Automatic Detection of Chimpanzee Vocalizations using Convolutional Neural Networks

A Thesis Presented to

The Faculty of the Computer Science Department

California State University Channel Islands

In (Partial) Fulfillment

of the Requirements for the Degree

Masters of Science in Computer Science

by

Chanakya Dev

Advisor: Jason Isaacs

June 2020

APPROVED FOR MS IN COMPUTER SCIENCE

[illegible] 07/09/2020

Advisor: Jason Isaacs — Date

[illegible] 07/09/2020

Bahareh Abbasi — Date

[illegible] 07/09/20

Vida Vakilian — Date

APPROVED FOR THE UNIVERSITY

Dr. Osman Özturgut 07/17/2020

Osman Özturgut — Date

Automatic Detection of Chimpanzee Vocalizatons using Convolutional Neural Networks
Title of Item

Machine Learning, Neural Networks, Automatic Sound Recognition, Chimpanzee
3 to 5 keywords or phrases to describe the item

Chanakya Dev Nakka
Author(s) Name (Print)

[illegible] 07/17/2020
Author(s) Signature Date

Automatic Detection of Chimpanzee Vocalizations using Convolutional Neural Networks

Chanakya Dev

July 17, 2020

Abstract

The organization, Greater Mahale Ecosystem Research and Conservation (GMERC), is focused on the study and conservation of various primate species in the region of Western Tanzania. For effective primatology studies, it is imperative to track and observe primate behavior at a distance. Such methods are non-intrusive, increase encounter durations and provide insights into behavior patterns that are found naturally. In this thesis, I propose a method to automatically detect chimpanzee vocalizations using machine learning techniques. The models can be deployed on micro-computers which are low cost and solar powered. This method can greatly increase encounter durations while directly reducing manual human effort. For this purpose, I have created a dataset using Urbansound8K and added two distinct kinds of chimpanzee vocalizations to define a multiclass classification problem. I have evaluated the effectiveness of various audio features and different classification techniques and have optimized them for 3 key factors: accuracy, memory footprint and processing speed. Our optimal solution was a 4 layered Convolutional Neural Network that used 40 Mel Frequency Cepstral Coefficients as input features. This method gave an accuracy of 81.8%, occupied less than 10 Mb of space and could perform inference every second on a commonly available micro-computer. Lastly, I outline how this framework can be used to localize the source of the audio post-classification that can aid the tracking and preservation of primates in the Issa Valley, Tanzania.

Contents

List of Figures

1 Introduction

1.1 Introduction to GMERC

The Greater Mahale Ecosystem Research and Conservation (GMREC) is a wildlife research team incorporated in Tanzania as GMERC Ltd [1]. It was previously known as the Ugalla Primate project and focuses its efforts on the ecology, behavior patterns and conservation of primates and other wildlife within the Greater Mahale Ecosystem (GME). This region is characterized primarily of miombo woodlands with significant patches of wooded grasslands, swamps and riverine forests. This area of western Tanzania is one of the most mosaic landscapes where chimpanzees live. It is open, seasonal and has low annual precipitation.

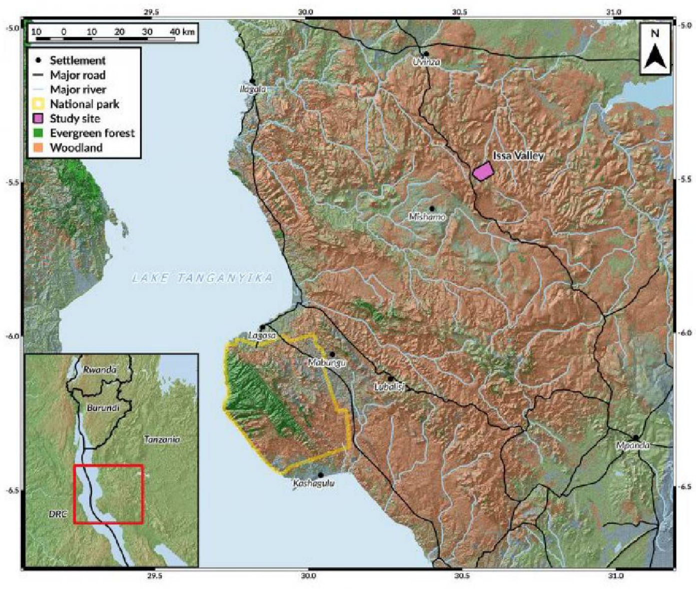


Figure 1: Study site from: About us, Greater Mahale Ecosystem Research and Conservation. https://gmerc.org/about-the-gmerc

The Issa Valley, in western Tanzania houses a study site as shown in Figure 1. Primatology research interest also stretches to other areas in the region including those along Lake Tanganyika, Gombe and Mahale Mountains

National Park. Since 2011, numerous chimpanzee sub-population habitats within the larger geographical areas of GME have been monitored and thus reflected in the name for the project.

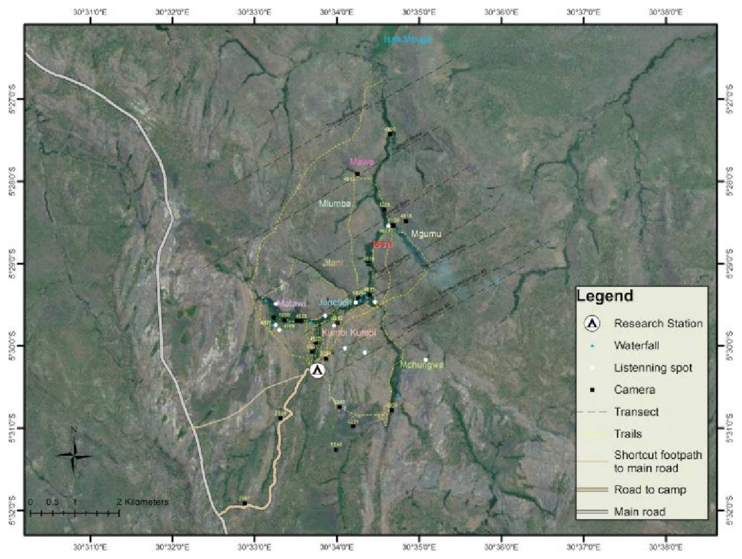


Figure 2: Long term research site from: About us, Greater Mahale Ecosystem Research and Conservation. https://gmerc.org/about-the-gmerc

The long term research site as shown in Figure 2 is habitat with savanna-woodland characteristics. The nature of this habitat is very important to understanding behavioral ecology of hominin. The key questions for the research focus on the social consequences of relatively low population densities and group sizes. It is a model for us to understand how we adapted to arid and mosaic habitats. Also, understanding the contexts of chimpanzees and other primates in these regions has the potential to provide insights regarding bipedalism and the explosion in brain sizes.

To be useful paleoenvironmental models, this information must be aggregated with fossil records and in conjunction with behavioral ecological theory to draw potential conclusions. From a conservation and an anthropological perspective, these apes are interesting to study. Since some of our earliest ancestors thrived in similar environments, understanding these ecosystems provide us glimpses into the origins of Homo-Sapiens. Sample association patterns with dung and hair sample studies allow for genotype extraction and inferences regarding group structure and organization [2], [3]. Specific to the greater apes, the conditions posed by these habitats demand novel and flexible social organizations, which is not usually a norm for their forest dwelling counterparts. Territories and group sizes are also determined using spatial distribution and clustering of sleeping nest sites [4].

Most of what we know about the great apes, our early ancestors who's DNA is 98% similar to ours comes from the studies restricted to groups which have developed tolerance towards human intervention. It is also to be noted that such approaches require investments in the form of time and faces ethical and feasibility issues for most of the wild populations. Whereas indirect means were used to conduct behavioral research, newer methods are being adopted to improve precision and efficiency [5]. Since 2018, chimpanzees nests in the valley could be identified and previous theories that were developed via indirect means could be revisited. The usage of Unmanned Aerial Vehicles (UAVs) increased contact durations and reduced contact distances dramatically. Whilst previously, a typical encounter would last for 1-2 hours and from a distance of 40-50 meters, now individuals could be tracked from 10-15 meters for an entire day. Although this method showed potential, some individuals were often difficult to track and thus remain elusive. In addition to this approach, acoustic monitoring using micro-computers provides an additional benefits to aid the tracking efforts. The stationary nature of these devices make it less intrusive in the habitat when compared to UAVs. Moreover, the devices are capable of automatically detecting chimpanzee vocalizations for weeks, with minimal human intervention.

1.2 Bio-acoustics

Bio-acoustics refers to the study of producing, dispersing and the reception of sounds in animals. The scope of the discipline covers the anatomical and neuro-physiological basis of acoustic mechanisms in organisms [6]. With specific reference to marine life, the term also implies the usage of sonar technology and biomass estimation [7].

Bio-acoustics research at Issa Valley was initially focused on chimpanzee long calls or hoots and their significance in selecting nesting sites and mediating parties [8]. These parties are organized in smaller numbers during the daytime and in significantly larger group sizes during the night. Given the smaller population densities and broader home ranges, the organization of such parties in this kind of a fragmented social system remains ambiguous.

It was not since 2006 that the simultaneous observation of caller and listener in the chimpanzee's socioecological context became a reality. It was often logistically difficult to observe and was a necessary validation for the chimpanzee long calls and their functional significance. Autonomous Recording Units (ARU's) by Cornell University, which have been historically successful for marine wildlife monitoring and localization have proven to be a boon for primatologists [9].

This approach enabled researchers to conduct audio based surveys in areas that were remote, inaccessible and unmanoeuvrable. The deployment of such systems in the Issa Valley made it possible to conduct long range observations (from >2km) and localization (from <1km) by detecting both natures of acoustic signals(hoots and shrieks) made by wild chimpanzees. The same devices were used to detect other fauna of interest like frogs, bushbacks and other primates. These devices can be solar powered and relay recorded sounds to a centralized receiver. To capture the variations induced by seasonality, 21 ARU's have been deployed. These devices are designed for capturing audio on a yearlong basis and seek to determine ranging behavior as a response to food availability, seasonality and weather.

Although much has been achieved through the implementation of ARU's, one limiting factor for primatology and other terrestrial applications remains automated detection and mining of vocalizations on a realtime basis. Applications with good classifier robustness that can tackle the high variability of acoustic signals in terrestrial ecosystems is essential for the monitoring efforts for terrestrial mammals. Such applications have attracted a lot of research efforts over the last decade [10].

Providing a solution to this problem is the central focus for our research, which attempts to monitor calls of interest in real-time. Leveraging the advent of improved embedded systems, classification techniques and remotely controlled ARU's, detection and transmission of acoustic information can be performed. With minimal human intervention and improved precision, accoustic estimation of chimpanzee density can be determined and compared with the results of previous research.

1.3 Main Contributions

This thesis focuses on developing a framework for acoustic monitoring of chimpanzee populations. Firstly, I overview the applications and advancements in automatic sound recognition. The classification of sound by machines is an ever evolving area of research. The biggest challenge to this field is to develop customer-centric applications that require a fair degree of robustness to real world acoustic noise and distortions.

Analyzing and classifying raw audio signals requires a tremendous amount of compute. These signals have to be transformed into their lower dimensional representations. Using such transformations, I aim to compress the audio clip and still retain most of the meaningful information. A wide perspective on acoustic features and their extraction procedures is provided. These features closely mimic the neural signals that are transformed in our body by the ear and auditory cortex.

Given the task of classifying chimpanzee 'hoots' and 'shrieks' from other sounds in nature, a description of the current state of research and development is provided. I review the merits and demerits of the traditional and newer classification approaches to understand what fits best for our current case. Since the predictions are performed on a smaller inference engine, it is imperative to develop a lightweight system. An overview of the system setup is given in Figure 3.

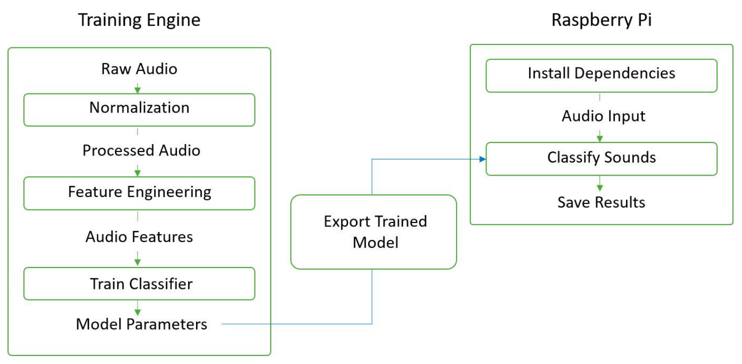


Figure 3: System Setup

1.4 Remaining Chapters

Chapter 2 highlights some of the key research in automatic sound recognition. Some of the prominent time and frequency domain features that are used for audio analysis have been investigated. Mel Frequency Cepstral Coefficients (MFCCs) which are some of the most widely used features of choice are revisited in greater detail. Also, Gammatone Cepstral Coefficients (GTCCs) have been experimented with, as viable alternative to MFCCs. For the classification tasks, Support Vector Machines were by far the most widely used classifiers untill they were replaced with perceptron based models. A wide perspective is then provided on the features and classifiers used in acoustic monitoring systems. A description of the current state of research and development is provided, after which the current experimental setup is discussed in Chapter 3. The results from the experiments and the final model to be used are covered in Chapter 4. In Chapter 5, I demonstrate how real-time detection is performed when the final model is deployed on a micro-computer. Future work and continued studies are discussed in Chapter 6.

