



**QUEEN'S
UNIVERSITY
BELFAST**

DETECTING IMPAIRED SPEECH TO HELP IDENTIFY STROKE

A dissertation submitted in partial fulfilment of the requirements for
the degree of BACHELOR OF SCIENCE in Computer Science

In

The Queen's University of Belfast

By

Pauric Donnelly

Tuesday, May 3rd, 2022

**SCHOOL OF ELECTRONICS, ELECTRICAL ENGINEERING and
COMPUTER SCIENCE**

CSC3002 – COMPUTER SCIENCE PROJECT

Dissertation Cover Sheet

Student Name: Pauric Donnelly

Student Number: 40227531

Project Title: RG03: Detecting Impaired Speech To Help Identify Stroke

Supervisor: Dr Richard Gault

Declaration of Academic Integrity

Before submitting your dissertation, please check that the submission:

1. Has a full bibliography attached laid out according to the guidelines specified in the Student Project Handbook
2. Contains full acknowledgement of all secondary sources used (paper-based and electronic)
3. Does not exceed the specified page limit
4. Is clearly presented and proof-read
5. Is submitted on, or before, the specified or agreed due date. Late submissions will only be accepted in exceptional circumstances or where a deferment has been granted in advance.

By submitting your dissertation, you declare that you have completed the tutorial on plagiarism at <http://www.qub.ac.uk/cite2write/introduction5.html> and are aware that it is an academic offence to plagiarise. You declare that the submission is your own original work. No part of it has been submitted for any other assignment and you have acknowledged all written and electronic sources used.

6. If selected as an exemplar, I agree to allow my dissertation to be used as a sample for future students. (Please delete this if you do not agree.)

Student's signature: Pauric Donnelly

Date of Submission: May 3rd 2022

Acknowledgements

I would like to thank Dr. Richard Gault for his help throughout this entire project. Since the allocation of projects, we met almost every week, and Dr. Gault was very helpful with every question I had for him and always guided me in the right direction whenever I had trouble with a concept. His help throughout the year does not go unnoticed and I thank him sincerely for it.

Abstract

Stroke is the second largest cause of death and disability-adjusted life-years in the world. Minimising the time to treatment for patients is extremely important. Impaired speech is a core symptom that medical professionals consider when identifying cases of stroke. Unfortunately, 1/3 of patients have a significant delay in treatment, partly due to a failure of patients, first-responders, and paramedics to recognize signs of stroke. A technology that can accurately detect speech impairment could support the diagnostic capabilities of non-experts and reduce delays in stroke treatment. This project will aim to develop computational models capable of classifying the presence (or absence) of impairment in audio files.

Contents

1. Introduction and Problem Area	5
2. Solution Description and System Requirements	7
2.1 Solution Description.....	7
2.2 Speech Classification	8
2.3 Impaired Speech.....	9
2.4 System Requirements	10
2.4.1 Functional Requirements	10
2.4.2 Non-Functional Requirements	11
3. Design	12
3.1 Choices in Model Architecture	12
3.1.1 Convolutional Neural Network.....	14
3.1.2 Long Short-Term Memory Network	18
3.2 Data Preparation and Pre-processing	20
3.3 Mobile Application Architecture	20
3.4 Potential Event and Error Handling Measures	22
4. Implementation	23
4.1 Implementation Decisions: Languages and Environments	23
4.1.1 Machine Learning Scripts.....	23
4.1.2 Front-End Application	23
4.1.3 Back-End Server.....	24
4.2 Software Libraries	25
4.3 Custom Dataset Pre-processing: Saarbruecken Database	27
5 Testing.....	30
5.1 Model Experimentation.....	30
5.1.1 Outline of Experimental Dataset	30
5.1.2 Experimental Methods – LSTM & CNN.....	32

5.1.3 Results.....	32
5.1.4 Investigating the influence of Tone on Classification	37
5.2 Functionality Testing: Saarbruecken Custom Dataset	40
5.3 Analysis: Reporting on Success of stroke detection	41
5.4 Mobile Application Testing	45
5.4.1 Front-End Testing.....	45
5.4.2 Backend Testing	45
6 System Evaluation and Experimental Results	45
6.1 Can Machine Learning Classify Impaired Speech?	45
6.2 Implications for Detecting Stroke through Speech Classification?	46
6.3 Success of Project and Further Work.....	46
References.....	49

1. Introduction and Problem Area

Stroke is the second largest cause of death and disability-adjusted life-years in the world. Impaired speech is a core symptom that medical professionals consider when identifying cases of stroke. Unfortunately, 1/3 of patients have a significant delay in treatment in part due to a failure of patients, first-responders and paramedics to recognise signs of stroke. A technology that can accurately detect speech impairment could support the diagnostic capabilities of non-experts and reduce delays in stroke treatment, since delays in seeking treatment for stroke is the major factor limiting the delivery of treatment [1]. This project will aim to develop computational models capable of classifying the presence (or absence) of impairment in audio files. This project will also include a collaboration with a Software Engineering project with a similar product to that which will be created, and these models will be incorporated with theirs.

In an exploratory study undertaken in [2], involving 26 women and 12 men, only 60.5% of patients were able to identify one risk factor of stroke and 55.3% of patients were able to accurately identify at least one symptom of stroke. Similarly in this study in [3], through a total of 553 patients who presented stroke symptoms or signs of stroke, only 32% of patients arrived to an emergency department within 1.5 hours of stroke onset, 46% of patients arriving within three hours and 61% within six hours. Furthermore, a study of 61,019 adults participating in a telephone survey found that only 17.2% of respondents overall could correctly classify all stroke symptoms and indicated to calling emergency services if they thought someone was having a stroke [4]. While factors such as sex and race influenced the delay of treatment, a large factor is the inability of medical professionals and other people to identify that a patient was suffering from stroke.

This is where a technology that can identify that a patient is experiencing a stroke would be extremely helpful and critical in helping patients receive a quick diagnosis and help them go to emergency departments in a quicker time than those studied above. The chosen area of focus is impaired speech, as slurred speech is one of the main symptoms of stroke [5]. There are a lack of studies into automated detection of stroke symptoms that focus on speech recognition. [6] found that in the contribution of Machine Learning (ML) in solving stroke-related problems, only 1 out of 39 studies report that they had identified focused on voice features as a dataset, in which this study focused on voice features that indicated a high BMI which could equate to a stroke risk [7]. Furthermore, [7] helps in the prevention of stroke and not in the diagnosis of stroke. There is also no evidence in this study [6] of which machine learning model

is best in terms of stroke audio classification, since [7] uses Logistic Regression (LR) and Random Forest (RF) models to train. Machine learning has proven to be beneficial in terms of voice recognition and classification tasks in recent years, however machine learning hasn't branched out into illness recognition as of yet. The proposed research project will investigate the use of machine learning models for the purpose of classifying audio from patients to diagnose them based on their speech signals. Successful completion of this project could inform future studies that look explicitly at stroke and the combination of symptoms that are commonly used in the diagnosis of stroke.

Therefore, the requirements and goals that are posed by this research paper are the following:

- Can machine learning classify impaired speech?
- If the above is true, which approach to ML is best at classifying impaired speech?
- Can the generated model be used in a real-world scenario to detect impaired speech?

Creating a method to identify when a person is having a stroke quickly based on the way they are speaking would go a long way to reducing time between stroke onset and arrival to an emergency department as seen in [3]. Likewise, it would also help encourage people to learn about recognising signs of stroke if there is a way to detect a major symptom.

2. Solution Description and System Requirements

2.1 Solution Description

Neural networks and speech classification have been used in many different research studies, such as in [8] which uses Convolutional Neural Networks (CNNs) to classify the intent of a sentence i.e., as a question, a reply, a remark etc., or in [9] which also proposes a CNN based approach to classify artificially generated speech signals from authentic speech signals. However, many of these studies do not explicitly look at classifying impaired speech, which is what this project will be researching.

The solution to the problem of classifying impaired speech was to construct a CNN as a baseline for the question as to whether a Neural Network can classify impaired speech. A CNN has been chosen for this as there have been successful studies done in the past such as [8], [9] which utilised this approach for different variations of healthy speech signals, so it would be a good starting point for this project to use a technology that has been used before in this field. How this will be carried out is by using a dataset of healthy and pathological samples of audio being trained using a binary classification model, with the predictions and metrics being analysed after testing has concluded to determine the success of the experiment. The dataset and pre-processing steps that would be implemented will be discussed in later chapters.

In addition, if the initial model is successful at classification then further models will be generated with different deep learning approaches, then additional model(s) will be generated with different deep learning approaches using the same training/testing data as before to analyse any performance changes as well as being able to determine which method of deep learning is the most optimal for impaired speech classification. A Long Short-Term Memory (LSTM) model will be generated, as this type of model has been shown to yield performance dividends in the past. Examples such as in [10] claim that because of their ability to learn temporal information would improve the performance of a model in classifying acoustic scenes, or in [11] which found that an LSTM had a better overall test accuracy at classifying construction site sounds compared to CNNs. Further work could be done to incorporate other Deep Learning concepts, such as an Autoencoder or a Transformer model, however for this project the focus will be on the LSTM approach.

Finally, the chosen model will be integrated into a mobile application. The application should have functionality that records audio from an individual and uploads this audio to a server that saves this audio. The audio will then be used to predict whether the individual may or may not

have stroke based on their speech, with the results of the prediction being passed back to the app to be displayed to the user. The results that the application displays to its users should help advise on whether a production scale application for stroke detection is a viable path for potential development in the future.

2.2 Speech Classification

Speech classification, also known as Natural Language Processing, is defined as “a set of tasks or problems of getting a program to automatically classify input utterance or audio segment into categories”[12]. There are multiple different use cases and real-world applications for speech classification. These include smart assistants like Siri on Apple devices, which uses a deep neural network (DNN) to identify when a person says “Hey Siri”, which will wake up the smart assistant to carry out whichever command the user wants it to do[13].

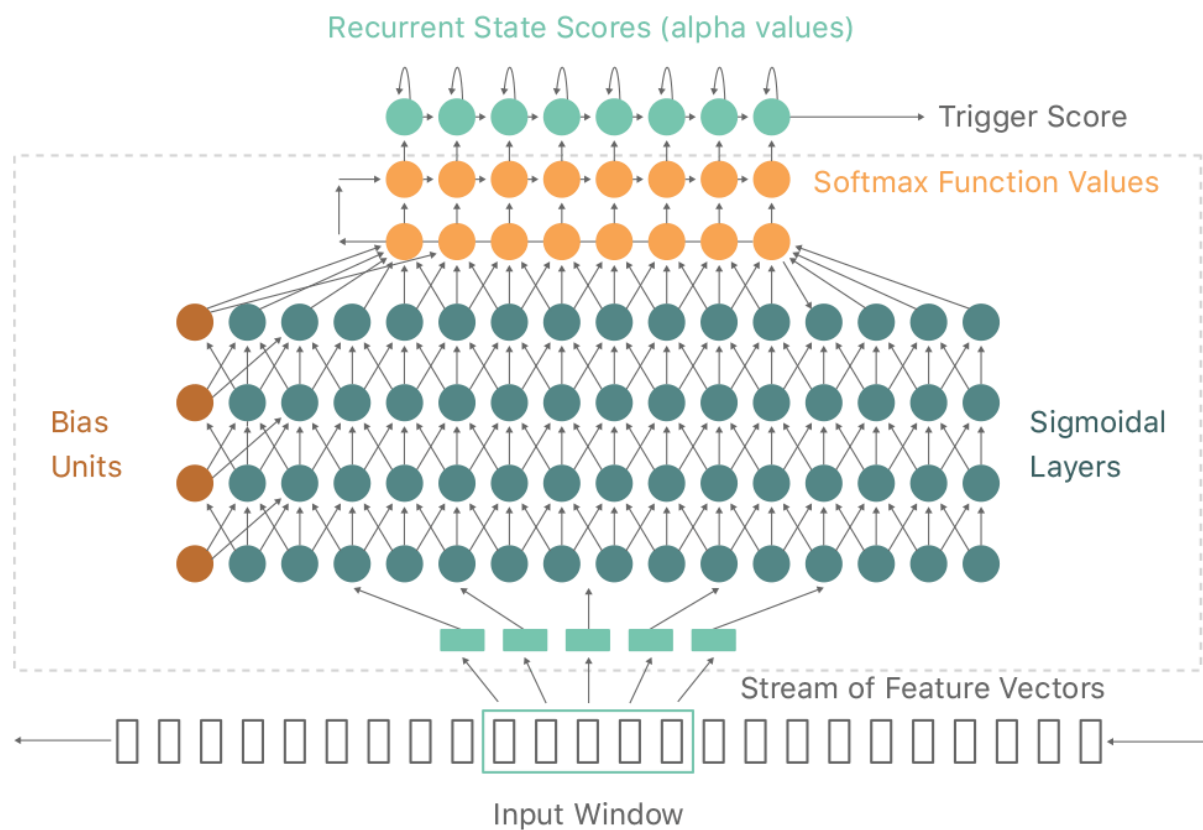


Figure 1 - "The Deep Neural Network used to detect "Hey Siri." The hidden layers are fully connected. The top layer performs temporal integration. The actual DNN is indicated by the dashed box."[13]

Another real world application for speech classification for the “Live Transcribe” feature in the Recorder application Google’s recently released Pixel 6 phone[14]. Thanks to its innovations in their new Google Tensor SoC technology, it uses artificial intelligence to classify what a person is saying in real-time and saves the words and sentences the person is saying into a

transcriptions file. It does this by having an on device CNN based upon previous work with the AudioSet[15] dataset to be able to both detect spoken words in an instant but also it allows them to not rely on data transfer between the mobile phone and a server to make the prediction for words spoken, meaning that any loss of internet connection would not impede on the model's performance at transcribing words.

The use cases explained above may provide a convenience benefit for a regular user, but these can provide some serious quality of life improvements too. In the example about live transcribe above, Google claimed[14] it can be a cheap and widely accessible way for deaf people to be able to interact in social situations and “empower this community”. The same can be said for the research and application being developed for this project. By undertaking research regarding Neural Networks detecting strokes through speech classification, there can be informed judgements made regarding this. When analysing neural networks related to stroke through their speech, and through the development of the application there is an opportunity to be able to allow stroke to be detected in sufferers quicker than previous studies have suggested, allowing them to get treatment quicker, helping reduce the overall impact of the disease and potentially increasing their quality of life in the long run.

2.3 Impaired Speech

Impaired Speech, also known as dysarthria, is defined by the NHS as “difficulty speaking caused by brain damage or brain changes later in life”[16]. There are multiple ways in which speech can be considered as impaired, such as slurred speech, a hoarse voice, issues with speaking in a consistent rhythm or difficulty moving the tongue or lips, among other things.

The main type of dysarthria associated with stroke is acquired dysarthria, which is when it occurs as a result of brain changes in later life[16]. The specific dysarthria condition that occurs with stroke is known as Aphasia, which is when speech is impaired as a result of damage to the left side of the brain[17]. Aphasia is a relatively common symptom in stroke patients, with one study reporting that out of 669 patients suffering from emergent acute stroke, 204 of these suffered with aphasia[18].

