University Of Galway

CT413 Final Year Project BSc (CSIT)

App For Anomic Aphasia

Thomas Meehan – 20425936

GitHub Repository Link - https://github.com/tomstuden/FinalYearProjectAppForAnomicAphasia.git

Supervised by

Dr Conor Hayes

School Of Computer Science

College Of Science and Engineering

Table of Contents

Introduction

This project aims to create an application to help address the challenges posed by anomic aphasia, a language disorder involving difficulty with word retrieval, In this report we will go through the objectives of the project, my initial background research, how I implemented my deep learning neural network, my front and backend implementation, my development processes, application evaluation and potential improvements that could be implemented in future. This report aims to offer insights into how this application was developed and the functionality it offers users.

1.1 - Overview Of Anomic Aphasia.

Anomic Aphasia (Andreetta, S., Cantagallo, A. and Marini, A., 2012) is a type of language disorder that can lead to people having difficulty in finding appropriate names for objects. Individuals with Anomic Aphasia can have perfect and intact grammar, comprehension and fluency but struggle to retrieve words, these individuals then must resort to vague descriptions to convey their intended meaning. This difficulty in word finding can significantly impact conversations leading to frustration from both the affected individual and others involved in the conversation.

Anomic Aphasia can result from a variety of causes and is usually a result of damage to the left hemisphere of the brain, where the language and speech is controlled, it can be caused by issues such as brain injury, strokes or neurological disorders and its severity can range based on several different factors.

1.2 – Project Overview

The initial scope for the project was defined by my Final Year Project supervisor Dr. Conor Hayes.

The scope involved assisting people with Anomic Aphasia by designing an application to recognise drawings made and return the word along with a visual image of the word and an audio clip of its pronunciations.

The scope evolved over the course of the project and the decision was made to develop an application along with a user interface, to allow users to draw their object and a word, alongside an image and audio clip of the word would be returned to the user. This was a large scope that allowed for numerous potential approaches to the problem and allowed space for me to find a solution that I felt was best suited both the problem and my skill set.

1.3 – Overview Of Project Objectives

This application will have 3 core components, the convolutional neural network (CNN), the Flask backend, and my frontend and user interface. The CNN will be developed in python and will be trained using the Google QuickDraw! Dataset, this will allow me to make accurate predictions on drawings made by the user. The Flask backend will allow me to communicate between the User Interface (UI) and the CNN processing and sending the relevant data between the two. Finally, the User Interface will allow the user to draw their object and submit this to the model to retrieve a predicted label.

1.4 – Core Objectives

The Core Objectives of this project aim to work together to provide the core functionality of the app, The core functionality or purpose of this application is to provide people with the condition of Anomic Aphasia a tool to aid them in their day to day lives.

In order to achieve this goal, there are a number of objectives that are essential to the project and its primary functions.

Providing a Clear and Accurate way for users to draw objects.

Initially a way is needed to provide users with a clear and readable User Interface to input their drawings. This allows the user to represent the item for which they wish to retrieve the word in their own way and should be simple, easy to use and represent these drawings accurately within it to allow these users to accurately capture and draw what they intended to.

Fetching Accurate Predictions on User Drawings

Another core objective will be providing accurate predictions on these drawings. This will involve using a machine learning approach to make accurate predictions on drawings provided by the user. This will involve developing a machine learning model capable of interpreting and analysing user inputs and producing accurate predictions on this data.

Developing a method of communication between User Interface and Machine Learning Model

A method of robustly communicating between the User Interface and the Machine Learning Model will need to be established, it is essential that this communication method efficiently and accurately represents both user inputs and returned predictions to the relevant components. A subpar communication system will result in inaccurate predictions being displayed and compromises the objectives of the project as a whole.

Display of Visual and Audio representation of Predictions to the User

Establishing a method of providing users with both Audio and Visual representations of the predictions will allow the results to be easier understood and interpreted by the user. Ensuring there is multiple ways of communicating the prediction to the user ensures that the user has a greater chance of using and understanding the applications results.

1.5 – Additional Objectives

Cross Platform Application

Initially being developed as a web application, ideally this would be transferred and deployed also as a mobile application. Allowing users of this application ease of access to it in real world situations where a web application may not be as useful.

Hosting backend Remotely

Hosting the backend remotely will allow seamless updates to the model to take place, allowing the model to be improved and scaled without experiencing down time and disrupting the applications availability.

1.6 – Report Overview

Introduction

This section of the report focuses on introducing the problem and provides an overview of Anomic Aphasia and introduces the project. It also states the objectives of the project and outlines the report overview, deadlines, constraints, and the deliverables of the project.

Background Research

This section involves researching existing techniques used to aid Anomic Aphasia, it also focuses on exploring different machine learning image classification techniques alongside using the Google QuickDraw! Dataset and exploring other individuals' approach to working with this dataset. This section also involves researching different methods of returning both visual and audio representations of the label and looking at different methods of user interface design and mobile capabilities. This section concludes with a look at the benefits of hosting the backend locally or remotely.

Implementation of Deep Learning Model

This section focuses on the technologies used when developing and training the deep learning model and involves various aspects such as the tools used, dataset

preprocessing, model construction and saving and recalling the model within the backend.

Implementation of Backend and Frontend

In this section we will look at the technologies used in the front and backend aspects of the project, including the communication between both aspects, the preprocessing done, Audio and Visual retrieval and the functionality of both the user interface and the front end, and the back-end component of the project. We will also discuss some of the challenges faced during implementing different aspects of the project.

Development Processes

This section of the report will cover the version control used throughout this project and will give an outline of both the development and testing procedures used in the project.

Application Evaluation

Within this section we will evaluate the performance of the application as a whole, focusing on the performance of the model and of both IBMs Watson Text-To-Speech Api and SERPs Google image retrieval API, both of which were used for retrieving visual and audio representations of the prediction from the model.

Potential Improvements and Next Steps

This area of the report will cover potential improvements that could be made to the application including improving the deep learning model, improving the APIs, compatibility with other platforms and implementing features such as user profiles and history within the application.

Conclusion

This part of the report summarizes the project and its findings and outlines my own personal view on the project.

Timelines

This section will contain process flow diagrams for the User Interface and Frontend, Backend, and the Deep Learning Model.

1.7 – Project Deadlines and Tasks

This part of the project discusses the deadlines faced for the project and its tasks.

- Project Options Posted – 12-September-2023
- Project and Supervisors Assigned – 25th – September-2023
- Initial Weekly Meeting and Discussion with Supervisor – 27th-September - 2023

- Research and Gaining Familiarity with the dataset – 1st November 2023– 20th November 2023
- Model Development Started – 22nd November-2023
- Project Definition Document Due – 27th – November- 2023(Extended 11th December)
- Initial UI Development Started – 22nd December 2023
- Model Development Completed – 28th December 2023
- Initial UI Development Completed – 25th Jan 2024
- Backend Started – 2nd February 2024
- Updated UI Developed – 15th February 2024
- Backend Completed – 10th March 2024
- Testing and Minor Improvements – 10th March 2024 – 28th March 2024
- Final Project Report Completed – 28th March 2024
- Final Project Demo – 2nd April 2024
- Final Project Viva Voce – 4th April 2024
- Mobile migration and Server migration – 20th March 2024 – 28th March 2024

1.8 – Constraints

- Time Constraints due to demands of other modules.
- Time Constraints due to submission date
- API Limitations, with SERP API having limited submissions per month.
- Reliance on relatively clear and well-drawn user image data for model to accurately predict label.

1.9 – Deliverables

- Project Definition Document – 27th November 2023, Pushed back to 11th December 2023
- Final Project Report – 28th March 2024
- Project Demonstration and Viva Voce – 2nd and 4th April 2024

1.10 – Declaration

I hereby declare that this final year project, which I now submit for correction, is entirely my own work
and has not been influenced by the work of others except where cited.

Thomas Meehan – 28/03/2024.

Background Research

2.1 – Current Anomic Aphasia Techniques and Aids

Most current Anomic Aphasia aids focus on speech therapy approaches aiming to help aid the individuals word retrieval and confidence over time. While this project was assigned to me by my supervisor, I conducted additional research to gain insights into existing technologies in this space.

It was while researching these approaches I encountered several technological aids for these individuals, such as Augmented Communication Applications such as CommunicAide (www.aphasiasoftwarefinder.org. n.d.) and iName it (Smartyearsapps. n.d.)., that allow individuals to communicate using visual aids when verbal communication is difficult. There are also several mobile applications that support individuals by offering picture difficulties, text to speech functions and synthesised speech functions. However, many of these apps also focus on therapeutic approaches with very few offering features that would support individuals in real world scenarios. This contrasted with my application which aimed to assist individuals in these real-world scenarios, providing them a way to retrieve words quickly and accurately.

It was after researching these current techniques I was able to identify there were very few efficient existing real-world approaches and applications that dealt with this issue in a quick and inobtrusive way.

2.2 – Machine Learning Classification with Image Recognition

I initially conducted research into the various ways machine learning algorithms could be used to deal with image classification.

Following this I was able to see that a deep learning approach focusing on using a Convoluted Neural Network (CNN) would allow me to accurately detect and predict objects that the user had drawn. When examining these I gained an insight into how

CNNs could be used to detect objects and identify patterns between its training data and unseen test images.

