Tampere University

Muhammad Ali Gulzar

# DETECTING EPILEPSY SEIZURE USING A SCREAM DETECTOR

Faculty of Information Technology and Communication Sciences

M.S.c Thesis

October 2023

# ABSTRACT

Muhammad Ali Gulzar: Detecting Epilepsy Seizure Using a Scream Detector
Tampere University

Master's Degree Programme in Information Technology
October 2023

A vast amount of data is available which can be processed and provide useful information to each person in interest of their application. One of the dominating applications to use data is the medical field and in specific, detecting epileptic seizure. There are various symptoms that can declare that a patient is having a seizure. Some of the symptoms include twitching, loss of consciousness or screaming. Detecting scream from patient’s whole night video is the main aim of this thesis. In recent years, seizure detection has been given much attention but this is an extremely challenging task due to lack of scream data available as well as mixture of dataset available.

To solve this task, deep learning approach has been adopted. Convolutional neural network (CNN) model has been used to construct a scream classifier. Convolutional neural networks have the ability to extract features from the input data and learn the features. CNN has the capability to compute the data piece by piece using sliding window which gives the benefit of fast computation. Main focus of this thesis was to learn how different number of hidden layers as well as neurons in these layers can benefit in making an accurate and stable scream classifier. Apart from that, it is analyzed how different datasets effect the learning as well as the output of applying the learned model on unseen data. Features for raw audio data were extracted using *audio melbands*. This feature is called melbands and are fed to the convolutional neural network. Moreover, effect of providing input data with different number of rows in analyzed. Lack of scream data makes it difficult to have an optimum scream classifier to be applied on patient’s videos. Multiple hidden layers are used in the model architecture. Dropout, normalization and Rectified Linear Unit (ReLu) technology have been used to generalize as well as improve the accuracy and stability of the model. Furthermore, the effect of various hyper-parameters on the results have been analyzed. The performance of the model is examined with respect to the input data as well as the configuration of the model.

# PREFACE

The research work presented in this thesis has been conducted at Neuro Event Labs in Tampere, Finland. This thesis is a culmination of my studies at Master's in Information Technology.

I am grateful to ALLAH ALMIGHTY for blessing me with considerable blessing and my family. I can’t thank enough to my supervisor, Esa Rahtu, for his support and excellent guidance throughout my work.

Tampere, 15 May 2020
Muhammad Ali Gulzar

# CONTENTS

## LIST OF SYMBOLS AND ABBREVIATIONS

| | |
|---|---|
| DBN | Deep Belief Networks |
| CNN | Convolutional Neural Networks |
| GPU | Graphic Processing Unit |
| SNN | Shallow Neural Network |
| NN | Neural Network |
| DL | Deep Learning |
| ReLu | Rectified Linear Unit |
| RNN | Reccurent Neural Network |
| MFCC | Mel Frequency Cepstral Coefficient |
| SVM | Support Vector Machine |
| HMM | Hidden Markov Model |
| GMM | Gaussian Mixture Model |

# 1. INTRODUCTION

There is vast amount of data available on-line in recent times and have ever growing importance. Its importance has grown so immensely that it can be said that data is the new oil of the digital economy. Researchers and practitioners apply varied techniques and methods that can extract useful information from the raw data. Applications ranging from camera of a mobile phone to an artificial intelligence robot uses data to provide foremost experience. One of the major domains that uses data is the medical field. Applications ranging from detecting breast cancer to fatty liver diseases [1][2]. In addition to the mentioned applications, data is used to detect *epileptic seizures*.

Epileptic seizure is a neurological disorder that affects people of all ages. Epileptic seizure causes brain activity of a person to become abnormal. This leads to the person having seizures or periods of unusual behavior, perception and sometimes loss of awareness. Some of the few seizure types include tonic, clonic, myoclonic, atonic and tonic-clonic. In 2015, there were 3.4 million people with epilepsy in the United States of America alone. Per World Health Organization, around 50 million people have epilepsy worldwide. Figure 1 shows the distribution of patients across United States of America as of 15 April 2020 provided by Centre for Disease Control and Prevention.

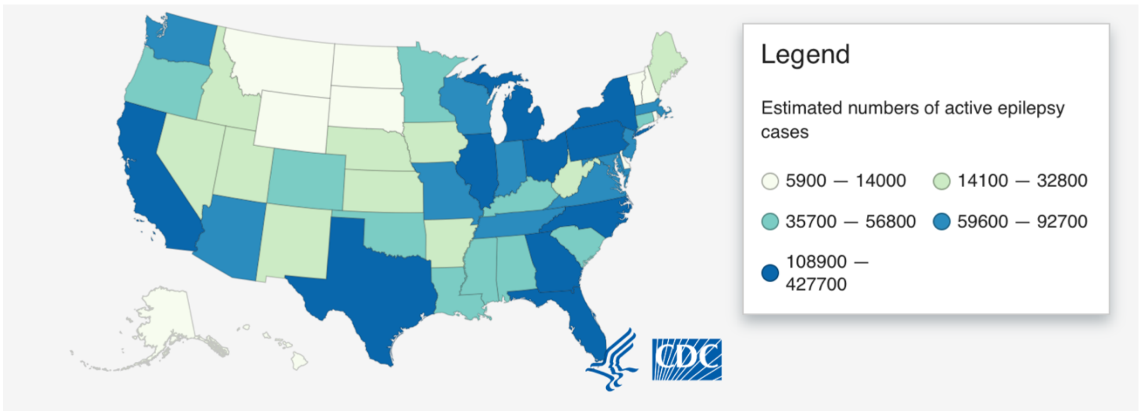

***Figure 1. Distribution of epilepsy patients in United States of America***

The foremost person gaining benefit to detect epilepsy seizures are the patients themselves. Early detection of seizures allows early rectification which leads to a higher probability of curing the disease in an accelerated time span. This benefit is the main reason that patients visits specialized doctors in the first place. The second person to gain benefit to detect epilepsy seizures using data are the doctors. Previously, cameras are installed in the room of the patient to record whole night videos. The doctors or nurses watch these videos to answer three main questions: At what time did the seizure occur; How long did the seizure last and what type of seizure did the patient had. After gaining information to these questions, doctor or nurse would prescribe certain methods to the patient to eliminate epileptic seizures. However, using data, a new technique is developed in the digital economy to detect epileptic seizures. Previously installed cameras provide whole night videos. These videos are then used by computers to detect epilepsy seizures and provide answer to the three main questions which were before provided manually by a doctor or a nurse.

As epilepsy is caused by abnormal activity in the brain, seizures can affect any part of the brain coordinates. This leads to different signs and symptoms of epilepsy which include temporary confusion, uncontrollable jerking movements of arms and legs, loss of consciousness or physic symptoms such as fear. Symptoms depend on the type of seizure. Most often, a person with epilepsy will have the same symptom each time he/she have a seizure. One of the physic symptoms include *screaming* which can be detected using data and machine learning.

Machine learning have seen rapid advancement over time. Many renowned companies and research centers have adopted to use machine learning techniques and methods to improve their efficiency and productivity. Machine learning has remarkable breakthrough in deep-learning in recent times. Deep neural network is a representation of a human brain architecture which have shown extraordinary results on many complex problems. Some of the examples of the neural network are Recurrent Neural Network (RNN), Deep Belief Network (DBN) and Deep Feed Forward (DFF) [7]. In this work to detect epileptics seizures, I will be using Convolutional Neural Network (CNN).

## 1.1 Problem Statement

Research in detecting epileptic seizure include collecting data, refining raw data, extracting useful features and use an appropriate machine learning approach to detect the symptoms of the seizure. One approach is adopted to solve epileptic seizure: classifier to detect if it the audio is a scream or not.

Classification of audio into scream and non-scream is difficult due to two main reasons:

- Lack of data for scream audio.
- Features of a scream can be corresponding to other sounds.

One of the main problems is that there is a lack of data of scream audio files [8]. Data is the source of teaching neural networks to classify between a scream and non-scream audio. However, solving the issue of lack of data does not settle the situation. It could be that the features of the scream, that are used to train the neural network, are corresponding to other sounds around the room of the patient which can lead to incorrect detection of seizures.

Despite of the limitations, challenge of this research is to have a machine learning model that correctly predicts the screams of patients having epileptic seizures with efficiency and accuracy.

## 1.2 Aim and Objectives

Two main aims are being targeted:

- Make a classification model that can detect if an audio is a scream or not with accuracy.
- Improve the computation speed of detecting scream from whole night videos.

For classification purpose, Convolutional Neural Networks is used to do achieve the aims. Its capabilities on performing machine learning on images are well known and explored. However, CNN have proved its efficiency and accuracy in various audio classifications problems [3]. CNN is made up of neurons with learnable weights. Each neuron in the CNN receives input values and takes a weighted sum over them. Finally, it is passed through an activation function and provides with an output.

# 2. FRAMEWORK AND DATASET

This chapter contains the information about the framework and dataset used to train machine learning models and for experimentation in this thesis.

## 2.1 Framework

For scream detection classification task, Keras is used to develop deep learning models. Keras is a programming framework and neural network library written in python programming language. It is a compact library which focuses on fast experimentation and allow to easily develop applications that requires deep learning. With the help of Keras, it is easy to write custom building blocks with very few restrictions. Keras can run on top of Tensorflow as well as Theano. Tensorflow was first developed for internal use by Google Brain. It is a library which is used for machine learning applications such as neural networks. In Tensorflow, we can also easily define the architecture on which the code should execute; whether on CPU or GPU. Furthermore, it provides a wide range of mathematics library to easily compute complex problems using high level language such as C++ or Python. On the other hand, Theano is a library for rapid numerical computation using python programming language. Theano is developed by MILA group in University of Montreal. However, for this experiment, Tensorflow has been used to develop a scream detection model. One of the factors which shows the growing popularity of Tensorflow instead of Theano is the number of search done in Google for the term Tensorflow and Theano which is show in Figure 2 and 3.

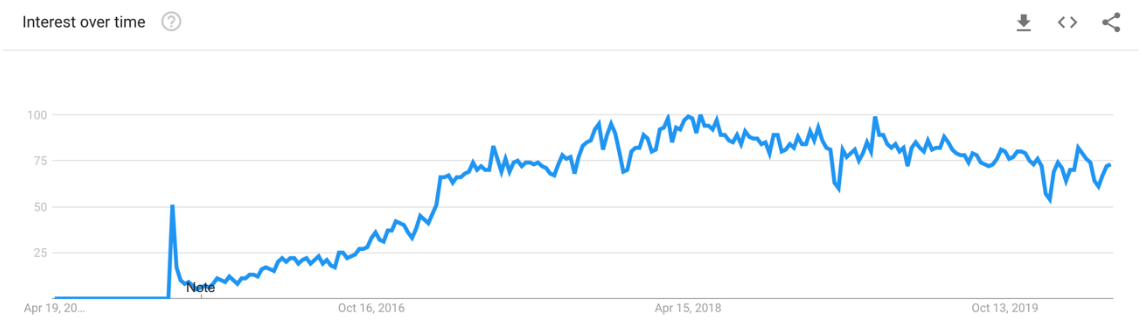

***Figure 2. Search trend for the term Tensorflow over the last 5 years worldwide***

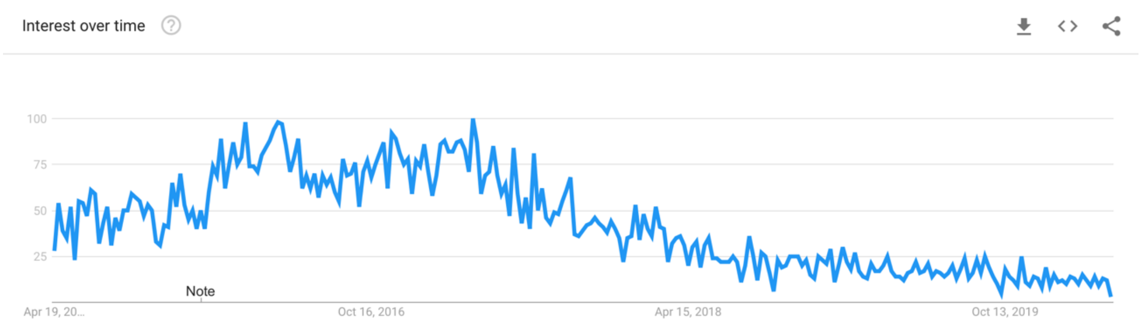

***Figure 3. Search trend for the term Theano over the last 5 years worldwide***

Audio files in raw form are no use to the neural network. We need to provide certain features of the audio that the neural network can learn from. For this purpose, we used the pipeline functionality which pass the audio files using ffmpeg package and extract audio melbands [35] using C++ code. Ffmpeg package is a software service which is used to handle video, audio and other multimedia files. It contains various features such as distortion, modulation, resampling and trimming media files. On the other hand, melbands are coefficients that are derived from a type of cepstral (It is a result of mathematical transformation in the field of Fourier transform) representation of the audio clip [4].

## 2.2 Datasets

There is a lack of dataset to train a neural network for scream detection. However, different techniques were adopted to provide enough data to the neural network to have a sustainable and accurate scream detection model.

### 2.2.1 Mivia Dataset

Mivia is a research lab at the University of Salerno which has expertise in Pattern Recognition and Computer Vision. Along with various research and publications, it provides public datasets in the field of audio analysis, biomedical image, graph and video analysis [5]. In the Mivia Audio Events Data section, there are various sounds but for this experiment, scream audio were used. There is a total of 6000 events which are equivalent to 7934,2 seconds of audio. In the documentation of Mivia, they divided the 6000 events of scream to 4200 training set and 1800 testing set. However, it was compounded to be used as one single dataset for training due to the lack of scream audio data [36].

### 2.2.2 Noise Dataset

As there is a lack of scream audio data, a new approach was adopted to enhance the dataset. An element called noise was developed using C++ code. It is an element which adds Gaussian random noise to the data [6]. There are 2 parameters set for this element which affect the Gauss-ian random noise; alpha which is the scale factor beta is the addition to the generated random number.

### 2.2.3 Csaba Dataset

Besides having scream audio data, there is a need for non-scream audio data to train a neural network for scream detection. For this purpose, we used CSIBE-RAW [37] dataset which include various common sounds such as baby cry, guitar and human speech. The format of the audio files is mono WAV with 16-bit depth and 44.1 kHz sampling rate.

