

# ECG Classification Using Two Class SVM and DNN

CORY DAVIS

SAYLI APHALE

# Contents

❑ Introduction

❑ Background

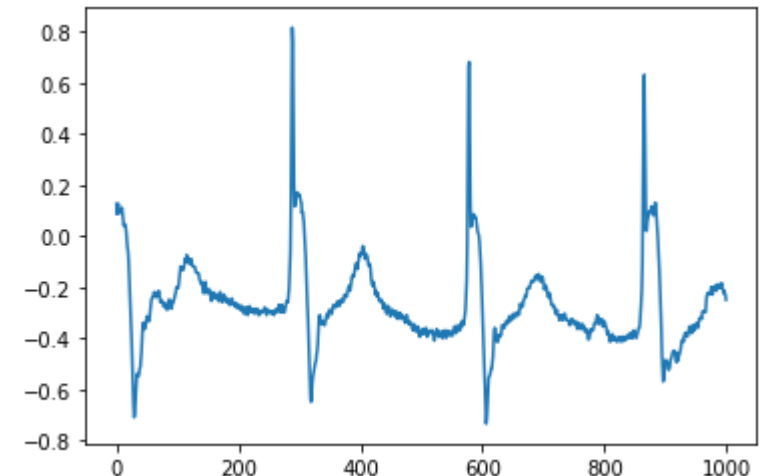
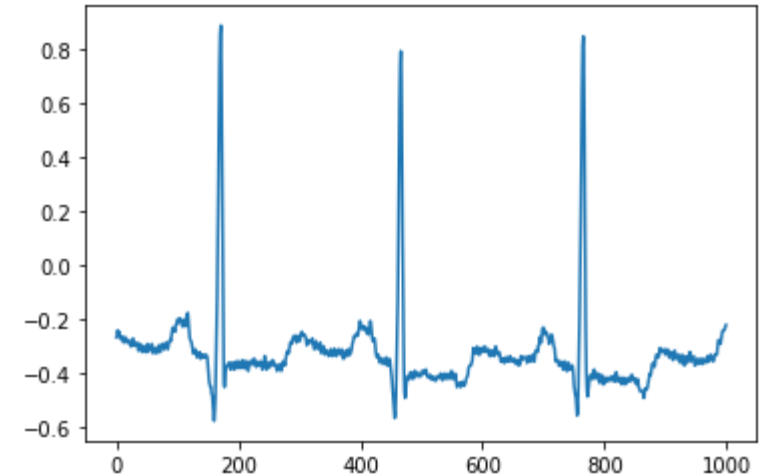
❑ Design of Implemented Sequential Model

❑ Implementation and Results

❑ Conclusion

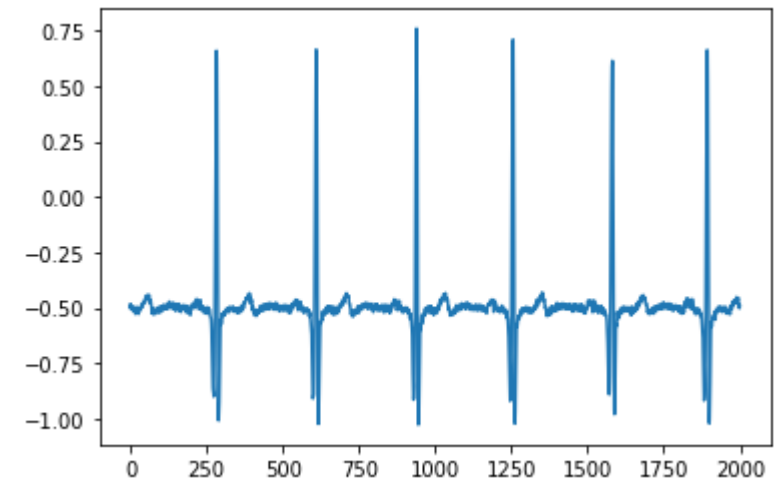
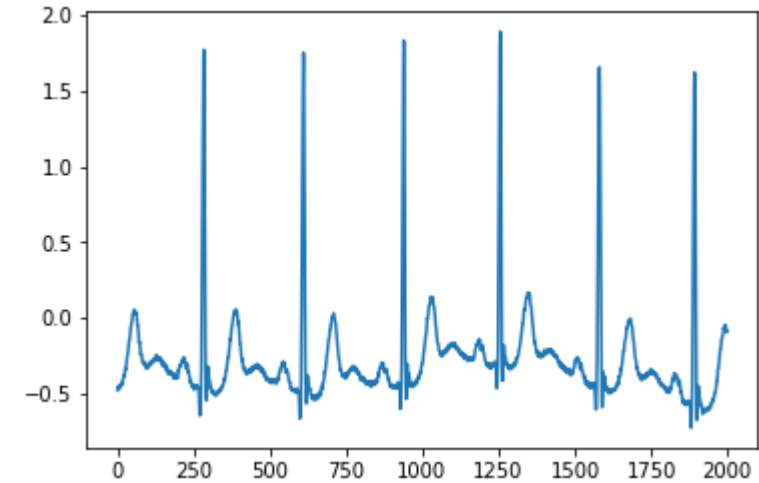
# Introduction

- ❑ Electrocardiograms (ECGs) are test to record heartbeats
- ❑ Measure electrical pulses of heartbeats
- ❑ Cardiologist reads the recordings to detect abnormalities and diagnose heart conditions
- ❑ MIT-BIH Arrhythmia Dataset
  - ❑ 48 recordings – 30 minutes per recording
  - ❑ Approximately 112,000 total heartbeats
  - ❑ Annotations of each peak provides beat type



