School of Computer Science and Engineering

Faculty of Engineering

The University of New South Wales

Module for Atrial Fibrillation Detection on a Smartphone

by

Silvia

Thesis submitted as a requirement for the degree of

Bachelor of Engineering in Computer Computer Science and Engineering (Honours)

Submitted: November 2022

Supervisor: Dr. Peter Brown

Student ID: z5386349

Abstract

This thesis aims to develop a solution that allows a Deep Learning (DL) Keras model to be able to work and make predictions based on Electrocardiogram (ECG) measurements on a smartphone. This enables the implementation of continuous AF monitoring which can lessen the risk of silent AF.

There are numerous studies that have developed their own ways of integrating AF monitoring system on a smartphone such as KardiaMobile by AliveCor and ECG for Apple Watch by Apple. However, none of them have implemented a feature that continuously monitor for AF locally.

The thesis project was able to successfully build a solution by converting the Keras model into a TensorFlow JavaScript Graph Model (TFJSGM), within the React Native (RN) development environment. Prediction accuracy is also shown to be on par with Keras's.

Acknowledgements

I would like to first and foremost thank my supervisor, Dr Peter Brown for his utmost guidance, and support throughout 2022. Software engineers Stephen Ong, Wincent Winarko, and Zhuoying Li, for providing numerous technical support and resources on mobile development. Dr Reza Argha for his expertise in machine learning. My assessor Dr Nigel Lovell, project manager Brice Lenfant, and other thesis students who participated in the weekly meetings. Finally, to my family and friends overseas and in Australia for their mental and emotional support.

Abbreviations

AF Atrial Fibrillation
API Application Programming Interface
AIT Austrian Institute of Technology
DL Deep Learning
NSR Normal Sinus Rhythm
OA Other Arrhythmia
TN Too Noisy
AF Atrial Fibrillation
DL Deep Learning
ECG Electrocardiogram
GSBmE Graduate School of Biomedical Engineering
HDL Hybrid Deep Learning
JS JavaScript
ML Machine Learning
NSW New South Wales
RN React Native
TCC TeleClinical Care
TF TensorFlow
TFJS TensorFlow JavaScript
TFJSGM TensorFlow JavaScript Graph Model
TFJSRN TensorFlow JavaScript React Native
TFLite TensorFlow Lite
UNSW University of New South Wales

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Atrial Fibrillation (AF) is one of the most common types of arrhythmias, which means irregular heartbeat or improper beating of the heart [FCG08]. AF increases one's risk to stroke and heart failure due to blood clotting in the heart [SSS14]. Given that AF is often asymptomatic [BBC+17], in other words, shows no symptoms, it may result in the risk of having it being underdiagnosed. Frequent monitoring via Electrocardiogram (ECG) may help doctors in aiding patients with the right medication or treatment [GKD+16, PTC+03] in the right time.

ECG is a medical recording of the heart's electrical signals [SY22]. It is non-invasive, safe, and the least costly way of measuring the heart's contractions. In this thesis project, the ECG measurement will be fed into a Deep Learning (DL) algorithm.

Deep Learning is a subset of machine learning that lets a machine learn from representation of data with multiple layers [LBH15]. To put it simply, if a picture of a cat were to be fed into a machine learning algorithm, in order to be able to determine which are the whiskers, ears, eyes, etc., the machine learning algorithm references back to what was extracted by human. On the other hand, a DL algorithm is able to make the prediction(s) by itself. Another example can be seen within Google Translate. The sentence "Cuaca hari ini sangat cerah" in Bahasa Indonesia if directly translated to English without DL will be "Weather today is very bright", which contextually wrong.

However, Google's DL algorithm is able to translate it to "The weather today is very sunny" (as seen in Figure 1.1) which is a more accurate translation.

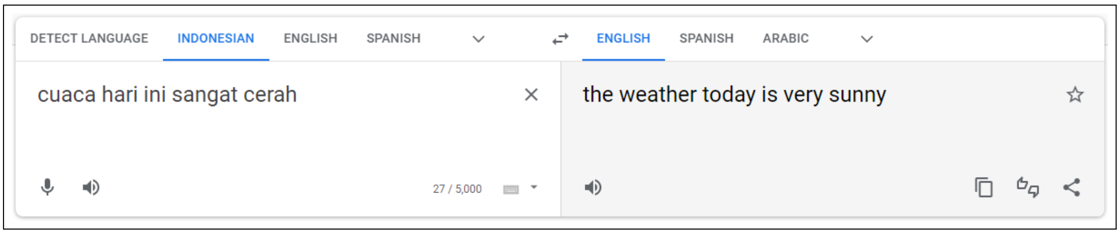

Figure 1.1: Deep Learning in Google Translate

1.1 Problem Statement

There is currently no existing AF detection apps that supports continuous monitoring for AF on a smartphone. Being able to continuously monitor a patient's heart activity is crucial as it: [CCP12, VLK$^+$22]

- Aids in detecting silent AF
- Helps in early diagnosis of AF
- Assists clinicians in providing early interventions via tailored treatment or therapies

Furthermore, existing AF detectors on smartphones relies on a backend. This method of detection requires a connection for it to be able to run and output a prediction. In a hypothetical scenario where a multitude amount of patients enables the continuous AF detection feature, there might be a possibility where the server can slow down. A server slow down may also cause a delay on a patient's detected outcome, which for AF patients are critical.

1.2 Aim

This thesis project aims to solve the problem of AF detection that server reliance to run predictions. The thesis's goal is to integrate an app module that allows AF to be detected locally by the smartphone itself. Being able to screen for AF independently without relying on a server enables the implementation for continuous monitoring of ECG measurements.

Chapter 2

Background

This section covers the background, and literature reviews on: (1)Similar works that are able to detect AF on a portable device. (2)Prior works which plays an important role in the development process of the module.

2.1 Similar Work

Detecting AF on a smartphone have been done previously with success. This section will discuss on those applications, how accurate it is according to researchers, its impacts, and what improvements can be done.

2.1.1 AliveCor KardiaMobile and Kardia App

KardiaMobile is a portable ECG device made by AliveCor. In comparison to the traditional way of getting ECG measurements, AliveCor was able to develop an ECG measuring device that only requires patients to place their fingers for 30 seconds on KardiaMobile, which afterwards sends its recording to their cross-platformed Kardia app in a smartphone device [MC21].

A study [WKE+20] have experimented in using the traditional ECG and the AliveCor KardiaMobile on 99 patients. Results from this experimentation shows that AliveCor's ECG monitor was able to have a highly accurate detection of AF, yielding a sensitivity (correctly identifying patients with AF [SHT20]) of 100% and specificity (correctly identifying patients without AF [SHT20]) of 94%.

There are, however, a few drawbacks to KardiaMobile and Kardia. In the app, patients may need to pay a fee if they were to request for a clinician's input of the ECG measurement they took [Ali21]. Patients are also limited to their latest record and are only able to view the records prior to that if they pay a subscription fee [Ali17]. Finally and most importantly, it does not support continuous monitoring [Slo18], a feature that this thesis project aims to work towards to.

2.1.2 Apple Watch and ECG App

The Apple Watch have been studied in the Apple Heart Study, in where the Apple Watch demonstrates to have a high positive predictive value of 0.84 and confidence interval (value that accurately reflects $419,297$ participants [CE09]) of 95%from its electrodes that generates single-lead ECG to detect AF [PMH+19].

The ECG app is easy to use, the readings from the sensors are reliable, and the watch support continuous heart rate monitoring via their optical light sensor [Jov15], not to be confused with continuous ECG monitoring for-which the watch does not support [Jov15], which again, is a feature that this thesis project aims to work on. On top of that, a patient must be of the age 22 and above to be able to use the ECG app [KKM+22] and is only compatible with Apple devices (not cross-platformed) [App22]. This thesis project, however, aims to cater to more patients on more than one platform, and not limiting to a certain age.

2.1.3 Comparison of PPG and single-lead ECG to Detect AF

The study aims to compare the diagnostic performance of photoplethysmogram (PPG) and single-lead ECG's proprietary smartphone apps' AF detection algorithm. Results from this study demonstrated that performance from both are equivalent. Where the specificity, sensitivity, accuracy of PPG are respectively, 94.1%, 97.6%, and 96.4%. On the other hand, ECG's are, 91.1%, 95.7%, and 94.1% [MBR+21].

Although the research does not correlate much with the current thesis project, the future works for this thesis does. The interchangeability of PPG and single-lead ECG means that if a PPG algorithm were to be made, it can be a good area to research on, and potentially another module to be built on top of this AF module. That being said, the study too does not implement continuous AF monitoring, which this thesis aims to build.

