NGEE ANN POLYTECHNIC

University of Glasgow

Diploma in Engineering Science

Product Design & Development (ESFYP)

Interim Report (30%)

Project ID: PS06

Project Title: Depression Detection Using Speech Analysis

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

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 technologies and audio processing technology 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

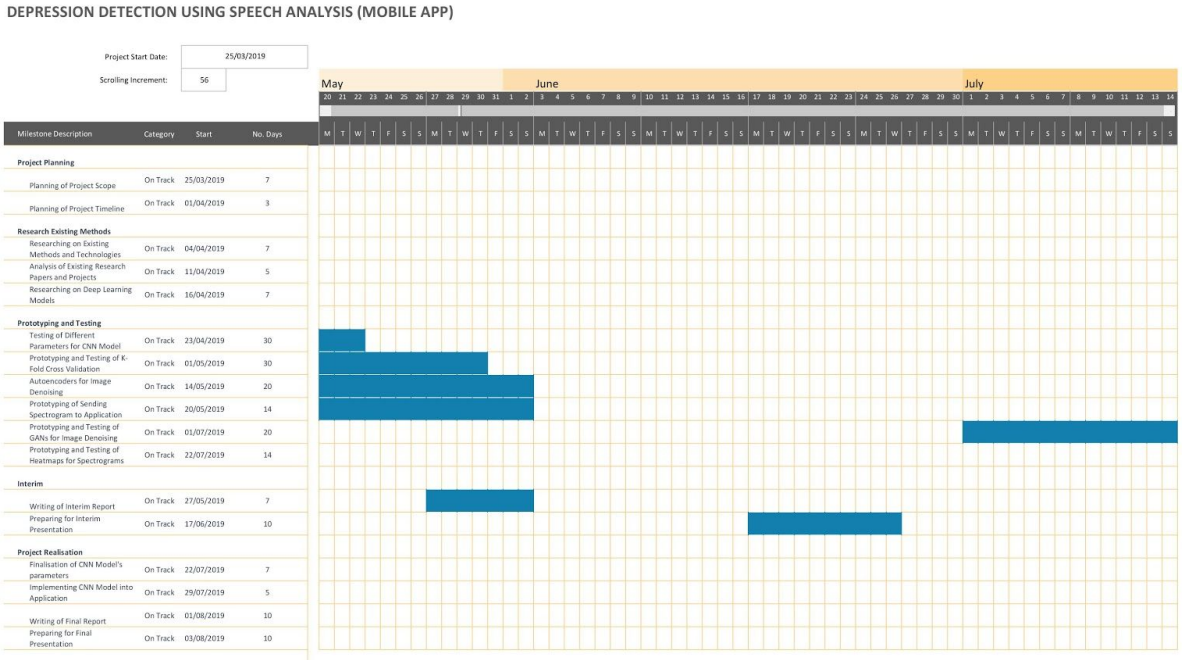

Figure 1: Gantt Chart (1/3)

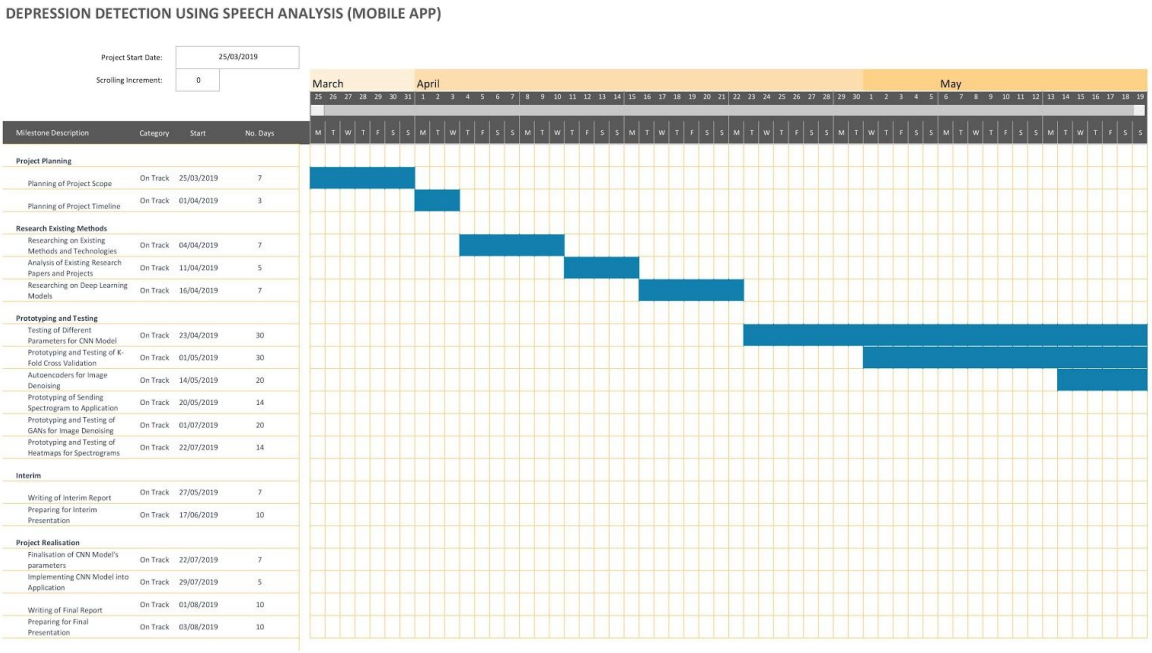

Figure 2: Gantt Chart (2/3)