# 3. RELATED WORK

This has been considerable research done on detection and classification of audio events [9], [10], [11], [12], [13], [14]. One of the focus of research in audio detection is scream detection. Various techniques as well as datasets have been used to resolve this problem. Various applications require scream detection as well as research done on them such as:

- Detection of scream and shouted speech in subway trains
- Scream sound detection for acoustic surveillance applications
- To monitor space including various environmental sounds and conversation voices

The following sections discusses old approaches for classification of scream classifier as well as difficulties obtained during the construction of a scream classifier.

## 3.1 Scream Detection Difficulties

Research demonstrates that the task of classifying an audio is not a straightforward task. In a research to make a scream and shouted speech classifier, the data is mixed. It means that the data contains occurrences belonging to different classes at the same time which is referred to as source separation problem in the research. To tackle this problem, some sort of pre-processing separation algorithm was required to have a useful dataset.

One of the problems in this research is insufficient information for differentiating scream and non-scream sounds. The individual frames do not contain ample information. As the research in using Support Vector Machine (SVM) classifier, individual frames of scream and non-scream sounds are highly overlapped in the feature space which will cause problems in this specific classifier. Furthermore, there is a lack of scream sound data available and cause imbalance between scream and non-scream sounds. For having a stable and accurate model, large dataset is always recommended but with an optimum level.

## 3.2 Scream Detection Approaches

Various approaches have been adopted for detection techniques with many different combinations of features and classifiers. Some of them include [14], [15], [16], [17], [18], [19], [20]:

- Mel Frequency Cepstral Coefficient (MFCC) / melbands with Gaussian Mixture Model (GMM)
- MFCC with Support Vector Machine (SVM)
- MFCC with Hidden Markov Model (HMM)
- MFCC with multiclass Adaboost
- MFCC and other spectral features classified with GMM
- MFCC and other spectral features classified with k-nearest neighbor (kNN)
- MFCC and other spectral features classified with random forest
- Gabor features classified with GMM
- Gabor features classified with SVM

There are other features and classifiers used beside the ones mentioned above. All of them are categorized under machine learning approach. Machine learning approach is based on scream and non-scream classification. This approach is further divided into two categories: supervised and unsupervised learning. Figure 4 shows the classification techniques used in the research.

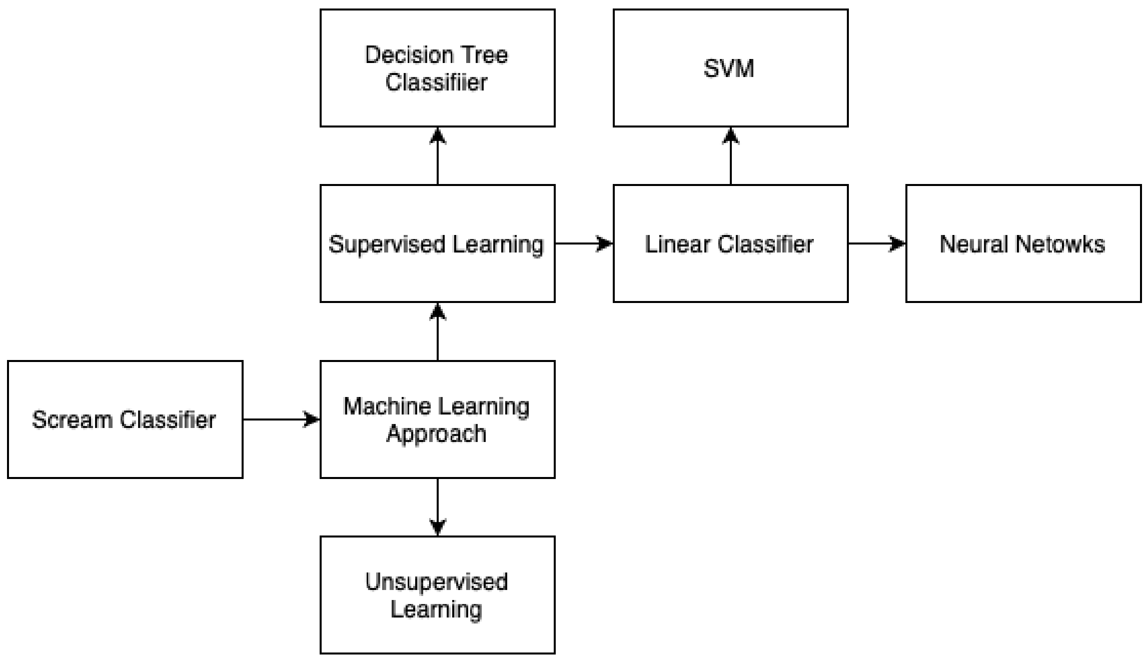

***Figure 4: Scream classification techniques***

- **Support Vector Machines (SVM):** To enhance the power to distinguish between scream and non-scream sounds, a sound-event partitioning method (SEP) have been adopted. SEP method provides the functionality of extracting multiple acoustic vectors from a single sound event. The experiment is based on 1000 sound events and after applying the SEP methods, Support Vector Machine (SVM) is used to classify between a scream and non-scream sound. This approach does not prefer the MFCC features to train a model which is due to, according to the research, provides 60% reduction in equal error rate (EER) when compared to using MFCC as features. SVM was also selected for its computation efficiency [8].

- **Neural Networks:** Deep learning approach was adopted in detecting scream and shouting sounds in a subway train [7]. In neural network approach, the Restricted Boltzmann Machines (RBM) was stacked into Deep Belief Network which in result was a final product of Deep Neural Network (DNN). RBM is a neural network with one layer of hidden units which have undirected connections with visible units. The interaction between visible and hidden units is through an Energy function as described below:

$$\boldsymbol{E}(\boldsymbol{v}, \boldsymbol{h}; \boldsymbol{\theta}) = -\sum_{i=1}^{V}\sum_{j=1}^{H} w_i v_i h_j + \frac{1}{2}\sum_{i=1}^{V} b_i v_i - \sum_{j=1}^{H} a_j h_j$$

DBN are composed of stacked RBM which learn various correlations between the data. As it is a classification problem, at the DBN is converted to a DNN as DNN has the capability to model the frontier between different classes through a discriminative rule based on gradient descent and error propagation. The data was divided into four categories of sound: Scream, shout, conversation and noise. This model used the MFCC features to train the model with the specifically using the 12th order of MFCC.

## 3.3 Perceptron

Perceptron is the mathematical representation of a biological neuron. In an actual neuron, when the strength of a signal exceeds a specific threshold, it produces a signal whereas perceptron base on the same principle. Perceptron is made up of four parts:

- Input values
- Weights
- Net sum
- Activation function

Each input is multiplied by a certain value called weight. Moving on, the weights of multiple inputs is summed together to represent the strength of the signal and an activation function is used to calculate the output value [22].

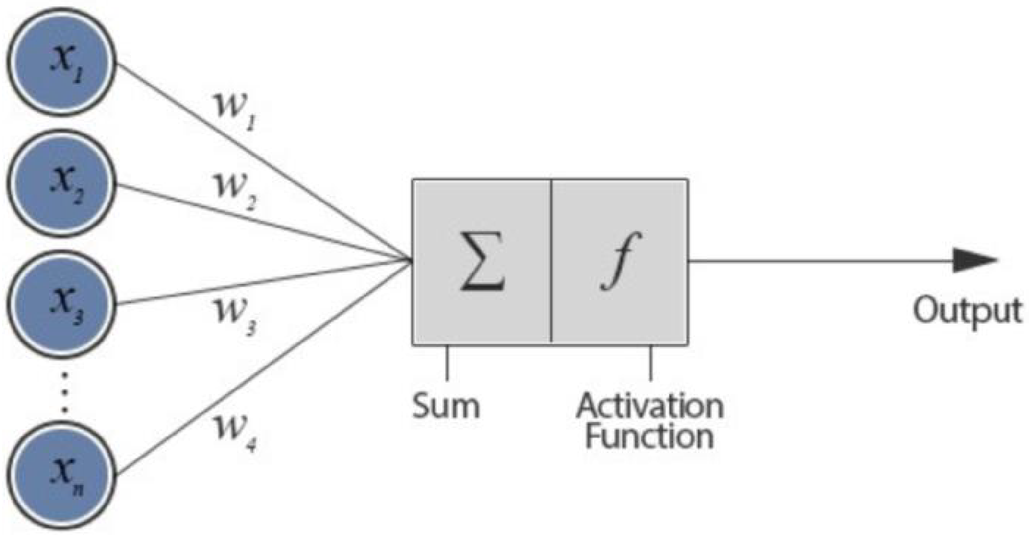

***Figure 5: Internal structure of an artificial neuron***

If we analyze figure 5, inputs (x1, x2, …., n) and weights (w1, w2, …, y) are real numbers which can be both negative and positive. All input values are multiplied with their respective weights assigned and added together. The result that is obtained through the stage of addition, it is passed through an activation function.

There are various activation functions available. One of the activation functions is a *step-function*. The following example fully demonstrates the workflow in a perceptron using a step-function activation function. A step function will deliver a result of 1 if it is above a certain threshold and 0 if it less than the threshold value. Example:

Input 1 (**x1**) = 0.4 and weight 1 (**w1**) = 0.5
Input 2 (**x2**) = 0.2 and weight 2 (**w2**) = 0.1
Threshold = 0.5

Perceptron calculation with step function:

$$(x_1 * w_1) + (x_2 * w_2) = (0.4 * 0.5) + (0.2 * 0.1) = 0.22 \rightarrow 0$$

As the signal strength (0.22) was less than the threshold (0.5), the result is 0.

## 3.4 Neural Network

A neural network can be considered as an artificial neural network that can receive, process and transmit information. In biological terms, it is made up of different cells and each cell perform a small part of computation of a bigger problem. Figure 6 shows a presentation of a neural network which consists of three layers:

- Input layer: Input data is transmitted through this layer
- Hidden layer: There can be multiple hidden layers between input and output layer and these layers perform computation on the input data provided through input layer.
- Output layer: This is the final layer in the neural network that produces the final output from the processed data.

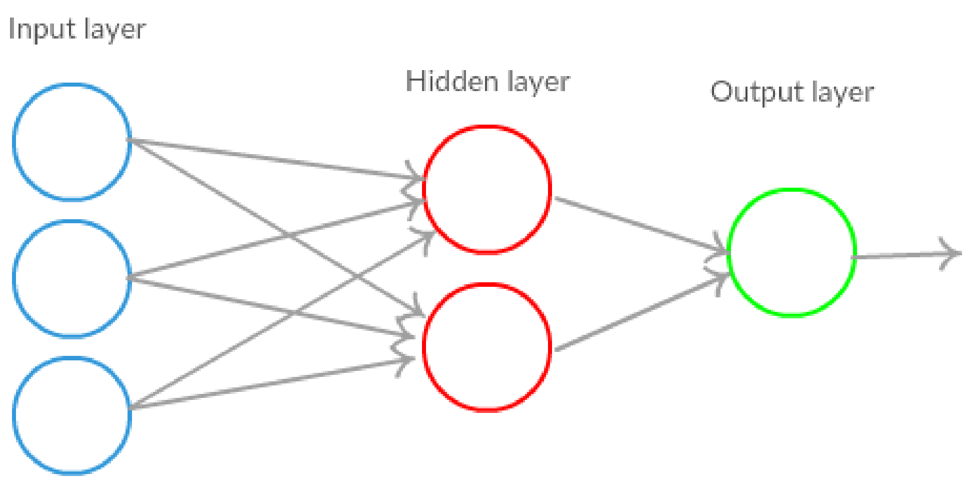

***Figure 6: A simplistic neural network representation***

Input layer is the first layer in any neural network. It does not have any weights and biases values associated with them. It takes the input signal and passes them to the next layer for processing. Each neuron in input layer should represent an independent variable that has some effect on the output of the neural network [23].

Hidden layer is also known as the middle layer. This layer performs different transformations to the input data with the help of an activation function. It extracts important features from the data that is provided from the previous layers or a layer. In a hidden layer, neurons are stacked vertically as shown in Figure 6. Figure 6 also demonstrates that there is one hidden layer in the neural network and the layer contains 2 neurons. When the task requires complex decisions, the usage of more than one hidden layer is recommended. However, it does not guarantee that increasing the number of hidden layers will be proportional to the accuracy of the results. Furthermore, there is a certain extent of adding hidden layers at which point, the accuracy will become constant or even decrease. Beside the number of hidden layers, the number of neurons in the hidden layer also affect the quality of the neural network. Low number of neurons compared to the complexity of the task being solved will result in *under-fitting*. Similarly, excessive number of neurons in a neural network will result in *over-fitting*.

Output layer is the last layer in the neural network that receives the input from the last hidden layer. This layer provides the results in a desired amount. Figure 6 represents that there is one neuron in the output layer which will produce one single value as a final result.

Weights play an important role in the training of a neural network. The calibration of the weights is done during the training of the model and the calibration is achieved by repeating forward and backward propagation.

### 3.4.1 Deep Learning

The works of perceptions from 1950 and multi-layer perceptron success are quite recent though the work was done several decades ago [42] [43]. There are many factors contributing to this phenomenon; the foremost being the accessibility of powerful computers. Now there is usage of graphical processing unit (GPU) instead of CPU for training machine learning models. Introduction of dropout, ReLU and proper random initialization all contribute to the success [44].

In classical machine learning approach, the classification is performed in such a way that specific features are extracted from the data and then use a general-purpose classifier on top such as support vector machine (SVM) [45]. However, in complex problems, extracting good features is difficult such as in computer vision [46].

Due to the recent advancements, deep learning has solved many problems including discussed in the previous paragraph. A deep model has several hidden layers of computations that automatically discovers more complex features to be used. The number of perceived regions in a deep architecture increase almost exponentially with the number of parameters by learning and combining multiple levels of representation [47].

### 3.4.2 Forward Propagation

In a neural network, the input layer provides the initial state of the data, propagates through the hidden layers and produce an output using an output layer. As discussed in section 3.3, weights are assigned to each value; at the first phase of training, the weights are assigned randomly. In forward propagation, these set of weights are applied to the input data where it finally produces an output. This procedure can be repeated various times and the occurrence depends on the complexity level of the task. Figure 7 shows a neural network with forward propagation in which weights are assigned randomly.

