TAMPERE UNIVERSITY OF TECHNOLOGY

Emre Cakir

Multilabel Sound Event Classification with Neural Networks

Master of Science Thesis

Examiner: Tuomas Virtanen, Heikki Huttunen
Examiner and topic approved by the Faculty of Computing and Electrical Engineering
on 3 September 2014

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Signal Processing
CAKIR, EMRE: Multilabel Sound Event Classification with Neural Networks
Master of Science Thesis, 63 pages
October 2014
Major: Signal Processing
Examiner: Tuomas Virtanen, Heikki Huttunen
Keywords: Sound event detection, Audio classification, Neural Networks, Multilabel

There are multiple sound events simultaneously occuring in a real-life audio recording collected e.g. at a busy street in rush hour. The events may include traffic noise, sound of rain, people talking etc. The humans are amazingly good at distinguishing these individual events, but as of yet, there is not any machine that can detect these events with (even close to) human accuracy. Polyphonic nature of the environmental audio recordings makes it hard to detect single sound events when many events are overlapping. With the gigantic audio database and state-of-the-art machine learning methods of the digital age, this is bound to change.

In this thesis, we use frequency-domain features to represent the audio input and multilabel deep neural networks (DNN) to detect multiple, simultaneous sound events in a real-life recording. We extract frequency-domain features from these recordings in short time frames. DNNs are artificial neural networks (ANN) with two or more hidden layers and they are especially good at modeling highly nonlinear relations and finding intermediate representations between system input and output. This is exactly the case in real-life sound event detection. Every feature extract is used as a training example and we train the neural network with these examples.

For the evaluation of this work, we focus on the performance of different topologies of DNNs used in this task. There are a large number of hyper parameters that define the structure of a DNN, such as the number of neurons in a layer, the learning rate used during learning, number of the hidden layers etc. The effects of each of these parameters are investigated in detail. A detection accuracy of 66.5% is achieved, which outperforms the state-of-the-art method by a large margin.

PREFACE

I would like to thank my thesis supervisors, Dr. Eng. Heikki Huttunen and Dr. Tuomas Virtanen for showing me a brilliant research path and helping me in every step I take. They truly changed my perception on academic research.

I would like to thank each and every member of our Audio Research Team, especially Toni Heittola, Oguzhan Gencoglu and Tuomo Tuunanen. They always provided me great motivation, guidance and fun research atmosphere.

I know I can not explain how grateful thing it is to have a caring family. I would like to thank my parents, Unal and Fatma, and my sister, Bilge for encouraging me to set sail on this unknown world for me called Finland and follow my dreams.

I would like to thank my Turkish buddies in Finland. Being far away from home would be a hell of a trouble without you people. I would also like to thank my Finnish comrades who helped me to adapt, discover and love Finnish culture.

I would like to thank Department of Signal Processing for providing me a unique audio database and funding to continue my research.

EMRE ÇAKIR
Tampere, October 2014

CONTENTS

TERMS AND DEFINITIONS

MIR	Music Information Retrieval
ASR	Automatic Speech Recognition
SED	Sound Event Detection
ANN	Artificial Neural Network
MFCC	Mel-frequency Cepstral Coefficient
LPCC	Linear Predictive Cepstral Coefficient
LPFC	Log-frequency Power Coefficient
LSF	Local Spectrogram Features
GMM	Gaussian Mixture Model
HMM	Hidden Markov Model
MP	Matching Pursuit
NMF	Non-negative Matrix Factorization
KNN	K-Nearest Neighbor
SVM	Support Vector Machine
DNN	Deep Neural Network
EM	Expectation Maximization
UBM	Universal Background Model
ZCR	Zero-crossing Rate
DFT	Discrete Fourier Transform
DCT	Discrete Cosine Transform
LPC	Linear Prediction Coding
DBN	Deep Belief Network
ReLU	Rectified Linear Unit
FFNN	Feed-forward Neural Network

RBF	Radial Basis Function
RNN	Recurrent Neural Network
SOM	Self-organizing Map
KL	Kullback-Leibler
SGD	Stochastic Gradient Descent
BP	Backpropagation

1. INTRODUCTION

The last decades of humanity witnessed a fast change of daily lifestyle with the introduction of computers in our lives. When we first encountered with the computers, most of us thought that they are too complex, too bulky and they will not be great use for people except scientists. With the help of the rapid developments in electronics, computers got smaller and easier to use. Consequently, they started to become a major factor in our day-to-day lives.

The more people get used to the computers, the more they started asking from them. The unnatural bond between humans and an electronic device shrinked swiftly in recent years. The people of the 21^{st} century are comfortable with carrying electronic gadgets with them anywhere they go to. Nowadays, we expect computers to help us understand the world around us and be aware of what is going on. We want electronic devices that can percept the physical context and assist their users in certain tasks, without any guidance. For this reason, context-aware devices have been a popular research area among the machine learning scientists. Some of the applications such as navigation assistance already take part in our daily lives, but they provide only limited context-awareness.

In fact, designing context-aware devices is easier said than done. Human beings always combine visual and audio information for perception of surroundings, but this is not still the case for machine learning systems. Nowadays, the vast majority of these devices are solely based on processing visual information. Although it is possible to obtain precise, accurate details of the surroundings from the image data, physical challenges exist. For the applications such as camera based indoor robot navigation [1], even a little loss of sight would cause serious problems.

On the other hand, audio information can provide additional awareness to these systems. One of the senses that we use most while interacting with the world is hearing. However, computers still fail badly, compared to humans, when it comes to interpreting and assigning meanings to the sounds. With the increase of available audio data and processing power, a new path opened for research and improvement for the scientists. Consequently, it led to dawn of a new branch of computer science, namely audio information retrieval.

Audio information retrieval is a very popular research area in recent years and it has many sub-branches, such as music information retrieval (MIR), automatic

speech recognition (ASR) and sound event detection (SED). Some of the MIR applications can be listed as musical instrument recognition [2; 3], automatic music transcription [4; 5] and musical genre classification [6; 7]. In [8], ASR is defined as the independent, computer-driven transcription of spoken language into readable text in real time. The research on ASR is mainly focused on improving the vocabulary size, noise robustness, speaker independence and continuous speech recognition [9; 10; 11]. SED systems take part in many applications such as audio based multimedia surveillance [12], audio keywords for video [13] and event detection in real-life recordings [14].

In this thesis, we tackle the area of real-life sound event detection in multi-label recordings by using artificial neural networks (ANN). The interpretation of the concept of sound event are various in the literature. As in [15], sound events may result from the ambient sounds in nature, such as wind on trees, sound of rain or insects chirping. In [16], sound events are interpreted as all the events in audio which are not based solely on music or speech signals. In our work, we use the second approach and deal with ambient nature sounds among music and speech signals all together.

Getting good results from a recognition system is fundamentally based on the selection of relevant features and the suitable classifier. In other areas of audio information retrieval other than multi-label SED, there are some traditional features and methods that have been used in ASR for a long time. These include e.g. mel-frequency cepstral coefficients (MFCC) [17], linear predictive cepstral coefficients (LPCC) [18] and log-frequency power coefficients (LPFC) [19] as features and Gaussian Mixture Models (GMM) and Hidden Markov Models (HMM) for data modeling. Real-life sound recordings typically have multiple sound events occuring simultaneously and it is hard to discriminate these individual sound events from the mixture signal. Therefore, this research area was not able to develop as fast as its other counterparts such as speech recognition, musical genre classification etc. The main problem in the recognition of real-life sound events is that traditional features and classification methods do not give as satisfactory results as other applications. We believe the reason is that these features and methods are not able to efficiently reflect the different temporal and spectral characteristics of the events occuring at the same time. For example, the MFCCs extracted from a mixture signal with overlapping sound events are not equal to the sum of the MFCCs of individual sound events.

Our motivation to tackle this problem comes from the yet unsatisfactory results in this intriguing topic. With the selection of correct features and by using the recent improvements in the machine learning such as deep neural networks, we believe that high accuracy event detection is possible in real-life recordings.

The thesis is organized as follows. Section 2 presents the theoretical background on audio features used in SED and artificial neural networks. Section 3 explains the methodology including preprocessing, feature extraction, neural network training and testing algorithms and a post-processing method. Section 4 reveals the evaluation details and results of several simulations. Finally, discussions on the results and suggestions for future research areas are pointed out in Section 5.

2. BACKGROUND

In order to understand the implemented system thouroughly, this section provides background information and literature review on topics such as sound event detection, audio features and artificial neural networks.

2.1 Sound Event Detection

The goal of SED systems is to recognize a sound event resulting from an individual sound source in a continuous audio signal. The main concern of the detection is to correctly recognize an event in a given extract of the recording. The exact location, *i.e.*, the start and end time of the event has a secondary importance, so the detection systems often have rather coarse time resolution. This is where SEDs differ from other audio based recognition systems such as speech and music recognition.

Previous SED research has typically been carried out for isolated sound events [20]. In each recording, there is a single sound event and the SED is trained to detect the event in a given time period. An SED system with a single output for each time instance is called monophonic SED. However in this thesis, SED is carried out in recordings collected from real-life environments. These environments have different acoustic characteristics due to the presence of human or nature activities. There are multiple overlapping sound events, therefore the detection of individual events is more challenging than in isolated recordings. An SED system that is able to detect multiple events in the same instnace is called a polyphonic SED system. Figure 2.1 illustrates the polyphonic nature of sound events in realistic environments.

SED systems first create a mid-level representation of the audio waveform in a recording. This process is called feature extraction. It is crucial to select the correct features to extract from an audio signal, because irrelevant features might cause a loss of valuable information and that would make the detection harder. Many events resulting from the nature, such as wind and rain sounds, have strong temporal domain signatures. Therefore, it is somehow necessary to extract the temporal domain information as well as spectral domain information in environmental sound recordings.

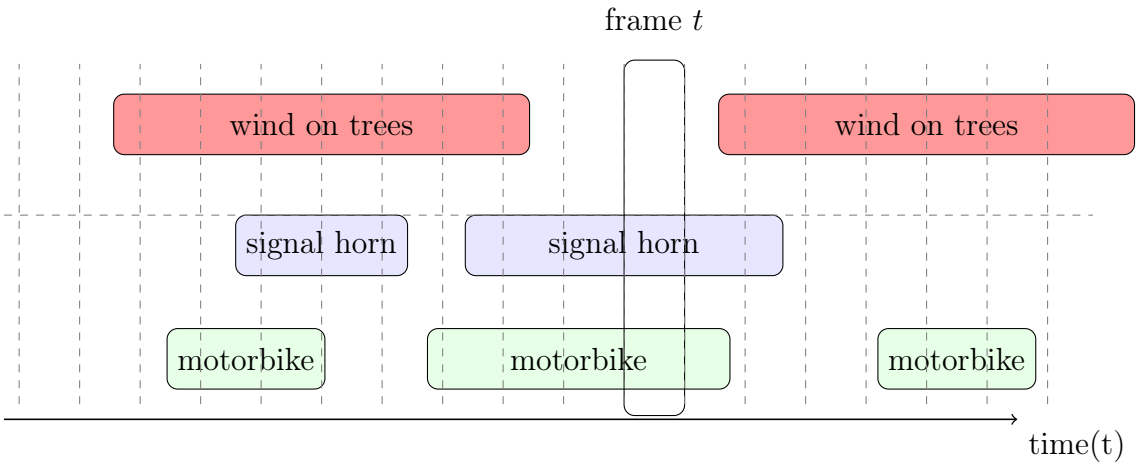

Figure 2.1: Overlapping sound events in a recording from a realistic environment. Frame t represents the short time frame from the recording where only signal horn and motorbike events are present.

2.2 Audio Features

Detection of audio events requires a good representation of input, as with most pattern recognition tasks. Using raw amplitude values of the audio signals for detection is a rare situation and most likely an unsuccessful method. Extracting noise robust, informative features from the sound recording can be seen as a good initialization and a first step forward.

Audio features for environmental sounds are especially important, when the sound is recorded in a real-life auditory scene. Multiple sound events with diverse characteristics are most likely occur simultaneously and some events might behave as noise for other events, making it harder for detection. Speech and music signals have harmonic structure, unlike environmental sounds, which are unstructured. Therefore, the traditional features that work succesfully in speech and music recognition systems might fail in environmental sound event detection. Several features from different domains have been experimented to characterize audio events. We briefly review common features used in SED, although not all of these are used in the implementation stage.

2.2.1 Time Domain Features

Time domain representation of a signal describes the changes in the signal with respect to time. In the case of audio signals, time-domain representations usually give little information, because audio signals are highly stochastic and additive noise can mask the individual sources in the sound. Some of the mostly used time domain features are as follows.

Zero-Crossing Rate Zero crossing rate (ZCR) is the rate of change of a signal from positive to negative. It can be seen as a simple measure of the frequency content of a signal and it is often calculated in short time windows. ZCR is calculated as

$$ZCR = \frac{1}{2N} \sum_{i=1}^{N} |\text{sgn}(u_i - u_{i-1})| \tag{2.1}$$

where u_i is the discrete time input, $i = [1, 2..., N]$, N is the number of samples from the input and $\text{sgn}(\cdot)$ is defined as

$$sgn(x) = \begin{cases} 1, & x > 0 \\ 0, & x = 0 \\ -1, & x < 0 \end{cases} \tag{2.2}$$

Short-time energy Short-time energy gives the information of amplitude variation over time in short frame and it is calculated as

$$E_k = \frac{1}{K} \sum_{i=1}^{N} |u_i w_{k-i}|^2 \tag{2.3}$$

where u is the discrete time input, k is the time index of the frame and w is the window of length N.

2.2.2 Frequency Domain Features

Frequency domain analysis illustrates how large portion of the input signal is found inside certain frequency bands over a range of frequencies. Time domain representation of a signal can be converted to frequency domain by using *transforms*. One of the most widely used transforms is called the *discrete Fourier transform* (DFT). The Fourier transform assumes stationary sinusoid signals in time domain. As for the audio signals, they are almost never stationary, always varying with time. Therefore, the DFT is taken in short time windows of length 10-100 ms by assuming the audio signal is stationary in that time interval. After taking the DFT of the window, there are several ways to proceed and extract frequency domain information.

Mel-band Energy The mel scale is a perceptual pitch scale that is designed so that the frequency distance between each band is perceptually equal for the listeners. The human auditory system is more sensitive to the frequency variations in low values than high values. Therefore, the frequency distances between consecutive mel bands in high frequencies is quite high compared to low frequency values. Frequency

f in Hertz is converted to m mel by using the formula

$$m = 2595 \log_{10}(1 + \frac{f}{700}) \tag{2.4}$$

where mel is the unit of pitch.

The mel-band energies of an audio signal are calculated as follows. After taking the DFT of each short time window, power spectrum is obtained by taking the square of the absolute value of the complex DFT. Then, mel-scale filter bank outputs are obtained by mapping the power spectrum onto the mel scale. Typically, filterbanks of 20 to 40 filters are applied to cover the whole frequency range. The process is illustrated in Figure 2.2.

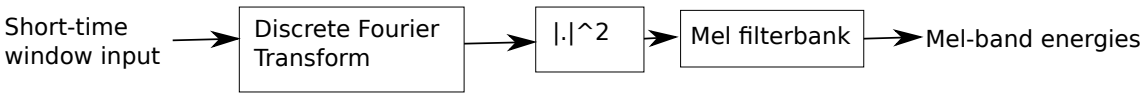

Figure 2.2: Mel-band energy extraction.

Mel-frequency Cepstral Coefficients Mel-frequency cepstral coefficients (MFCCs) are widely used features in speech, music and audio event recognition. They provide decorrelated features that work especially well in single-source environments.

MFCCs are calculated by taking the logarithm of mel-band energies and applying the discrete cosine transform (DCT). The motivation for taking the logarithm comes from human hearing. We do not perceive the loudness in the audio in a linear scale. When the amount of energy in an audio signal is doubled, the perceived loudness is not doubled. The logarithm operation makes the features more close to human perception. Log-filterbank energies are quite correlated, because the filterbanks are overlapping. To overcome this, the DCT is applied and typically the first 12 to 26 coefficients. This way, we keep the first coefficients that give the most information about the spectral shape. The first coefficient is discarded, because it basically gives the total log energy of the frame and does not give information about the spectral shape. The MFCC extraction process is illustrated in Figure 2.3.

Figure 2.3: MFCC extraction.

For sound signals it is beneficial to have the information on how the features change with respect to time. MFCCs alone give the power spectral envelope for a single frame. To model the dynamics of the MFCCs, *delta* (differential) coefficients are calculated as

$$d_t = \frac{\sum_{n=1}^{N} n(c_{t+n} - c_{t-n})}{2\sum_{n=1}^{N} n^2} \tag{2.5}$$

where d_t is the delta coefficient vector of length N for frame t and c_t is the MFCC vector for frame t. In some applications, delta-delta coefficients were found helpful as well.

Linear Predictive Cepstral Coefficients Linear Prediction Coding (LPC) is another common way to obtain spectral information. The signal u is predicted by a linear combination of its past values as

$$u_n \approx \sum_{i=1}^{p} a_i u_{n-i} \tag{2.6}$$

where a_i are the linear predictor coefficients and p is the LPC order. The coefficients are determined by minimizing the residual error energy using Levinson-Durbin recursion [21].

Linear predictor coefficients are often not used as features, but they are converted to linear predictor cepstral coefficients (LPCC). LPCCs are calculated recursively from LPCs. Unlike MFCCs, LPCCs do not relate with perceptual frequency scale.

Log-frequency Power Coefficients Log-frequency power coefficients (LFPC) are calculated in a similar fashion with Mel-band energies. The only difference is that the bandpass filters are not on the mel scale [19]. LFPCs characterize the distribution of the spectral energy across the sound input range of frequency.

