately it can classify new set of words similar to the training data.

The regression algorithms attempt to estimate the mapping function from the input variables to numerical or continuous output variables. Supervised learning if well fitted for problems where there is a set of reference points with which to train the model.

3.3.2 Unsupervised Learning

A self-learning algorithm that draws inferences from an unlabeled datasets. Base on the task, the algorithm can adopt different methods to organize the data.

- **Clustering:** During training, the model looks for a set of data that are similar to each other and groups them together. Observations similar in some sense are grouped in the same subset. The clustering is done by minimizing the sum of squares of distances between the corresponding cluster centroid and the data.
- **Anomaly Detection:** This is otherwise called outlier detection. It identifies rare observations or events which differ significantly from the majority of the data. Banks for example, use this algorithm to detect fraudulent transactions by tracking strange patterns in customers' purchasing behavior. For instance, if the same credit card is used in different countries within the same day.
- **Association:** An unsupervised learning model predicts attributes by looking at a couple of key attributes of a data point. It measures the correlation between the data point and other features, and mapped it to the subset that has the highest correlation coefficient. Online adverts or suggestions while surfing the internet are based on the previously visited links or websites.
- **Autoencoder:** Another unsupervised learning approach that tries to approximate the identity function, so as to predict an estimated value similar to the original value. It has three layers: an input layer, a hidden layer also known as the encoding layer, and a decoding layer as the output layer. It makes use of backpropagation, equating the target value to the input. However, it compresses the input data into a code, and try to recreate the input data from the compressed code.

  Since the data lack ground truth, it will be difficult to measure the accuracy of a model trained with unsupervised learning. Whereas, in many researches, label data is too expensive to get. Hence, the model looks up into the data and define some set of patterns to group the observations into subsets.

3.3.3 Semi-Supervised Learning

Semi-supervised learning is an algorithm which combines supervised and unsupervised machine learning techniques, which results from scenario whereby only small portion of the data could be labeled, and possible only small section of the dataset and its outliers are labeled, and larger proportion of the

data are unlabeled. It enhances supervised learning classification by minimizing errors in the labeled observations, and it must also be compatible with the distribution of the unlabeled observations. It also focuses to group a better defined clusters than those ones grouped by unlabeled data. It can be categorized as:

- **Transductive:** aims at predicting the labels of the unlabeled observations by considering both labeled and unlabeled observations together into account to train the model classifier.
- **Inductive learning:** It sees the given labeled and unlabeled data as the training example, and its aim is to use it to predict the unknown data.

3.4 Convolutional Neural Network (CNN)

3.4.1 Introduction

CNN was originally introduced by LeCun [19] and became popular after showing significant performance over other submission in the ImageNet ILSVRC challenge in 2012. Consequently, Convolutional Neural Networks have achieved great result in various tasks such as speech recognition, text classification, image recognition and so on.

Convolutional neural network comprises of four main operations: convolutions, non-linearity, pooling and classification, with an addition of batch normalization and dropout. The operations are stacked together as layers in order to perform convolution which are followed by a non-linearity function such as ReLU, sigmoid or logistic and tanh. This operation is repeated a few times before the pooling operation is used. If needed, batch normalization is applied after the convolution but before the non-linearity. After the network is sufficiently deep enough, and the initial input is sub-sampled by pooling to a size where it is manageable, it's then passed through to the classification layer of the network model.

3.4.2 Perequisite to CNN: Tensor and Vectorization

Discussing some prio knowledge necessary to understand how CNN runs will be the starting point in this section. In the concept of this research, boldface letters will be used to represent a vector, e.g., $\mathbf{x} \in \mathbb{R}^D$ is a column vector with D elements. And capital letters will be used to denote matrix, e.g., $X \in \mathbb{R}^{H \times W}$ is a matrix with H rows and W columns. This idea can be generalized to higher order matrices, like tensors, e.g.,

$$X \in \mathbb{R}^{H \times W \times D},$$

which is a tensor of third order, with Hieght H, Width W and Depth D having indices i, j, d, with:

$$0 \leqslant i < H$$

$$0 \leqslant j < W$$

$$0 \leqslant d < D.$$

We can also interpret this as a tensor containing D channels of matrices, i.e. every channel is a matrix with size $H \times W$. The first channel represents a tensor with an indices $i, j, 0$. If $D = 0$, the tensor becomes a matrix.
In a more explanatory way, a scalar value is a tensor of order 0; a vector is a tensor of order 1, and a matrix is a second order tensor. A good example of third tensor is a colored image, with dimension of $H \times W \times 3$, where the 3 represents its channels if it's stored in RGB format. However, one begins to wonder how an audio signal of one dimension can be treated like three dimensional features. In practice, we define a filter over the spectrum of the audio wave, and ensure sufficient snaps of the spectrum is taken to capture our targets.

Tensors are important concepts in CNNs, as the input, intermediate representation, and parameters in a CNN are usually tensors. Hence, it is worth noting that tensors of higher order, i.e. more than 3 are widely used in CNNs during kernel convolution. However, we can vectorize a tensor to a long vector, following a predefined order. An operation which can be easily done using the (:) operator in Matlab. For example; given matrix $A = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$,

$$\text{Vec}(A) = A(:) = (5, 6, 7, 8)^\intercal = \begin{bmatrix} 5 \\ 6 \\ 7 \\ 8 \end{bmatrix} \tag{3.1}$$

Hence, $\text{Vec}(A) = (5, 6, 7, 8)^\intercal$ in the above illustration. To vectorize a tensor of order 3, we could first vectorize its first channel (which is a matrix), before vectorizing the other remaining channels as well. The vectorized tensor is obtained by concatenating all the vectorized channels in this order. Tensors of higher orders can as well be vectorized in this manner.

3.4.3 Vector calculus and the chain rule

The CNN learning process is based on chain rule and vector calculus. Given z as a scalar, $z \in \mathbb{R}$, and $y \in \mathbb{R}^H$ as a vector. If z is a function of y, the partial derivative of z with respect to y can be defined as:

$$\left[\frac{\partial z}{\partial y}\right]_i = \frac{\partial z}{\partial y_i}. \tag{3.2}$$

In other words, $\frac{\partial z}{\partial \mathbf{y}}$ is a vector having the same size as y, and its i-th element is $\frac{\partial z}{\partial y_i}$.

It is also worth noting that $\frac{\partial z}{\partial \mathbf{y}^\intercal} = \left(\frac{\partial z}{\partial \mathbf{y}}\right)^\intercal$.

Now, suppose $\mathbf{x} \in \mathbb{R}^W$ is another vector, and $\mathbf{y}$ is a function of $\mathbf{x}$. Then, the partial derivative of $\mathbf{y}$ with respect to $\mathbf{x}$ can be defined as:

$$\left[\frac{\partial \mathbf{y}}{\partial \mathbf{x}^T}\right]_{ij} = \frac{\partial y_i}{\partial x_j}. \tag{3.3}$$

This partial derivatives is a $H \times W$ matrix, whose entry at the intersection of the i-th row and j-th column is:

$$\frac{\partial y_i}{\partial x_j}.$$

We can see that z is a function of $\mathbf{x}$ in a chain-like argument: a function maps $\mathbf{x}$ to $\mathbf{y}$ and another function maps $\mathbf{y}$ to z. The chain rule can be used to compute

$$\frac{\partial z}{\partial \mathbf{x}^T} \quad \text{as}$$

$$\frac{\partial z}{\partial \mathbf{x}^\intercal} = \frac{\partial z}{\partial \mathbf{y}^\intercal}\frac{\partial \mathbf{y}}{\partial \mathbf{x}^\intercal} \tag{3.4}$$

A sanity check for Equation (3.4) is to check the matrix or vector dimensions. Note that $\frac{\partial z}{\partial \mathbf{x}^\intercal}$ is a row vector with H elements, or a $1 \times H$ matrix. Also, recall that $\frac{\partial z}{\partial \mathbf{y}}$ is a column vector. Therefore, since $\frac{\partial \mathbf{y}}{\partial \mathbf{x}^\intercal}$ is an $H \times W$ matrix, the matrix multiplication between them is valid, and the result should be a row vector with W elements, which matches the dimensionality of $\frac{\partial z}{\partial \mathbf{x}^\intercal}$

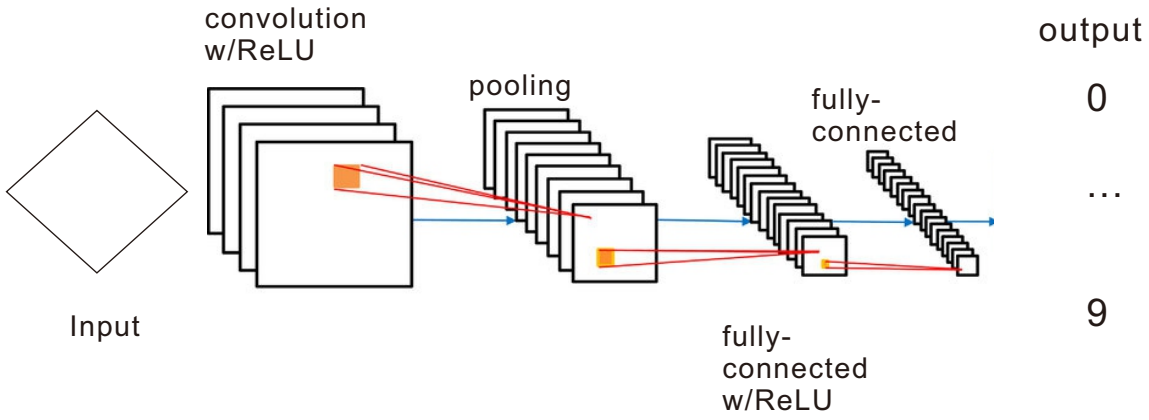


Figure 3.1: A simple CNN architecture with five layers

3.4.4 CNN Architecture

A CNN comprises neurons, which are organized into three spatial dimensions: width, height and depth of the activation volume. Unlike normal ANNs, the neurons within a given layer will only connect to small portion of the preceding region, which will be finally condensed to form $(1 \times 1 \times n)$ output, where n is the number of classes.
CNNs consist of three types of layers: convolutional layers, pooling layers and fully connected layers. These layers are stacked to form CNN's architecture.
As stated earlier, a CNN input feature is usually a tensor of dimension 3, such as Mel frequency cepstral coefficients, image with H rows and W columns, and 3 channels (R, G, B color channels). A CNN can as well handle higher order tensor inputs. The input features go through series of processing, usually called a layer, which could be a convolution layer, a pooling layer, a normalization layer, a fully connected layer, a loss layer, etc. Details of these layers are discussed in section 3.6, subsection 2, 8, and 9 respectively of this project.

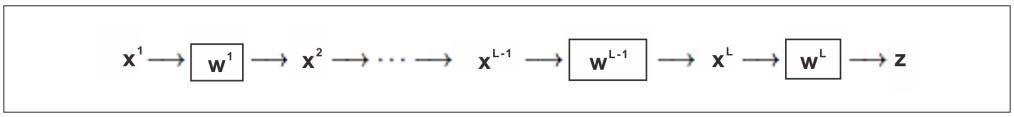


Figure 3.2: Schematic of layers in CNN

Figure 3.2 illustrates how a CNN runs layer by layer in a forward pass. The input is $\mathbf{x}^1$, usually an image (order 3 dimensional tensor), recall that audio wave is treated as an image (spectrum). It undergoes the processing in the first layer, which is the first box. We denote the parameters involved in the first layer's processing collectively as a tensor $\mathbf{w}^1$. The output of the first layer is $\mathbf{x}^2$, which also becomes the input to the second layer processing.

This processing continues till all layers in the CNN has been finished, and finally, output $\mathbf{x}^L$ is produced. One additional layer, which enhances method that learns good parameter values in the CNN is added for backward error propagation. Let's assume the problem at hand is a feature (examples: words, speakers, images) classification problem with C classes. A simple approach is to output $\mathbf{x}^L$

as a C dimensional vector, whose i-th entry encodes the prediction, with posterior probability of $\mathbf{x}^1$ coming from the i-th class. To formulate $\mathbf{x}^L$ as probability mass function, we can set the processing in the $(L-1)$-th layer as a softmax transformation of $\mathbf{x}^{L-1}$. The last layer in the chain is known as a loss layer. Assuming t is the corresponding target (ground-truth) value for the input $\mathbf{x}^1$, then a loss function or cost function can be used to measure the minimum square error between the CNN prediction $\mathbf{x}^L$ and the target t. For example, a simple loss function could be defined as:

$$z = \frac{1}{2}||\mathbf{t} - \mathbf{x}^L||_2^{\,2}, \tag{3.5}$$

Although more complex loss functions are usually used. This squared loss can be used in a regression problem, while, in a classification problem, the cross entropy loss is often used. The ground-truth in a classification problem is a categorical variable t. We first convert the categorical variable t to a C dimensional vector t. It should be noted that both $\mathbf{t}$ and $\mathbf{x}^L$ are probability mass functions, and the cross entropy loss measures the distance between them. Hence, we can minimize the cross entropy. Figure 3.3 explicitly models the loss function as a loss layer, whose processing is modeled as a box with parameters $\mathbf{w}^L$. In addition, some layers may not have any parameters, that is, $\mathbf{w}^i$ may be empty for some i. A good example is the softmax layer.

3.5 Learning Process in CNN

T. Mitchell stated that a computer program is said to learn from experience with respect to some class of tasks and performance measure, if its performance at tasks, improves based on the previous experience. This section summarizes how learning process occurs in CNNs and how it influences accuracy as a performance measure.

3.5.1 Forward Pass

Assuming all the parameters of a CNNs model $\mathbf{w}^1$,...., $\mathbf{w}^{L-1}$ have been learned, then the model can be used for prediction. Prediction only involves running the CNNs model forward. Considering the keywords we are to classify as an example. From Figure 3.3, starting from the input $\mathbf{x}^1$, we make it pass the processing of the first layer (i.e, the box with parameters $\mathbf{w}^1$), and get $\mathbf{x}^2$, which is then passed into the second layer, etc. Finally, we get $\mathbf{x}^L \in \mathbb{R}^C$, which estimates the posterior probabilities of $\mathbf{x}^1$ belonging to the C categories. Hence, we can output the CNN prediction as:

$$arg_i max\ x_i^L \tag{3.6}$$

The loss layer is not used while making prediction. It is only useful while learning a CNN parameters using a set of training examples. Now, the question is: how do we learn the model parameters?

3.5.2 Stochastic Gradient Descent (SGD)

Like many other learning systems, the parameters of a CNN model are optimized to minimize the loss z, That is, we want the prediction of the CNN model to match true labels as close as possible.

Let's assume a training example $\mathbf{x}^1$ is given for training such parameters. The training process involves running the CNN network in both directions. Firstly, we run the network in the forward pass to get $\mathbf{x}^L$, to achieve a prediction using the current model parameters. Instead of outputting a prediction, we will compare the prediction with the target $\mathbf{t}$ that matches $\mathbf{x}^1$, that is, continue running the forward pass till the last loss layer. And finally, we will achieve a loss z, which will now be a supervision signal, guiding the modification of the model parameters, using stochastic gradient descent as stated below:

$$(w^i)^{t+1} = (w^i)^t - \eta \frac{\partial z}{\partial (w^i)^t} \tag{3.7}$$

In Equation (3.7), with η as the learning rate, the partial derivative $\frac{\partial z}{\partial (w^i)^t}$ measures the rate of increase of z with respect to the changes in different dimensions of w^i, and it is otherwise known as gradient in mathematical optimization. Hence, in a small local region around the current value of w^i, moving w^i in the direction determined by the gradient will increase the loss value z. In order to minimize the loss function, we would update w^i along the opposite direction of the gradient. This updating method is known as the gradient descent.

In every update, the parameters will be changed by a small proportion of the negative gradient, controlled by the learning rate. One update based on x^1 will make the loss smaller if the learning rate is not too large. However, the loss of some other training examples become larger as a result of this. Hence, we need to update the parameters using all training examples(epoch). An epoch is achieved when all training examples have been used to update the parameters. In general, one epoch will reduce the average loss on the training set until the learning system fits the training data. Hence, we can repeat the gradient descent by iterating over the epoch and terminate at some point to get the CNN parameters.

If we update the network parameters using only gradient gotten from only one training example, it will result to an unstable loss function. That is, the average loss of all training samples will bounce up and down at very high frequency, because the gradient is estimated using only one training example instead of the whole training set. However, if we update the parameters using the gradient estimated from a subset of the training set, it is called the stochastic gradient descent. Contrary to single example based SGD, we can compute the gradient using the entire training set and then update the parameters. Hence, this batch processing involves a lot of computations because the parameters are updated only once in an epoch, and makes it impractical, especially when we have a large training data set.

Solution to this is to use a mini-batch of training dataset, to calculate the gradient using mini batch, and update the network parameters correspondingly. Stochastic gradient descent (SGD) (using the mini-batch strategy) is the mainstream approach to learn a CNNs parameters. We should also keep in mind that when mini-batch is used, the input tensor of the CNN will have an order of 4 ($H \times W \times D \times N$), where N is the mini-batch size.

3.5.3 Error Back Propagation

The last layer's partial derivatives can be computed easily, since $\mathbf{x}^L$ is connected to z directly and controlled by the network parameters $\mathbf{w}^L$, it is easy to compute $\frac{\partial z}{\partial \mathbf{w}^L}$, and only needed when $\mathbf{w}^L$ is not empty. In the same manner, it is as well easy to compute $\frac{\partial z}{\mathbf{x}^L}$. We will have an empty $\frac{\partial z}{\partial \mathbf{w}^L}$, and $\frac{\partial z}{\mathbf{x}^L} = \mathbf{x}^L - \mathbf{t}$. This implies that we have to compute two set of gradients for every layer: the partial derivatives of z with respect to the layer parameters $\mathbf{w}^i$, and that layer's input $\mathbf{x}^i$.
Hence, as seen from Equation (3.7), the term $\frac{\partial z}{\partial \mathbf{w}^i}$ can be used to update the current (i-th) layer's parameters. While, the term $\frac{\partial z}{\partial \mathbf{x}^i}$ can be used to update the network parameters backwards, i.e, (i-1)-th layer. In other words, $\mathbf{x}^i$ is the output of (i-1)-th layer and $\frac{\partial z}{\partial \mathbf{x}^i}$ is how $\mathbf{x}^i$ should be changed to reduce the loss function. However, we can interpret $\frac{\partial z}{\partial \mathbf{x}^i}$ as part of the error-supervision information projected from z backward till the present layer, and this backward propagation process using $\frac{\partial z}{\partial \mathbf{x}^i}$ continues to the (i-1)-th layer, which ease the learning process in CNN.
For an illustration, consider i-th layer as an example, when we are updating the i-th layer, the back propagation process from $(i+1)$-th layer must have been finished. In other words, we must have computed the terms $\frac{\partial z}{\partial \mathbf{w}^{i+1}}$ and $\frac{\partial z}{\partial \mathbf{x}^{i+1}}$, which are both stored in the memory stack and ready to be used. And the task is to compute $\frac{\partial z}{\partial \mathbf{w}^i}$ and $\frac{\partial z}{\partial \mathbf{x}^i}$.
Using chain rule as previously shown in section 3.2.3, we have:

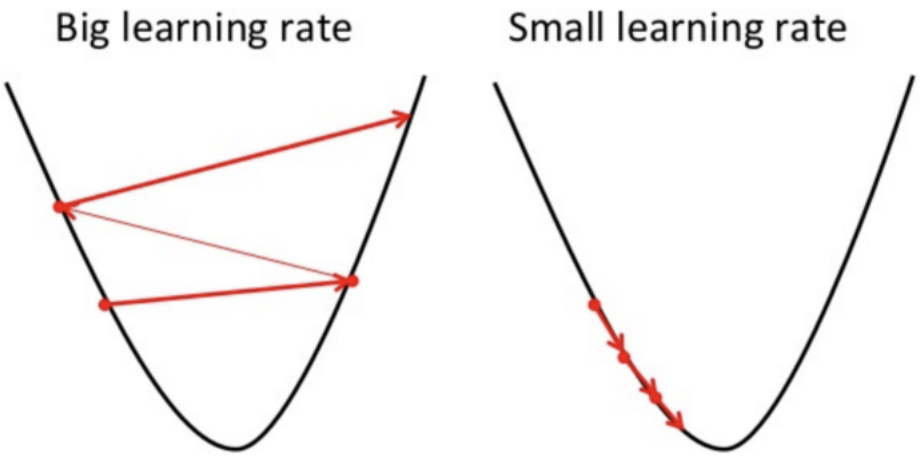


Figure 3.3: A: Big learning rate, B: Small learning rate

$$\frac{\partial z}{\partial(\text{Vec}(\mathbf{w}^i)^\intercal)} = \frac{\partial z}{\partial(\text{Vec}(\mathbf{x}^{i+1})^\intercal)} \frac{\partial \text{Vec}(\mathbf{x}^{i+1})}{\partial(\text{Vec}(\mathbf{w}^i)^\intercal)}, \tag{3.8}$$

$$\frac{\partial z}{\partial(\text{Vec}(\mathbf{x}^i)^\intercal)} = \frac{\partial z}{\partial(\text{Vec}(\mathbf{x}^{i+1})^\intercal)} \frac{\partial \text{Vec}(\mathbf{x}^{i+1})}{\partial(\text{Vec}(\mathbf{x}^i)^\intercal)} \tag{3.9}$$

Since $\frac{\partial z}{\partial \mathbf{x}^{i+1}}$ is already computed and stored in the stack, we only need to perform matrix reshaping operation (Vec) and transpose operation to compute $\frac{\partial z}{\partial(\text{Vec}(\mathbf{x}^{i+1})^\intercal)}$, which is the first term on the right hand side of Equations (3.8 and 3.9). To finally get the left hand side, we need to compute $\frac{\partial \text{Vec}(\mathbf{x}^{i+1})}{\partial(\text{Vec}(\mathbf{w}^i)^\intercal)}$ and $\frac{\partial \text{Vec}(\mathbf{x}^{i+1})}{\partial(\text{Vec}(\mathbf{x}^i)^\intercal)}$.
However, $\frac{\partial \text{Vec}(\mathbf{x}^{i+1})}{\partial(\text{Vec}(\mathbf{w}^i)^\intercal)}$ and $\frac{\partial \text{Vec}(\mathbf{x}^{i+1})}{\partial(\text{Vec}(\mathbf{x}^i)^\intercal)}$ are easier to compute than directly computing $\frac{\partial z}{\partial(\text{Vec}(\mathbf{x}^i)^\intercal)}$ and $\frac{\partial z}{\partial(\text{Vec}(\mathbf{w}^i)^\intercal)}$, because $\mathbf{x}^i$ is directly related to $\mathbf{x}^{i+1}$, via a function with parameters $\mathbf{w}^i$. For a moment, let's keep in mind these partial derivatives, as we will discuss it later in detail.