Once I identified this approach, I was able to conduct further research and examine existing CNN approaches (Saha, S. 2018). to image detection problems which greatly aided me in constructing my own CNN. I was also able to view different ways of evaluating and training my CNN to produce optimal results, avoiding both over and under fitting.

2.3 – Google QuickDraw! Dataset Research

When assigned this project, I was given the Google QuickDraw! (Quick, Draw! The Data 2019). Dataset by my supervisor.

This dataset contains over 50 million user generated drawings represented by their x and y points scaled to 256x256. These 50 million drawings are spread across 345 different categories providing a significant amount of training data for my model.

I was also able to use the Google QuickDraw! Web Application to investigate the relationship between how the drawings were constructed and how they were represented within the dataset. This allowed me to identify correlations between how the data was captured and then represented and provided me with not only a great insight into the dataset but also an insight into how my application could capture user input and represent this to the CNN (Guo, K., WoMa, J. and Xu, E., 2018).

2.4 – Kaggle Research – Exploring approaches to this problem

When conducting my research I encountered “Kaggle” (Kaggle, 2022). which is an online community platform for data scientists and machine learning enthusiasts.

Kaggle Logo

From here I was able to view user submissions for a Google QuickDraw! Competition which allowed individuals to create their own machine learning algorithms to correctly identify objects from this dataset. Using this I was able to research additional ways to create my Deep Learning Model and was also able to investigate how CNN approaches were used to tackle this problem.

This allowed me to gain an insight into how created models could interact with the dataset and provided me with an in depth look at solutions which provided me with new ideas and approaches to creating the Deep Learning Model.

In particular Nvantu's (kaggle.com. n.d.). approach allowed me to understand how to visualise and understand the data alongside offering great insight into how I could approach creating my own CNN to accomplish the model aspect of this project.

2.5 – API Research – Image and Audio APIs

I then moved onto researching different ways I could integrate visual and audio representations of my results into the application.

This involved researching various APIs that were designed for specific purposes. In terms of Image Search APIs there was a wide range available, such as the Contentful Images API. However, I focused on the "SERP – Google Images Results API" (Serpapi. n.d), which satisfied all requirements. This allowed me to scrape images from a defined google images search results page, it then returned the URL.

In terms of Audio, I looked in depth at Text-To-Speech APIs, after initially researching the Google TTS API (Google Cloud. n.d.)., I decided to use IBMs Watson TTS API (www.ibm.com. n.d.)., due to its more straightforward documentation and its impressive results, offering a variety of languages and accents within the API.

2.6 – User Interface Research

When researching User Interface approaches, I looked at some of my previous projects and ways to incorporate useful elements from these into this current project.

A primary element of focus within the User Interface was the drawing element as when researching I learned there were multiple ways to develop a drawing element largely depended on the framework used to build the application. I initially focused on using WPF from the .NET (Microsoft. n.d.). framework to construct an initial web application and the basic functionality, this was then converted to a Flutter Application which provided both web and mobile support.

Due to being unfamiliar with creating mobile applications, I spent significant time researching flutter and how to create applications within it.

2.7 – Mobile Capability Research

I conducted significant research on mobile compatible frameworks and how the application could be deployed as a mobile application, this involved researching compatible mobile emulators, such as Android Studio (Android Studio 2019). and Genymotion (Genymobile. n.d). and resulted in me migrating the app development to Flutter, which supports web, Android, and iOS app development.

2.8 – Exploring benefits of hosting backend locally or remotely

When researching the benefits of hosting the application locally or remotely on a server there were pros and cons for each approach, Hosting locally would provide me with the

ability to use the application offline however it would mean there was significantly less room for scalability.

Whereas hosting on a server would allow me to remotely update the backend in a much more seamless manner, improving the model and the other backend components without the need for constant updates and downtime, however there can be data privacy and cost concerns.

I decided to host the backend on a server due to the demands of the application and recognising that updating the model regularly would lead to increased accuracy with its predictions and it was therefor preferable to locally hosting.

Implementation of Deep Learning Model

3.1 – Technologies Used

To create this Convolutional Neural Network, I used the TensorFlow (TensorFlow 2019b). library, which offers a comprehensive array of libraries, tools, and resources for developing deep learning machine learning models within python. I also heavily relied on the Keras (TensorFlow 2019a). library within TensorFlow to construct my CNN using components from within the Keras and TensorFlow libraries, these included using the Sequential component from keras.models, I also imported Dense, MaxPooling2D and Conv2D layers from the keras.layers library. In terms of evaluating the model I imported metrics components such as top_k_categorical_accuracy and was able to visualise and plot results using matplotlib.

These CNNs were constructed using python due to its various supported libraries and extensive online documentation in this area, allowing me to easier learn and understand different approaches and problems that I encountered.

My models were constructed across both Jupyter Notebook and Google Colab environments. Their block-based approach allowed me to easier read and understand how different components of the python programs were interacting with each other.

3.2 - Data Preprocessing and Understanding

A significant amount of data understanding was required for this project. This was due to the formatting of the dataset being complex upon initial viewing.

The Google QuickDraw! Dataset contains over 50 million user drawings across 345 different categories. This data is distributed across 345 .csv files according to their category. The data is represented as a List of lines with each line containing separate

lists for the x and y float values of its points. These points were scaled within the values of 256x256 and due to the user generated nature of these drawings, the dataset also contained a column with true or false values based on whether the drawing was identified by Googles QuickDraw! Classification Algorithm. This helped to identify poorly drawn drawings and ensure the model was drawn using recognised or well-drawn drawings, identifying patterns within the drawings based on looking at the data directly was quite difficult.

Due to this the decision was made to convert these drawings to 64*64 images which would be fed to the model.

3.3 – Image Generation from Dataset

Due to the nature of the dataset the most reasonable approach seemed to be to convert these drawing points to images before feeding them into the model. This would allow me to view images and use these when training the model.

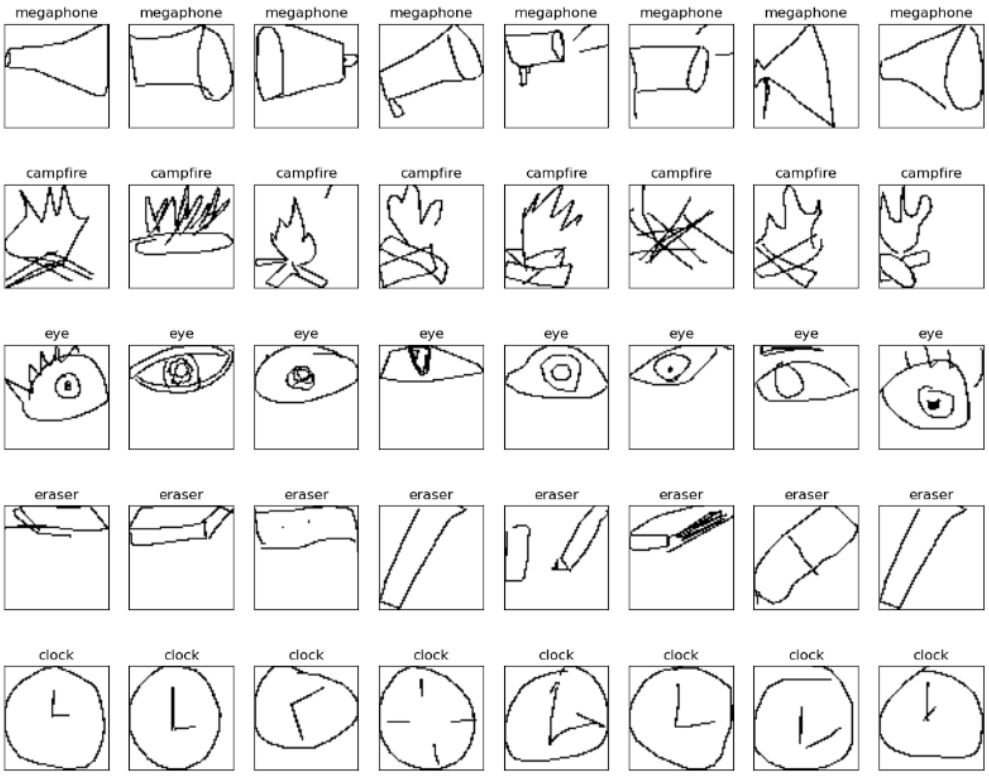


Image Generation Results

This involved pulling 40 random recognised images from 5 random categories and displaying them within a plot to visualise how the drawings are represented within the dataset. This also offers an insight into the data that will be used when training my CNN and allows me to understand the type of inputs the model expects when passing user generated data to the model.

3.4 – Model Creation

As mentioned before this initial model was created in Jupyter Notebook using python with TensorFlow and Keras libraries to allow me to build it.

I created a model that took a single grayscale image of height 64 and width 64 as an initial input.

This then fed into an initial convolutional layer (Conv2D) which has 32 kernels of size (3,3). This layer uses the ReLu activation function to introduce non-linearity to the model. Its padding is set to “same” ensuring that the dimensions of the output feature mirror those of the inputs. This layer applies kernels to the input image that extract features across different points of the image by detecting patterns such as edges and shapes. These layers allowed my model to extract meaningful features from the input images.

After each convolutional layer was added a max pooling layer (MaxPooling2D) layer was added with a pooling size of (2,2). This was done to help reduce the dimensions of the feature maps while retaining important information.

