Department of Electrical and Computer Engineering

Part IV Research Project

Project Report

Project Number: 97

Automated smartphone emergency callouts for heart disease sufferers

Author: Marcus Wong

Supervisor: Dr Avinash Malik

Secondary Supervisor: Dr Kevin Wang

8 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.

[illegible]

Marcus Wong

ABSTRACT: This research project explores the feasibility of an automated heart alert system, suitable to be worn comfortably by heart disease sufferers in their day-to-day lives. The system alerts for medical attention when anomalies in heart activity are detected. Physionet data was collected, cleaned and processed into heart rate variability (HRV) metrics. A back-propagation artificial neural network (ANN) was then developed using HRV metrics as inputs to classify heart condition events. The ANN achieved a 95.42% accuracy on PC for binary classification of ventricular tachycardia vs non-ventricular tachycardia. An Android smartphone app was built, with design considerations given to the timeliness and accuracy of the ANN model on a mobile device. Real-time R-R interval data is acquired on the app through a Bluetooth Low Energy connection between the smartphone and a Polar H7 Belt. Lastly, embedded software for the nRF52 was written to spoof diseased R-R interval readings. With the Polar H7 and nRF52 combined, testing concluded a real-time accuracy of 92.4%. Overall, the proposed system has been proved feasible. Possible future work may involve supporting the detection of more heart conditions and running clinical trials for medical device certification approval.

1. Introduction

Heart disease is a chronic condition that affects one in twenty people in New Zealand. A common side effect of this condition includes disruptions and irregularities to heart rhythm, known as heart arrhythmia. These events can be life-threatening if urgent medical attention is not provided.

Many heart disease sufferers are not monitored constantly in a clinical context. When not under medical care, emergency response time is important during a heart arrhythmia. Currently, there are many commercial solutions focused on providing one-click emergency callouts. These culminate in the form of a medical alert bracelet worn on the body, or a wall-mounted emergency button placed in a room.

However, literature has highlighted the issues faced with these existing systems, due to their fundamental reliance on human input [1]. Human input is often susceptible to misjudgment; studies have noted the tendency for at-risk individuals to delay calling emergency services due to underestimating the symptoms of life-threatening heart arrhythmia. There is also a clear correlation between heart disease sufferers and arthritis [2], as well as heart disease and dementia [3]. Both conditions affect a patient's physical and mental ability to call out emergency services, even if it is only one button press away.

A better solution is possible. By combining the fields of real-time heart monitoring, signal processing and machine learning, The proposed system reduces the dependence on human input by automatically calling emergency services when anomalies in heart activity are detected. The system is also low cost, uses familiar hardware (i.e. smartphone) and is simple to set up and use. The research outlined in this report describes the prototype development and design of the system.

2. System Diagram

The resulting system developed in this research project is outlined below.

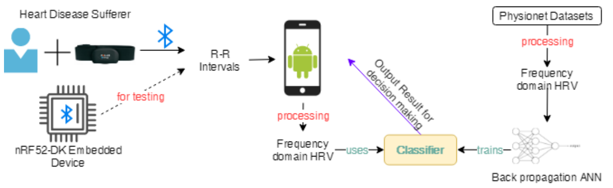


Figure 1: Components of the developed system

The overall app flow involves first connecting the smartphone to the Polar H7 belt. This is done via a drop down menu which displays all bonded Bluetooth Low Energy devices. Once connected, the belt will automatically send heart data known as a R-R interval, which is received and parsed by the smartphone. After fifty seconds, the app does further processing by converting the intervals into frequency domain HRV metrics and running the metrics through a trained machine learning classifier. A callout to emergency services is undertaken depending on the result.

Documentation in this report is structured chronologically in order of development, with machine learning conducted first. Discussions on the development of the machine learning classifier are explained in section 3. Discussions on smartphone app development and smart wearable integration are noted in section 4. Finally, embedded device development work is seen in section 5.

3. Machine Learning

3.1. Context

Machine learning was used in this project for heart condition classification. This was achieved by using an artificial neural network, feeding heart parameters as inputs. The choice of parameter was quantified by reviewing literature on different real-time heart monitoring approaches. Please refer to the Literature Review for more information.

As a result, data processing and subsequent machine learning work focused on heart rate variability (HRV)-based approaches to classification.

Frequency-domain based HRV metrics utilise power spectral density analysis to analyse heart condition. Power spectral density analysis is the technical process of decomposing a complex signal into simpler parts, whereby individual frequency components can be quantified in terms of their power or intensity. This analysis has been scientifically proven in medical journals to correlate to varying activity in the autonomic nervous system, depending on which spectrum bands were observed. The spectral bands found in literature to be useful in classifying heart condition [4] [5] were:

- very low-frequency power in the band 0.00-0.04 Hz, 'VLF'. This reading is often a predictor of inflammation and infection.
- Low-frequency power in the band 0.04-0.15 Hz, 'LF'. This reading correlates to sympathetic activity, which infers increasing heart rate among other uses
- High-frequency power in the band 0.15-0.4 Hz, 'HF'. This reading correlates to parasympathetic activity, which infers decreasing heart rate among other uses
- The ratio between low frequency and high-frequency LF/HF, correlating to sympathovagal balance, i.e. the level of 'nerve traffic' in the autonomic nervous system.

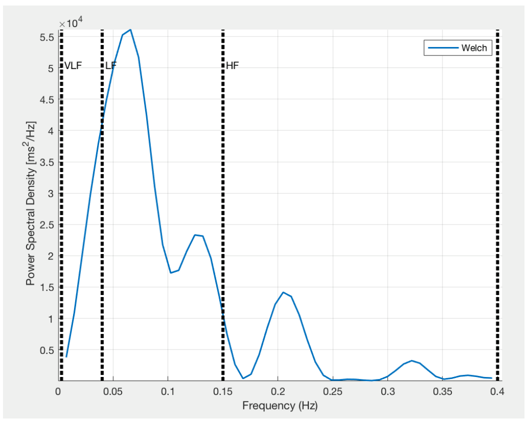


Figure 2: Spectrum analysis of a 2-minute ventricular tachycardia episode

Power extracted from the bands seen in Figure 2 serves as the HRV parameters used as inputs to the machine learning model.