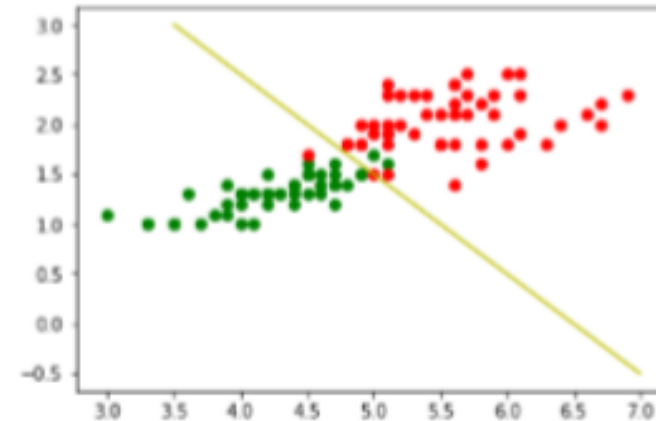
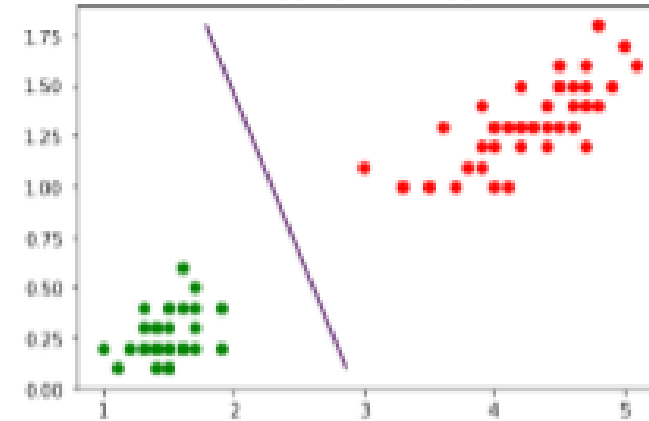
# Introduction

- ❑ Preprocessing
  - ❑ Data contains low-frequency variation
    - ❑ Baseline wander or drift
  - ❑ Data passes through a filter to remove this
  - ❑ Allows for better wave inspection and accurate data processing
- ❑ Pulses are isolated and resampled to length 200
- ❑ Processed using FFT



# Background: Support Vector Machine

- ❑ Supervised machine learning method for data classification
- ❑ This work uses two-class classification
- ❑ Goal of SVM is to determine hyperplane that correctly divides training data
- ❑ Clearly separated 2-D data is easy
- ❑ What about non-perfectly separable data?
- ❑ Hard SVM
- ❑ Soft SVM



# Background: Support Vector Machine

❑ C – optimization Coefficient

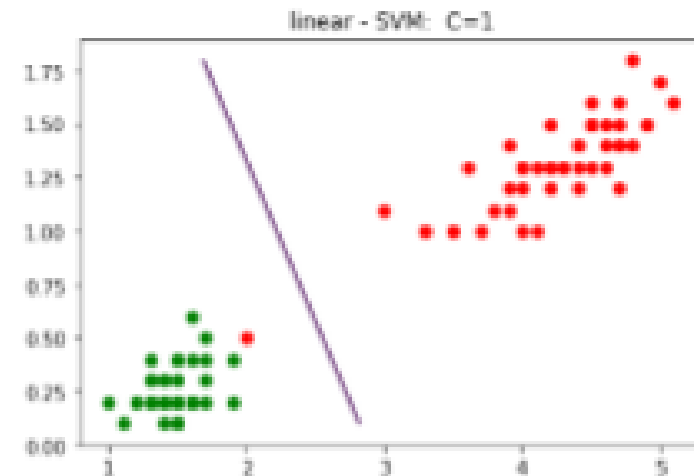
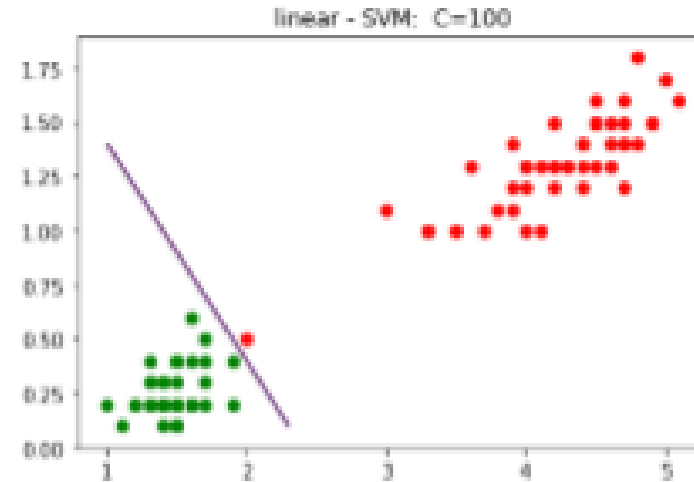
$$\min_{\beta, c, \eta} \quad \beta^T \beta + C \sum_{i=1}^n \eta_i$$

$$\begin{aligned} \text{Subject to} \quad & y_i(\beta^T x_i + c) \geq 1 - \eta_i, \quad \text{for } i = 1, 2, \dots, n \\ \text{and} \quad & \eta_i \geq 0, \quad \text{for } i = 1, 2, \dots, n. \end{aligned}$$

