Department of Electrical and Computer Engineering

Part IV Research Project

Project Compendium

Project Number: 97

# Automated smartphone emergency callouts for heart disease sufferers

Author: Marcus Wong

Supervisor: Dr Avinash Malik

Secondary Supervisor: Dr Kevin Wang

11 October 2018

# Declaration of Originality

This report is my own unaided work and was not copied from nor written in collaboration with any other person.

Marcus Wong

**ABSTRACT:** The research intent of this project is to explore the feasibility of an automated heart alert system, alerting for medical attention when anomalies in heart activity are detected. The research report has highlighted the actions undertaken to successfully develop such a system; the following compendium provides supporting material to the research report. It aims to provide further detail on the implementation of the proposed heart alert system, how to run/test the developed components, and to inform on any findings gathered during the research project that ultimately was not included in the final solution. Sections on data wrangling, machine learning, smartphone app development and embedded device development are discussed, with supporting material found in the references and appendices.

## 1. Introduction / Dependency Installation

The following compendium highlights the methods, testing instructions and research conducted for Project 97 of the University of Auckland ECE Part IV Projects. The intent of this research is to develop an automated emergency callout system for heart disease patients in life-threatening events.

The structure of the report includes discussing how to run any relevant components, procedures for testing that occurred during research, and any other research (if any) which was ultimately not included in the final solution.

The overall file structure includes:

- /signalprocessing, containing the files related to the initial data wrangling process, to process datasets found from Physionet. This involves the conversion of .ecg and .qrs files into more friendly .txt R-R interval files, as well as the use of the Python 'hrv' library to convert R-R interval samples into frequency domain heart rate variability (HRV) metrics. Scripts for parsing data into .csv format are also included. There are two datasets ultimately used in the final implementation; other files are included but discussed in Section 2.
- /machinelearning, containing the files to run the artificial neural network developed during this research project. K-fold cross validation was achieved via scikit-learn, and different analysis/processing tools like numpy/pandas have been used for data output. NOTE: This folder requires a virtualenv environment to be initialized. Detailed steps for this are included in Section 3.
- /PolarHeartRateApplication, containing code related to the Android app developed for automated emergency callouts. This code is run using Android Studio, with official support for version 4.3 (Jelly Bean) to version 7.0 (Nougat). However, the app has been previously tested on Android version 7.1 with no apparent issues. Instructions for testing are included in Section 4.
- /Bluetooth-le-spoofer, this contains the mbedOS files used to program the nRF52-DK; however, compiling from the source code provided is not recommended for simplicity's sake. Detailed instructions on how to run the code easier provided can be seen in Section 5.

Prerequisites for running the software packages required in this repository are in the requirements.txt file found in the root directory. These can be easily installed (using Python) with virtualenv. This can be installed by using pip:

$ pip install virtualenv

Then the following command can be invoked:

$ virtualenv –no-site-packages –distribute .env && source .env/bin/activate && pip install -r 'requirements.txt'

**Additional packages not included in requirements.txt are required. This includes command-line installation of wfdb for some data wrangling scripts, which can be installed here for all platforms here: https://www.physionet.org/physiotools/wfdb.shtml#applications. Three Python libraries: Keras, TensorFlow and json also require installation for machine learning scripts. These can be installed using pip:**

$ pip install keras
$ pip install tensorflow
$ pip install json

Software developed for this project was managed using the Git version control system, hosted on GitHub (https://github.com/marcusjzw/compsys700-new). This is a fork of an older repository hosted on GitHub by user *kalegevans* ((https://github.com/kalegevans/compsys700), my previous project partner who withdrew from the course

due to sickness. However, as seen in the history of both repositories, almost all the work has been developed by myself. The files developed are also provided locally with the Compendium submission.

To easily download any Physionet datasets, use the rsync command-line tool with this command:
$ rsync -Cavz physionet.org::nameOfDb /directory
where nameOfDb substitutes the desired database and /directory is the absolute path of where you want to save it.

## 2. Data Wrangling

All data wrangling related files explained in this section can be found by going to the root directory and entering the '/signalprocessing' subdirectory.

### 2.1. How to Run

#### *2.1.1. Onset_prediction_vtvf*

This folder contains dataset files downloaded from the Spontaneous Ventricular Tachyarrhythmia Database. The raw data can be found in the 'data/` subdirectory. In this subdirectory, it should be noticeable that files are tagged as 'mr', 'vt' or 'vf'.

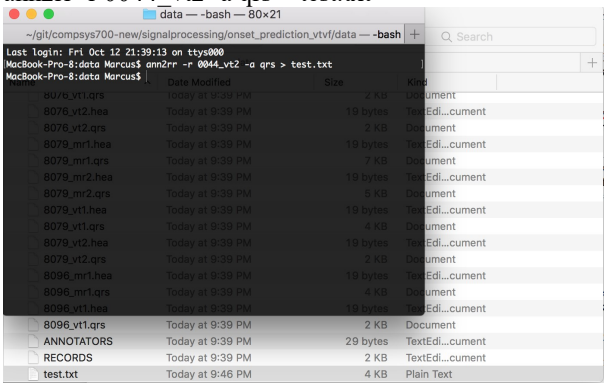
- 'vt' stands for ventricular tachycardia. These files were converted to R-R interval .txt files and added to the VT data used in the machine learning dataset.
- 'vf' means ventricular fibrillation; these files were omitted for reasons explained in the report.
- 'mr' means most recent. It contains the 1024 R-R intervals before 'ICD interrogation'. To understand this, it is known that all files in this database are captured via an ICD (Implantable Cardioverter Defribillator), used by heart disease sufferers. When a pacing event is required due to VT or VF, the latest 1024 R-R intervals in the device's memory are saved. 'MR' thus refers to a non-VT event, much like normal sinus rhythm.

'MR' events were mixed with normal sinus rhythm database data to create the full non-VT dataset. This was justified as MR data should provide the most accurate information to the neural network, where the patient recorded suffers from arrhythmia but is not currently undergoing an episode. This accurately reflects the target end user of the automated heart alert system.

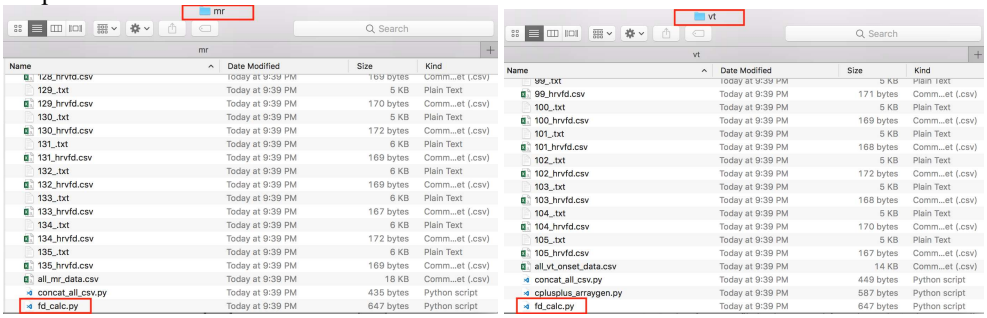
The 'mr/' folder shows the data extracted from mr-tagged samples, and the `RRdata1/vt/` folder shows the data extracted from vt-tagged samples. In both directories, the same flow was conducted to generate usable data:

1. The files downloaded from Physionet are seen in the `data/` subdirectory and are converted via wfdb. However, no script was used here. Manual wfdb commands were typed on console, as there was more hassle trying to write a script to sort through erratically numbered files, and also omitting vf-tagged files from being converted. To convert the data samples, use the following command on Bash/CMD:
   $ ann2rr -r <name-of-file> -a qrs > outputname.txt

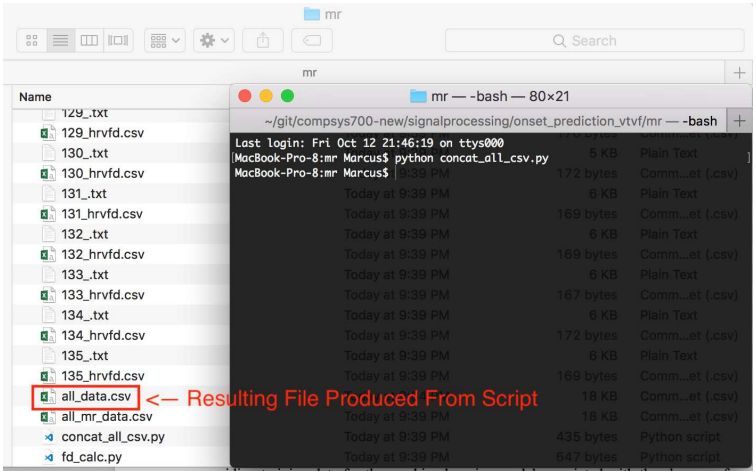
   For example:
   ann2rr -r 0044_vt2 -a qrs > test.txt



2. This should produce a file named test.txt which can be opened to show a new-line separated R-R interval in milliseconds. In my case, both mr and vt folders used the naming convention xxx_.txt, where xxx denotes a (potentially) three-digit number.
3. After separating samples by 'mr' or 'vt' into their respective folders, 'python fd_calc.py' is called. This is a script that exists on both 'mr' and 'RRdata1/vt' folders.




4. This calls the python `hrv` library to compute and output frequency domain HRV metrics in .csv format. In this script, parameters discussed in the report are chosen such as Welch's method for power spectral density estimation, Cubic interpolation, and linear detrending. This is done by navigating to the 'mr' folder or the 'RRdata1/vt' folder and calling 'python fd_calc.py'.
5. Lastly, after a .csv file has been created for each sample, 'python concat_all_csv.py' should be called to create `all_data.csv`. This was renamed to 'all_mr_data.csv' and 'all_vt_data.csv' in their respective folders for clarity purposes.




### *2.1.2. Nsr2db*

The folder named nsr2db contains data from the Normal Sinus Rhythm R-R Interval Database found here: https://www.physionet.org/physiobank/database/nsr2db/. These can be downloaded using the rsync package using the steps mentioned in section 1; alternatively, the data already exists in this folder. The objective of collecting this data was to obtain R-R intervals related to normal sinus rhythm, thereby providing training data for the machine learning model associated with the absence of ventricular tachycardia (from now on referred to as 'non-VT data'.

**NOTE:** The scripts in this folder require the installation of the wfdb command line.

The python script, *ecg_to_rri_script.py* can be run simply by navigating to the nsr2db folder is running: *python ecg_to_rri_script.py*. This script simply loops through all datasets and runs the wfdb command to convert .ecg files to .rri txt, as recommended in appropriate Physionet documentation (https://www.physionet.org/physiotools/wag/ann2rr-1.htm). However, the result contains a strange artefact – the first line of every .txt file has an unreasonable value as an R-R interval (usually a 5-digit number). To fix this, running 'python hotfix_firstline_irregular.py' runs through all the produced .txt files and deletes the first line.

The .txt data produced here was similarly run through frequency domain HRV conversion, as shown in section 2.1.1. Normal sinus rhythm data was mixed with 'mr' data from the previous section to form all non-VT data used in the machine learning dataset. This ultimately forms the 'all_data_onsets.csv' file seen on the root `signalprocessing` folder.

This .csv file contains the following columns:

[record number] [lf] [lfnu] [lf_hf] [total_power] [hfnu] [vlf] [hf] [outcome]

From these outputs, LF, HF, LF/HF and VLF will go on to be used in the neural network. The outcome column contains the class variable; either 0 (non-VT) or 1 (VT). The decision to omit the other inputs was to emulate literature as close as possible, as mentioned in the report.

### 2.2. Other Research

The last unmentioned folder, '/matlab-ecg-testing', was the result of unfruitful research. This involved the use of Physionet datasets which were ultimately not included in the final solution. This is due to the fact that these datasets were in .xws format, which is an ECG waveform. Using a MATLAB library called 'mhrv', these ECG waveforms were (attempted) to be converted into HRV. However, results were flawed due to the following issues:

1. MIT-BIH datasets [CITE], a database used during this implementation, have long periods of normal sinus rhythm of about 6-8 minutes. This is despite being tagged as an arrhythmia episode. This was not considered prior to data processing, so a lot of work had to be redone.
2. This requires extensive work with Lightwave, an online data splicing tool which would be required to splice out irrelevant data.
3. The Matlab library involved processing from ECG to R-R intervals, then from R-R intervals to frequency domain HRV. This involves a two-step process which leads more room for error. For simplicity's sake, due to finding the other datasets mentioned in Section 2.1, datasets already in R-R interval format were vastly preferred to avoid one unnecessary conversion step.
4. Among many techniques, a popular and fast method is known as the Fast Fourier Transform (FFT). The squared-magnitude components of an FFT produces a periodogram, which HRV parameters can be extracted from. This has been explored before in research and served as the initial approach for spectral density estimation in this project. However, relatively poor results (which are justified in the Report) led to the exploration of other methods.

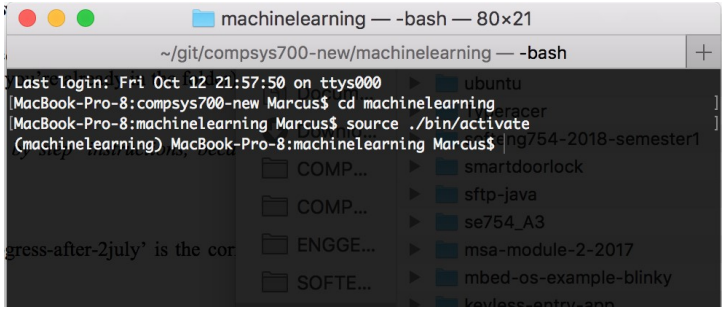
However, this Matlab library still proved useful later to generate power spectral density figures of ventricular tachycardia episodes shown in the report.

## 3. Machine Learning

The following section discusses the machine learning code developed in this section. The dependencies outlined in Section 1 are assumed to be installed. All content related to machine learning is found in the `/machinelearning/hrv_nn' folder.

