

Disease Prediction Android Application

Dissertation Project Report

"Haider Raoof"

17004364

A thesis submitted in part fulfilment of the degree of BSc (Hons) Computer Science

Supervisor: Dr Rob Holton

19 May 2022

Declaration

The candidate confirms that the work submitted is his/her own and that appropriate credit has been given where reference has been made to the work of others. The candidate agrees that this report can be electronically checked for plagiarism.

"Haider Raoof"

Abstract

This project details the development and implementation of a Disease Prediction Android Application that can provide its users an immediate predictive diagnosis of disease or condition after inputting their symptoms into the application. An artificial neural network model was trained using TensorFlow and Keras API in Python. This model was converted into a TensorFlow Lite model to be used as an asset within the Android application to perform on-device inference. In other words, perform disease predictions using the TensorFlow Lite model by utilising symptoms input into the application by the user. The application can predict a user's disease or condition from 41 different diseases/conditions and the accuracy of these predictions is 96.72%. The application developed also provides users other assistive features such as being able to save their disease prediction results, view past disease prediction results, view more information about their predicted disease and find nearby healthcare services.

Acknowledgements

I would like to thank my parents, brothers and close friends for their moral support whilst undertaking this project. I would also like to thank my supervisor, Dr Rob Holton, for his guidance whilst undertaking this project.

Table of Contents

Abstract	iii
Acknowledgements	iv
1. Chapter 1: Introduction	1
1.1 Project Description.....	1
1.2 Aims and Objectives	1
1.3 Challenges	2
1.4 Report Overview.....	2
2. Chapter 2: Literature Review.....	3
2.1 Introduction to Artificial Intelligence.....	3
2.2 Artificial Intelligence in Healthcare	4
2.3 Comparison of supervised ML algorithms	6
2.4 Technologies	8
3 Chapter 3: Requirements and Analysis	10
3.1 Project Description.....	10
3.2 Functional Requirements	11
3.3 Non-functional Requirements	12
3.4 UML Use Case Diagram	13
3.5 Evaluation and Testing	14
4 Chapter 4: Design	14
4.1 Project Folder Structure.....	14
4.2 Architecture and components.....	18
5 Chapter 5: Implementation and Testing.....	18
5.1 Dataset Selection and Pre-processing.....	18
5.2 Supervised ML Algorithm Selection.....	20
5.3 Build and Train ANN model	21
5.4 Test ANN model and Convert into TensorFlow Lite model.....	23
5.5 Application Description.....	24
5.6 Screenshots of Application	29
5.7 Predict Disease Functionality Implementation	30
5.8 Unit Testing	35
5.9 System Testing.....	36
6 Results and Discussion	42
7 Conclusions and Further Work	44

8	References	45
9	Appendices	50

1. Chapter 1: Introduction

1.1 *Project Description*

Often individuals experience symptoms related to poor health and need to be diagnosed quickly however usually it is not possible to get an immediate diagnosis. During the COVID pandemic many UK surgeries have stopped allowing face to face appointments, and this has exacerbated the issue of not being able to get a diagnosis and then possible health advice promptly. Instead, surgeries provide remote appointments such as telephone calls, online video calls and also through the use of email correspondence. This may leave many patients feeling inadequately assessed and may even lead to the lack of diagnosis or misdiagnosis of patients. This is an issue that this project aims to assist and minimise the impact of.

This project proposes to create an Android application that utilises Artificial Intelligence and more specifically machine learning (ML) so that a user will be able to input their symptoms into their android smartphone device and then immediately get a diagnosis of what condition they most likely have. The project will be focused around selecting the most appropriate machine learning algorithm to diagnose the user highly accurately and therefore implementing this algorithm. Furthermore, it will require data collection and pre-processing of the data to train the ML algorithm needed for the model. The smartphone application will also be of further use to the patient by providing them assistive features to help manage their condition which will be explained below.

1.2 *Aims and Objectives*

This project aims to provide accessibility to as many users/patients as possible with the development of an Android application that can be used for immediate diagnosis. The application will be developed to allow the user to be able to register to the application and login to the Disease Prediction Android Application. Therefore, a user will have individualised experience once logged into their account. The Android application will be developed utilising Android Studio and Java. It will utilise a trained machine learning model. First a dataset of symptoms and corresponding diseases will be acquired and used to train a machine learning algorithm in order to develop a model. The model will then be incorporated within the Android application. The use of this model will allow inference. This is the process of using the model to predict which condition the user has after the inputting of user symptoms by user input into the application. To provide further assistance to the user, alongside the result that appears, links to healthcare organisation's webpages relating to the condition they are predicted to have will be shown to the user. The user can open the link to receive further care and advice. The application will also allow the user to save their results which will be stored in the application. Therefore, the user can look at previous conditions the application predicted for the user. Furthermore, the application will be able to locate the nearest Hospital/GP practice/Pharmacy to the user. Overall, the application will be very useful and assistive in identifying a patient's condition and directing them with advice about their condition via healthcare organisation webpages or by referring to them to their nearest healthcare practitioner.

1.3 Challenges

This project and the development of the Disease Prediction Android Application present a variety of challenges that will need to be overcome for the successful development of the application and completion of the project. The first challenge will be to acquire the skills and knowledge of Android application development to make sure the application is developed to the fullest and all of its intended features are developed. Furthermore, my previous experience working with a machine learning algorithm has been to perform binary classification to identify if an individual is diabetic or not. The challenge presented with this project extends previous experience as it will require the selection of a machine learning algorithm to train, so that multi-class classification can occur as there will be multiple different diseases that can be classified based on the input symptom data. Selecting an appropriate dataset that is large enough and contains symptom associated to disease features will be imperative. If the dataset is too small, it may mean that the model cannot predict reliable results from which a reliable prediction of disease/condition will be useful for a user. A further challenge may also be presented in the process of pre-processing the data which has been selected to train the machine learning algorithm and to build the model. The dataset may require pre-processing to put it in a suitable format for the machine learning algorithm to be trained on it. This section describes some of the challenges that may need to be overcome for the successful completion of this project.

1.4 Report Overview

The chapters of this interim report are as follows:

Chapter 2 contains a literature review that has been conducted before the development of the application. The literature review defines artificial intelligence and highlights examples of its application in healthcare. Furthermore, it defines machine learning and provides a comparison of machine learning techniques that have been utilised in the literature to predict disease. Lastly, it describes the selection of technologies that will be used for the development of the Disease Prediction Android Application.

Chapter 3 contains a section detailing the requirements and analysis for the application. It details the functional and non-functional requirements of the application. Also, a UML use case diagram for the application as a system is presented. There is also an explanation of how evaluation and testing will occur for this project.

Chapter 4 contains a section detailing the design of this application. This is done by explaining the project folder structure in Android Studio for the application as it provides clarity for how it was designed. Also, a UML class diagram is provided to show the architecture and components of the Java classes used to implement the application.

Chapter 5 details the implementation and testing done develop the application. This section describes the following: dataset selection and pre-processing, the selection of a supervised machine learning algorithm, the building and training of an artificial neural network model, the testing of the model, an application description, screenshots of the application, an explanation of the predict disease functionalities implementation in code, unit testing and system testing.

Chapter 6 presents the results of the project and implementation of the application. Also, it provides some discussion on the deficiencies of the application and changes made to the original plan of the project. It also mentions how data privacy and cyber security were taken into consideration when developing this application.

Chapter 7 is a section on conclusions and further work. In this chapter the achievements and the results of project are stated. Suggestions for further work are provided. Lastly a reflection of the technical skills and practical knowledge that has been acquired by undertaking this project is provided in this chapter.

2. Chapter 2: Literature Review

2.1 *Introduction to Artificial Intelligence*

Artificial Intelligence (AI) is a term that holds multiple different definitions (Russell and Norvig 2021a). However, one of the definitions as defined by Russell and Norvig (2021a: 19) defines the field of AI as being “concerned with not just understanding but also building intelligent entities - machines that can compute how to act effectively and safely in a wide variety of novel situations”. This definition provides a brief explanation of what artificial intelligence involves. Furthermore, AI can be explained to be the use of computers to mimic the performance of human capabilities such as problem-solving and decision making (IBM Cloud Education 2020a). The field of AI can be seen to encompass the sub-fields of machine learning (ML) and deep learning (IBM Cloud Education 2020a). A more detailed definition of machine learning will be provided later.

AI in the modern world is being applied across many different applications, business sectors, industries and aspects of modern-day living. A brief explanation of some of these AI applications will be provided. AI is being applied in the e-commerce industry to create personalised shopping experiences with recommendation systems to recommend services and products to customers whilst virtual assistants and chatbots are utilising natural language processing to improve and make personable online shopping for users (Biswal 2021). AI can also be seen to have applications within the education sector, as AI technology is being used to automate administrative tasks so educators can focus on teaching, whilst also being used to develop personalised learning for students (Biswal 2021). AI is also having a large impact on modern lifestyles (Biswal 2021). AI has been applied for the creation of commercially available partially autonomous vehicles by many car manufacturers with the use of machine learning (Biswal 2021). Further AI applications on modern lifestyles can be seen with increased security measures with the use of facial recognition systems that recognise the user by identifying them via their face on devices such as smartphones and computers (Biswal 2021). AI can even be seen to be applied in the agricultural industry, through the use of computer vision, robotics, and machine learning applications (Biswal 2021). AI is able to identify deficiencies in soil, identify where weeds are growing and improve the yield of crop a farmer can produce (Biswal 2021). This list of AI applications is not exhaustive, and the use of AI can be seen in other business sectors and industries as well. However, most importantly for this literature review, it can be identified that there are a great number of applications of AI in the healthcare sector also (Biswal 2021).

2.2 Artificial Intelligence in Healthcare

The application of AI in healthcare can be seen to changing the medical field (Secinaro et al 2021). Researchers identify in the literature that the application of Artificial Intelligence in healthcare can be seen to be applied to four main fields which are: health service management, patient data and diagnostics, clinical decision making and predictive medicine (Secinaro et al 2021). The medical field currently has AI applications that are utilised in predictive, rehabilitative, surgical, clinical and disease diagnostic practices (Secinaro et al 2021). A brief explanation of some of these applications will be provided. Within healthcare, large volumes of medical data are collected and by utilising this data with machine learning techniques and data mining, solutions to a variety of complex medical problems can be developed (Secinaro et al 2021). It is this methodology that is used in the development for a lot of the AI applications that will be described below.

AI can be applied to provide clinical decision support to doctors as they can make critical decisions regarding patients faster and better due to AI's ability to recognise patterns of health complications (Arsene 2021). Furthermore, AI can be applied to create personalised medicine treatments for patients by performing analysis on patient data (Arsene 2021). AI is also assisting the process of drug discovery for pharmaceutical companies as it assists with discovering new drugs and shortening process times for the delivery of said drugs to the market (Arsene 2021). A relatively new application of AI can be seen with its use for patient pain management (Arsene 2021). By combining virtual reality and AI technologies, simulation of realities can be created that can be used as a source of distraction for patients from their pain and also decrease the reliance of opioids for pain relief (Arsene 2021). The development of clinical AI health applications can be seen to be valuable as it can be seen to be cost effective. This is exemplified by the potential \$150 billion annual saving the United States health economy could potentially make by the year 2026 due to AI health applications (Accenture 2021). Accenture identify the top 10 near-term impact AI applications that could provide this saving as: automated image diagnosis, preliminary diagnosis, clinical trial participant identifier, connected machines, dosage error reduction, fraud detection, administrative workflow assistance, virtual nursing assistants and robot-assisted surgery (Accenture 2021). These AI applications are in development for future impact, however there is a current AI application which is the most relevant and important application of AI in regard to this project. This is the use of AI to perform medical diagnostics and diagnose patients with specific diseases (Arsene 2021). The manner of how this is done, and examples of disease prediction will be described further below.

Disease prediction is commonly done with the use of supervised machine learning algorithms (Uddin et al 2019). Machine learning can be explained as the utilisation of computer algorithms that can learn and improve their performance by analysing input data in order to make predictions (Uddin et al 2019). Typically, these algorithms improve their accuracy of prediction by the introduction of more input data (Uddin et al 2019). Machine learning algorithms can be categorised into one of three categories by the type of learning they perform (Russell and Norvig 2021b). These are: supervised learning, unsupervised learning, and reinforcement learning (Russell and Norvig 2021b). Supervised machine learning algorithms work by training the algorithm using a dataset of labelled training data (Uddin et al 2019). Once trained, the algorithm is then tested against unlabelled test data (Uddin et al 2019). Supervised machine learning algorithms are used

for classification and regression problems (Uddin et al 2019). With classification problems, the output variable is discrete and therefore can be categorised into classes or categories for e.g., diabetic, or non-diabetic (Uddin et al 2019). With regression problems, the output variable is a number for e.g., the risk of a person developing cardiovascular disease (Uddin et al 2019). The commonly used supervised machine learning algorithms for disease prediction are: Logistic regression (LR), Support vector machine (SVM), Decision tree (DT), Random forest (RF), Naïve Bayes (NB), K-nearest Neighbour (KNN) and Artificial Neural Networks (ANN) (Uddin et al 2019). Within the literature, researchers utilise the use of these supervised learning algorithms for the prediction of different diseases.

In the literature, a variety of different diseases or the risks associated to them can be seen to be predicted by using supervised machine learning algorithms.

Islam et al (2018) present in their paper, the utilisation of machine learning algorithms to predict fatty liver disease. The researchers utilised the classification algorithms of RF, ANN, SVM and LR to build a predictive model for fatty liver disease (Islam et al 2018). The predictive model which utilised the LR algorithm presented the highest accuracy of prediction, at 0.707, in comparison to the other algorithms used (Islam et al 2018). Islam et al (2018) express the use of this predictive model for early prediction may assist clinicians with the diagnosis and treatment of fatty liver disease.

Ahmad et al (2013) present in their paper, predictive models to predict the recurrence rate of breast cancer in patients within two years. In this paper, the researchers utilised the SVM, ANN and DT algorithms to predict the recurrence of breast cancer (Ahmad et al 2013). The researchers identify the SVM prediction model as the best from the three models trained (Ahmad et al 2013). This is supported by the fact this model could be seen to have the highest accuracy of prediction of the three, with an accuracy of 0.957 (Ahmad et al 2013).

Patel et al (2013) present in their paper, the prediction of the diagnosis of heart disease using a reduced number of features and classification algorithms. Heart disease is used as a broad term to define a variety of conditions which affect the heart and blood vessels such as: coronary heart disease, congenital heart disease and congestive heart failure (Patel et al 2013). The researchers utilised supervised ML in the form of the classification algorithms of DT and NB to perform diagnosis of heart disease (Patel et al 2013). Of the classification algorithms used, the DT algorithm has the highest prediction accuracy at 99.2% (Patel et al 2013). However, training of a model using the DT algorithm took the longest in comparison to the other classification algorithms used (Patel et al 2013). This is an example of how diagnosis of heart disease can occur using supervised machine learning.

From the literature described, we can see that supervised machine learning algorithms can predict diseases such a fatty liver disease, heart disease, breast cancer and its recurrence in patients. These examples are not an exhaustive list of disease prediction, as prediction of different diseases to those mentioned can be seen in the literature. However, these examples do show that there is variation amongst the performance of supervised ML algorithm when performing disease prediction. This can be seen as researchers try to perform prediction using multiple supervised ML algorithms and compare their performance via assessment criteria such accuracy of prediction. Therefore, careful selection of the best suited supervised ML algorithm will be required for the training of the model of the Disease Prediction Android Application in order to attain a high accuracy of prediction.

2.3 Comparison of supervised ML algorithms

As mentioned, the commonly used supervised machine learning algorithms for disease prediction are: Logistic regression (LR), Support vector machine (SVM), Decision tree (DT), Random forest (RF), Naïve Bayes (NB), K-nearest Neighbour (KNN) and Artificial Neural Networks (ANN) (Uddin et al 2019). As demonstrated, the scientific literature also provides examples of how these algorithms have been used to predict different specific diseases. To assist with the selection of the supervised machine learning algorithm used for the application it is useful to understand the advantages and limitations of each of these algorithms.

Table 1 – Advantages and limitations of different supervised machine learning algorithms from Uddin et al (2019: 12)

Supervised algorithm	Advantages	Limitations
Artificial neural network (ANN)	<ul style="list-style-type: none"> - Can detect complex nonlinear relationships between dependent and independent variables. - Requires less formal statistical training. - Availability of multiple training algorithms. - Can be applied to both classification and regression problems. 	<ul style="list-style-type: none"> - Have characteristics of black box – user can not have access to exact decision-making process and therefore, - Computationally expensive to train the network for a complex classification problem. - Predictor or independent variables require pre-processing.
Decision tree (DT)	<ul style="list-style-type: none"> - Resultant classification tree is easier to understand and interpret. - Data preparation is easier. - Multiple data types such as numeric, nominal, and categorical are supported. - Can generate robust classifiers and can be validated using statistical tests. 	<ul style="list-style-type: none"> - Require classes to be mutually exclusive. - Algorithm cannot branch if any attribute or variable value for a non-leaf node is missing. - Algorithm depends on the order of the attributes or variables. - Do not perform as well as some other classifiers (e.g., Artificial Neural Network) (Atlas et al 1990).
K-nearest neighbour (KNN)	<ul style="list-style-type: none"> - Simple algorithm and can classify instances quickly. - Can handle noisy instances with missing attribute values. - Can be used for classification and regression 	<ul style="list-style-type: none"> - Computationally expensive as the number of attributes increases. - Attributes are given equal importance which can lead to poor classification. - Provide no information on which attributes are most effective in making a good classification.
Logistic regression (LR)	<ul style="list-style-type: none"> - Easy to implement and straightforward. - LR-based models can be updated easily. - Does not make any assumptions regarding the distribution of independent variable (s). - It has a nice probabilistic interpretation of model parameters. 	<ul style="list-style-type: none"> - Does not have good accuracy when input variables have complex relationships. - Does not consider the linear relationship between variables. - Key components of LR – logic models, are vulnerable to overconfidence. - May overstate the prediction accuracy due to sampling bias. - Unless multinomial, generic LR can only classify variables that have two states (i.e., dichotomous)

Naïve Bayes (NB)	<ul style="list-style-type: none"> - Simple and very useful for large datasets. - Can be used for both binary and multi-class classification problems - It requires less amount of training data. - It can make probabilistic predictions and can handle both continuous and discrete data. 	<ul style="list-style-type: none"> - Classes must be mutually exclusive. - Presence of dependency between attributes negatively affects the classification performance - It assumes the normal distribution of numeric attributes.
Random forest (RF)	<ul style="list-style-type: none"> - Lower chance of variance and overfitting of training data compared to DT, since RF takes the average value from the outcomes of its constituent decision trees. - Empirically, this ensemble-based classifier performs better than its individual base classifiers i.e., DTs. - Scales well for large datasets. - It can provide estimates of what variables or attributes are important in the classification. 	<ul style="list-style-type: none"> - More complex and computationally expensive - Number of base classifiers need to be defined. - It favours those variables or attributes that can take high number of different values in estimating variance importance. - Overfitting can occur easily.
Support vector machine (SVM)	<ul style="list-style-type: none"> - More robust compared to LR - Can handle multiple feature spaces. - Less risk of overfitting. - Performs well in classifying semi-structured or unstructured data such as texts, images etc. 	<ul style="list-style-type: none"> - Computationally expensive for large and complex datasets. - Does not perform well if the data have noise - The resultant model, weight and impact of variables are often difficult to understand, - Generic SVM cannot classify more than two classes unless extended.

