NEURAL NETWORK APPLICATIONS COURSE (CSE616) FINAL PROJECT – Final Presentation

Yomna Hesham Amin – G19093453

Ahmed Mohamed Farouk – 2000630

Introduction

This project implements the solution proposed in "**A Study on Arrhythmia via ECG Signal Classification Using the Convolutional Neural Network**" paper in Python, as the solution should have been implemented in Matlab but there's no reference for the code in the paper.

Paper Link:
https://www.frontiersin.org/articles/10.3389/fncom.2020.564015/full

Dataset:
https://physionet.org/content/mitdb/1.0.0/

Dataset in CSV:
https://www.kaggle.com/taejoongyoon/mitbit-arrhythmia-database

Our Implementation:
https://www.kaggle.com/yomnahesham/cse616-final-project

Paper Solution Brief

“This paper proposes a robust and efficient 12-layer deep one-dimensional convolutional neural network on classifying the five micro-classes of heartbeat types in the MIT- BIH Arrhythmia database. The five types of heartbeat features are classified, and wavelet self-adaptive threshold denoising method is used in the experiments”

“In the previous literature (Zubair et al., 2016; Acharya et al., 2017a,b; Yildirim et al., 2018; Atal and Singh, 2020), most of the works focus on the recognition of five main macro classes, namely Non-ectopic (N); Supraventricular ectopic (S); Ventricular ectopic (V); Fusion (F); Unknown (Q)”.

“There is very little effort devoted to classify the micro-classes of the ECG signal, hence it serves as our main motivation to study the micro-classification heartbeats, of five types, i.e., Normal (NOR), Left Bundle Branch Block (LBBB), Right Bundle Branch Block (RBBB), Atrial Premature (AP), Premature Ventricular Contraction (PVC).”