3.2. Data Acquisition/Processing

3.2.1. Data Sources

A well-known source for physiological data is Physionet, which hosts a multitude of heart condition datasets. This includes patient data covering heart arrhythmia conditions such as atrial fibrillation, ventricular tachycardia and ventricular fibrillation episodes.

Heart rate variability in the frequency domain can be calculated using R-R intervals over a period of at least fifty seconds [6]. An R-R interval is the period between successive heartbeats, whereby HRV measures the variation between these beats. R-R intervals can be calculated from an ECG reading, using the Pan-Tompkins algorithm [7]. However, due to the availability of datasets already in R-R interval format, this was not necessary for this study.

The datasets selected included the Spontaneous Ventricular Tachyarrhythmia Database and the Normal Sinus Rhythm RR Interval Database. The objective of these datasets was to achieve a machine learning model capable of binary classification between ventricular tachycardia (VT) and not ventricular tachycardia. This condition was specifically chosen for classification as VT events tend to precede the more life-threatening condition, ventricular fibrillation. In the context of the system, ideally ventricular tachycardia would be detected with emergency services called before the onset of more serious conditions.

All datasets were downloaded in batches through rsync, a command-line file transfer tool.

3.2.2. Data Formatting

Datasets were downloaded from Physionet. These are usually in obscure file formats, designed to be opened using Waveform Database (wfdb), a software package used for reading and writing formats used by Physionet databases.

In this project's implementation, .qrs and .vt files were converted to a friendlier format, for further processing to occur using Python libraries. This involved using the command line version of wfdb to convert beat annotation data to a .txt file. The .txt file has each R-R interval in milliseconds, separated by a new line (\n). A Python script was used to loop through all prospective files to perform batch conversions on large datasets.

Although data samples from the Spontaneous Ventricular Tachyarrhythmia database contained both ventricular fibrillation (VF) and ventricular tachycardia (VT) episodes, the former was deliberately omitted from datasets in this research project. The reasoning behind this is that detection of VT (as the 'precursor' condition for VF) allows for the timely calling of emergency services, ensuring medical professionals handle potential life-threatening events that follow. There is also a lack of VF related data to sufficiently train an accurate model.

3.2.3. Data Processing

The R-R interval text files gathered from the data formatting step outlined in section 3.2.2 required conversion to frequency domain HRV, to serve as the input parameters of the neural network. This involves power spectral density analysis, resulting in a figure like the graph shown in Figure 1.

MATLAB is a popular tool for such operations, with useful built-in functions for signal processing applications. However, due to its low portability and the existence of a BSD-licensed Python open source library simply known as *hrv* [8], Python was used in this project for processing.

In this process, three design decisions were made.

The first involved the *technique* used for spectrum analysis. As with most computerised techniques, the resultant analysis is only an approximation to the full integral solution. Consideration was required into which technique would balance computational complexity with accuracy, keeping in mind that both timeliness and accuracy are important metrics in this system.

The technique chosen for spectrum analysis (which reasoning is elaborated on in the Compendium) is known as Welch's method [9]. Welch's method is an incremental improvement over standard periodogram spectrum estimations, due to its ability to reduce noise in imperfect data. This is very important with heart data, which is a stochastic process that is susceptible to artefacts.

Noise reduction from the Welch's method is achieved by averaging. The original signal is first segmented into several data segments, and the average spectra of individual pieces are calculated via FFT. Thus, peaks are not as pronounced, and variance is reduced. Figure 3 below visually explains this concept.

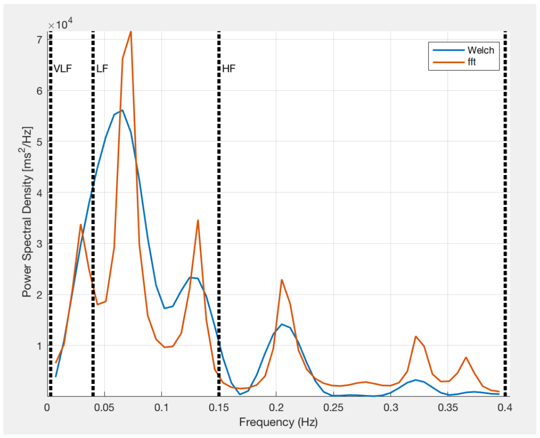


Figure 3: Comparing Fast Fourier Transform vs Welch's method on the sample in Figure 2

Another design decision made involved choosing an *interpolation method*. Interpolation involves constructing and estimating new data points given a set of existing data points obtained by sampling. Interpolation in this context helps to fill missing data points with relevant values.

Spline interpolation was used [10]. This involves using low-degree polynomials between each data point, choosing polynomials that fit smoothly together (called a "spline"). The result is a piecewise graph which has the most accuracy.

The last design decision involved the use of linear detrending with the R-R interval data provided. Detrending involves removing a trend from a time series sample, thus removing distortion from external factors. This was decided to be an important factor in ensuring the accuracy of the samples used for training and testing.

The resulting frequency domain parameters calculated from each sample were saved in .csv format, for easier future manipulation with Python libraries such as pandas. Data were formatted as shown, where each bracket signifies a unique column:

[Record no.] [LF] [LF/HF] [VLF] [HF] [Outcome]

The outcome column contained the class variable, whereby a '0' was printed in the absence of ventricular tachycardia, and a '1' was printed if the onset of ventricular tachycardia was present.

In total, three .csv files were produced: a file containing all relevant data from all datasets, one containing 75% of all data (later used for training) and one containing 25% of all data (later used for testing).

3.3. Neural Network Architecture

3.3.1. Choice of Tools/Languages

Keras is an open source deep learning library written in Python, which was used in this project for machine learning. The platform is a popular wrapper of TensorFlow, a framework used for building machine learning models. Keras is popular with researchers and novice data scientists, due to its focus on the high-level implementation of neural network models.

