ECG Analysis through Deep Learning and IoT devices

Paul Lussier

Advisor: Prof. Chen-Hsiang (Jones) Yu, Ph.D.

School of Computing and Data Science, Wentworth Institute of Technology

{lussierp, yuj6}@wit.edu

***Abstract*—The current standard for catching many heart conditions before symptoms arise is by getting a reading off of an electrocardiogram (ECG). While revolutionary in the detection and understanding of heart problems, ECG machines are costly and time consuming meaning many people don’t see the benefits of going to get one taken. This behavior directly opposes the possibility of catching the issue before it becomes problematic. With the rise in efficacy of Internet of Things technology we believe that it is possible for a small, affordable, and relatively unobtrusive device to be created with the goal of collecting and classifying an ECG using a raspberry pi and an AD8232 ECG sensor. These two pieces coupled with a deep learning model trained on data from PhysioBank, a large database of classified heart rhythms could provide a solution for people who do not want to go to the hospital for an ECG. We chose to focus on the following three rhythm types, Normal Sinus Rhythm, Congestive Heart Failure, and Atrial Fibrillation when training our model. Upon review of our system we found that IoT devices are prone to noise especially in the neighborhood of 63Hz so a large portion of our time was spent researching and implementing various noise reduction techniques. Some of the more successful of which included Real Fast Fourier Transform, and a high pass filter suggested to us by a doctor who regularly uses hospital grade electrocardiogram machines. Over the course of our research we created a host of functions to help us analyze and process the data we were collecting from the sensor. We collected all of these methods and placed them into a python class that can be leveraged by others to help them further the research we have started. When compared to classical machine learning methods, we found that the deep learning approach we employed was more accurate when given the same data. In the future we aim to flesh out an intuitive user experience as well as continue to evaluate the system and model on more users to get a sense of the real world accuracy.**

***Keywords— electrocardiogram, deep learning, convolutional neural network, noise reduction, Internet of Things***

# **I. Introduction**

Heart health is often touted as a major area for improvement especially in the United States. The social effects of COVID-19 coupled with the poor diet and exercise habits perpetuated by modern life take tolls in many ways, but few operate as quietly as heart disease. Early action is vital to the survivability of some of these conditions, however many are not picked up in time due to the need for an Electrocardiogram (ECG) to properly diagnose the problem before symptoms begin manifesting. With the advent of reliable consumer grade ECG sensors and the advances in Internet of Things (IoT) computing power the door has been opened to build a device that can read, prepare, and classify a person’s ECG reading at home or anywhere you can sit still for 30 seconds. Such a device could alleviate the financial burden on the patient as well as decrease the need for routine ECG for those already diagnosed with chronic conditions, freeing up medical staff to focus on other matters.

To test the viability of a device like this, we trained a convolutional neural network (CNN) with tensorflow[11] and loaded it onto a Raspberry Pi mini computer paired with an ECG sensor to create an all-in-one system capable of collecting, cleaning, and classifying the heart rhythm of a user. All without ever needing to send the users data to a server or doctor for classification which will keep costs down and privacy intact.

# **II. Related Work**

Given the logistic, cost, and privacy concerns precluding us from collecting patient ECG data for ourselves, we turned to “PhysioBank” - a database of anonymized biomedical data put together and maintained by “PhysioNet” that has long been trusted and used by the research community. The team at PhysioNet also has written a set of software libraries coined “PhysioToolkit” which allows researchers to easily import raw and metadata. “PhysioBank, PhysioToolkit, and PhysioNet: Components of a New Research Resource for Complex Physiologic Signals,” Goldberger et. al. [1] touts Physionet and its services as being able to replace the need for each team looking to make headway in the field of biomedicine to collect their own data. This reduced barrier to entry allows people like us without the backing to collect such data to still attempt to apply our expertise to the field.

Others to look into Deep Learning for ECG classification include Avanzato et. al. [2] who were able to design a CNN model capable of classifying normal rhythm, premature atrial beat, and premature ventricular contraction with 98.35% accuracy. Chatterjee et. al. [3] propose a pipeline utilizing various noise reduction algorithms to be used on ECG recordings depending on the kind of noise you hope to remove. Chatterjee et. al’s work [3] is aimed at reducing noise in hospital recordings not recordings from the types of IoT devices used in this study.

In order to read the ECG of the end user, we used an AD8232 SparkFun single lead heart rate monitor [4] designed to be a cost-effective way to measure the heart's electrical activity. Since the AD8232 emits an analog signal, a MCP3008 8-channel 10-bit analog to digital converter (ADC) with serial peripheral interface (SPI) interface[5] was required to convert to the digital signal recognizable by the raspberry pi running the system.

Much of the progress made during this research comes off of the legwork done by Shrestha et. al. [6], who were generous enough to teach us how to extract data from the PhysioBank and prep it for use as training data. Our research essentially picks up where Shrestha et. al. [6] left off with ECG classification models. They were able to achieve roughly 85% accuracy when classifying an ECG reading into three categories using classical machine learning models such as Random Forest, Support Vector, and K-Nearest Neighbor.

# **III. Proposed Idea and Hypotheses**

In this project we have a pair of hypotheses, first of which is that deep learning can be used to classify normal sinus rhythm, congestive heart failure, and atrial fibrillation ECG signals more accurately than classical machine learning models. Our second hypothesis is that IoT devices such as a raspberry pi can be used to create a mobile system to collect and classify the rhythm entirely on device.

# **IV. System Design**

Our system for collecting a user's ECG signal starts with the AD8232 sensor. In order to get a readable signal in the Raspberry Pi a MCP3008 ADC is required to convert the analog output of the AD8232 to a digital one. The MCP3008 output is connected to the Raspberry Pi according to documentation by the creators of the chip. The digital output is read from the SPI pins into a text file, each datapoint getting its own line. All the code for driving the data collection process along with a handful of functions needed to prepare and analyze the data have been compiled into a single python class for ease of recreation.

