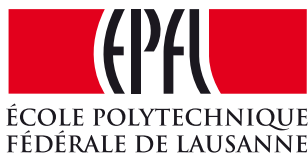


AUDIO EVENT DETECTION USING DEEP NEURAL NETWORKS

MASTER OF SCIENCE THESIS
École Polytechnique Fédérale de Lausanne



Author

Ashish Ranjan Jha
Master Student,
School of Computer Sciences,
École Polytechnique Fédérale de Lausanne
(Swiss Federal Institute of Technology, Lausanne)

Supervisors

Prof Pierre Dillenbourg (EPFL)
Dr. Fabien Cardinaux (Sony, Stuttgart)

Lausanne, EPFL, February 2016

Acknowledgements

I would like to thank my supervisor, Dr. Fabien Cardinaux for showing me a brilliant research path. His scientific approach, constant support and useful advice have helped me in my work and have also changed my perception towards academic research.

I would like to thank each and every member of the Speech and Sound Group at Sony especially Javier Alonso and Stefan Uhlich. They always helped me in staying motivated, gave guidance, and provided a fun research atmosphere. I would also like to thank Sony for giving me this opportunity of doing my thesis with their Stuttgart Technology Center, providing me with all their computational resources.

I would like to express my sincere gratitude to my academic supervisor, Prof. Pierre Dillenbourg for supervising my thesis. I am grateful for his recurrent support and relevant feedback. I would like to express my gratitude to EPFL, and all the professors for educating me computer science in all its breadth and depth.

Finally, I would like to thank my parents, Rani Jha and Bibhuti Bhushan Jha, and my sisters Nivedita, Sushmita and Shalini for the endless and selfless love and support they have given me over the years. Without their support, I could not come from India to Europe and pursue my dreams.

Lausanne, February 2016

Abstract

In a real-life audio recording, there are various audio events occurring either simultaneously or one at a time. These events may be anything ranging from traffic sounds to sounds of crying babies, adult female voice, etc. Human beings are highly efficient at detecting and distinguishing the various sounds they hear and till this date, no machine has reached anywhere close to the human accuracy for this task. Although the problem to detect every single audio event from a recording, where the events could potentially overlap, is a non-trivial problem. However, with the increasing size of audio datasets and the powerful machine learning models like the neural networks, it might soon be possible to reach the human accuracy. In this thesis, we have implemented neural networks as audio classifiers to detect single or multiple simultaneously occurring audio events in a given audio recording. We evaluate the performance of the neural networks and compare it with the performance of baseline models like gaussian mixture models or support vector machines. In a majority of experiments that we conducted, we find that the neural networks perform better than these models.

Key words: Audio event detection, neural networks, audio features, classifier

Contents

Acknowledgements	i
Abstract	iii
List of figures	vii
List of tables	ix
1 Introduction	1
2 Relevant Background	5
2.1 Audio Event Detection	5
2.2 Neural Networks	8
2.2.1 Deep Neural Networks	9
2.2.2 Convolutional Neural Networks	10
2.3 Related work	10
2.4 Datasets used	11
2.4.1 Dataset-I	12
2.4.2 Dataset-II	13
2.4.3 Dataset-III	15
3 Implemented Method	17
3.1 Overview	17
3.2 Classifiers	18
3.2.1 Baseline Models	18
3.2.1.1 k-Nearest Neighbors Approach	18
3.2.1.2 Support Vector Machines Approach	20
3.2.1.3 Gaussian Mixture Models Approach	22
3.2.2 Deep Neural Network Approach	23
3.2.2.1 Audio Features	23
3.2.2.2 Classification	25
3.2.3 Convolutional Neural Network Approach	25
3.2.3.1 Audio Features	26
3.2.3.2 Classification	26
3.3 Audio data augmentation experiment	30

Contents

4	Evaluation and Results	33
4.1	Overview	33
4.2	Dataset-I	34
4.2.1	Evaluation Procedure	34
4.2.2	Results	37
4.2.2.1	Baseline: k-Nearest Neighbors	37
4.2.2.2	Deep Neural Network	39
4.2.2.3	Convolutional Neural Network	41
4.3	Dataset-II	43
4.3.1	Evaluation Procedure	45
4.3.2	Results	47
4.3.2.1	Baseline: Support Vector Machine	47
4.3.2.2	Convolutional Neural Network	47
4.3.2.3	Standard baseline comparison	53
4.3.2.4	Data Augmentation	53
4.4	Dataset-III	57
4.4.1	Evaluation Procedure	58
4.4.2	Results	59
4.4.2.1	Results with method 1	59
4.4.2.2	Results with method 2	60
4.4.2.3	Standard baseline comparison	63
5	Conclusion	65
	References	70

List of Figures

2.1 MFCC Extraction Flowchart	7
2.2 Schematic representation of a neural network	8
2.3 Sample annotation for dataset-II	14
3.1 Audio Features Extraction Schematic for kNN model	19
3.2 Feature Extraction Schematic for SVM model	21
3.3 Audio Classification Framework	23
3.4 Audio Feature Extraction Schematic - DNN	24
3.5 DNN Network Architecture	24
3.6 MFCC matrix as an image for CNN	25
3.7 CNN Network Architecture	27
3.8 CNN model architecture for dataset-II	28
3.9 CNN model architecture for dataset-III	29
4.1 Zero-one error evaluation schematic	35
4.2 Evaluation by averaging the frame-wise predictions	36
4.3 Feature selection process	37
4.4 kNN - Confusion Matrix	38
4.5 DNN - Learning Curve	40
4.6 DNN - Confusion Matrix	41
4.7 CNN - Learning Curve	43
4.8 CNN - Confusion Matrix	44
4.9 Overall results for dataset-I	45
4.10 True positive rate and false positive rate	46
4.11 ROC Curve and Equal Error Rate	46
4.12 Weighted Binary Cross-Entropy	48
4.13 <i>Crowd</i> model- Learning Curve and EER Curve	50
4.14 <i>Crowd</i> model: 50-fold cross validation results	50
4.15 <i>Traffic</i> model- Learning Curve and EER Curve	50
4.16 <i>Traffic</i> model: 50-fold cross validation results	51
4.17 <i>Applause</i> model- Learning Curve and EER Curve	51
4.18 <i>Applause</i> model: 50-fold cross validation results	51
4.19 <i>Music</i> model- Learning Curve and EER Curve	52

List of Figures

4.20 <i>Music</i> model- EER curve for 5000 epochs	52
4.21 <i>Music</i> model: 50-fold cross validation results	52
4.22 EER curves - data-augmentation for <i>Crowd</i> class	55
4.23 EER curves - data-augmentation for <i>Traffic</i> class	55
4.24 EER curves - data-augmentation for <i>Applause</i> class	56
4.25 EER curves - data-augmentation for <i>Music</i> class	56
4.26 Overall results - data augmentation	57
4.27 Area Under ROC Curve	58
4.28 GMM Results	60
4.29 CNN architecture for dataset-III	62
4.30 GMM and CNN result comparison with evaluation method 2	63

List of Tables

2.1	Typical audio features	7
2.2	Urban8K dataset	12
2.3	Freesound dataset	13
2.4	CHiME dataset - audio classes	15
3.1	Parameters for MFCC extraction	19
3.2	Parameters for MFCC extraction for SVM for dataset-II	21
3.3	Parameters for MFCC extraction for dataset-III	22
3.4	Parameters for MFCC extraction for CNN for dataset-II	26
4.1	Classifiers and evaluation procedures for the three datasets	33
4.2	Urban8K dataset - review	34
4.3	DNN Hyperparamters - Dataset I	39
4.4	CNN Hyperparamters - Dataset I	42
4.5	Freesound dataset - review	45
4.6	SVM Results - Dataset II	47
4.7	CNN Hyper-parameters - Dataset II	48
4.8	Results comparison: CNN and SVM	53
4.9	Standard baseline comparison: CNN and SVM	53
4.10	Hyper-paramter setting: CNN with data augmentation	54
4.11	CHiME dataset - audio classes (review)	57
4.12	Hyper-parameter grid for CNN model	61
4.13	CNN results in comparison with the best GMM results	61
4.14	GMM results with evaluation method 2	62
4.15	Standard baseline comparison: CNN and SVM	63

1 Introduction

With the rapid advancements in the field of electronics and computer science, computers are nowadays expected to understand and be aware of the world around us. A lot of research has been done on electronic devices that can perceive the context and guide the user of the device based on the context. Navigation assistance, for example is such an application that is already a part of our daily lives, but it provides a limited context-aware information. Humans usually perceive their surroundings by combining the audio and visual information. However, most of the context-recognizing devices of today are based only on the visual information, which can be possibly sufficient to provide the required contextual information, but is prone to physical challenges. For example, in an indoor robot navigation system based on camera [1], even a slight loss of sight could lead to major errors in context recognition.

Audio information, on the other hand, can also provide additional contextual information. However, machines still perform far worse than humans when it comes to recognizing the context using audio information. In recent years though, with the increase in huge amounts of available audio data and computational power, extensive research has been, and is being carried out under the branch of computer science known as audio information retrieval (AIR)[2]. AIR's sub-branches, namely speech recognition, speaker identification, acoustic scene classification, audio event detection, etc. involve various machine learning models to give a desired performance. These machine learning models used for audio data have a huge potential of research and many of these are already used and have produced promising results.

Audio event detection (AED), as stated above, is one of the several sub-fields of AIR. There are various interpretations of the concept of an audio event in the literature. In [3], audio events are the ambient sounds in nature, winds on trees, sound of rain, etc. On contrary, in [4], audio events are all the events that are not solely based on music or speech signals. [5] presents a taxonomy defining various audio events. Hence, audio events are not precisely and uniquely defined in the literature and it can be said that the choice of the kind of audio events depends on the problem statement or the application area that is being tackled.

When the audio event detection task is performed using a machine learning model, the audio

data has to be processed and converted into audio features. In fact, getting good results from a machine learning model is largely based on the selection of input features. In AED, traditionally used features are the mel-frequency cepstral coefficients(MFCC), linear predictive cepstral coefficients (LPCC) and log-frequency power coefficients (LFPC). We use MFCCs as the features in our work because it has been most extensively used and has produced the best of results in AED literature. Also, the most used machine learning models in this field are the Gaussian Mixture Models (GMM) and the Hidden Markov Models (HMM). Recently, however, neural networks also have been used and have outperformed GMMs and HMMs. We also use neural networks as our machine learning model.

Neural networks are strong non-linear machine learning models composed of several layers, each layer containing several units. Due to their ability to analyze complex, multi-dimensional data, they are useful for complicated systems which are difficult to be expressed in compact mathematical formulas. Moreover, as the number of hidden layers increase, the network becomes deeper. Usually, training a deep neural network (DNN) is not as trivial as a shallow network. An additional unsupervised pre-training stage could sometimes be required for deep networks to be successful, for example in automatic speech recognition (ASR) [6]. Deep networks are used in a variety of machine learning areas such as image classification [7], natural language processing [8], feature learning [9], etc. and have given promising results. This is indeed a motivation for using DNN in areas such as audio events detection and classification. Another class of neural networks known as convolutional neural network (CNN) are inspired by the way the human visualization system works. CNNs significantly reduce the number of parameters of the machine learning model and capture local patterns from the data with shift invariance. CNNs are mostly used with image data, however they have also been used with audio data for tasks such as speech recognition [10], music genre classification [11], etc. In our work, we have designed and experimented with a DNN and a CNN model, for our problem.

In this thesis, we present an approach for tackling the problem of audio events detection using neural networks. We also present some results and insights that we obtained by applying neural networks on three different kinds of audio datasets. The audio datasets are basically set of audio recordings which are labeled with one or many of the audio events from a pre-defined set of audio events. Also, for one of the datasets, we artificially augment the audio data i.e. generate new data from the existing data to improve the machine learning model performance.

This thesis is organized as follows. Chapter 2 briefly reviews the relevant background on audio event detection, neural networks, related work in audio event detection, and finally an overview of the three different audio datasets used for the experiments. Chapter 3 discusses the different machine learning models that we implemented including the neural networks. This chapter describes all the machine learning steps including pre-processing, feature extraction, data division, network structure used and finally the network training methodology applied. This chapter also describes the artificial audio data augmentation experiments performed on one of the audio datasets. Chapter 4 presents the results obtained in our work and the

procedure followed for obtaining the results as well as the analysis of results. Finally, we conclude our findings under the conclusions chapter followed by references.

2 Relevant Background

In this chapter we provide a brief overview of various concepts and useful information related to the thesis. In the following we first discuss what audio event detection is, followed by a brief description of neural networks in the machine learning context. Then, we discuss the related work on audio event detection. We conclude this chapter by describing the audio datasets (set of recordings) that were used in the thesis.

2.1 Audio Event Detection

An audio event detection system performs two functions: (i) segmenting, and (ii) recognizing particular events in continuous audio recordings. In practice, the segmentation step is usually bypassed by breaking down the audio recording into audio clips/segments/chunks of a reasonable size (e.g. 4 seconds) and then assigning labels corresponding to the events to each chunk. This is similar to discretizing a continuous or analog signal into bits and assigning values to those bits such that the discretized signal closely resembles the original continuous signal.

For function (ii) i.e. recognizing particular events, most of the AED research till date is done with isolated sound events [12], where there is a single sound event in each recording and the audio event detection system is trained to detect a single event occurring in a given time period. Such systems with one audio event class per recording are called *monophonic*. If a single recording has more than one audio event, potentially overlapping with each other, then the audio event detection task is called *polyphonic*. In this thesis, we have worked on both monophonic and polyphonic systems.

The entire process of audio event detection is carried out in various steps: pre-processing, feature-extraction, training and testing, which are discussed below:

1. **Pre-processing** - Input audio data is pre-processed before being fed to the next stage i.e. feature extraction. General pre-processing operations include normalization, transfor-

mation, trimming, windowing, smoothing, etc. For e.g. in speech recognition systems, end-point detection [13] is a common pre-processing step.

2. **Feature Extraction** - Features are what is actually fed to the learning model as the input. Features are higher level representations compared to raw data, like frequencies instead of a raw temporal sample when using MFCCs for audio event detection. Features should be selected from the data in such a way that they contain enough information to properly represent the similarities between inter-class observations and variations between intra-class observations. In our work, MFCCs are the features that we extract from the audio recordings.
3. **Training** - Training phase in a machine learning model refers to the process of learning from labeled data. Audio event labels in a dataset are the audio event classes such as “Siren”, “Dog Barking”, “Gun Shot”, etc. that are annotated for each of the audio recordings. MFCC features are the inputs and labels are the outputs during the training of the machine learning model. Details of the DNN and CNN training procedures that were used in this thesis are presented in chapter 3. A further detailed explanation about training of neural networks and machine learning models in general, can be found in [14].
4. **Testing** - Once the model is trained, classification has to be performed on the testing data i.e. the unseen data. The testing data is completely isolated from the training data and represents the observations unseen to the system. The model’s performance is judged by evaluations done in this classification stage. Usually, in machine learning, the raw dataset is first pre-processed into usable features and then the transformed dataset is split into the training and testing sets in a ratio such that most of the data goes in training (e.g. training: 80%, testing: 20%).

However, for audio event detection, the train-test split cannot be performed in this manner. For example, if the task involves an event - “washing-machine sound”, then the dataset (having say, N number of recordings) might have sounds from different washing machines. If we simply break-down these recordings into say, M chunks and split these M chunks into training and testing sets, the training and testing sets might then contain chunks belonging to the same parent recording (i.e. the same washing-machine sound), which will make the machine learning system biased and we will obtain abnormally high classification accuracy. Hence, in audio event detection systems, the training and testing split happens on the raw dataset, and then these individual (training and testing) sets are pre-processed, so that all chunks of a parent recording go entirely either to the training set or to the testing set.

There are quite a number of audio features that can be extracted from the audio recordings. The features are primarily spectral, spatial, compressive or low-level features. Table 2.1 gives an overview of some commonly used features in audio tasks.

Table 2.1 – Typical audio features

Feature name	Abbreviation	Category
Mel Frequency Cepstrum Coefficients	MFCC	Spectral
Gammatone Frequency Cepstrum Coefficients	GFCC	Spectral
Zero Crossing Rate	ZCR	Low-level descriptor
Low Energy Rate	LER	Low-level descriptor
Spectral Centroid	SC	Low-level descriptor
Spectral roll-off	SRO	Low-level descriptor
Inter-aural time difference	ITD	Spatial
Inter-aural level difference	ILD	Spatial
Frequency band energy	FBE	Spectral
Voicing	N/A	Spectral
Linear Predictive Coding	LPC	Compressive

The spectral features indicate how much of the input signal is found inside a certain frequency bands over a range of frequencies. Discrete Fourier Transforms (DFT) are used to convert temporal information to spectral by taking small windows or frames (10-100 milliseconds) of the audio signals and assuming the audio signal is stationary in that window.

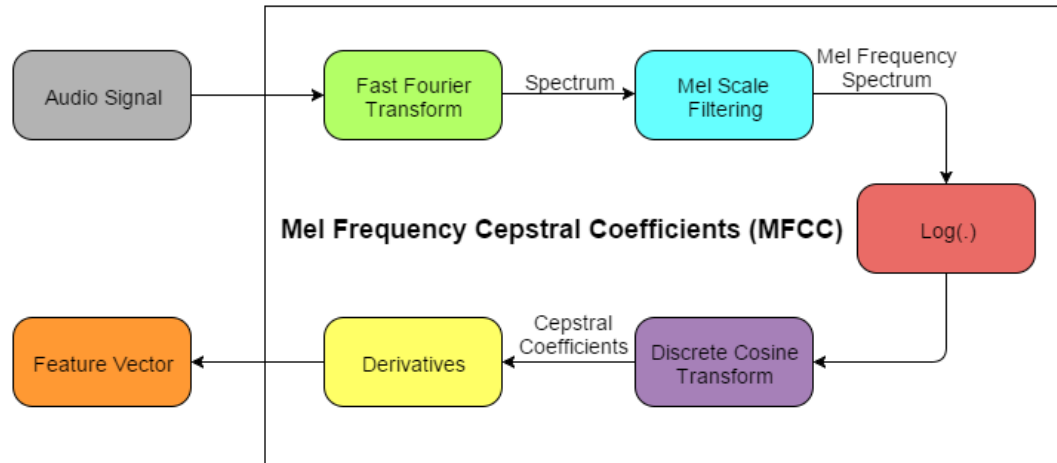


Figure 2.1 – MFCC Extraction Flowchart

We have used MFCCs as the audio features. They are widely used in audio event recognition systems and are extremely effective in single source environments. The concept of MFCC is inspired by the way the human auditory system works. These features are calculated by taking the log of mel-band energies and applying Discrete Cosine Transform (DCT). Logarithm is used for non-linearity because humans do not perceive loudness in audio in a linear scale. The MFCC extraction process is illustrated in figure 2.1. In the figure, it can be seen that after obtaining cepstral coefficients from the *Discrete Cosine Transformation* step, the coefficients are passed through an additional *Derivatives* step. This step provides us the final feature vector which includes the MFCCs, Delta MFCCs and Delta-Delta MFCCs. The Delta and Delta-Delta

MFCCs are respectively the discrete first-order and second-order derivatives of the MFCCs. A detailed description of the MFCC extraction process can be found in [15].

The most widely used classifiers in audio event detection are the GMM[13] and HMM[1]. Neural networks are now being increasingly used as well. We also use neural networks as our machine learning model for the audio event detection task and we discuss more about it in the next section.

2.2 Neural Networks

Artificial neural networks are machine learning models which are motivated by the way human brain works. Neural networks are defined in the literature 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" [16] [17]. Human brain has strong interpretation powers largely attributed to the structure composed of billions of interconnected neurons. Each neuron is a unit in the whole process and does the task of learning information.

For example, consider the task of handwriting recognition. If one was to create a computer program for this task by modeling individual characters and words, it would involve a huge number of permutations and combinations because of the large variety of natural handwritings. But, with the approach of neural networks, we basically try to imitate the way, the human brain would learn and understand the handwritten text.

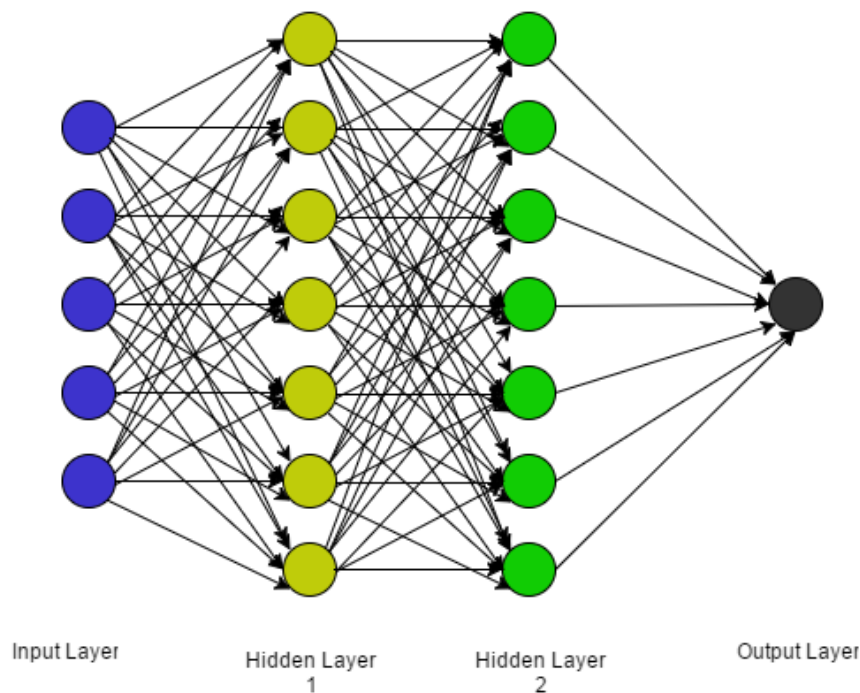


Figure 2.2 – Schematic representation of a neural network

A neural network is composed of a network of layers, where the first and last layers are called, the input and output layers respectively and the intermediate layers are known as the hidden layers. Figure 2.2 shows a schematic diagram of a neural network with an input layer, an output layer and two hidden layers. Each layer consists of neurons, which are the units of the network, represented by circles in the figure. The neurons of each layer are connected with the neurons of the preceding and succeeding layers and these connections are called the weights of the neural network. These weights are the actual parameters of the network that need to be fitted during the training process. The layers might (as in the figure) or not be fully connected to each other depending on the network architecture. During training, the input features are fed to the input layer of the network. A cost function is formulated as a function of the output of the neural network and the desired target output. The optimal set of parameters (or weights) are obtained by running an optimization algorithm such as stochastic gradient descent [18], that minimizes this cost function.