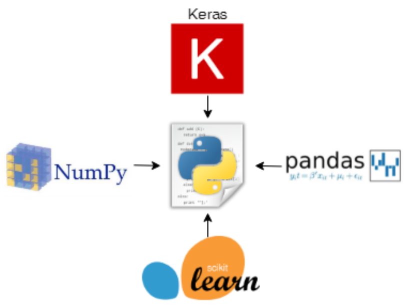


Figure 4: Visual representation of frameworks used

The ecosystem for machine learning in Python is unmatched. Having Python has the language of development is also useful for integrating other libraries for data manipulation (pandas), numerical analysis (numpy) and model validation (scikit-learn). These libraries were used in conjunction with the models developed using Keras for its respective purposes.

3.3.2. Design Decisions

Due to the heuristic nature of artificial neural network models, initial model configurations aimed to mimic the settings used in literature that proved successful in detecting serious heart conditions, such as congestive heart failure [11].

This meant the use of a back-propagation, gradient descent neural network. In simple terms, back-propagation involves the calculation of a gradient used to tune weights in the neural network. The gradient is calculated by analysing the error between the actual and desired output. Gradient descent, on the other hand, is an optimisation algorithm aimed at gradually minimising this error (formally known as a cost function), thus "training" the network. It does this by taking steps to approach the minimum of a function.

Based on literature, a starting configuration was decided as shown in Figure 4, using four neurons in the input layer to signify each HRV parameter, one hidden layer with 15 neurons and a sigmoid output layer.

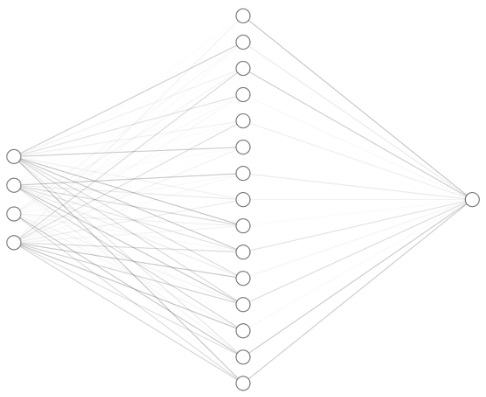


Figure 5: Visualisation of the initial neural network architecture, courtesy of the NN-SVG tool.

Design decisions were required to tweak the model to improve accuracy, as Keras reported an accuracy of only 71.86%. This was a poor result, considering a random guess for binary classification would tend to 50% given a large enough dataset. Experimentation with different neuron configurations was then conducted. Two points of focus involved tweaking *activation function* settings and choosing a better *neuron configuration*.

Activation functions determine whether a neuron chooses to 'activate' or not, given a set of inputs from the previous layer. For example, an activation function could be in the form of a step function which has two distinct states, 0 or 1.

In the context of artificial neural networks, popular activation functions are usually non-linear. This is done to achieve activations with a level of granularity (e.g. 0.56 instead of 1), which aids backpropagation decision making in future layers much better. In the model developed in this project, two non-linear functions were used in different layers of the system to achieve maximum accuracy.

The first non-linear function used in this project is known as the sigmoid function. The sigmoid function can be quantified as such:

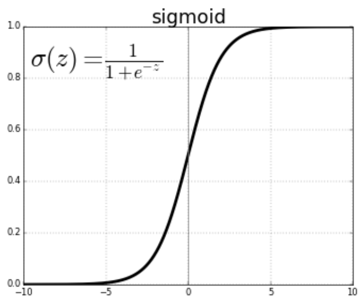


Figure 6: A sigmoid activation function

This function is used in the *output layer* of the model developed to determine whether the sole neuron should fire or not. The function is effective in classification problems - the steep curve between x = -2 and x = 2 ensures that small changes in the value of X will cause significant changes to Y (the output). By bringing Y values to either end of the curve, clear distinctions can be made for prediction.

The second non-linear function used is known as the ReLU function.

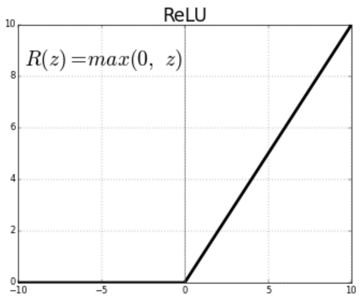


Figure 7: A ReLU activation function

This activation function is used in both *hidden layers* of the project's implementation. The advantage of ReLU is its sparsity of activation – in contrast to sigmoid and other popular activation functions, neurons tend to activate no matter what inputs are given, i.e. a weak activation could still produce an output of 0.0001. This is computationally expensive; ideally only a few important neurons would fire which reduce the time complexity of the operation.

With ReLU having an output of 0 for negative values of X, many neurons are not activated at all. This is important for classifier timeliness in an emergency callout system.

Neuron configuration was also considered carefully, to prevent overfitting. Surprisingly, there are many conflicting 'rules of thumb' regarding the number of hidden layers used, and the number of neurons in each hidden layer [12]. Developing objective measures for building neural network architectures is still a current topic of research, and final decisions regarding the number of neurons selected are discussed in section 3.3.4.

3.3.3. K-Fold Cross Validation

Initially, models were developed and tested using a classic train/test split. 75% of the dataset was used for training while 25% of the data was used for testing the accuracy of the predictor. However, this proved susceptible to bias with large variances between tests of the same model. A better solution was explored.

Cross-validation is a tool used to define the concept of accuracy. This involves testing the effectiveness of predictive models against unseen data, allowing potential problems like overfitting and poor real-time classification to be diagnosed.

K-fold is an industry standard technique for cross-validation, which was used in the model developed in this project. This technique was chosen due to its resourcefulness in 'resampling' data given a limited dataset. This was very helpful considering the (relative) lack of HRV data correlating to ventricular tachycardia. The technique is described in the pseudocode below:

```
1. The dataset is shuffled into k equally sized groups
2. Signal one group to be retained as the validation set
3. Signal the remaining k-1 groups to be used as the training set
4. The model is fitted on training data and tested on the validation set
5. Accuracy is recorded, and the model is discarded
6. The process is then repeated k times (i.e. the "folds"), whereby each group is used exactly once as the validation set
7. The accuracy of the model overall is the average accuracy of each fold
```

StratifiedKFold, a module from the popular scikit-learn Python library, was used to evaluate Keras models, where *k* = 4. Achieving proper k-fold cross validation was essential in ensuring that future neuron configuration tweaks could be accurately compared to one another. The culmination of this can be seen in the next section discussing the neuron heatmap.