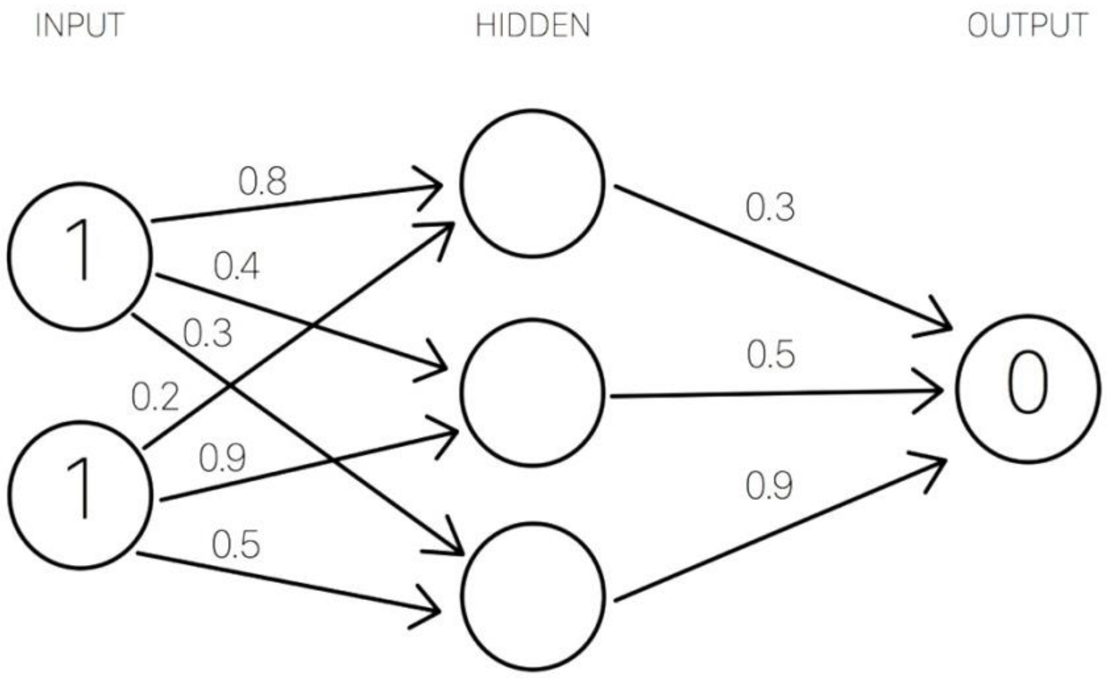

***Figure 7: Weight assigning in forward propagation***

### 3.4.3 Backward Propagation

When the forward propagation produces a result using random weights assigned, the result is then compared to the actual expected result to compute the error. To decrease the computed error, weights are propagated backwards by taking the derivative of the error according to each weight and subtracting the value from the specific assigned weight. Likewise, in forward propagation, the computation occurs at each layer in backward propagation. Forward propagation is a part of backward propagation but it occurs before backward propagation but both works together [24].

### 3.4.4 Activation Function

Neurons produce a signal value when the input value is multiplied with its respective weight, add bias and add all the results together. The decision if the neuron should be activated or not is decided by the activation function using the signal value. Without an activation function, a neural network is simply a linear regression model. Activation function produces non-linear transformation of the input data and the result is sent to the next layer.

There are various types of activation function including the three most common ones:

- **Step function:** Step function is a simplistic logical function. In simple terms, if a certain value *X* is greater than the threshold value, is will be activated, otherwise not. Hence, it can be referred to as a threshold-based activation function. Figure 8 [25] shows that the output is 1 if the signal value is greater than the threshold value which is 0. On the contrast, the output is 0 if it is less than 0, again using the reference of the threshold value.

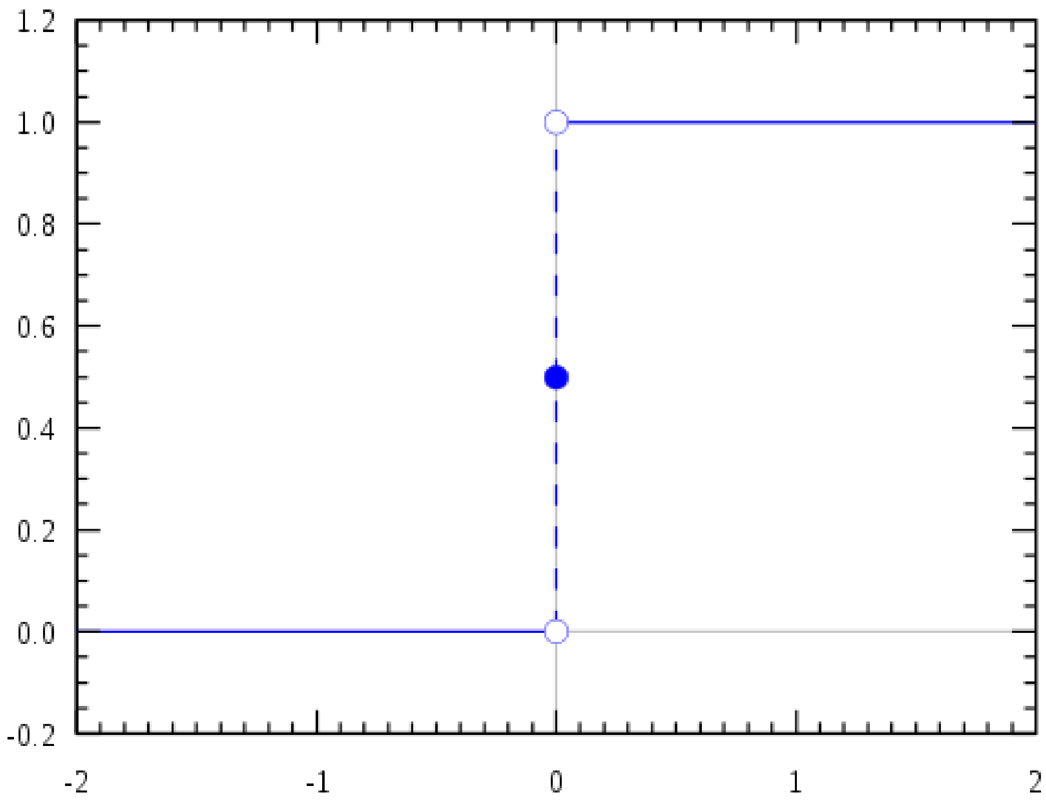

***Figure 8: Step function***

- **Sigmoid function:** Sigmoid function is non-linear in nature compared to step function. As shown in Figure 9, it has a S shaped curve and the output range is between 0 and 1. Furthermore, this function has the capability to bring the y-axis values to either end of the curve. This statement is supported by Figure 9 representation, as the y-axis values are very steep when the x-axis values are between -2 and 2 which means that any small change in x-axis values will result in a significant change in the y-axis.

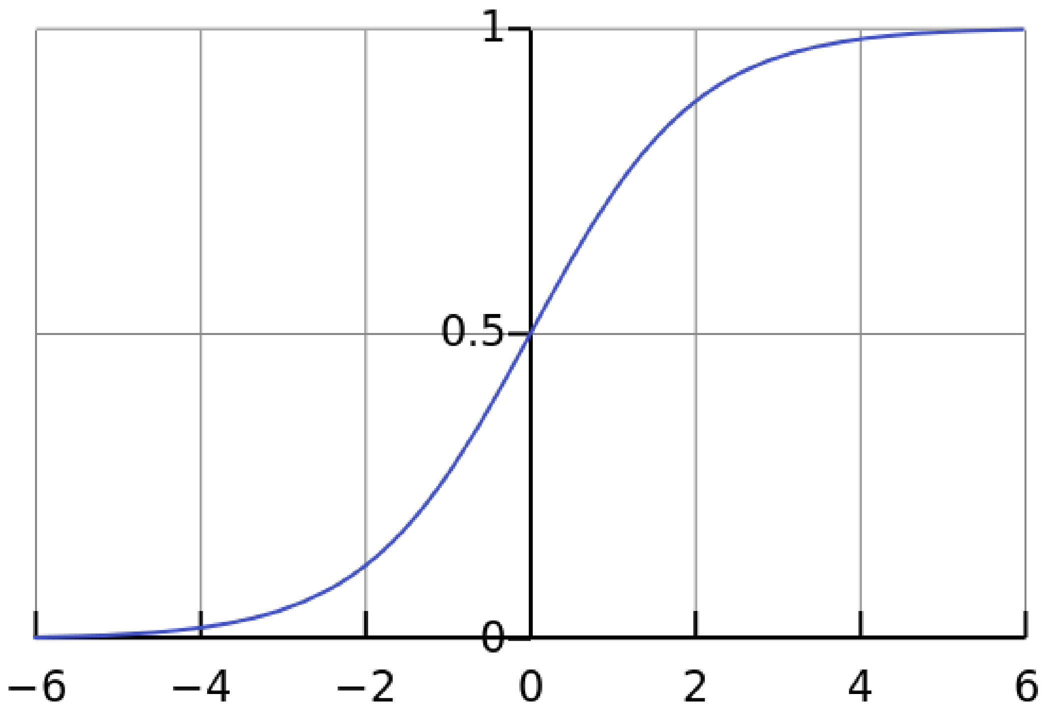

***Figure 9: Sigmoid activation function graph representation***

- **ReLu function:** ReLu (rectified linear unit) activation function can be easily understood by the following equation:

$$f(x) = \max(0, x)$$

The input value is either 0 or the value itself. The condition that determines is that if the input value is negative, it will be zero otherwise the input value itself. Figure 10 shows graph representation of ReLu but in reality, ReLu is non-linear in its nature. The output range of ReLu is from 0 to infinity, hence it not bounded. Furthermore, as it has simple mathematical operation, it is less expensive compared to other activation functions.

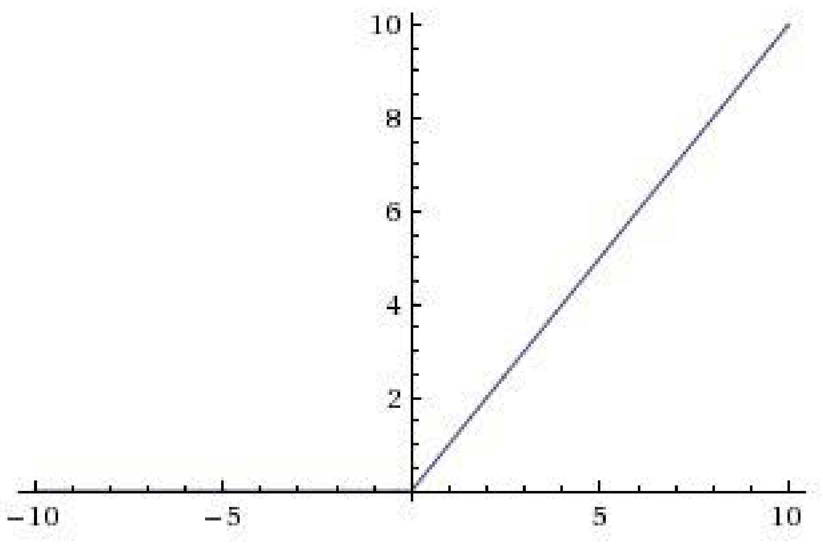

***Figure 10: ReLu function graph representation***

## 3.5 Convolutional Neural Network

This chapter discusses the important features of Convolutional Neural Network (CNN) and its importance in today's application. Furthermore, the comparison between human computation and learning power of neural networks is discussed. It is a human nature that no one starts to think about the problem statement or any situation from the start. Humans try to solve the problem in the best possible way with its previous knowledge.

Neural networks are splendid in image recognition task. However, if we consider that the image has large number of pixels that will be processed by the neural network, it would mean that the number of parameters for a neural network will increase. This situation will give arise to problems such as immense consumption of computational power and decrease in the performance of the neural network. That is why a simple neural network is not suitable for every case. Hence, Convolutional Neural Network is the most popular neural network that is used for image *classification* applications. Instead of weights assigned to each pixel in an image, a CNN has just enough weights to analyse a small patch of an image. It is like reading a sentence with a magnifying glass; eventually you read the whole sentence but you look at only small patch of the sentence at a specific time. This leads to having fewer parameters in the neural network which significantly improves the training time as well as require less amount of data to train.

Figure 11 illustrates the concept of computing an entire image piece by piece. The input layer provides data with dimensions 256 x 256. CNN can efficiently scan through the whole data chunk by chunk with a 5 x 5 window. The 5 x 5 window will slide through the whole image until the end. The sliding of the window usually occurs from left to right and top to bottom. Furthermore, it can be defined how fast the window should scan the whole data, defined by the term *stride.* As in Figure 11, if we set the stride 2, it would mean that the window 5 x 5 will slide 2 pixels every time until it spans the entire data. Hence, this is one of the main features that distinguishes CNN from other neural networks.

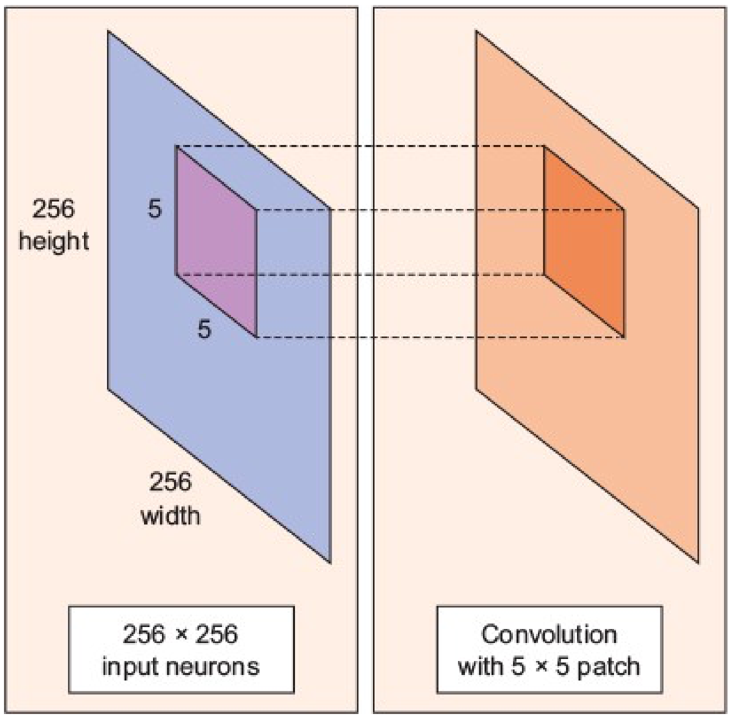

***Figure 11: Convolutional process on a data***

# 4. PROPOSED METHOD

This section will explain the techniques used in this research to have a stable and accurate model to detect seizure from patient’s video.