Setup of the MCP3008 was done following guides put out by the manufacturer of the chip, Adafruit. As can be seen in Figure 1, and in context in Figure 2, pin 1 of the MCP3008 is being used as the input pin. Going from 9 to 16 the connections are as follows, pin 9 and pin 14 connect to ground (GND). Pin 10 connects to the CE0 pin or pin 24 (see Figure 3 for Raspberry Pi pin names and numbers) on the Raspberry Pi. Pin 11 connects to the MOSI pin or pin 19 of the Raspberry Pi. Pin 12 of the ADC connects to the MISO pin or pin 21 of the Raspberry Pi. Pin 13 of the ADC connects to the SCLK pin aka pin 23. Pin 15 and 16 of the ADC connect to 3.3v power.

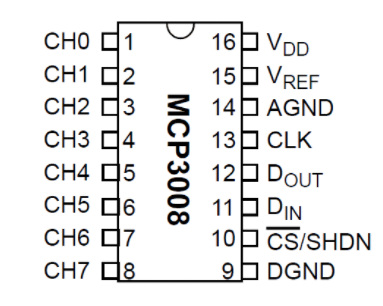


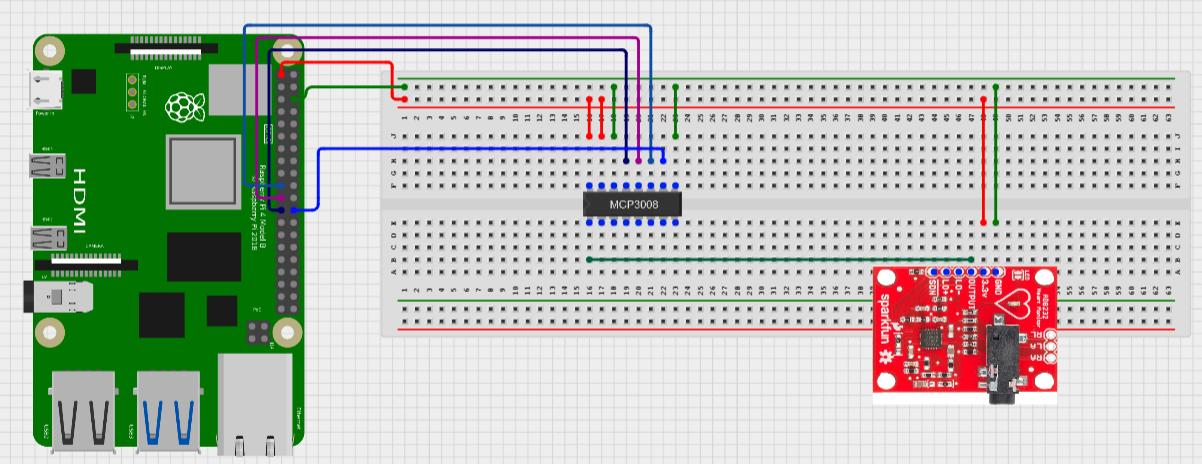
Figure 1. *Adafruits pin layout diagram for the MCP3008 analog to digital converter*

Figure 2. *Wiring diagram showing how each pin is connected in the context of the full system*