3.3.4. Neuron Heatmap

After experimentation with various 'best practice' equations for building accurate neural network models (with limited results), a neuron heatmap was developed to determine the best accuracy possible. This involved repeatedly running training and testing over a wide range of neuron configurations. Accuracy and standard deviation were recorded for each configuration.

A baseline of two hidden layers was used instead of one. One hidden layer did not provide high accuracy rates; given the complexity of differentiating VT and non-VT events (where VT is subtler than the congestive heart failure and fibrillation events classified in previous literature), two hidden layers proved more intuitive to achieve non-linear relationships.

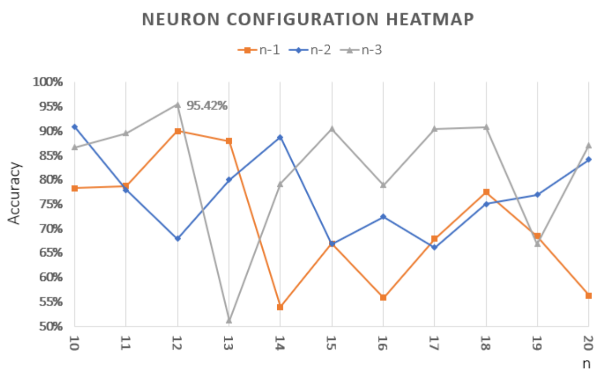


Figure 8: Neuron configuration heatmap

To explain Figure 8 above *n* denotes the number of neurons in the first hidden layer. *n – x,* where *x = {1, 2, 3...}* denotes the number of neurons in the second hidden layer relative to the first. Only three lines are shown, although more were tested. The highest accuracy can be seen marked on the map.

3.4. Results & Discussion

3.4.1. Results

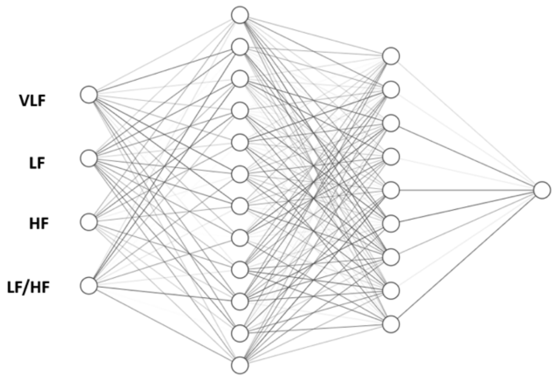


Figure 9: Visualization of final neural network architecture

As seen in the neuron heatmap, the configuration with 12 neurons in the first layer and 9 neurons in the second layer proved the most accurate at **95.42%.** Additionally, this configuration also had the smallest variance (not pictured), with a standard deviation of just 2.5% over ten training sessions.

The figure below highlights the confusion matrix for the model developed:

N = 242	Predicted: No VT	Predicted: VT
Actual: No VT	133	2
Actual: VT	9	96

Table 1: Confusion matrix results

$$Sensitivity = \frac{TP}{TP + FN} = 93.66\% \quad (1)$$

$$Specificity = \frac{TN}{TN + FP} = 97.96\% \quad (2)$$

$$Accuracy = \frac{TP + TN}{\sum population} = 95.42\% \quad (3)$$

3.4.2. Discussions on Accuracy

Despite good accuracy overall at 95.42%, the rate of false negatives is elevated compared to false positives. In the context of an emergency callout system, false negatives are by far more problematic than false positives. Not reporting a heart anomaly would degrade the system to the level of existing solutions (where manual input is required for a callout).

It is likely that slight overfitting may have occurred, causing the model to make predictions from noise artefacts. In the pursuit of optimizing accuracy, less consideration was taken on the 'type' of error that was occurring. From reflection, it may have been more advantageous to optimize false negative rates instead of overall accuracy.

Alternative accuracy metrics such as the *F1 Score* may have been more suitable in this project's context. In the total dataset, more training data was supplied for non-VT events (135) compared to VT events (105). This unevenness means that the baseline accuracy for binary classification is 56.25% instead of 50% if the system is randomly guessing.

The F1 Score is calculated without this bias. By calculating the weighted average of precision and recall, false positive and false negative rates are compared relative to their local sample sizes rather than the total population.

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} \quad (4)$$

$$precision = \frac{TP}{TP + FP} \quad (5)$$

$$recall = \frac{TP}{TP + FN} \quad (6)$$

In the context of the developed system, a rate of 2 false positives and 9 false negatives gives a precision rate of 97.96%, a recall rate of 91.43%, and a F1 Score of **94.57%**. This is slightly less than the more optimistic accuracy metric, due to poorer recall performance.

Upon further analysis, the summation of both ideas presented can be seen with the *F2 score* [13].

$$F_2 = 5 \cdot \frac{precision \cdot recall}{4 \cdot precision + recall} = 92.86\% \tag{7}$$

Taking the F2 score into account, the machine learning model developed in this project has an accuracy of **92.86%**. Mathematically speaking, the F2 score places more weighting to recall, due to the multiplier for precision in the denominator. As recall is mostly affected by the rate of false negatives, the F2 score is worse than other metrics. Overall, this as a good way to both normalise accuracy from uneven datasets and emphasise the impacts of false negatives.

4. App-Wearable Integration

4.1. Real-time Heart Monitoring

4.1.1. Polar H7 Wearable

The Polar H7 belt is an electrode-based heart rate monitor, currently used in fitness contexts. The belt is worn around the upper waist and communicates heart data wirelessly over the Bluetooth Low Energy protocol.

