Mini Project 1 - Arrhythmia Detection and Classification
Final Report

Blake Downey and Karthik Kumar

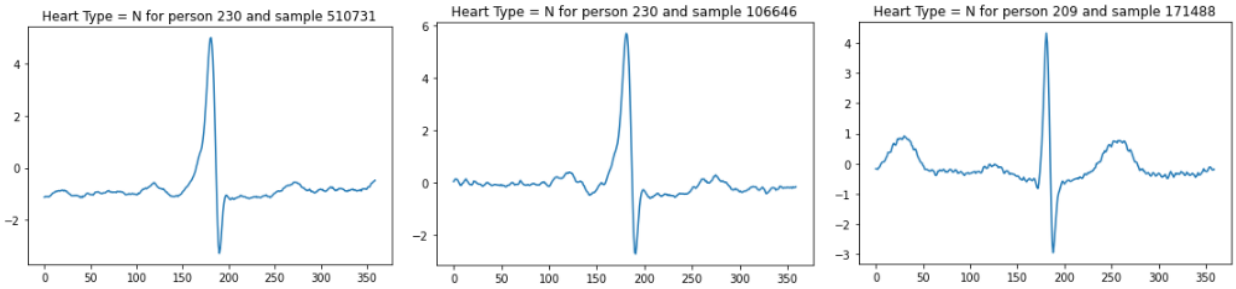
Team Member Contributions
Preprocessing: Blake and Karthik
AE: Blake
VAE: Karthik
Random Forest and Dense Net: Blake
CNN: Karthik
Addition of another feature: Karthik
Kaggle Normalization: Blake

Introduction
Arrhythmia, or irregular heart beat, detection is the topic of this mini project. In this assignment we were given time series data files from 44 patients, where 42 of them have 'MLII' measurements taken. To get this data into shape we have a few preprocessing functions that generate an nxd dataframe, where n is the heartbeat samples we extracted and d is the features, or the MLII values at each time step. We also make use of an autoencoder and a variational autoencoder to generate more samples of classes with less to adjust the class imbalance that is very present in the dataset. Along with this, we also used noise removal algorithms based on wavelets to clean the signals, and finally feeding the data to a few different ML and NN models.

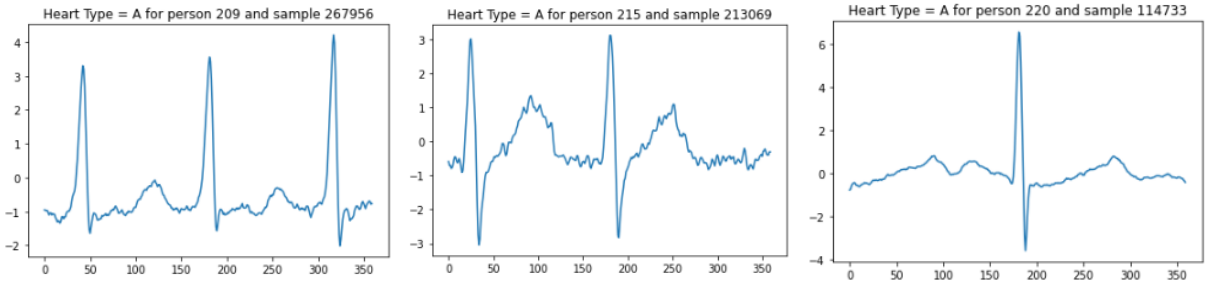
Answer 1) (30 points)
Answer 2) (5 points + 5 bonus marks)
Preprocessing
The data was given to us in 44 different csv files with annotation files to go along. Our task was to parse the annotations file and extract where the R peaks of each heartbeat were for each patient, and then take 180 samples to the left of the peak and 179 samples to the right of the peak. By doing this, we generate one sample of a heartbeat with a label of one of the valid classes (A, L, N, R, U, or V). In total, we extracted 98,312 samples of heartbeats, all labeled and having 360 features, representing the MLII values at each time step. While we loaded the data in for each patient, we decided to use a standard scaler from sklearn that removes the mean and divides by the standard deviation for the MLII measurements for that individual patient. We found that normalizing after the dataframe was fully constructed led to the heartbeat waves being distorted non uniformly, which is why we opted for the standard scaler to normalize by patient. After normalizing this way, the plots of the waves maintained their shape, and are simply rescaled to a smaller range of values that are better for the neural nets training, and was the desired result of normalizing.
The shape of our dataframe after preprocessing the files was 98,312x363. There were 363 features as we have type of heartbeat (class label), sample number and person number in the features for later use with the VAE. Some examples of the heartbeats can be seen below, 3 from