2.2 Prior Work

The AF detection module that will be made for this thesis project, will leverage from prior but still ongoing works that was a result for the combined efforts of researchers, students, and software engineers from UNSW and other universities. How the AF detection module can benefit from these findings will also be stated.

2.2.1 Pre-Trained DL Model that Assesses AF

A group of researchers and medical students have examined ECG classification algorithms and its arrhythmia classification accuracy. The algorithms used in this study are the Hybrid deep-learning (HDL) based algorithm, where the model is able to classify ECG recordings into four classes, one of them being AF; and genetic algorithm, that maximizes the average of F1-scores (predictive performance) which then will be used to select an optimal subset of classifiers to ensemble the classification for the DL AF detection algorithm [AA+].

TABLE III: **Results achieved by different AI-based models on a dataset of 636 STSL ECG traces captured by CONTEC PM10 ECG device.**

Classification Method	NSR	AF	OR	TN	(NSR+AF+OR)/3
		F_1 score: mean (SD)			F_1 score: mean (SD)
Pre-trained HDL Model	0.791	0.905	0.471	0.342	0.722
Pre-trained DL Model& Lead I ECGs	0.775	0.884	0.411	0.375	0.690
Pre-trained DL Model& Lead II ECGs	0.832	0.937	0.520	0.333	0.763
Pre-trained DL Model& Lead III ECGs	0.763	0.892	0.480	0.333	0.712
Pre-trained Method in [18]	0.819	0.833	0.449	0.400	0.701
Re-trained DL Model (classification layers only)	0.963 (0.017)	0.974 (0.015)	0.923 (0.005)	0.696 (0.125)	0.953 (0.009)
Re-trained DL Model (ensemble classifiers only)	0.971 (0.024)	0.972 (0.024)	0.933 (0.059)	0.676 (0.172)	0.961 (0.013)
Re-trained DL Model (entire DL model)	0.967 (0.023)	0.972 (0.022)	0.920 (0.053)	0.635 (0.148)	0.953 (0.032)
Re-trained HDL (entire DL model + ensemble classifiers)	0.971 (0.024)	0.972 (0.024)	0.933 (0.059)	0.676 (0.172)	0.959 (0.034)
Method in [18] trained on CinC dataset and our dataset	0.855 (0.050)	0.934 (0.051)	0.671 (0.021)	0.483 (0.173)	0.820 (0.023)
Method in [18] trained on our dataset	0.884 (0.041)	0.917 (0.012)	0.725 (0.078)	0.300 (0.184)	0.842 (0.030)

DL: deep learning, HDL: hybrid deep learning trained, SD: standard deviation, NSR: normal sinus rhythm, AF: atrial fibrillation, OR: other rhythms, TN: too noisy.

Figure 2.1: F1 scores of pre-trained model versus re-trained model [AA+]

As shown in Figure 2.1, by applying the transfer learning strategy, which is a process where a model for a problem is reused for another relating problem [WKW16], the study was able to increase the F1 score for both HDL and DL models from 0.884 0.937 to 0.972 0.974, outperforming conventional ECG classifiers.

By having more accurate predictions and lesser false positives, the AF detection module for smartphones may aid doctors in giving appropriate diagnosis and treatment for the patient in need.

2.2.2 TeleClinical Care (TCC) - Jadeite

TCC – Jadeite is an app designed in collaboration between the Graduate School of Biomedical Engineering (GSBmE) at UNSW Sydney, New South Wales (NSW) Government Health, and Austrian Institute of Technology (AIT). The app helps hospitals to monitor a patient's condition at home.

Patients are able to share their information simply by inputting information such as their weight, blood pressure, and oxygen saturation into the TCC – Jadeite app. These measurements are gathered via the weight scales, blood pressure monitor, and pulse oximeter machine that was given to the patient upon visiting the hospital. Patients will receive daily notifications to enter their measurements, and in situations where patients might forget to do so, a staff from the hospital may call the patient to check on their well-being. Figure 2.2 shows the layout of the app.

By adding in an AF detection module for the TCC – Jadeite app, both patients and health workers will be able to communicate on possible detected AF, thus aiding in better prescription and treatment for the patient.

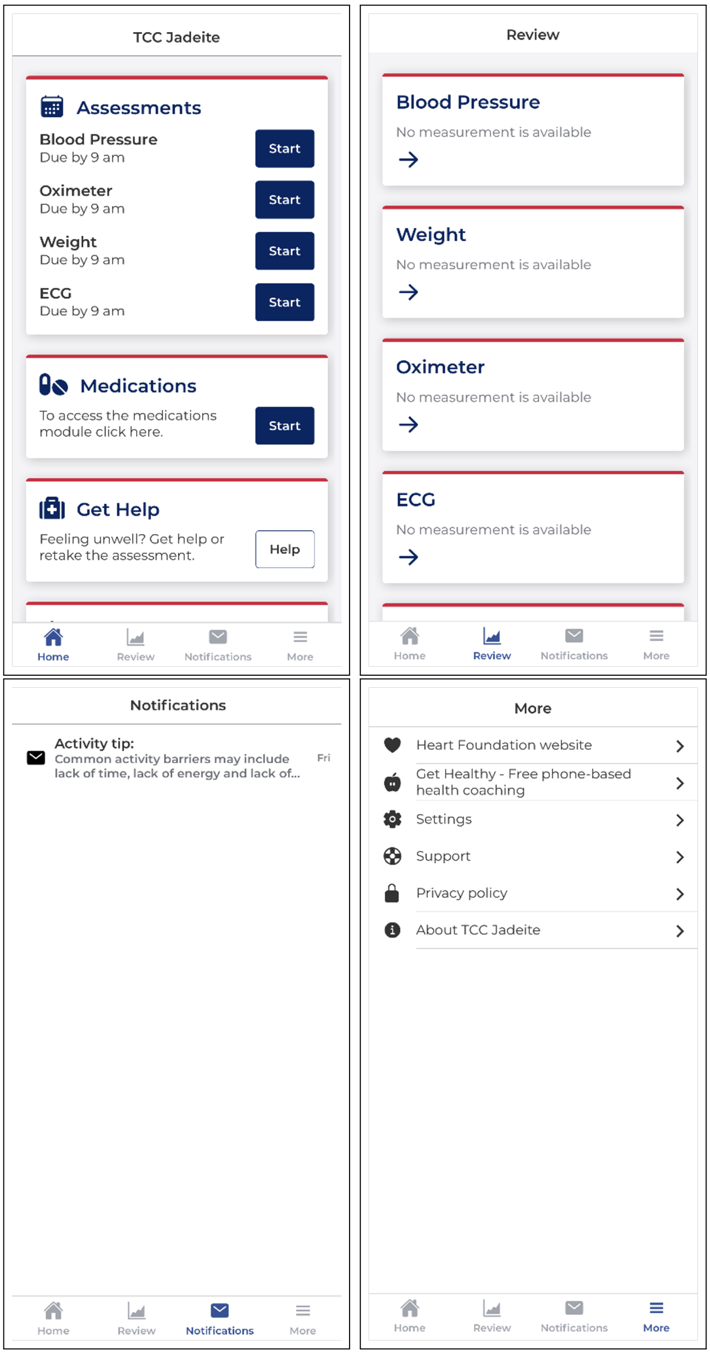

Figure 2.2: TCC – Jadeite app: home tab (upper left), review tab (upper right), notifications tab (lower left), more tab (lower right)

Chapter 3

Methodology

3.1 Functional Requirements

As shown in Figure 3.1, the requirements shown is of that enables a user that wants to use the module's features to be ensured that all crucial functionalities are covered.

Table 3.1: Functional Requirements

No.	**Description**
FR1	User should be able to toggle continuous AF detection module on and off
FR1	User should be able to view all past records of their ECG readings result
FR1	User should be able to filter past records of their ECG readings result
FR2	Module should notify user for detected AF
FR3	Module should be able to use DL model to detect for AF in the ECG recording
FR4	Module should be able to continuously monitor ECG measurement
FR5	Module should be able to detect possible AF from ECG readings

3.1.1 User Stories

Figure 3.2 shows a patient-centric view of the app. Features are told in the form of what the users want, and what can that feature benefit into.

Table 3.2: User Stories

No.	**As a/an**	**I want to**	**So that**
P1	Patient	Be able to monitor ECG continuously	I can be notified of a possible silent AF
P2	Patient	Be able to see an overview of the ECG result upon opening the app	I can take a look of my current status whenever I want to
P3	Patient	Be notified for possible detected AF	I know when to take precaution
P4	Patient	Be able to check past records of ECG readings result	I know when was the last time I have possible irregularities for AF
P5	Patient	Be able to filter my past records of ECG readings result	I can monitor the progression of my condition/symptoms

3.2 Non-Functional Requirements

The non-functional requirements are the desired state the module should be in. As shown in Figure 3.3, criteria involves user interface, experience, privacy, app performance, documentation, and testing.