The impact of this kind of impaired speech on this project is very significant. This is because the kinds of sounds being spoken by someone without aphasia compared to someone with aphasia are very different. This can be shown from the below Figure 2 showing the difference in a sound wave of “I” in the word “scissors” in Polish said by patient with a brain injury (top) and a healthy subject (bottom)[19].

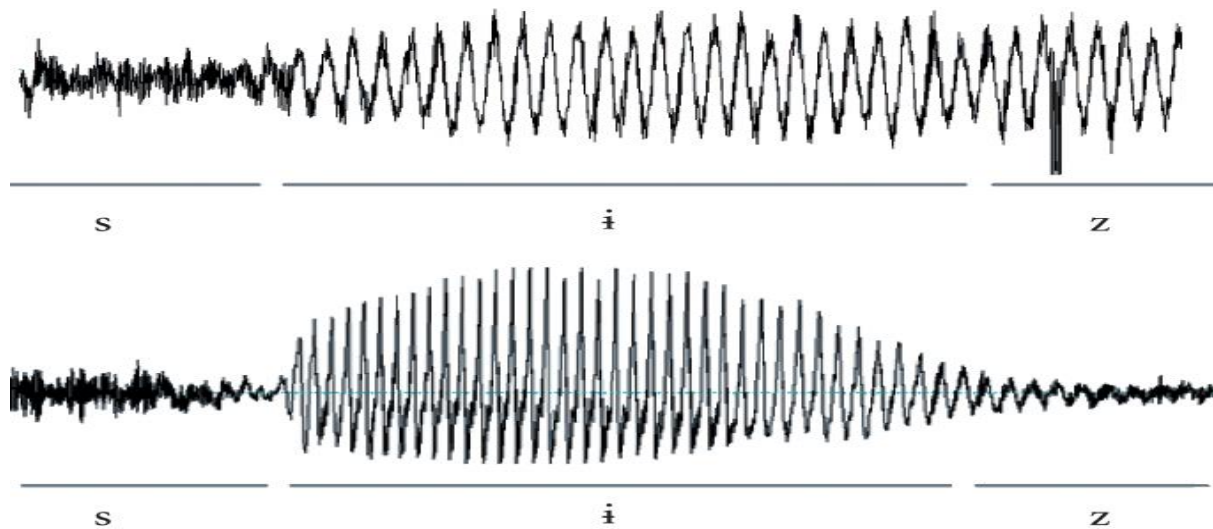


Figure 2 - "A close-up image of a sound wave of / I / in the word "scissors" pronounced as Polish /i/ produced by a TBI patient (top) and a control subject (bottom) "[19]

From the above Figure 2, the difference between the impaired patient and control patient is very clear to see. The control subject has a more distinct "I" sound compared to the impaired subject, which sees the "S", "I" and "Z" sounds drift into each other more than being pronounced as their own sounds. This study is helpful for the project as there can be a clear and defined difference between a healthy and impaired subject, leading to the potential that a well-defined dataset with similar waveforms to the above Figure 2 would potentially improve the performance of the Neural Network chosen to be developed for the application, and as mentioned previously the dataset will have an in-depth analysis in later chapters.

2.4 System Requirements

The requirements for the system are derived from the problem area as explained in Chapter 1.

2.4.1 Functional Requirements

The system should have the following:

Table 1 - Functional Requirements

ID	Description	Area	Importance
1.0	Design a Convolutional Neural Network to perform classification on a dataset of healthy and pathological speech samples.	Machine Learning	Essential
2.0	Design an additional LSTM model to compare to the baseline CNN model.	Machine Learning	Essential
3.0	Design a UI for the user to interact with the application. Should have text and a button for the user to record audio	Application (frontend)	Essential

4.0	Allow the user to press a button that will allow them to start and stop recording audio to be sent to the server for analysis.	Application (frontend)	Essential
5.0	Allow the user to upload the audio to a server so that predictions can be made against a machine learning model. This is done when the user stops recording, and a POST request is made to the server	Application (frontend)	Essential
6.0	Allow an audio file to be saved so it can be used in prediction against a machine learning model	Application (backend)	Essential
7.0	Process the audio so a prediction can be made to decide if the person has stroke or not	Application (backend)	Essential
8.0	Send the prediction from the server back to the frontend can display the outcome	Application (backend)	Essential
9.0	Display the outcome in the app so the user can make an assessment on whether the subject has stroke	Application (frontend)	Essential

2.4.2 Non-Functional Requirements

Some of these requirements have been addressed, however some could not.

The system could:

ID	Description	Area	Importance
2.1	Design an Autoencoder model to compare against the LSTM and CNN models. This would increase the depth of experimentation and give more model choices to use in the application	Machine Learning	Desirable
2.2	Design a Transformer model to compare against the LSTM and CNN models. This would increase the depth of experimentation and give more model choices to use in the application	Machine Learning	Desirable
3.1	Allow the user to preview the audio they had recorded before sending the audio to the server. This allows the user to decide if their audio is good enough to be predicted against or to record another sample	Application (frontend)	Desirable
5.1	Allow the server to handle multiple types of audio files for the prediction, i.e. .mp4 files for android and .wav files	Application (backend)	Desirable

	for iOS. This would allow the application to be ran on multiple operating systems, meaning the app can be used by more people on android and iPhone.		
6.1	Display a message indicating the progress of the prediction. This provides the user with useful information and lets them know the progress of the prediction	Application (frontend)	Desirable

3. Design

3.1 Choices in Model Architecture

As seen in Section 2.1, the models that have been chosen for classification are a CNN model and a LSTM model, which Sections 3.1.1 and 3.1.2 will go into more detail with. There are however a few parameters that will stay the same throughout both model approaches and other parameters that will vary throughout the experimentation.

Since this problem is a classification problem, the loss function being used by both models will be Categorical Cross-Entropy. A popular neural network function for measuring loss, it is defined as “the distance between the actual output probability and the expected probability. The smaller the cross entropy, the closer the distance between the 2 probability distributions”[20]. Cross Entropy was chosen instead of other loss functions such as Mean Squared Error (MSE) or Mean Absolute Error (MAE) as these loss functions are not guaranteed to minimise the loss, as “this is because MSE function expects real-valued inputs in range $(-\infty, \infty)$, while binary classification models output probabilities in range $(0,1)$ through the sigmoid/logistic function.”[21]. Next, Categorical Cross-Entropy was chosen instead of a Binary Cross-Entropy as the labels being passed into the model will be class labels of “Healthy” and “Pathological” instead of 0’s and 1’s that Binary Cross-Entropy expects. Cross-Entropy is defined by the below equation.

Equation 1 - Equation to calculate cross entropy, Assuming "p" is the probability distribution of the expected model output and "q" is the probability distribution of the actual model output[16]

$$H(p, q) = - \sum_{i=1}^n p(x_i) \log(q(x_i))$$

The activation function used for the output layers in both models will be the sigmoid activation function, represented by the below equation.

$$f(x) = \frac{1}{1 + e^{-x}}$$

The reason this was chosen as the output function is that, since the outputs of this function ranges between 0 and 1, it is especially useful in a classification problem where the probability of an output being either healthy or impaired needs to be applied to samples within the model to assess its performance.

There will also be some measures taken so that the issue of overfitting does not occur. Overfitting happens whenever a machine learning model adapts too much to a training dataset that, when it comes to testing the model against data that it has not seen before, the model will not perform as well as the metrics would suggest. Overfitting can be identified by comparing the training loss and validation loss curves on a loss chart. Below are examples of a model that has been overfitted and a model without overfitting in it.

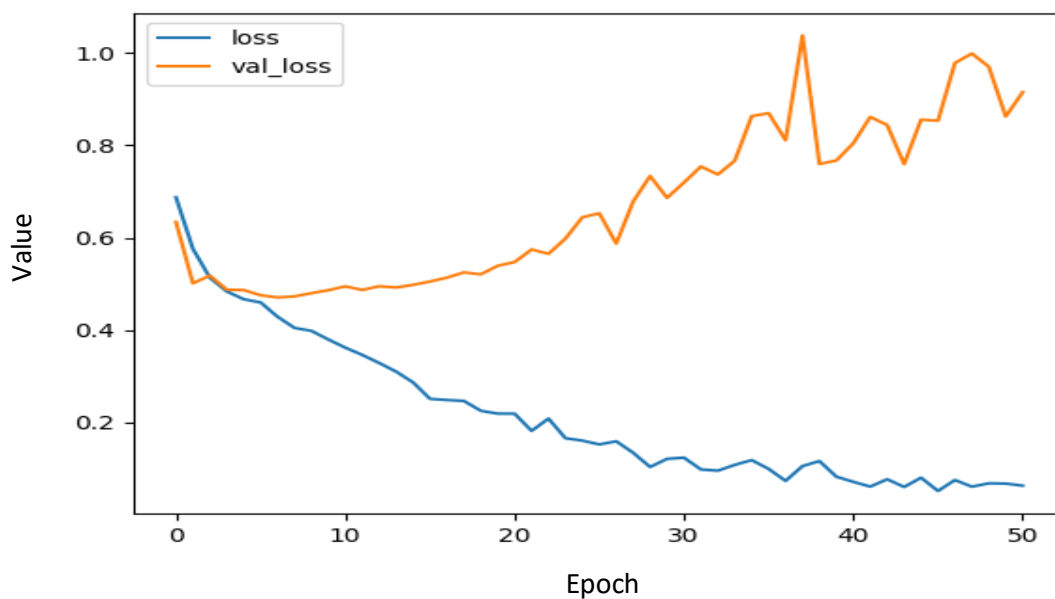


Figure 3 - Loss Chart of Model that has been overfitted to training set

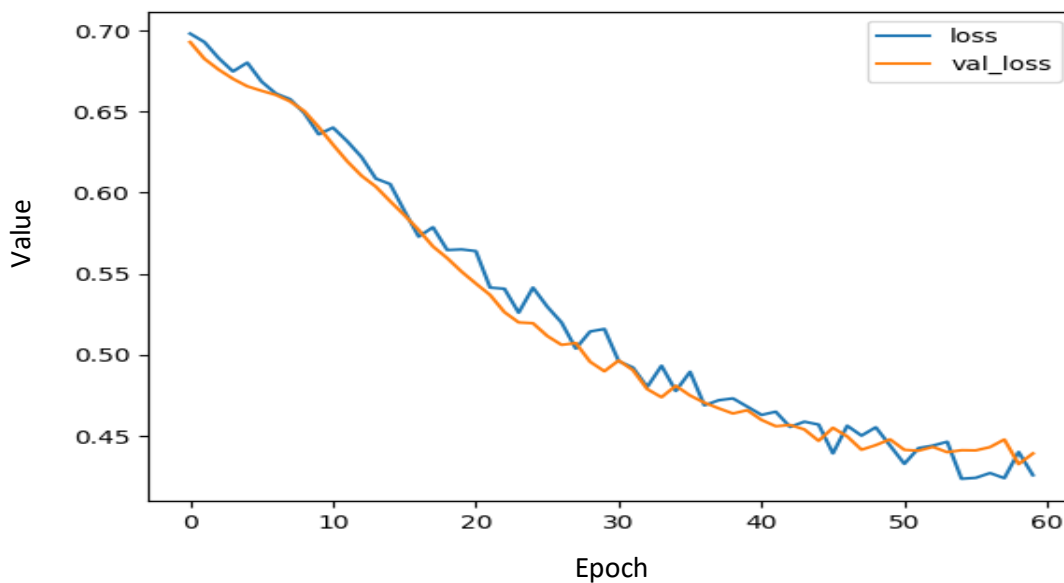


Figure 4 - Loss Chart of Model that has not been overfitted to training set

As you can see from the above Figures 3 and 4, overfitting can be identified within a model by an increase in validation loss over time while training loss decreases simultaneously. To mitigate the overfitting within the model there will be an Early Stopping call-back added to the model when fitting to monitor the validation loss and will stop training the model after 10 epochs with no improvement.

3.1.1 Convolutional Neural Network

The following Figure shows the architecture of the CNN model used to classify the audio.

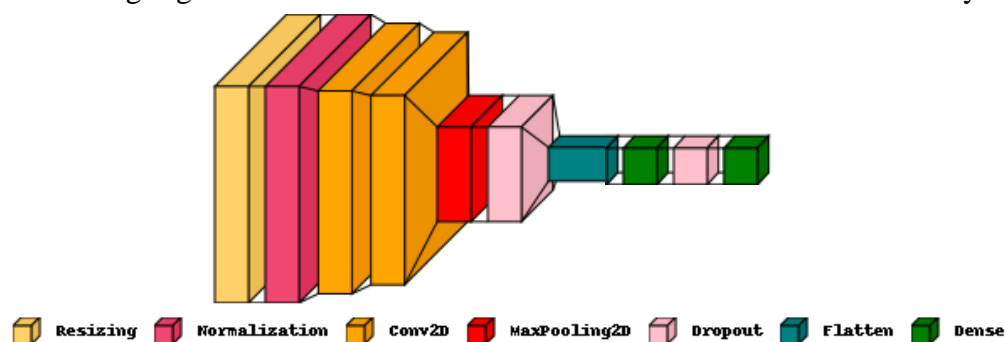


Figure 5 - Architecture of the CNN used for classification, illustrated using visualkeras as tabulated in Section 4.2

The network has 9 layers in total, comprising of:

- A resizing layer
- A normalisation layer
- 2 convolutional layers with a 'Relu' activation function

- A max pooling layer
- A dropout layer
- A flatten layer
- A dense layer with a 'Relu' Activation function
- Another dropout layer
- The output layer as discussed in 3.1

The resizing layer is needed so that the input to the layers can remain consistently the same size. By resizing the input to a smaller size, it will improve the scalability of the input and make the convolution in subsequent layers more reliable than without resizing, even though there is a risk of deforming some features with resizing input.

Once past the resizing layer, the data then will go through a normalisation layer. This will shift and scale the resized input so that the data has a distribution based around 0 and having a standard deviation of 1. This is computed using mean and variance values taken from the input which can then be used to adapt the data and restructures the layer to a normalised state. This will be useful as Neural Networks don't necessarily know the distribution of training data, so by normalising the data the future layers will have a decent foundation for calculating weights. There have also been studies carried out which shows that adding normalization to a Machine Learning problem can increase the classification accuracy of a model by an additional 2-5%[22].

The next 2 layers in this model are Convolutional Layers. Normal usage of these layers is for use in extracting features from images so the network can learn these features, usually by undertaking convolution, which is the process of passing an image through a filter, which returns a matrix of values known as activation values, with high values meaning there is a high chance of that feature within that matrix. The issue with using these layers in this project is that the samples are not images but audio samples. To do convolution the audio must be processed into an image format that can also demonstrate the values of the input at each position. This is why the input will be reformatted during pre-processing into spectrograms, specifically a Short-Time Fourier Transform (STFT) spectrogram, which specifies "complex amplitude versus time and frequency for any signal"[23]. This will provide an image that the convolutional layer can work with. More about the pre-processing steps undertook on the input will be discussed in future chapters.