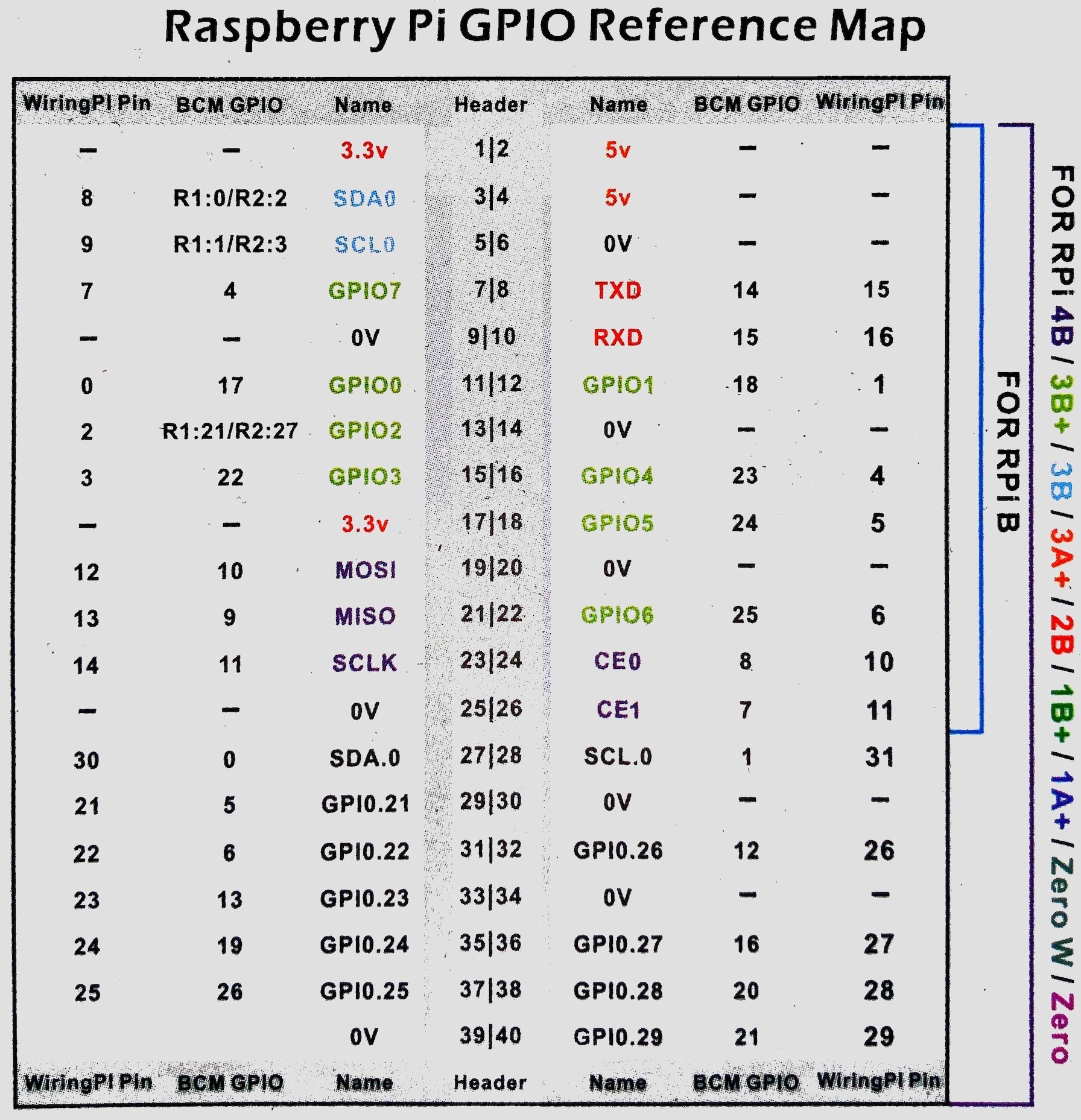
**

Figure 3. *Raspberry Pi GPIO pin reference map*

# **V. Implementation**

A. *Data Extraction and Preprocessing*

Inside of a Google Colab notebook we used data from PhysioNet’s database, PhysioBank, to train a Convolutional Neural Network (CNN) model that could classify an incoming ECG as either Normal Sinus Rhythm (NSR), Congestive Heart Failure (CHF) or Atrial Fibrillation (AF). Examples for each rhythm were extracted from PhysioBank using the PhysioToolkit library WFDB for python[7]. PhysioBank has a number of datasets pertaining to various heart rhythms, for our research we used MIT-BIH’s NSR and AF sets[1][8], and BIDMC’s CHF set[9].

In order to increase the number of datapoints we had available to feed to our model for training we segmented each of the entries from PhysioBank into chunks of 5000 values. Segmenting the data also normalized the length of the recordings being sent to the model for training since the raw data from physionet is collected over different time intervals, all of which are much longer than required for classification of the three types we hoped to classify. As discovered by K.H. Yoon et. al. [10] in their paper “Analysis of Statistical Methods for Automatic Detection of Congestive Heart Failure and Atrial Fibrillation with Short RR Interval Time Series”, records 04963 and 05091 of the MIT-BIH AF database were inaccurately annotated and records chf02 and chf06 of the BIDMC CHF set had high PAC or PVC frequency deeming them unusable in model training. As such we removed these records before training the model.

We determined that MIT-BIH NSR data was collected at a lower 128 samples per second which needed to be upscaled using linear interpolation to meet the 250 samples per second commonly cited as the standard for such classification and the rate used by the other two datasets.

Once all the data was segmented and properly scaled we took 2500 segments from each database and compiled them into a single array and randomized the ordering before splitting it up into a training set comprising 70% of the data, and a testing set with the other 30%. At this point the data still has a 5001st column holding a value 1-3 to denote which dataset each rhythm came from so we removed the last value from each row and added them to a new array. Since this was done after randomizing the data the values in these new labels arrays (one for testing and one for training) have the same indices as the data they were originally assigned to. We then took these integer classifications and one-hot encoded them.

B. *Model Training*

To train a model that could classify ECG signals into three categories we used Google's TensorFlow library[11] paired with Keras[12], an open source library that offers an interface to utilize neural networks in python. With these tools, we experimented with a large number of model configurations to find one that could get us the highest accuracy. The best performing of the models was structured as follows: Three one dimensional convolution layers each with 32 neurons and a relu activation, followed by a dropout layer to keep the model from overfitting. The next layer is a global average pooling layer which precedes two dense layers. The first of which has 64 neurons and a relu activation function. The second dense layer is the output layer, which has 3 neurons and a softmax activation function. The two dense layers are split by another dropout layer. See Figure 4 for a visualization of how the model is constructed. The model is compiled using the adam optimizer and a “categorical crossentropy” loss function. Finally we fit the model using Keras’s fit function which we had run 100 epochs to be sufficiently confident that the model was as accurate as it could be.

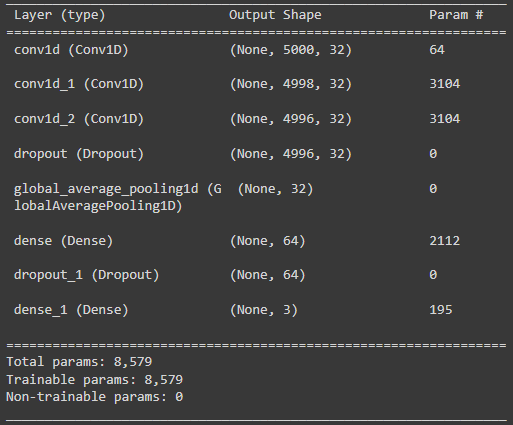


Figure 4. *Model summary including type of layers and number of parameters*

C. *Data Collection*

Recording a user's ECG signal is done by attaching the three leads of the ECG to the users chest as follows: red lead on the upper right of the chest, yellow lead on the upper left of the chest, and green lead on the lower left side of the rib cage (see Figure 5) making sure that each lead gets a new pad and is firmly and uniformly attached to bare skin. Once the user has attached all the leads and has plugged in the 3.5mm jack to the AD8232 it is very important to remain seated and still for the duration of the recording since the electrical activity of a moving muscle will interfere with the signal coming from the heart.