Neural networks have been invented in times as early as the 1940's. After a rising research till the 1970's, the interest on this topic subdued mainly because of the computational requirements that were much higher than the available technology of the time. In the 2000's the suppressed research interest started restoring with the improvements in computational technology and inventions of new optimized algorithms for neural networks.

To this date, there have been a variety of neural networks designed, experimented and tested in the literature. These neural networks differ in their topology and their learning algorithms. In the following, we discuss two such types of neural networks which we have used in the thesis.

2.2.1 Deep Neural Networks

DNNs are neural networks with two or more hidden layers [19] [20]. The motivation behind using a high number of hidden layers is to find more abstract features or concepts in the higher levels, defined in terms of the lower level features. These high-level features would then help to separate the independent distributions in the training data. A shallow network having a large number of neurons in its limited number of layers would rather be computationally exhaustive because for the same number of hidden layer units (neurons), a deep network is exponentially more efficient than a shallow network [21].

Another motivation in using DNNs comes from the analogy drawn with the human brain. Studies have shown that even for a simple visual object recognition task, human brain uses 10 layers of biological neurons. Intuitively, humans divide the problem into several layers of execution. Likewise, deep architectures represent the concepts one at a level and in the end make a composition of lower level concepts to get the final representation.

Training DNNs with the usual back-propagation algorithm [22] might not lead us to the global optimum solution or even a good local minimum point because of a high-dimensional

feature space created by a large number of layers. Although the latest research shows that deeper networks are lesser prone to a local minimum solution but still, deep networks may get stuck to a local minimum if the weights are initialized randomly. The introduction of a new unsupervised pre-training technique using auto-encoders [23] solved the initialization problem efficiently, where the layers (except the output layer) are trained greedily one by one in an unsupervised manner, then the whole network with pre-learned weights is trained as usual. Due to the great success of auto-encoders, we use the idea of pre-training deep neural network using auto-encoders, as well in our work.

2.2.2 Convolutional Neural Networks

CNNs are another type of neural networks that are widely used with images. These networks are inspired by the way the human visual mechanism works. It has been proven that living organisms tend to focus at a local area of the visual space at once, using a complex arrangement of visual cortex cells. These local areas or regions of the visual field are known as receptive fields and they are tiled to cover the entire visual field. The motivation behind using CNN is to exploit the spatially local correlation in 2D images. The architecture of CNN makes use of the local connections and tied weights followed by a pooling step which provides translation/shift invariant features. And, compared to DNNs, CNNs have much fewer parameters for the same number of hidden units which makes them faster and easier to train.

As the name suggests CNNs mandatorily have at least one convolutional layer. This layer has parameters that consist of a set of kernels. Kernels are basically filters that convolve around the two dimensions of the input image. These kernels produce feature maps from the input which consists of representations from the smaller parts of the input image. Also, there is usually one (or more) pooling layer(s) in CNNs where the input image (or feature map) is partitioned into a set of non-overlapping regions (rectangles) and for each such region, the pooling layer outputs one value. This value can be the maximum for a max-pooling layer, minimum for min-pooling, mean for mean-pooling and so on. Before the output layer, there is usually at least one fully-connected (hidden) layer in a CNN which connects all the convolved and pooled representations to the final output. CNNs are usually used with images, however we have tried to use it with audio data by interpreting an audio recording as an image (of audio frames). A detailed explanation about CNNs can be found in [24]

2.3 Related work

In the field of audio information retrieval, although audio event recognition has not been researched as extensively as speech and music recognition, several different methods have been proposed for audio event detection and classification in realistic environments. MFCCs and mel-bands are the most used features and GMM and HMM are the most used machine learning models in the existing literature work. Research for better features and better classification methods is in progress.

[25] presents a method for detection of overlapping sound events where the separated components assigned to individual event classes are learned as a non-negative dictionary in a coupled matrix factorization problem. [26] uses MFCC features and trains a Hidden Markov Model (HMM) classifier with consecutive passes of Viterbi algorithm. In [27], non-negative matrix factorization is used as a pre-processing step to decompose the audio signals into streams and then detect the most prominent event in each stream, but the maximum number of different overlapping events are assumed to be known a-priori. In [28], the spectrogram features are combined with GHT (Generalized Hough Transform) voting system to detect the overlapping sound events.

[29] presents a system for acoustic event detection from real life environments using a network of Hidden Markov Models. [30] gives an unsupervised approach for detection of human scream vocalizations from continuous recordings in noisy acoustic environments. [31] describes a method to extract time-frequency (TF) audio features by tensor-based sparse approximation for sound effects classification. [32] presents a biologically inspired model for sound event classification which combines spike coding with a spiking neural network (SNN). [12] proposes a deep neural network for recognizing isolated acoustic events such as footsteps, baby crying, rain, etc. [33] employs a deep neural network with a novel pre-training methodology for audio event classification. Similar to this, we also implement a deep neural network with an auto-encoder based pre-training process for the audio event detection task on one of our datasets.

2.4 Datasets used

The audio event detection task is subjective to the application, and to the kind of audio recordings (data) available in the dataset. There is no single uniquely defined taxonomy of sound events for which this task is to be performed. Different applications may demand different kinds of sound events, which in turn should involve different kinds of audio recordings. For example, if the application of the audio event detection task is to detect and identify “washing-machine sounds”, then it would be more meaningful to have audio recordings that contain sounds of washing-machines. Hence, the choice of the audio dataset (set of labeled audio recordings) is largely determined by the application we are aiming for.

Another factor that could determine the kind of audio dataset that is desired for the audio event detection task is the nature of the task, i.e. whether the task is a monophonic audio event detection task or a polyphonic audio event detection task. The sole difference between the two is that the latter considers the possibility of multiple audio events occurring **simultaneously**, whereas the former does not. The polyphonic detection task is reasonably closer to reality, but is usually more difficult to tackle as compared to a monophonic task.

For the thesis, the focus is not on any particular application but instead, on developing and applying neural networks on audio event detection tasks - whether monophonic or polyphonic. In the course of this work, we used three different kinds of dataset. First, we worked with a

monophonic audio dataset, i.e. each audio recording in this dataset was labeled with just one audio class label (no multiple simultaneously occurring events). After implementing and testing neural network models on this dataset, we experimented with the second dataset, which is a polyphonic audio dataset. Using the experience gained from dataset-I, we again implemented a neural network model for the second dataset. Finally, we worked on the third dataset, which is also a polyphonic audio dataset but has a more precise application i.e. the audio event classes in this dataset are much more specific as compared to the broadly defined audio event classes of dataset-II. Dataset-II is meant for outdoor events like *crowd*, *traffic*, *etc.*, whereas dataset-III is meant for a domestic, home environment. In the following, the relevant details and important characteristics of the three datasets are discussed.

2.4.1 Dataset-I

The first dataset that we used is a publicly available dataset known as the Urban8K dataset¹. As the name suggests this dataset has audio recordings from urban settlements. This dataset has audio classes such as *car horn*, *air conditioner*, *gun shot*, *etc.* that represent an urban context. We chose this dataset for our experiments for various reasons. Firstly, each audio recording in this dataset is labeled with just one audio event class label. Hence, this dataset is meant for a monophonic audio event detection system. Secondly, the recordings are short, with a maximum duration of four seconds. There is actually a trade-off between the (i)annotation effort, and (ii)loss of information in keeping the labeled recordings too short or too long respectively. Four seconds, which is the duration of audio recordings in this dataset, is a fair compromise between the two opposing factors. Lastly, this dataset has already been tested on some of the standard classifiers in the literature and hence we have some baseline results on this dataset to compare to.

There are 10 different audio event classes that are listed in table 2.2. Each audio recording

¹<https://serv.cusp.nyu.edu/projects/urbansounddataset/urbansound8k.html>

Table 2.2 – Urban8K dataset

Class	ID
air conditioner	0
car horn	1
children playing	2
dog bark	3
drilling	4
engine idling	5
gun shot	6
jackhammer	7
siren	8
street music	9

is labeled with one and only one of these audio event classes. The 10 different audio events are distributed (duration-wise) almost uniformly in the dataset. This means that a random classifier would detect the audio events with 10% classification accuracy on this dataset. The baseline results along with the further details of the dataset can be found in [5].

2.4.2 Dataset-II

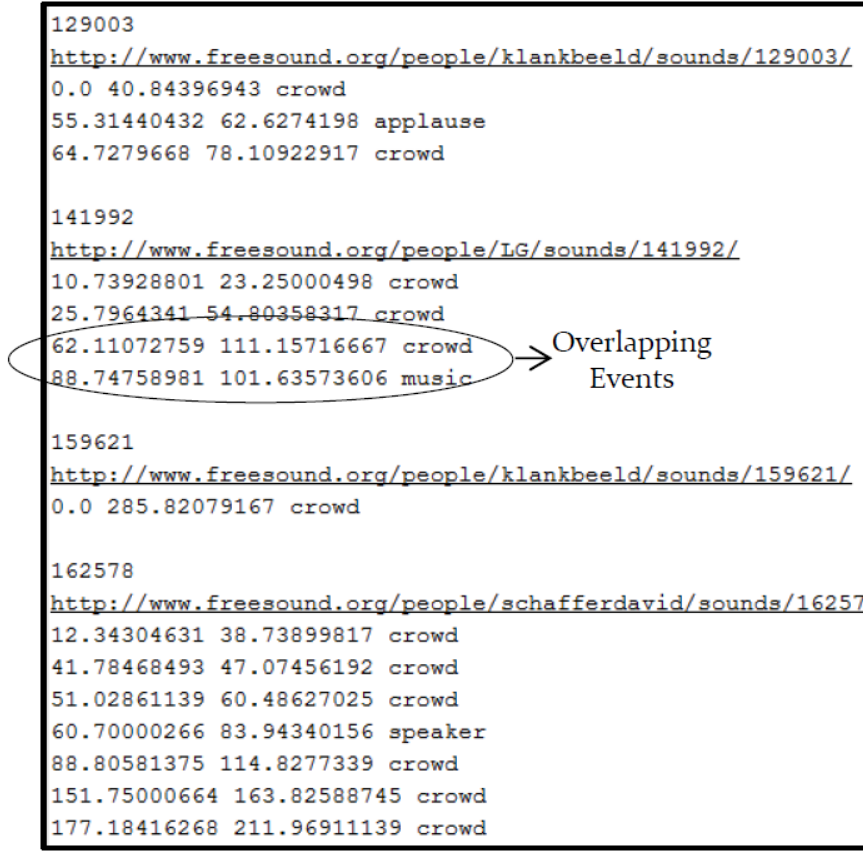
This dataset is a collection of audio recordings from *freesound*² website, which is a repository of audio files uploaded by users and covers a wide range of acoustic events. The annotations for these recordings are obtained from [33]. In this dataset, there are simultaneous occurrences of various audio events. Hence, this dataset is a polyphonic audio event dataset. The annotations involve 5 kinds of audio event classes. Wherever there are no annotations, we label that portion of the audio recordings as the sixth audio event - *others*. Thus, there are six audio event classes in total. The six classes along with their total time-share in the recordings are shown in table 2.3. A sample of the annotations is also provided in figure 2.3. The first line for each of the annotation blocks in the figure represents the audio file id, which is the primary key for identifying different audio files on the *freesound* website's database. The next line provides the link to download the audio file. Following that are the actual annotations with the first and second columns representing the start and end times (in seconds) of the audio event classes. The names of the respective audio event classes are provided in the third column.

Table 2.3 – Freesound dataset

Audio Class	Length (in minutes)
Crowd	45.625
Applause	26.958
Music	27.625
Traffic	34.375
Speaker	18.625
Others	53.083

Originally, as described in [33], this dataset had 84 audio recordings (228 minutes of data). But, we could procure only 69 (182 minutes) of those audio files because the other 15 were taken off from the website. Among the 15 lost files, as observed from the available annotations, most of the content belonged to the *music* class. More precisely, 20 out of the 84 audio recordings had *music* as an audio event annotated in them. Among these 20 recordings, 6 of them were almost entirely *music* sounds and all 6 of these were among the recordings that could not be procured. Hence, there was a significant loss of data belonging to the *music* class. In our experiments, to compensate for this loss, we therefore also tried adding 6 randomly chosen musical recordings from the *freesound* website with almost same total duration of data as was lost. For completeness, we present our results both with and without the extra *music* files.

²www.freesound.org



129003	http://www.freesound.org/people/klankbeeld/sounds/129003/	0.0 40.84396943 crowd
55.31440432	62.6274198	applause
64.7279668	78.10922917	crowd
141992	http://www.freesound.org/people/LG/sounds/141992/	10.73928801 23.25000498 crowd
25.7964341	54.80358317	crowd
62.11072759	111.15716667	crowd
88.74758981	101.63573606	music
159621	http://www.freesound.org/people/klankbeeld/sounds/159621/	0.0 285.82079167 crowd
162578	http://www.freesound.org/people/schafferdauid/sounds/162578/	12.34304631 38.73899817 crowd
41.78468493	47.07456192	crowd
51.02861139	60.48627025	crowd
60.70000266	83.94340156	speaker
88.80581375	114.8277339	crowd
151.75000664	163.82588745	crowd
177.18416268	211.96911139	crowd

Figure 2.3 – Sample annotation for dataset-II

The 69 recordings that we procured were of variable duration. Following the data pre-processing step used in [33], we sliced out audio segments of 5 seconds with an overlap of 2.5 seconds (between segments), from these recordings. This gave us approximately 4200 audio segments. Each of these segments were then treated as a training data instance for our machine learning models. The audio event labels for these segments were decided on the basis of the following criterion- in a given audio segment of 5 seconds, whichever audio event classes are labeled in the annotations within that 5 second duration, are to be considered as the audio event class labels for that audio segment. For example if there is a segment that begins from 0 seconds and ends at 5 seconds in the parent recording. And if in the annotations, we have *crowd* annotated from 2 seconds to 4 seconds, *applause* annotated from 6 seconds to 12 seconds, and *traffic* annotated from 4 seconds to 8 seconds, then the audio segment will have the following labeled classes: *crowd* and *traffic*, because both of them are (partly or wholly) present in that segment.

Despite of having the 6 audio event classes originally, we did all our experiments with just 4 audio event classes: (i)Crowd, (ii)Applause, (iii)Music, and (iv)Traffic. This is because the baseline results for this dataset published in [33] report the results for only these 4 classes. Hence, to make a valid comparison, we report our results as well for just these 4 classes.

Further details about the dataset can be found in [33]

2.4.3 Dataset-III

The third and last dataset we worked on is another publicly available dataset, known as the CHiME-Home³(Computational Hearing in Multisource Environments) dataset. As the name suggests, this dataset has sounds coming from multiple sources, hence it is a polyphonic audio events dataset. The recordings are confined to a home environment and in fact, all the recordings of this dataset are done at the same place inside a house.

Table 2.4 – CHiME dataset - audio classes

Label	Description
c	Child speech
m	Adult male speech
f	Adult female speech
v	Video game/TV
p	Percussive sounds, e.g. crash, bang, knock, footsteps
b	Broadband noise, e.g. household appliances
o	Other identifiable sounds

This dataset has a total of 6.8 hours of audio recordings with 7 possible audio event classes as listed in table 2.4. Similar to the previous two datasets, each audio recording in the dataset is of fixed length (4 seconds). Actually, the original recordings are sliced down to these fixed length excerpts (non-overlapping) and there are 1946 such audio excerpts in the dataset that we use for our experiments. The audio event class label for each audio excerpt is a set of the audio event classes. For example, the labels may be something like {c,m} or {c,f,v}, or {o}, etc. More details about the dataset can be found at [34] and [35]. The baseline results for this dataset are available at [35].

³<https://archive.org/details/chime-home>

3 Implemented Method

In this chapter we discuss the machine learning models that we implemented for performing the task of audio event detection. In the following, we first present a brief overview of the different approaches that we followed. Then we describe the various models or classifiers that we designed to perform audio event detection, including the audio features that we extracted, and also the models that were used to reproduce the baseline results. Finally, we describe the various data-augmentation techniques that we used to artificially increase the amount of audio data in dataset-II.

3.1 Overview

As discussed earlier in section 2.4, different audio event detection applications require different kinds of audio datasets, and our work involves three different datasets. These datasets are meant for different kinds of applications and are also of different nature i.e. dataset-I is a monophonic audio event dataset and datasets-II and III are polyphonic. In case of dataset-I, because each audio recording belongs to a single audio event class, we developed a single classifier for all audio event classes that can classify/detect the audio class, which a particular audio recording belongs to. However, for datasets-II and III, we developed as many classifiers as the number of audio classes in the dataset. That is, we designed n number of binary one-vs-all classifiers (n is the number of audio classes), one for each audio class x , where all the audio recordings having audio class label x are positive data samples and the rest of the recordings are data samples of the negative class for the binary classifier. This approach of using n different binary classifiers instead of one multi-class classifier provides us with the extra flexibility of fine-tuning the classifier for each class independently, as well as better analyze the performance of each classifier individually. However, by following this approach, we neglect the correlation between the occurrences of different audio classes.

For all the three datasets, we also have some baseline results from literature. Hence, for each dataset, our approach was to first try to reproduce the respective baseline results. Specifically, for dataset-I, the baseline results of literature were obtained using a *k-Nearest Neighbors (kNN)*

model. Hence, we trained a similar kNN model to reproduce those results. For datasets-II and III, the baseline models were *Support Vector Machines (SVM)* and *Gaussian Mixture Models (GMM)* respectively. Hence, we also designed a similar SVM and GMM model to reproduce the baseline results for these two datasets.

After reproducing the baseline results for each dataset, we designed a DNN and a CNN model to test their performance on these datasets and to observe if these classifiers perform better than the baseline models. We tested the performance of implemented DNN and CNN models on dataset-I, and upon observing a better performance of the CNN model compared to the DNN model, we tested the CNN model performance further on the other two datasets. For each of the three datasets, our CNN model had a similar architecture but owing to the characteristic differences between the datasets, we had to modify our CNN model to adapt to each dataset. The different models that we implemented are explained in the next section.

3.2 Classifiers

We have used MFCCs as the audio features in all our experiments. For the baseline results, we have tried to reproduce the same baseline models with the same audio feature set. And for the neural networks (CNN and DNN), we experimented with different model parameter settings and different audio feature-sets, to obtain promising classification results. In the following, for each approach, we discuss the audio features used, and the classification procedure.

3.2.1 Baseline Models

3.2.1.1 k-Nearest Neighbors Approach

The kNN model was designed to obtain baseline classification results on dataset-I. We consider the kNN results in [5] as the literature baseline results for dataset-I, and our aim was to achieve the same results using our kNN model.

Audio Features Motivated by the set of audio features used in [5], we generate, for each audio recording, a feature set that consists of the mean, standard deviation, minimum, maximum, median, skewness and kurtosis of the MFCCs as well as the mean and standard deviation of the delta- and delta-delta-MFCCs. All these operations (mean, standard deviation, etc.) are performed for each MFCC, over all frames of an audio recording.

Initially, we extract the 25 top MFCC coefficients from a frame of a recording and then by performing the above operations, we obtain a feature set of size 275 for one audio recording. Figure 3.1 shows the schematic diagram of the feature extraction process. We use Sony's WriteFeats¹ tool for extracting MFCC coefficients from the audio recordings with the extraction

¹Sony's *MatAudioLib* toolkit: for internal use only

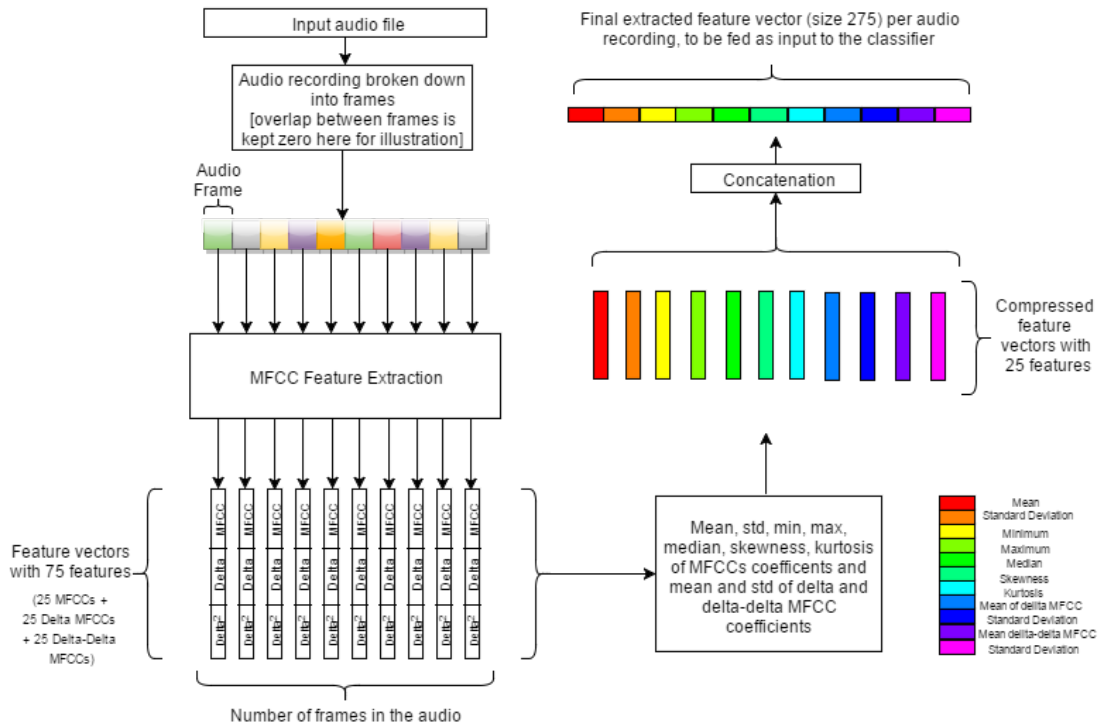


Figure 3.1 – Audio Features Extraction Schematic for kNN model

parameters mentioned in [5]. The values of parameters used in MFCC extraction are shown in table 3.1

Table 3.1 – Parameters for MFCC extraction

Parameter	Value
Number of MFCC coefficients	25
Number of mel filters	40
Frame length	23.2 (millisec)
Frame shift	11.6(millisec)
Maximum allowed frequency	22050(Hz)
Minimum allowed frequency	0(Hz)
Windowing type	Hamming
Normalize mel filters (yes/no)	Yes
Delta MFCC context (number of neighbor frames to be included)	21
Delta-delta MFCC context	11
Channel input	2 (channels)
Pre-emphasis value	0
Cepstrum lifter type	None
Cepstrum lifter value	0
Denoising in mel domain (yes/no)	No

Classification Following the kNN model specifications mentioned in [5], we train a k-nearest neighbors classifier with $k = 5$. A 10-fold cross validation experiment is conducted to obtain the classification performance results, where we ascertain that no two folds have audio slices/segments/excerpts from the same audio recording. The size of the input feature vector is 275 and the output of the classifier is a single class label, for e.g. 0 for class *air-conditioner*, 1 for *car-horn*, etc.