1.5 Key Terms

machine learning: Multi-disciplinary field that brings together concepts from computer science, statistics and data analysis to understand and extract information from the ever increasing amounts of data.

classification: Assigning something to a discrete set of possibilities. ex. spam or non spam, Democrat or Republican, blood type (A, B, AB, O).

supervised learning: The machine learning task of inferring a function from labeled training data consisting of pairs of input objects and a desired output value.

categorical data: Data that can be labeled or divided into groups. ex. race, sex, hair, color.

inference: Understanding how the output changes with respect to different input values.

WAVE/WAV: Waveform Audio File Format; a uncompressed, lossless audio file format standard for storing an audio bitstream on PCs developed by Microsoft and IBM.

2 Automatic Sound Recognition

With the advent of machine learning and embedded systems, automatic sound recognition (ASR) has attracted a lot of interest in the recent years. I will briefly review some of the important advancements in speech and non speech based domains. Like any classification exercise, the robustness of sound classification systems depends on the choice of features and classifiers used. The key difference between automatic recording units (ARUs) and ASR systems is that an ASR system is intended to process signals, apply machine learning techniques and recognize sounds automatically. The inputs can be speech or non-speech based on the system objectives. Preliminary research in ASR was related to content-based [11]–[13], speech [14], and non-speech [15] based audio classification. Eventually, diversified applications in audio like genre classification and instrument identification have been developed.

Other applications of non-speech based sound recognition are sound event recognition [16], environmental sound recognition [17], and audio surveillance [18]. Environmental sound recognition(ESR) in particular, poses a unique challenge because of the presence of numerous sounds in the environment which can be observed in different combinations at any given point in time. While the applications can be many, a generic method for building a sound recognition system follows three key steps: signal processing, feature engineering, and classification. Signal processing prepares and cleans the raw audio signal for feature engineering. Commonly, an audio signal is divided into frames with lengths ranging from 15-40 ms onto which a window function is panned across. This smoothens the signal and allows for further analysis. Commonly used sampling frequencies are 8KHz, 16KHz, 22KHz and 44KHz, depending on the Nyquist criterion and the frequency bands of the sound signals. For a waveform to be reconstructed correctly, the Nyquist criterion states that the sampling frequency must be greater than or equal to twice of the frequency to be sampled. Depending on the frequency selected, frame sizes are chosen accordingly (256, 512, 1024 samples for each frame). A certain amount of overlap is also incorporated when transitioning from one frame to another for preventing any loss of information around the frame edges. Inherently, time domain and frequency domain features are extracted from these signals which are often condensed and adequately representative. Most of the traditional features continue to be in use today with successful machine learning pipelines being built around them. Audio features may be broadly classified into:

- Time domain: features in the sound signal like amplitude, power and their temporal patterns.
- Frequency domain: semantically meaningful perceptual features which describe physical and statistical properties on the signal.
- Modulation frequency features: features representing long term variations in amplitude and frequency of the signal.
- Cepstral features: features approximating the modular and spectral envelopes.
- Eigen domain features: another long term feature representation with window durations ranging across several seconds.
- Phase space features: orthogonal feature information representation derived from linear models.

The most frequently used features for classification systems are cepstral features, followed by time and frequency domain features. Historically Support Vector Machines(SVMs) [12], [19], [20] were the most popular classifiers of choice. Other classification methods incorporated Hidden Markov Models (HMMs) [21], Neural Networks(NNs) [22], [23], k-Nearest Neighbors(kNNs) [24] and Gaussian Mixture Models(GMMs) [21]. Numerous systems incorporating hybrid and modified versions of these legacy classifiers are still being used today and are defined extensively in literature. Today, the development of deep learning has spread its horizons onto the audio domain as well, with significant amount of research being devoted to its applications in pattern recognition. With audio classification as a deterministic, closed set finite problem having sufficient labelled data at its disposal, deep learning is a practical approach for such problems.

Classification performance for ASR systems is generally measured via precision-recall metrics which can be given as a percentage of relevant results out of all the retrieved instances and relevant instances retrieved over the total amount of relevant instances. The F1 score, which is the harmonic mean of precision and recall may also be used in conjunction. In our study, I derive inspiration from the feature extraction principles and classification mechanisms developed for content-based audio classification [12], [13], surveillance [18] and ESR systems[21], [25].

Given its usage in the previous ASR systems for the same purpose [8], I have tried Support Vector Machines as choice of classifiers. I also give an overview of Neural Network architectures which have gained in prominence for developing speech recognition systems and other ASR systems as well. Finally, I propose the method that balances out adequate classification power, quick prediction time and low memory footprint.

2.1 Feature Engineering

Early works in acoustic monitoring observed much feasibility in marine ecosystems because sound propagates further in water than air. One such application also saw commercial success [11]. Using normalized Euclidean distances, a KNN classifier was built using abstract acoustic features like bandwidth, spectral centroid, loudness and pitch. Other features in the time-frequency domain used were short-time energy (STE), zero-crossing rate (ZCR), spectral flux (SF), spectral roll-off (SR) and subband energy (SBE). Even though these features are often revisited from time to time in recent applications, they are used as information that supplement some of the primary features used such as cepstral features. Because of the attention gained by cepstral features for speech and non-speech based applications, their performance on our test case will be investigated .

2.1.1 Mel Frequency Cepstral Coefficients

The human auditory perception is enhanced at lower frequency ranges than at higher frequency ranges. At lower frequencies, we are able to distinguish changes in pitch better. Mel frequency cepstral coefficients(MFCCs) were first introduced for the Muscle fish database [19]. When frequency bands are spaced equally on the Mel scale instead of the linearly spaced Hertz scale, it becomes intuitively easier to interpret sounds. Additionally since changes in frequencies are much more informative in a cepstrum, a combination of Mel scaled frequencies and their cepstrums give us MFCCs which are useful in classification based applications. Based on the procedure outlined by Sharan and Moir in [26], the windowed signal is first subjected to a discrete

Fourier transform(DFT) to compute the MFCCs:

$$X(k,t) = \sum_{n=0}^{N-1} x(n)w(n)e^{\frac{-2\pi ikn}{N}}, \qquad k = 0, ..., N-1 \tag{1}$$

where $x(n)$ is the sampled time domain signal, $w(n)$ is the window function, and N is the window length at frame t. Given a sampling frequency F_s , $X(k,t)$ represents the kth harmonic for frequency $f(k)$. Here, $f(k)$ is denoted as $f(k) = kF_s/N$. In order to convert frequencies in Hertz (f_{Hz}) to Mel scaled frequencies (f_{Mel}):

$$f_{Mel} = 1127log(1 + \frac{f_{Hz}}{700}) \tag{2}$$

For M_1 number of filters, the mth filter's central frequencies can be computed as:

$$f_{c\_m} = f_l + \frac{m(f_h - f_l)}{M_1 + 1}, \qquad m = 1, 2, ..., M_1 \tag{3}$$

After converting the frequencies to the mel scale, the minimum and maximum threshold frequencies are correspondingly scaled to f_l and f_h. To incorporate a degree of overlap, the filter boundaries are adjusted such that a filter begins at the central frequency of its preceding filter and ends at the central frequency of the next filter. For the mth filter and frame t, the filter bank energy outputs can be determined as:

$$E(m,t) = \sum_{k=0}^{\frac{N}{2}-1} V(m,k)|X(k,t)|^2, \qquad m = 1, 2, ..., M_1 \tag{4}$$

The frequency response is normalized and given as V(m,k). The filter bank energies are log compressed and passed through a discrete cosine transformation to give MFCCs:

$$c(i,t) = \sqrt{\frac{2}{M_1}} \sum_{m=1}^{M_1} ln(E(m,t))cos(\frac{\pi i}{M_1}(m - 0.5)), \qquad i = 1, 2, ..., L \tag{5}$$

Here, L is the cepstrum order. Normally, 20-25 filters are used in the bank in constant mel-frequency intervals spread across the entire input frequency range. For example, 23 filters in [17], [20], 20 filters in [18], [27] and 19 filters in [19] are used to sufficiently discriminate characteristics of input classes. For observing changes of MFCCs over time, its first order and second order derivatives are also used. These trajectories improve the performance of some classifiers. These derivatives are called the delta and double-delta coefficients and are computed as:

$$c_{\Delta}(i,t) = \frac{\sum_{d=1}^{D} d[c(i+d,t) - c(i-d,t)]}{2\sum_{d=1}^{D} d^2} \tag{6}$$

where t is the frame, i is the corresponding delta coefficient and value D which is normally set to 2. A similar approach can be used to compute the double-delta coefficients $c_{(\Delta-\Delta)}(i,t)$. Along each feature dimension, the mean and standard deviations are used as feature vector representations for each audio signal. It is a common practice to normalize the coefficients to remove distortions caused by environmental conditions. Although there are many techniques to normalize this data, Sharan and Moir in [26] suggest the usage of Cepstral mean variance normalization(CMVN) for this approach.

$$c_{CMVN}(i,t) = \frac{c(i,t) - \mu(i)}{\sigma(i)} \tag{7}$$

where $\mu(i)$ is the mean and $\sigma(i)$ is the variance for the dimension i. Cepstral Scaling(CS), another normalization method with [0 1] as bounds can also be used as:

$$c_{CS}(i,t) = \frac{c(i,t) - min(c(i))}{max(c(i)) - min(c(i))} \tag{8}$$

where $max(c(i))$ is the maximum and $min(c(i))$ is the minimum value for dimension i. The same equations can be used for normalization of the first and second order derivatives. Further normalization of background sounds can be achieved by first classifying the sounds as non-silent and silent based on the signal magnitude sum and a predefined threshold [19]. The resulting non-silent features are concatenated and used as a feature vector where further normalization can be performed. Cepstral features can later be combined with other perceptual features to improve classification performance.

When observing the performance of MFCCs under various noisy conditions, a large repository of environmental sounds is required to capture the effects of induced variations. Without this, MFCCs tend to perform rather poorly for both speech and non-speech based applications [16], [28]. Techniques like root compression and log spectrum compression are solutions for cepstral analysis problems where there is a significant amount of noise [14]. For low energy components, disproportionate variations in the cepstrum [29] can be effectively captured using log compression and peaks of the mel-filter banks. Root compression for the cepstral coefficients instead of log compression further improves the robustness of the classifiers when using MFCCs:

$$c(i,t) = \sqrt{\frac{2}{M_1}} \sum_{m=1}^{M_1} E(m,t)^{\gamma} cos(\frac{\pi i}{M_1}(m-0.5)), \qquad i = 1,2,...,L \tag{9}$$

Root compression is used on the filter bank energies with a γ root value where $0 < \gamma \leq 1$. When $\gamma = 1$, the resulting cepstral coefficients are linear with uncompressed filter bank energies. For low signal-to-noise conditions, γ is set closer to 1 and results in higher accuracies for these feature vectors [18]. One such application with air-conditioner noise as background used independent component analysis, a technique used for data separability in conjunction with MFCCs [30]. To enhance the robustness, this application uses a combination of two techniques in conjunction: a subspace based signal enhancement that takes into account the changing signal to noise ratio (SNR) and an independent component analysis using MFCCs as features. Finally, a multiclass SVM is used for the classification exercise. Other variations for deriving filter bank energies using Gammatone Cepstral Coefficients(GTCCs) and Power Normalized Cepstral Coefficients(PNCCs) [31] are also useful when the nature of the background noise is reverberant.

2.1.2 Gammatone Cepstral Coefficients(GTCCs)

While MFCCs still remain the most widely used features of choice, GTCCs have been gaining a lot of traction among the other cepstral features. Instead of using the Mel scale, a gammatone filter is used which derives its inspiration from the human ear anatomy. The cochlea's frequency selectivity can be determined using knowledge of our cavity structure and receptors. Patterson's [32] cochlea model is the most widely used model where band-pass filters are defined using equivalent rectangular bandwidth (ERB). Some implementations of these filterbanks have been proposed, with Apple Computer's work [33] being the most followed procedure for both speech [34] and non-speech [35] based applications. For extracting GTCCs the corresponding gammatone filter impulse response is given as:

$$g(r) = Ar^{j-1}e^{-2\pi Br}cos(2\pi f_c r + \phi) \tag{10}$$

where the amplitude is denoted as A, filter order as j , filter bandwidth as B , central frequency as f_c with phase ϕ and time r.

Each of the cochlea's filter bandwidths are desribed by the ERB [14], where ERB is the perceptual filter width along various points of the cochlear structure.

$$f_{cERB} = [(\frac{f_{cHz}}{Q_{ear}}) + (B_{min})^p]^{1/p} \tag{11}$$

The filter quality asymptotics are given as Q_{ear} at higher frequency ranges and the low frequency bandwidth threshold is given as B_{min}. We can then approximate the filter bandwidth by $B = 1.019 \times f_{cERB}$. There are different sets of parameters used for different cochlear models, ex. ($p = 1$, $B_{min} = 24.7$ and $Q_{ear} = 9.26$) as given in [34], ($p = 2$, $B_{min} = 125$, and $Q_{ear} = 8$) in [33] and ($p = 1$, $B_{min} = 22.85$, and $Q_{ear} = 7.23$) in [36]. In every model, a characteristic frequency and a specific bandwidth is resonated by the hair cells inside the cochlea. An overlap between the filters is indicated using a step factor parameter integrated with mappings between central frequencies and cochlear positions. The filter index and central frequencies can then be mapped as

$$f_{cm} = -Q_{ear}B_{min} + (f_h + Q_{ear}B_{min})e^{-ms/Q_{ear}}, \qquad m = 1, 2, ..., M_2 \tag{12}$$

where the filter bank's upper frequency is given as f_h and number of filters used is M_2. The step factor is given as s where

$$s = \frac{Q_{ear}}{M_2}log(\frac{f_h + Q_{ear}B_{min}}{f_l + Q_{ear}B_{min}}) \tag{13}$$

and f_l is the filter bank's lower frequency limit.