3.5.4 Importance of Learning Rate

How big the steps taking to the direction of the local minimum are determined by the so-called learning rate. It determines how fast or slow we will reach the optimal weights. For Gradient Descent to reach its local minimum, we have to choose an appropriate value for the learning rate, which is neither too low nor too high. If the steps are too big, it maybe not reach the local minimum because it will be bouncing back and forth between the convex function of gradient descent as shown on the left side of Figure 3.3. If the learning rate is choosen to be very small, gradient descent will eventually reach the local minimum but it will take too much time as shown on the right side of Figure 3.3.

3.6 Layers: Input, Output and Notations

So far, CNN architecture is clear, let's now focus on the different types of layers in CNN. To proceed, we would need to refine our notations. Assumption: consider the l-th layer, having tensor $\mathbf{x}^l$, input of an order 3, with $\mathbf{x}^l \in \mathbb{R}^{H^l \times W^l \times D^l}$, using index set$(i^l, j^l, d^l)$, to locate any specific element in $\mathbf{x}^i$. The index (i^l, j^l, d^l) refers to a single element in $\mathbf{x}^l$, which is in the d^l-th channel, and at a spatial point of (i^l, j^l) rows and column respectively. Practically, mini-batch method is usually used. Then

$\mathbf{x}^l$ becomes tensor of order 4 in $\mathbf{x}^l \in \mathbb{R}^{H^l \times W^l \times D^l \times N}$, where N is the mini-batch size.
In the next step, we assume the following zero based indexing convention.

- For simplicity in this evaluation, let $N = 1$
- $0 \leq i^l < \mathbf{H}^l$
- $0 \leq j^l < \mathbf{W}^l$
- $0 \leq d^l < \mathbf{D}^l$

In the l-th layer, a function will transform the input $\mathbf{x}^l$ to an output $\mathbf{y}$, which is considered as an input to the next layer. Important point: We can then say that $\mathbf{y}$ and $\mathbf{x}^{l+1}$ in fact refers to the same object, and let's assume that the output has size $\mathbf{H}^{l+1} \times \mathbf{W}^{l+1} \times \mathbf{D}^{l+1}$, and an element in the output is indexed by $(i^{l+1}, j^{l+1}, d^{l+1})$, $0 \leq i^{l+1} < \mathbf{H}^{l+1}$, $0 \leq j^{l+1} < \mathbf{W}^{l+1}$, $0 \leq d^{l+1} < \mathbf{D}^{l+1}$.

3.6.1 The ReLu Layer

A Rectified Linear Unit layer (ReLu) does not in any form change the input size, that is, $\mathbf{x}^l$ and $\mathbf{y}$ share the same size. In fact, the Rectified Linear Unit can be regarded as a truncation performed individually for each element in the input:

$$y_{i,j,d} = max\{0, \mathbf{x}^j_{i,j,d}\}, \tag{3.10}$$

$$\text{with :}$$
$$0 \leq i < \mathbf{H}^l,$$
$$0 \leq j < \mathbf{W}^l,$$
$$0 \leq d < \mathbf{D}^l.$$

Hence, there is no need for parameter learning in ReLU, because there is no parameter inside a ReLU layer. From Equation (3.10), it is obvious that

$$\frac{dy_{i,j,d}}{d\mathbf{x}^l_{i,j,d}} = [\mathbf{x}^l_{i,j,d} > 0], \tag{3.11}$$

where [.] is an indicator function, being 1 if its argument is true, and 0 otherwise.

Therefore, we have

$$\left[\frac{\partial z}{\partial \mathbf{x}^l}\right]_{i,j,d} = \begin{cases} \left[\frac{\partial z}{\partial \mathbf{y}}\right]_{i,j,d} & if \;\; \mathbf{x}^l_{i,j,d} > \;\; 0 \\ 0 \qquad otherise. \end{cases}$$

Note that $\mathbf{y}$ is an alias for $\mathbf{x}^{l+1}$. It should be noted that, the function $max(0, x)$ is not differentiable at $\mathbf{x} = 0$, therefore, Equation (3.11) is a bit problematic in theory. However, in practice, it is not an issue and ReLU can be used safely.

The main idea of using ReLU is to increase the nonlinearity of the CNN. Since the semantic information in speech and image are obviously nonlinear. The nonlinearity nature of ReLU function is illustrated in Figure 3.4

Assuming that $\mathbf{x}^l_{i,j,d}$ is one of the $\mathbf{H}^l \times \mathbf{W}^l \times \mathbf{D}^l$ features extracted by CNN layers 1 to $l-1$, which can either be positive, negative or zero. For instance, $\mathbf{x}_{i,j,d}$ may be positive if a region inside the input spectrum has certain patterns and $\mathbf{x}_{i,j,d}$ is negative or zero when such region does not exhibit

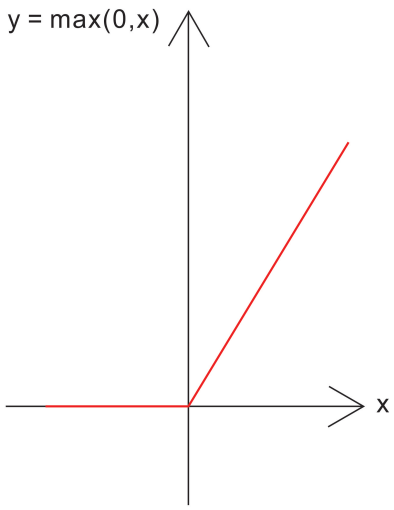


Figure 3.4: The ReLu function

these patterns. Hence, ReLU layer will map all negative values to 0, this means that $\mathbf{y}_{i,j,d}$ will be predicted only for spectrum possessing these patterns. Intuitively, this property is very useful for spotting complex patterns and objects.
There are many Other nonlinear transformations, which have been used in the researches of neural network to produce nonlinearity, for example, the logistic sigmoid function, which is given as:

$$y = \sigma(x) = \frac{1}{1 + exp(-x)}. \tag{3.12}$$

But logistic sigmoid works extremely worse than ReLU in CNN. Note that y is bounded between 0 and 1 if a sigmoid function is used, and

$$\frac{dy}{dx} = y(1-y), \ we \ have \ \frac{dy}{dx} \leq \frac{1}{4}. \tag{3.13}$$

During the backpropagation process, the gradient $\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y}\frac{dy}{dx}$ will have a smaller magnitude than $\frac{\partial z}{\partial y}$. This implies that, a sigmoid layer will reduce the magnitude of the gradient, and after series of sigmoid layer, the gradient will vanish (all its component will be very close to zero), which can create problem in gradient based learning.

In contrary, the ReLU layer maps the gradient of some features, which are less than the threshold in the l-th layer to 0, and these features are not activated (That is, they are not of interested again). For those futures that are activated, the gradient is back propagated without any change, which is good for SGD learning. The emergence of ReLU as a replacement for sigmoid is an important development in CNN, which significantly reduces the difficulty in learning CNN parameters and improves its accuracy. There are also more complex variants of ReLU, for example, parametric ReLU, Leaky ReLU and exponential linear unit, which are summarized in section 3.6 of this research.

3.6.2 Convolutional Layer

This layer plays an important role in CNN models by focusing on the use of learnable filters/kernels, which are usually small in spatial dimension but spread along the entire depth of the input feature.

The input feature of an audio signal is treated as picture, when the feature hits the convolutional layer, the layer convolve with each filter along the spatial dimension of the input to output a 2D activation map. As the kernel slides through the input, scalar product is calculated for each value in the kernel, and the network will learn which of its kernels that *fires* when they see a certain feature at a given spatial position of the input. This is known as activation. The center element of the kernel is positioned over the input vector, which is then calculated and replaced with a self weighted sum and any nearby pixels.
Training Artificial Neural Networks on input such as speech yields a model which is too big to train effectively. This generated the idea of connecting every neuron in a convolutional layer to only small portion of the input volume, and the dimensionality is called receptive field size of the neuron. The magnitude of the depth is usually approximately equals to the depth of the input.
For example, if the network has an input of dimension ($64 \times 64 \times 3$), and the receptive field is set to (6×6), we would have a total of 108 ($6 \times 6 \times 3$) weights on each neuron within the convolutional layer, where 3 is the magnitude of connectivity across the depth of the volume, unlike other ANNs models which would have yielded 12, 288 ($64 \times 64 \times 3$) weights. In addition, a CNN model reduces complexity by optimizing the output through the hyper parameters, which include; the depth, stride and zero-padding.

However, the depth of the output volume can be tuned through the number of neurons within the layer to the same region of the input. Reducing this hyperparameter will reduce the total number of neurons in the network, bas well as the accuracy of the model. We can also define the stride at which the kernel slide over the input feature. In this case, it is very important that we define to the effect that no useful information is lost. The value of the stride determine the amount of overlap in the layer. Zero-padding is used to moderate the dimensionality of the input feature and the output volume.

Given V as the input volume, R as the receptive field size, Z is the amount of zero padding, and S is the stride, the amount of padding can be calculated as follows:

$$\frac{(V - R) + 2Z}{S + 1}$$

If the calculated value doesn't give an integer, it implies that the stride has not been correctly set, and the neurons will be unable to accurately fit across the given input.

Despite efforts to curtail the complexity of the CNN model, the networks are still enormous. However, methods such as parameter sharing has been developed to curtail the overall number of parameter within the convolution layer. It works on the assumption that if a certain region is useful to compute at a set region, then it is more likely to be useful in some other region. Constraining each individual activation map within the output volume to the same weights and bias, there will be an enormous reduction in the number of parameters being produced by the convolutional layer. As a result, when backpropagation stage occurs, each neuron in the output represents the overall gradient, which can be summed up across the depth and thereby updating a single set of weights, as opposed to other ones.

3.6.3 Summary of Mathematics in Convolution Layer

Assuming we want to convolve a matrix with one convolution kernel(filter), and suppose our input feature is 3×4 dimension, and the kernel dimension is 2×2 as shown in the Figure 3.6
If we overlap the convolution kernel on top of the input matrix, we can calculate the product between each pairs of numbers corresponding to the same location in the kernel and the input, and we sum the products of these numbers to get a single number. For instance, if we overlap the filter with the top left portion in the input matrix, we can compute the convolution result at that spatial location as: $(1 \times 1) + (1 \times 4) + (1 \times 2) + (1 \times 5) = 12$. We then move the kernel down by one stride and compute the next convolution result as $(1 \times 4) + (1 \times 7) + (1 \times 5) + (1 \times 8) = 24$. We keep striding the kernel

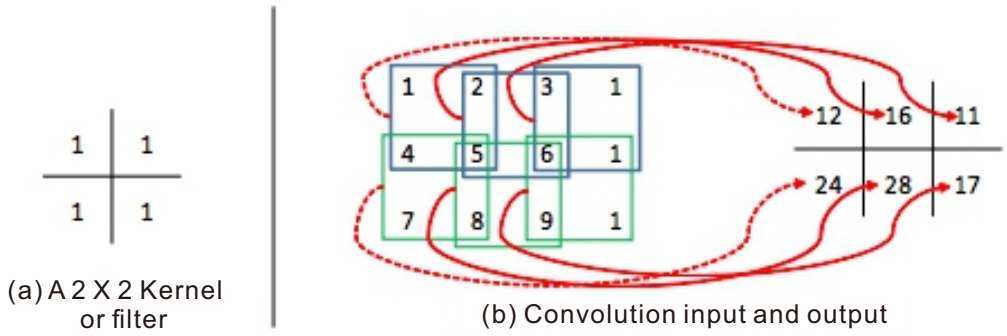


Figure 3.5: Simple Illustration [1]

down till it gets to the bottom border of the input matrix. Then, we return the kernel to the top, and move the kernel to its right by one element (pixel). We repeat the convolution for every possible pixel location until we have moved the kernel to the bottom right corner of the input matrix, as shown in Figure 3.5.

Suppose the input in the $l-th$ layer is a tensor of an order 3, with size $\mathbf{H}^l \times \mathbf{W}^l \times \mathbf{D}^l$. A convolution filter(kernel) is also a tensor of order 3, with size $\mathbf{H} \times \mathbf{W} \times \mathbf{D}^l$. When we overlap the kernel on top of the input tensor at the spatial location of (0, 0, 0), we then calculate the products of corresponding elements in all the $\mathbf{D}^l$ channels, then sum the $\mathbf{HWD}^l$ products to obtain the convolution result at this spatial location. Next, we stride the kernel from top to bottom and from left to right to complete the convolution.

A stack or multiple convolution kernels are usually used in a convolution layer. Assuming D kernels are used and each kernel spans a spatial dimension of $\mathbf{H} \times \mathbf{W}$, we represent all the kernels as f, where f is a tensor of order 4, being $\in \mathbb{R}^{\mathbf{H} \times \mathbf{W} \times \mathbf{D}^l \times \mathbf{D}}$. In the same manner, we use index variables $0 \leq \mathbf{i} < \mathbf{H}$, $0 \leq \mathbf{j} < \mathbf{W}$, $0 \leq \mathbf{d}^l < \mathbf{D}^l$ and $0 \leq \mathbf{d} < \mathbf{D}$ to indicate a specific element in the kernels. It is important to note that the set of kernels $\mathbf{f}$ refers to the same object as the notation $\mathbf{w}^l$ in Figure 3.3, and even if the mini-batch method is introduced as in the case of this research, the kernels remain unchanged.

As shown in Figure 3.6, there is spatial compression in the output dimension, compare to the spatial dimension of the input so long as the convolution kernel is larger than 1×1. Sometimes, we do a simple zero padding trick if we want the input and output feature to have the same height and width. For example, if the input dimension is $\mathbf{H}^l \times \mathbf{W}^l \times \mathbf{D}^l$ and the kernel size is $\mathbf{H} \times \mathbf{W} \times \mathbf{D}^l \times \mathbf{D}$, the output of the convolution result has size ($\mathbf{H}^l$-$\mathbf{H}$ + 1) x ($\mathbf{W}^l$-$\mathbf{W}$ + 1) x $\mathbf{D}$. In every channel of the input, if we pad $\left\lfloor \frac{\mathbf{H}-1}{2} \right\rfloor$ rows above the first row and $\left\lceil \frac{\mathbf{H}}{2} \right\rceil$ rows below the last row, also insert $\left\lceil \frac{\mathbf{W}-1}{2} \right\rceil$ columns to the left of the first column and $\left\lfloor \frac{\mathbf{W}}{2} \right\rfloor$ columns to the right of the last column of the input, the convolution result will be $\mathbf{H}^l \times \mathbf{W}^l \times D$ in size, That is, having the same spatial dimension as the input. $\lfloor . \rfloor$ is the floor functions. Zeros are usually use for padding the rows and the columns, though other values are also possible.

Stride is another important term used in convolution. In Figure 3.6, we convolve the kernel with the input matrix at every possible spatial location, which corresponds to the value of stride $s = 1$. However, if $s > 1$, every shifting of the kernel skips $s - 1$ locations (in other words, the convolution is performed once over every mel frequency cepstral coefficient (MFCC) both horizontally and vertically). Let's consider a simple case when the stride is 1 and no padding is used. Therefore, we have $\mathbf{y}$ or $(\mathbf{x}^{l+1})$ in $\mathbb{R}^{\mathbf{H}^{l+1}\times\mathbf{W}^{l+1}\times\mathbf{D}^{l+1}}$, with $\mathbf{H}^{l+1} = \mathbf{H}^l - \mathbf{H} + 1$, $\mathbf{W}^{l+1} = \mathbf{W}^l - \mathbf{W} + 1$, and $\mathbf{D}^{l+1} = \mathbf{D}$. The convolution procedure can be presented as:

$$\mathbf{y}_i^{l+1}, \mathbf{j}^{l+1}, \mathbf{d} = \sum_{i=0}^{\mathbf{H}} \sum_{j=0}^{\mathbf{W}} \sum_{d^l=0}^{\mathbf{D}^l} \mathbf{f}_{i,j,d^l,d} \times \mathbf{x}^l_{i^{(l+1)}+i, j^{(l+1)}+j, d^l}. \tag{3.14}$$

Equation (3.14) is repeated for all $0 \leq d \leq D = D^{l+1}$, and for all spatial location (i^{l+1}, j^{l+1}) that satisfies:

$$0 \leq i^{l+1} < \mathbf{H}^l - \mathbf{H} + 1 = \mathbf{H}^{l+1},$$
$$0 \leq j^{l+1} < \mathbf{W}^l - \mathbf{W} + 1 = \mathbf{W}^{l+1}.$$

In this equation,

$$\mathbf{x}_{i^{l+1}+i, j^{l+1}+j, d^l}$$

means element $\mathbf{x}^l$ indexed by the triplet $(i^{l+1} + i, j^{l+1} + j, d^l)$.
A bias factor b_d is usually added to $\mathbf{y}_i^{l+1}, \mathbf{j}^{l+1}, \mathbf{d}$ for regularization. A summary of convolution in neural network is shown in Figure 3.6, which depicts different processes, as applied in neural network training.
After convolution, a common practice is to reduce the size of a matrix by selecting the maximum values in the window, and padding it if need be, here in this example, we are applying padding of one, this is illustrated in Figure 3.7. After pooling, we apply our activation function "ReLU".