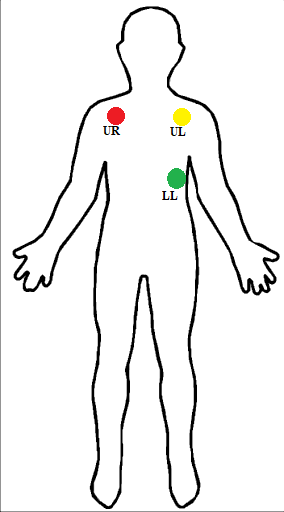
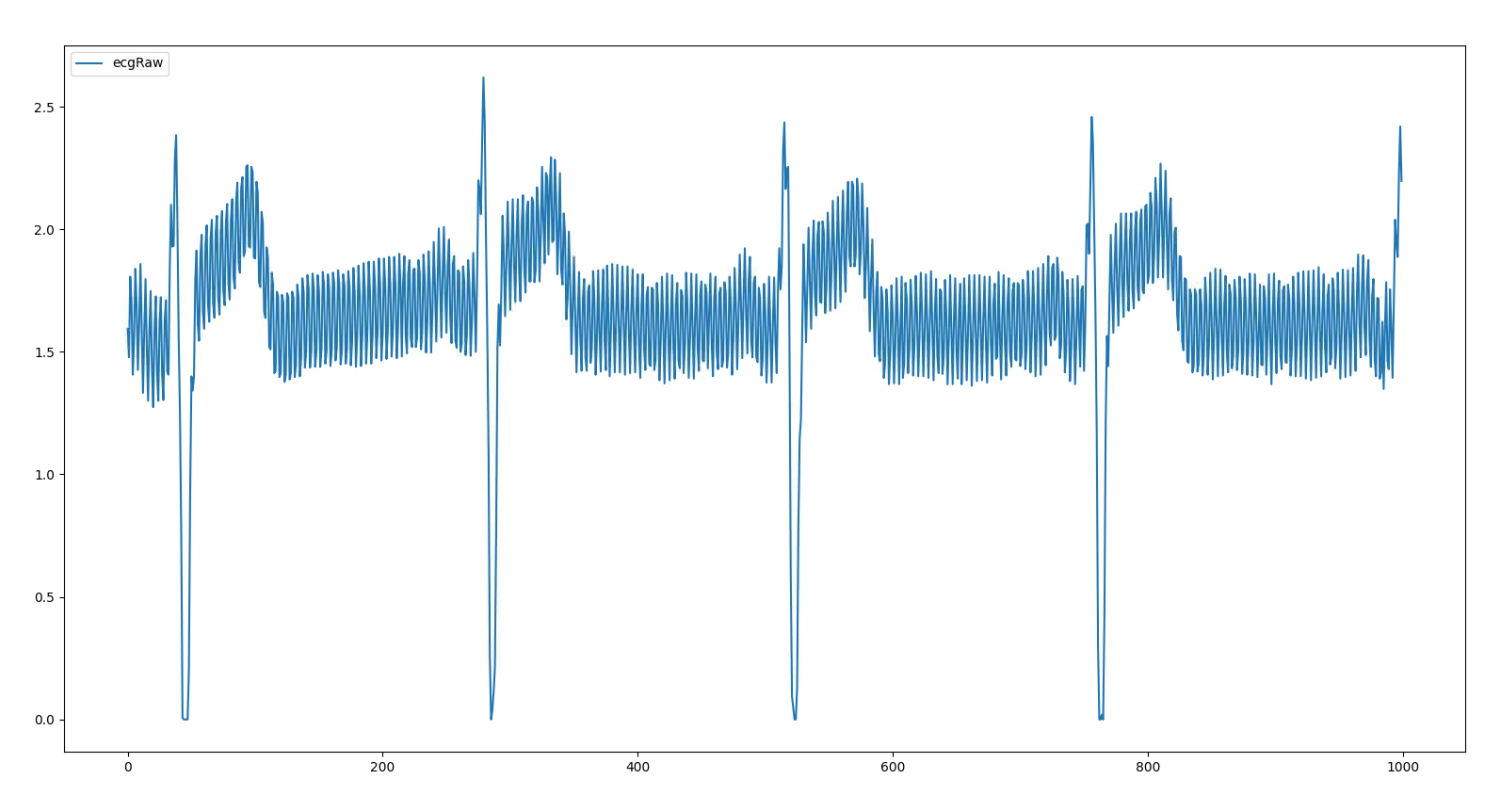


Figure 5. *Diagram for ECG lead placement where Red goes Upper Right (UR), Yellow goes Upper Left (UL) and Green goes Lower Left (LL)*

D. *Data Postprocessing and Denoising*

Before we can send a user's ECG recording to the model for a classification we must first do some post processing on the data. As with many Internet of Things (IoT) devices like the Raspberry Pi we are using to record user ECG data, noise is inevitable. As can be seen in Figure 6 the raw data coming out of the raspberry pi and AD8232 resembles an ECG but with significant noise. It is well documented that noise from the grid has a frequency of roughly 60Hz, and our tests indicate that our device has noise peaking in the 63Hz band. To address this issue we applied a Real Fast Fourier Transformation (RFFT) from the python library SciPy[13]. A RFFT is a variation on a classic Fast Fourier Transformation (FFT) which breaks a function into its constituent frequencies, see Figure 7.

Once we knew the hertz band of the noise was centered at 63Hz we were able to go into the RFFT and zero out the values in a window above and below 63Hz. After getting a RFFT with the noise at 63Hz removed, we inversed the RFFT to get the original data without the frequencies we determined as noise as constituent parts resulting in a clean signal. It is common for hospitals to employ a high pass filter to their data to remove noise in their ECG recordings so we applied such a filter to our data after the RFFT pass resulting in a reading that rests at 0, see Figure 8.