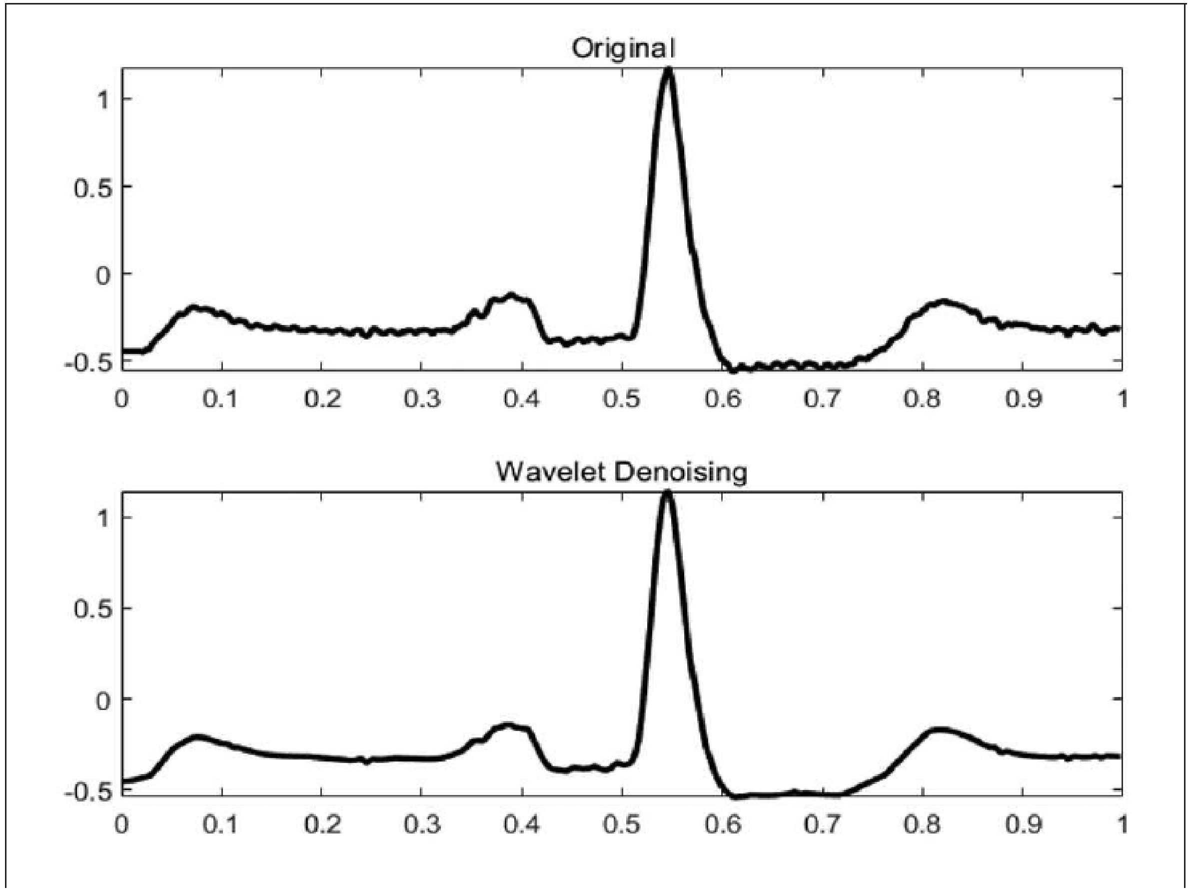
Paper Solution Steps

- Data Preprocessing
- Convolutional Neural Network

Paper Solution Steps – Data Preprocessing

- Data Denoising
  - Wavelet Transform with Sym4 Wavelet
  - Self-Adaptive Threshold
- Data Segmentation
  - Z-Score Normalization
  - Segmenting into Beats
  - 360 Sample/Beak
- Enhancement
  - Only 16 records are used
  - Rebalance the classes

Paper Solution Steps – Data Preprocessing



Paper Solution Steps – Convolutional Neural Network

- One-dimensional 12-Layer convolution with Average Pooling
- Dropout Layer
- Fully Connected Layer
- Softmax Output Layer
- Ten-Fold Cross Validation

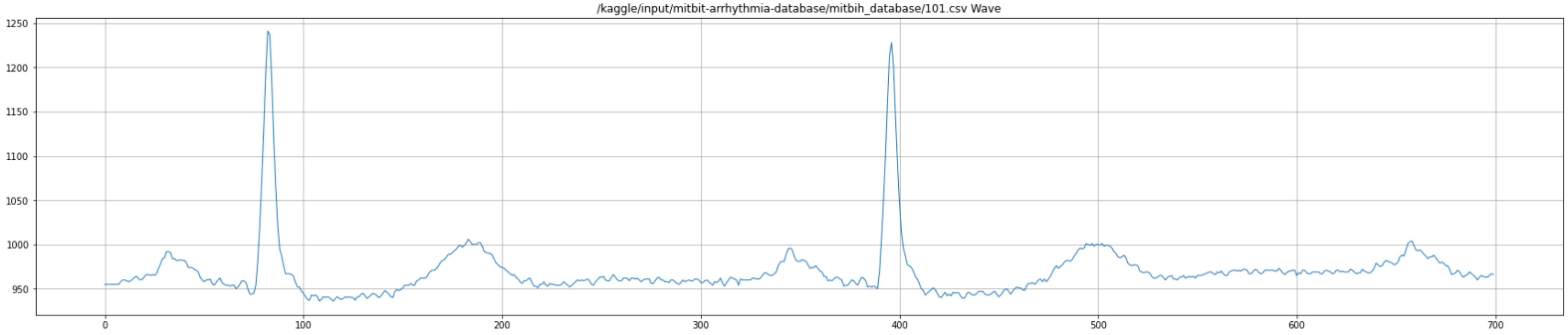
Paper Solution Steps – Convolutional Neural Network

Layers	Type	Output	Kernel size	Stride
Layer 1	Convolution	360*16	1*13	1
Layer 2	Average-Pooling	179*16	1*3	2
Layer 3	Convolution	179*32	1*15	1
Layer 4	Average-Pooling	89*32	1*3	2
Layer 5	Convolution	89*64	1*17	1
Layer 6	Average-Pooling	44*64	1*3	2
Layer 7	Convolution	44*128	1*19	1
Layer 8	Average-Pooling	21*128	1*3	2
Layer 9	Dropout	21*128	-	-
Layer 10	Fully-connected	1*35	-	-
Layer 11	Fully-connected	1*5	-	-
Layer 12	SoftMax	1*5	-	-

