Steve Su, Kristian Montoya

ECG Classification using RNN and CNN

Video link: part1: https://www.dropbox.com/s/sion3hr893j07xv/video_report_part_1.mp4?dl=0

part2: https://www.dropbox.com/s/9hrp8myvvr83gow/video_report_part_2.mp4?dl=0

github: https://github.com/steve303/heartECG.git

Objective

Our project topic covers the analysis of ECG's (electrocardiogram) of patient heartbeats to classify heart conditions. There are five classification categories. Zero denotes normal case and one through four denotes a certain type of heart abnormality. Each instance within our dataset represents an "image" of a patient's ECG over a 187 millisecond time interval where each millisecond is represented with a normalized data point that captures the ECG value at that given point in time. Our first objective is to use/test several classification models to find which models provide the highest overall accuracy. The study which we obtained the data used a CNN (convoluted neural network) model https://arxiv.org/pdf/1805.00794.pdf, however we wanted to know if a RNN (recurrent neural network) could produce better results since the data is time based. We also want to see if we can reproduce the original study's results with CNN.

Our second objective was to evaluate other techniques which could improve performance of our neural network models to address unbalanced data sets when there are few samples of certain categories. We experimented with data augmentation and a two stage NN (neural network) architecture. Regarding the two stage NN, one stage was responsible for just classifying whether the ECG signal was normal or abnormal. This model and its weights were saved and frozen then fed into another NN which then classified the specific heart abnormality, categories one through four. Through specialization of the tasks we hoped to optimize the NN design. Through data augmentation our goal was to balance the data set by increasing sample size of low frequency categories. This was done by injecting noise into the original signal to give additional data to train on.

Dataset Description

As stated above, each instance within our dataset represented a single recorded ECG signal. The training and validation data consisted of 87,553 and 21892 observations, respectively. The individual data points of each instance were normalized values between 0 & 1. These represented the feature values which totaled 187. Most signals did not have a value for each of the 187 features and was padded with zeros. A sample of the ECG signal is shown below in Figure 1. The data set was obtained through Kaggle web site https://www.kaggle.com/shayanfazeli/heartbeat.

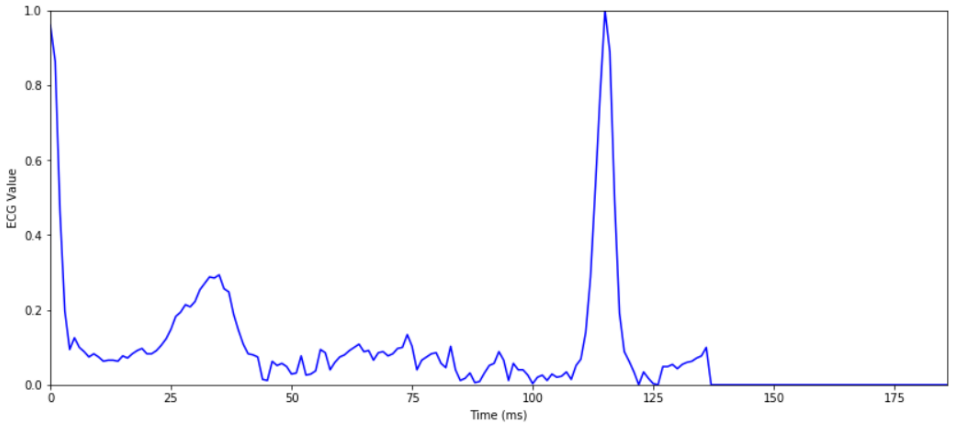
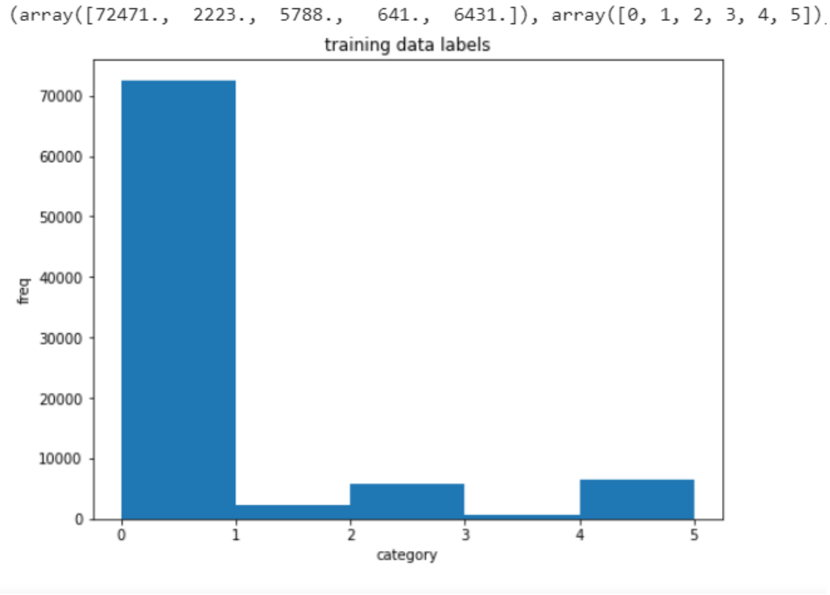


Fig. 1 Sample of ECG signal

The histogram below shows the training set labels, categories 0-5. There are a much larger number of normal samples compared to the abnormal samples. In the training set there were over 70,000 observations for the normal category and as few as 641 in the abnormal category. This can cause problems with ANN algorithms since it cannot train as well on these seldom seen samples. Categories two and three have a particularly a low count being less than 2.5% of the total.



```
(array([72471.,  2223.,  5788.,   641.,  6431.]), array([0, 1, 2, 3, 4, 5]))
```

Fig. 2 Histogram of training data labels

EDA/preprocessing

Figure 3 shows the average signal of the ECG plot for each of the five categories. Figure 4 shows sample signals of the individual plot overlaid with the average signal for its category. There seemed to be a wide variety in signal characteristics for some categories as the individual plot did not look similar to the average. Some preprocessing was done for the RNN model and will be discussed in that section of the report.

