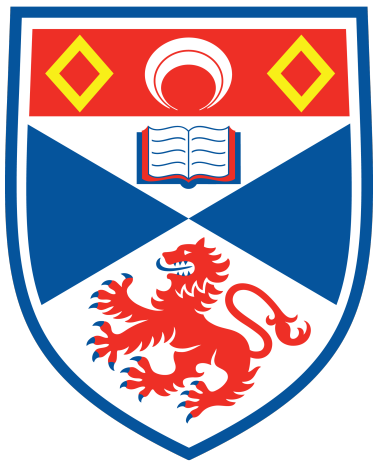
Robust Sound Event Classification Using Convolutional And Recurrent Neural Networks

Alexandros Michael - 160004864 Supervisor: Dr. Juan Ye

27 April 2020



CS4099 - Major Software Project

School Of Computer Science

University of St Andrews

1 Abstract

Audio is an information-rich medium that can be used to make sense of our environment, our immediate surroundings and to identify specific acoustic events. Ranging from applications in medicine to applications in environment detection, the domain of machine learning sound classification techniques is fairly broad. However, given the ubiquitous prevalence of smart, microphone enabled devices, it has the potential to rapidly expand.

Prior research in this field varies widely in approach depending on the intended use and the available datasets, using many different machine learning techniques. The focus of this project is to use two of the most promising neural network architectures, Convolutional and Recurrent Neural Networks, to classify 15 acoustic events that are typically associated with an office environment. Furthermore, we explore several data augmentation techniques, including the commonly used sound manipulation techniques and the novel SpecAugment. A number of experiments is conducted comparing the performance of multiple classification models which are evaluated on a manually recorded dataset, made to simulate a real-life deployment scenario.

Different approaches yielded very different results, some of which seem very promising. The LSTM network achieved the highest score, achieving a perfect classification accuracy of **100%** in one of our experiments, classifying all 56 manually recorded sound files correctly. In contrast, the highest score achieved by the CNN was a classification accuracy of **71.69%**.

2 Acknowledgements

I would first like to thank my supervisor, Dr Juan Ye, for all the tremendous help, insightful feedback and encouragement she has given me throughout the duration of this project, ensuring that I perform to the best of my abilities and making this work possible.

I would also like to thank my girlfriend Katherine, for her love, her encouraging words, her belief in me and for another one hundred and fourty three reasons that I can think of.

Lastly, I would like to thank my family for their unwavering support and love all throughout my life and the duration of my degree. To you, I owe everything that I am and that is why I dedicate this thesis to you.

3 Declaration

I declare that the material submitted for assessment is my own work except where credit is explicitly given to others by citation or acknowledgement. This work was performed during the current academic year except where otherwise stated. The main text of this project report is 11,371 words long, including project specification and plan. In submitting this project report to the University of St Andrews, I give permission for it to be made available for use in accordance with the regulations of the University Library. I also give permission for the report to be made available on the Web, for this work to be used in research within the University of St Andrews, and for any software to be released on an open source basis. I retain the copyright in this work, and ownership of any resulting intellectual property.

4 List of Tables

5 List of Figures

Contents

6 List of Abbreviations

AANN	Autoassociative Neural Network
ANN	Artificial Neural Network
CNN	Convolutional Neural Network
CQT	Constant Q-Transform
CWT	Continuous Wavelet Transform
DNS	Dominant Neighborhood Structure
DWT	Discrete Wavelet Transform
FFT	Fast Fourier Transform
GFCC	Gammaton Frequency Cepstral Coefficients
GMM	Gaussian Mixture Model
HMM	Hidden Markov Model
k-NN	k Nearest Neighbours
LPC	Linear Prediction Coefficients
LPCC	Linear Prediction Cepstrum Coefficients
MFCC	Mel Frequency Cepstral Coefficients
MLP	Multi-Layer Perceptron
PSG	Polysomnography
RNN	Recurrent Neural Network
SVM	Support Vector Machine
WT	Wavelet Transform

7 Introduction

7.1 Motivation & Applications

While the various Machine learning technologies and methodologies used to make use of the information conveyed by sound have been quickly advancing, the field of sound classification hasn't been researched nearly as much as the related field of image classification. Indeed, when using Google Scholar[4] the search query 'machine learning sound classification' yields 661,000 results whereas the search query 'machine learning image classification' yields a staggering 2,710,000 results. This shows that this is an up and coming field with plenty of room for improvement and given the increased prevalence of microphone enabled devices in our day to day life, we now have an opportunity to build the necessary tools to make sense of sound as an information-rich medium for a number of uses.

Machine learning sound classification techniques have been used to solve problems in various domains. The authors of 'Sleep staging using nocturnal sound analysis. Scientific Reports'[1] used non-contact microphones to record audio signals of patients during a polysomnography (PSG) study and trained a feedforward neural network to perform macro sleep stage estimation, an essential component for analysing sleep patterns and sleep quality.

Another example of the usefulness of sound classification appears in 'Sound Classification in Hearing Aids Inspired by Auditory Scene Analysis'[2]. In this paper, various classifiers, including neural networks, are trained and evaluated in classifying various sound classes, with the aim of automating the process of choosing the appropriate hearing aid program according to the present auditory scene and thus improving the user experience of individuals with hearing disabilities.

Furthermore, applications appear in the field of cardiology, as presented in the paper 'Heart Sound Classification Using Deep Structured Features'[3]. In this paper, the authors present a solution in which a deep CNN is used to extract features from heart sound recording audio signals and then a support vector machine (SVM) is used to classify whether the heart sounds are considered normal or abnormal which could hint to the presence of an underlying heart condition.

7.2 Problem Identification

The problem addressed in this project is the problem of the classifying auditory events that would typically take place in an office setting, a problem which falls in the broader problem category of sound event classification. To tackle

this, we incorporated the use of an artificial neural network (ANN). Specifically, since pattern recognition lies at the heart of the problem we have used a convolutional neural network (CNN) due to its ability to extract features from high level representations of data, in conjunction with a fully connected feed-forward neural network to carry out the classification task. Furthermore, since sound events can be viewed as temporal sequences we have also explored the use a recurrent neural network (RNN), due to its ability to recognise patterns in sequences of data.

7.3 Approaches

In the article 'Noise-Robust Sound-Event Classification System with Texture Analysis'[5], the authors explained how they approached the sound-event classification problem for detecting respiratory disease in livestock and detecting faulty railway parts as an image recognition problem. Indeed, the first step of their system is used to convert the sound signals into two-dimensional gray-level images. Following that they performed feature extraction by using the dominant neighborhood structure (DNS) technique and finally, they used these features to train four classifiers, a CNN, an SVM, a k-nearest neighbours (k-NN) classifier and a decision tree classifier. The performance of these classifiers were then tested in the presence of environmental noise as well as white Gaussian noise.

The paper 'Environment Sound Classification using Multiple Feature Channels and Attention based Deep Convolutional Neural Network'[6] outlines a different and novel approach of using multiple feature channels. While most research uses the Mel-Frequency Cepstral Coefficients (MFCC) extracted from audio signals, the authors also use Gammaton Frequency Cepstral Coefficients (GFCC), the Constant-Q transfrom (CQT) and Chronogram feature channels as inputs to a deep CNN, employed for classification. This approach yielded highly impressive results, performing very well on a number of benchmark datasets such as the ESC-50 for which it achieved a 88.50% accuracy which is beyond the human accuracy for the same dataset, at 81.30%.

The paper 'Robust audio sensing with multi-sound classification'[7] describes an approach in which the MFCC features are extracted from the audio signals of each sound file that are in turn fed into a CNN. Following that, the features produced by the CNN were fed into binary classifiers, one for each class, arranged in a stacked manner to enable the system to perform multi-sound classification.

7.4 Challenges

One of the biggest challenges in solving problems in this domain is the problem of finding a well annotated sound dataset. This is the main challenge high-

lighted in the paper 'A Dataset and Taxonomy for Urban Sound Research'[8] which examines the barriers to researchers in the area of sound classification. Annotating real-world data requires large amounts of effort and it is for this reason that '...datasets based on field recordings tend to be relatively small.' And since machine learning methods need a large amount of data to perform adequately, the lack of large datasets set a limit on the quality of the solutions we can provide. Having said that, there are indeed ways to compensate for this issue, for example performing data augmentation on the dataset, a method which we will be discussing later on. Furthermore, the authors mention an additional challenge, that is the fact that there is a lack of a common vocabulary among researches when working with urban sounds. As a result, it becomes very difficult to compare results across different studies and it also makes it difficult for researches to navigate in the right direction.

7.5 Objectives

We can summarise the objectives of our project as follows:

- Build and train a classification model with the ability to classify auditory events belonging to 15 different sound categories that typically occur in an office environment.
- Compare the effectiveness of different types of ANN architectures at the above task. Specifically, we will be comparing the performance of a CNN with that of a RNN.
- Examine the use of existing, large-scale audio classification models and integrate them in our solution.
- Use various data augmentation techniques, allowing our models to remain effective despite a variety in sound properties such as pitch and volume.

In addition to the objectives listed above we set out to achieve the following secondary objectives:

- Explore the differences in performance of our models when evaluated on manually recorded sound and sounds taken from a sound effects repository.
- Compare the effects different augmentation techniques have on our models.

7.6 Our Solution

The obvious first step was to obtain the dataset with which we would be using for training our machine learning models. We used a similar dataset

used by Haubrick & Ye (2019), a dataset which composed of files taken from the Freesounds repository[10], which is essentially a free to use, collaborative sound effect and field recording database. Though sounds files taken from here may not fully represent sound as it would appear in a real-life scenario, for example due to the lack of background noise, papers such as 'Non-Speech Audio Event Detection'[11] show that using such a dataset can actually benefit machine learning techniques as the files taken from such a repository are (usually) labelled accurately and are recorded tightly around the sound event.

After having chosen the dataset, we carry out a preprocessing on all the sound files. We noticed that a lot of the sound files had significant segments of silence in them. As a result the first preprocessing step was to remove the silent segments from the dataset. Furthermore, we noticed that the sound files for some sound categories were a lot longer than others. This would mean that the data distribution among sound categories would be very unbalanced. It became obvious that we would somehow need to overcome this issue so we opted for extending sound files that were less than 10 seconds long. Finally, in order to tackle the challenge of a dataset of limited size and to enable our model to generalise better on real sounds, we explored two different data augmentation techniques. The most obvious one was working directly on the sound files and manipulating their sound properties such as their volume and pitch. In addition we attempted to use the technique introduced in the paper 'SpecAugment: A Simple Data Augmentation Method for Automatic Speech Recognition', a technique that is applied on the inputs of our neural network.