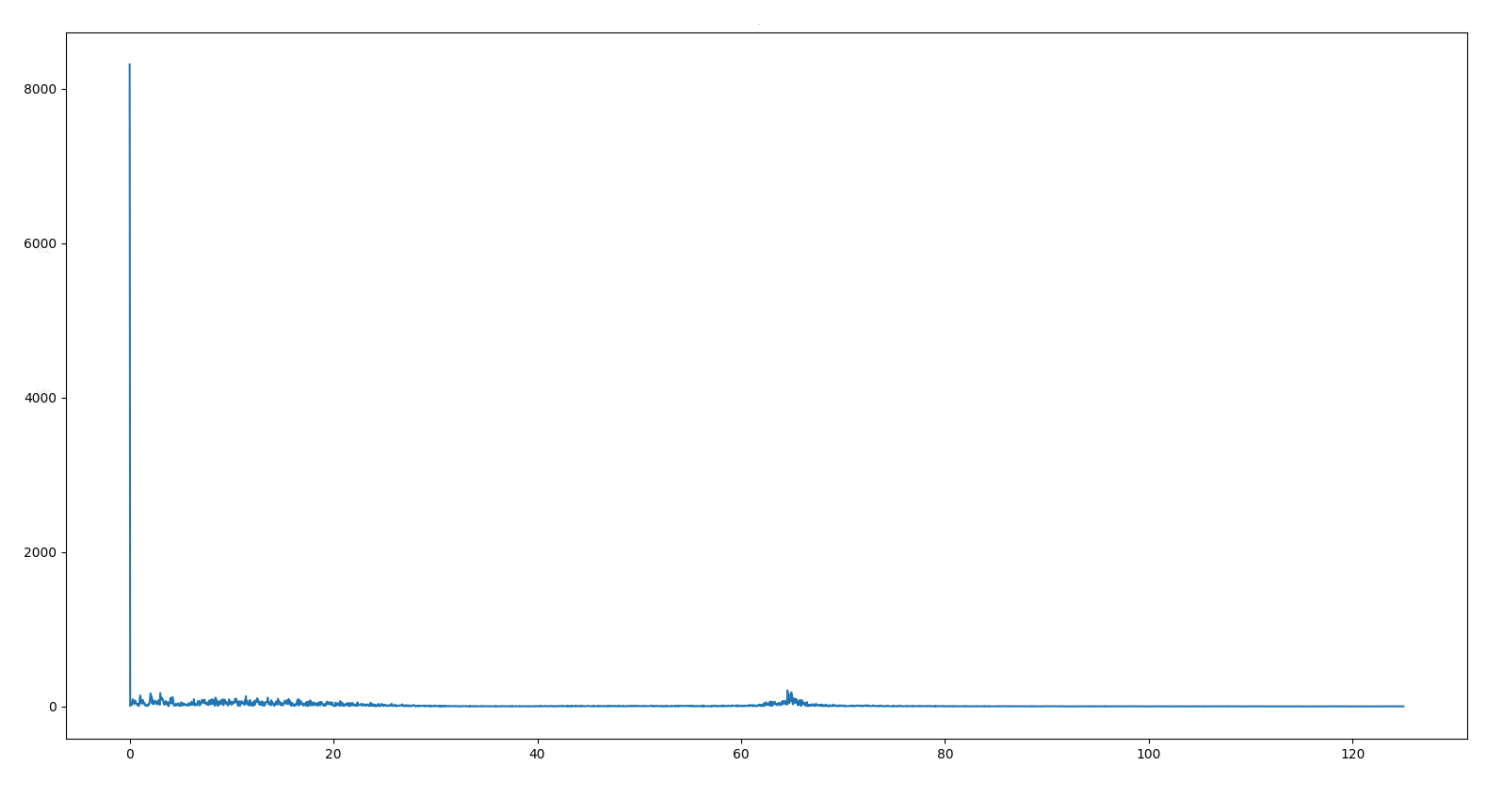
*Figure 6. 1000 raw data points from the AD8232 sensor.*

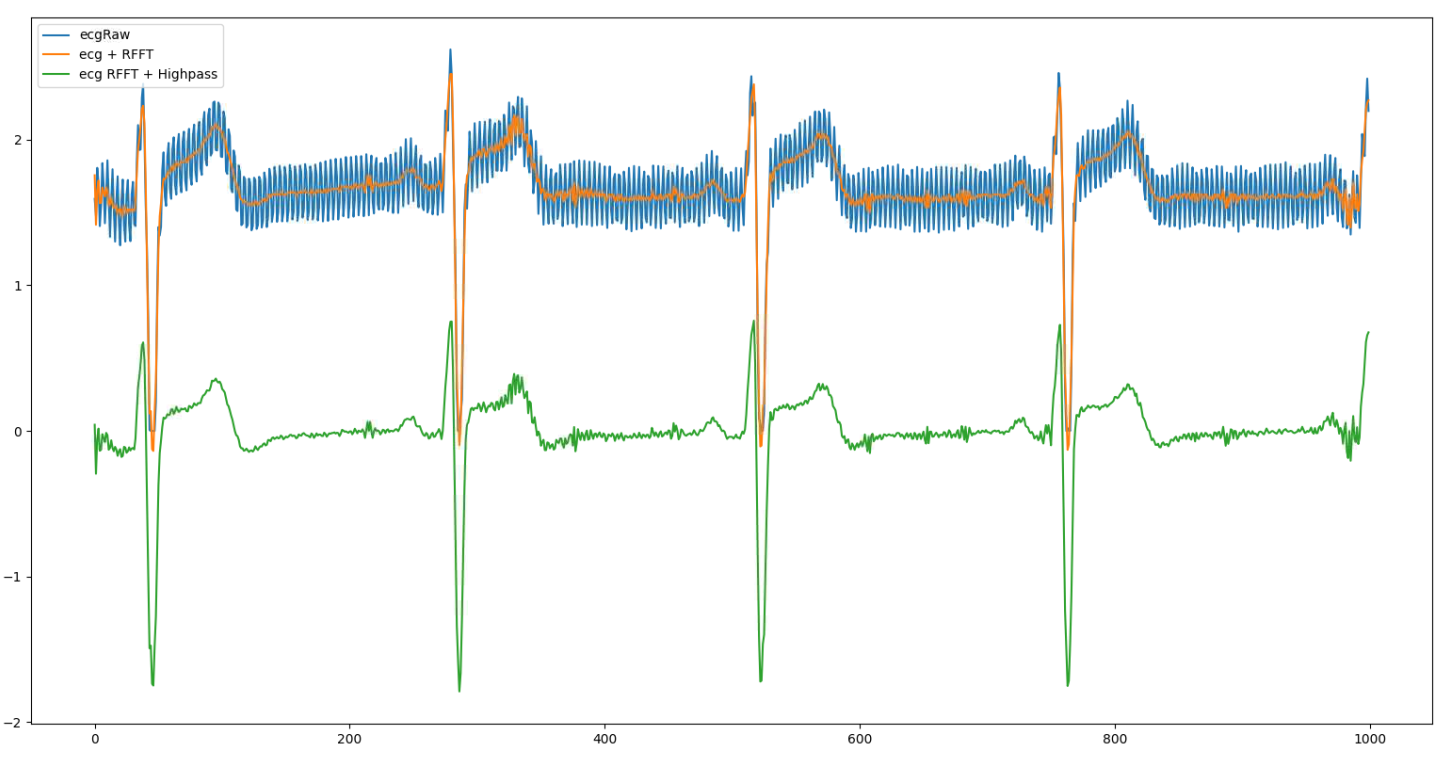
Figure 7. *RFFT frequency plot of the data seen in Figure 6.*

