# ECG Classification Using Two Class SVM and DNN

Cory Davis

Sayli Aphale

EE 5423 – Hardware Architecture for Machine Learning

Dr. Eugene John Ph.D.

## 1. Introduction

Electrocardiograms (ECGs) are tests that record heartbeats. It is an extremely useful test for determining health of a patient's heart. The recording displays the electrical pulses of the heart, and with the help of a cardiologist, can diagnose potential abnormalities. Diagnosing problems leads to proper treatment for the patient; these could include medication, lifestyle changes, or pacemaker implantation.

The process of manually reading and identifying abnormalities in ECG recordings can be a lengthy and tedious task. Applying machine learning techniques to ECG data is presently a vast field of research. The use of such techniques allows for significantly quicker identification of potential abnormalities and diseases. This research uses the MIT-BIH Arrhythmia dataset and trains an SVM model and a CNN to identify normal and abnormal heartbeats (Shown in Figure 1 a and b).

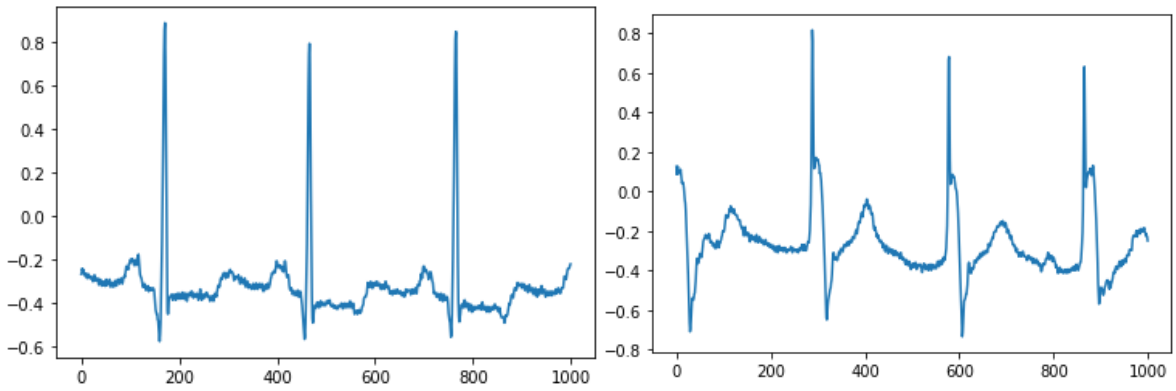


Figure 1. a) Normal Heartbeat b) Abnormal Heartbeat

### 1.1 MIT-BIH Arrhythmia Dataset

One of the most tested and accessible ECG data sets. The MIT-BIH Arrhythmia Dataset was recorded between 1975 and 1979 and distributed since 1980, being used as the standard test material for arrhythmia detectors. The database contains 48 recordings at 30 minutes per record, taken from 47 test subjects. The analog recordings were later digitized at 360 Hz per channel with 11-bit resolution over a 10-mV range. The dataset provides annotations at each beat with the index number and the pulse type [3].

## 1.2 Preprocessing

Prior to processing the MIT-BIH data in the models, the data was first preprocessed. Most ECG data has significant amounts of low-frequency variation (baseline wander or drift). To remove this, the data is passed through a filter to level out the signal. Reducing this wander helps in wave inspection and more accurate processing. Before and after shown in Figure 2 and 3.

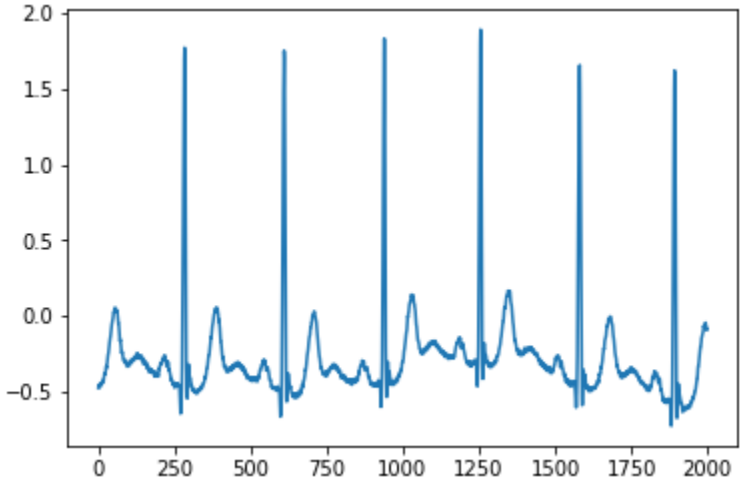


Figure 2. Raw ECG Waveform

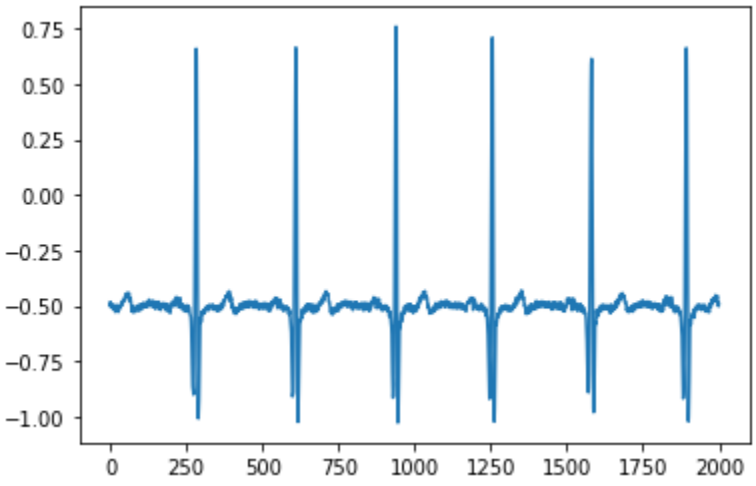


Figure 3. ECG Waveform After Filtering

After the baseline wander is removed, each heartbeat needs to be isolated for feeding into the model. Beats are isolated by finding the midpoints between each peak. The beats were then resampled into arrays of length 200. This data was than processed using the Fast Fourier Transform (FFT). As heartbeats are time dependent FFT was useful in analyzing the data. This filtered data was then delivered to the described SVM and NN models.