3.6.4 Reducing Complexity in Convolution: Product

Equation (3.14) looks pretty complex and time consuming. Hence, we look for a way of expanding $\mathbf{x}^l$ and simplify the convolution as a matrix product.
Before going into detail of this section, it's worth understanding the library **im2col** in python, MATLAB and many other programming languages. Given that $B = im2col(A, [m\ n], 'distinct')$, the library will rearrange discrete values of size m by n into columns, and returns the concatenated columns in matrix B . The $im2col$ function pads matrix A, if need be.
Let's consider example in Figure 3.5, and assuming a special case with $\mathbf{D}^l = \mathbf{D} = 1$, $\mathbf{H} = \mathbf{W} = 2$, and $\mathbf{H}^l = 3$, $\mathbf{W}^l = 4$. This means we are considering a channel of size 3×4 matrix (or the input feature) with a filter of size 2×2.

$$\underset{\text{Input matrix}}{\begin{bmatrix}1 & 2 & 3 & 1\\ 4 & 5 & 6 & 1\\ 7 & 8 & 9 & 1\end{bmatrix}} * \underset{\text{Filter}}{\begin{bmatrix}1 & 1\\ 1 & 1\end{bmatrix}} = \underset{\text{Output matrix}}{\begin{bmatrix}12 & 16 & 11\\ 24 & 28 & 17\end{bmatrix}}$$

Now, running a python library $B = im2co(A, [2, 2])$, we can have B as an expanded form of A:

$$\underset{\text{Matrix B}}{\begin{bmatrix}1 & 4 & 2 & 5 & 3 & 6\\ 4 & 7 & 5 & 8 & 6 & 9\\ 2 & 5 & 3 & 6 & 1 & 1\\ 5 & 8 & 6 & 9 & 1 & 1\end{bmatrix}}$$

Carefully looking at matrix B, we can observe that the first column of B correspond to the 2×2 portion in A, in a column-first order, which corresponds to $(\mathbf{i}^{l+1}, \mathbf{j}^{l+1}) = (0, 0)$. Also, the second to the last column of B correspond to the portion in A with $(\mathbf{i}^{l+1}, \mathbf{j}^{l+1})$ being $(1, 0), (0, 1), (1, 1), (0, 2)$ and $(1, 2)$

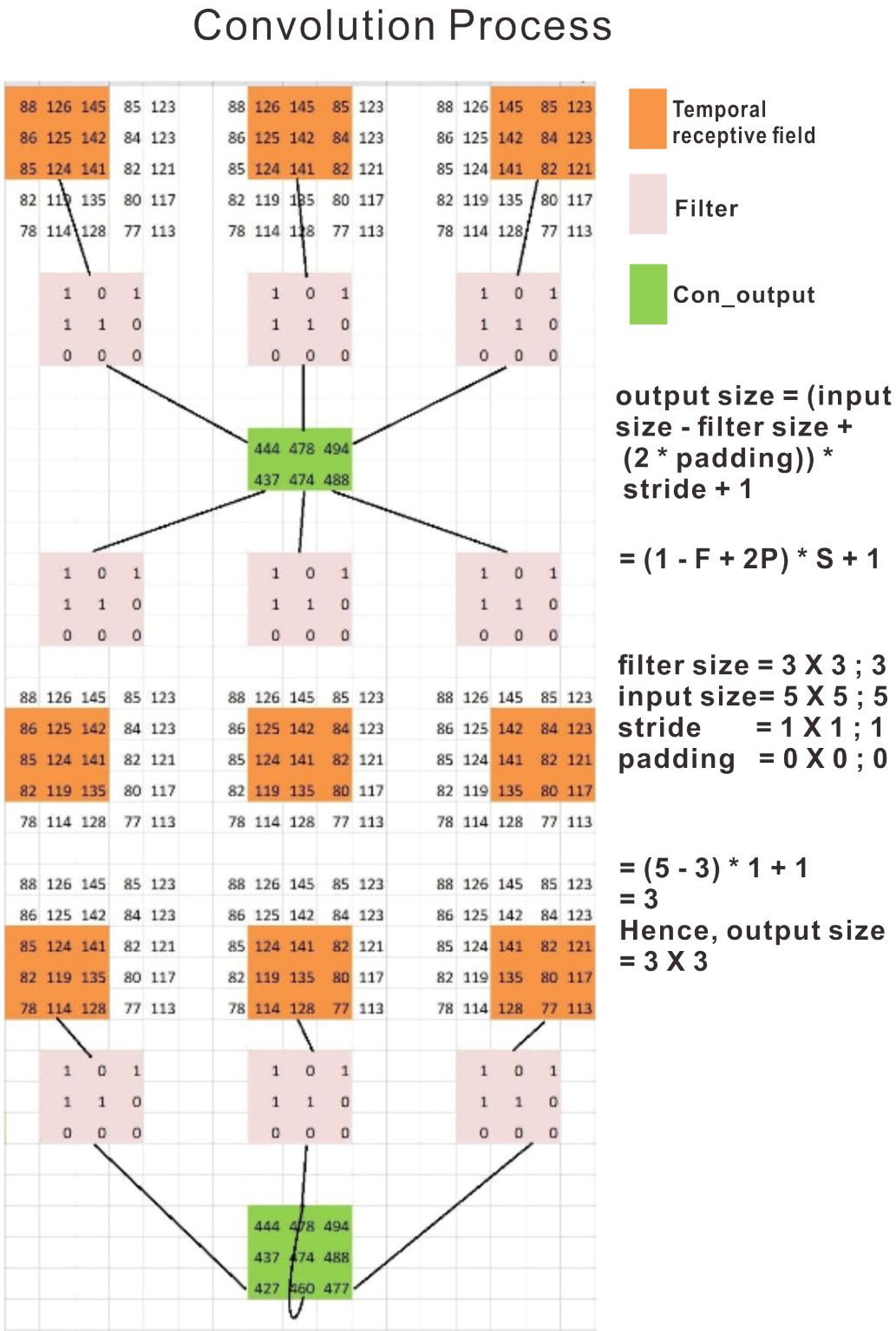


Figure 3.6: Complex Illustration [2]

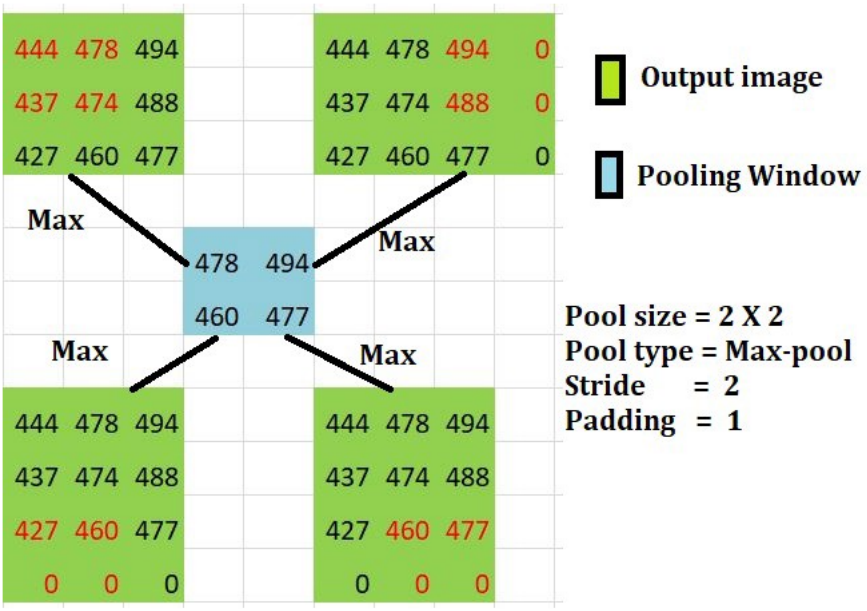


Figure 3.7: Max-Pooling [3]

respectively. The transpose of matrix B is known as the $im2row$ expression of A.
We will now vectorize the convolution kernel itself into a vector as:

$$\mathbf{B}^{\mathsf{T}} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 12 \\ 24 \\ 16 \\ 28 \\ 11 \\ 17 \end{bmatrix} \tag{3.15}$$

Reshaping Equation (3.15) will yield:

$$\begin{matrix} \{12 & \{16 & \{11 \\ 24\} & 28\} & 17\} \end{matrix} = \begin{bmatrix} 12 & 16 & 11 \\ 24 & 18 & 17 \end{bmatrix}$$

Which corresponds to the result of Figure 3.6.

3.6.5 General Form

If we reshape the vector in Equation (3.15) properly, we will get the output matrix resulting from convolution of the input matrix and the filter, as shown previously. Therefore, convolution is a linear operation. Hence, we can multiply the expanded input matrix and the vectorized filter to output a result vector, then reshaping this vector properly will give us the correct convolution results.

Henceforth, we can generalize this idea to more complex computations and formalize them. If $\mathbf{D}^l > 1$ (i.e., the input $\mathbf{x}^l$ channels is more than one), the expansion operator will first expand the first channel in $\mathbf{x}^l$, followed by the second channel, and so on till all $\mathbf{D}^l$ channels are expanded. The expanded channels will be stacked together, which implies, one row in the $im2row$ expansion will have $\mathbf{H}\times\mathbf{W}\times\mathbf{D}^l$ elements, instead of $\mathbf{H}\times\mathbf{W}$.
In a more formal way, assuming $\mathbf{x}^l$ is a third order tensor in $\mathbb{R}^{H^l\times W^l\times D^l}$, with one element in $\mathbf{x}^l$, having a triplet index of $(\mathbf{i}^l, \mathbf{j}^l, \mathbf{d}^l)$. Let us consider a set of convolution kernels $\mathbf{f}$, whose spatial extent

are all $\mathbf{H} \times \mathbf{W}$. Therefore, the $(im2row)$ expansion operator will convert $\mathbf{x}^l$ into a matrix $\phi(\mathbf{x}^l)$. And we will use two indexes (p, q) to index an element in this matrix. Therefore, the expansion operator will copy element at $(\mathbf{i}^l, \mathbf{j}^l, \mathbf{d}^l)$ in $\mathbf{x}^l$ to the $(p, q) - th$ entry in $\phi(\mathbf{x}^l)$.

We can therefore compute the corresponding $(\mathbf{i}^l, \mathbf{j}^l), \mathbf{d}^l$ triplet, given a fixed (p, q), since the following equations hold:

$$p = \mathbf{i}^{l+1} + (\mathbf{H}^l - \mathbf{H} + 1) \times \mathbf{j}^{l+1}, \tag{3.16}$$

$$q = \mathbf{i} + \mathbf{H} \times \mathbf{j} + \mathbf{H} \times \mathbf{W} \times \mathbf{d}^l, \tag{3.17}$$

$$\mathbf{i}^l = \mathbf{i}^{l+1} + \mathbf{i}, \tag{3.18}$$

$$\mathbf{j}^l = \mathbf{j}^{l+1} + \mathbf{j}. \tag{3.19}$$

If we divide Equation (3.17) by $\mathbf{H} \times \mathbf{W}$ and ceil it (i.e., take the integer part of the quotient), we can determine which channel $(\mathbf{d}^l)$ belongs to. In this same manner, we can find the offsets inside the convolution kernel as (i, j) such that $0 \leq \mathbf{i} < \mathbf{H}$, and $0 \leq \mathbf{j} < \mathbf{W}$. Also, q determines only one specific spatial position inside the convolution kernel by the triplet $(\mathbf{i}, \mathbf{j}, \mathbf{d})$.

Remember that the result of the convolution is $\mathbf{x}^{l+1}$, with spatial extent defines as: $\mathbf{H}^{l+1} = \mathbf{H}^l - \mathbf{H} + 1$, and $\mathbf{W}^{l+1} = \mathbf{W}^l - \mathbf{W} + 1$. If we now divide Equation (3.26) by $\mathbf{H}^{l+1} = \mathbf{H}^l - \mathbf{H} + 1$, it will give us the offset in the convolved result $(\mathbf{i}^{l+1}, \mathbf{j}^{l+1})$, or the top left spatial part of the region in $\mathbf{x}^l$, which is to be convolved with the kernel.

From the definition of convolution, we can use Equations (3.18 and 3.19) to find the offset in the input $\mathbf{x}^l$ as $\mathbf{i} = \mathbf{i}^{l+1} + \mathbf{i}$ and $\mathbf{j} = \mathbf{j}^{l+1} + \mathbf{j}$. To be more precise, the mapping from (p, q) to $(\mathbf{i}^i, \mathbf{j}^l, \mathbf{d})$ is one-to-one, but it should noted that the reverse mapping from $(\mathbf{i}^i, \mathbf{j}^l, \mathbf{d})$ to (p, q) is one-to-many, an essential part of neural networks that is used for backpropagation rules in convolution layer.
We will now employ the standard Vec operator to convert the set of convolution kernels $\mathbf{f}$ (tenssor of order 4) into a matrix. Start from one kernel, which can be vectorized into a vector in $\mathbb{R}^{HWD^l}$. Thus, we will reshape all convolution kernels into a matrix with HWD^l rows and D columns (recall that $D^{l+1} = D$), let this matrix be called F.

And finally, we arrive at a generalized equation to calculate convolution results with $\phi(\mathbf{x}^l)$ equals B^T:

$$\text{Vec}(\mathbf{y}) = \text{Vec}(\mathbf{x}^{l+1}) = \text{Vec}(\phi(\mathbf{x}^l)F) \tag{3.20}$$

NOTE:

$$\text{Vec}(\mathbf{y}) \in \mathbb{R}^{H^{l+1}W^{l+1}D},$$
$$\phi(\mathbf{x}^l) \in \mathbb{R}^{(H^{l+1}W^{l+1})X(HWD^l)},$$
$$and \quad F \in \mathbb{R}^{(HWD^l)XD}.$$

The matrix multiplication $\phi(\mathbf{x}^l)F$ yields a matrix of size $(\mathbf{H}^{l+1}\mathbf{W}^{l+1} \times D)$. And the vectorization of this resultant matrix yields a vector in $\mathbb{R}^{H^{l+1}W^{l+1}D}$, which corresponds to the dimension of $\text{Vec}(\mathbf{y})$

3.6.6 The Kronecker Product

A short route to the Kronecker product is needed to compute the derivatives.
Given:
Matrix $A \in \mathbb{R}^{m \times n}$ and matrix $B \in \mathbb{R}^{p \times q}$, the Kronecker product $A \otimes B$ is a $mp \times nq$ matrix, described as a block matrix

$$A \otimes B = \begin{bmatrix} a_{11}\mathbf{B} & \cdots & a_{1n}\mathbf{B} \\ \vdots & \ddots & \vdots \\ a_{m1}\mathbf{B} & \cdots & a_{mn}\mathbf{B} \end{bmatrix}$$

The following properties should be noted in the Kronecker product, which will be useful in this research work:

$$(\mathbf{A} \otimes \mathbf{B})^\intercal = \mathbf{A}^\intercal \otimes \mathbf{B}^\intercal \tag{3.21}$$

$$\text{Vec}(\mathbf{A} \times \mathbf{B}) = (\mathbf{B} \otimes \mathbf{A})\text{Vec}(\mathbf{X}) \tag{3.22}$$

Matrices $\mathbf{A}, \mathbf{X}$, and $\mathbf{B}$ must satisfy the multiplication principle of matrix, the dimension of matrices need to be multiplied must agree with matrix multiplication rules. We can therefor make use of $\otimes$ operator as follows:

$$\text{Vec}(\mathbf{y}) = \text{Vec}(\phi(\mathbf{x}^l)\mathbf{F}\mathbf{I}) = (\mathbf{I} \otimes \phi(\mathbf{x}^l))\text{Vec}(\mathbf{F}), \tag{3.23}$$

$$\text{Vec}(\mathbf{y}) = \text{Vec}(\mathbf{I}\phi(\mathbf{x}^l)\mathbf{F}) = (\mathbf{F}^\intercal \otimes \mathbf{I})\text{Vec}(\phi(\mathbf{x}^l)) \tag{3.24}$$

Thus, $\mathbf{I}$ is an identity matrix of proper size, whose size is determined by the number of column in $\mathbf{F}$, $\mathbf{I} \in \mathbb{R}^{D \times D}$ in Equation (3.23). Also, in Equation (3.24), $\mathbf{I} \in \mathbb{R}^{(\mathbf{H}^{l+1}\mathbf{W}^{l+1}) \times (\mathbf{H}^{l+1}\mathbf{W}^{l+1})}$.

3.6.7 Backpropagation Recap

As previously stated, we need to calculate two derivatives, i.e $\frac{\partial z}{\partial \text{Vec}(\mathbf{x}^l)}$, which will be used for back-propagation to the previous $((l-1)-th)$layer, and $\frac{\partial z}{\partial \text{Vec}(\mathbf{f})}$, which will be the basis of how the current $(l-th)$ layer will be updated. From Table 3.1, it should be noted that $\mathbf{f}$, $\mathbf{F}$ and $\mathbf{w}^i$ represents the same parameter (reshaping of the vector or tensor or matrix). We can also reshape $\mathbf{y}$ into a matrix $\mathbf{Y} \in \mathbb{R}^{(\mathbf{H}^{l+1}\mathbf{W}^{l+1}) \times \mathbf{D}}$, then $\mathbf{y}$, $\mathbf{Y}$ and $\mathbf{x}^{l+1}$ represents the same parameter (modulo reshaping).

From chain rule Equation (3.8), it is easy to compute $\frac{\partial z}{\partial \text{Vec}(\mathbf{f})}$ as:

$$\frac{\partial z}{\partial (\text{Vec}(\mathbf{F}))^\intercal} = \frac{\partial z}{\partial (\text{Vec}(\mathbf{Y})^\intercal)} \frac{\partial \text{Vec}(\mathbf{y})}{\partial (\text{Vec}(\mathbf{F})^\intercal)} \tag{3.25}$$

The first term on the right hand side of Equation (3.21) is already computed in the $(l+1)-th$ layer as $\frac{\partial z}{\partial (\text{Vec}(\mathbf{x}^{l+1}))^\intercal}$. The second term is also easy to compute as follows:

$$\frac{\partial z}{\partial (\text{Vec}(\mathbf{F})^\intercal)} = \frac{\partial ((\mathbf{I} \otimes \phi(\mathbf{x}^l))\text{Vec}(\mathbf{F}))}{\partial (\text{Vec}(\mathbf{y})^\intercal)} = \mathbf{I} \otimes \phi(\mathbf{x}^l) \tag{3.26}$$

Subject to the fact that $\frac{\partial \mathbf{X}\mathbf{a}^\intercal}{\partial \mathbf{a}} = \mathbf{X}$ or $\frac{\partial \mathbf{X}\mathbf{a}}{\partial \mathbf{a}^\intercal} = \mathbf{X}$, under the condition that the matrix multiplications are well defined. Hence, this gives:

$$\frac{\partial z}{\partial (\text{Vec}(\mathbf{F}))^\intercal} = \frac{\partial z}{\partial (\text{Vec}(\mathbf{y})^\intercal)} (\mathbf{I} \otimes \phi(\mathbf{x}^l)) \tag{3.27}$$

Taking the transpose, we have:

$$\frac{\partial z}{\partial \text{Vec}(\mathbf{F})} = (\mathbf{I} \otimes \phi(\mathbf{x}^l))^\intercal \frac{\partial z}{\partial \text{Vec}(\mathbf{y})} \tag{3.28}$$

$$= (\mathbf{I} \otimes \phi(\mathbf{x}^l)^\intercal)\text{Vec}\left(\frac{\partial z}{\partial \mathbf{Y}}\right) \tag{3.29}$$

$$= \text{Vec}\left(\phi(\mathbf{x}^l)^\intercal \frac{\partial z}{\partial \mathbf{Y}} \mathbf{I}\right) \tag{3.30}$$

$$= \text{Vec}\left(\phi(\mathbf{x}^l)^\intercal \frac{\partial z}{\partial \mathbf{Y}}\right) \tag{3.31}$$

Table 3.1: Variables, Alias and meanings. Note: Alias means that a variable can have different name or reshape into another form

	Alias	Size and Meaning
$\mathbf{X}$	$\mathbf{x}^l$	$\mathbf{H}^l\mathbf{W}^l \times \mathbf{D}^l$, the input tensor
$\mathbf{F}$	$\mathbf{f}, \mathbf{w}^l$	$\mathbf{HWD}^l \times \mathbf{D}$, $\mathbf{D}$ kernels, each $\mathbf{H} \times \mathbf{W}$ and $\mathbf{D}^l$ channels
$\mathbf{Y}$	$\mathbf{y}$, $\mathbf{x}^{l+1}$	$\mathbf{H}^{l+1}\mathbf{W}^{l+1} \times \mathbf{D}^{l+1}$, the output, $\mathbf{D}^{l+1} = \mathbf{D}$
$\phi(\mathbf{x}^l)$		$\mathbf{H}^{l+1}\mathbf{W}^{l+1} \times \mathbf{HWD}^l$, the im2row expansion of $\mathbf{x}^l$
$\mathbf{M}$		$\mathbf{H}^{l+1}\mathbf{W}^{l+1}$ $\mathbf{D}^{l+1} \times \mathbf{H}^l\mathbf{W}^l$ $\mathbf{D}^l$, the indicator matrix for $\phi\mathbf{x}^l$
$\frac{\partial z}{\partial \mathbf{Y}}$	$\frac{\partial z}{\partial \mathrm{Vec}(\mathbf{y})}$	$\mathbf{H}^{l+1}\mathbf{W}^{l+1} \times \mathbf{D}^{l+1}$, gradient for $\mathbf{y}$
$\frac{\partial z}{\partial \mathbf{F}}$	$\frac{\partial z}{\partial \mathrm{Vec}(\mathbf{f})}$	$\mathbf{HWD}^l \times \mathbf{D}$, gradient to update the convolution kernels
$\frac{\partial z}{\partial \mathbf{X}}$	$\frac{\partial z}{\partial \mathrm{Vec}(\mathbf{x}^l)}$	$\mathbf{H}^l\mathbf{W}^l \times \mathbf{D}^l$, gradient for $\mathbf{x}^l$, useful for back propagation

Note that Kronecker product in Equations (3.21 and 3.22) are used in the above derivation. We can therefore conclude that:

$$\frac{\partial z}{\partial \mathbf{F}} = \phi(\mathbf{x}^l)^\intercal \frac{\partial z}{\partial \mathbf{Y}}, \tag{3.32}$$

Which is used to update the parameters in the $l-th$ layer; the product of $\phi\mathbf{x}^T$(im2col expansion) and $\frac{\partial z}{\partial \mathbf{Y}}$(supervision signal from layer $(l+1)$) is the gradient with respect to the convolution parameters.

