Noise Robust Keyword Spotting Using Deep Neural Networks For Embedded Platforms

by

Ramzi Abdelmoula

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical & Computer Engineering

Waterloo, Ontario, Canada, 2016

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

The recent development of embedded platforms along with spectacular growth in communication networking technologies is driving the Internet of things to thrive. More complex tasks are now possible to operate in small devices such as speech recognition and keyword spotting which are in great demand. Traditional voice recognition approaches are already being used in several embedded applications, some are hybrid(cloud-based and embedded) while others are fully embedded. However, the environment surrounding the embedded devices is usually accompanied by noise. Conventional approaches to add noise robustness to speech recognition are effective but also costly in terms of memory consumption and hardware complexities which limit their use in embedded platforms. The purpose of this thesis is to increase the robustness of keyword spotting to more than one type of noise at once without increasing the memory footprint or the need for a denoiser while maintaining the recognition accuracy to an acceptable level. In this work, robustness in treated at the phoneme classification level as the phoneme based keyword spotting is the best technique for embedded keyword spotting. Deep Neural Networks have been successfully deployed in many applications including noise robust speech recognition. In this work, we use mutil-condition utterances training of a Deep Neural Networks model to increase the keyword spotting noise robustness. This technique is also used for a Gaussian mixture model training. The two approaches are compared and the deep learning proved to not only outperform the Gaussian approach, but has also outperformed the use of a denoiser system. This results in a smaller, more accurate and noise robust model for phoneme recognition.

Acknowledgements

I thank my supervisor, Prof. Karray, for many insightful conversations during the development of the ideas in this thesis, and for helpful comments on the text.

I thank Dr. Alaa for his inspirational help and Dr. Jiping for his help with understanding some of the tools used during this journey.

I thank my friends Ridha, Marwen, Arief, Haitham, Hayfa, Ahmed and Ibrahim for each, in his/her own way, helped the writing of this thesis. Wouldn't have been as easy without you!

My mother has been an inspiration throughout my life. She has always supported my dreams and aspirations, and if I do say so myself, I think she did a fine job raising me. I'd like to thank her for all she is, and all she has done for me.

Dedication

To my beloved parents and brother.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

DCT Discrete Cosine Transform 6

EM Expectation Maximization 7

KWS Keyword Spotting 7

LVCSR Large-Vocabulary Continuous Speech Recognition 7

DBN Deep Belief Networks 13

RBM Restricted Boltzmann Machines 14

SDA Stacked Autoencoders 14

SNR Signal to Noise Ratio 19

TIMIT Texas Instruments and Massachusetts Institute of Technology 11

WSJ World Street Journal 11

IoT Internet of Things 1

MMSE Minimum Mean-Square Error 32

HTK Hidden Markov Model Toolkit 37

Chapter 1

Introduction

1.1 Motivation

We live in the information age, where information is key to almost everything in our daily lives. From economy to military to science, the more information we have and the better our processing of it is, the more effective are our actions. Data science has been thriving in recent years. However, it has been established that most of the relevant data is not being recorded (or processed) for lack of appropriate technology and because of other challenges in storing, processing and mining huge amount of data. Thus the emergence of the Internet of Things (IoT) [25] that will help acquiring large amounts of data that is of better quality. In addition to the thriving of data science, speech recognition has been experiencing a resurrection in interest thanks to deep learning [28, 27]. An effective speech engine and natural language understanding makes a great human-machine interface which explains the great demand for such technology. Speech is one valuable and attractive data to be collected. This is due to the rich nature of speech, which is becoming among the most valuable information we can collect for further processing.
In the foreseeable futures, our homes, offices and streets will be connected and speech enabled. Speech engines that are deployed today and those under development by major speech companies such as Google and Apple are server and cloud based. Hence, every recorded speech utterance is sent to the servers via the Internet, gets processed and the appropriate command is sent back to the connected device to make the appropriate response. This promises a great deal of accuracy and speed thanks to the Internet infrastructure and servers processing power. This is considered a great step in the technological race for powerful human machine interaction. However, it also raises a serious privacy issue as

every word or sound anyone makes is recorded and sent for processing over the Internet. Even if the speech companies do not use the collected data for other than what is explicitly intended, a number of entities such as hackers could intercept the data and collect it for malicious use. Continuous surveillance is not an attractive idea to most individuals. However, it is possible to get the best of both worlds: privacy and technological advancement. The key to such compromise is to perform the speech recognition locally without having to send the data over the network. The speech utterance is not at risk of interception and the intelligent embedded system would be able to decipher the command and act accordingly. While small embedded platforms such as a switch, a fridge or a lamp are not equipped with the proper hardware to handle large vocabulary speech recognition, most designers of theses systems believe that a few trigger words and commands are more than enough to activate them. Thus, an efficient keyword spotting is capable of activating most IoT devices.
Keyword spotting on small embedded platforms comes with a few challenges including limited memory and processing power. In fact, the connected devices should remain cost effective at lower power consumption. In addition, IoT components are subject to acoustic interference as they would be placed all over the environment which makes robustness of the speech engine to noise an important aspect. Therefore, designing a small foot-print, low power consumption and robust keyword spotting engine would be the ideal solution to speech enabled IoT devices that preserve privacy and offer a comfortable interface with the users.

1.2 Thesis Contributions

The main goal of this thesis is to develop an effective approach for designing a noise robust small footprint phoneme classifier system which is also computationally efficient. This will be accomplished through two contributions. The first contribution is designing a system that has the ability to replace a software or hardware denoiser by including noisy data in the training process of the machine learning technique. This reduces the cost, memory requirement and increases the robustness of the speech engine and the processing speed. This is achieved by applying multi-condition training of the recognizer. The targeted noises are represented by injecting the corresponding noise into the training data enabling the model to simulate the noisy environments as it is trained with the clean and noisy data. Hence the denoising aspect is no longer a separate module but rather incorporated in the deep learning model itself. The second is to reach a good compromise between the clean and noisy data training, replacing the need for multiple noise models with a single one that

provides accurate and robust results for both clean and noisy conditions. This also serves the purpose of reducing memory footprint and decreasing the pre-processing delay caused by the model selection step.

1.3 Thesis Outline

The thesis starts with a detailed description of the major approaches used in the development process of the robust keyword spotting engine. We start in the second chapter by highlighting various speech recognition and keyword spotting algorithms. Along the way, we outline the theoretical background on Deep Neural Networks (DNN) speech enhancement tools. Following the presentation of the background material, the proposed approach to reach a robust and small keyword spotting engine is then presented in chapter 3. This includes the selected techniques alongside the appropriate test scenarios to test the performance of the proposed approach when compared with the state-of-the-art techniques. Finally, the test scenarios results are presented and discussed in chapter 4. A conclusion is drawn to assert whether the proposed approach has reached the expected outcome.

Chapter 2

Technical Background and Literature Review

2.1 Introduction

In this chapter, a general description of the speech recognition process and its main components is introduced. Next, the different methods of keyword spotting are highlighted and compared. The objective is to choose the best technique for keyword spotting to use in the process of designing a robust DNN based keyword spotting engine. Artificial Neural Networks (ANN) and their previous use in speech recognition is then described followed by more details on DNN. This chapter, concludes with a presentation of various speech enhancement approaches found in the literature.

2.2 Automatic Speech Recognition (ASR)

2.2.1 Definition

Speech recognition is the ability of a computer to identify and respond to the sounds produced in human speech [59]. We call Automatic Speech Recognizer (ASR) the system that identifies the speech utterance and translates them to text while we call Natural Language Understanding (NLU) the system that interprets the stream of utterances, allowing the interface machine to respond accordingly. This thesis only targets the first module, hence producing text given an audio signal.

2.2.2 Components of an ASR

An ASR is composed of two major components, namely a feature extractor followed by a decoder that requires a language model and an acoustic model to produce an estimate of the audio signal transcript. Figure 2.1 illustrates the relationship between the different modules and required elements for a fully functioning ASR [60].

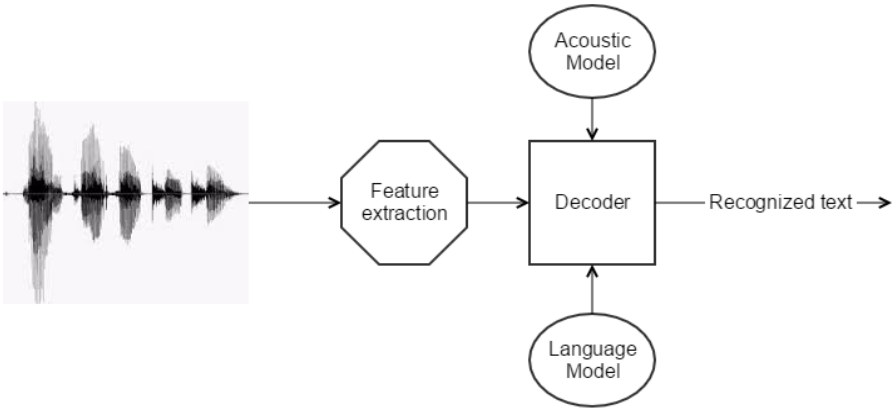

Figure 2.1: Speech recognition process

For feature extraction, there is a variety of techniques to derive features from the signal, each emphasizing on linguistically relevant or irrelevant information. Robustness can be addressed at the features level, making the decoding task easier [78]. However, it has been proved that enhancing the features for more robustness has minimal impact on the speech recognition process as a whole. Decoding the features is usually done in two steps. The first step is performing classification by any of the pattern recognition techniques which requires the use of an acoustic model. One example is using continuous classification of the feature vectors, by modeling the parameters of the distribution equation representing each class, commonly with Gaussian Mixture Models (GMM). A fairly new technique based on neural networks to classify a speech utterance by estimating the probability for different classes. The second step in the decoding process is hypothesis search, usually performed by the Hidden Markov model (HMM), a formulation that expresses the constraints of pronunciation and word sequence in a single finite-state network, for which efficient search and training algorithms are known. The combination of the dictionary, the phonetic description of each word, and the grammar which details the relationship between the dictionary words is called the language model [60].

2.2.3 Feature extraction

Representing the speech signal is possible in both time and frequency domains. Since the audio signal is in the time domain, using a similar representation, temporal waveforms [34], seem the best way to go. However, according to [34], using frequency domain representation is more useful as it better depicts the different sounds and is more accurate. In fact, the human ear also relies on frequency based sound's representation. Since most of the sound is not speech, continuous listening will collect much useless data and waste energy. Using Voice Activation Detection (VAD) enables the system to only extract the utterances that contain speech and ignore the irrelevant sounds and noises.

Mel-frequency cepstral coefficient (MFCC)s are the frequency domain features representation that are used in the scope of this thesis, but the audio signal input has to go through an analog to digital conversion before getting to the frequency domain representation. The latter is performed using a specific sampling rate depending on how much details we want to extract out of the analog signal. Having sampled clean speech data will need a representation that would have the benefits of the frequency domain representation but also what could be recovered from the time domain. The most commonly used MFCC features are extracted through performing a Discrete Fourier Transform (DFT) followed by a number of Mel scale filters generating the Mel filter bank. At last, Discrete Cosine Transform (DCT) is used to generate the MFCC [84]. Experiments prove that 13 MFCCs are the most representative to perform speech recognition [48]. To regain some of the temporal information lost when performing the DFT, the differential and the acceleration coefficient are added to the 13 MFCC reaching a vector of 39 features for each frame. The calculation of the differential and the acceleration coefficient helps estimate the temporal change in the signal [31]. Equation 2.1 illustrates the calculation of the deltas and double deltas of the 13 MFCCs.

$$d_t = \frac{\sum_{n=1}^{N} n(c_{t+n} - c_{t-n})}{2\sum_{n=1}^{N} n^2} \tag{2.1}$$

where d_t is a delta coefficient, from frame t computed using the MFCC coefficients c_{t+N} to c_{t-N}. Double delta coefficients are computed the same way, by replacing the MFCC coefficients with the deltas [4].

2.2.4 Hybrid HMM/GMM

A HMM is a finite state machine, where transitioning from one state l to another state m is performed every time frame t with probability a_{lm}. While entering the state m, an observation vector x_t is produced with a probability distribution $b_m(x_t)$. Figure 2.2 illustrates a HMM which has two states with no probability emission, one before the beginning of the chain and another after the final probability has been calculated. The remaining states have emission probability densities that are usually estimated by GMM [60, 19].

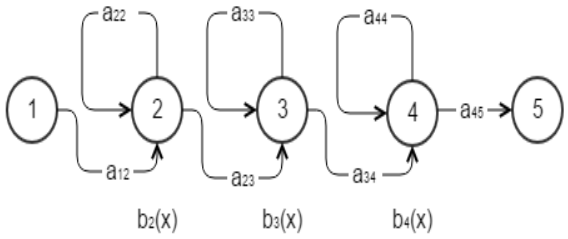

Figure 2.2: Hidden Markov Model states

For every emitting state, the HMM can stay at its current position or move to the adjacent state. Each of the speech units requires some acoustic data for training. The data could be from one or different speakers for better training of the HMM parameters.

HMMs in speech recognition are used to model the speech signal sequential structure, each of the states have a Gaussian density that helps model the signals spectral variability. With the Expectation Maximization (EM) algorithm, Gaussian models fit easily the data. In fact, they are quite effective in modeling the HMM states to MFCC. With the adequate number of mixtures, GMM can effectively model the probability distribution of the HMM states over their feature vectors. However, GMM in HMM-based acoustic models still suffer limitations motivating the research community for alternative models [60].

2.3 Keyword Spotting (KWS)

This section tackles in detail the well known four Keyword Spotting (KWS) methods, Large-Vocabulary Continuous Speech Recognition (LVCSR) KWS, Acoustic KWS and Phonetic Search KWS, followed by a discussion and comparison of these methods. The figures in this section are inspired from [54].

2.3.1 LVCSR-based KWS

The LVCSR translates the speech utterance into text and feeds it to the KWS engine. The text output of the speech recognizer is searched by the KWS engine to determine the existence and the positions of a specific list of keywords within the text output [77]. Applying the LVCSR based method in keyword spotting is performed over two steps. At first, using Viterbi search, a LVCSR that requires a large size acoustic and language models is performed on the audio utterance, generating text output. Next, a Keyword spotting procedure is used to identify the presence of the keyword list in the previously generated text. In [68, 53, 10], LVCSR keyword spotting method was tested giving quite unsatisfactory results of only 61.2% accuracy. Figure 2.3 illustrates the different components of an LVCSR based keyword spotting engine [55].