## 2. Background

### 2.1 Support Vector Machines

Machine Learning has two general types of learning models, based upon the labeling of the input data. Unsupervised learning looks at unlabeled data in order to determine patterns in a data set. Supervised learning infers patterns between data based on the labeling of the training data.

Support Vector Machines (SVM) are a supervised learning method used in machine learning. The primary purpose of SVM is data classification. SVM classification can mark data using as many classes as desired, however with this research only two-class classification is required. The goal of SVM is to determine a hyperplane that correctly divides the training data. In 2-D the hyperplane is a linear function, seen in Figure 4.

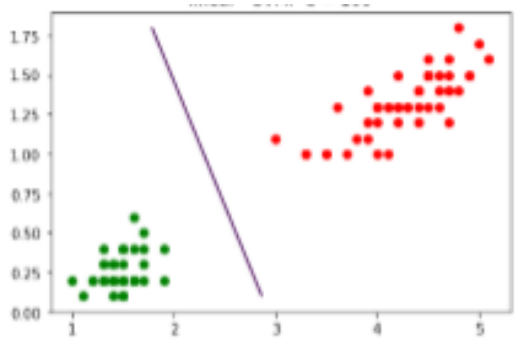


Figure 4. Chart Displaying Hyperplane to Separate Classes [6]

In testing this demarcation line, combined with the input data, will determine the predicted classification. This clear separation is called “Hard SVM.” However, there are many cases where the data is not perfectly separable (Figure 5). In this case an approximately correct hyperplane is obtained. This line correctly separates most of the data, except for a few cases.

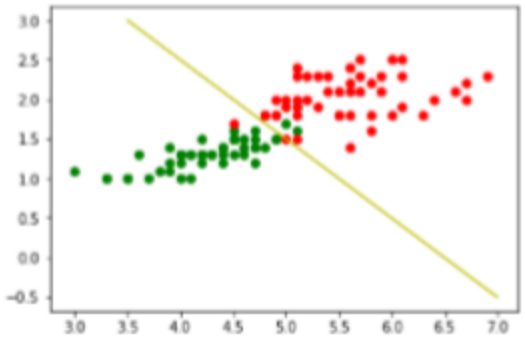

Figure 5. Non-perfectly Separable Data [6]

In cases where there is not a clear and distinct separation, “Soft SVM” is used.

$$\min_{\beta,c,n} \beta^T\beta + C\sum_{i=1}^{s} n_i \qquad \text{(Eq. 1)}$$

Subject to $y_i(\beta^T x_i + c) \geq 1 - n_i$ for i = 1,2,…,s

And $n_i \geq 0$

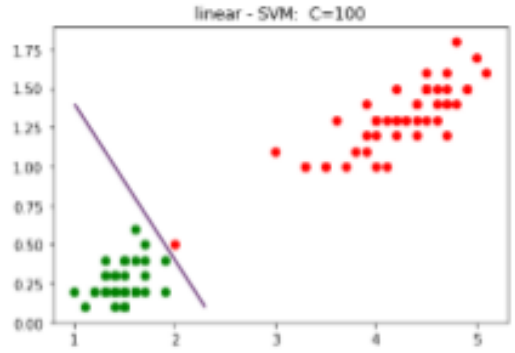


Figure 6. Hard SVM [6]

In cases of a large C, the line strictly separates the clusters (Figure 6), whereas a smaller C provides a more appropriate and stable fit (Figure 7). These examples only have two datapoints, so they are easily visualized. However, the dataset used in this research has 200 datapoints. The value of C must be optimized for high accuracy.

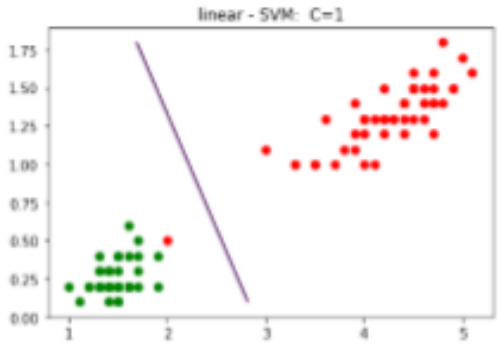


Figure 7. Soft SVM [6]

## 2.2 Neural Networks

Neural Networks are a set of algorithms, modelled loosely after the human brain, that are designed to recognize patterns. They interpret sensory data through a kind of machine perception, labelling or clustering raw input. The patterns they recognize are numerical, contained in vectors, into which all real-world data, be it images, sound, text or time series, must be translated.[8]

A Convolution Neural Network is (CNN) is constructed by stacking Multiple computation layers, through the computations of each layer a higher layer of abstraction of input data which can be named as feature map is extracted, to preserve essential and unique information [7]. The performance of a CNN primarily depends on the dept of the layers i.e. the type and number of layers used to design the CNN.

There are several types of Neural Networks some of which are: Fully connected NN, Convolution NN, Recurrent NN, Long-Short-Term Memory etc. Some of the well-known CNNs are AlexNet, VGG-16, ResNet50, Xception etc.