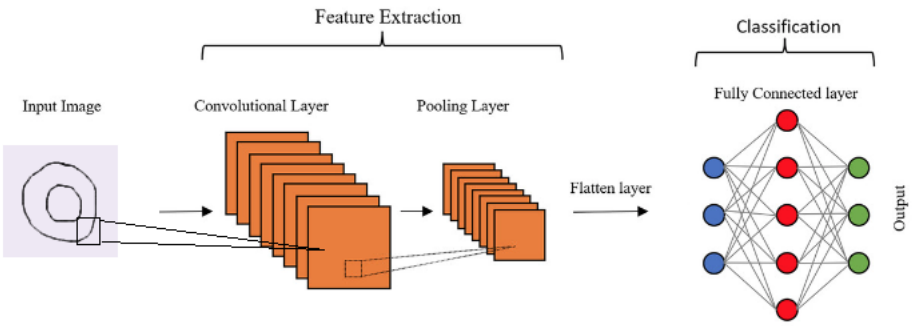
After the first max pooling layer a dropout layer was added. This was done to help prevent overfitting as the dropout layer sets a fraction of the inputs to 0 (in my case 0.2) during the training phase.

A flatten layer was then added, flattening the 2D feature maps into a 1D vector which was input into my fully connected dense layers.

Two dense layers follow this, the initial dense layer contains 680 neurons with ReLu activation. A dropout layer with a dropout value of 0.5 is added, followed by the final dense layer which has Num_Classes, 345 neurons with softmax activation. This layer outputs the probability distribution over the classes.

My output layer contains Num_Classes number of nodes again with the softmax activation function applied to produce the probabilities that the input image belongs to each class.

My model was compiled using the adam optimizer, this allows the neural network to adapt learning rates and allows for smoother and more stable optimization of the model.



High Level Representation of CNN – sourced from (Alsaleh, A. and Perkgoz, C., 2023).

I was then able to view a summary of my Convoluted Neural Network using model.summary().

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 64, 64, 32)	320
max_pooling2d (MaxPooling2D)	(None, 32, 32, 32)	0
conv2d_1 (Conv2D)	(None, 32, 32, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 16, 16, 64)	0
dropout (Dropout)	(None, 16, 16, 64)	0
flatten (Flatten)	(None, 16384)	0
dense (Dense)	(None, 680)	11,141,800
dropout_1 (Dropout)	(None, 680)	0
dense_1 (Dense)	(None, 340)	231,540

Summary of Created Model from model.summary()

3.5 – Model Training

My CNN was trained on the Google QuickDraw! Dataset. It was trained on batches of 1024 images of size 64*64. These images were generated using the dataset at random

allowing for better generalisation of the unseen test data in future. It also helped to evenly distribute the training across all 345 categories in the dataset, helping to ensure some were not overly represented while training.

The training stage consisted of 150 epochs with 16 steps per epoch. This allowed me to train the model on 16 384 samples per epoch, or a total of 2457600 samples throughout its training. As these were all randomly selected this offered appropriate coverage of the dataset.

Multiple combinations of training were tested including increasing the number of epochs and batch sizes to 250 and 2048 respectively. This did not make any significant difference to the models' training accuracy score and led to increased risk of overfitting, so the settings mentioned above were used to train the final model.

The data used to test this model was, like the data used to train it, sourced from the Google QuickDraw! Dataset and also consisted of testing and validating 1024 samples for 16 steps in each epoch. This was again done for 150 epochs. This data was also randomly selected allowing for accurate validation results per epoch.

```
Epoch 126/150
16/16 ━━━━━━━━━━━━━━━━━━━━ 150s 10s/step - accuracy: 0.5669 - loss: 1.6998 - val_accuracy: 0.7018 - val_loss: 1.1729
Epoch 127/150
16/16 ━━━━━━━━━━━━━━━━━━━━ 149s 10s/step - accuracy: 0.5706 - loss: 1.6592 - val_accuracy: 0.7056 - val_loss: 1.1756
Epoch 128/150
16/16 ━━━━━━━━━━━━━━━━━━━━ 149s 10s/step - accuracy: 0.5626 - loss: 1.7077 - val_accuracy: 0.7076 - val_loss: 1.1630
```

Screenshot of output while model was training and validating, includes accuracy scores for both

Following the completion of the models training and validation I was then able to save my model using the model.save("model.h5") command, this saved my trained model and allowed me to call it in the future.

3.6 – Model Evaluation

As the model was training and validating, I was able to get a sense of the accuracy levels, with each accuracy score for an epoch being output following the completion of that epoch. Following completion of the training I was receiving accuracy scores of approximately 59% on the training data and 74% on the validation data.

```
Epoch 150/150
16/16 ━━━━━━━━━━━━━━━━━━━━ 184s 12s/step - accuracy: 0.5905 - loss: 1.5741 - val_accuracy: 0.7394 - val_loss: 1.0353
```

Accuracy Scores on Training and Validation data for final Epoch

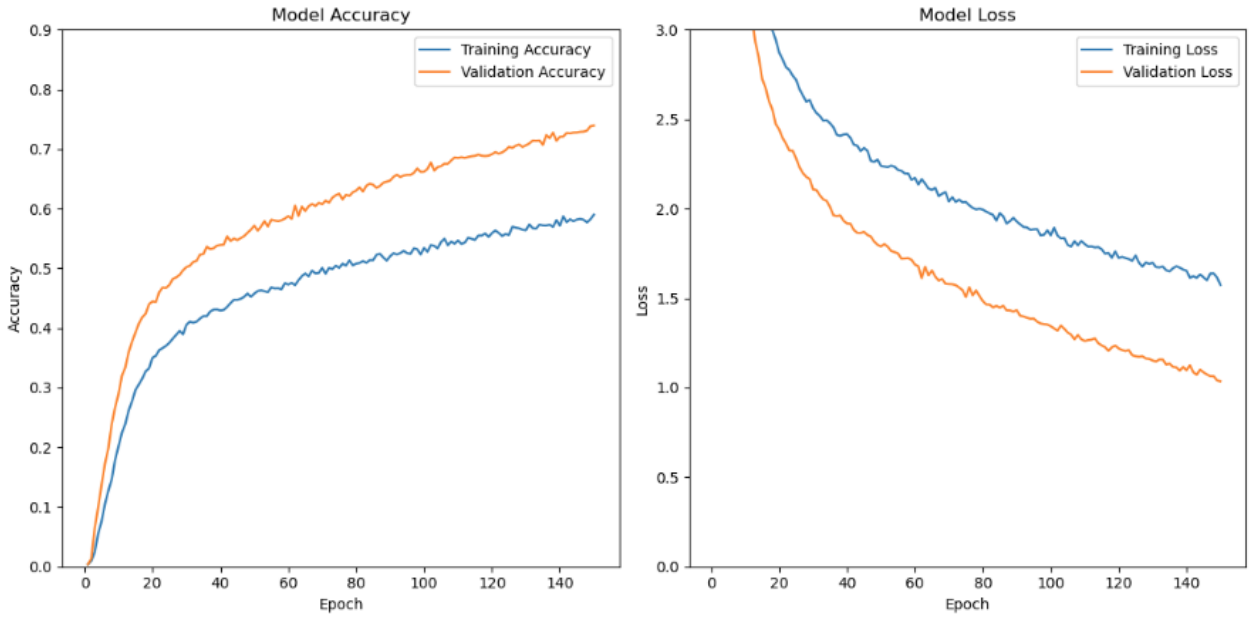
I was also able to observe the loss of each epoch, which is a representation of the difference between the models actual predictions and the actual label. This steadily

dropped over the 150 epochs the model was trained and tested for showing results of 1.05741 loss for the final epochs training data and 1.0353 for the validation data of the final epoch.

```
Epoch 150/150
16/16 ━━━━━━━━━━━━━━━━━━━━ 184s 12s/step - accuracy: 0.5905 - loss: 1.5741 - val_accuracy: 0.7394 - val_loss: 1.0353
```

Loss results for Training and Validation Data after final Epoch

Following the completion of the training and validation of my model I was able to plot the accuracy and loss results of the model for both the training and validation data over the 150 epochs, this allowed me a visual insight into the performance of my model and how it improved over the training period.



Graphs showing the accuracy and loss of the model on both the training and validation sets across 150 Epochs.

As we can see from these graphs the model showed consistent improvement across the 150 epochs reaching accuracy scores of approximately 74 and 59 percent on the validation and test sets respectively. We can also see the loss results consistently decrease over the training and validation period, both of these indicating a higher level of accuracy and less discrepancy between the models' predicted results and the actual results over time.

We can see the model made its most significant improvements between Epochs 1 – 40 increasing exponentially in this period before gradually tapering off and leading to more consistent loss and accuracy scores for the remainder of the training and validation period.

Unusually we can clearly see better loss and accuracy scores on the validation set for this model. This is likely due to the relatively high dropout rate introduced when the model is training, as this is not present during the validation process. A likely combination of this dropout rate and the fact that there is no major effect on a drawing having a label or not, as some drawings can have unique and clearly defined features which differentiate them from others without the use of labels.

Implementation of Front and Back End

4.1 – Technologies Used

Front End – When creating the front-end, I originally used the .NET platform to create a WPF application. This was due to familiarity with this technology and allowed me to experiment with creating the User Interface and the front-end functionality in an environment that I was comfortable in.

.Net Created by Microsoft.

