CS7641 Machine Learning Project Group 26

James Jun, Nikhil Sachdeva, Nalin Semwal, Tanmay Shishodia, Aishwarya Solanki

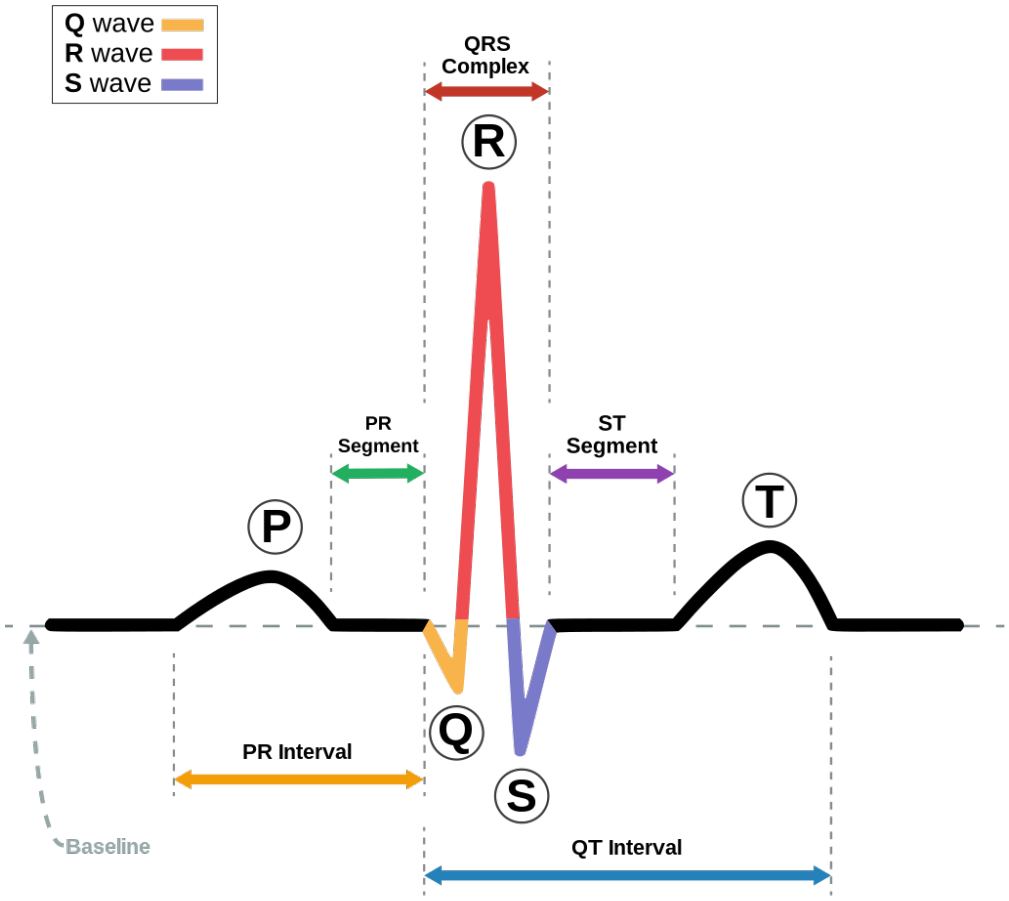
CS7641 Machine Learning Project Group 26

Classifying Cardiovascular Health from ECG Signals with Classical and Modern Machine Learning Techniques

Introduction

Electrocardiogram (ECG) signals represent the electrical activity of the human hearts and consist of several waveforms (P, QRS, and T). The duration and shape of each waveform and the distances between different peaks are used to diagnose heart diseases.

ECG analysis is an important diagnostic tool that helps doctors detect cardiovascular diseases and choose appropriate treatment for patients. This can be further enhanced by deploying ML models on the ECG data and execute classification techniques to identify abnormalities [1].



There are a lot of publications, which focus on various features of ECG, like RR intervals, R peak, QRS Complex width, ST segment length and so on. Various techniques like Discrete Wavelet Transform, Continuous Wavelet Transform, S-transform, Independent Component Analysis, and so on are used to extract that feature, and then normalized using Z-score, zero-mean, or Unity SD [1].

For the learning process, use of neural networks seems to be highly prevalent, with a lot of variants being used: Feed Forward Neural Network, Fuzzy Clustering Neural Network, Probabilistic Neural Network, Multilayer perceptron neural network, Radial Basis Function NN, etc. often in combination with Density Functional Theory, fuzzy logic, PDF, DWT, Discrete Cosine Transform, Fuzzy cmeans clustering, ICA, Principal CA, SVM, Particle Swarm Optimization, and so on [1].

In this project, We have 6 learning models that we are going to evaluate and analyze the performance by training and then comparing predictions on a test dataset with given values:

- KMeans
- Gaussian Mixture Model (GMM)
- Support Vector Machines (SVM)

- Fully Connected Networks (FCN)
- Long Short-Term Memory (LSTM)
- Transformers

We will use the PTB-XL dataset, which is based on raw signal data and composed of 21,387 clinical 12-lead ECG records of 10 seconds in length from 18,885 patients [2]. The records are further annotated by two cardiologists, conforming to the SCP-ECG standard [2][3]. For supervised learning, the dataset itself has five coarse superclasses, and 24 subclasses for diagnostic labels [2].

Problem Definition

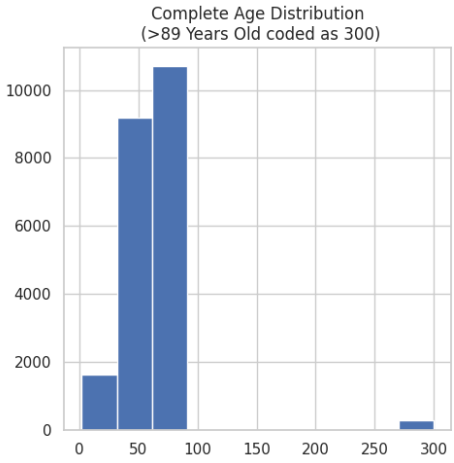
The problem in ECG analysis is the need for accurate and efficient interpretation of its signals, which can be time-consuming and prone to human error. This project aims to implement and compare different machine learning techniques to automate and enhance ECG analysis. By doing so, it addresses problems such as improving diagnostic accuracy, reducing the workload on healthcare professionals, enabling early detection of cardiac abnormalities, and ultimately enhancing patient care and outcomes in a more timely and cost-effective manner.

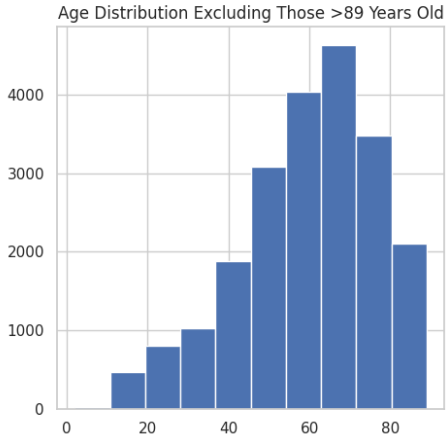
Data Collection

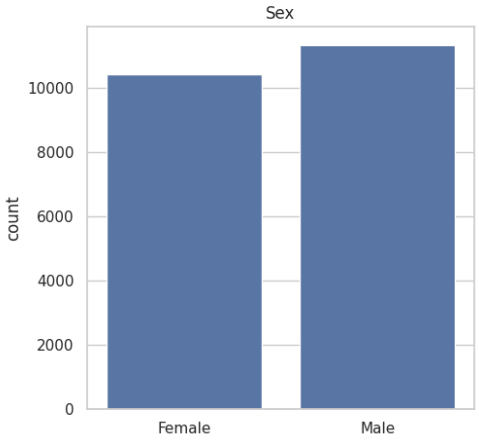
The PTBXL dataset we use in our project (https://physionet.org/content/ptb-xl/1.0.3/) is balanced with respect to sex (52% male and 48% female) and covers the whole range of ages from 0 to 95 years (median 62 and interquantile range of 22)[2]. The metadata corresponding to the waveform data was manually entered by a nurse. We are provided with 12 leads i.e. I, II, III, aVL, aVR, aVF, V1, V2, V3, V4, V5, V6 with their reference electrodes on the right with 400Hz as the original sampling frequency. For each record, the following steps were carried out:

- Initial ECG report was generated by a human cardiologist or an ECg device
- That was further converted to standard sets of codes - SCP-ECG
- Applicable information like heart axis during the scan was then extracted from the report
- To prove the accuracy of the report, a second opinion was considered of an independent cardiologist.
- Lastly, focusing on qualitativeness of the signal, the records were manually annotated by experts.

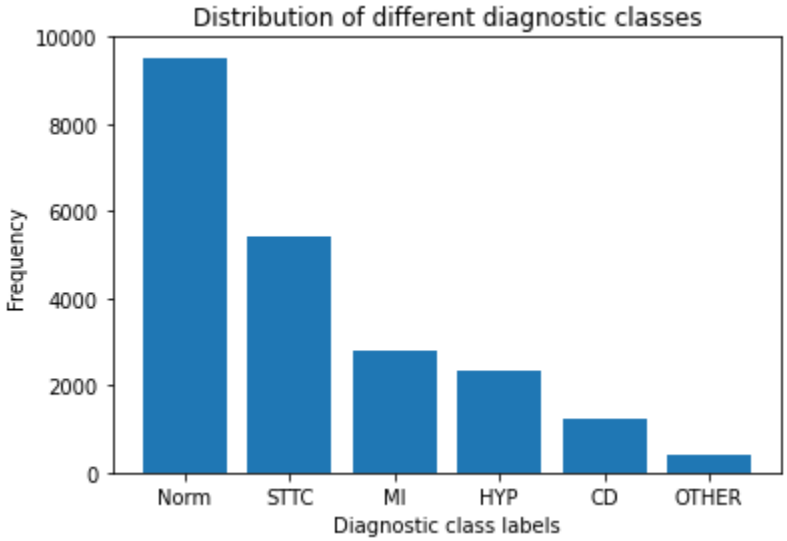
Below is a visualisation of the dataset described above:

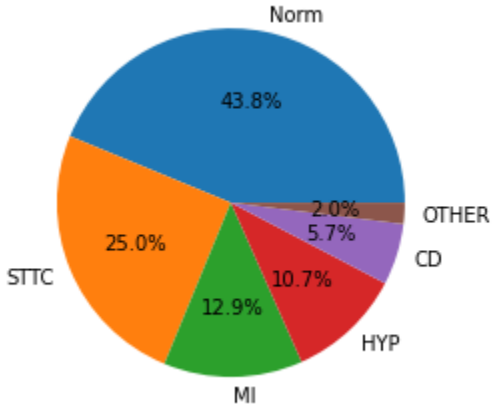






Further, we visualized the distribution of diagnostic codes within the PTB-XL dataset:





Methods

Classical methods performed manual feature engineering for ECG classification, typically involving modeling the different parts of the waveform and detecting irregularities [4]. With the advent of recurrent models like GRUs and LSTMs [5] came deep-learning focused methods that allowed the neural networks to process the waveform directly.

In this project, we first attempt to apply unsupervised methods, namely KMeans clustering and Gaussian Mixture Model (GMM), to this task. Then we apply the more advanced models, like SVM, FCNN, LSTM, and Transformers.

Unsupervised Methods

In this section, we describe our implementation of the unsupervised methods - KMeans and Gaussian Mixture Model (GMM).

Unsupervised learning models can comb through large amounts of data and discover atypical data points within a dataset. These anomalies can raise awareness around faulty equipment, human error, or breaches in security. Which is why we have selected these algorithms to be run on the large ECG dataset to help detect anamolies as a final goal.

KMeans clustering was originally used for signal data processing quantisation. The motivation for selecting this method is that many implementations of ECG dataset classfication have surfaced in the past decade and the accuracy achieved is much higher than other unsupervised models tested. Another factor is that KMeans is a very fast and an extremely simple algorithm along with being a very easy first model to apply on a dataset even though it is infact np-hard clustering. It aims to partition the dataset into n clusters in which each observation of the dataset is a part of a cluster with the nearest mean. In this method, we have to first estimate the number of clusters we want to choose. We can do that by various methods but the most famous one as well as the one used in this project is the Elbow Method. We then select random points as cluster centers and iteratively update those centers based on the accuracy of them being actual centers or cluster centroids of a defined cluster. Instead of randomly selecting the cluster centroids, we can also implement KMeans++ for a more accurate cluster center initialisation.

We are also implementing Gaussian Mixture model (GMM) on the dataset as it performs with a better accuracy than KMeans. Also, there have been popular implementations for ECG dataset using GMM[17] which have concluded better normal class accuracy compared to other unsupervised methods. Other factors behind selecting the GMM algorithm is that it runs fast and is the fastest algorithm in the hybrid model. The GMM algorithm has Agnostic properties which means it is highly flexible compared to other models. To find the optimal number of clusters for GMM, we are using KMeans as it is faster and much simpler. A comparative analysis of both the models is provided later in the report as well.