*Note: Before running anything in this section, initiate virtualenv!*

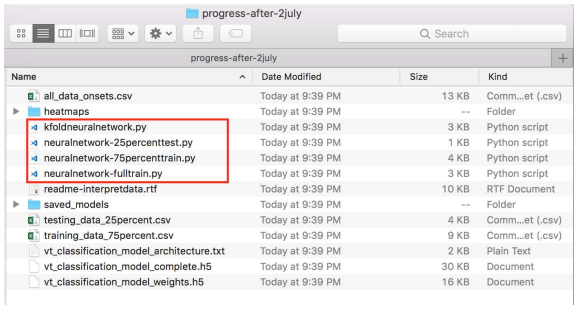
To start virtualenv, use the following command:
$ cd machinelearning (skip this step if you're already in the folder)
$ source ./bin/activate



*Note 2: This section is light on 'step by step' instructions because most developed work can be run by a simply command line call to run a script.*

### 3.1. How to Run / Test Procedures

In hrv_nn, there are two folders. 'progress-after-2july' is the correct folder. Discussions on the other folder can be found in Section 3.2.

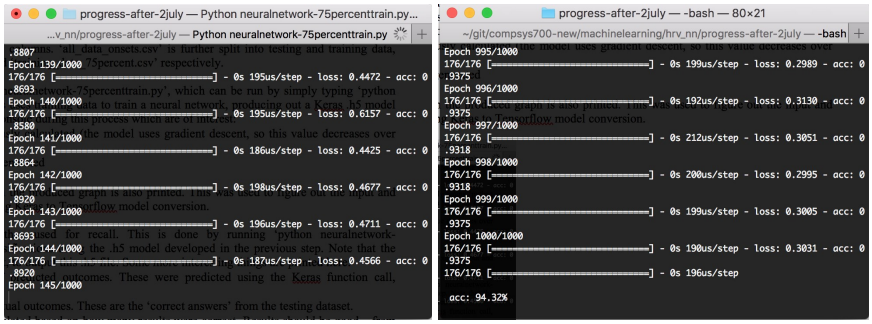



*The four Python scripts discussed in this section. Simply call 'python xxxx.py', where xxxx = name of the script*

To link to the previous section, the 'all_data_onsets.csv' file contains the data processed from the previous section. This is shown in Appendix A. Note there are no labels associated with this file; instead, 'readme-interpretdata.rtf' highlights the content of the different columns. 'all_data_onsets.csv' is further split into testing and training data, called 'testing_data_25percent.csv' and 'training_data_75percent.csv' respectively.

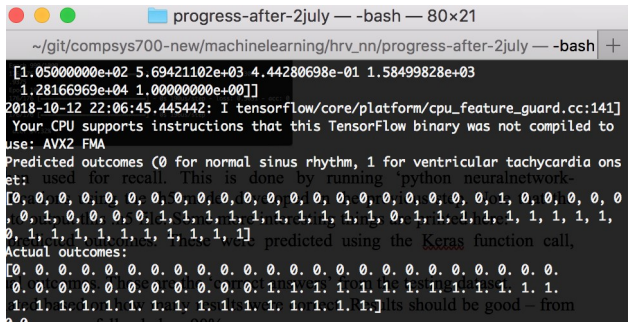
The first Python script of interest is 'neuralnetwork-75percenttrain.py', which can be run by simply typing 'python neuralnetwork-75percenttrain.py'. This uses training data to train a neural network, producing out a Keras .h5 model file. There are plenty of print outs to console during this process which are of interest:

- the loss function is continuously calculated (the model uses gradient descent, so this value decreases over time)
- accuracy with each epoch is generated
- the ending accuracy is output
- The names of nodes in the graph are also printed, used later during Keras to Tensorflow model conversion.

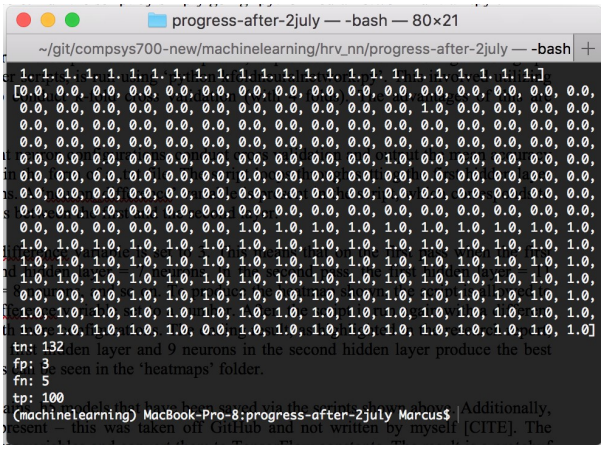



The produced Keras .h5 file are then used for recall. This is done by running 'python neuralnetwork-25percenttest.py', which predicts using the .h5 model developed in the previous step. Note that the previous Python script must be run first, to output this .h5 file. Some more interesting things are printed here:

- The script lists the array of predicted outcomes.
- The script lists the array of actual outcomes. These are the 'correct answers' from the testing dataset.



The next file of interest, ‘neuralnetwork-fulltrain.py’, can be run to calculate confusion matrix results using scikit-learn. These are printed out to the console. Run the script by simply going ‘python neuralnetwork-fulltrain.py’.




*Where tn = true negative, fp = false positive, fn = false negative, tp = true positive. Note that these results differ from the reported confusion matrix – the overall result is always similar but not exactly the same, due to neural networks being heuristic in nature.*

‘kfoldneuralnetwork.py’ is the script run to output neuron heatmap data, to produce the neuron configuration graph shown in the report. This, like all other scripts, is run using ‘python kfoldneuralnetwork.py’. This involved utilizing scikit-learn’s StratifiedKFold API, to conduct k-fold cross validation (with 4 folds). The advantages of this are discussed in the report.

The idea of this script is to try different neuron configurations, conduct cross validation and output the mean accuracy and standard deviation from all folds in the form of a .txt file. The script loops through setting the first hidden layer from between 10 neurons to 24 neurons. A ‘neuron_difference’ variable is present in the script, which corresponds to the difference in the number of neurons between the first and the second layer.

For example, by default, the neuron_difference variable is set to 3. This means that on the first pass when the first hidden layer = 10 neurons, the second hidden layer = 7 neurons. In the second pass, the first hidden layer = 11 neurons and the second hidden layer = 8 neurons, and so on. To produce the heatmap shown, the script is allowed to run to completion with the neuron_difference variable set to a number. After, the script is run again with a different “neuron difference”, to experiment with more configurations. The ending result, as highlighted in the research report, should see that the 12 neurons in the first hidden layer and 9 neurons in the second hidden layer produce the best results. The result of these experiments can be seen in the ‘heatmaps’ folder.

Lastly, the ‘saved_models’ folder contains .h5 models that have been saved via the scripts shown above. Additionally, a ‘keras_to_tensorflow.py’ script is present – this was taken off GitHub and not written by myself (https://github.com/amir-abdi/keras_to_tensorflow). The objective of this script is to freeze Keras variables and convert them to TensorFlow constants. The result is a protobuf file (.pb) that is compatible with the Android interface for machine learning models (TensorFlowInferenceInterface). This script can be called via this command line call:

python keras_to_tensorflow.py -graph_def True -input_model_file <nameOfFile>
where nameOfFile is ‘vt_classification_model_complete75PERCENT.h5’

The output .pb file finally concludes the experimentations with machine learning on PC, where the .pb model can be ported over to Android.