❑ Higher values of C creates a strict, and more unstable separation

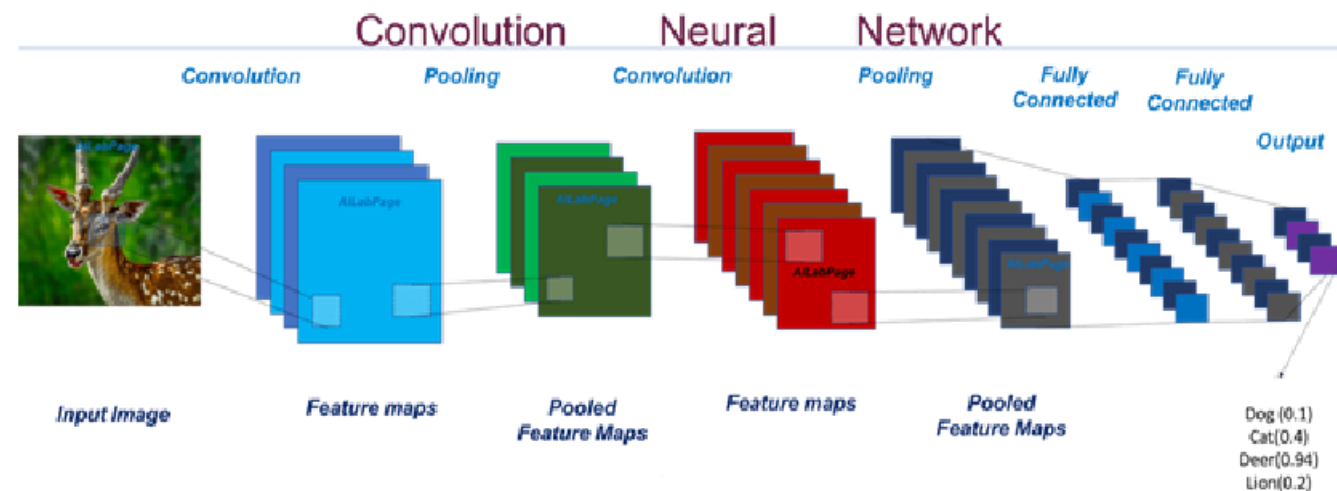
❑ Lower values of C creates a less restrictive and more stable fit.

❑ Range of C values must be tested for peak accuracy



# Background of Neural Networks

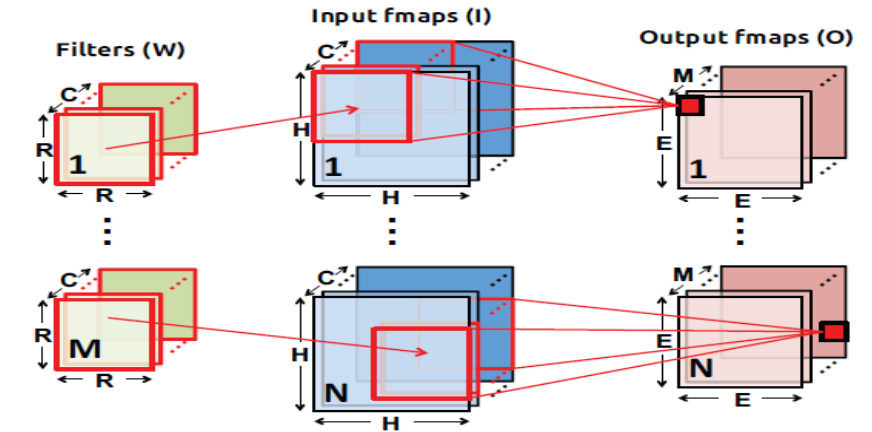
- ❑ Neural Networks Designed to recognize patterns
- ❑ Convolution Neural Networks are constructed by stacking Multiple
- ❑ Convolutional Neural Networks are constructed by stacking Multiple computational layers, each layer provides a higher layer of abstraction of Input data –Feature map.
- ❑ Performance of a CNN depends on the density of the network i.e. the type and number of layers used to design.
- ❑ Types of CNN – VGG-16, ResNet50, Xception, etc
- ❑ Types of layers in CNN
  - Input Layer
  - Hidden layer
  - Output layer
- ❑ Hidden layer carrier out feature extraction



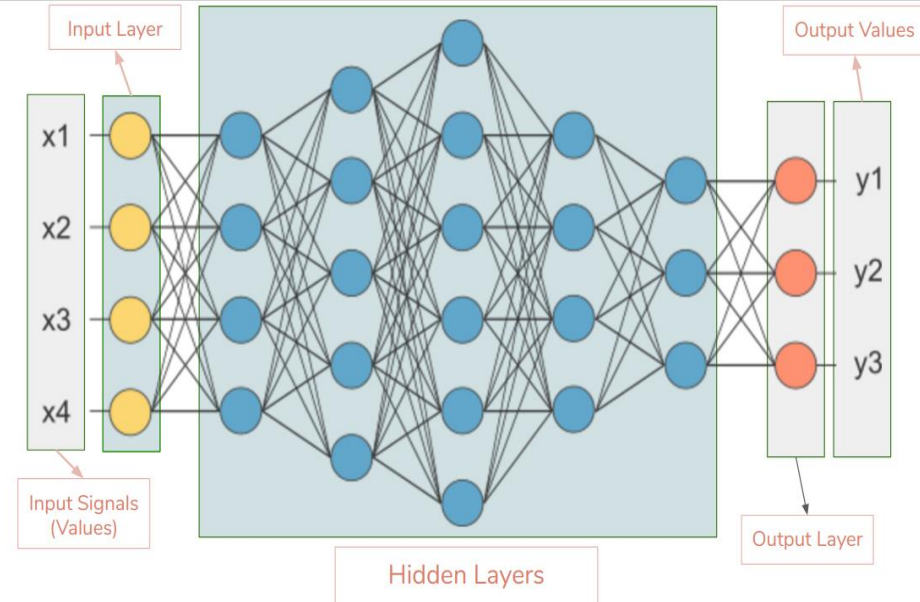
# Background of Neural Networks

## ❑ Hidden layers in CNN:

- Convolution layer : Uses Matrix filter to detect patterns in image
- Activation Function : Rectification of the feature map of the image
- Pooling layer : Identify edges, corners, etc along with down sampling
- Flatten layer: To convert 2-D array into single linear vector
- Fully connected layer : Classification of images



- ❑ CNN Decoder Ring states dimensions of the Input Feature maps, Filters , Output
- ❑ Today CNN are designed to be very dense to obtain : High performance Accuracy
- ❑ Time required to train dense CNN is large: Large Number of Layers & Dataset
- ❑ Training time can be sizable but the inference time should be lowest
- ❑ CNN application are user response based.
- ❑ Applications of CNN : Medical, Finance, Infrastructure