Figure 8. *In blue, the original raw data. In orange, the result of zeroing out a window around 63Hz using a RFFT. In green, the result of applying a high pass filter to the orange data.*

E. *User Interface*

Over the course of this research we created many python functions to aid in the processing of ECG recordings. We compiled all of our functions into a single python class called “ecgTools” that has a number of functions we wrote to analyze, process, and visualize ECG data.

In this class there are functions to pull data from PhysioBank or a local text file. A function that drives the AD8232 and records the data it outputs to a text file for easy access later on. The class also has multiple functions that perform the various pre- and post-processing techniques mentioned above. We added a predict function which allows a user to query our model for its classification, and functions that streamline the use of python plotting library matplotlib[14] so users can see a plot over time of the ECG recordings they are working with.

# **VI. Evaluation**

At 250 samples per second the 5000 samples required for each segment can be collected in just 20 seconds. Since the model is trained off of segments with 5000 data points, we need to collect 5000 from the user as well because the model needs the same sized array as what it was trained off of to make a prediction.

The CNN described above was our best performing version and scored a 96.93% accuracy when asked to classify the unseen ECG signals in the testing array. One important aspect of determining if a model is being overfit to the training data is to evaluate the loss. If the loss of the training data is lower than the testing or validation data loss then the model is likely overfit. See Figure 9 for the graph of each loss value over the 100 epochs. As can be seen, the loss and accuracy graphs both remain close together and do not cross indicating that this model is not overfit and the accuracy number of 96.93% can be accepted as a true indicator of the performance of the model.

When looking for equivalent studies and models to compare this model to, it is best to look at Shrestha et. al[4] since their work fed directly into ours and the same approaches were taken to segment the data. Shrestha et. al[4] used classical machine learning methods to create a model that could classify ECG recordings into the same three categories, and were able to achieve an accuracy of approximately 85%. We believe based on this comparison that deep learning and neural networks are a viable way to create models for classifying ECG data.

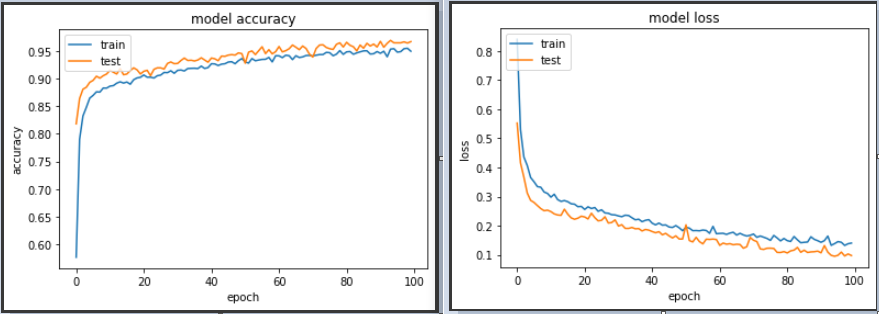
It must be noted that when this model is applied to data taken from the Raspberry Pi we are getting misclassifications, however currently the recordings are from an extremely small sample size because of the logistical difficulty of getting the system out to enough people. As it stands we are unsure if the model is overfit to the ideal environment of a hospital ECG where the training data came from or if there is an issue with the way the AD8232 is collecting the heart rate data, but we know that at least one user who has a verified NSR is consistently classified as AF. This inaccuracy also hinders our ability to analyze the effectiveness of our noise reduction methods. This topic is discussed more in future sections.

Figure 9. *Accuracy (Left) and Loss (Right) over 100 epochs.*

# **VII. Discussion**

Based on the testing data we set aside to determine the accuracy of our model it seems to be a very good classifier of ECG signals. However, as with all machine learning models the more data you can provide it the better, which was the reasoning behind segmenting the data during preprocessing. After getting the model onto the Raspberry Pi and feeding it the raw data from the AD8232 sensor we got a classification of Atrial Fibrillation which we believed to be inaccurate. The raw data coming out of the sensor is very noisy especially in the 60Hz range as discussed above so we were under the impression that the noise was causing the misclassification which started us on a path to find the best ways to denoise data, and in particular ECG data. Our first attempt to reduce noise came in the form of a simple moving average (SMA) and exponential moving average (EMA), two methods used primarily by stock analysts to determine trends. Figure 10 shows the raw data along with the SMA filtered data superimposed, and you can see the reduction in the noise is substantial, however our only real measure as to how well the noise is being reduced is by querying the model with the assumption that the user, in this case us, has a normal sinus rhythm. As it turned out, the model almost always sticks to its Atrial Fibrillation classification even when the noise is reduced using the industry standard FFT and highpass filter methods.

In order to really evaluate the accuracy of the system as a whole we would need to gather lots of users with the different rhythms we trained the model on, and record a proper hospital ECG at the same time to get a true signal to compare our AD8232 signal against. Since that is out of the scope of our study, we turned to experts for advice. Dr Andrew Ahn of PhysioQ[15] was kind enough to classify some of our signals and was quick to determine that the recordings were adequate enough after noise reduction to classify them one way or another. In the case of the recording we had been using to test our noise reduction attempts he is confident that the rhythm is NSR.

