Diploma in Engineering Science

Product Design & Development (ESFYP)

Final Report (50%)

Project ID: PS06

Project Title: Depression Detection Using Speech Analysis (Mobile App)

By:

Sim Yu Hui, Kellie (S10163148E)

Tan Hong Ray (S10177638K)

Supervisor:

Dr Harry Nguyen

School of Computing Science

University of Glasgow Singapore

Co-Supervisor:

Pham The Hanh

School of Engineering

Ngee Ann Polytechnic

Abstract

In recent years, depressive disorders have become a prominent cause of disability and burden worldwide. In Singapore, one in five elderly aged 75 and above show signs of depression. Real-time early detection of depression using mobile-based devices is promising, yet challenging due to its complex clinical characterisation. These characteristics include behavioural, emotional and cognitive symptoms. Currently, applications for self-help are out on the market but do not offer depression detection features.

Previously, we have introduced an Android application that is able to detect depression using real-time audio/speech analysis, and deep neural networks. This project aims to create an AI solution for detecting early signs of depression using speech features such as intensity decay, prosodic abnormalities and phonetic errors. Audio recordings with associated depression indications are provided for data training by the DAIC-WOZ Database.

In this report, our aim is to improve the way we detect depression and achieve better accuracy in our prediction. We leveraged on existing deep learning, denoising and audio processing technologies for our program. We also will further improve upon the android application we have developed. Our aim is to have an application to not only detect depression, but also to help the people affected by it.

Table of Contents

1. Introduction

Depression is a mood disorder characterized by physical (e.g., appetite disturbance, insomnia, loss of energy, psychomotor agitation), emotional (e.g., depressed mood, anhedonia), cognitive (e.g., low self-esteem, diminished ability to think, concentrate and make decisions) and behavioural symptoms (e.g., social isolation) "causing significant distress or severely impacting social, occupational or other important life areas" [1]. It is one of the most common mental disorders in the world with more than 300 million patients in 2015 [1]), it is the second cause of disability after ischaemia and it is the most important suicide risk factor among elderly people [1]. As a result of the above, depression requires prolonged and expensive medical treatments that result into a significant economic burden for both patients and society [1]. The development of automatic approaches for the detection of depression can help to reduce such costs by supporting the activity of clinicians and, in particular, by reducing the time they need to diagnose a patient as depressed.

Depressive disorders are a prominent cause of disability and burden worldwide. According to a 2010 Singapore Mental Health Study (SMHS), one in 10 Singaporeans will suffer from mental illness in their lifetime [2]. Not only that, studies have shown that many of these sufferers do not seek help [2]. According to Prof K Saktu, the Director of Medical Services at the Ministry of Health, "mental illness is often neglected due to a lack of understanding, misconceptions, discrimination and stigma of the disease." [3]. Due to this stigma and discrimination, the mentally ill tend to avoid seeing a psychiatrist for proper checkups and diagnosis. As a result, they end up suffering in silence and in severe cases, end up having suicidal thoughts. Currently, there are various technologies and ways to help the mentally ill. Some applications on the market offer chatbots, daily mood trackers or diaries and self-harm prevention methods through distraction. These applications are a good gateway for self-help, but are unable to detect or diagnose mental illnesses. Current methods of detecting for depression include consulting a psychiatrist and online mood tests such as M3 and PsychCentral. The Institute of Mental Health, a psychiatric hospital under the MOH, provides services for patients and their family.

Hospital-based services include satellite clinics and the Sunshine Wing [4]. Equipped with elderly and dementia-friendly features and facilities, it provides a supportive environment for rehabilitation [4]. There are also community-based services such as Aged Psychiatry Community Assessment Treatment Service (APCATS) and Occupational Therapy: Activities, Vocation and Empowerment (OcTAVE) Day Rehabilitation Centres [4]. APCATS provides assessment and treatment for mentally ill homebound or frail elderly patients [4]. OcTAVE provides psycho-social rehabilitation programmes, sheltered workshops and group activities for psychiatric outpatients [4]. These programmes focus on community-living skills, vocational training and self-care to help patients to integrate in the community [4].

However, rather than only focusing on post-diagnosis treatment, early detection is also crucial to ensuring the recovery of the patients. The government is also focusing on early detection. Targeted outreach efforts such as screening for depression in obstetric wards or amongst patients with chronic diseases have been set up in hospitals and the community [5]. The Community Health Assessment Team (CHAT) maintains an online portal where youths can access resources or make appointments for assessments [6]. The Response, Early Intervention, Assessment in Community Mental Health (REACH) programme supports schools in reaching out to students who need emotional support [6]. Training is provided to schools to strengthen their ability to identify, manage, and if necessary, refer at-risk children to specialists [6].

Last year, we introduced an application that utilised and combined various modern technologies such as logistics regression to detect depression in real-time. In this report, we present an improved version of the application that detects depression based on a user’s non-verbal vocal features. These features will be used in deep learning techniques, which will also be explored in this report. Similar to last year's, this application will not cure depression fully. Rather, it serves as a way out for the mentally ill who face stigma and discrimination and do not want to seek a professional diagnosis.

2. Project Outline and Objectives

2.1 Project Objectives

In this section, we will be sharing our learning objectives and end-goals of the project.

Through this project, we aim to

- Apply deep learning models to determine the probability of a user being depressed
- Provide an accessible and accurate method for users or psychologists to detect depression using non-verbal vocal features
- Enhance technology used for early detection of depression or other mental illnesses so as to allow for early detection of these illnesses
- Provide a visual representation of audio signal through the application, and if possible, highlight portions that link to depression being detected

Applications on the current market only offer ways to help cope with depression or other mental illnesses. These applications do not allow for the detection and diagnosis of depression. With the development of this application, there could be many meaningful advancements made in the medical industry with regards to the diagnosis of mental illnesses.

In order to achieve these objectives and goals, we will have to do research on the existing technologies and relevant software. More details as well as our project timeline can be found in the next few sections.

2.2 Gantt Chart

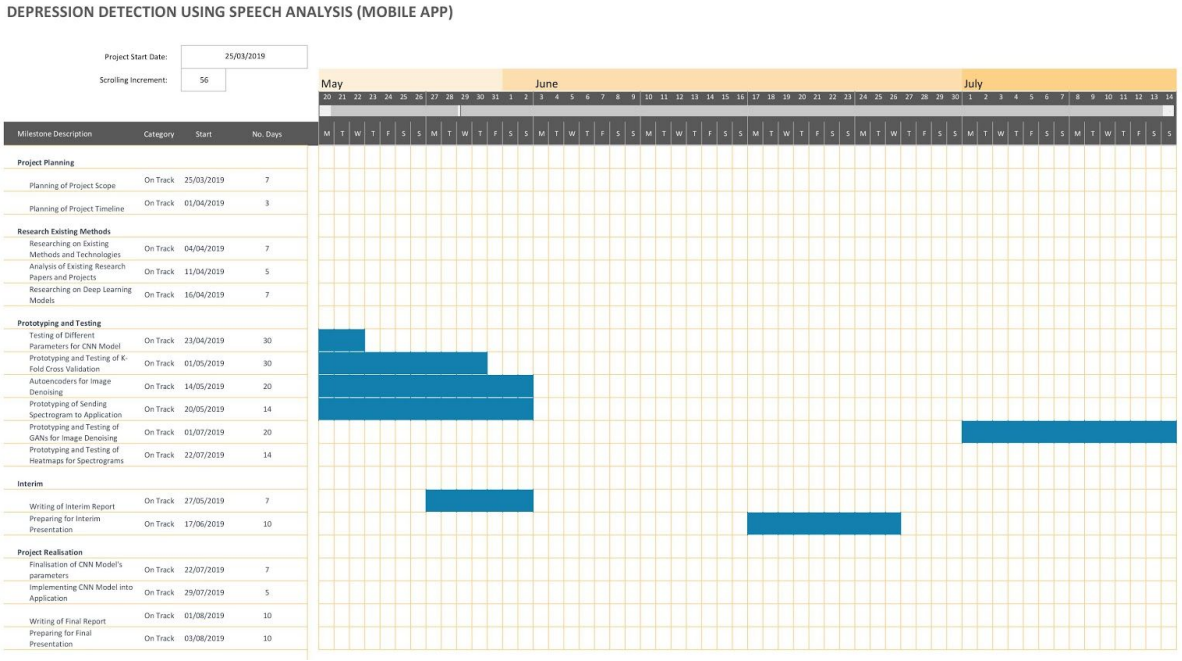


Figure 1: Gantt Chart (1/3)

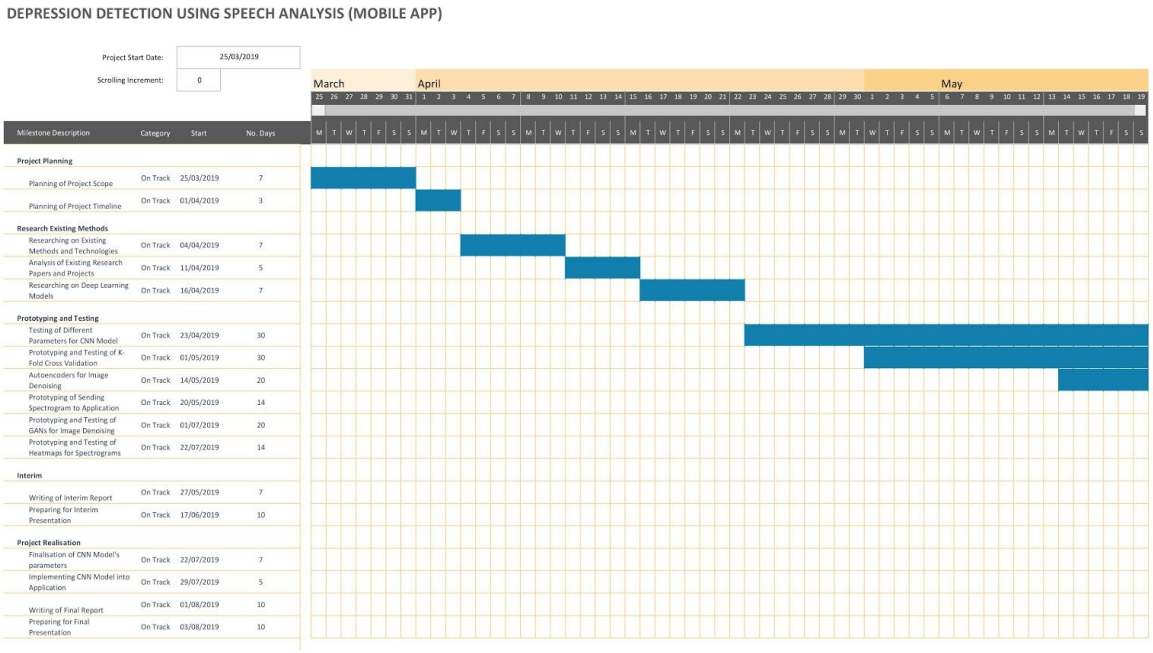


Figure 2: Gantt Chart (2/3)

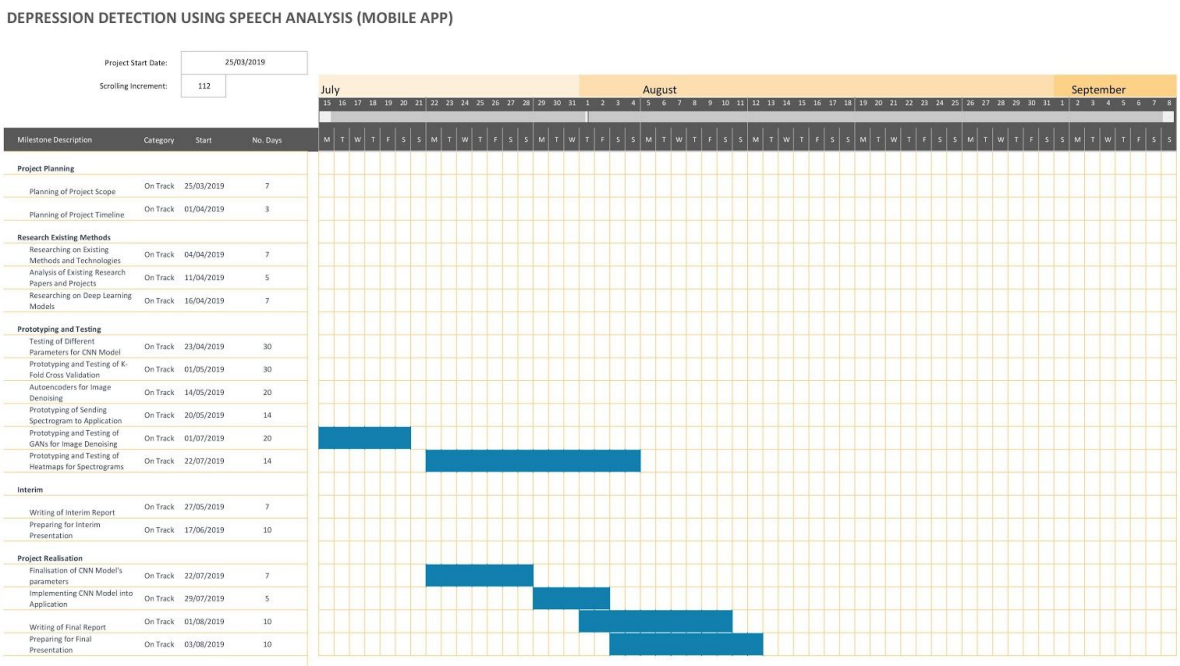


Figure 3: Gantt Chart (3/3)

3. Literature Research

3.1 Research on Existing Techniques

3.1.1 Introduction

Medical professionals currently use the Patient Health Questionnaire (PHQ) designed by the American Psychological Association to diagnose depression [7]. The brief form asks patients about their level of interest in regular activities, appetite and eating habits, ability to concentrate, and other queries designed to detect depression [7]. The questions are based on criteria from the Diagnostic and Statistical Manual of Mental Disorders (DSM) IV. Once diagnosed, depression can be treated. However, obstacles such as cost, mobility, and motivation may prevent depressed people from seeking the help they need [7].

In this section, we will explore 3 existing methods of detecting depression that have been made possible using technological advancements - Long-Short Term Memory Neural Networks, DepressionDetect, and a multi-modal method of Measuring Depression Symptom Severity from Spoken Language and 3D Facial Expressions. These methods are just 3 of many methods that have been published, but were chosen as they were the most relevant to our project.

3.1.2 Long-Short Term Memory Neural Network

Recently, researchers at the Computer Science and Artificial Intelligence Laboratory (CSAIL), are conducting a study aimed at detecting depression using a Long-Short Term Memory (LSTM) neural network to model audio and text transcriptions extracted from The Distress Analysis Interview Corpus (DAIC) [7]. This model may be able to evaluate mental health via cues found in speech patterns, such as monotone pronunciation and longer pauses between words [7]. A new model may be able to evaluate mental health via cues found in speech patterns, such as monotone pronunciation and longer pauses between words [7]. The DAIC consists of interactions between human subjects and a virtual agent [7]. 142 individuals were screened for depression through a human-controlled virtual agent [7]. The agent asked questions such as

‘How are you?’, and ‘Do you consider yourself to be an introvert?’ It gave feedback to the subject using natural responses like, ‘I see,’ and ‘Tell me more about that [7].’

The multi-modal method comprising of both audio and text had a subject-level mean absolute error (MAE) value of 4.97 and subject-level root mean squared error (RMSE) of 6.27 [7].

3.1.3 DepressionDetect

DepressionDetect uses the The Distress Analysis Corpus - Wizard of Oz (DAIC-WOZ) Dataset, in the aims of lowering the barrier of entry in seeking help for potential mental illnesses and supporting the diagnosis of mental health professionals [8]. Experiments were done using Conventional Neural Networks (CNN) to identify non-verbal speech features signifying depression [8]. More details about CNNs can be found in Section 3.3. Focusing on class imbalance and data representation/feature extraction, the model achieved an accuracy of 64.3% eventually [8].

Class imbalance occurred as the number of non-depressed subjects is about 4 times larger than that of depressed ones, hence potentially resulting in a classification “non-depressed” bias [8]. Furthermore, additional bias could have occurred due to the duration of the interviews [8]. To address these issues, each participant’s segmented spectrograms were cropped into 4 second slices. The participants were then sampled randomly in 50/50 proportion from each class. A fixed number of slices were sampled from each of these participants to ensure that the CNN had an equal interview duration per participant [8].

In terms of feature extraction, speech stimuli was visually represented as a spectrogram. A spectrogram is a visual representation of sound that displays the amplitude of the frequency components of a signal over time. It is able to maintain a high level of detail, including noise in the signal [8].

In this experiment, predictions were made on 4 second spectrograms. By using a majority vote of 40 predictions per participant to label the participant as depressed or not depressed, an accuracy of 55.5% was achieved [8].

3.1.4 Measuring Depression Symptom Severity from Spoken Language and 3D Facial Expressions

In [9], a machine learning method for measuring depressive symptom severity from de-identified multi-modal data was proposed [9]. The input to this model was audio, 3D video of facial keypoints, and a text transcription of a patient speaking during a clinical interview [9]. The output of this model was either a PHQ score or classification label indicating major depressive disorder. This method leveraged a causal convolutional network (C-CNN) to “summarize" sentences into a single embedding, which is then used to predict depressive symptom severity [9].

This model demonstrated an average error of 3.67 points (15.3% relative) on the clinically-validated Patient Health Questionnaire (PHQ) scale [9]. For detecting major depressive disorder, the model demonstrated 83.3% sensitivity and 82.6% specificity [9].

3.1.5 Conclusion

As mentioned in our introduction, there have been other methods for detecting depression that were published in recent years. However, based on our research, we found that not much development has been made in using cross validation to improve CNN models and denoising of audio and images prior to generating the model.

In the following sections, we will present our method, which provides an improvement in accuracy from existing methods.

3.2 Keras and Tensorflow

3.2.1 Keras

3.2.1.1 Introduction to Keras

Keras is one of the deep learning frameworks used. It offers consistent & simple APIs, minimizes the number of user actions required for common use cases, and provides clear and actionable feedback upon user error. Keras API is the official frontend of TensorFlow, via the tf.keras module [10]. Keras is a model-level library, providing high-level building blocks for developing deep learning models [11]. It does not handle low-level operations such as tensor products, convolutions and so on itself [11]. Instead, it relies on a specialized, well optimized tensor manipulation library to do so, serving as the "backend engine" of Keras [11]. At this time, Keras has three backend implementations available: the TensorFlow backend (an open-source symbolic tensor manipulation framework developed by Google), the Theano backend (an open-source symbolic tensor manipulation framework developed by LISA Lab at Université de Montréal), and the CNTK backend (an open-source toolkit for deep learning developed by Microsoft) [10].

3.2.1.2 Models

There are two main types of models available in Keras, namely the Sequential model and the Model class. In this section, we will introduce both types of models.

The Sequential model is a linear stack of layers that can be created by passing a list of layer instances to the constructor, or by adding layers via the .add() method [12]. The model expects a certain input shape, which can be declared in the first layer of the model [12]. Before training the model, the learning process has to be configured, which is done via the compile method. Models are trained on Numpy arrays of input data and labels, and are done using the fit function [12].

The Model class is used in the functional API. It can be instantiated by using model = Model(inputs=a, outputs=b) [13]. This model will include all layers required in the computation of b given a. If multi-input or multi-output models are needed, lists can be used as such: model = Model(inputs= [a1, a2], outputs= [b1, b2, b3]) [13].

3.2.1.3 Layers

In this section, three types of layers will be briefly explored, namely Convolutional, Core and Pooling. More details about them can be found in Section 3.3.

Convolutional layers available in Keras include Conv1D (e.g. temporal convolution), Conv2D (e.g. spatial convolution over images), SeparableConv1D (Depthwise separable 1D convolution), SeparableConv2D (Depthwise separable 2D convolution), DepthwiseConv2D (Depthwise separable 2D convolution), Conv2DTranspose (Transposed convolution layer/Deconvolution), Conv3D (e.g. spatial convolution over volumes), Conv3DTranspose (Transposed convolution layer/Deconvolution), Cropping1D (e.g. temporal sequence, it crops along the time dimension/axis 1), Cropping2D (e.g. picture, it crops along spatial dimensions such as height and width), Cropping3D (e.g. spatial or spatio-temporal data), UpSampling2D (repeats rows and columns of data by size[0] and size[1] respectively), UpSampling3D (repeats 1st, 2nd and 3rd dimensions of data by size[0], size[1] and size[2] respectively), ZeroPadding1D (e.g. temporal sequence), ZeroPadding2D (e.g. picture), ZeroPadding3D (e.g. spatial or spatio-temporal data) [14].

Core layers available in Keras include Dense, Activation, Dropout, Flatten, Input, Reshape, Permute, RepeatVector, Lambda, ActivityRegularization, Masking, SpatialDropout1D, SpatialDropout2D and SpatialDropout3D [15].

Pooling layers available in Keras include MaxPooling1D, MaxPooling2D, MaxPooling3D, AveragePooling1D, AveragePooling2D, AveragePooling3D, GlobalMaxPooling1D,

GlobalAveragePooling1D, GlobalMaxPooling2D, GlobalAveragePooling2D, GlobalMaxPooling3D and GlobalAveragePooling3D [16].

3.2.1.4 Preprocessing

In this section, we will be exploring 3 preprocessing functions, naely Optimizers, Activation, and Callbacks.

Given that a loss function is a mathematical way of measuring how wrong predictions are, optimizers tie together the loss function and model parameters by updating the model in response to the output of the loss function. In other words, optimizers shape and mold the model into its most accurate possible form by futzing with the weights [17]. The loss function is the guide to the terrain, telling the optimizer when it's moving in the right or wrong direction [17]. An optimizer is one of the two arguments required for compiling a Keras model. An optimizer can either be instantiated before passing it to model.compile(), or be called by its name. In the latter case, the default parameters for the optimizer will be used. Optimizers available in Keras include Stochastic gradient descent (SGD), RMSProp, Adagrad, Adadelta, Adam, Adamax and Nadam.