K-Means

In this section, we describe our implementation of KMeans, the preprocessing done for the same and the model generation steps. We also discuss our results and the possible future work. KMeans unsupervised clustering method is applied to ECG Dataset and the dataset is divided into accurate clusters. Preprocessing was the most daunting task for KMeans as it only works on numeric data because of Euclidean Distance being at the center of the algorithm.

Preprocessing

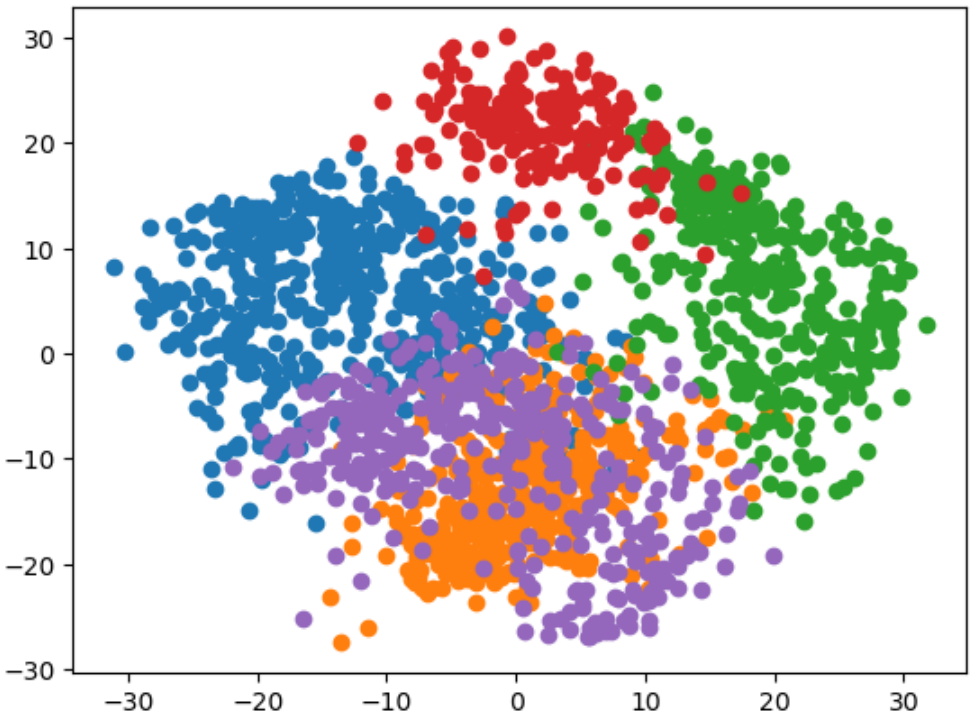
- **Data Cleaning** - We attempt - Removing duplicates, Removing irrelevant observations and errors, Removing unnecessary columns, Handling inconsistent data, Handling outliers and noise from the Dataset for it to be precisely useful for the KMeans algorithm. We remove

the NaN values as they are very low in number, we could also have set them to 0.0 for the sake of not removing values. KMeans is sensitive to outlier data points. In the dataset, we see ages more than 89 years have been assigned an age of 300 years. There were 29 such values which were removed to ease the further processing.

- **Quantize the categorical values** - The ECG Dataset has a lot of categorical columns values like [a, b, c] for us to run KMeans on that, we require them to be quantized to numeric dataset like [1,2,3]. We transform all the categorical columns to numeric values for further processing.
- **Scaling Features** - It is important for the features to be scaled closely in order for KMeans to work accurately, this also means, we handle the skewness that the data comes with. This data transformation step is very important and can be performed easily by min_max scaler to get the data between 0 and 1. You use MinMaxScaler when you do not assume that the shape of all your features follows a normal distribution.We could have also used Standard Scaler, a python library method, if we wanted data between -1 and 1. We have used a Standard Scaler in our project.
- **Dimensionality Reduction** - We attempt to use PCA for dimensionality reduction for KMeans clustering as taught in the class. Since it is difficult to visualize data with more than 3 features, we attempt to reduce the dimensionality using PCA.

Implementation

We built upon a generic implementation of KMeans algorithm on a large dataset. After loading the raw data set, We split our dataset into 90% training and 10% testing data. We implement the elbow method and calculate silhouette coefficients to determine the ideal number of clusters and the overall performance of KMeans. A glance at the dataset did reveal the distinct clusters possible. We can see 5 main classes forming - ‘CD’, ‘HYP’, ‘MI’, ‘NORM’, ‘STTC’ based on the dataset. As in unsupervised learning, we normally do not learn labels, it is hard to classify which label is which class. We could verify our understanding from the methods as mentioned above. We then flatten our raw dataset of 3D to 2D and fit the KMeans model on the same. This means that we deprive ourselves of the precision and accuracy of the whole raw dataset which induces loss. As you can see below, KMeans is unable to cluster the classes with a high accuracy as expected. As we can see from the below clustering, there is a major overlap between the purple (label 3) and orange (label 4) labeled classes as well as some minor overlap between other classes (like red (label 1) and green (label 2)). On the surface, KMeans is not able to give out excellent results on the dataset which opens us to try out other models.



GMM

In this section we will take a look at Gaussian mixture models (GMMs), which can be viewed as an extension of the ideas behind k-means, but can also be a powerful tool for estimation beyond simple clustering. We also discuss our results and the possible future work. GMM unsupervised clustering method is applied to ECG Dataset and the dataset is divided into accurate clusters.

Motivation

As we saw in the previous section, given simple, well-separated data, k-means finds suitable clustering results. Inside, KMeans is similar to GMM wherein there is an EM algorithm which essentially -

1. Chooses a starting point
2. Repeats till converged
    1. **E-step**: for each point, find weights encoding the probability of membership in each cluster
    2. **M-step**: for each cluster, update its location, normalization, and shape based on all data points, making use of the weights

Unfortunately, the k-means model has no intrinsic measure of probability or uncertainty of cluster assignments. Apart from that, when we think about our clusters being non-circular, KMeans is not able to correctly categorize them and forces some data points inside a circular

cluster and ends up mixing up some clusters by overlapping them. To be flexible with the cluster shape and size, we need the model to have more parameters and more flexibility. This is where GMM comes in the picture. GMM (Gaussian mixture Model) is a probabilistic model for clustering, which uses an EM (Expectation Maximization) algorithm for iterative calculation.[17] The Gaussian mixture model assumes that the data of each cluster conform to Gaussian distribution (also called normal distribution), and the distribution presented by the current data is the result of the superposition of Gaussian distribution of each cluster.

Preprocessing

We use the same preprocessed data that was used for KMeans.

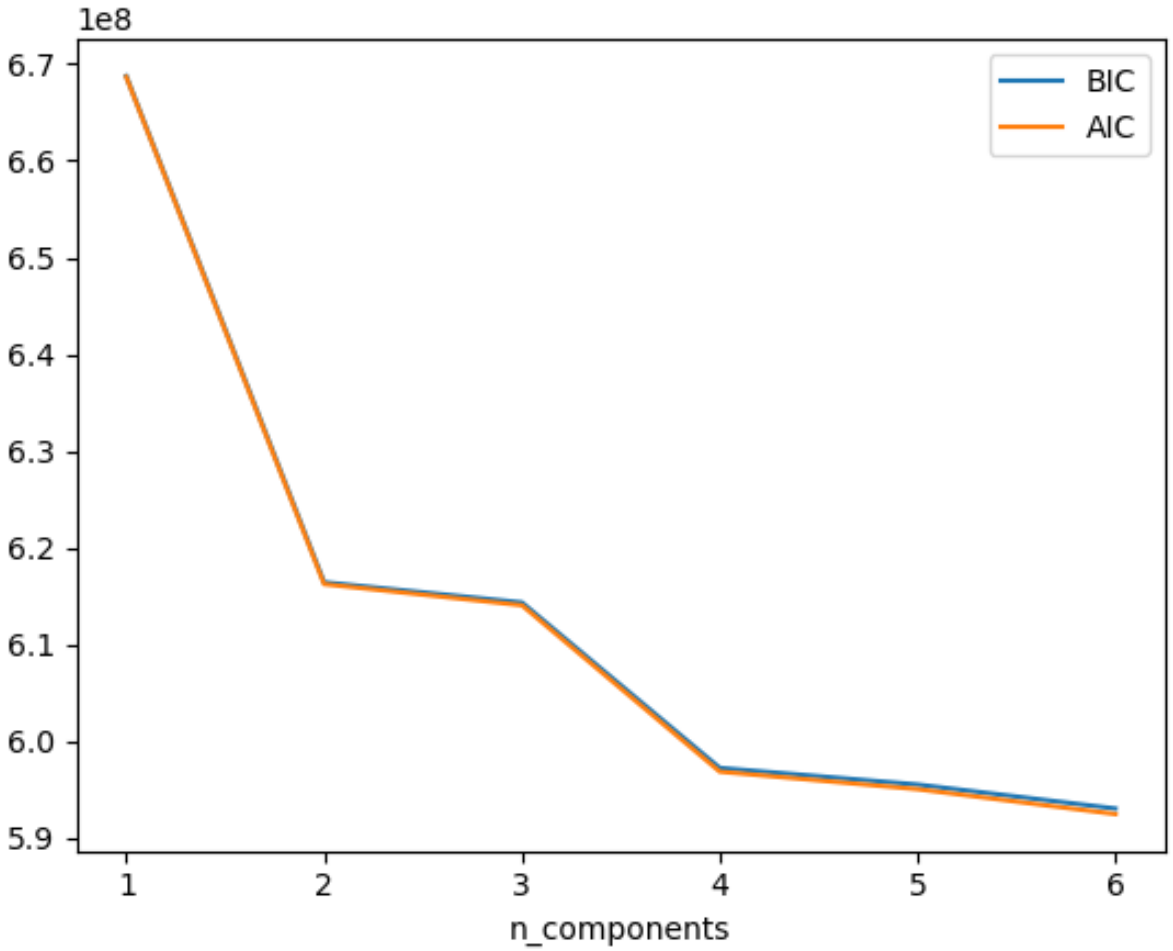
Implementation

We built upon a generic implementation of the GMM algorithm on a large dataset. After loading the raw data set, we split it into training and testing data. We used KMean to determine the ideal number of clusters as well as a glance at the dataset did reveal the 5 distinct clusters possible - 'CD', 'HYP', 'MI', 'NORM', 'STTC'. We could verify our understanding from the methods as mentioned above. Since, GMM is a probabilistic model under the hood, it was also helpful to visualize the probabilities of each point belonging to a particular cluster using the `predict_proba` method of the GMM object. We have implemented 2 GMM models, one with `covariance = 'diagonal'` and one with `covariance = 'spherical'`.

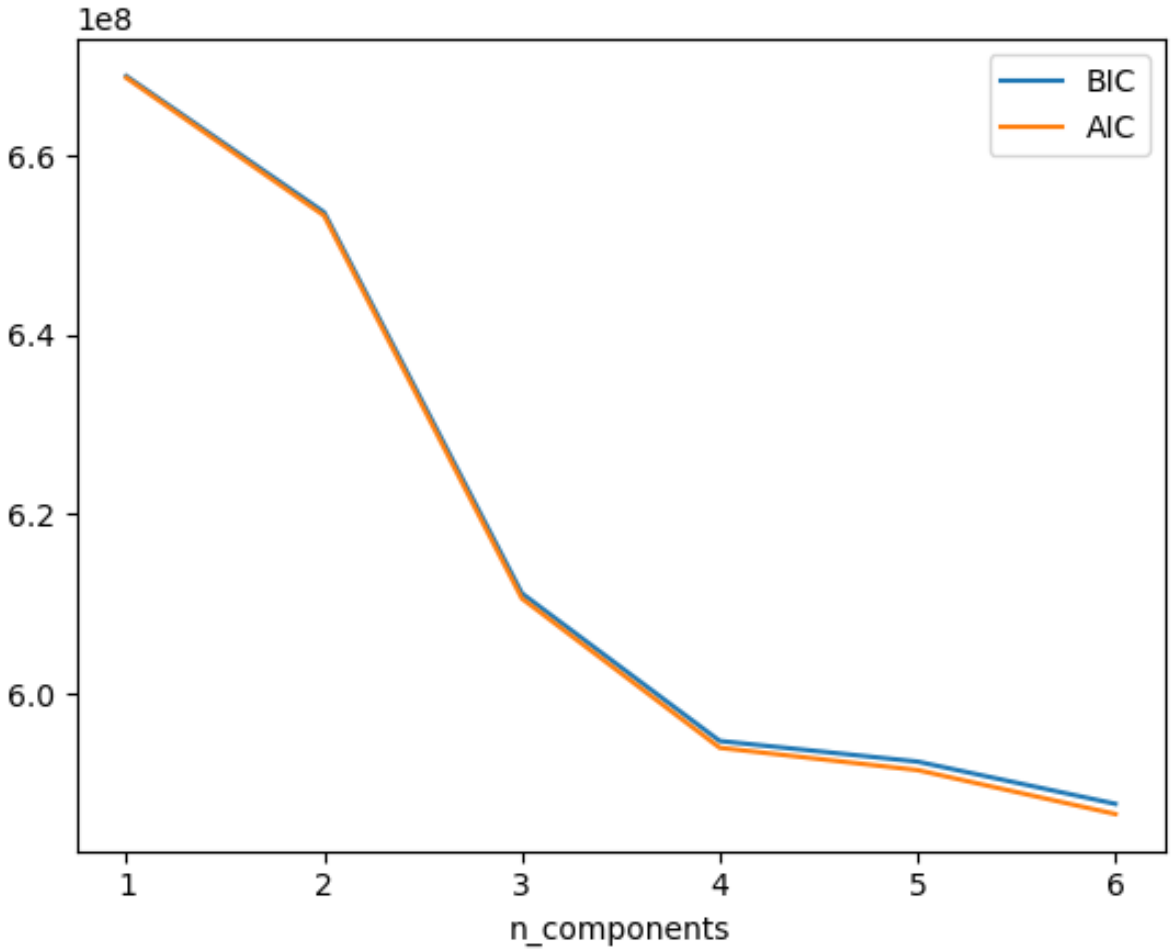
- Bayesian Information Criteria (BIC) is a method for scoring and selecting a model. It is especially helpful while selecting probabilistic models such as GMM itself. BIC is a criterion for model selection among a finite set of models. It balances the goodness of fit of the model and the complexity of the model, penalizing overly complex models.
- Akaike Information Criterion (AIC) is a measure used for model selection. It quantifies the goodness of fit of a statistical model while penalizing the model for its complexity. The model with the lowest AIC is considered to be the best.