Table 3.3: Non-Functional Requirements

No.	**Description**
NFR1	Consistent user interface and user experience. Allowing first time users to easily, and readily access the module
NFR2	User privacy. User's detection outcome to only be known by the user itself unless the user shares
NFR3	Acceptable performance. Allowing optimal performance of the whole prediction process.
NFR4	Well-documented. Enabling future developers to easily understand the code, and structure.
NFR5	Well-tested. Allowing the module to not be riddled with messy bugs, and so that module works as intended.

3.3 Agile Kanban

Agile is an iterative approach to project management and software development, so instead of doing big things at once, agile allows developers to work in small increments. In Agile Kanban, tasks will be divided into five sections: backlog, to put ideas and features that may or may not be implemented into the project; to do, to put features from backlog that are planned to be worked on; in progress, the features that are being worked on; testing, a working feature that is ready and waiting to be tested; done, where the feature is working, tested, and ready for deployment. Agile Kanban's flexibility with due dates, modifications, and accommodation to varying priorities tasks, makes it great for solo developers.

3.4 Timeline

The Figure 3.1 shows the expected progression of the thesis. By the end of the timeline, it is expected for the AF module to be implemented into TCC – Jadeite. As well as for the DL model to be able to fully function locally in a smartphone with good AF prediction accuracy.

3.5 Module Technologies

This section covers the main technologies used during the development of the AF module.

3.5.1 React Native

React Native (RN) is a framework used to develop multi-platformed apps [Pla22], which in this case is useful for this thesis project that aims to build an app available for iOS and Android.

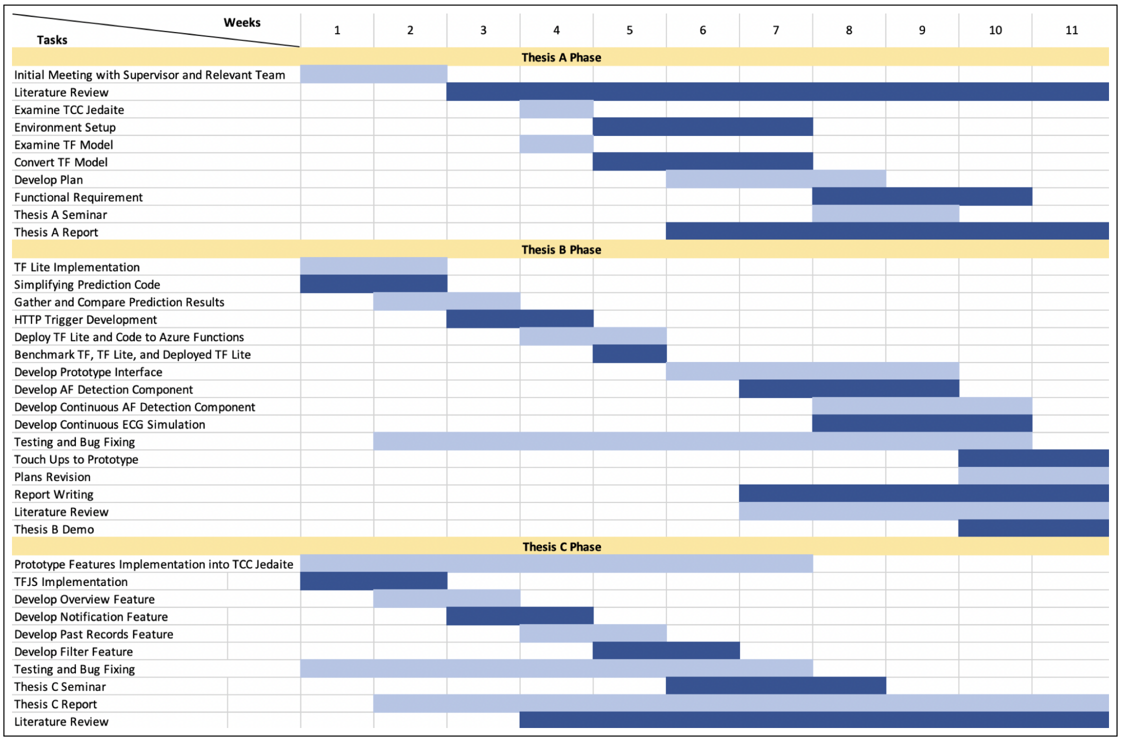

Figure 3.1: Thesis project timeline Gantt chart

3.5.2 TensorFlow

TensorFlow (TF) is a machine learning software library that is used primarily for deep learning applications [AAB+15]. In this case, the library has been previously used to develop an algorithm that has produced the DL model [AA+] that will be used in the conversion.

3.5.3 TensorFlow JavaScript

TensorFlow JavaScript (TFJS) is a JavaScript library that enables ML capabilities [AAB+15, STA+19]. These features includes developing ML, running models, and retraining models. Although web-based, it is interlinked to React Native extension of it.

3.5.4 TensorFlow JavaScript React Native

TFJS React Native (TFJSRN) is an extension of TFJS that allows bundling of resource [AAB+15, Ass20]. This would be further covered in Section 4.1.4.

3.6 Other Technologies

This section covers the other technologies used during experimentation via a different approach of developing the AF module.

3.6.1 TensorFlow Lite

TensorFlow Lite (TFLite) is a cross-platform deep learning framework where the main usage is to convert a pre-trained TF model into a TFLite model for the purpose of speed and storage optimization [AAB+15, Shu20].

3.6.2 Azure Functions

Azure Functions is Microsoft's serverless computing service, where machine resources are allocated dynamically [Mic22, CSP+15]. It triggers upon incoming request and cuts down on resources when idle for some time.

Chapter 4

Implementation

4.1 AF Module Development

4.1.1 Model Conversion

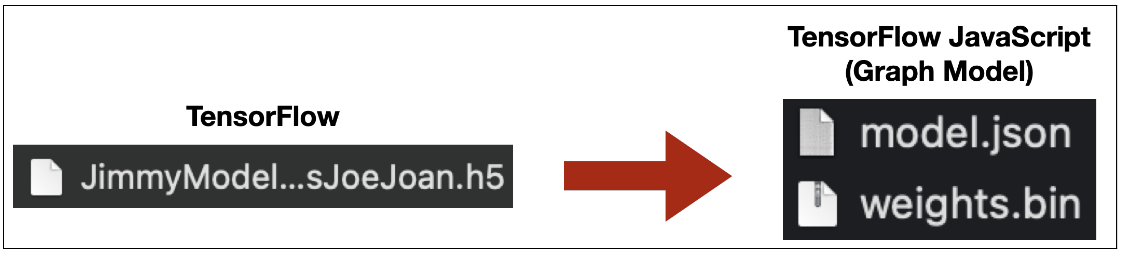

Figure 4.1: Keras to TFJS conversion

For TFJS to be able to make a prediction, a `model.js` and `weights.bin` of type bin file is needed (shown in Figure 4.1). Thus, for the conversion process, the `tfjs-convertor` [Ten22c] package is used. In the upcoming conversion, two output formats will be given: Graph Model, and Layers Model. For this project, Graph Model is chosen as Layers Model is not compatible with the custom classes defined in the Keras model [AA+]. Steps for conversion via the wizard are as follows: [Tenc]

1. In the command line run `tensorflowjs_wizard`
2. Type in the path to the eKeras file (i.e,. `score.h5`)

3. Select `Keras (HDF5)` as input format
4. Select `Tensorflow.js Graph Model` as output format
5. Select `No compression (Higher accuracy)` for compression type
6. Press enter (do not modify the value) when prompted to enter shard size
7. Press enter when prompted to enter metadata
8. Type in output directory
9. Once finished, go to the output directory
10. In the command line type `cat group1-shard1of33.bin... > weights.bin` to merge all weights into a single file.