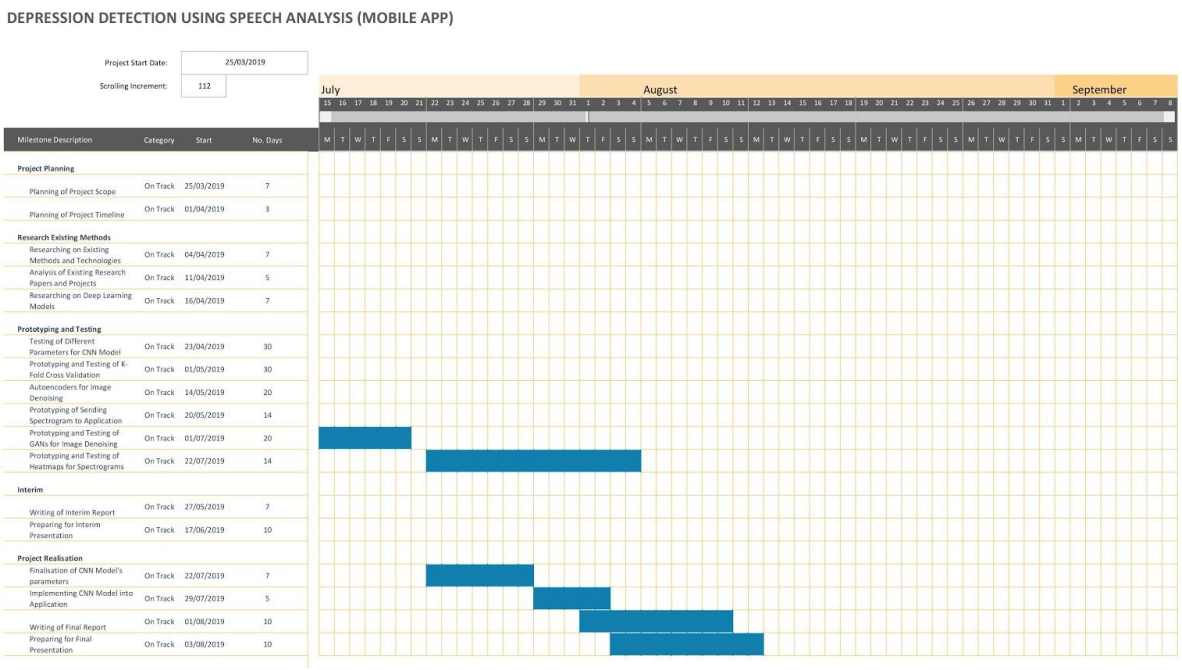

Figure 3: Gantt Chart (3/3)

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].

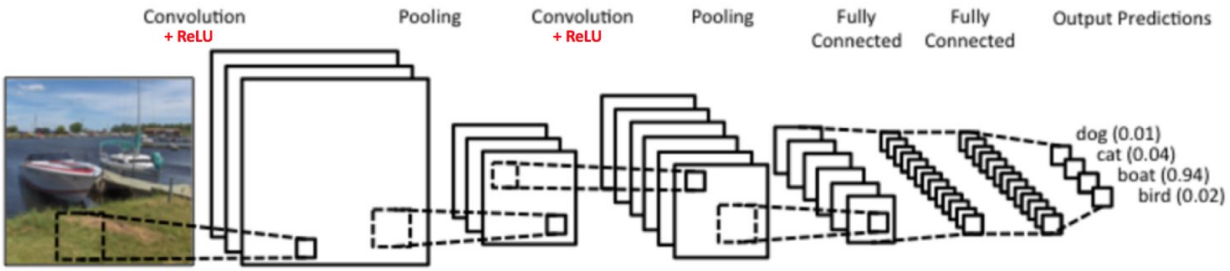

Figure 4: LeNet Architecture [22]

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]).

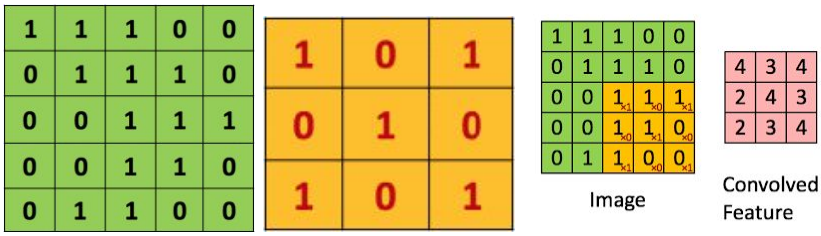

Figure 5: 5x5 image, 3x3 matrix and Convolved Feature [22]

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].

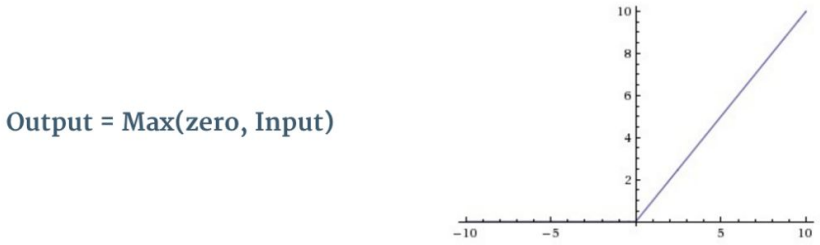

Figure 6: Graph of a ReLU operation [22]

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].

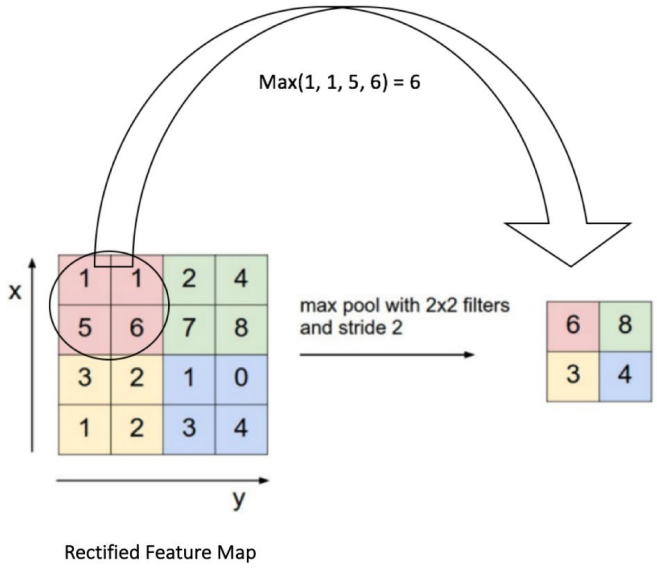

Figure 7: A Max Pooling operation on a rectified feature map [22]

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].