Unlike Bayesian procedures, BIC and AIC inferences are prior-free. Both AIC and BIC are scoring systems for model comparisons in classical statistics dealing with models with different numbers of free parameters. We score the two different models and keep the best model (the lowest BIC and AIC). Since, both Spherical and Diagonal GMM had similar AIC and BIC values, we can consider both are equally effective on the dataset.

AIC and BIC metrics for GMM with Diagonal covariance -



AIC and BIC metrics for GMM with Spherical covariance -



SVM

In this section we discuss SVM. Support Vector Machines (SVM) are a class of supervised machine learning algorithms used for classification and regression tasks. The motivation for applying SVM lies in its ability to handle nonlinear relationships, robustness in high-dimensional spaces, effective binary classification approach for multilabel problems, margin maximization for better generalization, and controlled complexity through kernel functions. However let's see how it performs on the given dataset.

Preprocessing

- Ages more than 89 years have been assigned an age in of 300 years according to the dataset documentation. There were 29 such values and were removed using the z-score outlier method.
- Missing values for height, weight, nurse and site were imputed using the mean.
- SCP Codes which contains our target labels is a JSON column with the likelihood for each label to be applicable on the patient. These were extracted out and a column was added for each label with either 0 or 1 indicating whether that diagnostic class is applicable to the

patient or not.
- Since SVM cannot handle multi label classification directly we perform a binary classification for each of these labels separately and measure it's performance.

Implementation

- The dataset was split into 80% train, 10% validation and 10% test sets.
- Possible features columns are age, sex, height, weight, nurse, and site. After performing correlation analysis plotted in the figure below, we notice that none of the feature are highly correlated except height and weight showing some considerable positive correlation.

image

- Next we performed PCA which aims to transform high-dimensional data into a new coordinate system (principal components) where the data's variance is maximized along the axes. It achieves this by identifying the directions (principal components) in which the data varies the most. The core of PCA is the choice of the number of components k, which is investigated by cumulative explained variance ratio such that we can preserve the information of original data. The image below shows the cumulative explained variance ratio for different values of k. We get maximum variance of 1 only when all features are considered, hence PCA doesn't help eliminate dimensions here.

Model training and backward feature selection are discussed in the Results and Discussion

section for SVMs.

Fully Connected Networks

Fully connected networks are a fundamental type of artificial neural network commonly used in various machine learning and deep learning applications, including in the analysis of Electrocardiogram (ECG) datasets. In this network, each neuron in a layer is connected to every neuron in subsequent layers. These networks typically consist of an input layer, one or more hidden layers, and an output layer. Despite the transition away from simpler models such as Fully Connected Networks to Convolutional Neural Networks (CNNs), these are still powerful tools in the analysis of complex datasets. In this project, we hope to see how well a Fully Connected Network can perform on the PTB-XL dataset.

Preprocessing

We removed any duplicates and any data points that do not have an assigned label. We opted to remove instead of generate labels for these data points as our dataset is quite large and should not suffer in performance from a few missing data points.

In the same way as performed for SVM, we scaled the signal values such that they lie within a range of 0 and 1. This is a common normalization technique used with signal-based data that we apply here as well to prevent any issues with bias towards features with larger magnitudes and may help the model converge more efficiently. As a result, Min-Max Normalization was applied to our data prior to training.

Implementation

Our final model used had following architecture:

- Fully Connected layer (FC1)
    - Input Features: 1000
    - Output Features: 1000
- Dropout: Applied with a probability of 0.8
- ReLU layer
- Fully Connected layer (FC2)
    - Input Features: 1000
    - Output Features: 1500
- Dropout: Applied with a probability of 0.8
- ReLU layer
- Fully Connected layer (FC3)

    - Input Features: 1500
    - Output Features: 6
- Softmax layer

Our final choice in architecture was selected based on iteratively training different types of fully connected network architectures on the PTB-XL dataset as seen in Table 1 below. Network Architecture (# input params, # output params) Best Validation Acc. Training Acc.

Table 1. Experimentation of different architectures for Fully Connected Network, where each fully connected layer had a Dropout layer and ReLU activation function. Batch size of 64, Adam Optimizer with learning rate = 0.001, and early stopping with patience of 10% of total epochs (20) was employed when performance began to degrade.

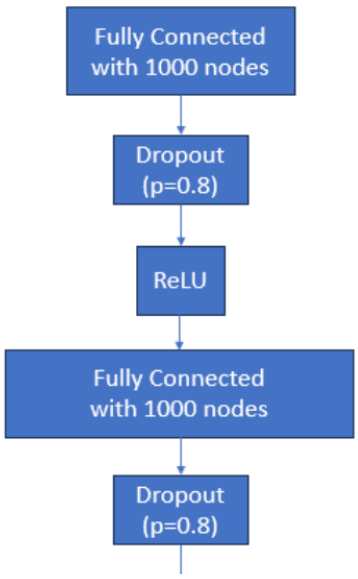
Network	Architecture (# input, # output params)	Validation Acc.	Training Acc.
FCN #1	FC1(1000, 2000), FC2(2000, 4000), FC3(4000, 6) Dropout=0.5	43.08% after 4 epochs	62.80%
FCN #2	FC1(1000, 2000), FC2(2000, 4000), FC3(4000, 8000), FC4(8000, 6) Dropout=0.5	43.88% after 6 epochs	63.87%
FCN #3	FC1(1000, 1000), FC2(1000, 2000), FC3(2000, 4000), FC4(4000, 6) Dropout=0.5	42.68% after 5 epochs	61.23%
FCN #4	FC1(1000, 1000), FC2(1000, 2000), FC3(2000, 6) Dropout=0.5	44.29% after 6 epochs	59.49%
FCN #5	FC1(1000, 1000), FC2(1000, 2000), FC3(2000, 6) Dropout=0.8	42.78% after 12 epochs	47.27%
FCN #6	FC1(1000, 1000), FC2(1000, 1500), FC3(1500, 6) Dropout=0.5	42.9% after 5 epochs	57.74%
FCN #7	**FC1(1000, 1000), FC2(1000, 1500), FC3(1500, 6) Dropout=0.8**	**45.71% after 18 epochs**	**44.57%**
FCN #8	FC1(1000, 1000), FC2(1000, 6) Dropout=0.5	38.28% after 20 epochs	38.9%

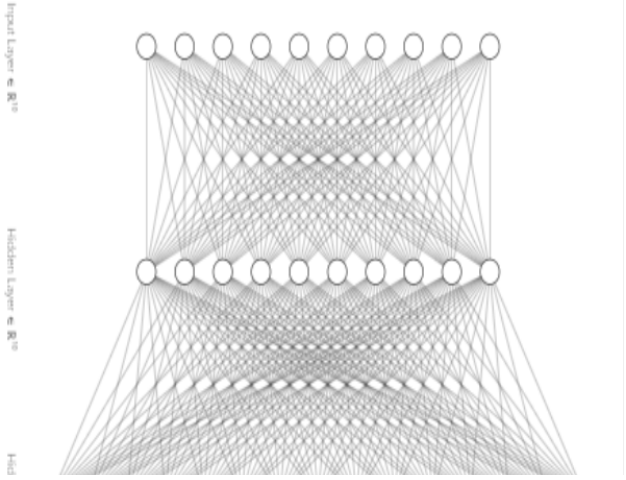
Network	Architecture (# input, # output params)	Validation Acc.	Training Acc.
FCN #9	FC1(1000, 1000), FC2(1000, 6) Dropout=0.8	37.7% after 4 epochs	38.82%

To begin, literature has very little to no implementations of independent fully connected networks on the PTB-XL dataset, typically opting for more simpler algorithms, such as in k-Means or SVMs, or incorporating them into more complex algorithms such as Convolutional Neural Networks, LSTMs, or Transformers. However, we wanted to investigate the balance of the ability of capturing more complex features in the waveforms with model complexity using FCNs to gauge the limits of this technique for ECG classification. In our current architecture, we use dropout layers to help prevent overfitting as a regularization technique by setting a random set of neurons to 0 during training. Further, we used ReLU as our activation function as we learned in class about its computational efficiency and success in capturing nonlinear relationships. Our feature maps are based on the 1000 features that are collected for each datapoint, and we limited the complexity to prevent further overfitting, which plagued much of our training.

The main purpose behind searching through different architectures for our Fully Connected Network was to limit the amount of overfitting that was occurring very early in our training. With the complexity present in our dataset, we initially opted for more layers with a larger number of parameters to hopefully account for this. However, this quickly resulted in overfitting of our model despite using ReLU activation function for nonlinear relationships and Dropout layers to help promote generalizability.

Instead, we moved toward reducing model complexity and increasing the dropout probability, which resulted in the highest validation accuracy. Here is the resulting network architecture we chose to move forward with:





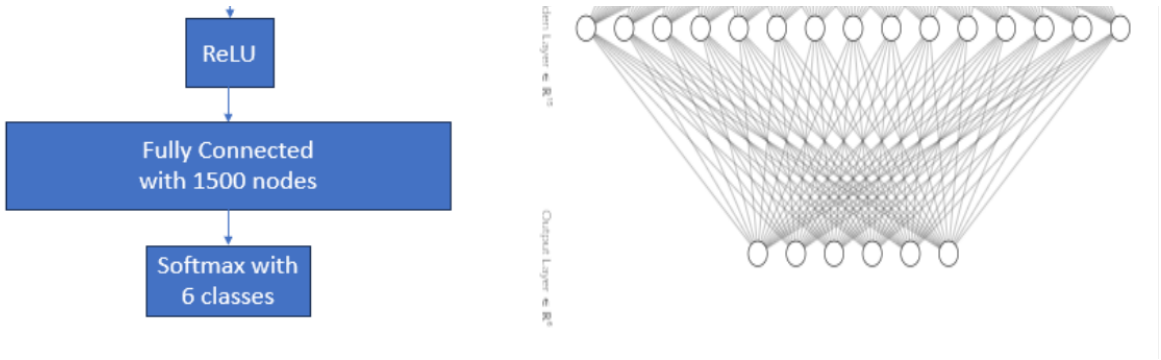


Figure 1. Fully Connected Network architecture for PTB-XL dataset

Furthermore, we performed hyperparameter tuning to try to identify the best way to train our model as shown in Table 2 below.

Table 2. Experimentation of different hyperparameters for training Fully Connected Network, using the architecture shown in Figure 1.