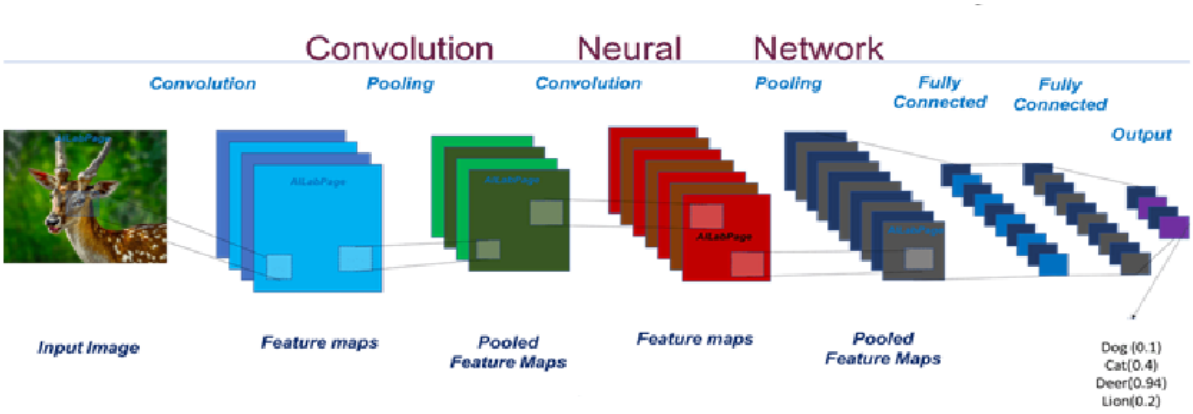


Figure 8. Convolution Neural Network

In CNN, there are three types of layers: the input layer, hidden layer and the output layer. The hidden layer carries out feature extraction by performing certain calculations and manipulations. There are multiple hidden layers like Convolution layer, Activation function layer, Pooling layer, etc. that perform the feature extraction. The Convolution layer uses a Matric filter and performs convolution operation to detect patterns in image. Activation functions like ReLu, Sigmoid are applied to the Convolution Layer to get a rectified feature map of the image. Pooling layer uses different filters to identify different parts of the image like edges, corners, body, feathers, eyes, beak, etc. It is also down sampling operation that reduces dimensionality of the feature maps.

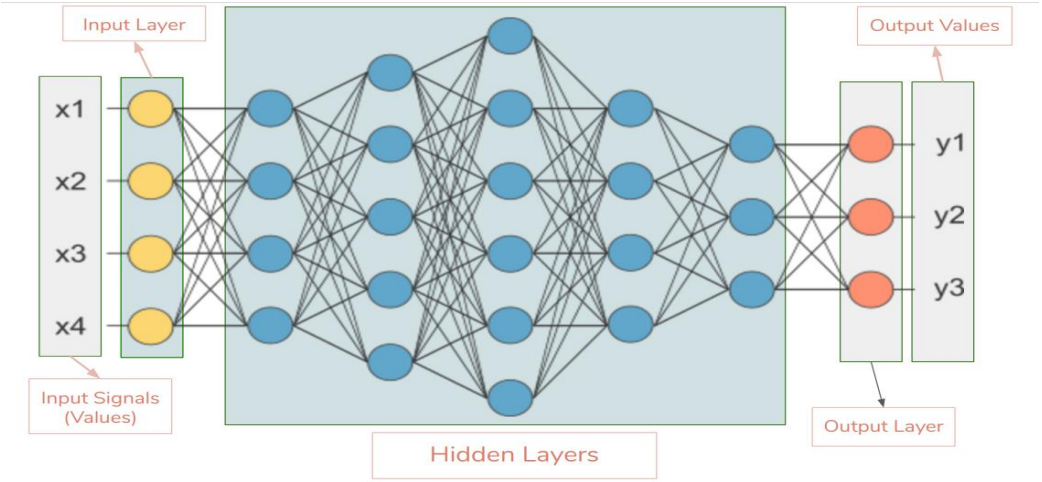


Figure 9. Layers in CNN

CNN also features other layers for the data processing like a flattening layer, flattening is the process of converting all the resultant 2-D or 3-D arrays from pooled feature map into single long continuous linear vector. The Flattened matrix from the pooling layer is fed as input to the fully connected layer to classify the image.

The CNN design has methodology in the way the input data is feed into the network for processing along with the dimensions of the input feature map as well as the dimensions of filters used for feature Etraction, pooling, the dense network. The CNN decoder ring can be seen in Figure 10.

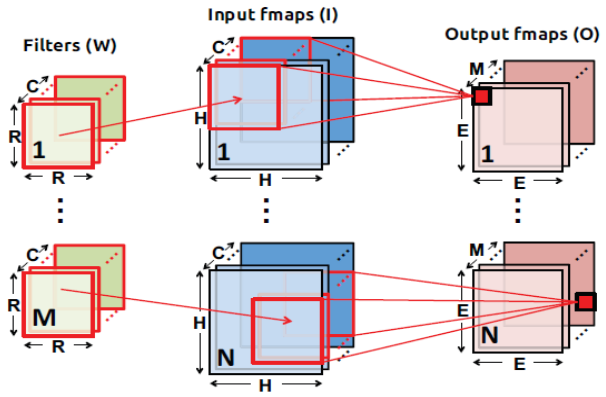


Figure 10. Feature Maps and Filter Dimensions [7]

Certain salient parameters of CNN decoder ring are as follows:

| | |
|---|---|
| N | Number of input fmaps/output fmaps (batch size) |
| C | Number of 2-D input fmaps/filters (channels) |
| H | Height of input fmap (activations) |
| W | Width of input fmap (activations) |
| R | Height of 2-D filter (weights) |
| S | Width of 2-D filter (weights) |
| M | Number of 2-D output fmaps (channels) |
| E | Height of output fmap (activations) |
| F | Width of output fmap (activations) |
| U | Stride of convolution |

Table 1. CNN Decoder Ring Parameters

Today CNN are designed to be very dense in-order to obtain high performance and accuracy. The time required to train such CNNs is considerably large with increase in the number of layers along with the particularity of dataset used, but the most important aspect of CNN is the response time i.e. the inference time when testing data is used. Training time can be as sizable as possible, but the inference time should be minimal as most of the CNN applications are user response based. CNN have many applications in varied fields such as medical, finance, infrastructure, weather forecasting, even detection etc.