# Design of Implemented Sequential Model

## ❑ Model : Keras Sequential

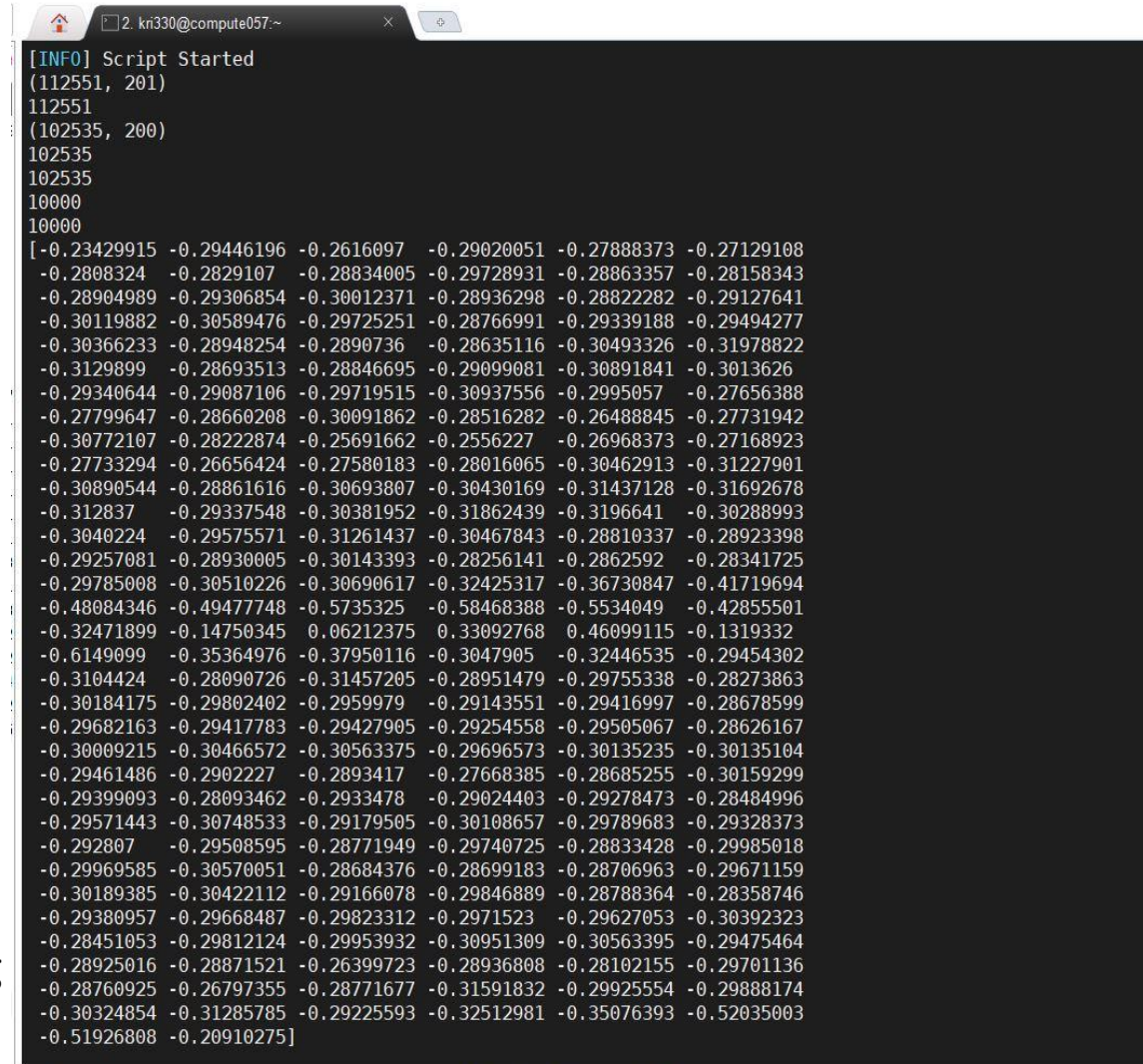
- Total 16 layers
- Convolution 1-D : 6 layers (input layer)
- MaxPooling 1-D : 2 layers
- GlobalMaxPooling : 1 layer
- Activation Function : ReLu
- Dense (fully connected) : 3 layers
- Before feeding input into Dense layer it is Flattened

## ❑ Model is then compiled, fitted and testing is performed

## ❑ Two sets of data:

- Set 1: 20,000 beats for Training, 2,000 beats for testing
- Set 2: 65,000 beats for Training, 10,000 beats for testing