We tried various distance measures to quantify the closeness of a data point with its neighbors, such as *euclidean*, *cosine*, *correlation-based similarity*, etc. and found that *cosine* distance was the best distance measure in our case. Hence, we used it for obtaining the kNN results. Complete implementation of this model was performed using Matlab.

3.2.1.2 Support Vector Machines Approach

An SVM model is a representation of the training examples as points in a space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New (test) examples are then mapped into that same space and predicted to belong to a category or class based on the side of the gap they fall on[36]. If the distance metric between the examples mapped as points in the classification space is linear, then the SVM performs linear classification. But, we can define other distance metrics which are not linear, making the SVM a non-linear and a more complex classifier. These distances are expressed using kernel functions. Inspired by [33], we use a radial basis function (RBF) kernel function for our SVM classifier. We use the SVM model to reproduce the baseline literature results for dataset-II. SVM is also used as the standard baseline model for dataset-III.

Audio Features We use MFCCs as the audio features for this model. Following the data pre-processing step used in [33], the audio recordings are split into audio segments of 5 seconds, and for each segment we extract 25 MFCC coefficients for each audio frame within the segment. Finally, we take the mean and standard deviation of the MFCC coefficients over all the frames of a recording. After concatenating the mean and standard deviation vectors, we get an input feature vector of size 50 for each audio segment of 5 seconds. The schematic diagram for the feature extraction process is shown in figure 3.2. We used Sony's WriteFeats² tool for extracting the MFCC coefficients from the audio recordings. The parameter setting used in extracting the MFCC coefficients are shown in table 3.2. Most of these settings are derived from [33].

Classification Following the approach used in [33], we used an RBF kernel function (with $\gamma = 0.2$). An RBF kernel function on two samples x and x' , represented as feature vectors, is

²Sony's *MatAudioLib* toolkit: for internal use only

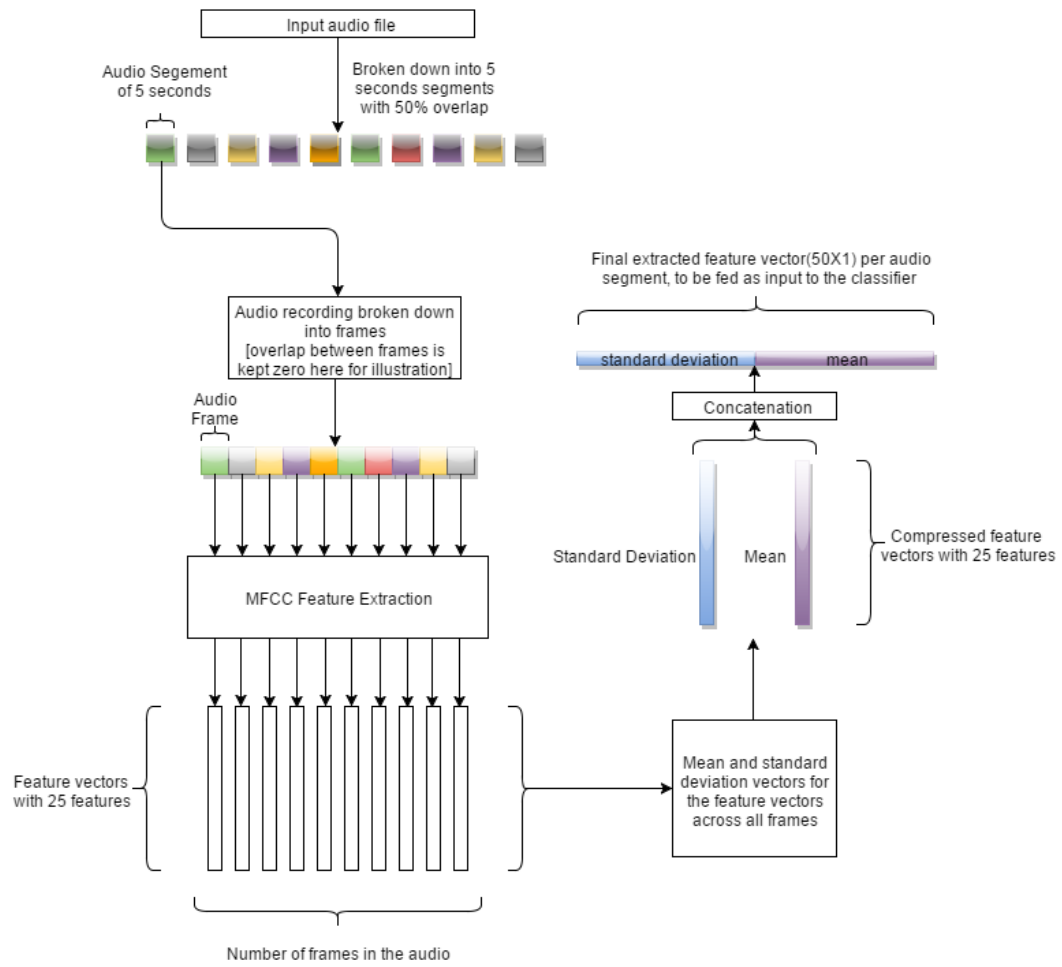


Figure 3.2 – Feature Extraction Schematic for SVM model

Table 3.2 – Parameters for MFCC extraction dataset-II

Parameter	Value
Number of MFCC coefficients	25
Number of mel filters	30
Frame length	46 (millisec)
Frame shift	23 (millisec)
Maximum allowed frequency	7500 (Hz)
Minimum allowed frequency	80 (Hz)
Windowing type	Hamming
Normalize mel filters (yes/no)	Yes
Channel input	2 (channels)

defined as:

$$K(x, x') = \exp(-\gamma \|x - x'\|^2)$$

Chapter 3. Implemented Method

For all other SVM parameters, we used the values mentioned in [33]. We trained 4 binary SVM classifiers for the 4 classes, by taking all the training data belonging to that class as positive examples and the rest of the data as negative examples. We normalized the input features to the range [0,1] with the scaling estimated from training data being applied to the test data. Matlab was used for implementing the SVM model.

3.2.1.3 Gaussian Mixture Models Approach

To reproduce the baseline literature results obtained on dataset-III, we implemented the GMM model. We extracted MFCC features out of the audio recordings and estimated GMMs on sequences of the MFCC features. GMM is often used in combination with MFCC features in the literature for tackling speech recognition or sound classification problems ([37], [38], [39]).

Audio Features The parameters for the extraction of MFCC features are derived from [35]. The parameter setting used in MFCC extraction from the audio recordings is shown in table 3.3. After discarding the first MFCC coefficient (which represents the frame energy), we get 13 MFCCs for each audio frame for each audio recording for estimating the GMMs. Complete feature extraction process is performed using Sony's WriteFeats³ tool.

Table 3.3 – Parameters for MFCC extraction for dataset-III

Parameter	Value
Number of MFCC coefficients	14
Number of mel filters	40
Frame length	21.33 (millisec)
Frame shift	10.67 (millisec)
Maximum allowed frequency	24000 (Hz)
Minimum allowed frequency	0 (Hz)
Windowing type	Hamming
Normalize mel filters (yes/no)	Yes
Channel input	2 (channels)

Classification For each of the 7 audio classes of the dataset, we estimate a pair of GMMs (as done in [35]). Using the expectation-maximization algorithm, we estimate full-covariance Gaussians. The number of Gaussians in the GMMs are varied as the values in the set: {1,2,4,8}. For predicting the presence or absence of an audio class, we calculate the log-likelihood ratio of the feature vector with respect to the estimated GMMs (details can be found in [35] and are also discussed in the results section). Scikit-learn⁴ was used to implement the GMM model.

³Sony's *MatAudioLib* toolkit: for internal use only

⁴<http://scikit-learn.org/0.15/>

3.2.2 Deep Neural Network Approach

We implemented a DNN classifier to test its classification performance on dataset-I. Similar to the kNN model, we use MFCCs as the features for this model. The DNN is trained with these features as input and the class labels of the audio event classes as output. Posterior probabilities of audio event classes are obtained as DNN predictions. During 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 transformed using softmax transformation and a zero-one classification error is then calculated for the testing examples. Figure 3.3 illustrates a high-level idea about the functioning of the classification system.

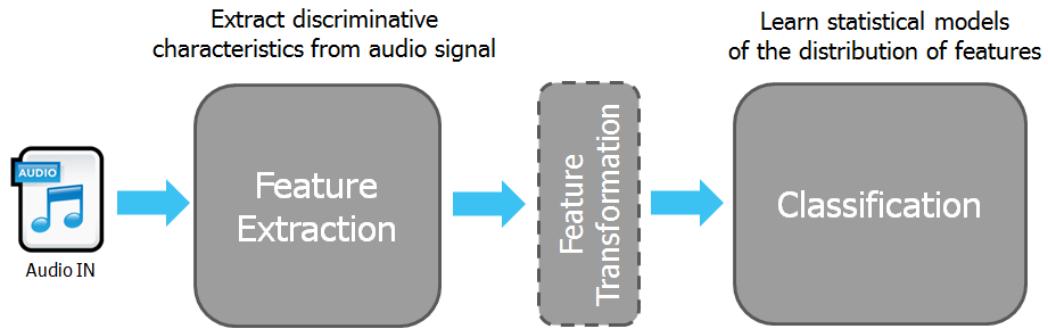


Figure 3.3 – Audio Classification Framework

3.2.2.1 Audio Features

Audio features are expected to provide a reasonable representation of the time-frequency characteristics of the audio signal. MFCCs are one of the most used and successful audio features for speech recognition and events classification tasks, hence we use MFCCs as the audio features for our model. We extract the top 25 MFCCs for each frame of an audio recording and also take the corresponding 25 delta-MFCCs and delta-delta MFCCs. This makes the input feature vector of size 75. Each audio recording is broken down into constituent frames and then the mean and standard deviation of the features vectors computed over all the frames are concatenated, resulting in a vector of size 150 ($75 + 75$). Figure 3.4 shows the schematic diagram for the feature extraction process. We used Sony's WriteFeats⁵ tool for extracting the MFCC features from the audio files/recordings. The parameter values used for MFCC extraction are the same as that used in kNN model (see table 3.1)

⁵Sony's *MatAudioLib* toolkit: for internal use only

Chapter 3. Implemented Method

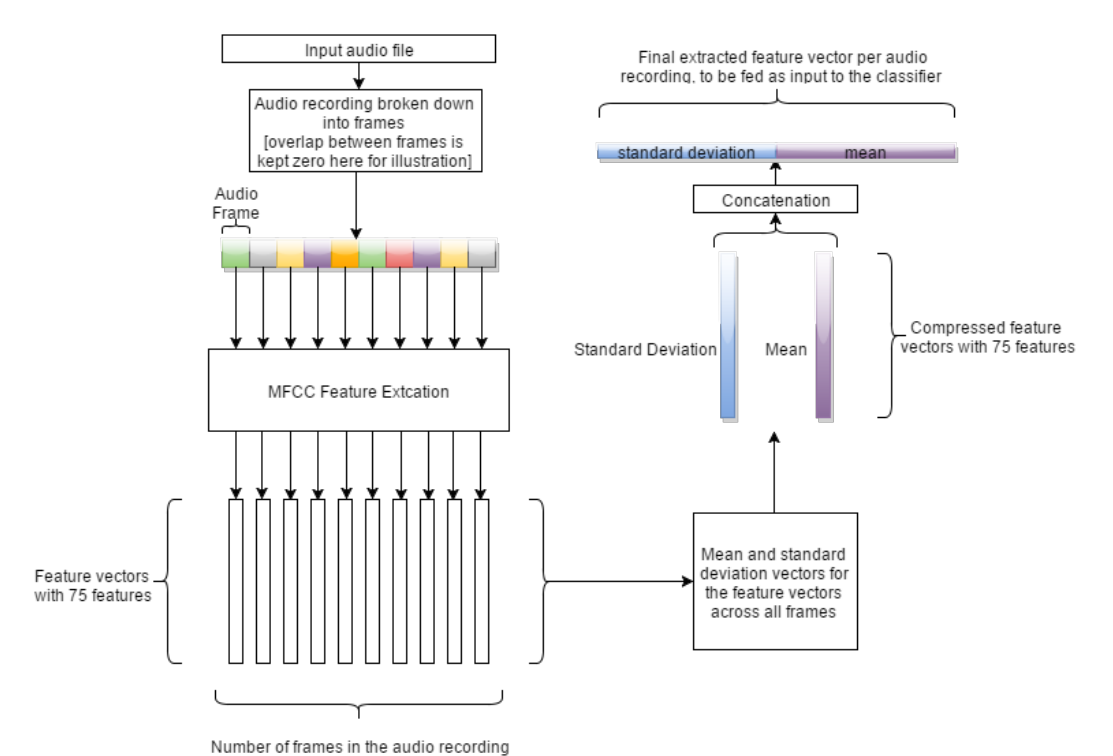


Figure 3.4 – Audio Feature Extraction Schematic - DNN

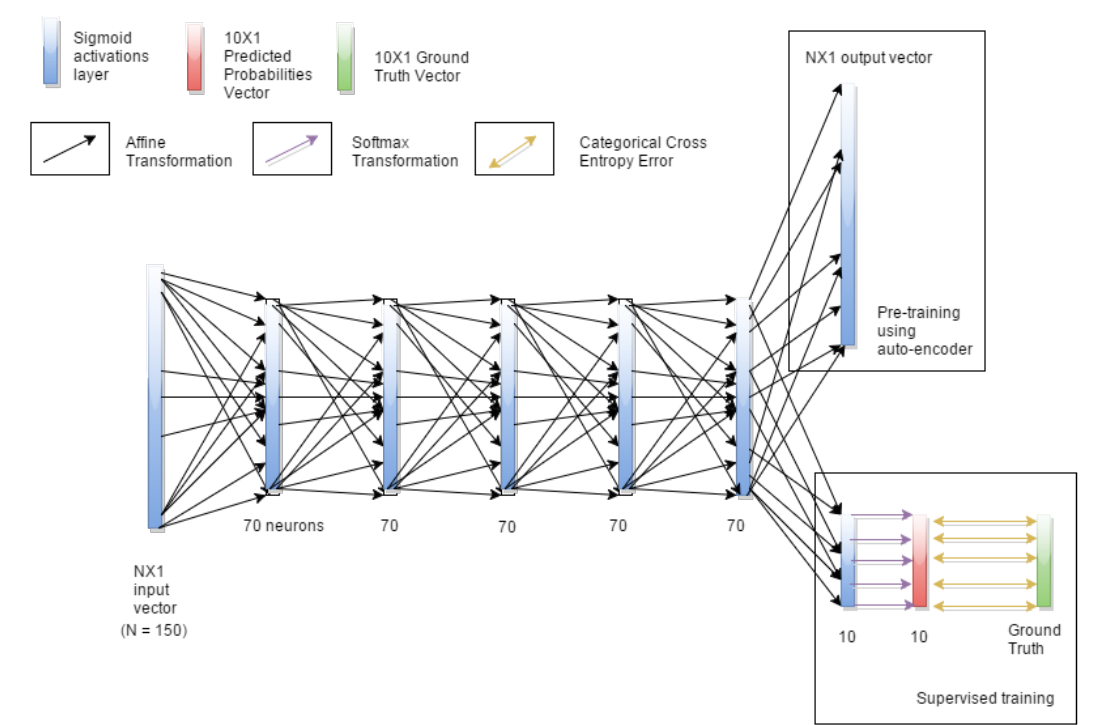


Figure 3.5 – DNN Network Architecture

3.2.2.2 Classification

The neural network architecture that we developed for this dataset is inspired from [12]. There are 5 hidden layers each with 70 neurons. The final layer outputs a vector of size 10 with the softmax probabilities for the 10 audio classes. We use sigmoid activation throughout the network. Instead of directly training the network with random initialization of neurons, we pre-train the network using an auto-encoder pre-training technique. The details of the network architecture are displayed in figure 3.5

Categorical cross-entropy is used as the cost function for the training process. Zero-one error is the error-metric used for the evaluation of the performance of this classifier. The output or target audio class labels are one-hot vectors of size 10. Complete implementation of the network architecture as well as the training and testing procedure is performed using sDeePy⁶, which is Sony's internal deep learning library.

3.2.3 Convolutional Neural Network Approach

In all three datasets, the audio recordings have a certain fixed length. In dataset-I, all audio recordings are 4 seconds long and similarly in dataset-II and dataset-III, they are 5 seconds and 4 seconds long respectively. This means that all the audio recordings consist of a constant number of frames. For each frame, we compute MFCC coefficients as part of the feature extraction step. Hence, we can imagine the audio recording transformed into MFCC coefficients, as an image where each MFCC coefficient for each frame of the audio recording is a pixel in the image. Figure 3.6 illustrates the transformation of a 4 second long audio recording into an MFCC matrix of size (24, 215), where 24 is the number of MFCC coefficients extracted per frame, and 215 is the number of frames in the recording. The MFCC matrix can be considered as an image, which is then split into 24 one-dimensional feature maps. These feature maps can be used as the input to CNN. Therefore, this manipulation facilitates us to use a CNN model on audio data just as it is used on images.

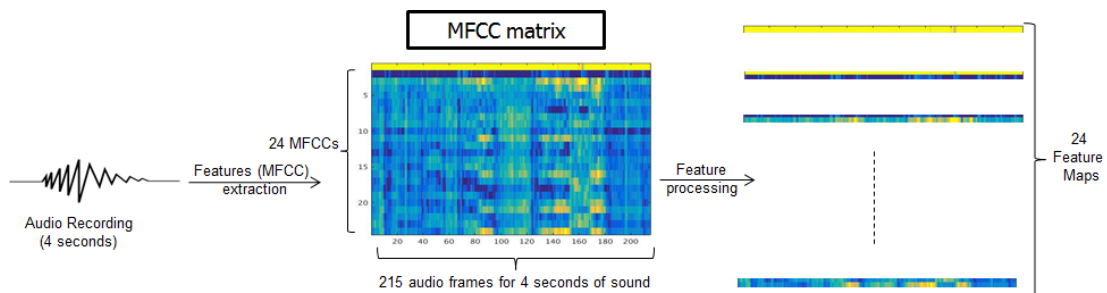


Figure 3.6 – MFCC matrix as an image for CNN

⁶sDeePy provides high-level APIs on top of the basic Theano functions which makes it convenient to create, train and test neural networks. It is meant for internal use only

Table 3.4 – Parameters for MFCC extraction for CNN for dataset-II

Parameter	Value
Number of MFCC coefficients	25
Number of mel filters	30
Frame length	30 (millisec)
Frame shift	10 (millisec)
Maximum allowed frequency	11025 (Hz)
Minimum allowed frequency	0 (Hz)
Windowing type	Hamming
Normalize mel filters (yes/no)	Yes
Channel input	2 (channels)

3.2.3.1 Audio Features

For dataset-I, based on the feature extraction procedure used in [5], we extracted the top 25 MFCC coefficients along with the corresponding delta-MFCCs and delta-delta-MFCCs for each frame of each audio recording. Similarly for dataset-II and dataset-III, we extracted 25 and 14 MFCC coefficients respectively for each audio frame of each audio recording, based on the feature extraction procedure used in related literature ([33], [35]). We ignored the first MFCC coefficient, which is usually done with MFCCs in audio event detection literature, because it just represents the log of the total energy of an audio frame, hence does not carry any unique or useful characteristic information. After excluding the first MFCC coefficient, we obtain input feature vectors of sizes 74, 24 and 13 per audio frame for datasets-I, II and III respectively.

A single audio frame is broken down into, say, N frames and since N is constant, each audio recording can be considered as a vector-of-frames of size N . Each frame, in turn, can be represented as a feature vector of size F (where F is 74 for dataset-I, 24 for dataset-II and 13 for dataset-III). Hence we can say that each audio recording is a one dimensional image which is represented by F feature maps. By this interpretation of audio recordings, it is possible to construct a CNN architecture that takes these F feature maps as its input.

We used Sony's WriteFeats⁷ tool for MFCC extraction and the parameter values used in extracting the MFCCs for dataset-I, dataset-II and dataset-III are given in tables 3.1, 3.4 and 3.3 respectively.

3.2.3.2 Classification

We designed a CNN model by interpreting audio recordings as images. However, for different datasets, the audio data has different specifications such as the length of the audio recordings or the number of audio classes in the dataset. Hence, we started with a CNN model for dataset-

⁷Sony's *MatAudioLib* toolkit: for internal use only

I and then for testing this model on datasets-II and III, we modified the network to adapt to the audio recordings and audio classes in the these datasets. In the following we describe the convolutional neural networks that we designed for the three datasets and the procedure that we used for training these networks.

Dataset-I We use the CNN architecture as shown in figure 3.7. This particular architecture is one of the many architectures that we experimented with, and while discussing the results, we present more variants of this CNN model. In this architecture, there is one convolutional layer with 74 different kernels of size (3,1) that convolve over the 74 respective input feature maps to produce 10 output feature maps. This layer is followed by a pooling layer, that max-pools 1 out of the 12 convolved features providing a compressed representation. Finally, we have an affine(fully-connected) layer that gives an output of size 10. A subsequent softmax transformation provides the softmax probabilities for the 10 classes. We use sigmoid activation function in all the layers, throughout the network.