However, when this was completed it became apparent that WPF was not suitable for deployment on mobile devices. I then made the decision to migrate to Flutter (Flutter 2019). and rewrite the User Interface and front-end functionality in this environment that is designed to be compatible with both web app and mobile deployment. Flutter is an open-source UI toolkit developed by Google to build applications and User Interfaces that are compatible across multiple devices, including both web applications and android applications, using flutter allowed me to implement a front end and provided a way for me to test both the front end and User Interface from my computer before deploying the front end of this application to an Android device.

Flutter, created in 2015 by Google.

Back End – For my backend I created a Flask (Flask n.d.). application written in python which provided a framework for building web frameworks and provides essential tools for building scalable backend systems. It includes functions such as handling HTTP requests and allows the backend to send and receive data to and from the frontend.

Flask: Developed in 2004 by Armin Ronacher

My backend functionality was written within this flask application using python. It uses Two APIs, SERPs Google Image Search API to retrieve images based on the predictions and IBMs Watson TTS API which convert the prediction label into an audio file.

4.2 – Front End Functionality

My Front Ends functionality consists of 8 core components separated into 3 main UI aspects and 5 others that work behind, processing data from the user and retrieving results from the back end.

These components focus on tackling separate aspects of the core requirements, such as allowing for accurate user drawing data to be passed to the backend, it also tackles the receiving of prediction data and aids the display of this user data to the user. Essentially this part of the project provides efficient and accurate communication to the backend, allowing user data and predictions to be processed, passed, received, and finally displayed to the user.

In this section we will be briefly discussing the functions of the User Interface but will be focusing on the components behind it in more depth

The front end retrieves the data drawn by the user as an image of size 400*400, this is then scaled to size 64*64 with the thickness of the lines scaled accordingly in order for it to be read by the model and sent to the Flask application, this is an essential component of the frontend as without this the model returns inaccurate predictions.

This scaled drawing is then displayed to the user before they confirm submission, displaying the scaled points and essentially the scaled image that will be fed to the model to retrieve a predicted label.

The frontend then adds each point to a List<Map<String, dynamic>> object which is a List of Maps and contains both the X and Y coordinates of each point, this List is then converted to a JSON String before being sent to the backend when the user confirms submission of the drawing.

Upon receiving a response from the Flask application in the form of a prediction label, an image URL, and an audio file path. This data is then fetched and returned as a label, an image and a .wav file to the user. This involves using an “Image.network” flutter widget which retrieves images based on a URL, this displayed the Image alongside the predicted label. The audio is retrieved by using the URL passed to it from the Flask application and assigning this to an” AudioPlayer” object which allows the .wav file to be played.

4.3 – Back End Functionality

The Back End contains most of the technical components for this application, hosting the model and both APIs the SERP image search API, and the IBM Watson TTS API.

My Backend is primarily responsible for the fetching of results based on the data passed to it from the front end, this part of the application ensures accurate predictions are made based on the data passed to it and is also responsible for fetching the prediction, along with audio and visual representations of the result. This part of the application communicates with the front end of the application to combine and help meet the core functionality and requirements of the application.

The backend is responsible for feeding the user input received by the frontend into the model and returning the top prediction for that image. It does this by initially converting the points retrieved from the frontend to an image array and feeding this 64*64 image to the model.

This returns a predicted label for the drawing which is both sent to the frontend and also passed to both APIs to retrieve additional data.

When this predicted label is retrieved it is passed into the SERP API with parameters for the query, this returns an image URL based on the search, the IBM Watson API also converts the label into a .wav audio representation of the word.

The backend then sends this data to the front end as a json response, containing the predicted label as a string, the image url, and the path to the audio file.

Output from Flask Backend Application correctly predicting "Donut".

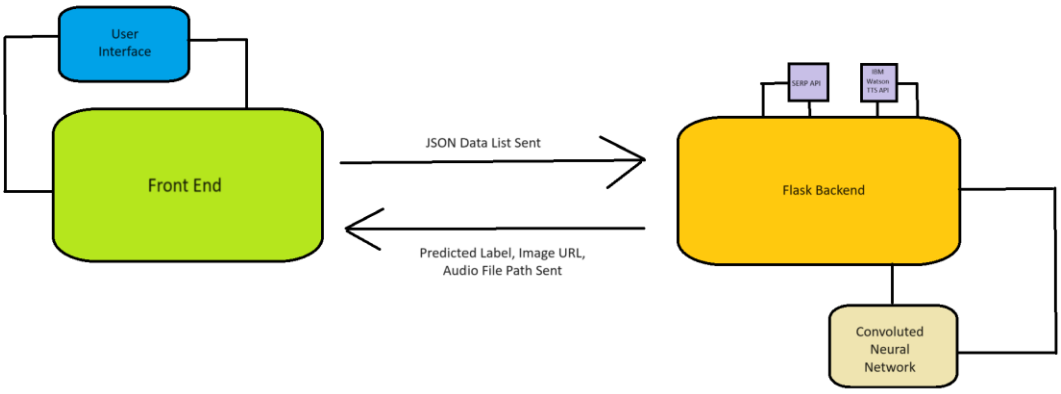
4.4 – Front – Back End Communication

My front and back end communicate using HTTP Post requests, with my frontend sending a list of points retrieved from the drawing element to the backend, this list is then processed into an image and fed to the model.

This was an essential aspect to the application, combining both the core aspects of both the front and back ends of the application together to fulfil the core requirements of the project.

The results from the model and the backends APIs are then sent as a JSON response to the frontend for display.

The frontend checks the response code for a "200" which indicates a successful response has been received. It then extracts the data from this and processes it to display correct results to the user.



Overview of Communication between Front and Back End

My Flask Backend application provides an endpoint to receive the drawing data which is also specified within the HTTP Post address alongside the server's address. This allows the frontend to accurately send the data to the correct location and retrieve the responses from this.

4.5 – APIs Used

I used two APIs for this project, both located in the Flask Application within the backend of my application. These APIs were responsible for both fetching images based on the models predicted label, and also creating a .wav audio file containing an audio representation of the predicted label returned.

For my image retrieval API, I used SERPs Google Image Search API, this allowed me to search for specific images based on a string fed to the API. This string contained the predicted label along with a specification to search for this label alongside the terms "Flaticon" and "icon png".

This tells the API to retrieve results primarily from a site "Flaticon" which provides icons in various formats, it also specifies to receive pngs to avoid any errors due to incorrect file compatibility when being represented by the frontend.

For my Text To Speech API, I originally attempted to use Googles TTS API however found that it was increasingly complex compared to alternatives, I then decided to use IBMs Watson TTS API which provided a more straightforward way of setting up the API.

This API returns an audio file based on a text and voice parameters, with the text being my predicted label. The voice parameter allows specification of the voice used for the TTS conversion. I left this as the default "en-US_AllisonV3Voice", as this provided what I was looking for in terms of clearly and concisely providing an English TTS result that clearly pronounced the word.

This .wav file was then saved to a folder and the JSON response contained both the Image retrieved by the SERP APIs URL and the file path for the .wav file returned by the IBM Watson TTS API.

4.6 – Server Implementation

Implementing a server was essential for providing Mobile support for the application, providing a place for the backend to run so that it could be accessible remotely. This aspect of the project involved collaborating with members of the faculty within the school of computer science within the University of Galway and allowed me to gain space for my backend on the computer science server.

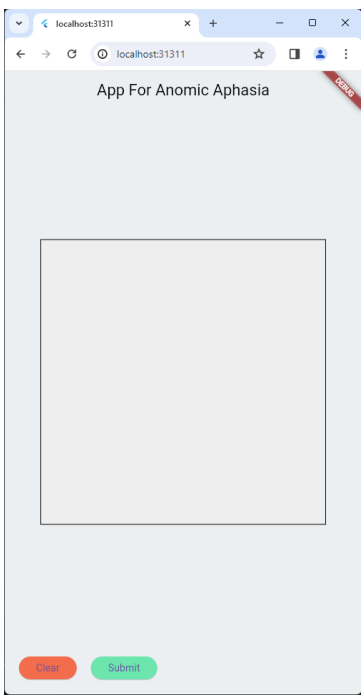
Migrating my backend to the server space involved moving not only the backend but also the various components that were used and called by the backend to the server, these included the model and various files containing data that allowed the model to retrieve labels based on results returned by the model. I also needed to create a file on the server space to hold the audio recordings returned by the IBM Watson TTS API.

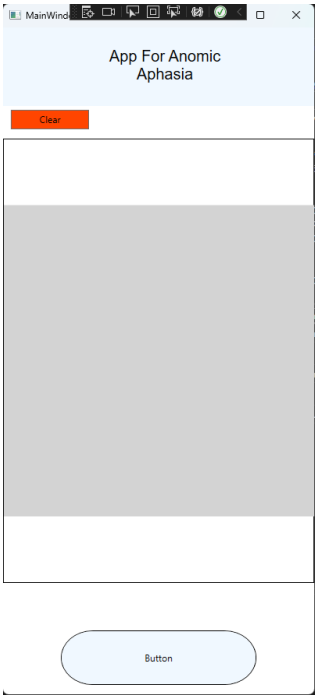
Hosting the backend remotely involved not only implementing the server, but also involved modifying aspects of my front end so features would be compatible, this included changing addresses that ips pointed to within the front ends code base, but also involved modifying my audio playing method, as originally the files would be saved locally by the backend before being called by the front. This change was made so that files would be saved remotely using the backend before being retrieved by the front end in a manner similar to that being used to retrieve the data from the predictions.