Using Table 1 (Uddin et al 2019:12) more insight is gained to help select the supervised machine learning algorithm for the application. The application will allow a user to predict the disease they may have, from a number of diseases. Therefore, as one class (a single disease) is being predicted from multiple classes (of diseases), the application will be performing multiclass classification. Critically, from Table 1 we can see that ANN, KNN and NB can all be used for classification problems. One of these may be potentially selected to be used for the application. However, NB is specifically mentioned to be a supervised machine learning algorithm that can perform multiclass classification in Table 1. Therefore, it will be the first out of the three algorithms to be inspected to see if it is suitable to use for the application. It should also be noted that Table 1 specifically mentions the generic form of SVM algorithm is not able to perform multi-class classification. Therefore, the generic form of this algorithm will definitely not be selected to perform prediction in the application. LR will also be not selected as it is suited to regression problems. Table 1 also informs us that issues may arise using ANN as it can be seen to be computationally expensive when training its network for classification problems that are complex. Seen as the classification problem is a multiclass classification problem it could be deemed as a complex classification problem. Table 1 shows a limitation of KNN which is that KNN may not be suitable as give it attributes an equal importance when being trained and this can result in poor classification. The attributes in a classification problem that are being addressed here refer to symptoms in the case of the application. In reality, it may be that a certain symptom for e.g., sneezing is more indicative to a specific disease such as influenza and therefore should not carry an equal importance to a different symptom such

as a cough. This shows another potential issue that must be considered before selecting the supervised machine learning algorithm that will be used for the application. If ANN, KNN and NB are deemed not suitable, the remaining algorithms of DT and RF will be inspected to see if they are suitable.

The supervised machine learning algorithm that will be selected for use to train the model for inference in the Disease Prediction Android application, will be explained and justified in the 'Chapter 5: Implementation and Testing' chapter of this report.

2.4 Technologies

To develop the Disease Prediction Android Application various technologies must be utilised. This section of the literature review will describe which technologies and programming languages have been selected for the development of the Disease Prediction Android Application.

Android Studio IDE – The Disease Prediction application is a mobile application for smartphone's utilising the Android operating system. Therefore, the development of this application will be done utilising Android Studio as the integrated developer environment (IDE). Android studio is the official IDE for android application development, and it provides useful features such as: an intelligent code editor, an android device emulator for system testing and in-built integration for external cloud-based services such as Firebase (Android for Developers n.d. a).

Java – There are multiple programming languages that can be used for the development of Android applications. These languages are Java, Kotlin.C++ and C# (Sims 2019). Both Java and Kotlin are recognised as official languages of Android development (Sims 2019). However, Java has been selected for the development of this application as I have a clear understanding of the language syntax and the use of Object Orientation Principles (OOP) within the language. Therefore, selection of this programming language should assist in being able to successfully develop the application.

Firebase – Authentication and database functionalities will be done utilising Firebase. Firebase is a cloud-based backend platform that is used to build iOS and Android applications by providing a variety of different backend services which are hosted in the cloud (Stevenson 2018). Some of the many different services that the Firebase platform provide are a realtime database, authentication, hosting, cloud Storage and a ML Kit (Firebase n.d. a). Applications can connect to these services by using client SDK's (Stevenson 2018). User login and saving/viewing predicted disease results will be performed by utilising Firebase's authentication and realtime database services respectively. The realtime database is a cloud-hosted NoSQL database, that allows for data syncing and storage in real time (Firebase n.d. b).

To perform ML in the Disease Prediction Android application different methods can be used. Inference is the process of using a trained ML model to perform a task (Android for Developers n.d. b). In the case of the Disease Prediction Android application this would be the process of using the model to predict which condition the user has after the inputting of symptoms by user input into the application. The process of inference can be done on-device or via a cloud-based service such as Firebase's ML Kit. Therefore, a decision was

made about which method of inference and subsequently which technologies will be utilised to perform ML in the application.

Table 2 – A comparison of on-device and cloud-based inference (Android for Developers n.d. b)

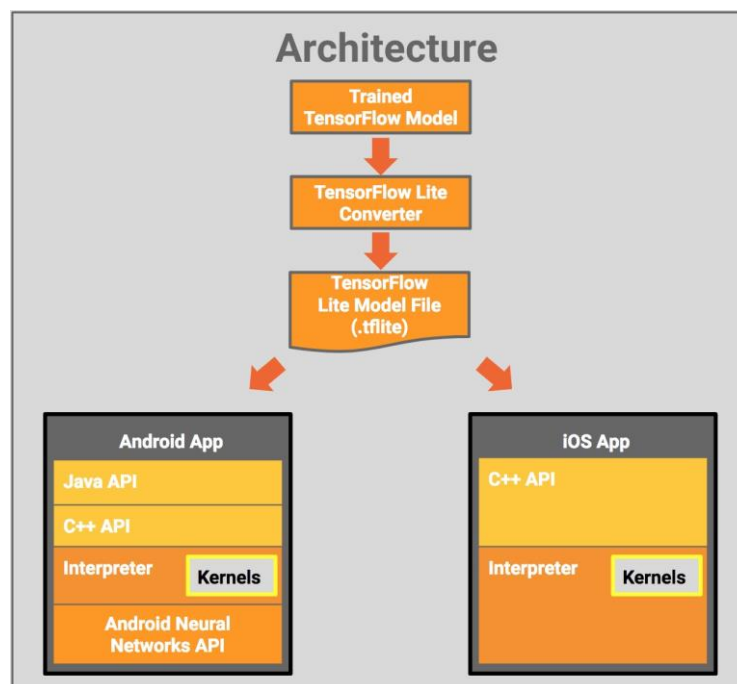
Issue	Cloud-based inference	On-machine based inference
Privacy	User data leaves device which may cause privacy issues.	Data doesn't leave the device at all.
Resources	Cloud-based resources are powerful, and storage is vast.	The devices computational power and storage may be limiting factors for performance.
Offline/Online	Cannot perform offline as a network connection is required.	Can perform offline which is advantageous if intended application is to be used where there is poor or no network access.
Cost	There is cost in the form of bandwidth for data transfer between the cloud/device for users and computing charges for the developers	There is cost in the form of battery usage and inference run time for the users
Latency	Higher latency due to limiting factors such as available bandwidth and out of sync communication with the cloud	Latency is lower than cloud-based inference in comparison and therefore provides a better real time experience

Initially, the use of cloud-based inference seemed an ideal approach due to the increased computational power provided by the cloud as shown in Table 2. The application is intended for a user be logged in to use it via the Firebase backend services. Therefore, it must be used with a network connection regardless and it would be okay if inference could only be performed if the device is online via cloud-based inference. However closer inspection of different cloud-based inference ML services has changed the selected approach. Many of the cloud-based inference services are for ML to be applied for common use cases in mobile applications. For example, the Firebase ML Kit, provides APIs to perform text image labelling, text recognition and landmark recognition (Firebase n.d. c). Another cloud-based inference service, Google Cloud provides APIs for ML applications such as image labelling, explicit content detection, speech recognition, natural language processing, translation and more within a mobile device (Google Cloud n.d.). The intended application of ML in the Disease Prediction application does not fit into any of the applications provided by APIs of cloud-based ML services. Therefore, the approach to performing ML in the application has changed and will be described below. Instead, we will utilise on-machine based inference. This process allows us to train a custom ML model that will allow for multi-class classification in order to predict a user's disease.

On-device inference does provide benefits such as its reduced latency and increased privacy as user data doesn't leave the device (Android for Developers n.d. b). From a user perspective, these benefits will provide the user with a greater user experience using the application. Below we will define the process required to do on-device inference for the Disease Prediction Android application.

TensorFlow is an open-source platform for machine learning that provides tool and libraries to build machine learning applications (TensorFlow n.d.). By using TensorFlow and its API's such as Keras, machine learning models can be built and trained (TensorFlow n.d.). The model for the Android application will be built and trained using Python and TensorFlow libraries on a development machine initially. This will be done using PyCharm, a Python integrated development environment (Jetbrains n.d.) TensorFlow Lite Converter, a TensorFlow class, can be used to convert the trained model into TensorFlow Lite model and specifically a '.tflite' file (TensorFlow 2021a). Once converted into a TensorFlow Lite model, the model as a '.tflite' file can be used as an asset within the Android application. Using the TensorFlow Lite model on-device in order to make predictions based on input data is inference and must be done through an interpreter (TensorFlow 2021b). Within an Android application, TensorFlow Lite inference can be done using either Java or C++ APIs (TensorFlow 2021b). So therefore, in the intended application, Java APIs for TensorFlow Lite inference will be utilised as an interpreter and this will allow the use of the TensorFlow Lite model to predict disease from user symptom input on an Android mobile device.

Figure 1 – Architecture required to use TensorFlow model in mobile applications from Moroney (2018)



3 Chapter 3: Requirements and Analysis

This chapter provides a detailed review of the requirements needed to be fulfilled for the development of the Disease Prediction Android Application. It also provides a description of the methods of evaluation and testing that will be done.

3.1 Project Description

This project description is an updated description from that mentioned in section 1.2 by mentioning the technologies which will be used as mentioned in section 2.4 of this report.

This project aims to provide accessibility to as many users/patients as possible with the development of an Android application that can be used for immediate diagnosis. The application will be developed to allow the user to be able to register to the application and login to the Disease Prediction Android Application. The ability to register and login will be provided with the use of Firebase Authentication. Therefore, a user will have individualised experience once logged into their account. The Android application will be developed utilising Android Studio and Java. It will utilise a trained machine learning model. First a dataset of symptoms and corresponding disease will be acquired and used to train a machine learning algorithm in order to build a model. The model will be built using TensorFlow and Python on a development machine. The model will then be incorporated within the Android application as a TensorFlow Lite model. The use of this model will allow inference, this is the process of using the model to predict which condition the user has after the inputting of user symptoms by user input into the application. To provide further assistance to the user alongside the result that appears, a link to a healthcare organisation's webpage relating to the condition they are predicted to have will be shown to the user. The user can open the link to receive further care and advice. The application will also allow the user to save their results which will be stored in the application via the Firebase Realtime database. Therefore, through the process of the application making connection to the Firebase Realtime database, the user can look at previous conditions the application predicted for the user. Furthermore, the application will be able to locate the nearest Hospital/GP practice/Pharmacy to the user and display it on a map. Overall, the application will be very useful and assistive in identifying a patient's condition and directing them with advice about their condition via healthcare organisation webpages or by referring to them to their nearest healthcare practitioner.

3.2 Functional Requirements

The functional requirements state what the Disease Prediction Android Application will be able to do in terms of features and functions. They are defined below as:

- Register – A user will be able to register an account to the application with an email address and a password.
- Login – A user will be able to get access to the features of the application upon logging in to their account with their credentials they used for registration.
- Logout – A user will be able to logout from their account and subsequently will not have access to the applications features.
- Symptom input - A user will be able to input up to seventeen symptoms with the use of drop-down menus to be used for disease prediction. Four symptoms are mandatory required from the user to perform disease prediction.
- Predict disease – A user will be able to gain a prediction of the disease/condition they have as the application uses a trained TensorFlow Lite model to do inference upon the symptoms they input and provide a prediction.
- View information about disease/condition – A user will be able to click a view more button that will take them to a healthcare organization's webpage which provides further information about the disease/condition the application predicted they have.

- Save result - A user will be able to save the result of their disease prediction and the date and time they acquired this prediction. This will be stored in the Firebase Realtime database.
- View past results – A user will be able to view past results for disease predictions they chose to save, as well as the date the prediction was done, as this information is retrieved from the Firebase Realtime database and presented to the user.
- Find nearby healthcare practitioner – A user will be able to find a nearby Hospital/GP practice/Pharmacy/Opticians to their location and this will be displayed to the user on a map.

3.3 Non-functional Requirements

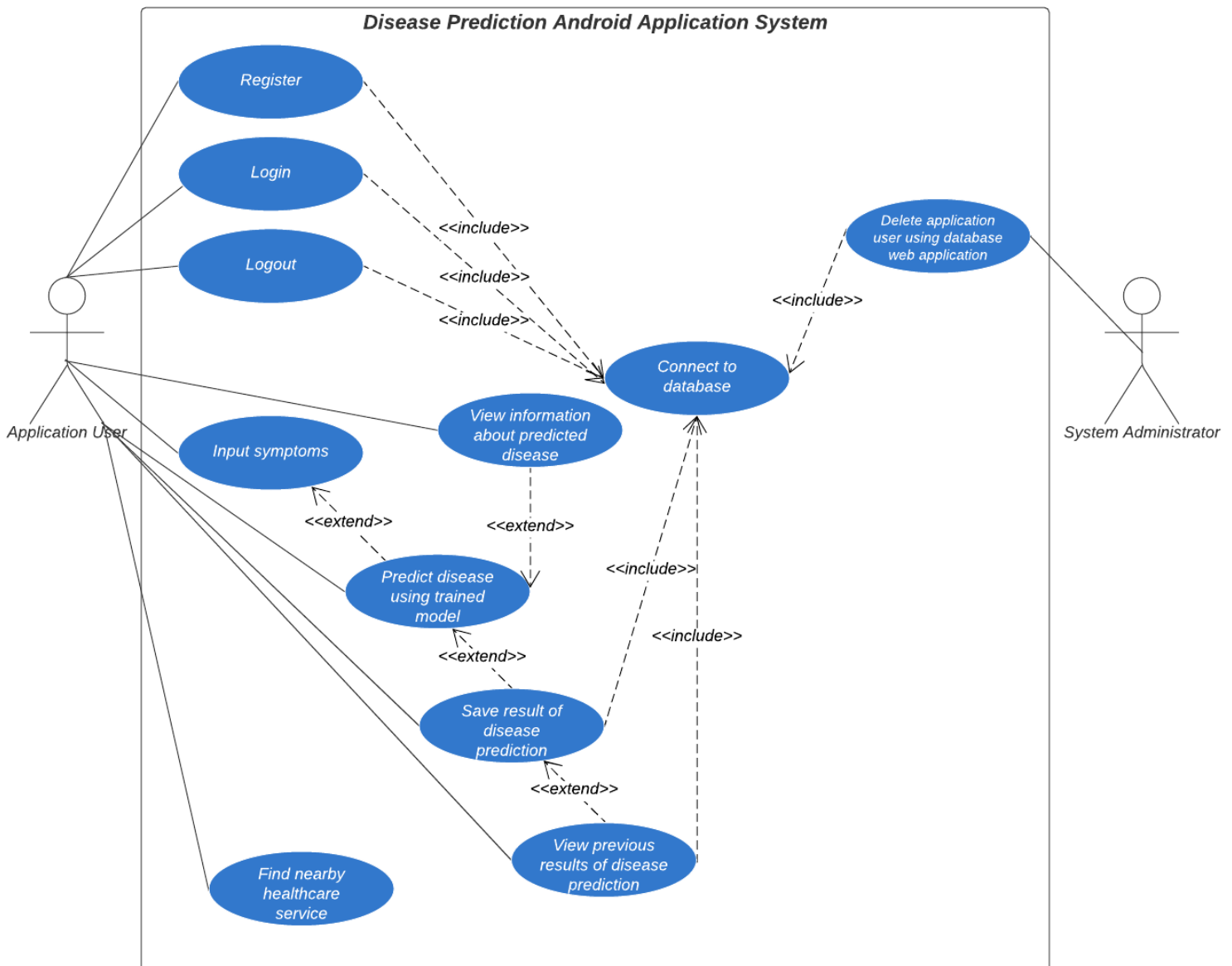
The non-functional requirements will describe the general properties of the Disease Prediction Android application and how the application should behave. They are noted as:

- Reliability – The application should be deemed reliable with the disease prediction results it provides. The TensorFlow Lite model used to predict results, must be able to predict to disease at a very high accuracy for the results the application provides to be deemed trustworthy and reliable.
- Usability – The application must be able to provide clear navigation between all its features so that a user can easily access them and therefore have a good user experience using the application. The styling for the user interface throughout the application must stay consistent in terms of color schemes, fonts, and layouts to also provide a good user experience of the application.
- Security - Confidentiality is maintained by requiring the user to register an account to the application with fewest fields possible in the form of an email address and password. Therefore, a user is only identifiable by their email address and a user identifier. Firebase generates per user a string made up of mixed characters and numbers to be a user identifier. The application developed reduces the risks and mitigates the damage done of any security breaches to it or the back-end services it utilizes with Firebase such as Firebase Authentication and the Firebase Realtime database. If a data breach is to occur and the results of disease predictions are leaked, the information is not personally identifiable as users registered to the application with just their email address and password. The saved results of disease predictions per user cannot be interpreted in a manner that personally identifies any user.
- Performance – The application must be able to provide all its features at runtime without issues and crashing. This will be assured through unit testing in development and by performing system testing once the application is completed.