There are 2 convolutional layers in the model. This was deemed appropriate because the 2nd convolutional layer can calculate the activation values from the previously calculated activation values from the 1st layer, which can help in greater determining the location of features within the image. However, adding these layers can lead to overfitting given that stacking these convolutional filters on top of an image allows more room for error and gives the weights an opportunity to lose control. The max pooling layer is then used to offset this, which is performed by applying a mask over the results of a convolutional layer and extracting the largest values within the mask to be used in layers deeper into the network, with the larger values indicating that there would be a feature within a certain area of the mask. This is represented using the below Figure 6, from this study [24].

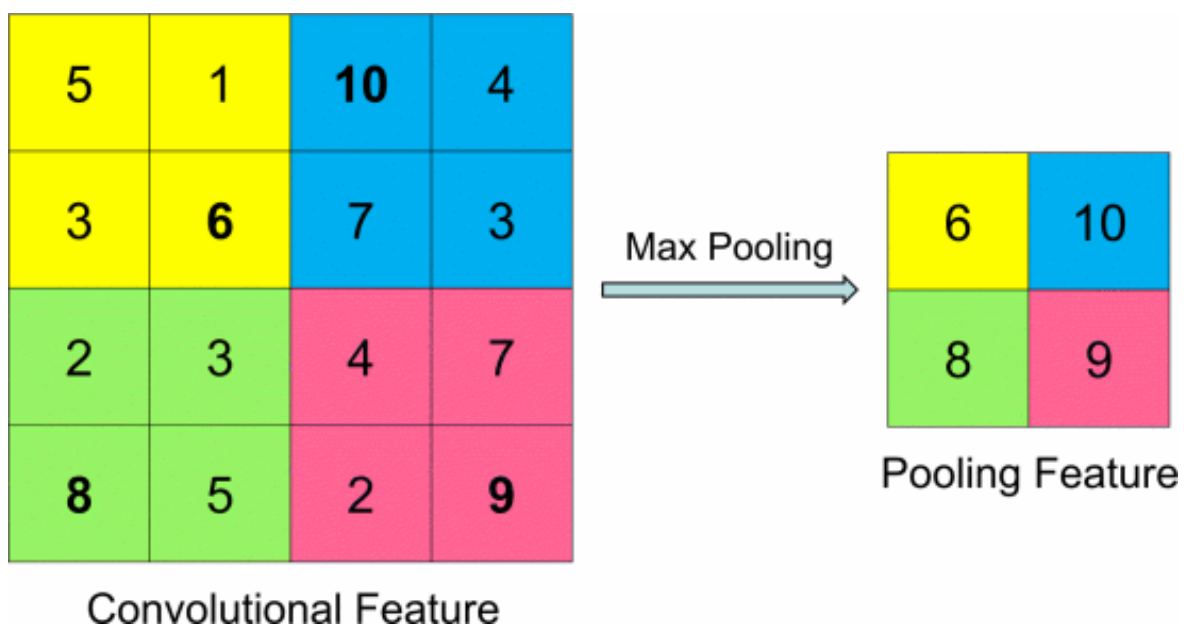


Figure 6 - An example of max pooling with a 2x2 mask[24]

After max pooling the input will go through a dropout layer. Dropout is a type of regularisation method that should help prevent overfitting within the model. In this paper[25], it explains that the concept behind dropout is to “temporarily delete some neurons in a hidden layer of the neural network with a fixed probability, simplify the model for training, and restore all neurons during testing”. Dropout has been shown to achieve good results in past studies, with this study [26] showing a drop in classification error when dropout is applied, as shown below in Figure 7.

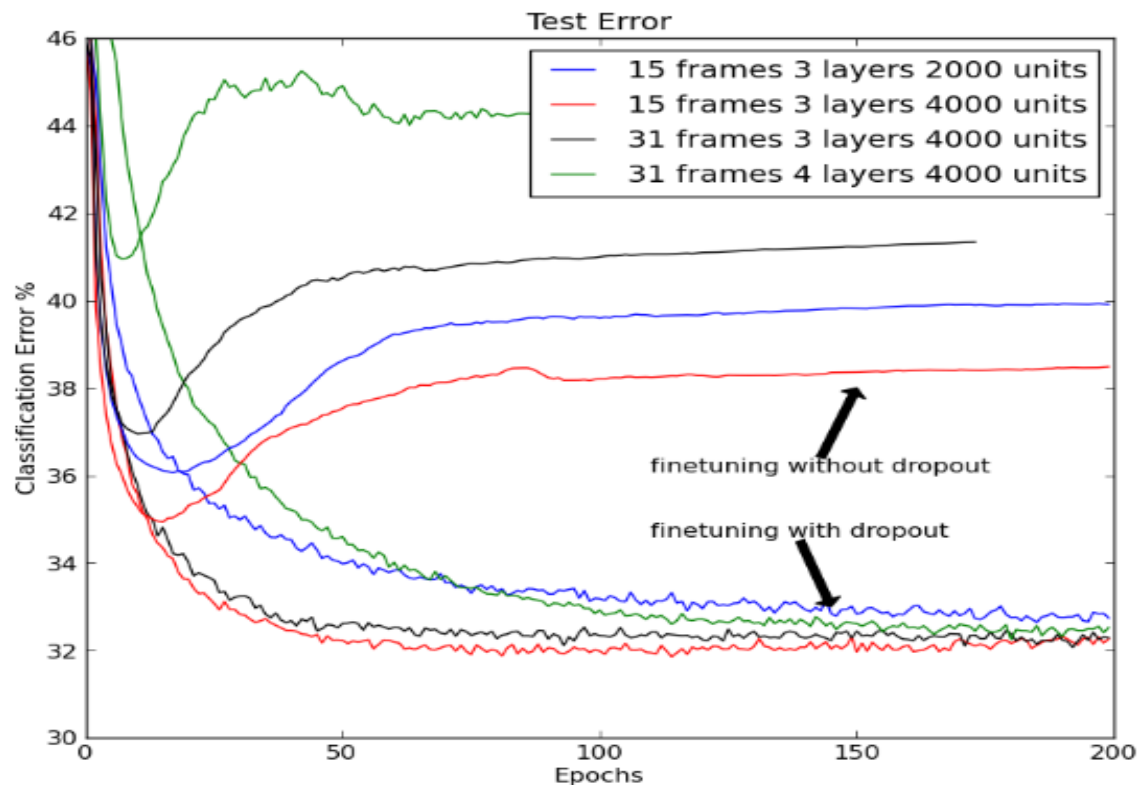


Figure 7 - Classification error rate on the TIMIT test benchmark, comparing machine learning models with different architectures with and without dropout applied. Taken from [26]

Before the input is passed through the final layers of the model, the dense and dropout layers, the input must pass through a flatten layer. This layer flattens the input into a 1-D matrix, so that the values that have passed through the previous layers can be handled by the next 3 layers more efficiently than they would have been in their multi-dimensional format. The final layers will finalise the model so that the classification can be done in the output layer, with the dropout getting rid of any unnecessary neurons before the data is classified as healthy or impaired by the output Dense layer.

3.1.2 Long Short-Term Memory Network

The LSTM network has a very similar structure to the CNN explained above. The network has 7 layers in total, which consists of:

- A normalisation layer
- 2 Bidirectional LSTM layers, with a ‘tanh’ activation function
- A dropout layer
- A flatten layer
- A dense layer with a ‘tanh’ Activation function
- Another dropout layer
- The output layer as discussed in 3.1

From above it is clear to see the similarities in the network. Both the CNN and LSTM network have the same normalisation layer to balance out the spectrogram inputs, and they both handle the input in the same way as the network gets deeper, with the same architecture in terms of dropout, flatten and dense layers. The critical difference between them are the layers between, with this network having bidirectional LSTM layers instead of convolutional and max pooling layers.

LSTMs are a concept in machine learning under the concept of Recurrent Neural Networks (RNNs). RNNs are described as “a class of neural network that are naturally suited to processing time-series data and other sequential data”[27]. They do this by having what is known as a recurrent cell (or layer) that updates every time a new input is passed to the RNN, then once the update is complete then the state of the cell is passed back to the model for when a new input is passed into the model. The issue with these types of basic RNN is that they only have a short-term memory in that, if a sequence is long enough then there is potential for the model to forget the information it has gathered in earlier stages, meaning that essential context may have been discarded from the model by the time it comes to predicting against inputs in a test set.

This is where LSTMs can help mitigate this issue of short memory. The architecture of the LSTM cell differs from a RNN cell in that, as well as having a recurrent cell in its build up, there is also a “memory cell” which stores information kept from all previous inputs so that the information is not lost with inputs later in model generation, as well as a “forget gate” which removes the information that is no longer relevant to the model with each new input, an “input gate” which helps with updating the information within the memory cell, and an “output gate”

which actually generates the output from the LSTM cell at that particular input. The comparisons between a basic RNN and an LSTM can be shown in the below Figure 8 from this paper[28].

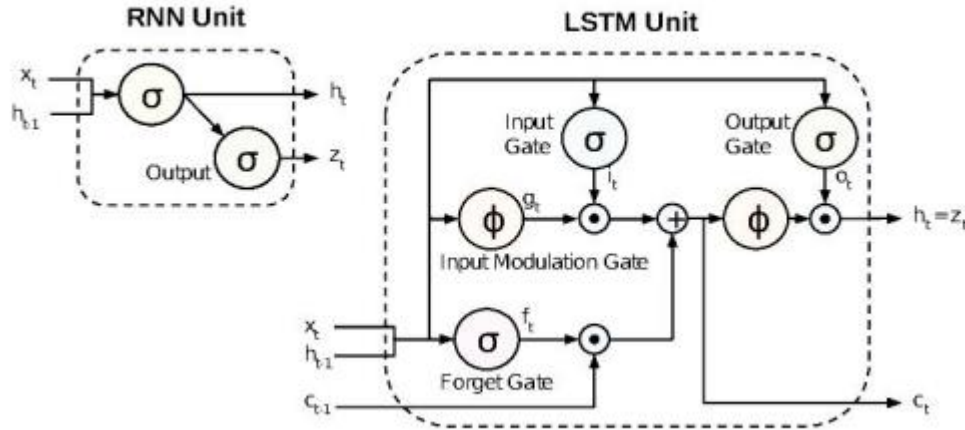


Figure 8 - Illustration showing the differences between a standard RNN cell and an LSTM cell, with additional memory cell and the various gates[28]

A regular LSTM network is a feed forward neural network, meaning that the sequence of information within the model flows forward from the past data to the future. However, the performance of the model can be improved when making the LSTM cell bidirectional. This will mean that instead of learning the sequence of information from past to future, the model will also be able to learn from flowing backwards (from future to past) simultaneously, and the differences between this can be represented in the below Figure 9 from this paper[29], and since audio samples are in themselves sequences of data in the form of sound, this bidirectional approach to classifying audio should in theory perform better than the CNN model proposed earlier.

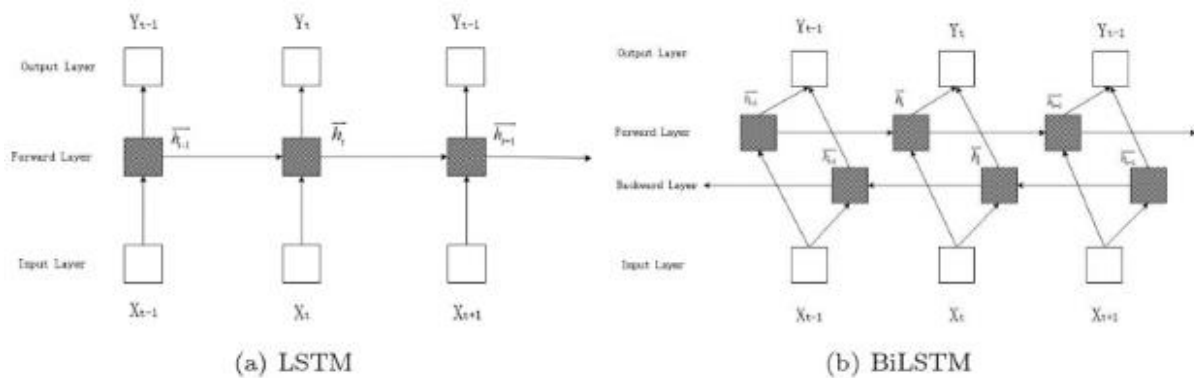


Figure 9 - An illustration showing the difference between a LSTM (a) network and a Bidirectional LSTM (b) network[24]

3.2 Data Preparation and Pre-processing

To make sure the data is in a state that it can be used by the Neural Networks, the data must be prepared so that it can be processed correctly. This procedure is known as pre-processing, and has been shown to have had a very large impact on the outcomes of machine learning experiments, with one study showing that data being passed through a neural network showed a 25.39% reduction in false positive rate when pre-processing steps were applied versus when no pre-processing was applied[30].

There are many pre-processing steps that can be applied to data, however there are 4 main categories of pre-processing that help get data in a usable state and have many subcategories. These are:

1. Data Cleaning – This is done to help “clean” the data that is passed in. This usually involves replacing missing values, modifying the samples to solve inconsistencies smoothing noisy data etc.
2. Data Integration – This is the process of taking data from multiple different sources and merging them to create one larger dataset. This is a good idea in theory, however in practice, this may be tricky, as data from different sources may have different attributes that make them stand out among other samples, and so there is a chance this degrades the data overall, so sometimes this step is not appropriate unless the data being merged has very similar attributes.
3. Data Reduction – This is the process of reducing the number of samples in the dataset to a point that makes analysing the data easier yet does not degrade or may even improve the performance of the model. This reduction may also improve the resources needed to run the model, such as occupying less storage space or RAM at execution time.
4. Data Transformation – This is the process of changing the format or structure of the data so that it is in a more desirable format for model generation.

Most of the above steps will be implemented within the chosen dataset for this model, however this will be explained more in Section 4.3

3.3 Mobile Application Architecture

The general overview of the mobile applications architecture is shown in the below Figure.