each class. In many of the classes below, like classes A, R and U, the 3 randomly selected signals differ from each other in the same class. This makes it very difficult to separate the classes just based on visually looking at the samples here. However, through some analysis of the class N signals, many of them have a similar shape and we can sort of pick those out from the rest. Also with classes L and R, they sort of have the same curves and and have similar abnormalities amongst the other samples in their respective classes which helps identify the samples as a particular class. It is quite impressive the effectiveness of these models given the complexity and variations of these heartbeat signals.

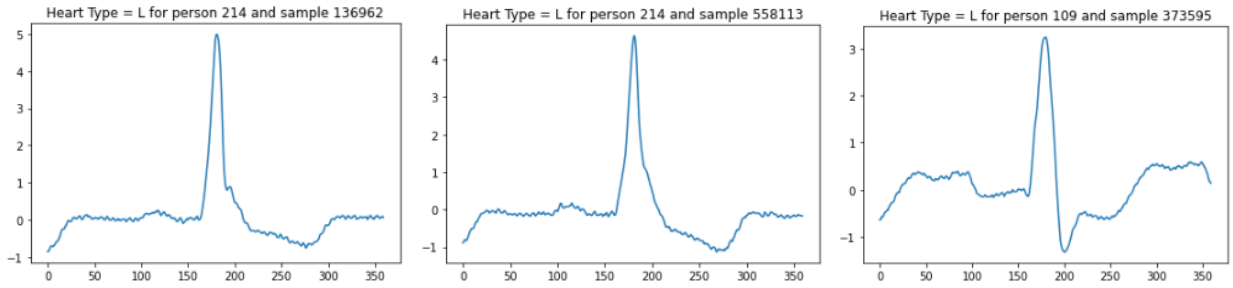