We chose to extract the MFCC features from the audio signals, in order to conform with the input required by our chosen CNN that is used later in the pipeline. CNNs have been widely used in the domain of image and sound classification due to their ability to extract the most important features from the inputs. The specific CNN architecture we've chosen for our solution is the VGGish model[13], which, as the name suggests, is based on the popular VGG network[14], used for image recognition.

For the purposes of this project we've also explored to use of an RNN for classification. This is because sound has a temporal component and thus naturally falls into the domain of data that RNNs are used for. While we still utilise the CNN in this approach, we only use it for its feature extraction capabilities and the actual task of classification is done by the RNN.

To evaluate the performance of our system using various combinations of choices, ranging from the type of data augmentation performed to which classifier is used, we carried out several experiments in which we used a manually recorded dataset as our evaluation dataset. This allowed us to see how well our models generalised to 'real-life' sounds in a potential deployment scenario.

8 Context Survey

In the following section we will be exploring in detail how various researchers have gone about solving different aspects of the classification problem.

8.1 Dataset

During our research, we've come across different methods of collecting the data required to train our machine learning models. For example, Sharma, Granmo & Goodwin (2020) have used pre-existing environment sound classification benchmark datasets, namely UrbanSound8K, ESC-10 and ESC-50. These datasets are composed of sound files belonging in 10 categories in the case of the former two and 50 in the case of the latter. Such datasets, due to the fact that they are known benchmarks, allow for researchers to compare and contrast the performance of various approaches. For example the `README` file of the ESC-50 Github repository[15] contains a table of various papers that have used the dataset along with the classification accuracy they have achieved. The problem with using a benchmark dataset is obviously that it doesn't solve the problem of training a machine learning model for a specific domain such as the classification of sounds that would occur in an office space. Furthermore, if the datasets themselves are poorly put together then all the reported results of research that is based on these datasets might not be fully reliable. For example in the same `README` file mentioned, it is stated that due to an unnoticed mistake in the preprocessing step data leakage might occur while training the classification models.

On the other side of the spectrum, Dafna, Tarasiuk & Zigel (2018) describe that they followed a data collection process by using a digital audio recording device with a directional microphone to acquire all the data needed to carry out their methodology. While this allowed them to maintain a problem-domain specific approach, one could argue that the data available to them was perhaps not enough to adequately train a classifier, as their dataset consisted of the recordings of 250 patients who were referred to a polysomnography study. Furthermore, in contrast to benchmark datasets, datasets such as the one used for this study are not publicly available and thus it makes it very difficult to replicate the reported results and/or compare this approach with a different one.

8.2 Data Augmentation

Techniques used to manipulate input data are widely used by researches, as it provides useful functions such as expanding the dataset, in cases where that

would be somewhat limited, and it also enables classification models to perform better despite the heterogeneity of sounds and their properties.

The technical report titled 'DCASE 2019 Challenge Task 5: CNN + VGGish'[17], describes that the authors subjected their raw input data to data augmentation by altering the pitch, the volume and by adding background noise. This was done to enable the model to generalise better after the training and validation process was complete. The motivation behind adding background noise to the raw audio signals, comes from the idea that in real-life, there is almost always some background noise present in one form or another, whereas the dataset, which might consist of sound effects, might be too clean making it unrepresentative of reality. The next step before machine learning can take place is the step of converting the sound files into a representation that can be understood by the model. In this case the authors chose to convert the audio files into log-mel spectrograms with 128 mel bins, to conform to the input requirement of the VGGish model that is used later on.

Park D. et.al (2019) followed a different approach. While they still converted the input audio into log-mel spectrograms, they did not perform data augmentation such as pitch and volume shifting on the raw audio signals themselves. Instead they chose to deform the log-mel spectrograms by applying time-warping, that is deforming the spectrograms in the time direction and by performing time and frequency masking which involves masking blocks of consecutive time steps or mel frequency channels. Reportedly, this novel approach yielded surprisingly good results by turning the over-fitting problem prevalent in neural networks into an under-fitting problem, which can be remedied by making training periods longer, and by making networks deeper.

In a similar manner, 'Recurrent Neural Networks For Polyphonic Sound Event Detection In Real Life Recordings'[21] outlines a data augmentation approach that is directly applied on the features extracted from the sound files. Their approach consists of three simple transformations, namely time stretching, sub-frame time shifting and blocks mixing. Time shifting mimics the process of slowing down and speeding up recording by stretching or compressing spectrograms using linear interpolation. Sub-frame time shifting also uses linear interpolation however in this case frames are interpolated in-between existing frames and thus the frame rate remains remains the same. Finally, block mixing combines different parts of the sound signals belonging in the same context by overlapping their spectrograms, in order to create new samples.

8.3 Feature Extraction

There are several ways in which features can be extracted from sound signals. 'Classification of audio signals using AANN and GMM'[23] sets out to train and use an autoassociative neural network (AANN) and a Gaussian mix-

ture model (GMM) to automatically classify and categorise sounds in a multimedia database. There were three sets of acoustic features extracted, namely linear prediction coefficients (LPC), linear prediction derived cepstrum coefficients (LPCC) and mel-frequency cepstral coefficients (MFCC). With linear prediction, each sample is predicted as a linear weighted sample of past samples and then the LPC are calculated by minimising the mean squared error over an analysis frame. A recursive relation was then used to convert the LPC into LPCC. On the other hand, the MFCC features are computed from the fast Fourier transform (FFT) power coefficients, that are later filtered by a triangular bandpass filter bank. Once the three sets of features were extracted, they were used to train the classification models. The AANN and the GMM that scored the highest evaluation scores were the ones that were trained with the MFCC features reaching an accuracy core of 93.1% and 92.6% respectively. The next best accuracy scores were reached by the classifiers trained with the LPCC features, with the AANN scoring 92% and the GMM 90% accuracy. The models trained with the LPC features performed the worst out of the three, with the AANN scoring 88% and the GMM scoring 86%.

Another technique commonly used to extract features from sound signals is wavelet transform (WT). The authors of 'Classification of Heart Sounds using Discrete and Continuous Wavelet Transform and Random Forests'[24] obtained three sets of features derived from applying Discrete Wabelet Transforms (DWT) Continuous Wavelet Transforms (CTW) and a combination of the two, from a dataset consisting of heart sounds with the purpose of training a classifier with the ability to diagnose cardiac pathologies. A series of experiments showed that the approach that used DWT registered the best results outperforming the CWT as well as the hybrid approach.

8.4 Classification

Across the papers researched we have seen a number of different types of classifiers being used. Buchler et. al (2005) used and compared a number of classifiers, namely a minimum-distance classifier, a Bayes classifier, a multilayer perceptron (MLP) and a Hidden Markov Model (HMMs), with the latter essentially being a statistical method widely used for speech recognition. In this study, this technique performed the best and its success was attributed due to the fact that it accounts 'for the temporal statistics of the occurrence of different sates in the features.'

The authors of 'Acoustic events detection with Support Vector Machines'[22] used a different type of classifier, namely a SVM using the RBF kernel function, to accurately detect gunshots in a noisy environment that could indicate a potentially dangerous situation. While they found that such a classifier is very effective in classifying gunshot sound events in the absence of noise, reporting

an accuracy of 99.89% in one of the experiments, its performance drastically falls down to 38.99% in the presence of background noise.

While we have seen several types of classifiers being used with varying degrees of success, CNNs are far and beyond the most widely used classifiers across the papers surveyed[3][5][6][7][17]. The success of this type of neural network in the task of sound classification is attributed to their ability to extract high level features out of sound and its ability learn complex patterns. Having said that, there is evidence that other types of networks such as RNNs have the potential to become the go-to tool for solving problems in this space as studies report state of the art performance achieved by using them[25], something that is attributed to their unique ability to model sequences.

9 Design

9.1 Workflow Overview

The initial step in our workflow was choosing an appropriate dataset for our solution. In this project we are not so interested in comparing the performance of our solution against a benchmark, rather we are building a solution for performing sound event classification in a typical office environment. This is also the application domain of the solution outlined in Haubrick P. & Ye J. (2019) so we used a subset of the dataset that has been previously collected on Freesound. We opted to omit 5 sound categories out of the 20 as we did not believe that they strictly belong in the domain of office sounds. As part of the evaluation process we have also manually recorded sound events belonging in those categories using a handheld recorder.

Following that, we carried out a preprocessing procedure. In this step, we first performed some data cleaning by checking whether the individual sound files are correctly annotated and whether the sound quality was adequate. Then, we removed silent sections from all the sound files. We defined silent parts as audio chunks equal or longer than 1 second, since the log-mel spectrograms used as inputs for the VGGish represent 960ms of audio data. After that, we extended sound files shorter than 10 seconds to overcome the imbalance in data across sound categories. As we'll show in one of the later sections, some sound categories had multiple times as much sound data in terms of seconds compared to others. The next step was to perform data augmentation on these sound files. We performed a number of data augmentation techniques on each original sound file such as increasing and decreasing the pitch and the volume. As a result from each sound file resulted a total of 5, the original one plus 4 variations of it. Lastly, we extracted the log-mel spectrograms from each sound file and we stored them so that they may be used by the VGGish later on in the pipeline. Note that the SpecAugment data augmentation strategy happens 'online' during program execution.

The first thing that takes place during program execution is loading up the names of the files that will comprise both the training and the testing set. Then, the appropriate Machine Learning model will be compiled and the log-mel spectrograms will be fed into it and for training to take place. Once training is complete, the model will be evaluated using the reserved testing set which will contain data from sound files that were not involved in the training process in order to give an accurate estimate of how well the model generalises. The evaluation produces metrics such as an accuracy rate the f-1 score, precision and recall, a confusion matrix as well as a more detailed classification report. These are all used to determine what strategy works best and be the basis of our discussion as to why each approach performs a certain way.