This concrete classification of the data gave us confidence in our noise reduction, but brought the integrity of the model into question. The model could be overfit to the clean and ideal conditions used to record the training data, when comparing our certified NSR data against an example from MIT-BIH’s NSR database (see Figure 11) the two do have different characteristics. That being said, Dr Ahn[15] explained to us that the real things he looks for when classifying NSR versus other rhythms is the P wave, T wave, QRS intensity, and RR distance (see Figure 12 for a labeled PQRST graphic). Since machine learning, especially deep learning, is such a black box when it comes to how it determines what to use as classification markers, we cannot determine with any certainty whether the model uses similar techniques to determine if a segment is NSR, AF or CHF.

One important question is do we want it to use the same methods as doctors? A major advantage of leveraging machine learning is to attempt to find nuance in the data that may go unnoticed by humans. If we hope to outperform human doctors then we would need to gather more pertinent data like the model will see in the field. Since we are just looking to provide an early warning to people in a cheap and affordable manor by means of alerting the user and doctor when a non-normal rhythm is detected from which the doctor can take next steps to confirm the rhythm on more capable machinery, we don’t feel the need to train a perfect model as we believe our time is better spent reducing noise and creating a cohesive user experience.

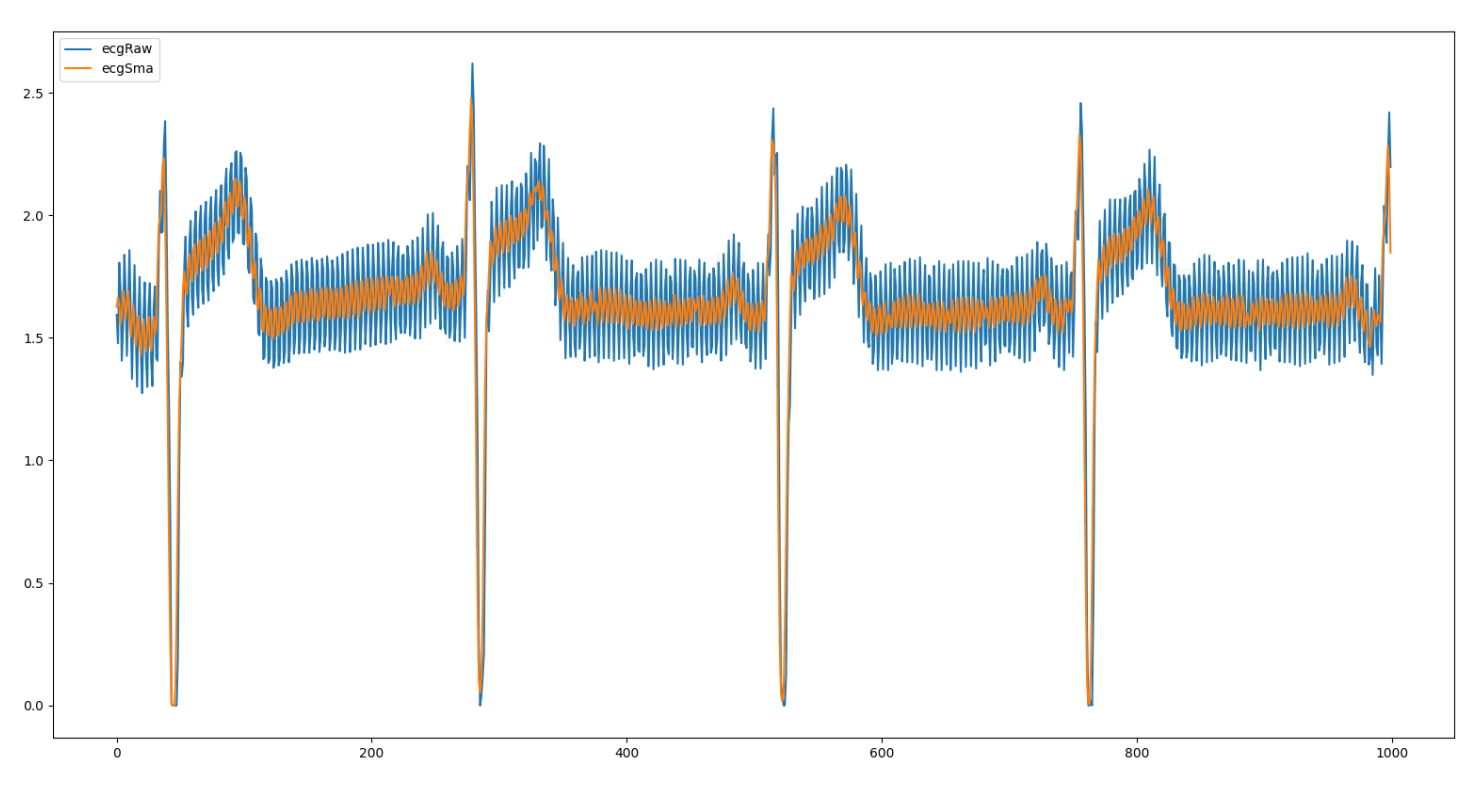


Figure 10. *In blue, raw ECG data, and in orange a SMA filter with a window size of 3.*

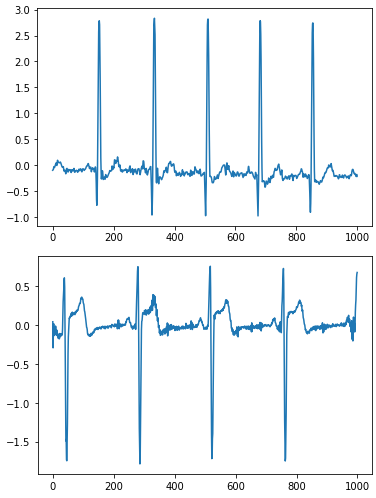
**

Figure 11. *Top depicts a resampled MIT-BIH NSR sample while bottom shows one of our noise reduced samples over the same time period*