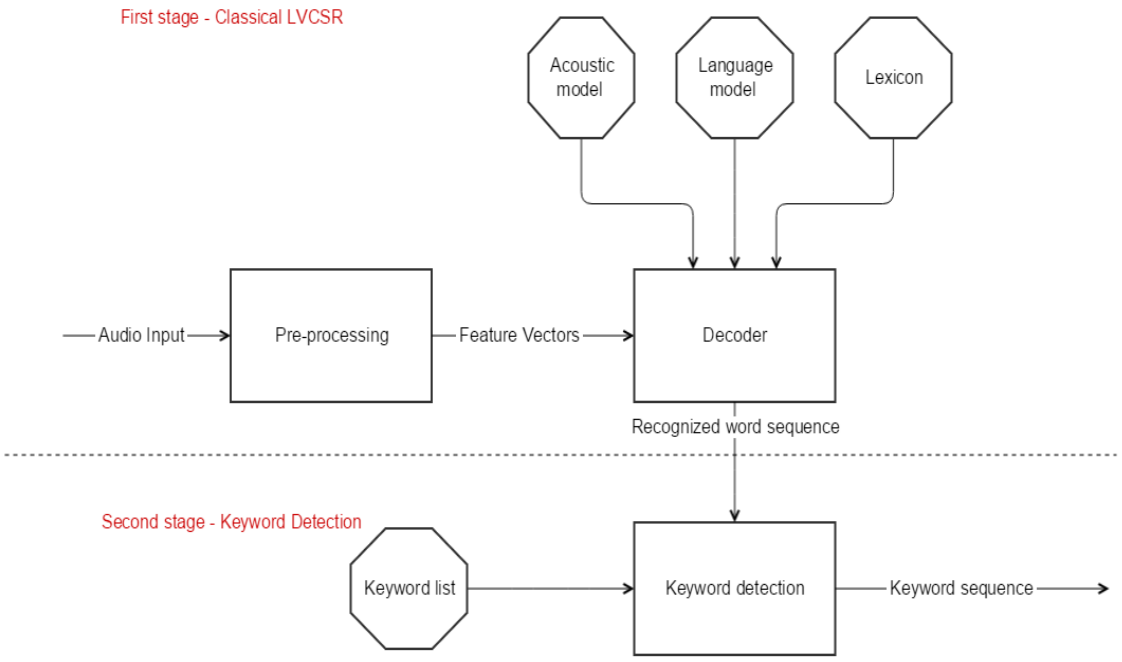

Figure 2.3: LVCSR based keyword spotting[55]

2.3.2 Acoustic KWS

Unlike the LVCSR, the acoustic KWS system does not estimate the whole text of the audio, yet it uses the same search algorithm. Instead of having one large acoustic model trained using representative data of the English language, this technique performs speech recognition based on a small subset of specific words alongside a general non-keyword model. Having a model for keywords and another for other words that are usually referred to as "garbage", the acoustic KWS can execute its search in one single step [68] as shown in Figure 2.4. In fact, Google [9] uses a similar method to train their trigger word "Okey Google" using DNN. Instead of having multiple models, they train their neural network using multiple occurrences of the trigger word with another training set classified as filler [55].

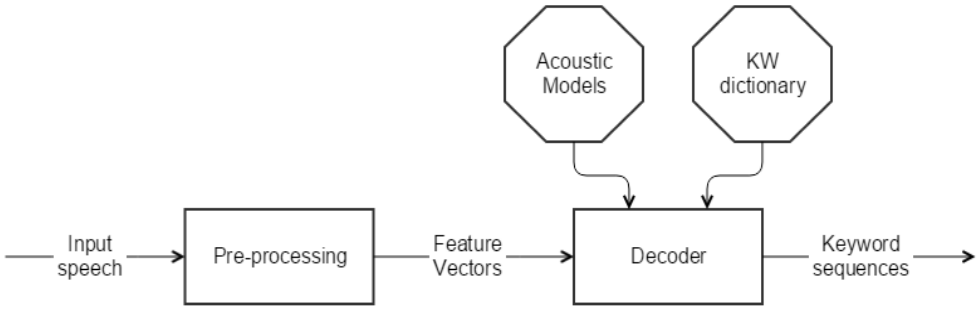

Figure 2.4: Acoustic based keyword spotting[55]

2.3.3 Phonetic search KWS

Phonetic search KWS consists of finding the keyword based on their phonetic transcript which is performed over two steps. The first step performs phoneme decoding that transforms the audio input to an array of phonemes in contrast of the LVCSR that produces a list of words [2, 64, 70]. The second consists of a phonetic search calculating the distance between the produced phonetic sequence from the previous stage with the list defining the keywords [2]. Figure 2.5 illustrates the different stages of the phonetic search process. It is noted that the engine requires a keyword list with their phonetic description alongside a general phoneme database. The latter is used by the decoder to produce an estimate of the phoneme sequence present in the input audio file [55].

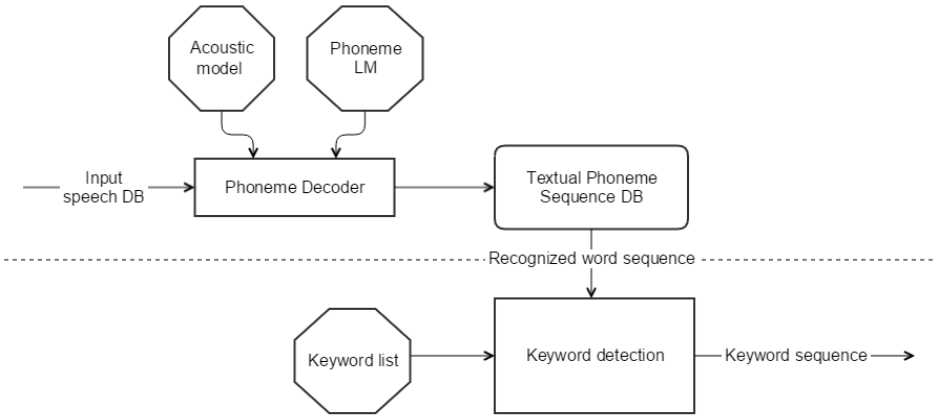

Figure 2.5: Phonetic search based keyword spotting[55]

2.4 Comparison: Performance Indexes

The major comparison metrics for the above mentioned KWS techniques are accuracy, processing time, keyword level flexibility and training data availability [32, 18, 46, 62]. These metrics are described next.

2.4.1 KWS accuracy

Relying on the textual transcript of the input speech, either in the case of LVCSR based or phonetic search based KWS, is subject to a great deal of mismatch due to disturbances in the testing environment. In fact, the quality of the microphone, ambient noise, mispronunciation, word overlap and even accents would usually lead to a poor performance of the decoder. The resulting text will be filled with substitution, deletion and insertion errors [6, 24]. The accuracy also depends on the quality of the database and how well it mimics real-life environment [6, 7]. Since the KWS, in both mentioned methods, relies on the decoder output, the accuracy is greatly influenced by the first stage results. The main difference between the phonetic search and the LVCSR based KWS is on the possibility of increasing the accuracy at the second stage. For the LVCSR KWS, the second stage is limited to searching the words and detecting whether a keyword is in the text or not, making it hard to improve accuracy. However, in the phonetic search, there is room for improvement as one keyword is composed of several phonemes. Hence, if even one or two

phonemes are mistakenly classified, using a smart second stage algorithm the correct keyword could be detected. Hence, in theory the accuracy in the phonetic search based KWS is better than the LVCSR based KWS.

2.4.2 Response time

On the overall complexity level, phonetic search based and LVCSR based KWS both rely on a two-step process: First in converting the speech to textual format (words for LVCSR and phonemes for phonetic search based). On the other hand, acoustic based KWS does not rely on any kind of transformation and performs on the speech input in one single step. Although searching for the correct keyword, while given a full text in the LVCSR technique, is quite fast, the first stage is considered a huge drawback when it comes to time response. This is caused by the large size of the vocabulary and the complexity of the models involved in the transcript generation stage.

Phonetic search implements phoneme recognition without the need of any lexicon or word level language model. Nonetheless, the second stage search calculates the distance between the phonetic textual representation from the first stage and the keywords phonemes. In contrast with LVCSR method that reaches the word level from the first stage, generating a hypothesis of the keywords is necessary for the phonetic search KWS, based on the sequence of phonemes produced by the first stage. However, acoustic KWS only needs the list of keywords without requiring any language model or phonetic description of the keywords. Requiring small vocabulary and operating on the input signal, the acoustic based KWS is optimal for real-time KWS in small databases. In [9], Google is using this technique for the trigger command recognizer on the handheld devices using Android operating systems. Yet, this also means that the model must be very accurate and well trained [70] to sidestep an excess of false alarms.

2.4.3 Training data availability

For LVCSR and phoneme search based KWS, any general database such as Texas Instruments and Massachusetts Institute of Technology (TIMIT) or the World Street Journal (WSJ) should be sufficient to train the model [41, 37]. On the other hand, Acoustic based KWS requires a database that is very specific for the specific keywords. Such a database is very hard and almost impossible to obtain, making this method exclusive to big companies with vast resources such as Google and Apple [9].

2.4.4 Keyword flexibility

Working with the smallest unit of speech, namely phonemes, the phonetic search based KWS is at a great advantage when it comes to keyword flexibility as LVCSR runs on a word sequence [7, 5, 76]. The phonetic search offers the user absolute freedom on changing the keywords as needed, provided that phonemes have no vocabulary constraints. One other flexibility aspect that phonetic search offers is the ability to change the keyword and not having to run the decoding stage again. The second stage is sufficient to spot newly defined keywords. In contrast, the textual transcript formed by LVCSR requires running both stages again as the engine is constrained with a vocabulary and a language model unless the keyword is already present in the language model. [7, 68, 45]. Given that keywords are usually domain specific terms and are not included in common lexicons [76, 24], then keyword flexibility represents a great disadvantage of the LVCSR technique.

Acoustic KWS is not adequate for frequent changing of the keywords. Since acoustic KWS is a one-step process that uses a keyword specific model, changing the keyword list would required a retaining of the model. Most real-world applications need keyword flexibility, alongside the fast real-time execution when applied to any type of speech input. According to [55], Phonetic search KWS technique is more efficient than LVCSR and acoustic-based methods. The focus of this thesis is phonetic search KWS, and the implementation of an algorithm for the reduction of computational complexity in the phonetic search KWS process.

The following table 2.1 summarizes the comparative study between the different KWS approaches[55].

KWS approach	**Keyword flexibility**	**Response time**	**KWS accuracy**	**Data availability**
LVCSR based	high	low	high	abundant
Acoustic based	low	high	high	scarce
Phonetic search based	high	medium	medium	abundant

Table 2.1: KWS comparison summary.

It is clear that there is no single best approach with high efficiency for all three metrics. However, for this thesis, more emphasis is put on keyword flexibility as embedded systems

are known to be versatile and are produced in big numbers, hence retraining for each keyword change would make the KWS engine less practical and less desirable. In addition, the data availability plays an important role as it excludes the acoustic KWS technique due to the lack of training data. Therefore the phonetic search based approach is adopted in this thesis as our platform for KWS.

2.5 DNN for Speech Recognition

Prior work for implementing feed-forward ANN instead of GMM within a HMM-based acoustic model has its advantages [47]. In order for the GMM to approximate the posterior probabilities, they require comprehensive detailed data distribution assumptions, while the ANN do not. ANN also let us use and apply different types of data, whether they are continuous or discrete [71]. Last but not least, ANN use the entirety of the training data to tune the network parameters and they have proved very efficient in modeling highly non-linear data. However, the ANN had many limitations when modeling speech data. This is due to the highly variant nature of speech [71]. Deeper networks (networks with many layers) have been proposed as the solution for more complexity to fit data such as speech, but the random initialization of the network parameters led to negative results when using Multi-Layer Perceptron [71]. In 2006, Hinton introduced the pre-training step to initialize the weights and therefore was able to solve the problem of initialization [28]. Deep Belief Networks (DBN) have been the most common technique for generatively pre-train deep networks where each layer extracts a higher representation and new structures from the input. A popular example that explains the abstraction aspect of the deep neural network is image processing. Figure 2.6 [33] shows the different abstractions that an image of a traffic sign could go through depending on the DNN depth. It can be seen that the first layer highlights the edges of the input image while decreasing the feature dimension from 32×32 to 28×28. The deeper we go the more abstract the features become.

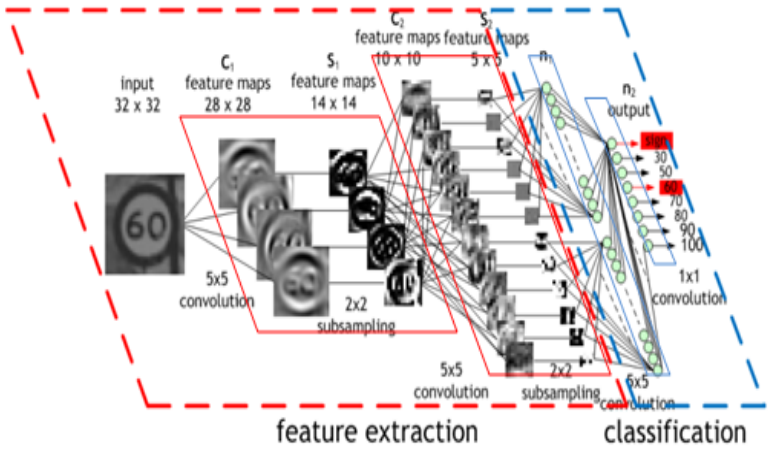

Figure 2.6: Abstraction layers of DNN [33]

.

Recent deep learning algorithms have shown that for acoustic modeling simulations, neural networks are more appropriate and very effective [3]. The two-steps algorithm to train neural networks goes as follows: The first step of the DNN process consists of initializing the weights of the network using generative pre-training, usually with Restricted Boltzmann Machines (RBM) or Stacked Autoencoders (SDA). Each of the layers offer a new abstract representation of the input data. Next, a discriminative back-propagation step takes over and performs the supervised learning step of the network. For pre-training deep networks, DBN is the most common method used in the literature. The pre-training reduces the likelihood of overfitting and makes the fine tuning stage less computationally intensive as the initialized weights may lead to a local or even global minima [28]. Thus, the backpropagation algorithm will reach convergence in less iterations. Hybrid HMM and DNN architectures for speech recognition acoustic modeling combine both, the sequential modeling of the HMM and the abstract representation of the DNN [58].

2.5.1 Stacked autoencoders (SDA)

A SDA model is used to learn new generic features, and as such is part of a representation learning system. It has been used for speech recognition in the recent years and has shown good prospects for DNN [74]. Stacked Autorncoders are, as the name implies, a stack of single-level autoencoders as shown in Figure 2.7. Hence the SDA are a deep learning

approach. SDA use the autoencoders as building blocks to create a deep network. While deep architectures can be more expressive and can extract more sophisticated features from input data, until relatively recently, deep networks were thought to be too difficult to train and as such of limited utility. As mentioned above, the breakthrough came when Geoffrey Hinton and his colleagues [28] showed how fast, layerwise greedy and unsupervised algorithms can be used to initialize a slower algorithm that fine tunes the learned weights and provides very good results on deep networks. This result revitalized machine learning research and deep architectures have become a tool of choice for a wide variety of classification and prediction problems [49].