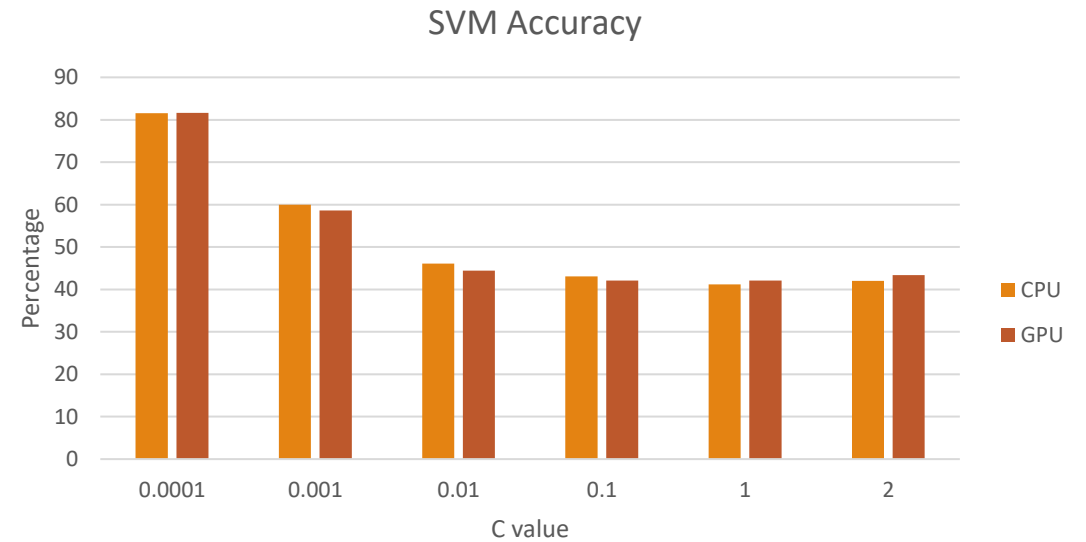
## ❑ Model run on CPU and GPU Nodes



```
[INFO] Script Started
(112551, 201)
112551
(102535, 200)
102535
102535
10000
10000
[ -0.23429915 -0.29446196 -0.2616097 -0.29020051 -0.27888373 -0.27129108
-0.2808324 -0.2829107 -0.28834005 -0.29728931 -0.28863357 -0.28158343
-0.28904989 -0.29306854 -0.30012371 -0.28936298 -0.28822282 -0.29127641
-0.30119882 -0.30589476 -0.29725251 -0.28766991 -0.29339188 -0.29494277
-0.30366233 -0.28948254 -0.2890736 -0.28635116 -0.30493326 -0.31978822
-0.3129899 -0.28693513 -0.28846695 -0.29099081 -0.30891841 -0.3013626
-0.29340644 -0.29087106 -0.29719515 -0.30937556 -0.2995057 -0.27656388
-0.27799647 -0.28660208 -0.30091862 -0.28516282 -0.26488845 -0.27731942
-0.30772107 -0.28222874 -0.25691662 -0.2556227 -0.26968373 -0.27168923
-0.27733294 -0.26656424 -0.27580183 -0.28016065 -0.30462913 -0.31227901
-0.30890544 -0.28861616 -0.30693807 -0.30430169 -0.31437128 -0.31692678
-0.312837 -0.29337548 -0.30381952 -0.31862439 -0.3196641 -0.30288993
-0.3040224 -0.29575571 -0.31261437 -0.30467843 -0.28810337 -0.28923398
-0.29257081 -0.28930005 -0.30143393 -0.28256141 -0.2862592 -0.28341725
-0.29785008 -0.30510226 -0.30690617 -0.32425317 -0.36730847 -0.41719694
-0.48084346 -0.49477748 -0.5735325 -0.58468388 -0.5534049 -0.42855501
-0.32471899 -0.14750345 -0.06212375 -0.33092768 -0.46099115 -0.1319332
-0.6149099 -0.35364976 -0.37950116 -0.3047905 -0.32446535 -0.29454302
-0.3104424 -0.28090726 -0.31457205 -0.28951479 -0.29755338 -0.28273863
-0.30184175 -0.29802402 -0.2959979 -0.29143551 -0.29416997 -0.28678599
-0.29682163 -0.29417783 -0.29427905 -0.29254558 -0.29505067 -0.28626167
-0.30009215 -0.30466572 -0.30563375 -0.29696573 -0.30135235 -0.30135104
-0.29461486 -0.2902227 -0.2893417 -0.27668385 -0.28685255 -0.30159299
-0.29399093 -0.28093462 -0.2933478 -0.29024403 -0.29278473 -0.28484996
-0.29571443 -0.30748533 -0.29179505 -0.30108657 -0.29789683 -0.29328373
-0.292807 -0.29508595 -0.28771949 -0.29740725 -0.28833428 -0.29985018
-0.29969585 -0.30570051 -0.28684376 -0.28699183 -0.28706963 -0.29671159
-0.30189385 -0.30422112 -0.29166078 -0.29846889 -0.28788364 -0.28358746
-0.29380957 -0.29668487 -0.29823312 -0.2971523 -0.29627053 -0.30392323
-0.28451053 -0.29812124 -0.29953932 -0.30951309 -0.30563395 -0.29475464
-0.28925016 -0.28871521 -0.26399723 -0.28936808 -0.28102155 -0.29701136
-0.28760925 -0.26797355 -0.28771677 -0.31591832 -0.29925554 -0.29888174
-0.30324854 -0.31285785 -0.29225593 -0.32512981 -0.35076393 -0.52035003
-0.51926808 -0.20910275]
```

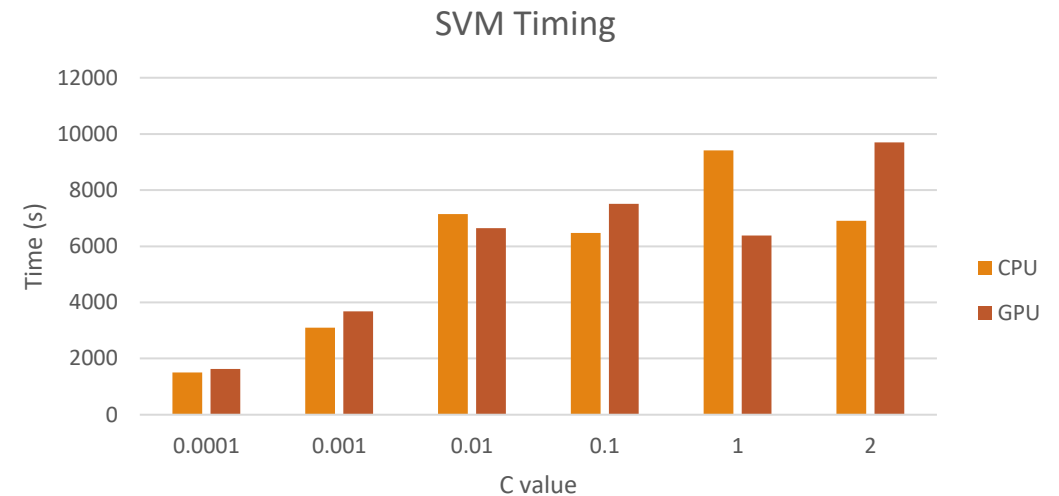
# SVM Results

- ❑ Tested using C values: 0.0001, 0.001, 0.01, 0.1, 1.0, 2.0
- ❑ C value – 0.0001 yielded 81% across CPU and GPU
- ❑ 0.5% difference in accuracy in favor of CPU
- ❑ By far the best accuracy
- ❑ Harder SVM yielded accuracy less than a coin flip
- ❑ The variation of “normal” heartbeats amongst patients, lends towards softer SVM