4.7 – User Interface Design and Functionality

Both my WPF and Flutter User Interface (UI) design followed similar patterns, consisting of 3 main components initially.

These were the Submit, Clear and Drawing Element components of the UI. The most challenging part of designing the User Interface was creating a drawing element to accurately capture user input.





Flutter UI *WPF UI*

The drawing element was the most important aspect of the UI with the other options needing this to function correctly in order to be applied. Within the initial WPF UI draft a canvas element was used to define the drawing area. Points were then captured every .2 seconds whilst the user was drawing. The movements of the mouse were tracked using mouse handling events, allowing data only to be gathered when the mouse was being clicked down.

When converting this to flutter this drawing element was replaced with a CustomPaint widget which was used to create a custom drawing area. Touch handling events such as onPanUpdate were used to track the users drawing input within the element and this was then represented within the drawing element using a CustomPainter to draw the lines according to the users' input.

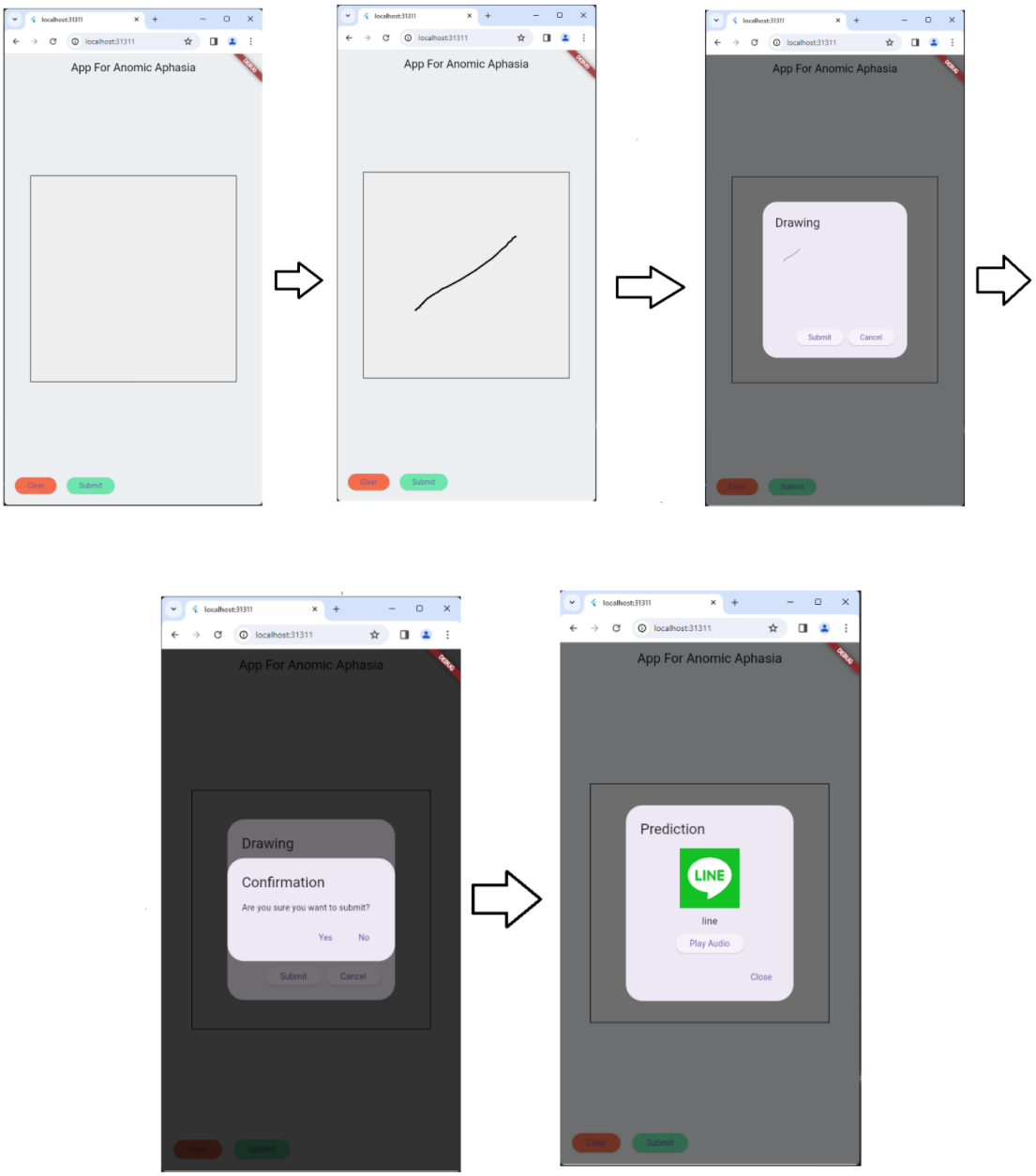
The drawing element aspect required capturing the user inputs in a precise and accurate manner and updating the UI accordingly. Storing the points drawn and scaling them was all handled within the front end of the application behind the UI.

A clear button was added in order to clear the canvas if the user was not happy with the drawing or made a mistake. This involved clearing all user input and displaying a Drawing Element Cleared message when the button was clicked.

A submit button was implemented when the user was satisfied with the drawing, this prompted to user to again confirm that they intended to submit the drawing then scaled all captured points and sent them to the backend for processing.

Following retrieval of a prediction, a pop up containing the predicted label alongside an icon of the image and a play audio button was shown to the user, displaying the predicted label of the drawing in Text, Visual and Audio format. The user could then close this and clear the drawing area to draw a new item.

The UI was developed early in the project as it was essential for confirming the model could accurately predict user drawn input. Despite the initially complex method for capturing the user input the UI is simple and contains all necessary functionality for the application to function as a whole.



Each Aspect of the UI when correctly guessing a "line".

4.8 – Challenges Faced During Implementation

While implementing the functional aspects of the application I encountered various issues, Here I will discuss some of the most problematic and challenging, these included rewriting the frontend using an unfamiliar language and environment, formatting user input correctly and representing the API results.

The largest issue faced while developing the functionality of the front and back end was by far the issue of rewriting the frontend to be compatible with web and mobile applications.

Initially written in WPF using C# I discovered there was little to no support for deploying WPF application either as a web or mobile app. This was due to WPF applications being standalone executables that run on local machines, meaning that deploying the WPF application as a Web or Android application was not feasible. This resulted in me needing to attempt a new approach for developing my frontend so it could be deployed as these.

This led me to use flutter which is primarily used for developing cross platform applications. Flutter uses a system of widgets not completely dissimilar to a WPF application, however these widgets are created in completely separate languages with WPF applications primarily using C# and XAML and Flutter primarily using Dart.

```
child: GestureDetector(
  onPanDown: (details) {
    setState(() {
      isDrawing = isWithinDrawingArea(details.localPosition, Size(400, 400));
      if (isDrawing) {
        points.add(DrawingArea(
          point: details.localPosition,
          areaPaint: areaPaint,
        )); // DrawingArea
      }
    });
  },
```

```
public async Task timerTask(MouseButtonEventArgs e)
{
    while (1!=0)
    {
        if (mouseDown == true)
        {
            pointReleased = (Vector)mousePoint;
            //Console.WriteLine("In Thread");

            Console.WriteLine("MousePressed at Point" + pointReleased);
            currentLine(pointReleased);
            //newStrokeDetected(pointReleased);
            await Task.Delay(TimeSpan.FromSeconds(.2));
        }
        else
        {
            Console.WriteLine("timer stopped");
            newStrokeEndDetected();
            break;
        }
    }
    //Thread.Sleep(100);
}
```

Comparison of Code for Logging of Data Points with Flutter Application on the left and WPF on the right

While both are Object Oriented Languages, Dart shares more similarities with JavaScript and Java than C#. The functionality from the WPF Application was completely rewritten and improved as a Flutter application, allowing me to deploy my

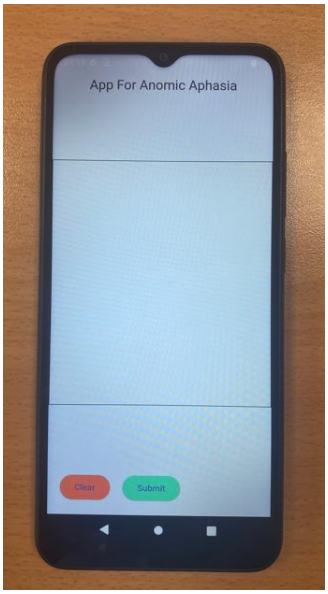
frontend as both a Web and Mobile application, Due to the initial WPF draft of the application having the same functionality as the updated Flutter Application this was more so an issue of learning how to use Dart and rewriting the application within flutter than issues within the functionality of the WPF draft of the application.

Another issue faced in the creation of an initial frontend was the formatting of user input to be sent to the backend, I initially put far too much focus on mirroring the user's data to match the format of data within the dataset and didn't allow for exploration of other ways to format the data so that it could be processed and read by the model. I learned from this while developing the flutter version of the application and used a much simpler and more straightforward approach of formatting the data as a series of x and y points and assigning these to maps within a list as opposed to logging the points as they were drawn and creating lists of lists like I did within the WPF application.

I also faced issues when displaying images in the UI from the URLs passed from the backend. This was due to the API retrieving and passing images that were not compatible, due to size or data type to the frontend. I managed to solve this by identifying compatible image sizes and formats and then specifying the API to search for these within the query.