Our Solution Steps – Data Preprocessing

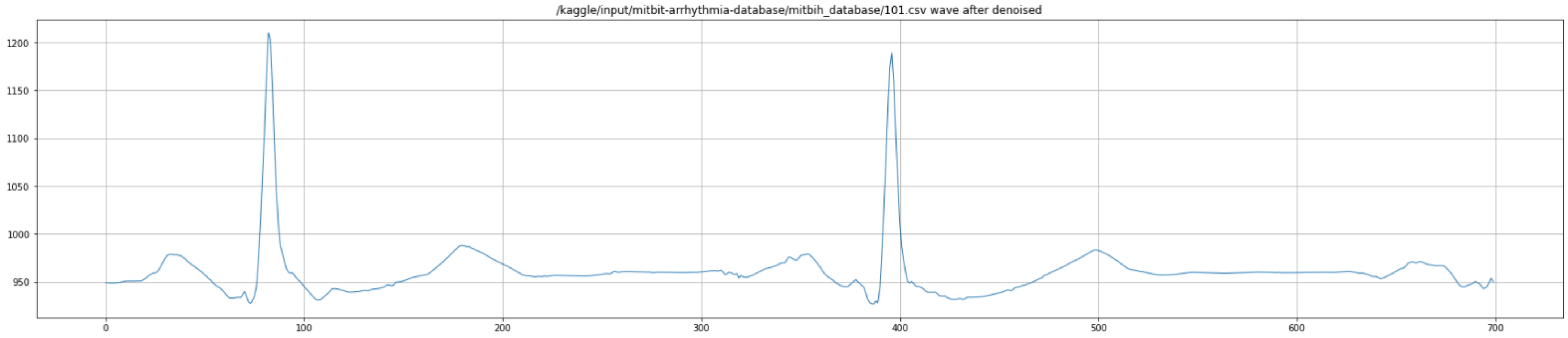
- Data Denoising
  - Wavelet Transform with Sym4 Wavelet *(DONE)*
  - Self-Adaptive Threshold *(Threshold is set to a fixed value instead)*
- Data Segmentation
  - Z-Score Normalization *(DONE)*
  - Segmenting into Beats *(DONE)*
  - 360 Sample/Beak *(DONE)*
- Enhancement
  - Only 16 records are used *(All records are used instead)*
  - Rebalance the classes *(DONE)*