A gammatone filter using 2nd order digital filters for four stages is explained in [33] and its corresponding implementation is provided in MATLAB [37]. A detailed analysis on the performance of MFCCs and GTCCs was performed in [35] where it was concluded that GTCCs outperform MFCCs for non-speech based applications in the lower frequency ranges.

2.1.3 Other representations in the time-frequency domain

Alternatively, one of the most popular time frequency image representations are done using the Short-time Fourier transform(STFT). Equally spaced frequency components are plotted against constant bandwidth increments. However, most of the audio signals for classification purposes have frequencies in the lower ranges than on the higher ranges. As a consequence, some information is lost for the lower frequency ranges and uninformative aspects pertaining to higher frequency ranges is retained in this kind of representation.

STFT spectrograms can be modified to use mel-filters and gammatone filters to give two different kinds of cochleagrams called Melspectrogram and Gammatone spectrogram respectively. The common aspect of both these representations is that smaller bandwidths are used for lower frequencies and larger bandwidths for higher frequencies. Since there are more components in the lower range of the spectrum, they are more suitable for regular audio classification tasks. Audio separation [38] and speech recognition [39] are some of the applications that derive features using image-based representations. For example, when a signal is windowed and scaled using the gammatone filter, each frequency component's energies can be represented as

$$C(m,t) = \sum_{n=0}^{N-1} |\tilde{x}(m,n)| w(n), \qquad m = 1,2,...,M_2 \tag{14}$$

where $C(m,t)$ is the mth harmonic corresponding to the gammatone filtered signal $\tilde{x}(m,n)$ and window function $w(n)$. The results obtained are then normalized in the range [0,1] to get the grayscale image intensity values for the cochleagram. This method of extracting features from cochleagrams has been shown to improve classification performance at low SNRs [38], [39]. Further improvement in performance can be achieved when aggregated with MFCCs and GTCCs.

2.2 Classification

In this work, I only focus on two classifiers, Support Vector Machines (SVMs) and Neural Networks (NNs). I provide a brief background and discuss their performance. The supervised classification approach consists of an preliminary training process that uses labelled data to train and develop a predictive model. This is followed by building a classifier that learns examples from the labelled data and performs predictions on unseen data. Classification of environmental sounds that do not consist of music or speech data is predominantly based on applying general classifiers like Gaussian Mixture Models (GMMs) and Hidden Markov Models (HMMs) because of the diverse and chaotic nature in their audio structures. One of the main problems with training Deep Neural architectures in a supervised manner is the amount of computational effort and labeled data required for efficient learning. While the former is in some part addressed on a universal basis by hardware advances and general-purpose GPU computing, the latter is very domain-dependent.

The post-training online process will use the trained model to generate predictions on unseen data and classify the sounds. This is performed on our ARU of choice - the Raspberry Pi 3. Google TensorFlow 1.9 officially supports the Raspberry Pi, making it possible to quickly install TensorFlow and start deploying machine learning applications with a Raspberry Pi.

2.2.1 Support Vector Machines

Initially developed for linear classification tasks in 1963 [40], Support Vector Machines (SVM's) have been used for non-linear datasets as well since 1992 [41]. This technique has since then gained a lot of attention along with the interests towards Automatic Sound Recognition [42]. Until the evolution of Deep Learning techniques for ASR, SVM's have widely been the classifier of choice for audio based classification tasks.

Hyperplanes: SVMs operate on the principle called a hyperplane that seeks to separate two classes by maximizing the distance between the corresponding hyperplanes. The simplest example is the dividing boundary of a single linearly separable dataset. Similarly, we can assume l training examples divided into two classes denoted as $(x_1, y_1), ..., (x_l, y_l)$. Here $x_i \in R^d$ is a feature vector having d dimensions which represents the ith training example. The corresponding labels are represented as $y_i \in \{-1, +1\}$. For obtaining the optimal hyperplane, we must minimize $||w||^2$ where $y_i(w \cdot x_i + c) \geq 1$. Here, $w \in R^d$ is a vector normal to the corresponding hyperplane and c is a constant.
Using the Lagrangian duality principle, we obtain:

$$w = \sum_{i=1}^{l} \alpha_i y_i x_i \tag{15}$$

Here the support vectors are x_i and the Lagrange multipliers are denoted as α_i where $\alpha_i > 0$. These support vectors represent the margin that satisfies $y_i(w \cdot x_i + b) = 1, i = 1, 2, ...l$. The offset to these margins can be determined as

$$b = y_i - w \cdot x_i \tag{16}$$

which is averaged over all the obtained support vectors.

For problems where a nonlinear decision boundary is required, the input vectors x are mapped through $\phi(x)$ into a space of higher dimensionality z to determine optimal hyperplanes. A kernel trick which was first seen in [41] can be used to create nonlinear classifiers. The inner product of $\Phi(x_i)$ and $\Phi(x_j)$ is computed using a non-linear kernel function $K(x_i; x_j)$. The Radial Basis Function (also known as Gaussian RBF) is commonly used as the kernel function:

$$K(x_i, x_j) = exp(-||x_i - x_j||^2/2\sigma^2) \tag{17}$$

where the width of the Gaussian function is denoted as $\sigma > 0$. The classifier thus obtained can be denoted as

$$f(x) = sgn\left(\sum_{i=1}^{l} \alpha_i y_i K(x_i, x) + b\right). \tag{18}$$

Multiclass Classification: Originally a binary classifier, the SVM has been modified into a multiclass classifier using a few novel techniques. The primary idea behind these techniques is the breaking down of the problem into several binary classification tasks. Three methods have hence been developed for tackling the problem of multiclass classification - one versus one, one versus all and Directed Acyclic Graphs (DAG).

Some of the earliest developments were seen for the one versus all method of SVM multiclass classification. Here, n independent classifiers are constructed for a n class dataset. Each of these classifiers are trained with samples from one class as positive and the rest as negative. At classification time, an example x is categorized based on the decision function

$$f(x) = \underset{i=1,2,\ldots,n}{\operatorname{argmax}}(w^i \cdot \Phi(x) + b^i). \tag{19}$$

The one versus one method performs classification between pairs of classes. An approach called as max-wins voting is used here. For an n class classification problem, $n(n-1)/2$ classifiers are constructed and two classes are taken at a time to construct each SVM. A test example can be classified as

$$f(x) = \underset{i=1,2,\ldots,n}{\operatorname{argmax}} \sum_{j=1, j\neq i}^{n} sgn(w^{ij} \cdot \Phi(x) + b^{ij}). \tag{20}$$

where i and j are classes.

The Directed Acyclic Graphs (DAG) method also takes two classes at a time but uses decision trees to perform classification during the test phase. Similar to the one versus one method, $n(n-1)/2$ classifiers are built during the training phase. The only difference is that by using decision trees, only $n-1$ classifiers are required for each classification during the test phase.

In general, SMVs outperform most classifiers except today's perceptron based models. In some cases, Hidden Markov Models (HMMs) perform better than linear SVMs [16]. For classification problems that are complex in nature, nonlinear SVMs are preferred. In [17], the results obtained from binary SVMs and HMMs are not significantly different. It is to be noted that iterating over the parameters c and σ can help in avoiding local minima and further improve the performance of the classifier.

2.2.2 Neural Networks

The introduction of deep learning has paved the way for several advancements in the field of Automatic Sound Recognition. Some of the earlier methods relied heavily on the way in which the acoustic features were preprocessed [30], [31], [33], [38]. Since manually labelling and annotating sound events is a cost intensive task, there are a limited number of datasets that are publicly available. However, Convolutional Neural Networks (CNNs) and data augmentation make it possible to analyze environmental sounds even when the training data is limited [21], [25].

Backpropagation: All Neural Networks are based on the principle of backpropagation. During initialization, the weights and biases and even filter values (in case of CNNs) are often randomized and do not represent meaningful features. It is only after exposing them to the labelled observations that the parameters get tuned accordingly. A single backpropagation iteration consists of a forward pass, computation of the loss function, a backward pass and parameter update.

When training examples are passed through the network, the first output will be unbiased and wont give preference to any output in particular since the weights and biases are randomly initialized. A loss function is then computed, for example, the mean squared error (MSE) in a standard regression task with N training examples is given by:

$$L = \sum_{i=1}^{N} 0.5(target_i - output_i)^2 \tag{21}$$

For the first few iterations, we observe high loss values. If we want to reach a point where the predicted examples represent their corresponding training examples, we need to minimize the loss function. Therefore, we need to revisit the weights and biases in our network that contribute to a high amount of loss for the network. Mathematically, this can be represented as a calculus problem where we are computing dL/dW for the layer weights.

The backward pass here determines the amount of loss that the individual weights have contributed to the final loss calculation. For updating the weights, we use a learning rate to update them and mitigate a portion of their loss contribution.

$$w = w_i - \eta \frac{dL}{dW} \tag{22}$$

where w_i is the initial weight and w is the updated weight after one backward pass. The learning rate η is the learning rate hyperparameter. Higher learning rates speed up the training process by making bigger updates to the weights during each iteration. However, they may sometimes imprecisely overshoot beyond their optimal values. Lower learning rates mean enhanced precision but a higher number of steps required to arrive at the optimal values. Commonly, the learning rate is set to 0.1 or 0.01. Alternatively, adaptive learning rates are used to arrive at the optimal learning rate automatically. The backpropagation iteration is repeated for sets of training examples (called batches). The number of iterations required depends on the size of the dataset and complexity of the network. Sometimes, the network might overfit to the training data where the function is fit closely to a limited number of observations and fails to generalize well for unseen data. To prevent overfitting, a technique known as early stoppage can be used where an arbitrary large number of epochs are defined during training. The process is stopped when there is no marginal improvement in performance on a hold-out cross validation dataset. Once the final parameter update for the final training example is done, we can assume that the network is sufficiently trained and ready to use on unseen data.

Convolutional Layer: A typical Convolutional Neural Network (CNN) architecture is characterized by groups of layers that are stacked together. Firstly, there are a limited number of input layers followed by combinations of convolutional layers, pooling layers, Fully Connected layers, and finally an output layer. The key difference between a CNN architecture and that of a multilayer perceptron is the introduction of convolutional and pooling layers that can be arranged in various ways. The convolutional operations determine the neurons that that are connected to certain regions in the input. Each operation computes a dot product of the weights and the small regions in the input space that they might be connected to. Pooling operations downsample the input vector along its width and height, resulting in a smaller output vectors. This operation reduces the overall number of parameters to be computed and therefore improves training speeds. A convolutional operation (CONV) is normally followed by a pooling operation (POOL) as seen in Figure 3.

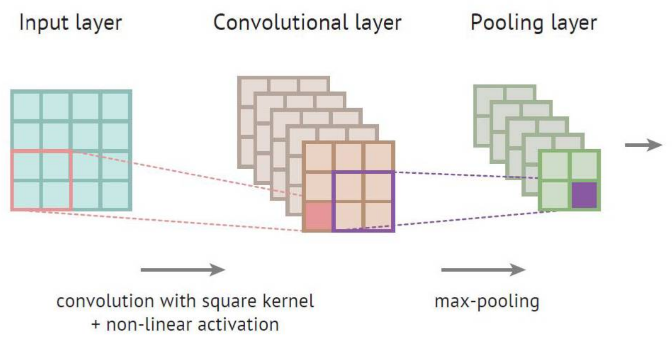


Figure 4: CONV and POOL operations [43]

The local structure for two dimensional input data (images) is rather unique to that of one dimensional input data (sound). Convolutional layers aim to take advantage of this structure by organizing hidden units in a special way. The convolutional operation processes only a small fraction of the entire input space (e.g. 3×3 pixel blocks) at once. The hidden unit is not connected to all the nodes of the previous layer. The weights associated with these hidden units represent a convolutional filter (kernel) which is used to scan the entire input space. After this filter is applied, a feature map is created. The same set of weights obtained can be used over and over again till the input space is completely scanned. By doing so, locally useful features (e.g. edges, shapes) can be detected in other regions within the input space. By looking for these specific features, most other insignificant features are ignored which results in lower costs of computation. This approach inherently makes the data robust to changes like translational shifts. A convolutional layer may have more than one filter. Even though sound could be considered a 1D signal, by creating spectrograms we can use the above 2D techniques traditionally developed for images.

To further reduce dimensionality and costs of computation, pooling layers are introduced which accumulate the information from adjacent cells in the feature map. Some of the common pooling operations are average-pooling (values in the cell are averaged) and max-pooling (highest value in the cell). Robustness to translations is further improved as a result of this downsampling.