Batch size	Learning Rate	# Epochs	Val Acc.	Training Acc.
64	0.001	18	45.71%	44.57%
128	0.001	6	42.36%	44.39%
256	0.001	7	44.35%	45.55%
32	0.001	9	43.21%	43.62%
64	0.005	8	36.25%	30.24%
64	0.01	6	35.26%	30.19%
64	0.0005	12	45.67%	45.68%
64	**0.0001**	**13**	**45.76%**	**47.55%**
64	0.00005	18	38.29%	39.27%
64	0.00001	20	31.58%	29.83%

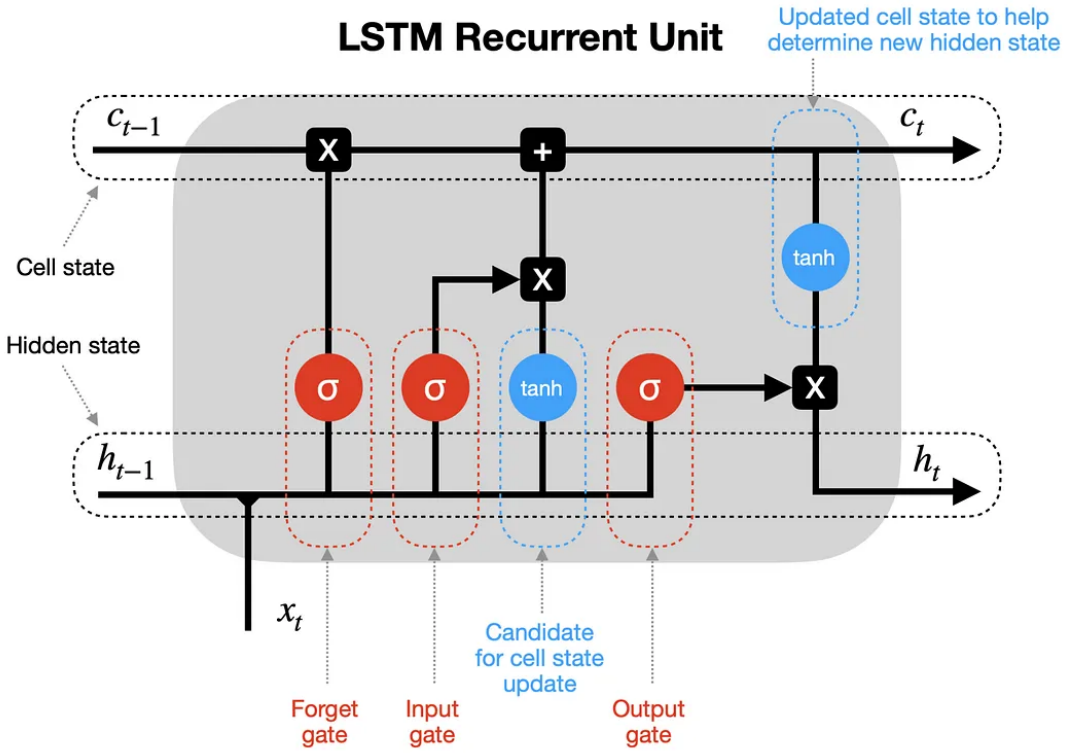
We use an Adam optimizer as it is claimed to be computationally efficient with little memory requirements and well suited for problems with large amounts of data or parameters, which our problem fits [15] Further, the use of the Adam optimizer is fairly straightforward to implement.

The hyperparameters were selected based on our investigation in Table 2. Learning rate of 1E-4

with a batch size of 64 provided the best validation accuracy. Our loss function was Cross Entropy Loss as it is commonly used in problems involving multi-class classification, and it has the ability to penalize the model more for classifying minority classes, which can also help manage imbalances in our dataset. We also trained over 20 epochs but employed early stopping to stop training the model when performance degraded.

LSTM

Recurrent Neural Networks are deep learning models, which handle time-series data very well. But they suffer from the problem of vanishing/exploding gradients. This problem is solved by the advent of LSTMs, which can better retain long-term data dependencies.



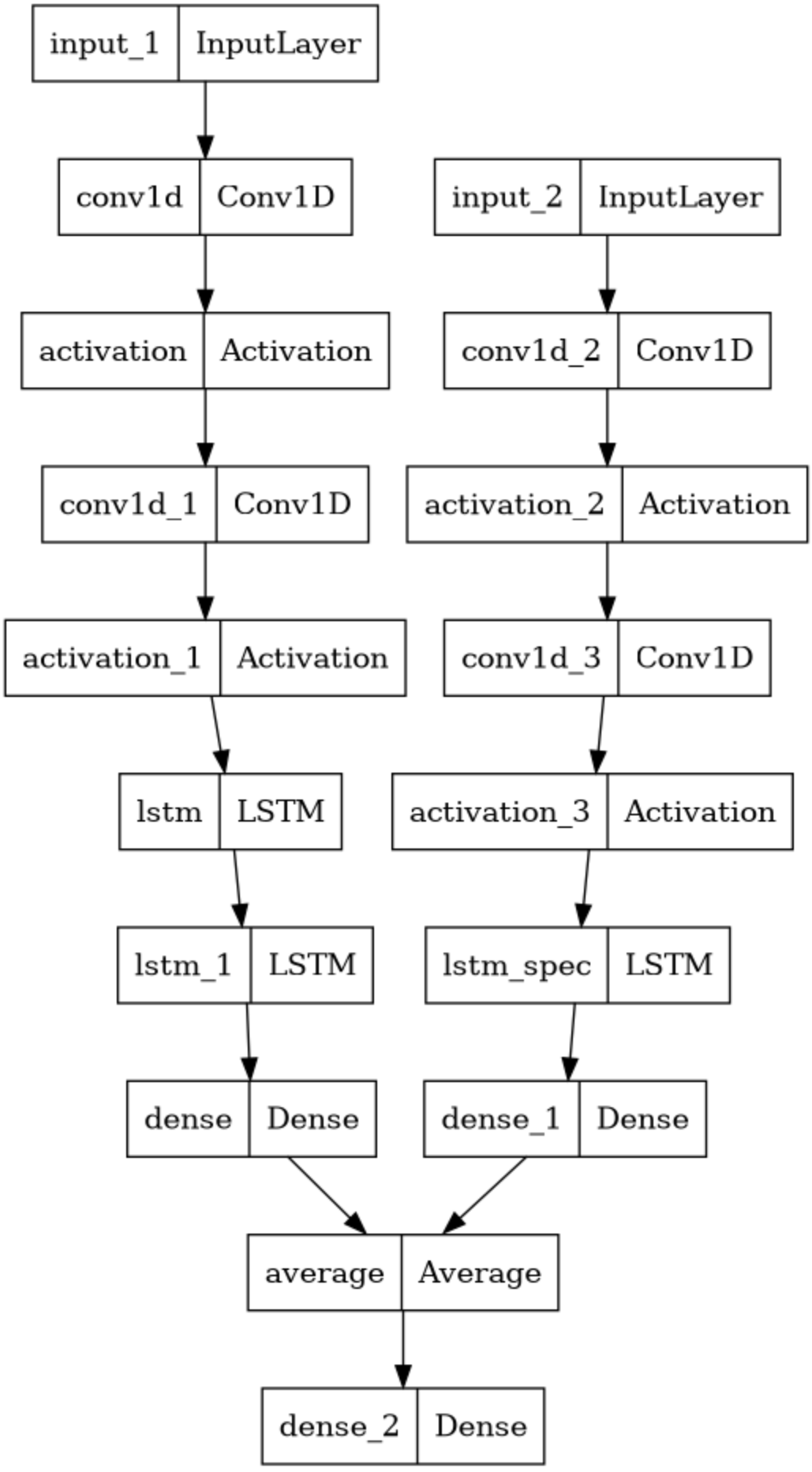
Preprocessing

- We first evaluate the data, weeding out datapoints that have more than 1 superclass diagnosis, to simplify classification and reduce the probabilistic nature of the classification.
- For the label encoding, the SKLearn MultiLabelBinarizer was used, and we essentially converted the 5 Superclass strings into a one-hot encoding format.
- For scaling the data, StandardScaler() method from the SKLearn library was used, to standardize features by removing the mean and scaling to unit variance.
- The train-validation-test split was performed according to the strat_fold column as

suggested by ECGNet authors (publishers of the data), with [0,8] folds used for training, [9] for validation, and [10] for testing.
- The scaled and split input is then converted into a custom format:
    - The first input stream gets the same data reshaped into 1 row x 1000 values per record.
    - The second input stream is a spectrographic representation of the ECG data, transformed using the spectrogram() method in the Scipy library, which better captures the frequency variations in the data.
- The data is batched into batches of 32 with a sliding window of 40 records and overlap of 10.

Implementation

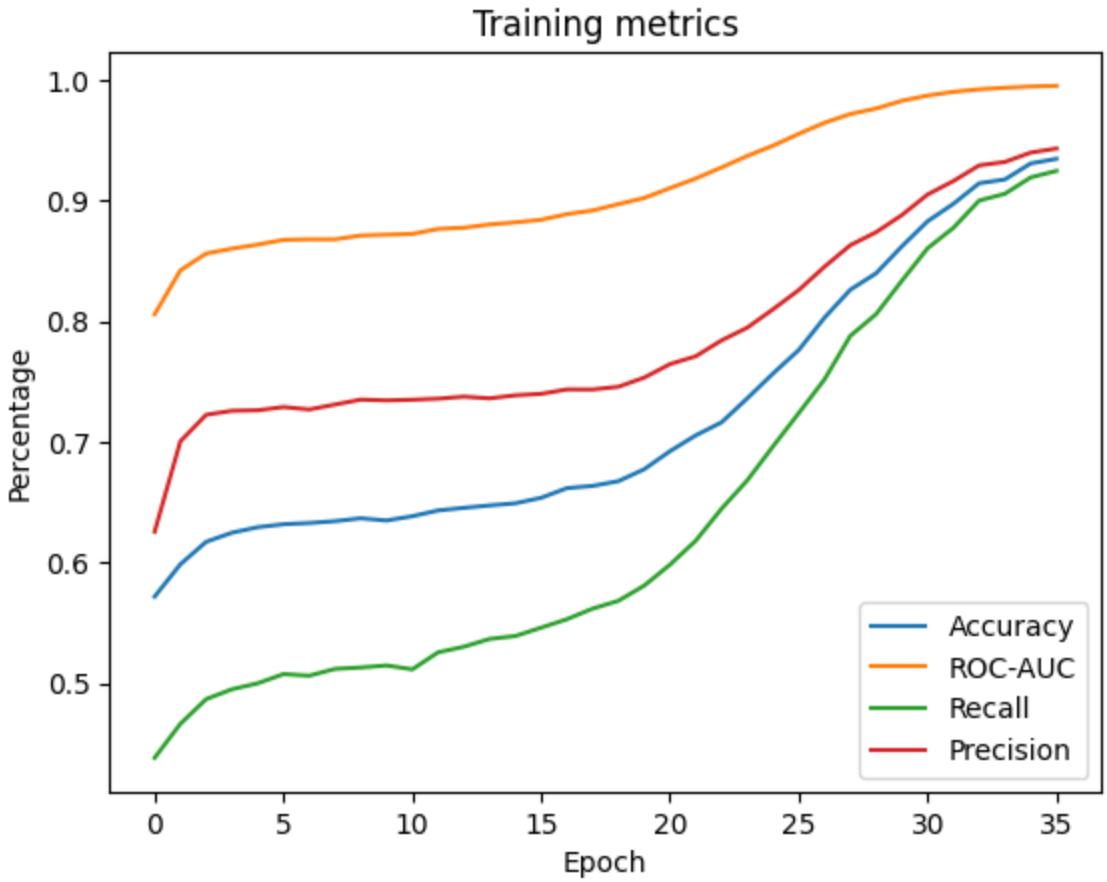
Convolutions perform very well for extracting features in signal and/or image data. They also serve to reduce the data size, which can reduce costs a fair amount even by a little bit of reduction. Hence, to reduce computational costs of running LSTM models directly on input, and to have them work only on captured/extracted data, we implemented an LSTM model with convolution layers that takes in pre-processed and scaled data in 2 varied input shapes as stated previously, and applies 2 different tracks of transformations, and averages out the output to get a final prediction.


input_1 InputLayer
conv1d Conv1D
input_2 InputLayer
activation Activation
conv1d_2 Conv1D
conv1d_1 Conv1D
activation_2 Activation
activation_1 Activation
conv1d_3 Conv1D
lstm LSTM
activation_3 Activation
lstm_1 LSTM
lstm_spec LSTM
dense Dense
dense_1 Dense
average Average
dense_2 Dense

LSTM layers on both tracks incorporate a dropout and recurrent_dropout, in order to prevent overfitting early. The convolution layers' sizes are kept as 32 and 64 for track 1, while the second track with extracted Spectrogram data has smaller convolution layers of size 6 and 16. For the activation layers, ReLu is used, and a Softmax Activated Dense layer at the end for classification.