Our Solution Steps – Data Preprocessing



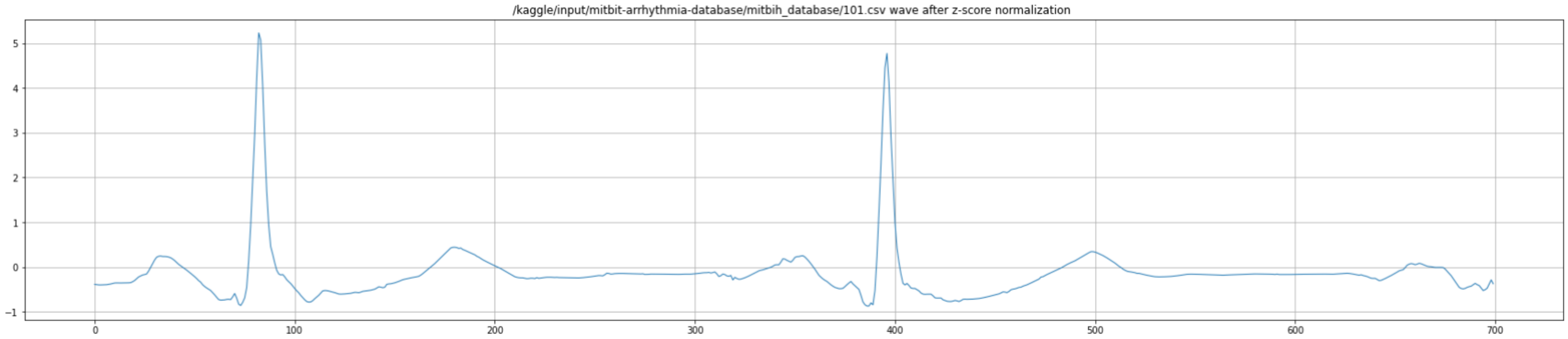
Original Signal

Our Solution Steps – Data Preprocessing



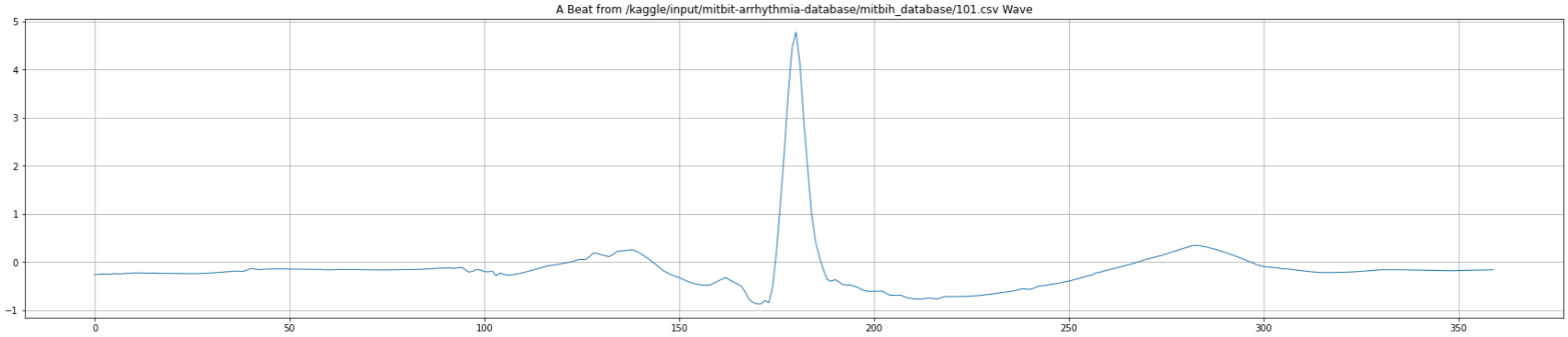
Denoised Signal

Our Solution Steps – Data Preprocessing



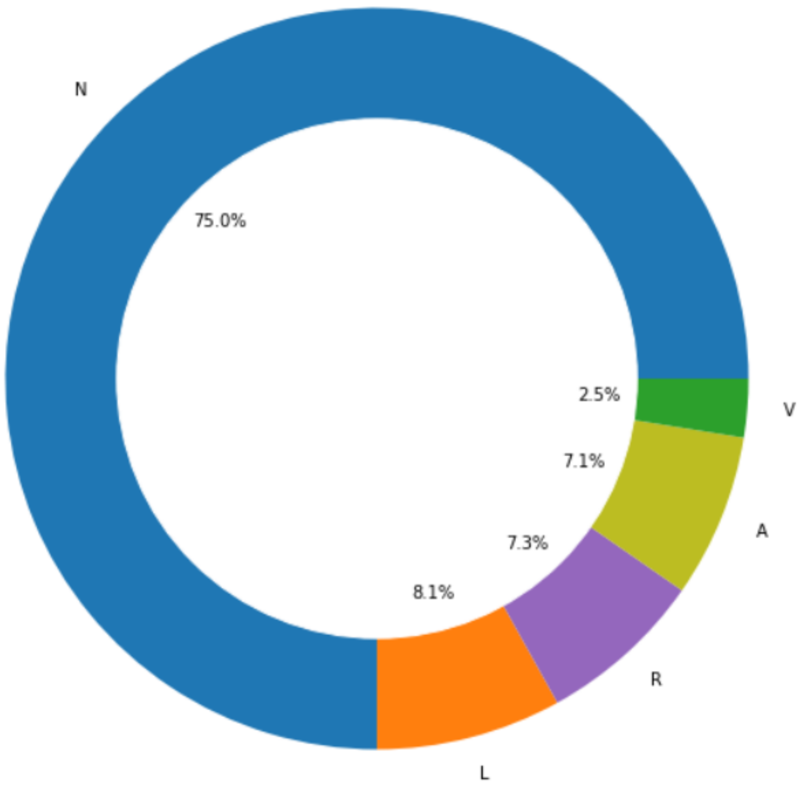
Normalized Signal

Our Solution Steps – Data Preprocessing



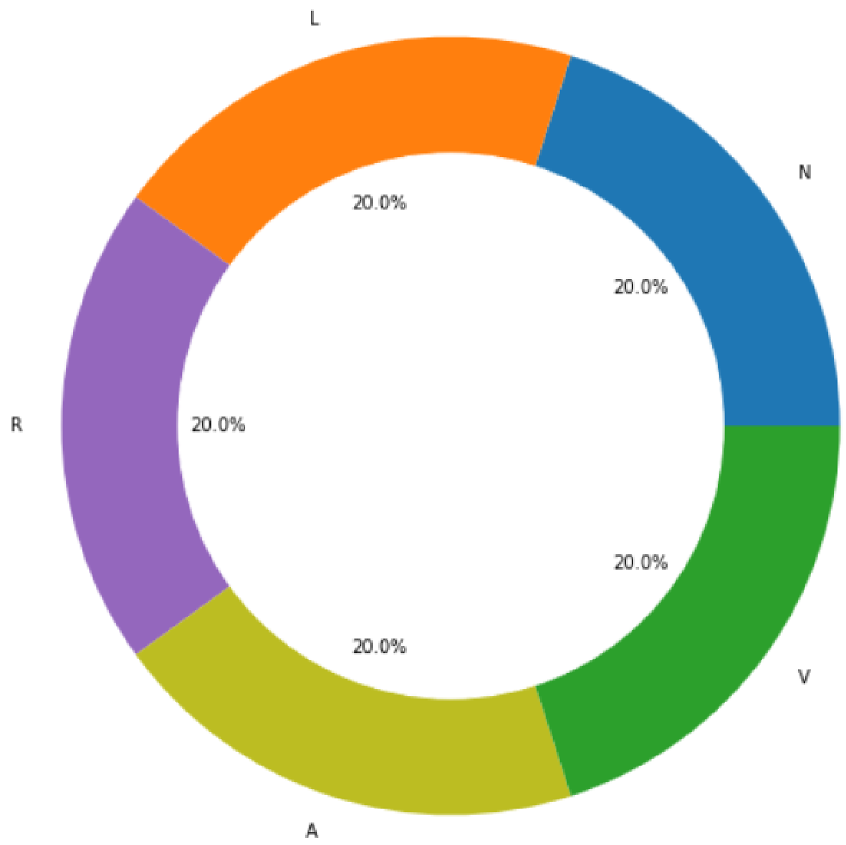
Segmented Beat

Our Solution Steps – Data Preprocessing



Classes Distribution

Our Solution Steps – Data Preprocessing



Classes Distribution after Rebalancing

Our Solution Steps – Convolutional Neural Network

- One-dimensional 12-Layer convolution with Average Pooling *(DONE)*
- Dropout Layer *(DONE)*
- Fully Connected Layer *(DONE)*
- Softmax Output Layer *(DONE)*
- Ten-Fold Cross Validation *(No Need – The used hyperparams result in great accuracy)*

Our Solution Steps – Convolutional Neural Network

```
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d_12 (Conv1D)           (None, 360, 16)           224
_________________________________________________________________
average_pooling1d_12 (Averag (None, 179, 16)           0
_________________________________________________________________
conv1d_13 (Conv1D)           (None, 179, 32)           7712
_________________________________________________________________
average_pooling1d_13 (Averag (None, 89, 32)            0
_________________________________________________________________
conv1d_14 (Conv1D)           (None, 89, 64)            34880
_________________________________________________________________
average_pooling1d_14 (Averag (None, 44, 64)            0
_________________________________________________________________
conv1d_15 (Conv1D)           (None, 44, 128)           155776
_________________________________________________________________
average_pooling1d_15 (Averag (None, 21, 128)           0
_________________________________________________________________
flatten_3 (Flatten)          (None, 2688)              0
_________________________________________________________________
dropout_3 (Dropout)          (None, 2688)              0
_________________________________________________________________
dense_6 (Dense)              (None, 35)                94115
_________________________________________________________________
dense_7 (Dense)              (None, 5)                 180
_________________________________________________________________
softmax_3 (Softmax)          (None, 5)                 0
=================================================================
Total params: 292,887
Trainable params: 292,887
Non-trainable params: 0
_________________________________________________________________
```