2.2.3 Time-Frequency Domain Features

Conventional frame-based methods are sometimes inefficient for audio event detection, when the audio is a mixture of sounds from multiple sources. Studies [22] show that human brain does not possibly use the frame-based approach, but it rather recognizes the features that are local and uncoupled in the frequency domain. Motivated by this fact, local spectrogram features (LSF) can be extracted from the region around the *keypoints* in the spectrogram [23]. Keypoints are selected to be at the discriminative sparse peaks occuring in a spectrogram so that sound events can be modeled with LSF clusters and their occurences in the spectrogram.

2.3 Artificial Neural Networks

An artificial neural network (ANN) is a pattern recognition method that is inspired by how the human brain processes information. In literature, ANN is defined as "massively parallel interconnected networks of simple (usually adaptive) elements and their hierarchical organizations which are intended to interact with the objects of the real world in the same way as biological nervous systems do" [24; 25]. Human brain is believed to be composed of billions of interconnected neurons of many layers. These neurons are specialized on learning information. For this reason, humans are magnificently good at interpreting the world that they see. Handwriting recognition is a simple example on the task. Tuned by different handwritings through many years, an average human brain is capable of understanding the handwritings of different people easily. When it comes to implementing this with a computer program, a reasonable approach would be to model the letters and characters. However, it is easier said than done. The reason is that the variations in handwritings of different people makes it really hard to find accurate models. This causes to a significantly lower success rate for computers than humans.

ANNs approach this problem in a similar way with human brain. It designates each letter (or, in general, training instance) as input, sets up an input layer, an output layer and (optional) hidden layer(s) of artificial neurons and starts to tune the network with each training instance. A cost function is selected to determine the error between the desired output and the estimated output. The task of training is to minimize this cost function iteratively. An ANN prototype is given in Figure 2.4.

The invention of ANN approach in pattern recognition dates back to 1940's. This may sound surprising due to the fact that significant improvement and research in the area is done in recent years. Actually, the idea was present way before the advent of computers. In 1943, the first artificial neuron was produced by the neurophysiologist Warren McCulloch and the logician Walter Pits [26]. They modeled a simple neural network using electrical circuits. With the advance of computers, the first hypothetical neural network was developed in 1950's. Perceptron was created by Frank Rosenblatt in 1958 [27] and backpropagation algorithm was found by Paul Verbos in 1975 [28], both of which are explained below in detail.

The reality is ANN dropped out of favor during 1970's. Although initially producing good results and giving huge promises, the computational requirements for ANN learning were found to be unreasonable considering the computer technology of the time. Moreover, the training data required to train an ANN properly was usually not available at that time. Another problem was that the initial values of network parameters affected the convergence duration a lot and there was not a

rule of thumb on how to perform initialization. Consequently, the funding on the field was decreased, other pattern recognition methods came onto stage and ANN research slowed down.

In 2000's, the research interest in the area is restored due to improvements in computational technology and the invention of new learning algorithms. The poor initialization problem is tackled successfully by the introduction a learning algorithm for Deep Belief Networks (DBN) in 2006 [29]. DBN uses unsupervised pre-training to initialize the network parameters. Nowadays, ANNs with multiple layers are trained with vast amount of training data and state-of-the-art results are obtained in many pattern recognition tasks. With the new inventions such as the *rectified linear unit* [30] and *dropout* [31] that reach same accuracy in a purely supervised manner, the unsupervised approach has become obsolete and the new trend is towards supervised learning.

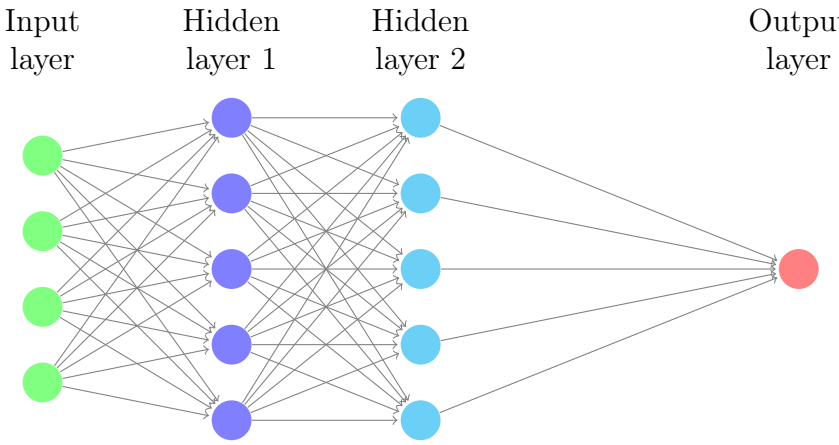

Figure 2.4: Symbolic representation of the ANN topology.

2.3.1 Neuron

Artificial neuron (interchangeably called as unit or node) is the basic element of the ANNs. Biological neurons transmit information (stimulus) between each other via the synapses. Depending on the amount of stimuli entering a neuron, it can either fire or not fire. ANNs try to model this phenomenon by weighing the inputs (stimuli) to an artificial neuron between the connections (synapses) of the neurons. Each neuron, except the ones in the input layer, receives inputs from previous layer of neurons and produces an output for the following layer of neurons. Each input is weighted by the corresponding weight parameter and resulting products are added up. Then, depending on a certain function, the artificial neuron produces an output. This function is called *activation function*. The artificial neurons mainly differ in their activation functions.

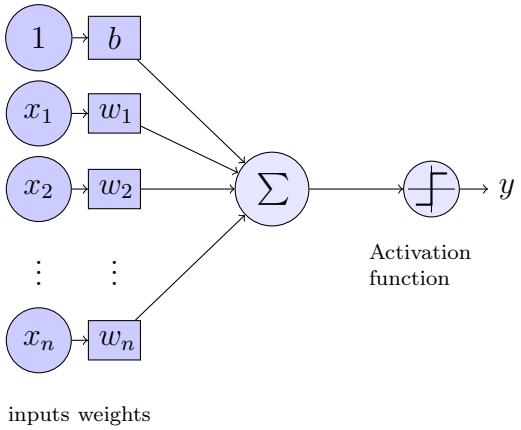

Figure 2.5: A single perceptron.

Perceptron Developed in 1958 by Frank Rosenblatt, perceptron used to be one of the most commonly used types of artificial neurons. A perceptron takes several inputs x_i, calculates the weighted sum of these inputs with weights w_i and gives an output y depending on a bias b.

Note that letters with bold characters (*e.g.* $\mathbf{x}$) represent vectors and the index i for x_i represents the i^{th} element of $\mathbf{x}$ vector. Capital, bold characters (*e.g.* $\mathbf{W}$) represent matrices. This notation will be used throughout the thesis.

The output y of a perceptron is calculated as

$$y(\mathbf{x}) = \begin{cases} 1, & \sum_{i=1}^{n} w_i x_i + b > 0 \\ 0, & \sum_{i=1}^{n} w_i x_i + b \leq 0 \end{cases} \tag{2.7}$$

where n is the number of inputs to the perceptron. Figure 2.5 illustrates the perceptron structure.

As explained, the perceptron uses a step function as the activation function. Although the step function is simple and intuitive, it may cause unstable behaviour. Suppose we want to update the weights so that a small change in the weight leads to a corresponding small change in the output, hopefully in the direction of the desired output. Unfortunately, this is not possible with the step function, because it only gives a discrete output. Therefore, updating the weights and biases during learning process is more effectively done with differentiable activation functions.

Sigmoid Neuron Sigmoid neuron is a very popular type of artificial neuron that uses sigmoid activation function σ. The output of the logistic sigmoid function is defined by

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{2.8}$$

When logistic sigmoid function is used to determine the output of the neuron, we end up with

$$\begin{aligned} y(\mathbf{x}) &= \frac{1}{1+e^{-z(\mathbf{x})}} \\ &= \frac{1}{1+\exp(\sum_{i=1}^{n} w_i x_i + b)} \end{aligned} \tag{2.9}$$

where $z(\mathbf{x}) = \sum_{i=1}^{n} w_i x_i + b$.

Sigmoid neuron output has the same upper and lower bound with perceptron output, namely 1 and 0. When z is very large and positive, $e^{-z} \to 0$ and $\sigma(z) \approx 1$. When z is very negative, $e^{-z} \to \infty$ and $\sigma(z) \approx 0$. The shape of the logistic sigmoid function is given in Figure 2.6.

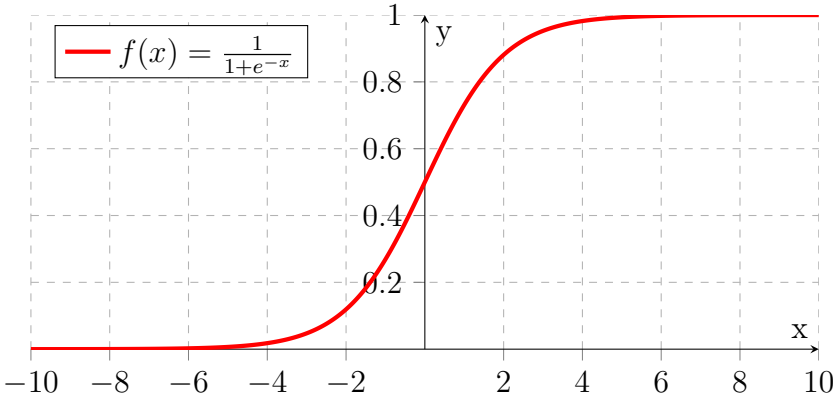

Figure 2.6: Sigmoid function.

As seen in Figure 2.6, sigmoid function is a smoothed version of a step function. This is certainly beneficial due to the fact that the function is continuously differentiable. It can be used to make small changes with the weights and biases to get small changes in the neuron output.

Rectified Linear Unit Rectified linear units (ReLU) have the activation function

$$f(z) = \max(0, z) \tag{2.10}$$

where z is the weighted sum of the inputs to the ReLU. This activation function has closer characteristics to biological neurons, because rectifier activation function allows truly sparse representations, which is also the case for biological neurons [32]. ReLU outputs are not bounded from above. A single unit with $\max(0, x)$ nonlinearity can be viewed as an approximation to a set of replicated binary units with tied

weights and shifted biases [30]. This can be mathematically expressed as

$$\sum_{i=1}^{\infty} \sigma(z - i + 0.5) \approx \log(1 + e^z). \tag{2.11}$$

The activation function $\log(1 + e^z)$ is called *softplus* and it is the smoothed version of the rectifier. The shapes of rectifier and softplus functions are presented in Figure 2.7.

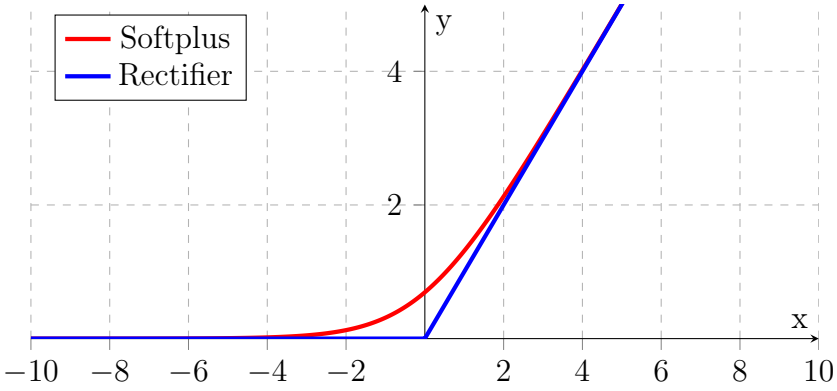

Figure 2.7: Rectifier and Softplus functions.

Maxout Neuron Maxout neurons are one of the newest type of neurons that can be seen as a modified version of the ReLUs [33]. Given an input vector $\mathbf{x}$ with length N, the output for maxout neuron h is given as

$$h(\mathbf{x}) = \max_{j \in [1,k]} z_j(\mathbf{x}) \tag{2.12}$$

where k is the number of affine feature maps (*i.e.* maxout pieces), $\mathbf{z}_j = \mathbf{x}^T \mathbf{w}_j + \mathbf{b}_j$ is the weighted sum of the inputs to h with weights w_j and bias b.

Unlike the other activation functions with 1-D weight vectors with length N for each neuron, the weight matrix $\mathbf{W}$ for the maxout neuron can be thought as having size $k \times N$. In the same fashion, the bias b for a maxout neuron is not a single value, but a vector with length k. Hidden units with maxout functions at each layer are divided into non-overlapping pieces and each piece generates a single activation via the max pooling operation. Finally, the largest weighted sum is given as activation output. Maxout activation function is simply equal to a ReLU function when a 0 is added to the the weighted sum set and the number of maxout pieces is set to 1. Maxout and ReLU are able to avoid the vanishing gradient problem thus allowing deeper NN topologies. The max pooling process is illustrated in Figure 2.8.

Maxout function can approximate an arbitrary convex function in a piecewise linear way [33]. It generates different activation functions for each neuron and it

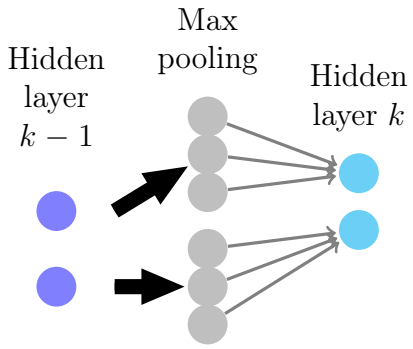

Figure 2.8: Maxout activation function with $N = 3$ feature mappings.

also learns the relationship between neurons in a single hidden layer. Unlike other activation functions, it is not bounded from above (*i.e.* no saturation problem) and it is locally linear at most points. Maxout networks usually work effectively without much pre-training, if any.

2.3.2 Layer

ANN layers are composed of artificial neurons grouped together. The neurons in each layer are connected with the neurons in the previous and next layer, *i.e.*, they get their inputs from the previous layer of neuron outputs and give their output to the next layer of neurons. Depending on their functionality, there are three different kind of neurons:

Input Layer Input layer is the first encounter of a training example $\mathbf{x}$ with the ANN. The neurons of the input layer is *passive*, i.e. they do not modify the data. Their sole function is to duplicate the value to their multiple outputs. In terms of weights, biases and activation function, the input layer can be seen as having $w = 0$, $b = x$ and $f(x) = x$ activation function.

Hidden Layer Hidden layers are the layers between input and output layer that are supposed to find different representations for the training examples. The term *hidden* just means "not an input or an output". Some ANNs have a single hidden layer and some have two or more hidden layers (also called the deep neural networks). The design of the hidden layers (number of neurons, activation functions etc.) take a huge part in the learning process, because they determine the complexity of the system. The usual way of determining the correct design is by doing grid search over hidden layer parameters.

Output Layer As it is evident from the name, output layer consists of neurons that give the posterior probabilities of the output of a training example in a detection

task. The number of neurons is equal to the number of possible outcomes of the example. Since the outputs are probabilities, they are in the range $[0, 1]$. For a detection system, these outputs can be thresholded to give binary outputs, i.e., "detected or not".

2.3.3 Learning Methods

The term *learning* for the ANN means that by processing training examples, the ANN output gets closer and closer to the desired output. This idea takes its roots from the human brain's learning, which uses the method of learning by example. The knowledge on how the brain learns is still in its infancy, but we do have a basic understanding of the process. It is believed that during the learning process, brain's neural structure is altered, increasing or decreasing the strength of its synaptic connections depending on their activity. ANN learning mimics this process by updating its weights and biases with each example. There are three different ways of learning used in ANNs: *supervised*, *semi-supervised* and *unsupervised*.

Supervised learning method is used when each training instance is associated with a set of labels. The network is provided with the desired output for the given input. For each instance it compares the desired output with estimated output and a cost value is determined. The network modifies the weights and biases according to a learning algorithm so that the cost value is minimized. The widely used learning algorithm for supervised ANNs, namely gradient descent algorithm, will be explained in detail in Section 3.3.1.

Semi-supervised learning systems often use a small amount of labeled data and a huge amount of unlabeled data [34]. It is particularly useful when the need for labels during the training is critical and the amount of testing data is huge.

In unsupervised learning, the labels for the training data are not available to the network. The ANN determines the cost solely from the estimated outputs. It tries to learn the similarities between the inputs and represent them in a more efficient way. Human visual system is believed to operate in such fashion. Although unsupervised learning is biologically more plausible, it is harder to find efficient cost functions and model the network in real-world problems. Besides, the number of possible outcomes is unknown and the performance is often poor when the training dataset is small. Unsupervised learning methods are widely used in clustering and mixture modeling.

2.3.4 Universal Approximation Theorem for ANNs

ANNs are universal approximators, *i.e.*, any continuous function can be approximately represented by ANNs with sufficient amount of complexity. The universal

approximation theorem for ANNs with a sigmoidal activation function was proved by Cybenko [35] and Hornik et al. [36]. Let I_N represent the N-dimensional unit cube which contains all possible input examples $\mathbf{x}$ where $x_i \in [0,1]$, i = 1,2,...N. Let $C(I_N)$ represent the space of continuous functions on I_N. Let $\sigma(\cdot)$ be a continuous, monotonically increasing, bounded function (e.g. logistic sigmoid). The finite weighted sum $\mathbf{y}$ is defined as

$$y_k(\mathbf{x}, \mathbf{w}) = \sum_i \alpha_{ki} \sigma(\sum_{j=1}^{N} w_{ij} x_j + b_j) \tag{2.13}$$

where α, b are real constants and $\mathbf{W}$ is the real-valued weight matrix. The weighted sums in the form $\mathbf{y}$ are dense in $C(I_N)$, meaning that given any $f \in C(I_N)$ and $\epsilon > 0$, there is a sum y that satisfies $|y(\mathbf{x}, \mathbf{w}) - f(\mathbf{x})| < \epsilon$ for all $\mathbf{x} \in I_N$. This shows that there always exists an ANN with a single hidden layer that can approximate an arbitrary, continuous, nonlinear, multidimensional function f with any rate of target accuracy.