3.6.8 Pooling Layer

While convolutional layers detect local features from its input, a pooling layers gradually reduces the dimensionality of the network, and thus further reduces the number of parameters and the computational complexity of the model, by semantically merging similar features by only keeping the maximum value or by averaging the values within a patch [20]. Usually, pooling layers are used after one or a few convolutional layers, continuously scaling down the size of the input as the network gets deeper, and thereby reducing the amount of computation. However, it should be noted that potentially valued information maybe discarded during pooling.

Due to the discarding of useful information by pooling layer, there are only two noticeable methods of max-pooling. The stride and filters of the pooling layers are both tuned to 2×2, which allows the layer to extend over the spatial dimensionality of the input. consequently, overlapping pooling may be used, where the stride is set to 2 with a kernel size set to 3. Having a kernel size above 3 might greatly decrease the performance of the model.

Other than max-pooling, CNN architectures may contain general-pooling. General pooling layers are comprised of pooling neurons which are able to perform a number of common operations including $L1/L2$-normalisation, and average pooling. However, this research work made use of max-pooling.

3.6.9 Fully Connected Layer

A fully connected layer refers to a layer if the computation of any element in the output $\mathbf{x}^{l+1}$ (or $\mathbf{y}$) requires all elements in the input $\mathbf{x}^l$. This layer takes an input volume from the output of the layer preceding it (such as Convolution, ReLU or pooling), and output a N dimensional vector, where N is the number of classes to be chosen from, which is decided based on the class that has the highest value of probability in the output vector, using softmax, or using some other activation functions. Fully connected layer looks at the resulting output of the previous layer, which represent the activation map of the input feature, and tries to check which features most correlate with the particular class.

Another common form of CNN architecture with convolutional layers being stacked between ReLus continuously before passing it through the pooling layer, and before going between one or more fully connected ReLus. To reduce the amount of complexity within a convolutional layer, a good idea is to split large convolutional layers into smaller sized convolutional layers. However, it is a fully connected layer, but can be computed as a convolution layer. Hence, it is of no need to derive learning rules for a fully connected layer separately.

Chapter 4

Feature Extraction and Performance Measures

4.1 Sound as data

This section details the basic of how to use audio data for keywords spotting models, the model's block as well as basic principles behind training a neural network. The data used in this research work consist of a set of 105,000 audio wave files from google, where people are saying ten different words: "yes", "no", "up", "down", "left", "right", "on", "off", "stop", and "go". In addition, unknowns and silence were included during classification. Each file is sampled into a vector with 16,000 Hz sampling rate. A common approach for words spotting is to first extract features from the raw audio wave. The usual features used in speech include spectrograms, log-Mel filter banks, and Mel-frequency cepstral coefficients (MFCC), which convert the raw wave into a time-frequency domain [21].Wei Dai and Chia Da [22] discussed how log-mel filter banks can be used as an input feature to train a neural network model. This work uses MFCC as the input feature, as we try to mimic the logarithmic scale of human hearing model.

4.1.1 Mel Frequency Ceptral Coefficient (MFCC)

The first step in audio processing is to extract the feature components of the audio signal, which are good for identifying the content and discarding other part which carries information like noise, emotion, etc.
MFCCs were first introduced by Davis and Mermelsten in the 1980's. Prio to the emergence of MFCCs, linear prediction ceptral coefficients (LPCCs) and linear prediction coefficients(LPCs) are the main feature used for audio processing in automatic speech recognition. Below are the implementation steps taken to extract audio features:

- Frame the signal into short frames.
- Calculate the power spectrum for each of the frame.
- Apply the Mel filterbank to the power spectrum of each filter.
- Take the logarithm of all filterbank energies.
- Take the Discrete Cosine Transform of the log filterbank energies
- Keep 12 discrete cosine transformed and discard the rest

There are other few more practice commonly done, sometimes the frame energy can be appended to each feature vector. Delta and delta-delta (the first and second derivatives of the input feature respectively) features can also be appended. Liftering is also usually applied to the final features.

4.1.2 Summary of Each Step

An audio signal constantly changes, but with an assumption that it is statistically stationary on a short time scales [23]. This is why we frame the signal into $20 - 40ms$ frames. If the frame is too short, we won't have enough samples to achieve reliable spectral estimate, and if too long, the signal

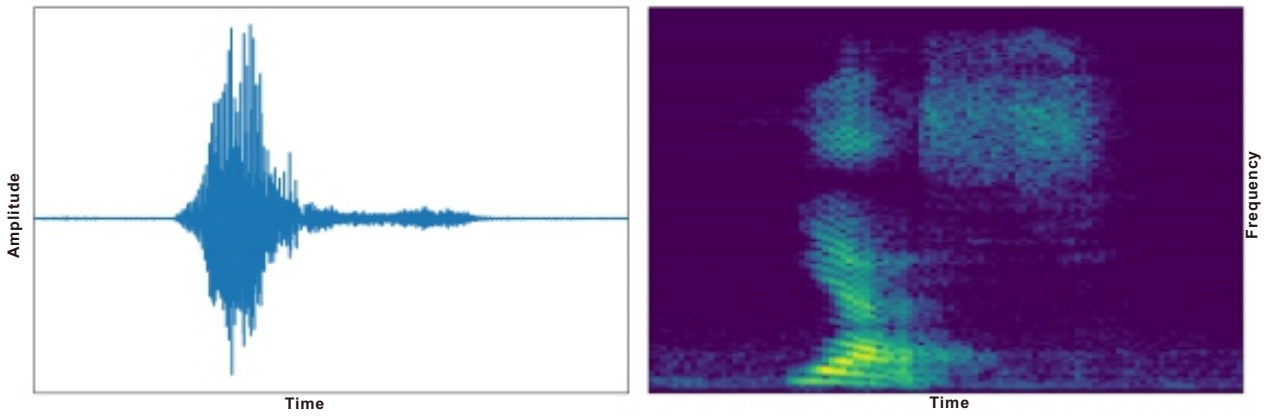


Figure 4.1: Figure showing the raw waveform (left) and log-spectrogram (right) of a word "yes" respectively

changes will be too much.

The idea of calculating the power spectrum of each frame is motivated by the human cochlea(an organ in human ear), which vibrate at different spots and wobbles small hairs in the ear according to the frequencies of the incoming sounds. Our periodogram performs the same job by identifying which frequencies are present in the frames.

Some unnecessary information which are not required for the words spotting or audio recognition are contained in the periodogram spectral estimate. In general the cochlea can not discern the difference between two frequencies that are closely spaced . This effect becomes more problematic as the frequencies increase. Therefore, we take clumps of periodogram bins and add them up to have an idea of how much energy contains in various frequency regions, which is done by the Mel filterbank: the first filter is very narrow and gives an indication of how much energy exists near zero Hertz. As the frequencies increase, the filters get wider as we become less concerned about variations, because we are interested how much energy occurs at each spot. The Mel scale indicates how to space our filterbanks and how wide to make them. See [24] for details on how to calculate the spacing.

After getting the filterbank energies, we take their logarithms. This is analogous to human hearing inability to fathom loudness on a linear scale. In order to double the percieved volume of a sound we need to put 8 times as much energy into it. if the sound is loud, large variations in energy may not sound all that different. This compression operation makes our features match more closely what humans actually hear. The effect of convolving channel coefficient with input speech can easily be removed in log ceptral domain, because this multiplication becomes additon. This is done by subtracting the ceptral mean from all the input vectors, and this is otherwise known cepstral mean subtraction, which is a channel normalisation approach [25].

Finally, we calculate the discrete cosine transform, this for two main reasons: Filterbanks overlap, which makes the filter energies highly correlated with one another. The discrete cosine transform decorrelates the energies. This implies that diagonal covariance matrices can be used to model the features. However, only 12 out of the 26 discrete cosine transform are made used, because the early coefficients contain the important information (low frequency), and the later coefficients include less-important information (higher frequency).

4.2 Performance Measures

A number of metrics are use to measure the performance of keyword detection algorithms. Getting a clear understanding of these metrics are very important for the purpose of discussing algorithms as well as understanding the results. This section gives a summary account of the metrics used to measure performance analysis of keyword detection algorithms.

4.2.1 True and Estimated Result Set

The estimate of a keyword detection system is a set of tuples, Ψ, representing putative keyword occurrences. Each tuple consists of an utterance label, a keyword tag, a start time and end time. When evaluating performance, this estimated result is scored against the true set of tuples, Γ, which the reference containing the set of keywords to be detected. The results are defined as follows:

$$\Gamma = \{\gamma_1, \gamma_2, ..., \gamma_N\} \tag{4.1}$$

$$\Psi = \{\psi_1, \psi_2, ..., \psi_N\} \tag{4.2}$$

$$where \quad \gamma_i = (u_i^\tau, w_i^\tau, s_i\tau, e_i^\tau) \tag{4.3}$$

$$\psi_j = (u_j^p, w_j^p, s_j^p, e_j^p) \tag{4.4}$$

$$u_i^\tau, u_i^p = i-th \text{ reference/putative occurrence's utterance label}$$

$$w_i^\tau, w_i^p = i-th \text{ reference/putative occurrence's keyword tag}$$

$$s_i^\tau, s_i^p = i-th \text{ reference/putative occurrence's keyword start time}$$

$$e_i^\tau, e_i^p = i-th \text{ reference/putative occurrence's keyword end time}$$

4.2.2 Hit Operator

The hit operator is used to determine whether a reference occurrence was successfully predicted. Given a reference occurrence γ and a putative result set, Ψ, then the occurrence of the true label is declared as it iff the mid-point of the true occurrence falls within the time boundaries of one of the putative occurrences in the Ψ, and the respective keyword labels and identifiers are equal. Mathematical representation on hit operation can be formulated as:

$$\gamma \ominus \Psi = \begin{cases} 0 & if \quad u_i^\tau \neq u_j^p \\ 0 & \quad\quad w_i^\tau \neq w_j^p \\ 0 & \quad\quad s_j^p < \frac{(s_i^r + e_i^r)}{2} \\ 0 & \quad\quad e_j^p < \frac{(s_i^r + e_i^r)}{2} \\ 1 & \quad\quad otherwise \end{cases} \tag{4.5}$$

4.2.3 Miss Rate

Miss rate tries to evaluate the occurrence of error in the predicted output. Given a reference set of tuple ot the targeted words Γ and a predicted result set Ψ, the miss rate is defined as the total number of elements of Γ that were not hit by at least one element in Ψ. This can be expressed as:

$$\begin{aligned} MissRate(\Gamma, \Psi) &= \frac{|\Gamma| - |HitSet(\Gamma, \Psi)|}{|\Gamma|} \\ where \quad HitSet(\Gamma, \Psi) &= \{\gamma \in |\Gamma| \sum_j (\gamma \ominus \psi_j) > 0\} \end{aligned} \tag{4.6}$$

Hit rate is defined as converse of miss rate:

$$HitRate(\Gamma, \Psi) = 1 - MissRate(\Gamma, \Psi) \tag{4.7}$$

4.2.4 False Alarm Rate (FAR)

False alarm rate is the number of false detection by the keyword spotting. Hence, false alarm can be seen as the member of the result set Ψ that does not match the true keyword in the reference set Γ. False alarm rate can be expressed as the number of false prediction by the keyword detector's result set, normalized by the total duration of the speech searched and the number of unique keywords searched for.
This can be mathematically expressed as:

$$\begin{aligned} FARate(\Gamma, \Psi, W, T) &= \frac{|FASet(\Gamma, \Psi)|}{|W| * T} \\ where \quad FASet(\Gamma, \Psi) &= \{\psi \in |\Psi| \sum_j (\gamma \ominus \psi) = 0\} \\ W &= \text{List of keywords being queried for} \\ T &= \text{Duration of speech searched in hours} \end{aligned} \tag{4.8}$$

This definition of false alarm rate is used when measuring the overall keyword detection performance of a system.

4.2.5 False Acceptance Rate

An alternative measure to the false alarm rate is the false acceptance rate, which is analogous to impurity of a result set. It measures the percentage of false alarms in the final result set. The number of false alarms in keyword detection result set, normalized by the total size of the result set. Mathematically, false alarm acceptance is defined as:

$$FaslseAcceptanceRate(\Gamma, \Psi) = \frac{|FASet(\Gamma, \Psi)|}{|\Psi|} \tag{4.9}$$

4.2.6 Receiver Operating Characteroistic Curves

A receiver operating characteristic curve (ROCs) is a plot of hit rate against false acceptance or alarm rate [26]. It provides a view of how keyword detection varies as the system is tuned. ROCs are excellent for evaluating accuracy at low false acceptance or alarm rates. However, the performance depreciates at low miss rate, making it difficult to understand how accuracy varies at low miss rates.

4.2.7 Detection Error Trade-off Plots

Detection error trade-off (DET) plot shows the plot of miss rate against false acceptance rate on a logarithm scale. Visualizing this measurements on a logarithm scale typically results in a linear operating characteristic curve that is more easier to interpret than ROC plot. Contrast to ROC, it provides good resolution at both low miss rate as well as low false acceptance rates.
However, DET may suffer from step effects within the low miss rate regions in some analyses. Such a set back indicates that the output scores of putative hits are segregated from the putative false alarms within the step region. It maybe also be observed when a small reference set is used, and this will be observed across all operating points since the lower bound of changes in miss rate would be $\frac{1}{\Gamma}$

		Predicted	
		Negative	Positive
Actual	Negative	p	q
	Positive	r	s

Table 4.1: Confusion matrix

4.2.8 Confusion Matrix

A very rare unique concept used to analyze and give a clear visual explanation of statistical classification problem is confusion matrix, also known as the error matrix. This is one of the major metrics used to describe the results of this research. The basic concept of confusion matrix is that it evaluates the number of correct and incorrect predictions, then further summarized it with the number of count values and breakdown into each classes.
A confusion matrix is a table that outlines different predictions and test results and contrasts them with real-world values [27]. Confusion matrices are used in statistics, data mining, machine learning models and other artificial intelligence (AI) applications. A confusion matrix can also be called an error matrix.
Consequently, many metrics can be depicted from a confusion matrix, a simple confusion matrix table will be used to analyze and summarize confusion matrix for better understanding. However, its entries are stated as follows:

- p is the number of correct predictions that an instance is correct
- q is the number of incorrect predictions that an instance is positive
- r is the number of incorrect predictions that an instance is negative, and
- s is the number of correct predictions that an instance is positive
- The accuracy (AC) is the proportion of the total number of predictions that were correctly predicted. It can be defined as:

$$AC = \frac{p+q}{p+q+r+s} \tag{4.10}$$

- The recall or true positive rate (TP) is the proportion of positive cases that were correctly classified, with the below equation:

$$TP = \frac{s}{r+s} \tag{4.11}$$

- The false positive rate (FP)is the proportion of negative cases that were incorrectly classified as positive:

$$FP = \frac{q}{p+q} \tag{4.12}$$

- The true negative rate (TN) is defined as the proportion of negative cases that were classified correctly:

$$TN = \frac{p}{p+q} \tag{4.13}$$

- The false negative rate (FN) is the proportion of positives cases that were incorrectly predicted as negative:

$$FN = \frac{r}{r+s} \tag{4.14}$$

- Precision (P) is the proportion of predicted positive cases that were correct:

$$P = \frac{s}{q+s} \tag{4.15}$$

- To consider a situation where the number of one case is far more than the other case, the accuracy determined above becomes inefficient. Hence, we can make use of the following for more descent evaluation:

$$g - mean_1 = \sqrt{TP * P} \tag{4.16}$$

$$g - mean_2 = \sqrt{TP * TN} \tag{4.17}$$

where $g - mean$ is the geometric mean of sensitivity.

4.2.9 Precision, Recall and F1-score

Precision is theratio of correctly predicted positive observations to the total predicted positive observations. High precision relates to low false positive rate, recall is the ratio of correctly predicted positive observations to all the observations in the actual class, and f1-Score is the weighted average of precision and recall. It takes both false positive and false negative into account, it's very useful in analyzing uneven data set. More details regarding these metrics can be found here [28].

Chapter 5

Regularization

5.1 Introduction

It has been shown that multilayer perceptions with at least one hidden layer can approximate any target mapping up to arbitrary [29]. Hence, the training data maybe over-fitted, giving a very high accuracy, while producing an extremely low accuracy during validation and testing. Regularization has been necessitated to avoid this calamity and give a better generalization performance on data which are new to the model. For clear assurance on model's performance during training, the training set is further divided into new training set and validation set. The model is then trained using the new training set and its generalization performance is evaluated using the validation set. However, there are different approaches to perform regularization. Usually, the training set is augmented to introduce some variances the network is expected to learn. Other approach could also add a regularization factor to control the complexity and avoid over-fitting as show in the equation below:

$$\tilde{E}_n(w) = E_n(w) + \eta P(w) \tag{5.1}$$

where $\tilde{E}_n(w)$ is the estimated weight, $W_n(w)$ is the input neuron weight, $P(w)$ influences the state of the solution and η is a balancing parameter.

$$P(w) = ||w||_2^2 = w^\intercal w \tag{5.2}$$

The key idea is to penalize large weights as they tend to cause overfitting.

5.2 Deterministic Regularization

Deterministic regularization method directly constrained the model based on its structure and training data. Pruning and weight penalties are the two common deterministic regularization methods used in deep learning.

5.2.1 Pruning

Pruning removes redundant neurons and scale the the size of the model down to the size required to solve the problem. Some pruning methods have been proposed to identify the redundant neurons, including skeletonization based on error gradient and optimal brain damage based on the Hessian of the error, considering a particular neuron [30]. Another effective pruning method is magnitude based pruning, which permanently drops connections that have low magnitude or whose magnitude is below the threshold, followed by training of the pruned network.

5.2.2 Weight Penalties

In weight penalty, a penalty factor is added to the cost function, to make it effective for simpler models over more complicated models, which is determined based on the weight magnitudes during training. Popular among weight penalization method is the ridge regularization, which add norm

of the network's weight to the cost function. This method continuously reduces the network weight during training.

Sparse weights are concept being used where some input features are less important or noisy. Usually, this occurs in high dimensional tasks, e.g. image classification. The idea of sparse weight can be fully exploited in this scenario. Hence, lasso regularization L_n has also been proposed [31], which adds the L_1 norm of the model weights to the cost function. Elastic regularization has also been proposed to combine both L_1 and L_2 norms of the weights [32].

Bridge regularization was also proposed by Frank and Friedman to optimize the norm of the weight penalty [33]. It has been proven that bridge regularization outperformed ridge, lasso and elastic-net in some regression problems. It was also applied in support vector machine, which produces very good results.

5.3 Stochastic Regularization

While deterministic regularization methods only depend on the network weights and the training data, stochastic methods add random noise to the model. It reduces the correlation between the neural activations, this generally results to an improved and robust performance and better generalization. It should be noted that stochastic method is equivalent to the deterministic approach when the randomness is marginalized. In reality, stochastic methods have outperformed deterministic methods of regularization in various scenarios [34].