This belt was used in this project due to its ability to output R-R interval values in real-time, over a protocol that is compatible with most modern smartphone devices. R-R interval values would be gathered on the smartphone and processed into HRV metrics in real-time, and then run through the classifier. Methodology detailing those steps are seen in sections 4.2.3 and 4.2.4.

The Polar H7 has also been the subject of literature. Multiple studies have proven the superior accuracy of the Polar H7 monitor over other commercially available devices, with R-R interval values correlating heavily (>98%) to the peaks found in an ECG reading [16] [17]. Overall, this makes the Polar H7 and the Bluetooth Low Energy stack an attractive option for this project.

4.1.2. Connection Establishment

The Bluetooth Low Energy (BLE) protocol stack consists of multiple ‘layers’ of protocols. Two intermediary protocols are known as Generic Access Profile (*GAP*) and Generic Attribute Profile (*GATT*) are of interest in connection establishment and data transmission.

The GAP layer handles connection functionality, such as device discovery, handshaking events and security features. In this project, GAP settings were configured to adhere to existing Polar H7 firmware settings.

A connection is established via a central-peripheral relationship. This involves wearing the Polar H7 belt to begin channel advertisement as a peripheral device. The central device (the smartphone) then configures to scan for channels, opting to initiate a connection event to the peripheral if the device UUID (Universally Unique Identifier) belongs to a Polar H7.

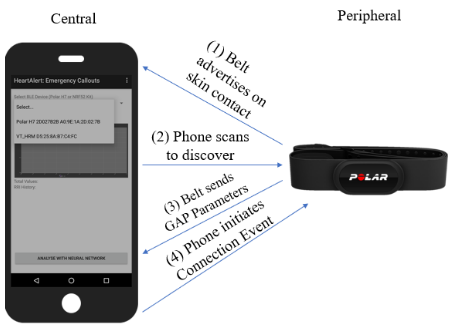
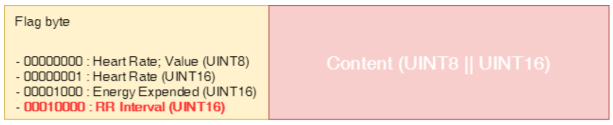

Central
Peripheral
HeartAlert: Emergency Callouts
Select BLE Device (Polar H7 or NRF52 Kit)
Select...
Polar H7 2D027B2B A0:9E:1A:2D:02:7B
VT_HRM D5:25:8A:B7:C4:FC
Total Values:
RRI History:
ANALYSE WITH NEURAL NETWORK
(1) Belt advertises on skin contact
(2) Phone scans to discover
(3) Belt sends GAP Parameters
(4) Phone initiates Connection Event
POLAR

Figure 10: Connection Establishment on the GAP layer

A device’s UUID is dependent on the *Services* it supports. In Bluetooth Low Energy, Services are a collection of characteristics that encapsulate device behaviour, existing on the GATT layer. In this layer, the format for data packets is specified to ensure transmission runs smoothly. The Polar H7 implements the Bluetooth Special Interests Group approved Heart Rate Service [16].

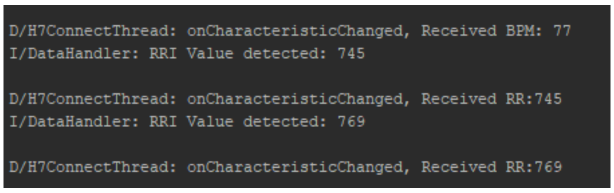
A characteristic existing in this service is known as Heart Rate Measurement, which contains information about R-R intervals in its data packet. The format of a Heart Rate Measurement data packet can be seen below:



Flag byte	Content (UINT8 || UINT16)
- 00000000 : Heart Rate; Value (UINT8) - 00000001 : Heart Rate (UINT16) - 00001000 : Energy Expended (UINT16) - 00010000 : RR Interval (UINT16)	

Figure 11: Data Format mandated for the Heart Rate Measurement characteristic

Given the information seen in Figure 11, the smartphone app was developed to check the first byte and perform bitwise operations to check for the presence of 0x10, i.e. whether bit 4 (counting from LSB) is high. If the bit is high, the data packet is read in as R-R interval data. This is repeated for receiving incoming beats per minute data packets, as BPM is also displayed on the app.



```
D/H7ConnectThread: onCharacteristicChanged, Received BPM: 77
I/DataHandler: RRI Value detected: 745

D/H7ConnectThread: onCharacteristicChanged, Received RR:745
I/DataHandler: RRI Value detected: 769

D/H7ConnectThread: onCharacteristicChanged, Received RR:769
```

Figure 12: Android Logcat results on parsed incoming data

The app was also developed to handle the occasional presence of more than one R-R interval received during one connection event. This is due to connection events being handled periodically, whereby in that time multiple R-R intervals could arrive (a standard R-R interval can be between 600-1200ms). Referral to Polar H7 documentation [19] was required to ensure the belt hardware's buffers were read by the smartphone app to capture all relevant data.

4.2. Software Architecture

4.2.1. Motivation

A smartphone application was proposed to test the feasibility of a portable system, given the relative reduction of computing power compared to a PC. Hypothesized issues included a lack of timeliness in general processing, inaccurate power spectral density estimations leading to reduced classifier accuracy, and issues with handling data from a heart monitoring device.

4.2.2. Design Choices

The application developed for this project is run on Android. As a result, the codebase for the app is written in Java.

Android was chosen because the platform can be easily integrated with TensorFlow machine learning models. This brings the challenge of converting the existing Keras models to compatible formats. This process is elaborated in section 4.2.4.

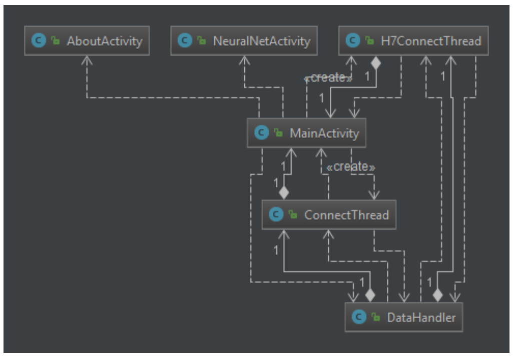


Figure 13: Android Application Class Diagram

The simplified class diagram outlined in Figure 13 showcases the relationships between different classes in the app (inherited Android classes are omitted). The dashed arrows represent dependency relationships, where solid lines with diamonds represent an aggregation relationship. The MainActivity class instantiates ConnectThread and H7ConnectThread, responsible for handling Bluetooth Low Energy connection events. Both classes interface with DataHandler to serve data.