2.3.5 Types of ANN

ANNs can be categorized into several groups depending on their topologies and learning algorithms. The type of ANN can significantly affect the performance in certain tasks. Due to the recent popularity of the topic, several types of ANN ideas have been developed and the most widely used types will be covered in this thesis.

Feed-Forward Neural Network Feed-forward neural networks (FFNNs) are one of the most common NN topologies. They consist of neurons arranged in layers, first layer taking the input and last layer producing the output. The layers in between do not produce any output to outside, therefore they are *hidden* layers. The information propagates forward from the input layer to the output layer via neuron connections.

FFNNs may have fully connected structure. In fully connected ANNs, each neuron in a layer is connected with each of the neurons in the previous and next layer. Non fully connected ANNs can be thought as having zero weights for certain connections. The symbolic representation in Figure 2.4 is an example of a fully connected feed-forward neural network.

Supervised learning is used in FFNNs. Each training example $\mathbf{x}$ is associated with a target $\mathbf{y}$ and the main goal is to minimize the error $\epsilon(\mathbf{y}, \hat{\mathbf{y}})$ between the target $\mathbf{y}$ and the FFNN estimate output $\hat{\mathbf{y}}$. FFNNs are widely used in classification tasks.

Radial Basis Function Network Radial basis function (RBF) networks use a similar approach with K-nearest neighbour classifiers [37]. Data centroids are

selected so that they sample the input domain suitably. Euclidian distance is calculated between the inputs and data centers. The distance value is given as input to a nonlinear function ϕ, the type of which is presumably not essential for the performance but usually selected as Gaussian [38]. The weights λ are updated with this output in the way that the further the neuron is from the data center, the less influence it has, so the corresponding connection weights are reduced. The output $f(x)$ is calculated as

$$f(x) = \lambda_0 + \sum_{i=1}^{N_c} \lambda_i \phi(||x - c_i||) \tag{2.14}$$

where $||\cdot||$ represents the Euclidian norm, c_i are the data centers and N_c is the number of data centers.

Recurrent Neural Network Recurrent neural networks (RNNs) allow their neurons to share their outputs with previous layer neurons, creating feedback cycles. This depicts that an RNN may sustain the temporal activations even in the absence of input [39]. Therefore, RNNs are dynamical systems with dynamical memory, while feed-forward networks are functions [40]. This brings an advantage to RNNs on modeling human brain activations. However, the complexity of RNN structures due to feedbacks make them less favourable in most cases over FFNNs. RNNs are mainly used in computational neuroscience, machine learning and nonlinear signal processing applications. An example of the RNN topology is illustrated in Figure 2.9.

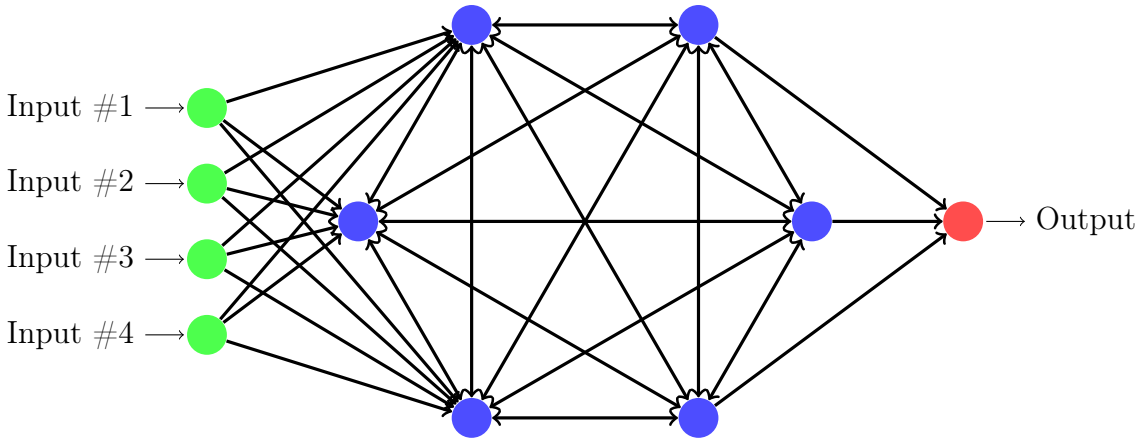

Figure 2.9: Recurrent neural network.

Self-Organizing Map Self-organizing map (SOM) [41] is a type of ANN that uses unsupervised learning to tune the neurons and weights. Also called *Kohonen map* after its inventor, SOMs are used to convert high-dimensional inputs to low-dimensional (mostly two-dimensional) discrete maps. They are particularly useful

in clustering applications.

SOMs are fully-connected feed-forward networks without any hidden layers. Depending on the inputs and weights, the *score* for each output neuron is calculated and the scoring function is often chosen as Euclidian distance

$$D(w_j, x_n) = \sqrt{\sum_{i=1}^{N}(w_{ij} - x_{ni})^2} \tag{2.15}$$

where $D(w, x)$ is the distance, w_{ij} is the weight for output neuron j, i=1,2,...N and x is the N-dimensional input vector. The output neuron with the best score (minimum distance) is designated as the *winning* neuron for that training instance. By taking the winning neuron as the center, output neighbourhood is determined according to distances and pre-selected neighbourhood size. Finally, weights of the neurons in the neighbourhood are updated by

$$\hat{w}_{ij} = w_{ij} + \eta(x_{ni} - w_{ij}) \tag{2.16}$$

where $\hat{w}_{ij}$ is the updated weight, n is the input neuron index and η is the *learning rate*. The learning rate and the neighborhood size are selected to be decreasing for convergence.

2.3.6 Advantages as a Classifier

Neural networks are distributed models. Connection weights and neuron biases produce the overall input-output response of the system. Each neuron makes up a basic part of highly complex structure. For a certain input, some neurons are switched on and some are off. Learning how to give responses to different stimuli enables ANNs to represent a nonlinear input-output mapping.

Neural networks can be trained to represent a wide range of pattern recognition tasks. Once the network is trained, the weighted connections extract the information from the raw input. Different combinations of connection weights give different responses for each input, making the neural network able to represent a diverse range of mapping problems.

Neural networks do not require any high level representations of the input. Due to their ability to represent complex input-output relations, there is no need to preprocess the ANN input with complex transformations. ANNs also do not require the input and output to have similar characteristics. To illustrate, the input of the ANN can be the energies in different frequency bands of an audio signal and the output can be given as different activity indicators of sound events.

2.3.7 Disadvantages as a Classifier

The number of neurons needed to approximate a function is not known beforehand and it mostly reduces to a trial-and-error procedure. Inadequate number of hidden neurons may cause insufficient complexity for representing the input-output relationships. On the other hand, using too many hidden neurons may cause the ANN to specifically work well on a certain training dataset and fail for generalization on other training examples. This famous phenomenon is called *overfitting*.

The number of hidden layers to be used in an ANN is not a known variable for the most cases. Intuitively, each layer of hidden neurons is expected to capture a different level of complexity in the input and multiple hidden layers might be needed for certain tasks. However, ANNs with multiple hidden layers tend to get stuck in local minima [42].

2.3.8 Deep Neural Networks

DNNs are ANNs with two or (most likely) more hidden layers [29; 43]. An ANN with a single hidden layer is the universal approximator. However, it sometimes does not perform sufficiently on some complex tasks, where the training data is scarce or the input simply does not have enough features. The aim of using multiple hidden layers is to find more abstract features in the higher levels, defined in terms of lower-level features [43]. These high-level, abstract features would help to separate the independent distributions in the training data.

Shallow network architectures, ANNs with one or two hidden layers, typically require a huge amount of neurons to find a good representation of their inputs. When the number of neurons is increased, the number of network parameters such as weights and biases consequently increase, causing a burdensome computational power requirement. Recent experiments show that a large amount of functions with many variables cannot be efficiently represented with a shallow architecture [44; 45].

Another inspiration to use deep architectures simply comes from the brain. Measurements have been made to find the distance and time a nerve signal travels in the body to transmit visual information. The results show that even for a basic visual object recognition, approximately 10 layers of biological neurons are involved. Simple tasks such as detection of edges in an object are associated with lower levels and recognition of fine details and sub-objects are related with higher levels.

Humans intuitively divide the problem into several levels of execution. Architects first design the low-level structure of the buildings and then insert the more complex elements into the plan. Programmers divide their code into basic functions, which are then merged to perform complex tasks. When someone starts learning to play an instrument, (s)he first tries to play some notes and basic sentences. Only after

long hours of training, (s)he can combine the knowledge and play some songs. These are some of the motivations behind the deep learning. Deep architectures represent concepts one at a level and in the end make a composition of lower level concepts to get the final representation.

The developments in computational technology paved the path for deep architectures. Training ANNs with multiple hidden layers was computationally burdensome due to the amount of ANN parameters involved. With the advent of parallel computing technologies, this burden is relatively eased. Very complex networks with 10-15 hidden layers still require a vast amount of computational power.

Despite the obvious benefits, designing deep architectures is easier said than done. The backpropagation algorithm works well for shallow architectures, but in deep architectures it may be problematic. Experiments showed that the training often gets stuck in a local minimum of cost function for deep networks, whereas better performance could be obtained if the global minimum was reached.

Initialization of the deep neural networks proved to have a crucial part in the performance of the deep networks. Typically, the network is initialized by assigning small random weights and biases for all neurons. Poor initialization increases the amount of training time drastically to converge to a satisfactory accuracy. The introduction of a new learning algorithm for deep belief nets (DBN) solved the initialization problem very efficiently [29]. The training method is as follows. First, the layers are trained greedily, one-by-one and in unsupervised manner. This leads to obtaining fairly good parameters in a fast manner and good initialization. DBNs can be used as stand-alone, unsupervised method or can be combined with supervised fine-tuning of the network parameters. DBNs were a huge breakthrough and obtained state-of-the-art results in several pattern recognition benchmarks. Thus, it once more attracted the attention of machine learning scientists into ANNs. DNNs were successfully used to obtain state-of-the-art results in classification tasks in image [46] and speech recognition [9]. With the advancements in parallel computing and inventions of new algorithms such as *maxout* [33] and *dropout* [31], unsupervised pre-training is even discarded and yet comparable results are achieved.

2.4 Previous work on Sound Event Detection

Since the dawn of pattern recognition and machine learning, environmental sound event recognition did not attract as much interest as its companions such as speech and music recognition. When the literature on the topic is examined, it needs hard work to find systems which use only audio data for classification. Due to the reasons such as polyphonic nature of environmental sounds, working on them apparently seemed challenging for researchers. The recognition rates were decreasing rapidly when the number of classes were increased to realistic values, making these systems impractical. Previous research on this area is related to areas such as audio-based context and scene recognition [47; 48] and audio-based events detection for sports video [49; 50].

Despite the scarcity of research in the area, several different methods have been proposed for audio representation and event detection in realistic environments. The traditional method for speech and music recognition has been using MFCCs as audio features and GMM-HMM for modeling these features. This has been implemented in environmental SED as well [14; 52], however the results were not satisfactory. Therefore, the search for new approaches in feature set and classification method has started. For preprocessing; methods such as wavelet transform [53], matching pursuit (MP) [15] and non-negative matrix factorization (NMF) [51] have been proposed. Time-domain features such as short-time zero crossing rate and short-time energy have been experimented in [15]. Other than GMMs, alternative classifiers such as K-nearest neighbor (KNN) [15], support vector machines (SVM) [53], Gaussian process [53] and Deep Neural Networks (DNN) [20; 57] were proposed in this particular task.

2.4.1 MFCC Based Methods

In [14], the task was to recognize and locate audio events in polyphonic long recordings collected from everyday environments. The recordings were annotated with 61 different audio event classes and MFCCs, delta-MFCCs and delta-delta-MFCCs were used as features. The experiments were conducted in both isolated events classification and polyphonic event detection. For the classification of isolated events; three-state, left-to-right fully connected HMMs were trained for each audio event class using the Expectation-Maximization (EM) algorithm. The likelihood for each observation sequence was found by Viterbi algorithm and the event corresponding to the HMM giving the largest likelihood was selected as output. Based on the simulations, a three-state left-to-right HMM with 4 to 16 Gaussians per state was a good choice with 52-54% recognition rate. For the event detection in continuous sequences, HMMs for each class were connected to a network HMM with equal tran-

sition probabilities for each model. The optimal sequence was decoded again using Viterbi algorithm, assuming that the system will output the most prominent event at a given segment of the polyphonic recording. The error rate for this system was calculated as 84%. This proposed method is not capable of detecting multiple sound events at a time, therefore it offers only limited usage in environments with large number of possible sound events.

Previously mentioned work was improved by Heittola et al. by doing sound source separation as a preprocessing step [54]. The recording was preprocessed using non-negative matrix factorization [51] and divided into four different tracks. The feature set and HMM modeling were implemented in a similar manner with [14] and the sound recordings were the same as well. SED was performed for each track with the same annotations, assuming that the tracks which do not contain the relevant classes will not show up as the output of the Viterbi algorithm. In order to prevent the large number of short events occuring as Viterbi output, a cost value was introduced between transitions. The accuracy inside 30 second blocks was calculated as 52%, almost twice of the baseline system. This work offers a great leap forward over its predecessor as it provides multi-label detection. However, the number of possible events at a time is fixed, which is an undesired restriction for event detection in realistic environments.

In [55], unsupervised sound source separation is used as a pre-processing stage to detect overlapping sound events. MFCCs have been used as audio features and a continuous density HMM with three state left to right topology have been used to model conditional feature distributions. In this work it is assumed that the target event is found in only one audio stream and the other streams contain overlapping sound events. The stream containing the target event is found by two different methods by using expectation maximization algorithm. First method is to find the most prominent stream and the second method is to eliminate possible streams one by one. Individual sound event models are trained with the obtained target stream. This work is selected as the baseline in this thesis for the following reasons. It is capable of doing polyphonic sound event detection on overlapping sound events, it uses the same sound database as ours for training and testing, it uses the same accuracy metric (F1 score) in one-second blocks. The highest overall accuracy obtained in this work is 44.9%.

A two-stage classification method for environmental sound events was proposed in [52]. This method again used MFCCs as features and GMM as classifiers, but it presented a novel idea for event classification. By following the idea that humans rule out unlikely events when they already know the context, the system first implemented context recognition. Then, based on the recognized context, a set of GMM models for event classes were used in detection stage. In this work, event detection

was done in two different approaches, one by determining the most prominent event at a time (monophonic) and the other by using multiple Viterbi passes. Apart from the event models, a universal background model (UBM) was also trained. In each pass, the decoded path was marked and the system was prevented from selecting the same event for the same time instance. Iteration stopped when all the time instances were marked with the event corresponding to UBM. The accuracy metric was f-measure and the accuracies were calculated block-wise: if an event was detected in any frame within the time block and the event actually occured at some point in that time block, it was regarded as correctly detected. The context recognition accuracies were 70.0% and 80.7% for 4-second and 20-second blocks. The SED accuracies only slightly increased when the context recognition stage was included. The authors claimed the reason for low increase might be due to error in context recognition causing the selection of wrong set of events in the detection stage.

DNN methods have been applied in various pattern recognition tasks and automatic sound event detection is one of them [58]. DBN with supervised fine-tuning was used for isolated acoustic event classification in [57]. Isolated sound events included environmental sounds such as birds singing, motorcycle, footsteps etc. After normalization and windowing, Mel-band energies and MFCCs were extracted from each frame. 120 most energetic frames from each sound event were used as input for the DBN with 5 hidden layers. Classification accuracy of 64.6% was obtained, which outperformed the GMM-HMM and shallow ANN methods.

2.4.2 Methods with Other Features

Apart from MFCCs, several other methods have been proposed for feature selection and extraction. Chu et al. used MP algorithm to decompose the audio signal into sets of basis vectors and then extracted the time-frequency domain features from the decomposed signals [15]. Their starting poing was that MFCCs do not work for environmental audio as well as they do for speech and music. They claimed the reason was that the diverse content of environmental audio consists of many sound sources with noise-like flat spectrum, preventing the audio to be modeled well by MFCCs. MP algorithm iteratively found the best representation with minimum residual energy, then extracted mean and standard deviaton features corresponding to frequency and scale parameters. They used GMM and KNN as classifiers and observed that GMM outperforms KNN in this task. When the time-frequency domain features were combined with MFCCs in feature set, it led to an increase of recognition accuracy by 21.4% over using only MFCCs as features.

Wavelet based features are also used in environmental sound event detection in [53]. First, wavelet packet decomposition trees of each audio signal are derived. Then, the features such as spectral centroid, sparsity, node energy and spectral

spread are extracted from the child nodes of the wavelet tree. The classifiers were selected as SVM and Gaussian Process (GP). Experiments are conducted in three different datasets. The first dataset consisted of individual sound events such as cat, doorbell, knock etc. The second dataset consisted of the mixture of these isolated sound events. The authors claim that the classifier struggles most when different sound events start and end at the same time. The third dataset is a collection of recordings from a natural environment with different species of birds singing simultaneously. The highest accuracy in the third dataset is obtained as 37.5% by using GP as classifier.

3. IMPLEMENTED SYSTEM

Environmental sound recordings are typically composed of overlapping sound events, which appear and disappear at certain time periods of the recording. The goal of the proposed system is to detect every sound event occuring at any instance of the recordings.

3.1 System Overview

Multi-label sound event recognition can be treated as a multi-label classification problem. The external input to the system is the sound recordings collected from everyday auditory scenes. To represent the characteristics of the sound events, frequency domain features are extracted in short time frames of data. Multi-label DNN is trained with these features as inputs and class activity indicators of sound events as target outputs. Posterior probabilities of sound events are obtained as DNN outputs. During DNN training, the network parameters are updated to minimize a cost function C, which is a function of the posterior probabilities and target outputs. During testing, the posterior probabilities are thresholded and a binary detection vector is obtained for training example. The illustration of the overall system can be found in Figure 3.1.