## 4.1 Audio Pre-processing

Audio in its raw form cannot be used to train a neural network. For this purpose, feature of an audio, melbands are extracted from audio files. To achieve this target, third party packages such as ffmpeg and libXtract are used. The following subsections explain how each package is used to accomplish the point of reference.

### 4.1.1 Audio Melbands Element

LibXtract is a simple and lightweight library of audio feature extraction functions. This library is available for anyone to use and even manipulate the original functions; this term is often referred as Open Source. LibXtract can be used in programming languages such as C++, python or java but for this research, C++ was used to implement the function. An element called “audiomelbands” was implemented to extract melbands from raw audio files. Program 1 shows variables used in this element. Variables of the name melBands, frequency and volumeCutLevel can be passed as a parameter. Variable melBands are the amount of melbands we want to extract from an audio file whereas volumeCutLevel is a reference point where the data is set to zero if the mean of the audio melbands is less than the volumeCutLevel.

```
struct AudioMelBandsElement::Private {
  int melBands = 25;
  int frequency = 48000;
  double volumeCutLevel = 0;
  std::uint32_t windowSize = 0;
  std::unique_ptr<xtract_mel_filter> filterBank;
  std::vector<std::vector<double>> filterData;
  std::vector<double *> filterPointers;
  double *hannWindow = nullptr;
  std::vector<double> emptyMelData;
};
```

***Program 1: Variables used in the element audiomelbands***

Moving ahead, Program 2 shows the main function of the audiomelbands element. As it is shown on line 1, the name of the function is process, the input data as a parameter is a matrix as well as the output data and return a boolean if the function runs successfully. Line 2 checks if the input data is in CV_32FC1 format. This format can be interpreted as 32 bit floats with 1 channel. Line 8 will execute another function, initLibxtract if the variable filterBank is false. By the name of the function it can be interpreted what the function will imply. It will simply initiate the libXtract library with the appropriate variables. From line 9 to 22, the function generates melbands for silence if the emptyMelData array is empty. As mentioned before, volumeCutLevel is used to set the value of melbands to zero if the mean of the melbands if less than the variable volumeCut-Level. Three variables are initiated with the name rawData, melData and secondMelData. Variable rawData is initiated with the same dimensions as the columns of the input data as shown on line 32. Finally, function called generateMelBands is executed to obtain the desired results.

```
1.  bool AudioMelBandsElement::process(const cv::Mat &in, cv::Mat &out) {
2.    if (in.type() != CV_32FC1) {
3.      std::cerr << "Error: The AudioMelBands element accepts only CV_32FC1 data\n";
4.      return false;
5.    }
6.
7.    if (!p->filterBank) {
8.      initLibxtract(uint(in.cols));
9.      if (p->emptyMelData.empty()) {
10.
11.       std::uniform_real_distribution<float> distribution(-0.001f, 0.001f);
12.       std::mt19937 engine;
13.       auto generator = std::bind(distribution, engine);
14.       std::vector<double> emptyData(p->windowSize, 0);
15.       std::vector<double> cloneMelbands;
16.
17.       std::generate_n(emptyData.begin(), p->windowSize, generator);
18.       p->emptyMelData = generateMelBands(emptyData);
19.       cloneMelbands = p->emptyMelData;
20.       p->emptyMelData.insert(p->emptyMelData.end(), cloneMelbands.begin(), cloneMelbands.end());
21.     }
22.   }
23.
24.   if (checkVolumeCutLevel(in) && !p->emptyMelData.empty()) {
25.     out = cv::Mat::zeros(1, int(p->emptyMelData.size()), CV_32FC1);
26.     for (int i = 0; i < int(p->emptyMelData.size()); ++i) {
27.       out.at<float>(0, i) = float(p->emptyMelData[i]);
28.     }
29.     return true;
30.   }
31.
32.   std::vector<double> rawData(in.cols, 0);
33.   std::vector<double> melData;
34.   std::vector<double> secondMelData;
35.
36.   for (int i = 0; i < int(rawData.size()); ++i) {
37.     rawData[i] = double(in.at<float>(0, i));
38.   }
39.
40.   melData = generateMelBands(std::vector<double>(rawData.begin(), rawData.begin() + p->windowSize));
41.   secondMelData = generateMelBands(std::vector<double>(rawData.end() - p->windowSize, rawData.end()));
42.   melData.insert(melData.end(), secondMelData.begin(), secondMelData.end());
43.
44.   out = cv::Mat::zeros(1, int(melData.size()), CV_32FC1);
45.   for (int i = 0; i < int(melData.size()); ++i) {
    out.at<float>(0, i) = float(melData[i]);
  }
  return true;
}
```

***Program 2: Main function “process” of the element audiomelbands***

Program 3 shows how the function generateMelBands is implemented. Line 9 shows the main function that is used from libXtract library; xtract_mfcc, to generate melbands of an appropriate audio file. My appropriate it means that it contains the data in CV_32FC1 format [34] and is not silent which is checked against the volumeCutLevel parameter.

```
1.  std::vector<double> AudioMelBandsElement::generateMel-
    Bands(const std::vector<double> &samples) {
2.    std::vector<double> windowedData(p->windowSize, 0);
3.    std::vector<double> spectrumData(p->windowSize, 0);
4.    std::vector<double> tempData(p->filterBank->n_filters, 0);
5.    double userData[4] = { double(p->frequency) / p->win-
    dowSize, XTRACT_MAGNITUDE_SPECTRUM, 0.0, 0.0 };
6.
7.    xtract_windowed(samples.data(), int(p->windowSize), p->han-
    nWindow, windowedData.data());
8.    xtract_spectrum(windowedData.data(), int(p->win-
    dowSize), userData, spectrumData.data());
9.    xtract_mfcc(spectrumData.data(), int(p->windowSize / 2), p->fil-
    terBank.get(), tempData.data());
10.   return tempData;
11. }
12.
```

***Program 3: Implementation of generateMelBands function***

### 4.1.2 Noise Element

As mentioned in Chapter 2, noise dataset is generated through the usage of an element implemented in C++ programming language. The main aim of this element usage is to generate more data to have a well-trained neural network that can provide satisfactory results on unseen data. My unseen data, it refers to the data that is neither used in training or testing and for this research, the unseen data is patient data having seizures. The implementation of this element is straight forward. Program 4 shows the variables as well as how the random gaussian noise [38] is generated. Function default_random_engine is used to create a generator that will create a random gaussian noise which will be further manipulated as show in program 5.

```
struct NoiseElement::Private {
  float alpha = 1.0f;
  float beta = 0.0f;
  float upperBound = 1.0f;
  float lowerBound = 0.0f;
  std::default_random_engine generator;
  std::normal_distribution<float> distribution;
};

  unsigned seed = std::chrono::system_clock::now().time_since_epoch().count();
  p->generator = std::default_random_engine(seed);
  p->distribution = std::normal_distribution<float>(0.0f, 1.0f);

```

***Program 4: Variables used and generation of random gaussian noise implementation***

In most of the lines of the code in program 5, it performs a check on the data type of the input data so that it could return appropriate data but the addition of the noise is the same in every case. Line 7 shows that first the random number is generated which is then multiplied by the variable alpha and added to beta. This number is every number in the data. After having a data with gaussian noise added to it, it performs a lower and upper bound clamping which is shown on line 8.

```
1.  bool NoiseElement::process(const cv::Mat &in, cv::Mat &out) {
2.  
3.    out = in;
4.    for (int i = 0; i < out.cols; ++i) {
5.      for (int j = 0; j < out.rows; ++j) {
6.        if (out.type() == CV_32FC1) {
7.          auto tempVal = out.at<float>(j, i) + (p->distribution(p->generator) * p->alpha + p->beta);
8.          out.at<float>(j, i) = std::clamp(tempVal, p->lowerBound, p->upperBound);
9.        } else if (out.type() == CV_64FC1) {
10.         auto tempVal = float(out.at<double>(j, i)) + (p->distribution(p->generator) * p->alpha + p->beta);
11.         out.at<double>(j, i) = static_cast<double>(std::clamp(tempVal, p->lowerBound, p->upperBound));
12.       } else if (out.type() == CV_16SC1) {
13.         auto tempVal = float(out.at<std::int16_t>(j, i)) + (p->distribution(p->generator) * p->alpha + p->beta);
14.         out.at<std::int16_t>(j, i) = static_cast<std::int16_t>(std::clamp(tempVal, p->lowerBound, p->upperBound));
15.       } else if (out.type() == CV_16UC1) {
16.         auto tempVal = float(out.at<std::uint16_t>(j, i)) + (p->distribution(p->generator) * p->alpha + p->beta);
17.         out.at<std::uint16_t>(j, i) = static_cast<std::uint16_t>(std::clamp(tempVal, p->lowerBound, p->upperBound));
18.       } else if (out.type() == CV_8SC1) {
19.         auto tempVal = float(out.at<std::int8_t>(j, i)) + (p->distribution(p->generator) * p->alpha + p->beta);
20.         out.at<std::int8_t>(j, i) = static_cast<std::int8_t>(std::clamp(tempVal, p->lowerBound, p->upperBound));
21.       } else if (out.type() == CV_8UC1) {
22.         auto tempVal = float(out.at<std::uint8_t>(j, i)) + (p->distribution(p->generator) * p->alpha + p->beta);
23.         out.at<std::uint8_t>(j, i) = static_cast<std::uint8_t>(std::clamp(tempVal, p->lowerBound, p->upperBound));
24.       } else {
25.         std::cerr << "Error: Unsupported type " << out.type() << std::endl;
```

```
        return false;
      }
    }
  }
  return true;
}
```

***Program 5: Main function of element noise***

### 4.1.3 Generating Melbands

Section 4.1.1 and 4.1.2 introduced elements that was used to extract melband feature from raw audio files. In this section, it is addressed how these elements work together along with ffmpeg package. Package ffmpeg is used to read the audio file with multiple parameters. Ffmpeg is a leading multimedia framework able to decode, encode, transcode, mux, demux, stream, filter and play anything humans and machines have created. It supports the most obscure ancient formats up to the cutting edge. Program 6 shows a section of bash script to accomplish the desire target. Line 1 prints out a message of noise element parameters along with the name of the files it is going to perform the computation. Parameter “-i” indicates that this is the audio file to be read. After reading the audio file using ffmpeg, noise element is used to add noise to the data. In this section of the script, noise elements use parameter with alpha equal to 0.2 and beta to -0.1. However, apart from this section, there are two executions performed in this same audio file. First one is the simple extraction of melbands without any noise and the other one along with noise element with parameter alpha equal to 0.4 and beta to -0.2. After extracting melbands, line 8 write the result in an output file in csv format.

```
1.  echo "Extracting mel bands of file with alpha=0.2 beta=-
    0.1 $filename"
2.  output_file="output.csv"
3.  ffmpeg -loglevel error -i ${filename} -f f32le -c:a pcm_f32le -
    ar 48000 -ac 1 pipe:1 \
4.  | pipeline-launch input stride=6400 \
5.  ! cast width=1600 height=1 type=5 \
6.  ! noise alpha=0.2 beta=-0.1 lowerBound=0 upperBound=1 \
7.  ! audiomelbands melBands=25 \
8.  ! datawrite outputFile=${output_file}
```

***Program 6: Bash script to extract melbands using audiomelbands and noise element***

## 4.2 Model Architecture

There are multiple ways to build neural network, by using multiple layers and different components [40]. In this scream classifier task, we use Convolutional Neural Network (CNN). A CNN contains one or more than one convolutional layer. These layers can either be completely interconnected or pooled. Following are the components and layers used in building the model for this scream detection task.

### 4.2.1 Convolutional and Pooling Layer

The word convolutional itself is the meaning; this layer performs convolving objects on one another. The convolutional layer is the core building block of CNN and carriers the main portion of the network's computational load. The main objective of the convolution is to extract features such as edges and corners from the input data and in this task, melbands. As we go deeper inside the network, the network starts identifying more complex features. Convolutional layers perform dot product between two matrices where one matrix is the set of learnable parameters and the other matrix is the portion of an melband spectrum. At the end of the convolutional process, we have a featured matrix which less parameters compared to the actual input data as well as more clear features than the real one [41].

After every convolutional layer, there is a pooling layer. The purpose of this layer is to decrease the computation power required to process the melband data. It is achieved by decreasing the dimensions of the featured matrix further more. The dominant features are extracted from a restricted amount of neighborhood. Figure 1 explains the procedure of pooling. The orange box is the input data and the highlighted part is the current state where the pooling computation will be performed, also known as pooling kernel. The blue box is the output data after pooling is performed. There are two different types of pooling and each provides a different result:

- **Max-pooling:** In this procedure, the maximum amongst all the values lying inside the pooling kernel is interpreted. Figure 12 is an example of max pooling. As we can see from the pooling kernel that the maximum amongst the values is 3, hence the result 3 is interpreted and given as an output data in blue box.

- **Average-pooling:** As the name suggests, the average among the values inside the pooling kernel is the result in the output data. If average-pooling is performed in Figure 12 example, the result would be *11/9* instead of 3.