The activation function takes into account the interaction effects in different parameters and does a transformation after which it gets to decide which neuron passes forward the value into the next layer [18]. Activations can either be used through an Activation layer, or through the activation argument supported by all forward layers [19]. Activation functions available in Keras include Softmax, Exponential Linear Unit (ELU), Scaled Exponential Linear Unit (SELU), Softplus, Softsign, Rectified Linear Unit (ReLU), Hyperbolic tangent (tanh), Sigmoid, Hard Sigmoid, Exponential (base e), and Linear [19].

A callback is a set of functions to be applied at given stages of the training procedure [20]. By passing a list of them to the .fit() method of the Sequential or Model classes, we can get a view on internal states and statistics of the model during training [20]. The relevant methods of the callbacks will then be called at each stage of the training. Callbacks available in Keras include

the Callback base class, BaseLogger, TerminateOnNaN, ProgbarLogger, History, ModelCheckpoint, EarlyStopping, RemoteMonitor, LearningRateScheduler, TensorBoard, ReduceLROnPlateau, CSVLogger and LambdaCallback[20]. An application of a callback would be using EarlyStopping to reduce overfitting [20]. More information about overfitting can be found in Section 3.4.2.

3.3 Convolutional Neural Networks (CNN)

3.3.1 Introduction to Neural Networks

Before we explain more about Convolutional Neural Networks, it is important for us to understand the definition of neural networks first. Neural networks are a set of algorithms, modeled loosely after the human brain, that are designed to recognize patterns [21]. They interpret sensory data through a kind of machine perception, labeling or clustering raw input [21]. The patterns they recognize are numerical, contained in vectors, into which all real-world data, be it images, sound, text or time series, must be translated [21]. They are used in clustering and classification applications [21].

3.3.2 Introduction to Convolutional Neural Networks

Convolutional Neural Networks (abbreviated as CNNs in this report) are a category of Neural Networks that are used effectively in areas such as image recognition and classification [22]. They have been successful in identifying faces, objects and traffic signs apart from powering vision in robots and self driving cars [22]. Lately, CNNs have been effective in several Natural Language Processing tasks (such as sentence classification) as well [22].

From the Latin convolvere, “to convolve” means to roll together [23]. For mathematical purposes, a convolution is the integral measuring how much two functions overlap as one passes over the other [23]. A convolution can be thought of as a way of mixing two functions by multiplying them [23].

3.3.3 LeNet Architecture

3.3.3.1 Introduction to LeNet Architecture

LeNet was one of the very first CNNs which helped propel the field of Deep Learning. This pioneering work by Yann LeCun was named LeNet5 after many previous successful iterations since the year 1988 [22]. At that time the LeNet architecture was used mainly for character recognition tasks such as reading zip codes, digits, etc. The following paragraphs will explain how it is used in classifying images. There have been several new architectures proposed in the recent years which are improvements over the LeNet, but they all use the main concepts from the LeNet [22].

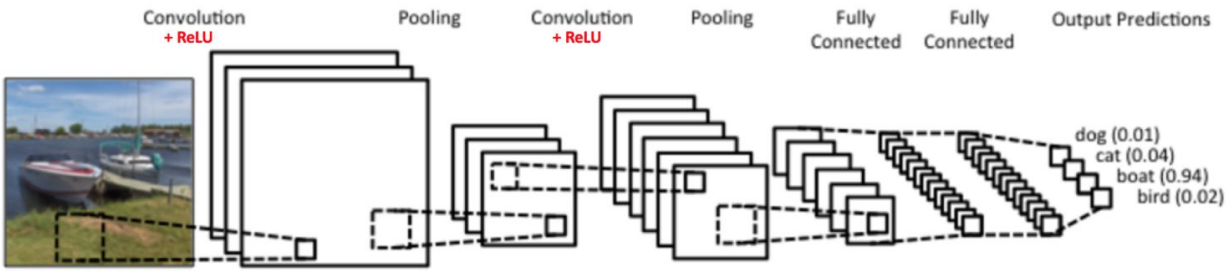


Figure 4: LeNet Architecture [22]

There are four main operations in the CNN shown in Figure 4 above. These operations will be explained in the following sections.

1. Convolution
2. Non Linearity (ReLU)
3. Pooling or Sub Sampling
4. Classification (Fully Connected Layer)

3.3.3.2 Convolution

The primary purpose of Convolution in case of a ConvNet is to extract features from the input image [22]. Convolution preserves the spatial relationship between pixels by learning image features using small squares of input data [22].

It is a known fact that every image can be considered as a matrix of pixel values in the range of 0 to 255, with 0 indicating black and 255 indicating white. Given a 5x5 image (in green below) whose pixel values are only 0 and 1 and a 3x3 matrix (in orange below), we can find that the result of the convolution of this image and the matrix can be represented as the Convolved Feature (in pink below [22]).

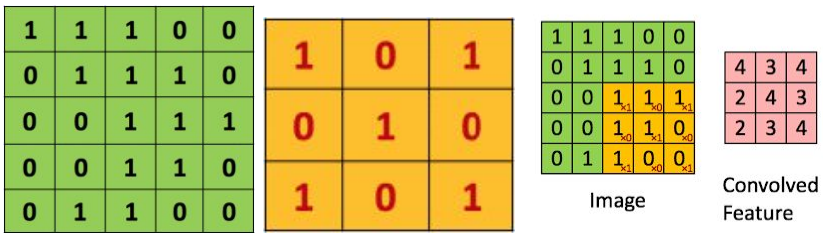


Figure 5: 5x5 image, 3x3 matrix and Convolved Feature [22]

In CNN terminology, the 3×3 matrix is called a ‘filter‘ or ‘kernel’ or ‘feature detector’ and the matrix formed by sliding the filter over the image and computing the dot product is called the ‘Convolved Feature’ or ‘Activation Map’ or the ‘Feature Map‘. It is important to note that filters acts as feature detectors from the original input image [22].

Since different values of the filter matrix will produce different Feature Maps for the same input image, we can perform operations such as Edge Detection, Sharpen and Blur just by changing the numeric values of our filter matrix before the convolution operation. In other words, different filters can detect different features from an image (for example: edges, curves etc).

In practice, although we still need to specify parameters (for example: number of filters, filter size, architecture of the network etc.) before the training process, a CNN learns the values of these filters on its own during the training process. The more number of filters we have, the more image features get extracted and the better our network becomes at recognizing patterns in unseen images [22].

The size of the Feature Map (Convolved Feature) is controlled by three parameters that we need to decide before the convolution step is performed:

1. Depth: Depth corresponds to the number of filters we use for the convolution operation. Using 3 filters would produce 3 different feature maps. Since these 3 feature maps are similar to stacked 2D matrices, the depth of the feature map would be 3 [22].
2. Stride: Stride is the number of pixels by which we slide our filter matrix over the input matrix. When the stride is 1, we move the filters one pixel at a time. When the stride is 2, the filters jump 2 pixels at a time as we slide them around. Having a larger stride will produce smaller feature maps [22].
3. Zero-padding: Sometimes, it is convenient to pad the input matrix with zeros around the border, so that we can apply the filter to bordering elements of our input image matrix [22]. A nice feature of zero padding is that it allows us to control the size of the feature maps. Adding zero-padding is also called wide convolution, and not using zero-padding would be a narrow convolution [22].

3.3.3.3 Non Linearity (ReLU)

ReLU stands for Rectified Linear Unit and is a non-linear operation. Its output is given by: Max(zero, Input) and can be seen in the graph below [22].

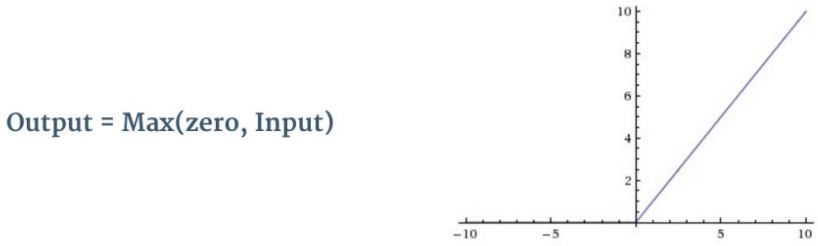


Figure 6: Graph of a ReLU operation [22]

ReLU is an element wise operation in the sense that it is applied per pixel. It replaces all negative pixel values in the feature map by zero [22].

Since convolution is a linear operation that involves element wise matrix multiplication and addition, we account for non-linearity in the real world data we want the CNN to learn by introducing non-linear functions. Examples of non-linear functions include ReLU, tanh and sigmoid, but ReLU has been found to perform best in most situations [22].

3.3.3.4 Pooling or Sub Sampling

Spatial Pooling (also called subsampling or downsampling) reduces the dimensionality of each feature map but retains the most important information. Spatial Pooling can be of different types (for example: max, average, sum etc.) [22].

In the case of Max Pooling, we define a spatial neighborhood (for example, a 2×2 window) and take the largest element from the rectified feature map within that window [22]. Alternatively, instead of taking the largest element we could also take the average or sum of all elements in that window. In practice, Max Pooling has been shown to work better [22].

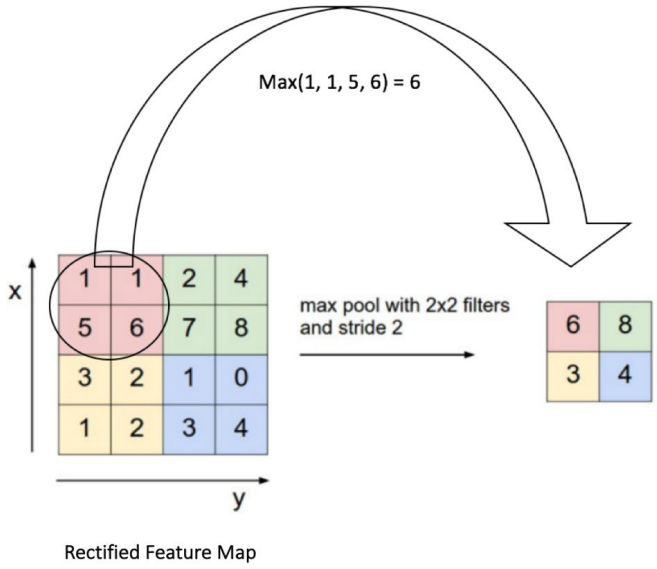


Figure 7: A Max Pooling operation on a rectified feature map [22]

The rectified feature map in Figure 7 was obtained after convolution + ReLU operation by sliding a 2×2 window by 2 cells (also called 'stride'). Taking the maximum value in each region, we find that the dimensionality of our feature map is reduced [22].

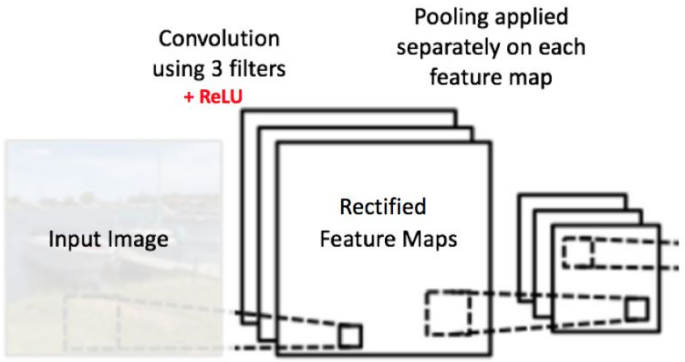


Figure 8: Pooling operation [22]

In Figure 8, since pooling operation is applied separately to each rectified feature map, we get three output maps from three input maps [22].

The function of Pooling is to progressively reduce the spatial size of the input representation. In particular, pooling

- makes the input representations (feature dimension) smaller and more manageable [22]
- reduces the number of parameters and computations in the network, therefore, controlling overfitting [22]
- makes the network invariant to small transformations, distortions and translations in the input image (a small distortion in input will not change the output of Pooling – since we take the maximum / average value in a local neighborhood) [22]
- helps us arrive at an almost scale invariant representation of our image, or rather "equivariant", allowing us to detect objects in an image no matter where they are located [22]

3.3.3.5 Fully Connected Layers

From the previous sections, we find that we have two sets of Convolution, ReLU and Pooling layers [22]. The 2nd Convolution layer performs convolution on the output of the first Pooling Layer using 6 filters, producing 6 feature maps. ReLU is then applied individually on all feature maps. We then perform Max Pooling operation separately on each of the rectified feature maps.

Together, these layers extract the useful features from the images, introduce non-linearity in our network and reduce feature dimension while aiming to make the features somewhat equivariant to scale and translation.

The Fully Connected layer is a traditional Multi Layer Perceptron that uses a softmax activation function in the output layer. The term "Fully Connected" implies that every neuron in the previous layer is connected to every neuron on the next layer [22].

The output from the convolutional and pooling layers represent high-level features of the input image. The purpose of the Fully Connected layer is to use these features for classifying the input image into various classes based on the training dataset. Apart from classification, adding a

fully-connected layer is also a cheap way of learning non-linear combinations of these features [22].

By using Softmax as the activation function in the output layer, the sum of output probabilities from the Fully Connected Layer can be 1. The Softmax function takes a vector of arbitrary real-valued scores and squashes it to a vector of values between zero and one.

3.3.3.6 Summary and Training

As discussed in the previous sections, the Convolution + Pooling layers act as Feature Extractors from the input image while Fully Connected layer acts as a classifier [22]. Since the input image is a boat, the target probability is 1 for Boat class and 0 for other three classes, and the target vector can be given by = [0, 0, 1, 0]

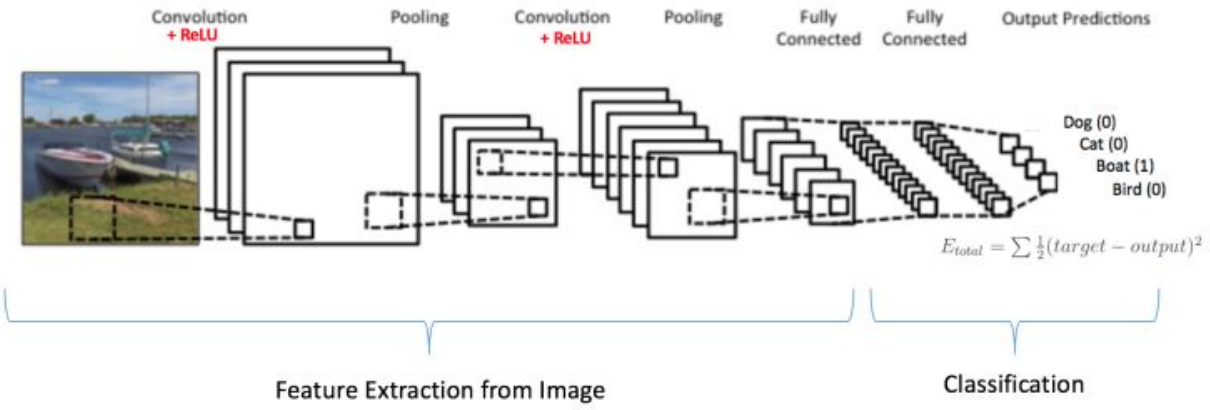


Figure 9: Summary and output of entire network [22]

The overall training process of the Convolution Network may be summarized as below:

- Step 1: Initialize all filters and parameters / weights with random values [22]
- Step 2: The network takes a training image as input, goes through the forward propagation step (convolution, ReLU and pooling operations along with forward propagation in the Fully Connected layer) and finds the output probabilities for each class [22]

    - Let's say the output probabilities for the boat image above are [0.2, 0.4, 0.1, 0.3] [22]
    - Since weights are randomly assigned for the first training example, output probabilities are also random [22]
- Step 3: Calculate the total error at the output layer (summation over all 4 classes) [22]
    - Total Error = $\sum$ ½ (target probability – output probability) ² [22]
- Step 4: Use Backpropagation to calculate the gradients of the error with respect to all weights in the network and use gradient descent to update all filter values / weights and parameter values to minimize the output error [22]
    - The weights are adjusted in proportion to their contribution to the total error [22]
    - When the same image is input again, output probabilities might now be [0.1, 0.1, 0.7, 0.1], which is closer to the target vector [0, 0, 1, 0] [22]
    - This means that the network has learnt to classify this particular image correctly by adjusting its weights / filters such that the output error is reduced [22]
    - Parameters like number of filters, filter sizes, architecture of the network etc. have all been fixed before Step 1 and do not change during training process – only the values of the filter matrix and connection weights get updated [22]
- Step 5: Repeat steps 2-4 with all images in the training set [22]

The above steps train the CNN, essentially meaning that all the weights and parameters have been optimized to correctly classify images from the training set [22]. When a new/unseen image is input into the CNN, the network would go through the forward propagation step and output a probability for each class. For a new image, the output probabilities are calculated using the weights which have been optimized to correctly classify all the previous training examples. If our training set is large enough, the network will be able to generalize well to new images and classify them into correct categories [22].

In general, the more convolution steps we have, the more complicated features our network will be able to learn to recognize. For example, in image classification, it may learn to detect edges

from raw pixels in the first layer, then use the edges to detect simple shapes in the second layer, and then use these shapes to deter higher-level features (for example: facial shapes in higher layers) [22].

3.4 Cross-Validation

3.4.1 Introduction to Cross-Validation

Validation is very important to machine learning as there is a need to validate the stability of the learning model and how well it will adapt to new data or in a practical scenario [24]. Cross-validation allows us to be sure that the model is actually picking up patterns from the data and not the inherent noise from the data, ensuring the quality of the model [24].

The goal of cross-validation is to define a dataset to test the model in the training phase (i.e. a validation dataset) in order to limit problems like overfitting and underfitting, and to get an insight on how the model will generalize to an independent dataset [24]. It is important that the validation and the training set should be drawn from the same distribution. Otherwise, it would make things worse [24].

In typical validation methods, the dataset will be split into 2 sets, a training set and a test set. These 2 sets of data do not overlap and contain different data. The training set will be used to train the model and test set will be used to test the model and determine the accuracy of the model. However, if the split made is not random (e.g. if one subset of our data has only people from a certain state, employees with a certain income level but not other income levels, only women or only people at a certain age), overfitting will occur. This is due to the fact that it is not certain which data points will end up in the validation set and the result might be entirely different for different sets [24].

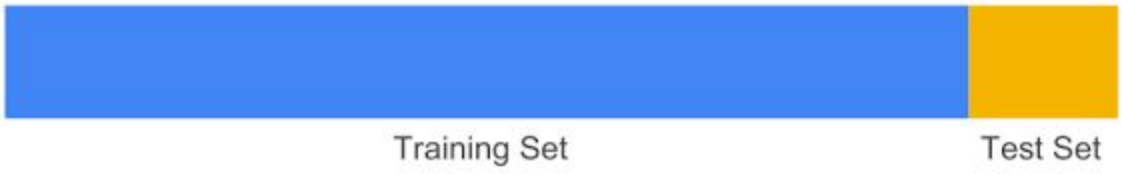


Figure 10: Cross-Validation - data is split into a training set and dataset

3.4.2 Overfitting

Earlier in the report, we briefly mentioned overfitting. Overfitting usually occurs when the model learns the parameters of a prediction function and tests it on the same data. A model that repeats the labels of the samples which it has just seen would have a perfect score, but would fail to predict anything useful on yet-unseen data.

For CNN models, it has been suggested that overfitting can be reduced by adding more data, using data augmentation, using architectures that generalize well, adding regularization (i.e. dropout, L1/L2 regularization), reducing architecture complexity [25].

In our case, to tackle overfitting, we have employed the use of k-fold Cross-Validation, which will be explained in the following section.

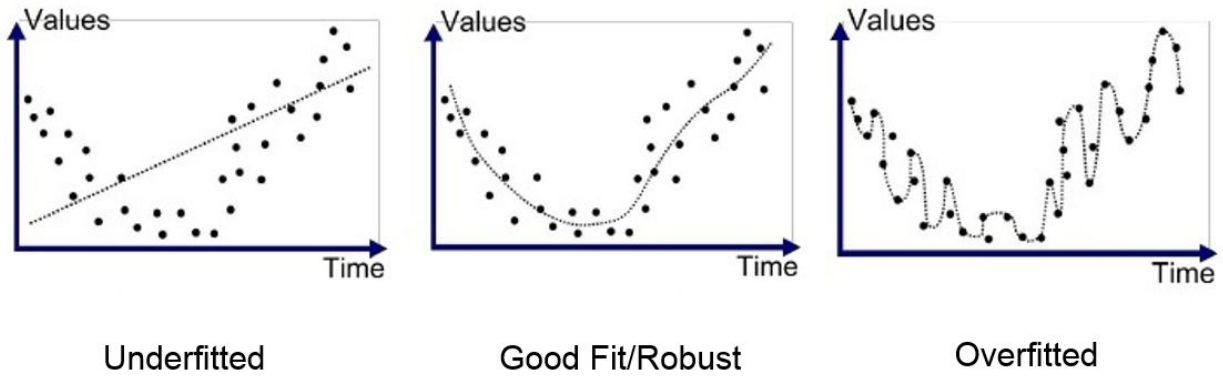


Figure 11: A simplified representation of overfitting

3.4.3 k-fold Cross-Validation

The gold standard for machine learning model evaluation is k-fold Cross-Validation, which provides a robust estimate of the performance of a model on unseen data [26]. It does this by splitting the training dataset into k subsets and takes turns training models on all subsets except one which is held out. Model performance is then evaluated on the held out validation dataset. The process is repeated until all subsets are given an opportunity to be the held out validation set. The performance measure is then averaged across all models that are created [26].

For example, k-fold Cross-Validation is often used with 5 or 10 folds. In other words, 5 or 10 models must be constructed and evaluated. 1 "fold" is used to test the network while the others are used to train it [26]. This reduces the chance of overfitting and increases the accuracy of the model in a real world scenario [26].