## 3. Design of Implemented Sequential Model

The NN model designed and implemented for detecting Arrythmia [7] is a Keras Sequential model. Two sets of training and testing data are run on the sequential model to predict the accuracy, performance and time required to perform the task.

The model is created and run on CPU and GPU node of Shamu and the timing and accuracy are noted. The MIT-BIH dataset is too heavy to be run on a personal device. The data is 2-D in shape where the first dimension is the length of the samples used for training/testing and other dimension is the number of columns. There are in total 201 columns out of which 200 are the data samples columns and the last one is the label which is separated to be separately feed in the model.

A Fast Fourier Transform is performed on data in-order to do the feature extraction of the data by taking only the real part of FFT. The data is then converted into NumPy array format, normalized and then given as an input to the CNN after reshaping it into 3-D.

The CNN Model is 16 layers deep, having 6 layers of Convolution 1-D which is the 1$^{st}$ layer in the model (the parsed is already flattened so no need to flatten the data), 2 layers of MaxPooling 1-D and one layer of GlobalMaxPooling 1-D (it is used for down-sampling and detect the edges, corners, etc.), the activation function used is ReLu for model (used for rectification of the feature maps) [7]. The 3-Dense layers (fully connected) are used for classification of data, the output layer has two neurons representing the classification into abnormal and normal heartbeat. Before inputting the data into the Dense layer, it is Flatten to form a continuous liner vector for the Dense layer [8]. The Figure depicts the 2-D array before normalization of data. The

Model is then complied and fitted; the Convolution 1-D layer input is 3-D so there is need for reshaping the data into 3-D array.

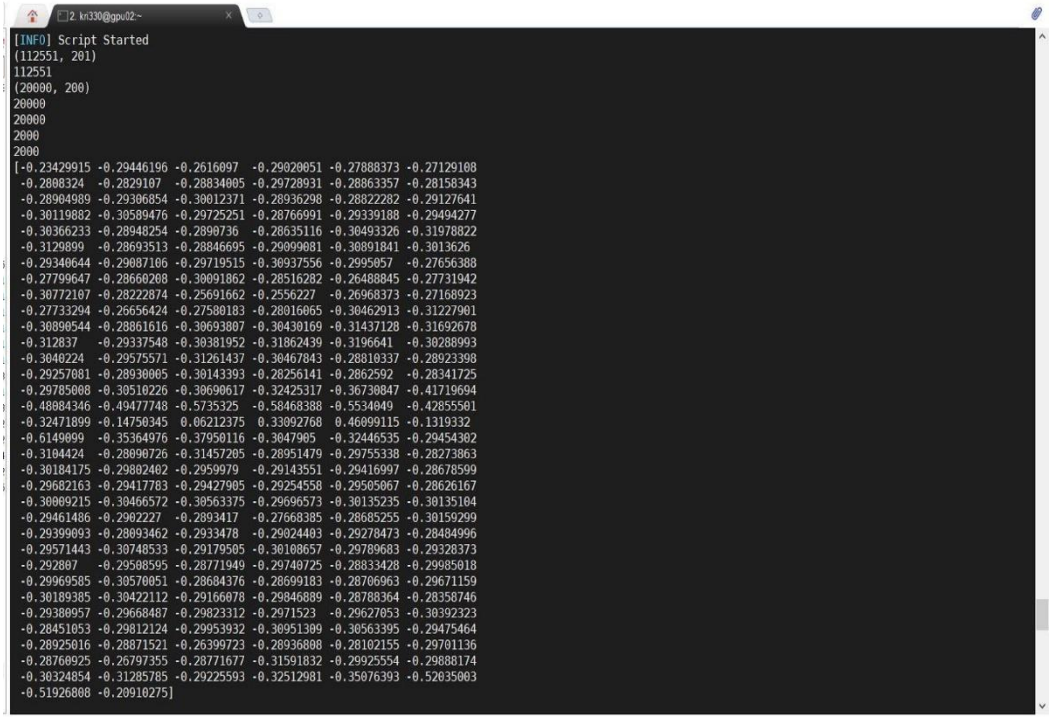


Figure 11. Size, Shape and 2-D representation of Data Matrix

## 4. Implementation and Results

To test the NN and SVM models, a high-performance CPU and GPU processing node was used. The goal was to test for variations between timing and accuracy of CPU versus GPU, as well as the efficiencies of using SVM versus NN. For SVM the following C values were used: 0.0001, 0.001, 0.01, 0.1. 1.0, and 2.0. Due to processing time constraints, the SVM training size was limited to 40,000 heartbeats.

For the NN, two dataset sizes were tested. Set 1: 20,000 beats for training and 2,000 beats for testing. Set 2: 65,000 beats for training and 10,000 beats for testing. The NN model is run on CPU and GPU node, for a range of epochs 10-50 with an increment of 5 epochs.