9.2 Data Collection

The Freesound dataset consists of sound files belonging in 15 sound categories. As obtained, the dataset can be summarised by the following table:

Sound category	**Number of files**	**Total length**	**Source**
Chair	14	99 seconds	FreeSound
Clear Throat	14	35 seconds	FreeSound
Coffee Machine	14	466 seconds	FreeSound
Coughing	14	103 seconds	FreeSound
Door Knock	14	83 seconds	FreeSound
Door Slam	14	28 seconds	FreeSound
Drawer	14	55 seconds	FreeSound
Falling Object	14	26 seconds	FreeSound
Footsteps	14	296 seconds	FreeSound
Keyboard	14	255 seconds	FreeSound
Laughing	14	110 seconds	FreeSound
Milk Steamer	8	157 seconds	FreeSound
Sink	14	327 seconds	FreeSound
Sneezing	14	48 seconds	FreeSound
Stirring	14	169 seconds	FreeSound
Total	**204**	**2257 seconds/37.62 minutes**	

Table 1: Details of the FreeSound dataset.

As the table indicates, all of these sound files originate from Freesound[10]. As we previously mentioned, sounds taken from a source such as Freesound have several benefits. They are well labelled and they are recorded tightly around the sound event. Furthermore, the research supports that they can be effectively used as training data for classifiers. The sound categories were carefully chosen so that the solution can be geared towards use in an office environment. We have removed 5 sound classes that were otherwise used in Habrick P., & Ye J. (2019) and those are the categories of Car, Conversation, Dishwasher Phone Ringing and Photocopier.

In order to evaluate how well our model generalises to real sound data, it was necessary to manually record sound of the above 15 categories using a recorder. To undertake this task we used a Homder TF-85 handheld recorded and the data collected can be summarised as follows:

Sound category	Number of files	Total length	Source
Chair	4	22 seconds	Manually Recorded
Clear Throat	4	23 seconds	Manually Recorded
Coffee Machine	3	125 seconds	Manually Recorded
Coughing	4	22 seconds	Manually Recorded
Door Knock	3	21 seconds	Manually Recorded
Door Slam	4	15 seconds	Manually Recorded
Drawer	4	24 seconds	Manually Recorded
Falling Object	4	11 seconds	Manually Recorded
Footsteps	4	24 seconds	Manually Recorded
Keyboard	4	38 seconds	Manually Recorded
Laughing	4	29 seconds	Manually Recorded
Milk Steamer	3	28 seconds	Manually Recorded
Sink	4	34 seconds	Manually Recorded
Sneezing	4	22 seconds	Manually Recorded
Stirring	3	22 seconds	Manually Recorded
Total	**56**	**457 seconds/7.62 minutes**	

Table 2: Details of the RealSound dataset.

9.3 Data Cleaning

We could trust that the sound files are correctly labelled and are of decent quality however we have no guarantee that that would be the case. For this reason, we went through each sound file one by one to check that the sound files are okay to use and that the label indicates the correct sound category. After a thorough inspection we came to the conclusion that the sound files contain audio data of adequate quality and the labelling was indeed correct.

9.4 Data Preprocessing

Data pre-processing encapsulates everything that is done on the sound files before machine learning takees place. This includes removing the silent chunks of the sound files, extending them beyond 10 seconds long and applying some data augmentation techniques.

9.4.1 Silence Removal

As mentioned before, we noticed that some sound files included extended silent sections. As a result this would fill the training set with data that is essentially noise, meaning that extended silent parts do not give us any useful information. That being said, shorter pauses or silent sections that are part of the sound signal can be very useful in detecting patterns. For this reason it is critical that we choose an appropriate threshold for what we deem to be silence. Since the VGGish inputs are log-mel spectrograms that represent 1 second of sound (960ms to be exact), it makes sense to set that time-threshold to exactly 1 second. Another threshold we had to determine was the loudness threshold. For that we used the decibels relative to full scale (dBFS) units, used for measuring amplitude levels in digital signals. We explored the use of several values but we ended up using the default threshold of the `pydub` module[27], which was -16 dBFS.

9.4.2 Sound File Extension

As seen in the table above, the amount of sound data for each category varies a lot. The 14 files for Door Slam amount to 28 seconds worth of sound whereas the 14 files for Coffee Machine amount to 466 seconds. In order to overcome this imbalance, we decided to take sound files that are shorter than 10 seconds and loop them until they reach or exceed the 10 second mark. For example if a sound file is 4 seconds long, it will be looped three times, making it 12 seconds long. We could've set a strict limit at 10 seconds, but that could possibly 'cut' the sound event, something which could obscure sound patterns. This is also the reason why we opted not to shorten the longer sound files.

9.4.3 Data Augmentation

At this point, we are ready to perform data augmentation on the sound files themselves, using a methodology similar to the one followed in Haubrick P., & Ye J. (2019). We carry out this process for several reasons. First, since we're producing more sound files out of the existing ones, we grow our dataset which will enable our machine learning to learn the sound event patterns better. Furthermore by manipulating the sound signals, it will hopefully allow our model to generalise better. The idea is that the model will become more sensitive to the actual sound patterns and will be able to classify events occurring in a real life scenario, despite background noise or a different pitch and/or volume. . The following are the possible perturbations applied on the original dataset:

- Increase volume by 5dB and 10dB.

- Decrease volume by 5dB and 10dB.
- Increase pitch by 3 or 6 semitones.
- Decrease pitch by 3 or 6 semitones.

9.5 Feature Extraction

After data augmentation is complete, we are ready to convert the audio files into a format that can be used for machine learning. This process is outlined below:

1. The audio is resampled to 16kHz single channel sound signal (mono).
2. A spectrogram is computed by decomposing the frames with a Short-Time Fourier Transform with a windows size of 25ms, a window hop of 10ms and a periodic Hann window.
3. The spectrogram that results from the above step, is split into 64 mel-spaced frequency bins, covering the range of 125-7500Hz.
4. The mel spectrogram is stabilised by taking the log and adding a small offset (log(mel-spectrum + 0.01)), where the small offset is used to avoid taking the logarithm of 0, an undefined operation.
5. These features are framed into examples of 0.96 seconds (non-overlapping), with each example covering 64 mel bands and 96 frames of 10ms each.

After the mel-spectrogram for each sound file is extracted, they are saved with the appropriate label indicating which raw audio file it represents which also indicates which sound category is belongs to. The dataset is now in a format that can be used by the program, which takes us to the machine learning portion of the pipeline.

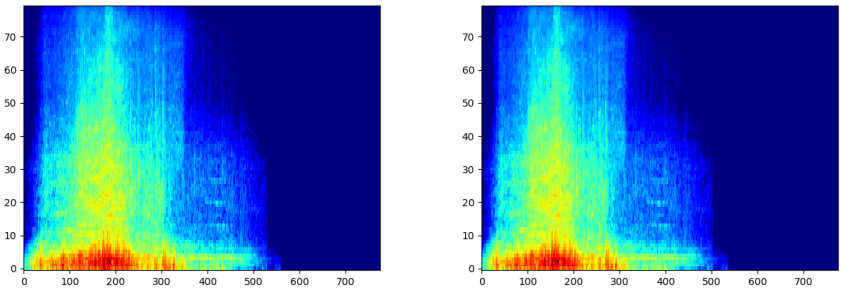
9.5.1 SpecAugment

We've already mentioned that this augmentation technique is applied on the extracted mel-spectrograms online during training. The techniques outlined in the paper that introduced them[12], are time warping, frequency masking and time masking.

- Time warping: This function warps the spectrogram along the time axis to the left or to the right by a certain distance τ.

- Frequency masking: This function masks horizontal strips of the spectrogram across a number of frequency channels f.
- Time masking: This function masks vertical strips of the spectrogram across a number of time steps t.

The following diagrams demonstrate the effects of each of the SpecAugment techniques:



(a) Base Spectrogram

(b) Time-warped Spectrogram

(c) Frequency & Time Masking

Figure 1: SpecAugment transformations

This technique was developed primarily to be used for speech recognition tasks, a problem domain in which it has achieved very promising results, outperforming all prior work on the LibriSpeech 960h and Switchboard 300h datasets. Another use of SpecAugment that showed promising results is described in the technical report 'Semi-Supervised Networks With Heavy Data Augmentations To Battle Against Label Noise In Audio Tagging Task'[18]. In this case, SpecAugment is used to regularise a CNN that is used as part of a system

that automatically tags audio, reducing the amount of manual effort required to build a large scale, labelled dataset. The audio clips used in this challenge were taken from FreeSound and cover a large number of topics, from sounds that one would typically find in a house such as microwave oven and toilet, to animal sounds such as dog and cat sounds.

SpecAugment is a simple yet effective augmentation technique that is computationally cheap to apply and we have seen that research shows that it yields promising results not only when applied on speech recognition datasets but also datasets that consist of sound events which bear many similarities to the dataset used in this project.

9.6 Machine Learning

This part varies depending on the model we're using. However, at the center of our approach lies the VGGish model, so it would be worthwhile to talk about it in a separate section.

9.6.1 VGGish

The VGGish model is a variant of the VGG model [14] used for large-scale image recognition (Configuration A with 11 weight layers). It was trained on a large annotated YouTube dataset, which later became known as YouTube-8M, and it was derived by applying the following changes to the VGG:

- The input was changed to 96x64 to for the log mel spectrogram audio inputs.
- The last group of convolutional and maxpool layers were dropped, leaving four groups of convolution/maxpool layers instead of five.
- A 128-wide fully connected layer is used in the end rather than a 1000-wide fully connected layer. This last layer acts as a compact embedding layer.

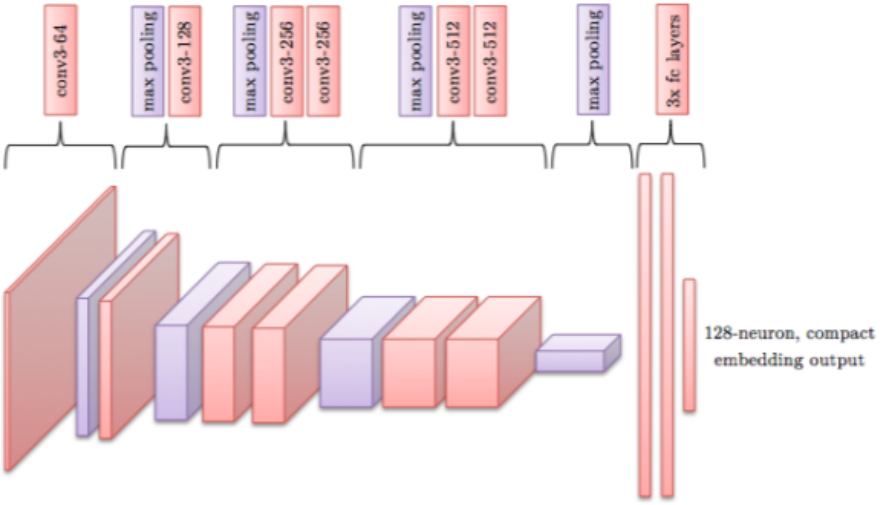


Figure 2: VGGish architecture

The VGGish model can be used as the following:

- As a feature extractor: Audio input is converted into a 128-D embedding which can then be used as input to a different classification model.
- As part of a larger model: In this case, we can add layers on top of the VGGish 128-D embedding and fine tune VGGish specifically to our task. This is useful if our dataset isn't drastically different compared to the original dataset.