Results

```
Epoch 1/60
556/556 [==============================] - 26s 44ms/step - loss: 0.5887 - accuracy: 0.7835 - val_loss: 0.1397 - val_accuracy: 0.9552
Epoch 2/60
556/556 [==============================] - 24s 43ms/step - loss: 0.1478 - accuracy: 0.9541 - val_loss: 0.1036 - val_accuracy: 0.9710
Epoch 3/60
556/556 [==============================] - 24s 44ms/step - loss: 0.1164 - accuracy: 0.9663 - val_loss: 0.0907 - val_accuracy: 0.9764
Epoch 4/60
556/556 [==============================] - 24s 43ms/step - loss: 0.0878 - accuracy: 0.9751 - val_loss: 0.0978 - val_accuracy: 0.9708
Epoch 5/60
556/556 [==============================] - 24s 44ms/step - loss: 0.0794 - accuracy: 0.9780 - val_loss: 0.0749 - val_accuracy: 0.9826
Epoch 6/60
556/556 [==============================] - 24s 44ms/step - loss: 0.0753 - accuracy: 0.9796 - val_loss: 0.0757 - val_accuracy: 0.9830
Epoch 7/60
556/556 [==============================] - 24s 44ms/step - loss: 0.0702 - accuracy: 0.9824 - val_loss: 0.0819 - val_accuracy: 0.9796
Epoch 8/60
556/556 [==============================] - 24s 43ms/step - loss: 0.0632 - accuracy: 0.9841 - val_loss: 0.0766 - val_accuracy: 0.9842
Epoch 9/60
556/556 [==============================] - 25s 46ms/step - loss: 0.0526 - accuracy: 0.9888 - val_loss: 0.0658 - val_accuracy: 0.9864
Epoch 10/60
556/556 [==============================] - 24s 44ms/step - loss: 0.0546 - accuracy: 0.9875 - val_loss: 0.0673 - val_accuracy: 0.9852
Epoch 11/60
556/556 [==============================] - 24s 44ms/step - loss: 0.0508 - accuracy: 0.9879 - val_loss: 0.0685 - val_accuracy: 0.9864
Epoch 12/60
556/556 [==============================] - 24s 43ms/step - loss: 0.0471 - accuracy: 0.9902 - val_loss: 0.0782 - val_accuracy: 0.9796
Epoch 13/60
556/556 [==============================] - 24s 44ms/step - loss: 0.0485 - accuracy: 0.9887 - val_loss: 0.0560 - val_accuracy: 0.9898
Epoch 14/60
556/556 [==============================] - 25s 44ms/step - loss: 0.0407 - accuracy: 0.9920 - val_loss: 0.0660 - val_accuracy: 0.9870
Epoch 15/60
```

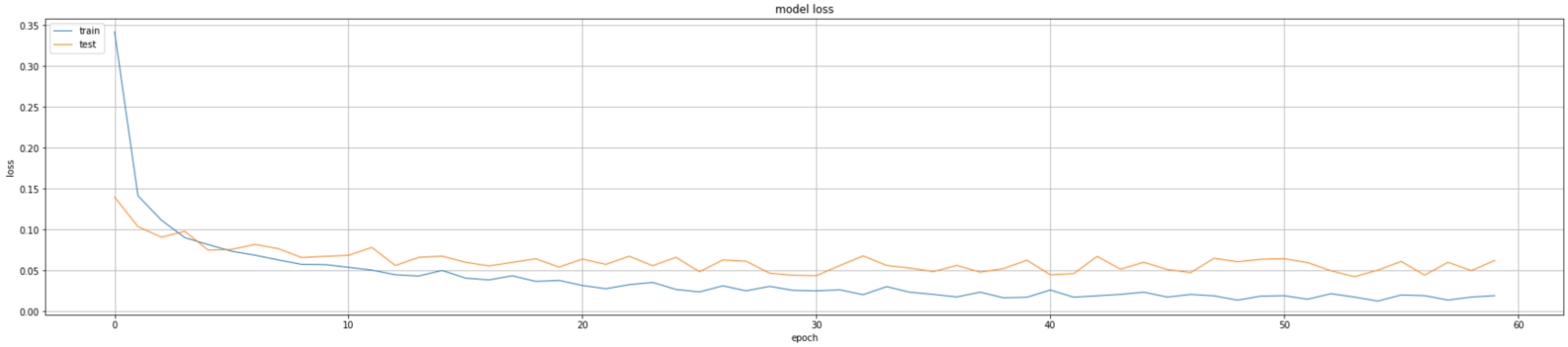
```
Epoch 16/60
556/556 [==============================] - 24s 43ms/step - loss: 0.0401 - accuracy: 0.9906 - val_loss: 0.0601 - val_accuracy: 0.9888
Epoch 17/60
556/556 [==============================] - 24s 43ms/step - loss: 0.0415 - accuracy: 0.9917 - val_loss: 0.0555 - val_accuracy: 0.9922
Epoch 18/60
556/556 [==============================] - 24s 44ms/step - loss: 0.0386 - accuracy: 0.9920 - val_loss: 0.0599 - val_accuracy: 0.9874
Epoch 19/60
556/556 [==============================] - 24s 43ms/step - loss: 0.0336 - accuracy: 0.9940 - val_loss: 0.0643 - val_accuracy: 0.9888
Epoch 20/60
556/556 [==============================] - 24s 44ms/step - loss: 0.0312 - accuracy: 0.9943 - val_loss: 0.0558 - val_accuracy: 0.9896
Epoch 25/60
556/556 [==============================] - 24s 43ms/step - loss: 0.0254 - accuracy: 0.9962 - val_loss: 0.0661 - val_accuracy: 0.9880
Epoch 26/60
556/556 [==============================] - 25s 44ms/step - loss: 0.0218 - accuracy: 0.9973 - val_loss: 0.0484 - val_accuracy: 0.9912
Epoch 27/60
556/556 [==============================] - 25s 44ms/step - loss: 0.0274 - accuracy: 0.9947 - val_loss: 0.0628 - val_accuracy: 0.9870
Epoch 28/60
556/556 [==============================] - 25s 44ms/step - loss: 0.0297 - accuracy: 0.9952 - val_loss: 0.0613 - val_accuracy: 0.9908
Epoch 29/60
556/556 [==============================] - 24s 44ms/step - loss: 0.0270 - accuracy: 0.9950 - val_loss: 0.0466 - val_accuracy: 0.9916
Epoch 30/60
556/556 [==============================] - 25s 44ms/step - loss: 0.0256 - accuracy: 0.9947 - val_loss: 0.0440 - val_accuracy: 0.9924
```