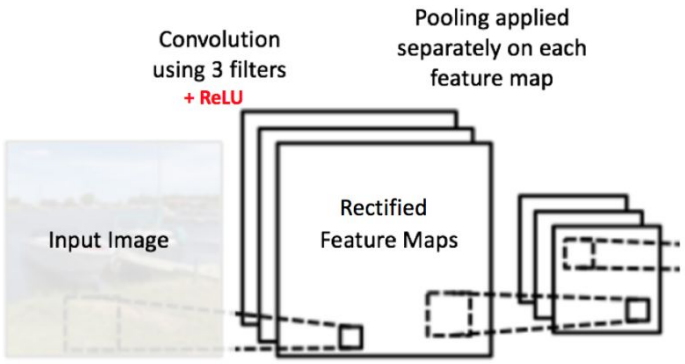

Figure 8: Pooling operation [22]

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]

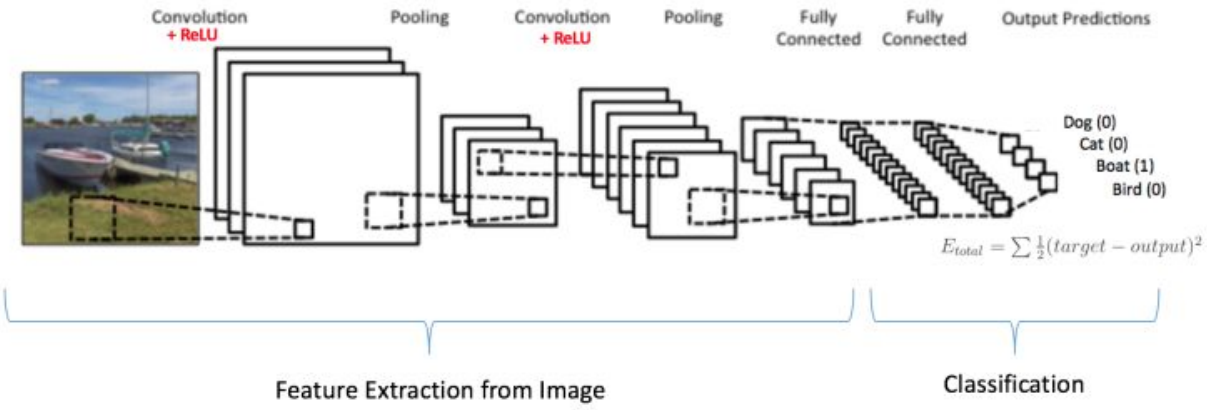

Figure 9: Summary and output of entire network [22]

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 = ∑ 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.

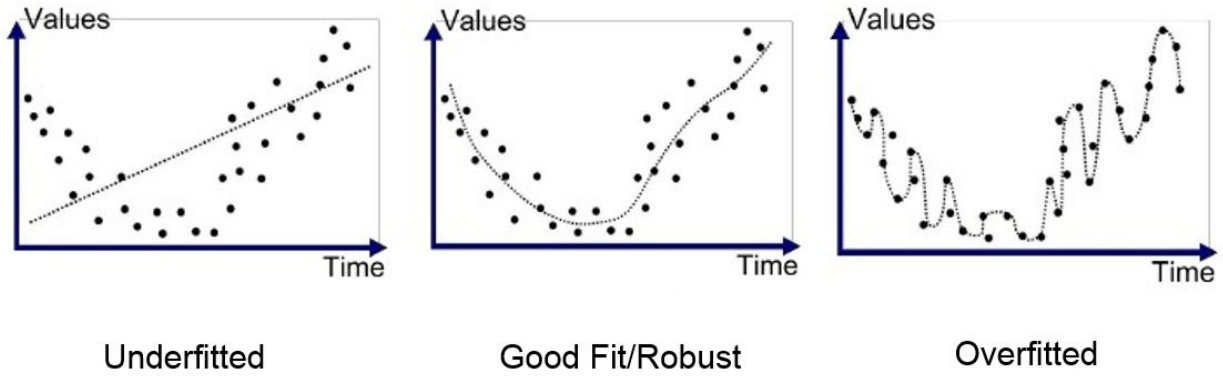

Figure 11: A simplified representation of overfitting

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].

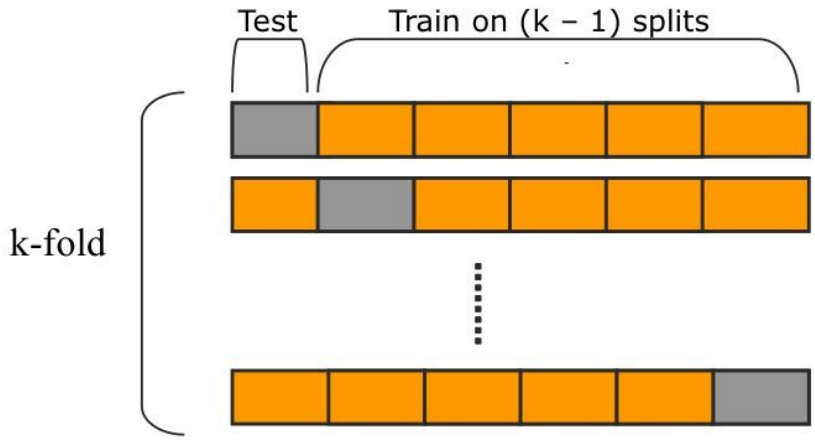

Figure 12: Visualisation of K-Fold Cross validation

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].