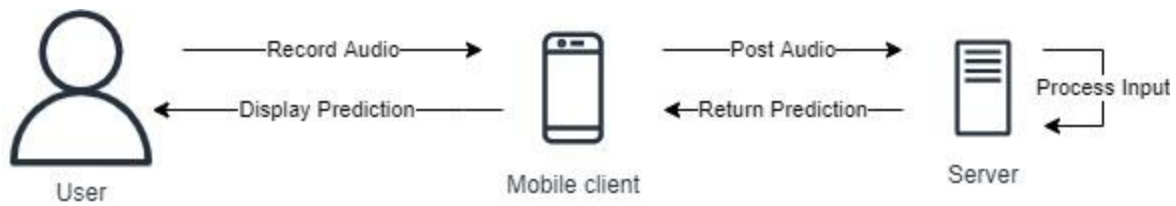


Figure 10 - Overview of the Mobile App Architecture

The architecture choice made for the mobile application was to use a client-server architecture, with using specifically a mobile device as the client. This decision was made to as mobile devices are extremely common in the real-world. According to [31], as of February 2022 there was a total of 6.259 billion smartphone users worldwide in 2021, with a projection of 6.567 billion by the end of 2022. Based on a worldwide population of 7.8 Billion people as of December 2021, derived from the UNs World Population Prospects 2019[32], it means that 80% of the worldwide population owned a smartphone by the end of 2021. As well as being a very common device, smartphones also have an abundance of features that would help in the collection of data, such as cameras and microphones. The microphones in particular will help in this case, since they will be necessary in recording the user's speech samples.

However most mobile phones may not have the computation power to run some of the machine learning predictions generated by their respective scripts. This therefore would introduce the server aspect of the application. For the client to be able to send audio data to the server, the recorded audio must be included in a HTTP request to the server from the client. In particular, this will be a HTTP POST request, which has the ability to tell the server to save the data that is included within this specific request. This request will be made to a specific URL known as an API Endpoint, which is where requests made by an application are carried out[33].

Once this POST request is made, the server then must have the ability to save the data locally to disk and the ability pass this data through to the chosen model as well, which will be used against the audio sample from the client to assess whether or not they have stroke. Once these predictions have been made, the server must then choose an output based on the prediction and return this prediction back to the client device to be used to inform the user of their stroke status. The architecture of the server side of the client-server architecture goes into more detail in the below Figure 11.

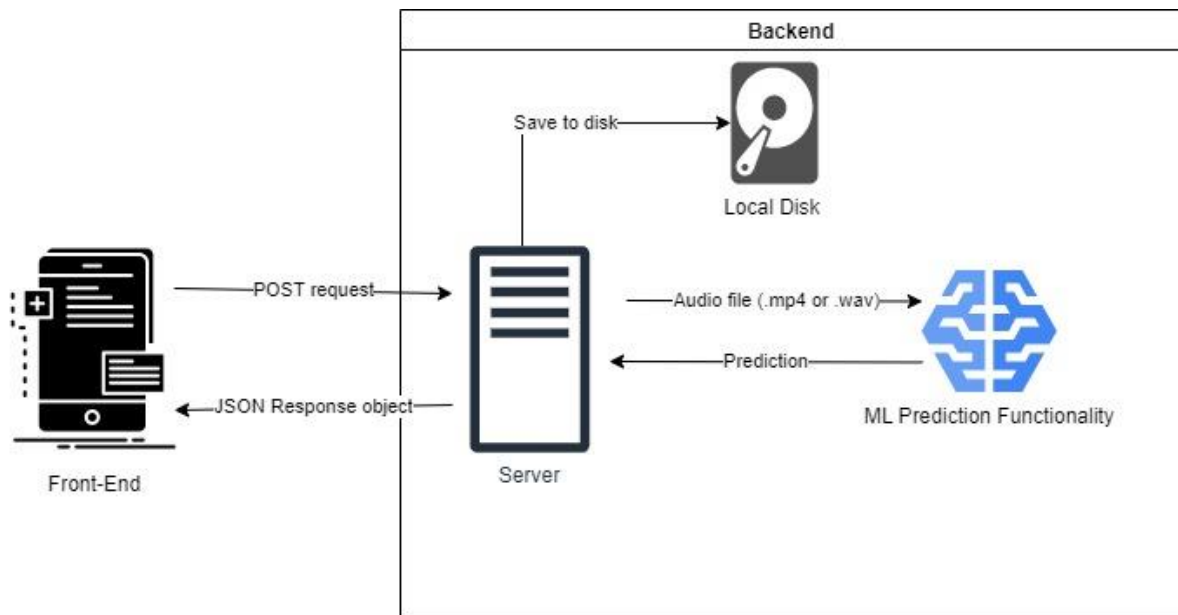


Figure 11 - Basic Architecture of the Backend

3.4 Potential Event and Error Handling Measures

Within the machine learning scripts, since the model was generated from a dataset that was already pre-processed, the events and errors that could potentially arise would be working with directories and retrieving the labels from the directories:

- When classifying the audio based on speech impairment alone, derive the class label based on the directory that the audio files are held in, i.e., Healthy or Pathological.
- Likewise, if the classification is based on both speech impairment and the tone of voice, derive the class label from the filename of the audio sample, which will have the impairment and tone on it alongside the sample ID, i.e., “4_Healthy_Neutral.wav”

Within the application, there is also some error handling measures that would need to be addressed in the backend when it comes to accessing certain recordings.

- Android and iOS both handle audio files in different fashions. The most compatible format for android is .mp4 for audio (sent in the audio format of .m4a), whereas iOS uses .wav files mainly. This needs to be accounted for on the server side of the application. If the file being sent over is an .mp4 file, then it must be converted to .wav
- Likewise, error handling will be added in the cases that certain audio files being passed into the server does not exist. This should be handled in an error message being sent to the user asking them to try again if the recording does not exist.

4. Implementation

4.1 Implementation Decisions: Languages and Environments

4.1.1 Machine Learning Scripts

The system was implemented using the Python programming language, specifically Python 3.9. This is a very powerful yet simple language to understand and is used in a variety of different applications, from API services to data science and machine learning projects such as this one. It has a vast array of libraries which can easily be installed on the command line to be used within these types of projects, as well as a large online community with tutorials and walkthroughs which helps when facing certain challenges in development. For these reasons it made sense to choose Python as the language used to implement the models.

The specific machine learning library being used in this project is Tensorflow. This is an open-source machine learning framework created by Google which utilises tensor objects to allow the creation of machine learning pipelines for many different project types, such as image classification, time-series predictions, Natural Language Processing projects, among other use cases. Tensorflow was chosen for this project because of its integration with Keras. Keras is an open-source deep learning API that allows developers to build, generate and evaluate machine learning pipelines, and paired with Tensorflows ability to manipulate data to fit the problem, with multiple methods of reading in various datatypes and having many pre-processing functions built-in. This alongside the fact that the Tensorflow website has many tutorials and guides to help a beginner build their own machine learning pipelines, it made sense to choose Tensorflow and Keras for generating the CNN and LSTM models.

Python, Tensorflow and Keras help in satisfying requirements 1.0 and 2.0 in that they allow the development of the CNN and LSTM models specified in the requirements, however they can also satisfy the non-functional requirements 2.1 and 2.2 also if the Autoencoder and Transformer models were to be developed in the future.

4.1.2 Front-End Application

Since it would be necessary to run on multiple mobile operating systems, there was a need to develop the front end with a technology that has a high portability and high adaptability so that the same files can be run on all operating systems. This was made possible by developing in JavaScript, a lightweight programming used by many web services and APIs[34]. There are many frameworks that can be utilised in JavaScript, such as Node.js[35], however the framework that was chosen for the front end of this application was Expo. Expo is a

framework built around the React-Native software library to develop applications for both Android and iOS using a single JavaScript codebase[36]. Expo makes the access of the sensors mentioned in Section 3.3 easily accessible through its SDK, meaning that it would be achievable to create a front end that can record audio on both iPhones and Androids, which can help satisfy requirements 3.0, 4.0, 5.0, 5.1 and 9.0 as a result, in that:

- Requirements 3.0, 4.0 and 5.0 can all be easily programmed using the JavaScript and Expo mentioned previously to create a UI that the user can interact with to send off the audio samples, while Requirement 9.0 can be satisfied by updating text parameters once the result from the server has been returned.
- Requirement 5.1 is partly satisfied due to the nature of Expo itself. It has audio libraries, which will be explained more in Section 4.2, that allow the same application to generate multiple types of audio file depending on the device interacting with it. It will understand if a device is Android or iOS and generate the correct format for that specific operating system

Using JavaScript and Expo/React-Native also has other benefits, like mentioned with Python in Section 4.1.1. There is a vast online community with tutorials and walkthroughs on different aspects of React-Native, as well as the Expo documentation itself being very detailed in explaining its various libraries. The React-Native library also has tools and functionality for implementing HTTP requests needed for the client side of the client-server architecture. This is done using the built FormData interface[37] to add the audio and other information such as file type and filename into a POST request. This will then get sent to the server using fetch[38], which is an asynchronous method that will send the POST request to the server and retrieve the output back from it in the form of a response object, which can then be broken down into specific parts so that the data being returned can be used to notify the user of what the server has predicted.

4.1.3 Back-End Server

The backend server uses the lightweight Rest API library known as Flask. It is a lightweight web application microframework that makes it easy to create HTML servers using Python[39], [40]. This server can retrieve the HTTP POST request made by the client side, retrieve the files sent in that request and be able to parse the audio file from the request. From there it can send the audio file to the prediction script, which will perform the same pre-processing steps as the model experimentation, which will be explained in greater detail in

Section 4.3, so to get the audio in a state that can be assessed against the model. This may include converting a .mp4 file to a .wav file if the audio comes from an Android device, using the AudioSegment function held within the pydub Python library[41]. Once this plus the pre-processing takes place, the script will make its prediction. The output from this script will determine the contents of the JSON response, with different messages being returned with different diagnoses. When the message is confirmed, it is added to a JSON object to be returned to the client side along with a status code of 200, implying that the request to the server was successful.

Using this helps satisfy a number of requirements:

- The functional requirements 6.0, 7.0 and 8.0 are satisfied since the audio gets saved to disk once the server processes the request (6.0), the prediction script held within the server is used to predict whether someone has stroke or not (7.0), and the client receives the result of the prediction via the JSON object returned (8.0)
- Requirement 5.1 is fully satisfied at this point. This is done via the prediction script in that if the audio being passed in is a .mp4 file, then it is converted to a .wav file as mentioned in the previous paragraph. This combined with the use of Expo means that multiple operating systems will be able to use this application.

4.2 Software Libraries

Below is a table which shows the software libraries used within the project, along with descriptions of what they are and how they helped in the generation of the models:

Table 2 - The software libraries used in development of the Neural Network pipelines and the mobile application

Library	Description	Area of System
Tensorflow	Used for importing the Tensorflow packages. Version 2.8.0 is used in this project	Machine Learning/Backend
Tensorflow.keras.models	Used for generating a sequential model for training deep neural networks	Machine Learning
Tensorflow.keras.layers	Used to define the layers within the deep neural networks. Imports functions such as Conv2D, MaxPooling2D, Bidirectional, LSTM etc.	Machine Learning
Tensorflow.keras.optimizers	Imports the optimizers to be used within the compilation of deep learning models, e.g., Adam	Machine Learning

Tensorflow.keras.losses	Imports the loss functions to be used within the compilation of deep learning models, e.g., CrossEntropy	Machine Learning
Tensorflow.keras.callbacks	Imports the call-backs to be used within the compilation of deep learning models, e.g., EarlyStopping	Machine Learning
Tensorflow.audio	Allows Tensorflow to be able to manipulate audio files so they can be turned into tensor objects	Machine Learning/Backend
Tensorflow.io	Allows Tensorflow to read in files and work with directories on the local machine	Machine Learning/Backend
Tensorflow.signals	Allows the tensor objects of audio signals to be interpreted as various types of formats, e.g., spectrograms	Machine Learning/Backend
Tensorflow.data	An API that allows the generation of datasets from tensor objects within the program	Machine Learning
Tensorflow.keras.models.load_model	Used in the backend to load the model generated in the machine learning scripts so that the audio passed to the server can be predicted against it	Backend
Tensorflow.errors	Used to implement error handling in the backend, e.g., if an incorrect filepath was passed to the prediction script	Backend
NumPy	Library used for handling matrices and vectors, since tensors are multi-dimensional vectors by default	Machine Learning/Backend
Matplotlib.pyplot	A python plotting library used to generate graphs and charts like the ones used in Figures 3 and 4	Machine Learning
Seaborn	Aids in generating the heatmaps in confusion matrices compiled during the testing stage of model generation	Machine Learning
OS	Needed to work with files held locally on the system	Machine Learning
Pathlib	Used for working with filepaths in the format that python can understand	Machine Learning
VisualKeras & PIL	Used to help generate and display the architecture of the CNN model shown in Figure 6	Machine Learning
Flask	Used for importing the Flask packages. Version 2.0.3 was used in this project	Backend

Flask.request	Allows the server to handle incoming HTTP requests from the client side	Backend
Werkzeug.utils.secure_filename	Allows files sent in HTTP requests to be saved with specific naming	Backend
json	Allows the output from the prediction to be sent back to the client in a format it will be able to parse	Backend
Pydub.AudioSegment	Allows the backend to manipulate .mp4 files into .wav files so the tensorflow.audio library can work with the sample passed in from an Android device	Backend
Expo-AV	Allows audio inputs into the mobile device to be handled, such as the starting/stopping of recording, saving of audio etc	Frontend
@flyerhq/react-native-android-uri-path	Used to parse URI filepaths passed in from mobile devices to an input readable by the Windows CLI	Frontend

4.3 Custom Dataset Pre-processing: Saarbruecken Database

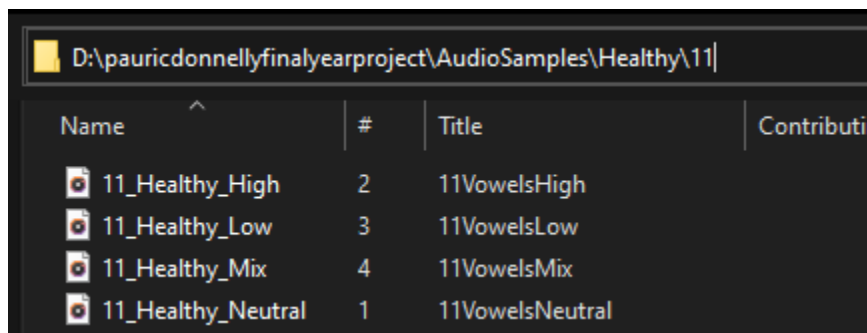
The dataset that is being used for this project is the Saarbruecken Voice Database[42]. This is a database with recordings of over 2000 people with different styles of audio recordings:

- “Recording of the vowels [i, a, u] produced at normal, high and low pitch
- Recordings of the vowels [i, a, u] with rising-falling pitch
- Recording of the sentence "Guten Morgen, wie geht es Ihnen?" ("Good morning, how are you?")” [8]

This dataset is appropriate for this use case as it holds samples of both healthy audio and pathological audio, which will be able to be used within both models to train, validate and test each model. The Saarbruecken database is also a publicly available database, so procurement of the data was a simple procedure.