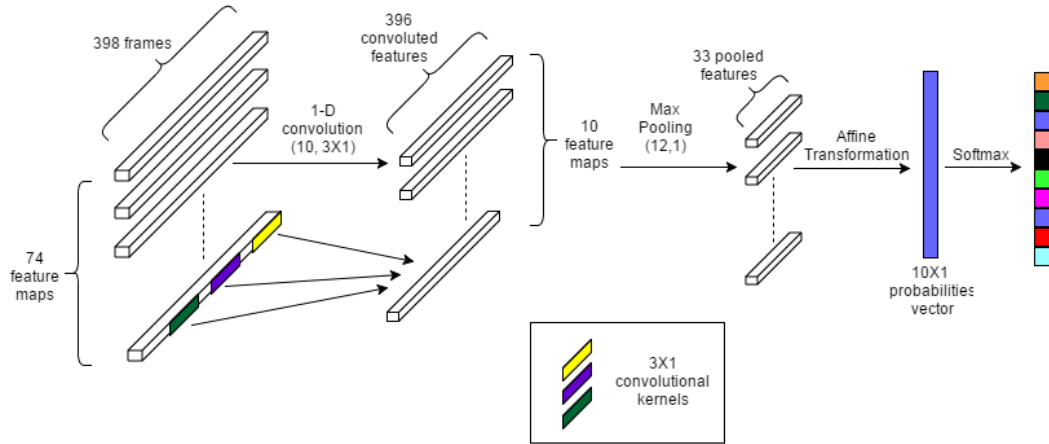


Figure 3.7 – CNN Network Architecture

We use a learning rate of 0.0003 for training with a batch size of 50 data instances. The training is done for 1000 epochs. We do not use momentum or dropouts or any other optimization. We maintain separate training and validation data samples, with 4500 training samples and 1200 validation samples. A data sample, here, simply means an audio recording that is pre-processed into 74 feature maps.

Categorical cross-entropy is used as the cost function for the training process. It is one of the most widely used metrics for monitoring errors in a multi-class classification problem. Inspired by the evaluation method of [5], zero-one error is used as the error-metric for evaluation of the classifier performance. The output/target audio class labels are one-hot vectors of size 10 as in the DNN model. Complete implementation of the CNN model is performed using Sony's sDeePy⁸ library.

⁸sDeePy provides high-level APIs on top of the basic Theano functions which makes it convenient to create,

Dataset-II The CNN model for this dataset is derived from the CNN model designed for dataset-I. The CNN architecture is shown in figure 3.8. There is 1 convolutional layer with 24 kernels of size (8,1) that convolve over the 24 respective feature maps to produce 4 output feature maps. This layer is followed by a pooling layer, that max-pools 1 out of the 16 convolved features resulting in a compressed representation. Subsequently, there is an affine/fully-connected layer that provides a single unit output from the 4 compressed feature maps. Finally, there is a sigmoid transformation over the single unit output that gives us the predicted probability of the binary classifier. We use tanh activation function for the hidden layers of the network. The network consists of 768 convolutional layer parameters and 54 fully connected parameters summing to a total of 820 parameters.

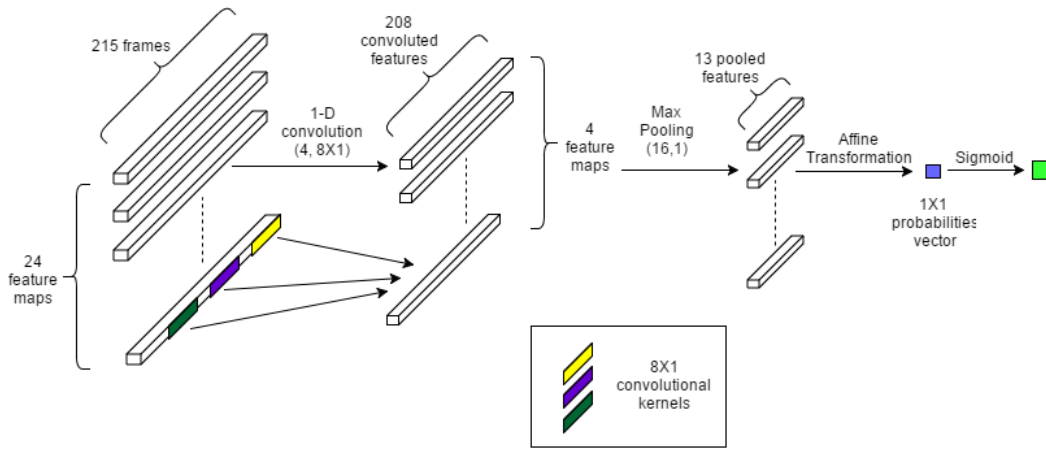


Figure 3.8 – CNN model architecture for dataset-II

We find the learning rate of 0.003, and a batch size of 50 suitable for the classification task. The training is done for 1000 epochs. We do not use momentum, dropouts or any other such optimization. We roughly have 3500 training and 800 validation data samples. Here, each data sample refers to an audio segment of 5 seconds which is pre-processed into MFCC features.

Weighted binary cross-entropy is used as the cost-function for the training process. Different classes are assigned different weights such that the weights are inversely proportional to the size of each class. These weights are then used as an extra parameter inside the standard cross entropy function to account for the skew in the proportion of the positive and negative samples of the classifier (details discussed in the results section). For the evaluation of the classifier performance, equal error rate is used as the error metric. The choice of these error metrics for the cost function and for the evaluation is motivated from [33]. The CNN architecture as well as the training and evaluation procedures are implemented using Sony's deep learning library - sDeePy⁹.

train and test neural networks. It is meant for internal use only

⁹sDeePy provides high-level APIs on top of the basic Theano functions which makes it convenient to create, train and test neural networks. It is meant for internal use only

Dataset-III Figure 3.9 shows the CNN architecture, which is similar to the architecture for dataset-II. There is 1 convolutional layer with 13 kernels of size (6,1) that convolve over the 13 respective feature maps to produce the 4 output feature maps. This layer is followed by a max-pooling layer of size 16. Max-pooling results in a compressed representation. Then, a fully-connected layer provides a single unit output from this compressed representation. Finally, a sigmoid transformation over the unit output provides us the probability of whether the audio recording belongs to a particular audio class or not, as predicted by the CNN classifier. The network consists of 312 convolutional layer parameters and 92 fully-connected layer parameters summing to a total of 404 total parameters.

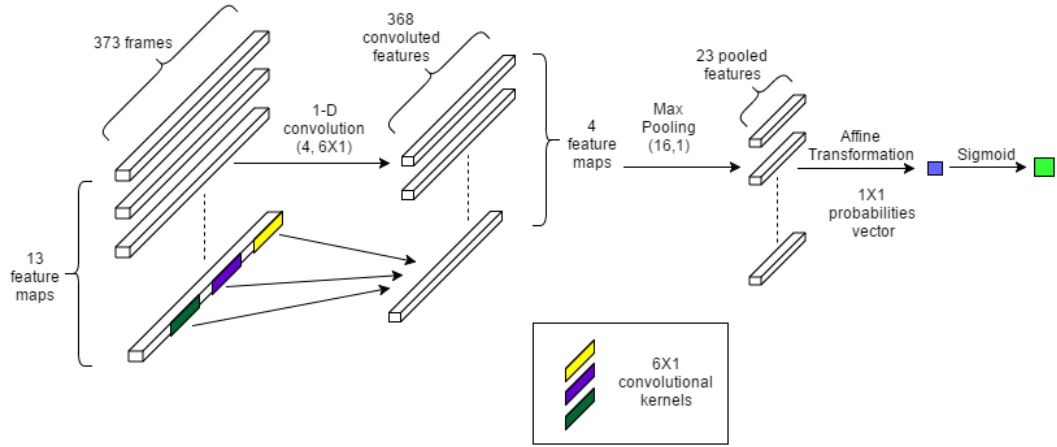


Figure 3.9 – CNN model architecture for dataset-III

We find the learning rate of 0.01 and a batch size of 50 suitable for this classifier. Training is done for 500 epochs. We do not use any further optimization in the training procedure like momentum, dropout, etc. The training set consists of approximately 1750 audio recordings and the testing set has 200 data samples. Each data sample here means an audio recording pre-processed into MFCC feature maps.

As used for dataset-II, we use weighted binary cross-entropy as the cost function to account for the skew between the number of data samples belonging to the positive and the negative class. The weights for each class are inversely proportional to the size of (number of data samples belonging to) each class. Details about the cost function are discussed in the results section. For the evaluation of the classifier performance, motivated from [35], we use AUC - area under ROC curve as the error metric. AUC is a reasonable error metric when evaluating the performance of a binary class in a situation where the positive and negative classes are not equal in size (number of data samples). Complete CNN implementation is done using Sony's deep learning library - sDeePy¹⁰.

¹⁰sDeePy provides high-level APIs on top of the basic Theano functions which makes it convenient to create, train and test neural networks. It is meant for internal use only

3.3 Audio data augmentation experiment

Dataset-II consists of approximately 4500 audio recording segments, each of length 5 seconds. 4500 data samples is not enough to completely utilize the learning potential of neural networks especially deep neural networks. In other words, with more data, the model performance could potentially improve further. This could be done in two ways: (i) manually annotating new audio recordings and adding to the dataset, (ii) generating new data samples from the available dataset, preserving the original class labels. We followed approach (ii) and this decision was motivated by two reasons:

1. Approach (i) is highly time-consuming and costly. For example, it can take up to 20 hours for an annotator to annotate 1 hour of audio data.
2. The success of artificial data augmentation in the field of computer vision, where images are cropped, mirrored, blurred, etc. and fed as new additional data samples. [40] used such an approach and proved to be a landmark in the field of computer vision.

Inspired from [41] and [42], we chose 6 different effects that could be applied on the original audio recordings to artificially generate new recordings which have the same audio event class labels. Those six effects are: (i) Changing pitch, (ii) Adding echo, (iii) Reducing noise, (iv) Adding tremolo effect, (v) Adding noise and (vi) Adding reverb. The only thing common in all these effects is that they do not change the overall duration/length of the recording, which is a necessary condition if we want to preserve the time-wise annotated labels. In the following, we discuss what these effects are and how were these implemented.

Pitch is a subjective property of sound which changes proportionally with the frequency. For example, a high pitch sound appears to be a sound with high frequencies. In this database, the audio classes are too broad to be affected by a change in the pitch of their audio recordings. For example, a crowd sound, even when up-pitched or down-pitched, will still sound like a crowd noise, and similarly the other classes. This would however not be the case if there were two such classes: male voice and female voice. Interestingly, by changing the pitch of the recordings of our dataset, we get reasonably differently sounding recordings, good enough to be called as new recordings. Thus, by creating one more version of the whole dataset (say up-pitched version), we get a new augmented dataset which is twice the size of the original (i.e. almost 9000 recordings).

Echo is another well-known sound effect in which the sound at time instance t is repeated with amplitude-fraction α at time instance $(t + \delta t)$, where δt is the echo delay, and α is the fraction of the original amplitude with which the sound echoes. In our dataset, adding echo to the recordings especially those belonging to the 'crowd' and 'applause' classes worked well. Because of the nature of these audio classes, adding echo did not change the meaning of the sound and just filled the audio recordings with more of the same content.

Reverberation is an effect similar but not equal to the echo effect. In reverberation, the sound

instance at time instance t is repeated not just at $(t + \delta t)$, but in a range of timings starting from t to $(t + \delta t)$. The intuition behind doing this is to emulate an acoustic environment (like a hall) where the sounds are getting reflected from all kinds of distances, shorter distances reflecting the sound repetitions that are quicker than the long distances. This multi-repetition of sounds produced at each instance creates an illusion that the recording is being played in a music hall. This effect was also effective in creating meaningfully different versions of the original audio recordings while preserving the audio labels.

Adding and reducing noise are two other effects that we applied to the dataset. For adding the noise, we used a random gaussian noise generator and added the noise on top of the recordings. The modified recordings sounded like having white noise due to poor quality of recording equipment or due to poor communication infrastructure, which can both happen in a real-world recording system. Removing noise was also helpful because most of the recordings in the dataset have a lot of noise especially for the *traffic* class, where the recording equipment is along a highway and even when there is no traffic, the winds or people crossing by are generating a lot of noise. By using this effect, we obtained a cleaner versions of the recordings while retaining the audio class labels.

Finally, tremolo is another effect where the amplitude of the recording is varied in a range (say 50% to 100%) periodically. This creates a "*wahwahwahwa...*" kind of effect on the sound, which is also observed in real world recordings quite often, due to the poor recording conditions, or interference of some periodic signals with the recorded audio.

Audacity¹¹, which is an open source audio editing software, was used in creating and adding all the above effects to the original audio recordings. The values for the different parameters involved in the various effects were kept as the default values provided by the software.

¹¹<http://audacityteam.org/>

4 Evaluation and Results

4.1 Overview

In this chapter, we present the evaluation procedure employed as well as the results obtained for the three datasets that were used in our experiments. For each dataset, we have first tried to reproduce the baseline results of the related literature, using their evaluation metrics, and then we have tested the performance of our DNN and CNN models on the datasets. However, in each dataset, the baseline results in respective literature are obtained using different machine learning models and different evaluation metrics. These metrics and models are shown in table 4.1.

It can be seen that both the classifier and the evaluation metric change as we change the dataset, which leaves us with no common baseline across datasets. To make a comparison of our implemented DNN and CNN models with a standard reference, we need a common set of baseline results, where a standard classifier and a standard evaluation metric is used. Dataset-I being a monophonic audio event dataset, however, cannot be compared to the other two datasets that are polyphonic. In dataset-I, we have used a single classifier that simultaneously classifies for all 10 audio classes. On the other hand, we have as many binary classifiers as the number of audio classes for the other two datasets. Nonetheless, for datasets-II and III, it is possible to have a common standard baseline. Therefore, in our results, we additionally present these standard baseline results by using SVM as the classifier and AUC as the evaluation metric.

Table 4.1 – Classifiers and evaluation procedures for the three datasets

Dataset	Classifier used	Evaluation metric
Dataset-I	kNN	Zero-one error
Dataset-II	SVM	Equal error rate (EER)
Dataset-III	GMM	Area under ROC curve (AUC)

Table 4.2 – Urban8K dataset - review

Class	ID
air conditioner	0
car horn	1
children playing	2
dog bark	3
drilling	4
engine idling	5
gun shot	6
jackhammer	7
siren	8
street music	9

4.2 Dataset-I

As mentioned earlier, the Urban8K dataset has one audio class per audio recording which makes it applicable for a monophonic audio event detection system. Table 4.2 reviews all the audio classes present in this dataset.

In the following, we discuss the evaluation procedure that we adopted to evaluate our results and then we present the results of our models trained on this dataset and compare those results with the baseline results available from literature.

4.2.1 Evaluation Procedure

Motivated by [5] which is the source of this dataset, we use zero-one classification error as the evaluation metric for our results. In the following, we explain how exactly is the zero-one error computed.

In our implemented DNN and CNN models, as there are 10 classes, the class labels or the ground truths for each of the 10 classes are first converted to one-hot¹ vectors of size 10. For example, a recording labeled as "air conditioner" (i.e. ID = 0) will translate to the one-hot vector [1 0 0 0 0 0 0 0 0 0], another one labeled as "drilling" (ID = 4) would be written as [0 0 0 0 1 0 0 0 0 0], and so on. Similar to the ground truth, the predictions made by the machine learning models are also vectors of size 10. But, instead of 0's and 1's, these vectors consist of probabilities. For example, if we have a prediction vector $\mathbb{P} : [p_1 \ p_2 \ p_3 \ p_4 \ p_5 \ p_6 \ p_7 \ p_8 \ p_9 \ p_{10}]$ for a certain audio recording, then p_1 indicates the probability of this audio recording being an "air-conditioner" sound, p_2 represents a "car-horn" sound and so on. The penultimate layer of the neural network contains units that are the sigmoidal transformations of the units (neurons) from the last hidden layer of the network. This penultimate layer is then passed through a softmax transformation that gives us the final prediction probability vector \mathbb{P} . The

¹one-hot vectors are binary vectors consisting of 0's and 1's with a single non-zero element

softmax transformation is expressed as:

$$\forall n \in [1, 10] : p_n = \frac{e^{q_n}}{\sum_{k=1}^{10} e^{q_k}}$$

where $q_i : i \in [1, 10]$ represents the sigmoid-transformed units of the penultimate layer.

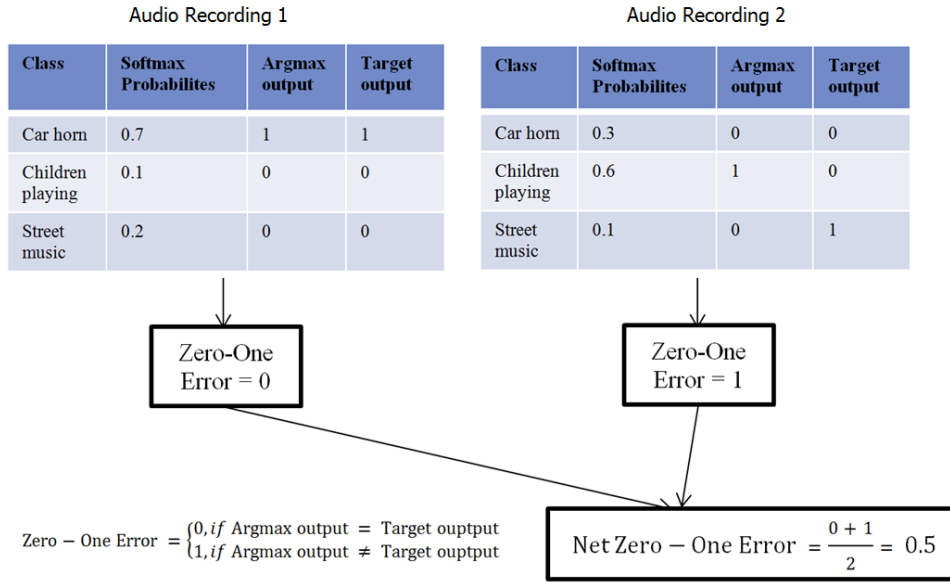


Figure 4.1 – Evaluation using zero-one error: a schematic

As we have both the ground truth and the prediction, they can now be compared to obtain the error. If the argmax of the prediction vector equals to the argmax of the ground truth vector, then the error is zero, else the error is one. This error is the error for one audio recording. The final error is simply the average of all such errors over all the audio recordings that are being used for testing/evaluation. Figure 4.1 shows a simplified schematic of the evaluation procedure for a toy dataset of 2 audio recordings with 3 audio classes.

For the above case, one audio recording means one training instance for the machine learning model. That is, one audio recording is labeled by a vector of size 10. There is another possibility, where one audio recording is broken down into individual frames where each frame becomes one training instance. In this case one audio recording still has a single ground-truth label vector, but n number of prediction vectors (where n is the number of frames in the recording). In that case, since direct comparison between the prediction and ground truth is not possible, we need to process the predictions. This is a part of the post-processing step, where we either (i) aggregate the prediction probabilities for each of the 10 classes across all the frames, where the aggregation function can be mean, median, maximum, minimum, etc. , or (ii) decide the argmax of the predictions on the basis of majority voting i.e. pick that class as the predicted output which has the highest probability in a majority of frames.

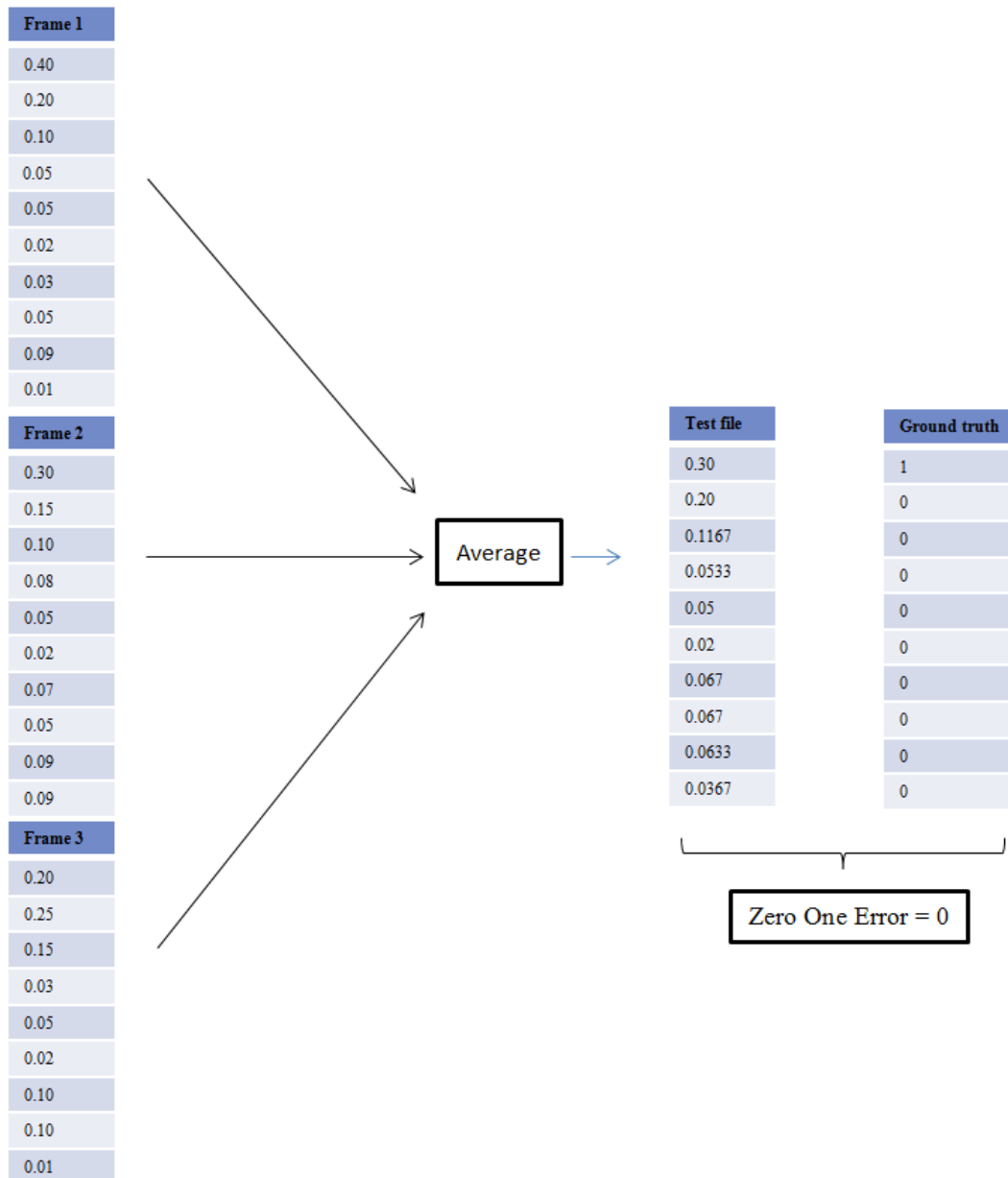


Figure 4.2 – Evaluation by averaging the frame-wise predictions: a schematic

Both approaches (i) and (ii) are commonly used in literature. We have used approach (i) wherever applicable, primarily because it was easier to implement. We tried different aggregation functions - *mean*, *maximum*, *mode* and *median* but *mean* gave us the most consistent results amongst all, hence we used it for evaluation. We also tried approach (ii) in one of the experiments and it gave the same final result as was given by approach (i), which is justifiable because *mean* also acts as a majority vote to some extent. That is, while computing *mean*, we are adding all the probabilities across the frames of an audio recording, and looking for the audio class having the highest sum of probabilities. In a similar way, under approach(ii),

the audio class having the majority should also usually have the highest sum of probabilities. Figure 4.2 shows a schematic of approach (i), consisting of the predictions from 3 frames, which are averaged and then compared to the ground truth.