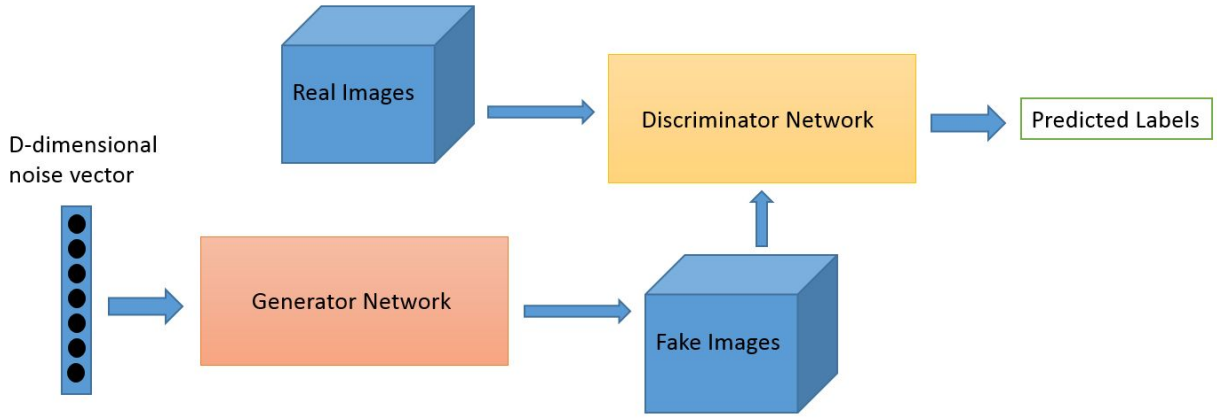

Figure 13: Visualisation of GAN networks [27]

3.6 Voice Analysis

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].

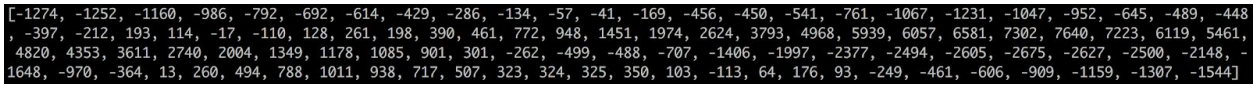

Figure 15: Sampled sound wave [28]

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.

[110.97481594791122, 166.61537247955155, 180.43561044211469, 175.09309469913353, 180.01686910959916, 176.00619977472167, 179.79737781786582, 173.53025213548219, 176.87177119846058, 170.42684732853121
, 159.26023828556598, 163.24469810981628, 149.15527353931867, 154.34196586290136, 151.46179061113972, 152.99674239973979, 143.98878156117371, 156.6033737693738, 155.78237530428544, 157.1793094101783
8, 146.2863229750967 9, 164.37233032929228, 158.12826564460 88, 147.23266451005145, 133.26597973863801, 116.51701000288 31, 116.85501120577126, 115.40519005123537, 120.85619013711488, 112.48406123161 09
1, 111.80244759457571, 92.590676871856431, 105.75863927434719, 95.673146446282971, 90.391748128064208, 79.3558180553 14899, 86.080143147713926, 84.748200268709567, 83.050569583779065, 86.207180262242
758, 90.252031938154076, 89.361567351948437, 90.917307309643206, 90.746777849123049, 86.726552726337033, 85.709412745066928, 95.938840816664865, 99.09254575917069, 96.632437741434885, 103.2396123166
6669, 105.80328302591124, 109.530292812347 07, 116.46408227060996, 129.20890691592615, 130.43460361780441, 138.15581799444712, 128.25056761852832, 138.14492240466387, 140.0352714810314, 128.151381394
29752, 123.93018478493934, 121.19289035588113, 119.03159255422509, 114.23027889344033, 119.17173421549 97, 101.02560719093093, 110.91192243698025, 106.04872005953503, 100.86977927980999, 92.123301579
000341, 94.376766266598295, 97.850709698634489, 113.37126364077845, 110.24526597732718, 113.72249347908021, 120.63960942628063, 122.06482553759932, 117.96716716036715, 120.87682744817975, 125.060973
81947157, 111.57319012901624, 115.54483708595507, 116.99850750130265, 114.40659619324526, 79.8695439808839 75, 104.83111191845597, 104.66218602004588, 104.91691734582642, 97.143620527536072, 78.43459
781117835, 82.214144782667248, 67.240072805959614, 66.578937262360313, 74.100307226086798, 64.861423011415653, 59.167561212002269, 62.479712687304911, 63.568362396107467, 55.906096471453267, 42.7908
02909362839, 55.693923524361097, 50.776364877715011, 41.196111220671298, 51.062413666348945, 58.493563858289065, 53.081835042922769, 73.060663128159547, 68.216252022361, 66.7701034934517, 59.76625
124915202, 35.413635503802389, 22.705615809958832, 16.458048045346381, 44.910670465379937, 59.282513769840705, 69.241393677323856, 81.778634874076346, 88.409923803546008, 94.688033733251245, 96.6408
67526244051, 91.806226496828543, 94.570526932206619, 99.250924315589074, 97.899164767741183, 75.176507616277235, 80.947474423758905, 71.859103451990862, 93.863684037461738, 96.757145639348298, 96.52
8614354976241, 99.366456533638413, 102.187176081769 04, 102.065966630232 35, 101.784931399110 82, 103.7883358299547, 99.915220403870748, 107.434784709299 35, 104.464495526206 18, 105.707898681952 98, 101.
105965413387 49, 100.757378315261 95, 91.742897073196886, 88.307278943069093, 90.936627732905492, 71.134275744339803, 72.504304977841457, 76.233185506299705, 63.281284410272761, 45.380164336858961, 43
.018963766250437, 49.133789791276826, 53.507751009532953, 48.586423555688746, -4.473077611302 8883, 50.833000650183408, 51.003802143009629, 39.577356593427531, 47.096919248906332, 55.442197175664383,
56.967128095484341, 49.383247263177985]

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

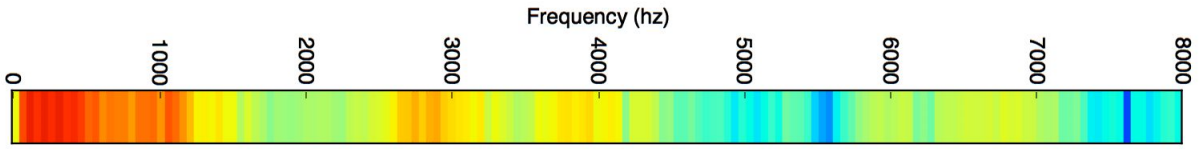

Figure 17: Data in Figure 16 converted into a chart [28]

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].