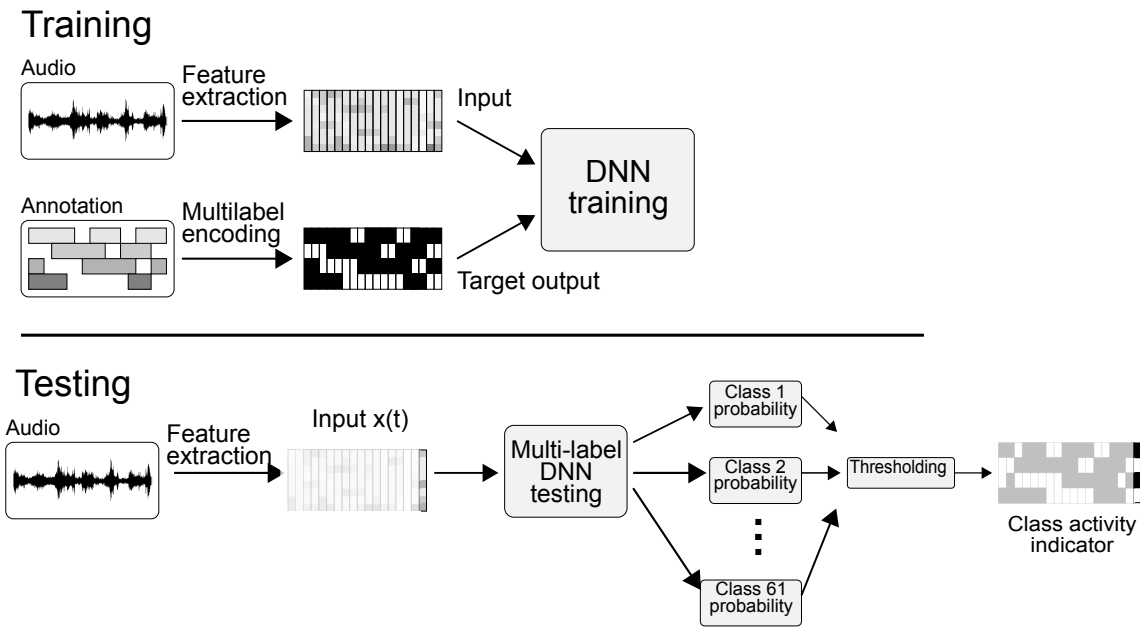

Figure 3.1: System overview.

3.2 Features

Audio features should offer good representation of the time-frequency characteristics of the signal. There are several steps to extract the features from a sound recording.

3.2.1 Pre-processing

Audio recordings are collected in various environments and the conditions in each environment typically differ from each other. The recorder may be too close to the sound sources or there may be too many sound events present at the same time. This would cause an amplitude range difference between recordings from each context and consequently degrade the classification performance. Since we would like to put more emphasis on the spectral characteristics of the recordings while extracting features, we do a peak amplitude normalization as the first phase of the preprocessing:

$$\hat{s}_k = \frac{1}{\max\limits_{i \in [1,\ldots,n]} |s_i|} s_k \tag{3.1}$$

where $\hat{s}$ is the normalized signal, s is the raw signal, $k \in [1, ..., n]$ and n is the number of samples in the signal.

For the next phase of preprocessing, frame-blocking and windowing is conducted. To obtain the spectral information from the sound data, the discrete Fourier transform is taken to model the signals with sinusoids. Fourier transform assumes these sinusoids to be stationary, whereas for the case of audio waves, they are varying

through time. Therefore, it is necessary to first divide the signal into frames.

The natural companion of frame-blocking is windowing. When a signal is divided into frames without windowing, there would be discontinuities at the boundaries, which is undesirable. For this reason, a window function is selected and multiplied by the whole signal to get smooth frames. In audio processing, the window function is usually selected to be some soft window such as Hanning, Hamming, triangle etc. For this work, Hamming window with 50 ms duration and 50% overlap is used, as Hamming window works generally well with the discrete Fourier transform. Hamming window **w** of length N is defined as

$$w_n = 0.54 - 0.46\cos(\frac{2\pi n}{N-1}), \tag{3.2}$$

where n = 1,2...N.

3.2.2 Feature Extraction

The goal of feature selection is to find a relevant feature set for the problem. By choosing the correct features, the amount of training data required to come up with a successful NN model can be significantly reduced. Consequently, the amount of computation during the training is reduced, too. Better accuracy can be obtained in less epochs by using only the relevant features.

In this thesis, experiments are conducted with frequency domain features such as Mel-band energies, log Mel-band energies and MFCCs.

Mel-band Energy Mel-band energy for each short time frame is extracted as explained in Section 2.2.2. The whole frequency range is mapped to 40 Mel bands and the energies are calculated for each band. These 40 energy values form a single instance vector **x** for the ANN.

As an alternative feature set, logarithm of the Mel-band energies are taken and used as features in different experiments for comparison.

MFCC After calculating the log mel-band energies, DCT of the energies is taken. 16 MFCCs are obtained by excluding the first coefficient and retaining the next 16 coefficients. The extraction process is explained in detail in Section 2.2.2.

3.2.3 Concatenation

Sound events in realistic environments usually take at least few seconds and some of the events have a noise-like, flat spectrum. Taking in mind that the time frames are of only 50 ms duration, one certainly expects a correlation between feature vectors extracted from consecutive time frames. This correlation information is prone to

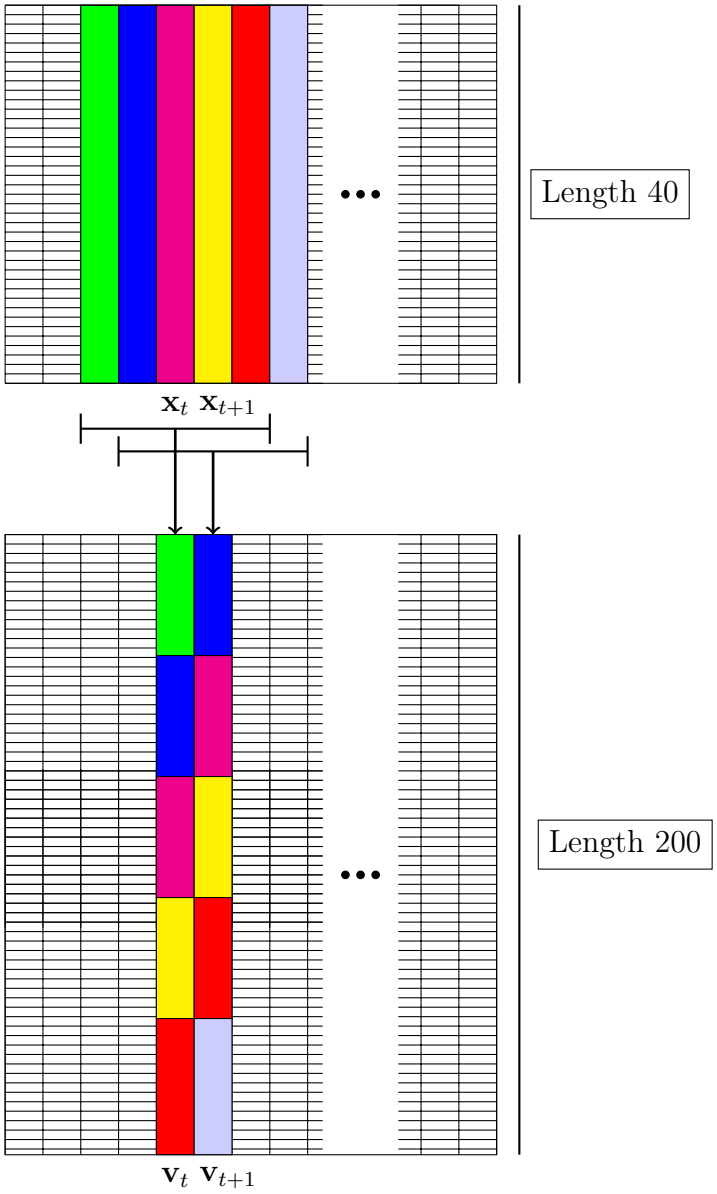

Figure 3.2: Concatenation of $\mathbf{x}_t$ with its 2 adjacent frames on both sides to form $\mathbf{v}_t$.

assist in multi-label classification. Therefore, concatenation method is used on a predetermined number of consecutive frames. The input frame $\mathbf{x}_t$ is concatenated with the frames before and after it to form concatenated input frame $\mathbf{v}_t$. Therefore, $\mathbf{v}_t = [\mathbf{x}_{t-N_{adj}}, x_{t-N_{adj+1}}, ..., x_t, x_{t+1}, ..., x_{t+N_{adj}}]^T$. The number of concatenated frames can be regarded as a parameter for the classification.

For every single frame, a new feature vector is obtained by this method. Therefore, the number of training instances stay the same. After concatenation, the resulting feature vector has a dimension of $|v| = (2 \times N_{\text{adj}} + 1) \times N_{\text{f}}$ where N_{adj} is the number of adjacent frames concatenated with the original frame and N_{f} is the number of features extracted from the short time frame. The concatenation process is illustrated in Figure 3.2.

3.3 Multi-label Classification

Multi-label classification is the general form of multi-class classification. A finite set of classes are given for multi-label classification. In multi-class classification, the task is to associate each learning example with one of the labels, whereas in multi-label classification, each example can be associated with multiple labels. Therefore, if the size of the label set is N, multi-label classifier can associate an example with 2^N different output vectors.

The first possible idea to solve a multi-label classification problem might be to decompose it into multiple single label multi-class classification problems. A network can be trained for each possible class and the results can be combined to get multilabel classification results. However this method ignores the correlation between the classes and therefore has weak expressive power [56]. Instead of this approach, a multilabel DNN is used for multilabel classification purposes in this thesis.

The input vector for the NN is the (concatenated) feature vector obtained from the recordings. The target output vector for the NN is determined by the manual annotations. Each sound event is annotated with its start and end time in the recordings. The temporal position of each input vector, *i.e.*, at what instant the features are extracted, is compared with the start and end times in the annotations. The target output vector for an input vector is a binary vector with length equal to the number of sound event classes. If a sound event is annotated as present for the same time instance with the input vector, the corresponding sound event class is marked with 1 in the target output vector. If a sound event class is marked with 0 in the target output vector, this means that sound event class does not appear at the same instant with the input vector.

3.3.1 DNN Training

Fully-connected feed-forward ANNs with multiple hidden layers are used for multi-label classification purposes in this thesis. Hidden layers are composed of maxout neurons and output layer is composed of sigmoid neurons (explained in Section 2.3.1). In this section, the learning procedure starting from the introduction of a training instance $\mathbf{x}$ until the update of the weights and biases will be explained. The notation will be as follows. The layer indices will be denoted as $l = 0, 1, 2..., M$ for the ANN with $M + 1$ layers total. a_j^l denotes the activation output of the j^{th} neuron in the l^{th} layer. b_j^l denotes the bias of the j^{th} neuron in the l^{th} layer. For the weight between the k^{th} neuron in the $(l-1)^{th}$ layer and the j^{th} neuron in the l^{th} layer, w_{jk}^l will be used. The maxout activation function will be denoted as $f(\cdot)$ and the sigmoid activation function will be denoted as $\sigma(\cdot)$.

To make the notation more compact with less indices, we define a weight matrix

$\mathbf{W}^l$ whose entries are the weights between $(l-1)^{th}$ and l^{th} layer, *i.e.*, its entries are the weights w^l_{jk} between the k^{th} neuron in the $(l-1)^{th}$ layer and the j^{th} neuron in the l^{th} layer. We also define activation output vector for l^{th} layer as $\mathbf{a}^l$ and bias vector as $\mathbf{b}^l$.

As explained in Section 2.3.1, the input layer ($l = 0$) simply passes the input x without any weighing operation, which gives $\mathbf{a}^0 = [x_1, x_2, ..., x_N]$ where N is the number of features in a single training instance $\mathbf{x}$.

The hidden layer activation outputs a^l_j are calculated as

$$a^l_j = f(\sum_k w^l_{jk} a^{l-1}_k + b^l_j) \tag{3.3}$$

or in the compact form

$$\mathbf{a}^l = f(\mathbf{W}^l \mathbf{a}^{l-1} + \mathbf{b}^l) \tag{3.4}$$

where $\mathbf{W}^l\mathbf{a}^{l-1}$ is the application of weight matrix to the activation outputs and $f(\cdot)$ is applied element-wise. Eventually, the activation outputs $\mathbf{a}^M$ from the output layer M is given as

$$\mathbf{a}^M = \sigma(\mathbf{W}^M \mathbf{a}^{M-1} + \mathbf{b}^M) \tag{3.5}$$

Until now, we fed a single training instance $\mathbf{x}$ all the way through the network and obtained the output vector $\mathbf{a}^M$ which is composed of the posterior probabilities for each class. For $\mathbf{x}$, the desired output $\mathbf{y} = \mathbf{y}(x)$ is already given by the annotations. We need to find a way to update the weights and biases of the network so that $\mathbf{a}^M$ approximates to $\mathbf{y}$ for all training inputs $\mathbf{x}$. In order to quantify the distance between the desired output $\mathbf{y}$ and the estimated output $\mathbf{a}^M$, a cost function C is is selected as Kullback-Leibler (KL) divergence [61] and it is calculated as

$$\begin{aligned} \mathrm{KL}(\mathbf{y}||\mathbf{a}^M) = \sum_{i=1}^{N} y_i \ln y_i - y_i \ln a^M_i \\ +(1-y_i)\ln(1-y_i) - (1-y_i)\ln(1-a^M_i) \end{aligned} \tag{3.6}$$

where N is the number of hidden neurons in the output layer. For binary y, as in our case, some terms in Equation (3.6) drop out and the resulting KL divergence is

$$C(w,b) = \mathrm{KL}(\mathbf{y}||\mathbf{a}^M) = \sum_{i=1}^{N} -y_i \ln a^M_i - (1-y_i)\ln(1-a^M_i). \tag{3.7}$$

This cost function is also called *cross-entropy cost function* as the way it is referred

in information theory.

The estimated posterior probabilities $\mathbf{a}^M$ are in the range $[0, 1]$ and the desired outputs y are either 0 or 1, therefore $C(w, b)$ is non-negative. Moreover, when y and $\mathbf{a}^M$ values are closer to each other, then the cost function C is closer to zero. The objective of the training is to make $C(w, b)$ as close as possible to zero. In order to do that, a set of weights $\mathbf{w}$ and biases $\mathbf{b}$ should be found which makes the cost minimum. This is done by updating them iteratively with their gradients. This algorithm is called *gradient descent algorithm*.

The gradient vector ∇C is defined as

$$\nabla C \equiv \left(\frac{\partial C}{\partial \mathbf{v}}\right) \tag{3.8}$$

where $\mathbf{v}$ represent all the weights and biases in the network. The change of C can be given as

$$\Delta C \approx \nabla C \cdot \Delta \mathbf{v} \tag{3.9}$$

where ΔC represents the change in C and $\Delta \mathbf{v}$ represents the change in $\mathbf{v}$. According to Equation 3.9, we can choose $\Delta \mathbf{v}$ so that it would make ΔC negative. For this reason, we choose

$$\Delta \mathbf{v} = -\eta \nabla C \tag{3.10}$$

where η is the learning rate of the system. Equation (3.10) gives $\Delta C \approx -\eta ||\nabla C||^2$, $\Delta C \leq 0$ and therefore C will decrease. The weights and biases will be updated as

$$\tilde{\mathbf{v}} = \mathbf{v} + \Delta \mathbf{v} = \mathbf{v} - \eta \nabla C \tag{3.11}$$

where $\tilde{\mathbf{v}}$ represents the updated weights and biases of the network. The learning rate η should be selected carefully. Too large η would cause to a violation of approximation and $\Delta C > 0$. Too small η would make the changes $\Delta \mathbf{v}$ too small and the convergence of the gradient descent will be slow. As far as our knowledge, there is no certain rule of thumb on the value of η and the optimum value is often found by grid search.

The updates on weights and biases in a single layer are given by Equation (3.11) as

$$\tilde{\mathbf{W}} = \mathbf{W} - \eta \frac{\partial C}{\partial \mathbf{W}} \tag{3.12}$$

$$\tilde{\mathbf{b}} = \mathbf{b} - \eta \frac{\partial C}{\partial \mathbf{b}}. \tag{3.13}$$

Recalling Equation (3.7), we need to calculate the KL-divergence over all training instances to find the cost C. In order to compute the gradient ∇C, ∇C_x should be computed individually for each instance and averaged, so that $\nabla C = \frac{1}{n} \sum_x \nabla C_x$.

When the number of training instances is very large, this process can be really slow. An alternative way is to select m random training instances and calculate the gradient ∇C over these m instances. This algoithm is called *stochastic gradient descent* (SGD) algorithm. The group of m training instances are called *mini-batches*. When m is selected large enough, this gives a good estimate of true gradient ∇C and speeds up the process. Then, Equations (3.12) and (3.13) turn out to be

$$\tilde{\mathbf{W}} = \mathbf{W} - \frac{\eta}{m} \sum_j \frac{\partial C_{\mathbf{x}_j}}{\partial \mathbf{W}} \tag{3.14}$$

$$\tilde{\mathbf{b}} = \mathbf{b} - \frac{\eta}{m} \sum_j \frac{\partial C_{\mathbf{x}_j}}{\partial \mathbf{b}} \tag{3.15}$$

where $\mathbf{x_j}$ denote a single training instance in the mini-batch. Until every training instance is processed, random mini-batches are selected and weights and biases are updated. The whole process is called an *epoch*. A new epoch is initiated right after one is completed.