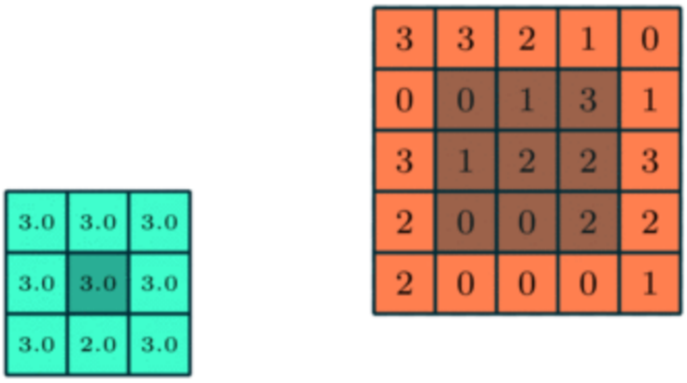

***Figure 12: Max Pooling procedure on a matrix***

### 4.2.2 Batch Normalization

Batch normalization is a method aimed at enhancing the overall accuracy and boosting the learning performance of neural networks [27]. The fundamental concept behind this approach involves normalizing the input of each layer by adjusting the standard deviation to '1' and the mean of the output activation to '0'. This normalization process is carried out on the output of each layer prior to applying the activation function, after which the resulting processed output is passed on to the subsequent layer. Advantages of batch normalization include:

- **Fast network training:** Due to extra normalization calculation, each iteration for training will be slower but the overall training will be faster.

- **Easier weight initialization:** It is a difficult process to assign weights when developing a deep neural network. Due to batch normalization, the initial starting weight sensitivity is reduced.

- **Add regularization:** Batch normalization adds minor noise to the network which enhances the learning capabilities of the neural network [28].

Figure 13 illustrates that batch normalization often helps in network training but this is not the case every time [29]. As we can see from the diagram, green and red lines have high accuracy value (y-axis) compared to the blue line.

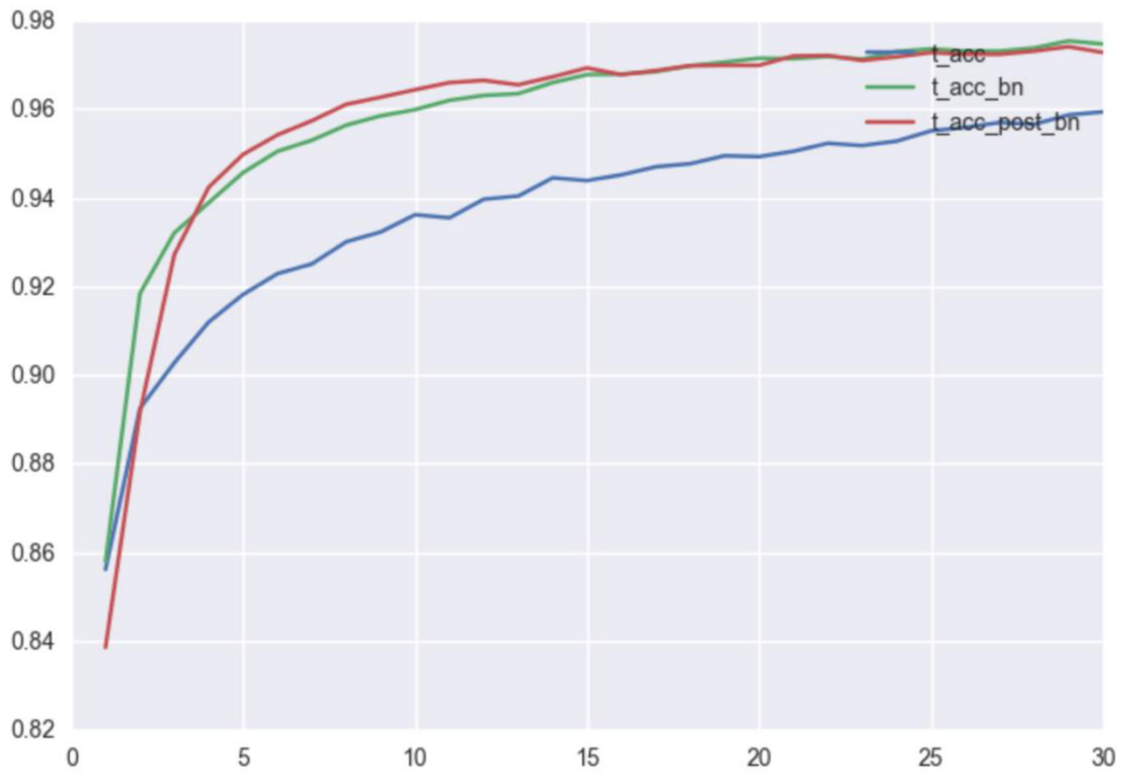

*Figure 13: Accuracy results with and without batch normalization*

### 4.2.3 Dropout Layer

Dropout technique is used to avoid overfitting [26]. Learning the training data too well is called over-fitting. There is a problem during training when a model learns the features and detail to an extent that it impacts the performance negatively on unseen data. To tackle this obstacle, dropout layer is used to avoid over-fitting. During training on a specific layer half of the neurons are switched off. This process improves generalization because it forces the layer to learn the same features with different neurons.

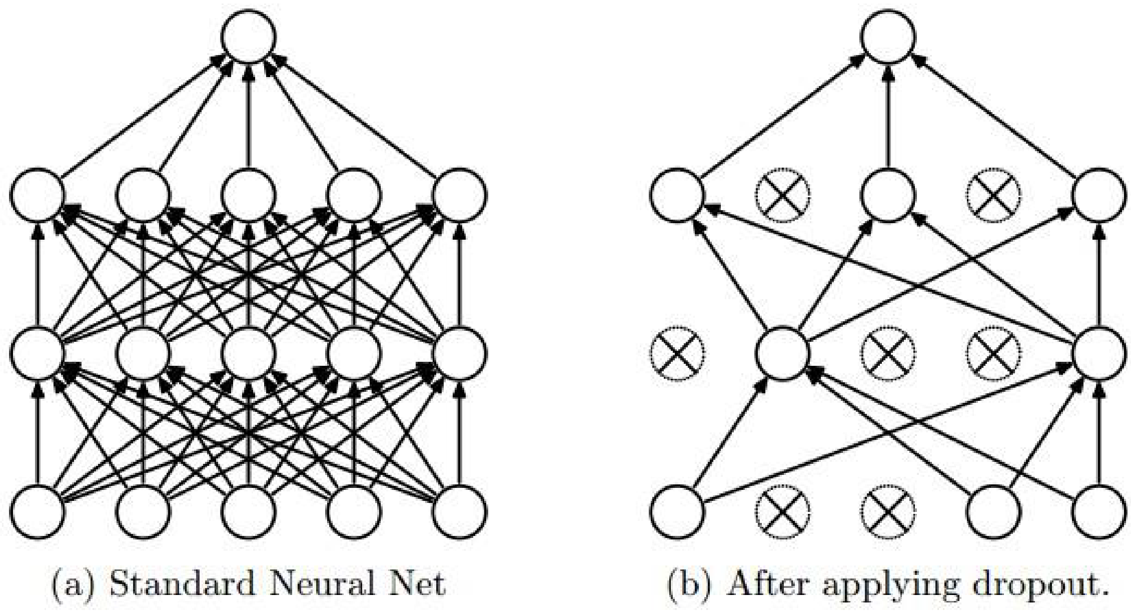

***Figure 14: Neural network before and after dropout***

In Figure 14.a, all the neurons are connected to each other whereas in Figure 14.b, there is an indication of deactivated neurons. Dropout helps models to increase accuracy and generalization better [26]. Figure 15 shows the effect of with and without dropout layer. Upon inspecting the Figure 15, we can see that classification error (y-axis) is decreasing with dropout layer.

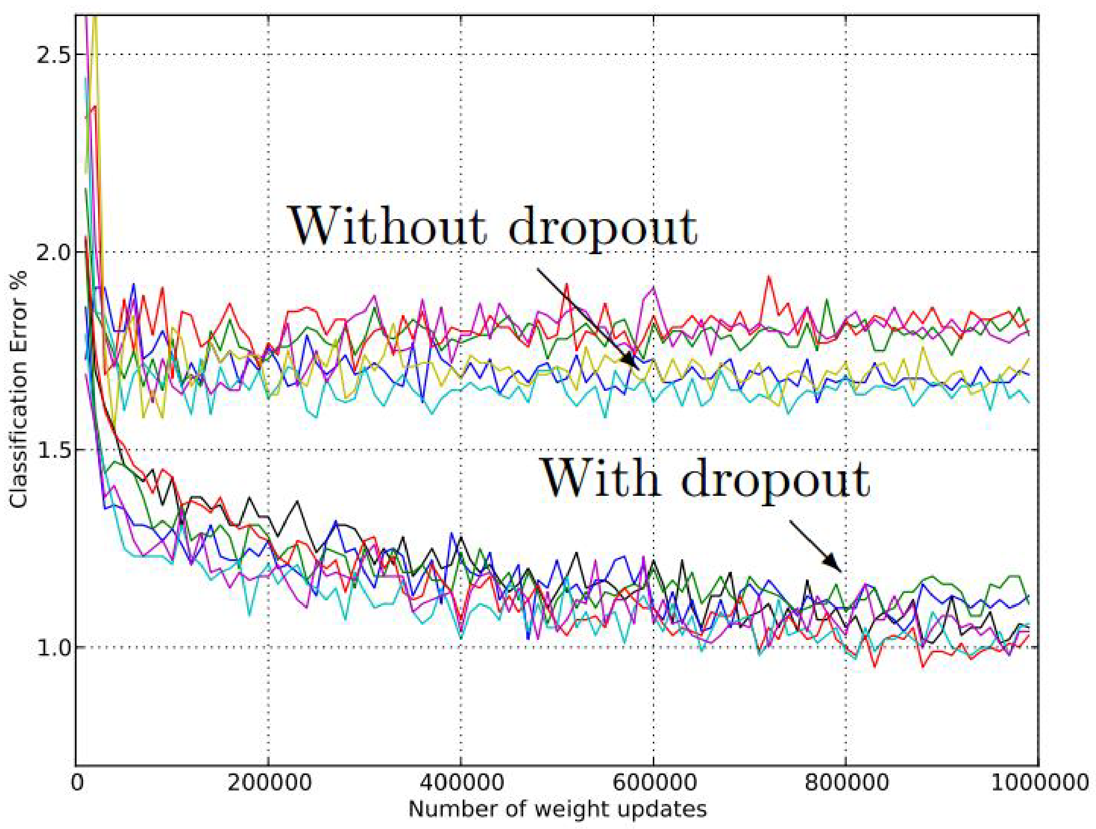

***Figure 15: Dropout effect on training of neural network***

### 4.2.4 Dense Layer

Dense layer is the simplistic layer in the neural network. A dense layer is a fully connected layer [30]. It connects each neuron from the current layer to the neurons of the next layer. Dense layer is usually followed by a non-linear activation function such as softmax or sigmoid. Figure 16 is the architecture of feed-forward network. The terminology, feed-forward, means no backward connections are allowed. In feedforward network, connections between nodes are only allowed from nodes in i-layer to i+1-layer. All arrows pointing in one direction and each neuron is densely connected to another neuron in the next layer.

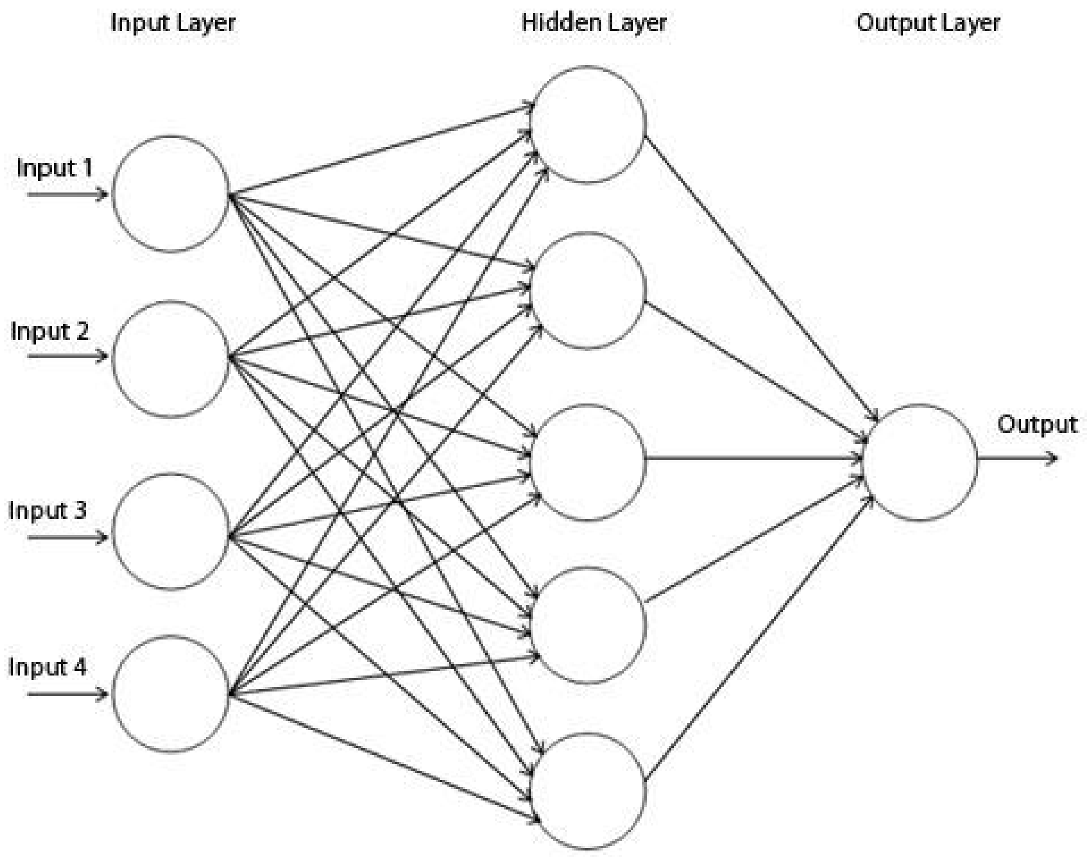

***Figure 16: Example of dense layer in neural network***

### 4.2.5 Network Topology

The layers mentioned in the above sections are used to construct a convolutional neural network for scream classifier task. Figure 6 shows the network architecture for this model. The model expects 50 melbands in each row. As the amount of training data incoming is unknown, the shape of the input tensor would be:

(AMOUNT_OF_DATA_ROWS, 50,1)

These tensors are provided to the Batch Normalization layer. From there, they are passed to the convolutional layer where it uses the *relu* activation function. The data is then passed onto pooling layer to dropout layer. From convolution to dropout, this process is repeated three times but with different parameters but the activation function, relu, stays the same over the whole network. Before passing the data to the fully connected layer, it passes through a flatten layer. Flatten layer converts *n* dimensional matrix of features into a vector that can be fed to the dense layers. Dense layers or commonly known as fully connected layers performs it computation. In the first layer, relu activation function is used whereas in the second one sigmoid is used. Finally, the third dense layer uses softmax activation function which provides a classifier result of “0” if it is a non-screaming data or “1” if it is a screaming data.