During app development, object-oriented principles were followed to ensure code quality and reusability. This involved encapsulating classes with getters and setters to read and write to private members. Classes were split to handle unique tasks – for example, the ConnectThread is designed purely to handle GAP layer interactions (starting discovery and UUID checking) while the H7ConnectThread handles GATT layer interactions (service discovery, parsing data). Multithreading was used in these two classes to improve app performance.

The DataHandler class follows the singleton design pattern. This pattern is useful for classes that coordinate actions across a system where only one instance should exist. In DataHandler's case, it aims to receive parsed data in from H7ConnectThread and notify UI classes when new data arrives.

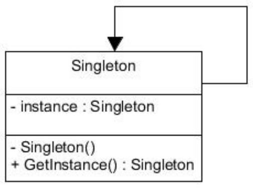
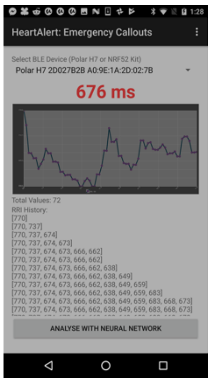


Figure 14: A Singleton class was used to make all classes use one instance of DataHandler

A static instance of DataHandler named *dd* is initialized in the class itself. For other classes to access DataHandler, a static method named *getInstance()* simply returns the *dd* object. In this way, classes can access DataHandler methods through a single instance (without instantiating their own).

The developed app uses the observer design pattern to allow front-end UI changes to be displayed upon incoming real-time data. For example, on the MainActivity screen there is a graph showing R-R interval data received, as shown below:


1:28

HeartAlert: Emergency Callouts

Select BLE Device (Polar H7 or NRF52 Kit)

Polar H7 2D027B2B A0:9E:1A:2D:02:7B

676 ms

Total Values: 72

RRI History:

[770]

[770, 737]

[770, 737, 674]

[770, 737, 674, 673]

[770, 737, 674, 673, 666, 662]

[770, 737, 674, 673, 666, 662]

[770, 737, 674, 673, 666, 662, 638]

[770, 737, 674, 673, 666, 662, 638, 649]

[770, 737, 674, 673, 666, 662, 638, 649, 659]

[770, 737, 674, 673, 666, 662, 638, 649, 659, 683]

[770, 737, 674, 673, 666, 662, 638, 649, 659, 683, 668, 673]

[770, 737, 674, 673, 666, 662, 638, 649, 659, 683, 668, 673]

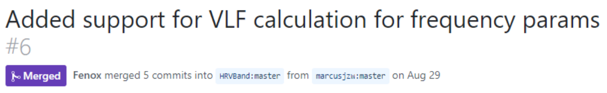
Figure 15: R-R interval graph seen in MainActivity

To update the UI in a timely fashion, the DataHandler class extends from a built-in Android class, *Observable*. Combined with MainActivity extending the built-in class *Observer*, DataHandler can *notifyAllListeners()* when a R-R interval is received and processed. This triggers MainActivity to call *update()* on a Runnable thread, which writes the R-R interval to a list and runs logic to display the graph.

4.2.3. HRV Conversion & Open Source Contributions

Although R-R interval to HRV conversion was implemented in prior steps, it was done so in Python for dataset processing purposes. This is not compatible with the Java-based Android platform.

An Apache-licensed Java library known as *hrv-lib*[18], was imported to the Android app to initially serve this purpose. However, it was soon discovered that its spectrum analysis missed the very low frequency (VLF) parameter, i.e. power in the 0.00 to 0.04 Hz frequency band. Due to this, time was spent in this research project to add VLF calculation functionality in the library.


Added support for VLF calculation for frequency params #6

Merged Fenox merged 5 commits into `HRVBand:master` from `marcusjzw:master` on Aug 29

Figure 16: Successful pull request to hrv-lib

Due to the use of a Façade design pattern, codebase changes were high-level and did not require changes to the underlying spectral estimation technique. Unit tests were written to validate functionality, and a pull request was made to integrate changes. The repository utilized Travis CI as a continuous integration service, which ran the unit tests automatically. GitHub repo contributor *Fenox* approved the successful pull request.

4.2.4. Neural Network Integration

After the successful integration of *hrv*-lib to the Android app, the developed neural network model was ported to Android. This involved the conversion of the model from Keras to TensorFlow.

Trained Keras models are saved in a .h5 file format. The Android platform, however, requires models in .pb format. The conversion from .h5 to .pb was conducted through a Python script that "froze" nodes, i.e. converted Keras variables to TensorFlow constants, saved neuron weights and pruned out any nodes that do not contribute to output predictions.

The resulting .pb file that was produced is known as a Protocol Buffer (*protobuf*) file. Protobufs are a form of serialized structured data, akin to XML and JSON. In this project, the .pb inference model was used as an input to TensorFlowInferenceInterface, a TensorFlow API compatible with Android used for fetching predictions from a protobuf formatted classifier.

The use of TensorFlowInferenceInterface API calls was used in NeuralNetworkActivity, an activity that can be entered after fifty seconds have elapsed. HRV metrics are fed in as inputs to the classifier using the *feed()* method. The classifier is then run using the *run()* method. Lastly, the *fetch()* method is used to return the prediction.

4.3. Results: Timeliness & Accuracy

Timeliness refers to how long the smartphone system takes to respond and act. It is naturally an important attribute to consider in an automated emergency callout system, to maximize survival chance during a life-threatening event.