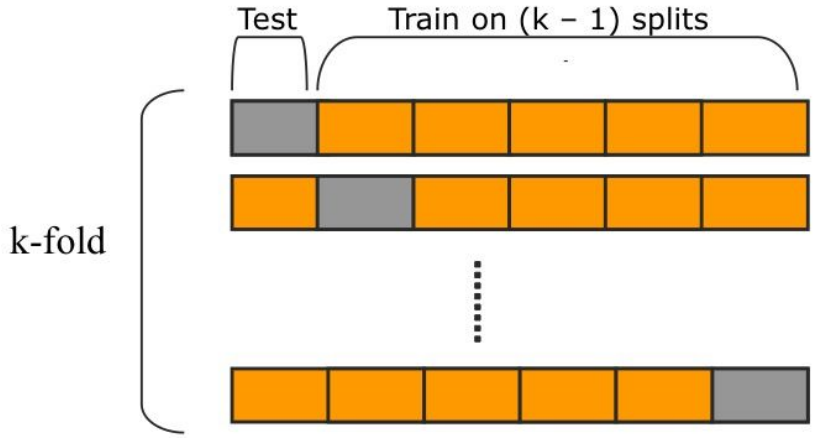


Figure 12: Visualisation of K-Fold Cross validation

3.5 Generative Adversarial Networks (GANs)

A Generative Adversarial Network (abbreviated as GAN in this report) consists of two networks, a generator and a discriminator. The main job of the generator is to generate new instances of data, while the discriminator tries to discriminate between the generated data and the actual data, deciding if the belongs in the dataset. Firstly, the GAN generator takes in random numbers and

returns an image. This generated image is fed into the discriminator alongside a stream of images taken from the actual, ground-truth dataset [27]. The discriminator takes in both real and fake images and returns probabilities, a number between 0 and 1, with 1 representing a prediction of authenticity and 0 representing fake. This results in a double feedback loop. The discriminator is in a feedback loop with the ground truth of the images, while the generator is in a feedback loop with the discriminator [27].

In simpler terms, a GAN can be thought of as the opposition of a counterfeiter and a cop in a game of cat and mouse, where the counterfeiter is learning to pass false notes, and the cop is learning to detect them. Since both the counterfeiter and cop sides are dynamic, each side comes to learn the other's methods in a constant escalation.

Since both nets are trying to optimize a different and opposing objective function (or loss function) in a zero-sum game, this is essentially an actor-critic model. As the discriminator changes its behavior, so does the generator, and vice versa. Their losses push against each other [27].

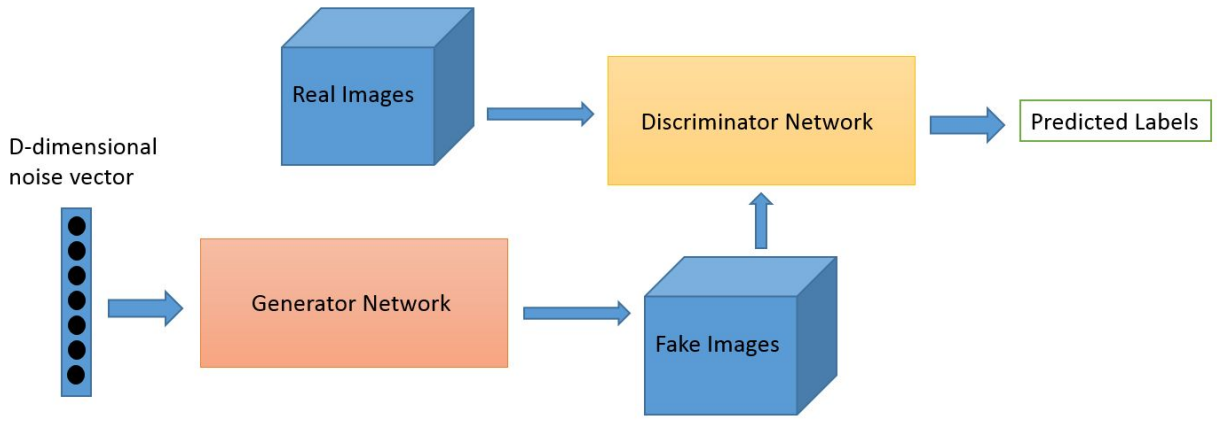


Figure 13: Visualisation of GAN networks [27]

3.6 Voice Analysis and Visualisation

3.6.1 Visual Representations

Sound can be transmitted as waves, and sound waves are one-dimensional. At every moment in time, they have a single value based on the height of the wave [28].

Figure 14: Visual representation of waveform [28]

To turn this sound wave into numbers, we can just record of the height of the wave at equally-spaced points, or in simpler words, sampling. A reading is taken thousands of times a second and a number representing the height of the sound wave at that point in time is recorded. Sampling a sound wave at 16000 times per second and only taking the first 100 samples, we can get a series of 100 numbers, where each number represents the amplitude of the sound wave at 1/16000th of a second intervals [28].

```
[-1274, -1252, -1160, -986, -792, -692, -614, -429, -286, -134, -57, -41, -169, -456, -450, -541, -761, -1067, -1231, -1047, -952, -645, -489, -448
, -397, -212, 193, 114, -17, -110, 128, 261, 198, 390, 461, 772, 948, 1451, 1974, 2624, 3793, 4968, 5939, 6057, 6581, 7302, 7640, 7223, 6119, 5461,
 4820, 4353, 3611, 2740, 2004, 1349, 1178, 1085, 901, 301, -262, -499, -488, -707, -1406, -1997, -2377, -2494, -2605, -2675, -2627, -2500, -2148, -
1648, -970, -364, 13, 260, 494, 788, 1011, 938, 717, 507, 323, 324, 325, 350, 103, -113, 64, 176, 93, -249, -461, -606, -909, -1159, -1307, -1544]
```

Figure 15: Sampled sound wave [28]

To ensure that no data is lost in between the readings, we use Nyquist Theorem to perfectly reconstruct the original sound wave from the spaced-out samples. The Nyquist Theorem states that we need to sample at least twice as fast as the highest frequency we need to record [28].

With this, we are able to obtain an array of numbers, with each number representing a certain amplitude of the sound wave [28].

Although it is possible to feed the array of numbers right into a neural network, trying to recognize speech patterns by processing these samples directly is difficult. As such, audio preprocessing has to be done. A prime example commonly used in preprocessing is Fourier Transform [28].

It breaks apart the complex sound wave into the simple sound waves that make it up [28]. Once we have those individual sound waves, we add up how much energy is contained in each one [28]. The end result is a score of how important each frequency range is, from low pitch (i.e. bass notes) to high pitch [28]. A number can be used to represent how much energy was in each 50Hz band of the 20 millisecond audio clip [28]. These can be seen easily when drawn on a chart.

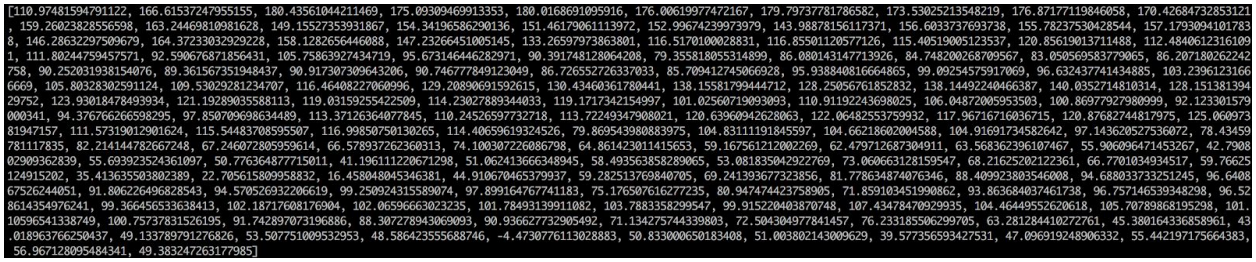


Figure 16: Fourier Transform of sampled sound wave from Figure 15 [28]

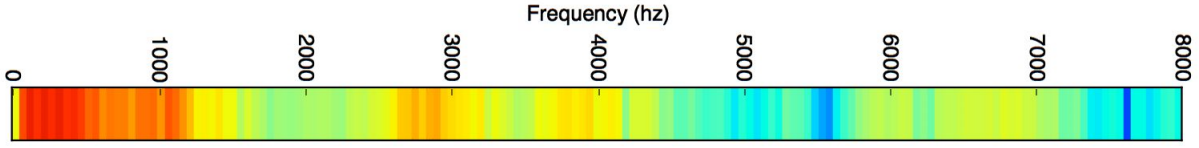


Figure 17: Data in Figure 16 converted into a chart [28]

Repeating this process on every 20 millisecond chunk of audio, we end up with a spectrogram. A spectrogram can be defined as an intensity plot (usually on a log scale, such as dB) of the

Short-Time Fourier Transform (STFT) magnitude [29]. The STFT is simply a sequence of Fast Fourier Transform (FFTs) of windowed data segments, where the windows are usually allowed to overlap in time, typically by 25-50% [29]. Windowing will be explained in the sections below. Spectrograms, which allow us to see musical notes and other pitch patterns in audio data, is an important representation of audio data [28]. Human hearing is based on a kind of real-time spectrogram encoded by the cochlea of the inner ear [28]. Parameters of the spectrogram include the window length M, window type (Hamming, Kaiser, etc.), hop-size R and FFT length N [28]. The window length M controls frequency resolution, the window type controls side-lobe suppression (at the expense of resolution when M is fixed), and the FFT length N determines how much spectral oversampling (interpolation) is to be provided [30]. The hop-size R determines how much oversampling there will be along the time dimension [30].

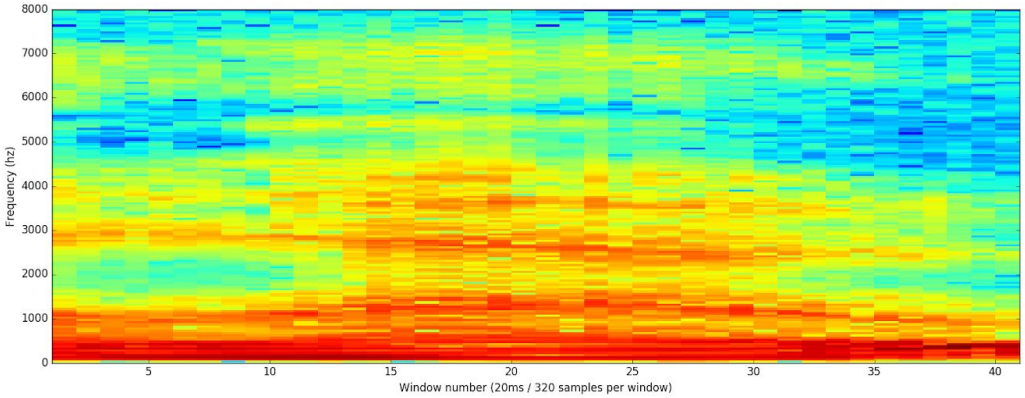


Figure 18: Example of a spectrogram [28]

Another visual representation method is a spectrum. As contrasted to waveform data, spectral data is displayed in the frequency domain (waveform data is in the time domain) [31]. Spectral information is obtained by applying a Fourier transform to waveform data [31]. Fourier transform converts the data to show the amplitude and phase of the vibration at different frequencies [31]. It decomposes the time domain information that contains multiple frequencies; however, any random or non-periodic impacts will be ignored with this method and they will not show up in the spectrum [31]. The crest factor (peak-to-average ratio/peak-to-average power

ratio/ratio between peak amplitude and RMS value) of a waveform can be calculated in the time domain and it helps determine if any impacting is occurring, and its severity [31].

Both spectral and waveform data have unique advantages when used in analysis [31]. For simple analysis where a single frequency is present in the data, either form of data can be used as shown in Figure 19 below. It is relatively simple to extract the amplitude, phase and frequency from both data sets [31].

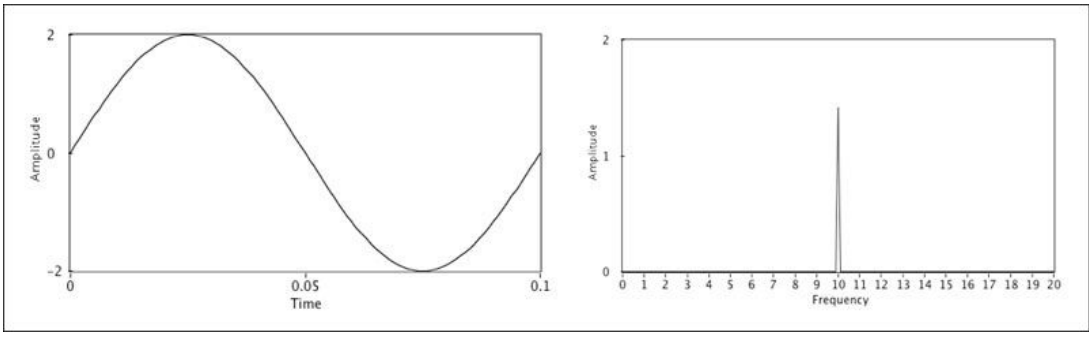


Figure 19. A waveform and the corresponding spectrum [31]

Unfortunately, most data is not as simple to analyze as like in Figure 20. Usually noise is present in the data, and multiple frequencies may be present. This makes it difficult to extract useful information from the waveform, and spectral data can be better for analysis. An example of this is shown in Figure 20.

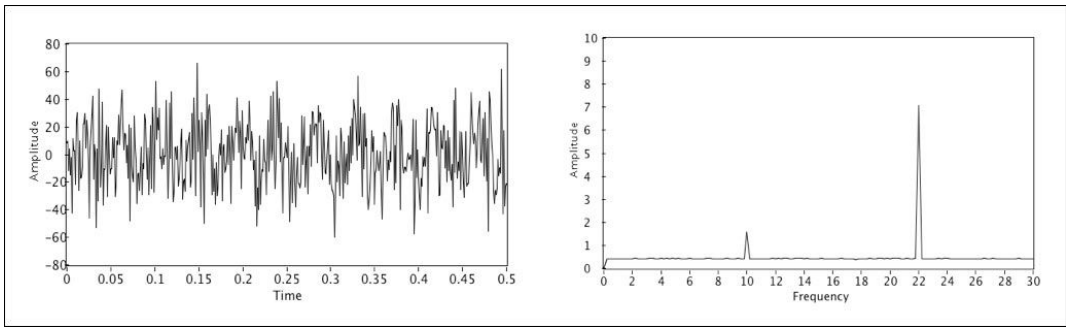


Figure 20: A waveform with multiple frequencies and noise and the corresponding spectrum [31]

As mentioned in previous sections, we worked on an Android application last year which was able to detect depression in real-time. In the development process, one of our main aims was to provide a visual representation of real-time audio coming in through the phone's microphones. To get this representation, we used Canvas (a class given in Android) to draw out the amplitude of the signal in real-time.

To conclude, we find that a neural network can find patterns in spectrograms more easily than raw sound waves and spectra. As such, spectrograms can be sent into neural networks such as our CNN model [28]. However, spectra can still be used as a form of visual representation and will be explored further in the sections below.

3.6.2 Framing

An audio frame is a group of audio signals and consists of exactly one sample per channel. An audio signal creates pressure which goes into the eardrum, causing the eardrum to vibrate, sending a signal to the brain. Audio signals can be simulated with microphones, which capture vibrations and change them into signals. If there is only one channel, the sound is mono and a frame is a single sample. If the sound is stereo, each frame consists of two samples [32].

Last year, in the development process of our Android application, we found that the sound we obtained from the microphones was mono. As such, our frame was a single sample. We collected data from the phone’s microphone at short time intervals. Following that, we implemented frame blocking by segmenting the audio signal into frames of 25 milliseconds with an overlap of 3/5 of the frame size (15 milliseconds). The moving window used was 10 milliseconds, 2/5 of the frame size.

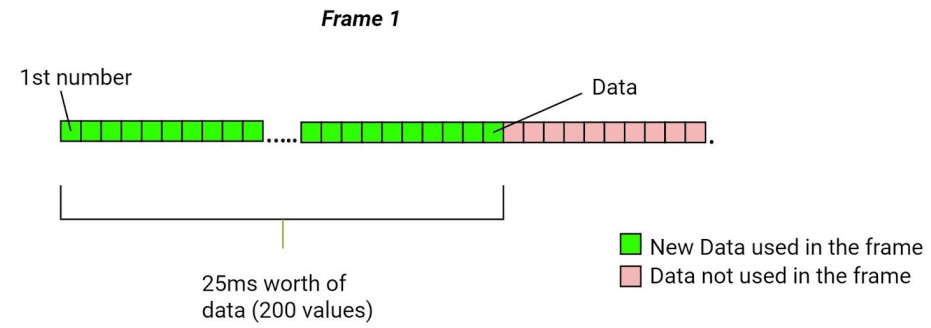


Figure 21: Frame 1 of data

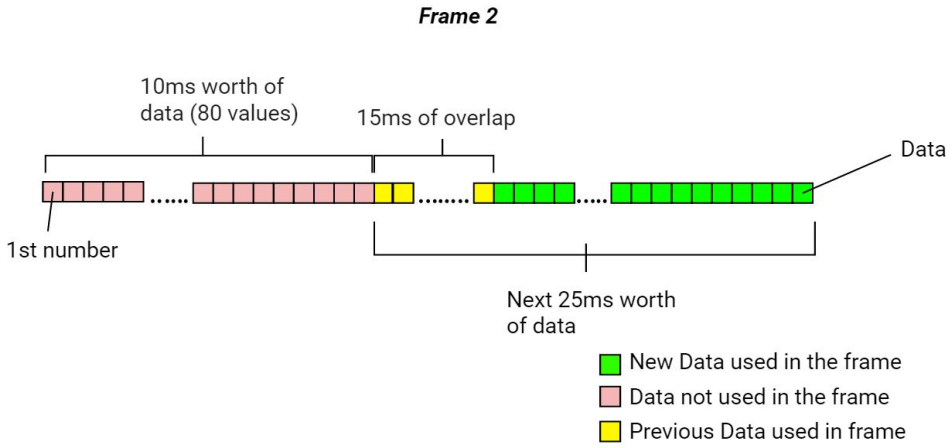


Figure 22: Frame 2 of data

Usually the frame size (in terms of sample points or shorts) is equal to a power of two in order to facilitate the use of Fast Fourier Transform (FFT). However, if this was not the case, we had to do zero padding to the nearest length of power of two [33].

In our application last year, the sample rate used was 8000 Hertz, which is usually used for speech signals, and the frame size/buffer size used was 8000 shorts per second. As such, over an audio frame duration of 25 milliseconds, we obtained 200 shorts. Zero padding was done on these 8000 shorts as well. Eventually, after applying frame blocking, we were able to obtain 100 frames over a period of 1 second. This essentially also meant that we obtained 20000 shorts per

second after the overlaps. By using frame blocking in our application, we were able to reduce any data lost in between frames and retain all the data we have obtained from the audio signal.

3.6.3 Windowing

In the analysis of audio signals, a short segment of a signal, rather than the whole signal, is usually used. One reason is due to the fact that the ear similarly Fourier analyzes only a short segment of audio signals at a time (on the order of 10-20 ms worth) [34]. Windowing essentially reduces the amplitude of the discontinuities at the boundaries of each finite sequence acquired by the digitizer [34]. Windowing consists of multiplying the time record by a finite-length window with an amplitude that varies smoothly and gradually toward zero at the edges [34]. This makes the endpoints of the waveform meet and results in a continuous waveform without sharp transitions [34].

Therefore, a spectrum analysis having time- and frequency-resolution comparable to human hearing must have the time-window limited accordingly [34]. As such, the proper method for extracting a "short time segment" of length N from a longer signal is to multiply it by a window function such as the Hann window, which is given by the equation [34]:

$$w(n) = \cos^2\left(\frac{n}{N}\pi\right), \quad n = -\frac{N-1}{2}, \ldots, -1, 0, 1, \ldots, \frac{N-1}{2}$$

In windowing, there are different types of window types such as rectangular, generalized Hamming and Blackman-Harris families (sums of cosines), Bartlett (triangular), Poisson (exponential), Kaiser (Bessel), Dolph-Chebyshev, Gaussian, and other window types [36]. In this section, we will be exploring the generalised Hamming window family in more detail.

The generalized Hamming window family is constructed by multiplying a rectangular window by one period of a cosine. The benefit of the cosine tapering is lower side-lobes. The price for this benefit is that the main-lobe doubles in width [37]. Two well known members of the

generalized Hamming family are the Hann and Hamming windows [37]. The Hamming and Hann window functions both have a sinusoidal shape [35]. Both windows result in a wide peak but low side lobes [35]. However, the Hann window touches zero at both ends eliminating all discontinuity [35]. The Hamming window doesn't quite reach zero and thus still has a slight discontinuity in the signal [35]. Because of this difference, the Hamming window does a better job of cancelling the nearest side lobe but a poorer job of canceling any others [35].

The center dotted waveform is the aliased sinc function $0.5 \cdot W_R(\omega) = 0.5 \cdot M \cdot \mathrm{asinc}_M(\omega)$ (scaled rectangular window transform). The other two dotted waveforms are scaled shifts of the same function, $0.25 \cdot W_R(\omega \pm \Omega_M)$ [37]. The sum of all three dotted waveforms gives the solid line. We see that there is some cancellation of the side lobes and the width of the main lobe is doubled [37].

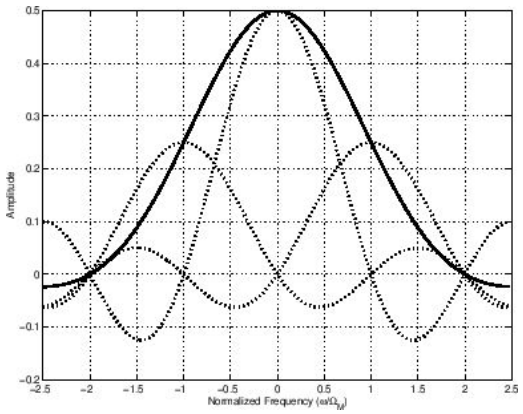


Figure 23: Construction of generalized Hamming window transform as a superposition of three shifted aliased sinc functions [37]

Hanning windows can be seen as one period of a cosine "raised" so that its negative peaks just touch zero [38]. Similarly, hamming windows can also be seen one period of a raised cosine [39]. However, the cosine is raised so high that its negative peaks are above zero, and the window has a discontinuity in amplitude at its endpoints (stepping discontinuously from 0.08 to 0) [39].