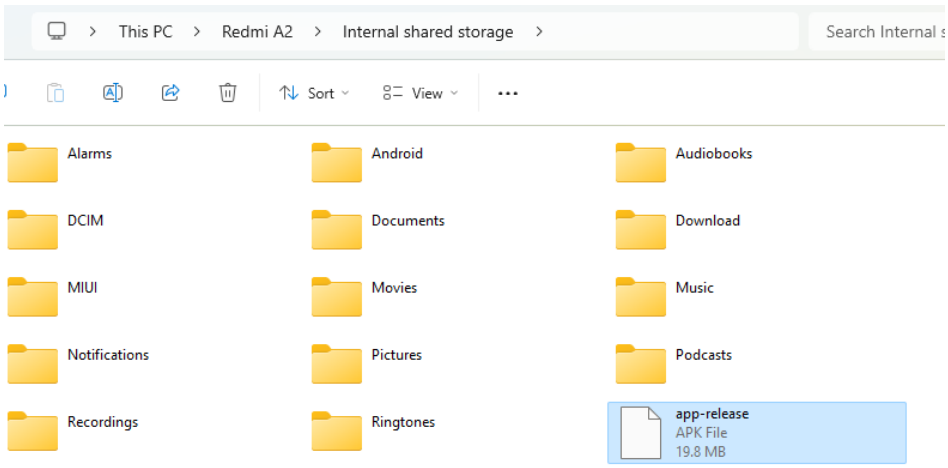
4.9 – Migration from Web to Mobile

Initially when I started developing the Flutter Application, I was able to use an Android Emulator to view the UI, this was done with only a mobile application in mind, however I encountered memory issues due to using the emulator and had to deploy the application as a Chrome Web App in order to view and interact with the UI. It was with this in mind that the Application was first developed as a Web App and then due to the cross-platform capabilities of Flutter it would be relatively straightforward to deploy the front end of the application on an android device.



Picture of Application deployed on android device.

I ensured the front end of the application was functioning as intended and was then able to create an APK of the flutter application, from here I was able to install and run this APK on my physical android device. Due to Flutters cross platform functionality it both applications use the same structure and files, with the flutter APK allowing the web app to be deployed as a mobile app without specifically altering aspects of the application for mobile compatibility.



Transferring Flutter Apk to Android Device.

However, the main difference in running the application as a web app or mobile app lies in the connection to the backend. With the web app the frontend was able to communicate with a locally based backend, however when implementing this on a mobile device the location of the backend needed to be on an accessible server. This allows the Mobile App to communicate with a remote server and reach the backend despite it being external, this meant that testing for the application was primarily done with the application as a web app before deploying my Flask Backend on a server and being able to reach it using my Mobile Application.

Migrating to Mobile involved changing the URL that the frontend was using to find the backend to the backends new address on the remote server. It also required altering and adding various permissions before generating the release apk to send to the mobile, such as adding permissions for the flutter application to access the internet, this was essential for communication between front and back ends of the application.

Development Processes

5.1 – Version Control

I used GitHub for version control allowing me to log changes made to each aspect of the code and save drafts of my code. Alongside logging changes made to my code this also provided me with a method of ensuring I had backups of functioning code if major issues arose, which gave me a level of security when making significant modifications to various aspects of the application.

GitHub also allowed me to record what changes were made to each version of the software and why. Easily providing a record of what I had most recently completed in the most recent commit.

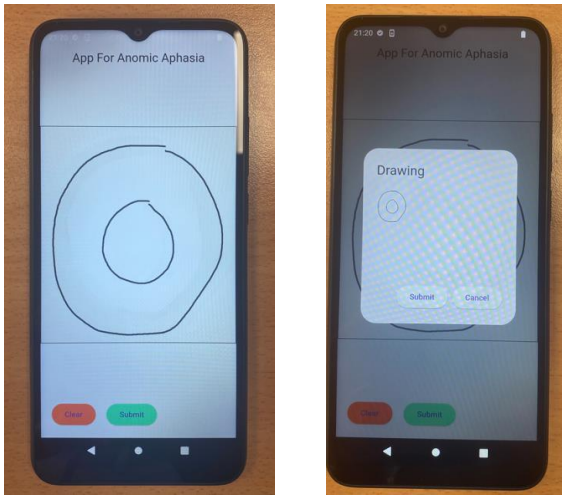
5.2 – Testing – Web

Most of my initial testing was done using the web app, this was largely due to the ease of communication between the frontend deployed as a web app and the backend deployed locally on my machine.

This meant that most of the functionality testing, and model testing was carried out here, ensuring everything functioned on the web app before I migrated the application to mobile. Virtually every aspect of the application was tested using the web app, including verifying model inputs, verifying communication between the front and back end, and ensuring the application worked as intended in its entirety.

5.3 – Testing – Mobile

Due to most of the application testing being done on the Web App, my mobile testing mainly consisted of ensuring that the backend was reachable from the mobile application. I also ensured that results coming to the Mobile Application were consistent with those being returned from the Web App when similar user inputs were passed from both.



Images of UI in different stages deployed on mobile

My mobile testing was largely made up of ensuring results on the Mobile App matched the standards of those returned by the Web App, due to the web app functioning to the desired level this allowed for confirmation both applications were working as intended.

Application Evaluation

6.1 – Model Performance Evaluation

The models performance, or the accuracy of the models' predictions largely lies on the user drawn image, with simple clear images returning correct results, and complex, poorly drawn images tending to have much lower accuracy.

However, despite this it is quite easy to understand in most cases why the model returns incorrect predictions, mistaking similar objects, such as a broom, for a rake. This is also evident when examining the originally model results, which contain a percentage of how confident the model is that the item its predicting is correct. This feature was used initially but was not brought through to be displayed in the final development of the application.

The model showed accuracy scores of approximately 74% on unseen data while testing and I would largely say this is consistent with results I have seen when testing the application over the past weeks. However, it cannot be understated the importance that having clear well drawn images have on the model.

Overall, I would say I am relatively happy with the models' predictions, as in the vast majority of cases well user drawn input will result in a correct prediction. And in most incorrect cases, it is quite simple to see the similarities between predicted and accurate labels, with similar features appearing in both.

6.2 – Text To Speech API Evaluation

After originally encountering issues when attempting to integrate the Google TTS API I decided to use the IBM Watson API. This API performed exactly as anticipated and required, clearly producing audio versions of the predicted labels with little to no error. Each result produced by the API matched the label given and for the most part was pronounced as intended. There were minor errors within the pronunciation of certain words due to the spelling within the dataset not directly reflecting the way the word is pronounced, for example "Donut" s pronunciation is represented as "Don-it" as opposed to the correct "Dough-nut". However, this is not necessarily a criticism of the API as it seems to directly create audio versions of the word passed to it as opposed to having a database of words and their pronunciations.

I also had no issues with exhausting API limits with this API as there was no cap placed on the number of queries.

6.3 – Image Search API Evaluation

This API also performed as required returning images relevant to the query posed with no errors, it allowed for easy query of each label and the option to modify each query to return specific image URLs, this proved useful due to some URLs being incompatible with display within my UI. One notable aspect of the API is that it tends to return the exact same image if the query remains the same, this proved problematic when the images being returned were not able to be displayed in the UI, however modifying the query to only search for compatible images completely removes this issue.

I made the decision to only retrieve Icons from this API mainly due to their efficient representation of the object they were portraying and also their consistent size, with most icons being created specifically for the purpose of being used in mobile or web applications. Modifying the query to only include icons from specific sites also helped ensure consistent images were returned by the API across all categories, improving the feel and flow of the application.

6.4 – User Feedback

My User Feedback stage consisted of gathering 5 random individuals and asking them to draw 5 drawings within the app, these individuals were given encouragement to ensure the images were well drawn but were allowed complete freedom over drawing in the application.

Results for each of these users were tallied and recorded, with 19 out of a total of 25 predictions being returned correctly, or 76% of the users tested predictions being correct. This lies in line with my own extensive testing receiving similar results, this also lines up with accuracy scores retrieved from the model during testing which were approximately 74%.

Users were also invited to provide feedback on their thoughts of the application, such as aspects of the application that they enjoyed and also aspects of the application they thought could be improved or additional features they would like to see implemented.

Users reported high levels of satisfaction with the User Interface with 4 users reporting they thought the responsiveness and layout of the User Interface was excellent. Users also enjoyed the simplistic design with 3 users highlighting the conciseness and clear display of predictions and their results.

Users also thought highly of the predictions returned by the model. Commenting on the model as accurate for the most part, with multiple users also expressing how they understood why the model made predictions it did in cases it got the predictions wrong.

Amongst the features that users said they would like to see implemented were, the addition of an erase and undo feature, increasing the canvas of the drawing elements size, the addition of more colours, and also the wait time decreased whilst the user was waiting for the UI to display predictions

Potential Improvements and Next Steps

7.1 – Improving Model

One major component that could be improved upon is the model, whilst demonstrating a relatively high success rate when making predictions on the unseen data this accuracy score of 73%, while this is acceptable for initial development of the application ideally this could be increased to upwards of 85% and account for more difficult drawings to consistently be predicted correct with a higher degree of accuracy.