4.1 SVM Results

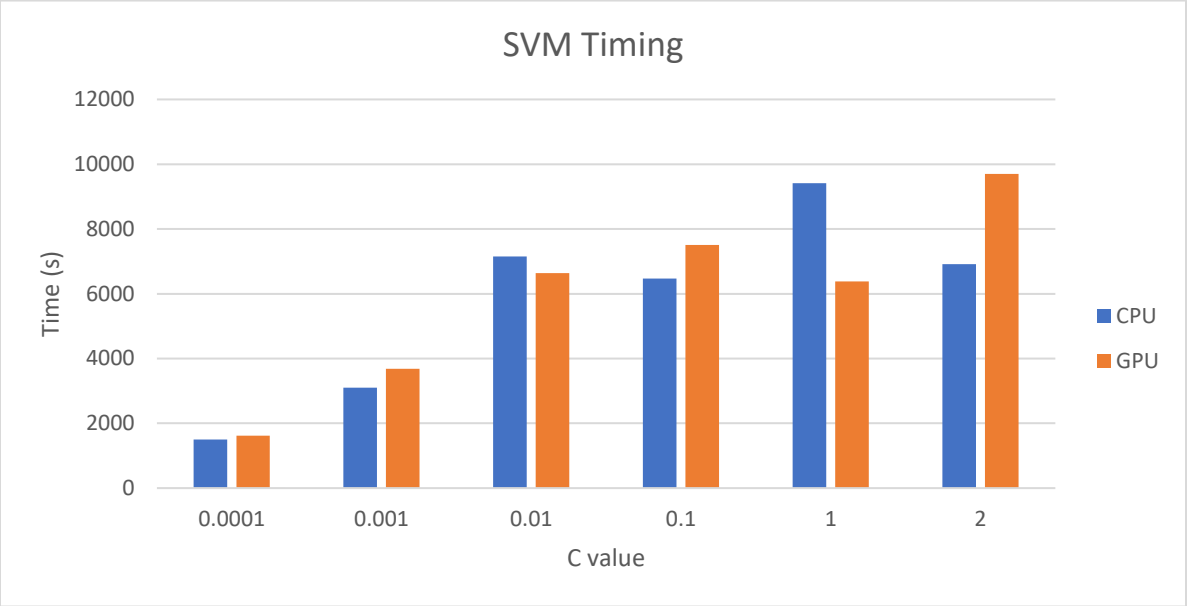


Figure 12. SVM Timing Results

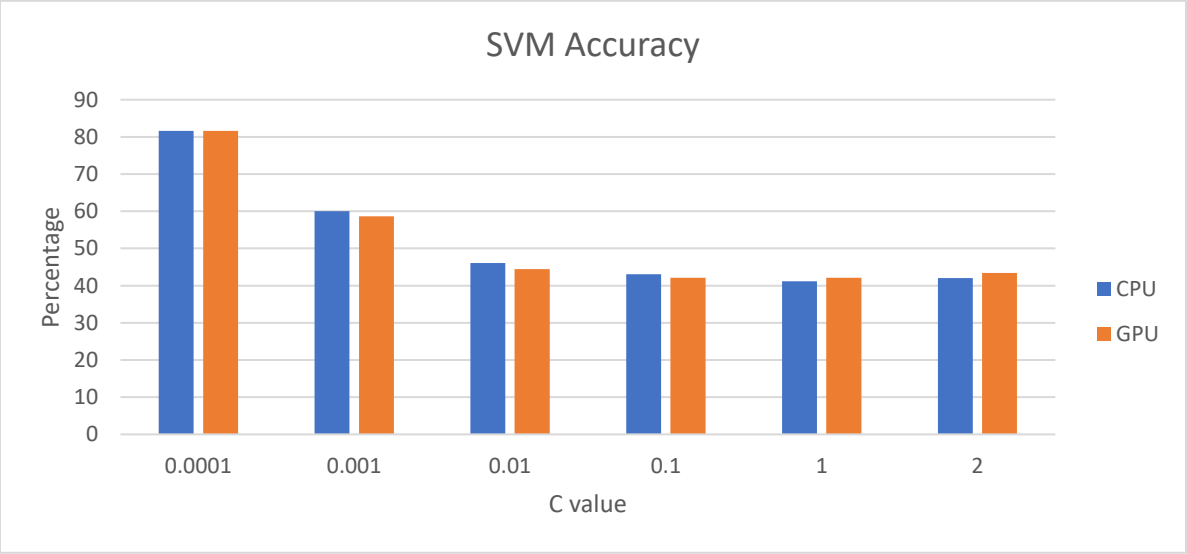


Figure 13. SVM Accuracy Results

For SVM the node used made very little difference in time to process or the accuracy. However, the GPU took 2.7% longer than the CPU to process the data. Additionally, the accuracy difference was negligible as the there was only a 0.5% difference slightly in favor of the CPU node.

In these tests, the extreme low C value, lending towards a “softer” SVM, yielded the highest level of accuracy at 81.6%. As the SVM was “hardened” the accuracy greatly dropped off before flattening out at about 41%. Normal heartbeats between patients can fluctuate greatly and the softer SVM allows for greater fluctuation while maintaining a significantly higher accuracy.

## 4.2 NN Results

### 4.2.1 CPU Node

The Figures 13, 14, 17, and 18 represent the results obtained on the CPU node for the two sets of dataset bifurcation. The charts constitute number of epochs on x-axis, the accuracy on y-axis of the testing data obtained for the model, time on y-axis required to create and process the model. The first set of number of training data and testing data are 20,000 and 2,000, respectively.