Backpropagation Algorithm The partial derivatives $\frac{\partial C}{\partial \mathbf{W}}$ and $\frac{\partial C}{\partial \mathbf{b}}$ of the cost function with respect to all weights and biases of the network are computed by a fast algorithm called *backpropagation (BP) algorithm* [28; 60]. BP algorithm is used to feed the error backwards from the output layer to the first layer and update the weights and biases accordingly.

BP algorithm is explained in four fundamental equations. Let us define the weighted input of the j^{th} neuron in the l^{th} layer as

$$\begin{aligned} z_j^l &= \sum_j w_{kj}^l a_j^{l-1} + b_k^l \\ &= \sum_j w_{kj}^l f(z_j^{l-1}) + b_k^l. \end{aligned} \tag{3.16}$$

where $f(\cdot)$ is the maxout function for hidden layers and sigmoid function for the output layer. Let us define an intermediate variable e_j^l to define the error in the j^{th} neuron in the l^{th} layer. The cross-entropy cost function $C(w, b)$ for a single training instance is defined as $C = -\mathbf{y} \log \mathbf{a}^M - (1 - \mathbf{y}) \log(1 - \mathbf{a}^M)$. The error e_j^l is defined as

$$e_j^l = \frac{\partial C}{\partial z_j^l} \tag{3.17}$$

and $\mathbf{e}^l$ notation will be used to denote the vector of errors in the l^th layer. The error

for the output layer M is then given as

$$e_j^M = \frac{\partial C}{\partial z_j^M} = \sum_k \frac{\partial C}{\partial a_k^M} \frac{\partial a_k^M}{\partial z_j^M} \tag{3.18}$$

by the chain rule. Recalling $a_k^M = \sigma(z_k^M)$ for the output layer with sigmoid activation, a_k^M only depends on z_k^M and $\frac{\partial a_k^M}{\partial z_j^M} = 0$ for $j \neq k$. Equation (3.18) then simplifies to

$$e_j^M = \frac{\partial C}{\partial a_j^M} \frac{\partial a_j^M}{\partial z_j^M} = \frac{\partial C}{\partial a_j^M} \sigma'(z_j^M). \tag{3.19}$$

where $\frac{\partial C}{\partial a_j^M} = -\frac{y_j(x)}{a_j^M} - \frac{1-y_j(x)}{a_j^M - 1}$ from the definition of the cost function C.

The error in the output layer is now defined and we need to define the error in each layer as the function of the error in the next layer. Again, starting from $e_j^l = \frac{\partial C}{\partial z_j^l}$ and using the chain rule

$$e_j^l = \sum_k \frac{\partial C}{\partial z_{k+1}^l} \frac{\partial z_{k+1}^l}{\partial z_j^l} \tag{3.20}$$

$$e_j^l = \sum_k \frac{\partial z_{k+1}^l}{\partial z_j^l} e_k^{l+1}. \tag{3.21}$$

Recalling Equation (3.16), we obtain

$$\frac{\partial z_{k+1}^l}{\partial z_j^l} = w_{kj}^{l+1} f'(z_j^l) \tag{3.22}$$

and substituting this into Equation (3.21), we get

$$e_j^l = \sum_k w_{kj}^{l+1} e_k^{l+1} f'(z_j^l) \tag{3.23}$$

At this point, we know how to compute the error e_j^l for each layer. Now, we will define the partial derivatives $\frac{\partial C}{\partial \mathbf{W}}$ and $\frac{\partial C}{\partial \mathbf{b}}$ in terms of e_j^l. Using the chain rule, the partial derivative $\frac{\partial C}{\partial w_{jk}^l}$ can be defined as

$$\frac{\partial C}{\partial w_{jk}^l} = \frac{\partial C}{\partial z_j^l} \frac{\partial z_j^l}{\partial w_{jk}^l} \tag{3.24}$$

Recalling Equation (3.16)

$$\frac{\partial z_j^l}{\partial w_{jk}^l} = a_k^{l-1} \tag{3.25}$$

and substituting Equations 3.17 and 3.25 into 3.24, we obtain

$$\frac{\partial C}{\partial w_{jk}^{l}} = a_k^{l-1} e_j^l. \tag{3.26}$$

Using the same approach as Equation (3.26), it can easily be proved that

$$\frac{\partial C}{\partial b_{k}^{l}} = e_j^l. \tag{3.27}$$

Equations 3.19, 3.23, 3.26 and 3.27 are the four fundamental equations of the BP algorithm.

When the BP algorithm is combined with SGD algorithm, the only difference is that the error $e_j^{x,l}$ and weighted input $z_j^{x,l}$ are computed as a vector over the whole minibatch with size m and $x = 0, 1, 2..., m-1$. The algorithm is repeated for a new mini-batch until all the training instances are processed. When the epoch is finished, a new epoch is initiated. Until the termination criterion is met, the DNN training continues. The termination criterion is defined by the user as usually the maximum number of epochs, a certain value of cost C or either one of these criteria is met. One can predict that training the network for very high number of epochs would bring high success. In fact, this only causes the weights and biases to adapt specifically for the training data and not generalize. This phenomenon is famously known as *overfitting*. Termination criteria are used to prevent overfitting. The pseudocode for the DNN training algorithm is given in Algorithm 1.

Momentum Momentum is an optimization method that helps the network to converge faster. Recalling Equation (3.11) in gradient descent algorithm, the weights of the network are updated as

$$\tilde{\mathbf{w}} = \mathbf{w} - \eta \nabla C \tag{3.28}$$

where $\tilde{\mathbf{w}}$ represents the updated weights, η is the learning rate and ∇C is the gradient vector. When the gradients are small, the updates have only a little effect. The process can be speeded up by introducing an additional *momentum* term to Equation (3.11) resulting in

$$\begin{aligned} \tilde{\mathbf{v}} &= \mu \mathbf{v} - \eta \nabla C \\ \tilde{\mathbf{w}} &= \mathbf{w} + \tilde{\mathbf{v}} \end{aligned} \tag{3.29}$$

where $\tilde{\mathbf{v}}$ represents the *velocity* variables for each weight and μ is the momentum coefficient. With consecutive updates in the right direction of gradient, the weights will be modified by larger amounts and the network will converge faster. However,

Algorithm 1 DNN training algorithm

1: $random\_initialize\_weights$;
2: $random\_initialize\_biases$;
3: $epoch \leftarrow 0$;
4: $processed\_data \leftarrow 0$;
5: $N \leftarrow total\_number\_of\_training\_instance$;
6: $m \leftarrow mini - batch\_size$;
7: **while** $Termination\_Criterion == False$ **do**
8: $x \leftarrow rand(m, N)$; ▷ Select m training inputs with indices from [1,...,N]
9: $a^{x,0} \leftarrow Training\_Data[x]$; ▷ Input a mini-batch of training input to DNN
10: **for** $l := 1$ **to** $M - 1$ **do** ▷ Total number of layers is M, indexed with $l = 0, 1, ..., M - 1$
11: $z^{x,l} = w^l a^{x,l-1} + b^l$; ▷ Feed the information forward
12: $a^{x,l} = f(z^{x,l})$; ▷ Find the activation outputs
13: **end for**
14: $e^{x,M} = \frac{\partial C^x}{\partial a^{x,M}} f'(z^{x,M})$; ▷ Error for the output layer
15: **for** $l := M - 2$ **to** 1 **do**
16: $e^{x,l} = w^{l+1} e^{x,l+1} f'(z^{x,l})$; ▷ Backpropagate the error
17: **end for**
18: **for** $l := M - 1$ **to** 1 **do**
19: $w^l \leftarrow w^l - \frac{\eta}{m} \sum_x e^{x,l} (a^{x,l-1})^T$; ▷ Weight updates backwards
20: $b^l \leftarrow b^l - \frac{\eta}{m} \sum_x e^{x,l}$; ▷ Bias updates backwards
21: **end for**
22: $processed\_data += m$; ▷ Increase the amount of processed data by m
23: $Training\_Data[x] = [\cdot]$; ▷ Empty the processed mini-batch in training data
24: $N -= m$; ▷ Decrease the amount of unprocessed data by m
25: **if** $processed\_data \neq total\_number\_of\_training\_instances$ **then**
26: **goto** 8; ▷ There are still unprocessed training inputs, start over
27: **else**
28: $epoch++$; ▷ Initiate new epoch and update the epoch number by incrementing with 1
29: $processed\_data \leftarrow 0$;
30: **end if**
31: $check\_termination\_criterion$;
32: **end while**
33: **Return** [w;b];

there is also a possibility to overshoot the global minimum, i.e. where the convergence should happen. Momentum coefficient μ value is selected between 0 and 1. When $\mu = 0$, the network is trained the same way in basic gradient descent algorithm. The optimal value of μ is found by grid search. During the first few epochs, the random initial values for weights and biases may create very large gradients, so it makes practical to start with a low momentum coefficient like 0.5 for a number of epochs and then increase the coefficient slowly later on [64].

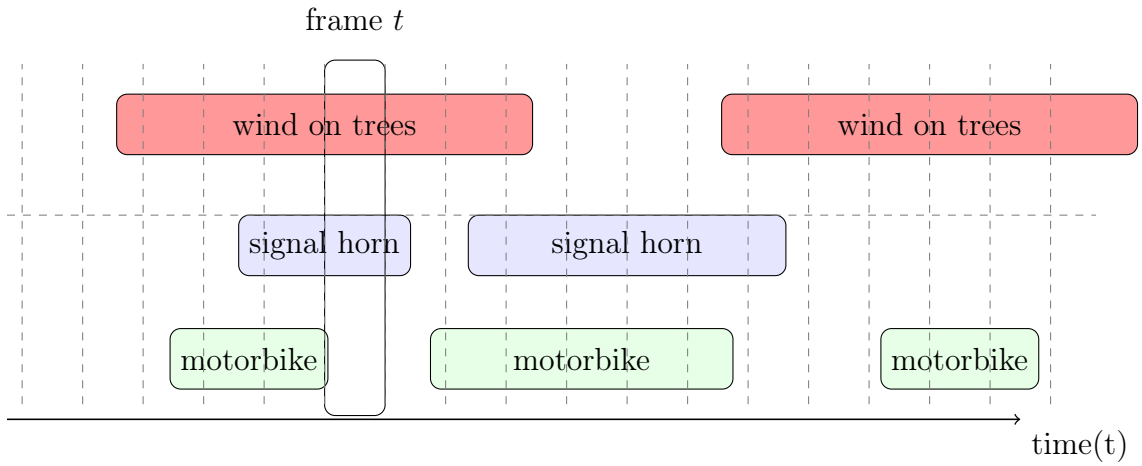

Figure 3.3: An example of overlapping sound events in a possibly realistic environment. Frame t represents the short time frame from the recording where only signal horn and motorbike events are present.

3.3.2 DNN Testing

After supervised training of the network, test data is used to evaluate the accuracy for the testing material. Test data does not include any examples that are also in the training data, so the network does not encounter with testing examples before they are tested. The weights and biases are not updated for any testing example. Each example $\mathbf{x}$ is fed to the network and the posterior probabilities $\mathbf{a}^M(\mathbf{x})$ are obtained from the output layer. In order to obtain the detected events and compare them with the desired outputs, a thresholding operation is necessary over posterior probabilities. The detection estimate $\hat{\mathbf{y}}$ is obtained by

$$\hat{y}_i(\mathbf{x}) = \begin{cases} 1, & a_i^M(\mathbf{x}) \geq 0.5 \\ 0, & a_i^M(\mathbf{x}) < 0.5 \end{cases} \tag{3.30}$$

where $i = 0, 1, ..., N-1$ and N is the number of possible outcomes for a training example. The binary estimate $\hat{\mathbf{y}}$ is then evaluated against the binary, desired output $\mathbf{y}$ for each testing example.This process is illustrated in Figures 2.1 and 3.1.

Table 3.1: Binary estimation process for frame t in Figure 3.3.

Class	Posterior probabilities	Binary estimates	Target outputs
Wind on trees	0.9	1	1
Signal Horn	0.45	0	1
Motorbike	0.25	0	0

3.3.3 Post-processing

Environmental sound events typically take at least a few seconds and the sound data is processed in short time frames of 50 ms. Therefore, one expects that when a sound event is first detected in a single frame, it is highly possible that it also appears in at least a few of the following frames, too. In addition, if an event is detected for 10 consecutive frames, then not detected for 2 frames and again detected for 10 consecutive frames, this does not seem realistic, either. This case usually occurs when the posterior probability of the event is very close to the binarizing threshold. Our experiments with DNNs showed us that DNN outputs may sometimes result with such noisy characteristics. In order to overcome this noise problem, *sliding window* method is proposed in this thesis.

Sliding window method is based on a simple but very efficient idea. For each sound event, the median of the binary output vector is found and the value is stored in a new matrix at the position of the first frame in the window. Then, the window is shifted one frame and the process is repeated until every test instance is used in a window. The algorithm for the post-processing method is given in Algorithm 2 and the process is illustrated with an example in Figure 3.4.

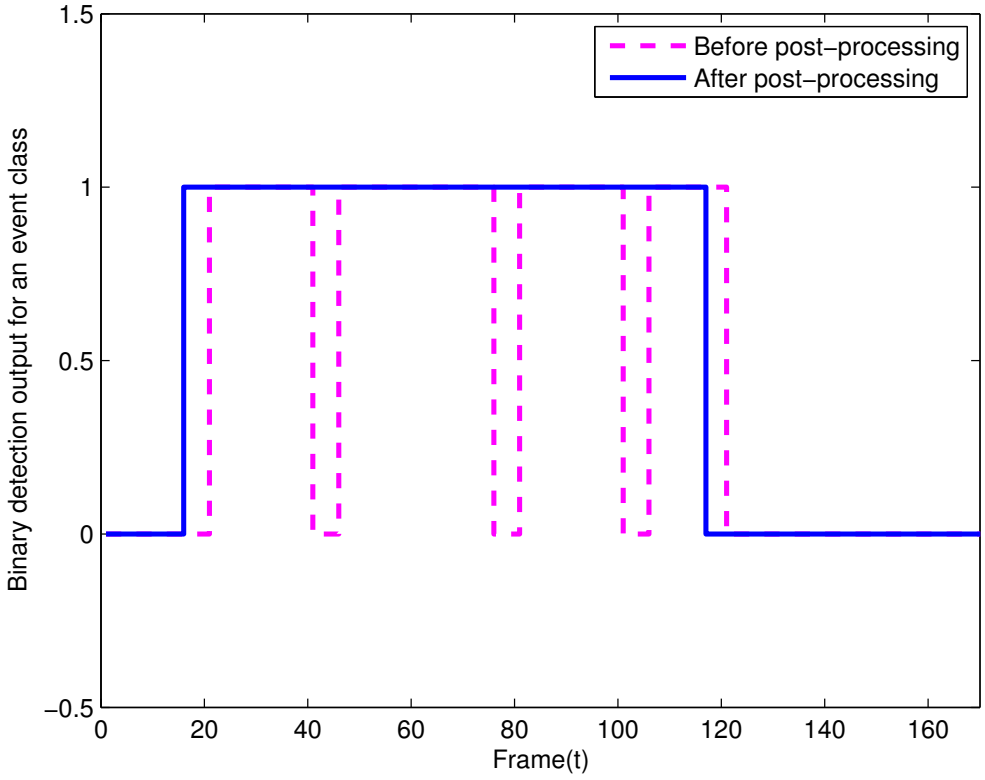

Figure 3.4: Sliding window method.

Algorithm 2 Sliding window algorithm

1: $X \leftarrow test\_output\_matrix$; ▷ X is the matrix containing binary estimates for each training instance
2: $Y := X$;
3: $N \leftarrow total\_number\_of\_training\_instance$;
4: $C \leftarrow total\_number\_of\_classes$;
5: $K \leftarrow number\_of\_frames\_in\_the\_sliding\_window$;
6: **for** i:=1 **to** N-K **do**
7: $window := X[:, i : i + K]$;
8: **for** j:=1 **to** C **do**
9: $Y[j, i] = median(window[j, :])$;
10: **end for**
11: **end for**
12: **Return** Y;

3.4 Division of Data

Supervised training models typically use two different datasets. One dataset is reserved for training purposes and the other one is reserved for testing purposes. Each dataset is composed of a huge amount of individual instances. Each instance is represented by two vectors: feature vector and target vector. Feature vector is composed of the features that are extracted from the specific instance. Target vector defines the desired outcome for the same instance. During training, feature vectors of the training instances are used as network input, whereas the target vectors are used for calculating the error between the desired and predicted output. During testing, the accuracy is calculated according to the chosen evaluation metric. Training dataset and testing dataset are completely separate, *i.e.* there are no instances that is used while both training and testing. This prevents the model to test itself over an instance that it has already been encountered.

The topology and learning algorithm of a DNN is dependent on multiple parameters. Some of these parameters can be listed as number of hidden units, number of hidden layers, learning rate, activation function, initial weight and bias values etc. Other than these structural parameters, the termination criterion for the training is another important parameter that determines the training procedure. Unfortunately, there are only a few guidelines on how to set these parameters for efficient learning. One usually has to conduct controlled experiments by fixing all the parameters except one and running experiments for several values of that parameter.

NN training is stopped when a certain termination criterion is met. This criterion is checked at the end of every epoch. The common way of stopping a NN training is to calculate the overall error for the model and see if it is decreasing anymore compared to previous epochs. This can be executed over training data, but since

the network will be trained for the same dataset in each epoch, the results may be overly optimistic. In other words, the network parameters may adapt to that specific training dataset and fail to generalize over unseen instances. This undesired situation, namely *overfitting*, can be avoided by using a third dataset. This dataset is called *validation dataset*. At the end of each epoch, validation dataset instances are given as inputs to the network and the difference between the predicted and desired outputs is calculated. If this value is less than or equal to a predetermined target error value, or if it does not change in many consecutive epochs, then the training is stopped. Validation instances differ from *training dataset* in the sense that their target output vectors are not used in the SGD algorithm during training. They also differ from *testing dataset* in the sense that the evaluation of the network is not conducted over them.