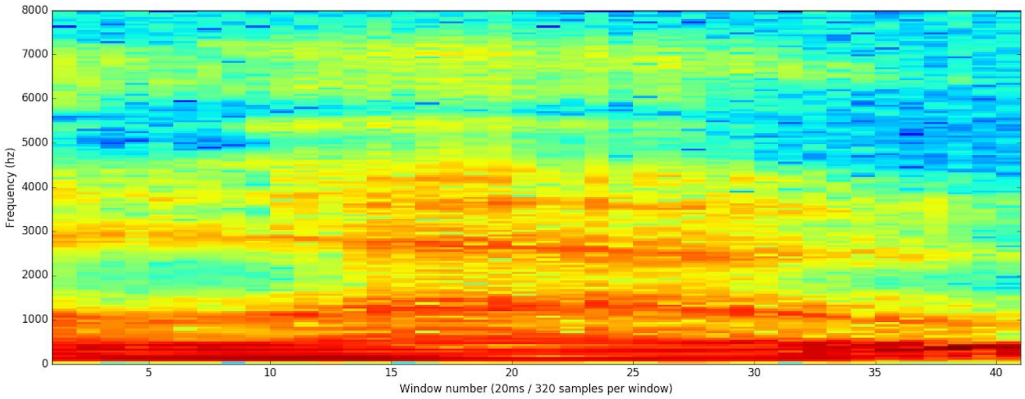

Figure 18: Example of a spectrogram [28]

As mentioned in previous sections, we worked on an Android application which was able to detect depression 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, since a neural network can find patterns in this kind of data more easily than raw sound waves, this type of data representation can be sent into neural networks [28].

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 [31].

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.

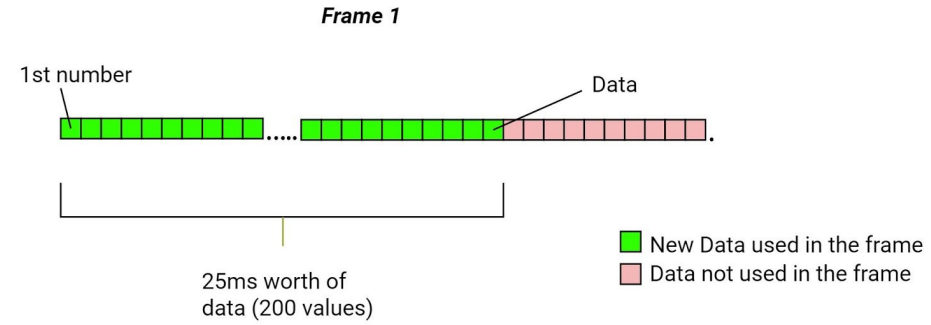

Figure 19: Frame 1 of data

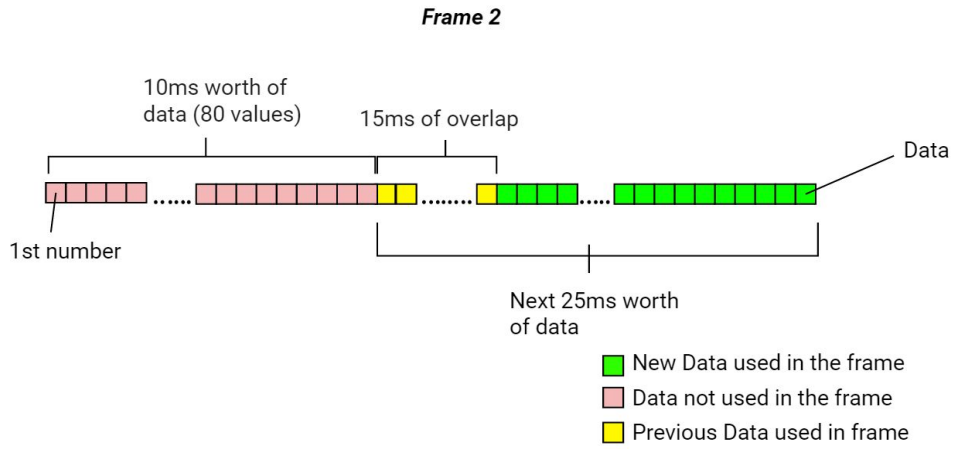

Figure 20: Frame 2 of data

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 [32].

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) [33]. 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 [33]. 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 [33]:

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

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 [35]. 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 [36]. Two well known members of the generalized Hamming family are the Hann and Hamming windows [36]. The Hamming and Hann window functions both have a sinusoidal shape [34]. Both windows result in a wide peak but low side lobes [34]. However, the Hann window touches zero at both ends eliminating all discontinuity [34]. The Hamming window doesn't quite reach zero and thus still has a slight discontinuity in the signal [34]. 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 [34].

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, $0.25 \cdot W_R(\omega \pm \Omega_M)$ [36]. 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 [36].

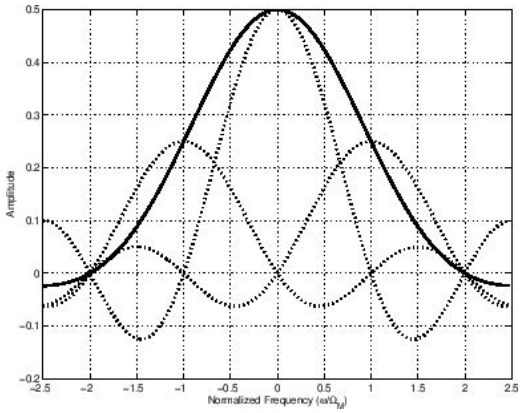

Figure 21: Construction of generalized Hamming window transform as a superposition of three shifted aliased sinc functions [36]

Hanning windows can be seen as one period of a cosine "raised" so that its negative peaks just touch zero [37]. Similarly, hamming windows can also be seen one period of a raised cosine

[38]. 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) [38].