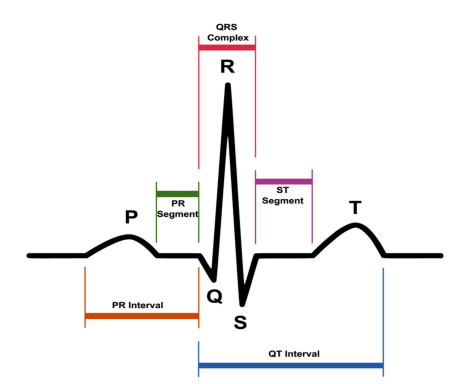


Figure 12. *Typical ECG wave with each characteristic labeled.*

# **VIII. Conclusions and Future Work**

In our study and implementation of deep learning to the ECG classification problem, we believe that it is a better method than classical machine learning. This conclusion comes by way of comparison to the work done by Shrestha et. al.[6] who used the same data from PhysioBank and came up with the data segmentation method we employed in this study. Shrestha et. al.[6] were able to get roughly 85% accuracy when using 5000 point segments compared to our roughly 97% accuracy. It must be stated that we do not know how well either model performs on a large sample of “real world” data, but as far as we can compare the two methods, deep learning scores better on the same data than the classical approach does. The secondary goal of this research was to determine if an IoT device such as a raspberry pi could be used as a foundation for a system that can read in, preprocess, classify, and display an ECG rhythm. With the advancements in chips and sensors the hardware is more than capable of powering nearly all of the steps of the process. We believe that the capabilities of the hardware, especially the raspberry pi, are sufficient to run an entirely mobile ECG classification program given that the model used to classify the rhythm is pretrained on more powerful hardware and loaded onto the raspberry pi.

That being said, it is unclear whether the deep learning approach applied in this study is viable in the real world so we hope to continue this work to determine that answer. We also hope to try other data collection methods suggested to us by Dr. Ahn[15] that may record a more hospital-like signal. In future work we would like to examine more features we could add to the dataset to hopefully “weight” characteristics like RR distance more heavily in the classification. Chatterjee et. al.[3] purpose various methods for denoising ECG recorded in a hospital but we did not have the time to evaluate their findings when applied to IoT collected data. Chatterjee et. al.[3] also describe some processes for evaluating the performance of ECG noise reduction algorithms which we think could be key in determining if the noise reduction approaches we took are viable. Another possible way of doing this may be to take inspiration from Hagerman and Olofssonn and their phase inversion technique [16] used to quantify how well hearing aid noise reduction algorithms perform. In terms of user experience, we intend to wrap the work we have done into a program or application that can be loaded onto a phone and communicate with the Raspberry Pi to give the user easy to use controls for reading their ECG and getting a response from the model.

# **IX. Acknowledgement**

##### **References**

1. Goldberger, A., Amaral, L., Glass, L., Hausdorff, J., Ivanov, P. C., Mark, R., ... & Stanley, H. E. (2000). PhysioBank, PhysioToolkit, and PhysioNet: Components of a new research resource for complex physiologic signals. Circulation [Online]. 101 (23), pp. e215–e220.
2. Avanzato R, Beritelli F. Automatic ECG Diagnosis Using Convolutional Neural Network. *Electronics*. 2020; 9(6):951. <https://doi.org/10.3390/electronics9060951>
3. Chatterjee, S., Thakur, R.S., Yadav, R.N., Gupta, L. and Raghuvanshi, D.K. (2020), Review of noise removal techniques in ECG signals. IET Signal Process., 14: 569-590. <https://doi.org/10.1049/iet-spr.2020.0104>
4. AD8232, S., (2021). SparkFun Single Lead Heart Rate Monitor - AD8232 - SEN-12650 - SparkFun Electronics. [online] Sparkfun.com. Available at: <<https://www.sparkfun.com/products/12650>>
5. MCP3008, A., (2021). Adafdruit 8 channel 10-bit ADC with SPI interface - MCP3008 - Adafruit. [online] Adafruit.com. Available at: <<https://www.adafruit.com/product/856>>
6. Shrestha, A., Yu, J., (2021). Remote Heart Monitoring - ECG data analysis using IoT Devices and Machine Learning.
7. GitHub. (2021). MIT-LCP/wfdb-python. [online] Available at: <https://github.com/MIT-LCP/wfdb-python>
8. Moody GB, Mark RG. A new method for detecting atrial fibrillation using R-R intervals. Computers in Cardiology. 10:227-230 (1983).
9. Baim DS, Colucci WS, Monrad ES, Smith HS, Wright RF, Lanoue A, Gauthier DF, Ransil BJ, Grossman W, Braunwald E. Survival of patients with severe congestive heart failure treated with oral milrinone. J American College of Cardiology 1986 Mar; 7(3):661-670.
10. K. H. Yoon *et al*., "Analysis of Statistical Methods for Automatic Detection of Congestive Heart Failure and Atrial Fibrillation with Short RR Interval Time Series," *2015 9th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, 2015, pp. 452-457, doi: 10.1109/IMIS.2015.88.
11. Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Rafal Jozefowicz, Yangqing Jia, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Mike Schuster, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.
12. Chollet, F., & others. (2015). Keras. GitHub. Retrieved from https://github.com/fchollet/keras
13. Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., … SciPy 1.0 Contributors. (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, *17*, 261–272. <https://doi.org/10.1038/s41592-019-0686-2>
14. [J. D. Hunter, "Matplotlib: A 2D Graphics Environment", Computing in Science & Engineering, vol. 9, no. 3, pp. 90-95, 2007](https://doi.org/10.1109/MCSE.2007.55).
15. Ahn, A. (2021, December 1) Personal interview [Personal interview]
16. Hagerman, Björn & Olofsson, Åke. (2004). A Method to Measure the Effect of Noise Reduction Algorithms Using Simultaneous Speech and Noise. Acta Acustica united with Acustica. 90. 356-361.