Hence, three dense layers have been used but there is no rule of thumb to find out efficient and reasonable network topology. It depends on the complexity of classification and the training data. However, one way to determine the number of layers is to add the layers until the test error is not improving.

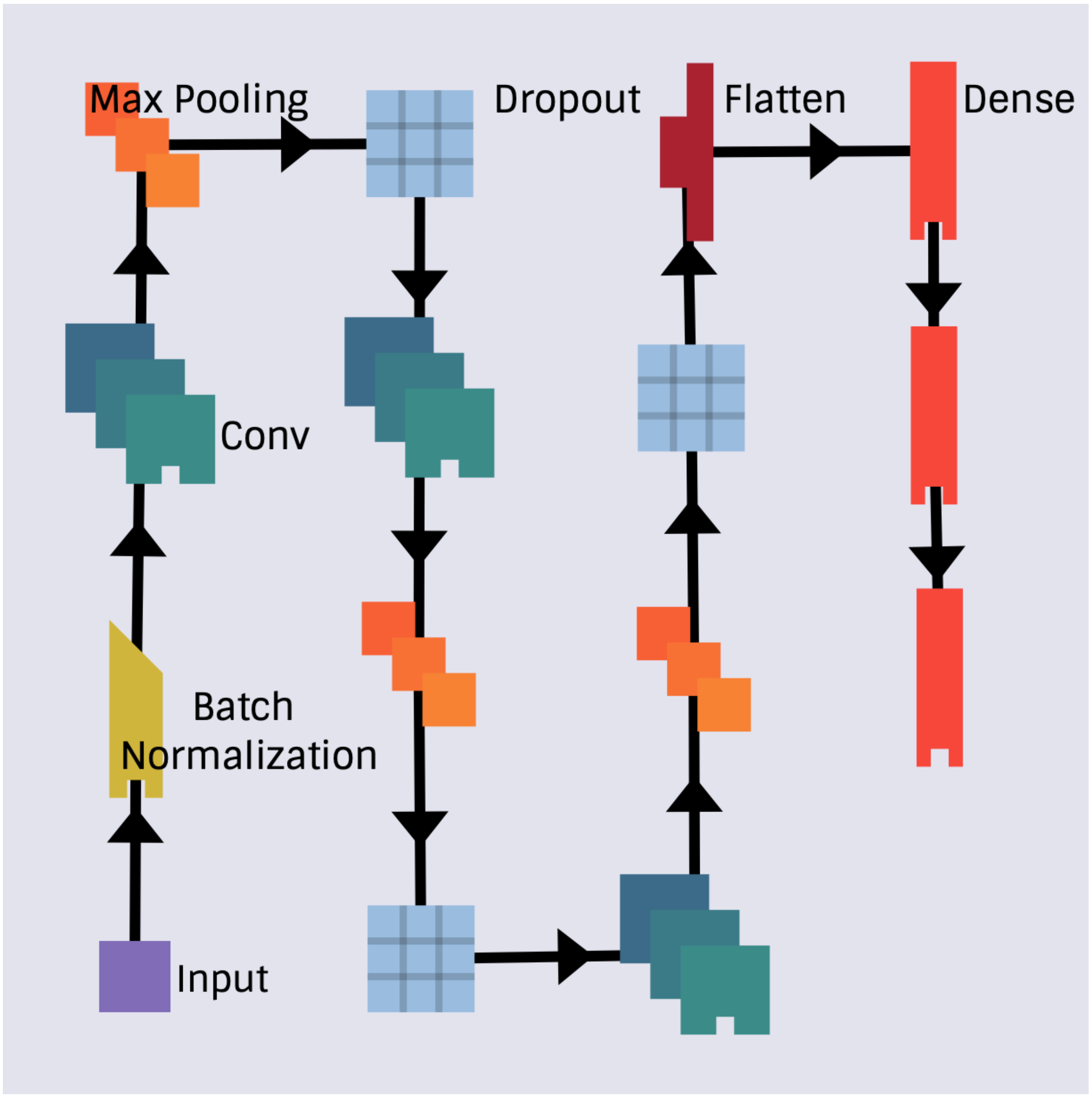

*Figure 17: Model neural network topology*

# 5. EXPERIMENTATION AND RESULTS

This section will discuss the evaluation metrics that are used to calibrate the performance of the model at an optimum level and conduct hyper-parameter tuning. Parameters that express properties of the such as its complexity or how fast it should learn. Hyper-parameter is usually fixed before training.

## 5.1 Hyper-parameters optimization

Number of hyper-parameters have noticeable effect on the computational and classification performance in the training process. Some of the hyper-parameters are interdependent on each other which makes it a challenging task to find optimum hyper-parameters. Hence, tuning hyper-parameter is a time-consuming process as they require multiple experiments and repetitions. Unfortunately, even after multiple testing, results can be inconclusive and this can be caused due to various reasons such as overfitting tuning. Some of the hyper-parameters used in this research are discussed in the following sub-sections.

### 5.1.1 Number of Neurons

The number of neurons determine the learning capacity of the neural network. Using an unnecessary number of neurons could lead to overfitting which will result in getting inaccurate result on unseen data. Furthermore, excessive number of neurons will also contribute to the poor performance in training such as taking longer training time. On the other hand, neurons less than the optimum level would result in under-fitting. Under-fitting is a term that describes a model which is not able to generalize to the new data as well have poor training statistics.

There are various methods that are used to calculate the number of neurons to be used in the hidden layer of a neural network but these rules are not authentic in way that the neurons vary from problem to problem. One way to calculate the number of neurons is:

- Number of neurons should be between the size of input and output layer
- Number of neurons should be between two-thirds of input layer
- Number of neurons should be less than double the size of the input layer

$$n < INPUT\_LAYER\_SIZE * 2$$

$n$ = Number of neurons
$INPUT\_LAYER\_SIZE$ = size of input layer

For this experiment, the following equation [31] was used to get a reasonable number of neurons for a start-up process but these changed over time during the whole research.

$$N_h = \frac{N_s}{(\alpha * \ (N_i + \ N_\sigma))}$$

$N_i = number\ of\ input\ neurons$
$N_s = number\ of\ samples\ in\ traning\ data$
$N_\sigma = number\ of\ output\ neurons$
$\alpha = arbitary\ scaling\ factor\ between\ 2\ and\ 10$

### 5.1.2 Batch Size

The purpose of the batch size is to evaluate how often the weights should be updated in the neural network. Objective function *f* is an average which is used over the whole batch of inputs. Intuitively, it can be deduced that large value of batch size the performance of the computation as well as it reduces variability of parameter over time.

The logic behind the usage of batch size can be explained in simple terms. It defines the number of input samples propagated through the neural network at one. For example, in the training process, there are 800 training samples and the batch size parameter is set to 200. The algorithm behind the logic will take 200 input samples from the dataset and trains the neural network. In the next process, it will take further 200 input samples. This process will be repeated until there is no input sample available to train.

Gradient descent is an algorithm used to calculate the coefficients or weights. It can be evaluated using Figure 1 that the mini-batch has abnormal activity when calculating the value for gradient descent. Furthermore, there is an even higher abnormal activity in the stochastic where the value of the batch size in stochastic is *1*. After doing experimentation, the default value of batch size was set to 512.

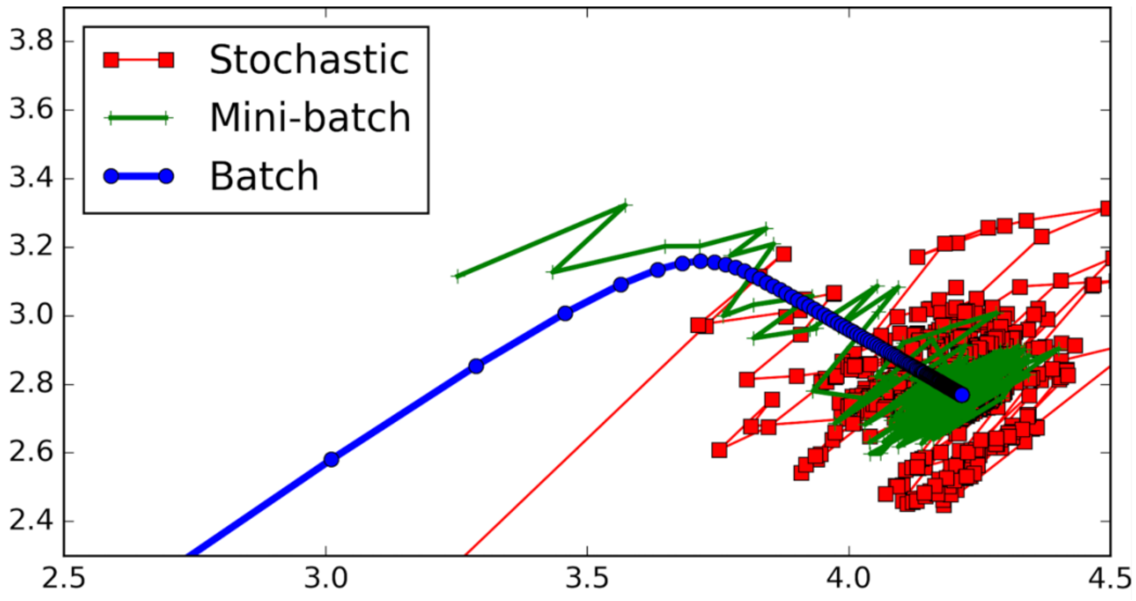

***Figure 18: Effect on gradient descent using different batch size***

As of today, we have very powerful machines that can perform computation at a great speed, using a large value batch size was not an issue in this research. The following discusses the advantages and disadvantages of using small number of batch size [32]:

1. **Advantages**:
   a. Using a small batch helps to train the neural network quickly.
   b. If there is a problem of memory in the machine, low batch size could resolve this issue as it takes less memory to train lower number of samples.
2. **Disadvantages:**
   a. It is harder to estimate the value of gradient descent using smaller batch size.

### 5.1.3 Dropout Rate

While using dropout layer which is explained in section 4.2.3, dropout rate is fixed to a value *p* for a specific layer in the neural network which will drop the number of neurons according to this particular value. Let us take an example. Dropout layer has neurons *n* = 1024 and *p* is set to 0.5. It means that 512 neurons will be dropped in this specific dropout layer. This can be verified by the following binominal equation where it is calculated that the probability of dropping 512 neurons is 0.025 [33].

$$Y = \sum_{i=1}^{1024} X_i \sim B_i(1024, 0.5)$$

$$P(Y = 512) = \binom{1024}{512} 0.5^{512} (1 - 0.5)^{(1024-512)} \approx 0.025$$

Figure 19 shows the probability of dropping fix number of neurons (1024) with various values of *p* ranging from 1 to 10. The x-axis shows the number of neurons whereas the y-axis represents the probability of dropping neurons. Again, the dropout rate depends on the application it is being used. In this specific task, 0.2 was used in all three dropout layers.

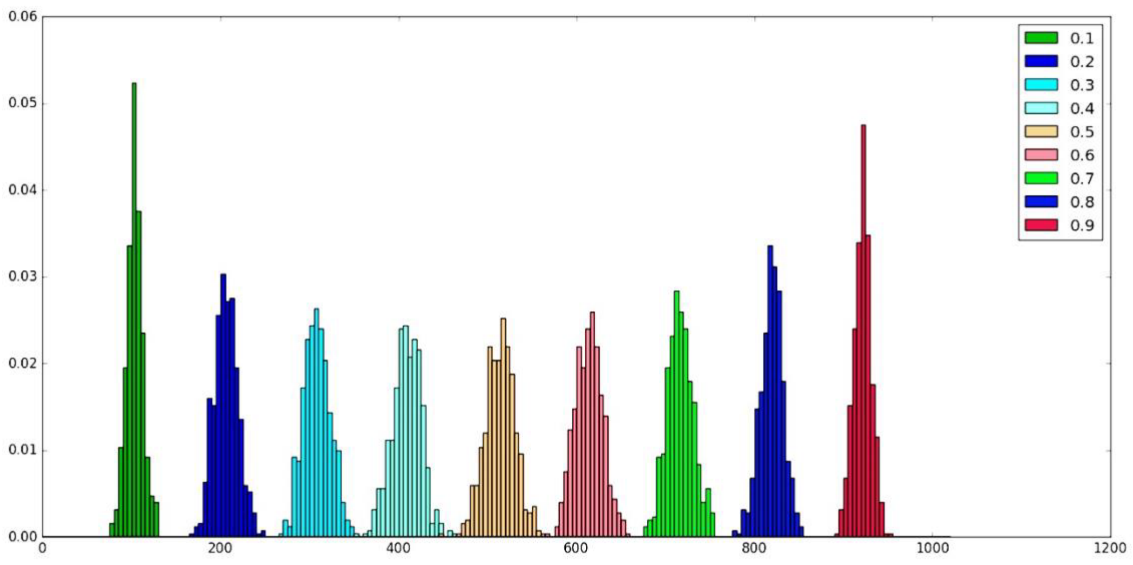

*Figure 19: Probability ratio of dropping out neurons*

## 5.2 Results

This section represents the distribution of dataset along with results obtained with different experiments conducted to have a reliable and stable scream classifier. There were various experiments conducted but the three main prominent ones are discussed below.