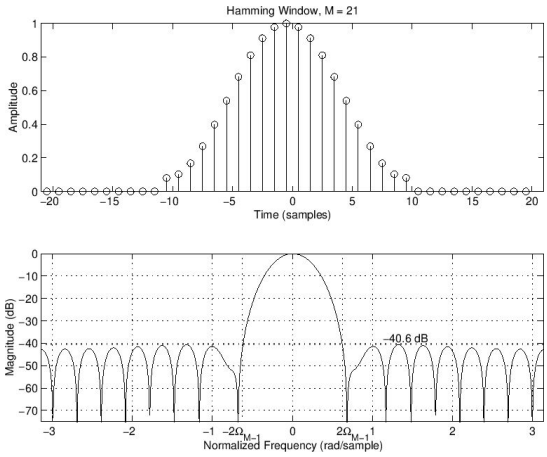

Figure 22: Hamming window and DTFT [38]

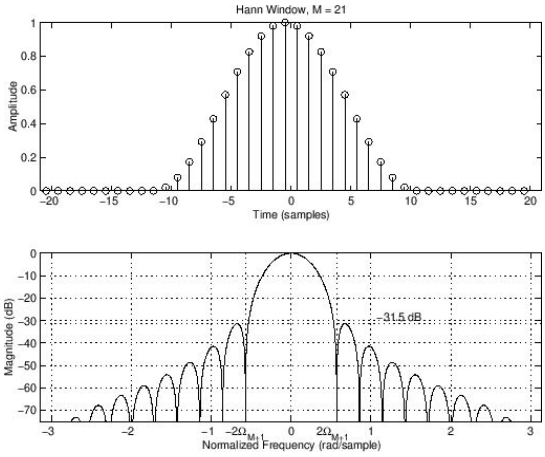

Figure 23: Hanning window and DTFT [37]

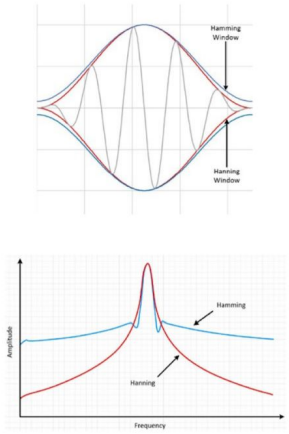

Figure 24: Hamming and Hann windowing result in a wide peak but nice low side lobes [34]

An example of how these windows can be used in speech analysis would be making Hamming windows be around 20 milliseconds [39]. 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 [39].

3.7 Image Denoising

3.7.1 Autoencoders

An autoencoder is made out of two parts: an encoder and a decoder [40]. 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 [40].

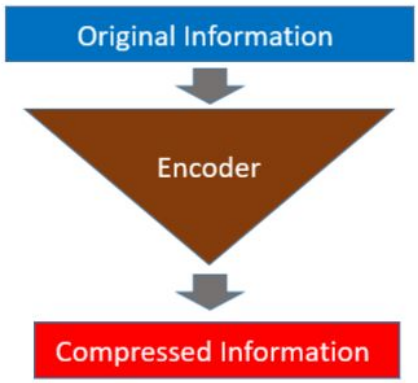

Figure 25: Encoder [40]

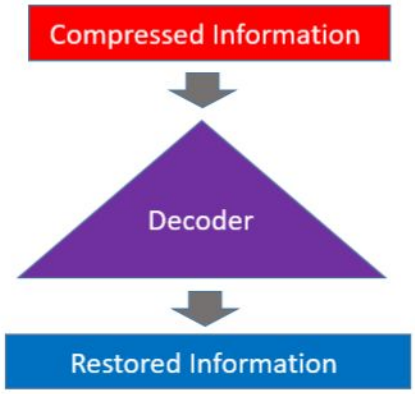

Figure 26: Decoder [40]

The autoencoder is a neural network that learns to encode and decode automatically [40]. Once learning is done, the encoder and decoder can be used independently [40].

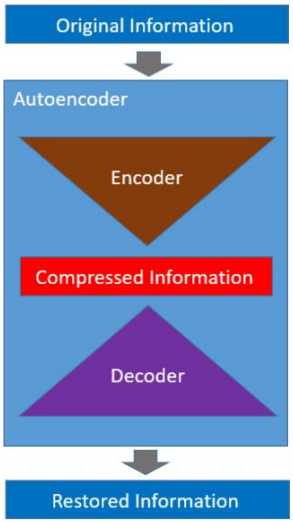

Figure 27: Autoencoder with encoder and decoder [40]

Autoencoders are data specific and do not work on completely unseen data structure [40]. For example, an autoencoder trained on numbers does not work on alphabets [40]. Another limitation is that the compression by an autoencoder is lossy [40]. As such, it does not perfectly restore the original information. Autoencoders can be useful for many different things, such as image noise reduction. For this, the autoencoder can be trained with the noisy data as input and have the original data as expected output [40]. During the training, the autoencoder learns to extract important features from input images and ignores the image noises because the labels have no noises [40]. The network can also be made deeper by adding more layers, improving its performance. For example, CNNs can be used for working on images to improve the quality of compression and decompression.

4. Experimental Results and Analysis

4.1 List of Experiments and Results

4.1.1 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 28: Model architecture

The validation accuracy 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 below. The web server was set up using an EC2 instance and Flask was used to connect the web server and the Android application.