4.1.2 Loading the Model

Upon launching the app, the TFJS model should be loaded right away. This is so that the app does not need to load the model repeatedly before each prediction. The loaded model is then set, and using the selector, it can now be selected from anywhere within the app. Figure 4.2 shows the code to load the model, and Figure 4.3 shows the variable that allows the model value to be retrieved.

```
async function loadModelAF() {
  const modelJson = require('../../modules/tcc-af/screens/Detector/models/model.json');
  const modelWeights = require('../../modules/tcc-af/screens/Detector/models/weights.bin');
  return await tf.loadGraphModel(bundleResourceIO(modelJson, modelWeights));
}

function* loadModel() {
  try {
    const model = yield call(loadModelAF);
    yield put(TFLoadModelAF(model));
    console.log('AF model ready');
  } catch (e) {
    Log.captureException(e);
  }
}
```

Figure 4.2: Loading of graph model

```
export const getPredictedAF = ({ app: { predictionsAF } }: ApplicationState) =>
  predictionsAF;
```

Figure 4.3: Model selector

4.1.3 Signal Filtering

Python Dependencies

The pre-processing procedure relies on some python dependencies with no equivalent alternative in React Native. Majority of it being that helps in filtering and normalising the signal. The python dependencies used are as follows:

- `numpy` → a Python library for scientific computing [HMvdW+20]
- `scipy.signal` → a toolbox for signal processing [VGO+20]
  - `resample` → the process of changing an existing signal's sampling rate [com]
  - `butter` → a type of filter that processes signals in a way that flattens the frequency to as flat as possible [But30, com]
  - `sosfiltfilt` → a forward-backward digital filter using cascaded second-order sections [com]

Filtering and Normalisation

Within the python environment, the process of filtering, and normalising the signal will occur. In the case of Figure 4.4, it uses the Butterworth filter followed by a forward-backward digital filter, and finally normalised. Once everything is done, file is saved.

Output

The final output of the file after the filtration, can be in seen in the following Figure 4.5. The raw sample file of 7500 lines has been significantly reduced to just 3000 lines, a

```python
def apply_filter(sig_array):
    sos = butter(11, (0.014, 0.3), 'bandpass', output='sos')
    filtered = sosfiltfilt(sos, sig_array)
    filtered = filtered / np.amax(filtered)
    if np.amin(filtered) < -1.0:
        filtered = filtered / abs(np.amin(filtered))
    return filtered
```

Filter (Python + Scipy)

```python
def normalise_length_zero_pad(sig, NL):
    signal_length = len(sig)
    if signal_length > NL:
        interval_start = (signal_length - NL) / 2
        return sig[np.int64(np.floor(interval_start)):signal_length - np.int64(np.ceil(interval_start))]
    elif signal_length < NL:
        pad_length = (NL - signal_length) / 2
        return np.pad(sig, (np.int64(np.floor(pad_length)), np.int64(np.ceil(pad_length))), 'constant')
    else:
        return sig
```

Normalise (Python + Numpy)

Figure 4.4: Filteration and normalisation process

60% decrease. However, it can also be observed that file size has increased 185.496% from 26.2 kilobytes to 74.9 kilobytes.

```
7500 lines (7500 sloc) | 26.2 KB

14
14
16
10
10
13
10
11
12
12
```

Before Filter + Normalise

```
3000 lines (3000 sloc) | 74.9 KB

1.343919709324836731e-02
-6.835613748989999294e-04
-3.686237707734107971e-02
-2.098015323281288147e-02
1.896347291767597198e-02
6.205459311604499817e-02
6.982202827930450439e-02
4.162625968456268311e-02
1.799553446471691132e-02
-9.775774553418159485e-03
```

After Filter + Normalise

Figure 4.5: Signal filtration output. Before(left), and after(right)

4.1.4 Process, Predict, Result

React Native Dependencies

- `@tensorflow/tfjs` → is a library that enables ML development in JS [Ten22a]
  - `expandDims` → is an operation that is adds an outer dimension to a single element [Tena]
  - `squeeze` → is an operation that discards dimensions of size one [Tena]
  - `arraySync` → is an operation that returns a tensor data as nested array [Tena]
  - `argMax` → is an operation that returns a tensor's set of element's maximum elements' indices [Tena]
  - `max` → is an operation that returns a tensor's maximum value across its dimension [Tena]
  - `loadGraphModel` → is an operation that loads a graph model given the path of the model definition [Tena, Ten22a]
- `@tensorflow/tfjs-react-native` → is an extension of TFJS that provides GPU accelerated execution of TFJS [Tena, Ten22a, Ten22b].
  - `bundleResourceIO` → is a IOHandler that supports loading of statically bundled models [Tenb]

Flow

Inside the app's RN environment, the measurements will undergo a final pre-processing procedure. Once done, that output will then be sent to be predicted using the TFJSGM. Afterwards, the prediction results will be compared to a threshold that determines whether the outcome is reliable or unreliable.

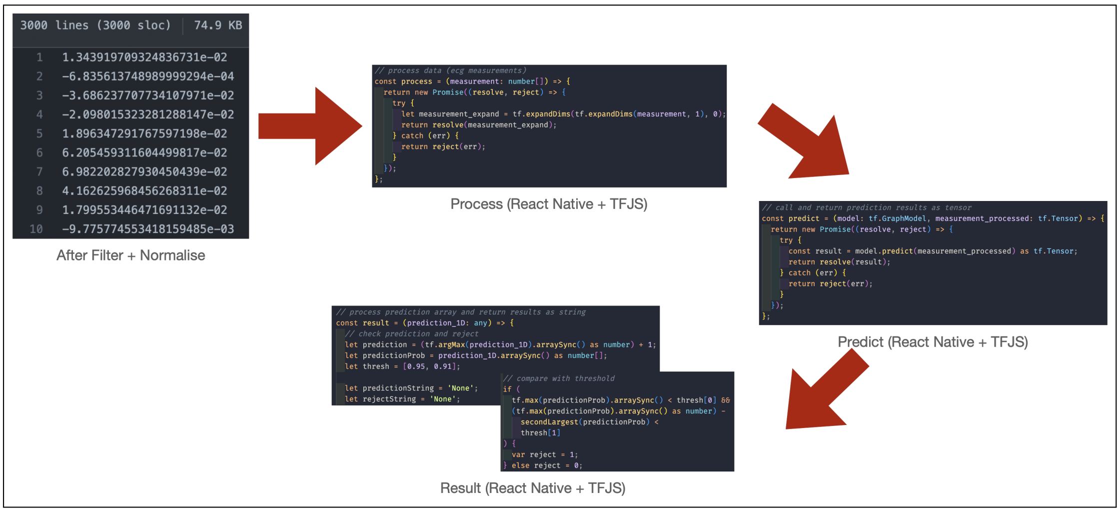

Figure 4.6: Flow of processing, predicting, and outputting

Process

Figure 4.7 shows the final pre-processing step that takes place in RN. This function takes in a parameter of ECG measurements. Using the TFJS library, the procedure expands the dimension twice, first on axis 1, and again on axis 0. Successful expansion returns a list of newly expanded array of ECG signals.

```
// process data (ecg measurements)
const process = (measurement: number[]) => {
  return new Promise((resolve, reject) => {
    try {
      let measurement_expand = tf.expandDims(tf.expandDims(measurement, 1), 0);
      return resolve(measurement_expand);
    } catch (err) {
      return reject(err);
    }
  });
};
```

Figure 4.7: Processing in React Native using TFJS

Predict

As observed in Figure 4.8 This procedure calls TFJS's predict function using the TFJSGM, and the list of ECG signals from Section 4.1.4. Once the prediction is made, the output will be returned as type tensor.

```
// call and return prediction results as tensor
const predict = (model: tf.GraphModel, measurement_processed: tf.Tensor) => {
  return new Promise((resolve, reject) => {
    try {
      const result = model.predict(measurement_processed) as tf.Tensor;
      return resolve(result);
    } catch (err) {
      return reject(err);
    }
  });
};
```

Figure 4.8: Predicting in React Native using TFJS

Result

Prior to this procedure, the tensor from Section 4.1.4, will be converted into a 1 dimensional array using the code `result.as1D()` which is another method that is provided by the TFJS library. Using the new 1D list, this procedure sets a value to *prediction*, and *predictionProb*. With those values, it will be compared with the defined threshold, *thresh*. If any of the value were to be out of the threshold's range, reject the reading. Else, set reject as 0, which indicates that reading is reliable.