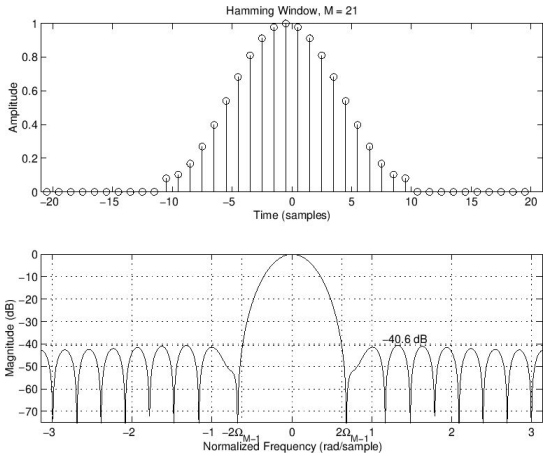


Figure 24: Hamming window and DTFT [39]

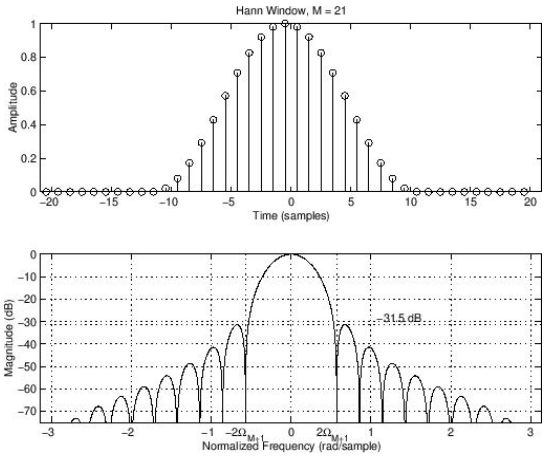


Figure 25: Hanning window and DTFT [38]

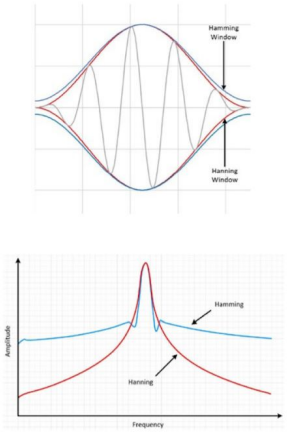


Figure 26: Hamming and Hann windowing result in a wide peak but nice low side lobes [35]

An example of how these windows can be used in speech analysis would be making Hamming windows be around 20 milliseconds [40]. This is short enough so that any single 20 milliseconds frame will typically contain data from only one phoneme, yet long enough that it will include at least two periods of the fundamental frequency during voiced speech, assuming the lowest voiced pitch to be around 100 Hz [40].

3.6.4 Sampling Rate and Pulse Code Modulation Data

The sampling rate is the number of samples of a sound that are taken per second to represent the event digitally [41]. For example, a sampling rate of 44.1kHz means that for every second, 44100 samples of the sounds are taken. The more samples that are taken, the more details are captured (e.g.: where the waves rise and fall) [41]. Hence, the quality of the audio will be higher. Also, the shape of the sound wave is captured more accurately [41]. Each audio sample will represent the amplitude of the signal at its given point of time [41]. The amplitude will be either be stored as an integer or floating point number, and then be encoded as a binary number [41].

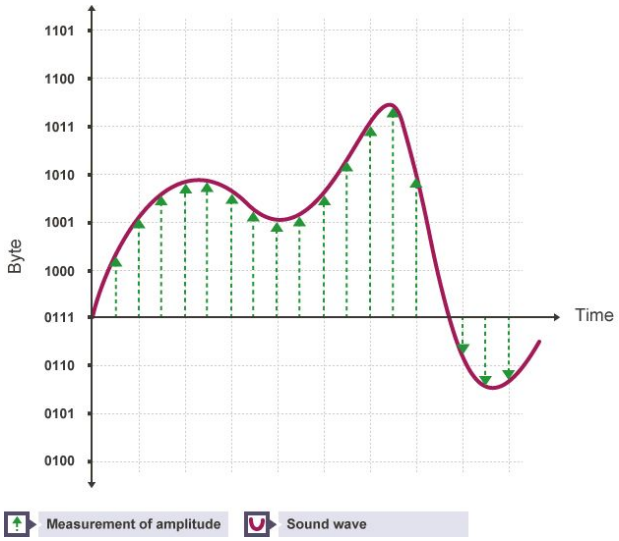


Figure 27: Graph of binary of audio signal for 0.1ms, given a 96kHz sampling rate [41]

Pulse code modulation (PCM) allows us to get a digital representation of an analog signal [42]. It takes samples of the amplitude of the analog signal at regular intervals [42]. The sampled analog data is changed to, and then represented by binary data [42]. PCM requires a very accurate clock [42]. The number of samples per second, ranging from 8,000 to 192,000, is usually several times the maximum frequency of the analog waveform in Hertz (Hz), or cycles per second, which ranges from 8 to 192 KHz [42].

The word pulse refers to pulses found in transmission lines, which are a natural consequence of two other almost simultaneously evolved analog methods: pulse width modulation and pulse position modulation, where each uses discrete signal pulses of varying widths or positions [42]. Otherwise, PCM has little similarity to these other forms of signal encoding [42]. These methodologies were introduced to the U.S. in the early 1960s as telephone companies began converting voice to digital signals to facilitate transmission between cities [42].

To obtain PCM from an analog waveform, the signal amplitude is sampled at regular time intervals [43]. The sampling rate is several times the maximum frequency of the analog waveform in cycles per second or hertz [43]. The instantaneous amplitude of the analog signal at each sampling is rounded off to the nearest of several specific, predetermined levels [43]. This process is called quantization [43]. The number of levels is always a power of 2 -- for example, 8, 16, 32, or 64 [43]. These numbers can be represented by three, four, five, or six binary digits (bits) respectively [43]. The output of a pulse code modulation is thus a series of binary numbers [43].

3.6.5 Android Audio APIs

3.6.5.1 AudioRecord

An AudioRecord object can read raw PCM data from the audio hardware [44]. However, the application must periodically poll the AudioRecord object and read audio data [44].

Upon creation, an AudioRecord object initializes its associated audio buffer that it will fill with the new audio data [45]. The size of this buffer, specified during the construction, determines how long an AudioRecord can record before "over-running" data that has not been read yet [45]. Data should be read from the audio hardware in chunks of sizes inferior to the total recording buffer size [45].

3.6.5.2 AudioTrack

An AudioTrack object can directly write raw PCM audio data [44]. The application can create multiple instances of AudioTrack to simultaneously play multiple audio streams [44]. However, each instance of AudioTrack must specify the buffer size on initialization [44]. This buffer size determines the minimum frequency at which data should be written to the AudioTrack object to avoid under-runs which in turn lead to glitches and stutters in the playback [44]. As the data is buffered in the Audio framework, the buffer size also affects the playback latency [44].

3.6.5.3 MediaRecorder

MediaRecorder records audio data from the specified source to the file whose path is specified in the setAudioSource() and the setOutputFile() APIs [44]. However, MediaRecorder cannot be used if additional sound processing is to be done on the audio data played or recorded [44]. Applications may frequently want to process the recorded data before saving, or apply filters like equalization, echo, reverb or custom filters [46]. For this, they need control over individual buffers of data before they are saved or played [46].

3.6.5.4 MediaPlayer

MediaPlayer provides the primary APIs used for playback [44]. A MediaPlayer object can be used to play audio data from a file or a specified URI/UR [44]L. The source is passed to the object using the setDataSource() API and applications can leverage this object if they want to play audio from a fixed source [44]. The MediaPlayer is most commonly used to play audio from compressed audio files (like MP3, AAC etc.) [44].

3.6.6 GraphView

GraphView is a library for Android to programmatically create flexible and nice-looking diagrams [47]. It is easy to understand, to integrate and to customize. Line Graphs, Bar Graphs, Point Graphs or a custom type of graph can be created [47].

Considering how customisable GraphView can be, it can be applied for visualisation of audio signals or maybe even other features that require graphs.

3.7 Denoising Images and Audio Signals

3.7.1 Introduction to Image Noise

Image noise is random variation of brightness or color information in images, and is usually an aspect of electronic noise [48]. It can be produced by the sensor and circuitry of a scanner or digital camera [48]. Image noise can also originate in film grain and in the unavoidable shot noise of an ideal photon detector [48]. Image noise is an undesirable by-product of image capture that obscures the desired information [48]. In this section, two types of image noise will be explored - Gaussian noise and Salt-and-pepper noise.

Gaussian noise is statistical noise having a probability distribution function equal to that of the normal distribution/Gaussian distribution [49]. It occurs in telecommunications, computer networking, communication channels and digital images [49]. It is usually additive and spatially uncorrelated, meaning that the noise for each pixel is independently and identically distributed [49]. Principal sources of Gaussian noise in digital images arise during acquisition e.g. sensor noise caused by poor illumination and/or high temperature, and/or transmission e.g. electronic circuit noise [50]. In digital image processing, Gaussian noise can be reduced using a spatial filter, though when smoothing an image, an undesirable outcome may result in the blurring of fine-scaled image edges and details because they also correspond to blocked high frequencies [50]. Conventional spatial filtering techniques for noise removal include: mean (convolution) filtering, median filtering and Gaussian smoothing [50].

Salt-and-pepper noise is a form of noise sometimes seen on images [51]. It is also known as impulse noise [51]. This noise can be caused by sharp and sudden disturbances in the image signal [51]. It presents itself as sparsely occurring white and black pixels [51]. An effective noise reduction method for this type of noise is a median filter or a morphological filter [51]. For

reducing either salt noise or pepper noise, but not both, a contraharmonic mean filter can be effective [51].

3.7.2 Introduction to Audio Noise

In audio, noise is generally any unpleasant sound and, more technically, any unwanted sound that is unintentionally added to a desired sound [52]. In recording sound, noise is often present on analog tape or low-fidelity digital recordings [52]. The standard audio cassette includes a layer of hiss on every recording [52]. When doing digital recording, the conversion of a sound file from 16-bit to 8-bit adds a layer of noise [52]. In this section, two types of audio noise will be explored - white noise and pink noise.

White noise is a sound that contains every frequency within the range of human hearing (generally from 20 Hertz to 20 kHz) in equal amounts [52]. Most people perceive this sound as having more high-frequency content than low, but this is not the case [52]. This perception occurs because each successive octave has twice as many frequencies as the one preceding it [52]. For example, from 100 Hz to 200 Hz, there are one hundred discrete frequencies [52]. In the next octave (from 200 Hz to 400 Hz), there are two hundred frequencies [52]. White noise can be generated on a sound synthesizer [52].

Pink noise is a variant of white noise [52]. Pink noise is white noise that has been filtered to reduce the volume at each octave [52]. This is done to compensate for the increase in the number of frequencies per octave [52]. Each octave is reduced by 6 decibels, resulting in a noise sound wave that has equal energy at every octave [52].

3.7.3 Denoising and its importance

Noise reduction is the process of removing noise from a signal [53]. All signal processing devices, both analog and digital, have traits that make them susceptible to noise [53]. Noise can be random or white noise with an even frequency distribution, or frequency dependent noise

introduced by a device's mechanism or signal processing algorithms [53]. Many noise reduction algorithms tend to alter signals to a greater or lesser degree [53].

Noise reduction techniques are especially important in digital signals to prevent their corruption [54]. For example, image denoising plays an important role in a wide range of applications such as image restoration, visual tracking, image registration, image segmentation, and image classification, where obtaining the original image content is crucial for strong performance [54].

Denoising is especially important in our case as if the noise is left in the spectrogram, the accuracy of the model might decrease as the model might start to pick up on the noise and think that the noise is what's causing it to be a positive/negative reading. This would decrease the validation accuracy of the model. Hence we have to denoise the images to further increase the accuracy.

3.7.4 Autoencoders for Image Denoising

An autoencoder is made out of two parts: an encoder and a decoder [55]. The encoder reduces the dimensions of input data so that the original information is compressed while the decoder restores the original information from the compressed data [55].

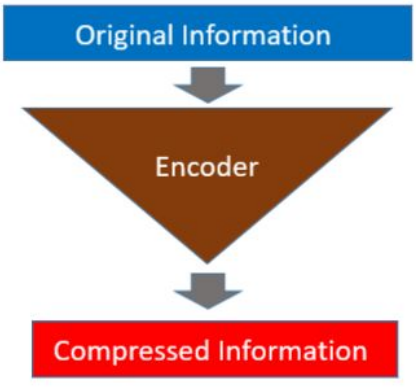


Figure 28: Encoder [55]

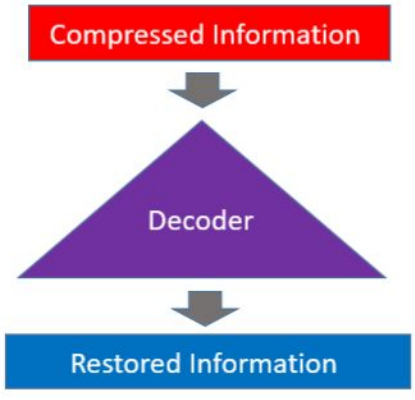


Figure 29: Decoder [55]

The autoencoder is a neural network that learns to encode and decode automatically [55]. Once learning is done, the encoder and decoder can be used independently [55].

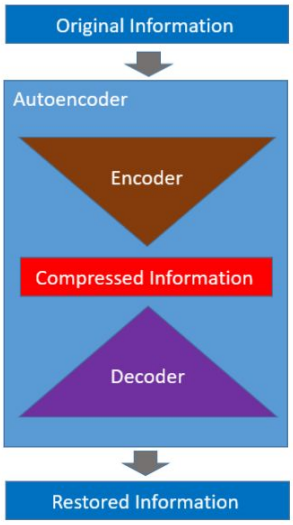


Figure 30: Autoencoder with encoder and decoder [55]

Autoencoders are data specific and do not work on completely unseen data structure [55]. For example, an autoencoder trained on numbers does not work on alphabets [55]. Another limitation is that the compression by an autoencoder is lossy [55]. As such, it does not perfectly restore the original information [55]. Autoencoders can be useful for many different things, such as image noise reduction [55]. For this, the autoencoder can be trained with the noisy data as input and have the original data as expected output [55]. During the training, the autoencoder learns to extract important features from input images and ignores the image noises because the labels have no noises [55]. The network can also be made deeper by adding more layers, improving its performance [55]. For example, CNNs can be used for working on images to improve the quality of compression and decompression [55].

3.7.5 CGAN for Image Denoising

The conditional generative adversarial network, or cGAN for short, is a type of GAN that involves the conditional generation of images by a generator model [56] Image generation can be conditional on a class label, if available, allowing the targeted generated of images of a given type [56]. This means that when the trained generator model is used as a standalone model to generate images in the domain, images of a given type, or class label, can be generated [56].

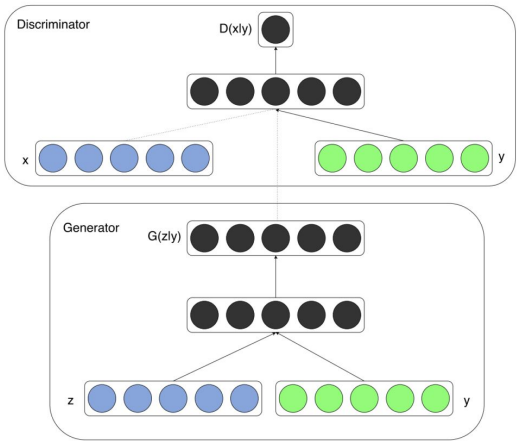


Figure 31 : Conditional Discriminator and Conditional Generator in a CGAN [57]

The CGAN for image denoising in [58] applies Gaussian random noise to the input data as the condition for the generator model.

3.7.6 OpenCV for Non-Local Means Image Denoising

There have been many image smoothing techniques (e.g. Gaussian Blurring, Median Blurring) that are able to remove small quantities of noise (to some extent) [59]. In those techniques, a small neighbourhood was taken around a pixel and operations such as gaussian weighted average and median of the values were done to replace the central element [59]. In short, noise removal at a pixel was local to its neighbourhood [59].

Noise is generally considered to be a random variable with zero mean [59]. Consider a noisy pixel, $p = p_0 + n$ where p_0 is the true value of the pixel and n is the noise in that pixel [59]. A large number of the same pixels can be taken from different images and their average can be computed [59]. Ideally, we obtain $p = p_0$ since the mean of noise is zero [59].

With this concept in mind, we find that we can use a set of similar images to average out the noise. Considering a small window (say 5x5 window) in the image, there is a huge chance that the same patch may be somewhere else in the image (e.g. in a small neighbourhood around it) [59]. These similar patches can be used together and their average can be computed [59].

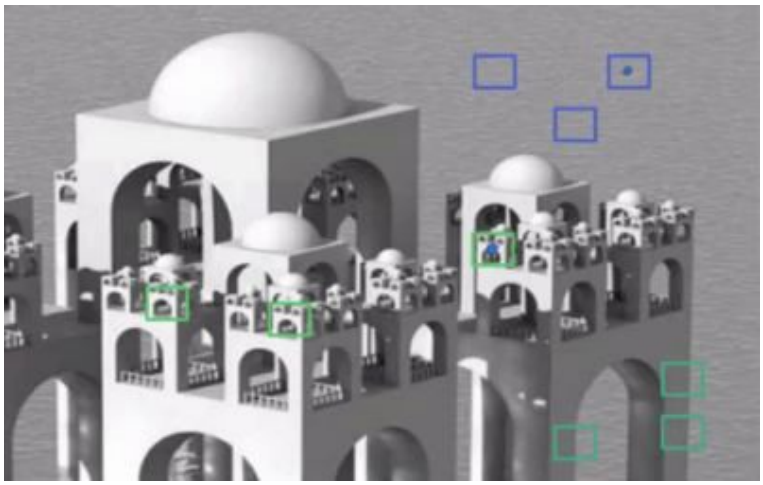


Figure 32: Sample Image for OpenCV Non-Local Means Image Denoising [59]

In Figure 32, we see that the blue patches in the image look similar and the green patches look similar. As such, we can take a pixel as well as a small window around it, search for similar windows in the image, average all the windows and replace the pixel with the result obtained [59]. This method is known as Non-Local Means Denoising. It takes more time compared to the blurring techniques above, but its result is very good [59].

In OpenCV, 4 variations of this technique are provided:

1. cv.fastNlMeansDenoising() - works with a single grayscale images [59]
2. cv.fastNlMeansDenoisingColored() - works with a color image [59]

3. cv.fastNlMeansDenoisingMulti() - works with image sequence captured in short period of time (grayscale images) [59]
4. cv.fastNlMeansDenoisingColoredMulti() - same as above, but for color images [59]

For these 4 variations, there are 3 common arguments for grayscale images (h, templateWindowSize and searchWindowSize) and 4 common arguments for coloured images (h, hForColorComponents, templateWindowSize and searchWindowSize):

1. h : parameter deciding filter strength. Higher h value removes noise better, but removes details of image, 10 is recommended [59]
2. hForColorComponents : same as h, but for color images only [59]
3. templateWindowSize : should be odd, 7 is recommended [59]
4. searchWindowSize : should be odd, 21 is recommended [59]

3.7.7 Noisereduce for Audio Denoising

Noisereduce is an algorithm that provides noise reduction using Spectral Gating in Python [60]. The algorithm requires two inputs, namely a noise audio clip containing prototypical noise of the audio clip and a signal audio clip containing the signal and the noise intended to be removed [60].

The algorithm uses 7 steps:

1. An FFT is calculated over the noise audio clip [60]
2. Statistics are calculated over FFT of the noise (in frequency) [60]
3. A threshold is calculated based upon the statistics of the noise (and the desired sensitivity of the algorithm) [60]
4. An FFT is calculated over the signal [60]
5. A mask is determined by comparing the signal FFT to the threshold [60]
6. The mask is smoothed with a filter over frequency and time [60]
7. The mask is applied to the FFT of the signal, and is inverted [60]

Noisereduce was installed using pypi (pip install noisereduce) and reference usages were available on GitHub at [61].

3.8 SQLite Database

SQLite is a software library that provides a relational database management system. It is light-weight in terms of setup, database administration, and required resources [62]. SQLite is self-contained, serverless, zero-configuration and transactional [62].

Normally, an RDBMS such as MySQL, PostgreSQL, etc, requires a separate server process to operate [62]. The applications that want to access the database server use TCP/IP protocol to send and receive requests [62]. This is called client/server architecture [62].

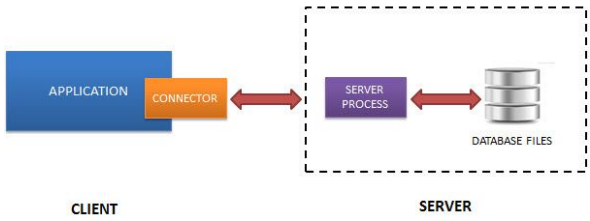


Figure 33: RDBMS client/server architecture [62]

SQLite is different as the database is integrated with the application that accesses the database. The applications interacting with the SQLite database read and write directly from the database files stored on disk [62].

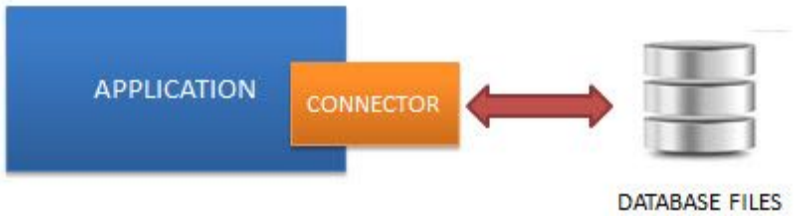


Figure 34 : SQLite serverless architecture [62]

SQLite uses dynamic types for tables [62]. In other words, any value can be stored in any column, regardless of the data type [62]. SQLite allows a single database connection to access multiple database files simultaneously [62]. This brings many nice features like joining tables in different databases or copying data between databases in a single command [62].