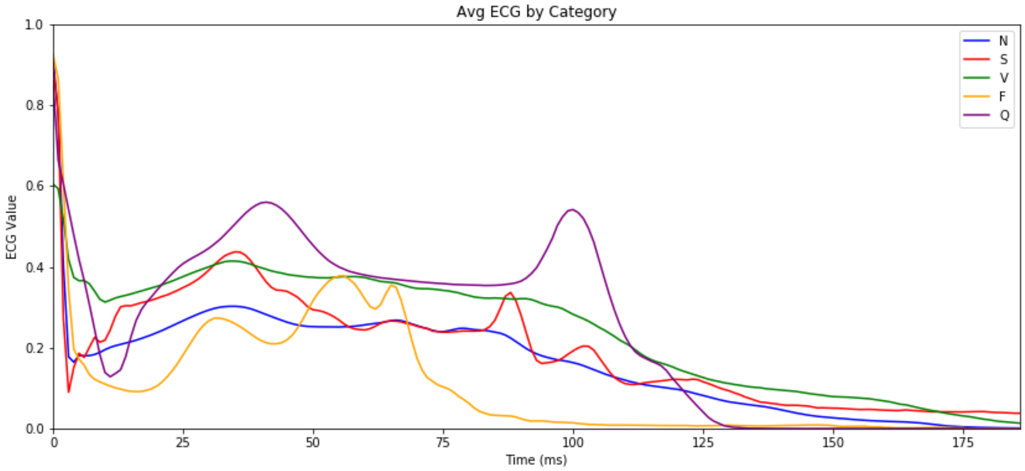


Fig 3. Average signal of each category N-0, S-1, V-2, F-3, Q-4

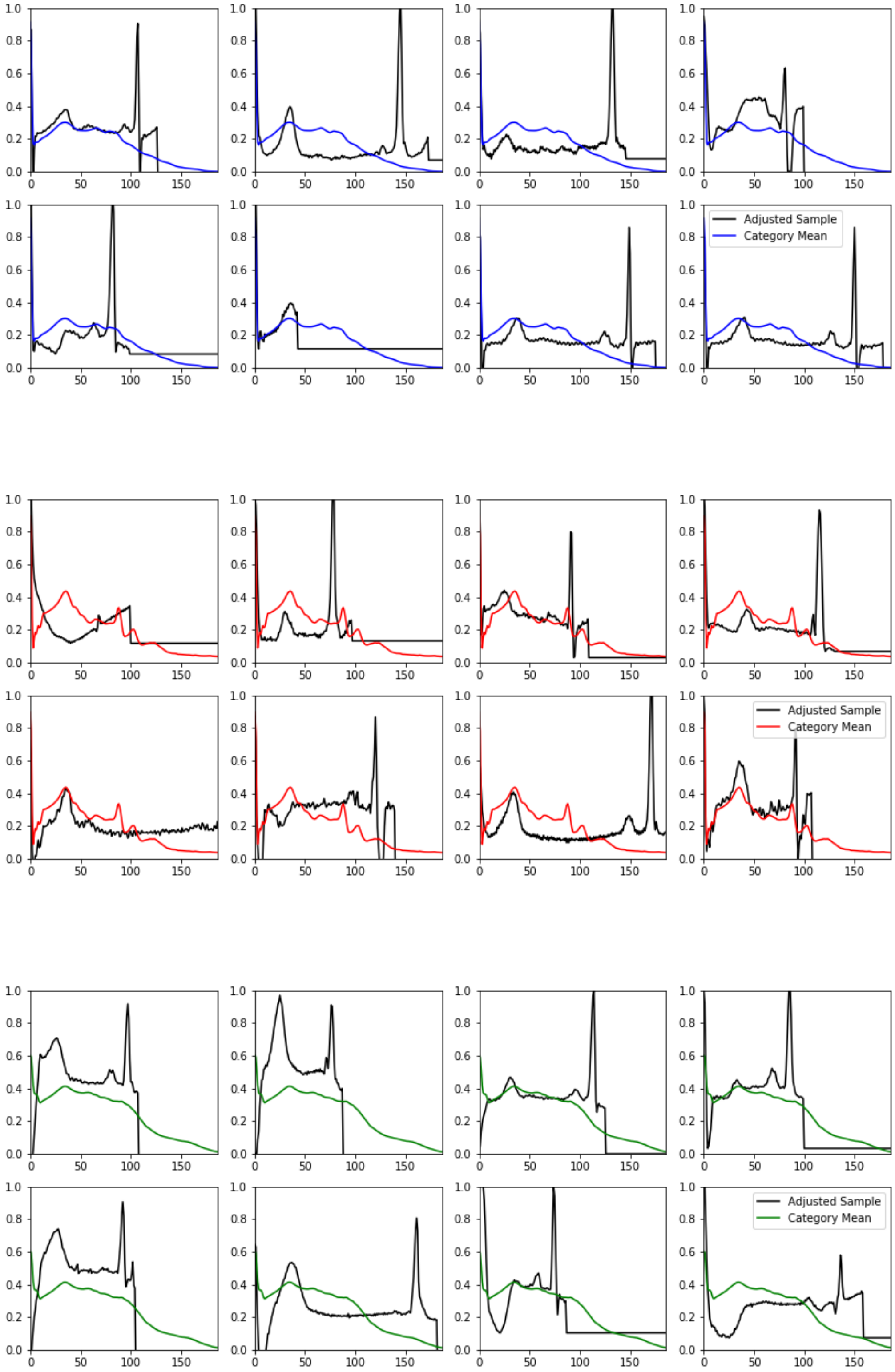


Fig 4. Sample ECG plots of categories 0, 1, and 2 (starting from the top)