Class N plots



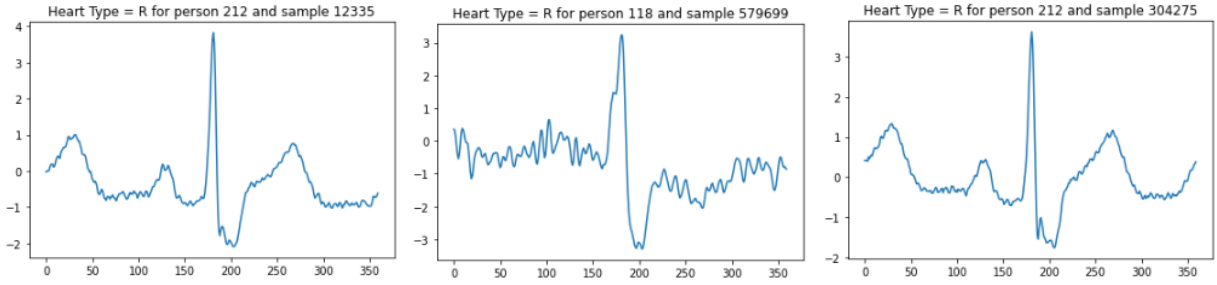
Class A plots



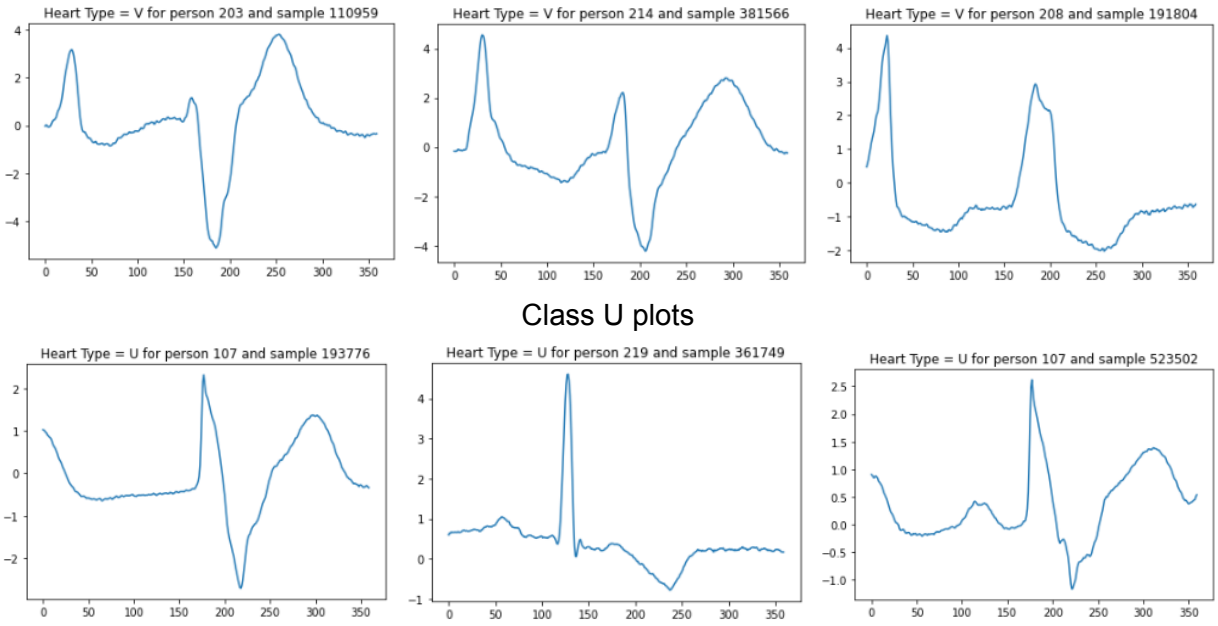
Class L plots



Class R plots

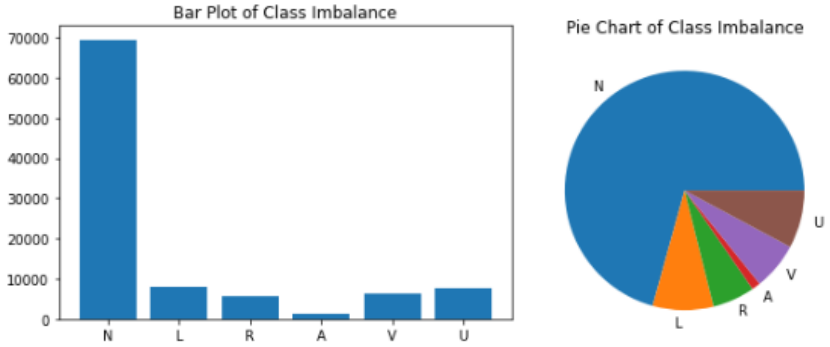


Class V plots



Answer 3) (25 points + 10 bonus marks)

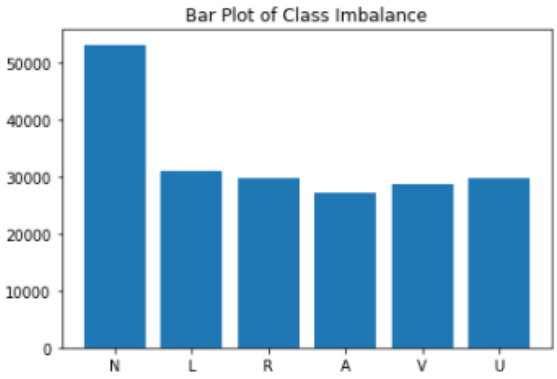
The classes were very imbalanced in this dataset, which makes sense as most of the heartbeats in any given patient should be class N, or normal heartbeats. We found that the classes had the percentages of total samples as follows: N: 70.700%, L: 8.210%, R: 5.701%, A: 1.176%, V: 6.396%, and U: 7.817%. Below are plots to help visualize the class imbalance, a bar plot and a pie chart. Showing this we can see that class N is the vast majority of samples.