Before the data is to be passed into these various models, the audio must be pre-processed before being used to train a model. This is required so that prediction estimation in both Machine Learning and Deep Learning models, proved in such studies as [30] as previously discussed. There were multiple steps that took place to improve the raw data:

- **Data Augmentation** - A drawback of the Saarbruecken database is that a proportion of the pathological samples have very similar characteristics to that of the healthy audio, in that it was very hard to tell from listening to the speech sample that the subject had a speech disorder. It therefore became necessary to gather pathological audio samples that could be discernible to the ear as “unhealthy”. The sample size of full .wav audio files was shortened to a total of 202, with an even split of healthy and pathological data amongst them. However, because of the nature of the audio samples, it became possible to apply a technique known as Data Augmentation, defined as “a suite of techniques that enhance the size and quality of training datasets such that Deep Learning models can be built using them”[43]. Since audio recordings in the Saarbruecken database had 4 different pitches of vowel sounds per audio sample, it became possible to extract all different variations of vowel sounds so that for each full audio sample given, 4 audio samples could be generated. This was done manually using Audacity, which is an “open-source audio editing software”[44]. Through editing these audio files, a total of 808 samples of audio were gathered. The directory hierarchy of these audio samples is saved as “AudioSamples/<AudioClass>/<PatientID>/<PatientID>_<AudioClass>_<SamplePitch>.wav” (as shown in Figure 11 below), and all of the audio files are hosted on the project Gitlab repository using Git Large File Storage[45].



Name	#	Title	Contributi
11_Healthy_High	2	11VowelsHigh	
11_Healthy_Low	3	11VowelsLow	
11_Healthy_Mix	4	11VowelsMix	
11_Healthy_Neutral	1	11VowelsNeutral	

Figure 12 - File Hierarchy of a sample of augmented data

- **Waveform Conversion** – At execution of the experiment script, the audio in the dataset is loaded into the script and decoded so the shape of the audio can be visually examined, as shown in Figure 12 below.

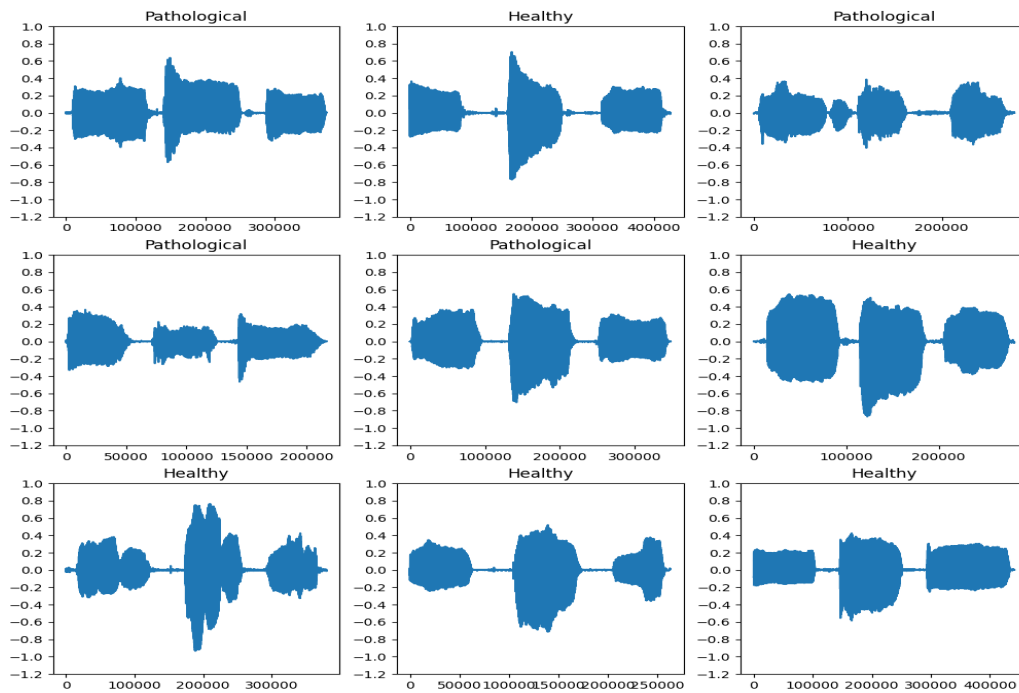


Figure 13 - Audio samples plotted as waveforms

- **Spectrogram Conversions** – The waveforms are converted to spectrograms via a STFT as explained previously. This results in a spectrogram for each audio sample in the dataset, like the one shown in Figure 13.

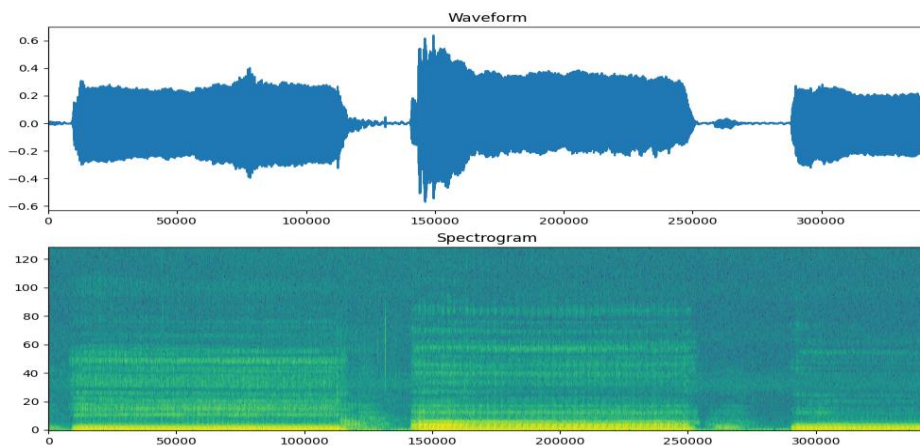


Figure 14 - Waveform represented as spectrogram

- **Padding** – The average shape of 100 waveforms is calculated and applied so that any audio sample that is lower than the average would have padding applied, so that all samples would be of a similar size. This is shown in Figure 14 below.

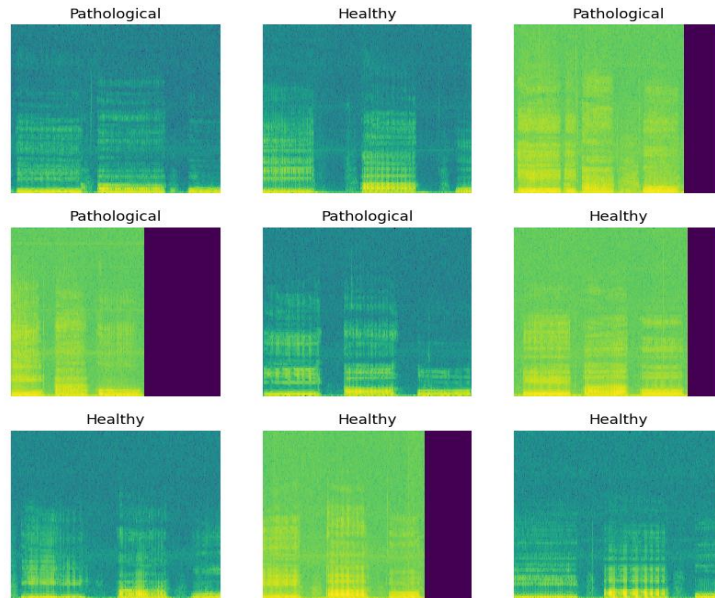


Figure 15 - Padded vs Un-Padded spectrograms

Once these steps are carried out the data can then be passed into the model, go through their normalization process, and hopefully be pre-processed to a degree that the model can work with the data seamlessly.

5 Testing

5.1 Model Experimentation

5.1.1 Outline of Experimental Dataset

For these experiments the dataset will be split into 3 different sets for training, validation and testing, with a ratio split of 70:15:15. The reason for this is due to the size of the dataset being relatively small compared to other benchmark audio datasets used in machine learning pipelines, such as the UrbanSound8k dataset[46]. This would mean that if the ratio between training and validation/testing was lower, then there would not be sufficient data to accurately evaluate the performance of the models and the results may be flawed. Therefore, a split of 70:15:15 has a large enough training size to train the models while also having an adequate validation/testing set size to accurately judge the models.

There will be multiple metrics used to evaluate the performance of the models. These values will be obtained by passing an unknown sample from the testing set and storing the result, with the total results being saved and displayed in a confusion matrix. Values within the matrix include True Positive (TP) predictions, True Negative (TN) predictions, False Positive (FP) predictions and False Negative (FN) predictions, with TP being correct healthy predictions, FP being incorrect healthy predictions, TN being correct pathological predictions, and FN being incorrect pathological predictions. These 4 values can be used to derive multiple performance measuring metrics. These are derived as follows (Note – let “n” equal the total number of samples within the test set):

- Accuracy – The percentage of all samples correctly classified in the test set. Defined as $\frac{TP+TN}{n}$
- Error Rate – The percentage of all samples incorrectly classified in the test set. Defined as $\frac{FP+FN}{n}$
- Recall – The percentage of positive samples correctly classified within the subset of all ground truth positive samples (also known as Sensitivity). Defined as $\frac{TP}{(TP+FN)}$.
- Precision – The percentage of positive samples correctly classified within the subset of all predicted positive samples. Defined as $\frac{TP}{(TP+FP)}$.
- Specificity – The percentage of negative samples correctly classified within the subset of all ground truth negative samples. Defined as $\frac{TN}{(TN+FP)}$.
- F-Measure (F1) – “The number of distinct test cases to detect the first program failure”[47]. Defined as $\frac{2*(Precision*recall)}{(Precision+Recall)}$.
- False Alarm Rate – The percentage positive samples incorrectly classified within the subset of all ground truth negative samples. Defined as $\frac{FP}{(TN+FP)} = 1 - Specificity$.

5.1.2 Experimental Methods – LSTM & CNN

The final architecture of the CNN and Bidirectional LSTM models is shown in the below Figures:

```
model = models.Sequential([
    layers.Input(shape=input_shape),
    # Downsample the input.
    layers.Resizing(32, 32),
    # Normalize.
    norm_layer,
    layers.Conv2D(32, 3, activation='relu'),
    layers.Conv2D(64, 3, activation='relu'),
    layers.MaxPooling2D(),
    layers.Dropout(0.25),
    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(num_labels, activation='sigmoid'),
])
```

Figure 16 - Final Architecture for CNN model

```
# Defining layers within Neural Network
model = models.Sequential()

model.add(Input(shape=input_shape))
model.add(norm_layer)
model.add(Bidirectional(LSTM(32, activation='tanh', return_sequences=True)))
model.add(Bidirectional(LSTM(64, activation='tanh')))
model.add(Dropout(0.25))
model.add(Flatten())
model.add(Dense(128, activation='tanh'))
model.add(Dropout(0.5))
model.add(Dense(num_labels, activation='sigmoid'))
```

Figure 17 - Final architecture of BiLSTM model

Both models will be run at 150 epochs, with the model generation stopping early if the validation loss does not improve after 10 epochs, as discussed in previous chapters. The learning rates and batch sizes within each experiment will be varied. This will be a combination of batch sizes 1, 32, 64, 128, 256 and the full batch of training and validation sets being passed in, and each of these batch variations will be run at learning rates of 0.0001, 0.001 and 0.01 to determine the optimum parameters for each model. It is important to note however that studies such as [48] suggest that an increase in batch size will result in an increase in test accuracy.

5.1.3 Results

The following tables are the results of experimentation of the CNN model at their different learning rates.

Table 3 - CNN Experiment results at LR = 0.0001

Model	LR	Batch	Accuracy	Error Rate	Recall	Precision	Specificity	F1 Score	False Alarm
CNN	0.0001	1	77	23	79	74	75	77	25
CNN	0.0001	32	74	26	74	74	73	74	27
CNN	0.0001	64	80	20	88	76	72	82	28
CNN	0.0001	128	80	20	89	76	72	82	28
CNN	0.0001	256	79	21	75	77	82	76	18
CNN	0.0001	Full	79	21	84	78	75	81	25

Table 4 - CNN Experiment results at LR = 0.001

Model	LR	Batch	Accuracy	Error Rate	Recall	Precision	Specificity	F1 Score	False Alarm
CNN	0.001	1	72	38	62	78	82	69	18
CNN	0.001	32	68	32	61	69	74	65	26
CNN	0.001	64	69	31	73	69	63	71	37
CNN	0.001	128	69	31	80	66	58	73	42
CNN	0.001	256	69	31	62	74	75	68	25
CNN	0.001	Full	72	28	76	74	67	75	33

Table 5 - CNN Experiment Results at LR = 0.01

Model	LR	Batch	Accuracy	Error Rate	Recall	Precision	Specificity	F1 Score	False Alarm
CNN	0.01	1	63	37	78	56	51	65	49
CNN	0.01	32	64	36	70	60	60	64	40
CNN	0.01	64	74	26	77	69	71	73	29
CNN	0.01	128	71	29	91	67	49	77	51
CNN	0.01	256	75	25	81	71	70	75	30
CNN	0.01	Full	80	20	97	76	57	85	43

From Tables 3-5 there are some interesting findings throughout. The best accuracy achieved by the CNN model was 80%. This was achieved with a LR of 0.0001 and batch sizes of 64 and 128, while the LR of 0.01 and a full batch size also achieving this. The worst accuracy was

achieved with a LR of 0.01 and a batch size of 1 with 63% accuracy. With a range of 17% between best and worst accuracy, there is a case to be made against the consistency of the CNN approach to impaired speech classification.

The recall metric continues this theme of inconsistency in the CNN approach. The best recall value occurs with a LR of 0.01 and a Full Batch size at 97%, while the worst occurs at a LR of 0.001 and a batch size of 32 with 61%. This is a range of 36% between both values, which shows that there are times when the model can correctly classify healthy samples at an almost perfect rate, and other times where the model can classify many healthy samples as pathological, further proving that the CNN approach is heavily inconsistent in its performance.

Precision continues to enforce how inconsistent this model can be. The best value was 78%, occurring twice at a LR of 0.0001 and a Full batch as well as a LR of 0.001 and a batch size of 1. The worst value is 56% occurring at a LR of 0.01 and a batch size of 1. This suggests that there are potentially a lot of pathological classifications being predicted as healthy as opposed to the vice versa happening. This because Precision includes False Positives in the calculation (which is incorrect healthy predictions as mentioned in Section 5.1.1), and since the precision highest and lowest precision values are both lower than the Recall, which includes False Negatives in its calculation (which is incorrect pathological predictions as mentioned in Section 5.1.1), an assumption can be made that the models are worse at correctly identifying pathological samples than healthy samples, which in the context of identifying stroke, is paramount as to misclassify a person suffering from stroke as healthy could have life threatening consequences.

This point is further clarified when looking at specificity, which could arguably be the worst performing metric from the CNN experiments. The best value was 82%, occurring at a LR of 0.0001 and batch size of 256, and at a LR of 0.001 and a batch size of 1. The worst value was 49%, occurring at a LR of 0.01 and a batch size of 128. The range between the highest and lowest values is 33% which not only emphasises how inconsistent the model performs, with some models having a decent classification rate of pathological samples and others more often predicting pathological samples as healthy, but again with highest and lowest values smaller than recall further proves the point that these models do not handle generally handle pathological samples as well as healthy samples.