Rectified Linear Units: In Neural Networks, the activation functions determine the output of nodes given the input values. They are analogous to logic gates in integrated circuits where the outputs can be 0 or 1 based on the logic and input values. Traditional activation functions like hyperbolic tangent and sigmoid were used for most of the Neural Network based architectures. Recent studies have replaced them with more viable solutions. One such activation function uses Rectified Linear Units (ReLUs) which is given as:

$$f(x) = max(0, x) \tag{23}$$

When compared to traditional activation functions, ReLUs are computationally faster and perform gradient propagation with better efficiency. Sigmoid units cause saturation in some cases. Also, their activation structure is much sparser than that of sigmoid units [44]. Even with a simple structure, the ReLUs still retain most of their discriminatory properties. These units however use random weight initialization which comes with a few drawbacks - some units, because of their initialization values always result in zero gradients. To address this issue, Leaky Rectified Units have been developed [45]. Leaky ReLUs have non zero slopes and effectively tackle the zero gradient problem [46].

Exponential Linear Units: Exponential Linear Units [47] or widely known as ELUs are the newer additions to the family of activation functions. They have a tendency to quickly converge the cost to zero and produce higher accuracy estimates. The key difference here is the inclusion of an extra constant α where $\alpha \geq 0$.

$$f(x) = \begin{cases} x & x > 0 \\ \alpha(e^x - 1) & x \leq 0 \end{cases} \tag{24}$$

The working of ELU is similar to that of ReLU with both serving as indentity functions for positive inputs. The key exception is that while ReLU smoothens sharply, ELU smoothens slowly as the output tends towards $-\alpha$. Unlike ReLU, ELU can produce negative results as well.

Dropout Learning: Neural Network architectures are capable of generalizing a wide variety of functions. Because of their flexibility, overfitting is sometimes a common problem where the algorithm might perform poorly on unseen examples. In the case of CNNs, this problem is slightly mitigated through weight sharing of the parameters. However, for any network, if the number of training examples are limited, the model may not generalize well for out-of-sample data.

To address the overfitting problem, a highly effective method known as dropout learning has been introduced [21]. Every hidden unit, in each iteration is assigned with a certain probability of removal (50% in the original paper). The rest of the learning procedure is performed normally. The random removal of nodes sometimes do not allow the network to learn complex dependencies and correlations between hidden units. In this way, clusters of hidden units that are strongly wired together tend to capture long range dependencies better. A network architecture that has been trimmed using dropout techniques produces better generalization capability by mitigating the effects of overfitting during the learning process.

Data Augmentation: Another factor that limits the generalization capacity of Neural Networks the the non availability of adequate training data. The publicly available datasets that are useful for sound classification have minuscule volumes when compared to those that are available for image classification research. For datasets of such limited nature, a solution known as data augmentation that has been successful for image recognition tasks can be revisited for audio classification as well.

This approach aims to expand the dataset by applying some deformations and still be classified according to the original labels of the training examples. For example, deformations like rotation, translation, scaling or mirroring would not change the semantic meaning of an image (e.g.a cat). Since all the deformed images carry the same label, the Neural Network attains the capability to see beyond such deformations for unseen data thus making the classifier more generalized. In the case of audio based data, deformations that were normally used are pitch shifting, noise addition, time stretching and time shifting [21], [48].

3 Experimental Setup

In this section, the experimental setup is discussed. For the successful application, I seek to evaluate the merits and demerits posed during each experiment iteration. In all cases, I aim to achieve a good classification accuracy. Since the inference is run on the Raspberry Pi, it is imperative to have a small sized model with a low memory footprint without compromising much on the accuracy estimates. A small sized model would also result in faster processing that allows for near-realtime inference on smaller engines. The experiments were conducted on a system with i5-7500 CPU @ 3.40 GHz, 16gb DDR4 RAM and NVIDIA GeForce GTX 1060 6GB GPU. Through our experimental setup, I seek to find answers the following questions:

1) What are the features of choice used when transforming the raw audio clip: MFCCs or GTCCs?
2) What are the advantages of Neural Networks over traditional machine learning models?
3) Can the Convolutional layers be used instead of the compute intensive Fully-Connected layers for audio data?
4) Is it possible to achieve a memory footprint less than 100MB without compromising on accuracy?
5) Given the limited compute capability of the Raspberry Pi, can I perform an accurate classification within a time window of 1 second?

Test Methods: In Section 3.1, I give a brief overview of the dataset that I will be working with. For our method of classification, the chimpanzee calls have to be supplemented with other annotated sound samples before training the classifier. By doing so, I attempt to make the classifier robust so that the chimpanzee calls are distinguishable from other sounds in the environment. Once the dataset is explored, I normalize it on various attributes like sampling rate, bit depth and number of channels to make the training process easier.

A baseline Convolutional Neural Network is described in Section 3.2. This iteration of the model is used as a benchmark to evaluate the performance associated with the choice of audio features, classifiers and network architectures. The model is initialized with two Convolutional layers and two

Fully Connected layers. Batch Normalization and Dropout are included to enhance model performance and stability.

Section 3.3 describes the process used to select the audio features used as inputs to the classifier. I evaluate the performance of Mel Frequency Cepstral Coefficents (MFCCs) and Gammatone Cepstral Coefficients (GTCCs), an emerging choice of features for audio classification tasks.

Support Vector Machines (SVMs), which were the most popular classifiers are revisited in Section 3.4. Here the performance of SVM and Neural Network based architectures is evaluated. The model parameters are fine-tuned using Grid Search to obtain the optimal set of values for the model. The performance of the best working model is evaluated with that of our baseline CNN.

In Section 3.5, I evaluate the merits and demerits of using Fully Connected Layers instead of Convolutional layers. Fully Connected layers look at all possible associations from the previous layer to determine the activation of a node whereas Convolutional layers look at only the adjacent nodes from the previous layer. With lesser chances of overfitting and comparable accuracy estimates, I evaluate if the usage of Convolutional Layers would be better suited to our task.

To fine-tune our model further after determining the architecture, I ran a Grid Search routine for the baseline model in Section 3.6. I iterate over various layer sizes, dropout probabilities and activation functions to determine the best performing set of hyperparameters for the model. Moreover, the number of MFCC channels were increased from 40 to 80 during feature extraction to improve granularity and therefore accuracy estimates.

This version of the model is evaluated in Section 3.7. It is then exported to the Raspberry Pi for the inference tasks.

3.1 Dataset Overview

The chimpanzee calls are combined with a publicly available dataset known as Urbansound8K [49]. This dataset consists of 8732 audio files which are each less than 4 seconds. These sounds have been captured from a variety of audio sources and labelled into 10 classes that are commonly encountered in urban environments - air conditioners, dogs barking, cars honking, children playing, drilling, idling of engines, gun shots, jackhammers, sirens and street music. These sounds are arranged into 10 folds for k-fold cross validation, a procedure used for re-sampling the data and evaluating model performance. Even though these sounds are not normally heard in a forest environment, adding these makes for a robust classifier that can detect chimpanzee calls even in the presence of other sounds. Moreover, this dataset has been used to evaluate a lot of audio classification approaches and thus gives us baseline performance metrics for comparison. One such work by Salamon and Bello [49] introduces a network that uses mel-spectrograms as features which are further reduced in dimensionality using Principal Component Analysis. The accuracy for the baseline network used was 68% and the best version that was developed gave an accuracy of 73%. By augmenting the chimpanzee calls collected on the research site with this dataset, I created a multiclass classification problem. This approach has an inherent tendency to reduce the amount of false positives that may be detected instead of the chimpanzee calls.

For the input data, in Table 1 we can see that the sample rates are not consistent and need to be normalized to a value which can still allow us to perform sound classification tasks. The chimpanzee calls recorded have a sampling rate of 11025Hz. Therefore, I have chosen a value of 11025 to which all the sounds can be downsampled. For 12 samples that were recorded at 8kHz, I have excluded them from the training set since downsampling all the sounds to 8kHz instead of 11kHz would cause a significant loss of information. Moreover since the number of excluded samples are relatively small, the resulting dataset is still be relatively balanced.

Sampling Rate	Value Counts
44100	5370
48000	2502
96000	610
24000	82
16000	45
22050	44
11025	39
192000	17
8000	12
11024	7
32000	4

Table 1: Sampling rates of sounds (Urbansound8K)

Since the chimpanzee audio files are recorded in mono, it is necessary to change the number of channels to 1 for all the files before training. In Table 2, we see that the audio clips in the Urbansound8K dataset have a significant amount of audio clips which are recorded in stereo. Audio signals recorded in stereo have two dimensions representing the channels. Data in the two channels are averaged together to get a mono signal.

Channels	Value Counts
2	7993
1	739

Table 2: Number of Channels (Urbansound8K)

The chimpanzee audio files have a bit depth of 16. In Table 3, we observe that the audio clips in the Urbansound8K dataset have varying bit depths. All the files are normalized to 16 bit to avoid any loss of information and consistency. I excluded the 52 samples with less than 16 bit depth as a part of the training set. The folder structure for the original dataset was re-organized so that the sound files are placed in their respective class folders. This makes it convenient to add or remove classes from this dataset.

Bit Depth	Value Counts
16	5758
24	2753
32	169
8	43
4	9

Table 3: Bit depth of the sounds (Urbansound8K)

The amplitudes of these sounds are broadly determined by the distance between the source and the receiver. To normalize this effect, I normalized the mean amplitudes each sound to a value equal to the total mean amplitude of all the sounds combined. Each sound is applied an amplitude gain value based on the difference of its mean to the total mean value. In this way, we can mitigate some of the bias that may be induced because of recording distance.

For some of the sounds which have durations of less than 4 seconds, I recombined them into a single file and split them into 4 second chunks. This ensures that the all the sounds have a length of 4 seconds, sampling rate of 11025Hz, single channel and a bit depth of 16.

Once the dataset is sufficiently normalized, I employed data augmentation methods like time shifting, noise addition, pitch shifting and amplitude changes with random probabilities to each sound sample. Their original label is preserved after applying the deformations. By doing so, the dataset is expanded to multiple times its original volume. After augmentation, I obtained a dataset consisting of 57246 training examples with about 4700 samples per class. The classifier thus built is resilient to overfitting and invariant of such deformations if they occur naturally.

3.2 Baseline Network

I conducted our experiments with a baseline network comprising of a CONV-POOL-CONV-POOL based architecture followed by two Fully Connected layers to evaluate the abstracted outputs. The network architecture can be generated using the model summary feature in Keras as shown in

Figure 5.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 40, 87, 16)        80
_________________________________________________________________
batch_normalization_1 (Batch (None, 40, 87, 16)        64
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 20, 43, 16)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 20, 43, 16)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 20, 43, 32)        2080
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 10, 21, 32)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 10, 21, 32)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 6720)              0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               860288
_________________________________________________________________
dropout_3 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 12)                1548
=================================================================
Total params: 864,060
Trainable params: 864,028
Non-trainable params: 32
_________________________________________________________________
```

Figure 5: Baseline CNN architecture

The Convolutional Layers ($conv2d\_1, conv2d\_2$) consist of filters that abstract out shapes that generally resemble their representations. The input layer's dimensions are accordingly adjusted as a 2-D feature vector after the transformations have been applied to the raw audio signals. Pixel groups that occupy an area equal to the square of the kernel size are transformed into one pixel when passed through a filter.

As we go through to the deeper layers, the output of the first Convolutional layer becomes an input to the second convolutional layer after any pooling operations . At this point, each input layer is a positional representation of where the lower level features(e.g. lines, curves) are detected. A similar filter mechanism when applied on the lower level features enables us to detect higher level features(e.g. edges at a angle, combination of edges) across a larger receptive field. By the end of the network, we obtain activation maps that represent the sum of the higher level sound patterns created by various sources.

Max-Pool layers ($max\_pooling2d\_1, max\_pooling2d\_2$) are usually followed by the Dropout layers ($dropout\_2, dropout\_3$). I used dropout to reduce the disproportionate dependency on smaller sets of features. If they are placed before the Max-Pool layers, this phenomenon is not upheld. The Max-Pool layers pick up the maximum value from their respective set of input values. The dropout layer thus will be of its optimal utility if it randomly picks up the maximum value from the set of input values. Therefore, the small feature dependencies are only removed when the maximum value has a dropout probability higher than the predefined value. The activation functions are followed by a layer of Batch Normalization ($batch\_normalization\_1$). By adjusting and scaling the activations, Batch Normalization improves training times, accuracy estimates and stability of the model.

Fully Connected Layers ($dense\_1, dense\_2$) aggregate the flattened input volume ($flatten\_1$) of the preceding layers and output an n dimensional vector where n is the number of classes in the original dataset. Each number in this output vector represents the probability associated with each class. The Fully Connected layers here determine which higher level features correlate to a certain class.

Before the training process takes place, the weights are randomly initialized. Each training iteration (i.e. a forward pass, computation of the loss function, backward pass and updating parameters) tunes the weights and biases of each layer to progressively build the classifier.

3.3 Feature Performance: MFCCs Vs GTCCs

Using the baseline network developed, I have evaluated the performance of Mel Frequency Cepstral Coefficients and Gammatone Cepstral Coefficients. The input nodes were changed according to the sizes of the feature vectors and were trained for 50 epochs with a batch size of 20 (the number of samples processed before the model is updated). The resulting accuracy and loss plots are then analyzed to determine the appropriate audio features of choice.

3.4 Traditional methods vs Neural Networks

In this experiment, I have trained an SVM model using the sklearn package and conducted hyperparameter optimization of the kernel function, γ, and c parameters using grid search on 70% of the training set. The following optimization parameters given in Table 4 have been used for evaluation:

Hyperparameter	range	selection
γ	1e-3, 1e-4, 1e-5, 1e-6	1e-5
c	1,10,20,30,40,50	20

Table 4: Hyperparameter selection for the SVM

I used small values for γ in powers of 10 to see the effect of γ at different magnitudes. We use small values for c to avoid precision issues from occurring when using small γ and large c values that can lead to the γ parameter being overpowered by the magnitude of c.

3.5 Fully Connected Layers vs Convolutional layers

Even though the usage of convolutional layers in the network architecture is domain dependant and was used primarily for image classification tasks, we have seen reasonable performance with respect to classification power and prediction times for audio based inputs. Alternatively, an architecture comprising of only Fully Connected (FC) layers can be trained with the expectation of better accuray estimates. FC layers are used to detect specific global configurations of the features detected by the lower layers in the net. They usually sit at the top of the network hierarchy, at a point when the input has been reduced (by the previous, usually convolutional layers) to a compact representation of features. Each node in the FC layer learns its own set of weights of all of the nodes in the layer below it. In Figure 6, we obtain the architecture and layer sizes of the Fully Connected network using the model summary feature in Keras.

3.6 Hyperparameter Tuning

To fine-tune the model, I attempt to improve upon the architecture design by running a Grid Search routine to evaluate the best set of hyperparameters. There are various hyperparameters that can be optimized like learning rate, layer size, dropout rate, activation functions and in some cases, even number of layers. I used Talos, an open source python library for evaluating the multitude of hyperparameter permutations. Talos incorporates grid, random, and probabilistic hyperparameter optimization strategies, with focus on maximizing the flexibility and efficiency as a result of random strategy. The