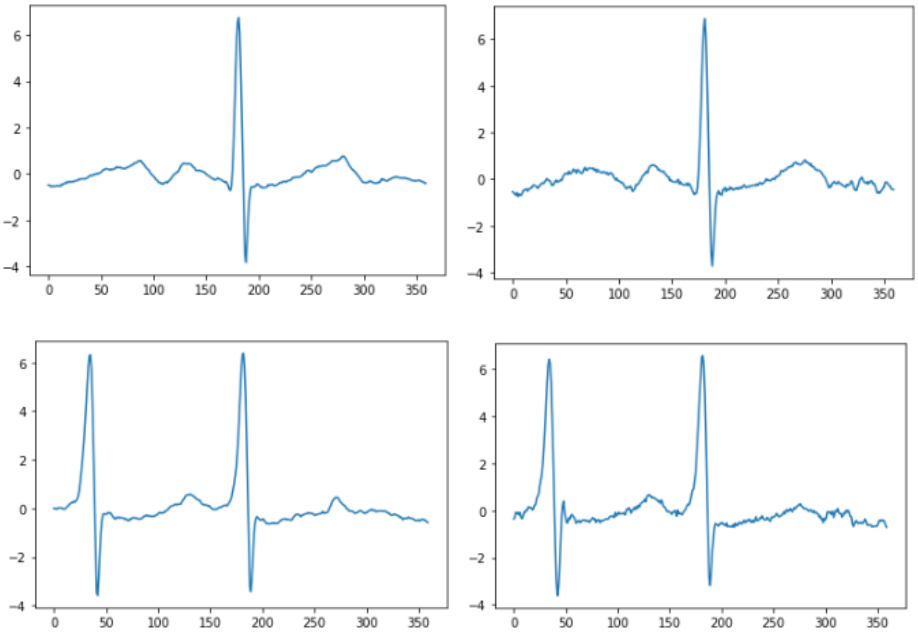
Autoencoder

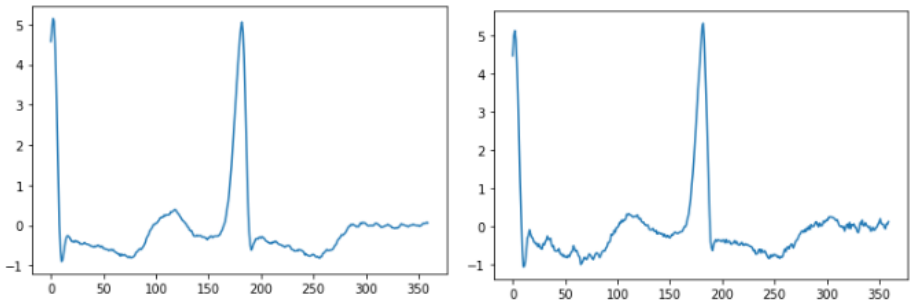
For the autoencoder we opted for a Keras network with a 2 layer encoder and 2 layer decoder. The latent space for this is 100-d. The full architecture and use of the AE can be found in the code. Since the problem exists with regular autoencoders where the latent space doesn't consistently map to a valid result, we encoded the signals, added random noise with a zero mean and 0.1 standard deviation, and then decoded these altered latent signals. We found that this gave pretty consistent results, and we were able to generate more samples for the classes that needed them, yet we found that it did not improve the baseline much for RF and not at all for the dense keras network. This may be because of the method for generating the new

samples using random noise in the latent space, or a poorly tuned hyperparameter set up. The number of rows for each class after using the autoencoder are as follows: N: 53173, L: 30900, R: 29813, A: 27125, V: 28662, and U: 29720. Due to memory constraints when generating the AE samples, we opted to generate ½ the size of the N class for each class, this results in the below graphs. As we can see the classes are more balanced, but due to the constraints on memory we see class N is still ~twice as much as each of the other classes.