The basic idea behind layerwise training is shown in Figure 2.7. After a layer is trained, the autoencoder output layer is discarded and the features (the y) are used as the input to the next layer. Hence the training is greedy and layerwise. The final step is to fine tune the network in a supervised fashion using the backpropagation algorithm.

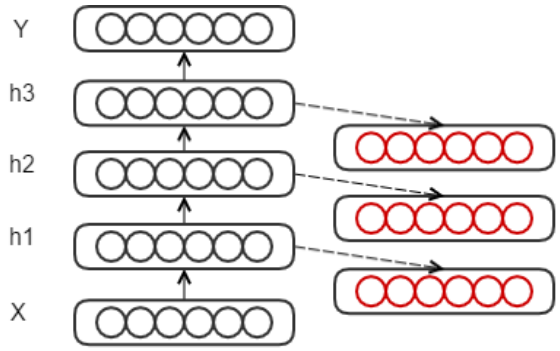

Figure 2.7: Stacked autoencoders

.

2.5.2 Deep belief networks (DBN)

In this section, the background required to understand the main prediction approach used in this research, namely DBN, is provided. The method is particularly suitable when dealing with a very large amount of data generated by complex interconnected systems. DBN are formed by stacking RBMs, which are then trained greedily using unsupervised algorithms [28]. An RBM is an undirected graphical model that has no connections within visible ($\mathbf{x}$) or hidden ($\mathbf{h}$) units, as illustrated in Figure 2.8.

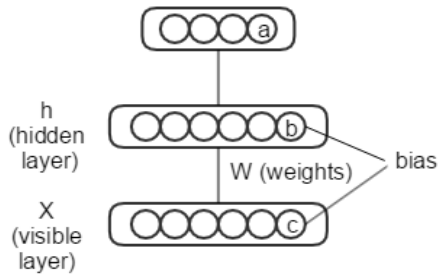

Figure 2.8: Restricted Boltzmann Machines

RBM are developed based on the inter-layer interaction energy and bias energy of each layer. Mathematically, the energy function of an RBM is defined as:

$$E(\mathbf{x}, \mathbf{h}) = -\mathbf{b}^\top \mathbf{x} - \mathbf{c}^\top \mathbf{h} - \mathbf{h}^\top \mathbf{W} \mathbf{x} \tag{2.2}$$

where $\mathbf{b}$ and $\mathbf{c}$ are bias vectors associated with the hidden and visible layers respectively, and where $\mathbf{W}$ is the weights matrix between the visible and hidden layers. We denote the set of parameters $\mathbf{b}, \mathbf{c}$, and $\mathbf{W}$ by θ. An RBM defines a distribution, which involves the hidden units, over the visible units. The distribution of observing a particular visible and hidden units configuration based on the energy function is given by:

$$P(\mathbf{x}, \mathbf{h}) = \frac{e^{-E(\mathbf{x},\mathbf{h})}}{Z} \tag{2.3}$$

where Z is the partition function. Using Equation 2.3, the distribution of observing a set of visible units is defined as:

$$P(\mathbf{x}) = \sum_{\mathbf{H}} P(\mathbf{x}, \mathbf{h}) = \exp \frac{-F(\mathbf{x})}{Z} \tag{2.4}$$

where $F(X)$ is known as the free energy term and H is the number of hidden layers, and is defined as follows:

$$F(\mathbf{x}) = \exp \left(\mathbf{c}^\top \mathbf{x} + \sum_{j=1}^{H} \log(1 + \exp(b_j + \mathbf{W}_{(j,:)} \mathbf{x})) \right) / Z \tag{2.5}$$

To train an RBM, the average negative log-likehood (NLL) function of all training points $(t = 1, ..., T)$ needs to be minimized. The NNL function is set as follows:

$$NLL = \frac{1}{T} \sum_{t} l(f(\mathbf{x}^{(t)})) = \frac{1}{T} \sum_{t} - \log P(\mathbf{x}^{(t)}) \tag{2.6}$$

Then, the NNL function is optimized with respect to θ using stochastic gradient descent algorithm; the result of the optimization is defined as:

$$-\frac{\partial \log P(\mathbf{x}^{(t)})}{\partial \theta} = \mathbb{E}_{\mathbf{h}}\left[\frac{\partial E(\mathbf{x}^{(t)}, \mathbf{h})}{\partial \theta} \,|\mathbf{x}^{(t)}\right] - \mathbb{E}_{\mathbf{x},\mathbf{h}}\left[\frac{\partial E(\mathbf{x}, \mathbf{h})}{\partial \theta}\right] \tag{2.7}$$

Where $\mathbb{E}$ designates the expectation with respect to the prior knowledge in the case of $\mathbb{E}_{\mathbf{h}}$ and with respect to the observation alongside the prior knowledge in the case of $\mathbb{E}_{\mathbf{x},\mathbf{h}}$. The second term in the right-hand side of the equation is hard to compute since we have to make an exponential sum over both $\mathbf{x}$ and $\mathbf{h}$. To address this problem, the authors of [8] proposed the contrastive divergence algorithm, which has three main steps:

1. Estimate a point $\tilde{\mathbf{x}}$ to replace the expectation $\mathbb{E}_{\mathbf{x},\mathbf{h}}$.
2. Estimate the point using Gibbs sampling.
3. Start the sampling chain at each observation $\mathbf{x}^{(t)}$.

As illustrated in Figure 2.9, the conditional independence of intra-layer units in an RBM allows us to apply the Gibbs sampling iteratively for sampling visible and hidden units.

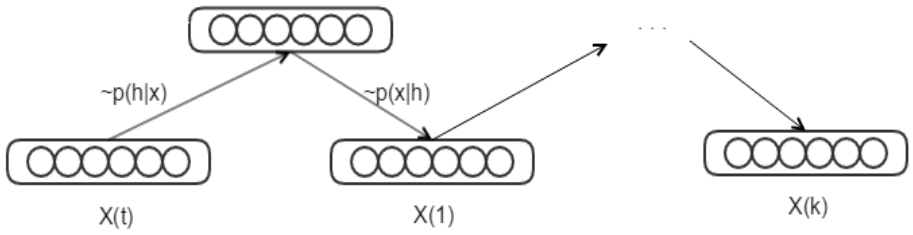

Figure 2.9: Illustration of Gibbs sampling

To perform the Gibbs sampling, the conditional distributions $P(\mathbf{h}|\mathbf{x})$ and $P(\mathbf{x}|\mathbf{h})$ have to be computed according to Equation 2.8.

$$P(\mathbf{h}|\mathbf{x}) = \prod_i P(h_i|\mathbf{x})$$
$$P(\mathbf{x}|\mathbf{h}) = \prod_k P(x_k|\mathbf{h}) \tag{2.8}$$

If $\mathbf{x}_k$ and $\mathbf{h}_j$ are binary units, the following applies

$$\begin{aligned} P(h_j = 1|\mathbf{x}) &= \frac{1}{1 + \exp(-(b_j + \mathbf{W}_{(j,:)}\mathbf{x}))} \\ &= \text{sigm}(b_j + \mathbf{W}_{(j,:)}\mathbf{x}) \\ P(x_k = 1|\mathbf{h}) &= \frac{1}{1 + \exp(-(c_k + \mathbf{h}^\top \mathbf{W}_{(:,k)}))} \\ &= \text{sigm}(c_k + \mathbf{h}^\top \mathbf{W}_{(:,k)}) \end{aligned} \tag{2.9}$$

After the negative sample $\mathbf{x}(k) = \tilde{\mathbf{x}}$ is estimated, the point estimate of the expectations in Equation 2.7 is computed as follows:

$$\begin{aligned} \mathbb{E}_{\mathbf{h}} \left[\frac{\partial E(\mathbf{x}^{(t)}, \mathbf{h})}{\partial \theta} \Big| \mathbf{x}^{(t)} \right] &\approx \frac{\partial E(\mathbf{x}^{(t)}, \tilde{\mathbf{h}})}{\partial \theta} \\ \mathbb{E}_{\mathbf{x},\mathbf{h}} \left[\frac{\partial E(\mathbf{x}, \mathbf{h})}{\partial \theta} \right] &\approx \frac{\partial E(\tilde{\mathbf{x}}, \tilde{\mathbf{h}})}{\partial \theta} \end{aligned} \tag{2.10}$$

Even with only one step sampling, i.e., $k = 1$, this algorithm has been proven to be successful for unsupervised training [8]. Using Equation 2.10, the parameter θ of the RBM are updated according to the following equation:

$$\theta^{(t+1)} = \theta^{(t)} - \alpha \left(\nabla_\theta (-\log P(\mathbf{x}^{(t)})) \right) \tag{2.11}$$

Where α is the learning rate. For each parameter $\mathbf{W}$, $\mathbf{b}$, and $\mathbf{c}$, the update equations are given below:

$$\begin{aligned} \mathbf{W}^{(t+1)} &= \mathbf{W}^{(t)} - \alpha \left(\mathbf{h}(\mathbf{x}^{(t)}) \mathbf{x}^{(t)^\top} - \mathbf{h}(\tilde{\mathbf{x}}) \tilde{\mathbf{x}}^\top) \right) \\ \mathbf{b}^{(t+1)} &= \mathbf{b}^{(t)} - \alpha \left(\mathbf{h}(\mathbf{x}^{(t)}) - \mathbf{h}(\tilde{\mathbf{x}}) \right) \\ \mathbf{c}^{(t+1)} &= \mathbf{c}^{(t)} - \alpha \left(\mathbf{x}^{(t)} - \tilde{\mathbf{x}} \right) \end{aligned} \tag{2.12}$$

where $\mathbf{h}(\mathbf{x}) \triangleq \text{sigm}(\mathbf{b} + \mathbf{W}\mathbf{x})$.

Furthermore, once the unsupervised training is done, the obtained parameters are used to initialize the neural networks that will be trained using a supervised back-propagation algorithm. In this work, the neural networks are trained to learn phonemes in a multi-class setting. Figure 2.10 depicts a multi-class classification with a soft-max layer on top of the stacked RBM.

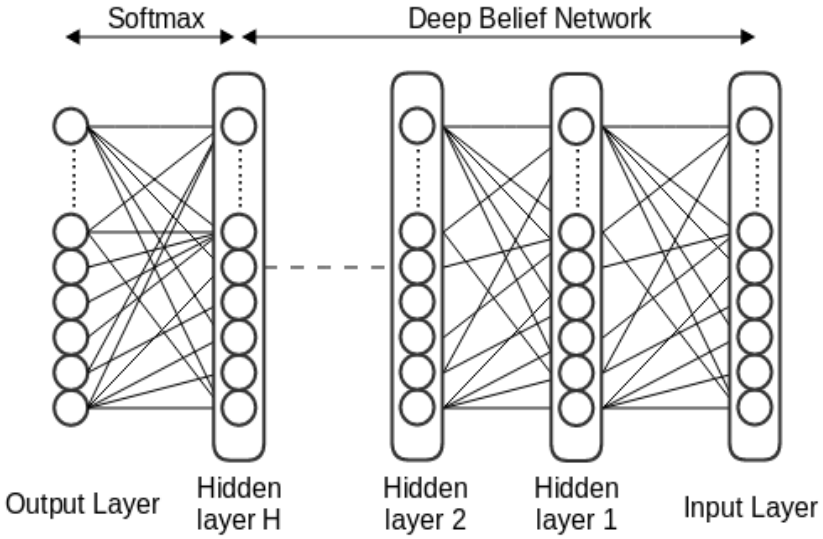

Figure 2.10: DBN architecture for multi-class classification

2.6 Noise Reduction

Speech recognition tests are usually carried out in ideal environments within laboratories, hence their accuracy level is substantially degraded when tested in real world conditions. The different noises and disturbances accompanying the highly variant testing environments create a considerable mismatch between the training and the testing set. Hence the accuracy degradation [57].

The latter mismatch has been the motivator behind more robust speech engines. Increasing robustness of the ASR can be achieved in the following three levels:
i) Improving the Signal to Noise Ratio (SNR) in the acoustic level by multiple speech enhancement approaches [81, 79]
ii) Choosing a parametric representation that is more robust to noise [23, 39] and
iii) Including both the noisy and clean signal in the modeling stage allowing the new model to recognize the speech under specific noisy environment [42].

In the scope of this thesis, the third type of speech enhancement technique is used. Dealing with the mismatch is a crucial step in the speech recognition development and its introduction to real world environments. Although a lot of research has been dedicated to speech enhancement over the last decades, most of the proposed approaches under

a multitude of assumptions were unsuccessful. [40, 14, 21]. Artificial Neural Networks, with only one layer (also known as shallow networks), have been introduced to act as nonlinear filters [69, 35]. Due to the simplistic representation of the shallow neural network and the usually small size of the training set, the results have not been encouraging. In addition, the random weight initialization, and the gradient-descent or ascent optimization get the network training process stuck in what is called local minimas or plateau [3]. This problem gets more pronounced when increasing the number of layers for a deeper architecture. Deeper networks are known to provide a better solution, but the latter problem has prevented any progress in that direction. In 2006, Hinton et al. [29] introduced a greedy layer-wise unsupervised pre-training that initializes the network weights avoiding the local minima obstacle, hence resurrecting the use of DNN. One of the first fields Hiton and al. chose to test the DNN was speech recognition. The obtained results are overwhelmingly better than the state-of-the-art recognizers [27, 83]. Binary classification on time-frequency approaches have also been tested, yet the time-frequency units proved not to be a good choice for classification. Ideal ratio mask estimation, in the MFCC features, is also performed using DNN for purposes of increasing the robustness of the ASR [56]. Another use of DNN is mapping a complex function from noisy to clean data using training data varying factors in noisy speech such as different speakers, different noise types, and SNRs in [79], leading to a major improvement. In fact 76% of subjective listeners prefer the DNN based clean speech over the other techniques.

2.7 Summary

In this chapter, The three major keyword spotting approaches, namely the acoustic, the LVCSR and the phonetic search based were highlighted and their performances compared. Phonetic search KWS had been shown to have more promising features and it is selected here as the KWS method. Recent use of DNN with Restricted Boltzmann Machine and Stacked Autoencoders in speech recognition was also described, suggesting that both pre-training algorithms should be tested and compared. Finally, the standard techniques for speech enhancement were presented.

This thesis focuses on training a DNN model using large size speech data under various conditions, encompassing clean, stationary noise contaminated, non-stationary noise contaminated and stationary then non-stationary noise contaminated to simulate various real-life environments. This is developed in the next chapter.

Chapter 3

Proposed Approach For Keyword Spotting Robustness

3.1 Problem Statement

Most of the speech databases that are commonly used by the industry as well as by researchers, such as TIMIT or WSJ contain large amount of data and are quite representative of the diversity of potential speakers [41, 37]. These databases encapsulate a multitude of aspect variations of the targeted speech domain. In fact, they include speakers of different gender, age and sometimes even with different accents [1]. This variety ensures high level of the database generality. Indeed, most of the speech recognition applications using such databases for training their models are known to be efficient and accurate [60].