In this project we have used the VGGish both as part of a larger model, fine tuning it so that we make used of the fact such a CNN that has been pre-trained on a massive dataset, and as a feature extractor, features which are then used to train a RNN.

Our decision to use VGGish was motivated by the amount of research that support the effectiveness of this particular network used as part of a sound classification system. In 'Simple CNN and vggish model for high-level sound categorization within the Making Sense of Sounds challenge'[16], the performance of the *SimpleMind* CNN is compared to that of VGGish with six fully connected layers added to it (named *NeverMind*) at the task of classifying audio files into five different classes, namely *Nature*, *Human*, *Music*,*Effects* and *Urban*. The system using the VGGish achieved superior results, scoring an 82% accuracy against 50% reached by *SimpleMind*. VGGish has also previously been used successfully as a feature extractor part of a larger system. 'Zero-Shot Audio

Classification Based On Class Label'[19], proposes a solution for zero-shot audio classification, in which VGGish is used to extract audio feature embeddings from recordings and Word2Vec[20] is used to generate class label embeddings. These two sets of embeddings are later used to train a bilinear model enabling it to measure the compatibility between an audio feature embedding and a class label embedding. Considering that a zero-shot approach is followed, the system achieves a remarkable performance on the ESC-50 dataset, scoring an average accuracy score of 26%.

9.6.2 Baseline Model

Our baseline model consists of the VGGish model, with the addition of a layer of 15 neurons (one for each sound class) using the softmax function at the end. What the softmax function does is it extends the idea of logistic regression to a multi-class problem. It essentially calculates the probability of the sound category an input belongs in and our prediction is that sound category with the highest probability. We used this simple, baseline model so that we can demonstrate if and how much better results our later improvements yield.

9.6.3 Fine Tuning Fully Connected Layers

This model is similar to the baseline model with the difference that we fine-tune the last 3 fully-connected layers. Fine tuning, summarised very well by this article[28],is essentially the process of tweaking the parameters of a pre-trained network so that it is better at solving our task. The idea is that initial layers are trained well enough for detecting higher, general features whereas the last few layers are learning the finer details. For this reason, we freeze the initial layers and only retrain the last few.

9.6.4 Fine Tuning Last Convolutional Layer

The idea here is pretty much the same as the above, however we freeze up fewer layers and we also tweak the last convolutional layer. Specifically, when we refer to the last convolutional layer, we refer to the last block of convolutional layers, namely the two conv3-512 layers seen in Figure 2. Note that we are also tweaking the last 3 fully connected layers in addition to the last convolutional layer.

9.6.5 RNN

As mentioned earlier a RNN is used because of the temporal component of the sound data. We used the VGGish network to extract the sequence of 128-D embeddings from each sound file, and then we use that as an input to the RNN. In this project we used a network with two Long Short-Term Memory layers (LSTM) as shown in the following diagram:

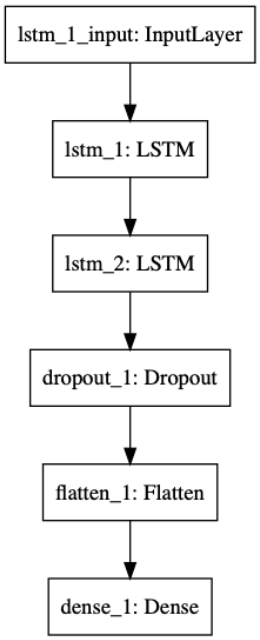


Figure 3: RNN architecture

A specific problem we had to address was the problem caused by the fact that different sound files have different lengths. For this reason, the length of the sequence of features extracted for each sound file varies widely and thus the timesteps component of the network's input shape cannot be predefined. In order to solve the problem of variable length input sequences, we padded the sequences with a dummy variable to a new longer desired length, a length equal to the longest sequence in our dataset[26].

9.7 Model Evaluation

The main evaluation point of each model was how well it generalises to the manually recorded dataset. For this reason we trained each model on the

FreeSound dataset and we then we used the manually recorded (RealSound) dataset for evaluation purposes. This allows us to compare each training approach followed, from the augmentation techniques to the network architecture itself. However we are also interested to see whether using a sub-set of the Freesound dataset that is set aside before training as a test set could provide accurate evaluation metrics in the absence of a manually recorded dataset which may not always be feasible to put together.

10 Implementation

In this section of our report we will be discussing the specifics of our project, from libraries used down to the details of how our system was implemented.

10.1 Data Organisation And Naming

Sound files belonging in the same sound category were kept in a separate directory and they were named using the following convention: `<sound_category>_<index>.wav`, where the index is a zero-based counter identifying a file in a specific category. For example the fifth file the 'ClearThroat' category is named `ClearThroat_4.wav`. Apart from the directory containing the sound files we also have directories for each of the subsequent steps making up the preprocessing procedure. This saves a lot of time as in the case where a change is made, it may not be necessary to carry out the entire preprocessing as we can simply start from where the changes have been made. Once the preprocessing is complete the resulting spectrograms are kept in a directory with a structure similar to that of the directory containing the sound files, with the difference that they're saved in the `.npy` format which is a binary files format[29]. By keeping these saved on disk, the program need not preprocess the raw sound files every time it is run, making execution quicker.

10.2 Preprocessing Implementation

The modules which implement the preprocessing procedure can be found in the `Preprocessing` directory of the project. Any files referred to in this section can be found there.

10.2.1 Silence Removal

The functionality is implemented in the `remove_silence.py` file. The program gets the directory where the original files can be found as a parameter and the resulting files are placed in a different directory which holds the files that have gone through this preprocessing step. The core functionality is achieved by the `split_on_silence` method of the `pydyb` library[27][30]. It is in this method where we define what should be considered as 'silence' in terms of segment length and amplitude.

10.2.2 Sound File Extension

This step is implemented in the `loop_sound.py` file. Once again, the program gets the directory where the audio files reside as a parameter and once it is complete, the resulting audio data is placed in a different one. The sound data is loaded using the `pydub` library and the program checks whether it is longer than a certain duration (default is 10s). If not, the sound data is appended to itself (looping) until it is indeed longer than the threshold.

10.2.3 Data Augmentation

Since the augmentation techniques applied on the raw sound files are almost the same as those used by Haubrick P. & Ye J. (2019), we used the same module[31] with a few adaptations so that it can be integrated into our pipeline and so that the resulting file names follow our own naming convention. As mentioned earlier the naming convention is `<sound_category>_<index>.wav`. However to cater for the augmented files, we've added another component to this turning it into `<sound_category>_<index>-<augmentation_type>.wav`. The augmentation type is used to indicate what data augmentation technique was applied. The following describes each of the possible labels:

- Increase volume: 'VCU'
- Decrease volume: 'VCD'
- Increase pitch: 'PCU'
- Decrease pitch: 'PCD'

For example the files `ClearThroat_4-VCU.wav` and `ClearThroat_4-PCD.wav` are the files resulting from applying the volume increase and the pitch decrease, respectively, on the `ClearThroat_4.wav` sound file.

10.3 Feature Extraction

The feature extraction process involves the conversion of audio waveform into an array of examples for the VGGish. This is done in the `vggish_input.py` module that can be found here[32]. Essentially, the audio waveform is taken and it is converted into a log-mel spectrogram which is represented by a 3-dimensional `numpy.array`[33] of shape `[num_examples, num_frames, num_bands]`. This array represents a sequence of 'examples' and each example represents a patch of the spectrogram which covers `num_frames` frames of audio, `num_bands`

mel frequency bands, and each frame covers a time window that's parameterised by the `STFT_HOP_LENGTH_SECONDS` variable in the `vggish_params.py` module[34].

For the purposes of our project, we've set the `STFT_HOP_LENGTH_SECONDS` parameter to be 10ms, and each example consists of 96 frames covering 64 mel frequency bands. Thus, each example would represent 960ms of audio. Hence, for an audio that is 5 seconds long, the feature extraction would produce a 3-dimensional numpy array of shape `[5, 96, 64]`.

10.3.1 Spec Augment Implementation

This augmentation technique is carried out on the fly just before training and it is applied directly on the features extracted, as described above. The paper that introduced this technique[12] does not provide an implementation so we were faced with the choice of building one from scratch or find an existing one. We ended up finding an implementation found here[35] that we adapted to fit our purpose. Specifically we had to restructure the output of so that it returns the augmented spectrogram in the appropriate shape of `[num_examples , num_frames, num_bands]`.

10.4 Machine Learning

In this section we will be discussing the various implementation details of the machine learning portion of this project. The files referred to in this section may be found in the `Classify` directory.

10.4.1 Settings

The file `classify_settings.py` contains a number of settings that can be adjusted and parameterised. For example, the path to the directory which contains the spectrograms to be used as a training set and a testing set are set in this file. This becomes important as this allows us to quickly train our models using different datasets without actually changing any of the core infrastructure.

10.4.2 VGGish Implementation

The most widely used implementation of VGGish and the one used by Haubrick P. and Ye J. (2019), is the one provided by tensorflow, found here[37].

Initially, we used this for this project as well, however, we run into some difficulties due to the fact that this implementation only supports TensorFlow v1.x while the GPU-enabled lab machines that we were using, only support GPU computing with TensorFlow v2.0 and v2.1.

10.4.3 Libraries

For the machine learning portion of this project, we decided to use Keras[38], a high level neural-network API using TensorFlow v2.1 as a backend. We chose Keras due to the fact that it's very user friendly, as it was built to allow fast prototyping and experimentation, and modular allowing us to easily design and build machine learning models to fit our purpose. For this reason we looked for and found a Keras implementation of VGGish (with the necessary pre-trained weights) here[39] something which allowed us to build new models on top of it and seamlessly integrate it in our pipeline.

10.4.4 Splitting The Dataset

The first step is to determine which spectrograms will be used for training and which will be used for testing. If the FreeSound dataset is used both for training and testing, the spectrograms are split in the two sets, a training set and a testing set, using the `sklearn.model_selection.train_test_split` method[36] with the latter being 20% of the entire dataset. If we are to use the FreeSound dataset for training purposes only and RealSound dataset for testing, we can simply load the spectrograms in their respective set without any splitting involved.