5.3.1 Dropout

Dropout is a technique used to approximate training a large number of neural networks with different structures in parallel. While training the algorithm, some neurons' outputs are randomly dropped out with a probability of $1 - p$, that is, they are temporarily removed from the network. It makes the training process noisy, ensuring nodes within a layer to probabilistically take on more or less responsibility for the inputs. It simulates a sparse activation from a given layer, which in turn forced the network to actually learn a sparse representation as a side-effect. A random binary mask vector $m = [m_1.....m_d]^T$ is sampled from a Bernoulli distribution with probability p

$$m \sim Bernoulli(p).$$

The random mask m is scaled with $\frac{1}{p}$, so additional changes are not needed during the model testing. The random mask is multiplied with the outputs of the preceding layer and the result is calculated as:

$$\tilde{a}^{l-1} = a^{l-1} \odot \frac{m}{p}, \tag{5.3}$$

$$a^l = \sigma(W_a^{l \sim l-1} + b^l), \tag{5.4}$$

With $\odot$ is the elementwise product, a and l are the activation function and layer respectively. In terms of weight disruption, dropout either turns off or scales all the outgoing weights from a neuron as stated below:

$$\tilde{W}_{i,j} = \begin{cases} 0 \quad if \quad m_j = 0, \\ \\ \frac{i}{p} W_{:,j} \quad if \quad m_j = 1 \end{cases} \tag{5.5}$$

However, the phenomenon of randomly dropping neurons in the network forces neurons to learn useful representations on their own rather than depending on other neurons. The weights are scaled with p during testing, equivalent to the effect of averaging over an ensemble of models. Such an aggregate of models is equivalent to an approximation to the Bayesian model averaging. For linear regression problems, dropout has been shown to be equivalent to penalizing the weights with L_2 norm.

5.3.2 Shakeout

Shakeout is another regularization technique used for both linear and multi-layer neural networks. The output units in one layer are the input features during the feed-forward computation of the units in the next layer, and the shakeout noises are introduced to the previous layer of the unit.
Considering the forward computation from layer l to layer $l+1$, for a fully-connected layer, the shakeout forward analysis is as follows:

$$u_i = \sum_j x_j[r_j W_{ij} + c(r_j - 1)S_{ij}] + b_i \tag{5.6}$$

$$x' = f(u_i), \tag{5.7}$$

Where i is the index of the output unit of layer $l+1$, and j represents the index of the output unit of layer l. The output unit of a layer is denoted by x, and the weight of the connection, also known as edge between x'_i and x_j is denoted by W_{ij}. While b_i is the bias of the ith unit, r is the regularization factor, S_{ij} represents the sign of the corresponding weight. After shakeout, the linear combination of u_i is sent to the activation function $f(.)$ to compute the corresponding output x'_i. However, during back-propagation, the gradient is computed with respect to each unit so as to propagate the error.
Mathematical analysis:

$$\frac{\partial u_i}{\partial x_j} = r_j(W_{ij} + cS_{ij}) - cS_{ij}. \tag{5.8}$$

And the weights are updated as follows:

$$\frac{\partial u_i}{\partial W_{ij}} = x_j(r_j + c(r_j - 1)\frac{dS_{ij}}{dW_{ij}}), \tag{5.9}$$

Where $\frac{dS_ij}{dW_ij}$ represents the derivative of a sgn function, which is not continuous at zero and thus the derivative is not defined, it can be approximated as $\frac{dtanh(W_ij)}{dW_ij}$. However, perturbing the weights, a random mask is produced with a probability of p, and the weights can be expressed as follows:

$$\tilde{W}_{i,j} = \begin{cases} -csgn(W_{ij}) & if m_j = 0, \\ \frac{i}{p} W_{:,j} & if m_j = 1 \end{cases} \tag{5.10}$$

Where c is the strength of L_1 regularization. Hence, instead of zeroing out weight, shake out sets it to a constant c with an opposite sign of the weight if the mask is zero, and adds the constant value c to the weights if the mask is one.

5.3.3 Dropout Variants

In dropout variants, neurons that are retrained in the current iteration are made more likely to be dropped in the subsequent iteration. Standout trains a different network, currently with the main neural network that predicts an adaptive dropout rate p; Monte-Carlo Dropout [35], where instead averaging by scaling the weights, multiple stochastic process are used to estimate the average, which gives a degree of the uncertainty in the prediction of the network; swap-out samples network models from a much bigger set of architectures, where neurons in each layer can be dropped, entire layers can be omitted or two layers can be combined.

Variation learning can also be used to improve network performance, instead of learning a value for each connection in the network, a probability distribution over each connection in the network is learned. If normal distribution is used to model it, the parameters in the network are doubled while performance is approaching that of dropout. However, most of the above variants of dropout are empirically insinuated and do not have special theoretical equivalence to deterministic regularization and model selection.

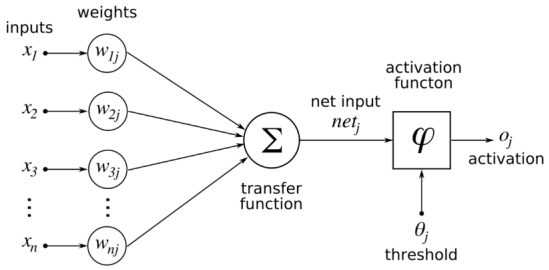


Figure 5.1: Activation Function

5.4 Batch Normalization

Batch normalization is used to accelerate Deep Neural Networks' training by making data standardization an integral part of the network architecture. Batch normalization can be categorized as another layer that can be added into the model architecture, just like the convolutional or fully connected layers. It details how to feed-forward the input and compute the gradients with respect to the parameters and its own input through a backward pass.

It is the general believe that the internal covariate shift is the main reason why deep neural models have been substantially slow during training. This emanate from the perspective that deep networks do not only have to learn a new representation at each layer, but account for changes that occur in their distributions. In general, coveriate shift is a popular problem in the community of machine learning, which occur often in real world scenario. This is usually handled with standardization or whitening processing step, but it is computational expensive, especially if the covariate shift occur throughout the network layers.

To curb this problem, Ioffe & Szegedy (2015) [36] defined the idea of batch normalization, which integrates normalization as a part of the model itself. It normalizes the output from a previous layer by using an estimated value of the mean and variance on a batch-level. Batch normalization also introduces two trainable parameters; gamma and beta, which scales and shift the normalized values, restoring the network's representation power. With batch normalization, higher learning rates can be used, as the normalization addresses the problems of vanishing and exploding gradients, and thereby stabilizing the training process.

5.5 Activation functions

The relationship between the input of a neural network and its weight is linear, which is a polynomial of one degree, it can only work for linear regression analysis, it can not handle a complicated and non-linear complex functions. Hence, activation functions are used to map input and response variable in this sense as shown in Figure 3.8. They introduce nonlinear properties to our neural network necessary to handle complicated and complex functions. It simply gets the sum from the product of the weights and inputs, as its own input and output for the next layer. However, with activation functions, neural networks are capable to learn and model high dimensional, non-linear big datasets, with complicated architecture and a lots of hidden layers, with which, the model use to extract knowledge from the complicated types of data, like videos, images, audios etc.

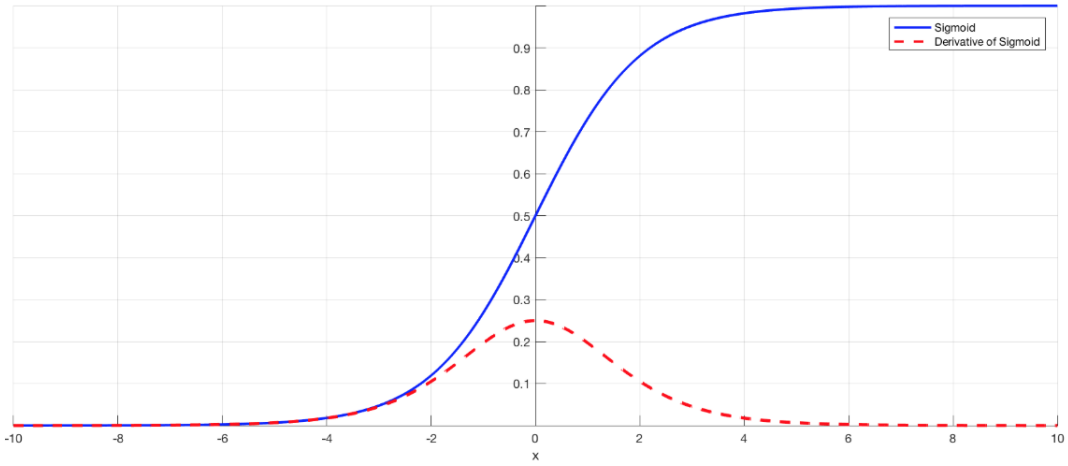


Figure 5.2: Sigmoid Graph

An important feature of activation functions used for neural network is the fact that they must be differentiable, which is essential for backpropagation optimization, while moving backwards to the network to compute the error gradients with respect to the network weights, which in turn optimize the weights using an optimizer in order to reduce losses. Frequently used activation functions in neural network are discussed below.

5.5.1 Sigmoid or Logistic Activation function

A logistic activation function is of the form:

$$f(x) = \frac{L}{1 + e^{-k(x-x_0)}} \tag{5.11}$$

where:
$L-$ The curve maximum value
$k-$ Steepness of the curve
x_0- x values of logistic's midpoint.

However, a standard logistic function is called sigmoid function with $k = 1, \quad x_0 = 0, \quad and \quad L = 1$. An S-shaped curve that ranges between 0 and 1. It takes a real-valued number and squashes it into range of number between 0 and 1. In particular, large negative numbers are mapped to 0 and large positive numbers are mapped to 1. The neuron does not fire if its value is 0, but fully saturated if its value is 1 and fires at the highest frequency.

However, its setbacks include:

- Vanishing gradient problem (a problem resulting from small gradient which making learning slow during backpropagation).
- Sigmoid saturate and kill gradients: The red dotted lines indicate the derivative, the gradient is almost zero at this region. During backpropagation, the small gradient will be multiplied with the gradient of this gate's output for the whole objective, the resulting value will be small and mapped to zero, this will prevent the neuron from firing, and no signal will flow through it. And

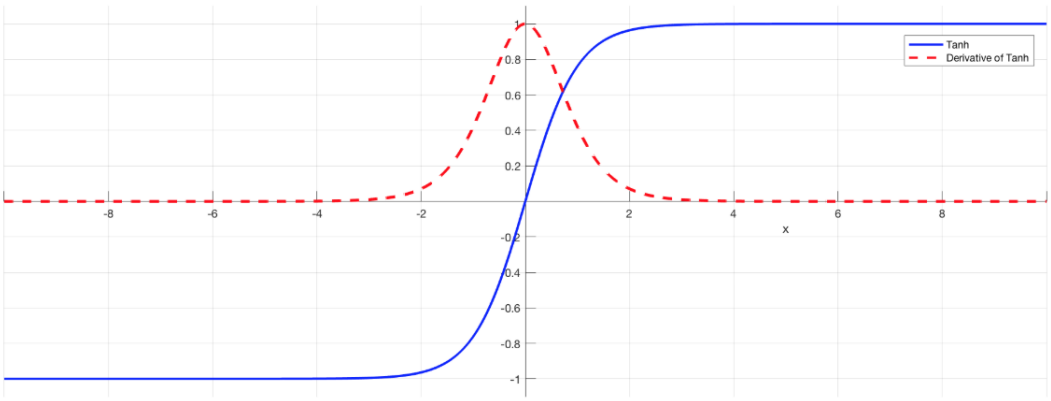


Figure 5.3: Hyperbolic Tangent function (Tanh)

if the weights initialization is too high, the neurons will get saturated, and network will fail to learn.

- It has a very slow convergence.
- Its output is not zero centered, which implies $0 < output < 1$, this makes optimization looks tedious. This is a bad property because neurons in later layers of processing would be receiving data that is not zero-centered. If the incoming data to the neuron is always positive, then during backpropagation, the gradient on the weights will be all positive or all negative, and this will lead to undesirable zig-zagging dynamics in the gradient updates for the weights.

$$S(x) = \frac{1}{1 + e^{-x}} \tag{5.12}$$

5.5.2 Hyperbolic Tangent Activation Function-Tanh

Tahn is of the form:

$$\mathbf{tanh(x)} = \frac{1 - e^{-2x}}{1 + e^{-2x}} \tag{5.13}$$

Here, its output is zero centered as it ranges from -1 to $+1$. Hence, it makes optimization easier, and it's preferred in practice over sigmoid, but still suffers from vanishing problem. The derivative as shown with the red dotted lines is given as

$$\mathbf{tahn'}(x) = 1 - tanh^2(x) \tag{5.14}$$

Since the derivative shape is similar to that of sigmoid, it is also interpreted that tanh also saturates and kill gradient.

5.5.3 Rectified Linear Units Activation Function

ReLU has become very popular and versatile in researches in the past couple of years. It has been shown that it had 6 times improvement in convergence over Tanh function[37]. It is argued that this is due to its linear, non-saturating form. The ReLU function is more effective than logistic and hyperbolic functions, in that it reduces computation cost. It's of the form:

$$\mathbf{f(x)} = max(0, x)$$

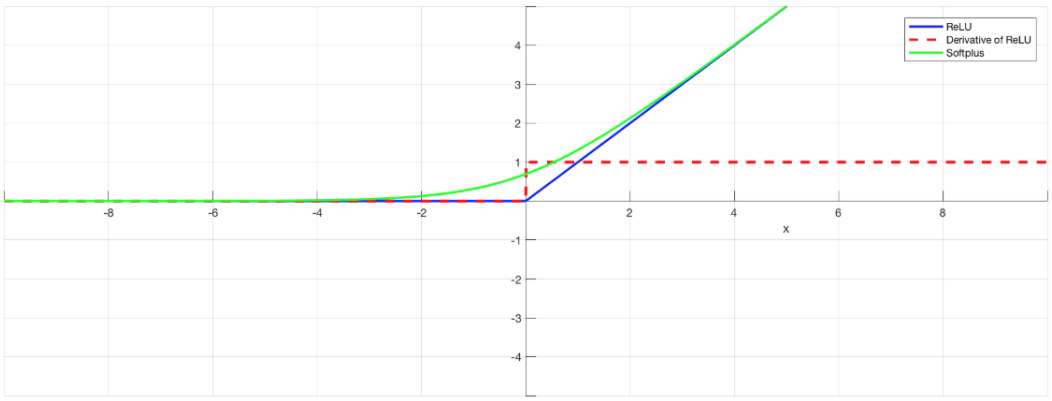


Figure 5.4: Rectified Linear Units (ReLU)

where x is the input to a neuron, it ranges from 0 to ∞. Its derivative is shown in the red plot in Figure 5.4.

Unfortunately, it can be fragile during training, which can make it die. If a large gradient flows through a neuron, it can update the weights in such a way that the neuron will never activate on any data point again and the gradient flowing through it will remain zero at that point. However, there is a smooth approximation to the rectifier, known as softplus function, which is indicated as green line in Figure 5.4, and it is defined in Equation (5.15) and its derivative is defined in Equation (5.16).

$$\mathbf{f(x)} = In(1 + e^{x}) \tag{5.15}$$

$$\mathbf{tahn'}(x) = \frac{1}{1 + e^{-x}} \tag{5.16}$$

From Equation (5.16), we can observe that the derivative of Softplus is the same as the logistic function. Therefore, both the ReLU and Softplus are largely similar, except near 0 where the softplus is very smooth and differentiable. However, it is pretty easier and economical to compute ReLU and its derivative than for the softplus function.

5.5.4 Leaky and Parametric ReLU

This activation function attempts to fix the dying problem of ReLU, instead of being zero when input is less than zero, leaky ReLU will have a small negative gradient.

$$\begin{cases} \mathbf{f(x)} & = 0.01\text{x}, \quad (\mathbf{x} < 0) \\ \mathbf{f(x)} & = \text{x}, \quad (\mathbf{x} \geq 0) \end{cases} \tag{5.17}$$

The activation function performs good, but its results are not always consistent. The slope in the negative region can also be made into a parameter of each neuron, which implies Parametric ReLU.

$$\begin{cases} \mathbf{f(x)} & = \alpha\text{x}, \quad (\mathbf{x} < 0) \\ \mathbf{f(x)} & = \text{x}, \quad (\mathbf{x} \geq 0) \end{cases} \tag{5.18}$$

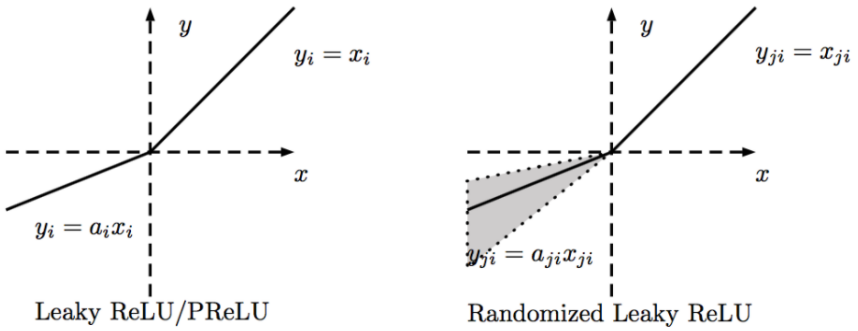


Figure 5.5: Leaky ReLU and Randomized Leaky ReLU

Where α is a constant less that 1. However, the parametric ReLU can be approximated as a maxout function as shown in equation.

$$\mathbf{f(x) = max(x, \alpha x)} \tag{5.19}$$

5.5.5 Randomized ReLU

Randomized ReLU is a randomized form of Leaky ReLU, where α is a random number. Slopes of its negative parts are randomized in a given range in the training, and then fixed during testing. It is reported that RReLU is capable of reducing overfitting due to its randomized nature in the Kaggle National Data Science Bowl (NDSB) competition. It's defined as follow:

$$\begin{cases} f(\mathbf{x}_{ji}) & = \mathbf{x}_{ji}, \quad if \ \mathbf{x}_{ji} \geq 0 \\ f(\mathbf{x}_{ji}) & = \alpha_{ji}\mathbf{x}_{ji}, \quad if \ \mathbf{x}_{ji} < 0 \end{cases} \tag{5.20}$$

$\mathbf{x}_{ji}$: the input of ith channel in jth sample.

$f(\mathbf{x}_{ji})$: the corresponding output after passing through the activation function.

However, note that:

- α_i is learned in parametric ReLU.
- α_i is fixed in Leaky ReLU.
- For Randomized ReLU, α_{ji} is takes random number during training, but fixed during testing.

Advantages of ReLU

- It is one sided compare to anti-symmetry of tanh.
- Efficient computation and gradient propagation: No vanishing or exploding problems.
- Sparse activation, in a randomly initialized network, only about 50 percent of hidden units are activated.
- Scale-Invariant: $\mathbf{max(0,}\alpha_i x\mathbf{)} = \alpha_i \ \mathbf{.} \ \mathbf{max(0,}x\mathbf{)}$.

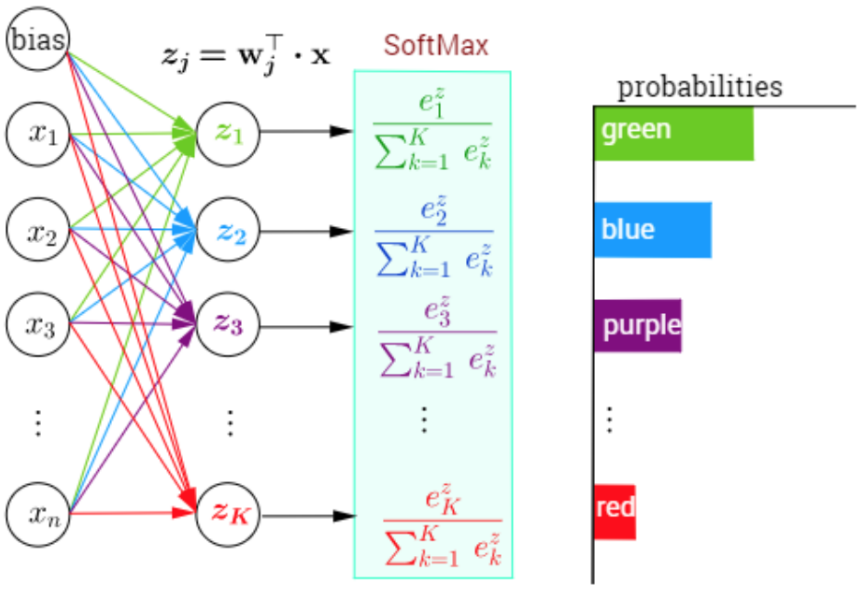


Figure 5.6: Illustration of Softmax

Disadvantages of ReLU

- It is not zero centered.
- It is not differentiable at zero.
- It not bounded, subjective to be blown off.
- Dying ReLU problem.