### 3.2. Other Research

The folder ‘progress-prior-2july’ located in ‘hrv_nn’ is archived and should not be used to run anything officially. Files of interest include ‘all_data.csv’, which contains wrangled data from the datasets not used in the final solution (i.e. datasets collected by the methods outlined in Section 2.2). This research was abandoned for having very poor accuracy

results, indicating a poor data formatting process. Despite showing 80-90% accuracy in neuron heatmap readings, accuracy was not reflective of 'true' recall rates.

This was uncovered with the following steps:

1. Splitting the 'all_data.csv' file into testing_data.csv and training_data.csv.
2. The file, 'neuralnetworkv3.py' was written and run using training_data.csv as training data. A .h5 Keras model file was output as a result.
3. Consequently, a python script, 'neuralnetworkv3_test.py' was developed and run to use testing_data.csv to fetch predictions based on the saved .h5 Keras model.
4. Results were poor.

On further research on why prediction and accuracy rates differed greatly, the issue of uneven data sizes was uncovered. The previous 80-90% accuracy ratings were essentially on par with 'random guesses', since much more non-VT data was supplied than VT data. After this result, further work was done to format data correctly (as seen in section 2.1), with much better recall rates using the datasets chosen in section 2.1.

## 4. Smartphone App Development/Polar H7 Integration

### 4.1. How to Wear and Configure the Polar H7

The electrode areas on the reverse side of the strap detect heart monitoring parameters. The transmitter device sends these detected parameters over a Bluetooth Low Energy connection.

To wear the belt:

1. Wet the electrode areas of the strap using water. Make sure they are well moistened for optimal accuracy.
2. 'Clip on' the transmitter to the strap.
3. Tie the strap around your chest, just below the chest muscles. Attach the hook to the other end of the strap.
4. Check that the Polar logo on the connector device is in an upright position.
5. Position the transmitter in the leanest area possible and rotate the chest belt slightly to the left side of your torso (i.e. close to the heart).

To connect the belt to the Android phone, preliminary pairing must be done for the app to discover the device. To do this, simply wear the belt (once worn, the transmitter automatically advertises itself) and follow these steps on your Android device:

1. Go to 'Settings'.
2. Go to 'Bluetooth'.
3. Find the Bluetooth device called 'Polar H7 xxxxxx', where xxxxxx is a hexadecimal string which is not relevant.
4. Tap on the device to pair the device to the phone. Occasionally, there have been situations where an error message stating "passkey incorrect" is displayed. Retrying usually fixes this problem - no passcode is required.
5. Done!

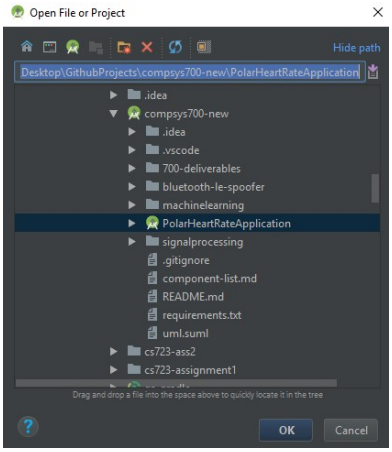
### 4.2. Running the Android App

Before starting, the Android phone used to test the app must be configured as a deployment device on Android Studio. If this is the case, skip the following steps. If not, these instructions should be followed:

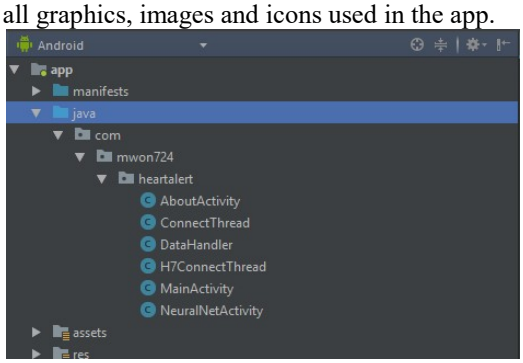
1. Connect your device to your computer.
2. Open the Settings app on your phone.
3. Scroll to the bottom and select About / About Phone.
4. Scroll to the bottom and tap 'Build number' 7 times.
5. Return to the previous screen to find 'Developer Options' near the bottom.
6. Open Developer Options, and then scroll down to find and enable USB debugging.

From the root directory, the folder named '/PolarHeartRateApplication' contains all application files. To run the code, Android Studio is required and can be downloaded free of charge at https://developer.android.com/studio/.

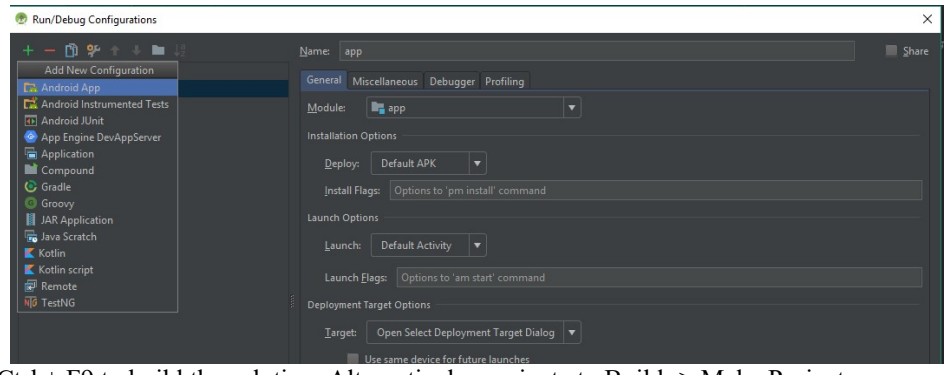
1. Open Android Studio, and Open a New Project. Navigate to the root directory of the project files. The folder named PolarHeartRateApplication should display the Android Studio logo to the left as shown. Highlight this folder and click OK.



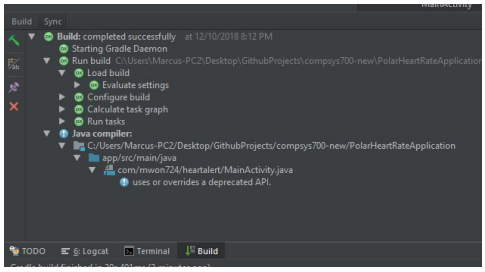

2. Android Studio will take a while to load for the first time. This includes checking gradle scripts for dependency management. Please allow up to 5 minutes for this.
3. The file structure is simple. The folder named 'heartalert' encloses the six classes used in the app, which are discussed in detail in the Report. The assets folder contains files related to the exported machine learning model. The res folder contains all graphics, images and icons used in the app.