3.9 Material Design

3.9.1 Introduction to Material Design

Material Design is a design language that Google developed in 2014 [63]. By basing an application on this design language and applying the principles based on this, an application can be made more interactive and the design can be enhanced.

Regardless of screen size, with Material Design, the application should still look good due to the margins and widths set. This allows for portability and smoothness no matter which device the application is deployed on.

3.9.2 Usages of Material Design

Some usages of Material Design include:

1. The material theme – this offers a new style of display for Android apps. It has in built systems widgets that allow the developer to quickly and easily transform the colour palette. There are also a bunch of default animations for touch feedback and other activity style transitions. [64]

2. List and card support – there are two widgets which support Material Design's list and card formats which include all styles and animations. The list widget is RecyclerView and the card widget is CardView. [64]
3. Material Design also added a change to the way shadows are displayed and they now have (in addition to the old x and y components) a z component which shows the elevation of view and affects:
    a. High z values lead to big shadows and low z values to small ones [64]
    b. High z values also determine that the component will appear above other views in the mix [64]
4. Animations are supported via APIs that allow the developer to build bespoke animations for touch feedback in the UI as well as changes in view states and activity transitions. These:
    a. Allow the application to respond to touch events by displaying touch feedback animations [64]
    b. Enable you to hide and reveal views using circular reveal animations [64]
    c. Allow the use of curved motion to make animations appear more natural to the user [64]
    d. Allow you to develop custom activity transition animations [64]
    e. Allow you to animate changes in one or more views using "view state change" [64]
    f. Show a full list of animations in state list drawables between view state changes [64]
    g. Touch feedback animations are also offered in several standard views (for example for buttons) these can be easily customized and then placed into your own custom views [64]

4. Experimental Results and Analysis

4.1 Pre-Processing Data

4.1.1 Classification and Folder Directories

In this approach, the participants audio is extracted from the .wav file using the transcript. Using AudioSegment.pydub module in Python [65], it is possible to slice and extract portions of a .wav file given the timings. As shown in Figures 35 and 36 below, we can see the differences when the patient's voice is not extracted and there is silence in the audio. The red boxes in Figure 36 shows the areas of silence which are not needed. It is difficult to distinguish the voices by looking at the spectrogram, however it is possible to see the change when there are words spoken.

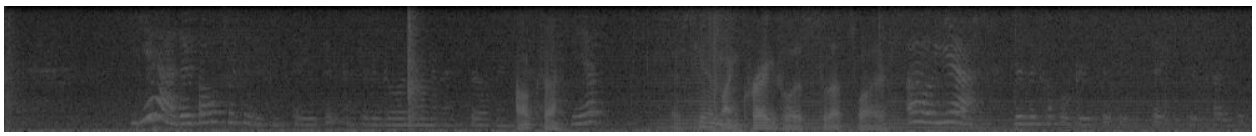


Figure 35: Sample of a spectrogram where patients voice is extracted

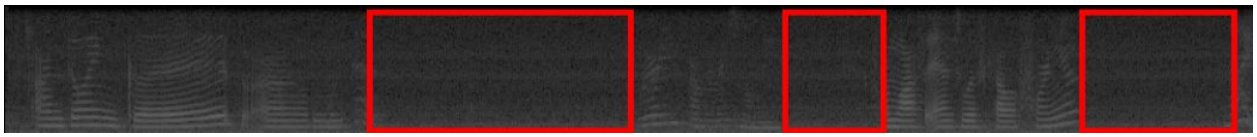


Figure 36: Sample of a spectrogram without extracting patients voice

This extracted audio file can then be further formatted into spectrograms. We split participants into training, test and validation folders based on [66]. Using the extracted audio file earlier, we processed the spectrograms based on this folder structure. Since we framed the spectrograms for 10 seconds of data, we ended up with at least 40 images per folder (denoted as “FilteredImages” in the following sections). However, as we will see in the sections below, we faced many memory errors and suspected that the computer was not able to process so many images at once.

As such, each folder was reduced to only have 1 image per folder (denoted as “FilteredImages2” in the following sections) during some of the experiments.

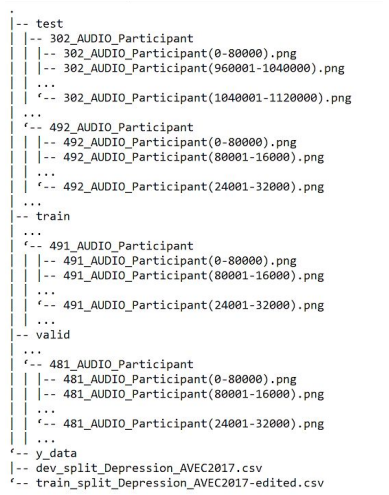


Figure 37: Sample Directory Structure (all images), FilteredImages

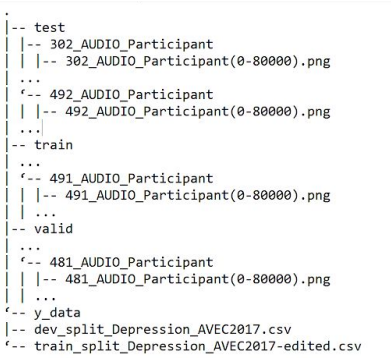


Figure 38: Sample Directory Structure (1 image per folder), FilteredImages2

4.1.2 Cropping Images using Numpy Slicing

A constant problem we faced in doing this project was the lack of memory (see Section 6 for more details). As such, computationally expensive programs took very long to run or could not run at all. To solve this problem, we decided to crop the images using Numpy slicing so as to test out the program. Since images can be converted into Numpy arrays (by using PIL’s Image module or by using OpenCV), we made use of this and sliced a Numpy array with shape (101,

1000, 1) to obtain a Numpy array with shape (101, 100, 1). When testing our code, we used the sliced Numpy arrays instead of the entire array. The Numpy array with shape (101, 1000, 1) was the original spectrogram image while the Numpy array with shape (101, 100, 1) was the cropped spectrogram image with the first 100 pixels out of the 1000. Attempts to use all 1000 pixels failed with a memory error. The code used for Numpy slicing can be found in Appendix B.

However, cropping images came with a shortcoming as well. Cropping images using this method caused a potential loss of crucial information. By using this method to crop images, we ignored 90% of the spectrogram - a Numpy array with shape (101, 900, 1). In this array, there could have been information that would have affected our accuracy. As such, we did not use this in our final model training, but used it to test out the code for the Autoencoders and GANs.

4.2 List of Experiments and Results

4.2.1 Original Setup

Similar to any other experiment, a control or original setup was needed for us to use as a reference point whenever we implemented more code to improve on the accuracy of the model. A Sequential model was used. As mentioned in the sections above, the model has different layers, and its architecture can be seen below. We trained on 4591 samples and validated on 1740 samples. The training and validation set were kept the same for all experiments, and was decided prior to the commencement of all experiments. The number of epochs used was 30 and the window size was 3x3. This model comprises of a combination of 2 Conv2D layers followed by 1 AveragePooling2D layer repeated thrice, followed by 1 Flatten layer and 3 Dense layers. The code for this can be found in Appendix B.

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 101, 1000, 32)	320
conv2d_2 (Conv2D)	(None, 101, 1000, 32)	9248
average_pooling2d_1 (Average	(None, 25, 333, 32)	0
conv2d_3 (Conv2D)	(None, 25, 333, 64)	18496
conv2d_4 (Conv2D)	(None, 25, 333, 64)	36928
average_pooling2d_2 (Average	(None, 6, 111, 64)	0
conv2d_5 (Conv2D)	(None, 6, 111, 96)	55392
conv2d_6 (Conv2D)	(None, 6, 111, 96)	83040
average_pooling2d_3 (Average	(None, 1, 37, 96)	0
flatten_1 (Flatten)	(None, 3552)	0
dense_1 (Dense)	(None, 256)	909568
dense_2 (Dense)	(None, 256)	65792
dense_3 (Dense)	(None, 1)	257
activation_1 (Activation)	(None, 1)	0

Total params: 1,179,041
Trainable params: 1,179,041
Non-trainable params: 0

Figure 39: Model architecture

In order to train the model, the following parameters were needed.

1. x_train & x_test: training dataset, which is initialized by loading the images as an numpy.array of shape (n,h,w,1) where n = number of images, h = height of image in pixel, w = width of image in pixel and followed by a 1 denoting the individual pixel.
2. y_train & y_test: consists of the PHQ8 binary label values as an numpy.array

Both x_train and x_test index corresponds to the labels in y_train and y_test respectively. The test set (x_test and y_test) are used in validation of the model. The y_test labels are not passed into the model for training but are used to validate if the model was able to accurately classify the x_test data. This produces the val_acc score during training.

In order to fit the model using the Sequential model's fit function, the following parameters were needed:

1. x=x_train
2. y=y_train
3. batch_size=32

4. epochs=30
5. validation_data=(x_test, y_test))

The validation accuracy (val_acc) of this model ranged from 55.1% to 59.0%. As it was not very high, we decided to employ various methods to improve its accuracy. These methods will be explained in the next few sections.

Apart from the deep learning algorithms applied, we also have an Android application in place. The framework for this can be found in Figure 40 below. The web server was set up using an EC2 instance and Flask was used to connect the web server and the Android application. This will be explored in further detail in Section 5.

Figure 40: Visualization of flow of events

4.2.2 With k-fold Cross-Validation

As mentioned in previous sections, k-fold Cross-Validation has been proven to be able to reduce overfitting and increase the accuracy of models.

We experimented with different parameters, such as the number of folds used (value of k), window size, and number of epochs used. A summary of the experiments we ran can be found below and the code for this can be found in Appendix B.

K-Fold Cross Validation	No. of Epochs	Window Size	No. of Folds		Amazon / PC?	Duration	Completed?	New Accuracy	Original Accuracy	Increase
#1	30	3x3	10		PC	about 1h	☑	0.7019565218	0.5511494253	0.1508070965
#2	30	2x2	10		PC	about 1h	☑	0.7230434781	0.5816091953	0.1414342828
#3	30	2x3	10		PC	about 1.5h	☑	0.6865217391	0.5609195404	0.1256021988
#4	30	3x3	5		PC	about 30min	☑	0.7530434783	0.5729885056	0.1800549727
#5	30	2x2	5		PC	about 2.5h	☑	0.7043478263	0.5902298851	0.1141179412
#6	30	3x3	4		PC	about 1.5h	☑	0.6983695652	0.5902298851	0.1081396802
#7	25	3x3	4		PC	about 1h	☑	0.7086956523	0.5568965516	0.1517991007
#8	25	2x3	4		PC	about 1.5h	☑	0.7135869563	0.5902298851	0.1233570712
#9	25	3x3	4		PC	about 1.5h	☑	0.7663043478	0.6000000001	0.1663043477

Figure 41: Summary of Experiments for k-fold Cross-Validation

We find that the most significant improvement (18.0%) was made when using k-fold Cross-Validation, with 30 epochs, a window size of 3x3, and 5 folds (Experiment #4). However, upon closer analysis, we realised that the model began to exhibit overfitting properties after 4 folds and 25 epochs. To prove this, we used those 2 parameters as an experiment as well, and found that that gave us the second most significant improvement of 16.6% (Experiment #9) and our third most significant improvement of 15.2% (Experiment #7). Experiment #9 and #7 used 25 epochs, a window size of 3x3, and 4 folds. Our least significant improvement (11.4%) can be seen in Experiment #5, with 30 epochs, a 2x2 window size, and 5 folds. As such, we decided not to continue with testing 2x2 window sizes.

In our code, we used Matplotlib to display the accuracy and validation accuracy as well as loss and validation loss of our model over time. From this, we found it easy to determine whether the model was overfitted, and if so, from which point onwards it was overfitted. An example of these plots can be seen below.

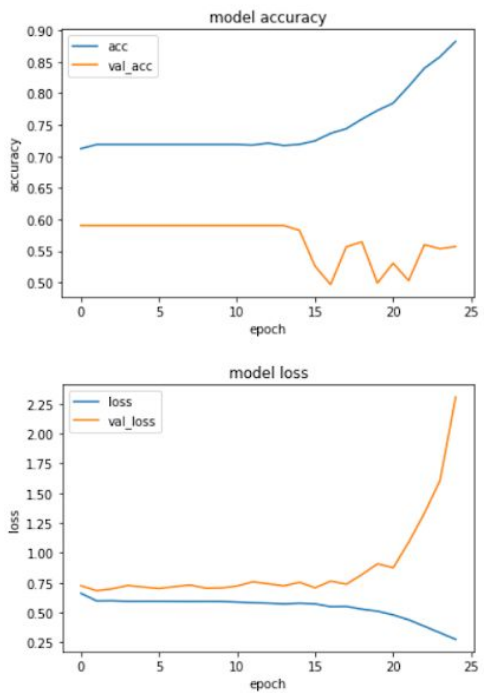


Figure 42: Before k-fold

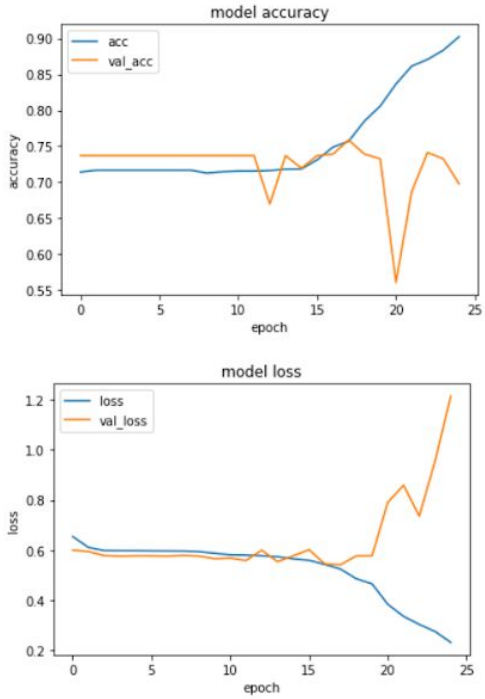


Figure 43: After k-fold (k = 4)

As can be seen in the plots above, after using k-fold Cross-Validation, the model accuracy and the validation accuracy match up more closely. Despite using k-fold Cross-Validation, our model still experienced overfitting at times, although the occurrence was significantly reduced. Thankfully, the accuracy of our model increased by an average percentage of more than 10% per experiment.

4.2.3 Autoencoders

We used autoencoders as a method for image denoising. We trained an autoencoder with the noisy spectrograms as x training and validation data and used the original spectrograms as y training and validation data. During the training, the autoencoder learnt how to extract important features from input images and ignored the image noises. In other words, we used model.fit(x=x_train_noisy, y=x_train_original, validation_data=(x_test_noisy, x_test_original)) when fitting the model.

We conducted 7 experiments for the autoencoders, all with varying parameters. A summary of these experiments can be found below and the code for this can be found in Appendix B.

Autoencoders	Batch Size	Cropped?	Folder Used	Noise Factor	Amazon / PC?	Duration	Completed?	Accuracy
#1	50	No	FilteredImages	0.5	new PC	crashed	☐	
#2	25	No	FilteredImages2	0.1	new PC	1 min	☑	0.003557948582
#3	25	No	FilteredImages2	0	new PC	1 min	☑	0.007010887813
#4	25	Yes	FilteredImages2	0.5	new PC	1 min	☑	0.0007676635533
#5	128	Yes	FilteredImages2	0.5	new PC	1 min	☑	0.001089532705
#6	128	Yes	FilteredImages	0.5	new PC	3 min	☑	0.005114746599
#7	128	Yes	FilteredImages	0	new PC	4 min	☑	0.00491684387

Figure 44: Summary of Experiments for Autoencoders

As can be seen in Figure 44 above, we experimented with many different parameters, namely the batch size, cropping the images (see Section 4.1.2), alternating between using FilteredImages and FilteredImages2 and the noise factor. Despite changing all of these parameters, the accuracy still remained very low for all experiments, with the highest being 0.702% (Experiment #3). However, upon closer analysis, we find that the noise factor for that experiment was 0, meaning that no noise was added to the spectrograms. Theoretically, the accuracy should have been quite high. However, the results strongly differed from what we expected. We suspect that the reason for this could be the fact that no two spectrograms are similar to each other, and hence using spectrograms for x_train, x_test, y_train and y_test could have caused this dip in accuracy (in contrast, the CNN model uses spectrograms for x_train and x_test but uses the PHQ8 binary values label for y_train and y_test).

We conducted 6 other experiments in addition to the 7 above. However, they either crashed or were using the wrong parameters. We only realised this crucial mistake towards the end of our project. However, we logged down our mistakes so as to ensure that we would not make them again.

Mistakes	Batch Size	Cropped?	Folder Used	No. of Epochs	Noise Factor	Amazon / PC?	Duration	Completed?	Accuracy	Mistake	K-Fold
#1	50	N/A	FilteredImages	10	0.5	Amazon	around 11h	☑	0.0074	used flatten, dense layers. model.fit(x=x_train_noisy, y=npY, validation_data=(x_test_noisy, npY_test)) rather than using training against the original images	
#2	32	N/A	FilteredImages	10	0.5	Amazon	about 13h	☑	0.005722269231	same as above	
#3	128	Yes	FilteredImages	50	0.5	new PC	5 min	☑	0.01988560825	same as above	
#4	128	Yes	FilteredImages	1000	0.5	new PC	crashed	☐		same as above	
#5	128	No	FilteredImages	20		new PC	about 20h	☑	0.7187976477	same as above	0.7167755993
#6	50	No	FilteredImages	20	0.5	new PC	about 17h	☑	0.000767469614	clipped images	

Figure 45: Summary of Experiments for Autoencoders (Mistakes)

For Experiments #1 to #5, we mistakenly added Flatten and Dense layers into our autoencoder. When fitting the model, we used model.fit(x=x_train_noisy, y=npY, validation_data=(x_test_noisy, npY_test)) rather than using training against the original images. This led to inaccuracy in the results and was not what we were truly testing for. For Experiment #6, we realised that our images were clipped using Numpy's clip function. This function threshold all the negative values to zero and all the values greater than one to one. This meant that all pixel values were between zero and one. However, this could have affected our accuracy as it was not what we aimed to achieve with autoencoders.

Based on all our experiments, we can infer that the code will not crash and would work either if we reduced the dataset size (by using FilteredImages2 instead of FilteredImages), or if we cropped the images using Numpy slicing (see Section 4.1.2). However, cropping images would cause a severe loss of information (almost 90% of the image).

Upon closer analysis, one possible reason for the low accuracy could be the fact that our input images were spectrograms, and some parts of the spectrograms could have been interpreted as noise rather than important features. To illustrate this, we will provide an example of our original spectrogram as well as one that has gone through the autoencoder.

Figure 46: First value in training and test data before adding noise

Figure 47: First value in training and test data after adding noise

4.2.4 GANs

In our project, we planned to use GANs to denoise the spectrogram. To test the concept, we first tried it on the MNIST database. The MNIST database of handwritten digits, which has a training set of 60,000 examples, and a test set of 10,000 examples, was used. In this case, the discriminator is trying to determine which data is from the dataset and which ones are not. At the same time, the generator is creating new, synthetic images that it passes to the discriminator. It does so in the hopes that they will be deemed authentic even though they are fake [27]. The goal of the generator is to generate passable handwritten digits, or in simple terms, to lie without being caught [27]. The goal of the discriminator is to identify images coming from the generator as fake. The discriminator network is a standard convolutional network that can categorize the images fed to it, a binomial classifier labeling images as real or fake and the generator is similar to an inverse convolutional network [27]. While a standard convolutional classifier takes an image and downsamples it to produce a probability, the generator takes a vector of random noise and upsamples it to an image. The first throws away data through downsampling techniques like Max Pooling, and the second generates new data [27]. Below are examples of the output of the generator of a typical GAN that uses the MNIST database.

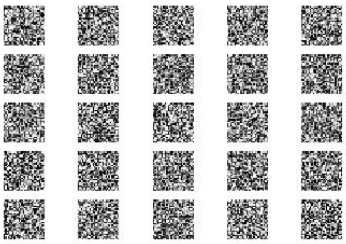


Figure 48: Output from first epoch

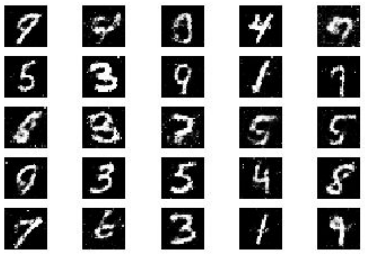


Figure 49: Output from last epoch

Based on our research, we found that there are many different types of GANs available. In our case, we used CGAN for image denoising, and the code for this can be found in Appendix B. This code was run for 20 epochs. Gaussian random noise was implemented for noise/artefact generation. Below are examples of the images generated.

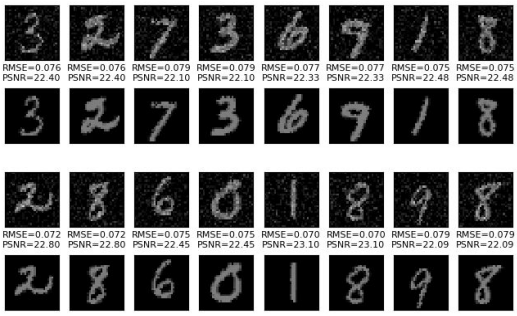


Figure 50: Output from epoch 0

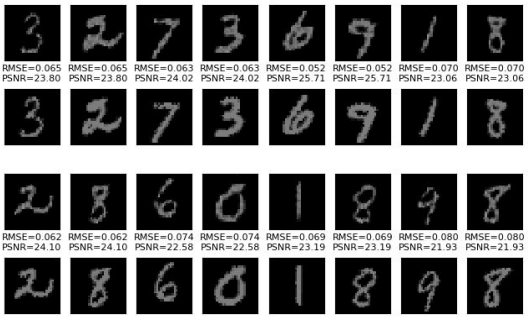


Figure 51: Output from epoch 20

As can be seen in the images above, the network was able to denoise the images. However, since our actual dataset is not MNIST but rather our spectrograms, we attempted to implement the code on the spectrograms. The code for this can be found in Appendix B.