The whole dataset is divided into non-overlapping training and testing datasets in this thesis. There are 10-11 recordings from each context. 2-3 of these recordings are used as testing dataset and 8-9 of them are used as training dataset. Five-fold cross validation is performed on this database, *i.e.*, 5 different combinations are made on the selection of recordings so that each recording is used in a testing dataset at least once. First fold is used to tune the parameters and the network is evaluated for the other four folds. The final results are presented by taking the average of the results for the last four folds.

4. EVALUATION

4.1 Database

The sound database used in thesis is provided by Audio Research Team under Multimedia Research Group in Tampere University of Technology [59]. The recordings are collected from real-life auditory scenes. There are a total of 103 recordings in the database and each of them are 10 to 30 minutes long. The total duration of the recordings is 1133 minutes. The recording is performed as a binaural recording, where the person responsible for recording wears in-ear microphones during the recording. The recording equipment consists of a Soundman OKM II Klassik/Studio A3 electret microphone and a Roland Edirol R-09 digital recorder. Recordings were done using 44.1 kHz sampling rate and 24-bit resolution. In this thesis, we are using monophonic versions of the recordings, i.e., two channels are averaged to one channel.

The recordings are collected from 10 different, real-life physical contexts: basketball match, beach, inside a public bus, inside a car, hallways, inside the office, restaurant, grocery, street and a track and field stadium. Basketball and track and field stadium contexts were selected as sports and leisure time scenes. Grocery and restaurant are public space scenarios which are difficult to work on due to high noise in realistic environment. Street, bus and car contexts were used to reflect the audio atmosphere while travelling. Finally, hallways and office contexts represent typical indoor work environments.

The annotation of the recordings is done manually by the same person. Each event is annotated with its name, start and end time in each recording. Some of the events occur multiple times in a single recording. The events are chosen to be easily recognized by perception; i.e.; a person without any hearing impairment can detect the same sequence of events by listening to the raw record. Some of the events include *brakes_squeak, cheering, referee whistle, cash_register, refrigerator* etc. Out of these events, the ones that occur 10 or more times in the database are categorized as 60 distinct event classes and 1 class is labeled as unknown for less frequent events. In each context, there are events annotated from 9 to 16 different event classes. Some of the events can be found in multiple contexts and some of the events are context specific; such as referee whistle and ball hitting floor. The number of annotated events and the total length of recordings per each recording

environment is given in Table 4.1. The number and duration of annotations for each class are not distributed equally, as some classes such as speech are more frequent than others, which is the normal case in a real-life environment. Event classes are given in Table 4.2.

During pre-processing and feature extraction, MIRtoolbox was used [65] . Pylearn2 machine learning library was used for the NN implementation [66]. Its functionality is based on Theano [67], therefore it allows us to use compile the codes by using Graphics Processing Unit (GPU) for fast operation.

Table 4.1: Number of events annotated per context and total length (in minutes) of recordings .

Context	**Number of annotated events**	**Total Length**
Basketball game	990	80
Beach	738	197
Inside a bus	1729	146
Inside a car	582	111
Office facility	1220	105
Hallway	822	100
Restaurant	780	96
Grocery shop	1797	88
Street	827	102
Track & field stadium	793	108

Table 4.2: Event classes used in the evaluation database.

No.	Event Class	No.	Event Class	No.	Event Class
1	'applause'	21	'click'	41	'refrigerator'
2	'background'	22	'coins_keys'	42	'road'
3	'ball_hitting_floor'	23	'coughing'	43	'seatbelt'
4	'beep'	24	'crowd_sigh'	44	'shoe_squeaks'
5	'bicycle'	25	'crowd_walla'	45	'shopping_basket'
6	'bird'	26	'dish_washer'	46	'shopping_cart'
7	'brakes_squeak'	27	'dishes'	47	'sigh'
8	'breathing_noises'	28	'dog_barking'	48	'signal_horn'
9	'bus'	29	'door'	49	'sliding_door'
10	'bus_door'	30	'engine_off'	50	'sneezing'
11	'car'	31	'footsteps'	51	'speech'
12	'car_door'	32	'keyboard'	52	'traffic'
13	'car_engine_starts'	33	'laughter'	53	'turn_signal_noise'
14	'cash_register'	34	'motor_noise'	54	'water_splashing'
15	'cat_meaowing'	35	'motorbike'	55	'wheel_noise'
16	'chair'	36	'mouse_scrolling'	56	'whistling'
17	'cheering'	37	'music'	57	'wind_on_trees'
18	'child'	38	'paper_movement'	58	'windscreen_wipers'
19	'clapping'	39	'pressure_release'	59	'wrapping'
20	'clearing_throat'	40	'referee_whistle'	60	'yelling'
61	'unknown'				

4.2 Evaluation Procedure

The detection accuracy of the proposed method is computed with F1 score, a widely used metric in sound event detection systems [14; 62; 63]. Accuracy is calculated inside non-overlapping one-second chunks of time frames and the final accuracy is found by taking the average of the block accuracies. F1 score is a measurement of the test accuracy in binary classification and it is calculated based on precision and recall.

As the first stage of evaluation, the detection estimates for the testing examples are divided into non-overlapping one-second chunks. Each testing example $\mathbf{x}$ is composed of sound features extracted in 50 ms time frames with 25 ms overlap. Therefore, $1000/(50-25) = 40$ consecutive time frames correspond to a one second block in the recording. Every $\mathbf{x}$ is associated with a detection estimate vector $\hat{\mathbf{y}}$ as explained in 3.3.2. The evaluation is done inside a 61x40 binary detection matrix $\mathbf{D}$, which is formed by combining 40 detection estimate vectors of length 61 (recall 61 is the number of event classes). In this matrix, the detected event classes in a second of recording are represented by 1.

The second stage is the comparison of the target outputs with the detected out-

puts. A binary target output vector $\mathbf{y}(\mathbf{x})$ with length 61 is present from the annotations for each testing example x. By combining 40 target output vectors together in the same fashion as detection estimate vectors, a target output matrix $\mathbf{T}$ is obtained. The comparison between $\mathbf{D}$ and $\mathbf{T}$ is done so that

- An event class is regarded as *correctly detected* if it is detected in any of the 40 frames in $\mathbf{D}$ and it is also annotated in any of the 40 frames in $\mathbf{T}$.
- An event class is regarded as *wrongly detected* if it is detected in any of the 40 frames in $\mathbf{D}$ but has no annotation for the same block in $\mathbf{T}$.
- An event class is regarded as *missed class* if it is annotated in any of the 40 frames in $\mathbf{T}$ but not detected in any of the 40 frames in $\mathbf{D}$.

The number of correctly detected event classes are obtained by counting the correctly detected event classes in respond to the above explanation in the corresponding block. The same is true for wrongly detected and missed event classes. Precision inside the block is calculated as

$$precision = \frac{\#correctly\_detected\_event\_classes}{\#correctly\_or\_wrongly\_detected\_event\_classes} \tag{4.1}$$

and recall inside the block is calculated as

$$recall = \frac{\#correctly\_detected\_event\_classes}{\#correctly\_detected\_event\_classes + \#missed\_event\_classes}. \tag{4.2}$$

F1 score is calculated as the harmonic mean of these two parameters

$$\text{F1 score} = \frac{2 \times precision \times recall}{precision + recall}. \tag{4.3}$$

F1 score for each one-second block is calculated by this method. Context-wise F1 score is calculated by taking the mean of the F1 scores of the block which is found in the recording of the corresponding context. Overall F1 score is obtained by taking the mean of context-wise F1 scores.

Table 4.3: Detection matrix $\mathbf{D}$

Class no.	Frame 1	Frame 2	Frame 3	Frame 4	Frame 5
1	0	0	1	1	0
2	0	1	1	0	0
3	0	0	0	0	0
4	0	1	0	0	0
5	0	0	0	0	0

Table 4.4: Target output matrix **T**

Class no.	Frame 1	Frame 2	Frame 3	Frame 4	Frame 5
1	0	0	1	1	1
2	0	0	0	0	0
3	1	1	1	0	0
4	0	0	0	1	0
5	0	0	0	0	1

The accuracy metrics are illustrated by an example. The block-wise comparison procedure is explained over the small-scale detection and target output matrices on Tables 4.3 and 4.4. Class 1 is regarded as correctly detected, because it is present in $D_{1,3}$, $D_{1,4}$ and at the same time $T_{1,3}$, $T_{1,4}$ and $T_{1,5}$. Class 2 is regarded as wrongly detected, because it is present in $D_{1,2}$, $D_{1,3}$ but it is not present in T. Class 3 is regarded as missed class, because it is not present in **D**, although it is present in $T_{1,1}$, $T_{1,2}$ and $T_{1,3}$. Class 4 is also regarded as correctly detected, because it is present in $D_{1,2}$ and at the same time $T_{1,4}$. Finally, class 5 is regarded as missed class, because it is not present in **D**, although it is present in $T_{1,5}$. As a result, we have 2 correctly detected classes, 1 wrongly detected class and 2 missed classes. This leads to 66.6% precision, %50 recall and 57.1% F1 score, which is used as detection accuracy for this specific block.

Block-wise F1 score metric is particularly useful in environmental sound event detection. It emphasizes the detection of a particular event class in a certain time block, rather than detecting the exact location inside the block [52]. Therefore, it works well with applications that require rather coarse time resolution, which is the case for environmental sound event detection.

4.3 Results

4.3.1 Configuration

DNN parameters are selected over a huge number of controlled experiments in this thesis. Among these experiments, the network setup with the highest overall accuracy is established by using the following parameters:

Table 4.5: Network parameter configuration for the model with highest accuracy.

Parameter	**Value**
Feature	Mel-band Energy
Number of concatenated frames	5
Number of hidden layers	2
Number of hidden neurons in each layer	800
Activation function for hidden layer neurons	Maxout
Number of neurons used in maxout pooling	2
Activation function for output layer neurons	Sigmoid
Learning rate	0.02
Initial weight and bias range	(-0.001,0.001)

Experiments in the results section are done by using the configuration in Table 4.5 as base point. In each subsection, the mentioned parameter value is spanned while keeping all the others fixed. This way, the observation of the effects of the individual parameters is aimed.

4.3.2 Effects of ANN Parameters

Effect of Frame Concatenation Frame concatenation is used to model the dynamic characteristics of the audio signal and to avoid missing the time information completely. The number of concatenated frames is spanned between 0 to 17 while keeping other parameters fixed as described in 4.3.1. The concatenation is done for each frame and its adjacent frames on both sides, *e.g.*, 7 concatenated frames setting is implemented for each frame and its 3 adjacent frames on both sides in time domain, as explained in 3.2.3. The results are given in Figure 4.1. The best performance is observed with 5 concatenated frames. It offers 5% increase in accuracy over the system without frame concatenation. Increasing the number of concatenated frames higher than 5 does not give any improvement in accuracy, although it vastly increases the computational power requirements.

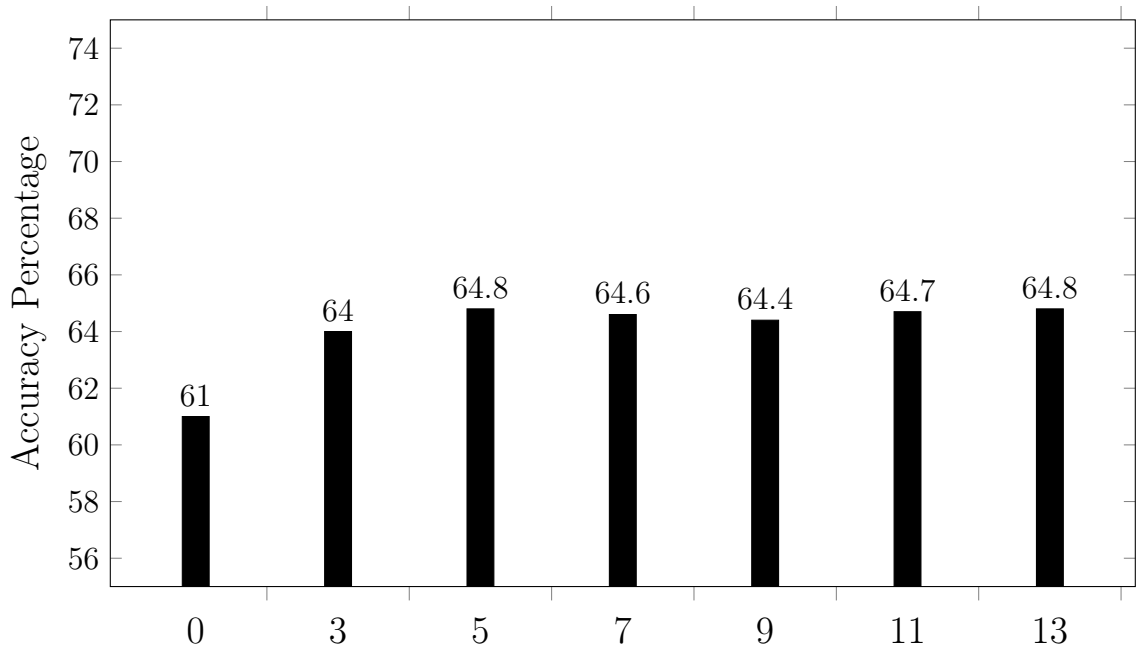

Figure 4.1: The detection accuracy as a function of number of concatenated frames.

Effect of Post-processing The post-processing method, namely *sliding window*, is applied as explained in Section 3.3.3. The window size is selected as 10 time frames, which spans 250 ms of audio data. The increase in accuracy for the post-processed output is clear in Figure 4.2.

The improvement by the post-processing can be explained by the noise in the posterior probabilities of the event classes, which is obtained from the output layer of the network. There are some occasions when the posterior probability for a class jumps significantly for a single frame compared to the frames before and after. This is not realistic since environmental sound events rarely occur for only 25 ms and disappear. If this noisy posterior probability exceeds the thresholded value, the corresponding event class is marked as detected in the one-second time block, which consists of 40 time frames. If the event class does not appear for that one-second block in reality, this single detection basicly poisons the whole block of frames and decreases the accuracy. Sliding window method helps to smooth these singularities and provide a better detection performance.

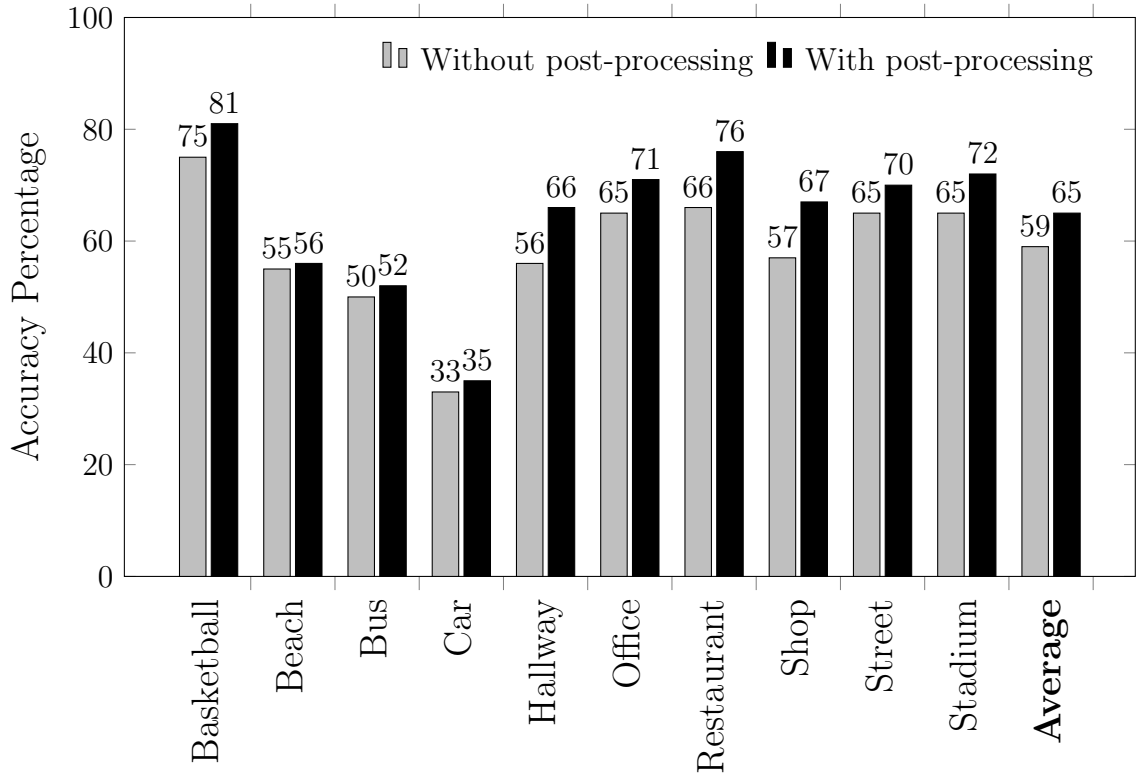

Figure 4.2: Context-wise detection accuracies (F1 score) for proposed system before post-processing and after post-processing.

Effect of the Number of Hidden Neurons The effect of the number of hidden neurons in each hidden layer is investigated in Figure 4.3. In this thesis, two hidden layers are used for ANN and the number of neurons are the same for both layers.