4.2 Continuous AF

4.2.1 Anticipated Input Type

To ensure compatibility with the continuous AF module, input type had to be anticipated. Figure 4.10 are screenshots taken from another thesis student from UNSW who

```
// process prediction array and return results as string
const result = (prediction_1D: any) => {
  // check prediction and reject
  let prediction = (tf.argMax(prediction_1D).arraySync() as number) + 1;
  let predictionProb = prediction_1D.arraySync() as number[];
  let thresh = [0.95, 0.91];

  let predictionString = 'None';
  let rejectString = 'None';

// compare with threshold
if (
  tf.max(predictionProb).arraySync() < thresh[0] &&
  (tf.max(predictionProb).arraySync() as number) -
    secondLargest(predictionProb) <
    thresh[1]
) {
  var reject = 1;
} else reject = 0;
```

Figure 4.9: Processing Result in React Native

worked on integrating a continuous ECG monitoring device [Tad22]. Retrieving the details from Figure 4.10, the calculated anticipated input average across 11 signals is 20 miliseconds per signal. It can also be seen in Figure 4.11, that the device outputs a

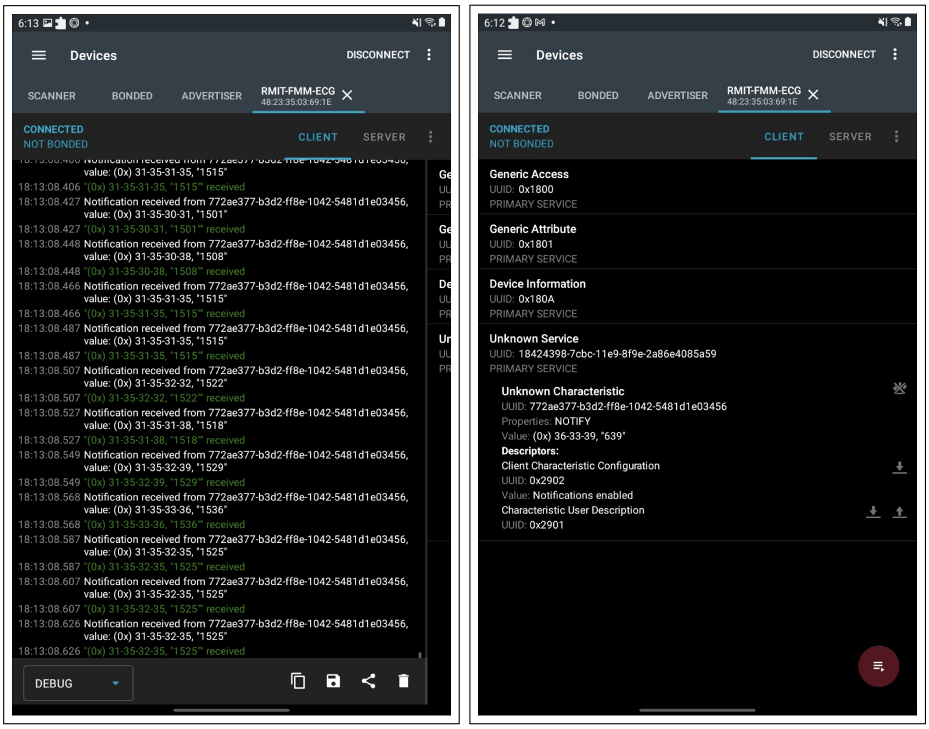

Figure 4.10: Input from Vlepis patch [Tad22]

string value that contains a number. However, the model accepts integers. Hence, the value should be converted into integers in the process.

Figure 4.11: Anticipated input type [Tad22]

4.2.2 Continuous ECG Simulation

Leveraging from the information from the previous section, an attempt is then made to create a continuous ECG simulation. In order to simulate a continuous ECG scenario, a custom function has been created. Algorithm 1 describes the procedure of how the aforementioned function works.

Algorithm 1: Continuous ECG Simulation

Input: $fSample$: Sample file, $rInterval$: ECG monitoring device reading speed(i.e., 30 milliseconds)

Output: List of ECG signal numbers that is read within pre-defined *timer*

1 Initialise *EmptyList* of type numbers
2 $SampleECGs \leftarrow fSample$ `// list of ECG signals of type numbers`
3 **while** $timer \geq 0$ **do**
4   **repeat** every $rInterval$
5   append *SampleECGs*'s value at index i into *EmptyList*
6         $i \leftarrow i + 1$
7   **until** $rInterval = 0$
`/* pre-defined timer can be of any reasonable value. i.e., 30 seconds */`
8 **if** $timer \leq 0$ **then**
9   run prediction with populated *EmptyList*

In Lines 1, and 2, a new empty list called *EmptyList* is created to store the data from *SampleECGs*. It is also observed that *SampleECGs* initialises its value to a list of ECG signals called $fSample$.

In Lines 3 up to 7, a loop occurs while *timer* (this project sets *timer* to 30 seconds) is not less than 0. Within that loop, for every $rInterval$ (this project sets $rInterval$ to 30 milliseconds), the value at $SampleECG$'s index position i is appended into $EmptyList$. Once that is done, i is increased by 1 so to move to the next index.

In Lines 8, and 9, when the *timer* reaches less than 0. Using the populated $EmptyList$ as parameter, run the prediction function.

4.3 Other Experimentation

This section covers other experimentation that was done. The reason for this section is to compare it against TFJS.

4.3.1 TFLite

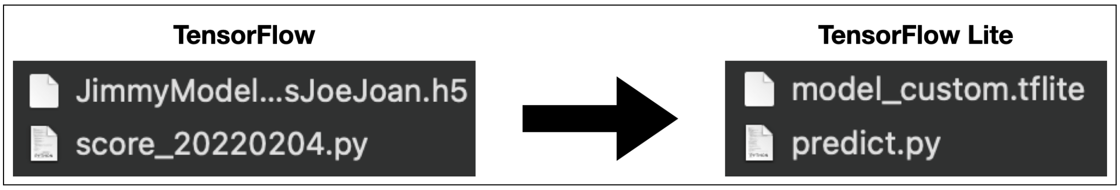

Figure 4.12: Keras to TFLite conversion

Just like TFJS, the Keras model can be converted into TFLite. The result of this conversion is that the model became 67% smaller in size. The difference is however, it only outputs as one single file, as opposed to a two parts namely model, and weights. This model and script is then deployed unto Azure Functions.

4.3.2 Azure Functions

Azure Functions is used to deploy the TFLite model, and the prediction script. HTTP trigger function is created, which works similarly to an API. By sending a HTTP request, in the form of a query string, a header, or `json` body; it triggers the prediction function, and returns the predicted outcome of the ECG readings.

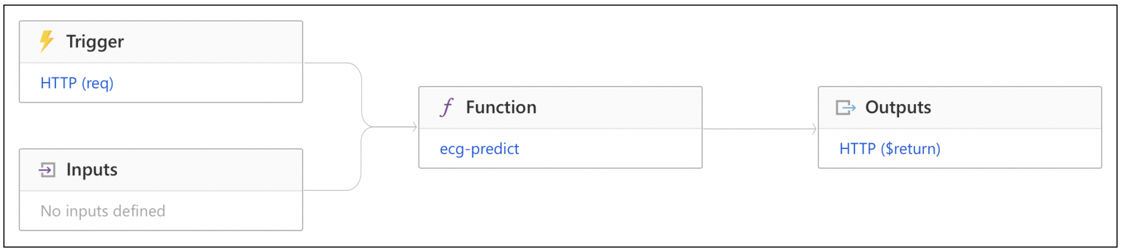

Figure 4.13: Azure Functions flow

Chapter 5

Evaluation

5.1 Prediction Results

5.1.1 Accuracy Comparison

Sample file `0008-2` is tested across 3 models. It is of type `.txt`. For TFJS, it is replaced with `p_0008-2` instead due to complications discussed in Chapter 4. Figure 5.1 shows the prediction outcomes all across the models.

The structure of the prediction output is that, it is a list of numbers which contains 4 values. Each value indicates the probability of the prediction, in which the higher it is, the more likely it is to be the predicted outcome. In order from left to right, the value is the probability of: AF, NSR, OA, and TN.

It can be seen that in Figure 5.1, the prediction probability of all models are very similar to each other. For example the NSR probability rate; the outcome from TFJS and TFLite differs with that of the Keras model only by a fourteen hundred-millionths. Thus, the prediction accuracy of all models are highly similar.

```
PredictionProb: [0.003147014882415533, 0.9517375230789185, 0.04081578925251961, 0.00429966114461422]
{"model": "model.json", "prediction": 2, "reject": 0}
```

```
PredictionProb:  [[0.00314701 0.9517375  0.04081573 0.00429966]]
{'prediction': 2, 'reject': 0, 'model': 'model_custom.tflite'}
```

```
PredictionProb:  [[0.00314702 0.95173764 0.04081572 0.00429966]]
{'prediction': 2, 'reject': 0, 'model': 'JimmyModelRelabeledTrainedOnCambellSamplesJoeJoan.h5'}
```

Figure 5.1: Prediction probability of sample 0008-2. In order from top to bottom: TFJS, TFLite, Keras model.

5.1.2 Classification Comparison

11 sample files are tested with 3 different models. Sample files are of types `.txt`, and `.json`. As for the models tested, it will be between TF(Keras), TFLite, and TFJSGM. Figure 5.1 shows a list of sample files and its expected results gathered by a group of researchers [AA+]; it is however, not a result predicative from any of the models.