TimingLogger, a built-in class on the Android platform, logs timing splits through simple method calls. By logging the time taken to finish HRV calculations, and the time taken to run and fetch from the neural network classifier, the total system latency can be measured as such:

1. Results in literature prove that all metrics required for HRV analysis can be estimated accurately with just 50 seconds worth of R-R intervals [11]. Thus, 50 seconds worth of R-R interval readings is gathered.
2. TimingLogger results for HRV conversion: 766ms
3. TimingLogger results for neural network processing time: 871 ms

Therefore, the total system latency is 51.637 seconds. Given that the average emergency service callout requires 7-10 minutes to arrive [20][21], this is an acceptable amount of latency.

The accuracy of the machine learning model was also retested on the app. This was done to ensure the smartphone-based power spectral density and power extraction process via *hrv-lib* was precise enough to produce similar levels of accuracy found on the original Keras model written in Python.

Testing involved wearing the belt and going through the process of R-R interval collection, conversion to HRV, and classification. From 50 repeated tests on a healthy male randomly sampled over four days, 48 tests came back with a healthy result, and 2 tests resulted in an "emergency callout" result. This hints that the app is functioning well; a false positive rate of 4% on a smartphone is congruent with the 97.96% precision rate calculated during model validation done on a computer.

5. Embedded Device Development

5.1. Motivation

As seen in the previous section, measuring the occurrence of "false positives" in real-time is easily done, due to the abundance of healthy human subjects. However, without a clinical trial, it is difficult to test the occurrence of "false negatives" in a patient at risk of ventricular tachycardia.

Thus, the use of an embedded device to 'spoof' diseased datasets was ideated. By following the same 'Heart Rate Service' data format as the Polar H7, a Bluetooth Low Energy enabled embedded device can be programmed to send R-R interval data from a diseased patient (taken from Physionet datasets). From this, a measure for the real-time accuracy of "false negatives" on the smartphone can be taken. The steps below highlight the progress made.

The device chosen for development is Nordic Semiconductor's nRF52 Development Kit, which contains a Bluetooth Low Energy chip.

5.2. Development

To speed up development time and to bypass MDK-ARM compiler limitations (explained further in the Compendium), the mbedOS platform was used. mbedOS is an open-source embedded operating system designed for the rapid deployment of embedded software, written in C++. Libraries are available to interface with Bluetooth Low Energy hardware peripherals on-chip.

Examples were used to establish template code for exposing a Heart Rate Service with a Heart Rate Measurement characteristic [16]. Minor tweaks were done to add bonding to prospective connections for the device to integrate seamlessly with the app (the app requires device bonding to discover connections).

The template code was designed to emit heart rate, instead of R-R intervals. As such, work was done to set flags and allow values to be sent in the R-R interval bit fields, as per the Heart Rate Measurement data packet format as shown in Figure [x].

Finally, .txt files of R-R intervals corresponding to VT-affected readings were simply loaded in memory in the form of a vector. Values were popped off when sent through BLE to the smartphone. In contrast to a file system, this simple solution was possible due to increasing the application stack size to 8KB, where R-R interval readings were roughly 4-6KB in size.

5.3. Results: Real-Time Classification

29 VT-affected samples were run through the nRF52 device to broadcast R-R intervals to the app. This was done by using held out data from the Spontaneous Ventricular Tachyarrhythmia database. From this, 3 samples reported a false negative with a result of "No anomalies detected".

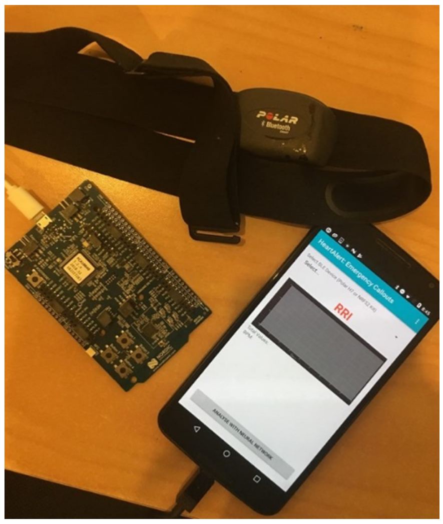


Figure 17: All three hardware components required for real-time testing: Polar H7, nRF52 and Android device

From the results collected from both the embedded device and through the Polar H7, the real-time classification of VT arrhythmia on a smartphone can be summarized in a confusion matrix.

N = 79	Predicted: No VT	Predicted: VT
Actual: No VT	48	2
Actual: VT	4	25

Table 2: Confusion matrix for real-time classification accuracy using the Polar H7 and nRF52 Dev Kit

This results in an accuracy of 92.4%. Overall, accuracy is only degraded slightly on a smartphone, from the 95.42% accuracy originally found on the model run on PC.

6. Conclusion

The novelty of the research proposed involves the combination of two research fields; studies in real-time heart monitoring techniques/HRV with smartphone-based neural networks. The resulting system was largely successful, with binary classification between VT and non-VT events achieving an accuracy of 95.42% on PC and 92.4% from real-time classification on a smartphone. Considerations and discussions were made about neural network design techniques, the use of cross-validation for model validation and alternative metrics like the F1 Score in this project. Design decisions were also made on software architecture and choice of design patterns in the development of the app.

The use of an embedded device to 'spoof' Heart Rate Service readings proved to be difficult, but successful. This allowed further testing of diseased states when no diseased patients are available.

Overall, this research project has contributed to science and technology by establishing the feasibility of a low-cost, automated emergency callout system that can be easily scaled up to support heart disease sufferers in New Zealand and beyond. The solution improves upon existing 'manual input' devices, and ultimately provides better survival chances during life-threatening events.

7. Future Work

There is plenty of future work to be conducted before a commercially available system can be launched. One aspect of the system that could be improved would be adding support for other common arrhythmias, such as atrial fibrillation. This would naturally allow more undesirable states to be picked up by the system, thus improving the usefulness of the system to heart disease sufferers. However, attempting this would require further development of the existing machine learning model beyond binary classification, which may exponentially increase the computational complexity of the classifier. Timeliness of the system, especially on a smartphone, will need to be closely monitored.