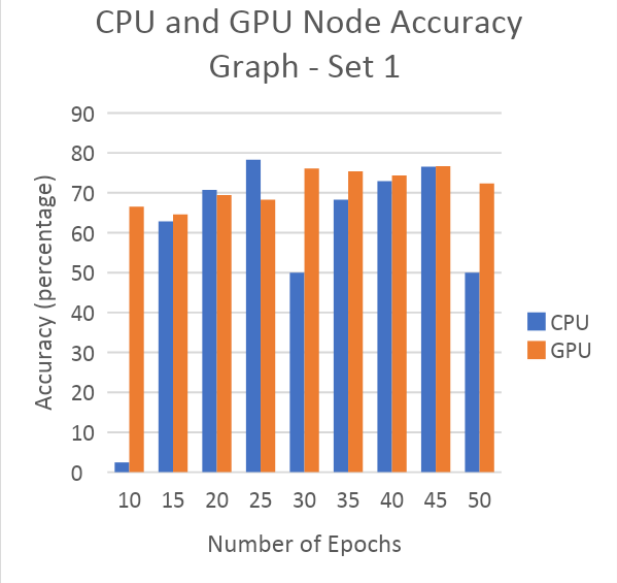


Figure 14

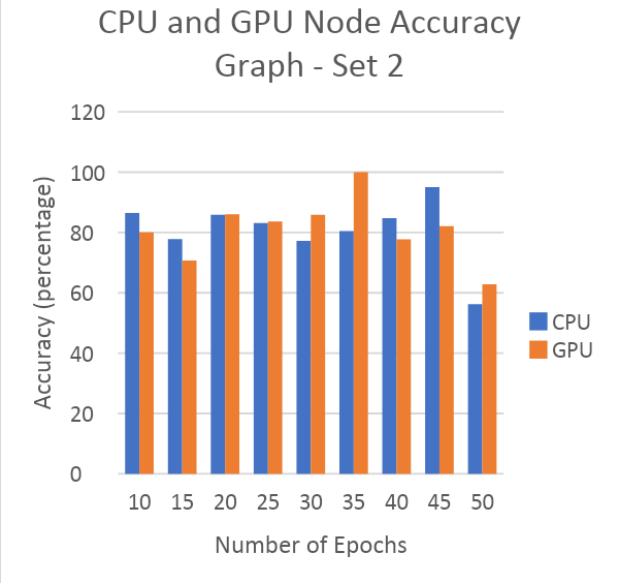


Figure 15

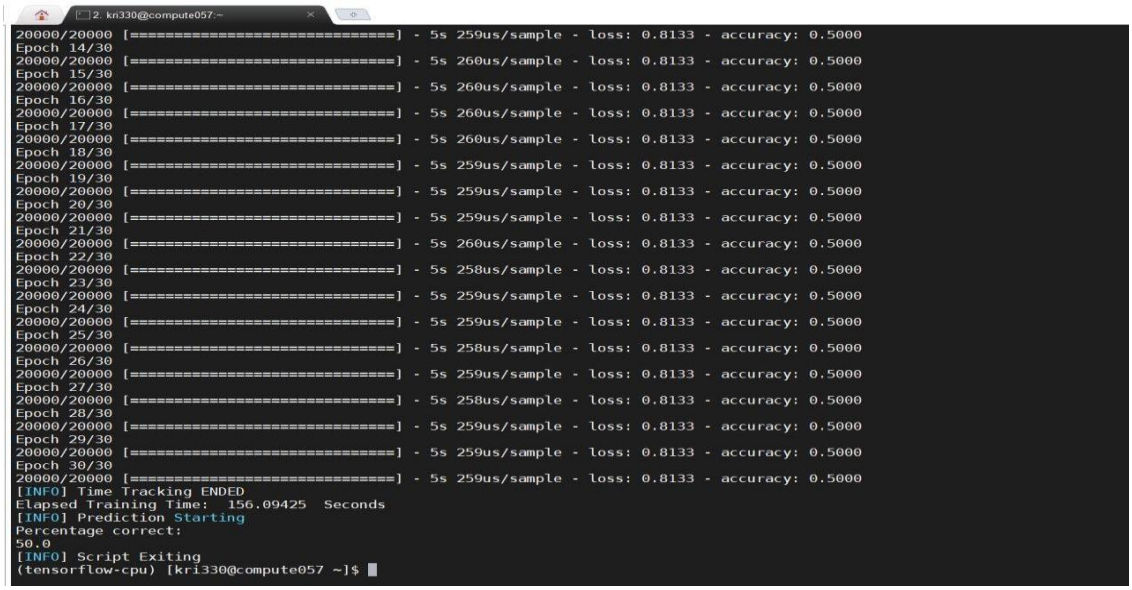


Figure 16. Results on CPU node for epochs = 30

The second set of number of training data and testing data are 65,000 and 10,000, respectively. The Graphs represents the values obtained for the same. It can be interpreted that with small amount of training and testing data the accuracy does not even go above 78.3% whereas the maximum accuracy obtained for 65,000 training and 20,000 testing is 95.05% for epochs = 45.

For set 1 data change in percentage is significant from epoch =10 to epoch = 15, and then there is a drop-in value for epoch =30 after initial increment followed by a rise till epoch = 45. Epoch =25 gives the maximum percentage correct of 78.3 in set 1 data, followed by a dip in amount at epoch = 50 it is 50%. Time in creating and processing the sequential model increases as the number of epochs are incremented. Since the training and testing data amount is limited the authenticity of the accuracy cannot be perfectly predicted.

With the increase in the training and testing samples there is a rise in the accuracy and time required to process the model. The minimum correct is 77.24% at epoch = 30 similar to set 1 and the maximum is 95.05% at epoch = 45. At epoch =50 there is a dip the in value.

It can be clearly estimated that for set 2 data epochs = 45 gives the best result and for set 1 epoch = 25 gives good result, any further increment in the number of epochs reduces the accuracy.