```
________________________________________________________________
Layer (type)                 Output Shape              Param #
================================================================
dense_13 (Dense)             (None, 256)               10496
________________________________________________________________
activation_13 (Activation)   (None, 256)               0
________________________________________________________________
dropout_10 (Dropout)         (None, 256)               0
________________________________________________________________
dense_14 (Dense)             (None, 512)               131584
________________________________________________________________
activation_14 (Activation)   (None, 512)               0
________________________________________________________________
dropout_11 (Dropout)         (None, 512)               0
________________________________________________________________
dense_15 (Dense)             (None, 256)               131328
________________________________________________________________
activation_15 (Activation)   (None, 256)               0
________________________________________________________________
dropout_12 (Dropout)         (None, 256)               0
________________________________________________________________
dense_16 (Dense)             (None, 12)                3084
________________________________________________________________
activation_16 (Activation)   (None, 12)                0
================================================================
Total params: 276,492
Trainable params: 276,492
Non-trainable params: 0
________________________________________________________________
```

Figure 6: Baseline FCN architecture

following hyperparameter configurations in Table 5 were evaluated and 80 MFCC channels were used for audio feature extraction.

Hyperparameter	range
Dense Layer size	64, 128, 256
Activation Fn.	elu, relu
Dropout	0.3, 0.4, 0.5 , 0.6

Table 5: Hyperparameter selection for the CNN

4 Results

This section elaborates on the results obtained from the various configurations in the Experimental Setup. Based on these results a final model is developed which is best suited for our current application objectives.

4.1 Experimental Setup Results

4.1.1 Feature Performance:

During Feature extraction, two choices of features were used. Results from the experiments showed that GTCCs do not perform better than MFCCs with respect to classification accuracy on the test set.

The model trained using GTCCs takes about seven times more space than the model trained using MFCCs. Also, the processing time for a single sample is much lower when using MFCCs. Even if the number of channels are increased in the favor of classification power, there would be a proportionate increase in the number of trainable parameters of the model which would result in bigger model sizes and slower processing speeds as seen in Table 6. Figure 7 and Figure 8 represent the accuracy plots while using MFCCs and GTCCs respectively. It can be observed that there is a significant improvement in accuracy when MFCCs are used. We can therefore conclude that further analysis using GTCCs would not be fruitful for the final inference task.

Feature Performance				
Features	channels	accuracy	processing time(s)	memory footprint(mb)
MFCCs	40	0.8424	0.168	10.175
GTCCs	64	0.6532	0.735	76.127

Table 6: Performance of MFCCs and GTCCs

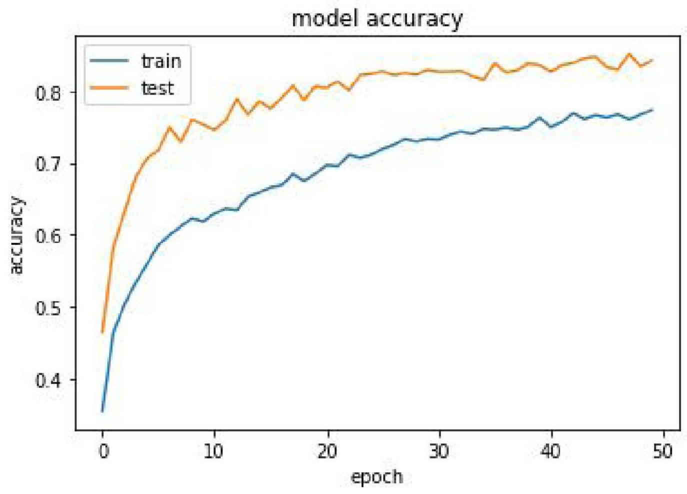


Figure 7: Accuracy of Baseline model using MFCCs as audio features

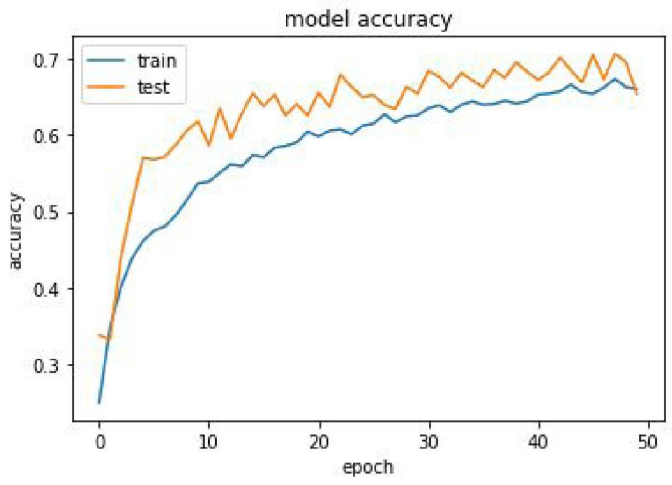


Figure 8: Accuracy of Baseline model using GTCCs as audio features

In Figure 7 and 8, we observe that the accuracy is higher on the test data than on the training data. This is a commonly seen phenomenon when using dropout. Dropout is a destructive process that limits the amount of information that is passed on to the subsequent layers in the network. This causes a higher amount of loss for the training set because it is harder for the network to predict the right answers. However, for the test set, all the neurons in the network are available which may increase the classifier performance.

4.1.2 SVM vs Neural Networks:

For the SVM classifier, a final accuracy of 0.830 was obtained on our dataset using MFCCs as input features. Since this value is slightly lower than what we had obtained when the Baseline Neural Network model was used (0.8434), I discarded this approach in favor of Neural Networks for improving classification power. Moreover, SVM's require fewer hyperparameters and are less flexible than Neural Networks.

4.1.3 Fully Connected vs Convolutional layers:

Preliminary results as seen in Figure 9 and Figure 10 show that classification power is slightly higher in the case of the CNN architecture(0.8709 vs 0.8644 after 100 epochs) when compared to a Fully Connected Network. When training, a percentage of the features are set to zero (50% in our case since we are using Dropout(0.5)). When testing, all features are used (and are scaled appropriately). So the model at test time is more robust - and can lead to higher testing accuracies.

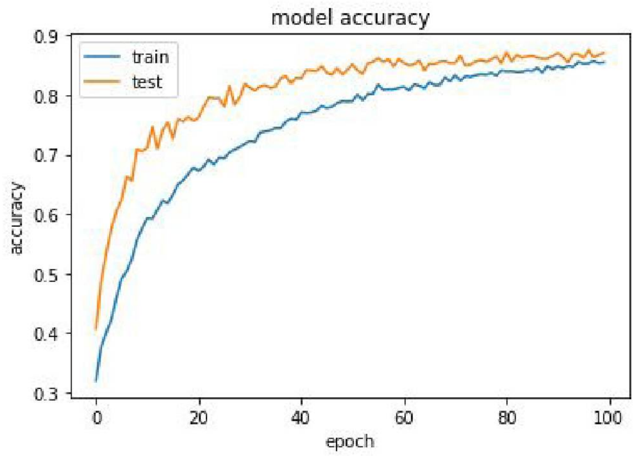


Figure 9: Accuracy of model with CONV layers and dropout trained over 100 epochs

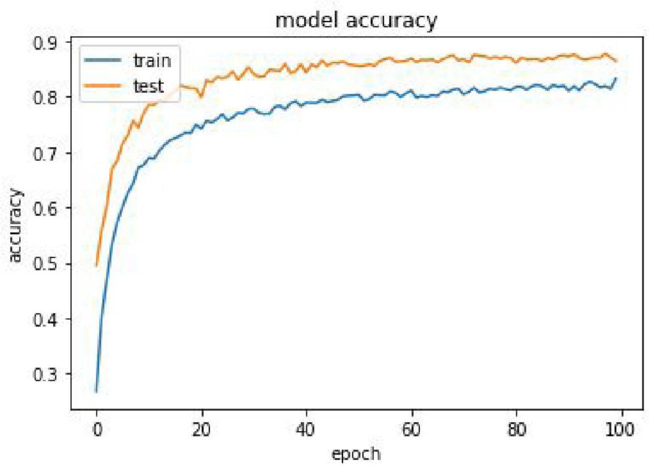


Figure 10: Accuracy of model with Fully Connected layers and dropout trained over 100 epochs

In Figure 11 and Figure 12, we observe that even though the accuracy estimates for both the architectures are comparable, the loss function for the Fully Connected network flattens much more quickly and stabilizes around a value of 0.8. On the other hand, the loss function for a CNN based architecture flattens at a value of 0.5. We can infer that the accuracy estimates can be improved for the CNN by fine-tuning the hyperparameters and training the model further.

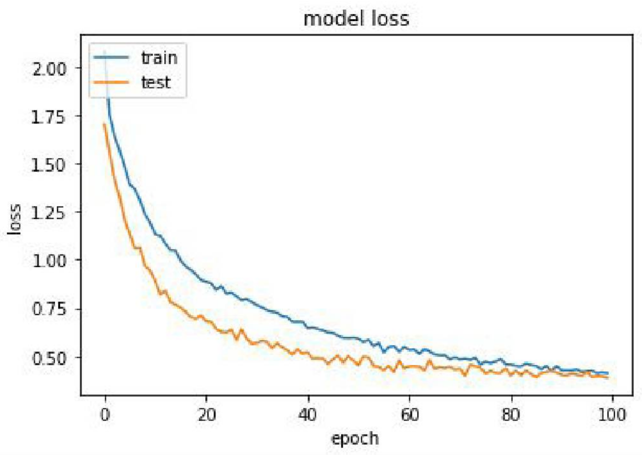


Figure 11: Loss plot of model with CONV layers and dropout trained over 100 epochs

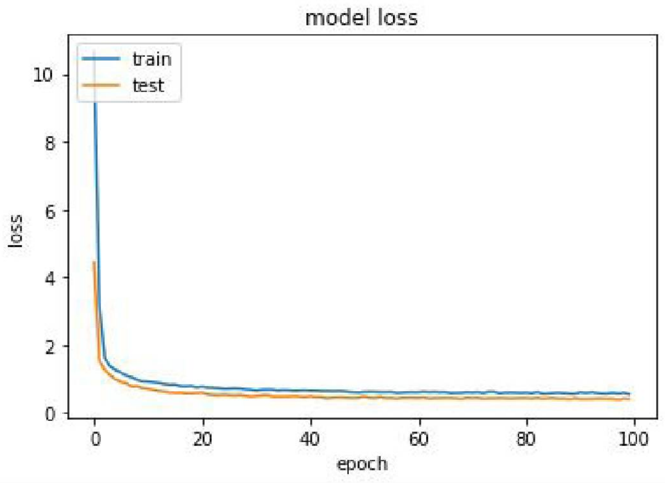
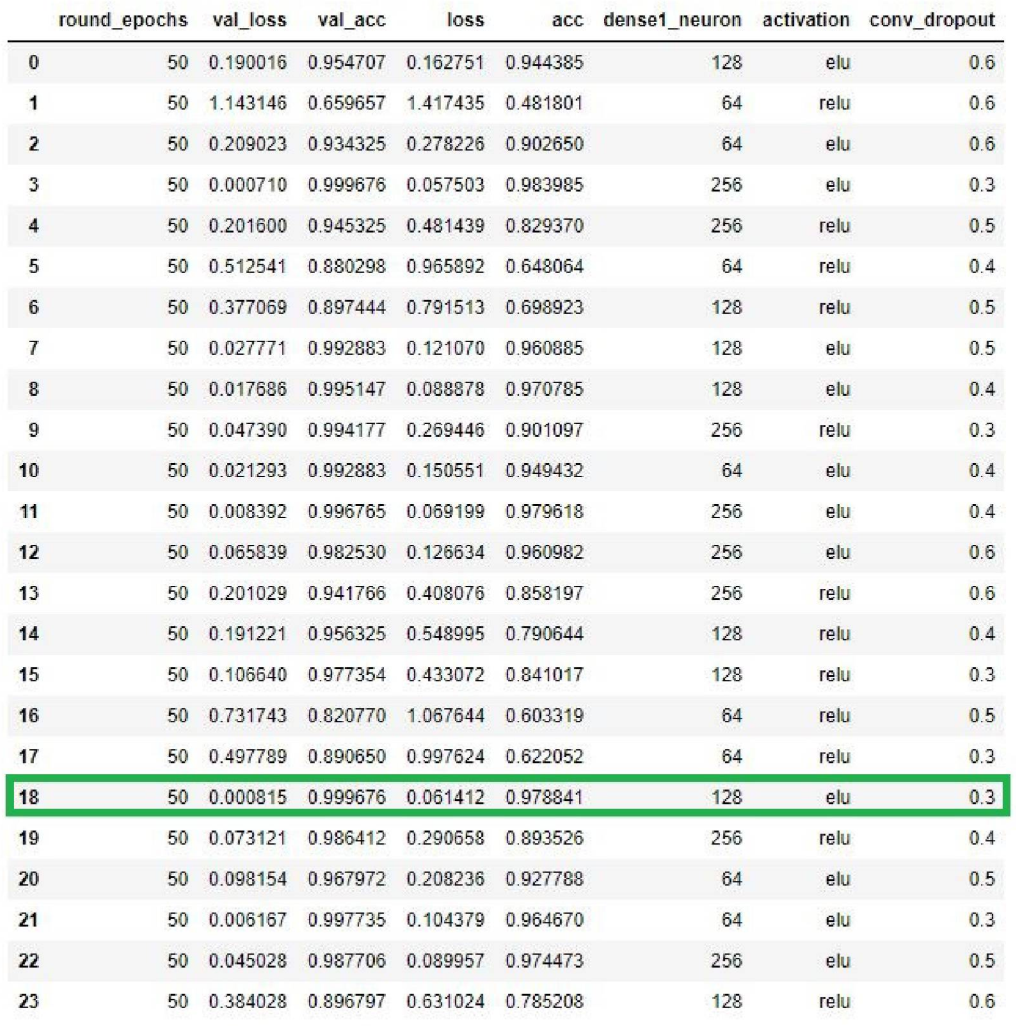


Figure 12: Loss plot of model with Fully Connected layers and dropout trained over 100 epochs

4.2 Final Model

Based on the results from the experiments, we can conclude that Neural Network architectures with convolutional layers work best for our problem statement. After running the Grid Search routine, the Talos framework output for the accuracy statistics can be seen in Figure 13.



	round_epochs	val_loss	val_acc	loss	acc	dense1_neuron	activation	conv_dropout
0	50	0.190016	0.954707	0.162751	0.944385	128	elu	0.6
1	50	1.143146	0.659657	1.417435	0.481801	64	relu	0.6
2	50	0.209023	0.934325	0.278226	0.902650	64	elu	0.6
3	50	0.000710	0.999676	0.057503	0.983985	256	elu	0.3
4	50	0.201600	0.945325	0.481439	0.829370	256	relu	0.5
5	50	0.512541	0.880298	0.965892	0.648064	64	relu	0.4
6	50	0.377069	0.897444	0.791513	0.698923	128	relu	0.5
7	50	0.027771	0.992883	0.121070	0.960885	128	elu	0.5
8	50	0.017686	0.995147	0.088878	0.970785	128	elu	0.4
9	50	0.047390	0.994177	0.269446	0.901097	256	relu	0.3
10	50	0.021293	0.992883	0.150551	0.949432	64	elu	0.4
11	50	0.008392	0.996765	0.069199	0.979618	256	elu	0.4
12	50	0.065839	0.982530	0.126634	0.960982	256	elu	0.6
13	50	0.201029	0.941766	0.408076	0.858197	256	relu	0.6
14	50	0.191221	0.956325	0.548995	0.790644	128	relu	0.4
15	50	0.106640	0.977354	0.433072	0.841017	128	relu	0.3
16	50	0.731743	0.820770	1.067644	0.603319	64	relu	0.5
17	50	0.497789	0.890650	0.997624	0.622052	64	relu	0.3
18	50	0.000815	0.999676	0.061412	0.978841	128	elu	0.3
19	50	0.073121	0.986412	0.290658	0.893526	256	relu	0.4
20	50	0.098154	0.967972	0.208236	0.927788	64	elu	0.5
21	50	0.006167	0.997735	0.104379	0.964670	64	elu	0.3
22	50	0.045028	0.987706	0.089957	0.974473	256	elu	0.5
23	50	0.384028	0.896797	0.631024	0.785208	128	relu	0.6

Figure 13: Best Set of Hyperparameters

Based on the results obtained in Figure 13, the configurations used in entries 3 and 18 gave the highest accuracy on the validation set(*val_acc*). I used 128 hidden nodes on the penultimate layer, a dropout rate of 0.3 and the elu activation function as used in entry 18. This is preferred over the configuration in entry 3 because of the smaller number of neurons in the dense layer which speeds up the training process and does not compromise on classification power. Using Keras' model summary feature, the architecture of the Final CNN model can be seen as given in Figure 15. An intuitive way of visualizing NN-architecture schematics [50] can be seen in Figure 14.

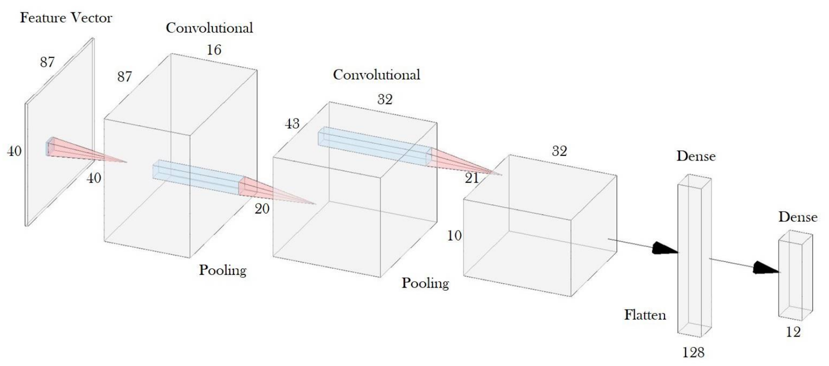


Figure 14: Architecture Diagram

A single input audio file has a sampling rate 11025 Hz, 16 bit depth, mono channel and length of 4 seconds. When MFCCs are extracted, I used a Hann window of hop length 512. For consistency, the windowed signals are padded with zeros and restricted to a length of n_fft. The value of n_fft is preferably a power of 2 for speeding up the FFT process. The number of rows in the Fourier transform matrix can be given as $(1 + n\_fft/2)$. In our case, I choose a value of 2048 for n_fft which corresponds to a duration of 186 milliseconds for an 11025 Hz sampling rate audio input. When 40 MFCC channels are used, I transformed the 4 second audio clip into a 2-dimensional vector of size (40×87). Each of these vectors is converted into an image-like representation which can be processed using CNNs. It is possible do this by reshaping them to add a depth of 1. This is analogous to a single channel (40×87) image. For the first convolutional layer, I used 16 filters to obtain 16 feature maps. Every pixel in each of the feature maps is an output of the convolutional layer. This convolutional layer extracts patterns like simple lines. I doubled the number of filters in the next convolutional layer ($conv2d\_4$) so