However, theses databases present a substantial downside, namely that they are recorded in a clean environment. In fact, testing the laboratory grade trained and tuned models under real-life circumstances results in poor accuracy. This is due to the fact that the model is not trained to deal with the noise accompanying the speaker utterances.
Noise is an important factor and not to be ignored, as both the speaker and the background can create noise that greatly disturbs the input signal making it hard for the decoder to decipher and recognize the spoken phonemes or words [52].

This is where the denoiser intervenes by trying to enhance the speech and estimate the original clean speech to feed it to the decoder. This should lead to better results depending on how efficient is the denoiser estimation. Denoisers are also trained with specific noises which make them dependent on the mismatch between the noise used for training and the

noise present while testing. Nevertheless, for the same type of noise, whether stationary or non-stationary, the denoiser is somehow efficient [63, 20, 13].

In an environment with changing types of noise, the convenient noise-specific denoiser model has to be chosen before applying it to the speech signal. Recently developed smart denoisers approximate the noise type and apply the correct model, one at a time. Denoising proves to be usually effective, even though it adds an overhead for the process. It also created a delay that is affects the real-time functionality which is at the core of every speech enabled system.

The next challenge is when the environment presents two or more different types of noises at the same time. The denoiser would eventually choose one model to filter out the noise and would fail to treat the other type of noise hence usually leading to poor accuracy![26].

One approach to solve coexistent noise types problem is to use the noisy data in the training stage to produce a model that handles more than one type of noise at the same time and not having to deal with the overhead of a denoiser. Since the available databases are composed of clean data, a noise adder is used to contaminate the data with specific noises generating the noisy data needed for the training. Creating different datasets using several noise samples produces a representative database of most real-world environments. However, the model becomes quite complicated and the traditional modeling approaches cannot capture the high level of non-linearity of the noisy data. This lead to the abandonment of the noise modeling technique and the focus on improving the denoiser-based method. With the recent development and success of deep learning, attempting to model the noisy utterance using deep learning may prove to be an excellent choice.

This thesis attempts to improve the recognition results in noisy environments using deep learning. The next section details the proposed approach for better accuracy using the noise modeling technique on a phoneme recognition engine. The results are compared to those of the denoiser-based method.

3.1.1 Noise robustness using deep neural networks (DNN)

Thanks to recent advancement of deep learning [28], DNN has been used successfully in a multitude of areas including speech recognition [27]. The next step is to use deep learning to enable the speech engine with noise robustness. Several methods have been used to incorporate noise robustness into the DNN training in the speech recognition area [65]. These methods are:
i) Multi-condition utterances training

ii) Denoising the utterances before the training
iii) Noise estimation incorporated into the network.
These approaches are similar to noise robustness techniques in the traditional HMM-GMM engines [75]. The following is a brief description of each method.

Multi-condition utterances training

Using multi-condition data for DNN training permits a higher level features learning by the network. These features are more robust to the noise effect on the overall classification. In this regard, DNN is considered as nonlinear feature extractor and also a nonlinear classifier. The lower layers represent discriminative features that are independent from the various conditions across the many acoustic conditions existing in the training data. Hence in multi-condition utterances training data, the input vector is a combination of the noisy utterances frames. Although the multi-condition technique is theoretically similar for both GMM and DNN, they are substantially different. For GMM, a Gaussian mixture directly models the data, hence the noise introduced variability is captured and modeled. In discriminative training, noisy features are discarded by the GMM while the deep neural network extracts helpful information using the nonlinear processing of the layers [75].

Enhanced features for DNN training

One intuitive solution to noisy data is to filter out the noise from the speech utterances before the training stage. Hence, using a speech enhancement technique reduces the noise effects on the input signal. The classifier learns any flaw introduced by the enhancement algorithm if the latter is used on both, the testing and training data. The HMM-GMM version of this technique is called feature space noise adaptive training [75, 17]. The same technique could directly be applied to the DNN training.

DNN noise-aware training

The last technique consists of adapting the model to the environment noise by introducing a noise estimation in the model itself. The noise model adapts the GMM parameters based on a model that determines how the clean speech is corrupted. Equation 3.1 describes the relationship between the noise, the noisy signal and the clean speech.

$$y = x + \log(1 + e^{n-x}) \tag{3.1}$$

Where x designates the clean speech, y is the noisy signal and n is the noise. The non-linearity of Equation 3.1 makes noise robustness a considerable challenge. Yet, thanks to various layers of nonlinear processing of DNN, the network is capable of learning the relationship from the data. This is done by appending each input with an estimate of the noise [75]. The DNN is informed of the noise and not adapted to it, therefore this technique is called DNN noise-aware training.

According to [75], multi-condition utterances training is the best approach for noise robustness, combined with the dropout technique, the accuracy is improved by 7.5% relative to the best published result in speech recognition. Therefore, the multi-condition training is adopted in this work as the DNN approach to achieve noise robustness.

3.2 Proposed Approach

In order to fulfill the need of the fast growing market of the IoT, the main goal of this work is to develop a keyword spotting engine that is robust to noisy environments and that could also sit on a platform with considerable memory and processing power constraints. This is accomplished by applying multi-condition utterances training of the recognizer. The targeted noises are injected into the training data enabling the model to simulate the noisy environments as it is trained with the clean and noisy data. Therefore dismissing the need for a denoiser as the robustness will be incorporated in the deep learning model. As mentioned in the previous chapter, the keyword spotting method used here is the phoneme search based. Indeed, this approach presents the best combination of excellent keyword flexibility with the ability to frequently change the list of keywords as desired with ease. Moreover, it ensures real-time response and satisfactory accuracy which is most important when it comes to any classification problem. Finally, the training data for the phoneme search based KWS is widely available.

Figure 3.1 illustrates the different modules of a KWS engine based on phoneme search. Figure 3.1 also depicts the data flow and what format the data takes going from one module to another [54]. The first module being the MFCC pre-processing, takes in the raw speech signal generating features that represent the most important information of the speech data. The second module is the most important as it decodes the speech features into phonemes producing the input for the next module. Finally, the keyword mapper module generates the list of detected keywords if they exist in the input signal [68].

The scope of this thesis only focuses on optimizing the phoneme decoder, in terms of small memory foot-print and faster response time while maintaining good robustness and accuracy.

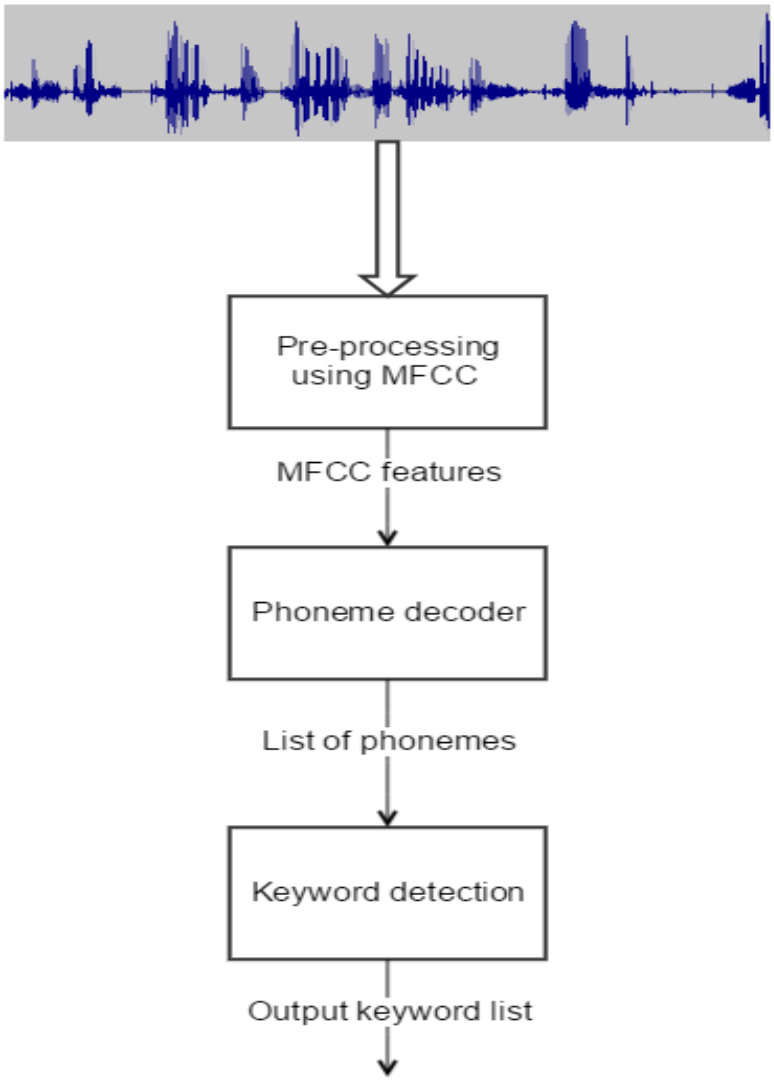

Figure 3.1: Phoneme search based keyword spotting engine architecture

Thanks to the recent successes of deep learning, in particular DNN, applied to various domains such as speech recognition [27], DNN are chosen to be the deep learning approach with which to implement the phoneme decoding task of the overall KWS process. As noted in the previous chapter, deep learning performs a pre-training step followed by a network fine-tuning step based on the standard backpropagation algorithm. The pre-training itself has been used in two different ways in the literature. The first method is to generate a new set of features using either RBM or SDA from the input training data. The new features are considered more abstract and are more representative than the original features. However, for speech recognition, MFCC is considered the standard representation and is used by most of the speech recognition engines in the market [28].
In this thesis we use the second method of deep learning which consists of using the pre-training stage to initialize the weights of the neural networks in preparation for the next fine-tuning process [33].

3.2.1 Customization of DNN for phoneme classification

The first stage of designing a DNN is the most important part in the training process. Therefore, choosing the correct pre-training algorithm would greatly influence the trained network accuracy. For speech recognition, the two main methods used in the literature are RBM and SDA which were already detailed in the previous chapter [28, 74].
To ensure that the best technique is used, both methods have been tested, the results are reported in the next chapter.

Speed and efficiency are two prominent features of DNN. However, some researchers do not prefer to use them as their success is dependent on a large number of uncertain factors, namely the training database and the empirical tuning of numerous parameters. The following section sheds some light on several parameters of the DNN and their importance to overcome common neural network problems.

How to overcome overfitting

One of the major shortcomings of neural networks is overfitting. The latter happens when a statistical model, neural networks in our case, interprets noise or random error instead of the underlying relationship. Overfitting is common when a model is extremely complex, such as having disproportionate parameters relative to the number of observations [61]. Training neural networks to model a robust phoneme recognizer is quite complex and requires the use of multiple layers with a substantial number of neurons for the different

layers. This represents a typical case where overfitting is very likely to happen. Hence, some precautions are taken to avoid overfitting [61, 66].

Database size As per the definition of overfitting, the major cause is having a small database and a large number of neurons and layers. To overcome this problem, larger datasets need to be used. In the scope of this work, the WSJ database is selected. In fact, the WSJ is composed of over 82,700 audio files with a sampling frequency of 100 samples *per* second. The databse offers around 60 million data points. This contributes to avoiding overfitting, at least not out of data points scarcity.

Early Stopping In this technique, the WSJ database is divided into three sets. First is the training subset, which is used to compute the gradient and update the network biases and weights. The second subset is called validation used for monitoring purposes. The error on the validation set is observed during the training stage. Normally, the validation error decreases during the initial phase of training, as does the training error. Nevertheless, as soon as the network starts overfitting the data, the error of the validation subset typically starts rising. After a certain number of iterations, validation error rises and the training is stopped, conserving the reached values of the weights and biases. Hence the name of this technique [80].

Adaptive learning rate As seen in the previous section, the validation error is typically used to sense overfitting and stop the training accordingly. Another use of the validation error to avoid overfitting is for it to be monitored for a different purpose. In this case to adapt the learning rate. In fact, if the validation error is almost constant, that usually means that the network is stuck in a local minima and continuing the training with a constant validation error contributes to overfitting. This technique updates the learning rate when the validation error is almost constant, allowing the network to break free from the local minima and move to a better local, or even the global, minima. Thus, this technique also helps avoiding overfitting [27].

Regularization An alternative technique to improve generalization is regularization. For this technique to operate, the performance function, which is usually the mean of sum of squares of the network errors on the training set, needs to change. Equation 3.2 describes the function usually used to train feed forward neural networks [30].

$$F = mse = \frac{1}{N}\sum_{i=1}^{N}(e_i)^2 = \frac{1}{N}\sum_{i=1}^{N}(t_i - \alpha_i)^2 \tag{3.2}$$

Where N is the number of neurons, e_i is the error, t_i is the label and α_i is the network output. To improve the generalization and avoid overfitting, it is needed to change the performance function by adding a term that represents the mean of the sum of squares of the network weights and biases $msereg = \theta mse + (1-\theta)msw$, where θ is the performance ratio, and msw is as follows

$$msw = \frac{1}{N}\sum_{j=1}^{n} w_j^2 \tag{3.3}$$

Where w_j designates the weights of the network. Using the modified version of the performance function allows the network to have smaller biases and weights, thus forcing a smoother response of the network making it less likely to overfit.

Determining the best value for the newly introduced performance ratio parameter is quite difficult. A larger than necessary ratio could result in overfitting. A ratio too small causes the network to overfit the training data.

It is preferable to automatically regulate the optimal regularization parameters. Bayesian framework processed by David MacKay [44] is one mechanism to reach automatic regulation. In this technique, the biases and weights of the network are random variables with certain distributions. The regularization parameters are linked to the variances affiliated with these distributions. Statistical techniques can help with the values of the parameters.

dropout This technique randomly drops units along with their connections from the neural network architecture during the training phase. Thus preventing the units from excessive co-adaption. The word dropout indicates dropping out visible and hidden units of a neural network. One unit drop means briefly removing it from the network. All incoming and outgoing connections of the unit are also removed as shown in Figure 3.2. The choice of the units to be dropped is random. In fact, every "thinned" network should perform well in the absence of the removed neurons. This requires each neuron to be more independent from the other neurons. Given that each neuron receives input from a random combination of neurons from a lower level layer, it gets noisier. In this regard, the dropout method introduces random noise to the training data. Dropout decreases the learning capacity of the deep neural network, hence leading to the generalization of the

model. Since a dropped out neurons is equivalent to having its activation fixed to 0, the dropout operation is the only change applied to the training algorithm [65]. While training, units dropout form a large number of distinct thinned networks. While validating, the use of one unthinned network with small weights approximates the effect of averaging the predictions of the thinned networks. This substantially reduces overfitting [66]. Dropout was successfully used on phoneme recognition with the TIMIT database in [30].