For machine learning models other than artificial neural networks, like the support vector machines or k-nearest neighbors, we do not need to convert the class labels into one-hot vectors. Instead, we can use the class IDs (as shown in table 4.2) as class labels for each of the 10 classes. Nonetheless, the evaluation metric remains the same i.e. the zero-one classification error. If the predicted label matches the ground truth label, then the error is zero, else one. Once again, we take the average of all the zero-one errors over all the audio recordings to calculate the final error. Notice that there is no need to calculate argmax in this method of error calculation.

Finally, apart from obtaining the zero-one classification error, we also measure the confusion matrix, which is a 10X10 matrix in this case (10 is the number of audio classes). Confusion matrix gives us an idea of the class-wise classification accuracy of the model/classifier. The class-wise performance of the classifier cannot be gauged from the zero-one classification error. Using confusion matrix, we can identify which classes in particular are the most difficult/easy to be classified or which pair of classes are confused the most often among each other, etc.

4.2.2 Results

4.2.2.1 Baseline: k-Nearest Neighbors

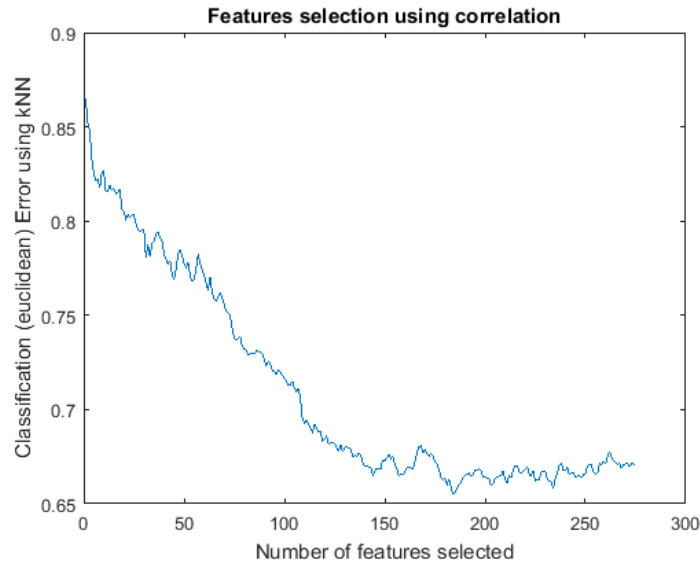


Figure 4.3 – Feature Selection Using Correlation

In an attempt to reproduce the kNN results published in [5], we implemented a kNN ($k=5$)

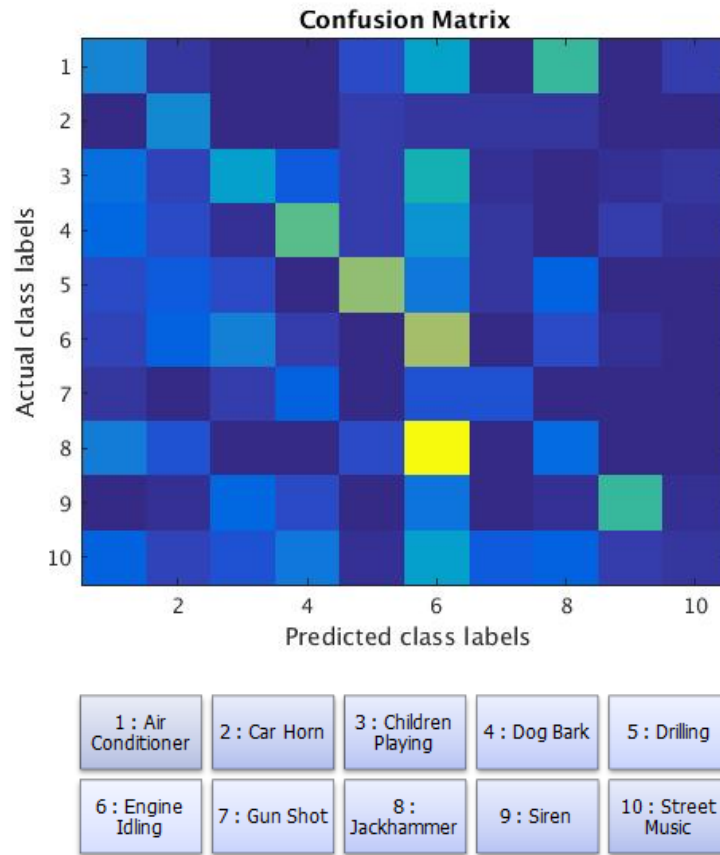


Figure 4.4 – Confusion Matrix for the kNN model

model in MATLAB and performed a 10-fold cross validation experiment. Euclidean distance was used as the distance (or error) metric for computing the errors in the kNN model. We obtained average classification accuracy of 33.4%. However the best result achieved in the literature is 56%. After failing to reproduce the results, we tried to contact the authors of [5] through which, we learned that the approach used in [5] uses an additional attribute selection process.

We tried to follow the described attribute selection procedure that says - “those features should be chosen which have a high correlation with the output and low correlation with other features”. Upon trying to keep the top k features (k varies from 1 to 275, where 275 is the size of the feature vector in our experiments) based on correlation, and evaluating the classification errors using the kNN model, we obtained the zero-one classification errors as shown in figure 4.3. From the figure, it can be seen that using attribute selection based on correlation, we achieve the best classification accuracy of approximately 35%, which is still poorer than the best literature result of 56%. Hence, we could not reproduce the results even by following the attribute selection step performed in [5]. We tried to further reach the authors to gain more insights on their implementation of the kNN model, but received no further reply.

At this point, we discontinued to work further on the kNN model for this dataset.

Figure 4.4 shows the confusion matrix for the kNN model. In the figure, class labels are labeled from 1 to 10 instead of 0 to 9. Blue color represents smaller values and yellow color represents higher values in the matrix (e.g. dark blue would mean a value close to zero). It can be seen that there is a noticeably high confusion between classes *Engine Idling* and *Jackhammer*. Overall, there are lots of confusions between the audio classes, which is evident by a lot of non-zero non-diagonal entries. We treat these results as our baseline results. Note that a random classifier would have an accuracy of 10% for this dataset because we have 10 uniformly distributed audio classes. Hence the kNN model, at least, performs better than a random classifier, as expected.

4.2.2.2 Deep Neural Network

Table 4.3 – Hyper-parameter setting for the DNN model

Hyper-parameter	Value
Feature type	MFCCs, delta MFCCs, delta-delta MFCCs
Size of input feature vector	150
Number of hidden layer	5
Number of hidden neurons in each layer	70
Number of neurons(units) used in output layer	10
Activation function for output layer	Sigmoid
Learning rate for training	0.1
Learning rate for pre-training	0.5
Pre-training type	Auto-encoder
Number of epochs for training	5000
Number of epochs for pre-training	300
Batch size for pre-training and training	100
Number of training data samples	5700
Number of validation data samples	1200
Cost function for training	Categorical cross entropy
Cost function for pre-training	L ₂ squared error
Error metric for training and validation	Zero-one classification error

The hyper-parameter values used for the network are primarily derived from [12] and then modified on the basis of experiments that we conducted for fine-tuning the network. A review of all the hyper-parameters is presented in table 4.3. Figure 4.5 displays the learning curve that shows the cost function, training and testing errors obtained by training the network for 5000 epochs.

By training the DNN with this hyper-parameter setting and subsequently testing on the test data, we obtained a classification accuracy of 40.8%. This is better than the best result obtained from our kNN model. But there is a limitation in the pre-processing step in our

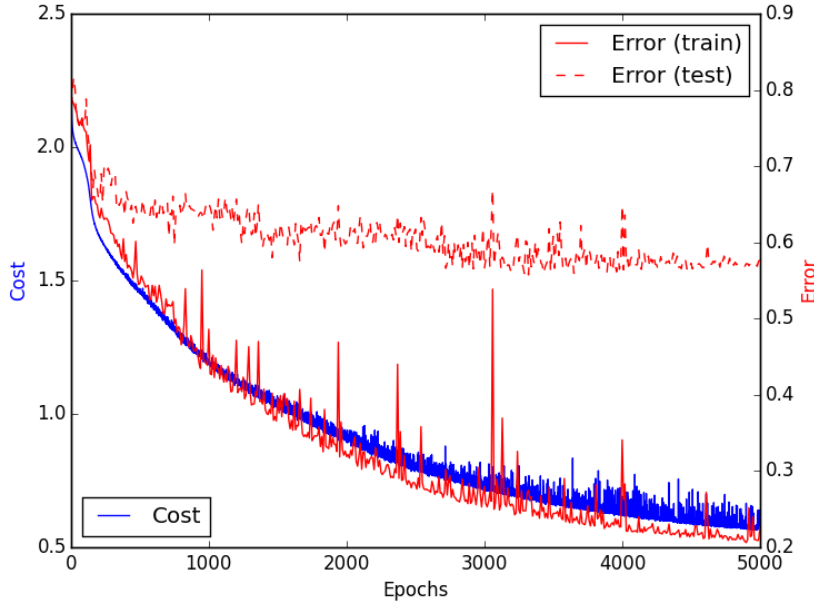


Figure 4.5 – Learning Curve for the DNN training

DNN model. We have compressed the frame-wise information of an audio recording into a single vector by taking the average and standard deviation of the feature vectors of all the frames. This compression leads to loss of information especially the contextual knowledge that flows between consecutive frames. This is a motivation to further improve the classification accuracy on this dataset by utilizing the contextual information, which is done with our next model - CNN.

In addition to the classification accuracy, we also obtained the confusion matrix for our DNN classifier which is shown in figure 4.6. First, it can be noticed from the figure that audio class *siren* has the highest classification accuracy. This could be because of the high distinctiveness of a siren sound from the other audio classes such that the spectral features for this class are clearly distinguishable. Secondly, many of the audio recordings seem to be blindly predicted as *air conditioner*, and there is a lot of confusion between the classes *air conditioner* and *engine idling*. This indicates that (i) the two classes share a lot of common spectral features that make them hard to be distinguished using our model (they also sound very similar when heard by a human), and (ii) *air conditioner* sounds sound similar to **white noise**, hence, even for a recording of *dog bark* sound, if there is some noise in the background, it could be misconceived as the *air conditioner* sound.

Lastly, the audio class *gun shot*, although seems to be classified correctly in most of the cases, is not well suited to the frames-compression method that we have used as a pre-processing step in our DNN model. Audio recordings of gun-shots would usually have only a couple of frames (out of hundreds) that actually contain the gun-shot sound, because it happens in

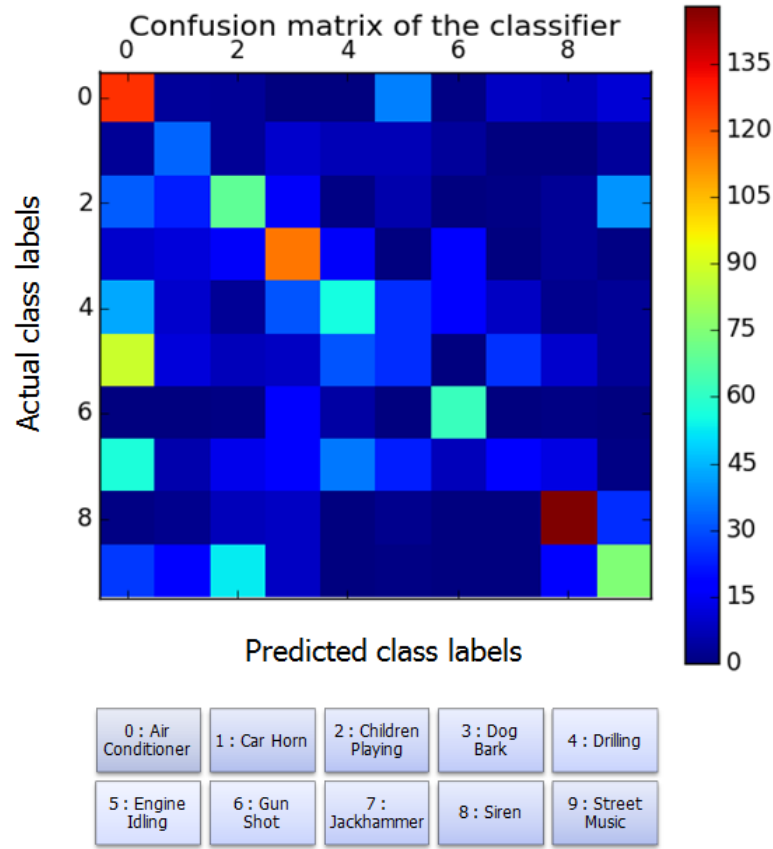


Figure 4.6 – Confusion Matrix of the DNN model

merely less than a fraction of a second. Hence, our method of taking the average and standard deviation of feature vectors across all frames is futile for such recordings. Because in our final feature vector, the real information from the frames having the actual features is heavily suppressed by the random noise from all other information-less frames.

4.2.2.3 Convolutional Neural Network

As mentioned in the previous section, the pre-processing step of the DNN model has a limitation that while compressing the information of all audio frames in an audio recording into a mean and standard deviation vector, the contextual information that could be obtained by capturing sequential changes between consecutive frames, is lost. To be able to capture the contextual information across the frames of an audio recording, we use CNN as the machine learning model.

CNN has a convolutional layer with kernels (or filters) that perform convolution operation on local regions of the data (MFCCs of the audio frames, in our case). This operation gives us local, shift invariant features in the form of gradients. These high-level features, potentially,

Table 4.4 – Hyper-parameter setting for the CNN model

Hyper-parameter	Value
Feature type	MFCCs, delta MFCCs, delta-delta MFCCs
Number of input feature maps	74
Number of output feature maps	10
Number of convolutional layers	1
Number of pooling layers	1
Number of hidden/affine layers	1
Pooling type	Max-pooling
Kernel Size	(3,1)
Activation function for the hidden and output layers	Sigmoid
Number of units/neurons in the output layer	10
Learning Rate for training	0.0003
Number of epochs of training	1000
Batch size	50
Number of training data samples	4500
Number of validation data samples	1200
Cost function for training	Categorical cross entropy
Error metric for training and validation	Zero-one classification error

carry more information about the audio recording than just the average of all frames, because the gradients provide information about the transitions happening between the consecutive frames of the recording. We started with a simple CNN model setting, and then finalized the hyper-parameters on the basis of conducted experiments with various hyper-parameter settings for fine-tuning the network. A review of all the hyper-parameter values that we used is presented in table 4.4.

The learning curve, with the cost function, training and validation errors for the 1000 epochs of training is shown in figure 4.7. From the learning curve it is evident that as the epochs progress, the model tends to fit better to the training data but the model performance on the validation data stagnates after 400 epochs.

We obtain a zero-one classification accuracy of 52.51% by testing the trained CNN model on the validation set. This result is much better than the previous two models - kNN and DNN. An explanation for this could be that in the CNN model, (i) we are able to retain the contextual information across the frames of an audio recording by performing the convolution step, and (ii) we are extracting the most useful information from each audio recording by performing the max pooling step, contrary to just taking the average of information from all frames, as in the DNN model.

Figure 4.8 shows the confusion matrix for this model. First it can be noticed from the matrix that the audio class *drilling* has the highest classification accuracy, which indicates that drilling sound is the easiest and the most distinctive to be classified by CNN model. Secondly, as

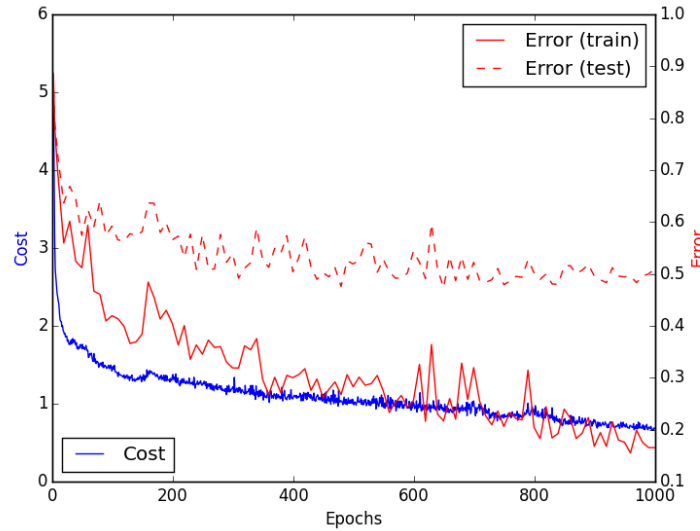


Figure 4.7 – Learning Curve of the CNN model

also noticed in the DNN model, many of the audio recordings are blindly predicted as *air conditioner*, and there is a lot of confusion between the following pairs of audio classes: (i) *engine idling* and *air conditioner*, (ii) *jackhammer* and *air conditioner*, (iii) *jackhammer* and *engine idling*, and (iv) *children playing* and *street music*. Pairs (i) and (iv) have also been reported to be confusing in the literature and would probably also sound confusing when heard by a human. And, the confusion in pairs (ii) and (iii) seems to be caused because there are long blank (noise) sequences in the jackhammer recordings, which might be wrongly interpreted as air-conditioner or engine-idling sounds. Actually, the audio class *jackhammer* is classified correctly for the least number of times and most of the jackhammer recordings are incorrectly classified as *air conditioner* or *engine idling*.

Figure 4.9 presents a comparison of our results from the three different models. It can be seen that considering our kNN model as our baseline, we have improved the classification accuracy by using the DNN model. And with CNN, we achieve an even better classification accuracy of 52.5%, which is our best result on this dataset. Note that we also report in the figure, the result from [5]. However, as stated earlier, we could not reproduce this result, which could indicate a difference in the evaluation protocol, which would make the results not comparable.

4.3 Dataset-II

As mentioned earlier, the *freesound* dataset is composed of a selection of publicly available audio recordings at the *freesound*² website. The annotations for these recordings are provided by [33]. This dataset has six potentially overlapping audio classes, which makes it a database

²www.freesound.org

suitable for a polyphonic audio event detection system.

Despite of the six available classes, the baseline results published in [33] uses only four classes: (i) Crowd, (ii) Applause, (iii), Music, and (iv) Traffic. For consistency and to be able to compare results, we also worked only with these 4 classes. In all our machine learning models, we trained 4 different one-vs-all binary classifiers for the 4 audio classes. For example, the binary classifier for class *crowd* has all the audio recordings having *crowd* in its annotations as the positive class samples and rest of the recordings as the negative class samples. In the end, we averaged the results obtained from the 4 binary classifiers to get the final result. Table 4.5 presents a review of the four classes along with the total length of the recordings that belong to each of these classes.

In the following, we first discuss the evaluation procedure that we used to reproduce the baseline results of [33]. Then, we present the results of our models trained on this dataset with this evaluation procedure, and compare those results with the replicated baseline results. Next,

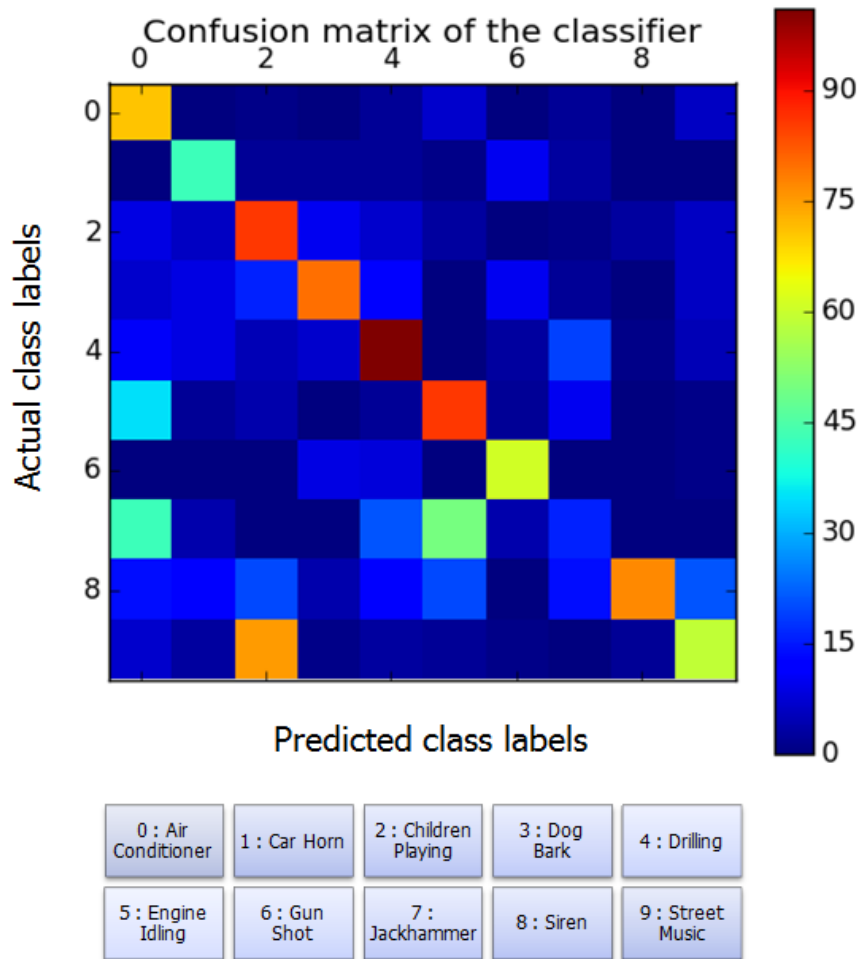


Figure 4.8 – Confusion Matrix - CNN model

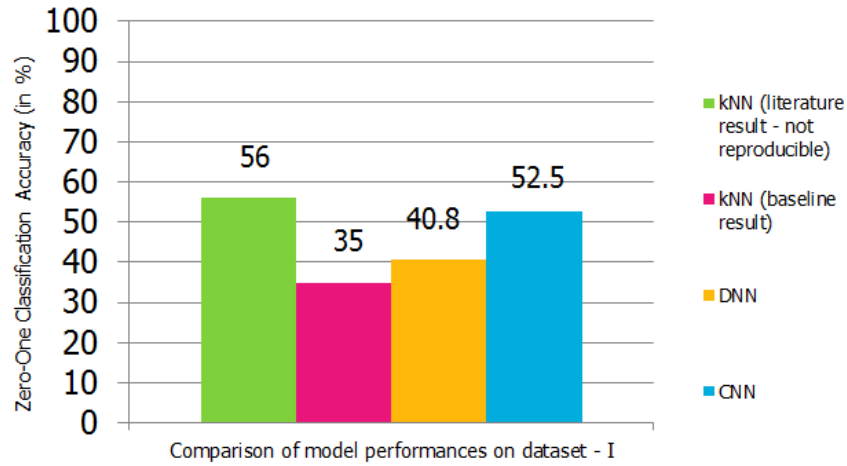


Figure 4.9 – Overall results for dataset-I

Table 4.5 – Freesound Dataset - review

Audio Class	Length (in minutes)
Crowd	45.625
Applause	26.958
Music	27.625
Traffic	34.375

we compare the performance of our implemented models with a standard baseline, which uses SVM as the classifier and AUC as the evaluation metric (as mentioned in section 4.1). Finally, we discuss the results of the artificial data augmentation experiments that we conducted on this dataset.