# SVM Results

- ❑ Tested using C values: 0.0001, 0.001, 0.01, 0.1, 1.0, 2.0
- ❑ the GPU took 2.7% longer than the CPU to process the data
- ❑ C value – 0.0001 was the fastest to process at about 1600s
- ❑ Processing time peaked at 9600s for GPU/C value – 2.0

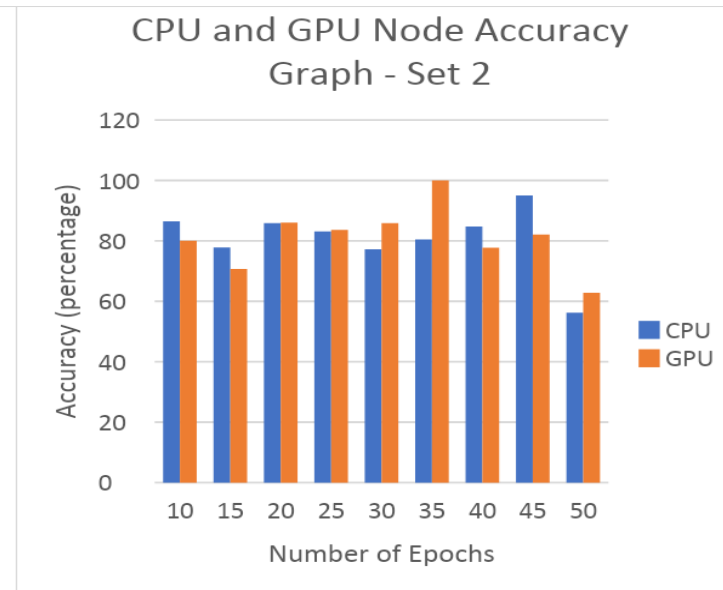
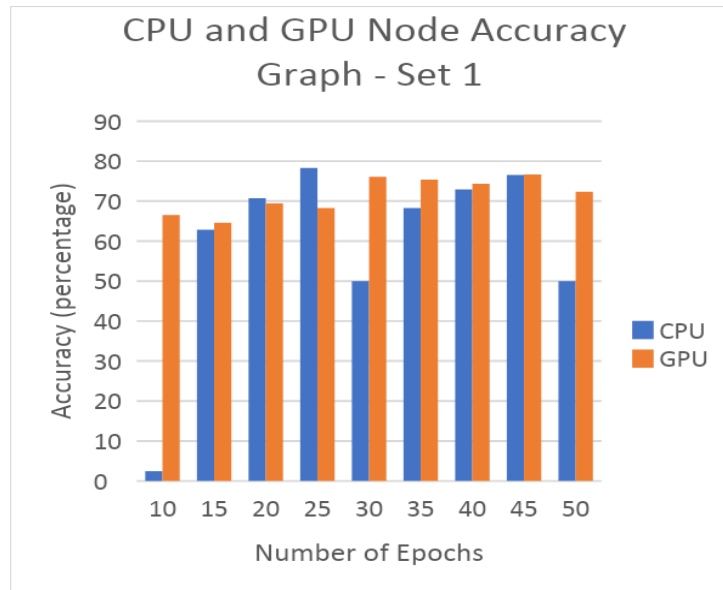


# Neural Network Results

## ❑ Accuracy

- Set 1- CPU Node: Range : 2.45% - 78.3%, Max at epoch =25
- Set 2- CPU Node: Range : 56.21% - 95.05%, Max at epoch = 45
- CPU Node: Decrease in correct percentage at epoch = 30 after initial rise
- Set 1- GPU Node: Range : 64.6% – 76.64% , Max at epoch = 45
- Set 2- GPU Node: Range : 68.20% - 100.0%, Max at epoch = 35
- GPU Node : Set 1: Epoch range 30-45 is ideal as the variation is limited & For set 2 epoch range 20-35 is giving best