Table 5.1: Expected results

Sample	Expected Results
p_0006-6	NSR
p_0007-1	NSR
p_0007-1	NSR
p_0007-1	NSR
p_0008-1	NSR
p_0008-2	NSR
p_0008-3	NSR
p_0009-1	OA
p_0009-2	AF
p_0009-3	AF
p_sample	AF

On the other hand, Figure 5.2 shows that there are some outcomes that are differs from what is expected. But those that are wrong are also the same all across all models. The results that do not match with the expected ones are namely `p_0008-3`, and `p_0001`.

Table 5.2: Prediction results and reliability

Sample	**Results (TF)**	**Results (TFLite)**	**Results (TFJS)**	**Reliability (TF)**	**Reliability (TFLite)**	**Reliability (TFJS)**
p_0006-6	NSR	NSR	NSR	Reliable	Reliable	Reliable
p_0007-1	NSR	NSR	NSR	Reliable	Reliable	Reliable
p_0007-1	NSR	NSR	NSR	Reliable	Reliable	Reliable
p_0007-1	NSR	NSR	NSR	Reliable	Reliable	Reliable
p_0008-1	NSR	NSR	NSR	Unreliable	Unreliable	Unreliable
p_0008-2	NSR	NSR	NSR	Reliable	Reliable	Reliable
p_0008-3	OA	OA	OA	Unreliable	Unreliable	Unreliable
p_0009-1	AF	AF	AF	Reliable	Reliable	Reliable
p_0009-2	AF	AF	AF	Reliable	Reliable	Reliable
p_0009-3	AF	AF	AF	Reliable	Reliable	Reliable
p_sample	AF	AF	AF	Reliable	Reliable	Reliable

5.2 Benchmarks

Table 5.3: Benchmark comparison

Model	**Platform**	**Average time per prediction (seconds)**
TF	Apple M1 Pro machine	25.4
TFLite	Apple M1 Pro machine	2.581
TFLite	Azure Functions	3.439
TFJS	iOS Simulator in Apple M1 Pro machine	17.86

As shown in Figure 5.3, the Keras model averages at 25.4 seconds per prediction. TFLite model averages at 2.581 seconds per prediction, which is 89.84% faster than the Keras model. TFLite model deployed on Azure averages at 3.439 seconds per prediction, which about 24.95% slower than TFLite model on a local machine, but still 86.46% faster than the Keras model. Finally, the TFJSGM averages at 17.86 seconds per prediction, which is about 29.69% faster than the Keras model, but 80% slower than TFLite deployed on Azure.

5.3 Why TFJS over TFLite

There are a couple of reasons to why TFJS is chosen over TFLite for this thesis project. Firstly, TFLite does not work locally in the mobile environment. For TFlite to work, the model and prediction code needs to be deployed on the cloud. In a previous experiment, this was done through Azure Functions. However, since the goal is to have the predictions to be done continuously, this may result a large amount of data traffic between the app and the cloud.

It is estimated that data being sent may range between ±75 to ±100 kilobytes. This only accounts for data sent from the app to the cloud, but not the other way around. So, if for example, 85 kilobytes of data are sent every minute, it may accumulate to 122.4 megabytes. Multiply that with 30 days, it can reach up to 3.67 gigabytes of data per month.

Hence, even though TFLite is faster, TFJS is picked as it allows more accessibility. This means that predictions can be made without an internet connection. Thus, there will be lesser risk of having a silent Atrial Fibrillation going undetected.

5.4 Testing

5.4.1 Compatibility Testing

The app initially is tested to be working on iOS and Android OS. However, after due to expo compatibility issues, the current app only works on iOS.

5.4.2 Usability Testing

The app was presented and reviewed at various stages by peers. Feedback given was put to use to improve and appropriate modifications are applied to the app.

5.5 AF Module Results

5.5.1 Home Screen

Figure 5.2: Home screen

AF Card

Upon entering the app, a home screen (see Figure 5.2) with various cards are displayed. One of them is the AF card, as seen in Figure 5.3. The AF card has a `Start` button that when is tapped will route the user to the detector screen (see Section 5.5.2).

Figure 5.3: AF home card

5.5.2 Detector Screen

Overview

In the detector screen (see Figure 5.4), user is greeted with an overview of the latest detection, as seen in Figure 5.5. The overview contains 4 values:

- Status of the detector, whether it's active or idle
- Latest detected result from the user's ECG measurement
- Reliability of the latest result
- Date and time of the latest result

If user has never used the detection feature prior, the values under each label will be displayed as `None`.

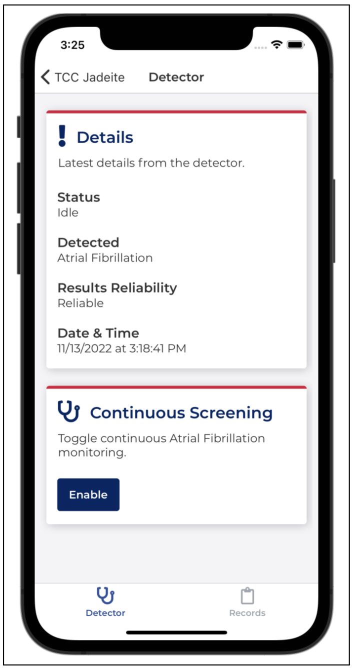

Figure 5.4: Detector screen

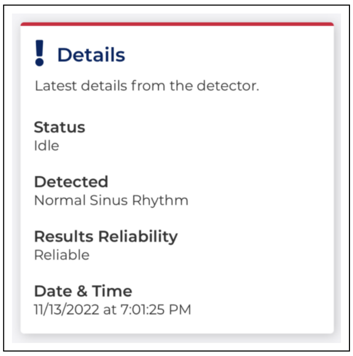

Figure 5.5: Detector overview

Detector Toggle

Under the overview, a card (see Figure 5.6) with a button can be seen. The button enables user to toggle the detector on and off.

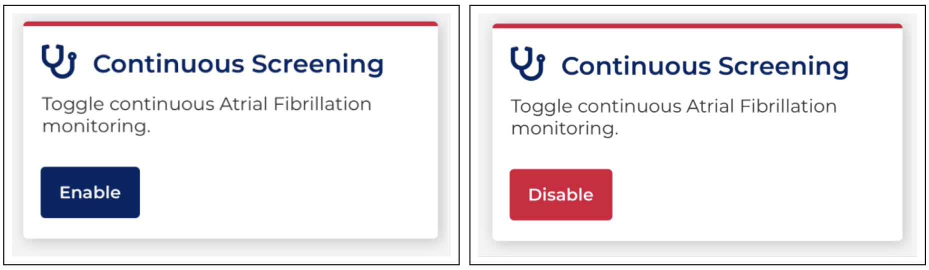

Figure 5.6: Continuous AF detector toggle with two states: inactive(left), and active(right)

5.5.3 Records Screen

In the records screen, as seen in Figure 5.7, shows a list of detected readings. There are 4 main readings: (1)NSR (2)AF (3)OA (4)TN. Reliability, date, and time are shown for each result.

Filters

The filter feature in Figure 5.8, allows users to filter by: (1)All (2)AF (3)OA. This attribute helps user to view crucial information. By filtering out unwanted data, the user will find it easier to keep track of their symptoms.

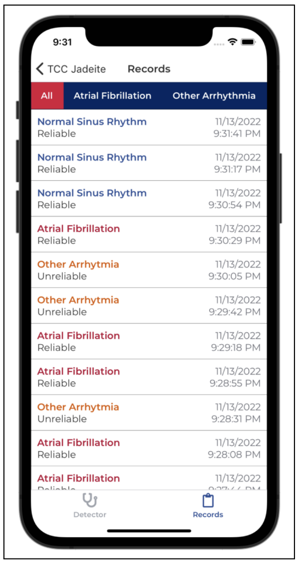

Figure 5.7: Detector records screen

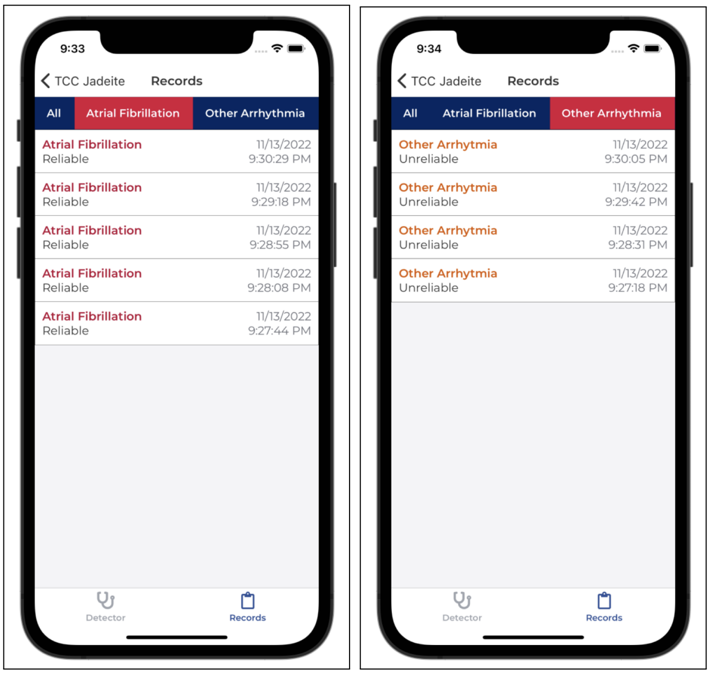

Figure 5.8: Detector records filter: AF filter(left), and OA filter(right)

Colour Coded Results

Readings are colour coded for easy viewing and identification. This allows both the user to quickly differentiate different symptoms at a glance.