### 5.2.1 Dataset Distribution

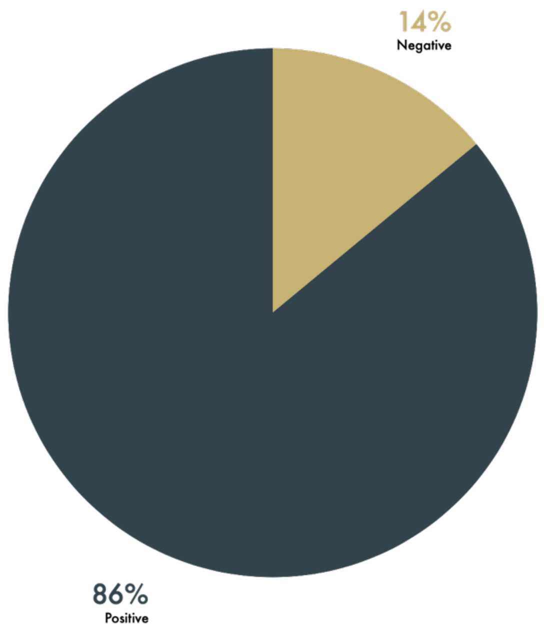

*Figure 20: Percentage of dataset distribution in negative and positive sections*

Figure 20 shows the distribution of dataset in terms of percentage. 14 percent of the dataset was dedicated to the scream audio section whereas 86 percent to the non-scream audio. The total number of samples were 60486 and in terms of numbers, there were 52022 samples as scream/positive and 8464 samples are non-scream/negative. Furthermore, 10 percent of the dataset was used as testing whereas 90 percent was used as training data.

### 5.2.2 Experiment 1

During this experiment, the model used *csaba* and *mivia* datasets. However, the following results were obtained when the trained model was executed on a test patient to detect seizures.

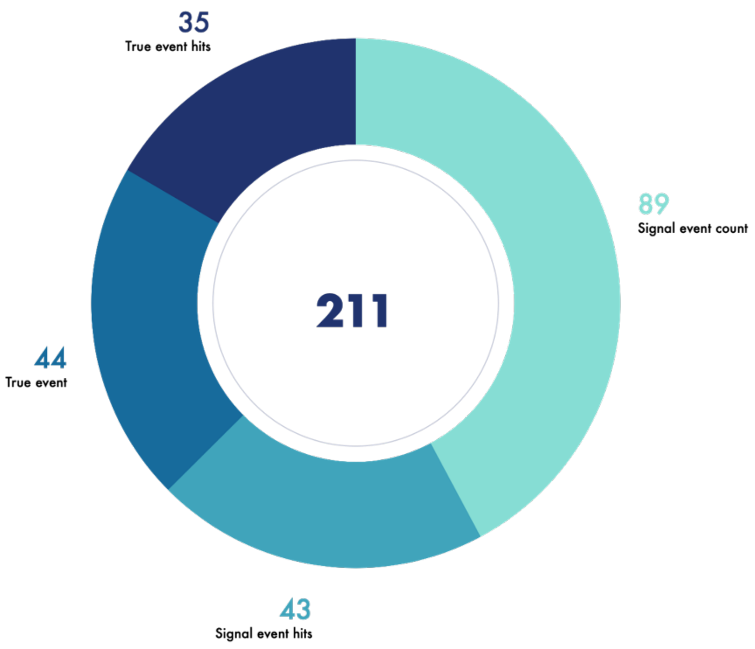

***Figure 21: Results obtained in Experiment 1 on test patient***

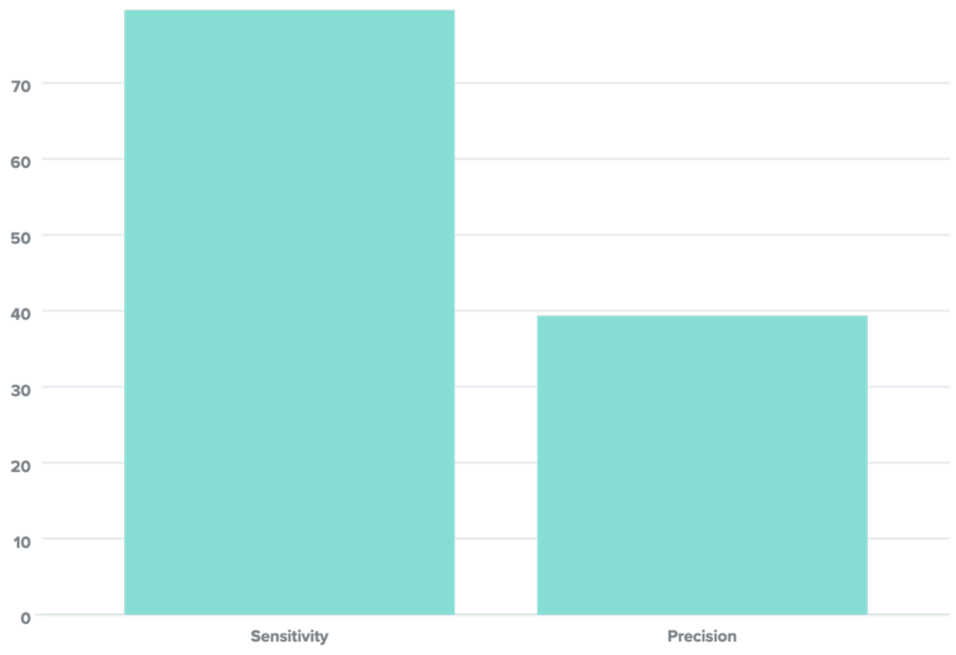

***Figure 22: Sensitivity and accuracy of model on test patient***

By observing figure 21 and figure 22, it could be implied that the model did not performed so well on the test data. Even though there is a high rate of true event hits, some of the hits were not screams. There is a very low value of precision, 39.3258 %, as it can be seen in figure 22.

During this experiment, the false positives of the patients were observed. It can be seen from the videos that the false positives were alarm clock and snoring sounds. Due to this observation, sounds such as *baby cry*, *alarm clock* and *snoring* from the csaba dataset (non-scream) were transferred to the scream dataset. This distribution and training of the new model was conducted in Experiment 3.

### 5.2.3 Experiment 2

This experiment was conducted to see how effectively the model will operate if it is trained without the mivia dataset which contained 800 audio files of scream. However, this experiment was a complete failure and the model did not detect a single scream on full night videos of the test patient.

### 5.2.4 Experiment 3

As it was deducted in experiment 1 that the false positives in the test patient were snoring and alarm clock sounds, a new model was trained with a new dataset. *Alarm clock sound, baby cry* and *snoring* sounds were transferred to the scream audio dataset. Intuitively this would mean that the percentage of scream and non-scream dataset have some impact on it with a minor percentage increase in scream dataset where as a minor decrease in non-scream dataset.

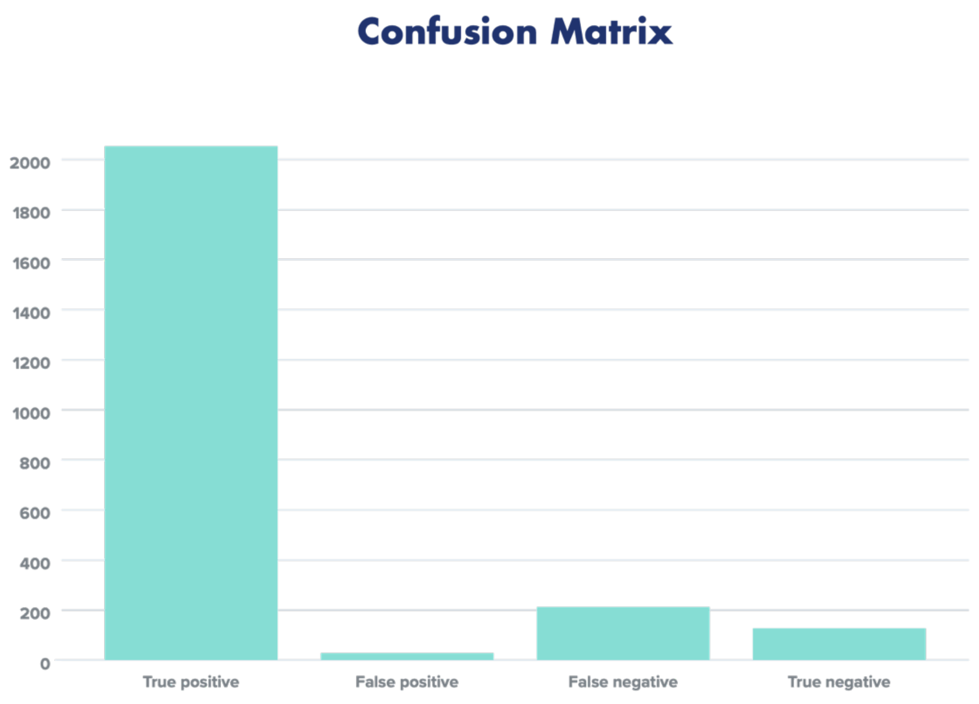

*Figure 23: Confusion matrix of the trained model on new dataset*

It can be seen from Figure 23 that the model performed relatively well on the new dataset. There is a huge decrease in the false positive values which account for only 1.35 percent or 28 samples.

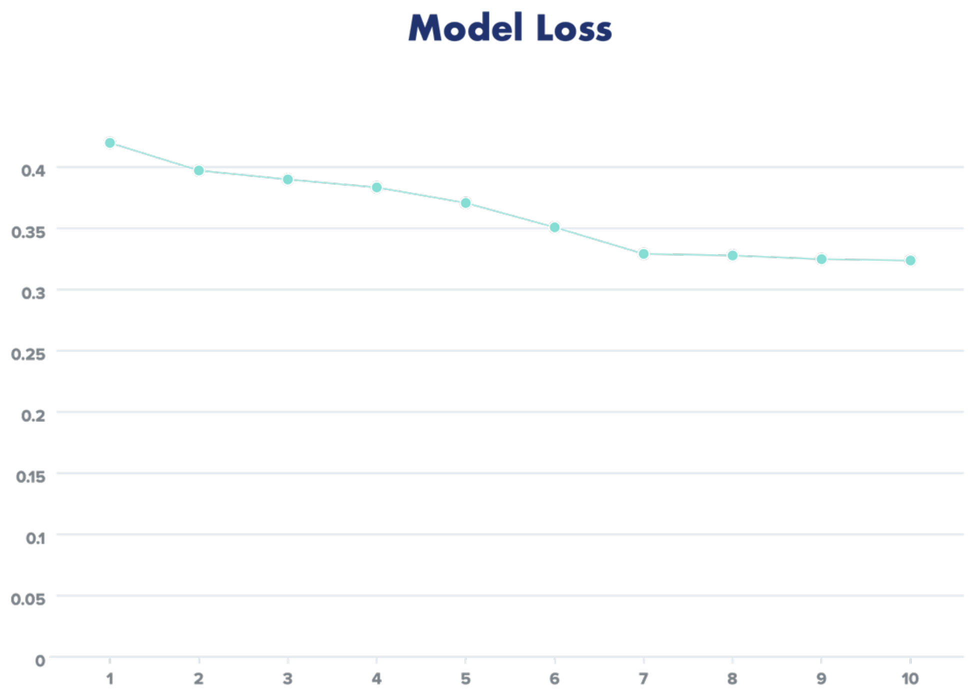

*Figure 24: Number of epochs against the loss rate*

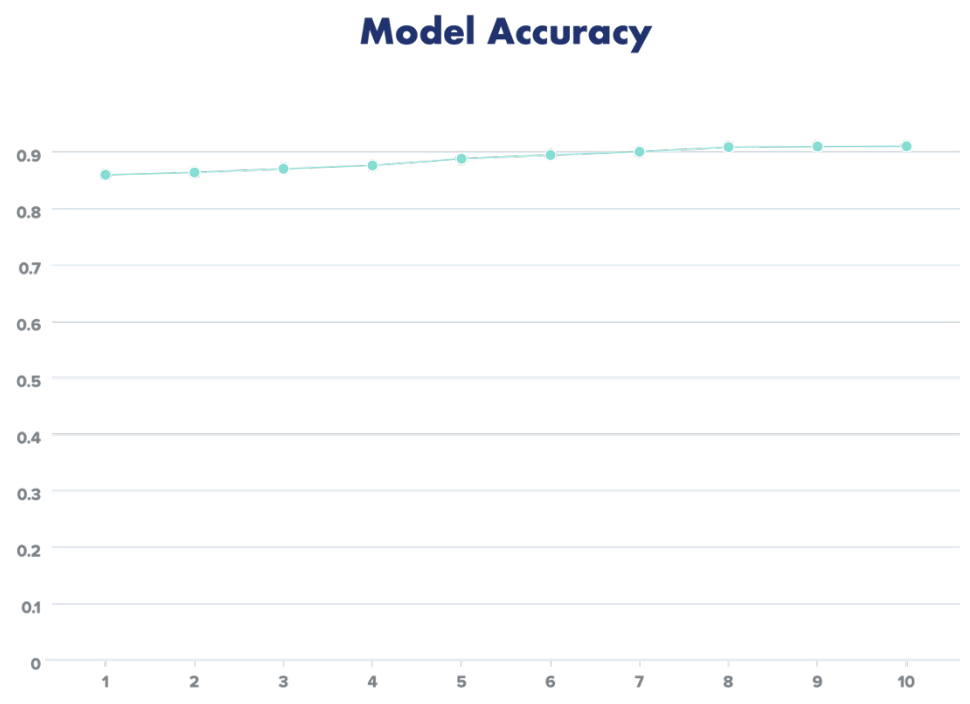

***Figure 25: Number of epochs against the model accuracy***

From Figure 25, it is clearly visible that the accuracy of the model increased with decreasing rate. The model achieved an accuracy of 0.9015 at the 10th epoch. Whereas the Figure 7 supports the accuracy statistics. The loss rate also decreased with each epoch achieving stability between 7 and 10 epochs.

Figure 26 and Figure 27 shows immense improvements on testing the model on the patient. There was a huge increase in precision from 39.325% in Experiment 1 to 84.3750 in the latest experiment. The inclusion of the *baby cry* sounds helped a lot in increase of accuracy as well as stability in the new model. Upon inspection, it was concluded that the melbands of the *baby cry* and *scream* have similarities which helped in detecting screams on the test patient data.