Increasing this accuracy score should aid with the model consistently returning more correct and less incorrect guesses, With the current model we see promising results when user drawn data resembles the training data, however improved training of the model would result in increased accuracy when the model is dealing with poorly drawn

objects. At the moment the model struggles to distinguish between similar items, being unable to conclusively say that a tree for example is not broccoli, with an improved model these “grey” area objects could be better classified and produce consistently better results and an improved user experience.

7.2 – Improving APIs

Due to the SERP APIs monthly search cap if this application were to be scaled or deployed to multiple users this cap would be met quickly, resulting in the image retrieval not working after a number of searches, while using SERPs API is manageable for testing and development purposes, large scale deployment of this application would call for either a subscription to a SERP plan that provides more monthly searches, or moving towards a different Image Search API that does not have a low monthly search cap.

7.3 – iOS Compatibility

This Application could be developed to be compatible with iOS Systems, Due to time and hardware constraints it was only possible to deploy this Application as a Web App and as an Android Application. Migrating and developing this application for iOS would increase the reachability of this application allowing users on both android and iOS devices to use this application.

7.4 – Profile and User History Implementation

A potential next step for this application could be the introduction of User Profiles, linking accounts to specific individuals, This could allow users access to a bank of previously drawn images and allow for easier word retrieval for people using the application. This could also result in improved accuracy as by saving the users drawings this data could be used to continually train and improve the model resulting in improved predictions. This could also be a method of gaining access to new data for research into people with Anomic Aphasia, offering insights into particular words particular people struggle with. Implementing Profiles could help link trends within people with the condition and allow for improvements in understanding and treating the condition.

7.5 – Dataset Expansion

At the moment there is only the possibility for classification across 345 categories, Improving and expanding this dataset could provide a larger bank of terminology for the users to access, aiding them in retrieving words for objects that simply are not yet in the dataset. This could be done in a number of ways, one potential approach could be to store user generated drawings and their labels, and use this to develop a new larger dataset, prompting the user to classify their drawing with a label if the model did not correctly predict their drawing, this dataset could then be used to aid and develop newer models with more possible results.

Conclusion

8.1 – Conclusion

In conclusion, This application provides a real world aid for people with anomic aphasia and offers them real time communication assistance from their mobile phone.

By using Machine Learning and a user focused design this application provides an interface for users to gain accurate labels for words they struggle with in their day to day life.

This project allowed me to fully immerse myself in the creation of a useful application that provides a solution to a real-world problem. This project allowed me to utilise a variety of skills that I picked up during my 4 years in University of Galway and also allowed me to practically use skills I developed during my 9-month professional placement.

This project also introduced me to a variety of new concepts, tasks, and approaches to solving practical issues, ranging from an introduction to Neural Networks and Machine Learning approaches, to the process of hosting backends on servers remotely and

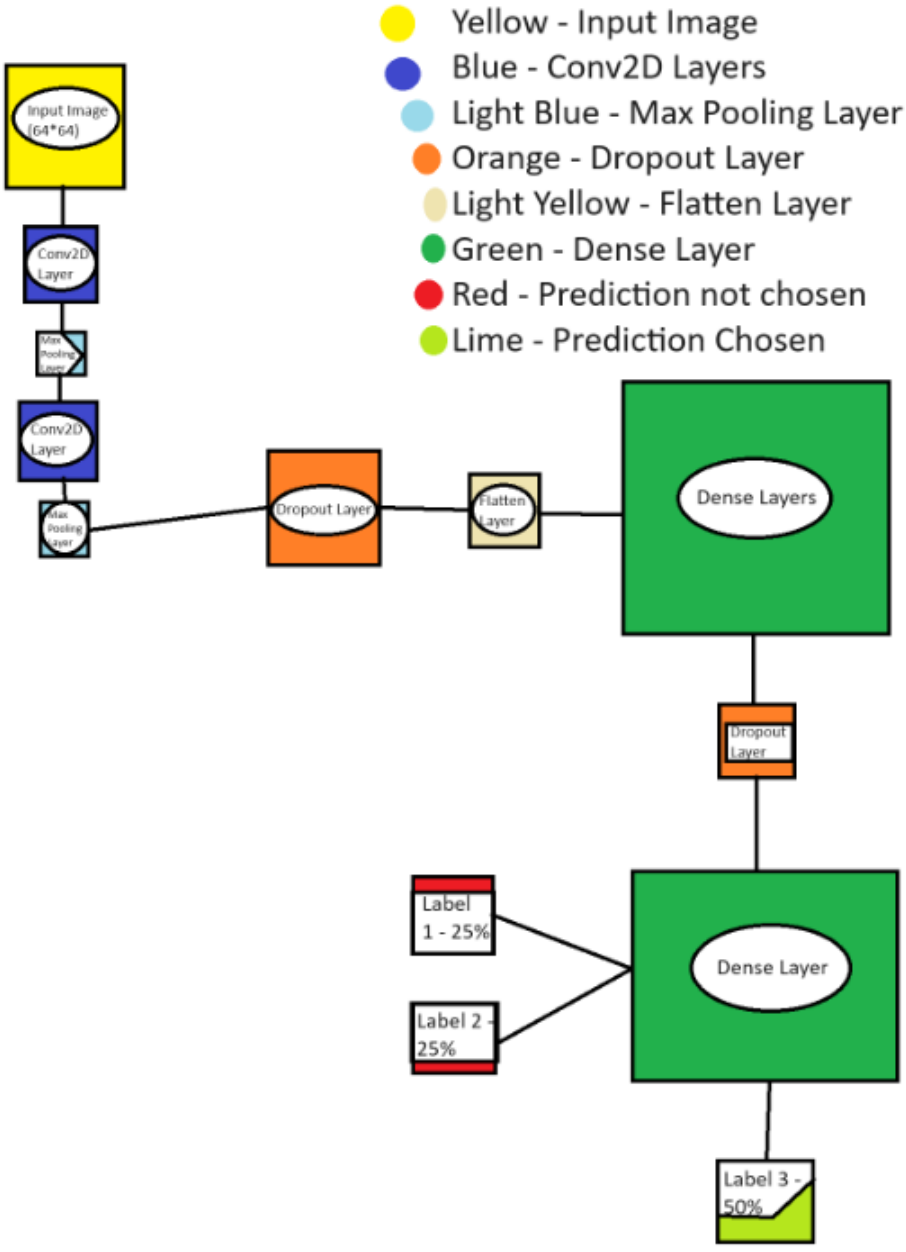
learning how to communicate between these and applications running on various different platforms.

This project also served as an introduction to APIs and their real world uses and capabilities. Using the two APIs that I did use also introduced me to a variety of APIs suitable for other tasks that could be useful in a wide range of different problems.

Overall, this project was a fantastic way to develop both my software skills and also an application that I believe has the potential to have a positive impact on the lives of people with Anomic Aphasia around the world.