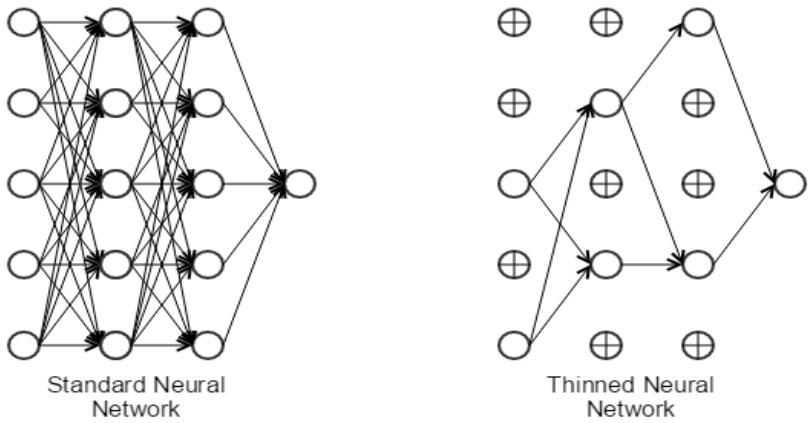

Figure 3.2: Dropout neural net model. Left: NN with 2 hidden layers. Right: A thinned NN after applying dropout to the network on the left [66].

Context padding

The acoustic signal is originally in the time domain, but going through the MFCC feature extraction process, more specifically through DFT, transforms the data into the frequency domain. This means that the data looses the temporal relationship between frames. To compensate for this loss, context padding is used to add more information about the acoustic signal [36]. Context padding consists on taking the frame in question alongside a specific number of frames from the right hand-side and from the left hand-side [36]. The purpose of this process is to take into consideration the dynamic nature of speech signal as identifying a phoneme often depends on more than the spectral features at a specific time. Indeed, it also depends on how features change over time. Applying context padding in speech recognition is performed by concatenating the MFCC features of the current frame f_t with those of the p prior and p posterior frames as seen in Figure 3.3. The input vector

V is a combination of context dependent frames of the speech utterance as shown in 3.4.

$$V_t = [f_{t-p}, ..., f_{t-1}, f_t, f_{t+1}, ..., f_{t+p}] \tag{3.4}$$

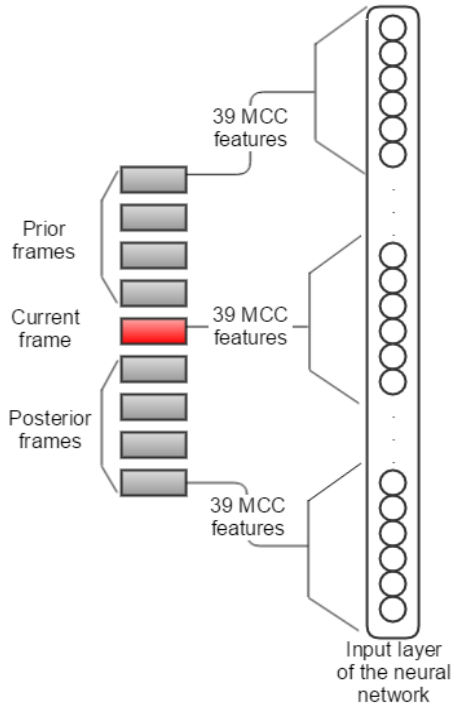

Figure 3.3: Frames context padding

Context dependent speech recognition has proved to be a breakthrough in speech recognition using deep neural networks as reported in [16, 72]. Context padding can be performed manually by creating a new set of data points that include the neighbouring frames features. A more convenient way would be on performing the context padding on the fly which is available is some libraries.

In the next section, we outline the essence of common approaches for adding noise and for denoising to an acoustic signal. This is known as artificial noise contamination and denoising, respectively.

3.2.2 Noise contamination

Training and testing the phoneme recognizer using noisy speech signal, in the absence of a noisy database, requires adding artificial noise to the clean database files. This involves selecting a noise sample file which can be of either types, stationary or non-stationary. Noise contamination is based on computing SNR of noise vs. speech. The energy calculation of a speech utterance requires adding up the sum of squares of all samples:

$$E_{Speech} = \sum_{i=0}^{N} s(i)^2 \tag{3.5}$$

$s(i)$ represents the speech utterances vector and E_{Speech} is the energy of the speech signal. Noisy samples can be generated or a dedicated noise sample file can also be used. The noise file could contain noises like babble, fan or any type of noises simulating real-life environment noises. Then the noise sample is truncated at the correct length. While using a noise sample file, it is critical to randomise the start position for every file to not have the same exact noise (of someone shouting or a door slam for example) at the same position in all the contaminated database files. In fact, not using random start position would have negative implications during the training stage of the classifier as it may confuse it with a pattern and therefore not contaminating the speech properly [43]. Next, the noise energy is computed as follows:

$$E_{Noise} = \sum_{i=0}^{N} n(i)^2 \tag{3.6}$$

$n(i)$ represents the noise vector used to calculate the noise energy E_{Noise}. The following equation details the calculation of the SNR between the speech utterance and the noise:

$$SNR = 10log_{10}(\frac{E_{Speech}}{E_{Noise}}) \tag{3.7}$$

Lacking a noise sample and only having a contaminated speech sample, the noise can be determined as follows:

$$n(i) = x(i)s(i) \tag{3.8}$$

Given $x(i)$ as the contaminated signal. Next, To reach the proper SNR, the noise is scaled by amplifying or reducing the amplitude, then added to the clean speech utterance. Assuming 5dB is the targeted SNR, the following formula is used:

$$K = \sqrt{\frac{E_{Speech}}{10^{5dB/10}}} \tag{3.9}$$

Where K is the scale factor. The last step would be creating the noisy signal using the noise sample $\hat{n} = Kn(i)$. Contaminated signal is then computed as follows [43]:

$$x(i) = s(i) + \hat{n}(i) \tag{3.10}$$

The next subsection discusses the denoising approach.

3.2.3 Denoising

In order to compare the use of a denoiser on the noisy speech input signal with the method implemented in this thesis, the performance of the DNN trained with clean data and tested with features of denoised speech have to be measured. Therefore, an efficient denoiser has to be selected. Noise could be stationary or non-stationary and for each type, a different method is used. Minimum Mean-Square Error (MMSE) estimation is based on minimizing Bayesian squared-error cost function [21]. It is known that mean-squared based distortion error of the log-spectra is a better match for phoneme recognition processing. Therefore, in [21] an algorithm based on log(MMSE) is employed, making the denoising process very efficient in handling stationary noise but not as efficient with non-stationary noises.

For non-stationary noisy speech data, a different approach [22] is used. In [22], for every time frame and every frequency range, the MMSE measure of noise power recursively updates the noise variance estimation. This estimation uses a spectral gain function determined by an iterative data driven training method. This method proves more efficient in non-stationary noise contaminated data.

In the following sections, the different experiments that are used to asses this work are detailed.

3.3 Test Scenarios

In this section, the different parameters of the proposed techniques in the phoneme classification process are discussed. Starting from the pre-processing features extraction and ending with the classification task, the reasoning behind the selected scenarios is presented

3.3.1 Feature extraction

The pre-processing stage is important as it is the first step in the phoneme recognition process, thus any poor choice in the parameters will propagate and will greatly influence

the final classification results. In addition, to fairly compare the proposed approach using DNN to the HMM/GMM based phoneme recognizer, similar feature extraction parameters have to be selected. Therefore, no changes are made to tune the pre-processing stage parameters. Instead, the optimal values that produce the best phoneme classification results for the hybrid HMM/GMM based recognizer are selected. The following are the different parameters used for feature extraction:

Number of MFCC features Most phoneme and speech recognition systems use 13 MFCC features as they best represent the speech information in the frequency domain. As detailed in chapter 1, to add temporal information to the features, the differential Δ and the acceleration $\Delta\Delta$ coefficients of the selected 13 MFCC features are added to the main features.

Sampling frequency Being the main units of a word, phonemes have shorter time length, hence the need to use a high sampling frequency. In this case, one frame is recorded every 10 ms of the speech utterance capturing enough data to represent short speech units such as phonemes.

Hamming Window length To reduce the spectral leakage of the signal while applying Fourier Transform, a Hamming window is used to add smoothness to the input signal. A 25 ms T hamming window is commonly used for speech recognition systems. For the scope of this work, the same frequency is selected.

3.3.2 Phoneme classification

Phonemes classification probability output is also the input to the keyword recognition algorithm, making the phoneme recognition process accuracy crucial to the final keyword detection result. Given the multitude and sensitivity of parameters to be tuned in the neural networks structure, cross validation is performed for each of the test scenarios. The following describes the different selected parameters for the DNN phoneme classifier, in both training and testing stages.

Context padding As previously explained, context padding is important to regain the lost temporal aspect of the speech data. For word classification in acoustic keyword spotting, the number of past and future features to be added are usually different. This is due

to the fact that some sounds or words usually come after or before the keyword in question. This present work focuses on smaller units of speech, namely phoneme. Hence the probability of phonemes or sounds neighbouring a specific phoneme should be the same on both sides. Therefore, the selected context padding features are the same for left and right. To decrease the data preparation cost in the testing stage, the number of neighbouring feature is kept small. In this case, the scenarios of 3, 4 and 5 features are tested.

Cross validation N-fold cross validation technique is most used to validate the results, especially in machine learning systems. Since the selected WSJ database contains almost 60 million data points, then using a 2-fold validation should be sufficient to validate the results.

Number of layers Given that the ultimate target of the keyword spotting engine is an embedded system, then the memory size of the model is critical and has to be kept to a minimum. The smaller the model, the more embedded systems it can fit. In addition, the classifier uses deep neural networks and not neural networks, which implies the use of more than one or two hidden layers. Hence the selected number of layers are 3, 4 and 5. The results are reported in the next chapter.

Number of Neurons For the same reasons listed in the previous paragraph, namely the need for a small model, the number of neurons per layer is also kept as small as possible. Since the size of the input layer is around 351 due to adding the 4 context frames on both sides, more or less 80, and the first layer is usually composed of more neurons than the feature vector size, then the selected number of neurons for the first hidden layer are 450, 500, 550 and 600 respectively. For the rest of the layers and in an attempt to minimize the size of the model, hence the number of neurons, the latter is decremented by 100 then incremented or decremented by 50 neurons to cover a fair range of possibilities. To tune the number of neurons for one layer X, the number of neurons in the rest of the layers is kept unchanged and the layer X neurons are varied according to the above detailed experimental scheme.

Learning rate Adaptive learning rate is one of the recently implemented methods to avoid overfitting as previously noted. A small learning rate would cause the network an excessive number of epochs to reach a minima in the gradient descent process. However a large learning rate would cause the error rate to fluctuate missing the minima, hence preventing a quick convergence. Therefore, we choose the starting value of the learning

rate as 0.08. The selected rate constantly decrements the training and validation error with no fluctuation. And since the used learning rate is adaptive, here is how it evolves: if the validation error reduction between two consecutive epochs is less than 0.001, the learning rate is scaled by 0.02 during each of the remaining epochs. Training terminates when the validation error reduction between two consecutive epochs falls below 0.0001. The minimum number of epoch is selected to be 20, after which scaling can be performed. The number of epochs in the pre-raining stage are selected to be 8, 10 and 12 as Restricted Boltzmann Machine and Stacked Autoencoders do not require many epochs to converge.

Dropout Proving to be one of the most effective methods to avoid overfitting, dropout technique is employed. Each of the layers has its own dropout factor, meaning the dropout does not have to be applied to all the layers. However, to boost the performance, all the layers should alternate a fraction of their neurons without training. Each layer can have a different dropout factor, but for this particular work, only one value is selected for all layers. The tested values are taken as 0.2, 0.3 and 0.4. The results are reported in the next chapter.

Data distribution As discussed in the cross validation section, 2-fold validation is selected for this work. Therefore, the MFCC features database is divided into 2 sets taking turn in being the training then the testing data. Since the databse is around 60 million data points, each of the two sets is composed of around 30 million frames. Each frame is represented by 39 MFCC features.

Having detailed the different phoneme recognition scenarios using clean data performed in this work, the next section describes the measures taken to enhance the noisy signal and make the phoneme recognizer more robust.

3.3.3 Speech enhancement process

Two major data preparation steps are needed to train the robust model and test its efficiency: noise contamination and denoising.

Noise contamination

Since the used database only contains clean speech and the chosen method to implement robustness include the noisy data in the training of the DNN, noise contamination is

performed. Given that the theory behind noise contamination is already detailed in the previous chapter. This section details the different contamination scenarios.
Since this step is about simulating a real-life environment, each noise contamination is tested using -10, 10 and 20 SNR values. We then select only the one that seems more realistic.
Noise can be either stationary or non-stationary, hence, more than one noise contamination has to be performed.

For stationary noise, the "Fan" noise sample is selected as it is a commonly present noise in our environment. Humans might not be aware of as our minds learned to ignore it due to its stationary nature and its continuous presence.

For non-stationary noise, "Babble" is what seems to be the best example that is always present and even the human ear cannot seem to ignore it due to the non-stationary and high frequency nature of such noise. However, "Babble" noise only simulates the chatter but not other background noises that are usually present in the environment. Therefore, the "Restaurant" noise sample is selected to act as the non-stationary noise in our experiments.

Finally, our real-life environments do not usually contain one specific noise, but rather a combination of them. For the scope of this work, a combination of the "Fan" and "Restaurant" noises is used to simulate this type of background noise.

Denoising

As previously explained, standard denoisers can focus either on stationary or non-stationary noise, but not both of them at the same time. However, smart denoisers simulate simultaneous denoising by periodically estimating the noise type and applying the corresponding denoising technique. In the scope of this thesis, a smart denoiser is used to ensure DNN noisy training is fairly compared to the state-of-the-art denoising technology.

3.4 Tools

This section provides a concise description on the various computational and algorithmic tools used in this work.

3.4.1 Hidden Markov toolkit (HTK)

Hidden Markov Model Toolkit (HTK) is a toolkit for building and manipulating hidden Markov models. HTK is primarily used for speech recognition research and is adopted by many speech recognition projects worldwide. HTK consists of a set of library modules and tools available in C source form. The tools provide sophisticated capabilities for speech analysis, HMM training, testing and results analysis. HTK is used during the pre-processing step of the KWS process and more specifically for the MFCC extraction step and the forced alignment that is needed to prepare the training and testing data for the DNN. HTK is also used for the training and testing of the GMM baseline approach adopted in this thesis [15].

3.4.2 Python Deep Neural Networks (PDNN) library

PDNN is a Python deep learning toolkit developed under the Theano environment. PDNN implements a complete set of models. Unsupervised learning (SDA, RBM), supervised learning (DNN, CNN) and multi-task learning can be conducted within a single framework. PDNN offers the simple use of DNN with all the required parameter settings, such as number of layers and neurons, context padding, dropout and adaptive learning rate. This library also offers the use of the Graphical Processing Unit (GPU) which is necessary given the size of the database which is composed of nearly 60 million data points. The use of the GPU helps to substantially reduce the required time needed to train the DNN. Since many tests are required in this work, GPU support is a necessity for the chosen DNN library. Hence, the PDNN library is selected [50].