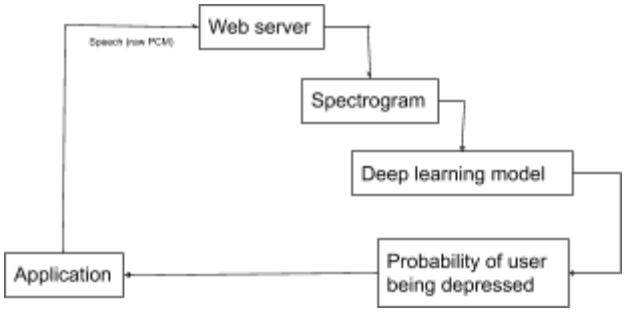

Figure 29: Visualization of flow of events

4.1.2 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

Figure 30: 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 around 15.2% (Experiment #7). 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 being overfitted, and if so, from which point onwards it was overfitted. An example of these plots can be seen below.

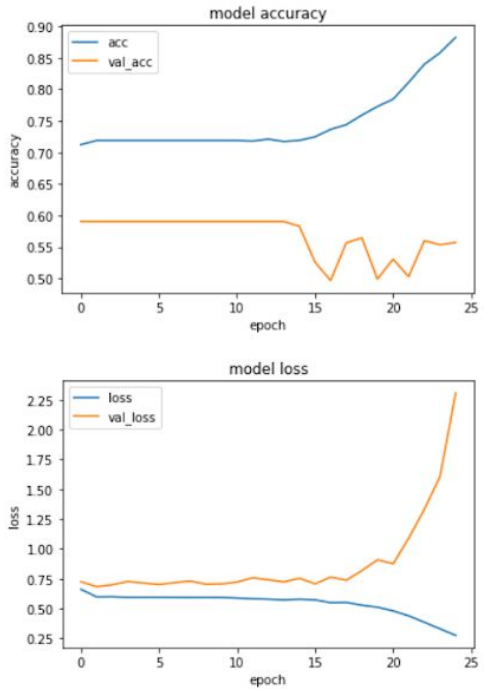

Figure 31: Before k-fold

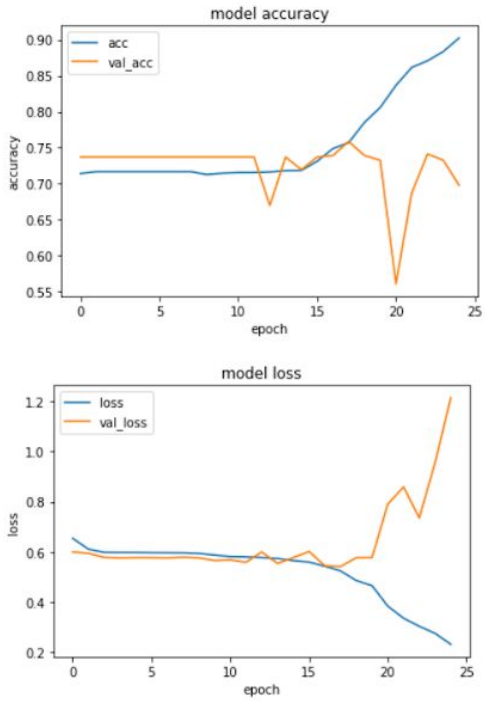

Figure 32: After k-fold (k = 4)

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.

4.1.3 Autoencoders

We used autoencoders as a method for image denoising. We trained an autoencoder with the noisy data as input and used the original data as expected output. During the training, the autoencoder learnt how to extract important features from input images and ignored the image noises.

We conducted two experiments for the autoencoders, one with a batch size of 50 and another with a batch size of 32. A summary of these experiments can be found below and the code for this can be found in Appendix B.

Experiment Name	Batch Size	No. of Epochs			Amazon / PC?	Duration	Completed?	Accuracy
#1	50	10			Amazon	around 11h	☑	0.0074
#2	32	10			Amazon	about 13h	☑	0.005722269231

Figure 33: Summary of Experiments for Autoencoders

The validation accuracy given by this autoencoder was 1.62%, which is scarily low. One possible reason for this 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 34: First value in training and test data before adding noise

Figure 35: First value in training and test data after adding noise

4.1.4 GANs

In our project, we plan 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.

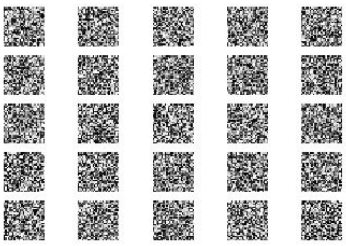

Figure 36: Output from first epoch

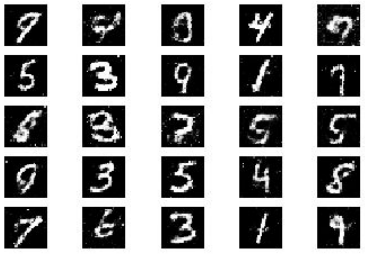

Figure 37: Output from last epoch

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.

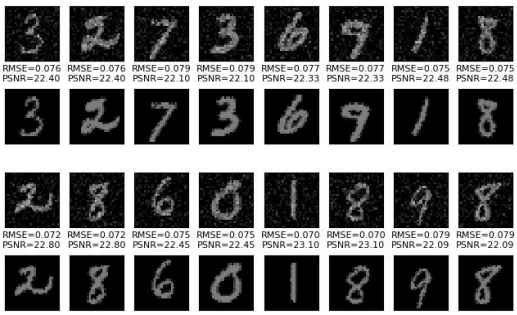

Figure 38: Output from epoch 0

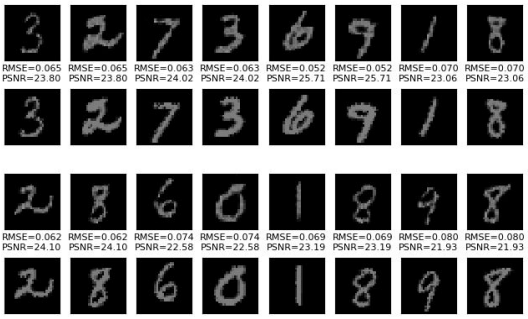

Figure 39: Output from epoch 20

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 will continue working on the code for the spectrograms. More details can be found in the next few sections.