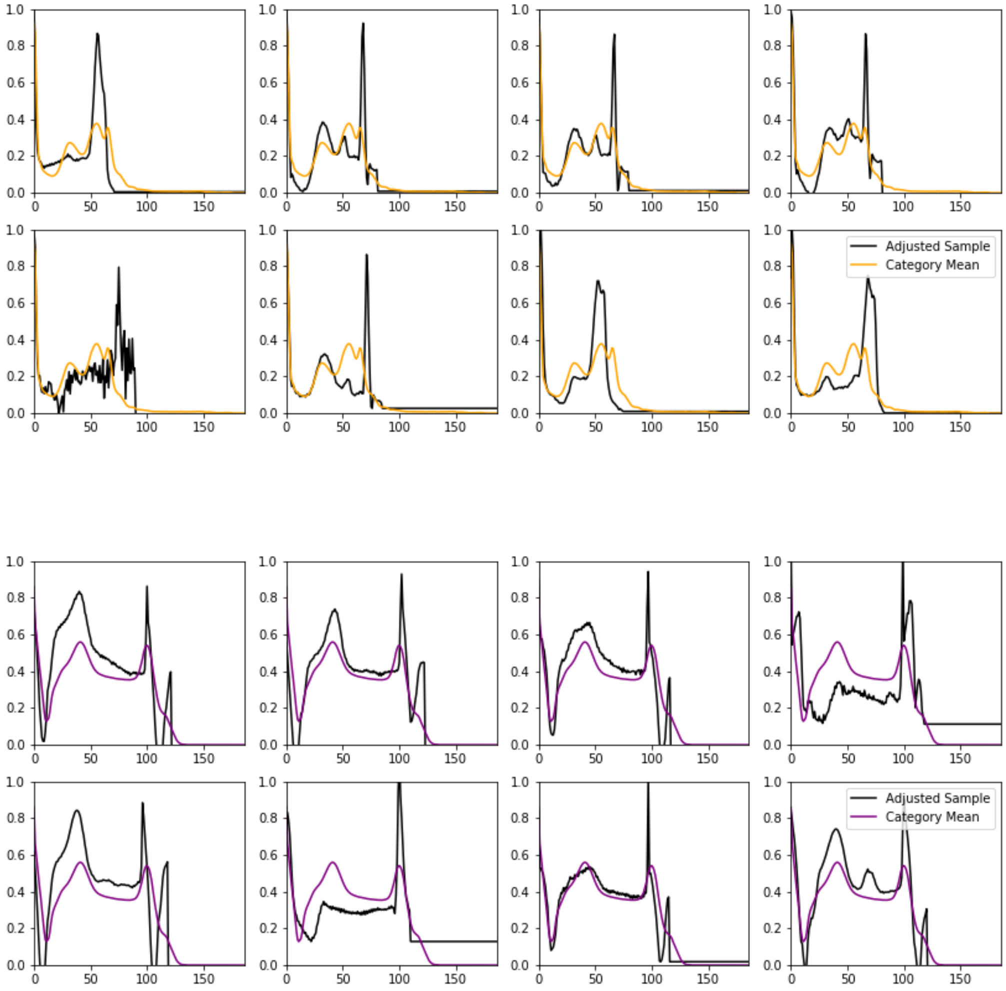
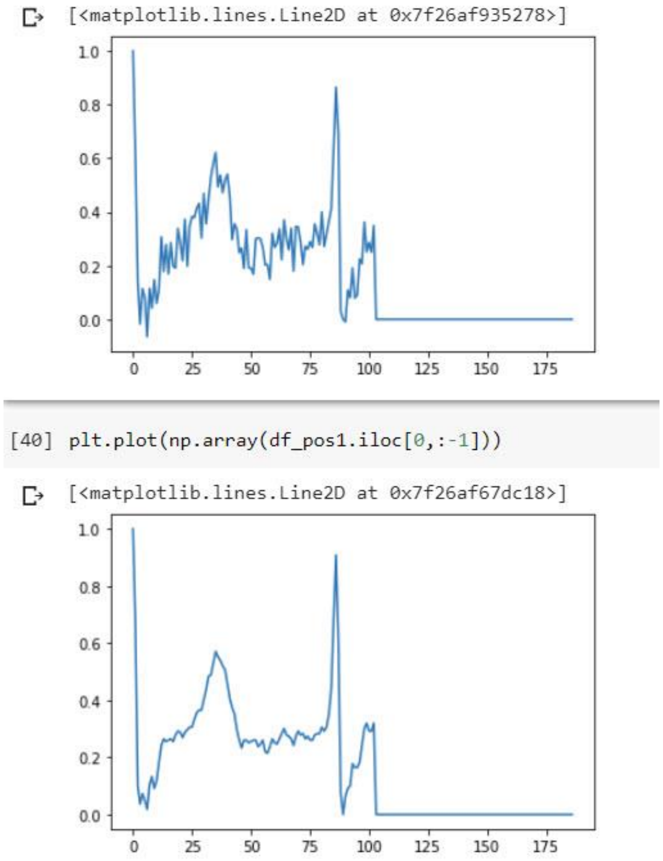


Fig 4 Continued. Sample ECG plots of categories 3 and 4

Data Augmentation Description

To improve the balance of data between normal and abnormal categories we augmented the abnormal data sets by injecting noise into its data. The normal training data set had up to 70,000 samples while the abnormal samples had a low of 641samples. A function called createnoise() was created to perform this task. From the four abnormal categories this function randomly injected a certain amount of noise into the signal. The figure below shows one signal with max noise at 10%. However, for the study we chose to use 6% as to not distort the signal too much. In the end, we balanced the data set so that each abnormal category would have a sample size of n=10,000. Furthermore, when we fit the data to our model, we only included 10,000 samples of the normal category instead of its 70,000 observations. Thus, for this new data set the total sample size was 50,000 for the five total categories, 0-5. Data augmentation was only performed in the training data set. The validation data set was left untouched.



```
[<matplotlib.lines.Line2D at 0x7f26af935278>]
```

```
[40] plt.plot(np.array(df_pos1.iloc[0,:-1]))
```

```
[<matplotlib.lines.Line2D at 0x7f26af67dc18>]
```