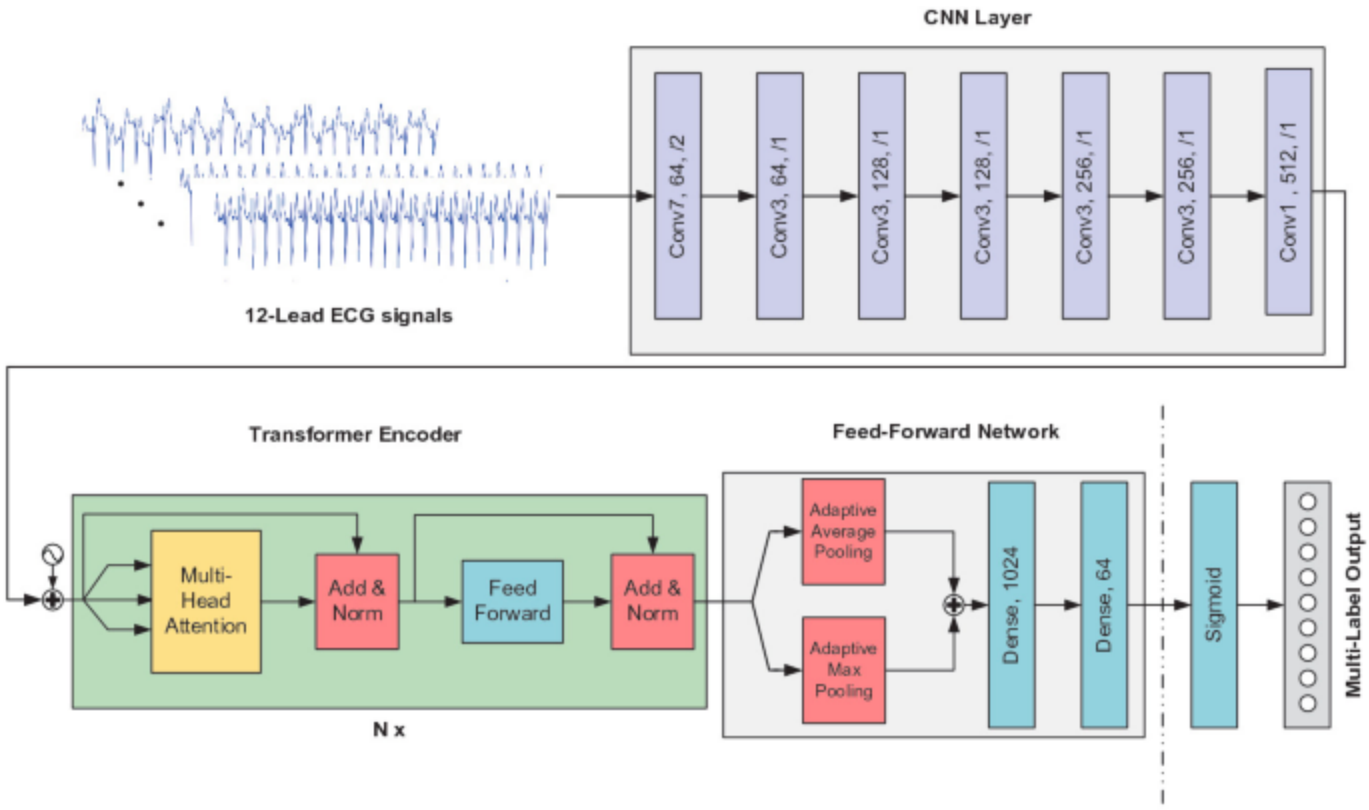
As before, the model is compiled to use Adam optimizer at a learning rate of 0.001, with loss captured by CategoricalCrossEntropy(), along with CategoricalAccuracy, Recall, Precision and ROC-AUC metrics all provided under Keras library.

The idea was to train the model over 50 epochs, but it was observed that the model converges faster, and it starts to overfit somewhere around the 15-25 epoch mark every single time. Hence, the training was cut off after 37 epochs (when the model is already overfit), to not taint the testing accuracy further, and the following metrics were observed:



Transformers

In this section, we describe our implementation of transformers and the associate preprocessing and postprocessing steps. We also discuss our results and next steps. Transformers are sequence-to-sequence unstructured models. That is, like the preceding Recurrent Networks and LSTMs, they simply take as input a sequence of tokens and are able to produce as output another sequence of tokens. This behavior can be trivially adapted for Classification Tasks; while there are many approaches for this, we have chosen to take the one popularized by BERT [14]. Herein, we append or prepend an extra black token to the input sequence. The output token at this position is passed through a final Linear output layer to obtain the final class probabilities.



Preprocessing

Due to the end-to-end nature of the Transformer architecture, no preprocessing is required as such. Many existing Transformer-based ECG classification methods, in fact, use none [13]. However, we considered what we were taught in class regarding preprocessing steps such as normalization. We learned that even though image data can be processed as it is and therefore requires no preprocessing, we should still prefer to apply normalization as it helps stabilize the learning process. In this vein, we perform 2 sets of experiments:

- Using only the raw and unchanged data. No preprocessing is applied whatsoever
- Applying the following 2 preprocessing steps -
    - Min-Max Normalization: This essentially scales the signal values such that they lie within a specific range; this kind of normalization is commonly used with signal-based data.
    - FIR Bandpass Filtering: A Finite Impulse Response filter helps us remove unwanted noise in a stable way; this filter has some properties that make it particularly suitable, such as avoiding compounding errors.

Since the total number of NAN values are very low, we choose to simply remove them. This is a multiclass classification dataset, which means there are cases where a signal has been assigned multiple output classes. In order to simplify our problem (the reasons for which are explained in results and discussion), we convert the dataset into a single class classification problem by

“exploding” these items. That is, if an input X is associated with two labels Y1 and Y2, we replace it with two distinct data items (X, Y1) and (X, Y2).

Implementation

Following previous work, we adopt a CRNN-based architecture for our model [11], which stands for Convolutional Recurrent Neural Networks. This is a combination of Convolutional and Recurrent Neural Networks. When it comes to signals, convolutions have been found to be the most effective for extracting features at early stages. The reason for this is the significant overlap between image and signal processing. Many low-level image processing techniques, in fact, consider images to be compositions of signals of different frequencies and wavelengths. We use VGG-16 as our CNN backbone. While there are more powerful CNNs available now, there is a reason we have chosen to go with a smaller model (explained in Transformer Results Discussion). Once low-level features have been extracted, a Recurrent architecture typically processes the signals as time-series data. In our model, the Recurrent architecture has been replaced by a SENET-inspired Transformer [12]. Without going into specifics, the Squeeze-and-Excitation modules used here are capable of recalibrating attention weights on-the-fly, which allows them to achieve greater performance with fewer parameters. The output from the Transformer module is passed through a final output Fully-Connected layer. Once Softmax has been applied, we can simply obtain the highest class probability as the final prediction.

Results and Discussion

Unsupervised Methods

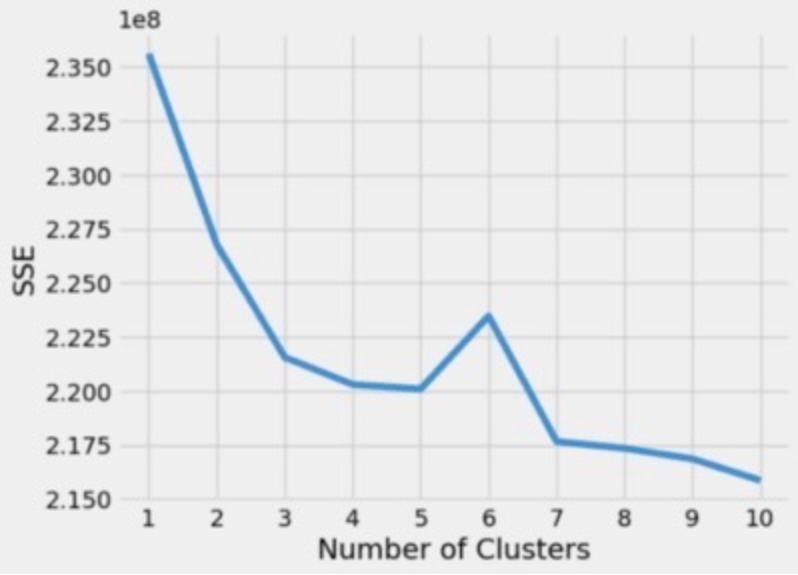
KMeans

In the Elbow method curve, we can see there’s an elbow at number of clusters = 5 and there is a spike right after that, that spike will smoothen out with the number of iterations but it did give us a cluster count of 5 which was accurate. The silhouette score is a score between -1 and 1, larger numbers indicate that samples are closer to their clusters than they are to other clusters. Our value for silhouette score is 0.15 which is low and there is room for improvement. Let’s discuss the performance part of the KMeans algorithm first. The 5 clusters that we got from our model labeled - [0 1 2 3 4] do correlate with our classes - [‘CD’ ‘HYP’ ‘MI’ ‘NORM’ ‘STTC’]. Although trying to figure out which label is which class in unsupervised learn can lead to generalisation issues if it actually works. Even if the clusters/classes are well separated, lack of true labels will prevent you from tuning hyperparameters and ensuring good generalisation. So, it is a theoretically possible concept, but as experienced during this project slightly unlikely to work in

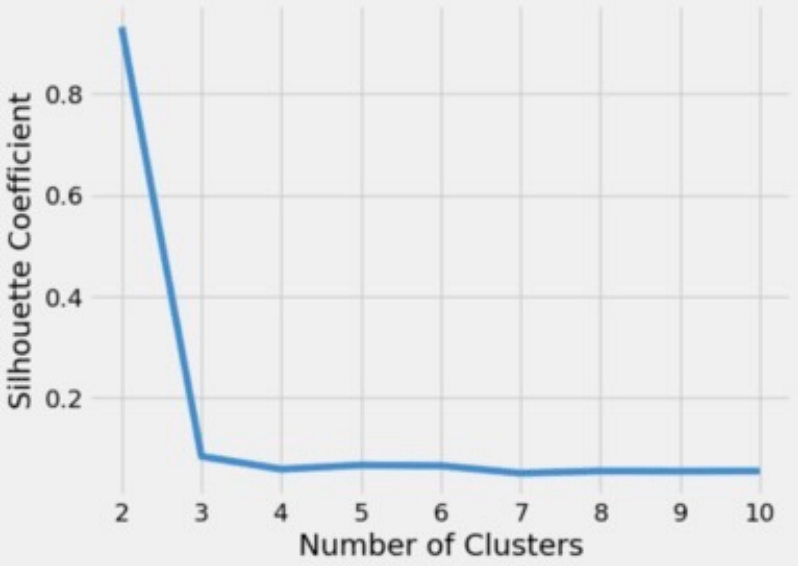
practice.

The time taken of the algorithm to finish training, fitting, predicting depends on a large number of factors, like the size of the dataset. In our case, with a large dataset, we could see KMeans taking quite some time to finish training, fitting, and testing. A future work can also include optimizing the performance of the algorithm run of the same. Instead of using KMeans from sklearn, we could implement a vanilla version or use other python libraries to compare performances.

Elbow Curve (Sum of Squared Error vs Number of Cluster):

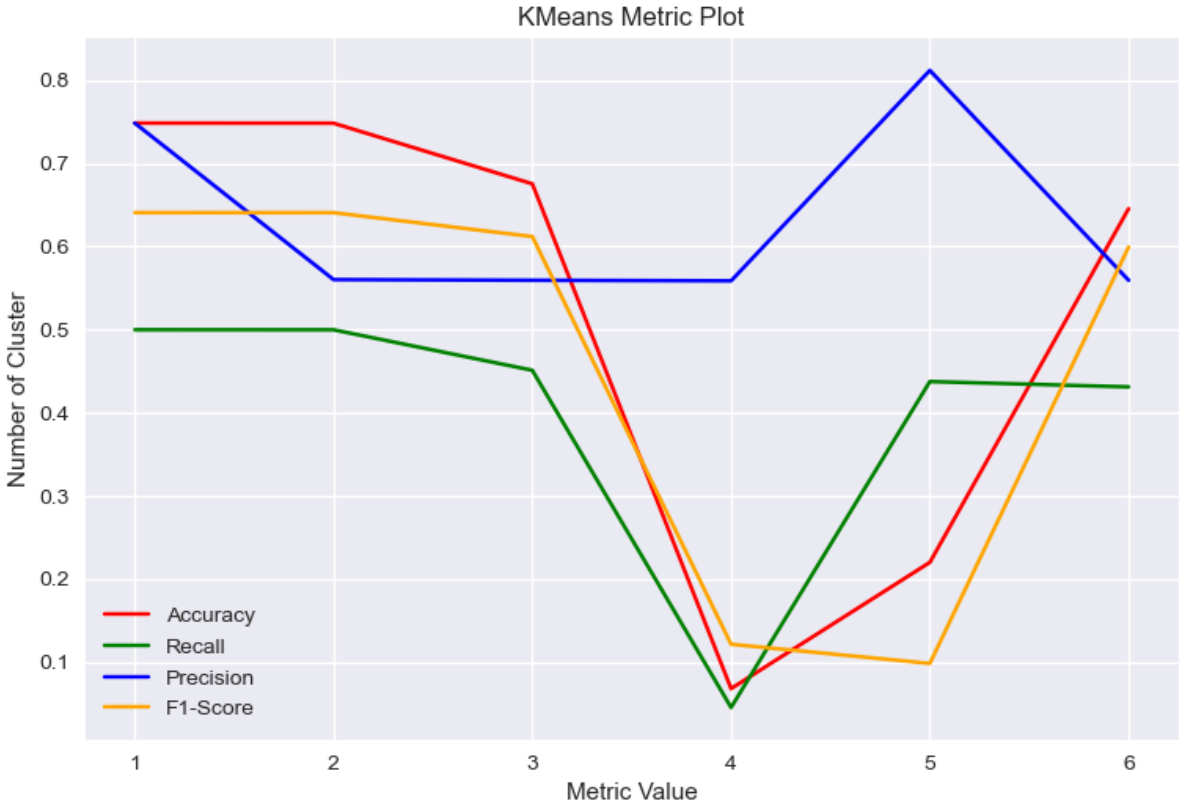


Silhouette Score (Silhouette coefficient vs Number of clusters):