3.4.3 Denoiser and noise adder

Based on the theory behind noise adding previously presented in the first chapter, a custom Matlab implementation is used as a noise adder for data contamination. An open source Matlab toolbox is also used for the smart denoising of the contaminated data.

3.4.4 General Purpose Graphical Processing Unit (GPGPU)

Training deep neural networks requires a great deal of computation and the use of a GPU has become a necessity for training deep networks. In our case, we use the Tesla-C2075 NVIDIA GPU card with 6 GBytes of embedded memory. The Tesla has 448 CUDA core,

delivering up to 515 Gigaflops of double-precision peak performance in each GPU. It offer levels L1 and L2 cache, supports the CUDA programming language and is connected through a high speed PCIe generation 2.0 interface. The use of such a powerful GPU makes it possible to train DNN models in a reasonable time frame.

3.5 Experiments Summary

Figure 3.4 summarizes the different experiments conducted in the scope of this thesis. Each of the experiments is characterized by a different colour. The data flows are detailed as follows:

Clean model with clean data The blue arrows show the different steps taken to train and test the clean data. Indeed, the clean data goes through the pre-processing module generating the correspondent MFCC features which are then split into two main subsets: the training subset that is used to train the DNN as previously described and the testing subset that is used to test the DNN module and outputting the final phoneme classification results.

Noisy model with noisy data The green arrows depict the flow of the data generating a noisy model and testing a different noisy subset. It is to be noted that noisy data, in both the training and testing subsets, are a combination of clean, fan contaminated, restaurant contaminated and fan then noisy contaminated.

Clean model with noisy data For comparison purposes, the clean data is also tested using the noisy model to determine how much accuracy is lost when including the noisy data into the training of the noisy model. The purple colour is given to this data flow.

Noisy model with noisy data The green arrows are assigned to describe what steps the data has taken to train and test the noisy model. The adopted path shows the use of the contamination module followed by pe-processing step to generate the training and testing data sets that are used to generate and test the noisy model.

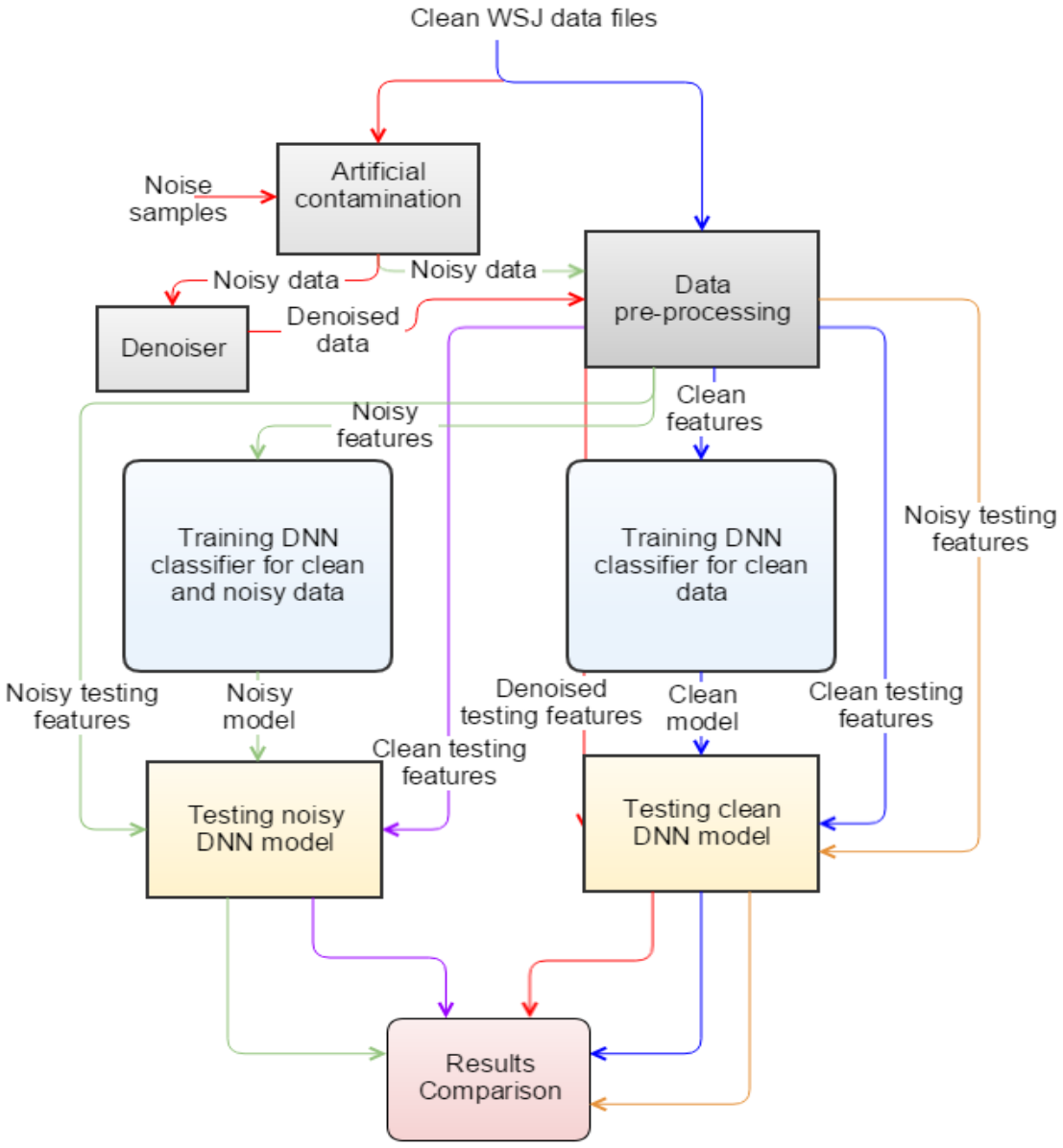

Figure 3.4: Data flow diagram

Clean model with noisy data To prove the necessity of using the noisy model rather than the denoiser, the clean model is also tested with the raw noisy data. The orange

colour in this case shows that the noisy testing features are used to test the performance of a clean trained DNN model.

Clean model with denoised data This is one of the main tests that are necessary to show that a noisy trained model is better than a clean model with a denoiser to interpret noisy speech. The red colour is chosen to describe the path taken by the testing data. Obviously, the use of the contamination module is necessary to corrupt the data, then it passes through the denoiser followed by the MFCC extractor which generates the features that are tested with the clean model.

Chapter 4

Experimental Results And Discussion

In this chapter, the different experiments leading to the tuning of the deep neural network and their results are presented. After the network has been tuned, the scenarios described in the previous chapter are performed and their corresponding results are reported. Finally, a discussion is conducted to evaluate the effectiveness of the proposed approach in this thesis.

4.1 Deep Neural Network Tuning

One of the drawbacks of DNN is the difficulty of selecting the network parameters, making the network tuning one of the major steps in any machine learning application that uses connectionist modeling. This section highlights the multiple steps taken to tune the neural networks. These experiments are very helpful in directing us forward. The final network parameters which are used to perform the test scenarios detailed in the previous chapter.

As has been mentioned, the following are conducted using a 2-Fold cross validation to generalize the results and draw conclusive decisions. Only clean WSJ data is used for the purpose of tuning the neural networks.

4.1.1 Layer configuration

To choose the most efficient number of layers the network should adopt, 3, 4 and 5 layers are tested and their output compared. The number of neurons per each layer is kept

constant to 600 with RBM as the pre-training technique. The number of epochs in the first training stage is 12 with an adaptive learning rate that also specifies the number of epochs of the second training stage.

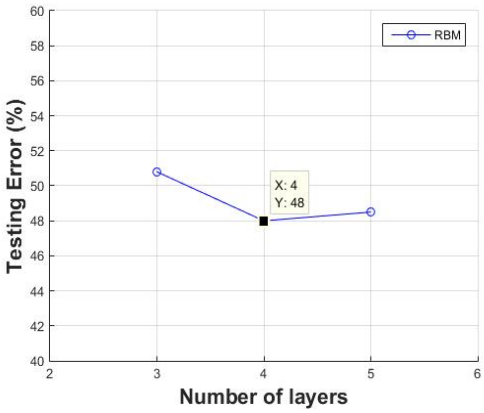

Figure 4.1: Comparison of WER (%) for different layer configurations

As can be seen in Figure 4.1, "4" is the number of layers that produces the lowest testing error rate. Therefore, for the rest of the experiments, all networks will have 4 hidden layers.

4.1.2 Neurons configuration

After selecting the best configuration regarding the number of layers, the number of neurons per layer has to be chosen based on the scenarios described in the previous chapter.
To tune the number of neurons of the first layer, the neurons of the other layers is kept unchanged and the neurons of the layer in question are varied. Once the best number of neurons is selected, the neurons are kept unchanged for the rest of the experiments.

First layer The tested number of neurons for the first layer are 450 to 550 with an increment of 50.

Figure 4.2 depicts the different results of varying the number of neurons on the first layer. It is clear that the best number of neurons for the first layer is 500 as it reaches an error of 46.7% whereas the other configurations vary between 47.1% and 48%.
Therefore, for the rest of the tests, the first layer his composed of 500 neurons.

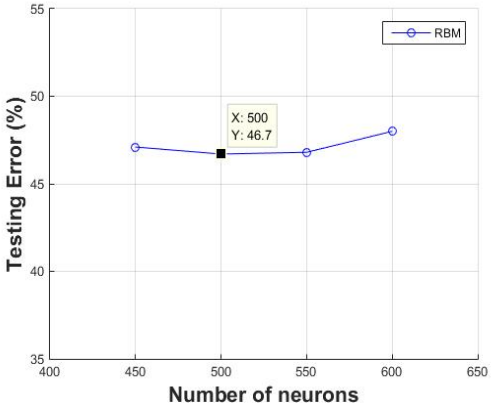

Figure 4.2: Comparison of WER (%) for different neurons configurations of layer 1

Second layer The number of neurons for the second layer is varied from 350 to 500 with an increment of 50.

Figure 4.3 shows two numbers of neurons that provide similar results, the first at 350 and the second at 450 neurons. The classifier's response time and the corresponding footprint are among the metrics used to measure the performance of the KWS that would use the tuned DNN as a decoder. Therefore, the selected number of neurons for the second layer is 350 as it offers almost the same accuracy performance while reducing the memory footprint and computational cost when compared with 450 neurons.

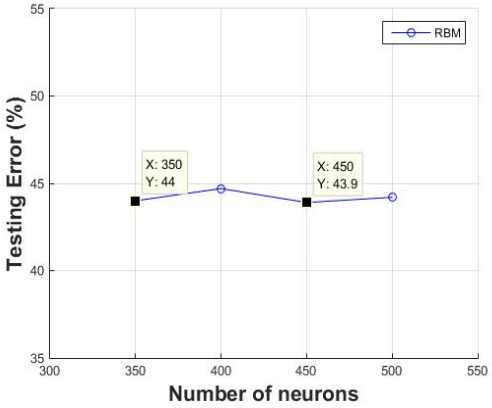

Figure 4.3: Comparison of WER (%) for different neurons configurations of layer 2

Third layer The number of neurons used to test the response of the third layer is varied from 100 to 250 with an increment of 50. Looking at Figure 4.4, it can be concluded that 200 neurons for the third layer offers the highest accuracy in the testing phase which is 43.2%. The rest of the configurations offer higher testing errors between 43.6% and 47%. Thus, for the remaining of the experiments, the third layer will be composed of 200 neurons.

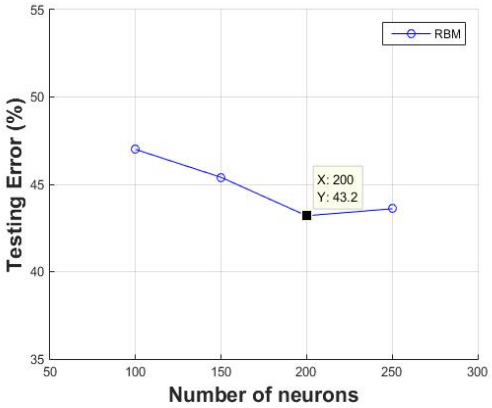

Figure 4.4: Comparison of WER (%) for different neurons configurations of layer 3

Fourth layer To tune the last hidden layer of the network, 50, 100, 150 and 200 are the number of neurons tested and the reported. Figure 4.5 illustrates the overall behaviour of the network. It can be concluded that 100 neurons results in the best performance, leading to the selection of 100 as the adopted number of neurons for the fourth hidden layer.

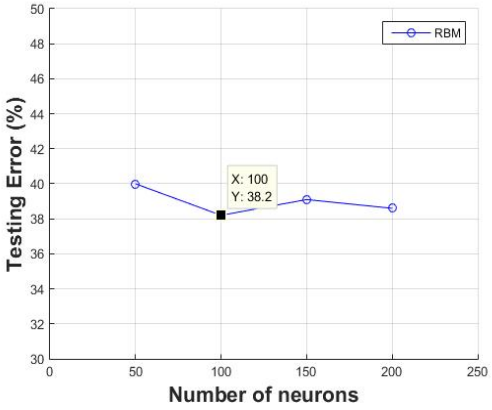

Figure 4.5: Comparison of WER (%) for different neurons configurations of layer 4

4.1.3 Pre-training approach

As described in the previous chapter, both stacked autoencoders and restricted Boltzmann machines are tested to select the best approach for the pre-training stage. The experiment is combined with testing verious number of epochs and selecting the one with the least error rate.

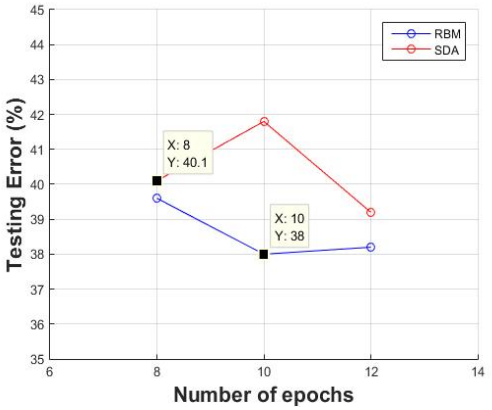

Figure 4.6: Comparison of WER (%) for RBM and SDA

Examining Figure 4.6, it is clear that for all the tested epoch numbers, RBM outper-

forms SDA in this case. Therefore RBM is selected as the pre-training technique to be used in the coming scenarios. Among the different epoch numbers selected to test the unsupervised learning stage, 10 seems to result in the lowest testing error. Hence, 10 epochs is chosen for the rest of the experiments.

4.1.4 Dropout selection

It is already established in the third chapter that the best dropout is one of the best techniques to avoid overfitting [66]. Therefore, it is tested with 0.2, 0.3 and 0.4 as the dropout rates for the chosen network structure.