Fig 5. ECG signal, upper plot is with data augmentation, lower plot is without data augmentation

RNN Model

Since heart ECG signals are time based we wanted to test whether we could get as good or better results than a CNN model which the authors used. The advantage of a RNN is that they are designed to remember past information when making a decision at the current time step. Furthermore in a RNN, the input data is fed sequentially with respect to time whereas a CNN is based on spatial locality. A simple RNN model was created using LSTM (Long short term memory) units with tanh activation function. We chose this unit versus a generic RNN unit because it handles exploding or diminishing gradients well.

In order to use the LSTM model the input data (float data type) had to be preprocessed into discrete values. The original ECG signal ranged from 0.0 to 1.0 and was broken into 64 bins. It was important to provide enough resolution but keep number of bins to a minimum for computation efficiency. Sixty-four bins seemed adequate as the non-digitized and digitized plots looked essentially the same.

The baseline model consisted of a LSTM layer with 32 outputs. This was then fed into a dense layer of 5 outputs, one for each category. The activation function was softmax. The learning rate was set to 0.0005 using Adam optimizer. Through optimization process this seemed to be a good balance for speed and accuracy (figure 6). Categorical cross entropy was chosen as the loss function. We will refer to this model as the baseline when comparing to other RNN models.

The cross entropy loss and accuracy were 0.183 and 0.948, respectively. The confusion matrix and precision/recall tables are presented in figure 7. One will notice that the recall numbers for categories one and three are particularly low, 0.25 and 0.50 respectively. At best, the model cannot correctly predict the total number of these cases greater than 50% of the time for these categories. The recall for categories two and four were better, 0.81 and 0.90 respectively. The recall for the normal case, category zero, had the best recall at 0.99. However, as mentioned earlier there were many more samples in the normal category compared to the abnormal categories in the training data set. This can cause performance problems because the neural network does not see the abnormal samples as frequently and can not learn as well for these less seen categories.

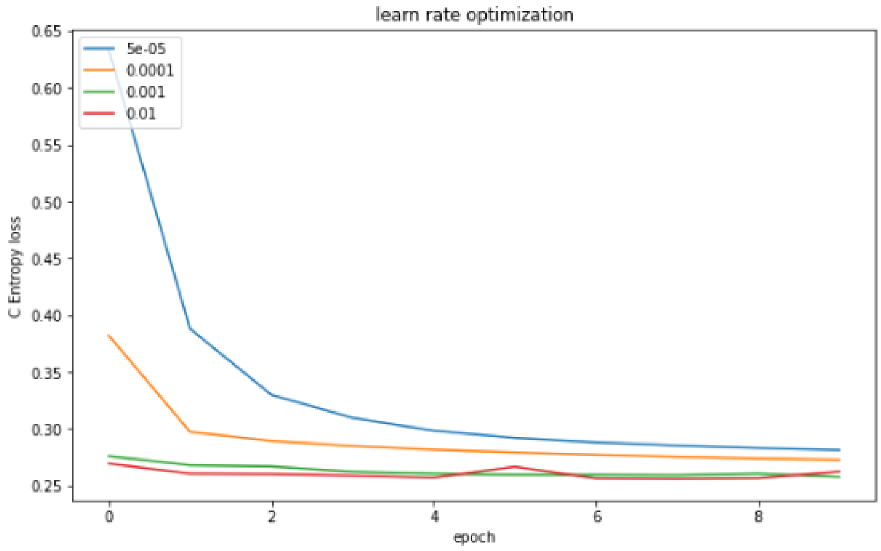
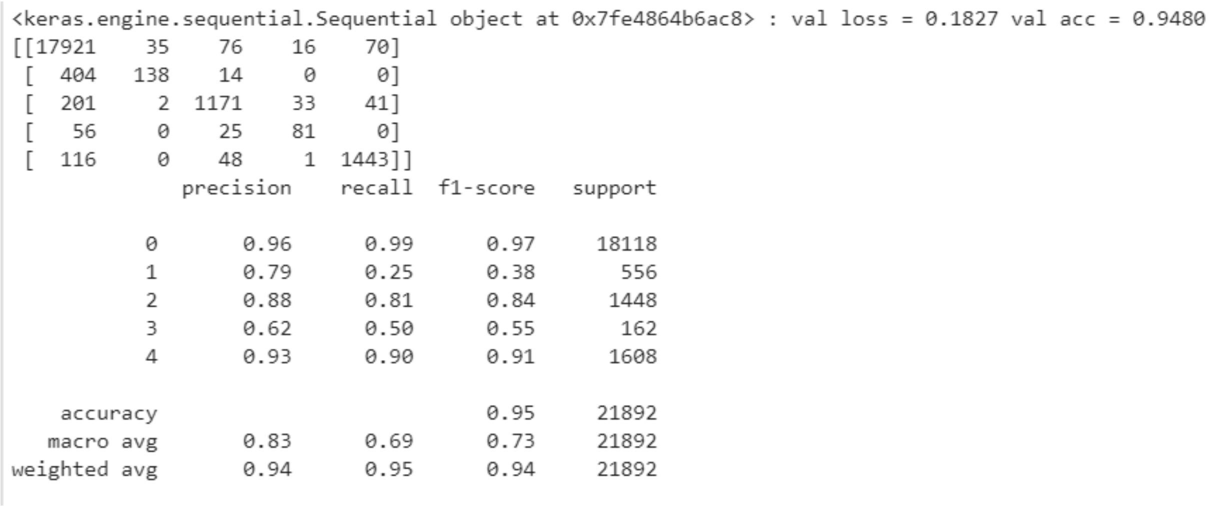