3.4 UML Use Case Diagram

The diagram below, Figure 2, is the UML use case diagram for the Disease Prediction Android Application as a system. As denoted by the diagram we can see that there are two actors and eleven use cases of the system. The two actors that interact with the system are noted as the Application User and the System Administrator. It should be noted that relationships exist between use cases denoted in the diagram. The extend relationship shows the dependency of a use case from the base use case it is extended from (Nishadha 2021). The extended use cases can be seen to be optional functionalities of the overall system (Nishadha 2021). For the include relationship, a use cases behaviour is included as part of the base use case's behaviour (Nishadha 2021). The included use cases can be seen to be mandatory for the base use case to be functional (Nishadha 2021).

Figure 2 – UML Use Case diagram for the Disease Prediction Android Application



3.5 Evaluation and Testing

This section describes how the implementation of the technical solution will be evaluated and tested during and after development.

The evaluation for the performance of the trained model will be done on the development machine. By utilising TensorFlow's inbuilt libraries, performance of the trained model can be measured. One of these performance metrics is accuracy. The selected model will be trained multiple times and a model with a prediction accuracy above 95% will be utilised as the asset for the Android application. The accuracy of the trained model will be presented in 'Chapter 5: Implementation and Testing' chapter of the report.

Where applicable in the technical solution, unit testing will be performed. Unit testing can be performed within Android Studio using the testing framework JUnit (Android for Developers n.d. c). By using this testing framework, test classes can be written that have test methods (Android for Developers n.d. c). The test classes will be submitted in the final technical solution and when ran in the Android Studio IDE will show how many test cases have passed or failed per testing class.

Furthermore, system testing will be done. System testing will be done to evaluate the performance of the Disease Prediction Android Application as a complete system. System testing is done in an end-to-end manner to test the functionalities of a system (tutorialspoint n.d.). This form of testing will be measured against the requirements specified in section 3.2 of this report. This form of testing and its results will be presented in 'Chapter 5: Implementation and Testing' of the final report.

4 Chapter 4: Design

This chapter of the report details how the Disease Prediction Android Application was designed. This section of the report is split into two sections. Initially, the project's folder structure in Android Studio is explained as it provides clarity on how the application was designed. Secondly, since the application is programmed in Java which is an Object Orientated programming language the project can be seen to have a class-based structure for its Java source code. Due to this, a UML Class diagram will be provided to detail the architecture and components of the application. It will do so by presenting the classes, variables, and methods of the code.

4.1 Project Folder Structure

The project folder structure for the Disease Prediction Android application can be split into five components. Those components are the: manifests folder, Java folder, assets folder, resources folder and Gradle scripts. An explanation of each will be provided below.

Manifest folder – The manifest folder contains the AndroidManifest.xml file which details many things about the application such as its android version, metadata, and other application component details (GeeksforGeeks 2022). It acts as the broker between the Android OS of a device and the application for e.g., the developer can define explicitly which page (activity) the application should open upon launch within this file (GeeksforGeeks 2022).

Java folder – This folder contains all the Java source code. The source code is where the functionality for the application is implemented.

Assets folder – This folder contains the diseasepredictionmodel.tflite file. This file contains the trained TensorFlow Lite model that will be used to perform on-device inference in order to predict disease based on user symptom input.

Resources folder – This folder contains aspects of the application such as images and xml files (GeeksforGeeks 2022). The xml files found within the layout folder contain xml markup that defines the user interface of the application (GeeksforGeeks 2022).

Gradle scripts – This folder contains files that define the build configuration of the application (GeeksforGeeks 2022). The build.gradle (Project) contains build scripts whilst build.gradle (Module) is where a user can define plugins and dependencies for the application (GeeksforGeeks 2022).

Figure 3 - The manifest, java, and assets folder of the Disease Prediction Android application

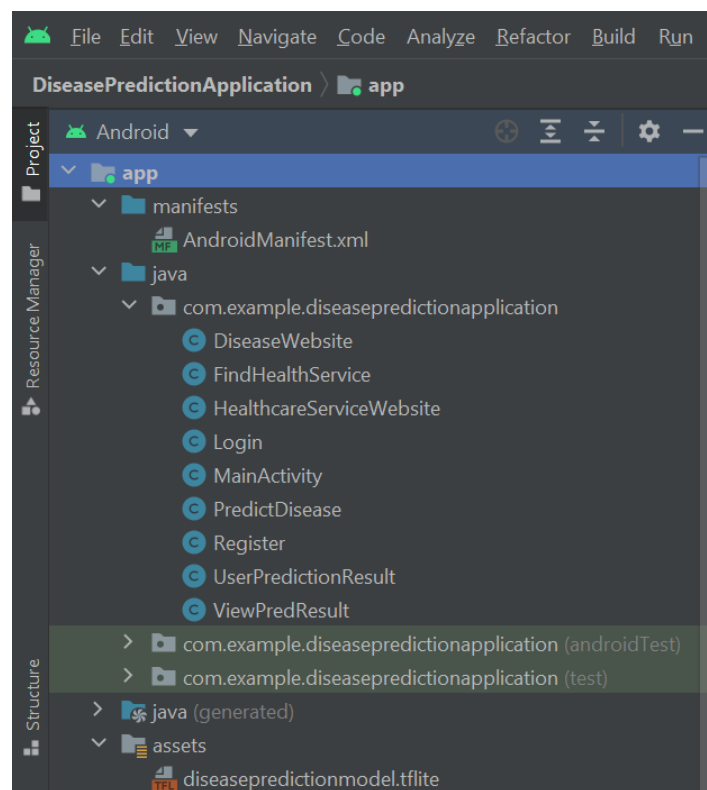


Figure 4 - The resources folder and Gradle scripts of the Disease Prediction Android application

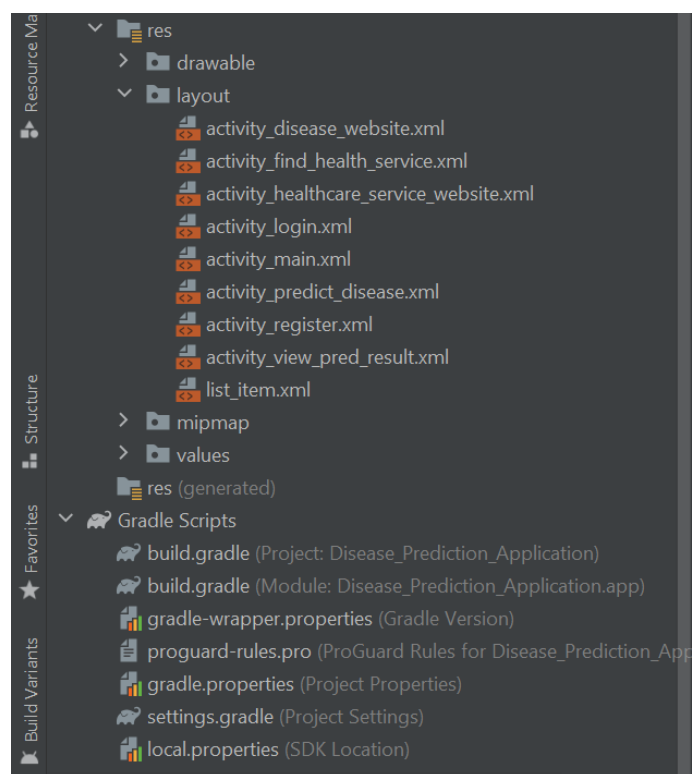
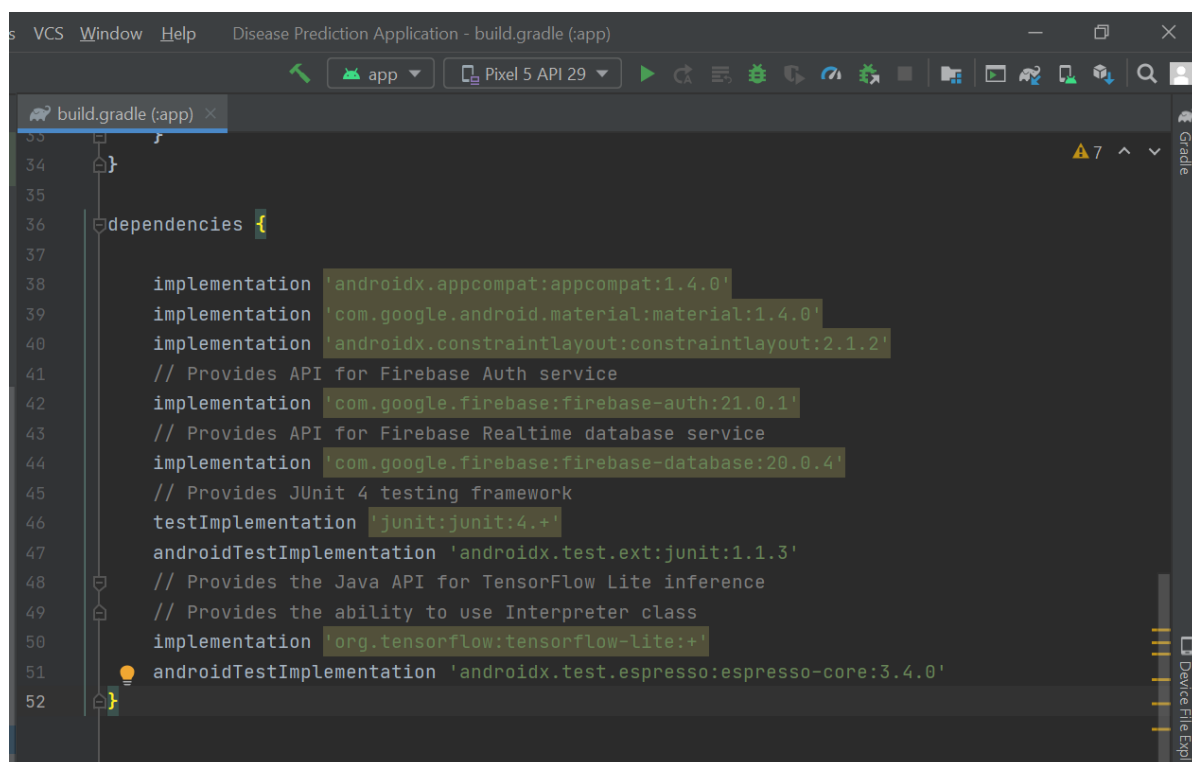


Figure 5 – Firebase Auth, Firebase Realtime database and TensorFlow Lite dependency declarations to provide API access in project

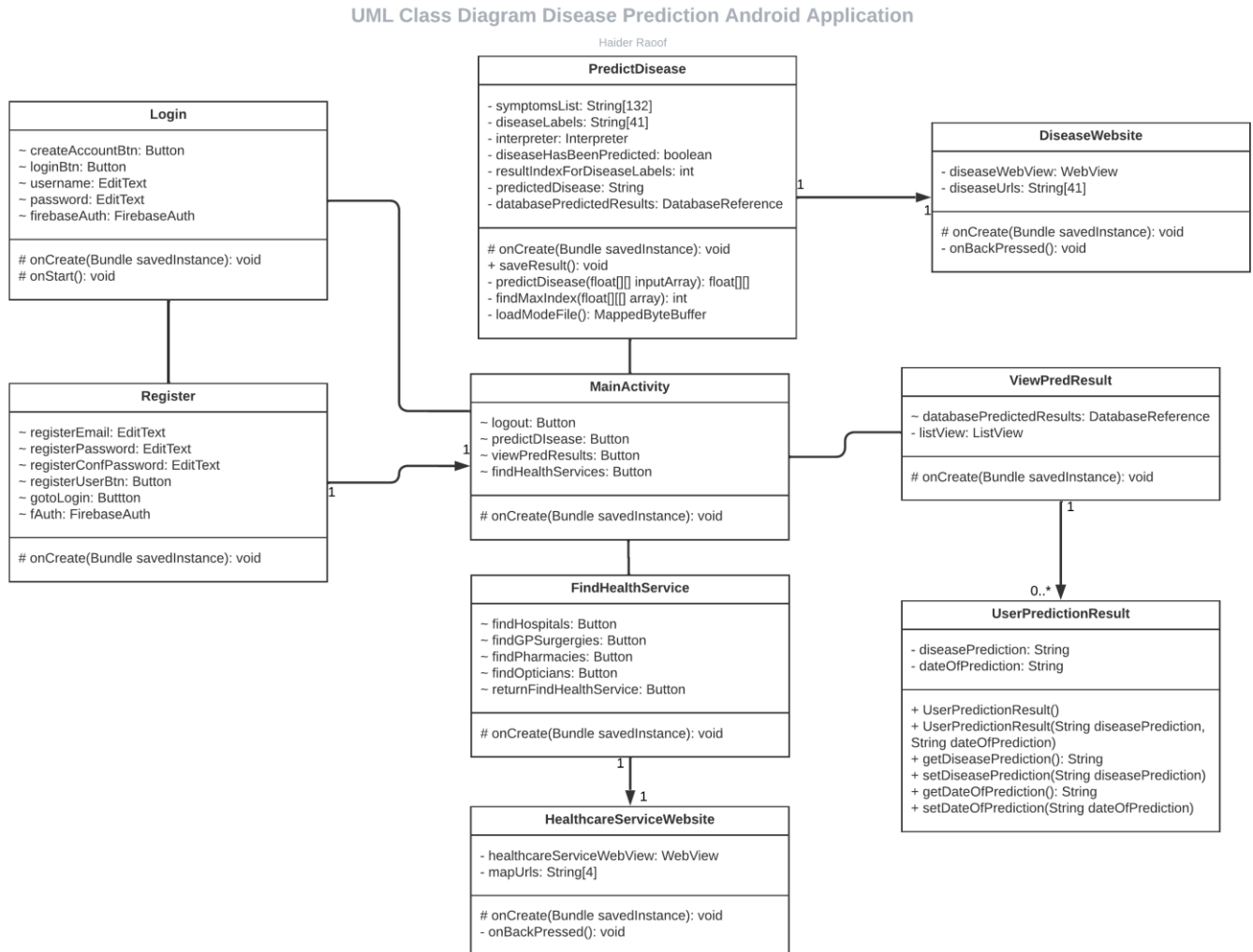


The lines `implementation com.google.firebase:firebase-auth:21.0.1` and `implementation com.google.firebase:firebase-database:20.0.4` are the Gradle dependencies for the Firebase Authentication and Firebase Realtime database services respectively. The declaration of the lines provides the APIs to interact with these services in my implementation. Android Studio has Firebase integration, so these dependencies are automatically declared once a Firebase service such as Firebase Authentication or the Firebase Realtime database are enabled and configured for a project. However, in an effort to provide properly cite these APIs, they can be seen to be made publicly available via Firebase's website (Firebase n.d. d). The line `implementation org.tensorflow:tensorflow-lite` is the Gradle dependency that provides access for the use of the Java API for TensorFlow Lite inference (TensorFlow 2021b). Using this API allows the use of the Interpreter class of this API in order to perform on-device inference (TensorFlow 2021b). Furthermore, the line `testImplementation junit:junit:4.+` plus provides the Junit 4 testing framework which can be used in Android Studio to perform unit testing (Android for Developers n.d. c) This section of the report was included to provide proper accreditation to the sources of these APIs which facilitated the implementation and testing of the application.

4.2 Architecture and components

A UML Class diagram will be provided here to denote the architecture and components of the application. This diagram will present the classes, variables and methods for the Java source code which implemented the functionality of the application.

Figure 6 – UML Class diagram for the Disease Prediction Android Application



5 Chapter 5: Implementation and Testing

5.1 Dataset Selection and Pre-processing

The dataset used to perform supervised machine learning is the ‘Training.csv’ dataset (Singh 2019). The dataset contains patient data, which details the symptoms a patient presented and the associated prognosis of disease per patient sample (Singh 2019). The source of the data provides two datasets in the form of ‘Training.csv’ and ‘Testing.csv’ however the Training.csv file provides substantially more rows of data in comparison to the Testing.csv file. Therefore the Training.csv file has been selected to train the selected supervised machine learning algorithm. The selected supervised machine learning algorithm will be stated in section 5.2 of this report.

The TensorFlow Lite model used for the Disease Prediction Android Application has been built, trained, and tested using a Python script named 'predictionmodel.py' (see Appendix C). In order to make the dataset suitable for the selected supervised ML algorithm to be trained and tested upon, pre-processing of the dataset was performed. A description of the data pre-processing will be provided below. It should be noted, the Python script utilises Python libraries from external sources. It utilises the pandas library which facilitates the storage of the dataset as a DataFrame object and handling of the data via this object (Pandas n.d.). It utilises the NumPy library which facilitates the use of NumPy arrays (NumPy n.d.). It also utilised the machine learning library scikit-learn for processing tools for the data (Scikit-learn n.d. a). It should also be noted that Anaconda was used to establish a conda environment to help set up this predictionmodel.py file to be able to run using TensorFlow 2.0 in PyCharm (Anaconda n.d.).