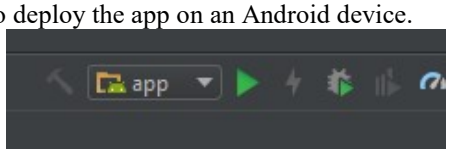

4. To run the app, ensure that 'app' is selected on the dropdown menu on the top right. **If it is selected, skip this step.** If it is not selected, click the dropdown menu and click 'Edit Configurations'. Press the '+' icon on the top left and select 'Android App'. Ensure that the Module selected in the General tab says 'app'. A screenshot of all relevant details has been provided:



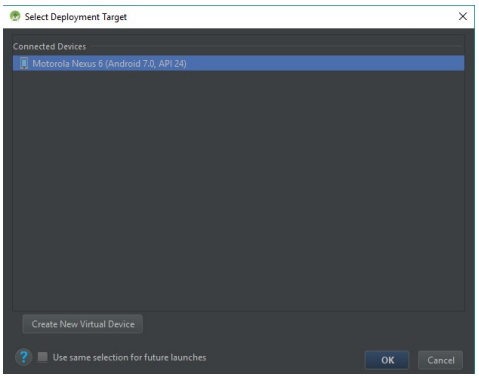

5. Press Ctrl + F9 to build the solution. Alternatively, navigate to Build -> Make Project.
6. A successful build is indicated with successful messages on the Build tab, as shown:



7. Click the Green Play button to deploy the app on an Android device.

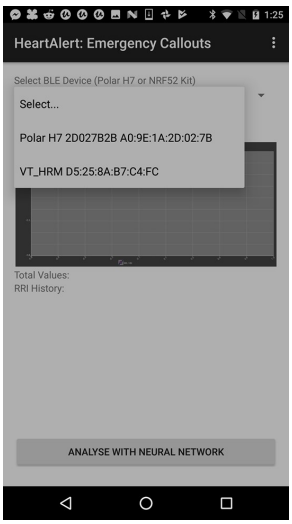



8. In the following screen, the connected Android device should display as a prospective deployment target. Click OK.



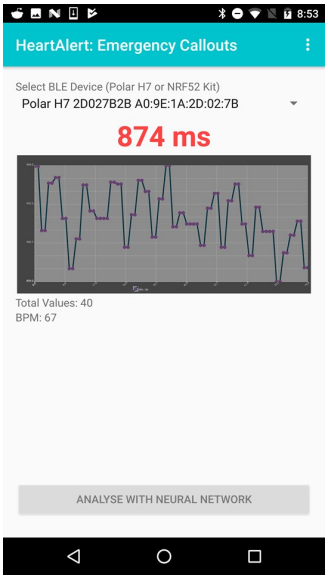

From this point, the .apk should install on the phone and launch automatically. Subsequent screenshots will now focus on the smartphone app. It is assumed Android Studio is kept running and the phone is kept plugged in, this way Logcat messages can be read. Logcat messages can be activated by clicking the tab at the bottom of the IDE.

1. Wear the Polar H7 Belt using instructions in Section 4.1.
2. On the initial screen, click the dropdown menu labeled 'Select…'. *This should display paired devices only*. If you have not paired the Polar H7, follow the instructions in Section 4.1 to pair. If you have not paired the embedded device, you can do so using the information in Section 5.1.
3. Select the device labeled Polar H7.

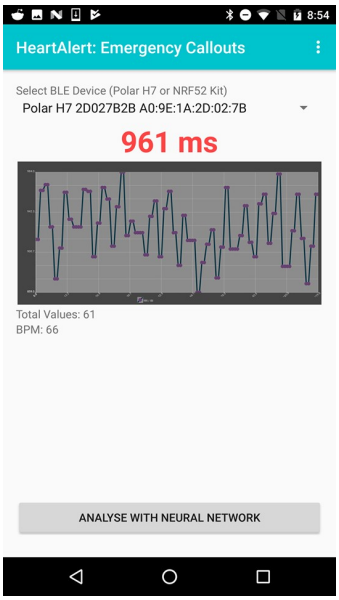



4. R-R intervals will immediately start being retrieved from the Polar H7 transmitter. The Logcat output should look like Figure 12 of the Report. R-R intervals are required to be collected for 60 samples – the 'Analyse with Neural Network' button is grayed out until then. Occasionally, two R-R intervals will be added – this is normal; the transmitter is sending two packets of R-R interval data.
5. Once 60 samples have been collected, tap 'Analyze with Neural Network'. This involves the processing of R-R data into frequency domain HRV (using the hrv-lib library), and then running through the classifier. A pop up message should appear determining the classification. After clicking OK, the R-R intervals processed and HRV metrics calculated are displayed on the screen. '

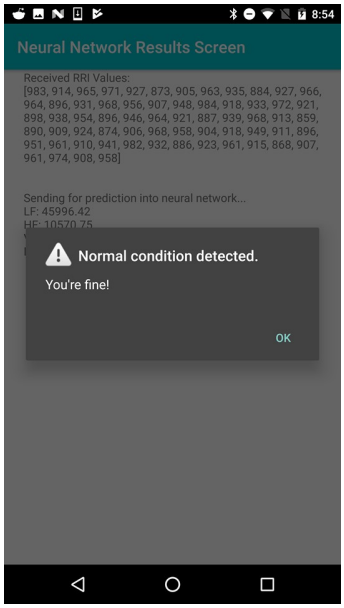
The following screenshots highlight the complete app flow:




*(1) Recording R-R intervals, only 40 values collected*




*(2) > 60 values collected, analyse button enabled.*




*(3) Classification popup – normal sinus rhythm detected*

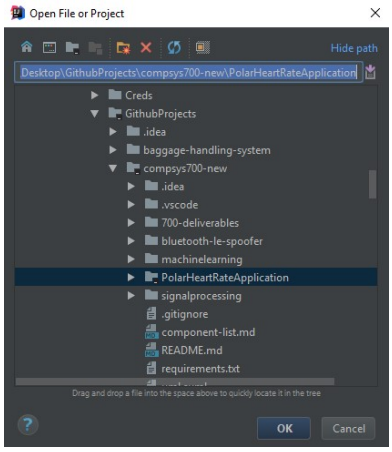
To output the metrics shown in Table 2 of the Report (the 2nd confusion matrix), this process was repeatedly taken over four days to obtain the relevant data (rate of true positives vs false positives, since it is known that every patient tested does not suffer from ventricular tachyarrhythmia). The other values present in the confusion matrix are retrieved via the embedded device, discussed in Section 5.3.

### 4.3. Class Diagram Generation

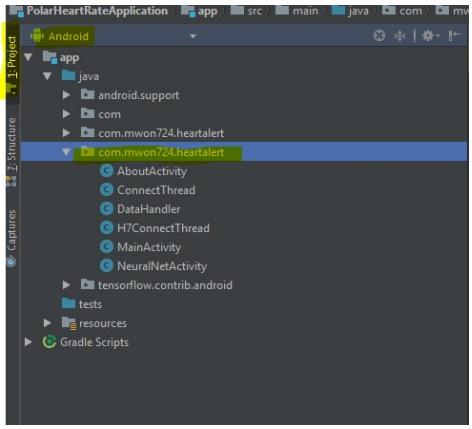
The class diagram found in the Report (Figure 13) can be generated using IntelliJ Ultimate. This is a paid software tool but is free for students (or anyone with an academic e-mail address). More details about this offer to acquire IntelliJ Ultimate can be found here at https://www.jetbrains.com/student/.