Both the training and the testing set, are made up of two components namely the 'X' component which is a vector of the actual data, and a 'y' component which is a vector that holds the class labels for each corresponding element of 'X'. The class labels are integers representing each sound class, as assigned in the `classify_settings.py` file.

Note that the process of organising the training and test sets is done at the sound file level and not at the example level. That is so that the different examples generated from the same sound file will not end up in both sets, something which would constitute data leakage.

10.4.5 Class Vector Encoding

Once the data has been loaded, we need to encode the category labels indicating which sound category a spectrogram belongs in (the 'y' component), in a

form that can be used by the machine learning model. For this purpose we used the `keras.utils.to_categorical` method which converts a class vector into a binary class matrix. This is also known as 'one hot encoding' an encoding in which a class variable is represented by a matrix made up of zeros except the index of the integer it represents, marked by a 1. For example the sound category 'Coughing' is represented by the integer 2. It's one hot encoding, given that we are using 15 different sound classes will be:

$$001000000000000 \tag{1}$$

With the digit indexed by 2 being 1, and the rest of the digits being 0.

10.4.6 Preparing The Inputs

The last data preparation step before machine learning can take place is to conform to the input format required by the VGGish, which is `numpy.array` of shape `[96, 64]`. We have previously mentioned that a sound file with a duration of 5 seconds would produce a spectrogram with shape `[5, 96, 64]`. As a result such a spectrogram will produce 5 samples. For example let's say a dummy dataset consists of 3 sound files, A, B and C:

- A: 3 seconds long.
- B: 4 seconds long.
- C: 6 seconds long.

Now, let's say A and C make up the training set and that B makes up the testing set. This means that the training set will consist of 9 samples and the test set will consist of 4 samples, with each sample being a `numpy.array` of shape `[96, 64]`.

Note: In the case when SpecAugment is used, the augmentation technique will be applied on the examples that make up the training set.

While this would be everything that's needed to by done when only VGGish is used, there is an additional step that needs to be made before we can use our inputs for the RNN. In this case, the sequences of 128-D embeddings for each sound file are obtained and, taking the example of the example dataset above, this would be a sequence of shape `[3, 128]` for sound file A, `[4, 128]` for B and `[6, 128]` for C. As previously mentioned however, the fact that these sequences are of different length mean that we can't predefine the input length of our network. We solve this problem by using the Keras `pad_sequences` method[41], which pads our sequences to a longer fixed length. The fixed length

is equal to the longest sequence in our dataset which, in our toy example, is the sequence extracted for sound file C. Thus, all of our sequences would be padded to have a shape of `[6, 128]`, and our input to our LSTM would be of shape `[batch_size, 6, 128]`.

10.4.7 Compiling The Models

After the data has been prepared for training, the machine learning model is defined and configured for training. When only the CNN is being used, we load it and we freeze all the layers apart from the ones we would like to fine tune to fit our task. Furthermore, an additional 15 neuron-wide softmax layer is appended to the end of the CNN which outputs the predicted sound category the input belongs to. The model is then compiled using the following arguments:

- `loss='categorical_crossentropy'`
- `optimizer='rmsprop'`
- `metrics=['accuracy']`

The loss argument is used to indicate which loss function is to be used during training. This is simply the function that is used to evaluate a neural network's weight configuration and the goal of the optimisation algorithm is to minimise it[40]. And since our task is multi-class classification, we use the categorical cross-entropy loss function. The optimizer argument refers to the said optimisation algorithm and in our case we used the RMSProp algorithm, which is an adaptive learning rate method proposed by Geoff Hinton[42]. Finally, the metrics argument defines the metrics to be evaluated by the model during training and testing. By using accuracy as this metric we are essentially saying that we would like to train the model to achieve the highest degree of accuracy possible.

In the cases we use a RNN the workflow looks a little bit different. Instead of simply appending a 15 neuron-wide softmax layer at the end of VGGish allowing it to perform classification, we use it as a feature extractor. The VGGish will produce a 128-D embedding for each input sample and sequences of these embeddings (each sequence representing a spectrogram generated from an audio file) are used as inputs to our RNN which is defined as in Figure 3 and compiled using the following arguments:

- `loss='categorical_crossentropy'`
- `optimizer='rmsprop'`
- `metrics=['accuracy']`

10.4.8 Model Evaluation

After training is complete, we use the testing set to evaluate our model's performance. The model predicts the sound category for each of the samples present in the testing set and then this can be used with our ground truth set, that is the set containing the correct classification for every test sample, to calculate various evaluation metrics. In order to produce comprehensive reports for a model's performance, we use the `sklearn.metrics.confusion_matrix`[43] and the `sklearn.metrics.classification_report`[44] methods. The former can be described as matrix C is such that $C_{i,j}$ that is equal to the number of observations known to be in group i and predicted to be in group j, and the latter is a text report containing the main classification metrics such as accuracy, recall and f-1 score.

11 Evaluation

11.1 Methodology

In this project we are interested in exploring how different machine learning techniques can affect a model's classification performance including:

- Different model architectures such as using a CNN and an RNN
- Various data augmentation techniques such as data augmentation techniques on the raw sounds files and using SpecAugment[12].

We are also comparing how well our models generalise on sounds that have been manually recorded compared to sounds that have been collected from FreeSound. To this end, we trained several models to cover all the possible configuration combinations to give a comprehensive picture of how each component affects the evaluation metrics.

11.2 Experiment Set Up

In order to understand what experiments have to be carried out we need to outline the different options we have when choosing what model architecture to use and which machine learning techniques to use.

11.2.1 Model Architecture

Perhaps the most obvious choice is that of which model architecture is to be used. The two approaches we are comparing are as follows:

- Use the VGGish CNN with a softmax layer in the end (referred to as **VGGish+** from here on in this section).
- Use the VGGish simply a feature extractor, with the features being used as input to an LSTM network for classification (referred to as simply **LSTM** from here on in this section).

The VGGish+ approach however, also offers the choice of fine-tuning. In order to explore how fine-tuning may affect our model's performance we have also formulated experiments with the following variables:

- VGGish+ with no fine-tuning.

- VGGish+ with fine-tuning the fully connected layers (See Figure 3).
- VGGish+ with fine-tuning the fully connected layers and the last convolutional layer. (conv3-512 layers).

11.2.2 Augmentation Techniques

As we've highlighted in the earlier sections there are two main augmentation techniques which we have explored namely applying augmentations manipulating aspects of the sound files used for training our models and SpecAugment which is applied on the generated spectrograms themselves. This allows us to compare trained models which have been trained using:

- No data augmentation
- Augmentations applied on the raw sound files which are later converted into spectrograms and used for training (referred to as **DA** from here on in this section).
- SpecAugment applied on the spectrograms used for training (referred to as **SA** from here on in this section).
- Both of the above techniques used together (referred to as **DA & SA** from here on).

11.2.3 Evaluation Set

In order to determine whether there is a significant change in the model's classification performance when using a sub-set of the data taken from FreeSound as our evaluation set and using the RealSound dataset as our evaluation set, we set up experiments for both cases.

11.3 Results

11.3.1 VGGish+ & No Fine-Tuning

The following are the results when FreeSound data is used for the training set and the testing set:

Augmentation	Training Accuracy	Validation Accuracy	Testing Accuracy
N/A	83.80%	77.51%	62.38%
DA	84.73%	83.48%	63.93%
SA	76.84%	75.64%	52.33%
DA & SA	74.95%	73.60%	49.36%

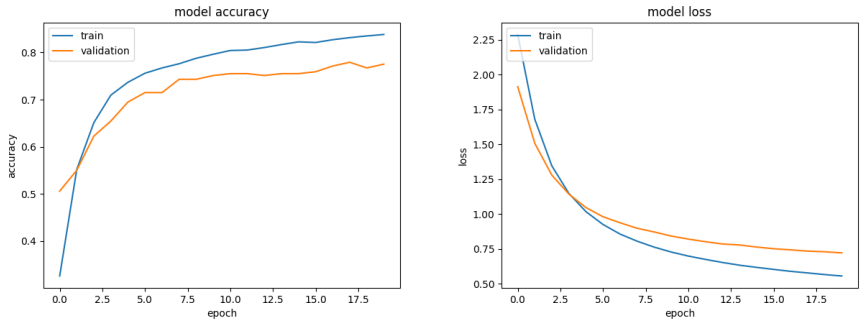
Table 3: Results for VGGish+ with no fine-tuning, evaluated on FreeSound.

The following are the results when the FreeSound dataset is used for training and the RealSound dataset is used for testing:

Augmentation	Training Accuracy	Validation Accuracy	Testing Accuracy
N/A	82.64%	82.81%	53.71%
DA	83.20%	81.91%	56.18%
SA	74.24%	72.19%	50.56%
DA & SA	73.05%	71.95%	50.11%

Table 4: Results for VGGish+ with no fine-tuning, evaluated on RealSound.

In both cases, we can see that there is a slight increase in accuracy when the DA augmentation technique is used. This may be attributed to the fact that the training dataset in this case is larger and to the fact that the augmentation causes perturbations to the original sound file, so the model becomes better at generalising to unseen sounds that are slightly different to the ones used during training. Furthermore it is evident that when SA is used the testing accuracy is actually worse than when no data augmentation is applied at all. Lastly, the testing accuracy when the RealSound dataset is used is lower than when the test set consists of a FreeSound subset, as expected. An examination of the training accurarcy and loss graphs these experiments produce revels that they follow a similar curve. The following are the graphs produced when the model is trained with data on which no data augmentation technique has been applied:



(a) Training Accuracy (b) Training Loss

Figure 4: Training accuracy and loss for model without augmented data and with no fine-tuning.