The following was done. A check was performed to see if any null values exist within the dataset. This returned false, so no processing of the dataset was done in regard to this. The dimensions of the dataset were observed to be 133 columns and 4020 rows. The 'Training.csv' file was observed in Microsoft Excel (Microsoft Excel n.d.). This was done to understand the dataset better. The first row presents the column headings. Therefore, there are 4019 records of patient data in this dataset. The first 132 columns each denote a symptom whilst the 133rd column is the 'prognosis' column that shows the disease an instance of a patient record has. The first 132 columns per instance are represented using nominal data. The presence of a symptom is denoted with a value of 1 and the absence of a symptom with the value of 0. The 133rd column is represented using categorical data in text format that presents the disease for that instance. A check was done to see the number of unique disease values in the dataset's 'prognosis' column. This returned a value of 41 to show there are 41 distinct disease values in the dataset. A check was then done to see number of duplicated rows there are in this dataset which showed 4616 rows to be duplicates. Duplicated rows were dropped whilst keeping the first occurrence of a row. This left a remaining 304 rows of records in the dataset.

The 'Testing.csv' in Microsoft Excel was also observed. This also has the same dimensions of 133 columns as the 'Training.csv' file. The first 132 being symptom columns and the 133rd being a 'prognosis' column. There are 41 records of patient data in this dataset. Each had a distinct disease value for the prognosis column. From this, the values of the 41 distinct disease values in the 'Training.csv' dataset were inferred.

Using the pandas' replace function, label encoding was performed to encode the prognosis column of the dataset. This was done as this column is not in a suitable format to predict disease with the selected trained supervised machine learning algorithm. Label encoding is the process of replacing categorical text data with numerical values (GreatLearn 2019). In the case of this dataset, numerical values from 0 to 40 denote a disease/label/class. A screenshot of how this was performed is shown in the figure below.

Figure 6 – Label encoding of the ‘prognosis’ column of the dataset in the Python script using pandas replace function.

```

12 # Use the pandas replace function to label encode the prognosis column of dataset - found all the disease names using
13 # the Testing.csv file provided from source of data
14 dataset.replace({'prognosis':{'Fungal infection':0, 'Allergy':1, 'GERD':2, 'Chronic cholestasis':3,
15 'Drug Reaction':4, 'Peptic ulcer disease':5, 'AIDS':6, 'Diabetes':7, 'Gastroenteritis':8, 'Bronchial Asthma':9,
16 'Hypertension':10, 'Migraine':11, 'Cervical spondylosis':12, 'Paralysis (brain hemorrhage)':13, 'Jaundice':14,
17 'Malaria':15, 'Chicken pox':16, 'Dengue':17, 'Typhoid':18, 'hepatitis A':19, 'Hepatitis B':20, 'Hepatitis C':21,
18 'Hepatitis D':22, 'Hepatitis E':23, 'Alcoholic hepatitis':24, 'Tuberculosis':25, 'Common Cold':26, 'Pneumonia':27,
19 'Dimorphic hemmorhoids(piles)':28, 'Heart attack':29, 'Varicose veins':30, 'Hypothyroidism':31, 'Hyperthyroidism':32,
20 'Hypoglycemia':33, 'Osteoarthritis':34, 'Arthritis':35, '(vertigo) Paroymsal Positional Vertigo': 36, 'Acne':37,
21 'Urinary tract infection':38, 'Psoriasis':39, 'Impetigo':40}}, inplace=True)

```

By using pandas pop function, the labelled column was separated from the dataset to create a labelled dataset. Therefore, two datasets were created, the dataset with the 132 features/symptom columns and the dataset with the prognosis/labels. The datasets were then split into training and testing data using an 80:20 split. From here the training and test sets were saved as NumPy arrays as the selected model can't use pandas DataFrame objects.

The four datasets after pre-processing are: training_dataset, training_labels, testing_dataset and testing_labels.

Figure 7 – The splitting of the dataset into the four required NumPy datasets after pre-processing of the data

```

63 # Get labelled column from dataset and create labelled dataset
64 dataset_labels = dataset.pop('prognosis')
65
66 # Split dataset into train test sets
67 # 80:20 split for training and test set
68 X_train, X_test, y_train, y_test = train_test_split(dataset, dataset_labels, test_size=0.2)
69
70 # Done to make the datasets in numpy array format to be used for the model.fit function
71 # Model can't use pandas dataframe objects, has to use numpy arrays
72 training_dataset = np.asarray(X_train)
73 training_labels = np.asarray(y_train)
74 testing_dataset = np.asarray(X_test)
75 testing_labels = np.asarray(y_test)
76

```

5.2 Supervised ML Algorithm Selection

The TensorFlow Lite guide states that a TensorFlow model can be converted into a TensorFlow Lite model (TensorFlow 2022). Incorrectly, an assumption was made that TensorFlow contained internal libraries that allowed the training of TensorFlow models using the NB, KNN, DT and RF algorithms which could then be converted into TensorFlow Lite models. This was incorrect. Models trained with the following supervised machine learning algorithms come from an external Python library called scikit-learn (Scikit-learn n.d. b). Instead, it was realised that only Keras trained models, which is a high-level API

TensorFlow self contains, can be converted from a TensorFlow model into a TensorFlow Lite model. Keras is a library that provides an interface for building, training, and testing of artificial neural networks (ANN) in Python (Keras n.d.). Therefore, selection of the supervised machine learning algorithm to be ANN in order to train a TensorFlow model and then convert it into a TensorFlow Lite model was enforced due to this misunderstanding. This is essential as the application requires a TensorFlow Lite model to be able to perform on-device inference in order to perform disease prediction based on user symptom input.

ANNs have the following structure: an input layer, one or many hidden layers and finally an output layer (IBM Cloud Education 2022b). At each layer there are neurons which connect to other neurons, and each neuron has a connection with an associated weight and threshold (IBM Cloud Education 2022b). If the scenario occurs where output of a neuron is above a threshold value, the neuron is activated and sends data to the next layer of the ANN otherwise no data is sent onwards (IBM Cloud Education 2022b).

As mentioned in section 2.3, ANNs can be used for classification problems. In the case of the problem defined here, we will be using the ANN to perform multiclass classification. It will be able to predict a disease class out of 41 disease classes based off input samples i.e., patient symptoms.

5.3 *Build and Train ANN model*

Figure 8 – Building and training of the ANN model in the Python script

```
# Define the model's architecture (layers) - an artificial neural network
# 3 layers with a Dense layer at every layer, so it is a fully connected network
model = keras.Sequential([
    # Input layer first
    # The number of input features is specified by input_dim i.e. the model expects rows of data with 132 features
    # Dense layer - every neuron is connected to every neuron in the next layer
    # Activation function is rectified linear unit
    keras.layers.Dense(132, input_dim=132, activation="relu"),
    # First hidden layer, also a dense layer so every neuron is connected to every neuron in previous and next layer
    # Number of neurons chosen for the hidden layer is 64
    # Activation function is also rectified linear unit
    keras.layers.Dense(64, activation="relu"),
    # Final layer is output layer with 41 neurons as we are classifying 41 classes (diseases)
    # Activation function softmax calculates values for each neuron so that the sum of those values equals 1
    # i.e. the probability of the network thinking it's a certain class
    keras.layers.Dense(41, activation="softmax")
])

# Compile the model
# Adam is the optimization algorithm used to update weights in the neural network
# Loss function tells us how different the predicted result is from the actual result
# Used to monitor the error by the neural network and should reduce as the weights update during and after each epoch
# Accuracy metric shows the percentage of predictions that identified the correct label (disease)
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Train model
# Epochs is the number of iterations through all the rows of the training dataset
# Batch size is the number of dataset rows considered, within an epoch, before the model weights are updated
model.fit(training_dataset, training_labels, epochs=30, batch_size=81)
```

Keras API was used to build, train, and test the ANN model (Keras n.d.). The ANN model was built and trained in the following manner. The ANN model was built with a three-layer architecture. An input layer, a single hidden layer, and an output layer. This ANN model is a fully connected network, so each neuron in a layer is connected to every neuron in the next layer once. The first layer is the input layer, and it has 132 neurons. The number of input features is specified by the `input_dim` parameter, so this model expects input rows of data with 132 features i.e., 132 symptoms per sample that the `training_dataset` and `testing_dataset` datasets will have. The activation function at this layer is rectified linear unit. An activation function defines how the weighted sum of inputs at a neuron in a layer is transformed into an output (Brownlee 2021). The hidden layer has 64 neurons and also has an activation function of rectified linear unit. The final layer, the output layer, has 41 neurons as the ANN model is classifying 41 diseases/classes/labels. The activation function at this layer is softmax. The softmax activation function calculates values for each neuron so that the sum of those values equals 1 i.e., the probability that the network thinks the prediction it makes is a certain class, for each class (Brownlee 2021).

The model was then compiled using the `compile` function. Adam was the optimisation algorithm used to update the weights of the ANN model. The loss function shows how different the predicted result is from the actual result and it is used to monitor the error by the neural network (Brownlee 2019). A loss function that can be used for multiclass classification is sparse cross entropy (Brownlee 2019). So, this was used for this ANN model and defined using `'sparse_categorical_crossentropy'` for the loss function parameter. Also, the metric of accuracy was used to judge the performance of the ANN model. By using accuracy, a result can be displayed when the model is tested to show the percentage of predictions the model did which identified the correct class (disease).

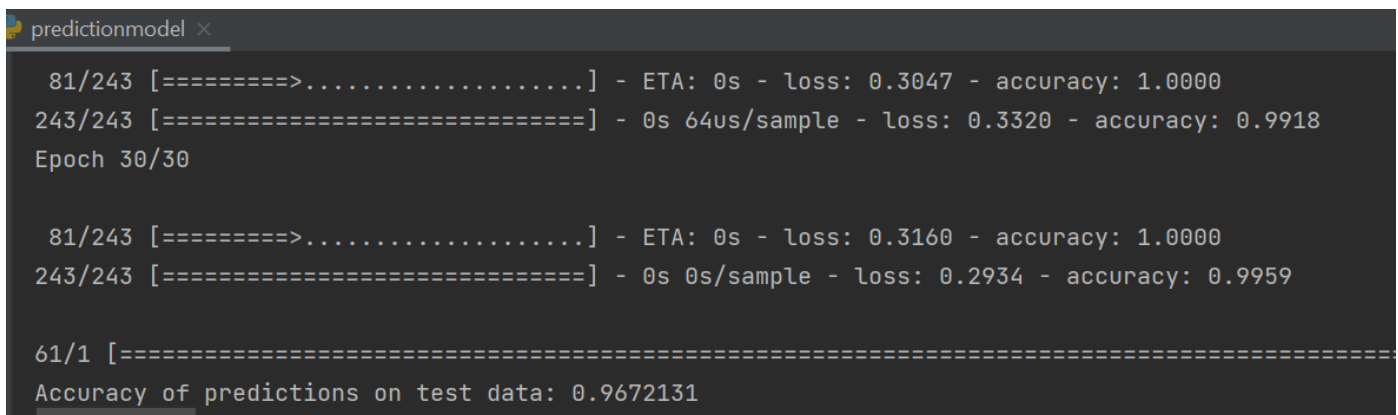
The ANN model is called to be trained upon the `training_dataset` and `training_labels` datasets using the `fit` function. The number of epochs was specified to be 30, whilst the batch size was specified to be 81. Epochs specifies the number of iterations the neural network goes through all the rows of the training dataset when training (Brownlee 2018). Batch size specifies the number of samples the neural network considers before updating its weights within an epoch (Brownlee 2018).

largest value and returning that index with NumPy package function `np.argmax()`, we can use that index to return the class label i.e., the predicted disease. This is done with an array of disease labels that was created earlier in the Python script (see Appendix C). This was done to assure the ANN model could correctly predict the disease Fungal Infection based off the input sample provided upon a run of the Python file.

Finally, the ANN model is converted into a TensorFlow Lite model and into a tflite file called 'diseasepredictionmodel.tflite'. This file can be used in Android Studio as an asset to perform on-device inference in order to predict disease based on user symptom input as shown in section Figure 3 of this report.

As mentioned earlier in section 3.5, the trained model is required to have a prediction accuracy of above 95%. The script was run manually repeatedly to build, train, and test the model. The accuracy score of the model was presented on in PyCharm console upon each run. Each run of the Python script meant the rewriting of the diseasepredictionmodel.tflite file and saving a different TensorFlow Lite model.

Figure 10 – Accuracy of the selected ANN model's predictions on test data



```
predictionmodel ×
81/243 [=====>.....] - ETA: 0s - loss: 0.3047 - accuracy: 1.0000
243/243 [=====] - 0s 64us/sample - loss: 0.3320 - accuracy: 0.9918
Epoch 30/30

81/243 [=====>.....] - ETA: 0s - loss: 0.3160 - accuracy: 1.0000
243/243 [=====] - 0s 0s/sample - loss: 0.2934 - accuracy: 0.9959

61/1 [=====]
Accuracy of predictions on test data: 0.9672131
```

The ANN model which was selected to be used for the application presented a prediction accuracy of 96.72% of on the test data. Therefore, the 'diseasepredictionmodel.tflite' saved upon this run was used as an asset for the Disease Prediction Android Application. This asset is the TensorFlow Lite model the application will utilise to perform on-device inference to predict disease based off user symptom input. It should be noted that since the dataset was split with an 80:20 train test split, 61 samples were used for testing whilst 243 samples were used for training of the ANN model.

5.5 Application Description

This section of the report will provide a description of the application and what developments have been in implementing its features. The submitted implementation of the application displays the implementation of all the functional requirements specified in section 3.2 of the report. Implementation of the logout, login and register functionalities of the prototype have been developed by utilising video tutorials (SmallAcademy 2021a-g). The register functionality works in the following manner. A user is required to enter an email address, their password, and a confirmation of their password once more on the Register page. Validation occurs to assure that the input fields are not empty as well as both passwords match. Warning messages appearing in the input fields if one or both are

empty when the user clicks the Register button. If input data is valid, the user's email address and password (in hashed form) are stored in the Firebase Authentication database. Exceptions are thrown and an error message is displayed in the app if: a user tries to register with an email address that has already been registered to the application, they don't provide an email address in the correct format or if the password is classed as invalid i.e., it is less than 6 characters in length. Once registered, a user is taken to the main landing page. This page will be described later. Also, the user is able to go to the Login page by clicking the Login button on the Register page.

The login functionality has been implemented on the Login page. If a user has registered to the application, they can login to the application with the email address and password they registered with. If their login credentials are found and match with those in the Firebase Authentication database, they are logged into the application and redirected to the main landing page. Once again validation occurs to assure that the input fields are not empty with warning messages appearing in the input field if any of them are empty. Furthermore, exceptions are thrown, and error messages are displayed in the app if: a user tries to login within an email address that has not been registered to the application, a user enters the wrong password for an account that has been registered to the application or a user doesn't enter an email address in the correct format. Also, a user is able to go to the Register page by clicking the Create Account button on the Login page.

Furthermore, the log out functionality has been implemented. At the bottom of the main landing page a user can click the Logout button and they will be redirected to the Login page. This functionality has been developed, as the user remains logged into their account on the application even if they exit the application on the mobile device and re-enter it. This is the case unless they click the Logout button. This has been done to stop the user having to login every time they want to access the application.

A description of the user interface from the main landing page will also be described. The main landing page has 4 buttons which are: Predict Disease, View Prediction Results, Find Healthcare Service and Logout. By clicking the Predict Disease button the user is redirected to Predict Disease page. By clicking the View Prediction Results button, the user is redirected to the View Prediction Results page. By clicking the Find Healthcare Service button the user is redirected to the Find Healthcare Service page. The Logout button has already been described.

Predict Disease button redirects to the Predict Disease page. The symptom input, predict disease, view information about disease/condition and save result functionalities have been implemented on this page. Symptom input functionality works in the following manner. There are 17 drop down menus for a user to select a symptom they're experiencing. The drop-down menus present 133 values to select from, with the default value being no symptom and the other 132 values being different symptoms to select from. The first 4 symptom drop down menus must be selected from to gain a disease prediction, whilst the following 13 drop down menus can be optionally and additionally selected from by a user to gain a disease prediction. This page is vertically scrollable so that all the drop-down menus and buttons on this page can be presented. The predict disease functionality works in the following manner. The page presents a Predict Disease button. Before the user presses the button, a place holder text appears below the button stating, "Result of disease

prediction appears here”. If a user selects at least the first four mandatory symptoms and then selects zero to any number of additional symptoms and then presses the Predict Disease button the application predicts their disease. The disease predicted is one of a possible forty-one diseases the application can predict for a user. This is done using the TensorFlow Lite model by the application and it presents it on the page in the format of “You are likely to have:” and then the disease the application has predicted for the user in place of the place holder text. If the user clicks the Predict Disease button having not selected four initial mandatory symptoms a warning message is displayed on screen stating, “You must enter four initial symptoms to predict your disease!”. This will occur in the following scenarios: if the user selects zero to three symptoms from the initial four drop down menus or they only selected symptoms from the other thirteen additional symptom drop down menus and then clicked the Predict Disease button. The view information about disease/condition functionally works in the following manner. The page presents a View Disease Information button underneath the place holder text/disease result. If a user clicks this button without acquiring a disease prediction a warning message appears on screen stating, “You haven’t acquired a disease prediction”. If a user does acquire a disease prediction via the application the user may click the View Disease Information button and a healthcare organisations webpage related to the disease will be presented to the user inside the application. This webpage provides the user further information about the disease/condition. The application will redirect a user to a webpage for every one of the 41 possible diseases/conditions it can predict for a user. Information about disease/conditions is provided from the following healthcare organisations. Information about multiple diseases/conditions are provided from NHS webpages on conditions (NHS n.d.) Information about multiple diseases/conditions are provided from webpages from Bupa (Bupa n.d.). Information about the disease Cholestasis is provided from a webpage by Healthline (Healthline 2018). Information about the condition of a drug reaction is provided from a webpage by NICE (National Institute for Health and Care Excellence) (NICE 2022). Information about the disease Hepatitis E is provided from a webpage by the British Liver Trust (British Liver Trust n.d.). Information about the disease Alcoholic Hepatitis is provided from a webpage by Johns Hopkins Medicine (Hopkins Medicine n.d.). The user can view the webpage or click the return button of their device to return to the Predict Disease page. Furthermore, there is a Save Result button on this page. If a user clicks this button without acquiring a disease prediction a warning message is displayed on screen stating, “You haven’t acquired a disease prediction”. If a user has acquired a disease prediction and they press this button the following things occur. A message appears on screen stating, “Your disease prediction result has been saved” and the result predicted for the user as well the date and time this button was clicked is saved and stored in the Firebase Realtime database. Finally, there is Return button at the bottom of the page which redirects the user to the main landing page.