The code was originally intended to run on 28x28 images, which is the size of images in the MNIST database. In terms of a Numpy array, each image had the shape (28, 28, 1). Even while running on these images, the hardware lagged and took close to 12h to complete. After adapting the code to suit our 101x1000 images (each image could be represented as a Numpy array with

shape (101, 1000, 1)), we faced a memory error. After much analysis, we found that this was probably due to the fact that the discriminator and generator networks were running concurrently on such huge images, hence leading to a memory error. As such, we implemented cropping of these images using Numpy slicing (see Section 4.1.2). Each image now was 101x100 and could be represented as a Numpy array with shape (101, 100, 1). This reduced the size of our dataset and the CGAN was able to be trained. However, the accuracy was very low. After trying to tweak the model and the parameters, the accuracy was still low. We feel that this might be due to the fact that after cropping the image, we have lost too much information for the image to be reconstructed. As such, we decided to not use it.

4.2.5 OpenCV Non-Local Means Image Denoising

As seen in Section 3.7.6, OpenCV can be used for non-local means image denoising. We implemented OpenCV's cv2.fastNIMeansDenoising() since our image was grayscale.

Figure 52: Before and after OpenCV Non-Local Means Image Denoising

However, as seen in Figure 52, the spectrogram obtained after denoising appears blurrer than the one before denoising. Some of the features of the spectrogram were not as clear as before. A summary of the experiments we ran can be found below and the code for this can be found in Appendix B.

OpenCV Denoising	Cropped?	Folder Used	No. of Epochs		Amazon / PC?	Duration	Completed?	Accuracy
#1	No	FilteredImages2	25		new PC	3 min	☑	0.5714285714
#2	No	FilteredImages	25		new PC	3h	☑	0.5959770114

Figure 53: Summary of Experiments for OpenCV Non-Local Means Image Denoising

Using our original setup in Section 4.2.1, we ran the CNN model (without k-fold Cross-Validation) on our denoised images. However, doing so, we found that our accuracy

dropped to 59.6% as seen in Figure 53. We tested our code on both FilteredImages2 and FilteredImages. Perhaps this could be due to the fact that this denoising method is typically applied to digital images and reducing noise such as Gaussian noise or salt and pepper noise. In these applications, the main goal is to make the images seem denoised, and make them look aesthetically better. However, our project required it to be applied to spectrograms of speech data and remove the parts that were not relevant to the features that our CNN model was extracting. It was possible that it could not differentiate the noise and actual human voice as expected, and could have removed some of the crucial features (explained above when mentioning the blurring of the spectrograms after the denoising). If this was the case, some of the features extracted usually through the CNN model could have been lost, hence causing a decrease in our accuracy.

4.2.6 Noisereduce

To prove that this algorithm is able to successfully remove noise, we tested it out with one of our own existing .wav files. The voiced speech of a typical adult male will have a fundamental frequency from 85 to 180 Hz, and that of a typical adult female from 165 to 255 Hz [67]. However, for good quality, a bandwidth of 80 Hz to 8000 Hz can be used [68]. As such, for our experiments with Noisereduce, the noise added had a minimum frequency of 400 Hz and a maximum frequency of 8000 Hz. The function to add noise was found in the Noisereduce GitHub repository. The entire process took less than 5 seconds for an audio file of 14 seconds. The code for the Noisereduce algorithm can be found in Appendix B.

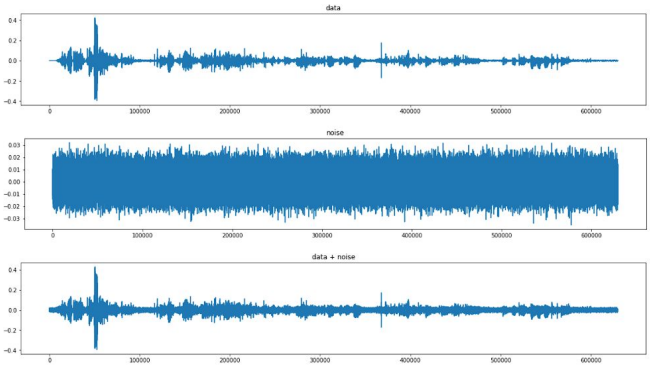


Figure 54: Visualisation of data in original .wav file, noise, and noise added to data in the .wav file

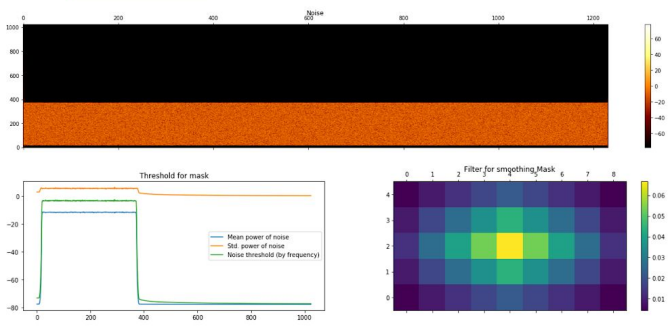


Figure 55: Visualisation of statistics of noise and threshold (based on statistics and sensitivity)

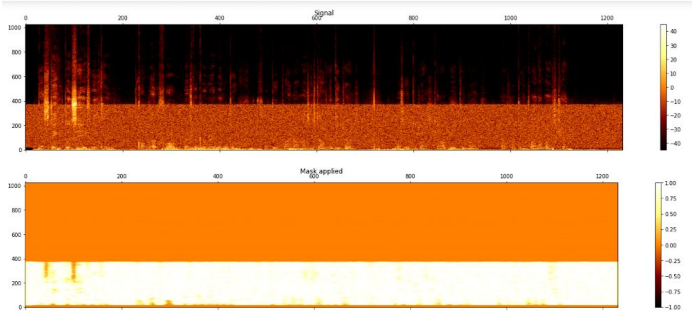


Figure 56: Visualisation of FFT of signal and mask applied (signal FFT vs threshold)

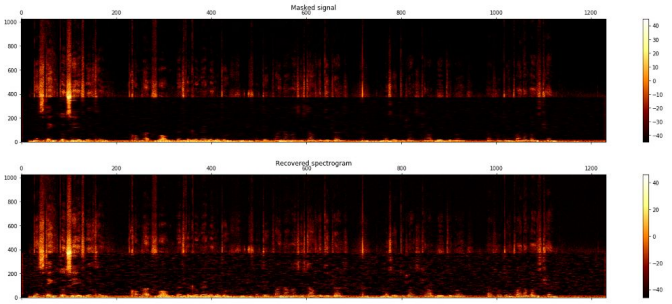


Figure 57: Visualisation of masked signal and recovered spectrogram

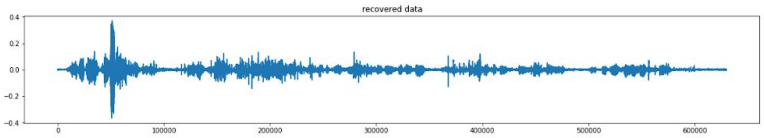


Figure 58: Visualisation of data after algorithm is complete

As can be seen above, noise can be removed from the audio data in our .wav file by using this algorithm. Hence, we implemented and adapted the code in our Flask application. By doing so, the .wav file recorded on our Android application can be processed and denoised in our Flask application after it has received the file.

Despite being unable to denoise the training and test images in our model, we are able to denoise the real-time audio coming in. Although we do not have a definite number, we are certain that this has increased the accuracy of our prediction.

4.3 Summary of Findings

For k-fold Cross-Validation, we have decided to use 25 epochs, a 3x3 window size, and 4 folds as the standard model structure.

For autoencoders, the validation accuracy was very low despite us varying parameters. Nevertheless, we were able to learn how to implement autoencoders based on our model and dataset. Autoencoders may not have been the best denoising method for our spectrograms, but have been proven in other applications to be highly accurate and precise.

For GANs, we were only able to implement image denoising CGAN on our spectrograms. There are many other GANs that have been developed but due to time constraints, we were not able to test them. Using CGAN, we were able to successfully denoise the images in the MNIST database. However, the denoising of our spectrograms was plagued with many problems from

low accuracy to memory errors. Nevertheless, it was a good attempt at denoising the spectrograms and has a lot of potential.

For OpenCV Non-Local Means Image Denoising, we find that the accuracy of our model decreased from 60.0% to 59.6%. As explained above, this could be due to the algorithm picking up the wrong parts of our spectrogram to denoise, hence leading to a decrease in accuracy. We decided not to use this in our final implementation in case it affected our accuracy when getting the prediction for the real time audio. Nevertheless, it was a good denoising technique, but perhaps one that was not meant for our spectrograms.

For Noisereduce, we have implemented the algorithm in our Flask application to denoise the audio before processing it into spectrograms (which in turn are processed through the CNN model).

Our detailed implementation can be found in Section 5 below and our future plans will be explored in more detail in Section 6.

5. Implementation

5.1 Summary and Full Setup

Figure 59 below shows a general rundown of how the system works. Upon pressing the start button in the application, the audio recording process is commenced. The data is periodically sent to the server. Upon receiving all the data (indicated when the user presses the stop button in the application), the audio is then processed through various means that we will cover later in the report. The backend server receives our data in the form of a .wav file, which is then converted into a spectrogram and is fed into our CNN model (with various enhancements that will be covered later). A prediction of the probability of the user being depressed is returned and it is sent back to the application, where it is displayed on the screen.

Figure 59: Visualisation of flow of events

5.2 Android Application

5.2.1 Application Summary

Our application will be the frontend to accessing the deep learning model for predictions. The application also serves as a platform for users to interact with our system. Here, the users will be able to record their voice, send it to the backend server, and return a prediction. Since the application is still a prototype, we have designed the application with both commercial use and

industry use (namely a hospital setting) in mind. Apart from just returning a prediction from the server, we have implemented a history log, daily mood tracker and audio visualisation tools into the application. We have also designed a preferences menu for users to adjust their preferences if needed, however, this has not been made functional yet.

The application will periodically send data to the back end server at small intervals. This will allow the application to receive prediction scores during the course of the screening, however this will not be displayed on the screen. When the user stops the recording process, the recorded audio is sent to the backend server and the final aggregated prediction scores will be presented.

5.2.2 Application Interface

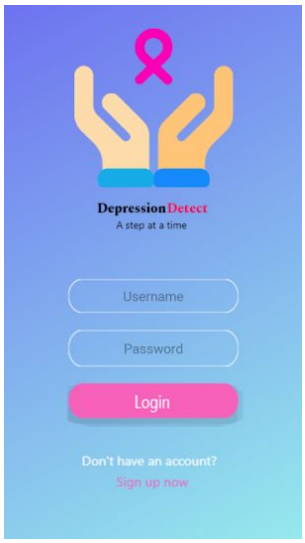


Figure 60: Login Screen

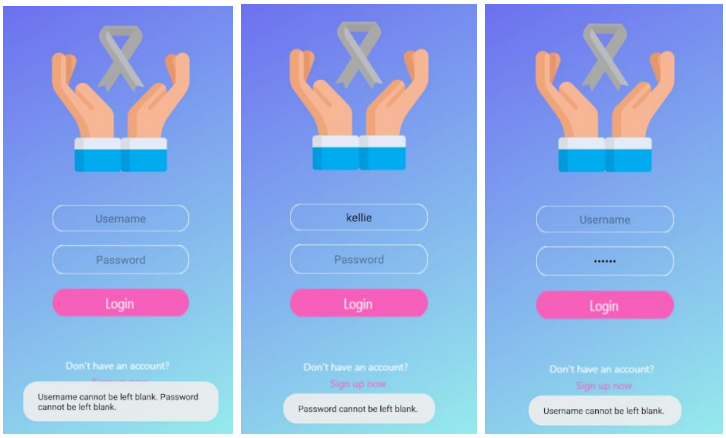


Figure 61: Toast messages shown when user does not key in right details

First we have the login screen (Figure 60). Upon launching the application, this will be the first screen the user sees. To use the application, the user will be prompted to enter his/her username and password. If either or both fields are not entered, a Toast message will be shown at the bottom of the screen and the user will not be allowed to login (Figure 61).

Figure 62: Request for Access to Microphone

Figure 63: Request for Access to Storage

Upon clicking the Login button, users who are using this application for the first time will be prompted with 2 permission alerts, namely access to the phone's microphone (Figure 62) and to external storage (Figure 63). These permissions are necessary as our application requires users to record audio using the built-in microphone on the device and also store information on the device's internal/external storage.

Identifications such as NRIC or FIN numbers was not used as the username as this could potentially breach Singapore's Personal Data Protection Act [69]. However, should this

application be deployed in a hospital or psychiatrist setting, the username could either be linked to the staff's ID or the patient's ID.

Should they wish to, users can change their username or password in the preferences menu (see below). However, as this application is just a prototype and this is not the main focus of our product, we did not verify the username and password. If we wanted to implement it, however, the authentication could be done with AccountManager, a class which provides access to a centralized registry of the user's online accounts [70]. The user enters credentials (username and password) once per account, granting applications access to online resources with "one-click" approval [70].

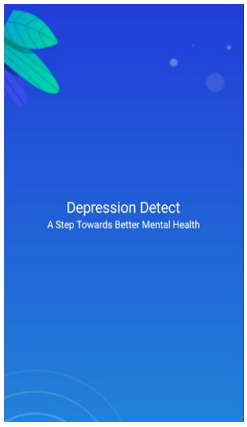


Figure 64: Animation

Figure 65: Menu Screen

Next we have the menu screen. A brief animation will play and the user will be brought to the menu screen. On this screen, users have 4 buttons to choose from, namely History, Detect, Preferences and Help. In our application, we have programmed all buttons except Help to work. The History, Detect and Preferences screens will be explained in more detail below.

The Help button was not part of our original aims and as such, we did not program it. However, it could serve as an avenue for users to seek help and reach out to a professional if needed. A chatbot was one of the potential ideas we had in mind as well.

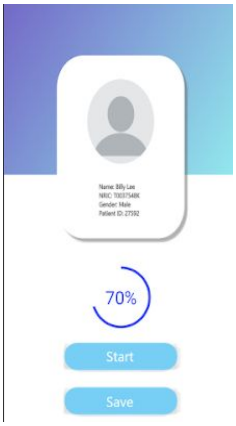


Figure 66: Detect Screen

This is the detect screen. The information of the user will be displayed here. Should this be deployed in a hospital setting, the patient's medical information can be displayed here (e.g. Name, NRIC, Gender, Patient ID). To commence the detection process and recording of audio, users can click on the start button, which will bring them to another screen. A circular progress bar has been implemented to indicate the results of the test. However, the results will only be shown after the user has successfully sent the data to the server fully. By default, the results will be 0.0%. The save button will allow the user to save the details of the test to the history log, which can be accessed through the history button in the menu.

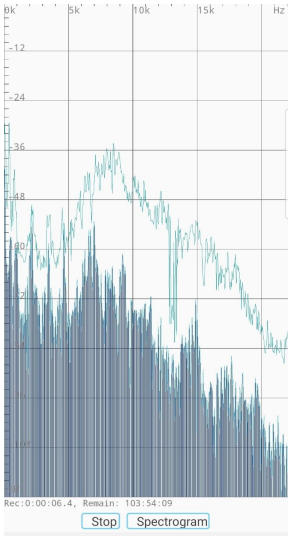


Figure 67: Spectrum of Real Time Audio

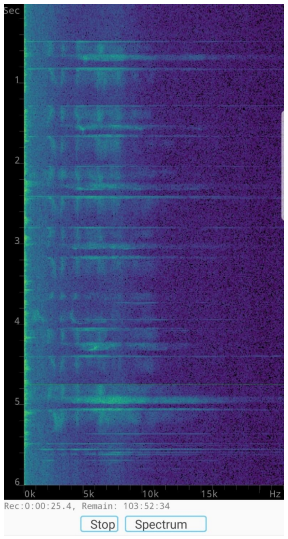


Figure 68: Spectrogram of Real Time Audio

Upon clicking the start button on the detect screen, the user will be brought to a recording screen. Based on our research in Section 3.6.1, we have decided to use a spectrum and a spectrogram for visualising the real time audio signal. More details about this will be explained in Section 5.2.4 and 5.2.5.

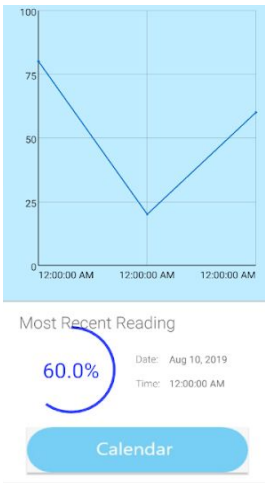


Figure 69: History Screen

Figure 70: Calendar Screen

On the history screen, the user will be able to see his/her history log as mentioned above. This is displayed and visualised through a graph. They will also be able to look at the prediction of their most recent reading alongside the corresponding date and time.

Upon clicking the calendar button, users will be directed to another screen (Figure 70). This allows for developments such as a daily mood tracker, where users can input their moods based on 5 moods ranging from extremely unhappy to extremely happy. This is based on [71], where it is encouraged for users to record their daily mood, creating a year in pixels. Below the calendar, a graph can be drawn to show users the frequency of the moods they have recorded through the application. As such, apart from the readings for the detection of depression being recorded, their moods can also be recorded, allowing depressed users to see that they can have good days despite their depression.

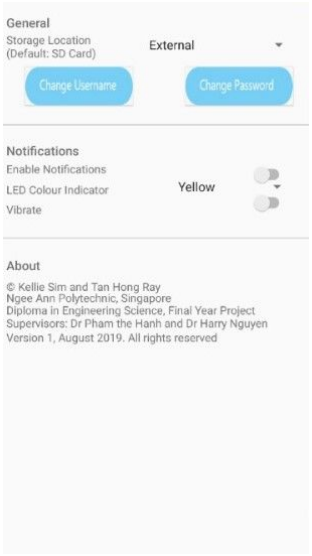


Figure 71: Preferences Screen

This is the preferences screen, where users can change settings such as storage location, their username/password and set custom notifications. As of now, this screen is not functional yet as most of the features have yet to be implemented in our application. For example, the storage location setting has not been implemented as it is set to the external storage by default in our

code. The username and password has not been fully implemented as mentioned above. The notifications system has not been implemented as well, but if it were to be implemented, users could choose to be notified about checking in and using the application daily.

However, most of the features seen here are based on the assumption that the application will be used commercially rather than in a hospital setting. In a hospital setting, perhaps settings more relevant to healthcare professionals can be made available.

5.2.3 Material Design

Although our application did not use the actual language, we have borrowed some of the design principles from Material Design. This allowed our application to be more aesthetically pleasing and user friendly. For example, we design the different buttons for different levels of importance, and where we want the attention of the user to be. We also added animations to transition from activity to activity to allow the application to feel more fluid.

5.2.4 Application Animations

Since animations can add visual cues that notify users about what is going on in the application, are especially useful when the UI changes state (e.g. when new content loads or new actions become available) and add a polished look to the application [72], we implemented animations in our application. These applications were mainly used when transitioning from one activity to the next.

5.2.5 Audio Recording

On the recording screen, the start button near the bottom of the screen initiates and calls an AudioRecord object. The audio data is returned in the form of PCM data, which is then processed and saved into a .wav file. This .wav file is automatically saved to the phone's or emulator's SD card. We will use this .wav file for processing using the machine learning model later on. The other Android APIs, AudioTrack, MediaRecorder and MediaPlayer were not used.

We did not use AudioTrack as playback was not required. We did not use MediaRecorder as our project required real-time audio processing and visualisation, which could only be done using AudioRecord. Furthermore, there was no need for any individual buffers of data in our application as it would be processed through the Flask app. We did not use MediaPlayer as it cannot be used if additional sound processing and visualisation is to be done on the audio data played or recorded [44].

The sampling rate used in our application was 8000 Hz. When recording, an audio visualisation will be displayed to the user and this visualisation will be explained in the next section.

5.2.6 Audio Visualisation

As mentioned in Section 3.6.1, there are many methods of visual representations of voiced data, namely waveforms, spectra and spectrograms.

We find that we can define the visual representations as such:

1. A waveform is a curve showing the shape of a wave at a particular time. [73]
2. A spectrum shows the frequency content of an entire signal and is a 1-dimensional function of amplitude (vertical axis) vs frequency (horizontal axis). [74]
3. A spectrogram shows how the frequency content of a signal changes over time and is a 2-dimensional function of amplitude (brightness or color) vs frequency (vertical axis) vs time (horizontal axis). [74]

All of these can be incorporated and displayed real-time through the Android application, achieving one of our objectives mentioned in Section 2.1.

5.2.7 SQLite Database

In our application, we used SQLite to store the time, date and results when the user has used the application. When the user opens the history screen, the application will retrieve the information

from the SQLite database and plot the information out onto a line graph as seen in Figure 69 above. This will allow the user to visualise the data and find trends in the data.

5.3 Backend Server

In our prototype, we used our laptops to host a backend server. On this backend server, RESTful APIs were called and the .wav file was taken in. The prediction was done on the server and it is eventually returned as a JSONObject which can then be displayed through the Android application.

The Android application and the backend server are connected through Flask. Flask is a micro framework library for Python that enables the front end application to connect to the server through the means of RESTful APIs [75]. Upon running the server, the model that was generated in Section 4 (after k-fold Cross-Validation) will be loaded using the load_model function found in keras.models. Loading the model prior to an API call is done to prevent taking up too much resources each time a call is made.

The API Service to make a prediction is '/users/<userID>/analysis/<sessionID>' where the <userID> and <sessionID> are variables for identifying and organizing the files. The userID in this case refers to the username entered by the user in the login screen while the sessionID is randomly generated at every instance of the application run. As shown in Figure 72, in the uploads folder, the uploads are organized in <userID>/<sessionID>. When a call is made, the .wav file is organized and uploaded in the respective directory. Spectrograms will also be generated based on the .wav file input.