```
________________________________________________________________
Layer (type)                 Output Shape              Param #
================================================================
conv2d_3 (Conv2D)            (None, 40, 87, 16)        80
________________________________________________________________
batch_normalization_2 (Batch (None, 40, 87, 16)        64
________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 20, 43, 16)        0
________________________________________________________________
dropout_4 (Dropout)          (None, 20, 43, 16)        0
________________________________________________________________
conv2d_4 (Conv2D)            (None, 20, 43, 32)        2080
________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 10, 21, 32)        0
________________________________________________________________
dropout_5 (Dropout)          (None, 10, 21, 32)        0
________________________________________________________________
flatten_2 (Flatten)          (None, 6720)             0
________________________________________________________________
dense_3 (Dense)              (None, 128)               860288
________________________________________________________________
dropout_6 (Dropout)          (None, 128)               0
________________________________________________________________
dense_4 (Dense)              (None, 12)                1548
================================================================
Total params: 864,060
Trainable params: 864,028
Non-trainable params: 32
________________________________________________________________
```

Figure 15: Final CNN architecture

that structures with higher complexity can be extracted. The outputs from the convolutional layers, followed by the pooling layer (*max_pooling2d_3*) represent space invariant, low level features extracted from the data. While these outputs can directly be connected to the final Fully Connected layer (*dense_4*), another Fully Connected layer (*dense_3*) can be added. This extracts non-linear relationships between higher level features. A dense layer size of 128 gave the best accuracy estimates for the model architecture. The final layer gives the probabilities associated with each of the 12 classes in our dataset.

Upon training the CNN model over the augmented dataset, with a batch size of 20 and 100 epochs, we obtain the following accuracy and loss plots as seen in Figure 16 and Figure 17. Since we are using the augmented dataset instead of the original dataset, we may observe a drop in overall accuracy but the classifier thus obtained will be invariant of deformations and is less likely to overfit:

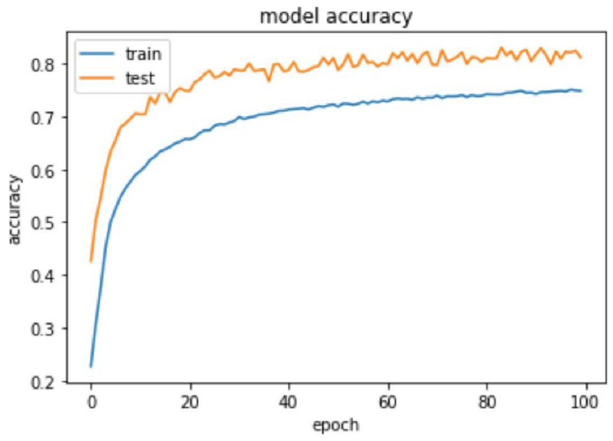


Figure 16: Model Accuracy

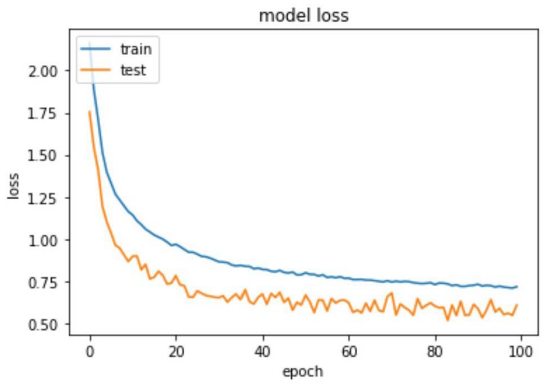


Figure 17: Model Loss

While accuracy is a very intuitive measure of performance, it alone is generally not enough to evaluate model performance effectively. Accuracy is simply the ratio of observations that are predicted correctly to the total number of observations. This metric works best when the dataset is mostly symmetric and there are equal numbers of false positives and false negatives. Optimizing purely for accuracy makes the model much more prone to detecting falsely labelled examples as positive. Intuitively, optimizing for a metric that mitigates the number of false positives is not ideal either as the number of true positives might be diminished as a result. To account for the model accuracy (correct predictions) and robustness (reducing the number of misses), we use a metric called the F1 score. The F1 score is the harmonic mean of two other metrics known as precision and recall.

Precision: Precision is the ratio of correctly labelled positive results to the total number of positive results predicted. This metric basically gives the number of predicted chimpanzee calls which were actually chimpanzee calls. This metric correlates to low false positive rates. Precision can be given as:

$$Precision = \frac{TruePositives}{TruePositives + FalsePositives} \tag{25}$$

Recall: Recall is the ratio of positive observations predicted to the total number of relevant samples (observations that should have been identified as positive). This metric tells the number of chimpanzee calls predicted correctly out out the total number of chimpanzee calls in the dataset. Recall can be given as:

$$Recall = \frac{TruePositives}{TruePositives + FalseNegatives} \tag{26}$$

To aggregate precision and recall, the harmonic mean is a better estimate than the arithmetic mean because the harmonic mean negates the effect of extreme values. The F1 score is given as:

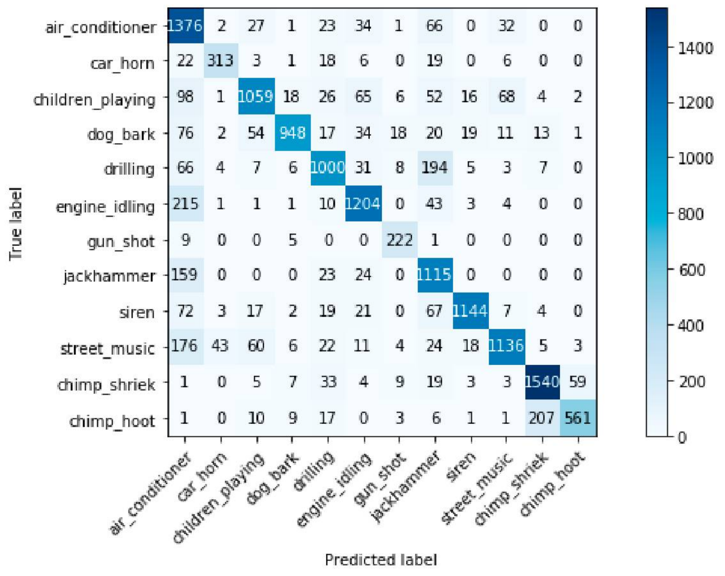
$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{27}$$

In Listing 1, the $10th$ and the $11th$ classes represent chimpanzee 'shrieks' and 'hoots' respectively. Since the two sounds have starkly different frequency ranges, it is imperative to separate them into two classes for model performance. The Classification Report created using sklearn metrics is given in Listing 1. Also, an intuitive way of visualizing the model accuracy is by using a Confusion Matrix. The Confusion matrix is a table layout where rows represent observations in the predicted classes while columns represent observations in the actual classes. The Confusion Matrix output and the Normalized Confusion Matrix for the classifier is shown in Figure 18 and Figure 19 respectively. Both of these representations have been generated using the confusion matrix feature of scikit-learn.