By clicking the View Predictions Results button on the main landing page, the user is redirected to the View Prediction Results page. The view past results functionality has been implemented on this page. The page is titled at the top “Your prediction result : prediction date”. This is the structure each saved result is presented with. All the results a user saved from previous disease predictions are dynamically loaded from the Firebase Realtime database and displayed in a list on screen. This page also has a Return button at the bottom of the page which redirects the user to the main landing page.

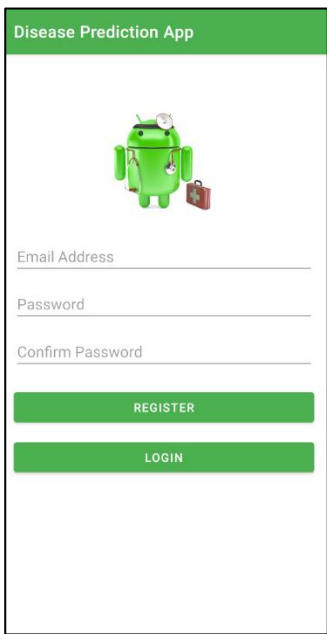
By clicking the Find Healthcare Service button on the main landing page the user is redirected to the Find Healthcare Service page. The find nearby healthcare practitioner functionality is implemented via this page. The page presents four buttons: Find Nearby Hospitals, Find Nearby GP Surgeries, Find Nearby Pharmacies and Find Nearby Optician. By clicking on any one of these buttons a Google Maps webpage is displayed in the application and this webpage displays the selected healthcare service practitioners nearby to the user on a map. By clicking the Find Nearby Hospitals button, a Google Maps webpage showing nearby hospitals is displayed in the application (Google Maps n.d. a). By clicking the Find Nearby GP Surgeries button, a Google Maps webpage showing nearby GP surgeries is displayed in the application (Google Maps n.d. b). By clicking the Find Nearby Pharmacies button, a Google Maps webpage showing nearby pharmacies is displayed in the application (Google Maps n.d. c). By clicking the Find Nearby Optician button, a Google Maps webpage showing nearby opticians is displayed in the application (Google Maps n.d. a). This page also has a Return button at the bottom of the page which redirects the user to the main landing page.

To develop the application a variety of different materials were utilised on top of those previously referred to train the ANN/TensorFlow Lite model. For overall knowledge of Android application development, an Android application Development course video series by freeCodeCamp.org was studied (freeCodeCamp.org 2020a-b). The course teaches how to present a webpage in the application. By applying this knowledge, I was able to implement the view more information about disease/condition and find nearby healthcare practitioner functionalities as these functionalities display webpages in the application. The course teaches how to create drop-down menus for a user to select from. By applying this knowledge, I was able to implement the symptom input functionality with drop down menus. A video tutorial on how to make an android page vertically scrollable by Coding Demos was studied (Coding Demos 2020). By applying this knowledge, I was able to change the xml mark up for the Predict Disease page, so that it is vertically scrollable in order to present all the drop-down menus and buttons of this page on screen in a presentable manner. To implement the predict disease functionality a variety of materials were utilised. A video series on Python neural networks using Keras by Tech with Tim was studied and followed (Tech with Time 2019a-d). This video series showed the implementation of a Keras built neural network model that could perform multiclass image classification. An online guide was also followed and studied which demonstrated how a Keras built neural network model can be built to perform binary classification to determine if test data displays a sample to be diabetic or non-diabetic (Brownlee 2019b). A video tutorial was viewed and followed which demonstrated an example of training a simple Keras built neural network model to predict the single numerical value output of a function and then perform inference with the model as a TensorFlow Lite model in an example Android application based on user input (Mobile Machine Learning 2020). An online guide was read which also demonstrated an example of using a Keras neural network model as a TensorFlow lite model to perform inference in an Android application and predict the single numerical value output of a function with a user entering an input value (Kota 2020). This guide was read to consolidate my understanding of how to utilise a trained TensorFlow Lite model in Android application to perform inference. By utilising the materials above, I was able to apply the knowledge gained to provide my own implementation to build ANN model that can perform multi-class classification of disease. I was also able to use this ANN model as a TensorFlow Lite model in the application to

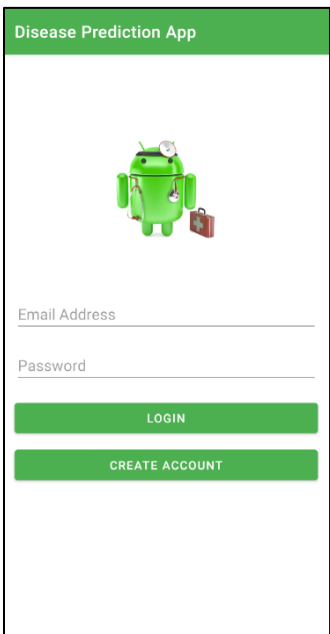
perform on-device inference to predict disease via user input. This was done with the use of the Java API for TensorFlow Lite inference and more specifically the Interpreter class of this API. A video by ProgrammingKnowledge teaches how to perform CRUD operations on the Firebase Realtime database for Android applications (Programming Knowledge 2021). By applying the knowledge gained from this video I was able to implement the save result functionality and view past results functionality. These functionalities both require interaction with the Firebase Realtime database.

5.6 Screenshots of Application

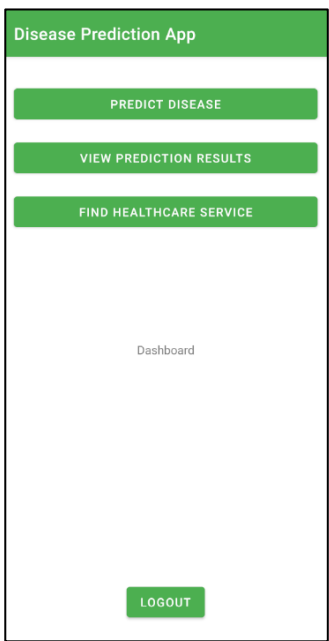
This section of the report will provide a screenshot of each page of the Disease Prediction Android application.



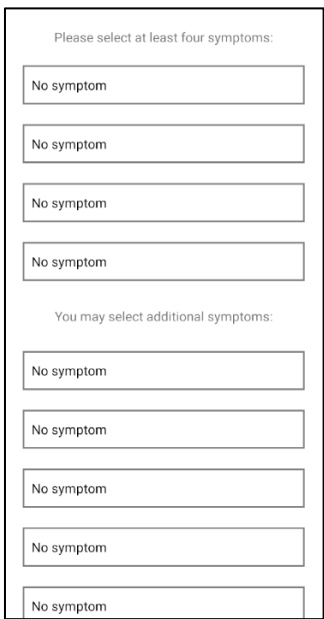
Register page



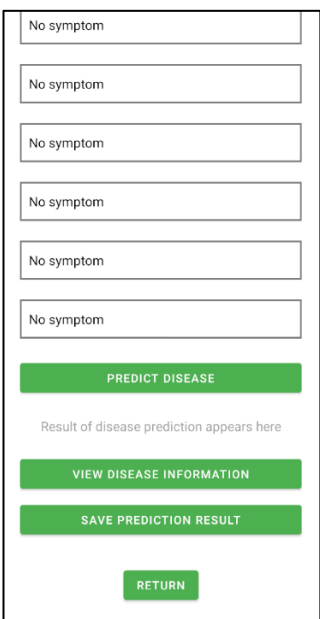
Login page



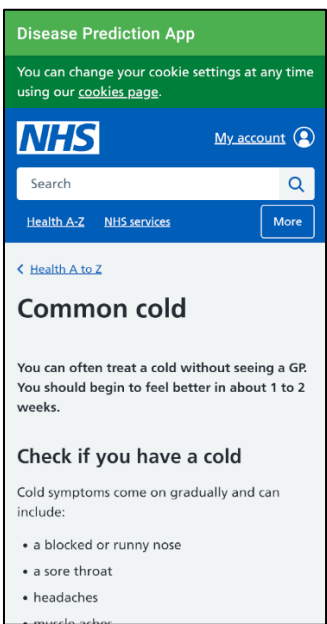
Main landing page



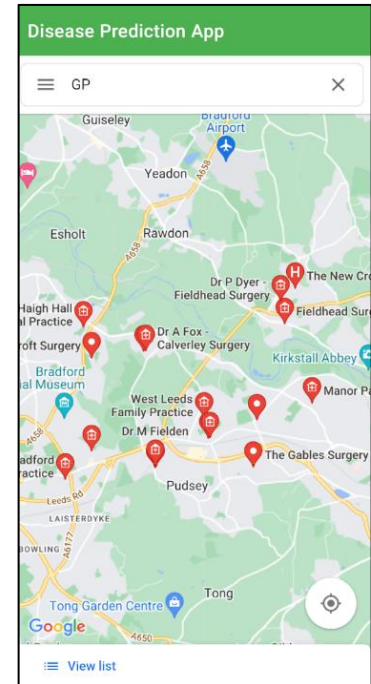
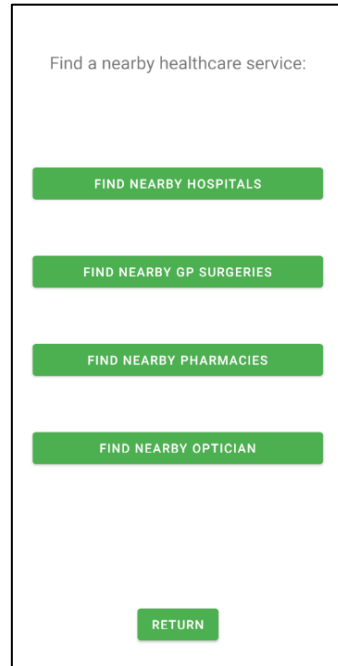
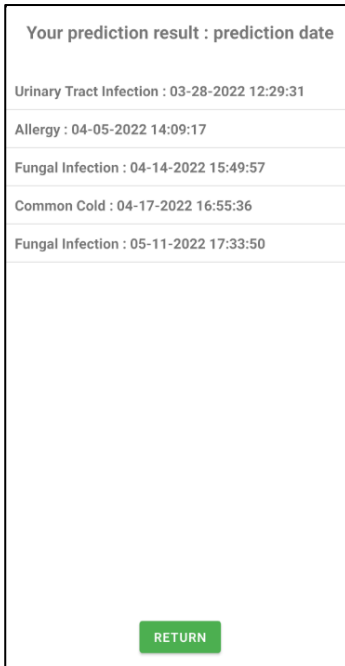
Predict Disease page



Bottom of Predict Disease page



Webpage displayed in the application for predicted disease



View Prediction Results page

Find Healthcare Service page

Google Maps webpage of nearby GP surgeries

It should be noted that if a user acquires a disease prediction via the application and then presses the View Disease Information button, a webpage from a possible 41 different webpages dependent of their predicted disease will be displayed to them.

Furthermore, only a screen shot of the Google Maps webpage presented to the user after clicking the Find Nearby GP Surgeries button is shown above. However, similar visual output is provided to the user if they click any of the other buttons for their required healthcare service.

5.7 Predict Disease Functionality Implementation

The predict disease functionality is the main functionality of the Disease Prediction Android Application. An explanation of how this functionality was implemented will be provided below alongside screenshots of the code.

Implementation of the functionality required the use of the Interpreter class from the Java API for TensorFlow Lite inference . By doing so, the ANN model that was trained and converted into a TensorFlow Lite model as the 'diseasepredictionmodel.tflite' file, can be used to perform on-device inference to predict disease based off user symptom input.

The predict disease functionality was implemented in the PredictDisease.java class. At the top of the class a string array of symptoms is declared as well as an additional value of 'no symptoms' at the first element. These values are presented to the user in order for them to select what symptoms from onscreen drop-down menus. Also, a string array of disease labels, for the 41 diseases that the application can predict a user is declared. This array of disease labels will be used to provide verbose output to the user about their disease prediction. Furthermore, an object of the Inference class named inference is declared. This

object will be used to load the model and drive on-device inference by executing the model on input data and returning a prediction via output data.

Figure 11 – Symptoms list array, disease labels array and Interpreter object declared.

```
// Symptom list
private String[] symptomsList = {"No symptom", "Itching", "Skin rash", "Nodal skin eruptions", "Continuous sneezing",
"Shivering", "Chills", "Joint pain", "Stomach pain", "Acidity", "Ulcers on tongue", "Muscle wasting",
"Vomiting", "Burning micturition", "Spotting urination", "Fatigue", "Weight gain", "Anxiety",
"Cold hands and feet", "Mood swings", "Weight loss", "Restlessness", "Lethargy", "Patches in throat",
"Irregular sugar level", "Cough", "High fever", "Sunken eyes", "Breathlessness", "Sweating",
"Dehydration", "Indigestion", "Headache", "Yellowish skin", "Dark urine", "Nausea", "Loss of appetite",
"Pain behind the eyes", "Back pain", "Constipation", "Abdominal pain", "Diarrhoea", "Mild fever",
"Yellow urine", "Yellowing of eyes", "Acute liver failure", "Fluid overload", "Swelling of stomach",
"Swelled lymph nodes", "Malaise", "Blurred and distorted vision", "Phlegm", "Throat irritation",
"Redness of eyes", "Sinus pressure", "Runny nose", "Congestion", "Chest pain", "Weakness in limbs",
"Fast heart rate", "Pain during bowel movements", "Pain in anal region", "Bloody stool",
"Irritation in anus", "Neck pain", "Dizziness", "Cramps", "Bruising", "Obesity", "Swollen legs",
"Swollen blood vessels", "Puffy face and eyes", "Enlarged thyroid", "Brittle nails",
"Swollen extremities", "Excessive hunger", "Extra marital contacts", "Drying and tingling lips",
"Slurred speech", "Knee pain", "Hip joint pain", "Muscle weakness", "Stiff neck",
"Swelling joints", "Movement stiffness", "Spinning movements", "Loss of balance", "Unsteadiness",
"Weakness of one body side", "Loss of smell", "Bladder discomfort", "Foul smell of urine",
"Continuous feel of urine", "Passage of gases", "Internal itching", "Toxic look (Typhus)",
"Depression", "Irritability", "Muscle pain", "Altered sensorium", "Red spots over body",
"Belly pain", "Abnormal menstruation", "Dischromic patches", "Watering from eyes",
"Increased appetite", "Polyuria (Excessive Urination)", "Family history", "Mucoid sputum", "Rusty sputum",
"Lack of concentration", "Visual disturbances", "Receiving blood transfusion",
"Receiving unsterile injections", "Coma", "Stomach bleeding", "Distention of abdomen",
"History of alcohol consumption", "Fluid overload", "Blood in sputum", "Prominent veins on calf",
"Palpitations", "Painful walking", "Pus filled pimples", "Blackheads", "Scurring", "Skin peeling",
"Silver like dusting", "Small dents in nails", "Inflammatory nails", "Blister", "Red sore around nose",
"Yellow crust ooze"};

// Diseases labels
private String[] diseaseLabels = {"Fungal Infection", "Allergy", "GERD", "Chronic Cholestasis",
"Drug Reaction", "Peptic Ulcer Disease", "AIDS", "Diabetes", "Gastroenteritis", "Bronchial Asthma",
"Hypertension", "Migraine", "Cervical Spondylosis", "Paralysis (Brain Hemorrhage)", "Jaundice",
"Malaria", "Chicken Pox", "Dengue", "Typhoid", "Hepatitis A", "Hepatitis B", "Hepatitis C",
"Hepatitis D", "Hepatitis E", "Alcoholic Hepatitis", "Tuberculosis", "Common Cold", "Pneumonia",
"Dimorphic Hemorrhoids (Piles)", "Heart Attack", "Varicose Veins", "Hypothyroidism",
"Hypertthyroidism", "Hypoglycemia", "Osteoarthritis", "Arthritis", "Paroxysmal Positional Vertigo (Vertigo)",
"Acne", "Urinary Tract Infection", "Psoriasis", "Impetigo"};

// Interpreter object using Interpreter class from Java API for TensorFlow Lite inference
Interpreter interpreter;
```

The interpreter object was initialised, and the model was loaded to the interpreter object with a loadModelFile() method call. It should be noted that the way the loadModelFile() method that was implemented in this application's implementation is done in the same way as demonstrated by Mobile Machine Learning in their video tutorial (Mobile Machine Learning 2020). The TensorFlow Lite Inference guide notes that an interpreter object can be initialised, and the model loaded to it utilised using a MappedByteBuffer (TensorFlow 2021b). Therefore, a decision was made to use this method to load the model as it returns a MappedByteBuffer. Another source which was studied during the project's development demonstrates how to load a TensorFlow Lite model to an Interpreter class object, demonstrated a very similar implementation of this method (Kota 2020). Due to this, this method can be deemed as boiler plate code. Nevertheless, the source of the code for this method has been referenced to in the implementation as well as within this report with this citation (Mobile Machine Learning 2020). This is shown in Figure 13.