Results

```
Epoch 31/60
556/556 [==============================] - 25s 44ms/step - loss: 0.0258 - accuracy: 0.9954 - val_loss: 0.0436 - val_accuracy: 0.9926
Epoch 32/60
556/556 [==============================] - 25s 44ms/step - loss: 0.0264 - accuracy: 0.9952 - val_loss: 0.0560 - val_accuracy: 0.9904
Epoch 33/60
556/556 [==============================] - 24s 44ms/step - loss: 0.0206 - accuracy: 0.9965 - val_loss: 0.0678 - val_accuracy: 0.9888
Epoch 34/60
556/556 [==============================] - 24s 43ms/step - loss: 0.0287 - accuracy: 0.9940 - val_loss: 0.0561 - val_accuracy: 0.9906
Epoch 35/60
556/556 [==============================] - 24s 44ms/step - loss: 0.0272 - accuracy: 0.9959 - val_loss: 0.0529 - val_accuracy: 0.9920
Epoch 36/60
556/556 [==============================] - 24s 43ms/step - loss: 0.0203 - accuracy: 0.9964 - val_loss: 0.0486 - val_accuracy: 0.9918
Epoch 37/60
556/556 [==============================] - 24s 43ms/step - loss: 0.0181 - accuracy: 0.9972 - val_loss: 0.0562 - val_accuracy: 0.9936
Epoch 38/60
556/556 [==============================] - 24s 43ms/step - loss: 0.0204 - accuracy: 0.9961 - val_loss: 0.0479 - val_accuracy: 0.9910
Epoch 39/60
556/556 [==============================] - 24s 44ms/step - loss: 0.0199 - accuracy: 0.9967 - val_loss: 0.0523 - val_accuracy: 0.9924
Epoch 40/60
556/556 [==============================] - 24s 44ms/step - loss: 0.0179 - accuracy: 0.9969 - val_loss: 0.0626 - val_accuracy: 0.9914
Epoch 41/60
556/556 [==============================] - 24s 44ms/step - loss: 0.0314 - accuracy: 0.9950 - val_loss: 0.0447 - val_accuracy: 0.9938
Epoch 42/60
556/556 [==============================] - 24s 43ms/step - loss: 0.0155 - accuracy: 0.9978 - val_loss: 0.0461 - val_accuracy: 0.9920
Epoch 43/60
556/556 [==============================] - 24s 43ms/step - loss: 0.0156 - accuracy: 0.9978 - val_loss: 0.0672 - val_accuracy: 0.9904
Epoch 44/60
556/556 [==============================] - 24s 44ms/step - loss: 0.0223 - accuracy: 0.9964 - val_loss: 0.0516 - val_accuracy: 0.9928
Epoch 45/60
```