The network's expressional power is highly dependent on the number of hidden neurons. Therefore, it is essential to have enough hidden neurons to represent the nonlinearities of the system. However, when the number of hidden neurons is increased to very high values, the number of weights and biases increase correspondingly and huge computational power requirements are encountered. Another risk of increasing the number of hidden neurons is that the weights and biases might adapt to the training data and cause to overfitting, especially if the number of training examples is not sufficient.

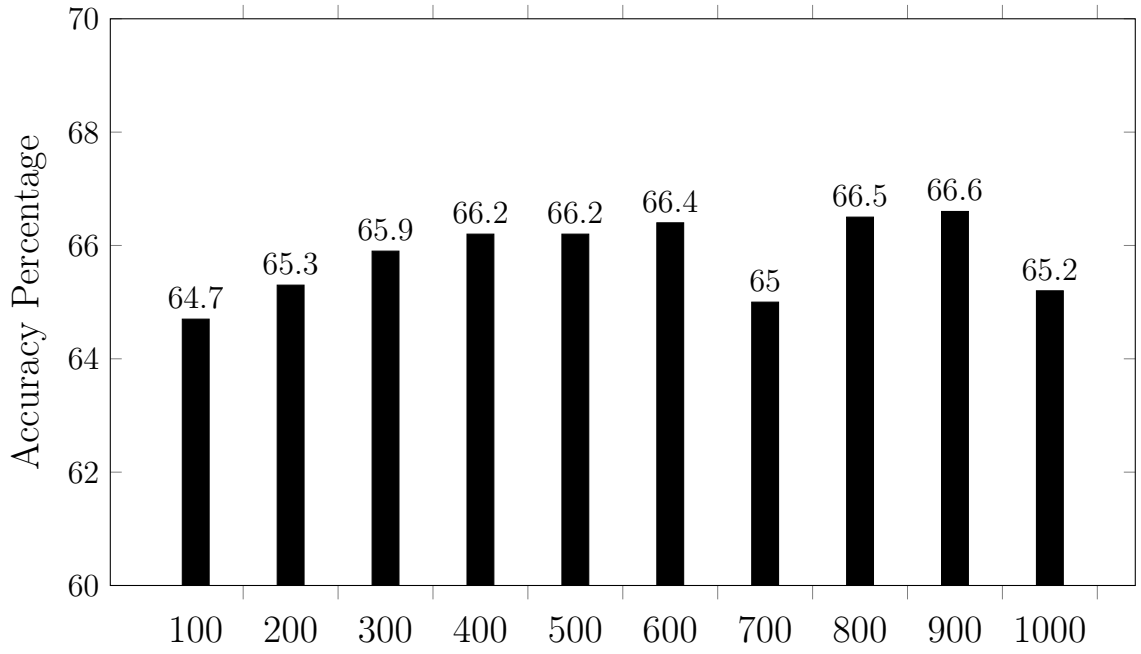

Figure 4.3: The detection accuracy as a function of hidden neuron numbers in each hidden layer.

Effect of Activation Function In order to determine the effect of the hidden unit activation function, a different DNN topology is experimented. Hidden layer activation functions are replaced from maxout function to logistic sigmoid function (see 2.3.1). As illustrated in Figure 4.4, the maxout function offers better accuracy in every context and overall. During both experiments, the parameters such as learning rate and the number of hidden neurons were selected the same. One may find a better initial configuration for sigmoid neurons by doing grid search over these parameters.

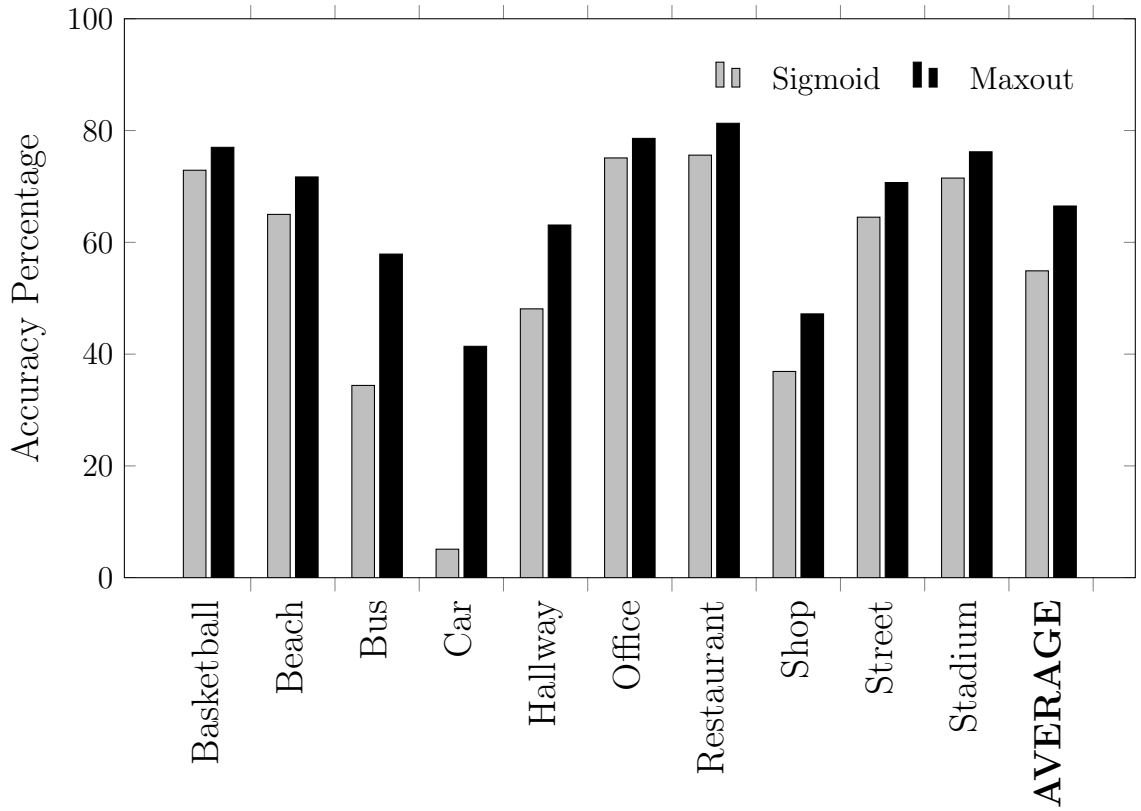

Figure 4.4: Context-wise detection accuracies (F1 score) for proposed system with maxout layers compared to the the system with sigmoid hidden layers.

Effect of Mini-batch Size Mini-batch size determines in what size of example batches the gradients of the cost function will be computed and the ANN parameters are updated (see SGD algorithm in Section 3.3.1). Using small-sized batches for cost function calculation gives a good estimate of the true values of the cost. However, when the mini-batch size is increased, the cost function estimated from the batches are not close enough to the true values and it decreases the detection accuracy. Note that there are around 2.1 million training examples extracted from the database. Therefore when the mini-batch size is selected as 2500, the weights are updated around 840 times in a single epoch. But still, it leads to a detection accuracy decrease of 20% compared to a system with mini-batch size of 100. Another observation is that when smaller mini-batch size is used, the system converges to an acceptable accuracy in less epochs than when a bigger mini-batch size is used. This is due to the fact that there are more weight updates in a single epoch for a small mini-batch size. Experiments on the effect of mini-batch size is illustrated in Figure 4.5.

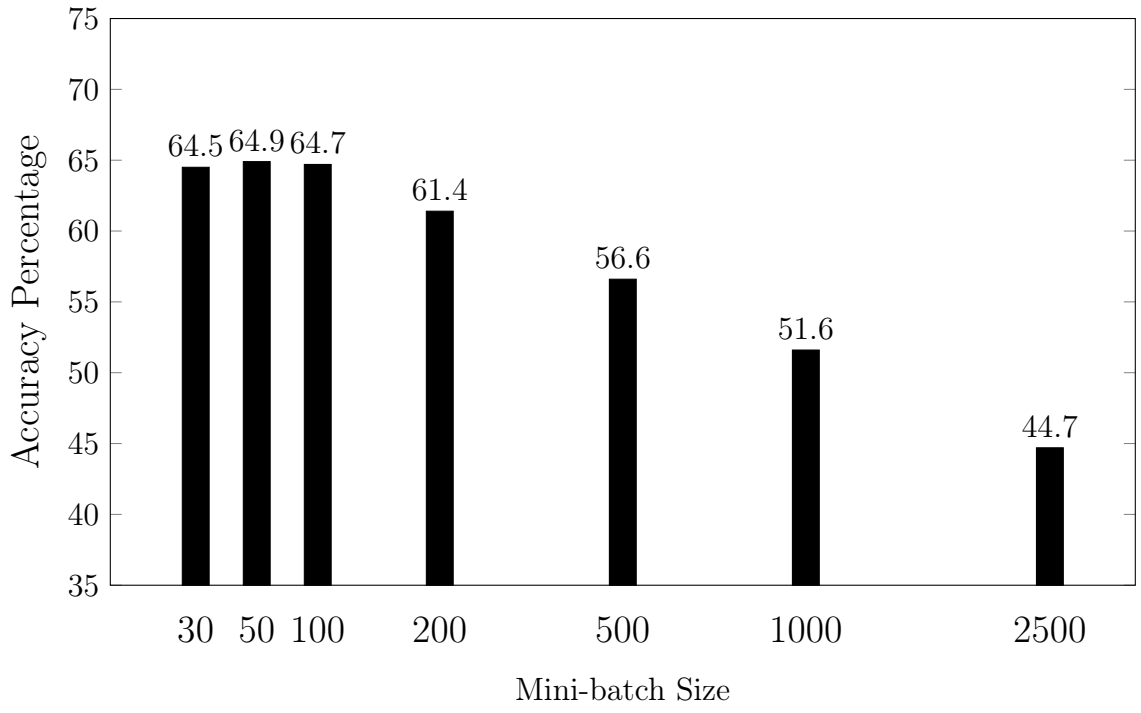

Figure 4.5: The detection accuracy as a function of mini-batch size.

Effect of Learning Rate Learning rate determines the size of the steps taken in each weight and bias update. Too low learning rate is a reason for slow convergence. Too high learning rate might result with overshooting the global minimum (see 3.3.1). As seen in Table 4.6, once a suitable range of learning rate is found, the network manages to converge to a high accuracy model. Learning rate is highly application-dependent, therefore a grid search over the possible values of it is usually necessary for different datasets.

Learning Rate	0.005	0.01	0.02	0.05	0.1	0.5	1.0
Accuracy	61.3	64.9	66.5	65.2	64.5	64.5	65.6

Table 4.6: The detection accuracy as a function of learning rate.

Effect of the Number of Neurons in Maxout Pooling The effect of the number of hidden neurons included in maxout pooling is experimented and the results are given in Table 4.7. When the number of pieces is increased, the maxout neuron learns the activation function with more expressive power, because there are more breakpoints in the approximated activation function curve for each neuron.

Number of Pieces	2	3	4	5
Accuracy	65.6	65.8	66.0	66.2

Table 4.7: The detection accuracy as a function of number of maxout pieces.

Effect of Momentum Momentum coefficient μ is spanned between 0.3 to 0.8 and the results are given in Figure 4.8. The overall detection accuracies are very close to each other, however when μ is higher, the network converges in less epochs.

Momentum	0.3	0.4	0.5	0.6	0.7	0.8
Accuracy	65.8	64.5	64.4	65.3	65.0	65.8
Termination epoch	91	102	92	92	79	79

Table 4.8: The detection accuracy as a function of number of momentum coefficient.

4.3.3 Context-wise Results

After examining the effects of several ANN parameters on the detection accuracy, the system configuration with highest accuracy is trained with three different feature sets.

For the MFCC feature set, 17 static MFCCs are extracted from 50 ms frames of audio data. The first MFCC is discarded since it gives the total log-energy of the frame and it is not informative.

For the Mel-band energy feature set, the frequency range of the recordings is divided to 40 Mel-bands and the energy inside each band is used as features. Another feature set is created by taking the logarithm of these Mel-band energies.

Accuracy in One-second Blocks The overall detection accuracies are calculated in one-second blocks for different features. The results are illustrated for each context in Figure 4.6.

All three ANN methods with different feature sets offer a considerable increase in accuracy over baseline method; which uses 16 MFCCs and their first and second time derivatives as features, a continuous density HMM with three state left-to-right topology for modeling feature distributions and GMM with 16 Gaussians to model the probability density function of observations in each state. We believe the success of ANN in this task is based on the high expressive power of ANNs on highly nonlinear functions. The polyphonic nature of the environmental audio makes it hard GMM-HMM systems to learn the correlations between sound event classes, however ANN deals with this problem in a significantly successful way.

When MFCCs are used as features, the overall accuracy in one-second blocks is 61%, which outperforms the baseline method by 16%. MFCC performance is 4% lower than Mel-band energy performance for this polyphonic environmental sound database. As also mentioned in [15], it can be argued that MFCCs are not able to capture the temporal domain signatures. These signatures are often seen in sound events resulting from natural sound sources, such as wind on trees and rain sound. When there are simultaneous sound events, these natural sound with broad flat spectrum are hard to recognize by using MFCCs as features.

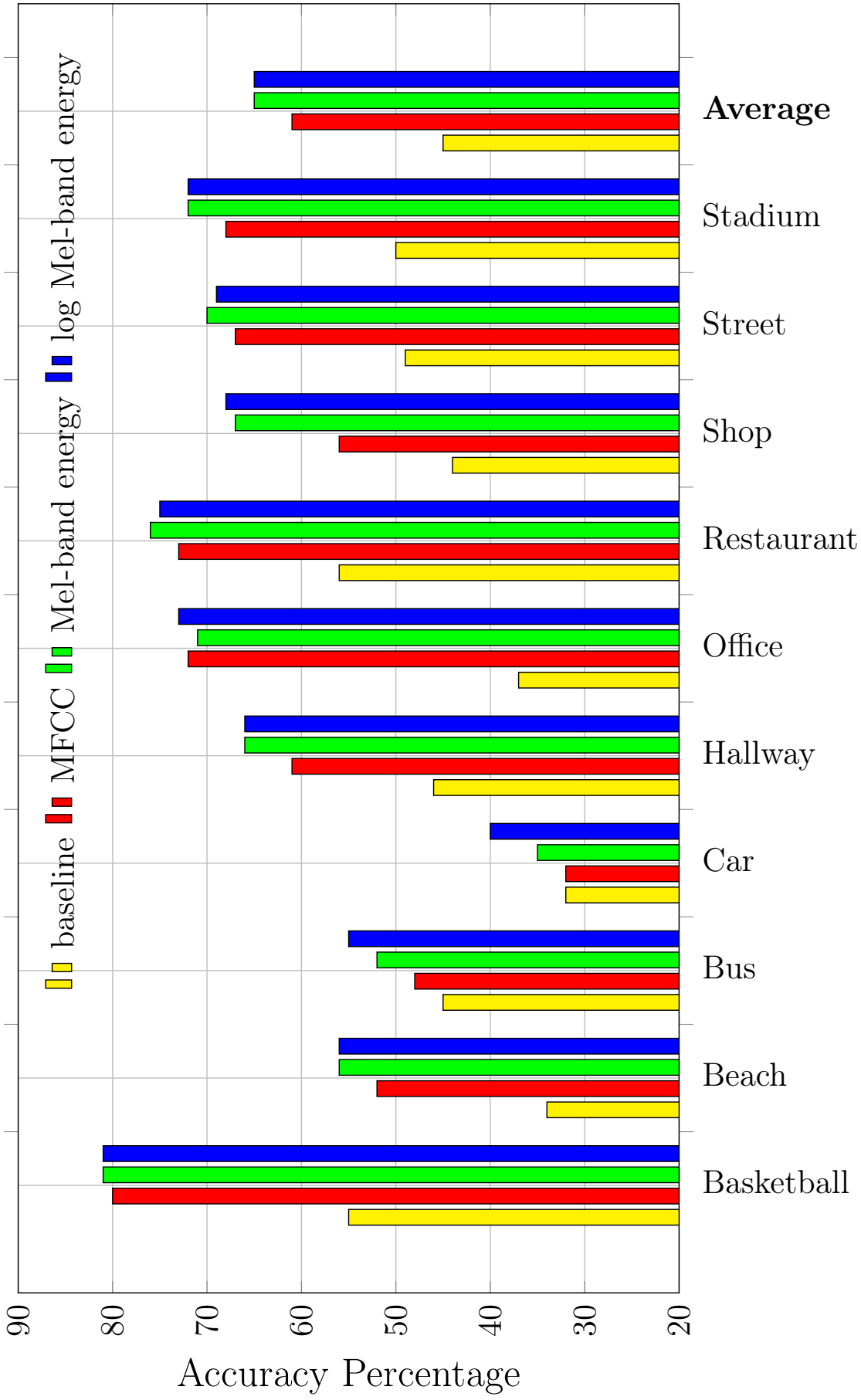

Figure 4.6: Context-wise detection accuracies (F1 score) for detection with MFCC, Mel-band energies and log Mel-band energies used as audio features.

Accuracy in Single Time Frames Another way of evaluating the detection accuracy is to calculate the F1 score inside each single time frame and averaging these values over the whole dataset. This brings the advantage of a higher time resolution compared to one-second block evaluation, but a considerably lower ($\approx 10\%$) decrease in detection accuracy. It can be said that ANN method is much better at detecting the sound events than locating them in the time domain. The results are illustrated for each context in Figure 4.7.