Figure 12 – Call to load the model to initialise the interpreter object

```
databasePredictedResults = FirebaseDatabase.getInstance().getReference();

try {
    // Load the model to the interpreter
    interpreter = new Interpreter(loadModelFile());
} catch (IOException e) {
    e.printStackTrace();
}
```

Figure 13 –The loadModelFile() method

```
/** A utility method used to load the model to use it for inference
 *
 * Utilised source code from a video tutorial posted by the Youtube channel
 * Machine Mobile Learning on Youtube at time stamp 11:46:
 * https://www.youtube.com/watch?v=63bCmY5uRto
 *
 * @return MappedByteBuffer
 * @throws IOException
 */
private MappedByteBuffer loadModelFile() throws IOException {
    AssetFileDescriptor assetFileDescriptor = this.getAssets().openFd("diseasepredictionmodel.tflite");
    FileInputStream fileInputStream = new FileInputStream(assetFileDescriptor.getFileDescriptor());
    FileChannel fileChannel = fileInputStream.getChannel();
    long startOffset = assetFileDescriptor.getStartOffset();
    long length = assetFileDescriptor.getLength();
    return fileChannel.map(FileChannel.MapMode.READ_ONLY, startOffset, length);
}
```

This method call loads the TensorFlow Lite model, “diseasepredictionmodel.tflite”, that was created using the predictionmodel.py Python script.

The following screenshots show what occurs when the user clicks the Predict Disease button on the Predict Disease page.

Figure 14 – Code when the user clicks Predict Disease button

```
TextView diseaseResult = findViewById(R.id.diseaseRes);
Button predictDiseaseButton = findViewById(R.id.predictDiseaseBtn);
// predict disease once button is clicked
predictDiseaseButton.setOnClickListener(v -> {
    // array to hold the values of user symptom input
    // The position of the item in the drop down (minus 1) corresponds
    // to the symptom in the symptom list
    int[] symptomInputs = new int[17];
    symptomInputs[0] = symptomOne.getSelectedItemPosition();
    symptomInputs[1] = symptomTwo.getSelectedItemPosition();
    symptomInputs[2] = symptomThree.getSelectedItemPosition();
    symptomInputs[3] = symptomFour.getSelectedItemPosition();
    symptomInputs[4] = symptomFive.getSelectedItemPosition();
    symptomInputs[5] = symptomSix.getSelectedItemPosition();
    symptomInputs[6] = symptomSeven.getSelectedItemPosition();
    symptomInputs[7] = symptomEight.getSelectedItemPosition();
    symptomInputs[8] = symptomNine.getSelectedItemPosition();
    symptomInputs[9] = symptomTen.getSelectedItemPosition();
    symptomInputs[10] = symptomEleven.getSelectedItemPosition();
    symptomInputs[11] = symptomTwelve.getSelectedItemPosition();
    symptomInputs[12] = symptomThirteen.getSelectedItemPosition();
    symptomInputs[13] = symptomFourteen.getSelectedItemPosition();
    symptomInputs[14] = symptomFifteen.getSelectedItemPosition();
    symptomInputs[15] = symptomSixteen.getSelectedItemPosition();
    symptomInputs[16] = symptomSeventeen.getSelectedItemPosition();

    // 2D input array for the model with size 1 * 132
    // Each of the 132 is representative of a symptom
    float[][] modelInputArray = new float[1][132];
}
```


An array called symptomInputs is created which holds the values of user symptom input. This is done by retrieving the selected item (symptom) position in the drop-down menu on screen. The position of the item i.e., symptom in the drop-down menu corresponds to a symptom in the symptom list. So, for e.g., if a user selects Itching this item is at position 1. The use of the method call getItemPosition() returns the positions of the selected items (symptoms) on screen for every one of the 17 drop down menus for symptom input presented on the Predict Disease page of the app.

Furthermore, a 2D float input array for the model, called modelInputArray, with a size of 1 x 132 is created. Each of the 132 elements will be representative of a symptom. Each element will denote the presence or lack of presence of a symptom.

Figure 15 – Code when the user clicks the Predict Disease button continued

```
// 2D input array for the model with size 1 * 132
// Each of the 132 is representative of a symptom
float[][] modelInputArray = new float[1][132];

// Assure disease prediction only occurs if user enters 4 symptoms
// Check to see if any or all of first 4 user inputs are No symptom (i.e. value of 0)
// Display toast and error message to the user if true otherwise predict disease
if (symptomInputs[0] == 0 || symptomInputs[1] == 0 || symptomInputs[2] == 0 || symptomInputs[3] == 0){
    // Display a Toast (error message to the user) tell them they must select 4 symptoms
    Toast.makeText(context, PredictDisease.this, "You must enter four initial symptoms to predict your disease!", Toast.LENGTH_SHORT).show
} else {
    for (int i = 0; i < symptomInputs.length; i++) {
        // If the value is not 0 (no symptom), indicate the presence of the symptom
        // In the modelInputArray by assigning a value of 1 in the corresponding element
        // Note, minus 1 to value as we added an extra drop down option of
        // No symptom to drop down menu
        if (symptomInputs[i] != 0) {
            int value = symptomInputs[i];
            int symptomPresentIndex = value - 1;
            modelInputArray[0][symptomPresentIndex] = 1;
        }
    }
    // Call the predictDisease method, passing the modelInputArray as an argument
    // and assign it the result to the modelOutputArray
    float[][] modelOutputArray = predictDisease(modelInputArray);
    // get index of element that is predicted to most likely be the disease
    resultIndexForDiseaseLabels = findMaxIndex(modelOutputArray);
    // use index to provide verbose output to the user
    predictedDisease = diseaseLabels[resultIndexForDiseaseLabels];
    diseaseResult.setText("You are likely to have: " + predictedDisease);
    // Assign the value of the boolean that denotes if a disease has been predicted or not as true
    diseaseHasBeenPredicted = true;
}
});
```

A check is done to assure a disease prediction will only occur if the user enters at least 4 symptoms from the initial four drop down menus presented on the PredictDisease page. A check is done to see, if the user selected the option “No symptoms” from any or all of the initial first four drop down menus presented in the Predict Disease page. No symptom holds an item position value of 0 for the drop-down menu. If so, an error message is displayed to the user stating they must enter four initial symptoms to predict disease. If this is not the case the following occurs. The modelInputArray is populated based on user symptom selection. A for loop iterates through the symptomInputs array in order to establish the modelInputArray to represent symptom presence and provide a sample for the model to perform a prediction on. If a value is not 0 (no symptom), indication of the presence of the symptom in the modelInputArray is done by assigning a value of 1 in the corresponding element. It should be noted that the value of the symptomInputs element is

subtracted by 1 to denote the symptom presenting. This is done as an extra drop-down option of No symptoms has been added to the drop-down menu. To clarify this a bit better, an example will be provided. If a user was to select the symptom Itching which has an item position of 1 from a drop-down menu on screen. The position item value of 1 would be subtracted by 1 to give a value of 0. Then in the modelsInputArray at position modelsInputArray[0][0], the value is assigned as 1 to denote the presence of the Itching symptom.

Then a call to the predictDisease method is made, while passing the now populated modelInputArray as an argument. The result of this method call, is the result of the prediction done by the TensorFlow Lite model and this will be assigned to a 2d float array called modelOutputArray.

Figure 16 – The predictDisease() method

```
//Method that takes user input and returns the output of prediction
//from the model
private float[][] predictDisease(float[][] inputArray){
    // 2D output array that is 1 * 41 for the 41 categories (diseases) of classification
    float[][] output = new float[1][41];
    // Call to TensorFlow Lite API class to run inference on model
    // input array passed to the run method and the result will be stored in the output array
    interpreter.run(inputArray, output);
    // return output array
    // output array, holds 41 elements, each element has a float value representing
    // how likely the model thinks the input is that disease
    // each element corresponds to a disease labels
    return output;
}
```

The predictDisease() method was implemented in the following manner. This method takes a parameter in the form of the modelInputArray and returns the output of the prediction done by the TensorFlow Lite model. A 2D float output array that has the dimensions of 1 x 41 is created, this represents the 41 categories (disease labels) of classification. Using the interpreter object, the run method is called and the inputArray i.e., the modelInputArray is passed to the run method. This method call executes the TensorFlow Lite model to perform a prediction on the input array and the result is stored in the output array. The output array holds 41 elements, and each element has a float value representing how likely the model thinks the input is that disease denoted at that index. Since the TensorFlow Lite model is an ANN model which last layer (output layer) used the softmax activation function. Each index of the output array is representative of a class/label. Moreover, each index of this output array corresponds to a disease label from the array of disease labels in Figure 11. The output array is returned by this method and assigned to the modelOutputArray variable in Figure 15.

Looking at Figure 15, the method call findMaxIndex is called and it passes the modelOutputArray as an argument. This is done to get the index of the element that has been predicted by the model to most likely be the disease.

Figure 17 – The findMaxIndex() method

```
// Method that mimics the behaviour of np.argmax() of numpy package in my python file
// Return the index of output array element that has the highest value predicted
private int findMaxIndex(float[][] array){
    int index = 0;
    float maxValue = 0;
    float[][] checkArray = array;
    for (int i = 0; i < checkArray.length; i++) {
        for (int j = 0; j < checkArray[i].length; j++) {
            if(maxValue<array[i][j]){
                maxValue = array[i][j];
                index = j;
            }
        }
    }
    // Since it's a 1 x 41 array, only need to return the index value of the 2nd dimension
    return index;
}
```

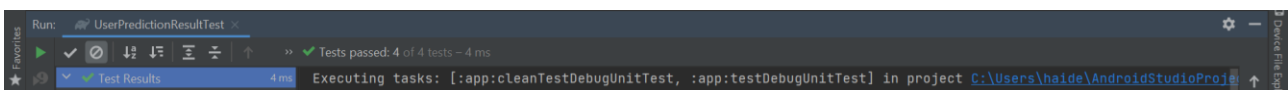
This method has been devised and implemented to mimic the behaviour of the np.argmax() method of the NumPy package in the predictionmodel.py Python script (see Appendix C). This method returns the index of the output array element that has the highest value assigned to it due to prediction by the model. Since the output array is 1 x 41, the implementation only needs to return the index value of the 2nd dimension of the array.

Looking at Figure 15. Once this index value is returned by the findMaxIndex() method it is assigned to the int variable resultIndexForDiseaseLabels. This variable is used to provide verbose output on screen to the user. By retrieving the String value from the diseaseLabels array in Figure 11 using the resultIndexForDiseaseLabels, the predicted disease is assigned to the String variable predictedDisease. Then a variable which is called diseaseResult (seen on Figure 14), which controls a text box presented on the Disease Predict box page, sets the result of the disease prediction on screen in the format “You are likely to have” + predictedDisease. The predicted disease, that the application has predicted for the user by, is then presented on screen to the user.

5.8 Unit Testing

Unit testing was performed for the application in Android Studio. Unit testing was performed for the methods of the UserPredictionResult class as it provides modular units of code that are testable using the JUnit 4 testing framework.

Figure 18 – Result of unit testing using the UserPredictionResultTest class



Four test cases were implemented to test the setters and getters of the UserPredictionResult class in the UserPredictionResultTest class. All four test cases passed showing that the behaviour of these methods is correct.

5.9 System Testing

As mentioned in section 3.5 of the report, system testing will be done to evaluate the performance of the Disease Prediction Android Application as a complete system. Test scenarios related to the functional requirements will be presented, as well their expected outcome and actual outcome upon testing. A result to describe if they have passed or failed will be shown. If a test scenario failed, a column to present what amendments were made in order for the test scenario to pass. A final column will show if a test scenario has passed after amendments.

Table 2 – System testing the Disease Prediction Android Application

Associated functionality	Test Scenario	Expected Outcome	Actual Outcome	Result (Pass or Fail)	Amendments made in order to pass test scenario	Result after amendments made (Pass or Fail)
Register	A user clicks the register button on the register page after inputting a valid email address and matching password and confirmation password that a both 6 characters in length.	User is registered to the application and redirected to the main landing page of the application.	User is registered to the application and redirected to the main landing page of the application.	Pass	N/A	N/A
Register	A user clicks the register button on the register page without inputting credentials into input fields.	Validation message is presented on screen saying the missing credential is required i.e., email is required, password is required, or confirmation password is required.	Validation message is presented on screen saying the missing credential is required i.e., email is required, password is required, or confirmation password is required.	Pass	N/A	N/A
Register	A user clicks the register button on the register page after filling in the input fields however the password and confirmation password entered are not matching.	Validation message presented on screen saying the passwords do not match.	Validation message presented on screen saying the passwords do not match.	Pass	N/A	N/A

Register	A user clicks the register button on the register page after entering details in the input fields and using an email address that has already been registered to the application.	Error message is displayed to the user saying the email address is already in use by another account.	Error message is displayed to the user saying the email address is already in use by another account.	Pass	N/A	N/A
Register	A user clicks the register button on the register page after entering details in the input fields however the email address provides is in an incorrect format.	Error message is displayed to the user saying the email address is badly formatted.	Error message is displayed to the user saying the email address is badly formatted.	Pass	N/A	N/A
Register	A user clicks on the register button on the register page after entering details in the input fields however the password provided is invalid i.e., it is less than 6 characters in length	Error message is displayed to the user saying the password is invalid, the password should be at least 6 characters in length.	Error message is displayed to the user saying the password is invalid, the password should be at least 6 characters in length.	Pass	N/A	N/A
Register	A user clicks the login button on the register page.	A user is redirected to the login page.	A user is redirected to the login page.	Pass	N/A	N/A
Login	A user clicks the login button on the login page after inputting an email address and password that have been previously registered to the application	A user is redirected to the main landing page of the application.	A user is redirected to the main landing page of the application.	Pass	N/A	N/A

Login	A user clicks the login button on the login page without inputting credentials into input fields.	Validation message is presented on screen saying the credential is missing i.e., email is missing, or password is required.	Validation message is presented on screen saying the missing credential is required i.e., email is missing, or password is required	Pass	N/A	N/A
Login	A user clicks the login button on the login page after entering details in the input fields however the email address provides is in an incorrect format	Error message is displayed to the user saying the email address is badly formatted.	Error message is displayed to the user saying the email address is badly formatted.	Pass	N/A	N/A
Login	A user clicks the login button on the login page after entering the input fields however they enter the wrong password for an account registered to the application.	Error message is displayed to the user saying the password is invalid or the user does not have an account.	Error message is displayed to the user saying the password is invalid or the user does not have an account.	Pass	N/A	N/A
Login	A user clicks the login button on the login page after entering details in the input fields however they have entered an email address that has not been registered to the application.	Error message is displayed to the user saying there is no user record corresponding to this identifier.	Error message is displayed to the user saying there is no user record corresponding to this identifier.	Pass	N/A	N/A
Login	A user clicks the create account button on the login page.	A user is redirected to the register page.	A user is redirected to the register page.	Pass	N/A	N/A

Logout	A user clicks the logout button on the main landing page.	A user is logged out of the application and is redirected from the main landing page to the login page.	A user is logged out of the application and is redirected from the main landing page to the login page.	Pass	N/A	N/A
Symptom input and Predict disease	A user clicks the predict disease button on the on the predict disease page after selecting a mandatory four symptoms via their dropdown fields.	The result of the disease prediction appears on screen below the predict disease button in the format “You are likely to have:” then the name of the predicted disease.	The result of the disease prediction appears on screen below the predict disease button in the format “You are likely to have:” then the name of the predicted disease.	Pass	N/A	N/A
Symptom input and Predict disease	A user clicks the predict disease button on the on the predict disease page however they have not selected a mandatory four symptoms via their dropdown fields.	Error message is displayed to the user saying you must enter four initial symptoms in to predict disease	Error message is displayed to the user saying you must enter four initial symptoms in to predict disease	Pass	N/A	N/A
Symptom input and Predict disease	A user clicks the predict disease button on the on the predict disease page after selecting a mandatory four symptoms and an additional up to 13 symptoms via their dropdown fields.	The result of the disease prediction appears on screen below the predict disease button in the format “You are likely to have:” then the name of the predicted disease.	The result of the disease prediction appears on screen below the predict disease button in the format “You are likely to have:” then the name of the predicted disease.	Pass	N/A	N/A

View information about disease/condition	A user clicks the view disease information button on the predict disease page after getting a prediction for their disease	A user is redirected to a healthcare organisations webpage which is presented in the application and provides information about the disease the application predicted for the user.	A user is redirected to a healthcare organisations webpage which is presented in application and provides information about the disease the application predicted for the user.	Pass	N/A	N/A
View information about disease/condition	A user clicks the view disease information button on the predict disease page without getting a prediction for their disease.	The application does not redirect the user to a healthcare organisations webpage in app.	The application crashed.	Fail	Changed the implementation so that a check is made to see if a user has had a disease predicted for them. If not, an error warning is displayed to user saying you haven't acquired a disease prediction.	Pass
Save result	A user clicks the save prediction result button on the predict disease page after getting a prediction for their disease.	The application presents a message to the user saying your disease prediction result has been saved. The predicted disease and the date and time of this prediction is saved in the Firebase realtime database.	The application presents a message to the user saying your disease prediction result has been saved. The predicted disease and the date and time of this prediction is saved in the Firebase realtime database.	Pass	N/A	N/A
Save result	A user clicks the save prediction result button on the predict disease page without getting a prediction for their disease	The application does not save a predicted disease or the time of prediction to the Firebase realtime database	The application saves solely the time of prediction to the Firebase realtime database i.e., the time the user clicked	Fail	Changed the implementation so that a check is made to see if a user has had a disease predicted for them. If not, an error warning is displayed to user saying you haven't	Pass

			the save button.		acquired a disease prediction. This prevents invalid results such as just saving the prediction date and time being saved to the Firebase realtime database.	
View past results	A user clicks on the view prediction results button on the main landing page after having saved previous disease predictions.	A user is redirected to the view prediction results page where any disease predictions a user chose to save are retrieved from the Firebase realtime database are displayed as well as the time and date the prediction was made.	A user is redirected to the view prediction results page where any disease predictions a user chose to save are retrieved from the Firebase realtime database are displayed as well as the time and date the prediction was made.	Pass	N/A	N/A
View past results	A user clicks on the view prediction results button on the main landing whilst they have not chosen to save any previous disease predictions.	A user is redirected to the view results page however no results are displayed to them.	A user is redirected to the view results page however no results are displayed to them.	Pass	N/A	N/A
Find nearby healthcare practitioner	A user clicks the find nearby hospitals button on the find healthcare service page.	A user is redirected to a google maps webpage which is presented in the application and shows hospitals nearby to the user.	A user is redirected to a google maps webpage which is presented in the application and shows hospitals nearby to the user.	Pass	N/A	N/A

Find nearby healthcare practitioner	A user clicks the find nearby GP surgeries button on the find healthcare service page.	A user is redirected to a google maps webpage which is presented in the application and shows GP surgeries nearby to the user.	A user is redirected to a google maps webpage which is presented in the application and shows GP surgeries nearby to the user.	Pass	N/A	N/A
Find nearby healthcare practitioner	A user clicks the find nearby pharmacies button on the find healthcare service page.	A user is redirected to a google maps webpage which is presented in the application and shows pharmacies nearby to the user.	A user is redirected to a google maps webpage which is presented in the application and shows pharmacies nearby to the user.	Pass	N/A	N/A
Find nearby healthcare practitioner	A user clicks the find nearby optician button on the find healthcare service page.	A user is redirected to a google maps webpage which is presented in the application and shows opticians nearby to the user.	A user is redirected to a google maps webpage which is presented in the application and shows opticians nearby to the user.	Pass	N/A	N/A

6 Results and Discussion

This section will provide some remarks and discussion on the results of this project. The Disease Prediction Android Application has successfully been implemented in its entirety and all the functional requirements specified in section 3.2 of the report have been implemented. The aims of this project have been successfully delivered as the application is able to provide a user an immediate diagnosis of their disease/condition after user symptom input. The application developed also provides the user with assistive features such as being able to gain more information about their predicted disease via health organisation's webpages, saving their disease prediction result, viewing their past disease prediction results, and locating nearby healthcare services to them in the application via Google Maps webpages.