Assuming IntelliJ Ultimate has been installed, the class diagram for the Android app can be generated with the following steps:

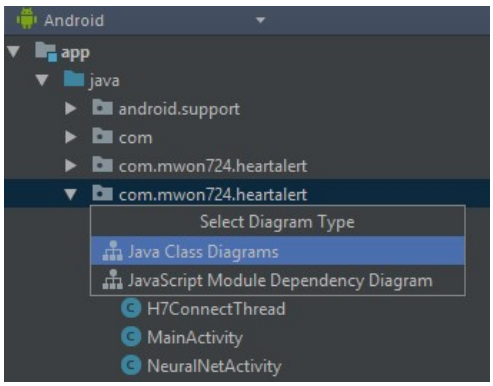
1. Open IntelliJ Ultimate, and open the PolarHeartRateApplication.



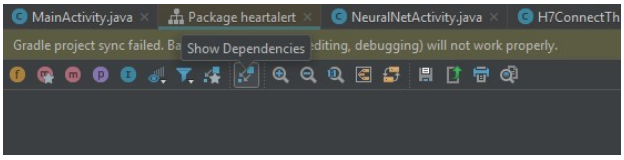

2. On the left panel of the IDE, ensure that the ‘Project’ tab is selected, ‘Android’ is selected as the dropdown menu option, and the folder ‘com.mwon724.heartalert’ is highlighted.



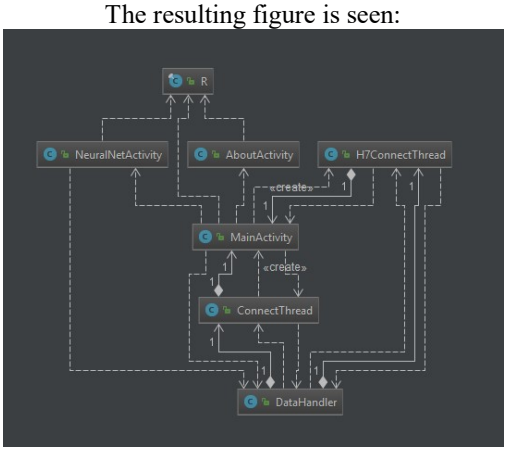

3. Press Ctrl + Alt + Shift + U. Alternatively, right-click the folder, hover over ‘Diagrams’, and select ‘Show Diagram’. The resulting pop up should look like this:



4. Hit Enter, and wait for the diagram to load. Then, click the ‘Show Dependencies’ Icon on the top toolbar, as shown here:




The resulting figure is seen:



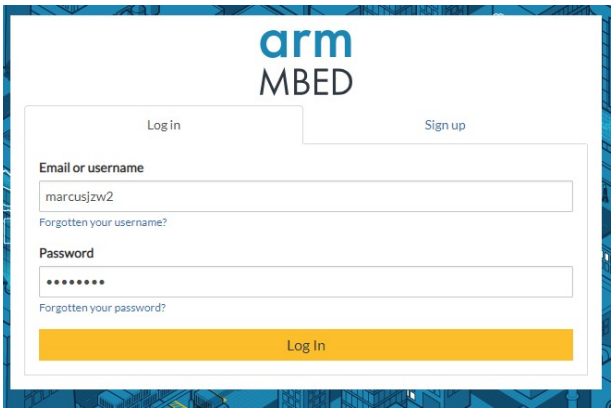

Some slight cleanups were made to the final report class diagram. This included removing redundant classes like built-in Android classes and the ‘R’ class for asset retrieval.

## 5. Embedded Device Development

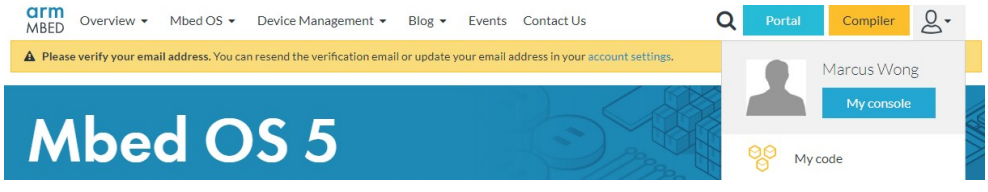
### 5.1. How to Run

An embedded software environment can be difficult to configure. To spare time dealing with GCC and Yotta configuration, the easiest way to run and flash the code written is to use the online Mbed OS compiler, which was used during development. Mbed OS is an open source operating system that supports the nRF52-DK. The steps to run and flash the code are as shown:

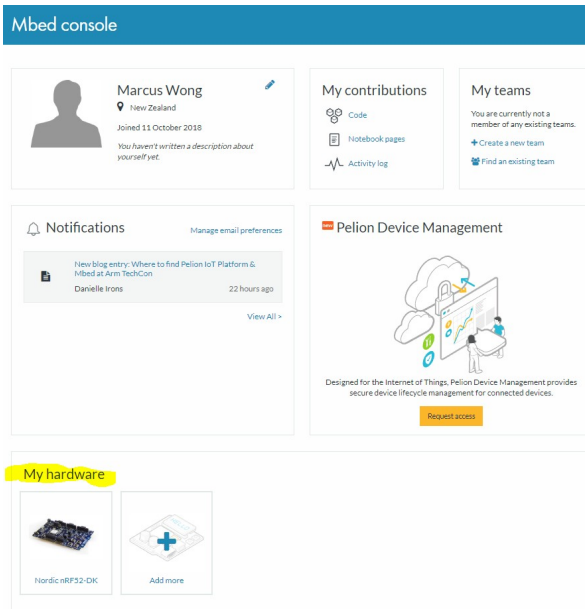
1. Go to https://os.mbed.com. A testing account has already been made for ease of use.
2. Enter the following credentials: <username: marcusjzw2> <password: uoa12345>



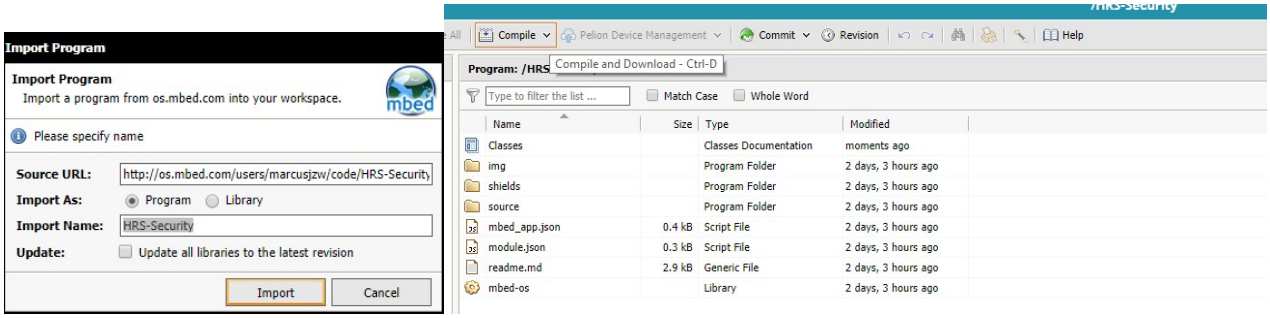

3. This account should already have the Nordic nRF52-DK compiler added in account configuration. You can check this by going to the top-right ‘user’ icon and clicking “My console”.