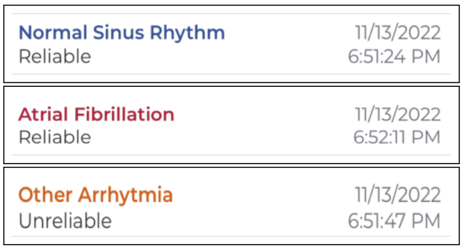

Figure 5.9: Colour coded results

5.5.4 Notifications

If an AF or OA is detected, the module will push a notification as seen on Figure 5.10. Notifications are pushed only when result is deemed reliable. This means that if an AF is detected but is unreliable, no notification will be pushed.

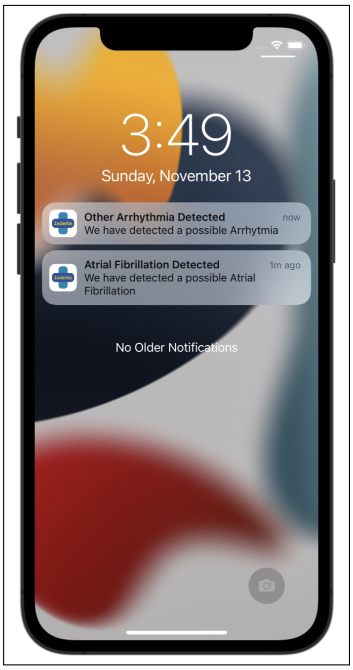

Figure 5.10: Notification feature

Chapter 6

Conclusion

This thesis report has explained the problem statement and aim for the development of a module for AF detection on a smartphone. Planned features are achieved. The DL model has successfully been converted from Keras to TFJS's Graph Model which allows it to work in the RN environment. The model is able to make predictions locally on a smartphone. Details surrounding the methodology, and implementation are covered. Last but not least, a thorough evaluation of the outcomes, and experiments; which justifies the direction taken in this thesis project.

6.1 Future Work

- Module for AF detection on a smartphone (PPG) → To allow a PPG DL algorithm to be implemented on top of the current AF detection module. This can further increase accessibility as users that cannot find or afford an ECG recording device, can now have another option to choose from.
- Integration with Vlepis patch → To allow real patients to be able to use the continuous AF detection module.
- Independent from Python dependencies → Being able to fully filter and process ECG readings in RN. As discussed before, the current way is to let the signals be partially filtered in Python before used in RN.

- Run in background → For continuous AF monitoring to be able to work even outside of the app.
- Store records in server → In this case, on KIOLA. i.e., all data gathered on a single day to be stored by the end of the day if connected to WiFi.
- Keeping TFJS up-to-date → Lookout for improvements or features that can further improve the functionalities and compatibilities.
- Further optimisation → Optimise the model configurations to increase prediction speed. i.e., by reducing model size.

Bibliography

[AA+] J. Magdy A. Argha, J. Li et al. Using a 1d transfer learning approach for abnormality detection from short-term single-lead ecg waveforms: Assessing the generalizability of an automated atrial fibrillation algorithm. Unpublished Manuscript.

[AAB+15] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[Ali17] AliveCor. *User Manual for Kardia™ by AliveCor®*, 2017.

[Ali21] AliveCor. Will i have to pay for a result of each ekg? https://alivecor.zendesk.com/hc/en-us/articles/115015830347-Will-I-have-to-pay-for-a-result-of-each-EKG-, 2021.

[App22] Apple. Take an ecg with the ecg app on apple watch. 2022.

[Ass20] Yannick Assogba. Tensorflow.js for react native is here!, 2020.

[BBC+17] Emelia J. Benjamin, Michael J. Blaha, Stephanie E. Chiuve, Mary Cushman, Sandeep R. Das, Rajat Deo, Sarah D. de Ferranti, James Floyd, Myriam Fornage, Cathleen Gillespie, Carmen R. Isasi, Monik C. Jiménez, Lori Chaffin Jordan, Suzanne E. Judd, Daniel Lackland, Judith H. Lichtman, Lynda Lisabeth, Simin Liu, Chris T. Longenecker, Rachel H. Mackey, Kunihiro Matsushita, Dariush Mozaffarian, Michael E. Mussolino, Khurram Nasir, Robert W. Neumar, Latha Palaniappan, Dilip K. Pandey, Ravi R. Thiagarajan, Mathew J. Reeves, Matthew Ritchey, Carlos J. Rodriguez, Gregory A. Roth, Wayne D. Rosamond, Comilla Sasson, Amytis Towfighi, Connie W.

Tsao, Melanie B. Turner, Salim S. Virani, Jenifer H. Voeks, Joshua Z. Willey, John T. Wilkins, Jason HY. Wu, Heather M. Alger, Sally S. Wong, and Paul Muntner. Heart disease and stroke statistics—2017 update: A report from the american heart association. *Circulation*, 135(10):e146–e603, 2017.

[But30] S. Butterworth. On the Theory of Filter Amplifiers. *Experimental Wireless & the Wireless Engineer*, 7:536–541, October 1930.

[CCP12] A. John Camm, Giorgio Corbucci, and Luigi Padeletti. Usefulness of continuous electrocardiographic monitoring for atrial fibrillation. *American Journal of Cardiology*, 110(2):270–276, Jul 2012.

[CE09] Douglas Curran-Everett. Explorations in statistics: confidence intervals. *Advances in Physiology Education*, 33(2):87–90, 2009. PMID: 19509392.

[com] The SciPy community. *SciPy documentation*.

[CSP+15] Marshall Copeland, Julian Soh, Anthony Puca, Mike Manning, and David Gollob. *Microsoft Azure: Planning, Deploying, and Managing Your Data Center in the Cloud*. Apress, USA, 1st edition, 2015.

[FCG08] F. H Fenton, E. M. Cherry, and L. Glass. Cardiac arrhythmia. *Scholarpedia*, 3(7):1665, 2008. revision #121399.

[GKD+16] David M. German, Muammar M. Kabir, Thomas A. Dewland, Charles A. Henrikson, and Larisa G. Tereshchenko. Atrial fibrillation predictors: Importance of the electrocardiogram. *Annals of Noninvasive Electrocardiology*, 21(1):20–29, 2016.

[HMvdW+20] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[Jov15] Emil Jovanov. Preliminary analysis of the use of smartwatches for longitudinal health monitoring. In *2015 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, pages 865–868, 2015.

[KKM+22] M. Kobel, P. Kalden, A. Michaelis, F. Markel, S. Mensch, M. Weidenbach, F. T. Riede, F. Löffelbein, A. Bollmann, A. S. Shamloo, I. Dähnert, R. A. Gebauer, and C. Paech. Accuracy of the apple watch iecg in children with and without congenital heart disease. *Pediatric Cardiology*, 43(1):191–196, Jan 2022.

[LBH15] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, May 2015.

[MBR+21] Markus R Mutke, Noe Brasier, Christina Raichle, Flavia Ravanelli, Marcus Doerr, and Jens Eckstein. Comparison and combination of single-lead ECG and photoplethysmography algorithms for wearable-based atrial fibrillation screening. *Telemed. J. E. Health.*, 27(3):296–302, March 2021.

[MC21] Savio Mathew and Ruth Chambers. Improving the utility and sustainability of novel health technology to improve clinical outcomes for patients: an east staffordshire experience of screening for atrial fibrillation with the alivecor kardiamobile. *BJGP Open*, 5(2), 2021.

[Mic22] Microsoft. Introduction to azure functions, 2022.