Below is a representation of various testing metrics at different number (1 to 6) of cluster for KMeans, we can see the accuracy is very low but we are getting a high precision value. So the

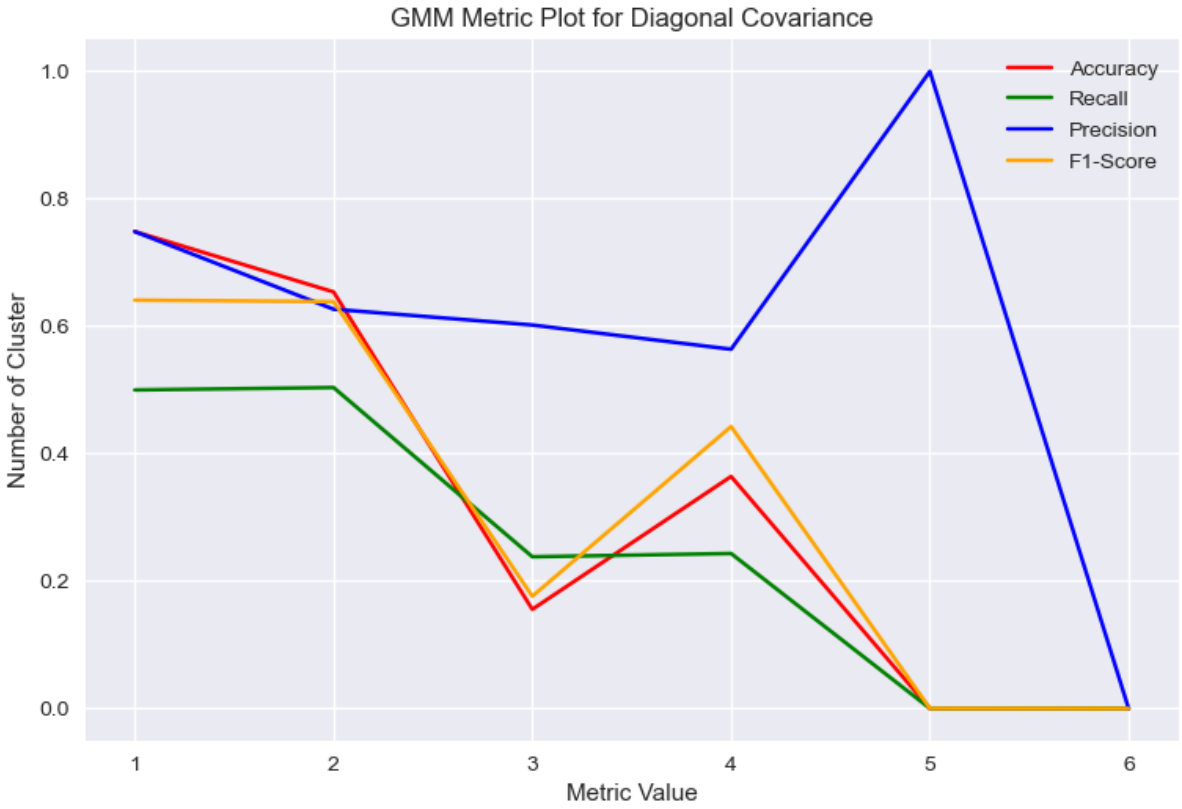
number of points put into the right cluster are less in number but the assignment is correct most number of times.



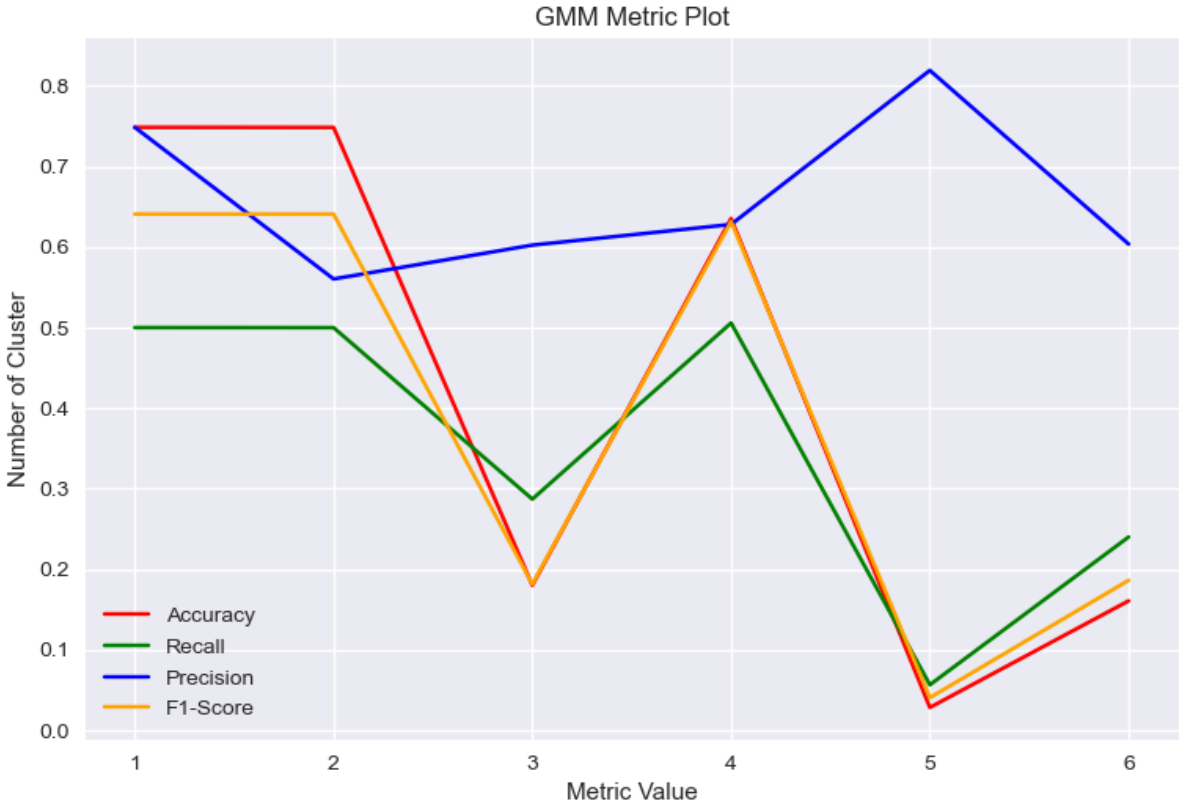
GMM

We used KMeans to find the optimal number of clusters to startup GMM. We also knew beforehand the actual classification of the dataset so it made sense to go with 5 number of clusters. GMM also outputs 5 distinct labels - [0 1 2 3 4] which can correspond to our 5 distinct classes as described in KMeans. GMM diagonal and spherical covariance models took similar time as compared to KMeans to run but GMM with full covariance takes a very large amount of time and doesn't finish running. That was a huge bottleneck as it was expected to be a better result than the methods implemented. This made us resort to diagonal or spherical covariance GMM models which are ofcourse comparable with KMeans in terms of several metrics. Also the fact that GMM with full covariance was not able to converge means that it is difficult to classify this dataset with an unsupervised method like GMM. This can be further proved by the average accuracy, precision, and recall scores found for the model as visualised below.

Various metrics for GMM with Diagonal covariance -

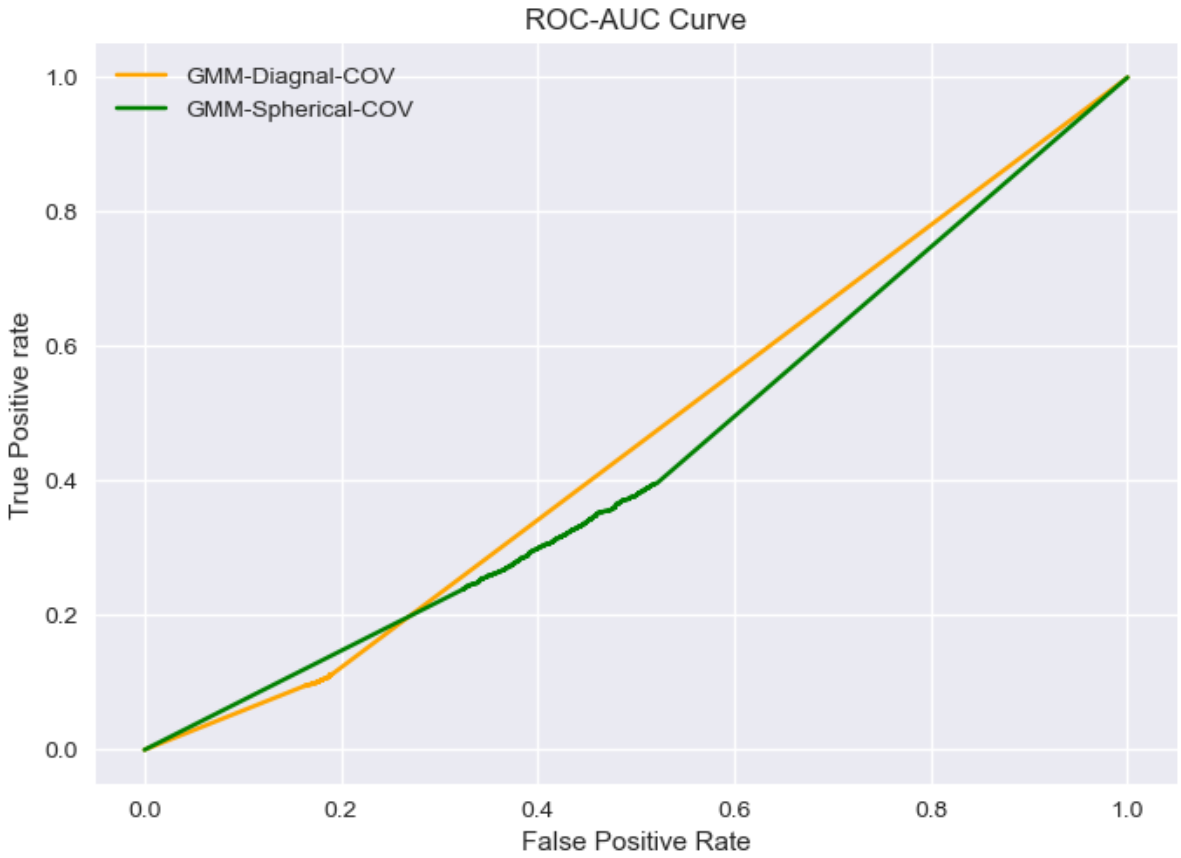


Various metrics for GMM with Spherical covariance -



The ROC AUC score tells us how efficient the model is. The higher the AUC, the better the model's performance at distinguishing between the positive and negative classes. An AUC score