4.2 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, since the validation accuracy is less than 2%, we will continue to work on the code so as to improve the accuracy. For GANs, we hope to be able to implement the code on spectrograms rather than the MNIST database of handwritten digits. Currently, the code seems to be working well on this database.

Our future plans will be explored in more detail in Section 6.

5. 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.

Following that, 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.

Apart from our personal computer, 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.

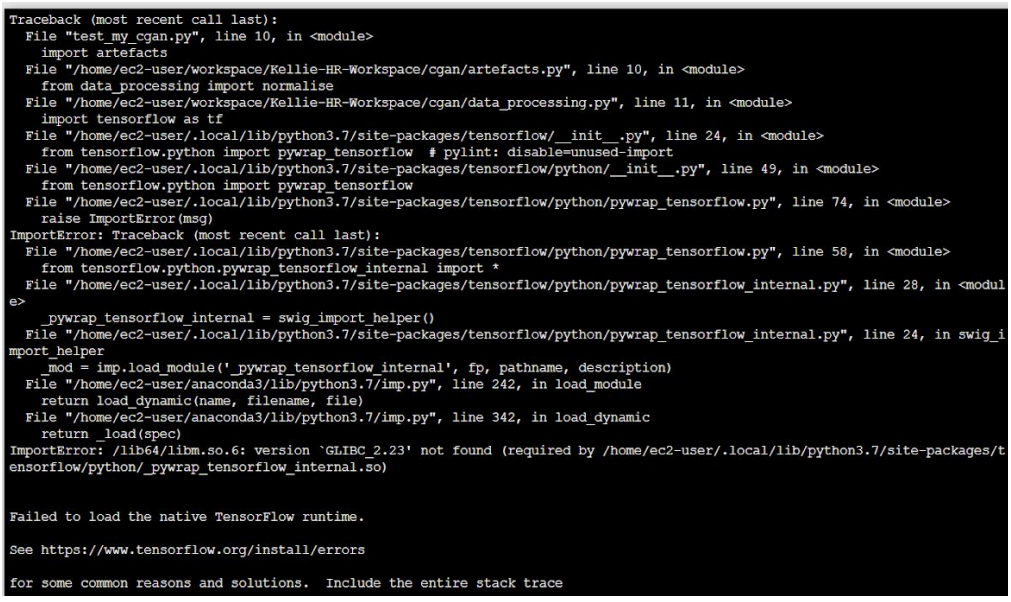

Figure 40: Error loading Tensorflow runtime

Due to time constraints, we were also not able to experiment much of the originally planned code for autoencoders as each run took more than 11 hours.

We tried to send the spectrogram from the web server to the application. However, we were met with a memory error. There was a lack of memory/RAM on the device used for testing.

6. 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.

At this point of our project, we have managed to achieve an average validation accuracy of 75%. 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 also learnt more about developing an Android application that is 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.

7. Future Plans

In the future, we aim to develop the application further, and improve the accuracy of our model. This section will be split into two parts, where we will explore both areas of future plans.

For the application, we plan to improve it by allowing the application to display the spectrogram in real time without having to send it through the server. We also 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).

For improving the accuracy of our model, we plan to implement denoising techniques such as GANs and autoencoders to provide a cleaner spectrogram to train our model. Currently, we have experimented on autoencoders, but the accuracy resulting from those experiments were around 1.62%. For GANs, we intend to test out different methods and implementations of GANs, such as StarGAN and wGAN. We also hope to be able to implement CGAN onto our dataset of spectrograms. Next, we plan to implement ensemble learning, which combines the predictions from multiple models, improving the accuracy. Following that, we also plan to test out different duration of audio signals used in the training and testing data. Currently, we are using 10 seconds of audio data but plan to experiment with 5 and 15 seconds of audio data. 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

```
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)
    pool1 = MaxPooling2D(pool_size=(2,2))(conv1)
    conv2 = Conv2D(64, (3, 3), activation='relu', padding='same')(pool1)
    pool2 = MaxPooling2D(pool_size=(2,2))(conv2)
    conv3 = Conv2D(128, (3, 3), activation='relu', padding='same')(pool2)

    #decoder
    conv4 = Conv2D(128, (3, 3), activation='relu', padding='same')(conv3)
    up1 = UpSampling2D((2,2))(conv4)
    conv5 = Conv2D(64, (3, 3), activation='relu', padding='same')(up1)
    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

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] “PCM Terminology and Concepts — alsaaudio documentation 0.8.4 documentation”, Larsimmisch.github.io, 2018. [Online]. Available: https://larsimmisch.github.io/pyalsaaudio/terminology.html.
[32] R. Jang, “12-2 MFCC,” Audio Signal Processing and Recognition. [Online]. Available: http://mirlab.org/jang/books/audiosignalprocessing/speechFeatureMfcc.asp?title=12-2 MFCC.
[33] “Spectrum Analysis Windows.” [Online]. Available: https://ccrma.stanford.edu/~jos/sasp/Spectrum_Analysis_Windows.html#10074.
[34] “Understanding FFTs and Windowing - National Instruments.” [Online]. Available: http://www.ni.com/en-sg/innovations/white-papers/06/understanding-ffts-and-windowing.html.
[35] “Spectrum Analysis Windows.” [Online]. Available: https://ccrma.stanford.edu/~jos/sasp/Spectrum_Analysis_Windows.html.
[36] Generalized Hamming Window Family.” [Online]. Available: https://ccrma.stanford.edu/~jos/sasp/Generalized_Hamming_Window_Family.html.
[37] “Hann or Hanning or Raised Cosine.” [Online]. Available: https://ccrma.stanford.edu/~jos/sasp/Hann_Hanning_Raised_Cosine.html.
[38] “Hamming Window.” [Online]. Available: https://ccrma.stanford.edu/~jos/sasp/Hamming_Window.html.
[39] “Spectrogram of Speech.” [Online]. Available: https://ccrma.stanford.edu/~jos/mdft/Spectrogram_Speech.html.

[40] 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.