Below are samples of the original A class vs the autoencoder generated A class. As we can see the original class A heartbeats (left images) just look like cleaner signals of the generated class A heartbeats (right images). This is because of the method we chose to use for adding noise in the latent space to generate a new signal. This method was effective at generating signals that look similar to the original inputs, which originally we thought would not test well as the signals really aren't that much different. However, looking at Table 1 below the graphs we can see that the autoencoder was very effective as the test set accuracy was 0.976 f1 score average on the dense feed forward neural network. The kaggle f1 score was 0.78139 using the cnn with this data generation method, which is slightly worse than the vae discussed below.



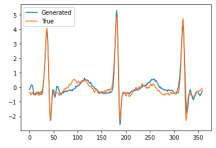
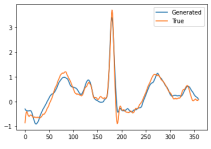
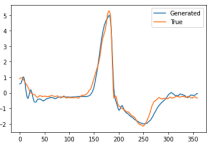
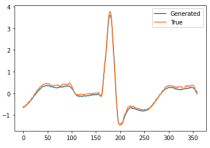
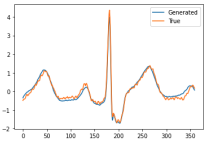


	f1	accuracy	precision	recall
Random Forest	0.98	0.98	0.98	0.98
Feed-Forward NN	0.975	0.975	0.976	0.975
CNN	0.973	0.973	0.973	0.973

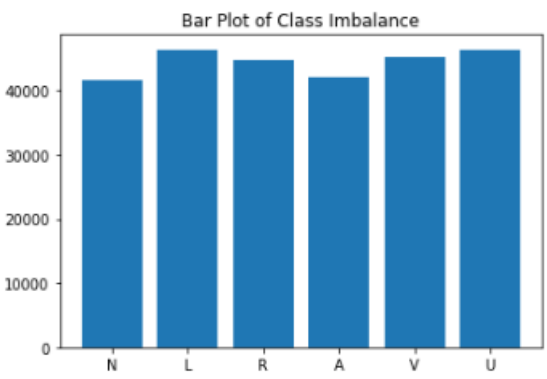
Table 1. Test set metrics using autoencoder generated samples

Variational Autoencoder

For the VAE we opted for a pytorch network with a 2 layer encoder and 2 a layer decoder. The latent space for this is 10-d. The biggest challenge with the VAE was the loss function. The loss function included the MSE and KL divergence. But the key learning here was that MSE had to be summed instead of taking the mean. The VAE produced very consistent results as shown in the table of plots below. With the generation of data, care was taken to ensure that the test data did not leak into the VAE. Hence the test data which included 20% of each sample class was always held out. In Table 2 below, we can note that the VAE was very effective in generating good data for the models to learn on. On kaggle the best f1 score we achieved was 0.79466.

A Class	U Class	V Class	L Class	R Class
				

The resulting balance on the classes after using the VAE is shown here. The number of samples per class are as follows: N: 41607, L: 46447, R: 44853, A: 42195, V: 45310, and U: 46340. The plots below really exemplify the data balance.

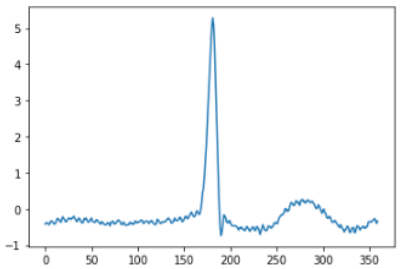


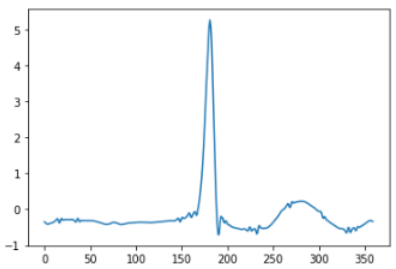
	f1	accuracy	precision	recall
Random Forest	0.961	0.958	0.965	0.959
Feed-Forward NN	0.964	0.964	0.965	0.964
CNN	0.964	0.964	0.965	0.964