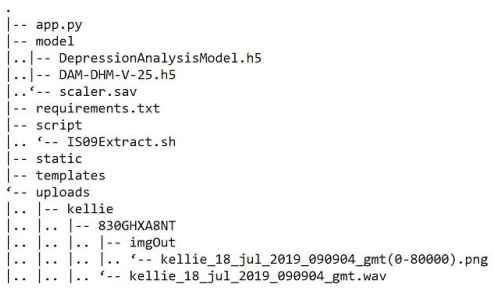


Figure 72: Sample Directory Structure, Backend Server

6. Problems Faced

In the course of any project, it is inevitable that problems are faced. For our project, due to various reasons such as time constraints or hardware problems, there were many issues that we were unable to solve.

From 25 March 2019 to 8 May 2019, we used an ASUS UX331UN laptop to run most of our code. This laptop has a NVIDIA GEFORCE 940MX GPU and Intel Core i7 Gen Processor. As such, the training of the models took a very long period of time. This resulted in a smaller number of experiments being run than expected.

From 9 May 2019 to 15 July 2019, most of the training was done on a personal computer with a NVIDIA GEFORCE GTX1060 GPU 3GB and Intel Core i5 Gen Processor. The small amount of Vram on the GPU has led to many errors as it is unable to handle large datasets and computationally intensive models.

From 16 July 2019 to 10 August 2019, we used a computer with a NVIDIA GEFORCE RTX2070 GPU 8GB and Intel Core i7 Gen Processor. Although the specifications of the computer were significantly better, we were unable to train as many experiments as planned as we were not able to use the computer over the weekend. This led to a lot of time wasted. Furthermore, we were unable to successfully run our code on the GPU version of Tensorflow. Despite spending almost a week trying to figure out why this was the case, we were unable to find any solution for this. This resulted in all the code being run on our CPU and hence taking much longer than expected (e.g. GPU was expected to take around 30 minutes, but CPU took 5 hours).

Apart from our personal computers, some of our training was done on an AWS Virtual Machine, which was connected using an EC2 instance. When trying to load our code, we were plagued

with this issue for a few weeks. As such, we were unable to run some of our code and a lot of time was wasted.

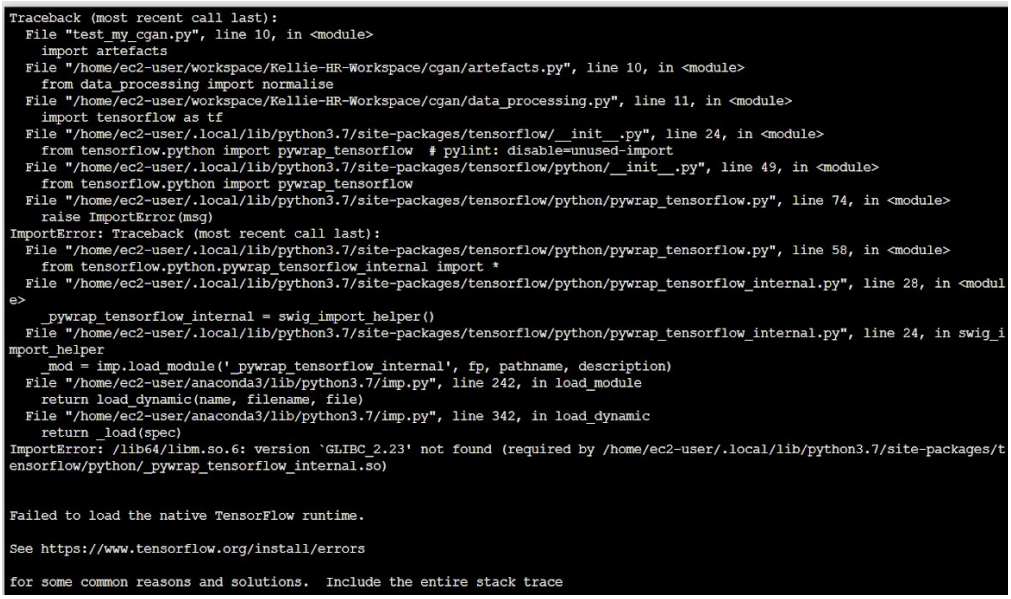


Figure 73: Error loading Tensorflow runtime

We tried to send the spectrogram from the web server to the application using Flask's send_file and send_from_directory methods. However, we were met with a memory error. There was a lack of memory/RAM on the device used for testing. As a result, we decided to display the real time spectrograms instead.

Hardware limitations also caused errors in the training of our models. In the training of the denoising with GANs and autoencoders, we were able to denoise the images from the MNIST dataset but were unable to train the models on the full spectrogram. This was due to a memory error as we did not have enough memory to train the models. To solve this issue, we tried to decrease the batch size and crop the images but this led to the accuracy being very low. More details about this can be found in the respective sections above.

7. Conclusion

Depression is a mood disorder characterized by physical, emotional, cognitive and behavioural symptoms “causing significant distress or severely impacting social, occupational or other important life areas”. However, due to the expensive cost of seeing a psychiatrist for a professional assessment or the stigma that comes with mental illnesses, many depressed people end up not diagnosed and this may result in them suffering in silence. As such, we aim to provide a reliable method to determine if a user is depressed. Even though this application will not cure depression fully, it serves as a way out for the mentally ill who face stigma and discrimination and do not want to seek a professional diagnosis.

We have managed to achieve an average validation accuracy of 75% after all our experiments. Through this project, we aim to apply deep learning models to determine the probability of a user being depressed, provide an accessible and accurate method to detect depression using non-verbal vocal features, enhance technology used for early detection of depression so as to allow for early detection, provide a visual representation of audio signal through the application, and if possible, highlight portions that link to depression being detected. With these aims in mind, we have learnt more about deep learning models and architecture, as well as different techniques to improve the accuracy of the models. We have studied and learnt more about both image and audio denoising techniques as well as their efficiency. We have also learnt more about developing an Android application. Through developing the various screens of the application, we have learnt about SQLite databases, animation and good design techniques. By tying in our knowledge from last year's audio recording and visualisation features in our application, we were able to come up with a whole system that was able to connect to a web server and send/receive JSON data from it.

This project would not have been possible without our supervisors, Dr Harry Nguyen and Dr Pham The Hanh.

8. Future Plans

Although it might be the end of our final year project, it need not be the end of this project. This application still has a lot of potential to be very impactful on society. If we were to work on the project further, we would try and implement a few things.

Firstly, we plan to show a heatmap of the spectrogram to display which part of the spectrogram caused our model to behave and produce an output in a specific way (i.e. circle/highlight where depression is detected by our model). This can be achieved by using TensorflowJS to visualise the various layers and activation maps in our model. We did attempt this but due to time constraints, we were not able to complete it. This visualisation might allow healthcare professionals to gain a better understanding of depression and mental illnesses as a whole.

Next, we aim to implement a help button in our Android application. This could serve as an avenue for users to seek help and reach out to a professional if needed. A chatbot was one of the potential ideas we had in mind.

Next, for improving the accuracy of our model, we attempted to implement denoising techniques such as GANs and autoencoders to provide a cleaner spectrogram to train our model. Although we did succeed in denoising images from the MNIST dataset through these methods, we were unable to replicate the results exactly with the spectrograms. If we were to work on this further, we would try to fix the issues we encountered or find other ways of denoising.

Lastly, we plan to test out if our CNN model is able to learn from raw audio rather than just the spectrogram images for our case. Research has been done on this application and the results were promising. As such, we plan to implement it in our context to hopefully improve the accuracy of our model.

Appendix A - List of Figures

Appendix B - Codes Used (Python)

```
def importImages(listOfImgDir):
    myFolder = []
    for folder in listOfImgDir:
        myImages = []
        for image in folder:
            # original image
            img = cv2.imread(image)

            # cropped image
            cropped_img = img[0:100, 0:1000]

            myImages.append(cropped_img)
        myFolder.append(np.array(myImages))
    return myFolder
```

Code for Numpy slicing/cropping

```
model = Sequential()
model.add(Conv2D(32, (2, 3), padding='same', input_shape=input_shape, activation='relu'))
model.add(Conv2D(32, (2, 3), padding='same', activation='relu'))
model.add(AveragePooling2D(pool_size=(4, 3)))
model.add(Conv2D(96, (2, 3), padding='same', activation='relu'))
model.add(Conv2D(96, (2, 3), padding='same', activation='relu'))
model.add(AveragePooling2D(pool_size=(4, 3)))
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dense(256, activation='relu'))
model.add(Dense(1))
model.add(Activation('sigmoid'))
model.compile(loss='binary_crossentropy',
              optimizer='adadelta', #adam
              metrics=['accuracy'])
print(model.metrics_names)
model.save_weights('model.h5')
callbacks = [EarlyStopping(monitor='val_acc', patience = 10)]
hist=model.fit(x=X_train, y=y_train, batch_size=32, epochs=numEpochs, callbacks=callbacks_list, validation_data=(X_val, y_val))
_, val_acc=model.evaluate(x=X_val, y=y_val, verbose=1)
model.load_weights('model.h5')
model.summary()
print("acc: ", np.mean(hist.history['acc']))
print("val_acc: ", val_acc)
```

Code for ‘Original Setup’

```
# summarise history for accuracy
plt.plot(hist.history['acc'])
plt.plot(hist.history['val_acc'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['acc', 'val_acc'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(hist.history['loss'])
plt.plot(hist.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['loss', 'val_loss'], loc='upper left')
plt.show()
```

Code for printing Matplotlib plots for accuracy and loss

```
# k-fold cross validation, k = n_folds

n_folds = 4
count = 0
cv_scores, model_history = list(), list()
for _ in range(n_folds):
    # split data
    X_train, X_val, y_train, y_val = train_test_split(newNPX, npY, test_size=0.10, random_state = np.random.randint(1,1000, 1)[0])
    # evaluate model
    model, test_acc = evaluate_model(X_train, X_val, y_train, y_val)
    count += 1
    cv_scores.append(test_acc)
    model_history.append(model)
    print('K-Fold has ran ', count, ' time(s)')

print('\nModel Accuracy after all K-Fold: ', (np.mean(cv_scores)))
```

Code for k-fold Cross-Validation

```
def autoencoder(input_img):
    #encoder
    conv1 = Conv2D(32, (3, 3), activation='relu', padding='same')(input_img)
    conv6 = Conv2D(32, (3, 3), activation='relu', padding='same')(conv1)
    pool1 = MaxPooling2D(pool_size=(2,2))(conv6)
    conv2 = Conv2D(64, (3, 3), activation='relu', padding='same')(pool1)
    conv7 = Conv2D(64, (3, 3), activation='relu', padding='same')(conv2)
    pool2 = MaxPooling2D(pool_size=(2,2))(conv7)
    conv3 = Conv2D(128, (3, 3), activation='relu', padding='same')(pool2)
    conv8 = Conv2D(128, (3, 3), activation='relu', padding='same')(conv3)

    #decoder
    conv4 = Conv2D(128, (3, 3), activation='relu', padding='same')(conv3)
    conv9 = Conv2D(128, (3, 3), activation='relu', padding='same')(conv4)
    up1 = UpSampling2D((2,2))(conv9)
    conv5 = Conv2D(64, (3, 3), activation='relu', padding='same')(up1)
    conv10 = Conv2D(64, (3, 3), activation='relu', padding='same')(conv5)
    up2 = UpSampling2D((2,2))(conv5)
    decoded = Conv2D(1, (3, 3), activation='sigmoid', padding='same')(up2)

    return decoded
```

Code for autoencoder, with encoder and decoder

```
noise_factor = 0.5
x_train_noisy = newNPX + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=newNPX.shape)
x_valid_noisy = newNPX_v + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=newNPX_v.shape)
x_test_noisy = newNPX_test + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=newNPX_test.shape)
x_train_noisy = np.clip(x_train_noisy, 0., 1.)
x_valid_noisy = np.clip(x_valid_noisy, 0., 1.)
x_test_noisy = np.clip(x_test_noisy, 0., 1.)
```

Code to add noise onto images in autoencoder

```
f = config.base_number_of_filters
k = config.kernel_size
s = config.strides
sz = config.train_size
c = config.channels
pad = padding_power_2((sz, sz))

if sz <= 128:
    raise RuntimeError("Input size must be larger than 128 for this U-Net model")

inputs = tf.keras.layers.Input((sz, sz, c), name="ginput")
inputs_pad = tf.keras.layers.ZeroPadding2D(pad, name="gpad")(inputs)
```

```
# Encoder layers
# Input is sz x sz x c
ge1 = tf.keras.layers.Conv2D(f, k, s, padding="same", name="geconv1")(inputs_pad)
# Input is sz2 x sz2 x f
ge2 = tf.keras.layers.LeakyReLU(config.leak, name="geact1")(ge1)
ge2 = tf.keras.layers.Conv2D(2*f, k, s, padding="same", name="geconv2")(ge2)
ge2 = tf.keras.layers.BatchNormalization(name="gebn2")(ge2)
# Input is sz4 x sz4 x 2f
ge3 = tf.keras.layers.LeakyReLU(config.leak, name="geact2")(ge2)
ge3 = tf.keras.layers.Conv2D(4*f, k, s, padding="same", name="geconv3")(ge3)
ge3 = tf.keras.layers.BatchNormalization(name="gebn3")(ge3)
# Input is sz8 x sz8 x 4f
ge4 = tf.keras.layers.LeakyReLU(config.leak, name="geact3")(ge3)
ge4 = tf.keras.layers.Conv2D(8*f, k, s, padding="same", name="geconv4")(ge4)
ge4 = tf.keras.layers.BatchNormalization(name="gebn4")(ge4)
# Input is sz16 x sz16 x 8f
ge5 = tf.keras.layers.LeakyReLU(config.leak, name="geact4")(ge4)
ge5 = tf.keras.layers.Conv2D(8*f, k, s, padding="same", name="geconv5")(ge5)
ge5 = tf.keras.layers.BatchNormalization(name="gebn5")(ge5)
# Input is sz32 x sz32 x 8f
ge6 = tf.keras.layers.LeakyReLU(config.leak, name="geact5")(ge5)
ge6 = tf.keras.layers.Conv2D(8*f, k, s, padding="same", name="geconv6")(ge6)
ge6 = tf.keras.layers.BatchNormalization(name="gebn6")(ge6)
# Input is sz64 x sz64 x 8f
ge7 = tf.keras.layers.LeakyReLU(config.leak, name="geact6")(ge6)
ge7 = tf.keras.layers.Conv2D(8*f, k, s, padding="same", name="geconv7")(ge7)
ge7 = tf.keras.layers.BatchNormalization(name="gebn7")(ge7)
# Input is sz128 x sz128 x 8f
ge8 = tf.keras.layers.LeakyReLU(config.leak, name="geact7")(ge7)
ge8 = tf.keras.layers.Conv2D(8*f, k, s, padding="same", name="geconv8")(ge8)
ge8 = tf.keras.layers.BatchNormalization(name="gebn8")(ge8)
# Input is sz256 x sz256 x 8f
```

```
# Decoder layers with skip connections
gd1 = tf.keras.layers.LeakyReLU(0.0, name="geact8")(ge8)
gd1 = tf.keras.layers.Conv2DTranspose(8*f, k, s, padding="same", name="gdconv1")(gd1)
gd1 = tf.keras.layers.BatchNormalization(name="gdbn1")(gd1)
gd1 = tf.keras.layers.Dropout(config.dropout_rate, name="gddrop1")(gd1)
# Input is sz128 x sz128 x 8f
gd1 = tf.keras.layers.concatenate([gd1, ge7], axis=3, name="gdcat1")
gd2 = tf.keras.layers.LeakyReLU(0.0, name="gdact1")(gd1)
gd2 = tf.keras.layers.Conv2DTranspose(8*f, k, s, padding="same", name="gdconv2")(gd2)
gd2 = tf.keras.layers.BatchNormalization(name="gdbn2")(gd2)
gd2 = tf.keras.layers.Dropout(config.dropout_rate, name="gddrop2")(gd2)
# Input is sz64 x sz64 x 8f
gd2 = tf.keras.layers.concatenate([gd2, ge6], axis=3, name="gdcat2")
gd3 = tf.keras.layers.LeakyReLU(0.0, name="gdact2")(gd2)
gd3 = tf.keras.layers.Conv2DTranspose(8*f, k, s, padding="same", name="gdconv3")(gd3)
gd3 = tf.keras.layers.BatchNormalization(name="gdbn3")(gd3)
gd3 = tf.keras.layers.Dropout(config.dropout_rate, name="gddrop3")(gd3)
# Input is sz32 x sz32 x 8f
gd3 = tf.keras.layers.concatenate([gd3, ge5], axis=3, name="gdcat3")
gd4 = tf.keras.layers.LeakyReLU(0.0, name="gdact3")(gd3)
gd4 = tf.keras.layers.Conv2DTranspose(8*f, k, s, padding="same", name="gdconv4")(gd4)
gd4 = tf.keras.layers.BatchNormalization(name="gdbn4")(gd4)
# Input is sz16 x sz16 x 8f
gd4 = tf.keras.layers.concatenate([gd4, ge4], axis=3, name="gdcat4")
gd5 = tf.keras.layers.LeakyReLU(0.0, name="gdact4")(gd4)
gd5 = tf.keras.layers.Conv2DTranspose(4*f, k, s, padding="same", name="gdconv5")(gd5)
gd5 = tf.keras.layers.BatchNormalization(name="gdbn5")(gd5)
gd5 = tf.keras.layers.concatenate([gd5, ge3], axis=3, name="gdcat5")
# Input is sz8 x sz8 x 4f
gd6 = tf.keras.layers.LeakyReLU(0.0, name="gdact5")(gd5)
gd6 = tf.keras.layers.Conv2DTranspose(2*f, k, s, padding="same", name="gdconv6")(gd6)
gd6 = tf.keras.layers.BatchNormalization(name="gdbn6")(gd6)
# Input is sz4 x sz4 x 2f
gd6 = tf.keras.layers.concatenate([gd6, ge2], axis=3, name="gdcat6")
gd7 = tf.keras.layers.LeakyReLU(0.0, name="gdact6")(gd6)
gd7 = tf.keras.layers.Conv2DTranspose(f, k, s, padding="same", name="gdconv7")(gd7)
gd7 = tf.keras.layers.BatchNormalization(name="gdbn7")(gd7)
# Input is sz2 x sz2 x f
gd7 = tf.keras.layers.concatenate([gd7, ge1], axis=3, name="gdcat7")
gd8 = tf.keras.layers.LeakyReLU(0.0)(gd7)
gd8 = tf.keras.layers.Conv2DTranspose(c, k, s, padding="same", activation="tanh",
                                      name="gdconvout")(gd8)
# Input is sz x sz x nc
```

```
outputs = tf.keras.layers.Cropping2D(pad, name="gcrop")(gd8)

model = tf.keras.models.Model(inputs=inputs, outputs=outputs, name="cond_gen")
```

Codes for image denoising CGAN - generator

```
f = config.base_number_of_filters
k = config.kernel_size
s = config.strides
sz = config.train_size
c = config.channels

inputs = tf.keras.layers.Input((sz, sz, c), name="dinput")

d0 = tf.keras.layers.Conv2D(f, k, s, padding="same", name="dconv0")(inputs)
d0 = tf.keras.layers.LeakyReLU(config.leak, name="dact0")(d0)

d1 = tf.keras.layers.Conv2D(2*f, k, s, padding="same", name="dconv1")(d0)
d1 = tf.keras.layers.BatchNormalization(name="dbn1")(d1)
d1 = tf.keras.layers.LeakyReLU(config.leak, name="dact1")(d1)

d2 = tf.keras.layers.Conv2D(4*f, k, s, padding="same", name="dconv2")(d1)
d2 = tf.keras.layers.BatchNormalization(name="dbn2")(d2)
d2 = tf.keras.layers.LeakyReLU(config.leak, name="dact2")(d2)

d3 = tf.keras.layers.Conv2D(8*f, k, s, padding="same", name="dconv3")(d2)
d3 = tf.keras.layers.BatchNormalization(name="dbn3")(d3)
d3 = tf.keras.layers.LeakyReLU(config.leak, name="dact3")(d3)

d4 = tf.keras.layers.Flatten(name="dflatout")(d3)

outputs = tf.keras.layers.Dense(1, name="ddenseout")(d4)

model = tf.keras.models.Model(inputs=inputs, outputs=outputs, name="cond_dsc")
```

Code for image denoising CGAN - discriminator

```
def importImages(listOfImgDir):
    myFolder = []
    for folder in listOfImgDir:
        myImages = []
        for image in folder:
            # original image
            img = cv2.imread(image)

            # denoised image
            denoised_img = cv2.fastNlMeansDenoising(img,None,10,7,21)

            myImages.append(denoised_img)
        myFolder.append(np.array(myImages))
    return myFolder
```

Code for OpenCV Non-Local Means Image Denoising

```
# load data
rate, data = wav.read("mywav.wav")
data = data / 32768

# add noise
noise_len = 7 # seconds
noise = band_limited_noise(min_freq=5000, max_freq=9000, samples=len(data), samplerate=rate)*10
noise_clip = noise[:rate*noise_len]
audio_clip_band_limited = data+noise

# denoise and write to file
noise_reduced = nr.reduce_noise(audio_clip=audio_clip_band_limited, noise_clip=noise_clip, verbose=True)
wav.write("mywav.wav", rate, noise_reduced)
```