5.5.6 Softmax

It is of the form:

$$P(\mathbf{y}=\mathbf{j}|\ \mathbf{x}) = \frac{e^{x^\intercal w_j}}{\sum_{k=1}^{\mathbf{K}} e^{x^\intercal w_k}} \tag{5.21}$$

An activation used for classification in multi-class problems in neural network. It is a general form of logistic function that squashes a K-dimensional vector from its real values to a K-dimensional vector of real values in the range of [0 and 1], that sum to 1. The class with highest probability is decided. As in Figure 5.6, the green class is decided, because it has the highest probability.
Other activation functions include: Step, ArcTan, SoftSign, Sinc, Gaussioan, Maxout, etc.

5.6 Cost Functions

Loss function is another crucial part of artificial neural networks, which is used to determine the mismatch between estimated or predicted value $\hat{\mathbf{y}}$ and the original value. It is always a positive value, where the accuracy of the model increases with decrease in the value of the loss function. Loss function is the hard core of empirical risk function as well as a significant component of structural risk function

[38, 39], which can be represented as:

$$\theta^* = \underset{\theta}{\operatorname{argmin}}\, \boldsymbol{L}(\theta) + \lambda.\Phi(\theta) \tag{5.22}$$

$$= \underset{\theta}{\operatorname{argmin}}\, \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{L}(\mathbf{y}^{(i)}, \hat{y}^{(i)}) + \lambda.\Phi(\theta) \tag{5.23}$$

$$= \underset{\theta}{\operatorname{argmin}}\, \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{L}(\mathbf{y}^{(i)}, f(x^{(i)}, \theta)) + \lambda.\Phi(\theta) \tag{5.24}$$

Where $\Phi(\theta)$ is the regularization factor, θ is the model's parameters to be learned, $F(.)$ is the activation function, and $X^{(i)} \in \mathbb{R}^m$ represents the training samples.

5.6.1 Mean Square Error (MSE)

A popular quadratic function used in evaluating the performances of a linear regression model, whose principle is based on optimizing the fitting that minimizes the sum of distance of all points to the regression line [39]. It can be defined as:

$$\boldsymbol{L} = \frac{1}{n}\sum_{i=1}^{n}(y^{(i)} - \hat{y}^{(i)})^2 \tag{5.25}$$

Where $(y^{(i)} - \hat{y}^{(i)})$ is the residual or error. MSE tries to minimize the sum of the error square.

5.6.2 Mean Square Logarithmic Error (MSLE)

This is a variant form of MSE, which is defined as:

$$\boldsymbol{L} = \frac{1}{n}\sum_{i=1}^{n}(log(y^{(i)} + 1) - log(\hat{y}^{(i)} + 1))^2 \tag{5.26}$$

It measures the differences between the estimated and the real values. MSLE is usually used when you are trying to reduce the amount of penalty on the huge difference between the predicted and the actual value, it improves learning process tremendously.

Following points should be noted:

- MSE and MSLE is same if both predicted and actual values are small.
- MSE > MSLE if either predicted or the actual value is big.
- MSE > MSLE if both predicted and actual values are big.

5.6.3 Mean Absolute Error (MAE)

MAE is used to measure the difference between statistical forecasts and the eventual outcomes, it is defined as:

$$\boldsymbol{L} = \frac{1}{n}\sum_{i=1}^{n} \mid (y^{(i)} - \hat{y}^{(i)}) \mid \tag{5.27}$$

Where $\mid . \mid$ means absolute value. As much as MAE and MSE are both used in predictive modeling, a lot of differences exist between them. While mathematical computations in MSE look friendly, that of MAE is complicated, it makes use of linear programming to compute its gradient. MAE performs more robust to outliers than MSE, because of the square in MSE, any error has a great influence in MSE than MAE, which use absolute value. However, when dealing with processes, where consequences of large errors are huge, compared to small error, MSE plays a vtal role than MAE. MSE corresponds to maximizing the likelihood of normal random variables.

5.6.4 Poisson

This is another loss function whose performance measurement is based on the degree of divergence of the predicted distribution to the actual distribution. It's usually employed to analyze bid data."It can be shown to be the limiting distribution for a normal approximation to a binomial where the number of trials goes to infinity and the probability goes to zero and both happen at such a rate that np is equal to some mean frequency for the process [39]." It is defined as:

$$\boldsymbol{L} = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}^{(i)} - y^{(i)}.log(\hat{y}^{(i)})) \tag{5.28}$$

where the '.' indicates the dot product.

5.6.5 Cross Entropy

Cross-entropy loss, or log loss, is used to measure the performance of a classification model, whose output is a probability value, that ranges between 0 and 1. Its loss increases as the probability of the prediction diverges from the actual label. It takes two probability distributions, $p(x)$: the true probability distribution, and the predicted probability distribution $q(x)$ as in Equation (5.29). Considering neural network, its calculation does not depend on the kind of layer that was used nor the activation function used, except for some activation functions that are not compatible, whose outputs cannot be interpreted as probability, say for instance, their outputs are negative values or the output does not sum up to 1.

$$H(p,q) = -\sum_{\forall x} p(x)log(q(x)) \tag{5.29}$$

It is usually used in binary classification. If the class is more than 2 (multiclass), as this case of this research work, it is called categorical cross-entropy, and it can be computed as:

$$\boldsymbol{L} = \frac{1}{n}\sum_{i=1}^{n}[y^{(i)}log(\hat{y}^{(i)}) + (1-y^{(i)})log(1-\hat{y}^{(i)})] \tag{5.30}$$

If the cross between the distributions is small, it means the two distribution are similar, the dissimilar increases as the cross between them increases. For more details and the mathematics behind it, see [40]

5.6.6 Negative Logarithmic Likelihood

Negative Log Likelihood loss function is another cost function that is popularly used in neural networks, it measures the accuracy of a classifier. It is suitable when the model outputs a probability for each class, instead of just the most likely class. It uses the idea of probabilistic confidence to measure accuracy softly, and has a tie to information theory. It is very similar to cross entropy (in binary classification) or multi-class cross entropy (in multi-classification) in its mathematical analysis. Negative log likelihood is defines as:

$$\boldsymbol{L} = -\frac{1}{n}\sum_{i=1}^{n} log(\hat{y}^{(i)}) \tag{5.31}$$

5.7 Optimizers

The major role of an optimizer is to update the weight parameters to minimize the loss function $\mathbf{J}(\theta)$ parameterized by the model's parameters $\theta \in \mathbb{R}^d$in opposite direction to gradient of the loss function with respect to the weight parameters. Loss function is a guide to the optimizer if it's moving in

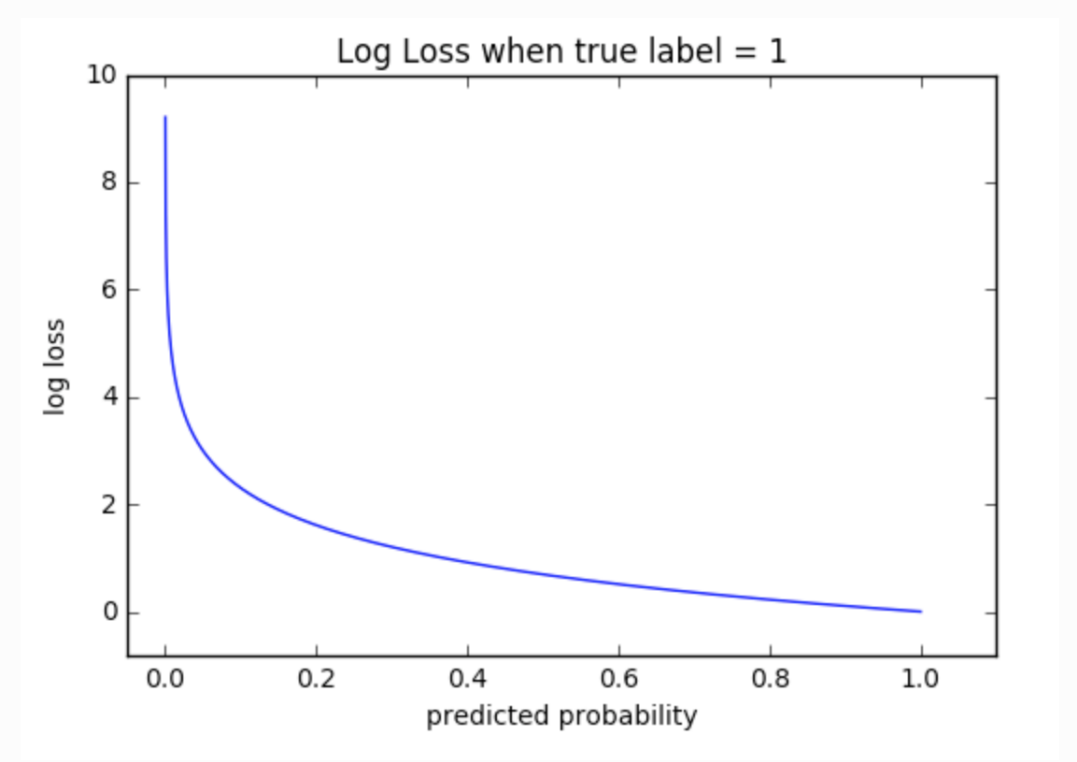


Figure 5.7: Cross Entropy; source: Machine Learning Cheatsheet

the right direction to reach the global minimum. Gradient descent is an iterative machine learning optimization algorithm to reduce the cost function and will help models to make accurate predictions [41]. Gradient descent is one of the most common algorithms used for optimization problems, and much more popular to optimize neural networks. To reach the global minimum, we need to determine how much step(the learning rate) we will take at a time.

Gradient descent are of three types, depending on how proportion of the training set with which we compute the gradient. And choosing the learning rate, we have to balance the trade-off between the speed of reaching the global optimum and the accuracy.

5.7.1 Batch Gradient Descent (BGD)

BGD computes the gradient of the loss function with respect to the parameter θ for the entire training dataset. It can be defined as:

$$\theta = \theta - \eta . \bigtriangledown_\theta J(\theta) \tag{5.32}$$

Computing the gradients for the whole data in only one update would be very slow. It is guarantied to converge at global minimum for quadratic cost function, which is convex in shape with only one minimum, but could converge at local minimum for non-convex cost functions.

5.7.2 Stochastic Gradient Descent (SGD)

Instead of computing gradient for the whole dataset in BGD, which is very inefficient, SGD approach for solving this problem is random choice of the next samples from the training set, that will be used to update the trainable parameters. That is, it uses random samples from the training dataset for each iteration and update the trainable parameters in the network. However, SGD performs frequent

updates with a high variance that makes the cost function to fluctuate heavily, and always overshoot. But with a very low learning rate, SGD shows almost the same properties as BGD.

It can be defined as:

$$\theta = \theta - \eta . \bigtriangledown_\theta J(\theta; \mathbf{x}^{(i)}; \mathbf{y}^{((i))}) \tag{5.33}$$

Where $(i + n)$ is the number of samples.

5.7.3 Mini-Batch Gradient Descent

A key optimizer that solves the challenges of BGD and SGD is mini-batch gradient descent. It computes the gradient and update the training parameters for every mini-batch.

$$\theta = \theta - \eta . \bigtriangledown_\theta J(\theta, \mathbf{x}^{(i:i+n)}, \mathbf{y}^{((i:i+n))}) \tag{5.34}$$

Hence, it reduces the variance of the parameter updates, which unarguably leads to more stable convergence. Mini-batch gradient descent is a good choice when training neural network, and sometimes referred to as SGD as acronyms.

5.7.4 Challenges of Mini-Batch Gradient Descent

- Very big challenge in choosing appropriate learning rate, a learning rate that is too small will lead to low convergence, while too big learning rate will make the cost function to fluctuate around the minimum or could lead to divergence.
- It applies the same learning rate to all the dataset. However, if the dataset is sparse with varying frequencies, it's good practice not to update all the dataset to the same extent, but perform more update for barely occurring samples.
- Another big problem encountered while minimizing a highly non-convex loss functions, common for neural networks is avoiding being trapped in their many suboptimal local minima. Sebastine Ruder [41] detailed how difficulties arise, not from local minima but from saddle points, that is, points where one dimension shoots up and another shoots down. These saddle points are usually surrounded by a plateau of the same error, that makes it hard for SGD to escape, as the gradient is close to zero in all dimensions.

However, numerous optimizers are used in Deep Neural Networks to tackle the challenges faced by the three variations of SGD. The popular ones are discussed below.

5.7.5 Momentum

Momentum is capable of enhancing solution to the navigation problem encountered by SGD, around the local optimum, rather than vibrating, it will accelerate the SGD in the relevant direction by adding a fraction γ (usually set to value between 0.7 to 09), of the previous update to the current one.

$$v_t = \gamma v_{t-1} + \eta \bigtriangledown_\theta \mathbf{J}(\theta) \tag{5.35}$$

$$\text{with} \quad \theta = \theta - v_t$$

It enhances faster convergence and reduced vibration.

5.7.6 Nesterov Accelerated Gradient (NAG)

However, momentum take a faster jump moving towards the right direction, without taking account for its current position and the gradient. NAG first make a big jump in the direction of recent accumulated gradient, measures the gradient and do some necessary correction, which in turn results to NAG updates.

$$v_t = \gamma v_{t-1} + \eta \bigtriangledown_\theta \mathbf{J}(\theta - \gamma v_{t-1}) \tag{5.36}$$
$$\text{with} \quad \theta = \theta - v_t$$

5.7.7 Adagrad

Now, that momentum can adapt the slope of the cost function and speed up SGD in turn, Adagrad adapt the updates to each individual parameter to perform larger or smaller updates depending on their respective relevance. More precisely, "Adagrad adapts the learning rate to the parameters with smaller updates for parameters relating to features that occur often, and larger update for barely occurring future's parameters [42]".
Adagrad makes use of different learning rate for every parameter θ_i at every time step t. Its major advantage is its ability to tune the learning rate automatically. However, it accumulates squared error in the denominator, which causes the learning rate to become infinitesimally small, and the algorithm cease to learn further. It can be defined as:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \in}} \odot g_t \tag{5.37}$$

Where $G_t \in \mathbb{R}^{dXd}$, and g_t is the gradient at time t.

5.7.8 Adadelta

Adadelta is an extension of Adagrad, which addresses the significant reduction in the learning rate, by restricting the value of the accumulated past gradient to a fixed value.

5.7.9 RMSprop

PMSprop is an adaptive learning rate method proposed by Geoff Hinton, but still unpublished. It divides the learning rate by an exponentially decaying average of squared gradients. Defined as:

$$E[g^2]_t = 0.9E[g^2]_{t-1} + 0.1g_t^2 \tag{5.38}$$

$$\theta_{t+1} = \theta_t \frac{\eta}{\sqrt{E[g^2]_t + \in}} g_t, \quad \eta = 0.001 \tag{5.39}$$

5.7.10 Adam

Apart from storing the exponential decaying average of previous squared gradients v_t, as in Adadelta and RMSprop, Adam stores an exponentially decaying average of past gradients. It has been shown that Adam works well in practice and performs significantly if compares to other adaptive learning-method algorithms.

5.7.11 AdaMax

Adam combines the relevant properties of the AdaGrad and RMSProp algorithms to develop an optimization algorithm that can handle sparse gradients on noisy problems.

5.7.12 Nadam

Nadam (Nesterov-accelerated Adaptive Moment Estimation) thus combines the best attributes of Adam and NAG [43]. Nadam applies acceleration to the parameters before calculating the gradients, then update it with the gradients calculated with the temporal parameters. This idea helps to avoid first-order optimization problem, which is gradients' explosion, in Recurrent Neural Networks.

For more details mathematics behind the optimizers discussed herein, and other optimizers used in neural network, please see: "overview of gradient descent optimization algorithms [41]". Thus; the practical application to show the performances of these optimizers are shown in Chapter 6 of this thesis.

Chapter 6

Implementation and Results

The key libraries used in the implementation include tensorflow(a deep learning framework that enable the user to build and train deep learning models), keras, Numpy [44], Scikit-learn [45], Pandas[46] and Librosa [47].

6.1 Pre-Experiment

Before the final implementation, a series of experiments are covered, this was to tune and select optimal values for our hyper parameters, select the best choice of optimizer, and activation function. These experiments include building a speaker recognition with keras, where the hyper parameters are tuned, and data augmentation is performed. The final implementation is summarized in the subsequent part of this chapter.

6.1.1 Speaker Recognition

In one of our pre-experiments, a speaker recognition model in keras is designed. This experiment is evaluated using the dataset from VoxCeleb; an audio dataset consisting of short clips of over seven thousand speakers spanning a wide range of different ethnicities, accents, professions and ages. The architecture used is adopted from "An analysis of Convolutional Neural Networks for Speech recognition [48]". The input feature to the model is 40-dimension log-filter-bank feature with up to second derivatives, which is used to augment the original input feature. However, Adam optimizer and ReLU activation function are used for the final experiment, with a learning rate of 0.0001.

A schematic diagram showing the flow of the experiment is shown in the Figure 6.1. The basic feature extraction and pre-processing as discussed in chapter three are adopted to generate the input features. This is followed by feedforward processing, which yields the predicted output,from which the error between the actual targets and the predicted speakers are estimated. This is used to update the network weights in the negative direction of the error's slope. This process is repeated until the error converges. Series of experiments are carried out to analyze the performances of the different optimizers and tuned hyper-parameters. The result of the experiment corresponding to the performances of the optimizers and the hyper-parameters are presented in Figure 6.2 to 6.5.

6.1.2 Results

In Figure 6.2, the performance accuracy on training and testing sets, using seven different optimizers are presented. While Adam optimizer showed the best performance, Adamax and Adadelta also showed a good performance, and Adagrad had the worst performance. This evaluation was necessary to verify our optimizer details as discussed in chapter three. In addition, the loss from the optimizers are presented in the Figure 6.3. It showed a reflection of its accuracy as expected.
Figure 6.4 shows the evaluation of the accuracy with respect to different epochs. It showed that the accuracy has reached an optimal value, a further increase in the training does not show a visible improvement.