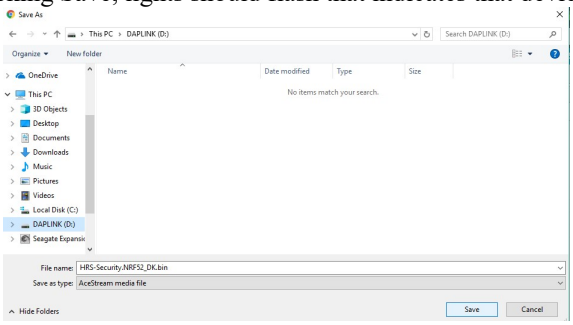
4. Under “my hardware”, it should say Nordic nRF52-DK. If it does not, simply click “Add hardware” and select the Nordic nRF52-DK.




5. Connect the nRF52-DK via Micro USB cable, to the PC. Ensure it is detected by the PC. It should come up as a Removable Storage device called ‘DAPLINK’. *Note: make sure the Power DIP switch is turned ‘ON’.*
6. Navigate here: [https://os.mbed.com/users/marcusjzw/code/HRS-Security](https://os.mbed.com/users/marcusjzw/code/HRS-Security). This is the public repository link for the source code developed in this research project. Click ‘Import to Compiler’.
7. The mbed OS Web IDE should load. Once it is done, click ‘Import’, then hit ‘Compile’.



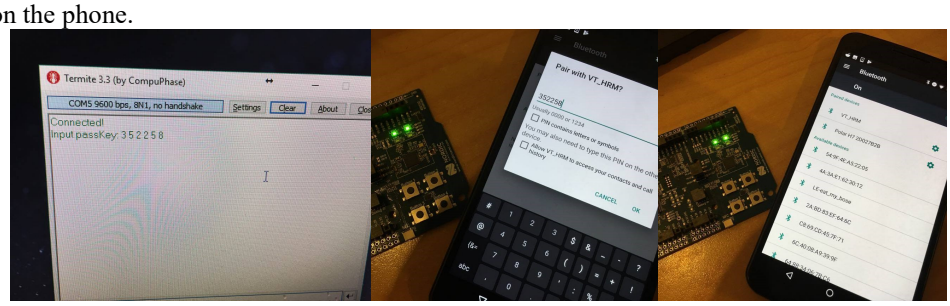

8. After a short while, a prompt will ask to download a .bin file.
9. Download the .bin file onto the nRF52-DK by directing the .bin to the root folder of the DAPLINK removable storage device. After clicking Save, lights should flash that indicates that device programming is occurring.



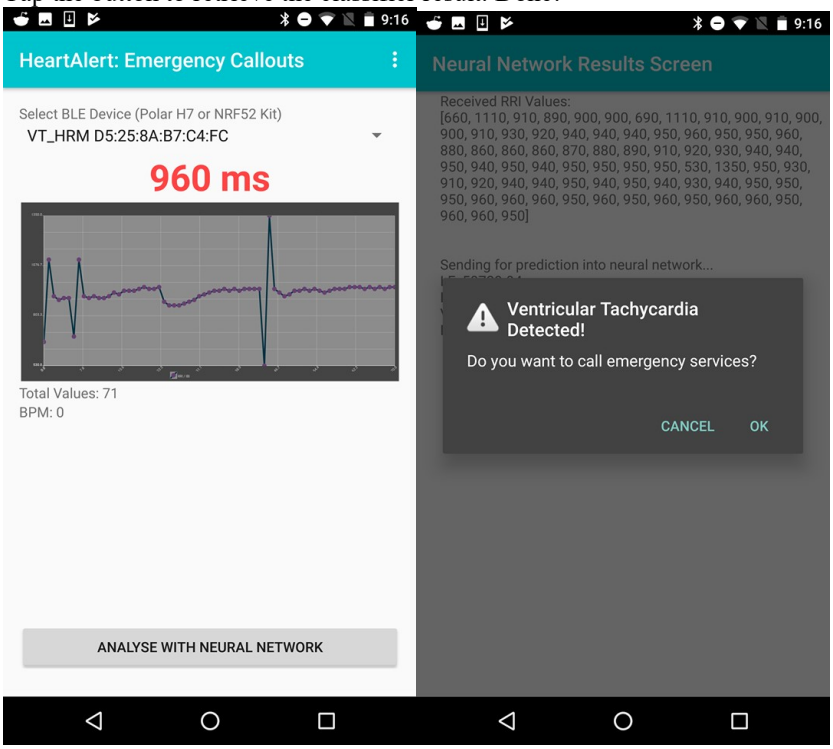

10. The lights on the device should be blank now. A power cycle will start running the software. This is verified through flashing lights occurring on LED1 (indicating advertising intervals) and a solid light on LED2.

With the device configured, the next step is to use the device on the developed app. To do this, follow these steps:

1. To connect the nRF52-DK on the phone, pairing must happen first. To do this, ensure that the device is still connected to the PC and open a serial program (Termite has been used and works). Make sure the correct COM port is configured to listen to the nRF52 Device.
2. Configure the connection with these settings: <baud rate: 9600> <data bits: 8> <stop bits: 1> <parity bits: 0>
3. On the Android phone, go to Settings -> Bluetooth -> Tap on the device called 'VT_HRM'. This is the name of the embedded device. A prompt on the phone should appear to ask for the passkey.
4. A passkey is a six-digit number which should now pop up on the serial program on the PC. Enter the six digits seen here on the phone.




5. The phone is now paired. You can now open Android Studio and deploy HeartAlert.apk (the application) using the steps described in Section 4.2. It is recommended to keep the Android phone plugged into the PC, so logging data can be checked.
6. Initiate the app and on the dropdown menu, select VT_HRM. **Make sure that after this, you wait for at least 30 seconds. This is because the socket automatically tries to discover a Polar H7 UUID – wait for the timeout!** The see all the processes going on during this wait, make sure Android Studio is configured at 'Logcat' viewing mode to see log messages, while the HeartAlert app is open on the phone.
7. Log messages should eventually start reporting receipt of R-R intervals. This is being done through the embedded device using the Heart Rate Measurement characteristic found in the Bluetooth-SIG approved Heart Rate Service, with the app parsing data transferred on the GATT layer correctly. This should be identical to the logs found in Figure 12 of the Report. The phone's UI should update to show the intervals, and the 'BPM' value should display 0.
8. After 60 intervals, the Neural Network button at the bottom of the app is clickable and received intervals can be classified.
9. Tap the button to retrieve the classifier result. Done!