Fig 6. Learning rate optimization RNN model

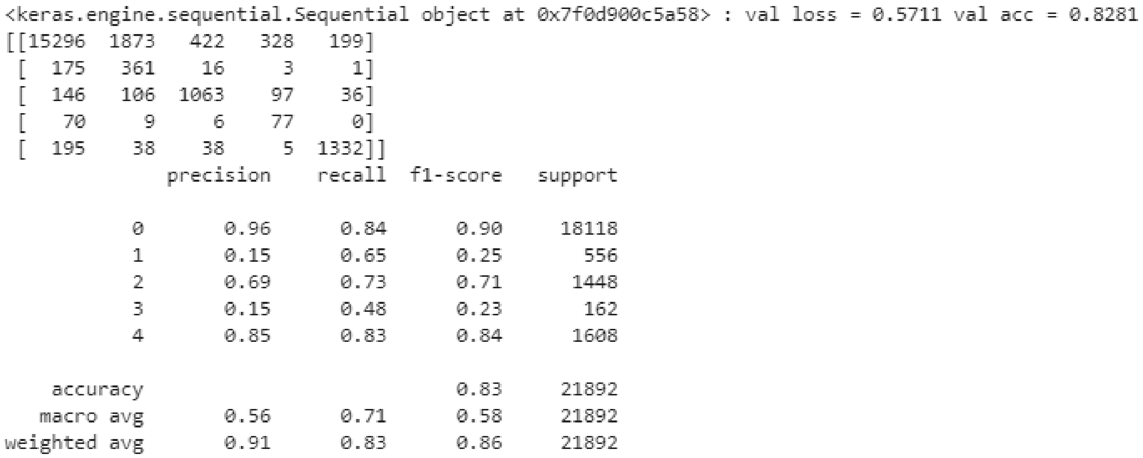


```
<keras.engine.sequential.Sequential object at 0x7fe4864b6ac8> : val loss = 0.1827 val acc = 0.9480
[[17921    35    76    16    70]
 [  404   138    14     0     0]
 [  201     2  1171    33    41]
 [   56     0    25    81     0]
 [  116     0    48     1  1443]]
              precision    recall  f1-score   support

           0       0.96      0.99      0.97     18118
           1       0.79      0.25      0.38       556
           2       0.88      0.81      0.84      1448
           3       0.62      0.50      0.55       162
           4       0.93      0.90      0.91      1608

    accuracy                           0.95     21892
   macro avg       0.83      0.69      0.73     21892
weighted avg       0.94      0.95      0.94     21892
```

Fig 7. Confusion matrix and precision/recall scores of baseline LSTM model

To address this data imbalance, we supplemented data for the abnormal categories with data augmentation, explained earlier. Now each category , zero through four, each have 10,000 data points for a total of 50,000 in this new data set. The same baseline LSTM model as above was run with this balanced data set for comparison. The same validation data set was used to create the confusion matrix and precision/recall table shown below, figure8.

The results were mixed. The overall loss and accuracy were, 0.571 and 0.828 respectively. These results were worse compared to the non-augmented data set, however the recall for category one improved from 0.25 to 0.65. There seemed to be a trade off for this improvement as categories zero, two, and four got worse, a drop of between 7 and 15% in recall. Category three was less affected between the non-augmented and augmented trials, 4% difference. Figure 9 summaries these comparisons. The training using this data set took quite a long time and perhaps with additional optimization and architecture changes the results could be improved.



```
<keras.engine.sequential.Sequential object at 0x7f0d900c5a58> : val loss = 0.5711 val acc = 0.8281
[[15296  1873   422   328   199]
 [  175   361    16     3     1]
 [  146   106  1063    97    36]
 [   70     9     6    77     0]
 [  195    38    38     5  1332]]
              precision    recall  f1-score   support

           0       0.96      0.84      0.90     18118
           1       0.15      0.65      0.25       556
           2       0.69      0.73      0.71      1448
           3       0.15      0.48      0.23       162
           4       0.85      0.83      0.84      1608

    accuracy                           0.83     21892
   macro avg       0.56      0.71      0.58     21892
weighted avg       0.91      0.83      0.86     21892
```

Fig 8. Confusion matrix and precision/recall scores of trained LSTM model using augmented data

	non-augmented	augmented	%difference
category	recall	recall	
0	0.99	0.84	-15.15%
1	0.25	0.65	160.00%
2	0.81	0.73	-9.88%
3	0.5	0.48	-4.00%
4	0.9	0.84	-6.67%