11.3.2 VGGish+ & FC Layers Fine-Tuning

The following are the results when FreeSound data is used for the training set and the testing set:

Augmentation	Training Accuracy	Validation Accuracy	Testing Accuracy
N/A	100%	93.17%	77.65%
DA	100%	99.45%	78.78%
SA	100%	99.33%	79.49%
DA & SA	100%	98.95%	81.05%

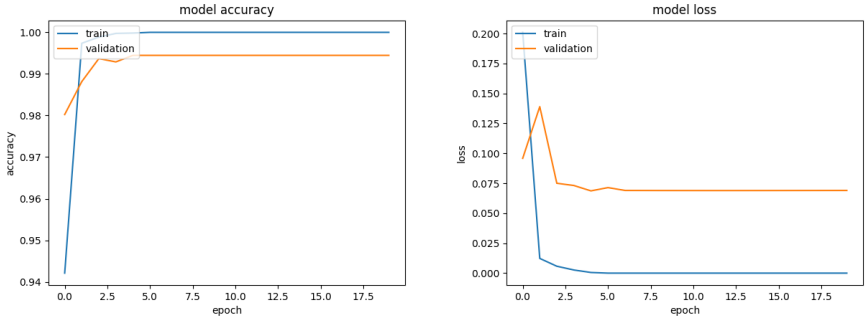
Table 5: Results for VGGish+ with fine-tuning the fully connected layers, evaluated on FreeSound.

The following are the results when the FreeSound dataset is used for training and the RealSound dataset is used for testing:

Augmentation	Training Accuracy	Validation Accuracy	Testing Accuracy
N/A	100%	97.19%	70.79%
DA	100%	99.20%	71.01%
SA	100%	99.06%	69.89%
DA & SA	100%	98.91%	70.11%

Table 6: Results for VGGish+ with fine-tuning the fully connected layers, evaluated on RealSound.

The first observation we can make is that thanks to the fine-tuning, our model is able to learn much better, reaching perfect training accuracy and a near perfect validation accuracy in every single experiment. Furthermore, SpecAugment does not cause the testing accuracy to significantly decrease, as seen in Tables 3 and 4. In fact, Table 5 shows a bump in testing accuracy when SA and DA & SA are used. Again, the testing accuracy achieved with RealSound is lower than when FreeSound is used as a testing set. By inspecting the training accuracy and loss graphs generated in the above experiment, they all follow a similar trend. As an example we will use the graphs generated when DA was used for training:



(a) Training Accuracy

(b) Training Loss

Figure 5: Training accuracy and loss for model trained with DA, fine tuning the FC layers

11.3.3 VGGish+ & CNN Layer Fine-Tuning

The following are the results when FreeSound data is used for the training set and the testing set:

Augmentation	Training Accuracy	Validation Accuracy	Testing Accuracy
N/A	100%	94.38%	79.21%
DA	71.83%	40.63%	38.61%
SA	99.67%	93.71%	68.03%
DA & SA	30.74%	32.04%	33.24%

Table 7: Results for VGGish+ with fine-tuning the last CNN layer, evaluated on FreeSound.

The following are the results when the FreeSound dataset is used for training and the RealSound dataset is used for testing:

Augmentation	Training Accuracy	Validation Accuracy	Testing Accuracy
N/A	100%	97.19%	71.69%
DA	34.87%	44.31%	49.44%
SA	32.22%	24.38%	33.93%
DA & SA	26.21%	25.73%	36.40%

Table 8: Results for VGGish+ with fine-tuning the last CNN layer, evaluated on RealSound.

The results here are fairly inconsistent and quite poor with the majority of experiments resulting in a testing accuracy of less than 50%. It is evident that fine tuning the last CNN layer has caused a significant performance degradation.

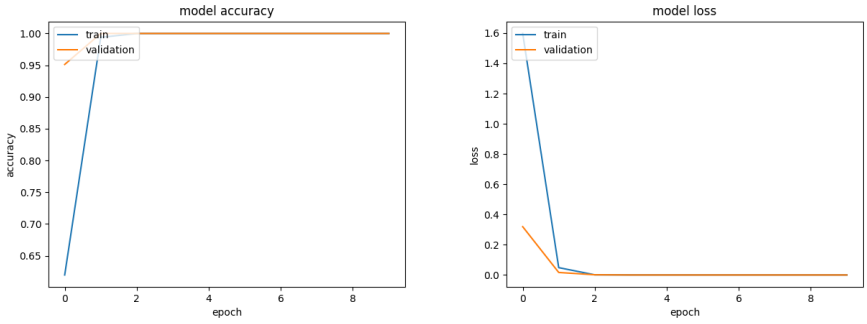
11.3.4 LSTM

In this section we will be looking at the results when VGGish is used as a feature extractor, features with which an LSTM network is trained and then used as a classifier. We focused on using the model's performance when evaluated on the RealSound dataset as this is a better basis of comparison as to how well the different models generalise to more realistic sounds.

Augmentation	Training Accuracy	Validation Accuracy	Testing Accuracy
N/A	100%	100%	100%
DA	100%	100%	58.93%
SA	100%	100%	76.79%
DA & SA	100%	100%	35.71%

Table 9: Results for LSTM network, evaluated on RealSound.

The first remarkable observation is that this network is very capable of learning the different sequences, achieving a perfect training and validation accuracy in all of the experiments. Even more remarkable is the fact that in the first experiment, the network achieves a perfect accuracy score on unseen data. Obviously this is a result which raised some suspicion. Specifically, we believed that there was some sort of data leakage involved. However, logs reveal that the model is indeed trained using the FreeSound dataset which consists of 204 sound files and it is tested on the previously unseen by the network RealSound dataset which consists of 56 sound files. The following experiments, however did not yield a similar result. The next best score resulted when the model was trained using SA, at 77%, a significant drop from 100% followed by a score of 59% and 36% when DA and DA & SA were used respectively. Despite the difference in testing accuracy, all the models achieved a perfect training and validation accuracy. The following are two of the graphs produced when the LSTM is trained:



(a) Training Accuracy (b) Training Loss

Figure 6: Training accuracy and loss for LSTM model

11.4 Results Analysis & Evaluation

Here, we will summarise the results of each model when evaluated on the RealSound dataset, in order to allow us to make a direct comparison. We will then discuss these results and the reasons as to why they differ.

Note: All of the results as well as accuracy and loss graphs, confusion matrices for each experiment, classification reports and program output, can be found in the `Classify/Results` directory.

11.4.1 Results Analysis

Model	Fine-Tuning	Augmentation	Testing Accuracy
VGGish+	N/A	N/A	53.71%
VGGish+	N/A	DA	56.18%
VGGish+	N/A	SA	50.56%
VGGish+	N/A	DA & SA	50.11%
VGGish+	FC	N/A	70.79%
VGGish+	FC	DA	71.01%
VGGish+	FC	SA	69.89%
VGGish+	FC	DA & SA	70.11%
VGGish+	FC & CNN	N/A	71.69%
VGGish+	FC & CNN	DA	49.44%
VGGish+	FC & CNN	SA	33.93%
VGGish+	FC & CNN	DA & SA	36.40%
LSTM	N/A	N/A	100%
LSTM	N/A	DA	58.93%
LSTM	N/A	SA	76.79%
LSTM	N/A	DA & SA	35.71%

Table 10: Summary of the performance of all the trained models, evaluated on RealSound.

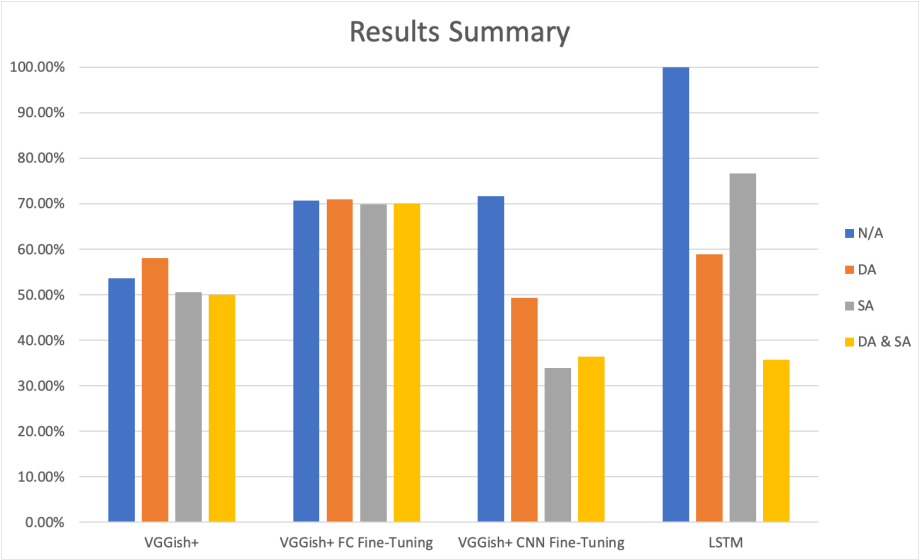


Figure 7: Graphical presentation of the results reported

From the table above we can make the following observations:

1. The model with the best performance when no data augmentation technique was applied was the LSTM, with an accuracy of 100%.
2. Using DA has improved the performance for VGGish+ without fine-tuning and for VGGish+ with fine-tuning the fully-connected layers. The improvement was quite small and it was 3.01% and 0.22% respectively.
3. Using DA has worsened the performance for VGGish+ with fine-tuning the fully-connected and the final CNN layers, and for the LSTM. The degradation of performance was -22.25% and -41.07%respectively.
4. Using SA has decreased the evaluation accuracy for all of the models using VGGish+. The decrease was -3.15%, -0.9% and -37.76% for no fine-tuning, fine-tuning the fully connected layers fine tuning the fully-connected and final CNN layers, respectively. Using SA has also decreased the performance of the LSTM by -23.21%.
5. Using DA & SA has also decreased the performance of all of the models. The decrease was -3.6%, -0.68%, -35.29 and -64.29% for each respective model.
6. The best result out of all of our models was 100% and it was achieved by the LSTM when trained on a non-augmented dataset.

7. When any kind of data augmentation was applied on the data used to train VGGish+ with fine-tuning the fully-connected and final CNN layers, the performance deteriorated quite severely.

8. When any kind of data augmentation was applied on the data used to train the LSTM, the performance worsened. However, excluding the 100% accuracy entry from our result set, the LSTM trained with data augmented using SA was the best-performing model, with an accuracy score of 76.79%.

9. When the fully-connected layers of VGGish+ were fine-tuned, its performance increased significantly when compared to its non fine-tuned counterpart.

10. When fine-tuning was extended to include the final CNN layer as well, the performance only increased when the model was trained on the non-augmented dataset. When trained with the other datasets, the performance degraded when compared to VGGish+ with no fine-tuning.

11. The model with the best performance when DA augmentation was applied was the VGGish+ with fine-tuning the fully-connected layers, with an accuracy score of 71.01%.

12. The model with the best performance when SA augmentation was applied was the LSTM, with an accuracy score of 76.79%.

13. The model with the best performance when DA & SA augmentations were applied was the VGGish+ with fine-tuning the fully-connected layers, with an accuracy score of 70.11%.