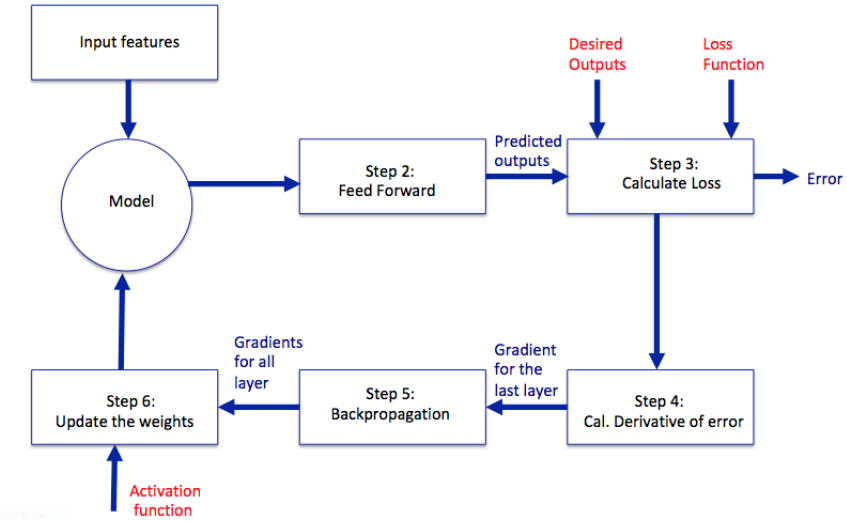


Figure 6.1: Flow of the experiment

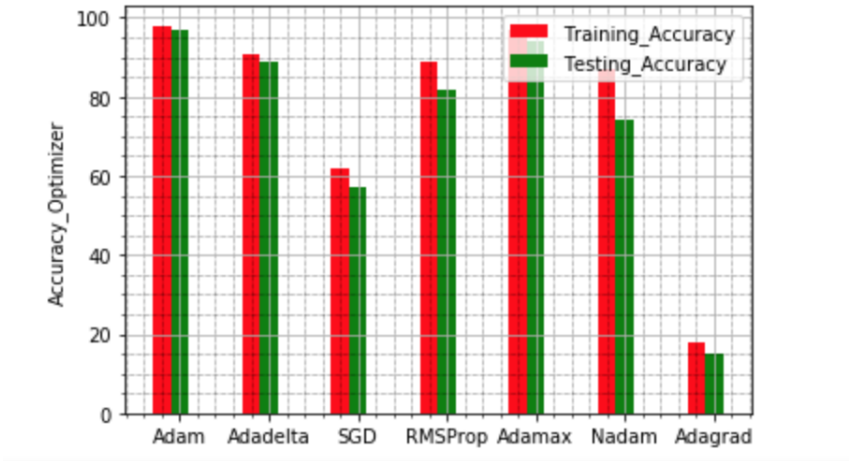


Figure 6.2: Accuracy with different optimizers

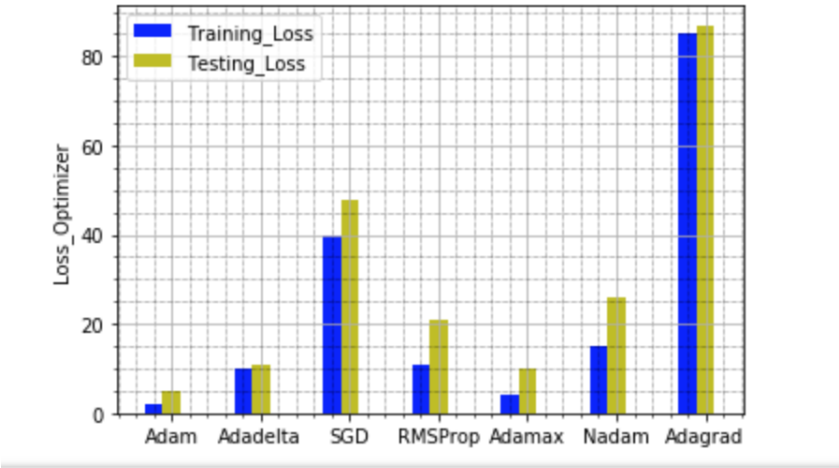


Figure 6.3: Loss with different optimizers

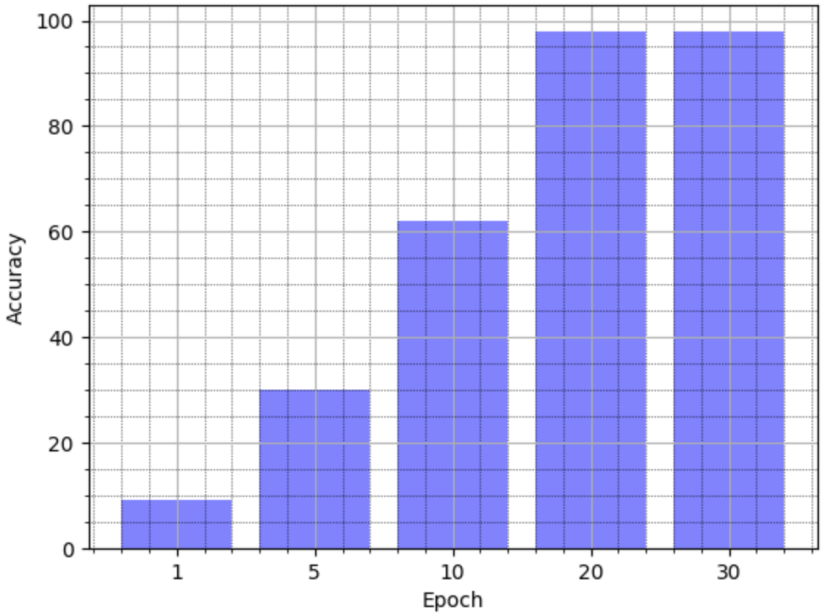


Figure 6.4: Performance analysis of epoch

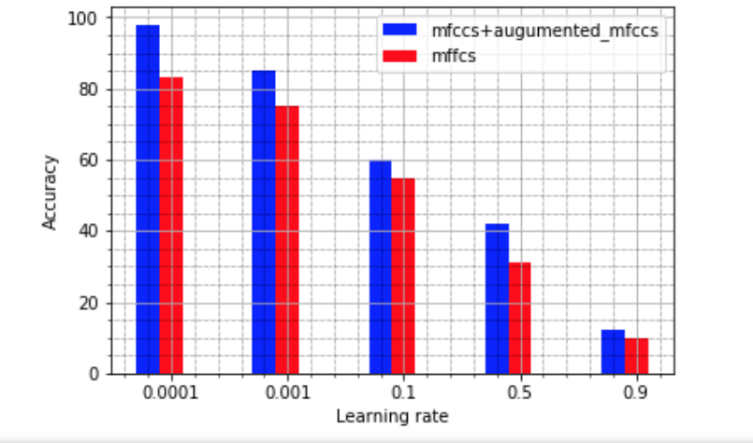


Figure 6.5: Performance analysis of learning rate

One of the most important hyper-parameters, which has significant influence on the latency and accuracy is the learning rate. It is the step size which determines to what extent newly acquired information overrides old information. In other words, it determines the extent at which the network weights are updated. From Figure 6.5, it can be observed that the performance of the model depreciates with increase in the leaning rate. It can also be observed that the model performed better with a well processed input features (with the first and second derivative of the MFCC concatenated with the input feature), than when the input feature is mel frequency ceptral coefficient gotten from a raw audio speech, without much of the pre-processings, this clarifies the improvement that can be achieved with further pre-processing of input features.

6.1.3 Words Recognition

As a further affirmation of our conclusion from previous subsection before proceeding to our final experiment, I evaluated the parameters that gave the optimal values during the parameters' tuning in the previous section. The experiment made use of CHiME-5 dataset. The results gotten from this experiment showed a promising and significant performance, as illustrated in the Figure 4.6, as displayed on the tensor board. The model recorded an accuracy of 98% and a loss of 0.01% However, the architecture of the experiment is arranged as:

- Convolution Layer
- Relu Activation
- Dropout
- Max Pool
- Convolutional Layer
- ReLU Activation
- Dropout
- Fully Connected Layer
- MatMul

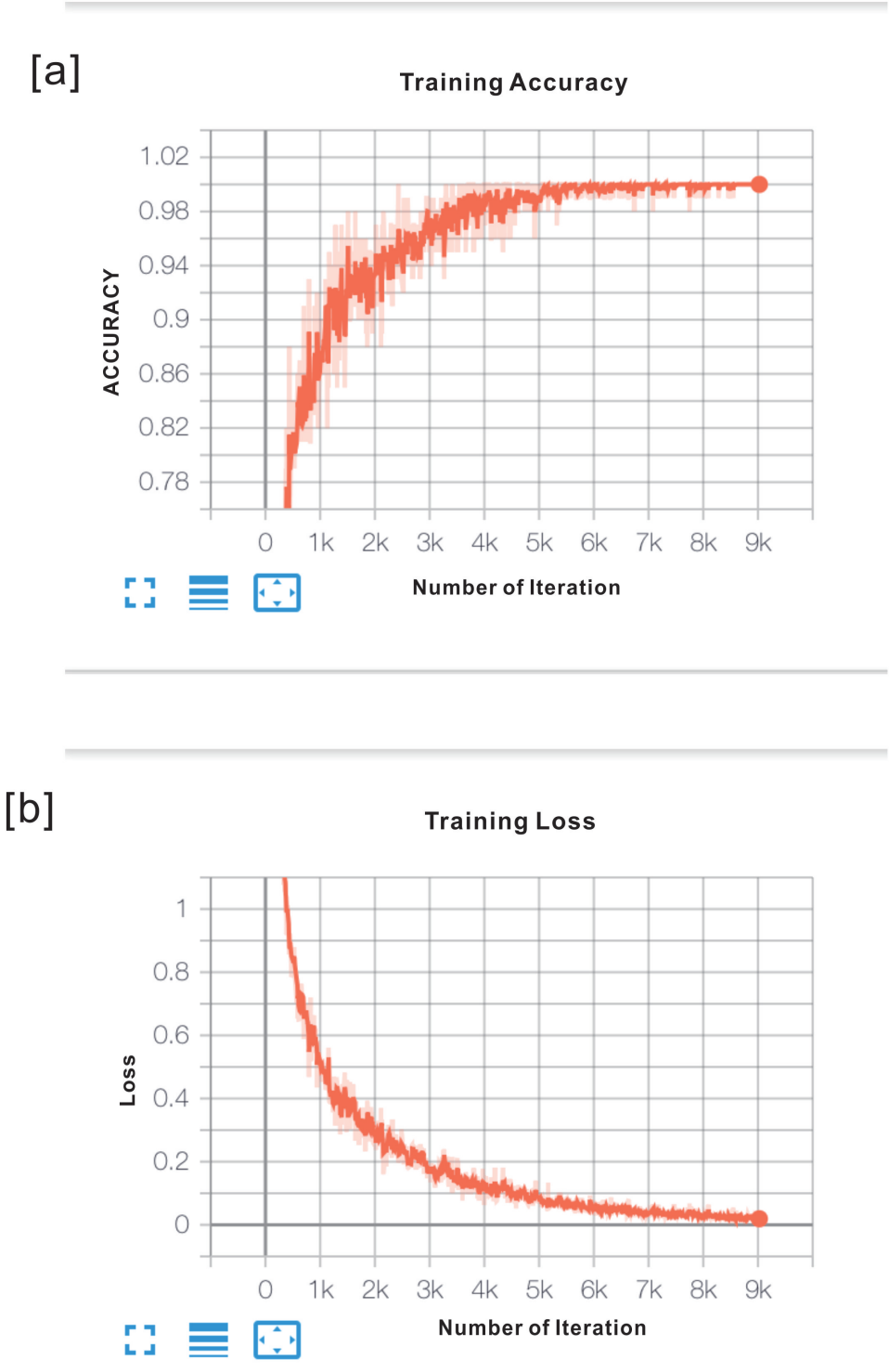


Figure 6.6: Accuracy and loss of the words recognition model

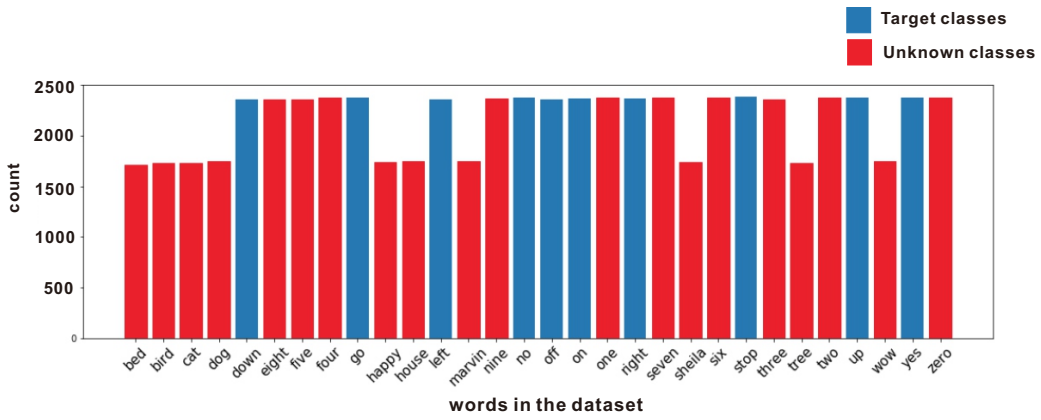


Figure 6.7: Dataset

6.2 Main Experiment

In this section, the datasets which is used to train and evaluate our method is discussed. Followed by summary of the implementation and the architectures of the models. To evaluate the performance of the model, a Low-latency singular value decomposition model (LSVD) is built , with which I compare the performance of the model. Finally, I present the evaluation metrics along with the outputs of the experiments.

6.2.1 Dataset

This research used the speech command dataset, which consists of 105,000 wave audio files of one second clips. It consists ten classes and the remaining words are classified as "unknown" and are used to improve the performance of the model to learn a representation for all words which are not in the 10 words to be classified. The last class in the set is "silence", which simply represents sample with no word. The files containing the words are arranged into folders by label, where the file names start with a hash representing the speaker, appended is a number representing the number of times that utterances by the same speaker appears in the data. The data is divided into training, validation, and testing sets with proportions shown on the result, by using the hash to ensure that same speaker does not appear in both sets. An application is considered robust if it can recognize audio, even in a noisy environment. To this end, a set of audio files which are denoted as background noise is provided. This was used to create a noisy training and validation sets, by manually adding white noise, recordings of machinery and household activity noise at signal to noise ratios (SNRs) randomly sampled between [-5dB, +10dB] to the clean data sets. Also, to cater for no matching audios, I made data for the silence class, as well as background noise. Since there is no complete silence in reality, the silence set is created from the noisy data, which was used during training. To further create distortion in the training set, time shifting was used to randomly offset the training data in time, so that a small portion at the start or at the end is cut off, and the opposite side is padded with zeros. Therefore, in total, there are 12 output labels: ten target words, one unknown class, and one silence class.

6.2.2 Model Architecture and Experimental Setup

Firstly, the convolution process remains the same as the one described in chapter three of this report. However, the architecture used in the experiment is based on the Convolutional Neural Network described in Small-footprint Keyword Spotting [49]. We choose this architecture because it's a simple structure, quick to train, and easy to understand. We grouped the audio samples into short segments with the help of our predefined window, and we find the spectrogram of each sample, which is repre-

Table 6.1: Training and validation results of the CNN and Low-LSVD models

	CNNs model	Low-LSVD model
Training accuracy	87.3%	76.4 %
Validation accuracy	89.5 %	80.2 %
Training loss	0.38	0.75
Validation loss	0.49	0.6

Table 6.2: Final testing of the CNN and Low-LSVD

	CNNs model	Low-LSVD model
Left	$(score = 0.79445)$	$(score = 0.72277)$
Right	$(score = 0.11001)$	$(score = 0.14566)$
$\_unknown\_$	$(score = 0.09554)$	$(score = 0.1319)$

sented as a vector of numbers. The human ear is more sensitive to some frequencies than others, We therefore convert the spectrogram to mel frequency cepstral qoefficients (MFCCs). The MFCCs form the input feature, which we feed into the convolutional neural network.

Furthermore, we build a low latency singular value decomposition model (low-LSVD), with the topology in the "Compressing Deep Neural Networks using a Rank-Constrained Topology [50]". Just as the CNN model, it was trained with the mel frequency cepstral qoefficients (MFCCs) of the audio waves, with a low rank based system consideration, which is based on the idea of low rank matrix factorization proposed in [51]. This reduces the number of computation done, as compared to the CNN model, though with a trade-off in the accuracy.

6.3 Results

The goal is to fine-tune the hyper parameters to obtain optimal values that will give a very good performance with our model. Also, to compensate the variation that could occur in model utilization by the end users, resulting from a noisy environment. This is achieved by missing the training set with background noise and a some silence samples, also the delta and double delta of the audio feature are concatenated with the input feature, and fed to the model. The results obtained from our CNN model are compared to the results of the low-LSVD model.
The results obtained from both models are summarized in Table 6.1 and 6.2. The left score implies percentage of correctly predicted words, while right score indicates words that are wrongly predicted, and the unknown is the percentage of false alarm.

The accuracy and loss curves of the the CNN and low-LSVD models are displayed on the tensorboard as shown in the Figures (6.8 and 6.9) respectively.
Obviously form the accuracy and loss curves, the training and testing accuracy converge and oscillate, which is a clear evidence that there is no overfitting during the training. However, the accuracy of the CNN model outperformed that of Low-LSVD model, and it loss is also minimized than the Low-LSVD loss.

It is observed that the validation accuracy is higher than the training accuracy, following are the possible reasons which can result to such scenarios:

- In neural network training, validation accuracy is computed after each epoch, which means the model must have trained better with the training set after each epoch. Whereas, training

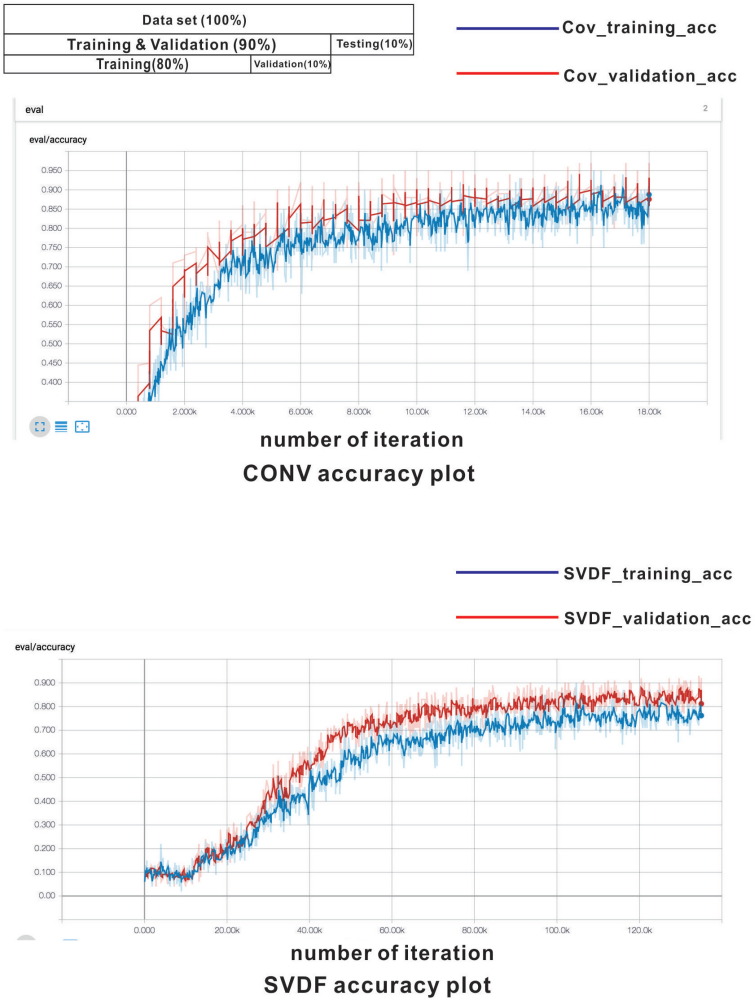


Figure 6.8: Accuracy curve

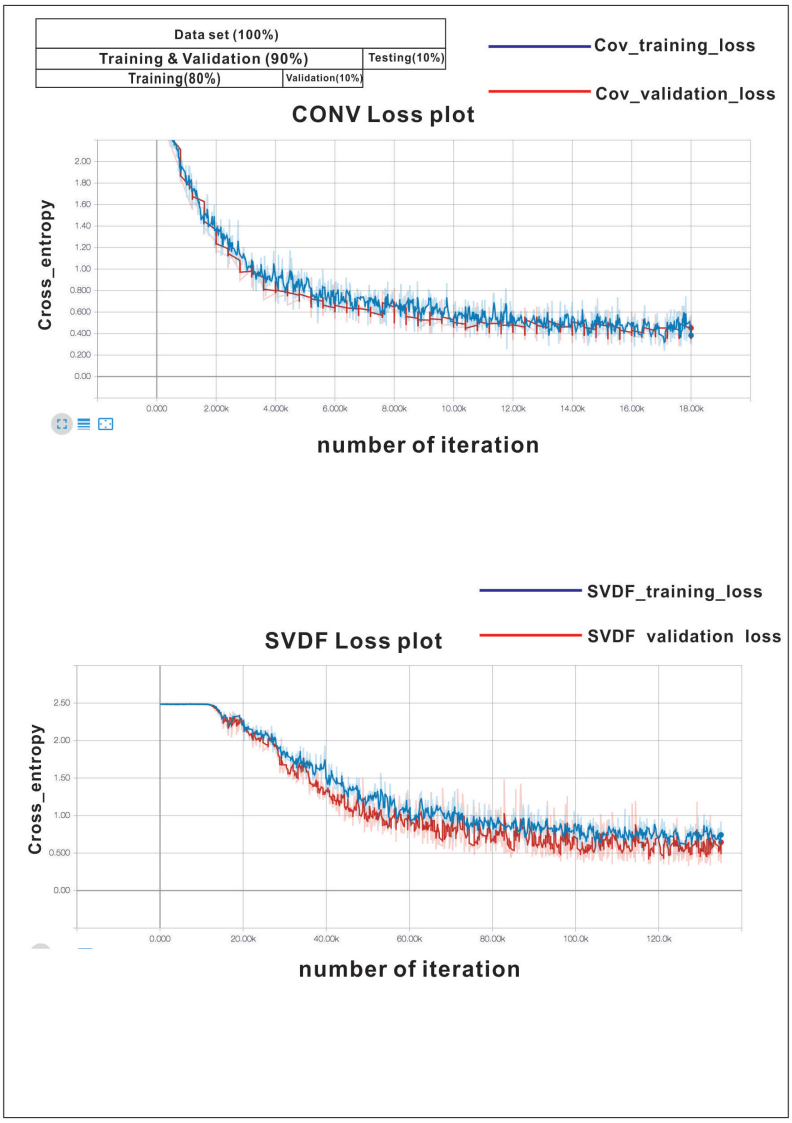


Figure 6.9: Loss curve

accuracy is computed while the training is still ongoing.