4.3.1 Evaluation Procedure

Motivated by [33], we use the equal error rate (EER) as the evaluation metric for our results. In the following, we explain how exactly is the EER computed.

Calculating EER on the receiver operating characteristics (ROC) curve [43] is a reliable way of estimating classification accuracy in case of binary classification problems with a skewed distribution of data in the positive and negative classes. In our case, in each of the 4 binary classifiers, the number of positive samples is much smaller than the number of negative samples, which indicates a major skew in class-wise data distribution. This justifies the use of EER as the error metric.

In the dataset, the audio recordings are 5 seconds long. For each audio recording r , and for

Chapter 4. Evaluation and Results

each binary classifier (pertaining to audio class b), a probability is predicted which is then compared to the ground truth binary label (0 or 1). This probability indicates the probability with which the recording r belongs to the audio class b . For a comparison between the prediction and ground truth, the predicted probability has to be converted to a binary number (0 or 1). This is done by thresholding the probability. For example, if the threshold is kept at 0.5, then if the prediction is greater than or equal to 0.5 (say 0.7), the thresholded prediction will be 1, else the thresholded prediction will be 0.

	Actually TRUE	Actually FALSE
Predicted as TRUE	True Positives (TP)	False Positives (FP)
Predicted as FALSE	False Negatives (FN)	True Negatives (TN)
True Positive Rate: $TPR = \frac{TP}{TP+FN}$		False Positive Rate: $FPR = \frac{FP}{FP+TN}$

Figure 4.10 – True positive rate and false positive rate

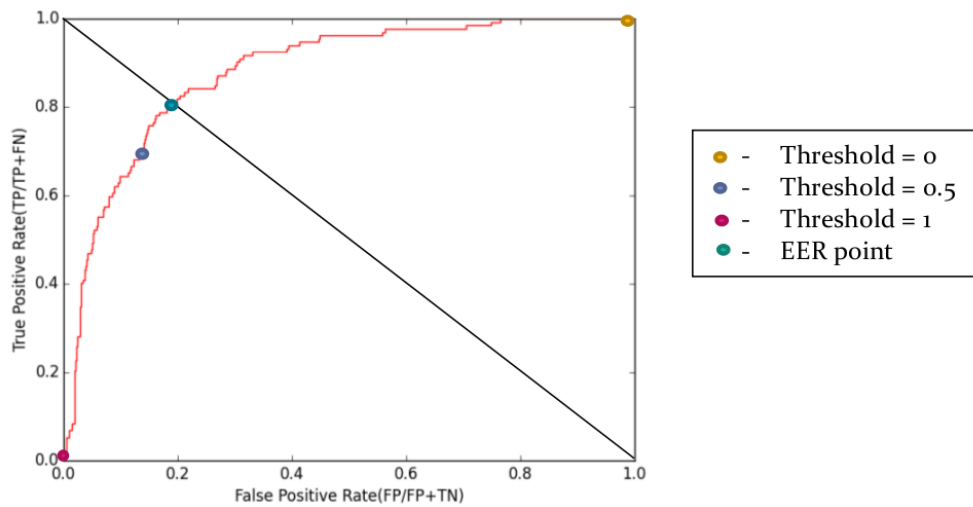


Figure 4.11 – ROC Curve and Equal Error Rate

A curve is plotted between the *true positive rate* (TPR) and the *false positive rate* (FPR) by varying the threshold from 0 to 1. This curve is known as the ROC curve. Figure 4.10 gives a summary about the TPR and FPR in binary classification. Figure 4.11 demonstrates a sample ROC curve and the EER point on that curve. EER is defined as the point on the ROC curve where the false rejection rate ($1-TPR$) equals the false acceptance rate (FPR). When performing k -folds cross validation, we provide the Standard Error (SE) measure along with the EER, where SE is the standard deviation in EER divided by the square root of k .

Table 4.6 – SVM class-wise as well as overall results (EER)

Audio Class	SVM (results from [33])	SVM (our results without compensation)	SVM (with compensation)
Crowd	19.82 ± 1.15	17.78 ± 1.19	17.45 ± 1.08
Applause	10.18 ± 0.81	12.49 ± 0.82	12.87 ± 1.07
Music	17.77 ± 1.15	37.83 ± 1.89	23.82 ± 1.82
Traffic	16.38 ± 0.77	15.69 ± 1.13	16.88 ± 1.22
Overall	16.04 ± 0.55	20.95 ± 1.26	17.75 ± 1.30

4.3.2 Results

4.3.2.1 Baseline: Support Vector Machine

SVM model parameters are mainly derived from [33]. We used an RBF kernel function (with $\gamma = 0.2$). We trained 4 different one-vs-all binary classifiers for the 4 classes: (i) Crowd, (ii) Applause, (iii), Music, and (iv) Traffic. We considered those audio recordings as the positive data samples for an audio class, which have at least one of its frames belonging to that audio class. Using EER along with SE as the error metric, we performed a 50-fold hold-out cross validation experiment with roughly 80% of training data and 20% of validation data in each fold.

Some of the audio recordings which belonged to *music* class in the original dataset used in [33], are no more available in the freesound repository. Hence we compensated for this by manually adding a proportionate length of randomly chosen musical recordings from the same website³. We obtained the results from our SVM model both with and without the externally added *music* audio recordings. The results are shown in table 4.6, where the error is represented as $EER \pm SE$. The overall results are calculated by simply averaging the EER and SE across all the 4 classes.

4.3.2.2 Convolutional Neural Network

Similar to the SVM model, we have 4 different CNN models for the 4 classes, each model being a binary classifier. A review of all the selected hyper-parameters for the CNN is presented in table 4.7. All the 4 CNN models are trained with the same set of hyper-parameters except the *applause* class, for which the CNN model is trained for 500 epochs (instead of 1000 epochs) because it learns faster as compared to the other class models. We performed a 50-fold hold-out cross validation experiment with roughly 80% of training data and 20% of validation data in each fold.

The cost function used is a modified cross-entropy function to account for the skew in data distribution between the positive and negative classes. We call it a weighted binary cross entropy function and its idea is derived from [33]. The formula for this error metric along with

³www.freesound.org

the weights used in the case of each class are shown in figure 4.12. The weights of the classes are inversely proportional to the size of the respective class. In the following, we discuss the results of each of the 4 models starting with *crowd*, followed by the overall (averaged) results.

Table 4.7 – Hyper-parameter setting for the CNN model

Hyper-parameter	Value
Feature type	MFCCs, delta MFCCs, delta-delta MFCCs
Number of input feature maps	24
Number of output feature maps	4
Number of convolutional layers	1
Number of pooling layers	1
Number of hidden/affine layers	1
Pooling type	Max-pooling
Kernel Size	(8X1)
Activation function for the hidden and output layers	Tanh
Number of units/neurons in the output layer	1
Learning Rate for training	0.003
Number of epochs of training	1000
Batch size	50
Number of training data samples	3800
Number of validation data samples	800
Cost function for training	Weighted Binary cross entropy
Error metric for training and validation	Weighted Binary cross entropy

Weighted Binary Cross-entropy: Weighted with coefficient 'k' inversely proportional to the class size				
class	Crowd	Applause	Music	Traffic
k	4.5215	7.6522	7.4676	6.0012
$Error = k * (-y * \log(y^{pred})) + (-(1 - y) * \log(1 - y^{pred}))$				

Figure 4.12 – Weighted Binary Cross-Entropy

Crowd model Figure 4.13 shows the learning curve and the epoch-wise EER error obtained on the training and validation sets for one of the 50 folds. As can be seen from the EER curve, the best EER obtained on the test data is 9.18%. The learning curve hints towards training the network for more than 1000 epochs, but upon training the same network for 5000 epochs, we obtained the same best EER on the test data as in the case of 1000 epochs.

Figure 4.14 shows the 50-folds cross-validation results for this model. The error for each fold in the figure is the best EER (in %) obtained on the test data for that fold. We obtain an overall result ($EER \pm SE$) of 14.86 ± 0.86 from the 50 fold cross-validation experiment which is a considerable improvement over the EER of 17.45% obtained from the SVM model.

Traffic model Figure 4.15 shows the learning curve and the epoch-wise EER curve obtained upon training the network for one of the 50 folds. From the learning curve, it can be seen that after around 200 epochs, the network starts over-fitting to the training data. From the EER curve, it can be seen that we obtain the best EER of 15.25% on the test data for this fold.

Figure 4.16 shows the 50-fold cross-validation results for this model. We obtain an overall result ($EER \pm SE$) of 11.77 ± 0.72 , which is much better than the EER of 16.88% obtained using the SVM model.

Applause model As stated earlier, this model is trained only for 500 epochs contrary to 1000 epochs in the case of other models because the CNN model for this class was observed to be quick to train and would generally start to overfit rapidly beyond 500 epochs. Figure 4.17 shows the learning curve as well as the EER curve obtained by training the model for one of the 50 folds. From the learning curve it is evident that the network starts overfitting to the training data by the end of 200 epochs. As can be seen from the EER curve, we obtained the best EER of 12.72% on the test data for this particular fold.

Figure 4.18 shows the 50-fold cross-validation results for this model. We obtain an overall result ($EER \pm SE$) of 4.87 ± 0.44 , which is much better than the EER of 12.87% obtained using the SVM model.

Music model Figure 4.19 shows the learning curve and the EER curve obtained by training this CNN model with one of the 50 folds for 1000 epochs. The learning curve clearly indicates that network could be trained for more epochs. Hence, we trained it further for 5000 epochs. Figure 4.20 shows the EER curve for 5000 epochs. For this particular fold, we obtained the best EER of 15.89% when trained until 1000 epochs and the best EER of 15.54% when trained until 5000 epochs.

Figure 4.21 shows the 50-fold cross-validation results for this model. We obtain an overall result ($EER \pm SE$) of 24.04 ± 1.23 , which is slightly poorer than the EER of 23.82% obtained using the SVM model.

Overall Results Comparison: CNN vs SVM Table 4.8 shows the comparison of results of all 4 classes obtained from the different models. The results are given as $EER \pm SE$, where EER is the equal error rate and SE is the standard error. It can be noted that the implemented CNN model performs best for all classes except *music*. Interestingly, the results of the CNN model for *music* class are even poorer than our own implemented SVM model. The results of the experiments performed in [33] also show that for the *music* class, SVM performs better than DNN, which seems to concur with our findings.

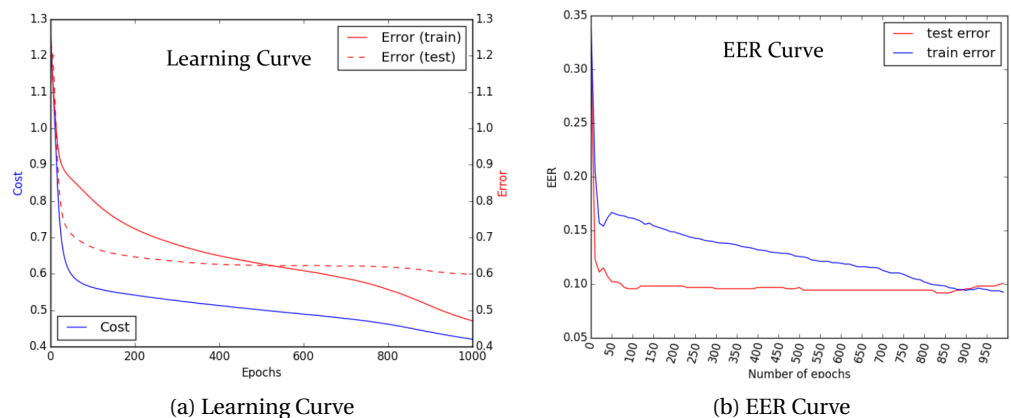


Figure 4.13 – Learning Curve and EER Performance of *Crowd* CNN model

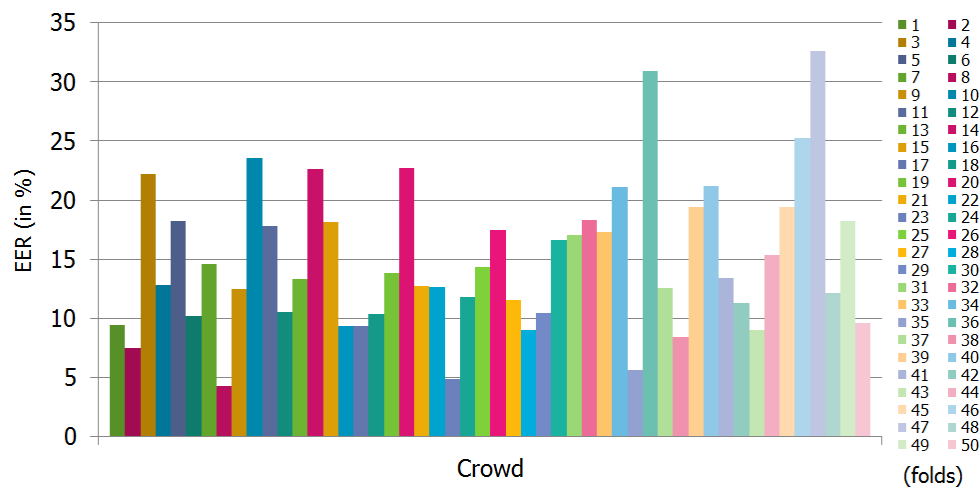


Figure 4.14 – *Crowd* model: 50-fold cross validation results

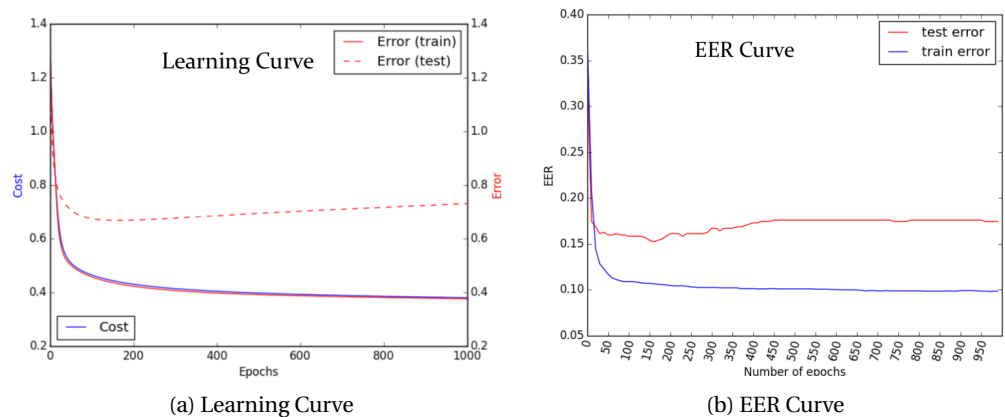
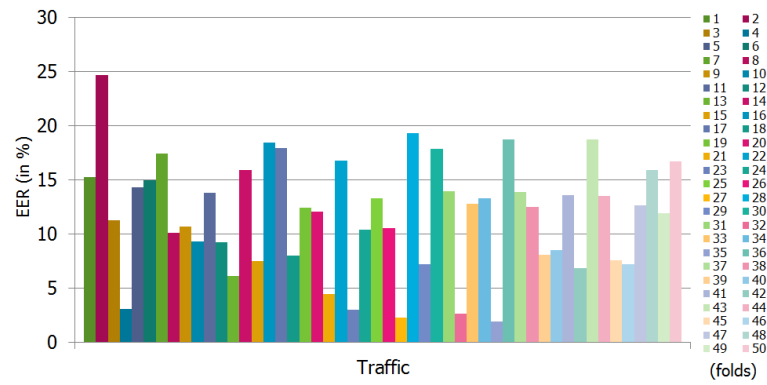
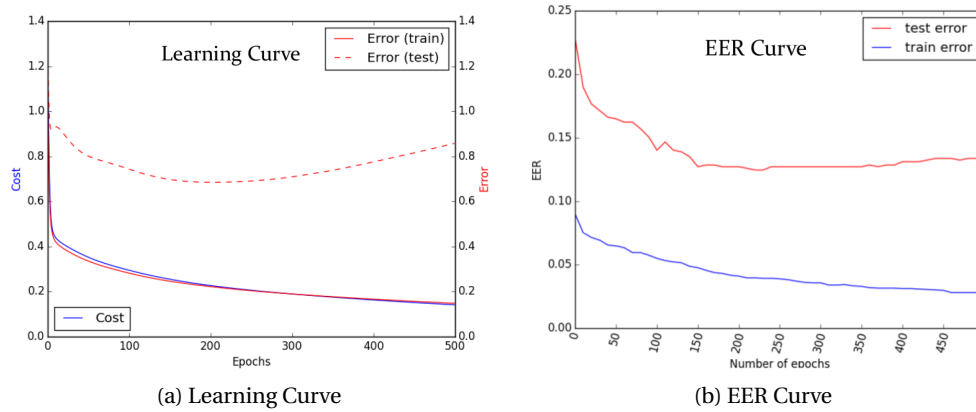
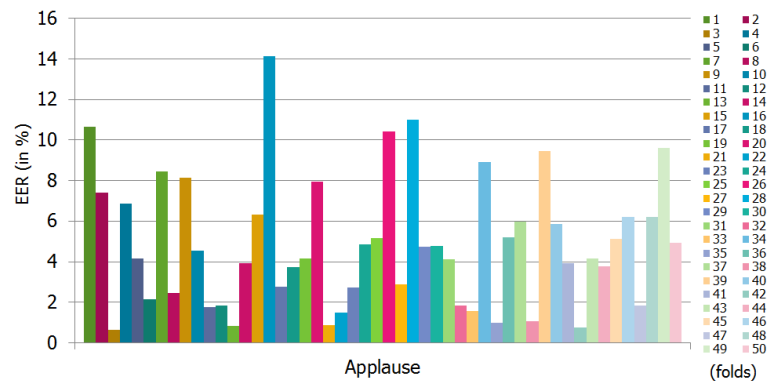


Figure 4.15 – Learning Curve and EER Performance of *Traffic* CNN model

Figure 4.16 – *Traffic* model: 50-fold cross validation resultsFigure 4.17 – Learning Curve and EER Performance of *Applause* CNN modelFigure 4.18 – *Applause* model: 50-fold cross validation results

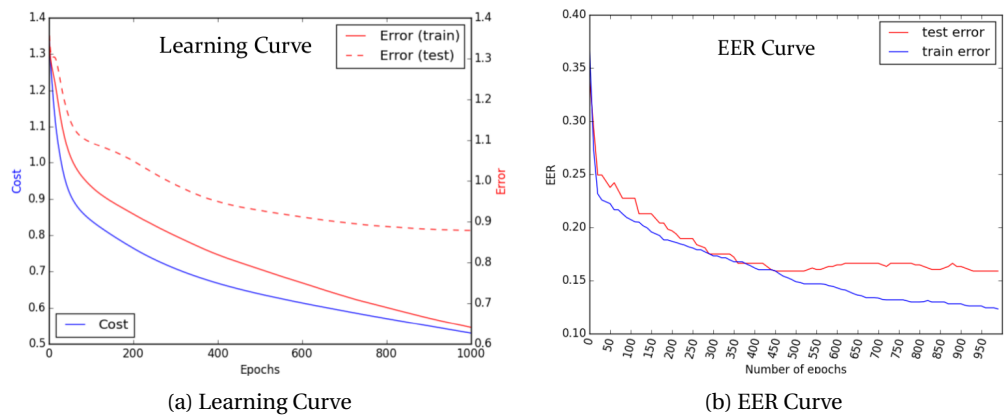


Figure 4.19 – Learning Curve and EER Performance of *Music* CNN model

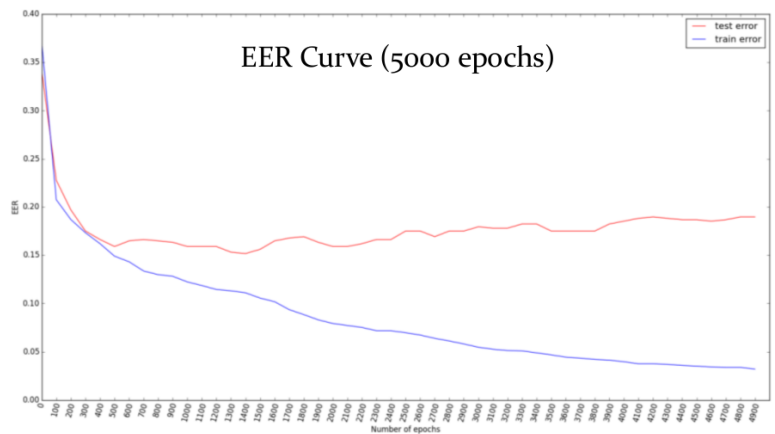


Figure 4.20 – *Music* model - EER curve for 5000 epochs

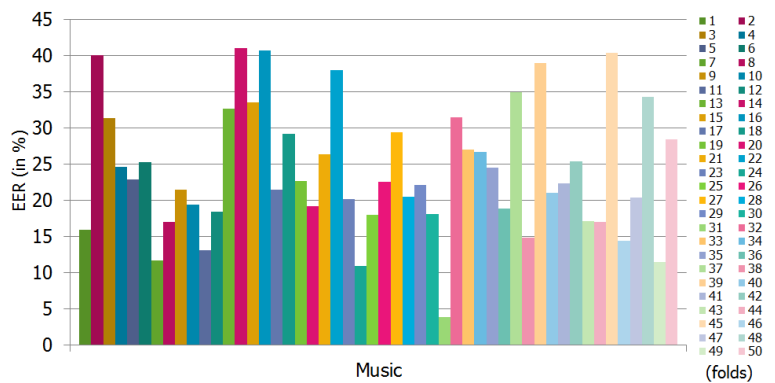


Figure 4.21 – *Music* model: 50-fold cross validation results

Table 4.8 – Results comparison: CNN and SVM

Audio Class	SVM (results from [33])	SVM (our implementation)	CNN
Crowd	19.82 ± 1.15	17.45 ± 1.08	14.86 ± 0.86
Applause	10.18 ± 0.81	12.87 ± 1.07	4.87 ± 0.44
Music	17.77 ± 1.15	23.82 ± 1.82	24.04 ± 1.23
Traffic	16.38 ± 0.77	16.88 ± 1.22	11.77 ± 0.72
Overall	16.04 ± 0.55	17.75 ± 1.3	13.89 ± 0.81

One thing is however clearly visible that compared to the SVM results of [33], our (SVM and CNN) results for the *music* class are poorer. This is, as stated earlier, primarily due to loss of some *music* data from the original dataset.