The application utilises an ANN model that showed a prediction accuracy of 96.72% on test data as seen in Figure 10. This ANN model was converted into a TensorFlow Lite model for which the application was able to perform inference and execute predictions using this model. This means the application can predict disease based off user symptom input with a

prediction accuracy of 96.72%. The prediction accuracy of the model is greater than the specified accuracy of 95% that was specified in section 3.5 of the report as an evaluation metric of the model. However, due to this fact there is a possibility that the application will incorrectly predict the disease a user has and misdiagnose them. A misdiagnosis prediction may predict a more severe disease/condition, and this may unnecessarily alarm and worry the user.

Another limitation and criticism of this application is that it is limited to only being able to predict 41 diseases for the user based on user symptom input. The application was limited to this due to the dataset used to train the ANN/TensorFlow Lite model. This also may mean that application provides a disease prediction that is a misdiagnosis of what the user may be experiencing as the application cannot predict for that disease/condition.

Furthermore, the use of an application in this manner of selecting symptoms and gaining a prediction from the application isn't very reflective of what occurs in the real world setting with a doctor/healthcare professional's diagnosis of disease. There are more factors that affect the diagnosis these individuals may consider such as the duration a symptom has been presenting. For example, the presenting of a cough that a patient has had for 3 weeks in comparison to a cough a patient has had for longer may influence the diagnosis in different ways. This application does not work to this level of complexity when considering symptoms and therefore the reliability of the disease predictions the application provides may be questioned.

Due to the reasons provided above, this application can be deemed to be not suitable to be offered to the public at its current state. This developed application is more likeable to a proof-of-concept application. It requires further work to address the deficiencies defined above.

The completion of this project differed to the original plan as detailed in the interim report. Originally it was intended, that the supervised machine learning algorithms of NB, KNN, DT, RF were all used to train a model and predict disease. Firstly, this would be to see if these models could be trained to perform disease prediction (multiclass classification) and secondly if multiple supervised ML algorithms were able to successfully train a model, the model that presented the best accuracy of prediction would be the one to be converted into a TensorFlow Lite model and used in the application to predict disease. However, a misunderstanding was made that changed how the project was progressed. The TensorFlow Lite guide states that a TensorFlow model can be converted into a TensorFlow Lite model (TensorFlow 2022). Incorrectly, an assumption was made that TensorFlow contained internal libraries that allowed the training of TensorFlow models using the NB, KNN, DT and RF algorithms which could then be converted into TensorFlow Lite models. This was incorrect. Models trained with the following supervised machine learning algorithms come from an external Python library called scikit-learn (Scikit-learn n.d. b). Instead, it was realised that only Keras trained models, which is a high-level API TensorFlow self contains, can be converted from a TensorFlow model into a TensorFlow Lite model. Keras is a library that provides an interface for building, training, and testing of artificial neural networks (ANN) in Python (Keras n.d.). Therefore, selection of the supervised machine learning algorithm to be ANN in order to train a TensorFlow model and then convert it into a TensorFlow Lite model was enforced due to this

misunderstanding. The original plan of selection and evaluation metric from the interim report was to select a single model that presented the best prediction accuracy out of the supervised ML algorithms listed above. This was no longer possible and so instead the prediction accuracy of the ANN/TensorFlow Lite model was set to have to be greater than 95% in order for it to be deemed suitable for use in the application.

The project was developed with considerations about data privacy and cyber security held in mind. Confidentiality is maintained by requiring the user to register an account to the application with fewest fields possible in the form of an email address and password. Therefore, a user is only identifiable by their email address and user identifier. Firebase generates per user a string made up of mixed characters and numbers to be a user identifier. The application developed reduces the risks and mitigates the damage done by any security breaches to it or the back-end service it utilizes with Firebase such as Firebase Authentication and the Firebase Realtime database. If a data breach is to occur and the results of disease predictions are leaked, the information is not personally identifiable as users registered to the application with just their email address and password. The saved results of disease predictions per user cannot be interpreted in a manner that personally identifies any user.

7 Conclusions and Further Work

To conclude, the Disease Prediction Android Application has successfully been implemented in all its entirety. An ANN model was trained using Python, Keras API and TensorFlow on a development machine in order to predict disease i.e., perform multiclass classification. This model can perform predictions with an accuracy of 96.72%. This model was converted into a TensorFlow Lite model, which was utilised within the application to perform on-device inference. In other words, perform disease predictions based off user symptom input into the application. The application can predict a disease/condition for a user from 41 different diseases/conditions. The application was also developed to provide the user with other assistive features in regard to their health. Such as being able to view webpages in the application from healthcare organisations which provide further information about their predicted disease/condition. Furthermore, users are provided the ability to save their disease prediction results and view their past disease prediction results in the application. This has been achieved by utilising the Firebase Realtime database. As well as this, the application provides users information about a range of their nearest healthcare service providers by presenting the user Google Maps webpages within the application.

Future work for this project could be approach in two different ways. An individual approach and in a collective team-based approach. The application could have the added functionality of being able to predict the skin diseases/conditions a user has if they take a picture of their skin using a phone's camera. This would be done using a Keras built neural network model to perform multiclass image classification of skin conditions/diseases. The Keras built neural network model could be converted into a TensorFlow Lite model and used within the Android application as an asset to perform inference and predict what skin condition/disease a user has based on the image the user has taken. The implementation of this functionality would further advance the Disease Prediction Android Application.

Following a collective approach, a much-improved Disease Prediction Android Application could be developed. This improved application would address the deficiencies defined of the Disease Prediction Android Application in ‘Chapter 6: Results and Discussion’. This improved application would predict disease at a much higher accuracy (above 99% accuracy), predict as many diseases as possible and also consider the complexities of how symptoms affect the diagnosis of a disease/condition. It would be intended that this version of the application would be suitable for use by the public. A team could be established that takes the knowledge and advice of medical professionals who understand and can define how symptoms affect the diagnosis of disease/conditions. If this information and understanding can be acquired from this team, a complex artificial neural network model could be built and trained to predict a user’s disease/condition. Since this model is expected to be complex, it may require a large amount of computational resources and due to this may have to utilise cloud-based inference. The application would have to be able to interact with a cloud which would store the complex artificial neural network model that is being used to predict disease based off user input.

By undertaking this project, I have acquired many technical skills and practical knowledge. Firstly, I have gained an understanding of how artificial neural networks work and how to build, train and test them using the Keras API and TensorFlow in Python. I have gained an understanding of how to use ANNs to perform multiclass classification. Furthermore, I have gained technical skills and know how in the form of learning Android application development. I have gained an understanding of how to incorporate machine learning into Android Applications with the use of a TensorFlow Lite model and the Java API for TensorFlow Lite inference to perform disease prediction based off user symptom input. As well as this, I have gained an understanding of how NoSQL databases are structured and used in the form of the Firebase Realtime database. This builds upon my past knowledge and experience which was solely related to SQL databases, with past experiences using the MySQL database. The project has also developed my ability to perform time management. Careful planning and the breaking down of tasks at the start of each semester, as demonstrated by the Gantt charts in my portfolio, have facilitated the completion of this project in its entirety. Overall, undertaking this project has been a very rewarding and insightful experience which has provided me with a variety of technical skills and practical knowledge.

8 References

- 1) Accenture (2020) *Artificial Intelligence: Healthcare’s New Nervous System*. [Technical report] Accenture. https://www.accenture.com/t20171215T032059Z__w__/us-en/_acnmedia/PDF-49/Accenture-Health-Artificial-Intelligence.pdf#zoom=50 Accessed 3 January 2022.
- 2) Ahmad, L. G., Eshlaghy, A. T., Poorebrahimi, A., Ebrahimi, M. and Razavi, A. R. (2013) Using Three Machine Learning Techniques for Predicting Breast Cancer Recurrence. *Journal of Health & Medical Informatics* 4(2), 1-3. <http://dx.doi.org/10.4172/2157-7420.1000124> Accessed 8 January.
- 3) Anaconda (n.d.) <https://www.anaconda.com/>
- 4) Android for Developers (n.d. a) *Features overview*. <https://developer.android.com/studio/features> Accessed 19 January 2022.
- 5) Android for Developers (n.d. b) *Build smarter apps with machine learning*. <https://developer.android.com/ml> Accessed 19 January 2022.

- 6) Android for Developers (n.d. c) *Build local unit tests*. <https://developer.android.com/training/testing/local-tests#java> Accessed 23 January 2022.
- 7) Arsene, C. (2021) *Artificial Intelligence in Healthcare: the future is amazing*. Healthcare Weekly. <https://healthcareweekly.com/artificial-intelligence-in-healthcare/> Accessed 3 January 2022.
- 8) Atlas, L., Cole, R., Muthusamy, Y., Lippman, A., Connor, J., Park, D., El-Sharkawi, M. and Marks, R. J. (1990). A performance comparison of trained multilayer perceptrons and trained classification trees. *Proceedings of the IEEE* 78(10), 1614-1619.
- 9) Biswal, A. (2021) *AI Applications: Top 14 Artificial Intelligence Applications in 2022*. <https://www.simplilearn.com/tutorials/artificial-intelligence-tutorial/artificial-intelligence-applications> Accessed 1 January 2022.
- 10) British Liver Trust (n.d.) *Hepatitis E*. <https://britishlivertrust.org.uk/information-and-support/living-with-a-liver-condition/liver-conditions/hepatitis-e/>. British Liver Trust. Accessed 7 March 2022.
- 11) Brownlee, J. (2018) *Difference between a Batch and Epoch in a Neural Network*. <https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/>. Machine Learning Mastery. Accessed 14 May 2022.
- 12) Brownlee, J. (2019a) *How to Choose Loss Functions When Training Deep Learning Neural Networks*. <https://machinelearningmastery.com/how-to-choose-loss-functions-when-training-deep-learning-neural-networks/>. Machine Learning Mastery. Accessed 14 May 2022.
- 13) Brownlee, J. (2019b) *Your First Deep Learning Project in Python with Keras Step-by-Step*. <https://machinelearningmastery.com/tutorial-first-neural-network-python-keras/>. Accessed 8 February 2022.
- 14) Brownlee, J. (2021) *How to Choose an Activation Function for Deep Learning*. <https://machinelearningmastery.com/choose-an-activation-function-for-deep-learning/>. Machine Learning Mastery. Accessed 14 May 2022.
- 15) Bupa (n.d.) *Health information A to Z*. <https://www.bupa.co.uk/health-information/a-to-z>. Bupa. Accessed 7 March 2022.
- 16) Coding Demos (2020) *How to Make Android Constraintlayout Scrollable Using Android ScrollView*. [Video] <https://www.youtube.com/watch?v=DpFNfQzhKQM>. Accessed 2 March 2022.
- 17) Firebase (n.d. a) *Products / Build*. <https://firebase.google.com/products-build> Accessed 19 January 2022.
- 18) Firebase (n.d. b) *Firebase Realtime Database*. <https://firebase.google.com/products/realtime-database> Accessed 19 January 2022.
- 19) Firebase (n.d. c) *Firebase Machine Learning*. https://firebase.google.com/docs/ml#cloud_vs_on-device Accessed 19 January 2022.
- 20) Firebase (n.d. d) *Available libraries*. <https://firebase.google.com/docs/android/setup#available-libraries> Accessed 8 May 2022.
- 21) freeCodeCamp.org (2020a) *Android Application Development for Beginners – Full Course*. [Video] <https://www.youtube.com/watch?v=fis26HvvDII>. Accessed 31 January 2022.
- 22) freeCodeCamp.org (2020b) *Android Application Development for Beginners – Full Course (Part 2)*. [Video] <https://www.youtube.com/watch?v=RcSHApwXAQ>. Accessed 31 January 2022.
- 23) GeeksforGeeks (2022) *Android Project folder Structure*. <https://www.geeksforgeeks.org/android-project-folder-structure/?ref=lbp> Accessed 8 May 2022.
- 24) Google Cloud (n.d.) *Cloud API's*. Google. <https://cloud.google.com/apis#section-6> Accessed 19 January 2022.

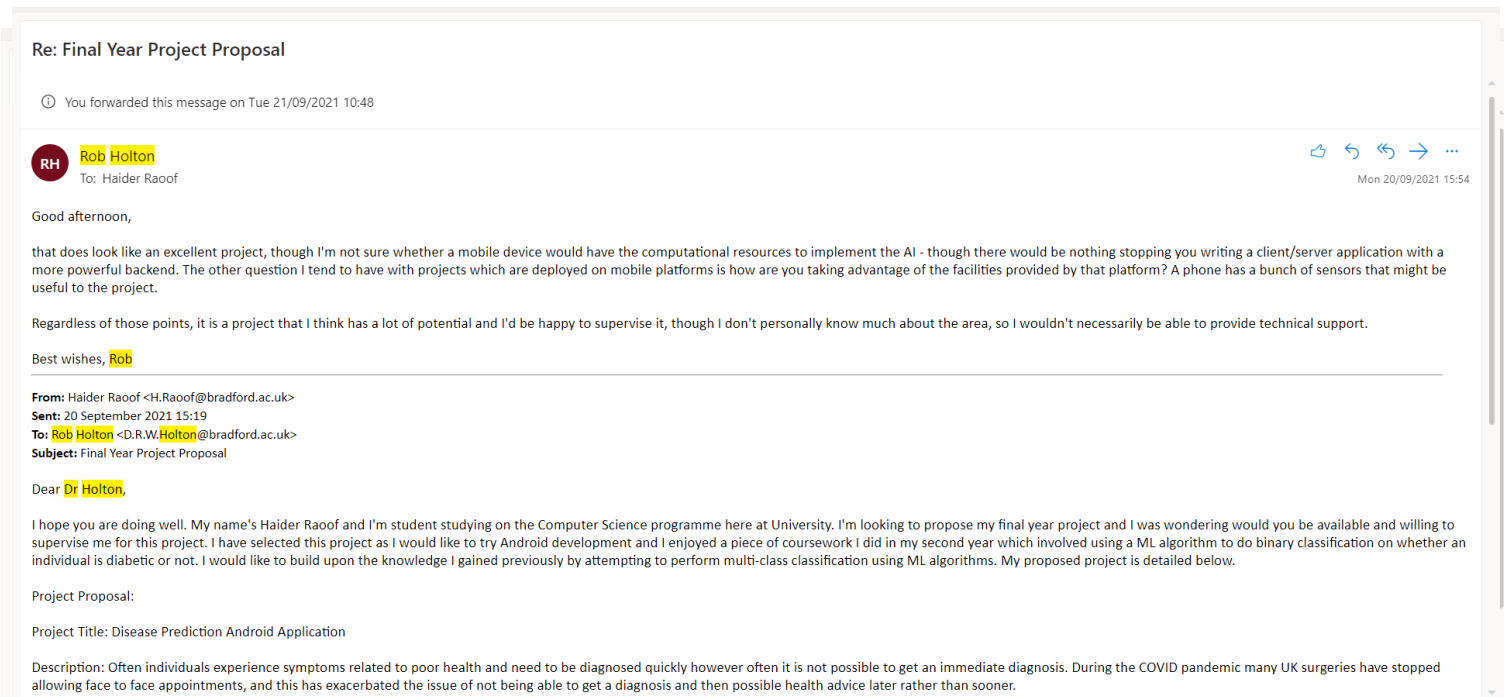
- 25) Google Maps (n.d. a) *Maps result via search term hospital*. <https://www.google.com/maps/search/hospital/>. Google. Accessed 14 March 2022.
- 26) Google Maps (n.d. b) *Maps result via search term GP*. <https://www.google.com/maps/search/GP/>. Google. Accessed 14 March 2022.
- 27) Google Maps (n.d. c) *Maps result via search term pharmacy*. <https://www.google.com/maps/search/pharmacy/>. Google. Accessed 14 March 2022.
- 28) Google Maps (n.d. d) *Maps result via search term opticians*. <https://www.google.com/maps/search/opticians/>. Google. Accessed 14 March 2022.
- 29) GreatLearning (2021) *Label Encoding in Python Explained*. <https://www.mygreatlearning.com/blog/label-encoding-in-python/> Accessed 15 February 2022.
- 30) Healthline. (2018) *Everything You Should Know About Cholestasis*. <https://www.healthline.com/health/cholestasis>. Healthline. Accessed 7 March 2022.
- 31) IBM Cloud Education (2020a) *Artificial Intelligence (AI)*. IBM. <https://www.ibm.com/uk-en/cloud/learn/what-is-artificial-intelligence> Accessed 1 January 2022.
- 32) IBM Cloud Education (2020b) *Neural Networks*. IBM. <https://www.ibm.com/cloud/learn/neural-networks>. Accessed 14 May 2022.
- 33) Islam, M., Wu, C., Poly, T. N., Yang, H, and Li, Y. J. (2018) Applications of Machine Learning in Fatty Live Disease Prediction. In: *40th Medical Informatics in Europe Conference MIE 2018*, Gothenburg, Sweden, 24-26 April 2018. IOS Press. 166-170.
- 34) JetBrains (n.d.) *PyCharm*. <https://www.jetbrains.com/pycharm/>. Accessed 13 May 2022.
- 35) Johns Hopkins Medicine (n.d.) *Alcoholic Hepatitis*. <https://www.hopkinsmedicine.org/health/conditions-and-diseases/hepatitis/alcoholic-hepatitis>. Accessed 7 March 2022.
- 36) Keras (n.d.) *Keras*. <https://keras.io/>. Accessed 13 May 2022.
- 37) Kota, S.D.K. (2020) *Running ML Models in Android Using Tensorflow Lite*. <https://medium.com/analytics-vidhya/running-ml-models-in-android-using-tensorflow-lite-e549209287f0>. Accessed 11 February 2022.
- 38) Microsoft Excel (n.d.) *Microsoft Excel*. <https://www.microsoft.com/en-us/microsoft-365/excel> Accessed 12 May 2022
- 39) Mobile Machine Learning (2020) *Train machine learning model and develop Android Application (Example)*. [Video] <https://www.youtube.com/watch?v=63bCmY5uRto>. Accessed 9 February 2022.
- 40) Moroney, L. (2018) *Architecture required to use TensorFlow model in mobile applications*. [Image] <https://blog.tensorflow.org/2018/03/using-tensorflow-lite-on-android.html> Accessed 20 January 2022.
- 41) NHS (n.d.) *Health A to Z*. <https://www.nhs.uk/conditions/>. NHS. Accessed 7 March 2022.
- 42) NICE (2022) *Adverse drug reactions*. <https://cks.nice.org.uk/topics/adverse-drug-reactions/>. National Institute for Health and Care Excellence. Accessed 7 March 2022.
- 43) Nishadha (2021) *Use Case Diagram Relationships Explained with Examples*. <https://creately.com/blog/diagrams/use-case-diagram-relationships/> Accessed 23 January 2022.
- 44) NumPy (n.d.) *NumPy*. <https://numpy.org/> Accessed 4 February 2022.
- 45) Pandas (n.d.) *pandas*. <https://pandas.pydata.org/> Accessed 4 February 2022.