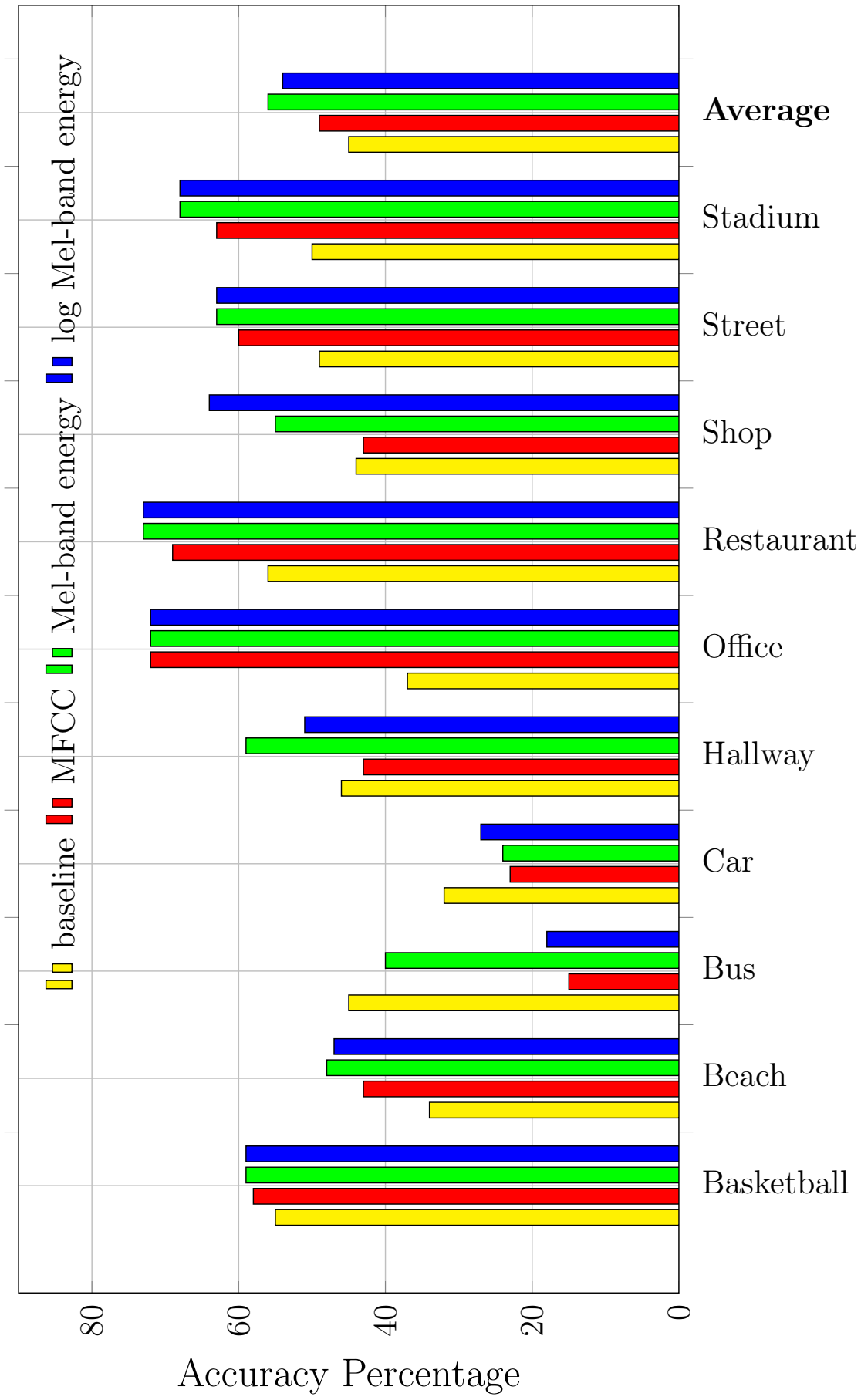

Figure 4.7: Context-wise detection accuracies (F1 score) for detection with MFCC, Mel-band energies and log Mel-band energies in single time frames.

5. CONCLUSIONS

In this thesis, we tackled the area of multilabel sound event detection in realistic environments. We proposed to use the spectral domain features such as MFCCs and Mel-band energies for audio data representation. For multilabel classification, ANNs with multiple hidden layers were proposed.

The experiments have been conducted on an audio database collected from real-life contexts such as inside a bus, in an office, in a restaurant etc. Each recording includes multiple overlapping sound events in almost all time instances, therefore the task was to detect individual sound events from a mixture of sound signals.

In order to find the ANN parameters for highest accuracy, many experiments have been conducted. The effect of several parameters on the detection accuracy have been experimented and explained. F1 score in one-second block was chosen as the evaluation metric during the tests. The highest overall accuracy is calculated as 65%. A comparison was made with the baseline system, which uses GMM-HMM classification method with MFCC features on the same database. We showed that multilabel ANNs provide a significant increase in accuracy (20% increase) over this baseline method.

From our experiments, we conclude that using Mel-band energies as audio features gives a slightly higher accuracy compared to MFCCs, which are the traditional features used in single label event detection and speech recognition. Our reasoning on this is the sum of MFCCs of individual sound events is not equal to MFCCs extracted from the mixture signal. We also conclude that multilabel classification with ANNs provide better results compared to GMM-HMM methods due to its high expressional power on nonlinear functions.

We noticed from the experiments that frame concatenation and sliding window methods can be really beneficial in polyphonic SED systems. We also noticed that the values of the learning parameters such as learning rate, mini-batch size and momentum coefficient can have significant importance in ANN learning. Finding a useful range of these parameters can significantly speed up the learning process and result in better accuracy.

For the future work, the performance can be increased by implementing regularization methods on ANN such as weight decay, L1-L2 penalties and dropout. We observed that when the number of hidden layers is increased, the model suffers from

overfitting. The ways of decreasing the overfitting for ANNs with more hidden layers can be investigated in the future. The post-processing method can be improved as well with the future research.

REFERENCES

[1] Biswas, J., & Veloso, M. (2012, May). Depth camera based indoor mobile robot localization and navigation. In IEEE International Conference on Robotics and Automation (pp. 1697-1702).

[2] Eronen, A. (2001). Automatic musical instrument recognition (Master's thesis). Tampere University of Technology, Finland.

[3] David, B., & Richard, G. (2004, June). Efficient musical instrument recognition on solo performance music using basic features. In Audio Engineering Society Conference: 25th International Conference: Metadata for Audio.

[4] Benetos, E., Dixon, S., Giannoulis, D., Kirchhoff, H., & Klapuri, A. (2012, October). Automatic Music Transcription: Breaking the Glass Ceiling. In International Society for Music Information Retrieval Conference (pp. 379-384).

[5] Dessein, A., Cont, A., & Lemaitre, G. (2010). Real-time polyphonic music transcription with non-negative matrix factorization and beta-divergence. In 11th International Society for Music Information Retrieval Conference (pp. 489-494).

[6] Guaus, E., & Herrera, P. (2006, October). Music genre categorization in humans and machines. In Audio Engineering Society Convention 121.

[7] Tzanetakis, G., & Cook, P. (2002). Musical genre classification of audio signals. In IEEE Transactions on Speech and Audio Processing, 10(5), pp. 293-302.

[8] Stuckless, R. (1994). Developments in real-time speech-to-text communication for people with impaired hearing. In M. Ross(Ed.), Communication access for people with hearing loss (pp.197-226). Baltimore, MD: York Press.

[9] Dahl, G. E., Yu, D., Deng, L., & Acero, A. (2012). Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. In IEEE Transactions on Audio, Speech, and Language Processing, 20(1), pp. 30-42.

[10] Reichl, W., & Ruske, G. (1995). Discriminative training for continuous speech recognition. In Proceedings of the European Conference on Speech Communication and Technology (pp. 537-540).

[11] Gemmeke, J. F., Virtanen, T., & Hurmalainen, A. (2011). Exemplar-based sparse representations for noise robust automatic speech recognition. In IEEE Transactions on Audio, Speech, and Language Processing, 19(7), pp. 2067-2080.

[12] Atrey, P. K., Maddage, M. C., & Kankanhalli, M. S. (2006). Audio based event detection for multimedia surveillance. In IEEE International Conference on Acoustics, Speech and Signal Processing, vol. 5, pp. V-V.

[13] Xu, M., Maddage, N. C., Xu, C., Kankanhalli, M., & Tian, Q. (2003, July). Creating audio keywords for event detection in soccer video. In IEEE 2003 International Conference on Multimedia and Expo, vol. 2, pp. II-281).

[14] Mesaros, A., Heittola, T., Eronen, A., & Virtanen, T. (2010, August). Acoustic event detection in real life recordings. In 18th European Signal Processing Conference (pp. 1267-1271).

[15] Chu, S., Narayanan, S., & Kuo, C. C. (2009). Environmental sound recognition with time-frequency audio features. In IEEE Transactions on Audio, Speech, and Language Processing, 17(6), pp. 1142-1158.

[16] Tran, H. D., & Li, H. (2011). Sound event recognition with probabilistic distance SVMs. In IEEE Transactions on Audio, Speech, and Language Processing, 19(6), pp. 1556-1568.

[17] Muda, Lindasalwa and K.M., Begam and Elamvazuthi, I (2010). Voice Recognition Algorithms using Mel Frequency Cepstral Coefficient (MFCC) and Dynamic Time Warping (DTW) Techniques. Journal of Computing, 2 (3), pp. 138-143. ISSN 2151-9617.

[18] Zbancioc, M., & Costin, M. (2003, July). Using neural networks and LPCC to improve speech recognition. International Symposium on Signals, Circuits and Systems, vol. 2, pp. 445-448.

[19] Nwe, T. L., Foo, S. W., & De Silva, L. C. (2003). Speech emotion recognition using hidden Markov models. Speech communication, 41(4), pp. 603-623.

[20] O. Gencoglu, T. Virtanen, H. Huttunen. (2014). "Recognition of Acoustic Events Using Deep Neural Networks," in European Signal Processing Conference (EUSIPCO 2014).

[21] Durbin, J. (1960). The fitting of time-series models. Revue de l'Institut International de Statistique, pp. 233-244.

[22] Allen, J. B. (1994). How do humans process and recognize speech?. In IEEE Transactions on Speech and Audio Processing, 2(4), pp. 567-577.

[23] Dennis, J., Tran, H. D., & Chng, E. S. (2013). Overlapping sound event recognition using local spectrogram features and the generalised hough transform. Pattern Recognition Letters, 34(9), pp. 1085-1093.

[24] T. Kohonen. State of the art in neural computing. In Proceedings, IEEE First International Conference on Neural Networks, pp 179-190, San Diego, June, 1987.

[25] S. Haykin, Neural Networks: A Comprehensive Foundation, 2nd Edition, Prentice-Hall, 1999.

[26] McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics, 5(4), pp. 115-133.

[27] Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. Psychological Review, 65(6), p. 386.

[28] Werbos, P. (1974). Beyond regression: New tools for prediction and analysis in the behavioral sciences. PhD thesis, Harvard University, 1974.

[29] Hinton, G., Osindero, S., & Teh, Y. W. (2006). A fast learning algorithm for deep belief nets. Neural computation, 18(7), pp. 1527-1554.

[30] Nair, V., & Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines. In Proceedings of the 27th International Conference on Machine Learning (pp. 807-814).

[31] Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. arXiv preprint arXiv:1207.0580.

[32] Glorot, X., Bordes, A., & Bengio, Y. (2011). Deep sparse rectifier networks. In Proceedings of the 14th International Conference on Artificial Intelligence and Statistics. JMLR W&CP ,vol. 15, pp. 315-323.

[33] Goodfellow, I., Warde-Farley, D., Mirza, M., Courville, A. and Bengio, Y. (2013). Maxout Networks. In International Conference on Machine Learning.

[34] Chapelle, O., Schölkopf, B., Zien, A. (2006). Semi-supervised learning. Cambridge: MIT press.

[35] Cybenko, G. (1989). Approximation by Superpositions of a Sigmoidal Function. Math. Control Signals Systems, vol. 2, pp. 303-314.

[36] Hornik, K., M. Stinchcombe, and H. White. Multilayer Feedforward Networks are Universal Approximators. Neural Networks, vol. 2, 1989, pp. 359-366.

[37] Fix, E., Hodges, J.L. Discriminatory analysis, nonparametric discrimination: Consistency properties. Technical Report 4, USAF School of Aviation Medicine, Randolph Field, Texas, 1951.

[38] Chen, S., Cowan, C. F., & Grant, P. M. (1991). Orthogonal least squares learning algorithm for radial basis function networks. In IEEE Transactions on Neural Networks, 2(2), pp. 302-309.

[39] Mikolov, T., Karafiat, M., Burget, L., Cernocky, J., & Khudanpur, S. (2010, September). Recurrent neural network based language model. In INTERSPEECH (pp. 1045-1048).

[40] Lukosevicius, M., & Jaeger, H. (2009). Reservoir computing approaches to recurrent neural network training. Computer Science Review, 3(3), pp. 127-149.

[41] Kohonen, T. (1990). The self-organizing map. Proceedings of the IEEE, 78(9), pp. 1464-1480.

[42] de Villiers, J., and E. Barnard. Backpropagation Neural Nets with One and Two Hidden Layers. IEEE Transactions on Neural Networks, vol. 4, 1992, pp. 136-141.

[43] Bengio, Y. (2012). Deep Learning of Representations for Unsupervised and Transfer Learning. In International Conference on Machine Learning (pp. 17-36).

[44] Braverman, M. (2011). Poly-logarithmic independence fools bounded-depth boolean circuits. Communications of the ACM, 54(4), pp. 108-115.

[45] Bengio, Y., and Delalleau, O. Shallow versus deep sum-product networks, 2011. The Learning Workshop, Fort Lauderdale, Florida.

[46] Wan, L., Zeiler, M., Zhang, S., Cun, Y. L., & Fergus, R. (2013). Regularization of neural networks using dropconnect. In Proceedings of the 30th International Conference on Machine Learning (pp. 1058-1066).

[47] Eronen, A. J., Peltonen, V. T., Tuomi, J. T., Klapuri, A. P., Fagerlund, S., Sorsa, T. & Huopaniemi, J. (2006). Audio-based context recognition. In IEEE Transactions on Audio, Speech, and Language Processing, 14(1), pp.321-329.

[48] Clarkson, B., Sawhney, N., & Pentland, A. (1998). Auditory context awareness via wearable computing. Energy, 400(600), p. 20.

[49] Baillie, M., & Jose, J. M. (2003). Audio-based event detection for sports video. In Image and Video Retrieval (pp. 300-309). Springer Berlin Heidelberg.

[50] Xiong, Z., Radhakrishnan, R., Divakaran, A., & Huang, T. S. (2003, April). Audio events detection based highlights extraction from baseball, golf and soccer games in a unified framework. In 2003 IEEE International Conference on Acoustics, Speech, and Signal Processing, vol. 5, p. 632.

[51] Raj, B., Virtanen, T., Chaudhuri, S., & Singh, R. (2010). Non-negative matrix factorization based compensation of music for automatic speech recognition. In INTERSPEECH (pp. 717-720).

[52] Heittola, T., Mesaros, A., Eronen, A., & Virtanen, T. (2013). Context-dependent sound event detection. EURASIP Journal on Audio, Speech, and Music Processing, 2013(1), pp. 1-13.

[53] Siahaan, E. (2013). Single and Multi-Label Environmental Sound Recognition with Gaussian Process (Master's thesis). National Central University, Taiwan.

[54] Heittola, T., Mesaros, A., Virtanen, T., & Eronen, A. (2011, September). Sound event detection in multisource environments using source separation. In Workshop on machine listening in Multisource Environments (pp. 36-40).

[55] Heittola, T., Mesaros, A., Virtanen, T., and Gabbouj, M. Supervised Model Training for Overlapping Sound Events Based on Unsupervised Source Separation, the 38th International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2013), pp. 8677-8681, Vancouver, Canada, 2013.

[56] Zhang, M. L., & Zhou, Z. H. (2006). Multilabel neural networks with applications to functional genomics and text categorization. In IEEE Transactions on Knowledge and Data Engineering, 18(10), pp. 1338-1351.

[57] Gencoglu, O. (2014). Acoustic Event Classification Using Deep Neural Networks (Master's thesis). Tampere University of Technology, Finland.

[58] Cakir, E., Virtanen, T., Huttunen, H. (2014). Polyphonic Sound Event Detection Using Multilabel Deep Neural Networks. In 2015 IEEE International Conference on Acoustics, Speech, and Signal Processing (submitted).

[59] Heittola, T., Mesaros, A., Eronen, A., & Virtanen, T. (2010). Audio context recognition using audio event histograms. In Proc. of the 18th European Signal Processing Conference (EUSIPCO 2010) (pp. 1272-1276).

[60] Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1988). Learning representations by back-propagating errors. Nature, 323.9, pp. 533-536.

[61] Kullback, S., & Leibler, R. A. (1951). On information and sufficiency. The Annals of Mathematical Statistics (pp. 79-86).

[62] Zhou, X., Zhuang, X., Liu, M., Tang, H., Hasegawa-Johnson, M., & Huang, T. (2008). HMM-based acoustic event detection with AdaBoost feature selection. In Multimodal Technologies for Perception of Humans (pp. 345-353). Springer Berlin Heidelberg.

[63] Zhuang, X., Zhou, X., Hasegawa-Johnson, M. A., & Huang, T. S. (2010). Real-world acoustic event detection. Pattern Recognition Letters, 31(12), pp. 1543-1551.

[64] Hinton, G. (2010). A practical guide to training restricted Boltzmann machines. Momentum, 9(1), p. 926.

[65] O. Lartillot and P. Toiviainen. A Matlab Toolbox for Musical Feature Extraction From Audio, In International Conference on Digital Audio Effects, Bordeaux, 2007.

[66] Ian J. Goodfellow, David Warde-Farley, Pascal Lamblin, Vincent Dumoulin, Mehdi Mirza, Razvan Pascanu, James Bergstra, Frederic Bastien, and Yoshua Bengio. "Pylearn2: a machine learning research library". arXiv preprint arXiv:1308.4214.

[67] Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., & Bengio, Y. (2010, June). Theano: a CPU and GPU math expression compiler. In Proceedings of the Python for scientific computing conference (SciPy), vol. 4, p. 3.