4.3.2.3 Standard baseline comparison

We used SVM as the classifier, AUC as the evaluation metric and then conducted a 10-fold cross validation experiment to obtain a standard baseline result on this dataset. This result is then compared with our CNN result. Details about using AUC as the evaluation metric are discussed in section 4.4.1. Table 4.9 shows the results of all 4 audio classes expressed as $AUC \pm STD$, where AUC and STD are respectively the mean and standard deviation in the AUCs across the 10 folds, and both AUC and STD are expressed in percentages.

Table 4.9 – Standard baseline comparison: CNN and SVM

Audio Class	SVM	CNN
Crowd	89.51 ± 8.71	92.03 ± 5.61
Applause	92.95 ± 7.09	97.82 ± 1.41
Music	79.57 ± 10.37	84.83 ± 6.21
Traffic	87.80 ± 10.06	92.28 ± 4.20
Overall	87.46 ± 9.06	91.74 ± 4.36

From the results, it can be seen that CNN performs better than SVM in all the audio classes. More importantly, the standard deviation in the CNN performance is consistently lower than SVM across the 10 folds. This shows that CNN does not just perform better than SVM, but is also more robust against variations in the audio data. This characteristic is important for a classifier to make it usable for practical applications.

4.3.2.4 Data Augmentation

After applying the CNN model to the dataset that was available, we attempted to artificially augment the data in order to make the model more robust against variations in the recordings. Also, the original dataset does not have enough data samples to fully realize the potential of a

CNN or DNN, hence it was a motivation to increase the amount of data.

Table 4.10 – Hyper-parameter setting: CNN with data augmentation

Hyper-parameter	Value
Learning Rate	0.003
Batch Size	50
Number of epochs	500
Original training samples	3300
Validation samples	800

As stated in section 3.3, we used 6 different ways of replicating the recording with label-preserving variations: (i) Changing pitch, (ii) Adding echo, (iii) Reducing noise, (iv) Adding tremolo effect, (v) Adding noise, and (vi) Adding reverb. Trying all the different permutations of these 6 kinds of data augmentations demanded a long time and/or heavy computational power, hence, doing a 50-fold cross-validation experiment each time became unfeasible. So, we experimented with each of these methods only for a single split of training and validation data. Most of the CNN hyper-parameters are the same as those used in our earlier CNN experiments with this dataset. Table 4.10 reviews briefly the set of hyper-parameters used in this experiment.

We trained 8 networks for each of our 4 classes. The only difference between these 8 networks was the input (training data) fed to them. The term *original* is used hereafter to represent the original dataset that we obtained from [33]. Using the 6 above-mentioned ways of augmenting data, we created 6 additional copies of our dataset by applying the 6 different kind of augmentation techniques one-by-one to the original dataset. For convenience we name these 6 new copies of dataset as: (i) *echo* - when echo is added to the original data, (ii) *noise-addition* - when noise is added to the original data, (iii) *reverb* - when reverb is added, (iv) *noise-reduction* - when noise reduction is applied to the original data, (v) *pitch* - when the pitch of original recordings is increased by 20%, and (vi) *tremolo* - when the tremolo effect is added to the original recordings. Hence the 8 networks that we trained have the following respective input data:

- (I) *original*
- (II) *original + echo*
- (III) *original + noise-addition*
- (IV) *original + reverb*
- (V) *original + noise-reduction*
- (VI) *original + pitch*
- (VII) *original + tremolo*
- (VIII) *original + echo + noise-addition + reverb + noise-reduction + pitch + tremolo*

Figures 4.22, 4.23, 4.24 and 4.25 show the 8 different EER curves for each of the 8 models for all 4 classes: *crowd*, *traffic*, *applause* and *music* respectively. Note that in the figures, *everything* refers to the case VIII where all six augmented datasets along with the original dataset is fed to the CNN. From the EER curves, it can be observed that in general, the network overfits to the training data but the network performance on the testing data is different for different kinds of data augmentation techniques. For e.g., if we look at the EER curves for *noise* augmentation, the testing error (colored red) strongly decreases in case of *applause* and *music* models. Also, it can be seen that although the various data augmentation techniques have positive effects on some of the models, however, for *traffic* class, we do not observe any improvement due to data augmentation.

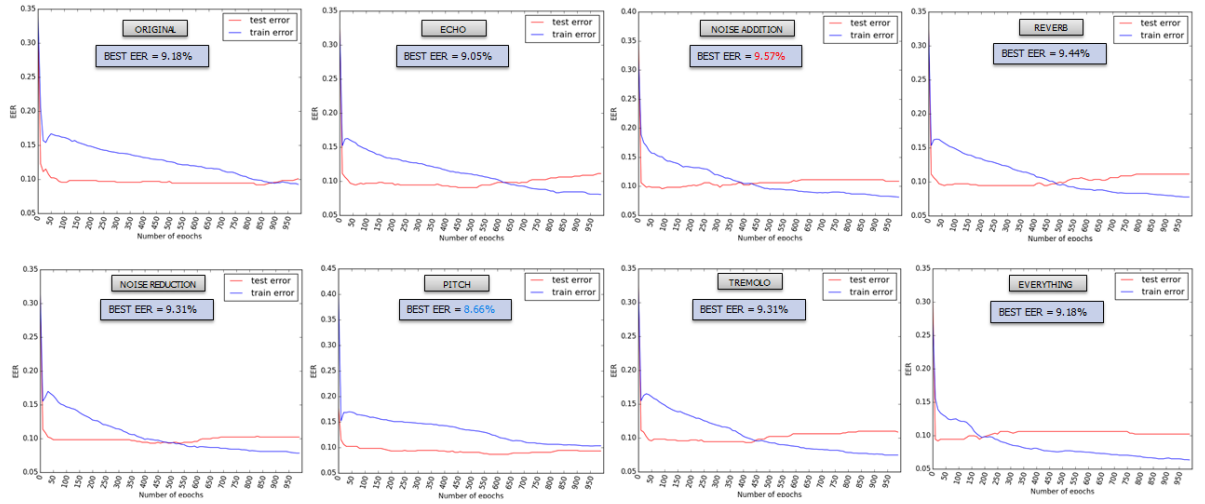


Figure 4.22 – EER curves for the 8 different CNN models with data-augmentation for *Crowd* class

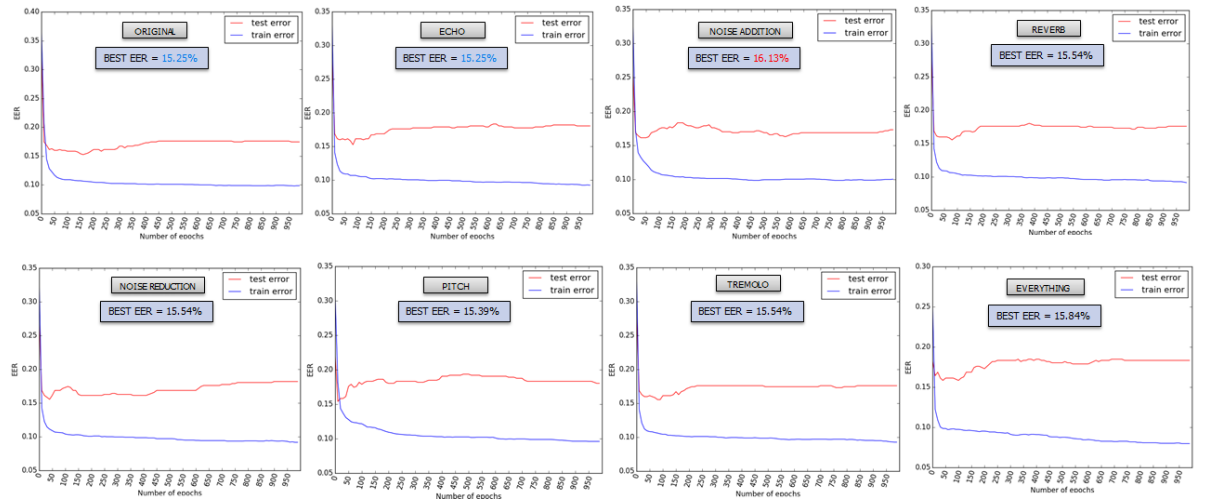


Figure 4.23 – EER curves for the 8 different CNN models with data-augmentation for *Traffic* class

Chapter 4. Evaluation and Results

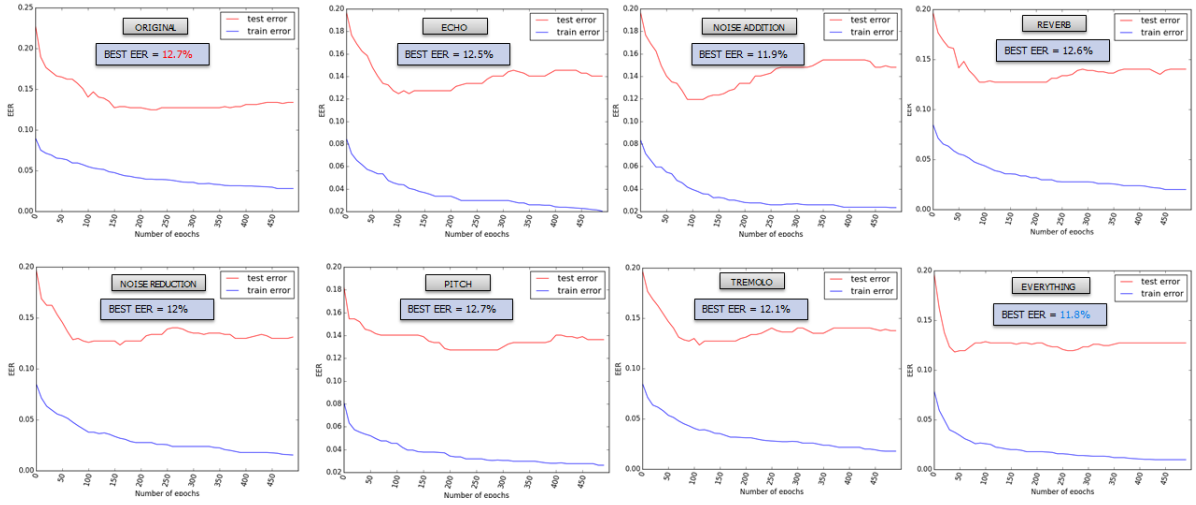


Figure 4.24 – EER curves for the 8 different CNN models with data-augmentation for *Applause* class

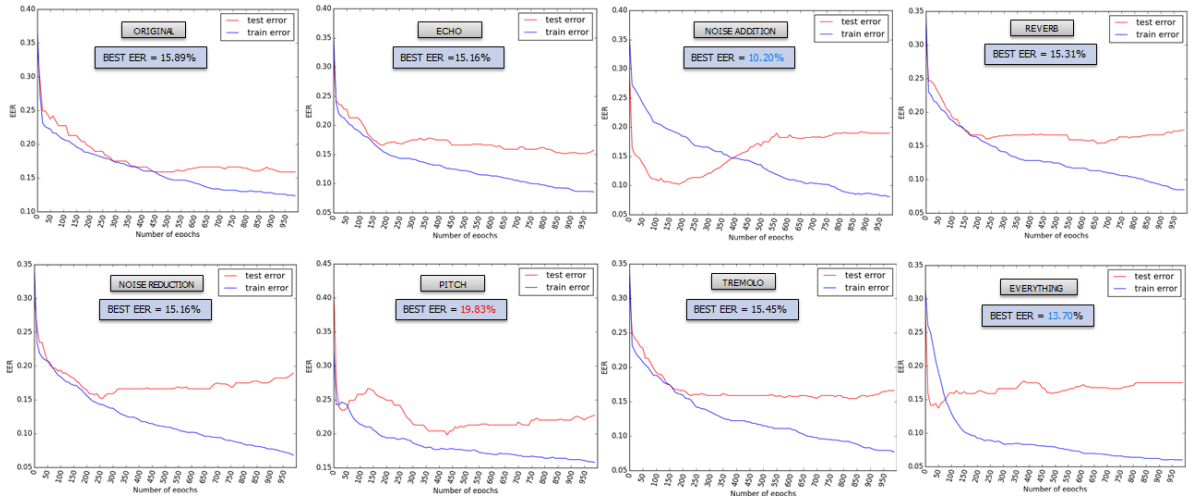


Figure 4.25 – EER curves for the 8 different CNN models with data-augmentation for *Music* class

Figure 4.26 compares the different augmentation approaches for all 4 audio classes. While there are notable improvements in performance using, for example, *noise-addition* or *noise-reduction* methods for the *applause* class, or *pitch* method for *crowd* class, etc. but there are also cases where data augmentation does not help, for example, in the *traffic* class. These results are obtained using a single fold experiment, and hence it is difficult to generalize any inferences that we make from the observations. For a better understanding of the effect of audio data augmentation, it would require a set of experiments where (i) different parameter settings of different augmentation techniques are tried (e.g. increasing the pitch by 10%, 20%, 30% and so on), and (ii) a k-fold cross validation experiment is conducted in order to have a better generalization of results.

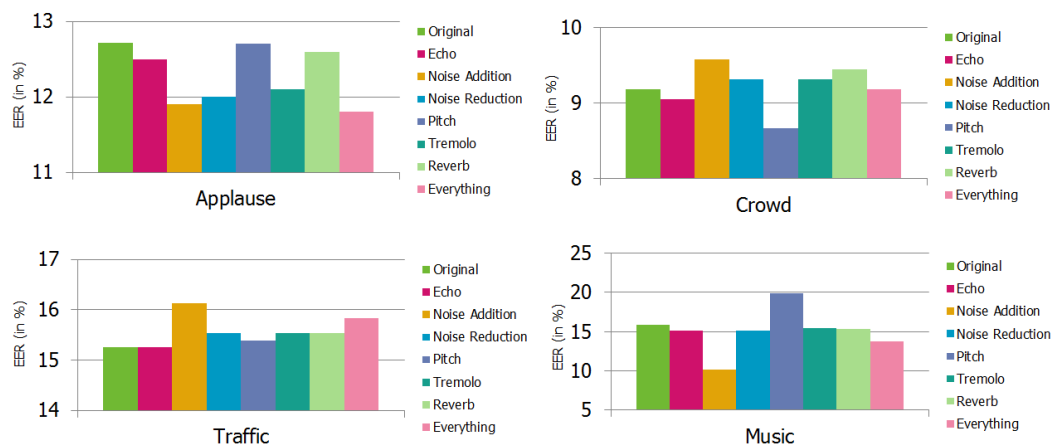


Figure 4.26 – Overall results comparing different data augmentation methods across the 4 audio classes

4.4 Dataset-III

The CHiME dataset is another dataset useful for performing polyphonic audio event detection. As discussed earlier, this dataset is composed of sounds from a domestic/home environment. The dataset consists 7 different audio classes which are reviewed in table 4.11.

First we tried to reproduce the baseline results published in [35] for this database using GMM. Then we examined the results produced by our implemented CNN model on this database. Finally, we compared the performance of CNN with a standard baseline, using SVM as the classifier and AUC as the evaluation metric. For the CNN model, we trained 7 different one-vs-all binary classifiers, one for each class. This is similar to the approach we followed for dataset-II.

In the following, we discuss the evaluation procedure that we used to replicate the baseline results of [35]. Then, we present the results of our CNN model and compare these results with the reproduced baseline results. Finally, we compare the CNN performance with a standard baseline result.

Table 4.11 – CHiME dataset - audio classes (review)

Label	Description
c	Child speech
m	Adult male speech
f	Adult female speech
v	Video game/TV
p	Percussive sounds, e.g. crash, bang, knock, footsteps
b	Broadband noise, e.g. household appliances
o	Other identifiable sounds

4.4.1 Evaluation Procedure

Motivated by [35], we use AUC - area under (ROC) curve as the evaluation metric to evaluate the performance of our models. In the following we explain how the AUC is computed.

Besides EER (discussed in section 4.3.1), AUC is another reliable metric to estimate the classification accuracy of a binary classifier when there exists a skew in data distribution between the positive and negative classes. When we train 7 different classifiers per class, the number of positive data samples is much smaller than the number of negative data samples. Hence, AUC is a promising error metric for our case.

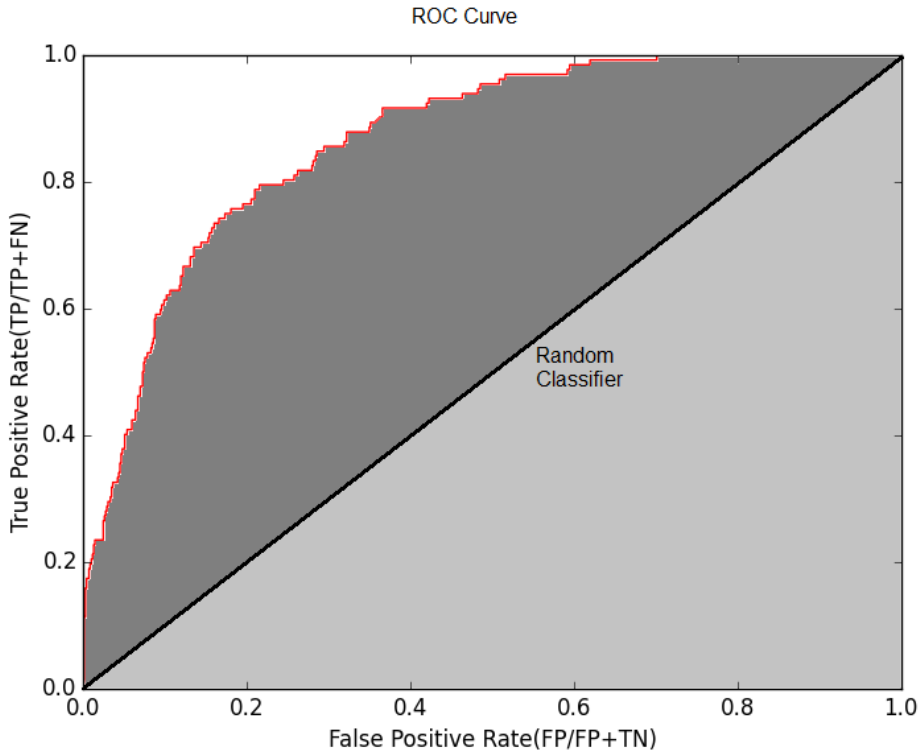


Figure 4.27 – Area Under ROC Curve

In this dataset, the audio recordings are 4 seconds long. Each audio recording is fed as one training data instance to the classifier (CNN) which then outputs a probability. This probability indicates whether the recording belongs to the positive or the negative class depending on a high value (say 0.9) or a low value (say 0.1) respectively. The ROC (Receiver Operating Characteristic) curve is plotted between the *true positive rate* (TPR) and the *false positive rate* (FPR), which are computed by the comparison of model predictions and the ground truth (details about ROC curve discussed in section 4.3.1). AUC is the area under this computed ROC curve. The higher the area, the better the classification performance of the classifier. A purely random classifier would have an AUC score of 50% and an ideally perfect binary classifier would have an AUC score of 100%. Figure 4.27 shows the AUC for a binary classifier

and a random classifier. It can be seen that the area under curve for the random classifier is 50%. Next, we discuss the two different ways in which we calculated the AUC in a 10-fold cross validation experiment.

1. **Method 1** - Similar to [35], we perform 10-fold cross validation, holding one out of the 10 folds of data as the testing/validation set. By doing this repeatedly 10 times for the 10 folds, we get the predictions for the 10 different folds of testing data. In the end we concatenate the predictions of all these 10 folds of testing data (which ultimately equals to the full dataset) and then compute the AUC by comparing these predictions to the ground truth value. This is done for each of the 7 classifiers, hence, we obtain 7 different AUC scores, one for each classifier.
2. **Method 2** - Method 1 seems an inappropriate way of evaluating the performance of a classifier from a 10 fold cross-validation experiment because in each fold, the training dataset is different and hence the model weights, that are learned, are different. Each of the 10 binary classifiers may then have a different optimal threshold (the probability value above which all predictions are 1 or TRUE) and hence, combining the predictions of all such classifiers to calculate AUC would require the 10 individual classifiers to compromise for a lower threshold value which can be called as the **common optimal threshold** for all the 10 classifiers. This optimal threshold is not the same as the individual optimal thresholds of each of the 10 classifiers. It would be equivalent to saying that the average of 100 numbers is equal to the average of the averages of these 100 numbers grouped in the set {20; 10; 30; 15; 25}.