- 46) Patel, S. B., Yadav, P. K. and Shukla, D. P. (2013). Predict the diagnosis of heart disease patients using classification mining techniques. *IOSR Journal of Agriculture and Veterinary Science* 4(2), 61-64.
- 47) ProgrammingKnowledge (2021) *Firestore Tutorial for Android App development*. [Video]. <https://www.youtube.com/watch?v=SV9pJqR41KI>. Accessed 21 March 2022.
- 48) Russell, S, and Norvig. P. (2021a) Introduction. In Russell, S., and Norvig. P. (editors) *Artificial Intelligence: a Modern Approach*. 4th edition. Pearson Education. 19-53.
- 49) Russell, S, and Norvig. P. (2021b) Learning from Examples. In Russell, S., and Norvig. P. (editors) *Artificial Intelligence: a Modern Approach*. 4th edition. Pearson Education. 669-733.
- 50) Scikit-learn (n.d. a) *scikit-learn Machine Learning in Python*. <https://scikit-learn.org/stable/> Accessed 4 February 2022.
- 51) Sci-kit learn (n.d. b) *Supervised learning*. https://scikit-learn.org/stable/supervised_learning.html. Accessed 13 May 2022.
- 52) Secinaro, S., Calandra, D., Secinaro, A., Muthurangu, V. and Biancone, P. (2021) The role of artificial intelligence in healthcare: A structured literature review. *BMC Medical Informatics and Decision Making* 21(1), 1-23.
- 53) Sims, G. (2019) *I want to develop Android apps — What languages should I learn?* Android Authority. <https://www.androidauthority.com/develop-android-apps-languages-learn-391008/> Accessed 19 January 2022.
- 54) Singh, R. (2019) *Symptom checker dataset*. <https://www.kaggle.com/datasets/rabisingh/symptom-checker?select=Training.csv>. Accessed 14 February 2022.
- 55) SmallAcademy (2021a) *Firestore Login & Register App with Email | Part – 1 | Connecting App to Firestore*. [Video] https://www.youtube.com/watch?v=dMB27peGUrK&list=PLlGT4GXl8_8dm7OeLB5SiVZAPXeSq9i2Z&index=1 Accessed 4 January 2022.
- 56) SmallAcademy (2021b) *Firestore Login & Register App with Email | Part – 2 | Creating Login and Register Activity*. [Video] https://www.youtube.com/watch?v=Bo7fCxYQYdg&list=PLlGT4GXl8_8dm7OeLB5SiVZAPXeSq9i2Z&index=2 Accessed 4 January 2022.
- 57) SmallAcademy (2021c) *Firestore Login & Register App with Email | Part – 3 | Login layout design*. [Video] https://www.youtube.com/watch?v=rtTOQyvvjLo&list=PLlGT4GXl8_8dm7OeLB5SiVZAPXeSq9i2Z&index=3 Accessed 4 January 2022.
- 58) SmallAcademy (2021d) *Firestore Login & Register App with Email | Part – 4 | Register Layout Design*. [Video] https://www.youtube.com/watch?v=5jReLwZBztk&list=PLlGT4GXl8_8dm7OeLB5SiVZAPXeSq9i2Z&index=4 Accessed 4 January 2022.
- 59) SmallAcademy (2021e) *Firestore Login & Register App with Email | Part – 5 | Data Extraction and Validation*. [Video] https://www.youtube.com/watch?v=FPqACcPE6Wo&list=PLlGT4GXl8_8dm7OeLB5SiVZAPXeSq9i2Z&index=5 Accessed 5 January 2022.
- 60) SmallAcademy (2021f) *Firestore Login & Register App with Email | Part – 6 | Register User*. [Video] https://www.youtube.com/watch?v=iPv7auErERo&list=PLlGT4GXl8_8dm7OeLB5SiVZAPXeSq9i2Z&index=6 Accessed 5 January 2022.
- 61) SmallAcademy (2021g) *Firestore Login & Register App with Email | Part – 7 | Login User*. [Video] https://www.youtube.com/watch?v=T-L7hJPXXUk&list=PLlGT4GXl8_8dm7OeLB5SiVZAPXeSq9i2Z&index=7 Accessed 5 January 2022.

- 62) Stevenson, D (2018) *What is Firebase? The complete story abridged*. <https://medium.com/firebase-developers/what-is-firebase-the-complete-story-abridged-bcc730c5f2c0> Accessed 19 January 2022.
- 63) Tech with Tim (2019a) *Python Neural Networks – TensorFlow 2.0 Tutorial – What is a Neural Network*. [Video] https://www.youtube.com/watch?v=OSoDdkle0o4&list=PLzMcbGfZo4-lak7tiFDec5_ZMIItiIfmj. Accessed 4 February 2022.
- 64) Tech with Tim (2019b) *Python Neural Networks – TensorFlow 2.0 Tutorial – Loading & Looking at Data*. [Video] https://www.youtube.com/watch?v=wu9IH1Xvdd4&list=PLzMcbGfZo4-lak7tiFDec5_ZMIItiIfmj&index=2. Accessed 4 February 2022.
- 65) Tech with Tim (2019c) *Python Neural Networks – TensorFlow 2.0 Tutorial – Creating a Model*. [Video] https://www.youtube.com/watch?v=cvNtZqphr6A&list=PLzMcbGfZo4-lak7tiFDec5_ZMIItiIfmj&index=3. Accessed 6 February 2022
- 66) Tech with Tim (2019d) *Python Neural Networks – TensorFlow 2.0 Tutorial – Using the Model to Make Predictions*. [Video] https://www.youtube.com/watch?v=RqLD1INA_cQ&list=PLzMcbGfZo4-lak7tiFDec5_ZMIItiIfmj&index=4 Accessed 6 February 2022
- 67) TensorFlow (2021a) *TensorFlow Lite converter*. <https://www.tensorflow.org/lite/convert/index> Accessed 20 January 2022.
- 68) TensorFlow (2021b) *TensorFlow Lite inference*. <https://www.tensorflow.org/lite/guide/inference> Accessed 20 January 2022.
- 69) TensorFlow (2022) *TensorFlow Lite guide*. <https://www.tensorflow.org/lite/guide>. Accessed 13 May 2022.
- 70) TensorFlow (n.d.) *An end-to-end open source machine learning platform*. <https://www.tensorflow.org/> Accessed 20 January 2022.
- 71) tutorialspoint (n.d.) *System Testing*. https://www.tutorialspoint.com/software_testing_dictionary/system_testing.htm Accessed 23 January 2022.
- 72) Uddin, S., Khan, A., Hossain, M.E., and Moni M.A. (2019) Comparing different supervised machine learning algorithms for disease prediction. *BMC Medical Informatics and Decision Making* 19(1), 1-16.

9 Appendices

Appendix A – The original specification of the project as agreed with, Dr Rob Holton, my supervisor via email correspondence.



Appendix B – A statement about the relevance of my project to the Computer Science degree programme I am studying.

The project undertaken matches my degree programme in the following manner. In my project there was the use of mathematical methods to design and implement a solution to the problem of predicting disease from a user's symptom input in an Android application in order to provide an immediate diagnosis. Mathematical methods were utilised in the form of supervised machine learning which was done using an artificial neural network. A general understanding of the mathematics used for forward propagation and a back propagation was crucial, in order to understand the how an artificial neural network performs supervised learning and updates it's weights. An artificial neural network was successfully trained and used within an Android application to predict disease based on user's symptom input.

Appendix C – Python script, predictionmodel.py, used to build, train, and test an ANN model which is then converted into a TensorFlow Lite model for the Disease Prediction Android Application

```
e Refactor Run Tools VCS Window Help testEnv - predictionmodel.py

predictionmodel.py x
1 import pandas as pd
2     import tensorflow as tf
3     import numpy as np
4     from tensorflow import keras
5     from sklearn.model_selection import train_test_split
6     from tensorflow import lite
7
8     # Classify a persons disease based upon the symptoms they have
9     # 133rd column of the datasets is 'prognosis'
10    # There are 41 categories/diseases/labels in this column
11    # 133rd column is the label the model will predict i.e. the disease a person has
12    # This example uses 80:20 train test split
13
14    # Read the dataset into a dataframe object
15    dataset = pd.read_csv("Training.csv")
16
17    # Preprocessing of dataset
18
19    # Check to see if there is the existence of any null values in the dataset
20    print("Are null values present in the training dataset? ", dataset.isnull().values.any())
21    # Returns false so there is no null values in the dataset
22
23    # Find out the dimensions of the dataset
24    # The datasets dimensions are 133 columns and 4020 rows of data
25    print(dataset.info())
26
27    # Find out the number of unique disease values in the dataset prognosis column
28    # 41 distinct diseases in dataset
29    numberOfDiseasesTraining = len(pd.unique(dataset['prognosis']))
30    print("Number of diseases in dataset:", numberOfDiseasesTraining)
31
32    # To prevent the model just doing pattern recognition and not generalising well
33    # I will drop duplicate rows from the dataset
34
35    # Number of duplicated rows of data in dataset
36    # 4616 rows
37    print("Number of duplicate rows in dataset: ", dataset.duplicated().sum())
38
39    # Remove duplicated rows whilst keeping the first occurrence of a row
40    dataset.drop_duplicates(subset=None, keep="first", inplace=True)
```

```

Refactor Run Tools VCS Window Help testEnv - predictionmodel.py

predictionmodel.py
37 print("Number of duplicate rows in dataset: ", dataset.duplicated().sum())
38
39 # Remove duplicated rows whilst keeping the first occurrence of a row
40 dataset.drop_duplicates(subset=None, keep="first", inplace=True)
41
42 # Find out the dimensions of the dataset after removing duplicates
43 # The datasets dimensions are 133 columns and 304 rows of data after removing duplicates
44 print(dataset)
45
46 # The 132 symptom columns of the dataset are nominal data with values of either 0 and 1
47 # to signify the presence of a symptom, 0 means not presenting whilst 1 means it does present in a person
48
49 # The prognosis column of the dataset is categorical data that is in a text format. It's not in a suitable
50 # format to predict disease with the model, so label encoding to put the diseases into a numerical data format will be done
51
52 # Use the pandas replace function to label encode the prognosis column of dataset - found all the disease names using
53 # the Testing.csv file provided from source of data
54 dataset.replace({'prognosis':{'Fungal infection':0, 'Allergy':1, 'GERD':2, 'Chronic cholestasis':3,
55 'Drug Reaction':4, 'Peptic ulcer disease':5, 'AIDS':6, 'Diabetes':7, 'Gastroenteritis':8, 'Bronchial Asthma':9,
56 'Hypertension':10, 'Migraine':11, 'Cervical spondylosis':12, 'Paralysis (brain hemorrhage)':13, 'Jaundice':14,
57 'Malaria':15, 'Chicken pox':16, 'Dengue':17, 'Typhoid':18, 'hepatitis A':19, 'Hepatitis B':20, 'Hepatitis C':21,
58 'Hepatitis D':22, 'Hepatitis E':23, 'Alcoholic hepatitis':24, 'Tuberculosis':25, 'Common Cold':26, 'Pneumonia':27,
59 'Dimorphic hemmorhoids(piles)':28, 'Heart attack':29, 'Varicose veins':30, 'Hypothyroidism':31, 'Hyperthyroidism':32,
60 'Hypoglycemia':33, 'Osteoarthritis':34, 'Arthritis':35, '(vertigo) Paroymsal Positional Vertigo':36, 'Acne':37,
61 'Urinary tract infection':38, 'Psoriasis':39, 'Impetigo':40}}, inplace=True)
62
63 # Get labelled column from dataset and create labelled dataset
64 dataset_labels = dataset.pop('prognosis')
65
66 # Split dataset into train test sets
67 # 80:20 split for training and test set
68 X_train, X_test, y_train, y_test = train_test_split(dataset, dataset_labels, test_size=0.2)
69
70 # Done to make the datasets in numpy array format to be used for the model.fit function
71 # Model can't use pandas dataframe objects, has to use numpy arrays
72 training_dataset = np.asarray(X_train)
73 training_labels = np.asarray(y_train)
74 testing_dataset = np.asarray(X_test)
75 testing_labels = np.asarray(y_test)
76

```

```

76
77 # Labels of diseases so when testing we have meaningful output after prediction done in this python file
78 diseaseLabels = ['Fungal infection','Allergy','GERD','Chronic cholestasis','Drug Reaction', 'Peptic ulcer disease',
79 'AIDS','Diabetes ', 'Gastroenteritis','Bronchial Asthma','Hypertension ', 'Migraine','Cervical spondylosis',
80 'Paralysis (brain hemorrhage)','Jaundice','Malaria','Chicken pox','Dengue','Typhoid','hepatitis A','Hepatitis B',
81 'Hepatitis C','Hepatitis D','Hepatitis E','Alcoholic hepatitis','Tuberculosis','Common Cold','Pneumonia',
82 'Dimorphic hemmorhoids(piles)','Heart attack','Varicose veins','Hypothyroidism','Hyperthyroidism', 'Hypoglycemia',
83 'Osteoarthritis','Arthritis','(vertigo) Paroymsal Positional Vertigo','Acne','Urinary tract infection',
84 'Psoriasis','Impetigo']
85
86 # Define the model's architecture (layers) - an artificial neural network
87 # 3 layers with a Dense layer at every layer, so it is a fully connected network
88 model = keras.Sequential([
89     # Input layer first
90     # The number of input features is specified by input_dim i.e. the model expects rows of data with 132 features
91     # Dense layer - every neuron is connected to every neuron in the next layer
92     # Activation function is rectified linear unit
93     keras.layers.Dense(132, input_dim=132, activation="relu"),
94     # First hidden layer, also a dense layer so every neuron is connected to every neuron in previous and next layer
95     # Number of neurons chosen for the hidden layer is 64
96     # Activation function is also rectified linear unit
97     keras.layers.Dense(64, activation="relu"),
98     # Final layer is output layer with 41 neurons as we are classifying 41 classes (diseases)
99     # Activation function softmax calculates values for each neuron so that the sum of those values equals 1
100     # i.e. the probability of the network thinking it's a certain class
101     keras.layers.Dense(41, activation="softmax")
102 ])
103
104 # Compile the model
105 # Adam is the optimization algorithm used to update weights in the neural network
106 # Loss function tells us how different the predicted result is from the actual result
107 # Used to monitor the error by the neural network and should reduce as the weights update during and after each epoch
108 # Accuracy metric shows the percentage of predictions that identified the correct label (disease)
109 model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
110
111 # Train model
112 # Epochs is the number of iterations through all the rows of the training dataset
113 # Batch size is the number of dataset rows considered, within an epoch, before the model weights are updated
114 model.fit(training_dataset, training_labels, epochs=30, batch_size=81)
115

```