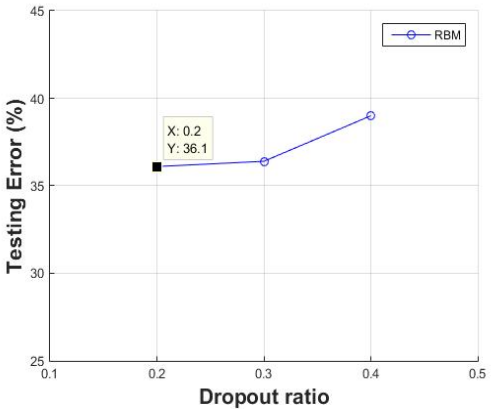

Figure 4.7: Comparison of WER (%) for different dropout ratios

Figure 4.7 proves that the dropout technique enhances the performance of the network. In this case, the dropout ratio 0.2 seems to outperform the other ratios reaching an error rate of 36.1%. Thus, 0.2 of the network neurons are to be dropped out at each layer while training.

4.1.5 Context padding selection

As outlined in the previous chapter, the use of context padding promises a great improvement in the testing error. In this experiment, 3, 4 and 5 context padding on each side are tested and their results are analysed.

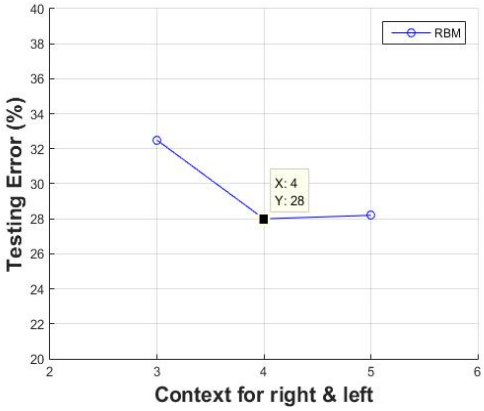

Figure 4.8: Comparison of WER (%) for different context padding frames

In Figure 4.8, "4" seems to be the best combination of frames to be added to the current frame. The total number of frames involved in a single frame classification is now "9". Given that each frame is represented by 39 features, the total number of features in one data point comes to a total of "351" features per frame. Adding "4" frames to the equation dropped the testing error from 36.1% to 28% which is the best result for all the different parameters combinations that have been tested.

4.1.6 Network final configuration

The previous results suggest that the best network configuration to be used for the noise handling scenarios is composed of 4 layers. The neurons distribution throughout the layers is: 351 for the input layer, 500 for the first layer,350 for second layer, 200 for the third layer, 100 for the pre-last layer and finally 40 for the output layer as there are 40 phonemes to classify. The dropout is set to 20%. The number of epochs for the pre-training is set to 10 with RBM as the preferred algorithm for the unsupervised learning. The number of epochs for the backpropagation is automatically set through the adaptive learning rate mechanism detailed in the previous chapter. The number of frames on the right and left hand-side to be included in the data point is set to "4".

4.2 Test Scenarios Results

At this point the network is tuned and configured to be used for the test scenarios detailed in the previous chapter. This section presents the different scenarios results followed by their interpretation. A conclusion is drawn on the effectiveness of the adopted technique. To determine the efficiency of the DNN, it has to be compared to the state-of-the-art phoneme classifier, namely HMM/GMM [15] based. In the scope of this thesis, the reference model is a mono-phone 3-state HMM for 40 phonemes with the probability distribution on each state being 5 mixture GMM. For the following tests, the use of noisy data is considered. As previously mentioned, noisy data are a combination of clean (25%), "Fan" (25%), "Restaurant" (25%) and Fan convoluted with "Restaurant" contaminated data. The DNN is tested with both, context dependent (CD) and context independent (CI) for fair comparison with GMM and to see the impact of context padding on the testing error of the DNN.

4.2.1 Clean data modeling

The first test sets used only the clean data of the WSJ database for model training.

Clean Data Testing This test is performed to compare the efficiency of the DNN based model with the GMM based model to determine if the DNN is more accurate than GMM for phoneme classification under clean data for training and testing.

Modeling approach	Training data	Testing data	Testing error
GMM	Clean	Clean	41%
DNN (CI)	Clean	Clean	34%
DNN (CD)	Clean	Clean	28%

Table 4.1: Comparison of WER (%) for GMM and DNN Clean data

Table 4.1 shows the superiority of the DNN approach over the GMM with a substantial improvement of 7% in the testing error. Using context dependent input to the DNN with "4" frames on each side further improves the accuracy by 4%. This proves that DNN is better than GMM in the phoneme classification of clean data. This has already been established in recent literature as detailed in chapter 2 [60].

Noisy Data Testing The following test is performed as a reference to prove that a clean model performs poorly when tested with noisy data. Table 4.2 depicts the high Word Error

Modeling approach	Training data	Testing data	Testing error
GMM	Clean	Noisy	65%
DNN	Clean	Noisy	62%

Table 4.2: Comparison of WER (%) for GMM and DNN Noisy data

Rate (WER) rate when using both approaches with noisy data on clean models. This is an expected outcome. Still the DNN model performs slightly better. Therefore, the approach adopted in this thesis to overcome the noisy environment condition is detailed in the following section.

4.2.2 Noisy data modeling

In this section noisy data is used to model GMM and DNN models and a conclusion is drawn from comparing their respective results.

Noisy Data Testing The purpose of training noisy models is to improve the performance of the phoneme decoder when tested under noisy condition, thus the noisy DNN model. The latter model is then compared to the GMM model that is also trained with noisy data and the best approach under both clean and noisy environment is selected to carry the phoneme classification for the KWS process.

Modeling approach	Training data	Testing data	Testing error
GMM	Noisy	Noisy	61%
DNN (CI)	Noisy	Noisy	54%
DNN (CD)	Noisy	Noisy	43%

Table 4.3: Comparison of WER (%) for GMM and DNN Noisy data

Figure 4.3 clearly shows the poor performance of the GMM modeled with noisy data as it only decreased the testing error by 4% when tested with noisy data. On the other hand, DNN efficiently modeled the noisy training data, as the testing error decreased by 19% for the context independent bringing the classification error to 43% which is only 15% away from the clean model with clean testing data. The Context dependent DNN also decreased the testing error by 8% which still outperforms the GMM model by 7%

Denoised Data Testing The previous tests proved that using a DNN model with noisy training outperforms the GMM counterpart with the same training data. But it could be argued that the use of a denoiser could replace the DNN noisy training. Thus, this test is performed to monitor the behaviour of the clean model with denoised testing data.

Modeling approach	Training data	Testing data	Testing error
DNN (CI)	Clean	Denoised	58%
DNN (CD)	Clean	Denoised	54%
DNN (CI)	Noisy	Noisy	54%
DNN (CD)	Noisy	Noisy	43%

Table 4.4: Comparison of WER (%) for DNN denoised data

The illustrated results in table 4.4 prove that using a denoiser prior to feeding the testing data to the clean DNN model improved the performance by 4%. Whereas the DNN noisy trained model gives a testing error of 54% which is substantially better considering that there is no use of a denoiser. Performing the context padding improved accuracy by an extra 4% when using denoised data. Context dependent noisy DNN training outperforms the denoised clean model DNN by a considerable 11%. Removing the denoiser from the system and reaching a better accuracy for noisy data is one of the contributions of this thesis.
Better performance under noisy condition only implies that the system will have two models one for noisy and the other for clean data.

Context dependent input features reduced the testing error considerably in the various above tests. Therefore, the rest of the DNN experiments are performed using context padding with "4" frames on each side of the current frame.

Clean Data testing To prove that the noisy trained DNN noisy model could be the sole replacement of multiple models without the need for a denoiser, which is usually expensive to deploy in real world applications, testing the noisy model in clean conditions and comparing it to GMM is required.

Table 4.5 depicts the degradation of the DNN noisy model when compared to the DNN clean model using clean testing data. The degradation is about 10% but the noisy model still outperforms the clean model in noisy conditions by 19%.

This is considered as a good compromise when it comes to the overall behaviour of the classifier. However, a better compromise could be reached by increasing the clean data component currently at 25% in the noisy data set.

Modeling approach	Training data	Testing data	Testing error
DNN	Clean	Clean	28%
DNN	Noisy	Clean	38%

Table 4.5: Comparison of WER (%) for DNN clean data

4.2.3 Increased clean data ratio

Increasing the clean data ratio in the mixed training error from 25% to 50% may bring an increase of the testing error under noisy conditions while it also promises an increase of the performance when tested with clean data. In fact, the model is less familiar with the noisy data and more familiar with clean data as the proportions shift toward a half and half ratio.

Modeling approach	Training data	Testing data	Testing error
DNN (25% Clean)	Noisy	Noisy	43%
DNN (50% Clean)	Noisy	Noisy	46%
GMM (25% Clean)	Noisy	Clean	45%
DNN (25% Clean)	Noisy	Clean	38%
DNN (50% Clean)	Noisy	Clean	31%

Table 4.6: Comparison of WER (%) for DNN Clean data ratio

Table 4.6 depicts the expected degradation of the noisy DNN model with the noisy testing data from 43% to 46%. However, the testing error only decreased from 38% to 31%. The same test is performed using GMM generating a testing error of 45% which is quite higher than the DNN.
This confirms the superiority of the DNN over GMM when testing the noisy model with clean data. This presents a better compromise for both condition.

4.2.4 Speed and memory consumption

DNN are more accurate classifiers than GMM (once trained) but at the same time DNN is very slow to train compared to GMM. The size of the data, the deep nature of the used neural networks that add exponential complexity with each added layer and the significant number of epochs that the deep neural network requires to converge to a local minima

makes it very slow to train. Due to the nature of this study, several experiments are needed to obtain consistent and reliable results. Therefore, speeding up the training process may be helpful but is not crucial for this work. The dropout technique used in this work to avoid overfitting also affects the neural network training speed. Since a percentage of the neurons is dropped from every layer to favour the generalization of the network, their connections are also ignored during the weights calculations. This reduces the training time depending on the dropout value for each layer. In our case, we use a dropout of 0.2 which makes the training speed up around 20%. This speed up has also been reported in the literature [66].

DNN reach the top of their potential when they are deep (many layers) and having many neurons per layer [12]. Training them on a traditional Central Processing Unit (CPU) would require months. The high data transfer latency limits the multi-threading programming making it not suitable for this situation. Nonetheless, recent parallel neural networks for graphics cards GPUs have solved the training speed limitation of the DNN [11]. GPU designed code for classification should be up to two orders of magnitude faster than the CPU [67, 73]. In [85], the use of 4 GPUs made the DNN code about 4-6 times faster. Using the parallel programming DistModel platform with 3 GPUs achieved a 2.6 times speed up according to [51]. In our case, only one NVIDIA Tesla GPU was available and has been used to train the different models needed for this thesis. The actual speed up was not calculated as the CPU took a seemingly very long time to train the model, which makes the speed up seem very high, as the GPU only takes a dozen hours to train the model. The DNN training speed is not the only issue as the decoding of a DNN is also slower than the GMM decoding due to the high number of float multiplications required for a classification especially for deep networks. According to [38], using a GMM model on embedded systems is twice as fast as using a DNN for a speech recognition decoding task. However, there are a number of techniques that speed up the decoding of DNN to reach a speed equal to that of GMM. Theses techniques are the following: Using fixed point operations and frame skipping technique [38]. In this thesis, none of the mentioned techniques has been used, but they will be used once the network is tuned and tested on embedded systems as the decoding speed is also crucial in realt-time embedded applications such as keyword spotting.
Although the DNN decoding process is more complex than the GMM decoding, the memory footprint does not follow the same pattern. Indeed, according to [38], a DNN model with 1.48M parameters outperforms the GMM in accuracy, with a disk size of only 17% of the GMMs. This is considered a major advantage for deep neural networks as small memory consumption enable smaller embedded platform to be speech enabled. In our case, the GMM model is only twice as large as the deep learning model while the DNN is best in terms of accuracy.

4.2.5 Summary and discussion

The above results indicate that using DNN instead of GMM to train the phoneme classifier model substantially improves the performance of the decoder. Indeed, using clean data for both testing and training, DNN model outperformed the GMM based model by 13% in accuracy. Furthermore, using a mix of noisy and clean data to model the DNN classifier prove to be a better alternative than using a denoiser in noisy conditions. It also avoids the case where the clean data is mistakenly considered as noisy data, leading the denoiser to disturb the clean utterance and producing corrupt data that is wrongly classified.

Despite dropping the testing error by 19%, the introduction of noisy data in the training set increased the phoneme classification error rate by 10%. A 50% balance between the clean and the noisy data when training the DNN model actually presents better overall results. In fact, while the noisy tests error rate drops by 16% instead of 19%, the clean testing data only drops by 3%. In summary, adding the noise to the training set drops the clean classification accuracy by 3% while it increases the noisy data classification by 16% This means that the newly trained noisy model can replace both models(clean and noisy), hence using one single model instead of two and not having to classify the data as clean or noisy. This makes the recognition process faster and less greedy in terms of memory footprint. In fact, one model in stead of two models reduced the memory requirements and the processing time is decreased by not having to detect the noise level and later load the corresponding model.

Therefore, it is safe to say that the noisy trained DNN model outperforms GMM and the use of a denoiser under all circumstances. In addition, it can replace the use of two model and only focus on a single process using DNN to classify more than one type of noise. The DNN model size is half the size of the GMM model, whereas it is slower at its current state.

Chapter 5

Conclusion and Future Work

Using DNN mixed data training approach promised to enhance the system robustness to background noises without the need for a hardware or a software denoiser. This technique also allows us to replace the two model approach, one for noisy and the other for clean environment, by a single model with a small penalty in terms of accuracy but with large gain in terms of less model complexity. According to the reported results, the DNN model reached a phoneme classification error rate of 46% whereas the denoiser processed data reached large error rate of 54%. This proves the superiority of the proposed approach in terms of robustness over the use of a denoiser and over the use of other modeling approaches such as HMM/GMM that only reached a testing error rate of 61%. The same noisy trained model tested with clean data also offers an excellent error rate of 31% with only a small degradation of 3% when compared to the clean trained model (28%) without introducing any additional complexity compared to standard DNN classification process. Both results combined proves that the mixed data trained model is capable of processing both clean and noisy data without the need of a denoiser, a noise sensing tool or other models. These results are reached while reducing, not only the memory footprint as the DNN model is shown to be much smaller than the GMM model and removing the denoisr also saves the memory space it occupies, but also with a possibility of not increasing the processing time, which are the overall goal of this thesis. Indeed, the DNN model is just a few hundreds of kilobytes large which is smaller than the GMM model size as detailed in the previous chapter. The absence of the denoiser and the noise sensing tool means less memory and less processing time as only the DNN multiplications are needed to reach a phoneme classification with no need for a denoiser. However, the DNN decoding is slower than GMM decoding which will be rectified when introducing the mentioned speedup techniques for DNN in the previous chapter.