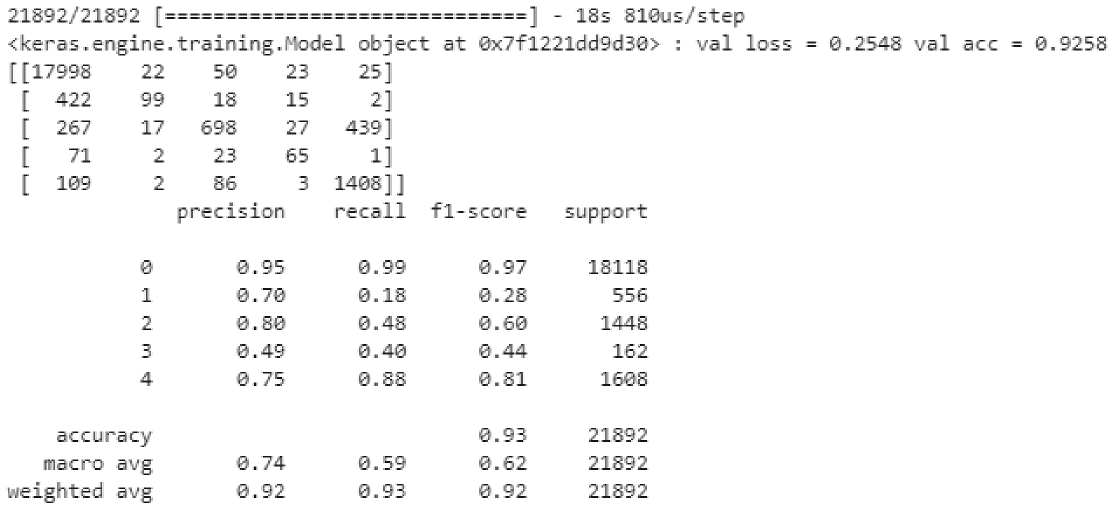
Fig 9. Comparison of recall scores LSTM model with and without data augmentation. Note, the same validation data set was used in this comparison.

Our second attempt to improve the recall results used a two stage architecture. The goal was to break up the tasks for each stage. The first stage classified whether the ECG was abnormal or not and the second stage determined the exact category. The first stage consisted of a RNN with LSTM units (tanh activation) of 32 outputs followed by a dense layer (relu activation) with 16 outputs and final dense layer with 2 outputs with softmax activation to determine the normal or not normal label. Again, Adam optimizer and cross entropy loss were chosen for the model. Additionally the labels of the training data had to be changed to 0 or 1 for normal or not normal.

The first stage was run independently of the second stage. After many epochs of training data the loss and accuracy settled at 0.153 and 0.954, respectively. After training was complete these weights were saved and frozen when run in conjunction with the second stage. In order to join with the second stage the last layer with softmax activation was stripped along with its weights. The second stage will have the final classification units.

The second stage of the model consisted of a dense layer (relu activation) with 16 outputs followed by another dense layer with 5 outputs to determine the classification with softmax activation. The Keras Model API was required to make this type of construction. To freeze the weights of the first stage network the trainable parameter was set to false and confirmed by the number of non-trainable params output. When it was time to run the model, the original (non-augmented) training data (labels 0-4), was used. With the pre-loaded weights the time to run was much shorter.

The model did not perform as well as our original baseline LSTM model. The loss and accuracy were 0.255 and 0.926 respectively, whereas the original LSTM model had a loss and accuracy of 0.183 and 0.948 respectively. The recall scores for the abnormal categories did not do so well either (figure 10). All scores were between 2 and 40% lower than the original baseline LSTM model. See figure 11 for comparisons. One explanation why this model did not do so well is that the first stage did not do a good enough job classifying between normal and not normal so these errors were carried over to the second stage. Improving the first stage is one avenue to improve the overall model. Additionally, adding more flexibility to the second stage could help make up for the short comings of the first stage. Adding additional layers or neurons could help address this.



```
21892/21892 [==============================] - 18s 810us/step
<keras.engine.training.Model object at 0x7f1221dd9d30> : val loss = 0.2548 val acc = 0.9258
[[17998    22    50    23    25]
 [  422    99    18    15     2]
 [  267    17   698    27   439]
 [   71     2    23    65     1]
 [  109     2    86     3  1408]]
              precision    recall  f1-score   support

           0       0.95      0.99      0.97     18118
           1       0.70      0.18      0.28       556
           2       0.80      0.48      0.60      1448
           3       0.49      0.40      0.44       162
           4       0.75      0.88      0.81      1608

    accuracy                           0.93     21892
   macro avg       0.74      0.59      0.62     21892
weighted avg       0.92      0.93      0.92     21892
```

Fig. 10 Confusion matrix and precision/recall scores of the two stage LSTM model

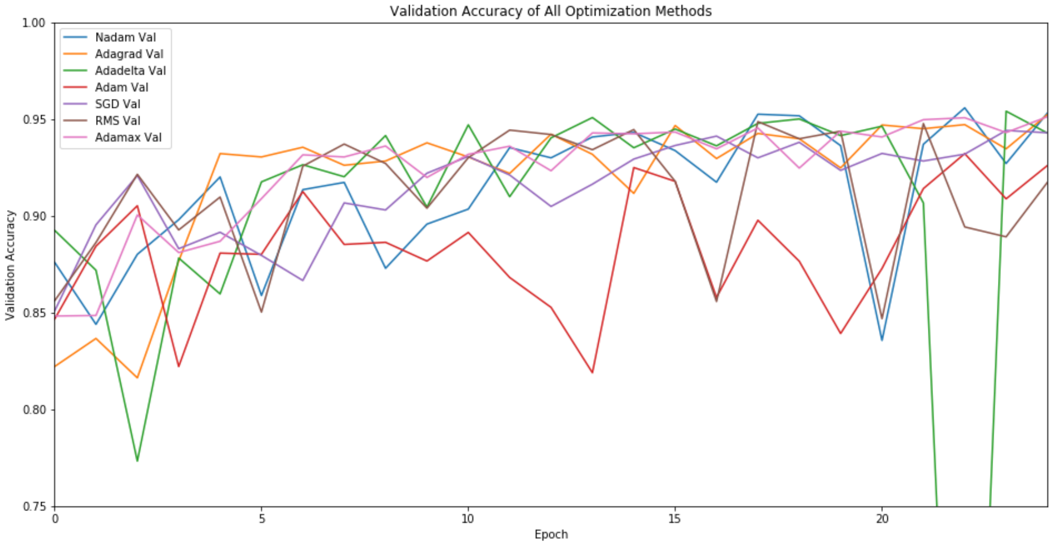
category	LSTM baseline recall	2 stage LSTM recall	%difference
0	0.99	0.99	0.00%
1	0.25	0.18	-28.00%
2	0.81	0.48	-40.74%
3	0.5	0.4	-20.00%
4	0.9	0.88	-2.22%