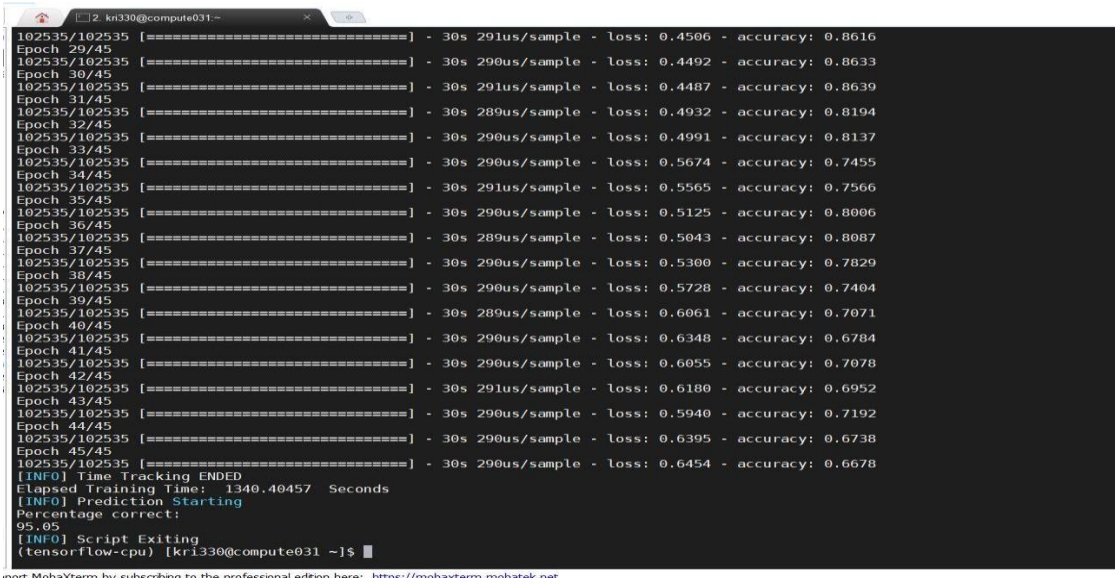


Figure 17. Results on CPU node for epochs = 45

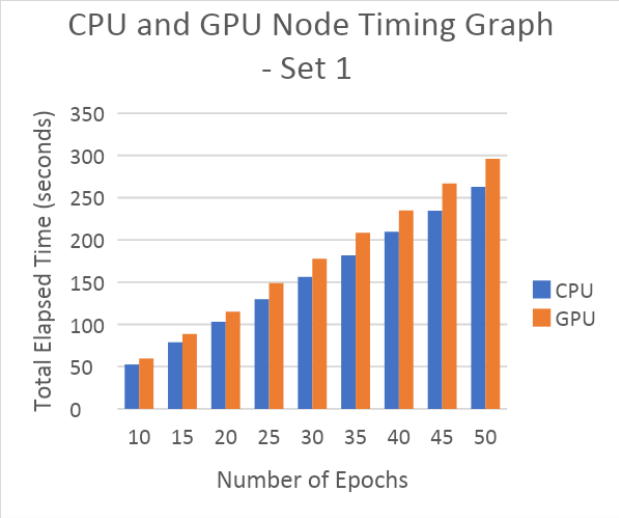


Figure 18

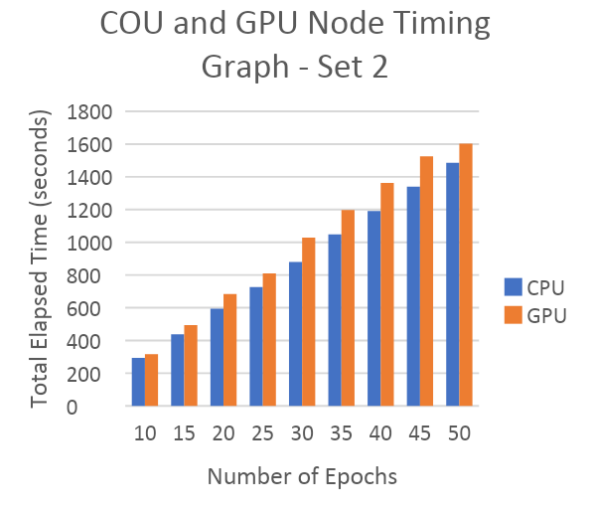


Figure 19

4.2.2 GPU Node

On the GPU node as well, the model is run Graph 1, Graph 2, Graph 3 and Graph 4 represent the values obtained for set 1 and set 2 of data. From Graphs for set one it is clearly interpreted that the accuracy range is form 64.6% – 76.64% and the maximum accuracy is obtained at epoch = 45. At epoch = 50 there is dip in the correct value. In both sets the accuracy is dropped at epoch = 15.

The maximum percentage correct is 100 for set 2 data at epoch = 35 and the range of set 2 accuracy is 6.82% - 100%. For set 1 the epoch range between 30-45 is ideal as the variation is limited. For set 2 epoch range 20-35 is giving best outcome.