```
              precision    recall  f1-score   support

           0       0.61      0.88      0.72      1562
           1       0.85      0.81      0.83       388
           2       0.85      0.75      0.80      1415
           3       0.94      0.78      0.86      1213
           4       0.83      0.75      0.79      1331
           5       0.84      0.81      0.83      1482
           6       0.82      0.94      0.87       237
           7       0.69      0.84      0.76      1321
           8       0.95      0.84      0.89      1356
           9       0.89      0.75      0.82      1508
          10       0.87      0.92      0.89      1683
          11       0.90      0.69      0.78       816

    accuracy                           0.81     14312
   macro avg       0.84      0.81      0.82     14312
weighted avg       0.83      0.81      0.81     14312
```
Listing 1: Classification Report



	air_conditioner	car_horn	children_playing	dog_bark	drilling	engine_idling	gun_shot	jackhammer	siren	street_music	chimp_shriek	chimp_hoot
air_conditioner	1376	2	27	1	23	34	1	66	0	32	0	0
car_horn	22	313	3	1	18	6	0	19	0	6	0	0
children_playing	98	1	1059	18	26	65	6	52	16	68	4	2
dog_bark	76	2	54	948	17	34	18	20	19	11	13	1
drilling	66	4	7	6	1000	31	8	194	5	3	7	0
engine_idling	215	1	1	1	10	1204	0	43	3	4	0	0
gun_shot	9	0	0	5	0	0	222	1	0	0	0	0
jackhammer	159	0	0	0	23	24	0	1115	0	0	0	0
siren	72	3	17	2	19	21	0	67	1144	7	4	0
street_music	176	43	60	6	22	11	4	24	18	1136	5	3
chimp_shriek	1	0	5	7	33	4	9	19	3	3	1540	59
chimp_hoot	1	0	10	9	17	0	3	6	1	1	207	561

Figure 18: Confusion Matrix

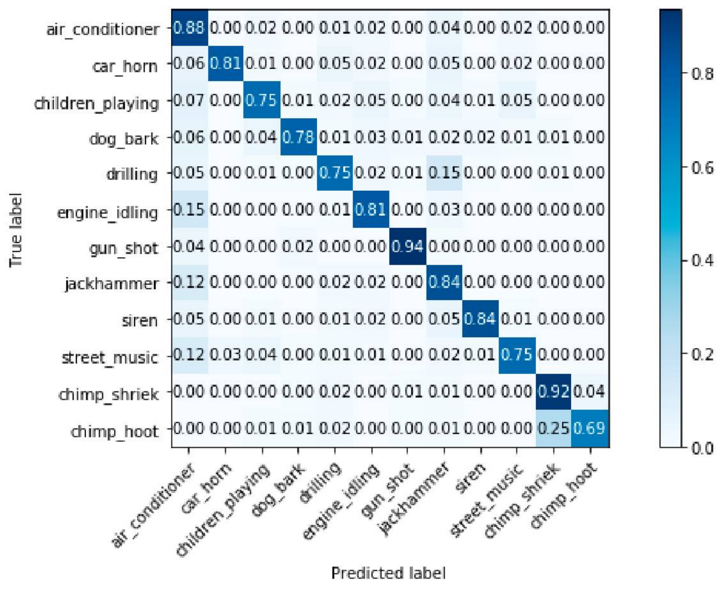


Figure 19: Normalized Confusion Matrix

We observe that for both of the chimpanzee shrieks and hoots, fairly good F1 scores of 0.89 and 0.78 are obtained. When all the classes are taken into consideration, we get a 12×12 confusion matrix with the Y axis showing the true labels in the test set and the X axis showing the class assigned to an observation associated with that true label. Each element (i, j) in the matrices represent the number/proportion of items with an actual class i on the X axis that were categorized into class j on the Y axis. We obtain a relatively good proportion (0.92) of chimpanzee shrieks labelled to their actual class. Even though this proportion is comparably lower for chimpanzee hoots (0.69), it is to be noted that a significant proportion of hoots are classified as shrieks (0.25). This result is expected since the chimpanzee hoots are acoustically similar to chimpanzee shrieks than any of the other classes in the dataset.

5 ARU: Realtime Detection on Raspberry Pi

The Raspberry Pi 3 Model B+ is a product in the Raspberry Pi 3 family. It hosts a Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC @ 1.4GHz CPU and 1GB LPDDR2 SDRAM. It can be solar powered and attached with a desirable microphone to capture audio inputs.

5.1 Audio Preprocessing and Inference

As of writing, the latest 1.9 release of TensorFlow can be installed on the Raspberry Pi 3 from pre-built binaries using Pythons pip package system. The inference engine uses the Librosa package in Python3 for audio feature extraction.

Since our Neural Network is designed to take 4 seconds of audio as input for feature extraction, we need to chunk the audio input stream into 4-second blocks before performing the inference. The input audio signal is downsampled to 11kHz with a bit depth of 16 and a single channel. To detect chimpanzee calls every second, each 1-second chunk of the audio stream is appended to the last 3 seconds of the the audio stream input as shown in Figure 20. By doing so, the Neural Network has the requisite input vector size and audio input is scanned every second. If a chimpanzee call is detected above a certain probability threshold, the respective audio clip is saved onto the local memory of the Raspberry Pi. These clips can be periodically extracted for further analysis.

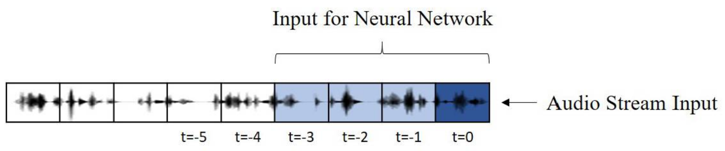


Figure 20: Inference Window

5.2 Performance on a real-time application

When inference is performed in realtime, I obtained the predictions every second with a processing time of 0.91 seconds on an average using the

Raspberry Pi. Most of the chimpanzee calls were accurately detected with their timestamps except for some instances which qualitatively resembled dog barks more than chimpanzee calls. The above approach can also be used to scan a folder for audio files and identify chimpanzee vocalizations in the clips. Additionally, the timestamps and the class prediction probabilities can be logged into a .txt file for identifying the file names and the four second time windows in which chimpanzee calls were detected.

6 Conclusion

In this study, we have seen the applicability of Neural Network architectures that can be deployed and used for inference on devices with limited computational capability. With detection possible in realtime, this approach is a significant improvement over the previous ASR systems.

Using an audio repository consisting of chimpanzee recordings, I initially classified them into two different kinds of vocalizations - 'shrieks' and 'hoots'. I then used Urbansound8K, a publicly available dataset with 10 labelled classes and merged it with the labelled chimpanzee audio recordings. The new dataset was then cleaned and normalized for integration flexibility. In order to determine the best classifier given the limited compute capability, I evaluated simple sound classification approaches using Support Vector Machines, Fully Connected Neural Networks and Convolutional Neural Networks. Even though the training and inference times are a fraction of what Neural Network architectures would take, SVMs gave an accuracy of 0.830 which was deemed as unsatisfactory. On the other hand, the usage of Fully Connected layers gave comparable accuracy estimates but had significantly large compute requirements. Based on our observations, Convolutional Neural Networks brought together good accuracy estimates, low memory footprints and fast prediction times with the right choice of audio features. It is also concluded that MFCCs perform better, have faster processing speeds and lower memory requirements for our classification task.

The dataset was then subjected to various data augmentation approaches to make the classifier robust and invariant to certain deformations. I obtained a dataset consisting of 57246 training examples with about 4700 samples per class. The final classifier gave an accuracy of 81.18% on the validation set and F1 scores of 0.89 and 0.78 for the two chimpanzee classes while requiring just 10MB of storage. The classifier is robust and identifies chimpanzee vocalizations in a variety of environmental conditions.

Previously, audio data was recorded over a range of hours and relayed back to a central processing unit where it is processed for further analysis. With the new approach, audio data of interest can be stored onto the onboard memory while rejecting the rest. This allows for recordings at a higher bitrate, which can subsequently improve the quality of training data for future classifiers. Moreover, as and when a chimapnzee call is recorded, researchers can react quickly thus opening up a host of possibilities for primatology study.

7 Future Work

Future work based on this study can be categorized into classifier improvement and localization. While the classifier can be used to detect chimpanzee vocalizations within locally stored audio files or natural environments, it is based on a relatively low quality of input data (sampling rate of 11025 Hz). This can be improved in the future using recordings with high sampling and bit rates. With the ever increasing compute capacity of small, low cost computers, cutting edge classification approaches that use Recurrent Neural Networks (RNNs) can also be used. An approach using RNNs would simply require sound samples of interest and ambient background sounds for their effectiveness.

It is possible to localize the chimpanzee vocalizations by exporting the project's image file onto multiple Raspberry Pi devices. When 3 or more devices are used and their spatial coordinates are known, we can determine the source location. Since the speed of sound in air is 343 meters per second at standard temperature and pressure, the timestamp at which the sounds are detected can give us an approximate distance between the source and the receiver after taking latency and processing time into consideration. Using radio modules like LoRa and LoRaWAN for the Raspberry Pi, the detection triggers can be transmitted to a centralized computing engine where it is possible to triangulate the source of the sound.

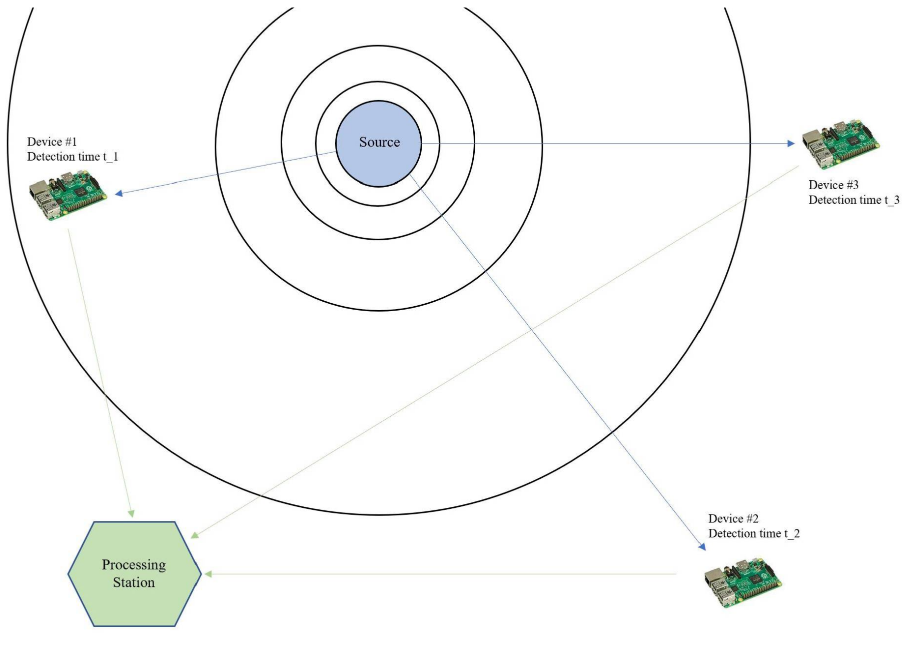


Figure 21: Localization

Effective localization of chimpanzees based on their sounds allows for tracking their movements at a distance without any interference. Some aspects of chimpanzee behavior like population densities, grouping patterns and broader home ranges can be determined with further analysis of the localized sounds. While this project lays out an outline for detecting chimpanzee calls using simple Neural Network architectures, it can also serve as a framework for identifying other sounds in nature. The classification module can be redesigned to detect other fauna and possibly even poachers.

References

[1] GMERC. *Greater Mahale Ecosystem Research and Conservation Website*. URL: http://gmerc.org/.

[2] Gordon Luikart, Nils Ryman, David A. Tallmon, et al. "Estimation of census and effective population sizes: The increasing usefulness of DNA-based approaches". In: *Conservation Genetics* 11.2 (2010), pp. 355–373. ISSN: 15660621. DOI: 10.1007/s10592-010-0050-7.

[3] Maureen S. McCarthy, Jack D. Lester, Eric J. Howe, et al. "Genetic censusing identifies an unexpectedly sizeable population of an endangered large mammal in a fragmented forest landscape". In: *BMC Ecology* 15.1 (2015), pp. 1–15. ISSN: 14726785. DOI: 10.1186/s12898-015-0052-x.

[4] Ammie K. Kalan, Alex K. Piel, Roger Mundry, et al. "Passive acoustic monitoring reveals group ranging and territory use: A case study of wild chimpanzees (Pan troglodytes)". In: *Frontiers in Zoology* 13.1 (Dec. 2016), p. 34. ISSN: 17429994. DOI: 10.1186/s12983-016-0167-8. URL: http://frontiersinzoology.biomedcentral.com/articles/10.1186/s12983-016-0167-8.

[5] Noémie Bonnin, Alexander Van Andel, Jeffrey Kerby, et al. "Assessment of Chimpanzee Nest Detectability in Drone-Acquired Images". In: *Drones* 2.2 (2018), p. 17. DOI: 10.3390/drones2020017.

[6] Taylor Francis Online. *Bioacoustics journal*. URL: http://www.bioacoustics.info/.