Another potential point for future work would be to expand the app to more platforms outside Android. From elementary research, it is possible to convert Keras models to CoreML, the machine learning framework for iOS. By supporting both iOS and Android, over 97% of the total mobile operating system market is covered [21].

Lastly, clinical trials will need to be conducted with real patients to test results further. Certifications from the FDA and similar boards will likely use clinical trial results to determine system effectiveness. This is a necessary step towards gaining approval to distribute and sell medical devices.

8. Acknowledgements

Many thanks to my primary supervisor, Avinash Malik, for guiding my project along – especially during the machine learning parts of the project. Your advice was invaluable for ensuring the resulting model was properly tested and validated.

I'd also like to acknowledge my prior project partner, Kale Greenwood-Evans, who regrettably had to withdraw from the research project in early May due to sickness. The timing was unfortunate; get better soon.

9. References

[1] J. R. Finnegan *et al.*, "Patient Delay in Seeking Care for Heart Attack Symptoms: Findings from Focus Groups Conducted in Five U.S. Regions," *Prev. Med.*, vol. 31, no. 3, pp. 205–213, Sep. 2000.

[2] C. Turesson, A. Jarenros, and L. Jacobsson, "Increased incidence of cardiovascular disease in patients with rheumatoid arthritis: results from a community-based study," *Ann. Rheum. Dis.*, vol. 63, no. 8, pp. 952–955, Aug. 2004.

[3] B. N. Justin, M. Turek, and A. M. Hakim, "Heart disease as a risk factor for dementia," *Clin. Epidemiol.*, vol. 5, pp. 135–145, Apr. 2013.

[4] Task Force of The European Society of Cardiology and The North American Society of Pacing and Electrophysiology, "Heart rate variability: Standards of measurement, physiological interpretation, and clinical use," *Eur Heart J*, vol. 17, p. 28, 1996.

[5] H. Usui and Y. Nishida, "The very low-frequency band of heart rate variability represents the slow recovery component after a mental stress task," *PLoS ONE*, vol. 12, no. 8, Aug. 2017.

[6] L. Salahuddin, J. Cho, M. G. Jeong, and D. Kim, "Ultra Short Term Analysis of Heart Rate Variability for Monitoring Mental Stress in Mobile Settings," in *2007 29th Annual International Conference of the IEEE Engineering in Medicine*

and Biology Society, Lyon, France, 2007, pp. 4656–4659.

[7] V. Timothy, A. S. Prihatmanto, and K. H. Rhee, "Data preparation step for automated diagnosis based on HRV analysis and machine learning," in *2016 6th International Conference on System Engineering and Technology (ICSET)*, 2016, pp. 142–148.

[8] "rhenanbartels/hrv: A Python package for heart rate variability analysis." [Online]. Available: https://github.com/rhenanbartels/hrv.

[9] "Low-Complexity Welch Power Spectral Density Computation - IEEE Journals & Magazine." [Online]. Available: https://ieeexplore-ieee-org.ezproxy.auckland.ac.nz/abstract/document/6525419. [Accessed: 07-Oct-2018].

[10] G. D. Clifford and L. Tarassenko, "Quantifying Errors in Spectral Estimates of HRV Due to Beat Replacement and Resampling," *IEEE Trans. Biomed. Eng.*, vol. 52, no. 4, pp. 630–638, Apr. 2005.

[11] M. I. Obayya and F. E. Z. Abou-Chadi, "Classification of heart rate variability signals using higher order spectra and neural networks," in *2009 International Conference on Networking and Media Convergence*, 2009, pp. 137–140.

[12] F. S. Panchal and M. Panchal, "Review on Methods of Selecting Number of Hidden Nodes in Artificial Neural Network," p. 10, 2014.

[13] ClustEval, "F2-Score," *Clusteval | Integrative Clustering Evaluation Framework*. [Online]. Available: https://clusteval.sdu.dk/1/clustering_quality_measures/5. [Accessed: 07-Oct-2018].

[14] S. Gillinov *et al.*, "Variable Accuracy of Wearable Heart Rate Monitors during Aerobic Exercise," *Med. Sci. Sports Exerc.*, vol. 49, no. 8, pp. 1697–1703, Aug. 2017.

[15] S. W. Cheatham, M. J. Kolber, and M. P. Ernst, "Concurrent Validity of Resting Pulse-Rate Measurements: A Comparison of 2 Smartphone Applications, the Polar H7 Belt Monitor, and a Pulse Oximeter With Bluetooth," *J. Sport Rehabil.*, vol. 24, no. 2, pp. 171–178, May 2015.

[16] "Heart Rate Service." [Online]. Available: https://www.bluetooth.com/specifications/gatt/viewer?attributeXmlFile=org.bluetooth.service.heart_rate.xml. [Accessed: 07-Oct-2018].

[17] "H6, H7, H10 and OH1 Heart rate sensors - Polar Developers." [Online]. Available: https://developer.polar.com/wiki/H6,_H7,_H10_and_OH1_Heart_rate_sensors. [Accessed: 23-Aug-2018].

[18] "HRVBand/hrv-lib - A Java library that provides methods to calculate HRV." [Online]. Available: https://github.com/HRVBand/hrv-lib.

[19] "Emergency Ambulance Service national performance reports," *Ministry of Health NZ.* [Online]. Available: https://www.health.govt.nz/new-zealand-health-system/key-health-sector-organisations-and-people/naso-national-ambulance-sector-office/emergency-ambulance-services-eas/performance-quality-and-safety/emergency-ambulance-service-national-performance-reports. [Accessed: 07-Oct-2018].

[20] H. K. Mell, S. N. Mumma, B. Hiestand, B. G. Carr, T. Holland, and J. Stopyra, "Emergency Medical Services Response Times in Rural, Suburban, and Urban Areas," *JAMA Surg.*, vol. 152, no. 10, pp. 983–984, Oct. 2017.

[21] "Mobile Operating System Market Share Worldwide," *StatCounter Global Stats*. [Online]. Available: http://gs.statcounter.com/os-market-share/mobile/worldwide. [Accessed: 07-Oct-2018].