- Also, regularization such as dropout, is only applied on training dataset, which readjust its accuracy to avoid overfitting.

6.3.1 Streaming Accuracy

However, most audio applications are run on a continuous audio stream, instead of just small audio clips. A typical way to test a model in this scenario is to apply it repeatedly at varying offsets in time and average the outputs over a short window to give a good prediction. To test the effectiveness of the model on a long audio wave, our CNN model is tested against a long audio of about ten minutes, with the same sample rate of (16,000 samples per second), as the trained model. A label that indicates where each of the target words occur in time was provided. The two information are saved as a comma-separated text file, where the first column represents the label and the second column indicates the time in seconds from the start of the time that it occurs. During feature extraction, I used high sampling rate to ensure that the target words are captured, I took the mfccs of the overlapping frames and average over them, and this was fed as testing feature to my model. The result of the testing is as follow: 47.0% matched, 43.0% correctly, 9.0% wrongly, 0.0% false positive.

The matched percentage showed how many target words are in the long audio wave. While the correctly percentage showed how many of the target words are correctly detected; which implies that 43% from 47% target words of the total words in the audio stream.
However, wrongly percentage indicates how many words are identified as target words rather than background noise. The false positive percentage indicates how many words are falsely identified where no speech is actually present.

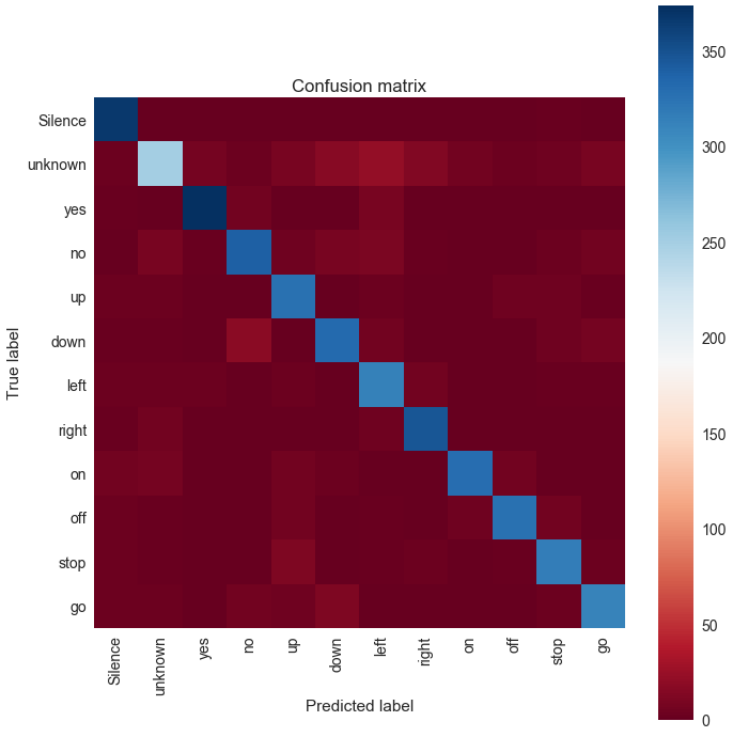
6.3.2 Confusion Matrix

This section depicts the confusion matrix from the outcome of the CNN mode. Each row is a label representing the set of samples predicted by the model, therefore, the first row represents all the audio clips that are predicted to be silence, second row “unknown”, third row “yes”, and so on. While, each column represents the ground truth in the same order as the predicted labels. This matrix is more informative than the accuracy and loss curve, in that it does not only show the accuracy and loss score, but also summarizes the mistakes made by the model. The intersection of each ground truth and predicted label indicates the correct prediction by the model, while other values are wrong predictions. A perfect confusion matrix will be a diagonal matrix, with other elements above and below it set to zero. Figure 4.10 shows the confusion matrix of our CNN model. The color bar indicates the mapping of the color to a linear scale. While Table 6.3 and 6.4 summarize the performance measures, which were discussed in chapter four of this thesis.

6.3.3 Analysis of SNR Effect

Figure 6.11 depicts the effect of the SNR on the model. The left score implies percentage of correctly predicted words, while right score indicates words that are wrongly predicted, and the unknown is the percentage of false alarm. The performance of the model depreciates with decrease in the SNR as shown in the Figure 6.11.

Figure 6.10: Confusion matrix of the CNN model



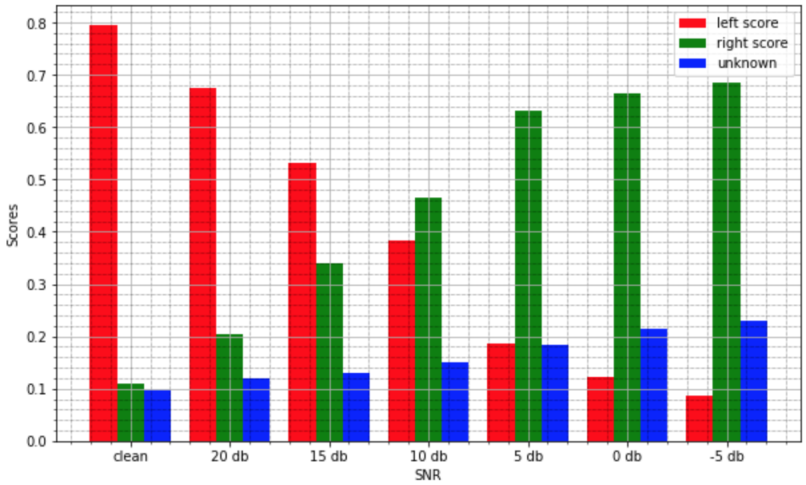
	Accuracy	TruePosiveRate	FalsePositiveRate	TrueNegativeRate	FalseNegativeRate	g_mean1	g_mean2
Silence	0.991508	0.991914	0.008530	0.991470	0.008086	18.354172	19.101335
unknown	0.967179	0.712251	0.010484	0.989516	0.287749	14.630143	15.728284
yes	0.991049	0.942065	0.004040	0.995960	0.057935	18.938226	19.299971
no	0.980032	0.869565	0.009077	0.990923	0.130435	17.534161	18.355211
up	0.982786	0.931429	0.012728	0.987272	0.068571	16.789854	17.940200
down	0.979573	0.883289	0.011307	0.988693	0.116711	17.127672	18.144832
left	0.979114	0.918129	0.015691	0.984309	0.081871	16.171823	17.580471
right	0.989672	0.955923	0.007261	0.992739	0.044077	17.895159	18.560185
on	0.988983	0.909091	0.003756	0.996244	0.090909	17.766603	18.131758
off	0.989213	0.926346	0.005245	0.994755	0.073654	17.529040	18.035658
stop	0.985081	0.905714	0.007986	0.992014	0.094286	16.968621	17.733258
go	0.983475	0.893983	0.008733	0.991267	0.106017	16.749038	17.586229

Table 6.3: Performance metrics

	Precision	Recall	F1-Score
yes	0.95	0.94	0.95
no	0.90	0.89	0.90
up	0.93	0.90	0.92
down	0.87	0.87	0.87
left	0.88	0.87	0.88
right	0.93	0.91	0.92
on	0.93	0.92	0.93
off	0.89	0.87	0.88
stop	0.90	0.89	0.90
go	0.89	0.87	0.88

Table 6.4: Precision, F1-Score and the Recall of the CNN model

Figure 6.11: Analyzing the effect of SNR on the model



Chapter 7

Conclusion and future work

Keyword detection is present in a wide range of electronic devices. The derived utility of such device for the end user is to enjoy high accuracy, compact models that run locally. In thesis, the state-of-the-art of this technology is summarized.
A deep convolutional and low-latency singular value decomposition networks were successfully trained, to add more expressive performance and better generalization for keyword detection models. Network-in-network principles were applied, which amongst others include data augmentation with background noise, human made noise and white noise, batch normalization, delta and delta-delta concatenation to build a very promising convolutional neural network model and low-latency singular value decomposition model.

In research field of automatic speech recognition, being familiar with the key ideas, pros and cons of a model is essential in order to leverage it in global research, so as to improving performance, these are presented herein. Firstly, a summary of the pitfalls of neural networks was presented. Secondly, a description of the applications of keyword detection are discussed, which had drew many researchers across the globe to provide solutions to the challenges facing the development of this system. This is followed by the mathematical analysis of the models used in this research, in such a way to reducing the complexity and training cost.

Furthermore, the proposed models uses audio files from the Google Speech Commands datasets, as feature inputs. This is used to feed both the CNN and LSVD models, the accuracy recorded from both models showed that CNN performed better than LSVD, though in a trade-off to computation complexity affliated with CNN model, which is far reduced in LSVD model. The performance analysis of different optimizer showed that Adam and Adamax have best performance in term of accuracy compared to others. Some of the hyper-parameters investigated include the learning rate, epoch, and the SNR. We observed that the accuracy decreases with increase in learning rate, and perform better at high SNR.

However, one of the open challenges for possible future work is to optimize the model structure parameters, using genetic algorithm for good quality and performance. Keyword detection has attained expectation close to human parity, but it usually requires a large amount of computation. Thus, an interesting research topic is to optimize the model structure to meet the requirements for low latency and computation, which will achieve a similar or an improved performance of these models. It is also of great interest to look into how to choose the hyper parameters with genetic algorithm rather than tuning them manually.

Bibliography

[1] Javier Tejedor, Doroteo T Toledano, Paula Lopez-Otero, Laura Docio-Fernandez, Luis Serrano, Inma Hernaez, Alejandro Coucheiro-Limeres, Javier Ferreiros, Julia Olcoz, and Jorge Llombart. Albayzin 2016 spoken term detection evaluation: an international open competitive evaluation in spanish. *EURASIP Journal on Audio, Speech, and Music Processing*, 2017(1):22, 2017. (Cited on page 1.)

[2] Lisa Amini, Pascal Frossard, Effrosyni Kokiopoulou, and Oliver Verscheure. Method and system for robust pattern matching in continuous speech for spotting a keyword of interest using orthogonal matching pursuit, March 22 2016. US Patent 9,293,130. (Cited on page 4.)

[3] Lubos Smídl and Josef V Psutka. Comparison of keyword spotting methods for searching in speech. In *Ninth International Conference on Spoken Language Processing*, 2006. (Cited on page 4.)

[4] Guy Alon. Key-word spottingthe base technology for speech analytics. *Natural Speech Communications*, 2005. (Cited on page 5.)

[5] Duarte Raposo, André Rodrigues, Soraya Sinche, Jorge Sá Silva, and Fernando Boavida. Industrial iot monitoring: Technologies and architecture proposal. *Sensors*, 18(10):3568, 2018. (Cited on page 5.)

[6] Charles Chow. System and method for keyword detection in a controlled-environment facility using a hybrid application, December 28 2010. US Patent 7,860,722. (Cited on page 5.)

[7] Anirudh Raju, Sankaran Panchapagesan, Xing Liu, Arindam Mandal, and Nikko Strom. Data augmentation for robust keyword spotting under playback interference. *arXiv preprint arXiv:1808.00563*, 2018. (Cited on page 5.)

[8] Giulia Borghini and Valerie Hazan. Listening effort during sentence processing is increased for non-native listeners: A pupillometry study. *Frontiers in neuroscience*, 12:152, 2018. (Cited on page 5.)

[9] Lin-shan Lee, James Glass, Hung-yi Lee, and Chun-an Chan. Spoken content retrievalbeyond cascading speech recognition with text retrieval. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 23(9):1389–1420, 2015. (Cited on page 6.)

[10] Hojjat Salehinejad, Joseph Barfett, Parham Aarabi, Shahrokh Valaee, Errol Colak, Bruce Gray, and Tim Dowdell. A convolutional neural network for search term detection. In *2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, pages 1–6. IEEE, 2017. (Cited on page 6.)

[11] C Myers and L Rabiner. Connected word recognition using a level building dynamic time warping algorithm. In *ICASSP'81. IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 6, pages 951–955. IEEE, 1981. (Cited on page 6.)

[12] Albert JK Thambiratnam. *Acoustic keyword spotting in speech with applications to data mining*. PhD thesis, Queensland University of Technology, 2005. (Cited on page 6.)

[13] Cory S Myers and Lawrence R Rabiner. A comparative study of several dynamic time-warping algorithms for connected-word recognition. *Bell System Technical Journal*, 60(7):1389–1409, 1981. (Cited on page 6.)

[14] Xiong Wang, Sining Sun, Changhao Shan, Jingyong Hou, Lei Xie, Shen Li, and Xin Lei. Adversarial examples for improving end-to-end attention-based small-footprint keyword spotting. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6366–6370. IEEE, 2019. (Cited on page 7.)

[15] Ronald L Breiger, Robin Wagner-Pacifici, and John W Mohr. Capturing distinctions while mining text data: Toward low-tech formalization for text analysis. *Poetics*, 68:104–119, 2018. (Cited on page 7.)

[16] Matthew Nicholas Stuttle. *A Gaussian mixture model spectral representation for speech recognition*. PhD thesis, University of Cambridge, 2003. (Cited on page 9.)

[17] Mingyang Wu and DeLiang Wang. A two-stage algorithm for one-microphone reverberant speech enhancement. *IEEE Transactions on Audio, Speech, and Language Processing*, 14(3):774–784, 2006. (Cited on page 9.)

[18] Matiur Rahman Minar and Jibon Naher. Recent advances in deep learning: An overview. *arXiv preprint arXiv:1807.08169*, 2018. (Cited on page 10.)

[19] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. (Cited on page 12.)

[20] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015. (Cited on page 27.)

[21] Fawaz S Al-Anzi and Dia AbuZeina. The capacity of mel frequency cepstral coefficients for speech recognition. *International Journal of Computer and Information Engineering*, 11(10):1162–1166, 2017. (Cited on page 29.)

[22] Wei Dai, Chia Dai, Shuhui Qu, Juncheng Li, and Samarjit Das. Very deep convolutional neural networks for raw waveforms. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 421–425. IEEE, 2017. (Cited on page 29.)

[23] Adrien Meynard and Bruno Torrésani. Spectral analysis for nonstationary audio. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 26(12):2371–2380, 2018. (Cited on page 29.)

[24] James Lyons. Mel frequency cepstral coefficient (mfcc) tutorial. *Practical Cryptography*, 2015. (Cited on page 30.)

[25] Lawrence R Rabiner, Ronald W Schafer, et al. Introduction to digital speech processing. *Foundations and Trends® in Signal Processing*, 1(1–2):1–194, 2007. (Cited on page 30.)

[26] Jason Arndt and Lynne M Reder. Word frequency and receiver operating characteristic curves in recognition memory: evidence for a dual-process interpretation. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 28(5):830, 2002. (Cited on page 32.)

[27] Kevin Markham. Simple guide to confusion matrix terminology. *data school*, 25, 2014. (Cited on page 33.)

[28] Cyril Goutte and Eric Gaussier. A probabilistic interpretation of precision, recall and f-score, with implication for evaluation. In *European Conference on Information Retrieval*, pages 345–359. Springer, 2005. (Cited on page 34.)

[29] Simon S Du, Jason D Lee, Yuandong Tian, Barnabas Poczos, and Aarti Singh. Gradient descent learns one-hidden-layer cnn: Don't be afraid of spurious local minima. *arXiv preprint arXiv:1712.00779*, 2017. (Cited on page 35.)

[30] Xin Dong, Shangyu Chen, and Sinno Pan. Learning to prune deep neural networks via layer-wise optimal brain surgeon. In *Advances in Neural Information Processing Systems*, pages 4857–4867, 2017. (Cited on page 35.)

[31] Edward D Huey, Seonjoo Lee, Gayathri Cheran, Jordan Grafman, and Davangere P Devanand. Brain regions involved in arousal and reward processing are associated with apathy in alzheimers disease and frontotemporal dementia. *Journal of Alzheimer's Disease*, 55(2):551–558, 2017. (Cited on page 36.)

[32] Hui Zou and Trevor Hastie. Regularization and variable selection via the elastic net. *Journal of the royal statistical society: series B (statistical methodology)*, 67(2):301–320, 2005. (Cited on page 36.)

[33] Gilles Gasso, Alain Rakotomamonjy, and Stéphane Canu. Recovering sparse signals with a certain family of nonconvex penalties and dc programming. *IEEE Transactions on Signal Processing*, 57(12):4686–4698, 2009. (Cited on page 36.)

[34] Shai Shalev-Shwartz and Ambuj Tewari. Stochastic methods for l1-regularized loss minimization. *Journal of Machine Learning Research*, 12(Jun):1865–1892, 2011. (Cited on page 36.)

[35] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059, 2016. (Cited on page 37.)

[36] Satoshi Iizuka, Edgar Simo-Serra, and Hiroshi Ishikawa. Globally and locally consistent image completion. *ACM Transactions on Graphics (ToG)*, 36(4):107, 2017. (Cited on page 38.)

[37] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. (Cited on page 40.)

[38] Jianqing Fan. *Local polynomial modelling and its applications: monographs on statistics and applied probability 66*. Routledge, 2018. (Cited on page 44.)

[39] Katarzyna Janocha and Wojciech Marian Czarnecki. On loss functions for deep neural networks in classification. *arXiv preprint arXiv:1702.05659*, 2017. (Cited on pages 44 and 45.)

[40] Kamil Nar, Orhan Ocal, S Shankar Sastry, and Kannan Ramchandran. Cross-entropy loss and low-rank features have responsibility for adversarial examples. *arXiv preprint arXiv:1901.08360*, 2019. (Cited on page 45.)

[41] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016. (Cited on pages 46, 47, and 49.)

[42] Maren Mahsereci and Philipp Hennig. Probabilistic line searches for stochastic optimization. In *Advances in Neural Information Processing Systems*, pages 181–189, 2015. (Cited on page 48.)

[43] Timothy Dozat. Incorporating nesterov momentum into adam. 2016. (Cited on page 49.)

[44] Travis E Oliphant. *A guide to NumPy*, volume 1. Trelgol Publishing USA, 2006. (Cited on page 50.)

[45] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011. (Cited on page 50.)

[46] Wes McKinney. Pydata development team." pandas: powerful python data analysis toolkit, release 0.18. 1" dated may 3, 2016. (Cited on page 50.)

[47] Brian McFee, Colin Raffel, Dawen Liang, Daniel PW Ellis, Matt McVicar, Eric Battenberg, and Oriol Nieto. librosa: Audio and music signal analysis in python. In *Proceedings of the 14th python in science conference*, pages 18–25, 2015. (Cited on page 50.)

[48] Jui-Ting Huang, Jinyu Li, and Yifan Gong. An analysis of convolutional neural networks for speech recognition. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4989–4993. IEEE, 2015. (Cited on page 50.)

[49] Tara Sainath and Carolina Parada. Convolutional neural networks for small-footprint keyword spotting. 2015. (Cited on page 55.)

[50] Preetum Nakkiran, Raziel Alvarez, Rohit Prabhavalkar, and Carolina Parada. Compressing deep neural networks using a rank-constrained topology. 2015. (Cited on page 56.)

[51] Tara N Sainath, Brian Kingsbury, Vikas Sindhwani, Ebru Arisoy, and Bhuvana Ramabhadran. Low-rank matrix factorization for deep neural network training with high-dimensional output targets. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6655–6659. IEEE, 2013. (Cited on page 56.)