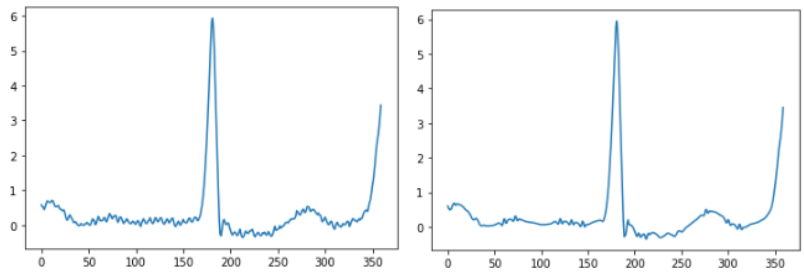
Table 2. Test set metrics using VAE generated samples

Answer 4) (10 points)

The denoising method that we opted for was the wavelet denoising method in the pywt library. We originally tested two different denoising methods, both using wavelets but the other was from the skimage package which was less successful at removing the noise from the signals. This method first applies the discrete wavelet transform (dwt) to the input sample, one sample at a time, and computes a threshold for values in the signal to remove the frequency of. The function then iterates each sample's frequency component and removes it if it's below the threshold, and uses the inverse dwt function (waverec in pywt) to construct the noiseless signal. This method was far more successful than the other that we tried. Below are some examples of the noise removal in action, with the left images being before removal and right images being after noise removal. The left images show some bumpy noise along the bottom, which is effectively removed in the right images. These are two examples of the many that this noise removal worked on that we analyzed to ensure that the method worked for all the signals that we tried!







Answer 5) (5 points)
As previously shown in the metrics printed above, we chose a random forest classifier as our machine learning/non-deep learning model. We were actually very pleased with the results that we achieved using the random forest classifier as it was in comparison or better than the results of the nn / cnn for the AE and VAE generated samples. However the f1 scores for the random forest classifier did not hold as well in comparison with a kaggle score of 0.75013. We did a few rounds of hyper-parameter tuning until we converged on very good results. The only parameters that we tuned were the number of estimators and max_depth of each tree. We tested with min_samples_split as well but found that the default value actually gave us the best validation and test results. The by class breakdown is below for the random forest classifier, which we can note that it did not perform as well on class A as the deep learning models did.

	Precision	Recall	F1
A	0.42	0.75	0.54
L	0.98	0.99	0.98
N	0.99	0.96	0.98
R	0.98	0.99	0.98
U	0.88	0.90	0.89
V	0.88	0.96	0.92

Metrics on Test Set by Class Using RF (with VAE upsample)

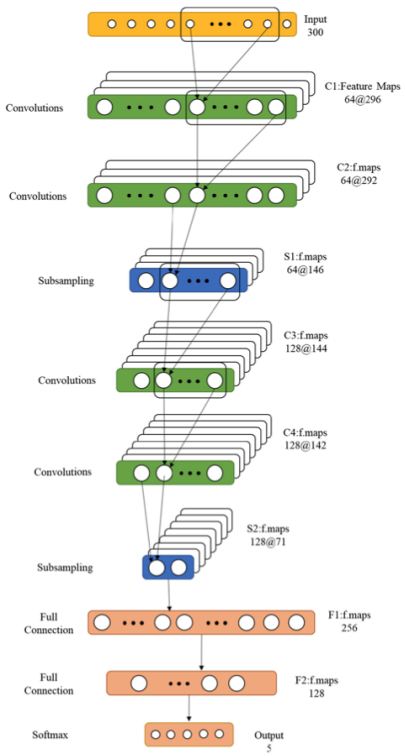
Answer 6) (10 points)
For the feed forward dense network we chose to use the Keras backend to build it for simplicity. We built a 6 layer dense network, each layer having a relu activation besides the last layer with a softmax activation. In the later half of the network we have a dropout ratio of 0.6 which majorly helps us account for overfitting in the network. Since our labels were letters, we used a label binarizer from sklearn and trained the network with a call back to track the lowest validation loss epoch which can be seen in answer 9 below. The by class breakdown for the nn is below and as we can see comparing to the above RF results, the nn did significantly better on class A, but suffered small losses in classes U and V. Overall, we were very impressed by the effectiveness of the nn model, which also tested very well on the kaggle data.