Therefore, the proposed approach is clearly an attractive alternative to the standard keyword spotting techniques available in the market. Testing more noises and reaching better compromises in terms of clean and noisy data proportions in the training set would be the focus of the next step in this research.
The following step involves developing a small and fast phoneme mapping algorithm that would be the module taking the phoneme classifier output and deciding if a keyword is spoken in the given speech utterance.
Finally, an end-to-end systems needs to be developed and tested on embedded platforms to report real-life experiments and results.

References

[1] Wall street journal-based continuous speech recognition (csr) corpus, 1994.

[2] Guy Alon. Key-word spottingthe base technology for speech analytics. *Natural Speech Communications*, 2005.

[3] Yoshua Bengio. Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127, 2009.

[4] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[5] Lukáš Burget, Jan Černockỳ, Michal Fapšo, Martin Karafiát, Pavel Matějka, Petr Schwarz, Pavel Smrž, and Igor Szöke. Indexing and search methods for spoken documents. In *Text, speech and dialogue*, pages 351–358. Springer, 2006.

[6] John Butzberger, Hy Murveit, Elizabeth Shriberg, and Patti Price. Spontaneous speech effects in large vocabulary speech recognition applications. In *Proceedings of the workshop on Speech and Natural Language*, pages 339–343. Association for Computational Linguistics, 1992.

[7] Peter S Cardillo, Mark Clements, and Michael S Miller. Phonetic searching vs. lvcsr: How to find what you really want in audio archives. *International Journal of Speech Technology*, 5(1):9–22, 2002.

[8] Miguel A Carreira-Perpinan and Geoffrey E Hinton. On contrastive divergence learning. In *Proceedings of the tenth international workshop on artificial intelligence and statistics*, pages 33–40. Citeseer, 2005.

[9] Guoguo Chen, Carlos Parada, and Georg Heigold. Small-footprint keyword spotting using deep neural networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, pages 4087–4091. IEEE, 2014.

[10] I-Fan Chen, Chongjia Ni, Boon Pang Lim, Nancy F Chen, and Chin-Hui Lee. A novel keyword+ lvcsr-filler based grammar network representation for spoken keyword search. In *Chinese Spoken Language Processing (ISCSLP), 2014 9th International Symposium on*, pages 192–196. IEEE, 2014.

[11] Dan Ciresan, Ueli Meier, and Jürgen Schmidhuber. Multi-column deep neural networks for image classification. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3642–3649. IEEE, 2012.

[12] Dan C Ciresan, Ueli Meier, Jonathan Masci, Luca Maria Gambardella, and Jürgen Schmidhuber. Flexible, high performance convolutional neural networks for image classification. In *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, volume 22, page 1237, 2011.

[13] Israel Cohen and Baruch Berdugo. Speech enhancement for non-stationary noise environments. *Signal processing*, 81(11):2403–2418, 2001.

[14] Israel Cohen and Sharon Gannot. Spectral enhancement methods. In *Springer Handbook of Speech Processing*, pages 873–902. Springer, 2008.

[15] CUED. The hidden markov model toolkit, 2015.

[16] George E Dahl, Dong Yu, Li Deng, and Alex Acero. Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *Audio, Speech, and Language Processing, IEEE Transactions on*, 20(1):30–42, 2012.

[17] Li Deng, Alex Acero, Mike Plumpe, and Xuedong Huang. Large-vocabulary speech recognition under adverse acoustic environments. In *INTERSPEECH*, pages 806–809, 2000.

[18] Satya Dharanipragada and Salim Roukos. A multistage algorithm for spotting new words in speech. *Speech and Audio Processing, IEEE Transactions on*, 10(8):542–550, 2002.

[19] Sean R Eddy. Hidden markov models. *Current opinion in structural biology*, 6(3):361–365, 1996.

[20] Yariv Ephraim and David Malah. Speech enhancement using a minimum-mean square error short-time spectral amplitude estimator. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 32(6):1109–1121, 1984.

[21] Yariv Ephraim and David Malah. Speech enhancement using a minimum mean-square error log-spectral amplitude estimator. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 33(2):443–445, 1985.

[22] Jan S Erkelens and Richard Heusdens. Tracking of nonstationary noise based on data-driven recursive noise power estimation. *Audio, Speech, and Language Processing, IEEE Transactions on*, 16(6):1112–1123, 2008.

[23] Jort F Gemmeke, Tuomas Virtanen, and Antti Hurmalainen. Exemplar-based speech enhancement and its application to noise-robust automatic speech recognition. In *International Workshop on Machine Listening in Multisource Environments*, pages 53–75, 2011.

[24] Michal Gishri, Vered Silber-Varod, and Ami Moyal. Lexicon design for transcription of spontaneous voice messages. In *LREC*, 2010.

[25] Bin Guo, Daqing Zhang, and Zhu Wang. Living with internet of things: The emergence of embedded intelligence. In *Internet of Things (iThings/CPSCom), 2011 International Conference on and 4th International Conference on Cyber, Physical and Social Computing*, pages 297–304. IEEE, 2011.

[26] Philip Harding. *Model-based speech enhancement*. PhD thesis, University of East Anglia, 2013.

[27] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *Signal Processing Magazine, IEEE*, 29(6):82–97, 2012.

[28] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[29] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, 2006.

[30] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[31] Md Afzal Hossan, Sheeraz Memon, Mark Gregory, et al. A novel approach for mfcc feature extraction. In *Signal Processing and Communication Systems (ICSPCS), 2010 4th International Conference on*, pages 1–5. IEEE, 2010.

[32] David A James and Steve J Young. A fast lattice-based approach to vocabulary independent wordspotting. In *Acoustics, Speech, and Signal Processing, 1994. ICASSP-94., 1994 IEEE International Conference on*, volume 1, pages I–377. IEEE, 1994.

[33] Derek Janni. Introduction to deep neural network, 2015.

[34] Jean-Claude Junqua and Jean-Paul Haton. *Robustness in automatic speech recognition: Fundamentals and applications*, volume 341. Springer Science & Business Media, 2012.

[35] Shigeru Katagiri. *Handbook of neural networks for speech processing*. Artech House, Inc., 2000.

[36] Dušan Krokavec. Context dependent phoneme recognition. In *Text, Speech and Dialogue*, pages 252–257. Springer, 1999.

[37] Lori Lamel and Jean-Luc Gauvain. High performance speaker-independent phone recognition using cdhmm. In *EUROSPEECH*, volume 93, pages 121–124, 1993.

[38] Xin Lei, Andrew Senior, Alexander Gruenstein, and Jeffrey Sorensen. Accurate and compact large vocabulary speech recognition on mobile devices. In *INTERSPEECH*, pages 662–665, 2013.

[39] Ding Liu, Paris Smaragdis, and Minje Kim. Experiments on deep learning for speech denoising. In *Proceedings of the annual conference of the International Speech Communication Association (INTERSPEECH)*, 2014.

[40] Philipos C Loizou. *Speech enhancement: theory and practice*. CRC press, 2013.

[41] Carla Lopes and Fernando Perdigão. Phone recognition on the timit database. *Speech Technologies/Book*, 1:285–302, 2011.

[42] Xugang Lu, Yu Tsao, Shigeki Matsuda, and Chiori Hori. Speech enhancement based on deep denoising autoencoder. In *INTERSPEECH*, pages 436–440, 2013.

[43] James Lyons. Adding noise of a certain snr to audio files, 2014.

[44] David JC MacKay. Bayesian interpolation. *Neural computation*, 4(3):415–447, 1992.

[45] Jonathan Mamou and Bhuvana Ramabhadran. Phonetic query expansion for spoken document retrieval. In *INTERSPEECH*, pages 2106–2109, 2008.

[46] Jonathan Mamou, Bhuvana Ramabhadran, and Olivier Siohan. Vocabulary independent spoken term detection. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 615–622. ACM, 2007.

[47] Austin Marshall. Artificial neural network for speech recognition. *2nd Annual Student Research Showcase*, 2005.

[48] Martin F McKinney and Jeroen Breebaart. Features for audio and music classification. In *ISMIR*, volume 3, pages 151–158, 2003.

[49] David Meyer. Introduction to autoencoders. 2015.

[50] Yajie Miao. Kaldi+ pdnn: building dnn-based asr systems with kaldi and pdnn. *arXiv preprint arXiv:1401.6984*, 2014.

[51] Yajie Miao, Hao Zhang, and Florian Metze. Distributed learning of multilingual dnn feature extractors using gpus. 2014.

[52] Pedro J Moreno. *Speech recognition in noisy environments*. PhD thesis, Carnegie Mellon University Pittsburgh, 1996.

[53] Petr Motlicek, Fabio Valente, and Igor Szoke. Improving acoustic based keyword spotting using lvcsr lattices. In *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*, pages 4413–4416. IEEE, 2012.

[54] Ami Moyal, Vered Aharonson, Ella Tetariy, and Michal Gishri. Keyword spotting methods. In *Phonetic Search Methods for Large Speech Databases*, pages 7–11. Springer, 2013.

[55] Ami Moyal, Vered Aharonson, Ella Tetariy, and Michal Gishri. *Phonetic search methods for large speech databases*. Springer Science & Business Media, 2013.

[56] Arun Narayanan and DeLiang Wang. Ideal ratio mask estimation using deep neural networks for robust speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 7092–7096. IEEE, 2013.

[57] Javier Ortega-García and Joaquín González-Rodríguez. Overview of speech enhancement techniques for automatic speaker recognition. In *Spoken Language, 1996. ICSLP 96. Proceedings., Fourth International Conference on*, volume 2, pages 929–932. IEEE, 1996.

[58] Jia Pan, Cong Liu, Zhiguo Wang, Yu Hu, and Hui Jiang. Investigation of deep neural networks (dnn) for large vocabulary continuous speech recognition: Why dnn surpasses gmms in acoustic modeling. In *Chinese Spoken Language Processing (ISCSLP), 2012 8th International Symposium on*, pages 301–305. IEEE, 2012.

[59] Lawrence Rabiner and Biing-Hwang Juang. Fundamentals of speech recognition. 1993.

[60] Sihem Romdhani. Implementation of dnn-hmm acoustic models for phoneme recognition. 2015.

[61] Christian Schittenkopf, Gustavo Deco, and Wilfried Brauer. Two strategies to avoid overfitting in feedforward networks. *Neural networks*, 10(3):505–516, 1997.

[62] Daniel Schneider. *Holistic vocabulary independent spoken term detection*. PhD thesis, Universitäts-und Landesbibliothek Bonn, 2012.

[63] Björn Schuller, Martin Wöllmer, Tobias Moosmayr, and Gerhard Rigoll. Recognition of noisy speech: A comparative survey of robust model architecture and feature enhancement. *EURASIP Journal on Audio, Speech, and Music Processing*, 2009:5, 2009.

[64] Frank Seide, Peng Yu, Chengyuan Ma, and Eric Chang. Vocabulary-independent search in spontaneous speech. In *Acoustics, Speech, and Signal Processing, 2004. Proceedings.(ICASSP'04). IEEE International Conference on*, volume 1, pages I–253. IEEE, 2004.

[65] Michael L Seltzer, Dong Yu, and Yongqiang Wang. An investigation of deep neural networks for noise robust speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 7398–7402. IEEE, 2013.

[66] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[67] Daniel Strigl, Klaus Kofler, and Stefan Podlipnig. Performance and scalability of gpu-based convolutional neural networks. In *Parallel, Distributed and Network-Based Processing (PDP), 2010 18th Euromicro International Conference on*, pages 317–324. IEEE, 2010.

[68] Igor Szöke, Petr Schwarz, Pavel Matejka, Lukás Burget, Martin Karafiát, Michal Fapso, and Jan Cernockỳ. Comparison of keyword spotting approaches for informal continuous speech. In *Interspeech*, pages 633–636, 2005.

[69] Shinichi Tamura. An analysis of a noise reduction neural network. In *Acoustics, Speech, and Signal Processing, 1989. ICASSP-89., 1989 International Conference on*, pages 2001–2004. IEEE, 1989.

[70] Kishan Thambiratnam and Sridha Sridharan. Dynamic match phone-lattice searches for very fast and accurate unrestricted vocabulary keyword spotting. In *ICASSP (1)*, pages 465–468, 2005.

[71] Edmondo Trentin and Marco Gori. A survey of hybrid ann/hmm models for automatic speech recognition. *Neurocomputing*, 37(1):91–126, 2001.

[72] Michael Trost. Context-dependent deep neural networks: Breakthrough of neural networks for speech recognition, 2015.

[73] Rafael Uetz and Sven Behnke. Large-scale object recognition with cuda-accelerated hierarchical neural networks. In *Intelligent Computing and Intelligent Systems, 2009. ICIS 2009. IEEE International Conference on*, volume 1, pages 536–541. IEEE, 2009.

[74] Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *The Journal of Machine Learning Research*, 11:3371–3408, 2010.

[75] Tuomas Virtanen, Rita Singh, and Bhiksha Raj. *Techniques for noise robustness in automatic speech recognition*. John Wiley & Sons, 2012.

[76] Roy G Wallace, Robert J Vogt, and Sridha Sridharan. A phonetic search approach to the 2006 nist spoken term detection evaluation. 2007.

[77] Mitchel Weintraub. Lvcsr log-likelihood ratio scoring for keyword spotting. In *Acoustics, Speech, and Signal Processing, 1995. ICASSP-95., 1995 International Conference on*, volume 1, pages 297–300. IEEE, 1995.

[78] XIAO Xiong. *Robust speech features and acoustic models for speech recognition*. PhD thesis, 2009.

[79] Yong Xu, Jun Du, Li-Rong Dai, and Chin-Hui Lee. An experimental study on speech enhancement based on deep neural networks. *Signal Processing Letters, IEEE*, 21(1):65–68, 2014.

[80] Keun-Hyeok Yang, Ashraf F Ashour, and Jin-Kyu Song. Shear capacity of reinforced concrete beams using neural network. *Int. J. Concrete Struct. Mater*, 1(1):63–73, 2007.

[81] Nima Yousefian and Philipos C Loizou. A dual-microphone speech enhancement algorithm based on the coherence function. *Audio, Speech, and Language Processing, IEEE Transactions on*, 20(2):599–609, 2012.

[82] Dong Yu, Frank Seide, Gang Li, and Li Deng. Exploiting sparseness in deep neural networks for large vocabulary speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*, pages 4409–4412. IEEE, 2012.

[83] Xiao-Lei Zhang and Ji Wu. Denoising deep neural networks based voice activity detection. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 853–857. IEEE, 2013.

[84] Fang Zheng, Guoliang Zhang, and Zhanjiang Song. Comparison of different implementations of mfcc. *Journal of Computer Science and Technology*, 16(6):582–589, 2001.

[85] Pan Zhou, Cong Liu, Qingfeng Liu, Lirong Dai, and Hui Jiang. A cluster-based multiple deep neural networks method for large vocabulary continuous speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 6650–6654. IEEE, 2013.