outcome





# Neural Network Results

## Accuracy : CPU Node

```
2. kn330@compute057:~  
20000/20000 [=====] - 5s 259us/sample - loss: 0.8133 - accuracy: 0.5000  
Epoch 14/30  
20000/20000 [=====] - 5s 260us/sample - loss: 0.8133 - accuracy: 0.5000  
Epoch 15/30  
20000/20000 [=====] - 5s 260us/sample - loss: 0.8133 - accuracy: 0.5000  
Epoch 16/30  
20000/20000 [=====] - 5s 260us/sample - loss: 0.8133 - accuracy: 0.5000  
Epoch 17/30  
20000/20000 [=====] - 5s 260us/sample - loss: 0.8133 - accuracy: 0.5000  
Epoch 18/30  
20000/20000 [=====] - 5s 259us/sample - loss: 0.8133 - accuracy: 0.5000  
Epoch 19/30  
20000/20000 [=====] - 5s 259us/sample - loss: 0.8133 - accuracy: 0.5000  
Epoch 20/30  
20000/20000 [=====] - 5s 259us/sample - loss: 0.8133 - accuracy: 0.5000  
Epoch 21/30  
20000/20000 [=====] - 5s 260us/sample - loss: 0.8133 - accuracy: 0.5000  
Epoch 22/30  
20000/20000 [=====] - 5s 258us/sample - loss: 0.8133 - accuracy: 0.5000  
Epoch 23/30  
20000/20000 [=====] - 5s 259us/sample - loss: 0.8133 - accuracy: 0.5000  
Epoch 24/30  
20000/20000 [=====] - 5s 259us/sample - loss: 0.8133 - accuracy: 0.5000  
Epoch 25/30  
20000/20000 [=====] - 5s 258us/sample - loss: 0.8133 - accuracy: 0.5000  
Epoch 26/30  
20000/20000 [=====] - 5s 259us/sample - loss: 0.8133 - accuracy: 0.5000  
Epoch 27/30  
20000/20000 [=====] - 5s 258us/sample - loss: 0.8133 - accuracy: 0.5000  
Epoch 28/30  
20000/20000 [=====] - 5s 259us/sample - loss: 0.8133 - accuracy: 0.5000  
Epoch 29/30  
20000/20000 [=====] - 5s 259us/sample - loss: 0.8133 - accuracy: 0.5000  
Epoch 30/30  
20000/20000 [=====] - 5s 259us/sample - loss: 0.8133 - accuracy: 0.5000  
[INFO] Time Tracking ENDED  
Elapsed Training Time: 156.09425 Seconds  
[INFO] Prediction Starting  
Percentage correct:  
50.0  
[INFO] Script Exiting  
(tensorflow-cpu) [kri330@compute057 ~]$
```

```
2. kn330@compute031:~  
102535/102535 [=====] - 30s 291us/sample - loss: 0.4506 - accuracy: 0.8616  
Epoch 29/45  
102535/102535 [=====] - 30s 290us/sample - loss: 0.4492 - accuracy: 0.8633  
Epoch 30/45  
102535/102535 [=====] - 30s 291us/sample - loss: 0.4487 - accuracy: 0.8639  
Epoch 31/45  
102535/102535 [=====] - 30s 289us/sample - loss: 0.4932 - accuracy: 0.8194  
Epoch 32/45  
102535/102535 [=====] - 30s 290us/sample - loss: 0.4991 - accuracy: 0.8137  
Epoch 33/45  
102535/102535 [=====] - 30s 290us/sample - loss: 0.5674 - accuracy: 0.7455  
Epoch 34/45  
102535/102535 [=====] - 30s 291us/sample - loss: 0.5565 - accuracy: 0.7566  
Epoch 35/45  
102535/102535 [=====] - 30s 290us/sample - loss: 0.5125 - accuracy: 0.8006  
Epoch 36/45  
102535/102535 [=====] - 30s 289us/sample - loss: 0.5043 - accuracy: 0.8087  
Epoch 37/45  
102535/102535 [=====] - 30s 290us/sample - loss: 0.5300 - accuracy: 0.7829  
Epoch 38/45  
102535/102535 [=====] - 30s 290us/sample - loss: 0.5728 - accuracy: 0.7404  
Epoch 39/45  
102535/102535 [=====] - 30s 289us/sample - loss: 0.6061 - accuracy: 0.7071  
Epoch 40/45  
102535/102535 [=====] - 30s 290us/sample - loss: 0.6348 - accuracy: 0.6784  
Epoch 41/45  
102535/102535 [=====] - 30s 290us/sample - loss: 0.6055 - accuracy: 0.7078  
Epoch 42/45  
102535/102535 [=====] - 30s 291us/sample - loss: 0.6180 - accuracy: 0.6952  
Epoch 43/45  
102535/102535 [=====] - 30s 290us/sample - loss: 0.5940 - accuracy: 0.7192  
Epoch 44/45  
102535/102535 [=====] - 30s 290us/sample - loss: 0.6395 - accuracy: 0.6738  
Epoch 45/45  
102535/102535 [=====] - 30s 290us/sample - loss: 0.6454 - accuracy: 0.6678  
[INFO] Time Tracking ENDED  
Elapsed Training Time: 1340.40457 Seconds  
[INFO] Prediction Starting  
Percentage correct:  
95.05  
[INFO] Script Exiting  
(tensorflow-cpu) [kri330@compute031 ~]$
```

Want MahaYarn live easter-eggs in the professional edition here: <https://mhaayarn.mahata.net>