Fig. 11 Comparison of recall scores between baseline LSTM and two stage model

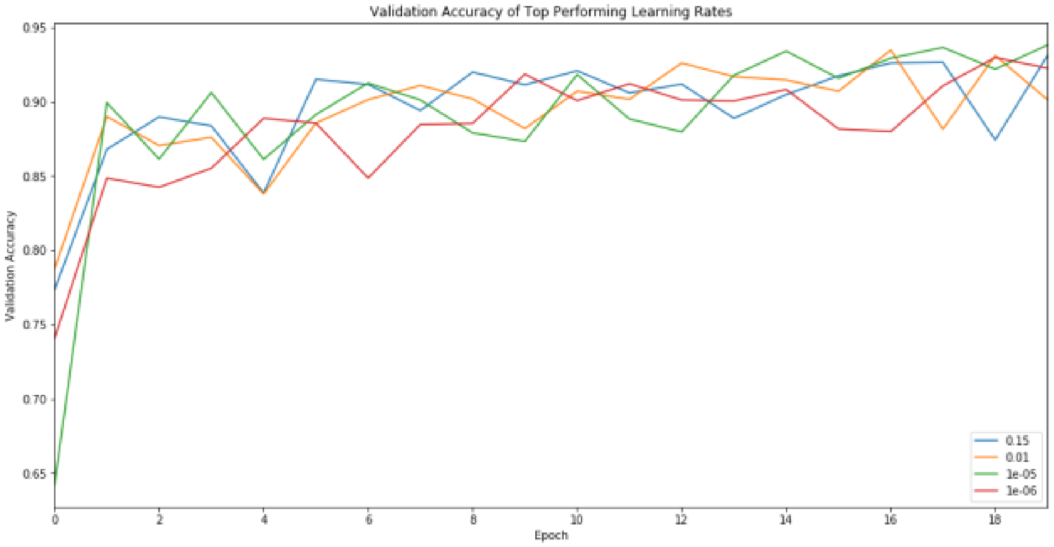
CNN Model

The architecture of our CNN is pretty straight forward: Two 1-D convolution layers, a 1-D pooling layer, and two dense layers (we also normalize the inputs between each layers). We experimented with other archetypes/layouts but found that this gave the best results of all our attempts.

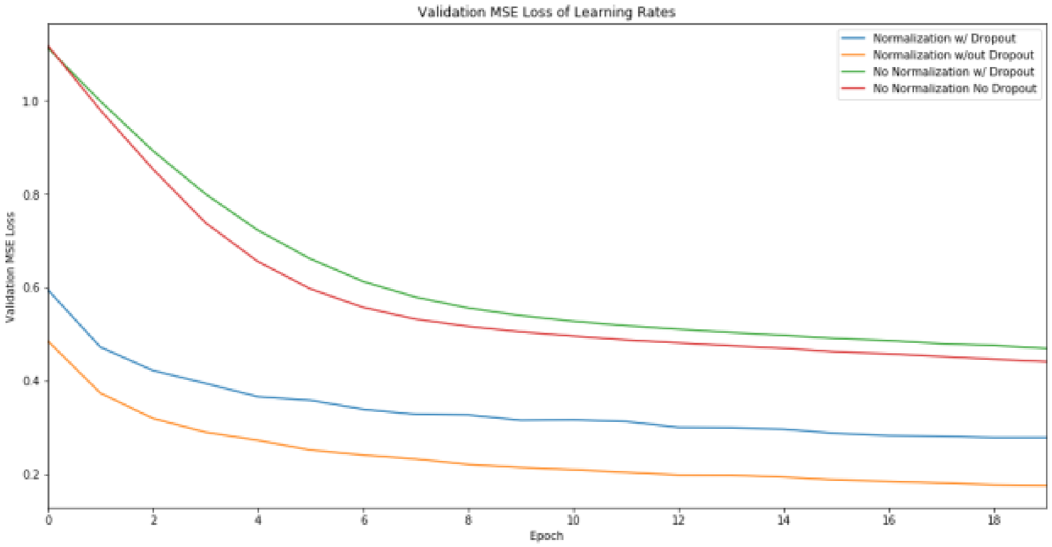
Once we established the layout for the CNN, we set out to find the optimal optimizing method. We tested the following methods: Nadam, Adagrad, Adadelta, Adam, SGD, RMS, and Adamax.