	Precision	Recall	F1
A	0.72	0.73	0.73
L	0.98	0.99	0.99
N	0.99	0.97	0.98
R	0.96	1.00	0.98
U	0.85	0.89	0.87
V	0.87	0.95	0.91

Metrics on Test Set by Class Using NN (with VAE upsample)

Answer 7) (40 points)

We use the CNN architecture proposed in the paper “X. Xu and H. Liu, "ECG Heartbeat Classification Using Convolutional Neural Networks," in IEEE Access, vol. 8, pp. 8614-8619, 2020, doi: 10.1109/ACCESS.2020.2964749”. The architecture proposed by the authors consists of 4 1D-Convolution layers with 2 max pooling layers and 3 linear layers for classification. The ReLu function was chosen as the activation function. The input layer had to be modified to use 360 nodes instead of 300 nodes and the input to the linear layer required 86*128 nodes. Dropout layers were added after the convolution layer 1 and layer 3. The loss function used to train this network was the cross entropy function. The weights were updated using the Adam solver.

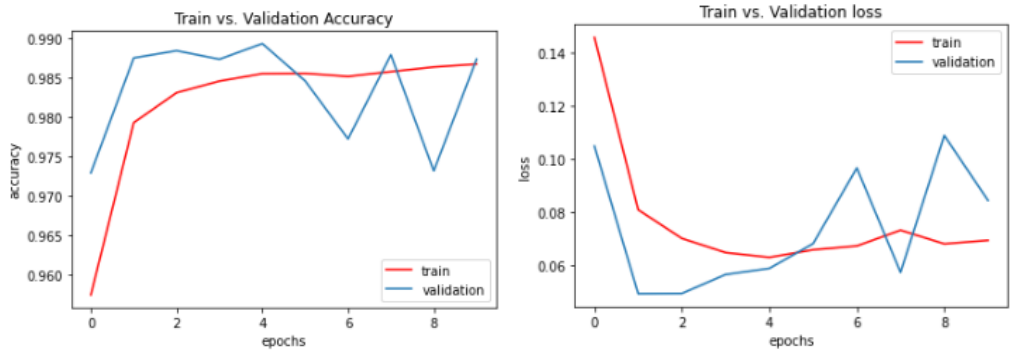


Proposed Architecture

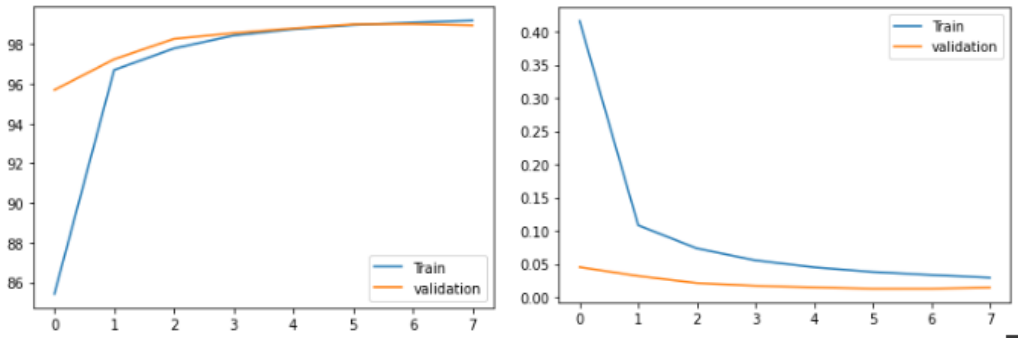
The paper suggests that the accuracy across 107679 heart beats was 99.4%. With our model for 98312 samples (entire dataset) the accuracy was 97%. It is hard to compare the exact classes as the nomenclature is not the same as what we are using. However, one thing was clear: the A class continued to perform worse than the other classes and remained at around 89%. The f1/precision and recall scores are shown in the table above.