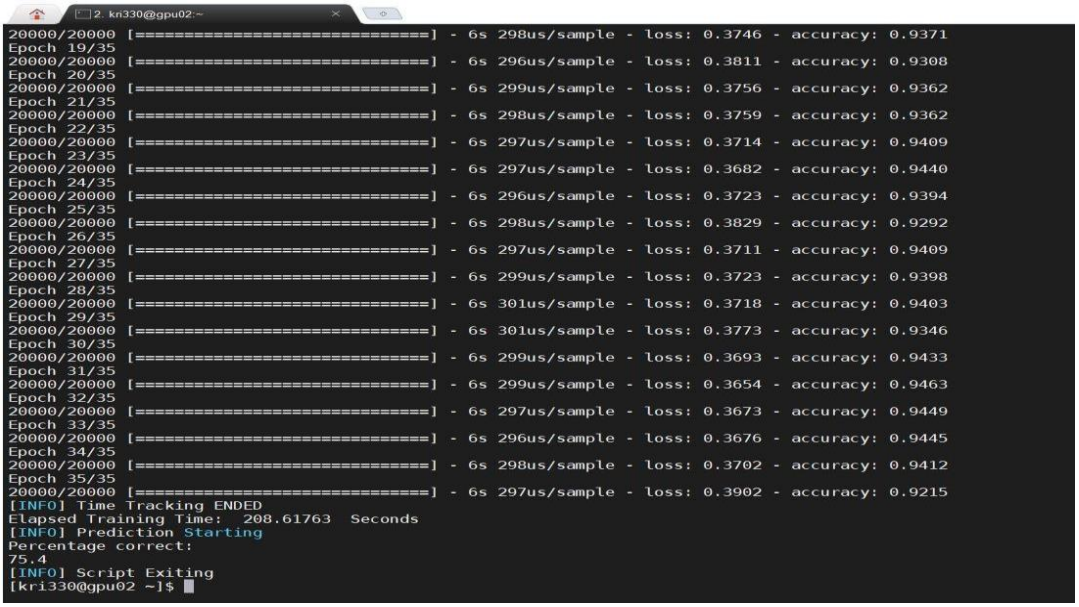


Figure 20. Results on GPU node for epochs = 35

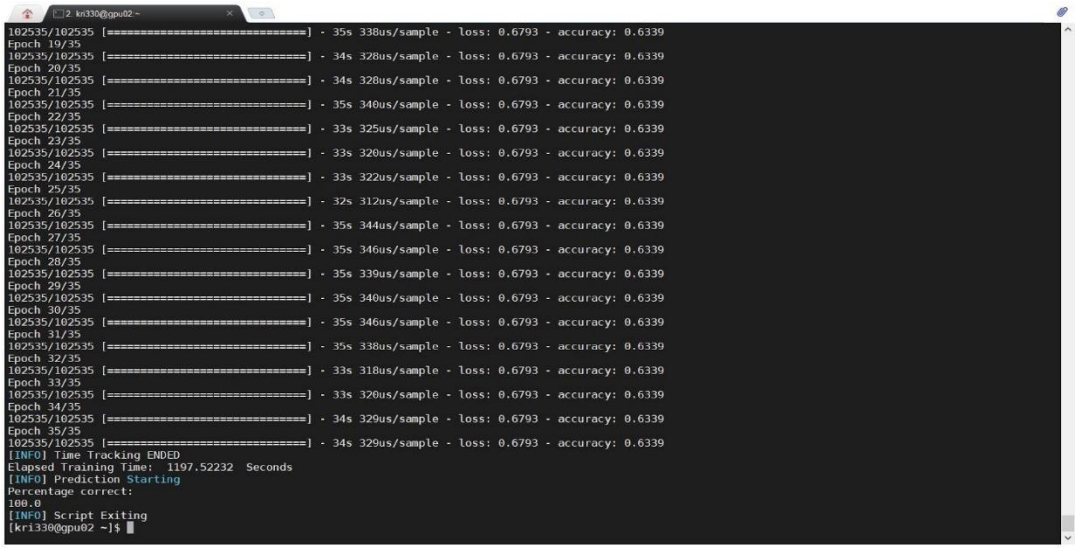


Figure 21. Results on GPU node for epochs = 35

4.2.3 Comparison CPU and GPU Node

The time required to process each epoch of the model on CPU node and GPU node is in seconds, the difference between the time is only a few second more on GPU node than that of CPU node for set 1. As the number of data samples increases the time gap increases little, the time taken to on GPU is still more than CPU.

For set 1 data the is not much of a difference in accuracy just 2% and the CPU is performing better than GPU, on the contrary for large dataset the GPU performs better than CPU by 6 % at lesser number of epochs.

It can be clearly understood that depending on the size of data both the nodes will require different epoch counts to give the best results. Even though the time consumed is more, the GPU node performs better than the CPU node for the designed model.

## 5. Conclusion

There are many ways to perform machine learning. This research tests two of those methods for comparison, SVM and CNN. SVM is a supervised learning technique that excels at classification of data. CNN models a network to recognize patterns between data and excels in this form of classification. It is clear from the data that CNN performed significantly better than SVM. CNN peaked at 95% whereas the optimal SVM peaked at roughly 82% accurate. In addition, the CNN was significantly fast than SVM. Using set 2 and 50 epochs, the CNN took 1600s to process. The shortest time for SVM was also approximately 1600s, while the largest processing time was 9600s. For processing time per point of accuracy, the CNN far outclasses SVM. Typically, a GPU would be expected to process this data faster than a CPU. However, in this research the GPU overall took about 3% longer to finish calculations.

**References**


[1] Moody GB, Mark RG. The impact of the MIT-BIH Arrhythmia Database. IEEE Eng in Med and Biol 20(3):45-50 (May-June 2001). (PMID: 11446209)

[2] Goldberger AL, Amaral LAN, Glass L, Hausdorff JM, Ivanov PCh, Mark RG, Mietus JE, Moody GB, Peng C-K, Stanley HE. PhysioBank, PhysioToolkit, and PhysioNet: Components of a New Research Resource for Complex Physiologic Signals (2003). Circulation. 101(23):e215-e220

[3] "MIT-BIH Arrhythmia Database." MIT-BIH Arrhythmia Database v1.0.0, 24 Feb. 2005, www.physionet.org/content/mitdb/1.0.0/.

[4] Martis, Roshan Joy, et al. "Automated Screening of Arrhythmia Using Wavelet Based Machine Learning Techniques." Journal of Medical Systems, vol. 36, no. 2, 2010, pp. 677–688., doi:10.1007/s10916-010-9535-7.

[5] Übeyli, Elif Derya. "ECG Beats Classification Using Multiclass Support Vector Machines with Error Correcting Output Codes." Digital Signal Processing, vol. 17, no. 3, 2007, pp. 675–684., doi:10.1016/j.dsp.2006.11.009.

[6] Banerjee, Taposh. "EE 5263: Machine Learning Lecture Notes." 7 Nov. 2019.

[7] Yu-Hsin, Joel Emer and Vivienne Sze. "Eyeriss: A Spatial Architecture for Energy-Efficient Dataflow for Convolution Neural Networks". 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture.

[8] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, Olivier Temam. "DianNao: A Small-Footprint High Throughput Accelerator for Ubiquitous Machine Learning". Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems.

[9] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, et al. "In-Datacentre Performance Analysis of a Tensor Processing Unit" Google, Inc., Mountain View, CA USA 2017. In-Datacentre Performance Analysis of a Tensor Processing Unit. In Proceedings of ISCA '17, Toronto, ON, Canada, June 24-28, 2017, 12 pages.

[10] "Tf.keras.Sequential : TensorFlow Core v2.1.0." TensorFlow, www.tensorflow.org/api_docs/python/tf/keras/Sequential.

[11] "The Sequential Model API." Sequential - Keras Documentation, keras.io/models/sequential/.