The F1 score does hint at some improvements. The highest score occurs at a LR of 0.01 and a full batch size with 85%, and the worst result of 64% occurring at the same LR but with a batch

size of 32. While this does show that the models are performing better than some of the stats suggest, this metric can sometimes be deceiving. The model that has the highest F1 score at 85% also has the third lowest specificity score at 57%, meaning that the F1 score is more of a reflection on a models ability in handling healthy samples than pathological samples, which in this context of stroke detection, may not be the most helpful.

To conclude the analysis on the CNN model, the biggest trend throughout testing that was discovered is how inconsistent the performance of the CNN can be. There are some runs that have high accuracy and combination of other metrics that reflect well on the model, but there are some runs that do not classify the data well at all, and the metrics reflect this. It may be the case that the inconsistency of the CNN may be main reason why the NN is not chosen for the final solution in the mobile application.

The following tables are the results of experimentation of the Bidirectional LSTM model at their different learning rates.

Table 6 - BiLSTM Experiment Results at LR=0.0001

Model	LR	Batch	Accuracy	Error Rate	Recall	Precision	Specificity	F1 Score	False Alarm
BiLSTM	0.0001	1	78	22	77	78	79	77	21
BiLSTM	0.0001	32	82	18	82	87	82	84	18
BiLSTM	0.0001	64	81	19	83	79	79	81	21
BiLSTM	0.0001	128	78	22	85	71	71	78	29
BiLSTM	0.0001	256	83	17	92	79	74	85	26
BiLSTM	0.0001	Full	79	21	84	80	74	82	26

Table 7 -BiLSTM Experiment Results at LR=0.001

Model	LR	Batch	Accuracy	Error Rate	Recall	Precision	Specificity	F1 Score	False Alarm
BiLSTM	0.001	1	75	25	73	80	78	76	22
BiLSTM	0.001	32	84	16	89	79	80	84	20
BiLSTM	0.001	64	83	17	84	80	81	82	19
BiLSTM	0.001	128	79	21	83	75	74	79	26
BiLSTM	0.001	256	75	25	83	71	68	76	32
BiLSTM	0.001	Full	76	24	72	78	80	75	20

Table 8 - BiLSTM Experiment Results at LR=0.01

Model	LR	Batch	Accuracy	Error Rate	Recall	Precision	Specificity	F1 Score	False Alarm
BiLSTM	0.01	1	80	20	74	80	85	77	15
BiLSTM	0.01	32	78	22	87	70	71	77	29
BiLSTM	0.01	64	78	22	89	70	68	78	32
BiLSTM	0.01	128	79	21	88	76	68	81	32
BiLSTM	0.01	256	83	17	92	78	72	85	28
BiLSTM	0.01	Full	81	19	85	75	78	79	22

From Tables 6-8 the trends show that the BiLSTM models display a different pattern from the previous CNN model in that the results do not vary widely and follow a consistent path throughout the experimentation process. This can be shown from the initial accuracy metrics. The highest accuracy is 84% from a model with a LR of 0.001 and a batch size of 32, and the lowest accuracy of 75% from a model with the same LR but with a batch size of 1 and 256 respectively. Given that the range between the lowest and highest is 9%, this already shows a clear improvement on the CNN, with the BiLSTM approach showing signs that it's architecture may help in keeping results more stable than the CNN approach.

Recall continues this trend. The highest value is 92% at a LR of 0.0001 and 0.01, with a batch size of 256. The lowest value is 72% at a LR of 0.001 and a full batch size. Although the range is greater than the accuracies range at 20%, it is still lower than the CNN's range of 36%, further proving the point that the BiLSTM is a more stable approach to impaired speech classification than CNNs.

Precision is where the BiLSTM starts to really separate itself from the CNNs. The highest value is 87% at a LR of 0.0001 and a batch size of 32, while the lowest value is 70% at a LR of 0.01 and batch sizes of 32 and 64 respectively. This is a vast improvement on the numbers given from the CNN, with the BiLSTMs highest value greater than the CNNs by 9% and the BiLSTMs lowest value greater than the CNNs by 14%. These differences show how the BiLSTM models ability to correctly classify pathological samples is much better than the CNN, and could prove that the BiLSTM model is the more optimal model to choose for the mobile application, since correctly classifying impaired speech is paramount for this use case.

This trend is continued with the Specificity metrics. The highest specificity value is 85% at a LR of 0.01 and a batch size of 1, while the lowest value is 68% a LR of 0.01 and batch sizes of 64 and 128, as well as at a LR of 0.001 and a batch size of 256. While these values are not as good as precision, with both highest and lowest values being 2% lower than their precision counterparts, these values are still much higher than the specificity metrics in the CNN models. The highest value in the BiLSTM is greater than the CNNs by 3%, while the lowest value is higher in the BiLSTM by 19%. This furthers the point that the BiLSTM model performs better at classifying impaired speech than the CNN, since specificity is a metric targeting the classification of impaired speech rather than classifying healthy speech.

Lastly, the F1 score proves both trends being presented in these findings are correct in that the BiLSTM model performs better than the CNN at a more consistent rate, and that the BiLSTM model can do a better job at classifying the pathological samples than the CNN. The highest F1 score was 85%, coming at a batch size of 256 with LRs of both 0.0001 and 0.01 respectively. The worst value was 75% coming at a LR of 0.001 and a full batch size. With a range of 10% between the worst and best values, the BiLSTM model has a smaller range than the CNN by 9%, meaning that the BiLSTM model has a more consistent F1 score and in general should perform relatively better than the CNN as a result.

To conclude the analysis of the BiLSTM model, it is clear to see that this model performs relatively similar over most runs. It does a very good job at classifying healthy audio while also doing a good job at classifying impaired speech at the same time, thanks to its good recall, precision and specificity scores. The ranges between the values in the various metrics are small, meaning that the BiLSTM performs well consistently over the various combinations of metrics. With these points in mind it would be wise to assume that the BiLSTM model has the statistics to show it is an adequate model to be integrated within the mobile application.

5.1.4 Investigating the influence of Tone on Classification

With the pre-processing measures splitting the samples up into not just healthy and pathological but also into 4 different styles of tone, there was a concern that instead of the models classifying the input by speech pathology, the models would be classifying instead by the tone of voice. To prove whether this was the case experiments were run with changing the inputs to both models, changing the labels from “Healthy/Pathological” to “Healthy/Pathological_Tone”. The 8x8 confusion matrices produced by these experiments are in the below Tables 9 and 10.

Table 9 - CNN Confusion Matrix for Tone Classification

Groundtruth	Healthy_High	11	1	2	2	3	0	0	1
	Healthy_Low	2	5	1	5	0	2	0	0
	Healthy_Mix	4	4	3	1	0	0	1	0
	Healthy_Neutral	3	1	1	3	0	0	1	2
	Pathological_High	1	0	2	2	4	4	2	3
	Pathological_Low	0	2	2	1	2	9	2	1
	Pathological_Mix	1	0	3	0	4	0	3	4
	Pathological_Neutral	0	0	1	2	2	1	1	3
		Healthy_High	Healthy_Low	Healthy_Mix	Healthy_Neutral	Pathological_High	Pathological_Low	Pathological_Mix	Pathological_Neutral
Prediction									

Table 10 - LSTM Confusion Matrix for Tone Classification

Groundtruth	Healthy_High	8	2	1	1	2	0	3	0
	Healthy_Low	2	13	1	0	0	1	0	0
	Healthy_Mix	7	1	5	0	0	0	0	0
	Healthy_Neutral	2	0	0	4	0	0	0	2
	Pathological_High	2	0	3	1	7	3	2	0
	Pathological_Low	1	1	1	2	2	4	6	0
	Pathological_Mix	0	1	2	0	4	3	6	2
	Pathological_Neutral	0	0	0	2	1	1	0	9
		Healthy_High	Healthy_Low	Healthy_Mix	Healthy_Neutral	Pathological_High	Pathological_Low	Pathological_Mix	Pathological_Neutral
Prediction									

From the above Tables 9 and 10, there are some interesting trends that can be dissected. The main difference between both matrices is that the BiLSTM model has a higher accuracy than the CNN model, with correct predictions being more consistent in the BiLSTM model. This shows that the BiLSTM model is a better model to choose if classifying by tone. The main similarity between both these models however is more important than the differences, in that there are very few misclassifications between speech impairment for both models. If the matrices are examined closely most misclassifications come when the impairment is correct, but the tone is incorrect, with a significant amount of pathology misclassification combinations

not occurring and having a value of 0 within the matrices in the majority of cases. This revelation is important in that it confirms that the models can classify between a healthy sample and an impaired sample, while not letting the tone of voice impact the classification.

5.2 Functionality Testing: Saarbruecken Custom Dataset

This testing will mainly focus on the impact that trimming and padding the input during the pre-processing stage has on the outcome of training the models. The hypothesis here is that if the majority of samples under the healthy class or the pathological class have sample lengths that differ, i.e., healthy samples being shorter on average than pathological samples and vice versa, then there could be a case to be made that the models learn the impairment of the samples not through the differences in voice but through the differences in sample lengths. The below Figure 19 shows the rough spread of audio lengths, with the top group being pathological samples and the bottom group being healthy.

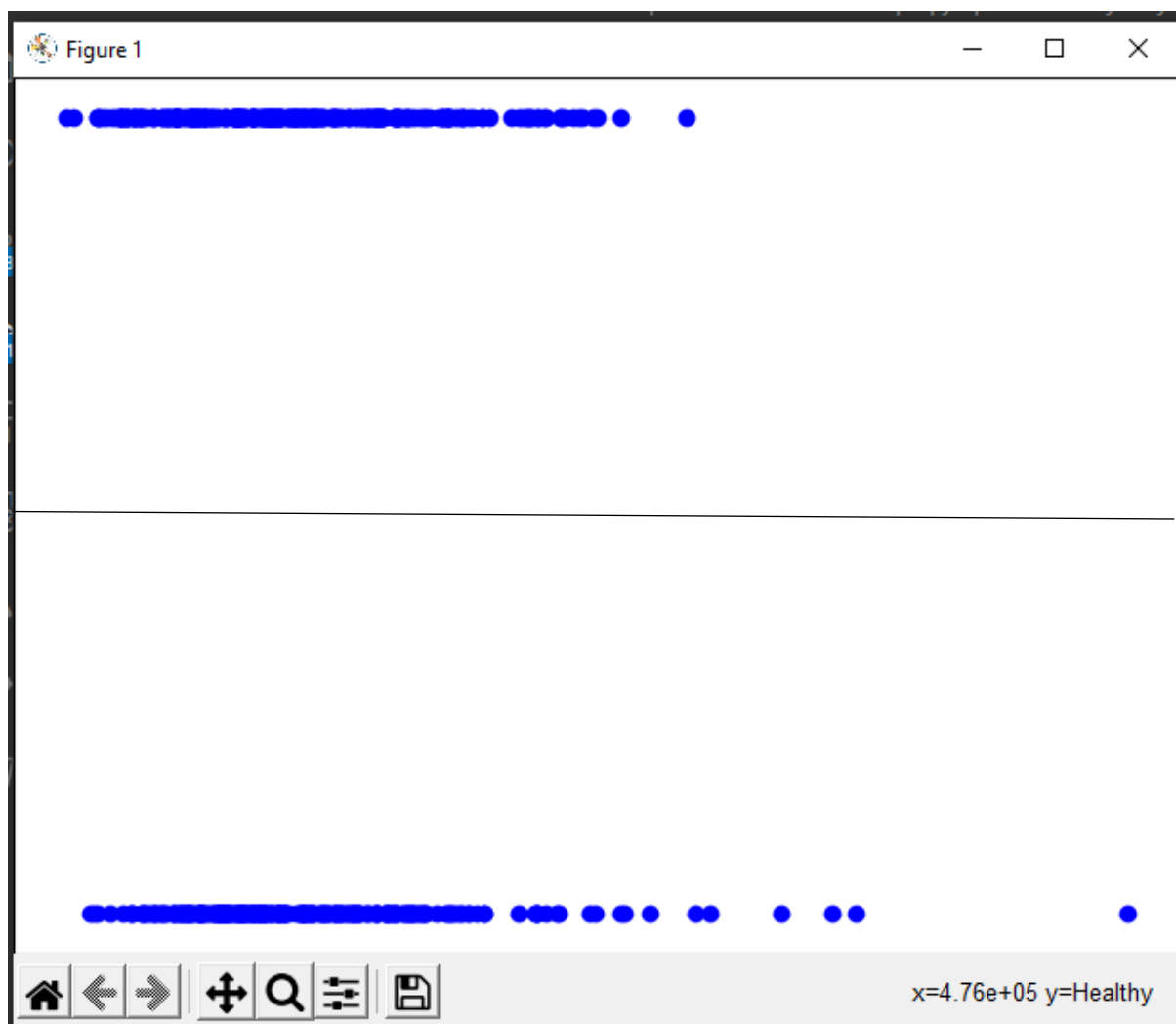


Figure 18 - The rough distribution of audio samples in the dataset

A more in-depth probability density chart showing the distribution of audio samples within the dataset is shown in the below Figure 20.

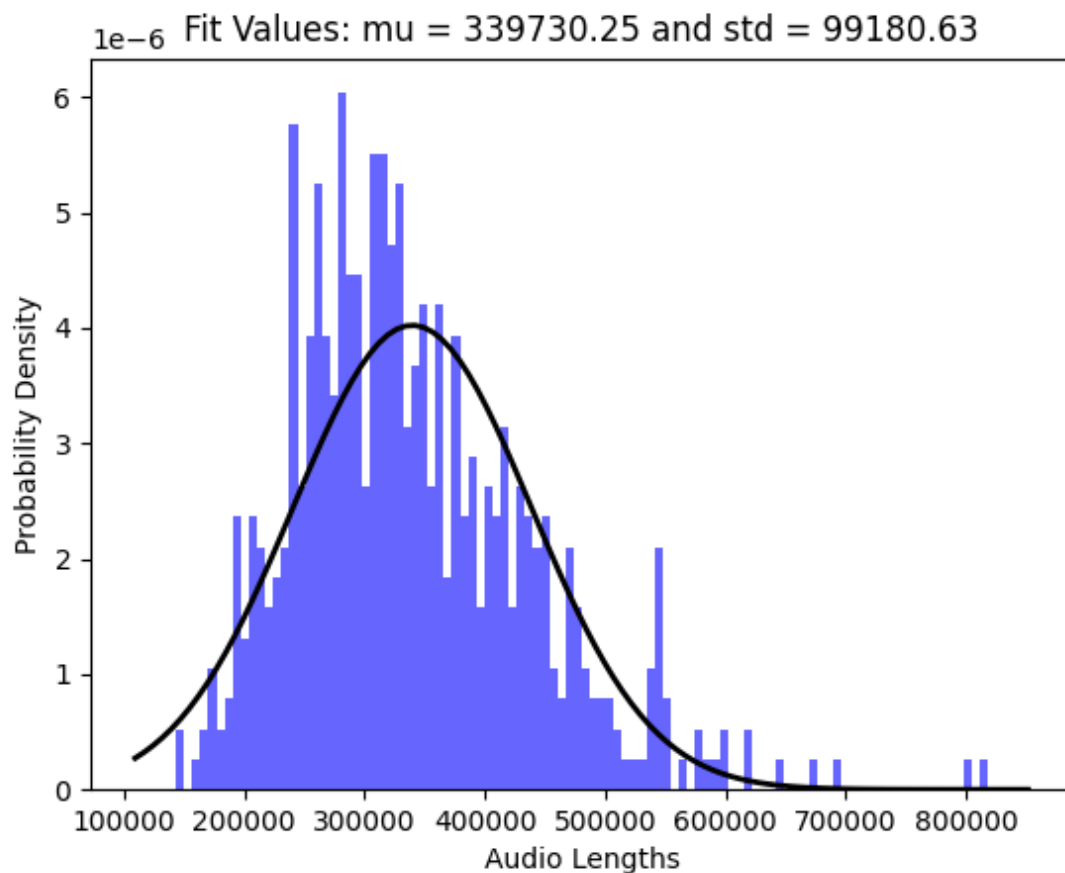


Figure 19 - Distribution Graph of the lengths of audio samples

As you can see from both Figures 19 and 20, although there are roughly 3-4 healthy outliers that are much longer than the rest of the samples, the distribution of the audio is roughly the same, with the distribution graph showing a relatively normal distribution. This shows that the length of the data should not influence the outcome of the model and proving that the dataset is fit for purpose when it comes to machine learning problems.