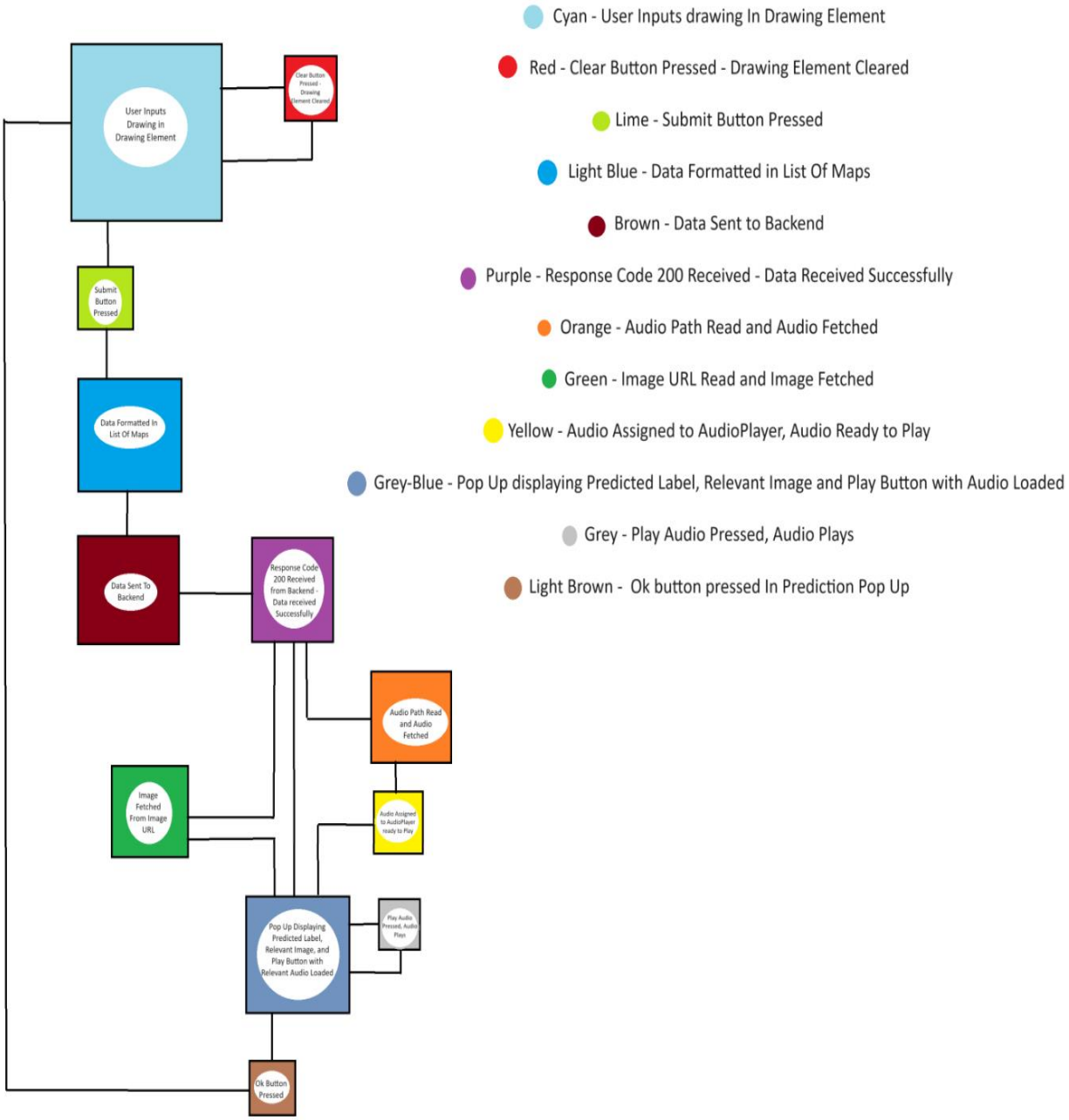
Process Flow Diagrams

9.1 – Model



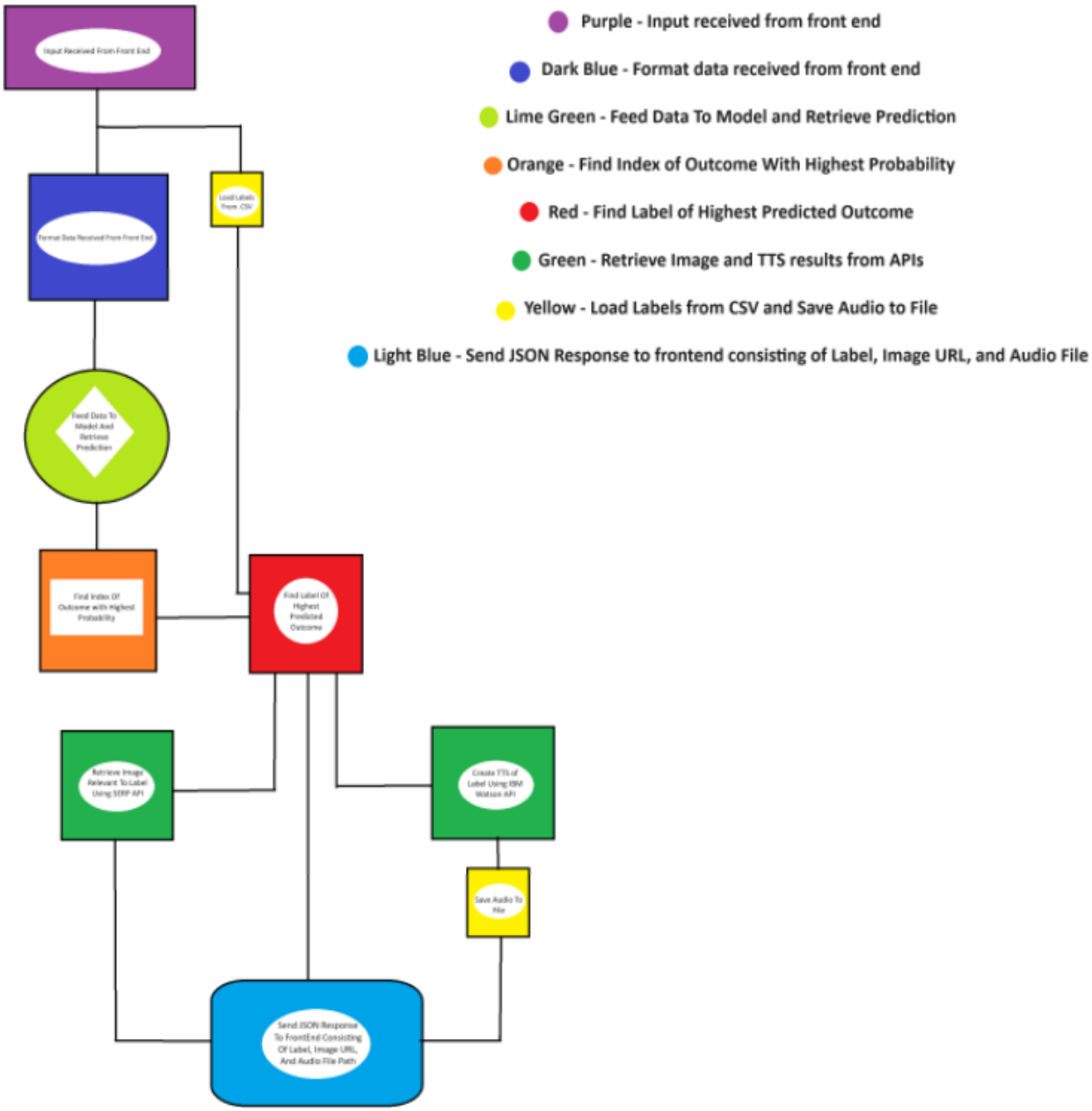
Process Flow for Model – Diagram available in full in GitHub Repository

9.2 – UI and Frontend



Process Flow Diagram for UI and Frontend -Diagram available in full in GitHub Project Repository.

9.3 – Backend



Process Flow Diagram for Flask Backend Application – Diagram available in full in GitHub Project Repository

Acknowledgements

I could not complete this report without thanking the people that have helped me throughout this project, I received a large amount of help and support from a large number of people.

I would like to thank my supervisor, Dr. Conor Hayes, for his support and guidance throughout every aspect of this project, from being instrumental in almost every stage of this project, offering expertise and advice on any areas I was struggling with to putting me in contact with various people who would aid me in completing different aspects of the project. Conor was more than happy to meet weekly with me which provided extra motivation and deadlines to each and every aspect and problem I faced.

I would like to thank Joe O'Connell who was vital in allowing me to migrate this application to mobile platforms, Joe helped me when hosting the backend on the University of Galway server, walking me through the process of how to launch applications and host various components on a remote server. Joe also was happy to answer any queries I had in relation to this aspect of the project and was able to offer different approaches on how this aspect of the project could be completed.

I would also like to thank my family and friends for offering their support and guidance throughout this project and additionally for providing user feedback and advice on how to manage my time and balance this project with other college deliverables. This allowed me to ensure that I stayed on top of all aspects of the course while completing this project.

Bibliography

Alsaleh, A. and Perkgoz, C., 2023. A space and time efficient convolutional neural network for age group estimation from facial images. PeerJ Computer Science, 9, p.e1395.

Andreetta, S., Cantagallo, A. and Marini, A., 2012. Narrative discourse in anomic aphasia. Neuropsychologia, 50(8), pp.1787-1793.

Android Studio (2019). *Download Android Studio and SDK tools*. [online] Android Developers. Available at: https://developer.android.com/studio.

Flask (n.d.). *Welcome to Flask — Flask Documentation (3.0.x)*. [online] flask.palletsprojects.com. Available at: https://flask.palletsprojects.com/en/3.0.x/.

Flutter (2019). *Flutter - Beautiful native apps in record time*. [online] Flutter.dev. Available at: https://flutter.dev/.

Google Cloud. (n.d.). *Cloud Text-to-Speech API*. [online] Available at: https://cloud.google.com/text-to-speech/docs/reference/rest.

Guo, K., WoMa, J. and Xu, E., 2018. Quick, Draw! doodle recognition. Stanford University.

Kaggle (2022). *Kaggle: Your Home for Data Science*. [online] Kaggle.com. Available at: https://www.kaggle.com/.

kaggle.com. (n.d.). *Google Quick, Draw! Challenge*. [online] Available at: https://www.kaggle.com/code/nvatuan/google-quick-draw-challenge [Accessed 28 Mar. 2024].

Microsoft. (n.d.). *.NET | Free. Cross-platform. Open Source.* [online] Available at: https://dotnet.microsoft.com/en-us/.

Quick, Draw! The Data (2019). *Quick, Draw! The Data*. [online] Withgoogle.com. Available at: https://quickdraw.withgoogle.com/data.

Saha, S. (2018). A Comprehensive Guide to Convolutional Neural Networks—the ELI5 way. [online] Towards Data Science. Available at: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

Smartyearsapps. (n.d.). *Aphasia Therapy App | IName it | Smarty Ears Apps for Language*. [online] Available at: https://www.smartyearsapps.com/iname-it/ [Accessed 28 Mar. 2024].

TensorFlow (2019a). *Keras | TensorFlow Core | TensorFlow*. [online] TensorFlow. Available at: https://www.tensorflow.org/guide/keras.

TensorFlow (2019b). *TensorFlow*. [online] TensorFlow. Available at: https://www.tensorflow.org/.

www.aphasiasoftwarefinder.org. (n.d.). *CommunicAide | Aphasia Software Finder*. [online] Available at: https://www.aphasiasoftwarefinder.org/communicaide [Accessed 28 Mar. 2024].

www.ibm.com. (n.d.). *IBM Watson Text to Speech*. [online] Available at: https://www.ibm.com/products/text-to-speech.

Serpapi. (n.d). Images Results. Available at: https://serpapi.com/images-results

Genymobile. (n.d). Genymotion: Android Emulator for app testing. Available at: https://www.genymotion.com/