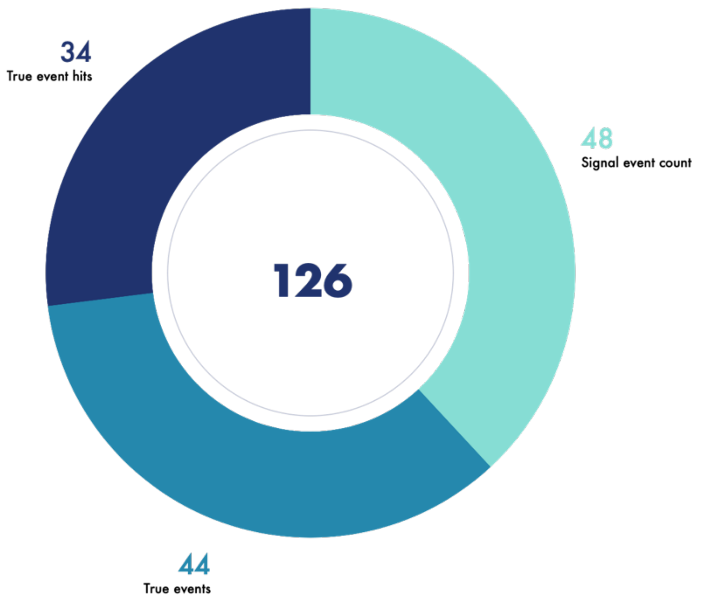

*Figure 26: Results obtained in Experiment 3 on test patient*

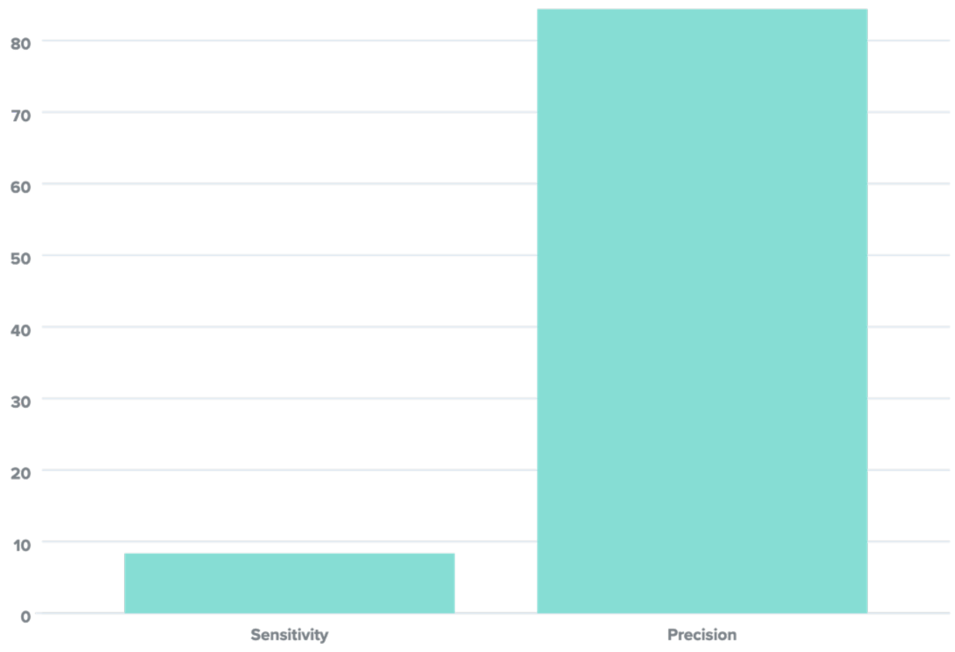

*Figure 27: Sensitivity and accuracy of model on test patient*

Table 1 shows the performance comparison between 2 main experiments conducted in this task to develop a scream classifier.

| Model | Accuracy | Scream dataset | Non-scream dataset |
|---|---|---|---|
| **Experiment 1** | 39.3258 | Mivia | Csaba |
| **Experiment 3** | 84.3750 | Mivia + (baby cry, alarm sounds and snoring) | Csaba – (baby cry, alarm sounds and snoring) |

***Table 1: Comparison of 2 different models***

# 6. CONCLUSION

The difficulty of making a classifier depends on various factor; one of the major factors is the number of samples available to train a neural network. In this thesis, the capabilities of neural network have been demonstrated to have a scream classifier. The model was trained on three different types of datasets:

- Csaba (non-scream) and Mivia (scream) [Dataset 1]
- Csaba (non-scream), Mivia (scream), baby cry (scream), alarm clock (scream), snoring (scream) [Dataset 2]

The model did not perform well on the dataset 1 as the false positives consisted of sounds such as snoring and alarm clock. Hence, these sounds were transferred from Csaba dataset to scream dataset as shown in dataset 2. However, there were other reasons for the model not performing too well which might include dropout ratio, number of neurons or even number of layers as there is no rule of thumb to figure out the optimum level. The model architecture was designed to take 30 rows of input data but this row could be changed every time a new model is been trained. Changing the number of rows of input data had minor effects on the results. Upon inspection, the optimum number of rows was found to be 30 which gave the best results on the test patient. If the number of rows was increased, the model produced invaluable results due to over-fitting. During experiment, neural network architecture size was increase and decreased to test the performance as the starting experiments were expensive in terms of time and resource consuming. Furthermore, variations of batch size and dropout value also effected the result. If there is a higher value of batch size, it was noticeable increase in the variance over each epoch. Ultimately, deeper networks need large amount of training and testing data to achieve good results along with having powerful CPU and GPU to perform faster computation.

Further research in this thesis could place by taking the following steps:

- Tuning of hyper-parameter will have significant improvement on the results.
- Using Google Coral to increase the computation speed on the testing data as the testing data is full night videos of patients.
- Testing the dataset on a regressor neural network instead of a classifier. A regressor will result in a numerical value ranging in between 0-1. Some experiments are already done with adding some convolutional and pooling layers to the existing neural network used in this thesis along with using sigmoid activation function instead of softmax.
- Testing the trained model on more than one test patient. As only one test data could not guarantee that the trained model is a perfect one, it is better to test it on more than one test patient.

# REFERENCES

[1] Danna Voth," Using AI to detect Breast Cancer", IEEE Computer Society.

[2] Chieh-Chen Wu, Wen-Chun Yeh, Wen-Ding Hsu, Md. Mohaimenul Islam, Phung Anh (Alex) Nguyen, Tahmina Nasrin Poly, Yao-Chin Wang, Hsuan-Chia Yang and Yu-Chuan (Jack) Li, "Prediction of fatty liver disease using machine learning algorithms", Elsevier.

[3] Shawn Hershey, Sourish Chaudhuri, Daniel P. W. Ellis, Jort F. Gemmeke, Aren Jansen, R. Channing Moore, Manoj Plakal, Devin Platt, Rif A. Saurous, Bryan Seybold, Malcolm Slaney, Ron J. Weiss, Kevin Wilson, "CNN Architectures for Large-Scale Audio Classification", International Conference on Acoustics, Speech and Signal Processing (ICASSP) - IEEE (2017)

[4] Jacek Grekow, "Audio features dedicated to the detection and tracking of arousal and valence in musical compositions", Journal of Information and Telecommunication

[5] Pasquale Foggia, "Audio Surveillance of Roads: A System for Detecting Anomalous Sounds", University of Salerno

[6] Benoit B.Mandelbrot, "A Fast Fractional Gaussian Noise Generator", Water Resources Research

[7] Laffitte, Pierre; Sodoyer, David; Tatkeu, Charles ; Girin, Laurent, "Deep neural network for automatic detection of scream and shouted speech in subway train", 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), March 2016

[8] Lei, Baiying ; Mak, Man-Wai, "Robust scream sound detection via sound event partitioning", Multimedia Tools and Applications, 2016, Vol.75(11)

[9] S. Chu, S. Narayanan and C.-C.J. Kuo, "Environmental sound recognition with time-frequency audio features", IEEE Trans. on Audio Speech and Language Processing, vol. 17, no. 6, pp. 1142-1158, Aug. 2009.

[10] M. Crocco, M. Cristani, A. Trucco and V. Murino, Audio surveillance: a systematic review, Oct. 2014.

[11] J. Dennis, D. T. Huy and S. C. Eng, "Image feature representation of the subband power distribution for robust sound event classification", IEEE Trans. on Audio Speech and Language Processing, vol. 21, no. 2, pp. 367-377, Feb. 2013.

[12] M. Fernández-Delgado, E. Cernadas, S. Barro and D., "Do we need hundreds of classifiers to solve real world classification?", J. of Machine Learning Research, vol. 15, pp. 3133-3181, 2014.

[13] I. McLoughlin, H. Zhang, Z. Xie, Y. Song and W. Xiao, "Robust sound event classification using deep neural networks", IEEE/ACM Trans. on Audio Speech and Language Processing, vol. 23, no. 3, pp. 540-552, Mar. 2015.

[14] A. Diment, E. Cakir, T. Heittola and T. Virtanen, "Automatic recognition of environmental sound events using all-pole group delay features", European Signal Processing Conference, pp. 734-738, Aug. 30 - Sept. 4 2015.

[15] X. Wu, H. Gong, P. Chen, Z. Zhong and Y. Xu, "Surveillance robot utilizing video and audio information", J. of Intelligent and Robotic Systems, vol. 55, no. 4, pp. 403-421, 2009.

[16] J. Pohjalainen, T. Raitio and P. Alku, "Detection of shouted speech in the presence of ambient noise", Interspeech, pp. 2621-2624, Aug. 28–31 2011.

[17] W. Huang, T.-K. Chiew, H. Li, T. S. Kok and J. Biswas, "Scream detection for home applications", IEEE Conference on Industrial Electronics and Applications, pp. 2115-2120, 2010.

[18] B. Lei and M. Mak, "Sound-event partitioning and feature normalization for robust sound-event detection", Int. Conf. on Digital Signal. Processing, pp. 389-394, Aug. 20-23 2014.

[19] S. Ntalampiras, I. Potamitis and N. Fakotakis, "On acoustic surveillance of hazardous situations", *IEEE Int. Conf. on Acoustics Speech and Signal Processing*, pp. 165-168, Apr. 19-24 2009.

[20] Y. Lee, D. Han and H. Ko, "Acoustic Signal Based Abnormal Event Detection in Indoor Environment using Multiclass Adaboost", *IEEE Trans. on Consumer Electronics*, vol. 59, no. 3, pp. 615-622, Aug. 2013.

[21] J. Wang, C. Lin, B. Chen and M. Tsai, "Gabor-based nonuniform scale-frequency map for environmental sound classification in home automation", *IEEE Trans. on Automation Science and Engineering*, vol. 11, no. 2, pp. 607-613, Apr. 2014.

[22] J. M. Nazzal, I. M. El-Emary and S. A. Najim, “Multilayer Perceptron Neural Network (MLPs) For Analyzing the Properties of Jordan Oil Shale,” IDOSI Publications, King Saudi Arabia, 2008.

[23] J. Yosinski, J. Clune and Y. Bengio, “How transferable are features in deep neural networks?” Cornell University.

[24] R. Kishore and T. Kaur, “Backpropagation Algorithm: An Artificial Neural Network Approach for Pattern Recognition,” 2012.

[25] G. F. D. Duff and D. Naylor, “Differential Equations of Applied Mathematics,” 1996.

[26] N. Srivastava, G. Hinton and A. Krizhevsky, “A Simple Way to Prevent Neural Networks from Overfitting,” University of Toronto, Toronto, Ontario, 2014.

[27] S. Ioffe and S. Ioffe, “Batch Normalization: Accelerating Deep Network Training b y Reducing Internal Covariate Shift,” Google Inc., USA.

[28] J. Collis, “Glossary of Deep Learning: Batch Normalisation,” Jun 27.

[29] Rui and Shu, “A Gentle guide to using batch normalization,” 2016.

[30] N. E. West and T. J. O'Shea, "Deep Architectures for Modulation Recognition," U.S. Naval Research Laboratory and Virginia Polytechnic Institute and State University, Washington, D.C.

[31] S. Lawrence, L. Giles and A. C. Tsoi, "What Size Neural Network Gives Optimal Generalization? Convergence Properties of Backpropagation," University of Maryland, 1996.

[32] P. Goyal, P. Dollar, R. Girshick, P. Noordhuis and A. Tulloch, "Accurate, Large Minibatch SGD," Facebook.

[33] P. Baldi and P. Sadowski, "The Dropout Learning Algorithm," University of California, Irvine.

[34] Waleed Yousef, "Matlab vs. OpenCV: A Comparative Study of Different Machine Learning Algorithms", Cornell University

[35] Sulistijono, I.A. ; Urrosyda, R.C. ; Darojah, Z, "Mel-Frequency Cepstral Coefficient (MFCC) for music feature extraction for the dancing robot movement decision", Lecture Notes in Computer Science

[36] Mivia Research Lab, "Mivia dataset for scream sound", University of Salerno

[37] Csava Kertéz, "Common sounds in Bedrooms (CSIBE) Corpora for Sound Event Recognition of Domestic Robots", University of Tampere

[38] Butterfield, Andrew ; Ngondi, Gerard Ekembe ; Kerr, Anne, "Gaussian Noise", A dictionary of Computer Science

[39] Kassam, Saleem A.; Thomas, John B, "Sound detection in non-gaussian noise", New York Springer

[40] IEEE, "IEEE Standard for Artificial Intelligence and Expert System Tie to Automatic Test Equipment (AI-ESTATE): Overview and Architecture"

[41] Sichkar, V; Kolyubin, S, "Effects of Various Dimensions Convolutional Layer Filters on Traffic Sign Classification Accuracy"

[42] Alex Krizhevsky, Ilya Sutskever, Geoffrey E Hinton, "Imagenet classification with deep convolutional neural networks," in Advances in Neural Information Processing Systems (NIPS) 2012, pp. 1097–1105.

[43] Alex Graves, Abdel-rahman Mohamed, Geoffrey Hinton, "Speech recognition with deep recurrent neural networks," in IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) 2013, pp. 6645–6649.

[44] Xavier Glorot, Yoshua Bengio, "Understanding the difficulty of training deep feedforward neural networks," in International conference on artificial intelligence and statistics 2010, pp. 249–256.

[45] Bernhard Scholkopf, Alexander J Smola, “Learning with kernels: support vector machines, regularization, optimization, and beyond”. 2001, MIT press.

[46] Yoshua Bengio, Olivier Delalleau, Clarence Simard, “Decision trees do not generalize to new variations,” in Computational Intelligence, vol. 26, no. 4, pp. 449–467, 2010.

[47] Yoshua Bengio, Yann LeCun, “Scaling learning algorithms towards AI,” in Large-scale kernel machines, vol. 34, no. 5, 2007.