In following sections we will be attempting to explain the most important observations we have noted above.

11.4.2 Results Evaluation : Observation 1

Because of the seemingly excellent accuracy score of this model, we looked into the experiment to see whether there were any mistakes in our methodology and in our code and we found that the experiment was carried out as intended. To convince the reader of its legitimacy, we have included screenshots of the logs and the classification report, which can also be found in the submission in the file `Classify/Results/Baseline/lstm_baseline_rs.txt`.

```
Train on 163 samples, validate on 41 samples
Epoch 1/10
 - 21s - loss: 1.5978 - accuracy: 0.6196 - val_loss: 0.3195 - val_accuracy: 0.9512
Epoch 2/10
 - 21s - loss: 0.0486 - accuracy: 0.9939 - val_loss: 0.0167 - val_accuracy: 1.0000
Epoch 3/10
 - 21s - loss: 0.0010 - accuracy: 1.0000 - val_loss: 0.0013 - val_accuracy: 1.0000
Epoch 4/10
 - 21s - loss: 1.0387e-04 - accuracy: 1.0000 - val_loss: 2.3149e-05 - val_accuracy: 1.0000
Epoch 5/10
 - 21s - loss: 4.5435e-05 - accuracy: 1.0000 - val_loss: 2.5237e-06 - val_accuracy: 1.0000
Epoch 6/10
 - 21s - loss: 1.0794e-06 - accuracy: 1.0000 - val_loss: 5.3789e-07 - val_accuracy: 1.0000
Epoch 7/10
 - 21s - loss: 4.6659e-07 - accuracy: 1.0000 - val_loss: 1.4538e-07 - val_accuracy: 1.0000
Epoch 8/10
 - 21s - loss: 1.1994e-07 - accuracy: 1.0000 - val_loss: 5.2336e-08 - val_accuracy: 1.0000
Epoch 9/10
 - 21s - loss: 1.0312e-07 - accuracy: 1.0000 - val_loss: 2.6168e-08 - val_accuracy: 1.0000
Epoch 10/10
 - 21s - loss: 1.1409e-07 - accuracy: 1.0000 - val_loss: 1.1630e-08 - val_accuracy: 1.0000
```

(a) Training Log

	precision	recall	f1-score	support
Chair	1.00	1.00	1.00	4
ClearThroat	1.00	1.00	1.00	4
CoffeeMachine	1.00	1.00	1.00	3
Coughing	1.00	1.00	1.00	4
DoorKnock	1.00	1.00	1.00	3
DoorSlam	1.00	1.00	1.00	4
Drawer	1.00	1.00	1.00	4
FallingObject	1.00	1.00	1.00	4
FootSteps	1.00	1.00	1.00	4
Keyboard	1.00	1.00	1.00	4
Laughing	1.00	1.00	1.00	4
MilkSteamer	1.00	1.00	1.00	3
Sink	1.00	1.00	1.00	4
Sneezing	1.00	1.00	1.00	4
Stiring	1.00	1.00	1.00	3
accuracy			1.00	56
macro avg	1.00	1.00	1.00	56
weighted avg	1.00	1.00	1.00	56

(b) Classification Report

Figure 8: Training log and classification report for LSTM model trained with non-augmented data.

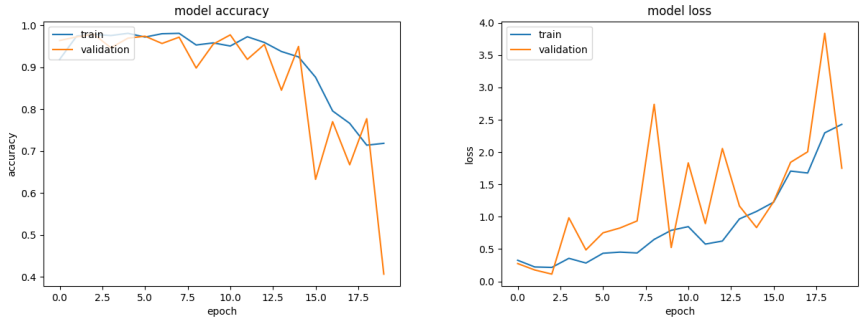
From the above it is evident that there was no data leakage involved in this experiment, as the training set consisted of 204 samples, equal to the number of sound files in FreeSound and the test set consisted of 56 samples, equal to the number of sound files in RealSound (see Tables 1 and 2). We continue to look as to why the result is what it is and we will expand on what we can do to solidify the claim when we discuss future work.

11.4.3 Results Evaluation : Observation 2

Using a data augmentation such as DA has two main effects. The first one is that it multiplies the amount of training data available and for that reason our model is more likely to learn more effectively. Furthermore, by introducing perturbations such as pitch and volume differences and the introduction of white noise, the models will generalise better to sound events that occur in more realistic settings. These effects can be used to explain the observation of an improvement in classification accuracy.

11.4.4 Results Evaluation : Observation 3

While we would expect that using DA would lead to an improvement in classification accuracy, that was not the case here. In the case for VGGish+ with fine-tuning the fully-connected and final CNN layers, we suspect that the reason behind this degradation in performance is the problem of overfitting. To validate our claims, we inspected the graphs plotting the accuracy and loss during training:



(a) Training Accuracy

(b) Training Loss

Figure 9: Training accuracy and loss for model trained with DA, fine-tuning the final CNN layer & the FC layers.

11.4.5 Results Evaluation : Observation 4

Using SA has only caused the performance of our models to deteriorate and there are several reasons as to why this is the case. The first reason may be the fact that the SpecAugment parameters that we used when applying this technique, namely τ, f and t, were not optimal for our task. Furthermore, as seen in the paper[12], SpecAugment was developed for speech recognition and

not for the task of sound event classification and for that reason we can suggest that this technique may not be the most effective data augmentation technique for the problem we attempted to tackle in this project.

11.4.6 Results Evaluation : Observation 5

Since we have found that SA has adversely affected the performance of our models, it is no surprise that when it is used in conjunction with DA it leads to the same result.

11.4.7 Results Evaluation : Observation 6 & 8

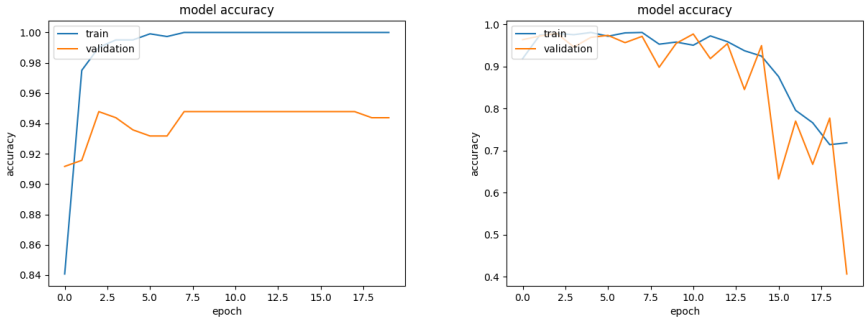
As seen in Figure 8, when the LSTM is trained with non-augmented data it achieves an accuracy of 100%. The second best performing model is once again the LSTM when trained with data augmented using SA. We can thus conclude that the approach of using VGGish as a feature extractor and then train and use an LSTM for classification is the most effective one.

11.4.8 Results Evaluation : Observation 9

The increase in performance was quite significant when the fully-connected layers were fine-tuned compared to when they weren't, with the increase being 17.08%, 14.83%, 19.33% and 20% in the respective experiments with the same dataset. The reason for this is quite obvious and it's because by fine tuning these layers we are able to make use of the pre-trained VGGish and make it better suited for our task. The backpropagation function in VGGish+ with no fine-tuning only changes the weights of the final 15 neuron-wide softmax layer whereas un-freezing the other layers allows for our model to better capture patterns that are specific to our task.

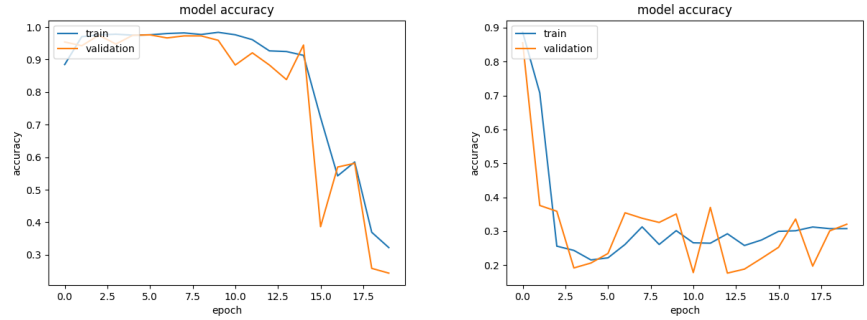
11.4.9 Results Evaluation : Observation 10

While including the final CNN layer in the set of layers to be fine-tuned increased the performance of the model when trained with the non-augmented dataset, specifically by 17.98%, it caused a performance decrease when the model was trained with data augmented with DA, SA and DA & SA. The change was -6.74%, -17.26 and -13.71% respectively. We can understand the reasons behind these results by looking at the training and validation accuracy graphs generated during training:



(a) No augmentation

(b) DA



(c) SA

(d) DA & SA

Figure 10: Training and validation accuracies for VGGish+ model with fine tuning the fully-connected and final CNN layers.

We can see that while the models learn pretty quickly, just like when when no data-augmentation techniques are used, initially reaching a high training accuracy this later this plummets. The reason behind this is that these models, because of the larger dataset, overfit to the data, hence the degradation in performance.

12 Conclusions & Suggested Future Work

12.1 Conclusion

The goal of this project was to use Deep Neural Networks for the purpose of classifying sound events that are typically associated with an office environment. We have explored various approaches such as trying different model architectures including a pre-trained CNN, with and without fine-tuning to different degrees, as well as a RNN. We also looked into different data augmentation techniques not only applied on the raw sound files but also techniques applied on the spectrograms generated from them. We then carried out a number of experiments comparing the performance of each of the models and we then investigated the reasons behind the differences between them.

The set of results gathered from the different experiments have been somewhat unexpected. For example we were surprised that fine-tuning the final CNN layer would actually lead to (mostly) a large decrease in performance when compared to the baseline model. Furthermore, after coming across the promising results found in the paper that introduced SpecAugment[12] we were surprised to see that when we applied it, it caused our models to perform worse than when no augmentation technique was used. Lastly, when DA was applied it did not always lead to better performance and when it did lead to an increase it was not by a significant amount.