5.3 Analysis: Reporting on Success of stroke detection

From the results gathered in Section 5.1.3 the best results at each learning rate for each model are displayed in the below Tables 11 and 12.

Table 11 - CNN optimum parameters for each learning rate

Model	LR	Batch	Accuracy	Error Rate	Recall	Precision	Specificity	F1 Score	False Alarm
CNN	0.0001	128	80	20	89	76	72	82	28
CNN	0.001	128	69	31	80	66	58	73	42
CNN	0.01	Full	80	20	97	76	57	85	43
Average Values			76.3	23.6	88.6	72.6	62.3	80	37.6

Table 12 - BiLSTM Optimal Parameters for each Learning Rate

Model	LR	Batch	Accuracy	Error Rate	Recall	Precision	Specificity	F1 Score	False Alarm
BiLSTM	0.0001	256	83	17	92	79	74	85	26
BiLSTM	0.001	32	84	16	89	79	80	84	20
BiLSTM	0.01	256	83	17	92	78	72	85	28
Average Values			83.3	16.6	91	78.6	75.3	84.6	24.6

From the above Tables 11-12 and the larger Tables 3-8 in Section 5.1.3 it is very clear to see how the recurrent nature of the BiLSTM model performs much better than the CNN model, with the average values of all the metrics being more favourable on the BiLSTM model than the CNN, with on average:

- The accuracy and error rate are 7% better on average in the BiLSTM model than the CNN model
- The recall rate 2.4% is better on average in the BiLSTM model than the CNN model
- Precision is 6% better on average in the BiLSTM model than the CNN model
- Specificity and false alarm rate are 13% better on average in the BiLSTM model than the CNN model
- F1 is 4.6% better on average in the BiLSTM model than the CNN model

However, this does not mean that the CNN model is not a good model either. The average accuracy for the CNN is 76.3%, with the highest being 80%, while the average for the BiLSTM being 84% with the average of 83.3%. These results show that the models are competent in the CNN's case and very good in the BiLSTMs case at classifying between healthy and impaired speech. The metric that is the best for both models is the recall, with the CNN scoring as high

as 97% with a LR of 0.01 and a full batch, and the BiLSTM scoring a high of 92 from 2 out of the 3 best runs. This means that both models perform very well in predicting healthy samples as healthy and keeps the number of incorrectly predicted pathological numbers at a minimum.

Worryingly the lowest metric for both models is specificity, with an average of 62.3% for the CNN on average and a 75.3% on average for the BiLSTM. This mean that the models do not perform as well when it comes to classifying pathological samples correctly like it does healthy samples.

The best model out of all these experiments would be the BiLSTM model running at a LR of 0.001 and a batch size of 32. This is because the high accuracy at 84% means it can predict healthy and impaired speech at a high level, the high recall at 89% means it has a low rate of false impaired misclassification, and a good specificity of 80% means that the rate of false healthy misclassification is low as well. This model is the best performer out of the 36 experiments run. The loss chart, accuracy chart, and confusion matrix for this run are in the Figures 19, 20 and 21 below.

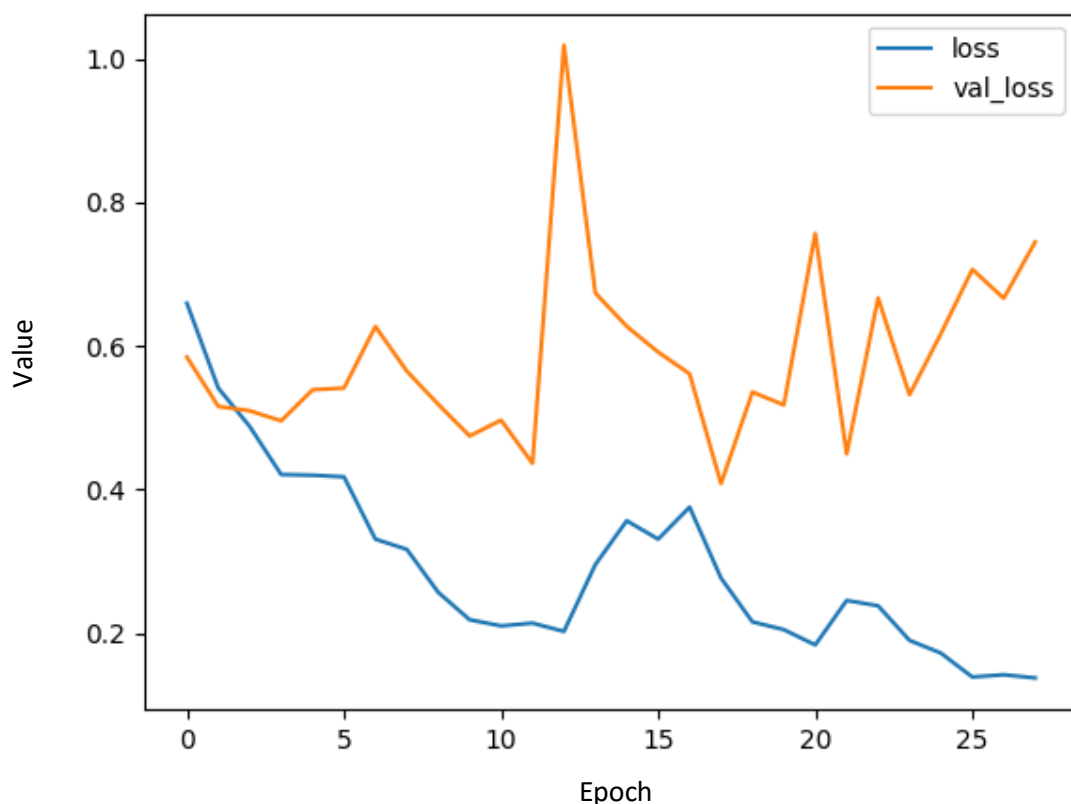


Figure 20 - Loss Graph for BiLSTM model with most optimal parameters

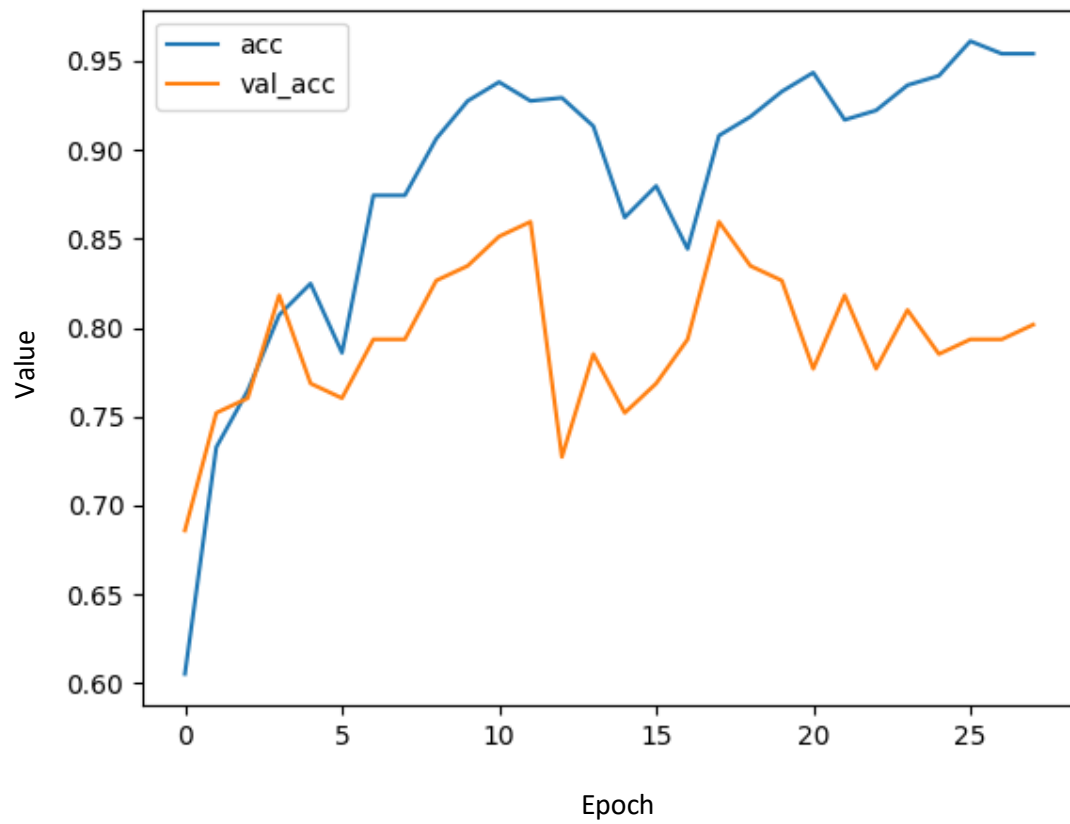


Figure 21 - Accuracy Graph for BiLSTM model with most optimal parameters

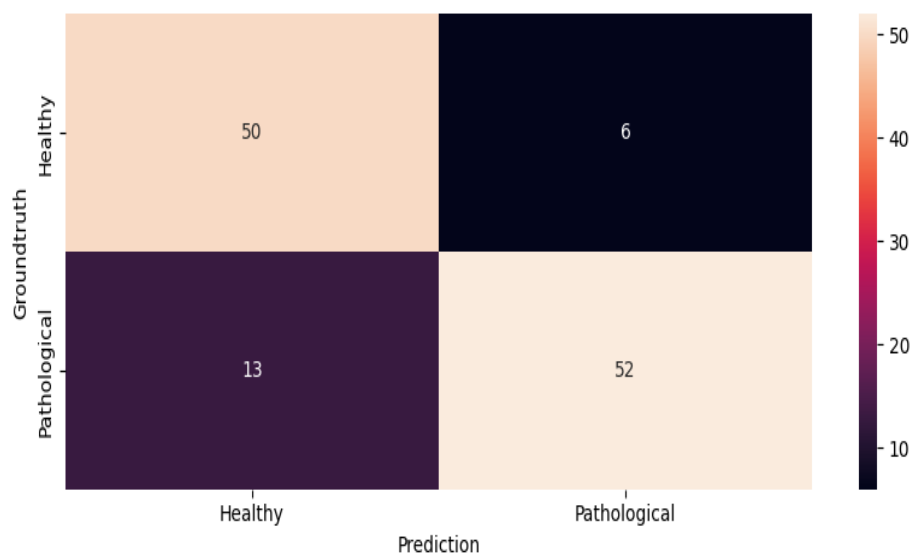


Figure 22 - Confusion Matrix for BiLSTM model with most optimal parameters

5.4 Mobile Application Testing

5.4.1 Front-End Testing

There was very basic unit testing added to the front end using the testing framework known as Jest. Jest is specifically mentioned in the Expo documentation as the unit testing framework to use since it “is the most widely used JavaScript unit testing framework”[49]. The main aims of the tests that were added ensured that the application rendered correctly and that the number of assets in the UI was correct. This is shown in the below Table 13.

Table 13 - Test Suite for the Front-End

Test ID	Expected Behaviour	Pass/Fail
1	Unit test to ensure that the App generates the correct number of Assets (A title, a button and a text asset)	Pass
2	Unit test to ensure that the UI renders correctly (Snapshot test so that UI remains consistent[49])	Pass

5.4.2 Backend Testing

Basic unit testing was undertaken in the backend, so that the predictions script was returning behaviour as predicted. These were written using the unittest framework that has already been built into Python[50]. The tests included making sure that healthy and pathological audio samples were tested against the model and returned expected predictions, as well as passing in a fake audio file to test that the error handling was robust; as depicted in Table 14.

Table 14 - Test Suite for Back-End Functionality

Test ID	Expected Behaviour	Pass/Fail
1	Unit test using a healthy audio sample, prediction script should predict this sample is healthy	Pass
2	Unit test using a pathological audio sample, prediction script should predict this sample is pathological	Pass
3	Unit test using a mock audio file path, method should return an error message saying the file does not exist	Pass

6 System Evaluation and Experimental Results

6.1 Can Machine Learning Classify Impaired Speech?

The answer to this question is yes, machine learning can classify impaired speech. This is shown from the high accuracy scores from models and other metrics like specificity with the BiLSTM model that support this. Throughout the testing of the BiLSTM model the specificity percentages were greater than 80%, meaning that the models were able to correctly pick up on

impaired speech and not classify them as false healthy samples. While these numbers are not as high as recall, it is still a high enough percentage to say that the model can pick up the differences between healthy and impaired speech.

Likewise, the testing undertaken on the dataset through the tone classification and the impact padding and trimming has on the dataset proves that the model can classify based on health of the patient and not on any blemishes within the dataset, furthering the point that machine learning can classify impaired speech.

6.2 Implications for Detecting Stroke through Speech Classification?

The implications for detecting stroke via speech classification are quite extensive. This research can be applied to methods for detecting stroke, such as a stroke detection app that is able to record sound from a potentially pathological sample, be able to predict against a trained model that is able to differentiate healthy voices from voices that have stroke and return the prediction to the patient. Depending on the outcome of the prediction the patient can make the decision on whether the person needs medical attention. This should reduce the delay from when a patient has a stroke and when they get medical treatment as mentioned in [1].

As well as this, an application that can identify stroke in a patient may also teach people that are not familiar with symptoms of stroke that speech is one of the main symptoms to look out for, and by running the application on various stroke sufferers then different nuances in their impaired speech may stand out more to the public and might encourage people to be more aware when someone is struggling to talk the signs and what to look out for.