Hence, we followed the standard method used in k-fold cross validation experiments, where we calculate the AUC in each fold i.e. each model (for each fold) is evaluated individually. Finally we take the mean (and standard deviation) of all the 10 AUC values to obtain the final AUC on the dataset. This method is more robust since we are not assuming that different classifiers can be judged on the same threshold scale and also this method is more transparent as it gives us the standard deviation in the AUC values which can give us an idea of how different are the folds from each other.

4.4.2 Results

We present the results on the baseline GMM model as well as our implemented CNN model using both the evaluation procedures - method 1 and method 2.

4.4.2.1 Results with method 1

Baseline: Gaussian Mixture Models We develop and train the GMM models as an attempt to reproduce the baseline GMM results published in [35]. We have 7 audio classes, hence, for each class we train a pair of GMMs, one pertaining to the positive class and the other to the negative class. As a part of the experiment, the number of gaussians are varied using the values in the set {1,2,4,8}. We estimate full-covariance

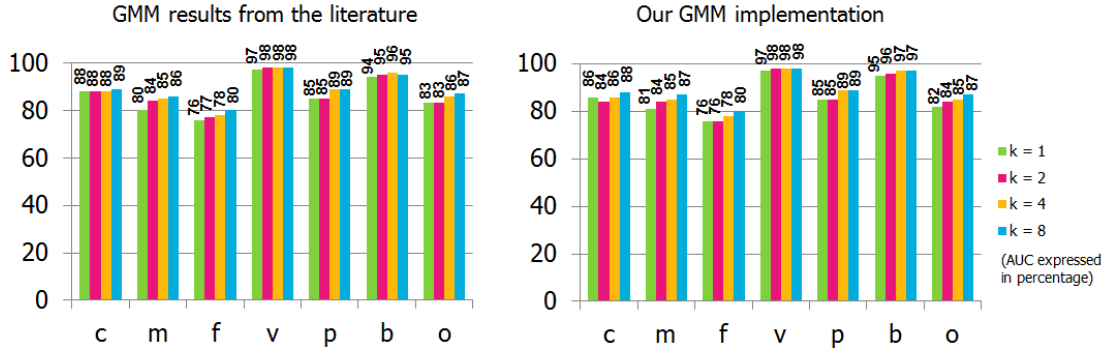


Figure 4.28 – GMM Results: Baseline results from literature (left), and results from our implemented GMM (right)

gaussians and finally for each audio class x and for each audio recording r , prediction is made by computing the log-likelihood ratio of the feature-vector corresponding to r , with respect to the pair of estimated GMMs. This prediction indicates the probability with which r belongs to class x .

As discussed in the previous section, we perform a 10-fold cross validation experiment for each of the 7 audio classes. The AUCs computed for each audio class are displayed in figure 4.28 along with the baseline results of [35]. It can be seen that our GMM implementation closely reproduces the results from [35]. The next step is then to train and test this dataset on our CNN model with the aim of getting better results than those obtained using GMM.

Convolutional Neural Network We train 7 different CNN models, one per audio class and perform a 10-fold cross-validation experiment as done in the GMM experiments. However, we tried a grid of hyper-parameter values for each of the 7 CNN models. Basically we experimented with 16 different permutations of controllable hyper-parameters for each of the 7 CNN models. Table 4.12 shows the 16 different sets of hyper-parameter values that we tried as our CNN model configurations.

Table 4.13 shows the best AUC performance (in %) obtained from these CNN model configurations along with the permutation IDs that indicate the sets of hyper-parameter values (from table 4.12) that give those best CNN results. In this table, we also compare these results with the best GMM baseline literature results for each of the 7 classes.

4.4.2.2 Results with method 2

Baseline: Gaussian Mixture Models Using the same model setting as in the previous GMM setup, we conducted a 10-fold hold-out cross validation experiment by keeping 90% of data in the training set in each fold. However, in this case, instead of accumulating the predictions of all the 10 folds and calculating the AUC on the whole predictions set, we calculated individual AUCs on predictions of each of the 10 folds.

Table 4.12 – Hyper-parameter grid for CNN model

ID	Learning Rate	Epochs	Batch-size	Output feature maps	Kernel size	Max-pool size
1	0.01	500	50	7	(7,1)	(10,1)
2	0.003	1000	50	7	(7,1)	(10,1)
3	0.003	1000	100	7	(7,1)	(10,1)
4	0.03	500	20	7	(7,1)	(10,1)
5	0.001	1000	50	7	(7,1)	(10,1)
6	0.01	500	50	26	(7,1)	(10,1)
7	0.01	500	50	15	(7,1)	(10,1)
8	0.01	500	50	7	(5,1)	(12,1)
9	0.01	500	50	7	(9,1)	(8,1)
10	0.01	500	50	15	(5,1)	(12,1)
11	0.01	500	50	4	(9,1)	(8,1)
12	0.001	1000	50	26	(14,1)	(11,1)
13	0.001	1000	50	10	(3,1)	(11,1)
14	0.01	500	50	7	(3,1)	(11,1)
15	0.01	500	50	5	(3,1)	(17,1)
16	0.01	500	50	13	(17,1)	(18,1)

Table 4.13 – CNN results in comparison with the best GMM results

Class label	Best GMM result	Best CNN result	Permutation IDs
c	89	89	(8,16)
m	86	81	(6,7,8,10,16)
f	80	77	(15,16)
v	98	99	(1,2,4)
p	89	87	(15,16)
b	96	97	(9,14,15)
o	87	84	(16)

Table 4.14 shows the results obtained in the form of the mean and standard deviation of AUC values (in %) for each of the 7 classes and for the different number of gaussians in the set {1,2,4,8}. These results are clearly different (poorer) than the GMM results that we obtained by using method 1 for evaluation. One reason here could be that the distribution of data into the 10 folds using our hold-out cross validation process results in a few “bad” folds which, in turn, results in a lower mean value of AUC with a high standard deviation.

Convolutional Neural Network Similar to the GMM experiments (using method 2), we conducted a 10-fold cross validation experiment with our CNN model to compare the performance of the CNN model with the GMM model. With the change in the evaluation procedure (from method 1 to method 2), we had to try new hyper-parameter

Table 4.14 – GMM results with evaluation method 2

Class label	Number of Gaussians			
	1	2	4	8
c	78.58 ± 17.56	80.50 ± 14.93	77.67 ± 17.29	77.96 ± 17.62
m	55.29 ± 28.44	56.75 ± 30.65	69.3 ± 25.83	70.12 ± 25.88
f	67.99 ± 14.28	62.93 ± 12.96	59.73 ± 17.78	67.61 ± 12.31
v	97.77 ± 2.24	97.42 ± 3.65	97.59 ± 3.67	97.70 ± 3.62
p	81.85 ± 13.15	82.42 ± 12.88	82.29 ± 12.66	83.04 ± 13.22
b	97.04 ± 6.14	97.30 ± 5.64	97.05 ± 6.58	96.71 ± 8.08
o	71.54 ± 12.31	73.83 ± 14.99	81.69 ± 10.85	80.83 ± 10.45

settings and by doing that, we found the architecture shown in figure 4.29 as the best performing CNN architecture. As can be seen from the figure, we kept the kernel size as (4,1) and 7 output feature maps after the convolutional layer which are then followed by a max-pooling layer of size 10. In the end, there is an affine (fully-connected) layer followed by a sigmoid transformation.

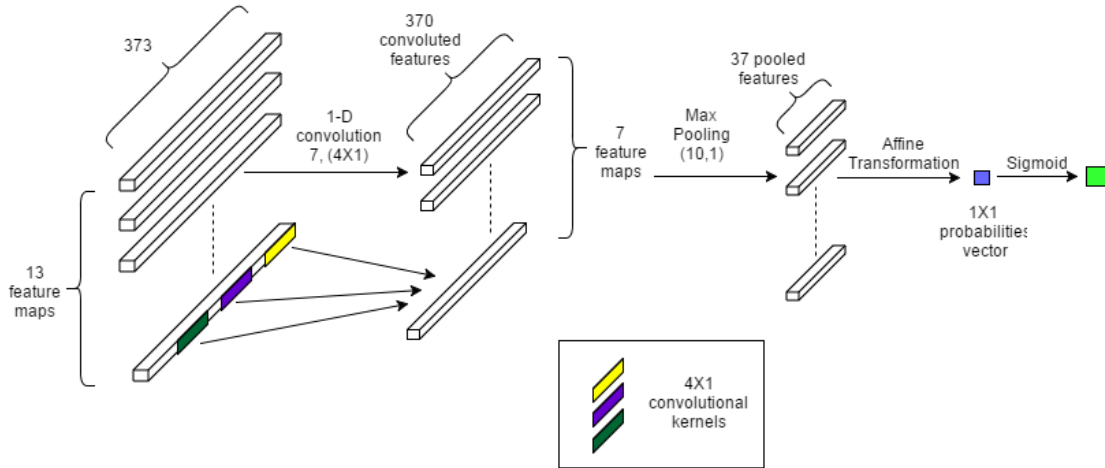


Figure 4.29 – CNN architecture for dataset-III

Figure 4.30 shows a comparison between the results of the CNN model and the GMM model with 8 gaussians, using method 2 as the evaluation procedure. The bars represent the mean value of the AUC obtained over the 10 folds of the cross validation. The exact values of the mean and standard deviations of the AUC for CNN model are shown in table 4.15 and for the GMM model, these values are listed in table 4.14. It can be seen that the CNN model outperforms the GMM classifier for most of the audio classes with this evaluation method.

The standard deviations in the AUC values are also lower in the CNN results which shows that the CNN results are relatively more stable across folds. This consistency in performance of a classifier across different folds of dataset is important for usage in

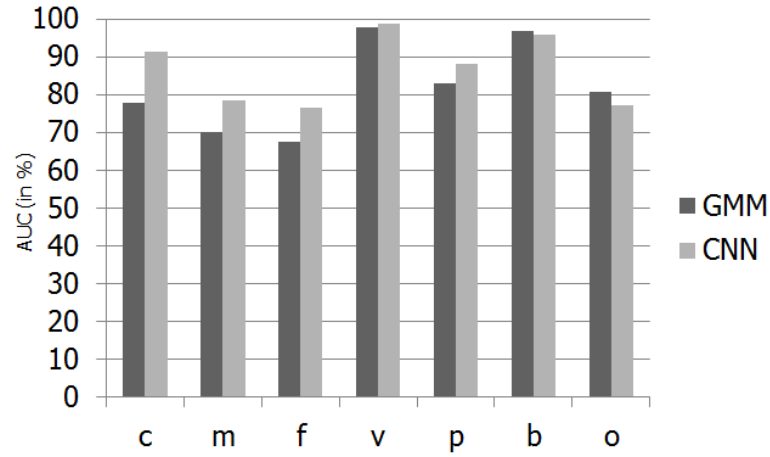


Figure 4.30 – GMM and CNN result comparison with evaluation method 2

practical applications. Because in a real-life scenario, a device which uses audio event detection for context recognition, must be robust to the variations in the quality and type of audio data that it receives. In our experiments, CNN has shown that robustness by maintaining a small variance in the accuracy across different folds.

4.4.2.3 Standard baseline comparison

We used SVM as the classifier (with $\gamma = 0.2$), AUC as the evaluation metric and conducted a 10-fold cross validation experiment to obtain a standard baseline result on this dataset. This result is compared to the CNN results obtained using method 2. Evaluation using AUC is done by following the procedure mentioned in section 4.4.1. Table 4.15 shows the standard baseline comparison results of the 7 audio classes expressed as $AUC \pm STD$, where AUC and STD are respectively the mean and standard deviation in the AUCs across the 10 folds, and both AUC and STD are expressed in percentages.

Table 4.15 – Standard baseline comparison: CNN and SVM

Audio Class	SVM	CNN
c	88.45 ± 1.96	91.64 ± 1.37
m	81.89 ± 5.80	84.12 ± 4.77
f	78.77 ± 5.89	79.33 ± 2.83
v	97.90 ± 1.09	98.65 ± 0.59
p	85.94 ± 3.39	89.33 ± 1.58
b	87.09 ± 12.34	94.53 ± 5.92
o	79.74 ± 5.40	81.79 ± 2.30

From the results, it can be seen that for all 7 audio classes, CNN performs better than the SVM baseline classifier. And also the standard deviation in CNN performance is always lower than SVM, which shows its robustness for these audio classes.

5 Conclusion

In this thesis, we tackled the problem of audio event detection in scenarios that closely resemble real-life events. We used three different kinds of datasets to cover different types of environments and to cover the cases of both, monophonic and polyphonic audio event detection. We used MFCCs for representing the audio data, and neural networks as classifiers for the classification task.

The three datasets respectively include sounds from urban environments, everyday outdoor sounds, and sounds within a domestic (home) environment. Each audio recording includes single (monophonic) or multiple (polyphonic) overlapping audio events, hence the task was to detect individual audio events from a mixture of audio events in a recording.

Different kinds of neural networks - DNN and CNN were proposed to detect these audio events. To find the best fitting parameters of these models, various experiments were conducted. The effect of varying several hyper-parameters to improve the neural networks' performance has been experimented. Different kinds of evaluation metrics were employed such as the area under ROC curve, equal error rate, etc. to measure the classification accuracy of our proposed classifiers and to compare our results with the baseline results. On two of the three datasets, our proposed neural network model surpassed the performance of the baseline classifiers such as SVM and GMM.

From the experiments, we conclude that using CNN as the classifier with MFCC as audio features gives a reasonably higher accuracy than the traditional classifiers like GMM or SVM. Our reasoning is that for traditional classifiers, the MFCCs across all the audio frames of a recording are usually aggregated to form a compressed feature vector. This leads to loss of information especially loss of the contextual information that flows between neighboring frames. In CNN, we feed an audio recording as a one dimensional image with pixels referring to the frames of the recording. CNN, then tries to find some local patterns within this image representation. This helps in retaining the useful contextual information as we are not aggregating information from all frames irrespective of the knowledge of the contents of each frame. We also notice that artificially augmenting the audio data does not necessarily always help and

depends on the method used to generate the new recordings from the already available recordings.

By trying different permutations and combinations of the hyper-parameter values for the CNN, we examined the effect that they have on the performance of the model. Finding an optimal range of these hyper-parameters can noticeably improve the learning process. We conducted a hyper-parameter grid-based experiment on the CNN model to obtain such an optimal range of hyper-parameters.

For future work, the CNN classifier accuracy can possibly be further improved by adding further optimizations like momentum, dropouts, weight decay, etc. to the learning procedure. The DNN model that we implemented suffered from over-fitting and hence regularization techniques could be applied to reduce the model variance. Finally, for polyphonic audio event detection, we implemented one-vs-all binary classifiers for each audio event class. This has a limitation that we neglect the possibility of correlation between occurrences of different audio classes. Therefore, a single multi-class classifier for all audio classes can be implemented and experimented-with in the future.

Bibliography

- [1] J. Biswas and M. Veloso, "Depth camera based indoor mobile robot localization and navigation," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pp. 1697–1702, IEEE, 2012.
- [2] J. Foote, "An overview of audio information retrieval," *Multimedia systems*, vol. 7, no. 1, pp. 2–10, 1999.
- [3] S. Chu, S. Narayanan, and C. J. Kuo, "Environmental sound recognition with time-frequency audio features," *Audio, Speech, and Language Processing, IEEE Transactions on*, vol. 17, no. 6, pp. 1142–1158, 2009.
- [4] H. D. Tran and H. Li, "Sound event recognition with probabilistic distance svms," *Audio, Speech, and Language Processing, IEEE Transactions on*, vol. 19, no. 6, pp. 1556–1568, 2011.
- [5] J. Salamon, C. Jacoby, and J. P. Bello, "A dataset and taxonomy for urban sound research," in *Proceedings of the ACM International Conference on Multimedia*, pp. 1041–1044, ACM, 2014.
- [6] D. Yu, L. Deng, and G. Dahl, "Roles of pre-training and fine-tuning in context-dependent dbn-hmms for real-world speech recognition," in *Proc. NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2010.
- [7] D. Ciresan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pp. 3642–3649, IEEE, 2012.
- [8] R. Collobert and J. Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning," in *Proceedings of the 25th international conference on Machine learning*, pp. 160–167, ACM, 2008.
- [9] D. Yu, M. L. Seltzer, J. Li, J.-T. Huang, and F. Seide, "Feature learning in deep neural networks-studies on speech recognition tasks," *arXiv preprint arXiv:1301.3605*, 2013.

- [10] O. Abdel-Hamid, A.-r. Mohamed, H. Jiang, L. Deng, G. Penn, and D. Yu, "Convolutional neural networks for speech recognition," *Audio, Speech, and Language Processing, IEEE/ACM Transactions on*, vol. 22, no. 10, pp. 1533–1545, 2014.
- [11] Q. Kong, X. Feng, and Y. Li, "Music genre classification using convolutional neural network,"
- [12] O. Gencoglu, T. Virtanen, and H. Huttunen, "Recognition of acoustic events using deep neural networks," in *Signal Processing Conference (EUSIPCO), 2014 Proceedings of the 22nd European*, pp. 506–510, IEEE, 2014.
- [13] K. Yamamoto, F. Jabloun, K. Reinhard, and A. Kawamura, "Robust endpoint detection for speech recognition based on discriminative feature extraction," in *Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings. 2006 IEEE International Conference on*, vol. 1, pp. I–I, IEEE, 2006.
- [14] T. Hastie, R. Tibshirani, J. Friedman, and J. Franklin, "The elements of statistical learning: data mining, inference and prediction," *The Mathematical Intelligencer*, vol. 27, no. 2, pp. 83–85, 2005.
- [15] V. Tiwari, "Mfcc and its applications in speaker recognition," *International Journal on Emerging Technologies*, vol. 1, no. 1, pp. 19–22, 2010.
- [16] T. Kohonen, "State of the art in neural computing," 1987.
- [17] S. Haykin and N. Network, "A comprehensive foundation," *Neural Networks*, vol. 2, no. 2004, 2004.
- [18] S.-i. Amari, "Backpropagation and stochastic gradient descent method," *Neuro-computing*, vol. 5, no. 4, pp. 185–196, 1993.
- [19] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural computation*, vol. 18, no. 7, pp. 1527–1554, 2006.
- [20] Y. Bengio, "Deep learning of representations for unsupervised and transfer learning," *Unsupervised and Transfer Learning Challenges in Machine Learning*, vol. 7, p. 19, 2012.
- [21] G. F. Montufar, R. Pascanu, K. Cho, and Y. Bengio, "On the number of linear regions of deep neural networks," in *Advances in Neural Information Processing Systems*, pp. 2924–2932, 2014.
- [22] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Cognitive modeling*, vol. 5, p. 3, 1988.
- [23] D. Erhan, P.-A. Manzagol, Y. Bengio, S. Bengio, and P. Vincent, "The difficulty of training deep architectures and the effect of unsupervised pre-training," in *International Conference on artificial intelligence and statistics*, pp. 153–160, 2009.

- [24] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
- [25] A. Mesaros, T. Heittola, O. Dikmen, and T. Virtanen, "Sound event detection in real life recordings using coupled matrix factorization of spectral representations and class activity annotations," in *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*, pp. 606–618, IEEE Computer Society, 2015.
- [26] T. Heittola, A. Mesaros, A. Eronen, and T. Virtanen, "Context-dependent sound event detection," *EURASIP Journal on Audio, Speech, and Music Processing*, vol. 2013, no. 1, pp. 1–13, 2013.
- [27] T. Heittola, A. Mesaros, T. Virtanen, and M. Gabbouj, "Supervised model training for overlapping sound events based on unsupervised source separation.," in *ICASSP*, pp. 8677–8681, 2013.
- [28] J. Dennis, H. D. Tran, and E. S. Chng, "Overlapping sound event recognition using local spectrogram features and the generalised hough transform," *Pattern Recognition Letters*, vol. 34, no. 9, pp. 1085–1093, 2013.
- [29] A. Mesaros, T. Heittola, A. Eronen, and T. Virtanen, "Acoustic event detection in real life recordings," in *18th European Signal Processing Conference*, pp. 1267–1271, Citeseer, 2010.
- [30] M. K. Nandwana, A. Ziaei, and J. H. Hansen, "Robust unsupervised detection of human screams in noisy acoustic environments,"
- [31] X. Zhang, Q. He, and X. Feng, "Acoustic feature extraction by tensor-based sparse representation for sound effects classification," in *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*, pp. 166–170, IEEE, 2015.
- [32] J. Dennis, T. H. Dat, and H. Li, "Combining robust spike coding with spiking neural networks for sound event classification," in *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*, pp. 176–180, IEEE, 2015.
- [33] Z. Kons and O. Toledo-Ronen, "Audio event classification using deep neural networks.,"
- [34] H. Christensen, J. Barker, N. Ma, and P. D. Green, "The chime corpus: a resource and a challenge for computational hearing in multisource environments.," in *INTERSPEECH*, pp. 1918–1921, Citeseer, 2010.
- [35] P. Foster, S. Sigtia, S. Krstulovic, J. Barker, and M. D. Plumbley, "Chime-home: A dataset for sound source recognition in a domestic environment," in *Applications of Signal Processing to Audio and Acoustics (WASPAA), 2015 IEEE Workshop on*, pp. 1–5, IEEE, 2015.

Bibliography

- [36] Wikipedia, “Support vector machine — wikipedia, the free encyclopedia,” 2016. [Online; accessed 7-January-2016].
- [37] D. Stowell, D. Giannoulis, E. Benetos, M. Lagrange, and M. Plumbley, “Detection and classification of acoustic scenes and events,”
- [38] B. Defréville, F. Pachet, C. Rosin, and P. Roy, “Automatic recognition of urban sound sources,” in *Audio Engineering Society Convention 120*, Audio Engineering Society, 2006.
- [39] J.-J. Aucouturier, B. Defreville, and F. Pachet, “The bag-of-frames approach to audio pattern recognition: A sufficient model for urban soundscapes but not for polyphonic music,” *The Journal of the Acoustical Society of America*, vol. 122, no. 2, pp. 881–891, 2007.
- [40] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in neural information processing systems*, pp. 1097–1105, 2012.
- [41] J. Schlüter and T. Grill, “Exploring data augmentation for improved singing voice detection with neural networks,”
- [42] T. Ko, V. Peddinti, D. Povey, and S. Khudanpur, “Audio augmentation for speech recognition,” in *Proceedings of INTERSPEECH*, 2015.
- [43] A. P. Bradley, “The use of the area under the roc curve in the evaluation of machine learning algorithms,” *Pattern recognition*, vol. 30, no. 7, pp. 1145–1159, 1997.