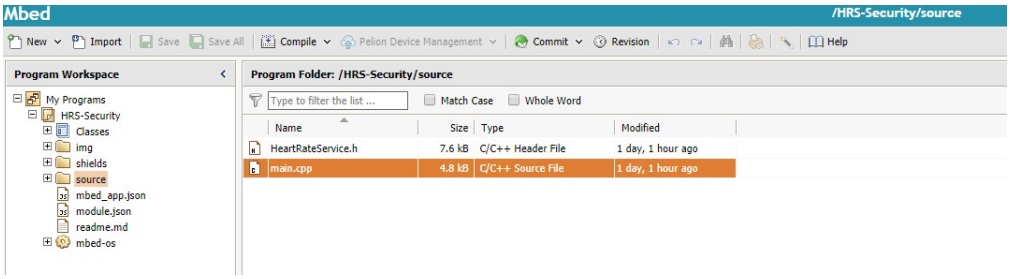


Currently, pressing ‘OK’ on the ventricular tachycardia detected screen does not do anything in this prototype, to avoid real emergency callouts being conducted. In a commercial solution, we would not have this screen – instead, a background service will initiate a call to emergency services automatically, sending phone GPS location information along with the request.

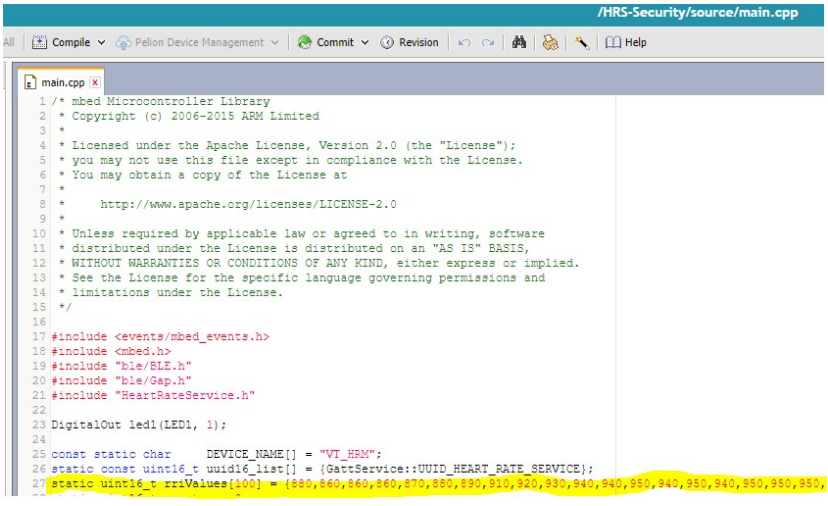
### 5.2. How to Load New VT Datasets

The default code tests with a sample of R-R intervals which are classed as VT. The initial plan was to utilize a filesystem structure on the embedded device to host different VT testing data; however, this was abandoned due to time constraints. Instead, to swap out different files, the following steps can be employed:

1. From the root directory of the project files, navigate to ‘signalprocessing/onset_prediction_vtvf/RRdata1/vt/’. This should be a familiar directory, containing all processed VT datasets from Section 2.
2. There is a file called ‘cplusplus_arraygen.py’. The objective of this Python script is to produce a code snippet, which is pasted onto the mbed OS nRF52-DK source code, to swap out the datasets emitted. The classifier present on the smartphone has been trained with 75% of data. The .txt files in this folder from 77_.txt up to and including 105_.txt are all examples of VT which has not been used in training data. Thus, they are used as testing data.
3. Call the script using the following command. You can replace ‘77_.txt’ with any .txt file from 77 to 105:
   $ python cplusplus_arraygen.py 77_.txt
4. A code snippet will be generated and printed to console. Now, open the mbed OS Web IDE with the nRF52-DK embedded code project opened, using the steps outlined previously. Navigate to the main.cpp file inside the ‘source’ folder.




5. Copy the code generated by the script onto Line 27 of the main.cpp file. The line to be replaced should be obvious; it is the long line intiating all values in the rriValues array.




6. Click Compile, program the device using the .bin file generated, as per instructions in the previous section.
7. Test the connection using the app. You (usually) do not need the re-pair/bond the device again, but there has been a single occasion where repairing is necessary.

### 5.3. Testing Procedures

In order to test the real-time classification accuracy of the machine learning model with VT-affected data, The steps outlined in the previous was repeated to go through all data between 77_.txt and 105_.txt in the 'signalprocessing/onset_prediction_vtvf/RRdata1/vt/' folder. The results recorded are shown in Table 2 of the Research Report (i.e. the second confusion matrix).

### 5.4. Other Research

Some initial research was spent exploring Windows PC to Bluetooth Low Energy functionality, to spoof VT datasets without needing an embedded device. However, due to poor platform support, this was dropped in favour of a dedicated Bluetooth Low Energy device.

Planned approaches using the nRF52-DK involved using example source code found in the Nordic SDK to implement a custom service known as 'Nordic UART Service'. The Service would allow transmission and receipt of Bluetooth Low Energy data over serial programs, as well as emulating the flow control mechanisms of the UART protocol itself using Bluetooth Low Energy Descriptors. However, the issue with this approach would be the development work required to rehaul the Android app. Due to the Polar H7 following Heart Rate Service, the app would need to be rewritten to also support data unpacking and parsing for a different service altogether. Because of this, the approach of implementing Heart Rate Service directly on the nRF52-DK was decided on.

Initial Heart Rate Service experimentation involved the use of examples in the Nordic SDK. However, they became a challenge to run due to the 32KB limit placed on free licenses of the MDK-ARM compiler (More information can be found here: http://www2.keil.com/mdk5/editions/lite). The two remaining choices were to move to a free compiler (e.g. GCC) or use the fact that the nRF52-DK is mbedOS-compatible. To speed up prototyping time, mbedOS was chosen. Due to ample support for Bluetooth Low Energy in the form of high level C++ APIs, development progressed better.

## 6. Conclusions

The steps to run, configure and test the software and firmware developed in this research project are outlined in this compendium. This included Python scripts for data wrangling, Python scripts for machine learning and cross validation, Java code for an Android app capable of Bluetooth Low Energy and TensorFlow model integration, and firmware in C++ for the spoofing of ventricular tachycardia data over Heart Rate Service. Overall results from this research project are promising.

## Appendix A

*Note: All appendices refer to files already included in the submission, so it does not make sense to copy them again. Instead, the Appendices will highlight in detail where to find specific files that are used to generate Figures/Tables in the submission. Note that most of these files are already discussed in the body of the Compendium.*

The final dataset used for machine learning (after data wrangling) can be found in the following folder, relative to the root directory:

'/machinelearning/hrv_nn/progress-after-2july/all_data_onsets.csv'

The .csv has no labels. Instead, the labels are in:

'/machinelearning/hrv_nn/progress-after-2july/readme-interpretdata.rtf'

## Appendix B

The Python code used to generate the confusion matrix (Table 1) can be found in '/machinelearning/hrv_nn/progress-after-2july/neuralnetwork-fulltrain.py

## Appendix C

The .txt outputs used to generate neuron heatmap data, along with the spreadsheet used to draw the neuron heatmap, can be found in '/machinelearning/hrv_nn/progress-after-2july/heatmaps'.

## Appendix D

The nRF52-DK datasheet can be found online. Alternatively, it has been downloaded into '/700-deliverables/nrf52-dk-datasheet.

## Appendix E

Heart Rate Service documentation can also be found online, here:
https://www.bluetooth.com/specifications/gatt/viewer?attributeXmlFile=org.bluetooth.service.heart_rate.xml&u=org.bluetooth.service.heart_rate.xml