With that being said, some of the results are quite encouraging. Most notably, the performance of the LSTM has been better than expected, with the testing accuracy reaching 100% in one of the experiments. The other model that brought promising results is the VGGish+ with fine-tuning the fully-connected layers, with the highest testing accuracy it reached being 71.01%.

Overall, this project has been successful in exploring various machine learning techniques and methodologies to carry out the task we've set out to complete, and we've revealed some of the more promising approaches that may be followed in the future.

12.2 Future Work

The first task we would undertake in a future iteration of this project is to investigate the effectiveness of the LSTM approach, given that one of the experiments resulted in a perfect classification performance. We would like to attempt to replicate the experiment, perhaps with a larger testing dataset, something which would validate or invalidate the claim suggested by our experiment.

Furthermore, we would attempt to tackle the problem of overfitting that we

came across when we fine-tuned the final CNN layer of VGGish+. The graphs in Figure 10 show that the model's training accuracy deteriorates in subsequent training epochs from a point onward and for that reason one could suggest that using a technique such as early stopping[45], would be beneficial.

Lastly, because of the underwhelming effects of the data augmentation techniques we used in this project, we would also explore more of these in the hope of finding one such that it can be reliably used to improve our model's performance.

13 Implications Of Covid-19

As per the instruction of the project coordinator I will be outlining the effects the Covid-19 pandemic has had on this work.

The most significant implication has been the lack of access to the computer labs which have been my work environment since the beginning of my degree. This, in combination with the fact that fully adjusting to a new working environment proved to take more time than anticipated, caused my productivity to decrease.

Furthermore, the closures introduced some technical and practical difficulties. The datasets used for this project are located on the school's `BigTMP` file-system and so direct access to them was not possible from the day of closure onwards. In addition, conducting experiments on my personal machine was not a viable option as the size of the datasets in combination with the lack of appropriate hardware caused my computer to crash on several occasions.

With regards to lost functionality, had the closures not happened I would've conducted a more comprehensive suite of experiments to further consolidate the conclusions we have made from the current set of results.

References

[1] Dafna, E., Tarasiuk, A., & Zigel, Y. (2018). Sleep staging using nocturnal sound analysis. Scientific Reports, 8(1). doi: 10.1038/s41598-018-31748-0

[2] Büchler, M., Allegro, S., Launer, S., & Dillier, N. (2005). Sound Classification in Hearing Aids Inspired by Auditory Scene Analysis. EURASIP Journal On Advances In Signal Processing, 2005(18). doi: 10.1155/asp.2005.2991

[3] Tschannen M., Kramer T., Marti G, Heinzmann M., & Wiatowski T. (2016). Heart Sound Classification Using Deep Structured Features. Dept. IT & EE, ETH Zurich, Switzerland. doi: 10.22489/CinC.2016.162-186

[4] Google Scholar. (2020). Retrieved 27 March 2020, from https://scholar.google.com/

[5] Choi, Y., Atif, O., Lee, J., Park, D., & Chung, Y. (2018). Noise-Robust Sound-Event Classification System with Texture Analysis. Symmetry, 10(9), 402. doi: 10.3390/sym10090402

[6] Sharma J., Granmo O., & Goodwin M. (2020). Environment Sound Classification using Multiple Feature Channels and Attention based Deep Convolutional Neural Network. University of Agder, Norway, arXiv:1908.11219v6. 27 Feb 2020

[7] Haubrick P.,& Ye J.(2019) Robust audio sensing with multi-sound classification. School of Computer Science, University of St Andrews, UK

[8] Salamon J., Jacoby C., & Bello J. (2014). A Dataset and Taxonomy for Urban Sound Research. Music and Audio Research Laboratory & Center for Urban Science and Progress, New York University

[9] Cotton C., & Ellis D (2011). Spectral Vs. Spectro-temporal Features For Acoustic Event Detection. LabROSA, Dept. of Electrical Engineering Columbia University

[10] Freesound - Freesound. (2020). Retrieved 29 March 2020, from https://freesound.org/

[11] Non-speech Audio Event Detection Portelo J., Trancoso I., Neto J., Abad A., & Serralheiro A. (2009). Non-Speech Audio Detection. INESC-ID Lisboa, Portugal, IST, Lisboa, Portugal, Military Academy, Portugal

[12] Park S., Chan W., Zhang Y., Chiu C., Zoph B., Cubuk E., & Le Q. (2019). SpecAugment: A Simple Data Augmentation Method for Automatic Speech Recognition. Google Brain. arXiv:1904.08779v3

[13] Hershey S., Chaudhuri S., Ellis P., Gemmeke F., Jansen A., Moore C., Plakal M., Platt D., Saurous A., Seybold B, Slaney M. CNN architectures for large-scale audio classification. InAcoustics, Speech and Signal Processing

(ICASSP), 2017 IEEE International Conference on 2017 Mar 5 (pp. 13 1-13 5). IEEE.

[14] Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556. 2014 Sep 4 2014

[15] karolpiczak/ESC-50. (2020). Retrieved 29 March 2020, from https://github.com/karolpiczak/ESC-50

[16] Guyot, Patrice. (2018). Simple CNN and vggish model for high-level sound categorization within the Making Sense of Sounds challenge.

[17] Tompkins D., & Nichols E. (2019). DCASE 2019 Challenge Task 5: CNN+VGGish. Microsoft, Dynamics 365 AI Research.

[18] Zhang J., & Wu J. (2019). DCASE 2019 Task 2: Semi-Supervised Networks With Heavy Data Augmentations To Battle Against Label Noise In Audio Tagging Task'.

[19] Xie, Huang & Virtanen, Tuomas. (2019). Zero-Shot Audio Classification Based on Class Label Embeddings.

[20] Mikolov, Tomas & Sutskever, Ilya & Chen, Kai & Corrado, G.s & Dean, Jeffrey. (2013). Distributed Representations of Words and Phrases and their Compositionality. Advances in Neural Information Processing Systems. 26.

[21] Parascandolo G., Huttunen H., & Virtanen T. (2016). Recurrent Neural Networks For Polyphonic Sound Event Detection In Real Life Recordings. Department of Signal Processing, Tampere University of Technology. arXiv:1604.00861v1

[22] Vavrek, Jozef & Pleva, Matus & Juhár, Jozef. (2010). Acoustic events detection with Support Vector Machines.

[23] Dhanalakshmi, P., Palanivel, S., & Ramalingam, V. (2011). Classification of audio signals using AANN and GMM. Applied Soft Computing, 11(1), 716-723. doi: 10.1016/j.asoc.2009.12.033

[24] Balili, Christine & Sobrepena, Ma & Nava l, Prospero. (2015). Classification of heart sounds using discrete and continuous wavelet transform and random forests. 655-659. 10.1109/ACPR.2015.7486584.

[25] Phan H., Koch P., Katzberg F., Maass M., Mazur R., & Mertins A. (2017). Audio Scene Classification with Deep Recurrent Neural Networks. Institute for Signal Processing, University of Lubeck. arXiv:1703.04770v2

[26] Brownlee, J. (2020). Data Preparation for Variable Length Input Sequences. Retrieved 20 April 2020, from https://machinelearningmastery.com/data-preparation-variable-length-input-sequences-sequence-prediction/

[27] jiaaro/pydub. (2020). Retrieved 2 April 2020, from https://github.com/jiaaro/pydub

[28] Consulting, A. (2020). Keras Tutorial : Fine-tuning pre-trained models — Learn OpenCV. Retrieved 2 April 2020, from https://www.learnopencv.com/keras-tutorial-fine-tuning-using-pre-trained-models/

[29] numpy.lib.format — NumPy v1.17 Manual. (2020). Retrieved 6 April 2020, from https://docs.scipy.org/doc/numpy/reference/generated/numpy.lib.format.html

[30] jiaaro/pydub. (2020). Retrieved 6 April 2020, from https://github.com/jiaaro/pydub/blob/master/pydub/silence.py

[31] pmhaubrick/CNN-Polyphonic-Audio-Classification. (2020). Retrieved 6 April 2020, from https://github.com/pmhaubrick/CNN-Polyphonic-Audio-Classification/blob/master/Augment/augmentation.py

[32] tensorflow/models. (2020). Retrieved 6 April 2020, from https://github.com/tensorflow/models/blob/master/research/audioset/vggish/vggish_input.py

[33] numpy.array — NumPy v1.17 Manual. (2020). Retrieved 6 April 2020, from https://docs.scipy.org/doc/numpy/reference/generated/numpy.array.html

[34] tensorflow/models. (2020). Retrieved 6 April 2020, from https://github.com/tensorflow/models/blob/master/research/audioset/vggish/vggish_params.py

[35] KimJeongSun/SpecAugment_numpy_scipy. (2020). Retrieved 9 April 2020, from https://github.com/KimJeongSun/SpecAugment_numpy_scipy

[36] sklearn.model_selection.train_test_split — scikit-learn 0.22.2 documentation. (2020). Retrieved 9 April 2020, from https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

[37] tensorflow/models. (2020). Retrieved 9 April 2020, from https://github.com/tensorflow/models/tree/master/research/audioset/vggish

[38] Home - Keras Documentation. (2020). Retrieved 9 April 2020, from https://keras.io/

[39] DTaoo/VGGish. (2020). Retrieved 10 April 2020, from https://github.com/DTaoo/VGGish

[40] Brownlee, J. (2020). Loss and Loss Functions for Training Deep Learning Neural Networks. Retrieved 10 April 2020, from https://machinelearningmastery.com/loss-and-loss-functions-for-training-deep-learning-neural-networks/

[41] Sequence Preprocessing - Keras Documentation. (2020). Retrieved 20 April 2020, from https://keras.io/preprocessing/sequence/

[42] An overview of gradient descent optimization algorithms. (2020). Retrieved 10 April 2020, from https://ruder.io/optimizing-gradient-descent/index.html#rmsprop

[43] sklearn.metrics.confusion_matrix — scikit-learn 0.22.2 documentation. (2020). Retrieved 10 April 2020, from https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html

[44] sklearn.metrics.classification_report — scikit-learn 0.22.2 documentation. (2020). Retrieved 10 April 2020, from https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html

[45] Brownlee, J. (2020). Use Early Stopping to Halt the Training of Neural Networks At the Right Time. Retrieved 15 April 2020, from https://machinelearningmastery.com/how-to-stop-training-deep-neural-networks-at-the-right-time-using-early-stopping/