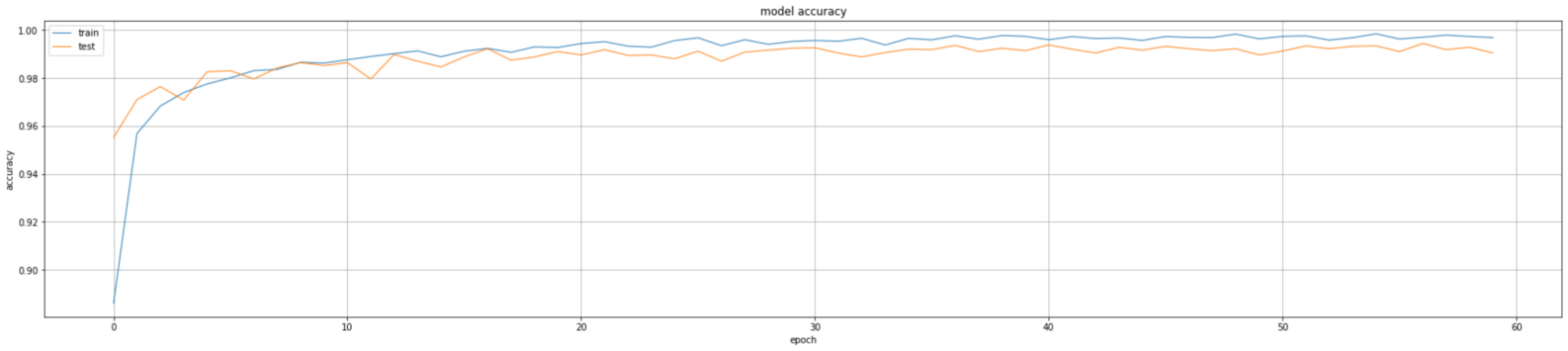
```
Epoch 46/60
556/556 [==============================] - 25s 45ms/step - loss: 0.0206 - accuracy: 0.9963 - val_loss: 0.0511 - val_accuracy: 0.9932
Epoch 47/60
556/556 [==============================] - 25s 44ms/step - loss: 0.0184 - accuracy: 0.9975 - val_loss: 0.0473 - val_accuracy: 0.9922
Epoch 48/60
556/556 [==============================] - 25s 44ms/step - loss: 0.0223 - accuracy: 0.9956 - val_loss: 0.0649 - val_accuracy: 0.9914
Epoch 49/60
556/556 [==============================] - 24s 44ms/step - loss: 0.0140 - accuracy: 0.9986 - val_loss: 0.0607 - val_accuracy: 0.9922
Epoch 50/60
556/556 [==============================] - 25s 44ms/step - loss: 0.0136 - accuracy: 0.9980 - val_loss: 0.0636 - val_accuracy: 0.9896
Epoch 51/60
556/556 [==============================] - 25s 44ms/step - loss: 0.0231 - accuracy: 0.9961 - val_loss: 0.0643 - val_accuracy: 0.9912
Epoch 52/60
556/556 [==============================] - 24s 44ms/step - loss: 0.0141 - accuracy: 0.9978 - val_loss: 0.0596 - val_accuracy: 0.9934
Epoch 53/60
556/556 [==============================] - 24s 43ms/step - loss: 0.0253 - accuracy: 0.9953 - val_loss: 0.0495 - val_accuracy: 0.9922
Epoch 54/60
556/556 [==============================] - 24s 44ms/step - loss: 0.0190 - accuracy: 0.9968 - val_loss: 0.0424 - val_accuracy: 0.9932
Epoch 55/60
556/556 [==============================] - 24s 44ms/step - loss: 0.0125 - accuracy: 0.9985 - val_loss: 0.0503 - val_accuracy: 0.9934
Epoch 56/60
556/556 [==============================] - 24s 44ms/step - loss: 0.0197 - accuracy: 0.9966 - val_loss: 0.0611 - val_accuracy: 0.9910
Epoch 57/60
556/556 [==============================] - 24s 43ms/step - loss: 0.0184 - accuracy: 0.9978 - val_loss: 0.0442 - val_accuracy: 0.9944
Epoch 58/60
556/556 [==============================] - 24s 44ms/step - loss: 0.0148 - accuracy: 0.9979 - val_loss: 0.0600 - val_accuracy: 0.9918
Epoch 59/60
556/556 [==============================] - 24s 44ms/step - loss: 0.0159 - accuracy: 0.9973 - val_loss: 0.0498 - val_accuracy: 0.9928
Epoch 60/60
556/556 [==============================] - 24s 44ms/step - loss: 0.0195 - accuracy: 0.9970 - val_loss: 0.0623 - val_accuracy: 0.9904
```

Results



Model Loss

Results



Model Accuracy

Results

Finding the loss and accuracy of the model

```
score = model.evaluate(test_x, test_y)

print('Test Loss:', score[0])
print('Test accuracy:', score[1])
```

```
157/157 [==============================] - 2s 11ms/step - loss: 0.0623 - accuracy: 0.9904
Test Loss: 0.06230679154396057
Test accuracy: 0.9904000163078308
```

Model Loss &
Accuracy