# Neural Network Results

Accuracy : GPU Node

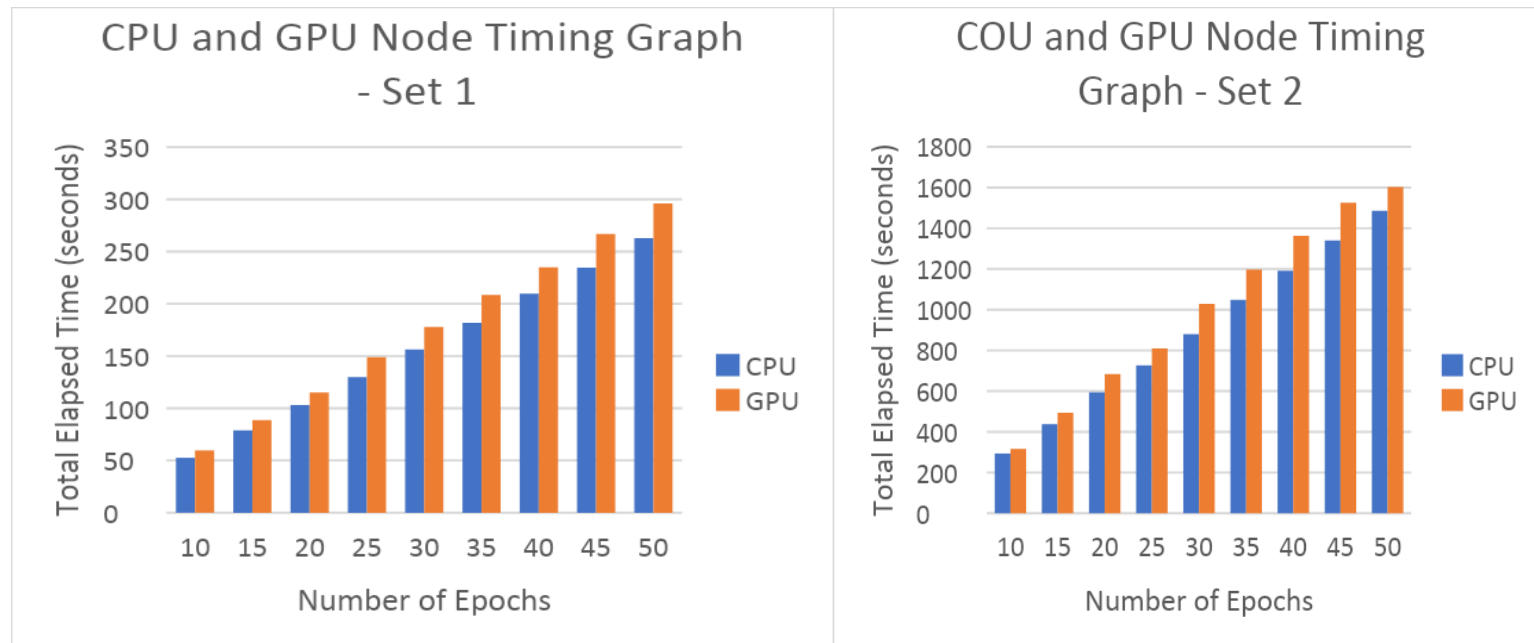
```
20000/20000 [=====] - 6s 298us/sample - loss: 0.3746 - accuracy: 0.9371
Epoch 19/35
20000/20000 [=====] - 6s 296us/sample - loss: 0.3811 - accuracy: 0.9308
Epoch 20/35
20000/20000 [=====] - 6s 299us/sample - loss: 0.3756 - accuracy: 0.9362
Epoch 21/35
20000/20000 [=====] - 6s 298us/sample - loss: 0.3759 - accuracy: 0.9362
Epoch 22/35
20000/20000 [=====] - 6s 297us/sample - loss: 0.3714 - accuracy: 0.9409
Epoch 23/35
20000/20000 [=====] - 6s 297us/sample - loss: 0.3682 - accuracy: 0.9440
Epoch 24/35
20000/20000 [=====] - 6s 296us/sample - loss: 0.3723 - accuracy: 0.9394
Epoch 25/35
20000/20000 [=====] - 6s 298us/sample - loss: 0.3829 - accuracy: 0.9292
Epoch 26/35
20000/20000 [=====] - 6s 297us/sample - loss: 0.3711 - accuracy: 0.9409
Epoch 27/35
20000/20000 [=====] - 6s 299us/sample - loss: 0.3723 - accuracy: 0.9398
Epoch 28/35
20000/20000 [=====] - 6s 301us/sample - loss: 0.3718 - accuracy: 0.9403
Epoch 29/35
20000/20000 [=====] - 6s 301us/sample - loss: 0.3773 - accuracy: 0.9346
Epoch 30/35
20000/20000 [=====] - 6s 299us/sample - loss: 0.3693 - accuracy: 0.9433
Epoch 31/35
20000/20000 [=====] - 6s 299us/sample - loss: 0.3654 - accuracy: 0.9463
Epoch 32/35
20000/20000 [=====] - 6s 297us/sample - loss: 0.3673 - accuracy: 0.9449
Epoch 33/35
20000/20000 [=====] - 6s 296us/sample - loss: 0.3676 - accuracy: 0.9445
Epoch 34/35
20000/20000 [=====] - 6s 298us/sample - loss: 0.3702 - accuracy: 0.9412
Epoch 35/35
20000/20000 [=====] - 6s 297us/sample - loss: 0.3902 - accuracy: 0.9215
[INFO] Time Tracking ENDED
Elapsed Training Time: 208.61763 Seconds
[INFO] Prediction Starting
Percentage correct:
75.4
[INFO] Script Exiting
[kri330@gpu02 ~]$
```

```
102535/102535 [=====] - 35s 338us/sample - loss: 0.6793 - accuracy: 0.6339
Epoch 19/35
102535/102535 [=====] - 34s 328us/sample - loss: 0.6793 - accuracy: 0.6339
Epoch 20/35
102535/102535 [=====] - 34s 328us/sample - loss: 0.6793 - accuracy: 0.6339
Epoch 21/35
102535/102535 [=====] - 35s 340us/sample - loss: 0.6793 - accuracy: 0.6339
Epoch 22/35
102535/102535 [=====] - 33s 325us/sample - loss: 0.6793 - accuracy: 0.6339
Epoch 23/35
102535/102535 [=====] - 33s 320us/sample - loss: 0.6793 - accuracy: 0.6339
Epoch 24/35
102535/102535 [=====] - 33s 322us/sample - loss: 0.6793 - accuracy: 0.6339
Epoch 25/35
102535/102535 [=====] - 32s 312us/sample - loss: 0.6793 - accuracy: 0.6339
Epoch 26/35
102535/102535 [=====] - 35s 344us/sample - loss: 0.6793 - accuracy: 0.6339
Epoch 27/35
102535/102535 [=====] - 35s 346us/sample - loss: 0.6793 - accuracy: 0.6339
Epoch 28/35
102535/102535 [=====] - 35s 339us/sample - loss: 0.6793 - accuracy: 0.6339
Epoch 29/35
102535/102535 [=====] - 35s 340us/sample - loss: 0.6793 - accuracy: 0.6339
Epoch 30/35
102535/102535 [=====] - 35s 346us/sample - loss: 0.6793 - accuracy: 0.6339
Epoch 31/35
102535/102535 [=====] - 35s 338us/sample - loss: 0.6793 - accuracy: 0.6339
Epoch 32/35
102535/102535 [=====] - 33s 318us/sample - loss: 0.6793 - accuracy: 0.6339
Epoch 33/35
102535/102535 [=====] - 33s 320us/sample - loss: 0.6793 - accuracy: 0.6339
Epoch 34/35
102535/102535 [=====] - 34s 329us/sample - loss: 0.6793 - accuracy: 0.6339
Epoch 35/35
102535/102535 [=====] - 34s 329us/sample - loss: 0.6793 - accuracy: 0.6339
[INFO] Time Tracking ENDED
Elapsed Training Time: 1197.52232 Seconds
[INFO] Prediction Starting
Percentage correct:
100.0
[INFO] Script Exiting
[kri330@gpu02 ~]$
```

# Neural Network Results

## ❑ Total Elapsed Time

- Time in creating and processing the sequential model increases as the number of epochs are incremented
- With the increase in the training and testing samples there is a rise in time required to process the model
- GPU takes more time CPU





# Comparison CPU and GPU Node

- ❑ The time required to process each epoch of the model on CPU node and GPU node is in seconds
- ❑ Difference between the time is only a few second more on GPU node than that of CPU node for set 1
- ❑ 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



# 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.
- ❑ Using set 2 and 50 epochs, the CNN took 1600s to process
- ❑ Shortest time for SVM was also approximately 1600s, while the largest processing time was 9600s
- ❑ 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.