[7] Herman Medwin, Clarence S. Clay, and Timothy K. Stanton. *Fundamentals of Acoustical Oceanography*. Vol. 105. 4. 1999, pp. 2065–2066. ISBN: 012487570X. DOI: 10.1121/1.426950.

[8] Alex K. Piel. "Temporal patterns of chimpanzee loud calls in the Issa Valley, Tanzania: Evidence of nocturnal acoustic behavior in wild chimpanzees". In: *American Journal of Physical Anthropology* 166.3 (2018), pp. 530–540. ISSN: 10968644. DOI: 10.1002/ajpa.23609.

[9] Stefanie Heinicke, Ammie K. Kalan, Oliver J. J. Wagner, et al. "Assessing the performance of a semi-automated acoustic monitoring system for primates". In: *Methods in Ecology and Evolution* 6.7 (2015), pp. 753–763. ISSN: 2041210X. DOI: 10.1111/2041-210X.12384.

[10] Daniel T. Blumstein, Daniel J. Mennill, Patrick Clemins, et al. "Acoustic monitoring in terrestrial environments using microphone arrays: Applications, technological considerations and prospectus". In: *Journal of Applied Ecology* 48.3 (2011), pp. 758–767. ISSN: 00218901. DOI: 10.1111/j.1365-2664.2011.01993.x.

[11] Erling Wold, Thom Blum, Douglas Keislar, et al. "Content-based classification, search, and retrieval of audio". In: *IEEE Multimedia* 3.3 (1996), pp. 27–36. ISSN: 1070986X. DOI: 10.1109/93.556537.

[12] Guodong Guo and Stan Z. Li. "Content-based audio classification and retrieval by support vector machines". In: *IEEE Transactions on Neural Networks* 14.1 (2003), pp. 209–215. ISSN: 10459227. DOI: 10.1109/TNN.2002.806626.

[13] Lie Lu, Hong Jiang Zhang, and Hao Jiang. "Content analysis for audio classification and segmentation". In: *IEEE Transactions on Speech and Audio Processing* 10.7 (2002), pp. 504–516. ISSN: 10636676. DOI: 10.1109/TSA.2002.804546.

[14] Sadaoki Furui. "Cepstral Analysis Technique for Automatic Speaker Verification". In: *IEEE Transactions on Acoustics Speech and Signal Processing* 29.2 (1981), pp. 254–272.

[15] Enrique Alexandre, Lucas Cuadra, Manuel Rosa, et al. "Feature selection for sound classification in hearing aids through restricted search driven by genetic algorithms". In: *IEEE Transactions on Audio, Speech and Language Processing* 15.8 (2007), pp. 2249–2256. ISSN: 15587916. DOI: 10.1109/TASL.2007.905139.

[16] Jonathan Dennis, Huy Dat Tran, and Haizhou Li. "Spectrogram image feature for sound event classification in mismatched conditions". In: *IEEE Signal Processing Letters* 18.2 (2011), pp. 130–133. ISSN: 10709908. DOI: 10.1109/LSP.2010.2100380.

[17] Selina Chu, Student Member, Shrikanth Narayanan, et al. "Environmental Sound Recognition With TimefffdfffdfffdFrequency Audio Features". In: *Language* 17.6 (2009), pp. 1142–1158. ISSN: 15587916. DOI: 10.1109/TASL.2009.2017438. URL: http://ieeexplore.ieee.org/xpls/abs%7B%5C_%7Dall.jsp?arnumber=5109766.

[18] Roneel V. Sharan and Tom J. Moir. “Noise robust audio surveillance using reduced spectrogram image feature and one-against-all SVM”. In: *Neurocomputing* 158 (2015), pp. 90–99. ISSN: 18728286. DOI: 10.1016/j.neucom.2015.02.001. URL: http://dx.doi.org/10.1016/j.neucom.2015.02.001.

[19] Guodong Guo, Stan Z. Sz Li, and Guodong Guo. “Content-based Audio Classification and Retrieval using SVM Learning”. In: *First IEEE Pacific-Rim Conference on Multimedia, ...* 8.i (2000), pp. 619–625. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.138.7193%7B%5C&%7Drep=rep1%7B%5C&%7Dtype=pdf%7B%5C%%7D5Cnhttps://www.microsoft.com/en-us/research/publication/content-based-audio-classification-and-retrieval-using-svm-learning/.

[20] Asma Rabaoui, Manuel Davy, Stéphane Rossignol, et al. “Using one-class SVMs and wavelets for audio surveillance”. In: *IEEE Transactions on Information Forensics and Security* 3.4 (2008), pp. 763–775. ISSN: 15566013. DOI: 10.1109/TIFS.2008.2008216.

[21] Geoffrey Hinton, Li Deng, Dong Yu, et al. “Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups”. In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 82–97. ISSN: 10535888. DOI: 10.1109/MSP.2012.2205597.

[22] Jinhai Cai, Dominic Ee, Binh Pham, et al. “Sensor Network for the Monitoring of Ecosystem: Bird Species Recognition”. In: *2007 3rd International Conference on Intelligent Sensors, Sensor Networks and Information*. 2007, pp. 293–298. ISBN: 978-1-4244-1501-4. DOI: 10.1109/ISSNIP.2007.4496859. URL: https://ieeexplore.ieee.org/abstract/document/4496859/%20http://ieeexplore.ieee.org/document/4496859/.

[23] Oguzhan Gencoglu, Tuomas Virtanen, and Heikki Huttunen. “RECOGNITION OF ACOUSTIC EVENTS USING DEEP NEURAL NETWORKS”. In: *2014 22nd European Signal Processing Conference (EUSIPCO)* (2004), pp. 506–510.

[24] Jinglan Zhang, Kai Huang, Mark Cottman-Fields, et al. “Managing and Analysing Big Audio Data for Environmental Monitoring”. In: *2013 IEEE 16th International Conference on Computational Science*

and Engineering. IEEE, Dec. 2013, pp. 997–1004. ISBN: 978-0-7695-5096-1. DOI: `10.1109/CSE.2013.146`. URL: `http://ieeexplore.ieee.org/document/6755327/`.

[25] Zvi Kons and Orith Toledo-Ronen. “Audio event classification using deep neural networks”. In: *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH* August (2013), pp. 1482–1486. ISSN: 19909772.

[26] Roneel V. Sharan and Tom J. Moir. “An overview of applications and advancements in automatic sound recognition”. In: *Neurocomputing* 200 (2016), pp. 22–34. ISSN: 18728286. DOI: `10.1016/j.neucom.2016.03.020`. URL: `https://www.sciencedirect.com/science/article/pii/S0925231216300406`.

[27] Woo Hyun Choi, Seung Il Kim, Min Seok Keum, et al. “Acoustic and visual signal based context awareness system for mobile application”. In: *IEEE Transactions on Consumer Electronics* 57.2 (2011), pp. 738–746. ISSN: 00983063. DOI: `10.1109/TCE.2011.5955216`.

[28] Xiaoxia Zhang and Ying Li. “Environmental Sound Recognition Using Double-Level Energy Detection”. In: *Journal of Signal and Information Processing* 04.03 (2013), pp. 19–24. ISSN: 2159-4465. DOI: `10.4236/jsip.2013.43b004`.

[29] Vivek Tyagi and Christian Wellekens. “On desensitizing the mel-cepstrum to spurious spectral components for robust speech recognition”. In: *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings* I (2005), I/529–I/532 Vol. 1. ISSN: 15206149. DOI: `10.1109/ICASSP.2005.1415167`.

[30] Jhing-Fa Wang Cai-Bei Lin Jia-Ching Wang Hsiao-Ping Lee and Cai-bei Lin. “Robust Environmental Sound Recognition for Home Automation”. In: 5.1 (2007), pp. 1–7. URL: `papers3://publication/uuid/85D3E57C-9CCE-419A-98EC-6131D3C21562`.

[31] Chanwoo Kim and Richard M. Stern. “Power-Normalized Cepstral Coefficients (PNCC) for Robust Speech Recognition”. In: *IEEE/ACM Transactions on Audio Speech and Language Processing* 24.7 (2016), pp. 1315–1329. ISSN: 23299290. DOI: `10.1109/TASLP.2016.2545928`.

[32] R. D. Patterson, K. Robinson, J. Holdsworth, et al. "Complex Sounds and Auditory Images". In: *Auditory Physiology and Perception.* 1992. 2013, pp. 429–446. DOI: `10.1016/b978-0-08-041847-6.50054-x`.

[33] Malcolm Slaney. "An Efficient Implementation of the Auditory Filter Bank". In: *Apple Computer Technical Report* 1.35 (1993), p. 40. ISSN: Apple Computer Technical Report #35. URL: `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.81.4406%7B%5C&%7Drep=rep1%7B%5C&%7Dtype=pdf`.

[34] Octavian Cheng. "Performance Evaluation of Front-end Processing for Speech Recognition Systems". In: *The University of Auckland School of Engineering Report* 621 (2005), p. 33.

[35] Xavier Valero and Francesc Alias. "Gammatone cepstral coefficients: Biologically inspired features for non-speech audio classification". In: *IEEE Transactions on Multimedia* 14.6 (Dec. 2012), pp. 1684–1689. ISSN: 15209210. DOI: `10.1109/TMM.2012.2199972`. URL: `http://ieeexplore.ieee.org/document/6202347/`.

[36] Donald D. Greenwood. "A cochlear frequencyfffdfffdfffdposition function for several speciesfffdfffdfffd29 years later". In: *The Journal of the Acoustical Society of America* 87.6 (June 1990), pp. 2592–2605. ISSN: 0001-4966. DOI: `10.1121/1.399052`. URL: `http://asa.scitation.org/doi/10.1121/1.399052`.

[37] Malcolm Slaney. "Auditory toolbox. Version 2 : A Matlab Toolbox for Auditory Modeling Work". In: *Apple Computer Company: Apple Technical Report* 45 (1998), pp. 1–41. ISSN: 15297853.

[38] Bin Gao and W. L. Woo. "Wearable audio monitoring: Content-based processing methodology and implementation". In: *IEEE Transactions on Human-Machine Systems* 44.2 (2014), pp. 222–233. ISSN: 21682291. DOI: `10.1109/THMS.2014.2300698`.

[39] Eric W. Healy, Sarah E. Yoho, Yuxuan Wang, et al. "An algorithm to improve speech recognition in noise for hearing-impaired listeners". In: *The Journal of the Acoustical Society of America* 134.4 (2013), pp. 3029–3038. ISSN: 0001-4966. DOI: `10.1121/1.4820893`.

[40] A. Lerner V. Vapnik. "Pattern recognition using generalized portrait method". In: *Automation Remote Control* 4 (6) (1963), pp. 774–780.

[41] Bernhard E. Boser, Isabelle M. Guyon, and Vladimir N. Vapnik. "A training algorithm for optimal margin classifiers". In: (2004), pp. 144–152. DOI: 10.1145/130385.130401.

[42] Moran-Cervantes Janeth. "Modeling Wildlife Population with Sound Identification". In: May (2018).

[43] Karol J. Piczak and J. Karol. "ESC". In: *Proceedings of the 23rd ACM international conference on Multimedia - MM '15*. New York, New York, USA: ACM Press, 2015, pp. 1015–1018. ISBN: 9781450334594. DOI: 10.1145/2733373.2806390. URL: http://dl.acm.org/citation.cfm?doid=2733373.2806390.

[44] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. "Deep Sparse Rectifier Neural Networks Xavier". In: 15 (2011), pp. 315–323. ISSN: 15324435. DOI: 10.1.1.208.6449. arXiv: 1502.03167. URL: http://proceedings.mlr.press/v15/glorot11a/glorot11a.pdf.

[45] Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. "Rectifier nonlinearities improve neural network acoustic models". In: *Icml '13* 28 (2013), p. 6. URL: http://www.stanford.edu/%7B~%7Dawni/papers/relu%7B%5C_%7Dhybrid%7B%5C_%7Dicml2013%7B%5C_%7Dfinal.pdf.

[46] Bing Xu, Naiyan Wang, Tianqi Chen, et al. "Empirical Evaluation of Rectified Activations in Convolutional Network". In: (2015). arXiv: 1505.00853. URL: http://arxiv.org/abs/1505.00853.

[47] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. "Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)". In: (2015), pp. 1–14. arXiv: 1511.07289. URL: http://arxiv.org/abs/1511.07289.

[48] Brian Mcfee, Eric J Humphrey, and Juan P Bello. *A SOFTWARE FRAMEWORK FOR MUSICAL DATA AUGMENTATION*. Tech. rep. URL: https://bmcfee.github.io/muda.

[49] Justin Salamon, Christopher Jacoby, and Juan Pablo Bello. "A Dataset and Taxonomy for Urban Sound Research". In: *Proceedings of the ACM International Conference on Multimedia - MM '14*. 2014. ISBN: 9781450330633. DOI: 10.1145/2647868.2655045.

[50] Alex Lenail. *NN SVG*. URL: http://alexlenail.me/NN-SVG/LeNet.html.