Answer 8) (10 points)

Shown below are the train and validation curves for accuracy (left) and loss (right) for the feed forward neural network that we used. As we can see the network converged very quickly, starting with low loss and higher accuracy after just 1 epoch of training and then bouncing around after that. The model that was saved by using keras callback for the nn model was actually after epoch 2 of training. This can also be seen in the graph as the network started to overfit to the training data after just the second epoch. Because the scale is so small on the y axis and there were a small number of epochs, the jumping around in validation accuracy and loss is exaggerated visually.

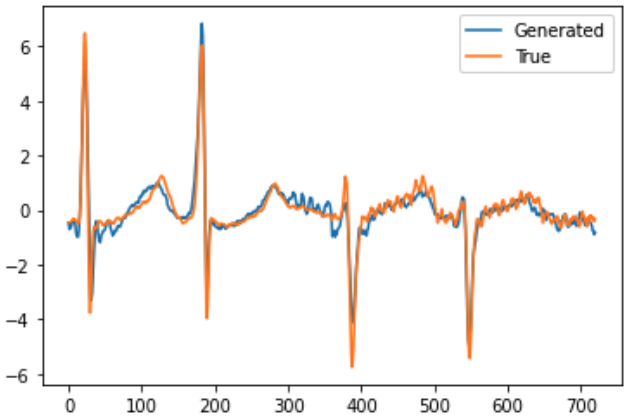


The two graphs shown below, cnn accuracy (left) and loss (right) over 8 epochs are much smoother in comparison. Like the feed forward nn, we can see that the network was able to converge very quickly, but then have a very smooth curve down the rest of training.



Answer 9) (15 points)

We chose the V1 feature as it was the second most prominent feature after the MLII feature. Here, we concatenated the V1 to the MLII feature and each signal now consisted of 720 samples. The CNN and VAE were modified to account for 720 samples instead of 360 samples. As seen in the table below the CNN with 2 features tends to perform better than the prior model that includes only a single feature. Unlike the MLII, the V1 tends to follow almost the inverse of MLII as shown in the figure. This tends to provide more information to the model thereby helping it to perform better.



	f1	accuracy	precision	recall
CNN - 2 features	0.978	0.978	0.979	0.978
CNN -1 feature	0.964	0.964	0.965	0.964

Test set with features MLII and V1 (with VAE upsample)

Answer 10) (10 points)

Print/plot examples or time-series snippets of mis-classified arrhythmia (False positives) and also false negatives. Why might this have happened?

Using the test set results from the cnn predictions we compared the predictions to the ground truths and labeled a few plots below. The false positive samples (bottom row) were when the model predicted class A and ground truth said class N and the false negative samples (top row) were when the model predicted class N when the ground truth said class A.

Looking at the upper left image, a false negative, we can note that it looks very similar to the class N data in the graphed heartbeats on page 2 of this report. The R peak here has a very similar pattern to the R peak of the 3rd class N heartbeat. Another interesting result is the 3rd image here in the first row which predicted N, but looks very similar to the class A graphed signals on page 2 with close R peaks in succession.

The false positives are interesting here, the middle image in the second row looks to me like a class N signal, which it is, but the model predicted class A. With the VAE/AE given that some of the class A is similar to N it is possible that during oversampling it made it harder for the model to capture the difference between the two classes.

These discrepancies can most likely be chopped up to the model not being able to capture every possible case of each class, or to mislabeled data. Since each R peak was labeled by hand, it would not be impossible for certain peaks to be mislabeled.