Code for Noisereduce

Appendix C - Codes Used (Java)

```java
public class restClient extends Application {
    private static final String BASE_URL = "http://172.17.41.97:5000";

    private static AsyncHttpClient client = new AsyncHttpClient();

    public static void get(String url, RequestParams params, AsyncHttpResponseHandler responseHandler) {
//        System.out.println("Get Request: " + getAbsoluteUrl(url));
        client.get(getAbsoluteUrl(url), params, responseHandler);
    }

    public static void post(String url, RequestParams params, AsyncHttpResponseHandler responseHandler) {
        client.post(getAbsoluteUrl(url), params, responseHandler);
//        client.post(getAbsoluteUrl(url), )
    }

    public static void getByUrl(String url, RequestParams params, AsyncHttpResponseHandler responseHandler) {
        client.get(url, params, responseHandler);
    }

    public static void postByUrl(String url, RequestParams params, AsyncHttpResponseHandler responseHandler) {
        client.post(url, params, responseHandler);
    }

    private static String getAbsoluteUrl(String relativeUrl) { return BASE_URL + relativeUrl; }
}
```

Code for connecting Android application and Backend Server

```java
    public static String SAVED_FILE;
    public static String SESSION_ID;
    public static String USER;

    public static String getSAVED_FILE() { return SAVED_FILE; }

    public static void setSAVED_FILE(String filename) { SAVED_FILE = filename; }

    public static void setSESSION_ID() { SESSION_ID = randomAlphaNumeric( count: 10); }

    public static String getSESSION_ID() { return SESSION_ID; }

    public static void setUSER(String username) { USER = username; }

    public static String getUSER() { return USER; }

    private static final String ALPHA_NUMERIC_STRING = "ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";
    public static String randomAlphaNumeric(int count) {
        StringBuilder builder = new StringBuilder();
        while (count-- != 0) {
            int character = (int)(Math.random()*ALPHA_NUMERIC_STRING.length());
            builder.append(ALPHA_NUMERIC_STRING.charAt(character));
        }
        return builder.toString();
    }
```

Code for generating SessionID

```
void postCall() {
    RequestParams reqParam = new RequestParams();
    String filePath = Environment.getExternalStorageDirectory().getAbsolutePath();
    filePath += "/DepressionAnalysis";
    filePath = PredictiveIndex.getSAVED_FILE();
    System.out.println("postCall() print: " + filePath);

    File theFile = new File(filePath);
    try {
        reqParam.put( key: "file", theFile);
    } catch (Exception e) {
        e.printStackTrace();
    }
    String url;
    url = "/users/" + PredictiveIndex.getUSER() + "/analysis/" + PredictiveIndex.getSESSION_ID();
```

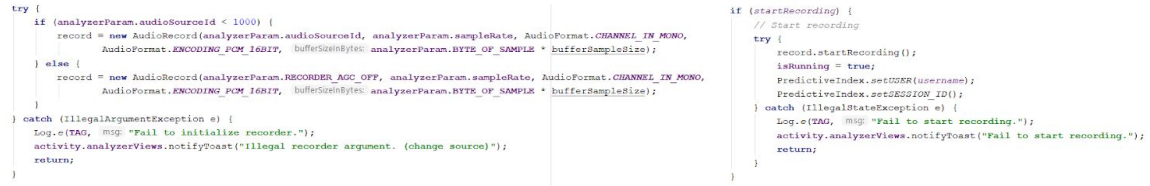
```
restClient.post(url, reqParam, new JsonHttpResponseHandler() {
    @Override
    public void onSuccess(int statusCode, Header[] headers, JSONObject response) {
        // If the response is JSONObject instead of expected JSONArray
        Log.d( tag: "API",  msg: "Post : " + response);
        try {
            JSONObject serverResp = new JSONObject(response.toString());
            myResponses.add(serverResp);
            getResponses();
        } catch (JSONException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
    @Override
    public void onSuccess(int statusCode, Header[] headers, JSONArray timeline) {
        // Pull out the first event on the public timeline
        System.out.println("Status: " + statusCode + "\nHeaders: " + headers + "\nResponse: " + timeline);
        getResponses();
    }
    @Override
    public void onFailure(int statusCode, Header[] headers, String str, Throwable throwable) {
        // Pull out the first event on the public timeline
        System.out.println("Status: " + statusCode + "\nHeaders: " + headers + "\nResponse: " + str);
    }

    @Override
    public boolean getUseSynchronousMode() { return false; }
});
}
```

```
public static double dispPrediction;
public static boolean receivedPrediction = false;

public void getResponses() {
    while (!myResponses.isEmpty()) {
        JSONObject jsonArray = myResponses.poll();
        if (jsonArray != null) {
            System.out.println("getResponses Called");
            try {
                double prediction = jsonArray.getDouble( name: "Prediction");
                System.out.println("Patient shows " + String.format("%.2f", prediction) + "% signs of Depression");
                dispPrediction = prediction;
                receivedPrediction = true;
            } catch (Exception e) {
                e.printStackTrace();
                System.out.println("Error in API Response");
            }
        }
    }
}
```

Code for getting prediction from Backend Server



```
try {
    if (analyzerParam.audioSourceId < 1000) {
        record = new AudioRecord(analyzerParam.audioSourceId, analyzerParam.sampleRate, AudioFormat.CHANNEL_IN_MONO,
                AudioFormat.ENCODING_PCM_16BIT,  bufferSizeInBytes: analyzerParam.BYTE_OF_SAMPLE * bufferSampleSize);
    } else {
        record = new AudioRecord(analyzerParam.RECORDER_AGC_OFF, analyzerParam.sampleRate, AudioFormat.CHANNEL_IN_MONO,
                AudioFormat.ENCODING_PCM_16BIT,  bufferSizeInBytes: analyzerParam.BYTE_OF_SAMPLE * bufferSampleSize);
    }
} catch (IllegalArgumentException e) {
    Log.e(TAG,  msg: "Fail to initialize recorder.");
    activity.analyzerViews.notifyToast("Illegal recorder argument. (change source)");
    return;
}
```

```
if (startRecording) {
    // Start recording
    try {
        record.startRecording();
        isRunning = true;
        PredictiveIndex.setUSER(username);
        PredictiveIndex.setSESSION_ID();
    } catch (IllegalStateException e) {
        Log.e(TAG,  msg: "Fail to start recording.");
        activity.analyzerViews.notifyToast("Fail to start recording.");
        return;
    }
}
```

Code for AudioRecord initialisation

```
graphView.getViewport().setScrollable(true);
graphView.getViewport().setScalable(true);
graphView.addSeries(dataSeries);
graphView.getGridLabelRenderer().setNumHorizontalLabels(3);
graphView.getGridLabelRenderer().setHumanRounding(false);
graphView.setBackgroundColor(Color.rgb( red: 152,  green: 236,  blue: 255));

dataSeries.setDrawDataPoints(true);
dataSeries.setDataPointsRadius(5);

// set manual Y bounds
graphView.getViewport().setYAxisBoundsManual(true);
graphView.getViewport().setMinY(0);
graphView.getViewport().setMaxY(100);
```

Code for GraphView line graph initialisation

```
public class DatabaseHandler extends SQLiteOpenHelper {
    public DatabaseHandler(Context context) { super(context, name: "Database1", factory: null, version: 4); }

    @Override
    public void onCreate(SQLiteDatabase sqLiteDatabase) {
        String CreateTable = "CREATE TABLE Table2 (xValue INTEGER, yvalue INTEGER)";
        sqLiteDatabase.execSQL(CreateTable);
    }

    @Override
    public void onUpgrade(SQLiteDatabase sqLiteDatabase, int oldVersion, int newVersion) {
        sqLiteDatabase.execSQL("DROP TABLE IF EXISTS Table2");
        onCreate(sqLiteDatabase);
    }

    public void insertToData(long ValX, int ValY) {
        SQLiteDatabase sqLiteDatabase = this.getWritableDatabase();
        ContentValues contentValues = new ContentValues();
        contentValues.put("xValue", ValX);
        contentValues.put("yValue", ValY);

        sqLiteDatabase.insert( table: "Table2", nullColumnHack: null, contentValues);
    }


}
```

Code for SQLite DatabaseHandler

```
public void insertData(double pred) {
    long xValue = new Date().getTime();
    int yValue = (int) pred;

    dbHandler.insertToData(xValue, yValue);

    dataSeries.resetData(grabData());
    graphView.getGridLabelRenderer().setLabelFormatter(new DefaultLabelFormatter() {
        @Override
        public String formatLabel(double value, boolean isValueX) {
            if (isValueX) {
                txtViewTime.setText(sdf.format(new Date((long) value)));
                return sdf.format(new Date((long) value));
            } else {
                txtViewTime.setText(super.formatLabel(value, isValueX));
                return super.formatLabel(value, isValueX);
            }
        }
    });
}

public DataPoint[] grabData() {
    String[] column = {"xValue, yValue"};
    @SuppressLint("Recycle") Cursor cursor = sqLiteDatabase.query( table: "Table2", column, selection: null, selectionArgs: null, groupBy: null,
            having: null, orderBy: null);

    DataPoint[] dataPoints = new DataPoint[cursor.getCount()];

    for (int i = 0; i < cursor.getCount(); i++) {
        cursor.moveToNext();
        dataPoints[i] = new DataPoint(cursor.getLong( columnIndex: 0), cursor.getInt( columnIndex: 1));
    }
    return dataPoints;
}
```

Code for inserting new predictions into SQLite Database and displaying it on GraphView

References

[1] F. Scibelli et al., Depression Speaks: Automatic Discrimination between Depressed and Non-depressed Speakers based on Nonverbal Speech Features. 2018.

[2] J. Tan, "Study: S'poreans think mental illness is sign of personal weakness," The New Paper, 8 October 2015. [Online]. Available: http://www.tnp.sg/news/singapore/study-sporeans-think-mental-illness-sign-personal-weakness

[3] Institute of Mental Health, "Healthy Minds, Healthy Communities," National Mental Health Blueprint, pp. 03-31, December 2010.

[4] Institute of Mental Health, "Loving Hearts, Beautiful Minds," December 2014. [Online]. Available: https://www.imh.com.sg/uploadedFiles/Publications/IMH_CorporateProfile.pdf

[5] A. Khor, in Singapore Mental Health Conference 2013, Singapore, 2013.

[6] A. Khor, in National University of Singapore Students Political Association (NUSPA) Social Policies Forum 2017 'Mental Health in Singapore', Singapore, 2017.

[7] L. Reed, "Detecting depression with AI", Science Node, 2018. [Online]. Available: https://sciencenode.org/feature/Detecting%20depression.php.

[8] K. Kiefer, DepressionDetect - A Machine Learning Approach for Audio based Depression Classification. 2017.

[9] A. Haque, M. Guo, A. S. Miner, and L. Fei-Fei, “Measuring Depression Symptom Severity from Spoken Language and 3D Facial Expressions,” arXiv:1811.08592 [cs, eess], Nov. 2018 [Online]. Available: http://arxiv.org/abs/1811.08592. [Accessed: 01-Jun-2019]

[10] “Why use Keras - Keras Documentation.” [Online]. Available: https://keras.io/why-use-keras/.

[11] “Backend - Keras Documentation.” [Online]. Available: https://keras.io/backend/.

[12] “Guide to the Sequential model - Keras Documentation.” [Online]. Available: https://keras.io/getting-started/sequential-model-guide/.

[13] “Model (functional API) - Keras Documentation.” [Online]. Available: https://keras.io/models/model/.

[14] “Convolutional Layers - Keras Documentation.” [Online]. Available: https://keras.io/layers/convolutional/.
[15] “Core Layers - Keras Documentation.” [Online]. Available: https://keras.io/layers/core/.
[16] “Pooling Layers - Keras Documentation.” [Online]. Available: https://keras.io/layers/pooling/.
[17] “Introduction to Optimizers,” Algorithmia Blog, 07-May-2018. [Online]. Available: https://blog.algorithmia.com/introduction-to-optimizers/.
[18] “Optimizers - Keras Documentation.” [Online]. Available: https://keras.io/optimizers/.
[19] “Activations - Keras Documentation.” [Online]. Available: https://keras.io/activations/.
[20] “Callbacks - Keras Documentation.” [Online]. Available: https://keras.io/callbacks/.
[21] “A Beginner’s Guide to Neural Networks and Deep Learning,” Skymind. [Online]. Available: http://skymind.ai/wiki/neural-network.
[22] ujjwalkarn, “An Intuitive Explanation of Convolutional Neural Networks,” the data science blog. 10-Aug-2016 [Online]. Available: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/.
[23] “A Beginner’s Guide to Convolutional Neural Networks (CNNs),” Skymind. [Online]. Available: http://skymind.ai/wiki/convolutional-network.
[24] G. Drakos, “Cross-Validation,” Towards Data Science, 16-Aug-2018. [Online]. Available: https://towardsdatascience.com/cross-validation-70289113a072.
[25] R. Ruizendaal, “Deep Learning #3: More on CNNs & Handling Overfitting,” Towards Data Science, 12-May-2017. [Online]. Available: https://towardsdatascience.com/deep-learning-3-more-on-cnns-handling-overfitting-2bd5d99abe5d.
[26] J. Brownlee, “Evaluate the Performance Of Deep Learning Models in Keras,” Machine Learning Mastery. 25-May-2016 [Online]. Available: https://machinelearningmastery.com/evaluate-performance-deep-learning-models-keras/.
[27] “A Beginner’s Guide to Generative Adversarial Networks (GANs),” Skymind. [Online]. Available: http://skymind.ai/wiki/generative-adversarial-network-gan.

[28] A. Geitgey, “Machine Learning is Fun Part 6: How to do Speech Recognition with Deep Learning,” Medium. 24-Dec-2016 [Online]. Available: https://medium.com/@ageitgey/machine-learning-is-fun-part-6-how-to-do-speech-recognition-with-deep-learning-28293c162f7a.
[29] “Spectrograms.” [Online]. Available: https://ccrma.stanford.edu/~jos/mdft/Spectrograms.html.
[30] “Classic Spectrograms.” [Online]. Available: https://ccrma.stanford.edu/~jos/sasp/Classic_Spectrograms.html#20468.
[31] Xyobalancer.com. (2012). Spectral Vs. Waveform Data. [online] Available at: http://www.xyobalancer.com/xyo-balancer-blog/spectral_vs_waveform_data.
[32] “PCM Terminology and Concepts — alsaaudio documentation 0.8.4 documentation”, Larsimmisch.github.io, 2018. [Online]. Available: https://larsimmisch.github.io/pyalsaaudio/terminology.html.
[33] R. Jang, “12-2 MFCC,” Audio Signal Processing and Recognition. [Online]. Available: http://mirlab.org/jang/books/audiosignalprocessing/speechFeatureMfcc.asp?title=12-2 MFCC.
[34] “Spectrum Analysis Windows.” [Online]. Available: https://ccrma.stanford.edu/~jos/sasp/Spectrum_Analysis_Windows.html#10074.
[35] “Understanding FFTs and Windowing - National Instruments.” [Online]. Available: http://www.ni.com/en-sg/innovations/white-papers/06/understanding-ffts-and-windowing.html.
[36] “Spectrum Analysis Windows.” [Online]. Available: https://ccrma.stanford.edu/~jos/sasp/Spectrum_Analysis_Windows.html.
[37] Generalized Hamming Window Family.” [Online]. Available: https://ccrma.stanford.edu/~jos/sasp/Generalized_Hamming_Window_Family.html.
[38] “Hann or Hanning or Raised Cosine.” [Online]. Available: https://ccrma.stanford.edu/~jos/sasp/Hann_Hanning_Raised_Cosine.html.
[39] “Hamming Window.” [Online]. Available: https://ccrma.stanford.edu/~jos/sasp/Hamming_Window.html.
[40] “Spectrogram of Speech.” [Online]. Available: https://ccrma.stanford.edu/~jos/mdft/Spectrogram_Speech.html.

[41] "BBC Bitesize - GCSE Computer Science - Encoding audio and video - Revision 2", BBC Bitesize. [Online]. Available: https://www.bbc.com/bitesize/guides/z7vc7ty/revision/2
[42] "What is Pulse Code Modulation (PCM)? - Definition from Techopedia", Techopedia.com. [Online].
Available: https://www.techopedia.com/definition/24128/pulse-code-modulation-pcm
[43] "What is pulse code modulation (PCM)? - Definition from WhatIs.com", SearchNetworking. [Online]. Available:
https://searchnetworking.techtarget.com/definition/pulse-code-modulation-PCM
[44] G. Gokul, Y. Yan, K. Dantu, S. Ko and L. Ziarek, "Real Time Sound Processing on Android", in The 14th International Workshop, University at Buffalo, The State University of New York, 2016, p. 2.
[45] Android Developers. (2018). AudioRecord | Android Developers. [online] Available at: https://developer.android.com/reference/android/media/AudioRecord
[46] P. Bevelacqua, "Fourier Transform", Thefouriertransform.com. [Online]. Available: http://www.thefouriertransform.com/#introduction
[47] J. Gehring, Android Graph Library for creating zoomable and scrollable line and bar graphs.: jjoe64/GraphView. 2019 [Online]. Available: https://github.com/jjoe64/GraphView.
[48] "Image noise", En.wikipedia.org. [Online]. Available:
https://en.wikipedia.org/wiki/Image_noise.
[49] Arora, "Gaussian noise", Slideshare.net. [Online]. Available:
https://www.slideshare.net/AnchalArora11/gaussian-noise.
[50] "Gaussian noise", En.wikipedia.org. [Online]. Available:
https://en.wikipedia.org/wiki/Gaussian_noise.
[51] "Salt-and-pepper noise", En.wikipedia.org. [Online]. Available:
https://en.wikipedia.org/wiki/Salt-and-pepper_noise.
[52] "What is audio noise? - Definition from WhatIs.com", WhatIs.com. [Online]. Available:
https://whatis.techtarget.com/definition/audio-noise.
[53] "Noise reduction", En.wikipedia.org. [Online]. Available:
https://en.wikipedia.org/wiki/Noise_reduction.

[54] "Image Denoising | Vision and Image Processing Lab", Vision and Image Processing Lab. [Online]. Available: https://uwaterloo.ca/vision-image-processing-lab/research-demos/image-denoising.
[55] N. Shibuya, “How to Reduce Image Noises by Autoencoder,” Towards Data Science, 01-Nov-2017. [Online]. Available: https://towardsdatascience.com/how-to-reduce-image-noises-by-autoencoder-65d5e6de543.
[56] J. Brownlee, "How to Develop a Conditional GAN (cGAN) From Scratch", Machine Learning Mastery, 2019. [Online]. Available: https://machinelearningmastery.com/how-to-develop-a-conditional-generative-adversarial-network-from-scratch/.
[57] M. Mirza and S. Osindero, “Conditional Generative Adversarial Nets,” arXiv:1411.1784 [cs, stat], Nov. 2014 [Online]. Available: http://arxiv.org/abs/1411.1784
[58]
B. Cottier, Tensorflow/Keras implementation of a Conditional Generative Adversarial Network (CGAN) model that can be used for image denoising or artefact removal.: bencottier/cgan-denoiser. 2018 [Online]. Available: https://github.com/bencottier/cgan-denoiser.
[59] “OpenCV: Image Denoising.” [Online]. Available: https://docs.opencv.org/3.4/d5/d69/tutorial_py_non_local_means.html.
[60] T. Sainburg, “Noise reduction using spectral gating in python,” Tim Sainburg, 07-Jul-2018. [Online]. Available: ./noise-reduction-python.html.
[61] T. Sainburg, Noise reduction / speech enhancement for python using spectral gating: timsainb/noisereduce. 2019 [Online]. Available: https://github.com/timsainb/noisereduce.
[62] “What is SQLite? Top SQLite Features You Should Know,” SQLite Tutorial. [Online]. Available: http://www.sqlitetutorial.net/what-is-sqlite/.
[63] “Google’s Material Design - Android Design Language,” The Interaction Design Foundation. [Online]. Available: https://www.interaction-design.org/literature/article/google-s-material-design-android-design-language.

[64] “Google’s Material Design - Android Design Language,” The Interaction Design Foundation. [Online]. Available: https://www.interaction-design.org/literature/article/google-s-material-design-android-design-language.
[65] “Audiosegment documentation,” https://media.readthedocs.org/pdf/audiosegment/stable/audiosegment.pdf , 2019, [Online; 5-February-2019].
[66] J. Gratch, R. Artstein, G. Lucas, G. Stratou, S. Scherer, A. Nazarian, R.Wood, J. Boberg, D. DeVault, S. Marsella, and D. Traum, “The distress analysis interview corpus of human and computer interviews,” InLREC, pp. 3123–3128, 2014.
[67] “acoustics - What is the meaning of ‘frequency of a human voice’?,” Physics Stack Exchange. [Online]. Available: https://physics.stackexchange.com/questions/76463/what-is-the-meaning-of-frequency-of-a-human-voice.
[68] “frequency - What is the bandwidth of human speech?,” Signal Processing Stack Exchange. [Online]. Available: https://dsp.stackexchange.com/questions/36505/what-is-the-bandwidth-of-human-speech.
[69] ADVISORY GUIDELINES ON THE PERSONAL DATA PROTECTION ACT FOR NRIC AND OTHER NATIONAL IDENTIFICATION NUMBERS. Singapore: Personal Data Protection Commission Singapore, 2018 [Online]. Available: https://www.pdpc.gov.sg/-/media/Files/PDPC/PDF-Files/Advisory-Guidelines/Advisory-Guidelines-for-NRIC-Numbers---310818.pdf.
[70] “AccountManager,” Android Developers. [Online]. Available: https://developer.android.com/reference/android/accounts/AccountManager.
[71] B. Journal, “Year in Pixels,” Bullet Journal. [Online]. Available: https://bulletjournal.com/blogs/bulletjournalist/deep-dive-year-in-pixels.
[72] “Animations Overview,” Android Developers. [Online]. Available: https://developer.android.com/training/animation/overview.

[73] “waveform noun - Definition, pictures, pronunciation and usage notes | Oxford Advanced Learner’s Dictionary at OxfordLearnersDictionaries.com.” [Online]. Available: https://www.oxfordlearnersdictionaries.com/definition/english/waveform.
[74] “Terminology: spectrum, spectrogram, spectrograph, sonogram, etc,” Signal Processing Stack Exchange. [Online]. Available: https://dsp.stackexchange.com/questions/2426/terminology-spectrum-spectrogram-spectrograph-sonogram-etc.
[75] A. Ronacher, “Flask,” http://flask.pocoo.org