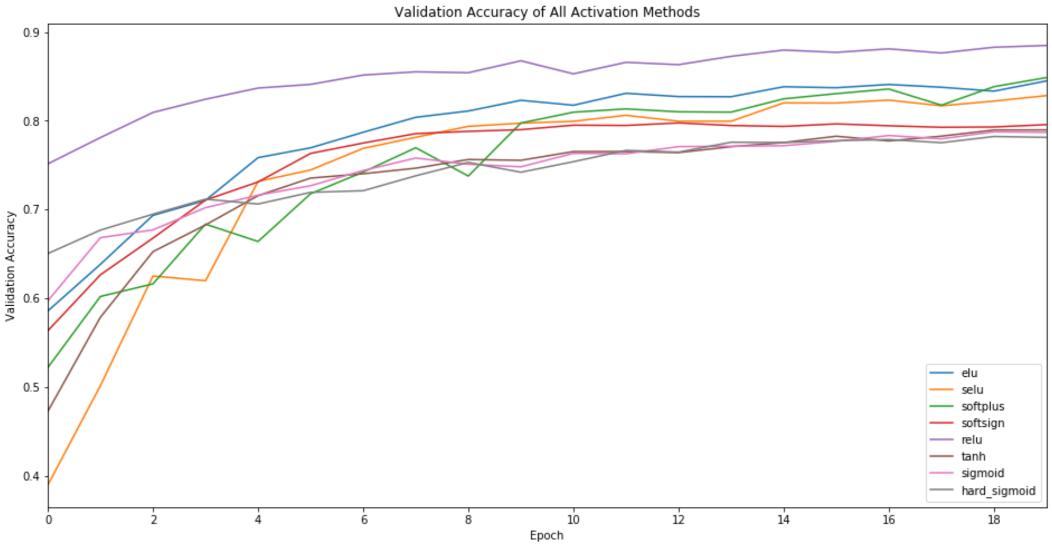


Of all the tested methods, Adamax gave the best performance (with the highest accuracy, lowest loss, and most stable learning session). From there, we looked towards the model's learning rate, activation method, and whether or not to include our normalizing/dropout layers in our model in an attempt to maximize the model's performance. After all our tests, we found that a learning rate of 1e-5, RELU activation method, inclusion of normalizing layers, and exclusion of the dropout layers gave the best performance. The graphics for all of this are included below.



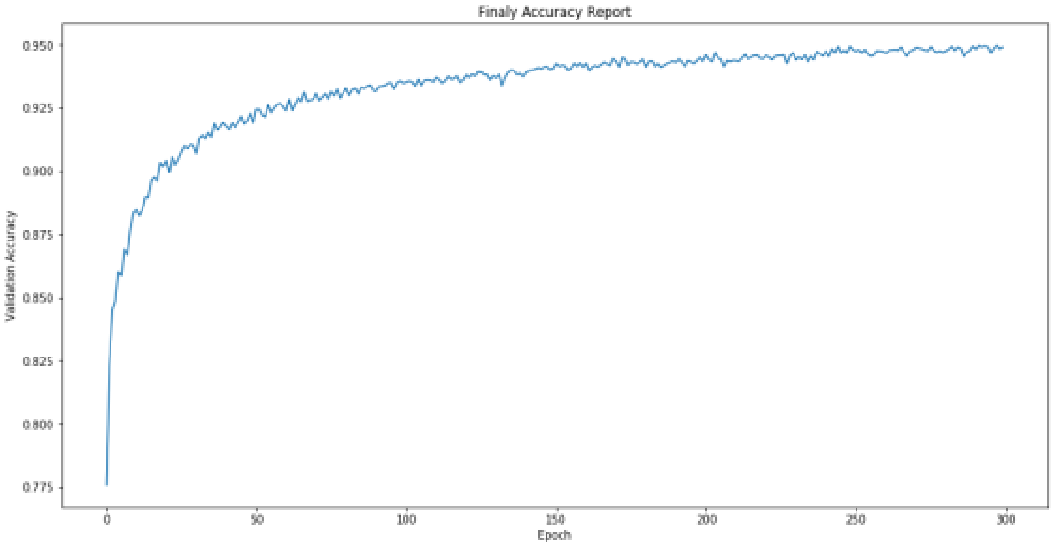
Steve Su, Kristian Montoya

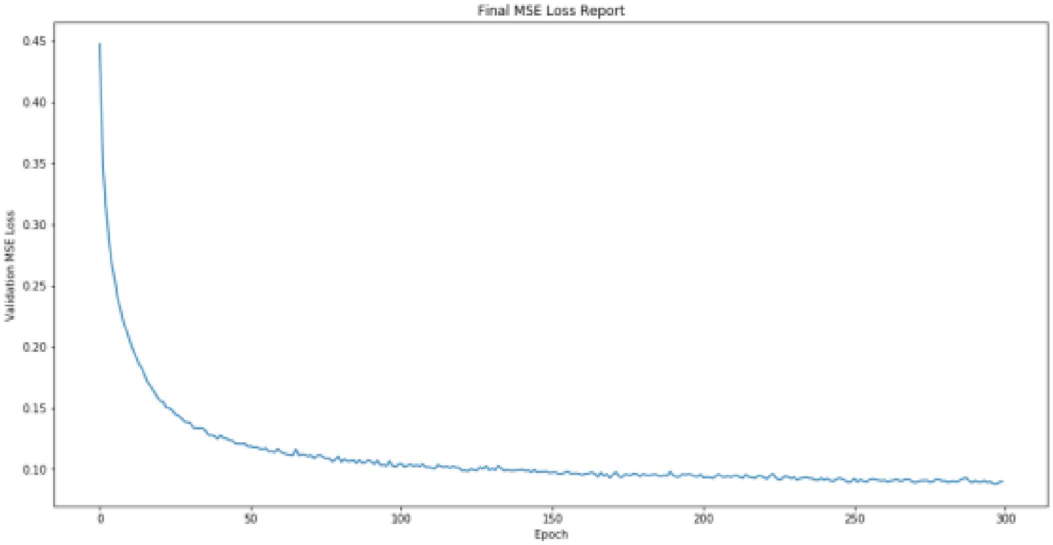




Steve Su, Kristian Montoya

Final CNN results: Validation accuracy: 95%, MSE Loss: 0.04





Summary

The two techniques we tried to improve recall performance with RNN model had mixed results. The data augmentation technique helped in classification of category one and boosted its recall score by 160% but at a cost to the scores of the other categories. However, with additional optimization and architecture changes this could be improved. Additionally, the percent noise injected in the data during the augmentation process could also be influential. In this study it remained constant at 6%.

The second technique using two stage architecture had less promising results. The model never achieved an accuracy greater than 93% and it not improve in any of the recall scores. It is hypothesized that the first stage did not do well enough classifying normal or not normal samples and that these errors carried over to the second stage. The model used in this study was purposely designed to be simple to get results quickly, but perhaps more flexibility is needed to improve performance.

Our baseline RNN/LSTM and CNN models, both performed similarly with accuracy results of nearly 95%. The authors of the original study also had accuracy results of 95% but their recall scores were much better, in the 0.90s.