[Pla22] Meta Platforms. React native, 2022.

[PMH+19] Marco V. Perez, Kenneth W. Mahaffey, Haley Hedlin, John S. Rumsfeld, Ariadna Garcia, Todd Ferris, Vidhya Balasubramanian, Andrea M. Russo, Amol Rajmane, Lauren Cheung, Grace Hung, Justin Lee, Peter Kowey, Nisha Talati, Divya Nag, Santosh E. Gummidipundi, Alexis Beatty, Mellanie True Hills, Sumbul Desai, Christopher B. Granger, Manisha Desai, and Mintu P. Turakhia. Large-scale assessment of a smartwatch to identify atrial fibrillation. *New England Journal of Medicine*, 381(20):1909–1917, 2019. PMID: 31722151.

[PTC+03] Richard L. Page, Thomas W. Tilsch, Stuart J. Connolly, Daniel J. Schnell, Stephen R. Marcello, William E. Wilkinson, and Edward L.C. Pritchett. Asymptomatic or “silent” atrial fibrillation. *Circulation*, 107(8):1141–1145, 2003.

[SHT20] Amelia Swift, Roberta Heale, and Alison Twycross. What are sensitivity and specificity? *Evidence-Based Nursing*, 23(1):2–4, 2020.

[Shu20] Li Shuangfeng. Tensorflow lite: On-device machine learning framework. *Journal of Computer Research and Development*, 57(9):1839, 2020.

[Slo18] T. V. Slooten. Alivecor kardia mobile heart monitor 2018. 2018.

[SSS14] Mohammad Shenasa, Hossein Shenasa, and Mona Soleimanieh. Update on atrial fibrillation. *The Egyptian Heart Journal*, 66(3):193–216, 2014.

[STA+19] Daniel Smilkov, Nikhil Thorat, Yannick Assogba, Ann Yuan, Nick Kreeger, Ping Yu, Kangyi Zhang, Shanqing Cai, Eric Nielsen, David Soergel, Stan Bileschi, Michael Terry, Charles Nicholson, Sandeep N. Gupta, Sarah Sirajuddin, D. Sculley, Rajat Monga, Greg Corrado, Fernanda B. Viégas, and Martin Wattenberg. Tensorflow.js: Machine learning for the web and beyond, 2019.

[SY22] Chhabra L Sattar Y. *Electrocardiogram.* StatPearls Publishing, 2022.

[Tad22] Sai Tadala. Integrating the vlepis patch system for physiological measurement with a frontend application. Bachelor's thesis, School of Computer Science and Engineering, The University of New South Wales, 2022.

[Tena] TensorFlow. *TensorFlow API.*

[Tenb] TensorFlow. *TensorFlow.js API React Native.*

[Tenc] TensorFlow. *TensorFlow.js Convertor Manual.*

[Ten22a] TensorFlow. Tensorflow.js, 2022.

[Ten22b] TensorFlow. Tensorflow.js adapter for react native, 2022.

[Ten22c] TensorFlow. Tensorflow.js convertor, 2022.

[VGO+20] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[VLK+22] Eemu-Samuli Väliaho, Jukka A. Lipponen, Pekka Kuoppa, Tero J. Martikainen, Helena Jäntti, Tuomas T. Rissanen, Maaret Castrén, Jari Halonen, Mika P. Tarvainen, Tiina M. Laitinen, Tomi P. Laitinen, Onni E. Santala, Olli Rantula, Noora S. Naukkarinen, and Juha E. K. Hartikainen. Continuous 24-h photoplethysmogram monitoring enables detection of atrial fibrillation. *Frontiers in Physiology*, 12, 2022.

[WKE+20] Felix K. Wegner, Simon Kochhäuser, Christian Ellermann, Philipp S. Lange, Gerrit Frommeyer, Patrick Leitz, Lars Eckardt, and Dirk G. Dechering. Prospective blinded evaluation of the smartphone-based alivecor kardia ecg monitor for atrial fibrillation detection: The peak-af study. *European Journal of Internal Medicine*, 73:72–75, Mar 2020.

[WKW16] Karl Weiss, Taghi M. Khoshgoftaar, and DingDing Wang. A survey of transfer learning. *Journal of Big Data*, 3(1):9, May 2016.

Appendix

A.1 Prototype

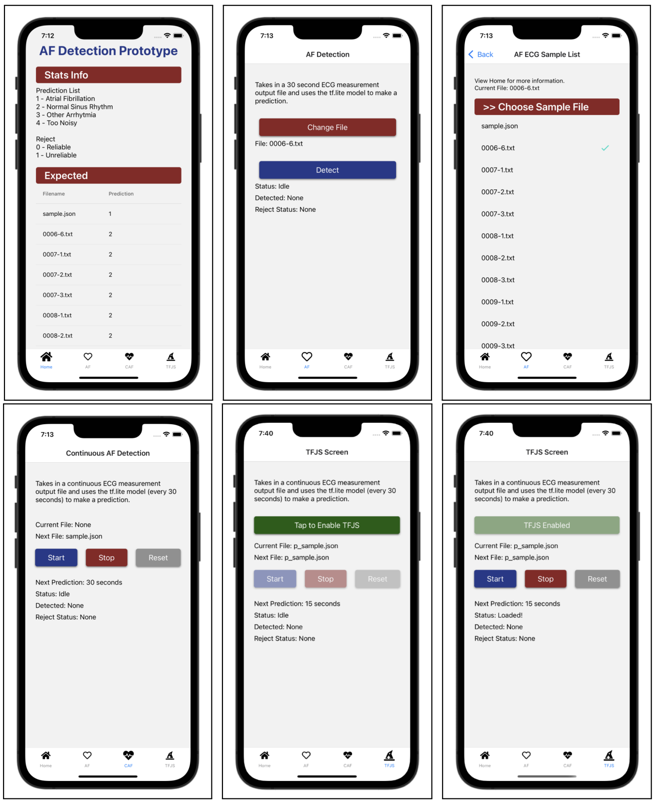

Figure A.1: Prototype. In order from top-left to bottom-right: home screen, AF detection screen, AF detection sample list, continuous AF screen, continuous AF using TFJS screen(inactive), continuous AF using TFJS screen(active)

A.2 Kanban Board: Trello

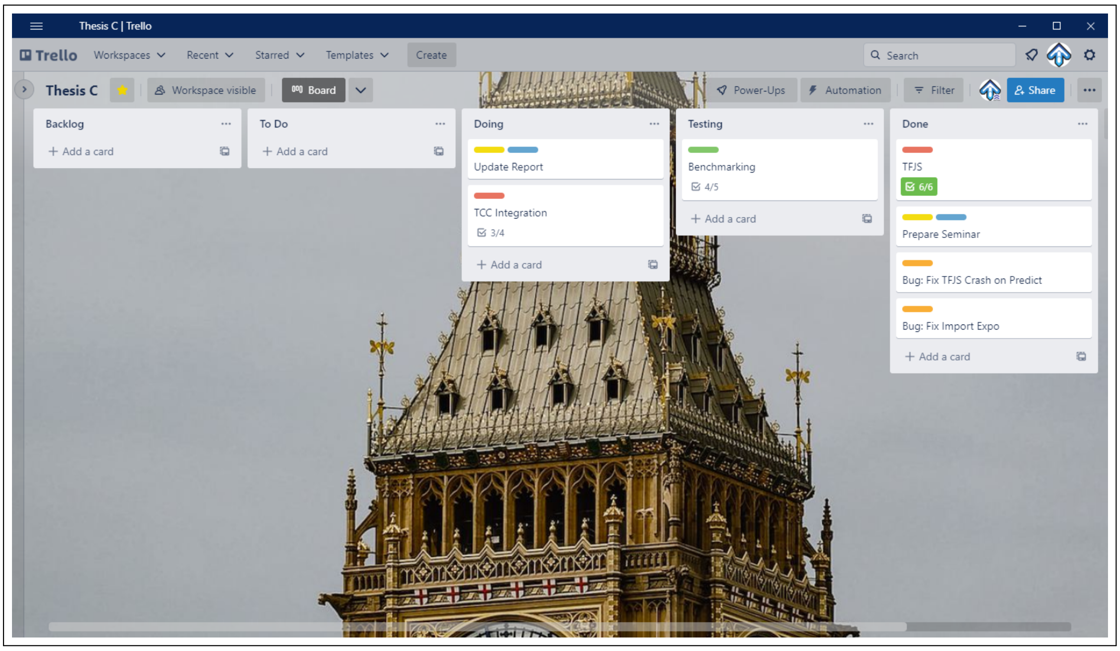

Figure A.2: Using Trello for Kanban