6.3 Success of Project and Further Work

Overall, the project was a success on the machine learning aspect of the project. The main objective of the project was achieved in having the ability to classify impaired speech, with some very promising metrics to go along with it. However, even the best combination of parameters had a problem of overfitting, even with extra parameters added to combat this issue, as seen in Figure 20. That being said, further work could delve deeper into the kinds of models being used to classify, maybe layering multiple types of Neural Network such as LSTM into CNN or vice versa, or even a new technology altogether such as the Transformer concept. Further work could also be done potentially with improving the dataset, potentially reaching out for volunteers who have or have had stroke to analyse their speaking patterns and record some samples to be used. This could be a better approach than trimming and editing various

files like had been done in the pre-processing section and could potentially be more relevant since they are targeting stroke patients specifically.

With the project, there were many challenges faced. The researcher had only recently been introduced to Python and was still a novice on its use, therefore the tutorials made handling difficulties within the project a lot easier. Similarly the researcher had very limited experience with JavaScript and no experience using Expo, therefore the tutorials and walkthroughs on sites such as stackoverflow.com [51] were relied on heavily to help with the development of the application.

However, the most significant weakness of the project is the application. At the beginning this project was meant to integrate with a software engineering project which aimed to identify stroke via arm weakness using cameras on a mobile application. When trying to setup this application problems arose in running Expo application and Flask servers, with errors crashing the app and the server most times on startup. In conversations with other researchers using the same application on a different project it was understood that the application may be broken in many aspects as, in order to reflect the latest updates to python and JavaScript, the coding had to be refactored and updated regularly. Therefore, to overcome this, an application was developed from scratch by the researcher to be used in the place of the existing application, despite the existing application being the preferred solution of choice.

The next problem with the application was not the programming of the application but the model being compared against it. As mentioned a few paragraphs ago, there was evidence that even with the most optimal parameter combination, the models still seemed to overfit. This then resulted in a model that, when used in the real-world i.e. in this application, it would result in all samples being sent to the server returning that the subject has stroke and to seek medical attention. Multiple test subjects were used during the testing process and despite this each subject returned a stroke diagnosis, even though the testing as outlined in Section 5.4.2 would suggest that with tests passing this should not be the case. However, the test audio clips used are from the same dataset mentioned in Section 4.3, so the model should be able to tell the difference between audio samples it has already seen. This kind of performance is not acceptable, since this undermines the promising results shown in Sections 5.1.3 and 5.3. To help overcome this issue in any further research a different approach to classifying impaired audio could be used, for example the Transformer concept.

To conclude, the project was a success in answering the main question of the project. ML can classify impaired speech from healthy speech, and Tables 3-8 and 11-12 prove this. However, future research should ensure better handling of the software components to gain a better insight into the project itself and to allow for more consistent results.

References

Git repository: <https://gitlab2.eecs.qub.ac.uk/40227531/pauricdonnellyfinalyearproject>

- [1] D. K. Moser *et al.*, ‘Reducing delay in seeking treatment by patients with acute coronary syndrome and stroke: a scientific statement from the American Heart Association Council on cardiovascular nursing and stroke council’, *Circulation*, vol. 114, no. 2, pp. 168–182, Jul. 2006, doi: 10.1161/CIRCULATIONAHA.106.176040.
- [2] J. Zerwic, S. Young Hwang, and L. Tucco, ‘Interpretation of symptoms and delay in seeking treatment by patients who have had a stroke: Exploratory study’, *Heart & Lung*, vol. 36, no. 1, pp. 25–34, Jan. 2007, doi: 10.1016/j.hrtlng.2005.12.007.
- [3] C. R. Lacy, D.-C. Suh, M. Bueno, and J. B. Kostis, ‘Delay in Presentation and Evaluation for Acute Stroke’, *Stroke*, vol. 32, no. 1, pp. 63–69, Jan. 2001, doi: 10.1161/01.STR.32.1.63.
- [4] K. J. Greenlund *et al.*, ‘Low public recognition of major stroke symptoms’, *American Journal of Preventive Medicine*, vol. 25, no. 4, pp. 315–319, Nov. 2003, doi: 10.1016/S0749-3797(03)00206-X.
- [5] ‘Stroke - Symptoms’, *nhs.uk*, Oct. 24, 2017.
<https://www.nhs.uk/conditions/stroke/symptoms/> (accessed Jan. 11, 2022).
- [6] M. S. Sirsat, E. Fermé, and J. Câmara, ‘Machine Learning for Brain Stroke: A Review’, *Journal of Stroke and Cerebrovascular Diseases*, vol. 29, no. 10, p. 105162, Oct. 2020, doi: 10.1016/j.jstrokecerebrovasdis.2020.105162.
- [7] B. J. Lee, K. H. Kim, B. Ku, J.-S. Jang, and J. Y. Kim, ‘Prediction of body mass index status from voice signals based on machine learning for automated medical applications’, *Artificial Intelligence in Medicine*, vol. 58, no. 1, pp. 51–61, May 2013, doi: 10.1016/j.artmed.2013.02.001.
- [8] Y. Gu, X. Li, S. Chen, J. Zhang, and I. Marsic, ‘Speech Intention Classification with Multimodal Deep Learning’, in *Advances in Artificial Intelligence*, Cham, 2017, pp. 260–271. doi: 10.1007/978-3-319-57351-9_30.
- [9] D. M. Ballesteros, Y. Rodriguez-Ortega, D. Renza, and G. Arce, ‘Deep4SNet: deep learning for fake speech classification’, *Expert Systems with Applications*, vol. 184, p. 115465, Dec. 2021, doi: 10.1016/j.eswa.2021.115465.
- [10] S. H. Bae, I. Choi, and N. S. Kim, ‘ACOUSTIC SCENE CLASSIFICATION USING PARALLEL COMBINATION OF LSTM AND CNN’, p. 5, 2016.
- [11] M. Scarpiniti, D. Communiello, A. Uncini, and Y.-C. Lee, ‘Deep Recurrent Neural Networks for Audio Classification in Construction Sites’, in *2020 28th European Signal Processing Conference (EUSIPCO)*, Jan. 2021, pp. 810–814. doi: 10.23919/Eusipco47968.2020.9287802.
- [12] ‘Speech Classification — NVIDIA NeMo 1.7.2 documentation’.
https://docs.nvidia.com/deeplearning/nemo/user-guide/docs/en/stable/asr/speech_classification/intro.html (accessed Apr. 11, 2022).
- [13] ‘Hey Siri: An On-device DNN-powered Voice Trigger for Apple’s Personal Assistant’, *Apple Machine Learning Research*.
<https://machinelearning.apple.com/research/hey-siri> (accessed Apr. 11, 2022).
- [14] ‘Real-time Continuous Transcription with Live Transcribe’, *Google AI Blog*.
<http://ai.googleblog.com/2019/02/real-time-continuous-transcription-with.html> (accessed Apr. 12, 2022).
- [15] J. F. Gemmeke *et al.*, ‘Audio Set: An ontology and human-labeled dataset for audio events’, New Orleans, LA, 2017.

- [16] ‘Dysarthria (difficulty speaking)’, *nhs.uk*, Oct. 18, 2017.
<https://www.nhs.uk/conditions/dysarthria/> (accessed Apr. 14, 2022).
- [17] ‘Aphasia’, *nhs.uk*, Oct. 20, 2017. <https://www.nhs.uk/conditions/aphasia/> (accessed Apr. 14, 2022).
- [18] M. B. Maas *et al.*, ‘The Prognosis for Aphasia in Stroke’, *Journal of Stroke and Cerebrovascular Diseases*, vol. 21, no. 5, pp. 350–357, Jul. 2012, doi: 10.1016/j.jstrokecerebrovasdis.2010.09.009.
- [19] M. Połczyńska and Y. Tobin, ‘Comparing and contrasting phonological processes in adult dysarthria and first language acquisition’, 2011, pp. 245–265.
- [20] Y. Li, W. Shi, G. Liu, L. Jiao, Z. Ma, and L. Wei, ‘SAR Image Object Detection Based on Improved Cross-Entropy Loss Function with the Attention of Hard Samples’, in *2021 IEEE International Geoscience and Remote Sensing Symposium IGARSS*, Jul. 2021, pp. 4771–4774. doi: 10.1109/IGARSS47720.2021.9554061.
- [21] R. Khan, ‘Why using Mean Squared Error(MSE) cost function for Binary Classification is a bad idea?’, *Medium*, Dec. 08, 2020.
<https://towardsdatascience.com/why-using-mean-squared-error-mse-cost-function-for-binary-classification-is-a-bad-idea-933089e90df7> (accessed Apr. 28, 2022).
- [22] M. Shanker, M. Y. Hu, and M. S. Hung, ‘Effect of data standardization on neural network training’, *Omega*, vol. 24, no. 4, pp. 385–397, Aug. 1996, doi: 10.1016/0305-0483(96)00010-2.
- [23] ‘The Short-Time Fourier Transform’.
https://ccrma.stanford.edu/~jos/sasp/Short_Time_Fourier_Transform.html (accessed Jan. 13, 2022).
- [24] Y. Zheng, B. K. Iwana, and S. Uchida, ‘Discovering Class-Wise Trends of Max-Pooling in Subspace’, in *2018 16th International Conference on Frontiers in Handwriting Recognition (ICFHR)*, Aug. 2018, pp. 98–103. doi: 10.1109/ICFHR-2018.2018.00026.
- [25] J. Wang and L. Liu, ‘A Neural Network Sparseness Algorithm Based on Relevance Dropout’, in *2019 IEEE 6th International Conference on Industrial Engineering and Applications (ICIEA)*, Apr. 2019, pp. 480–484. doi: 10.1109/IEA.2019.8715060.
- [26] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, ‘Improving neural networks by preventing co-adaptation of feature detectors’, *arXiv:1207.0580 [cs]*, Jul. 2012, Accessed: Apr. 20, 2022. [Online]. Available: <http://arxiv.org/abs/1207.0580>
- [27] R. DiPietro and G. D. Hager, ‘Chapter 21 - Deep learning: RNNs and LSTM’, in *Handbook of Medical Image Computing and Computer Assisted Intervention*, S. K. Zhou, D. Rueckert, and G. Fichtinger, Eds. Academic Press, 2020, pp. 503–519. doi: 10.1016/B978-0-12-816176-0.00026-0.
- [28] T. Saji, ‘Unsupervised Sentiment Analysis using small recurrent language models’, Dec. 2019. doi: 10.13140/RG.2.2.35638.52806.
- [29] G. Liu and J. Guo, ‘Bidirectional LSTM with attention mechanism and convolutional layer for text classification’, *Neurocomputing*, vol. 337, pp. 325–338, Apr. 2019, doi: 10.1016/j.neucom.2019.01.078.
- [30] P. Chandrasekar and K. Qian, ‘The Impact of Data Preprocessing on the Performance of a Naive Bayes Classifier’, in *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*, Jun. 2016, vol. 2, pp. 618–619. doi: 10.1109/COMPSAC.2016.205.
- [31] ‘Smartphone users 2026’, *Statista*.
<https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/> (accessed Apr. 29, 2022).

- [32] ‘World Population Prospects - Population Division - United Nations’. <https://population.un.org/wpp/DataQuery/> (accessed Apr. 29, 2022).
- [33] ‘What is an API endpoint?’, *Cloudflare*. <https://www.cloudflare.com/learning/security/api/what-is-api-endpoint/> (accessed Apr. 29, 2022).
- [34] ‘JavaScript | MDN’. <https://developer.mozilla.org/en-US/docs/Web/JavaScript> (accessed Apr. 29, 2022).
- [35] ‘Node.js - MDN Web Docs Glossary: Definitions of Web-related terms | MDN’. <https://developer.mozilla.org/en-US/docs/Glossary/Node.js> (accessed Apr. 29, 2022).
- [36] ‘What is Expo and why it matters for app development’, *Moze*, Feb. 21, 2022. <https://www.mozestudio.com/journal/what-is-expo-and-why-it-matters-for-app-development/> (accessed Apr. 29, 2022).
- [37] ‘FormData - Web APIs | MDN’. <https://developer.mozilla.org/en-US/docs/Web/API/FormData> (accessed Apr. 29, 2022).
- [38] ‘fetch() - Web APIs | MDN’. <https://developer.mozilla.org/en-US/docs/Web/API/FormData> (accessed Apr. 29, 2022).
- [39] ‘Welcome to Flask — Flask Documentation (2.1.x)’. <https://flask.palletsprojects.com/en/2.1.x/> (accessed Apr. 29, 2022).
- [40] ‘Flask’, *Pallets*. <https://palletsprojects.com/p/flask/> (accessed Apr. 29, 2022).
- [41] ‘pydub/API.markdown at master · jiaaro/pydub’, *GitHub*. <https://github.com/jiaaro/pydub> (accessed Apr. 29, 2022).
- [42] B. Woldert-Jokisz <bogwol@coli.uni-sb.de>, ‘Saarbruecken Voice Database’, <http://stimmdatenbank.coli.uni-saarland.de>, May 23, 2007. http://www.stimmdatenbank.coli.uni-saarland.de/help_en.php4 (accessed Jan. 12, 2022).
- [43] C. Shorten and T. M. Khoshgoftaar, ‘A survey on Image Data Augmentation for Deep Learning’, *Journal of Big Data*, vol. 6, no. 1, p. 60, Jul. 2019, doi: 10.1186/s40537-019-0197-0.
- [44] ‘About’, *Audacity* ®. <https://www.audacityteam.org/about/> (accessed Jan. 12, 2022).
- [45] ‘Git Large File Storage’, *Git Large File Storage*. <https://git-lfs.github.com/> (accessed Jan. 12, 2022).
- [46] ‘UrbanSound8K’, *Urban Sound Datasets*. <https://urbansounddataset.weebly.com/urbansound8k.html> (accessed Jan. 13, 2022).
- [47] T. Y. Chen, F.-C. Kuo, and R. Merkel, ‘On the statistical properties of the F-measure’, in *Fourth International Conference on Quality Software, 2004. QSIC 2004. Proceedings.*, Sep. 2004, pp. 146–153. doi: 10.1109/QSIC.2004.1357955.
- [48] S. L. Smith, P.-J. Kindermans, C. Ying, and Q. V. Le, ‘Don’t Decay the Learning Rate, Increase the Batch Size’, *arXiv:1711.00489 [cs, stat]*, Feb. 2018, Accessed: Jan. 13, 2022. [Online]. Available: <http://arxiv.org/abs/1711.00489>
- [49] ‘Testing with Jest’, *Expo Documentation*. <https://docs.expo.dev/guides/testing-with-jest> (accessed Apr. 29, 2022).
- [50] ‘unittest — Unit testing framework — Python 3.10.4 documentation’. <https://docs.python.org/3/library/unittest.html> (accessed Apr. 29, 2022).
- [51] ‘Stack Overflow - Where Developers Learn, Share, & Build Careers’, *Stack Overflow*. <https://stackoverflow.com/> (accessed Apr. 29, 2022).