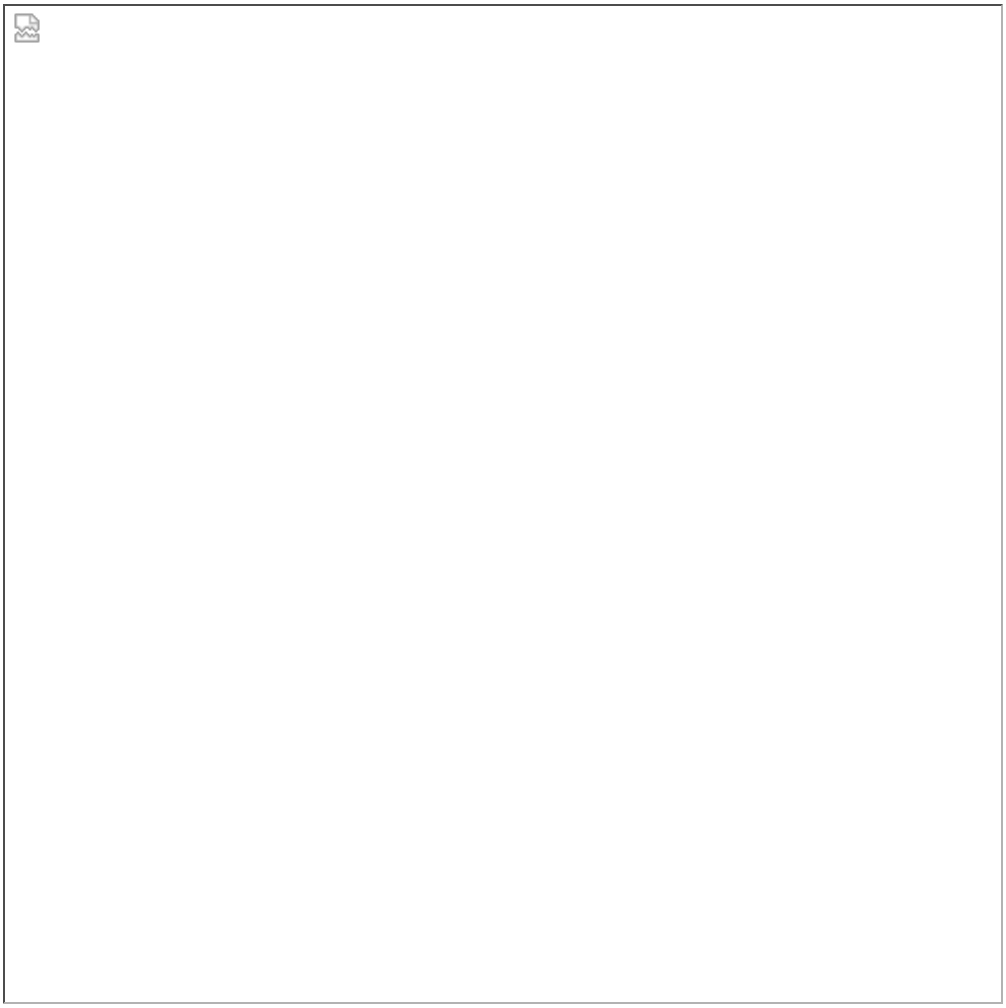
of 1 means the classifier can perfectly distinguish between all the Positive and the Negative class points. Below are the ROC-AUC curves for both the models which show us that they are both equally effective on the ECG dataset -

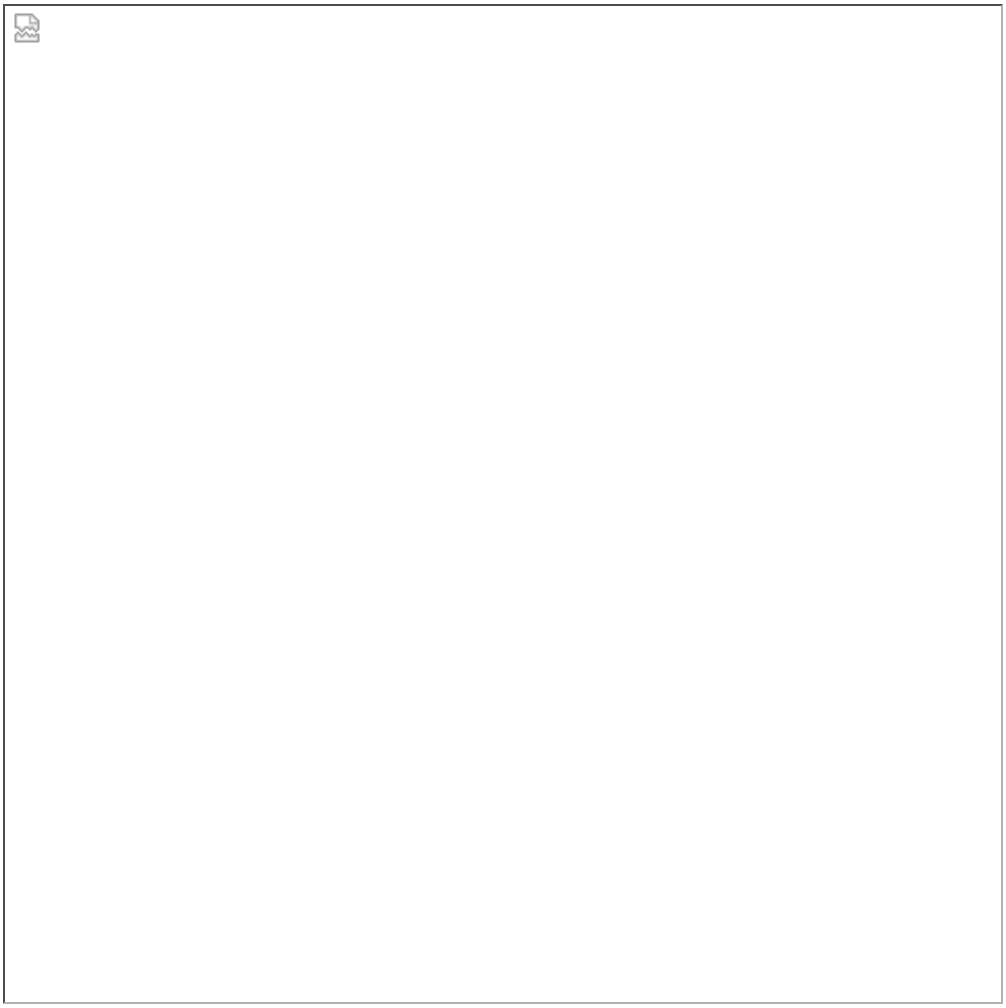


Supervised Methods

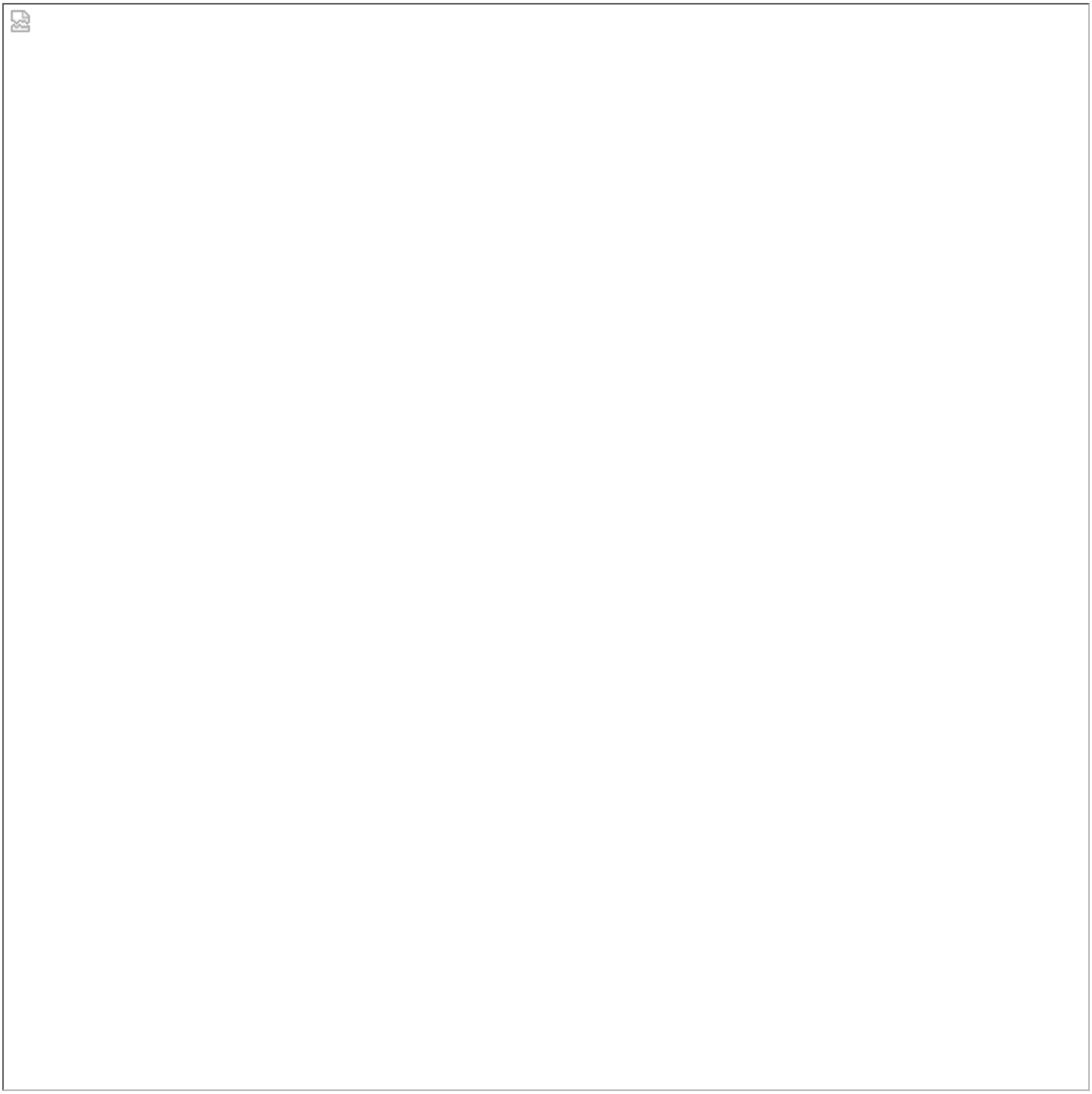
SVM

- Since SVM can handle high dimensional data we first train the model with all features involved. SVM needs data to be in the same scale hence we first transform the values. We got the following accuracies for validation and test set:

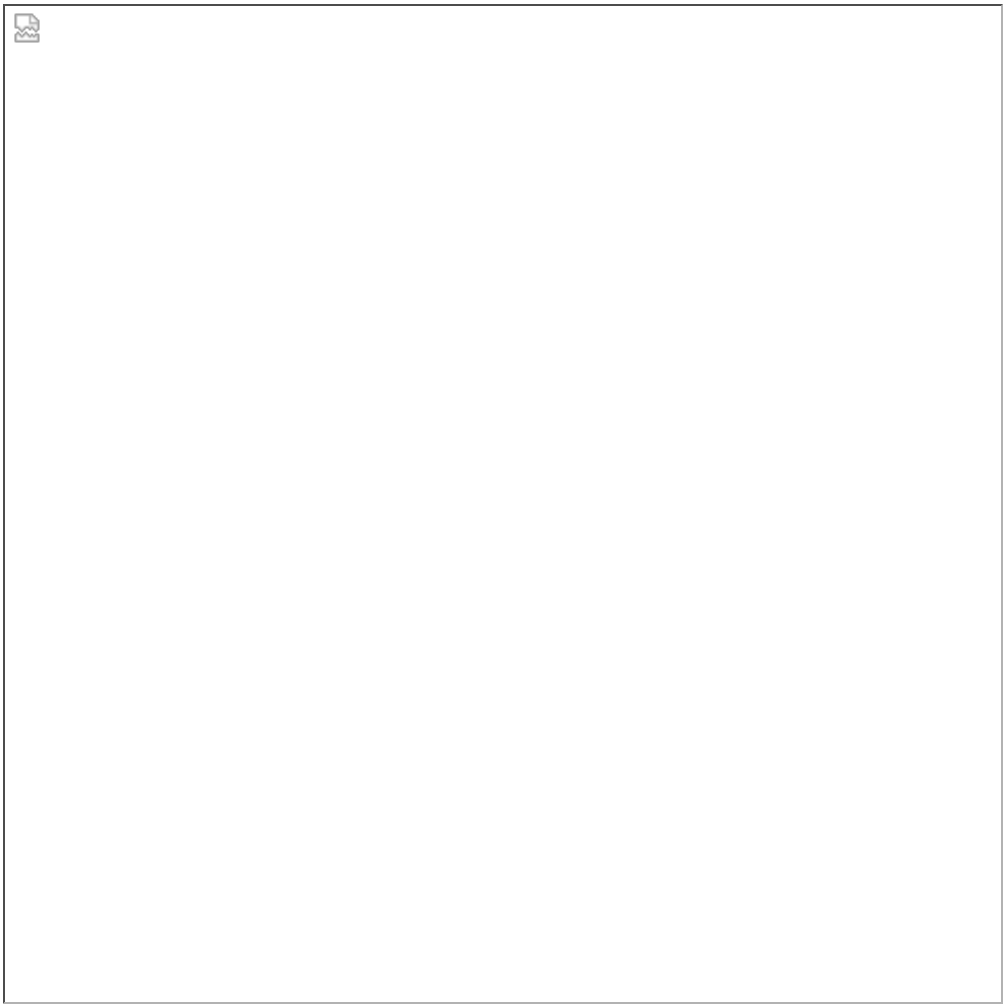


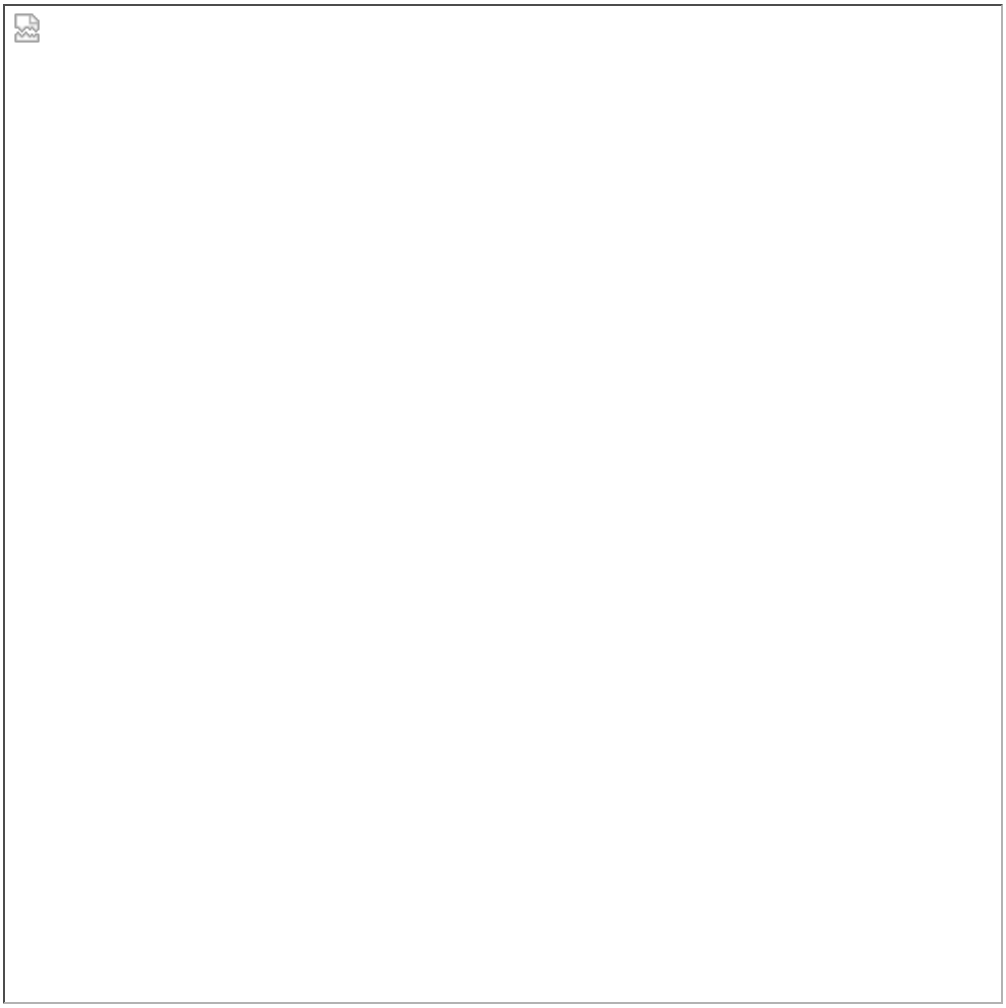


- To eliminate features, we applied Backward Feature Selection using Linear SVM itself. Backward feature selection is a technique used to iteratively remove features from a model to improve its performance or efficiency. For each diagnostic class we noticed two features which were important than others which varied for each class. The feature importance is calculated using the coefficient associated with the feature after training SVM. The values of the feature importance for each class is plotted below.



- Top two features for each class was selected and we train, validate and test the model again considering only these two features according to the class. We notice no drop in accuracy and hence with just two features we can get the same accuracies.





We trained and tested SVM with 3 kernels: linear, RBF, and sigmoid. If you notice on the table below we get quite good accuracies however an f1 score of 0 for linear and rbf kernel and very low scores for sigmoid kernel. Now what's really happening here? Having high accuracy but low f1 score indicates that the model is not effectively capturing the classes. This variation between accuracy and F1 score occurs when dealing with imbalanced datasets, where one class dominates the predictions, leading to high accuracy but poor performance in correctly identifying the minority class. Thus SVM performs quite poorly. Sigmoid kernel does get valid values of f1 score because we can use it as the proxy for neural networks due to it's non linear tanh equation.

Kernel	Diagnostic Class	Accuracy	F1 Score
Linear	NORM	0.67	0.55
	MI	0.75	0
	STTC	0.77	0
	HYP	0.88	0

Kernel	Diagnostic Class	Accuracy	F1 Score
	CD	0.78	0
RBF	NORM	0.66	0.51
	MI	0.75	0
	STTC	0.77	0
	HYP	0.88	0
	CD	0.78	0
Sigmoid	NORM	0.56	0.53
	MI	0.70	0.08
	STTC	0.73	0.14
	HYP	0.77	0.13
	CD	0.72	0.08

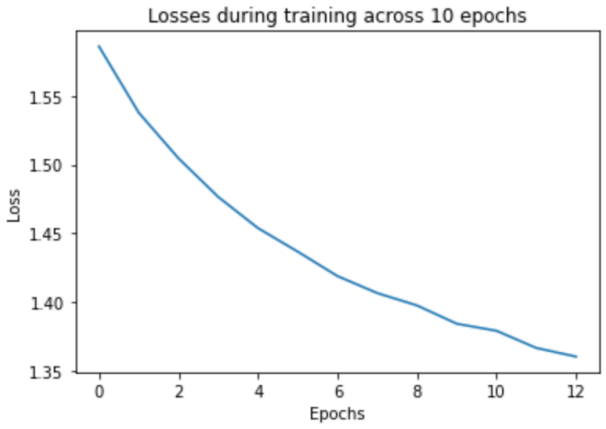
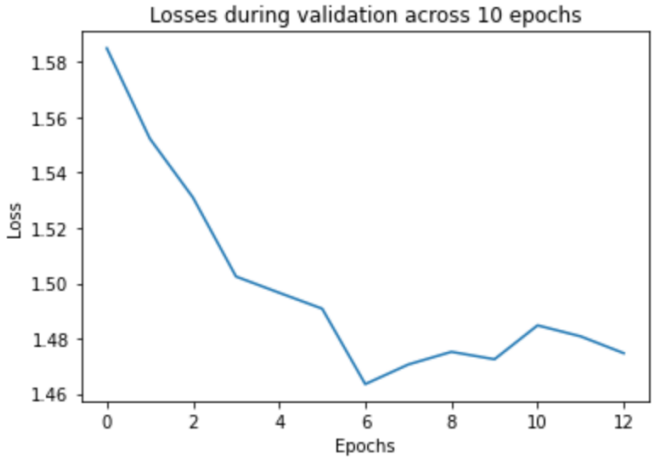
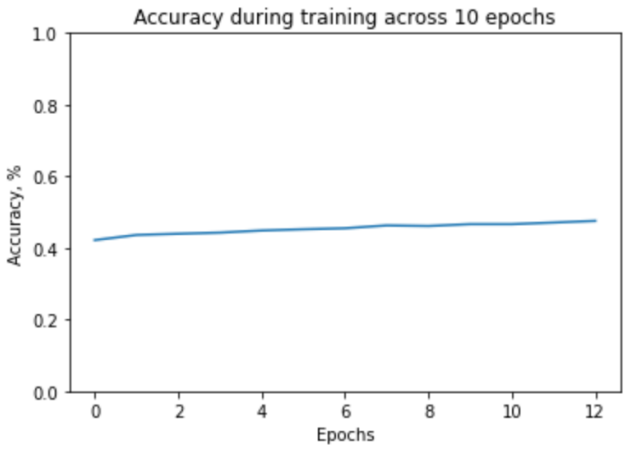
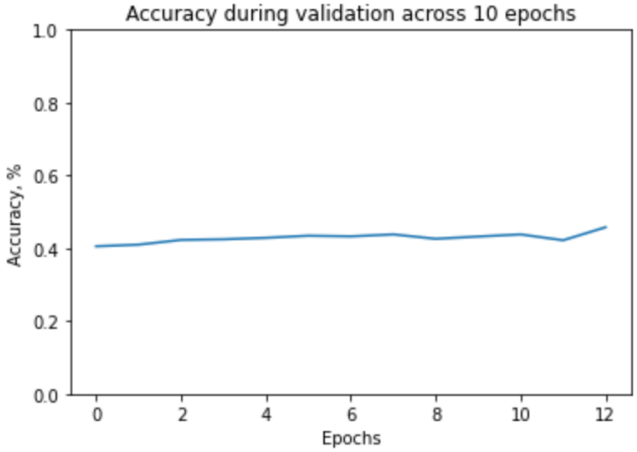
Fully Connected Network (FCN)

Our dataset was split into training (90%) and testing (10%) sets as recommended by the creators of PTB-XL [1]. We use the version of the dataset which labels diagnostic superclasses. After the preprocessing mentioned above has been applied, we end up with 5 distinct output classes - ‘CD’, ‘HYP’, ‘MI’, ‘NORM’, ‘STTC’.

Results

Our final results are shown below for training and testing:

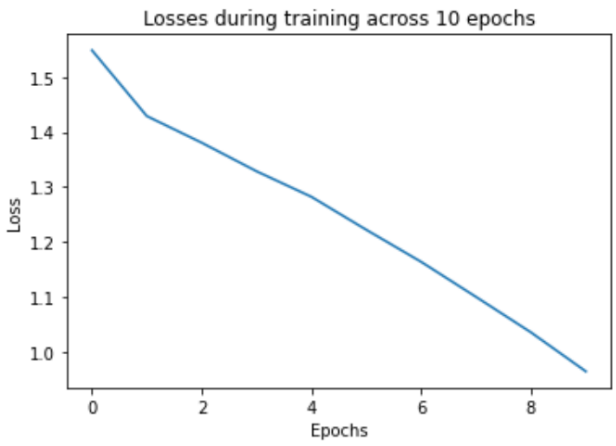
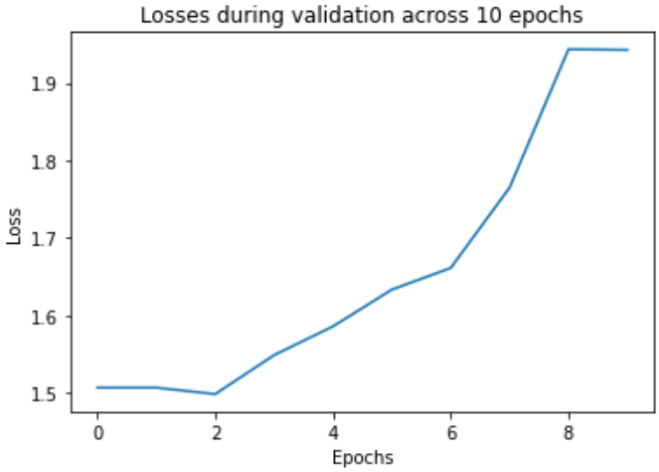
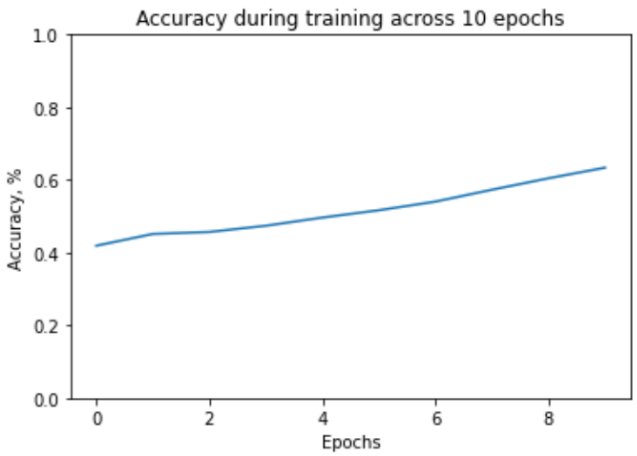
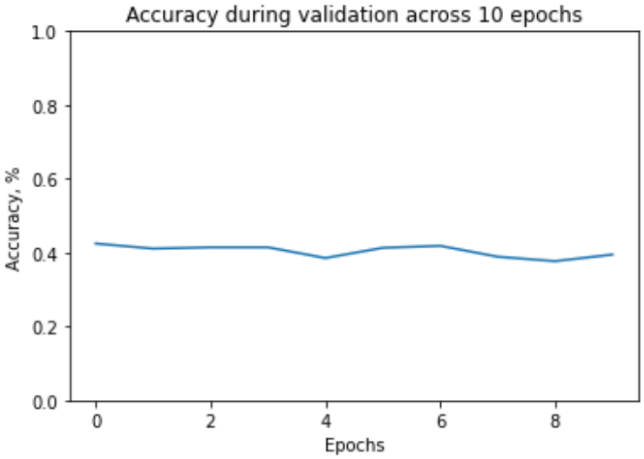
Train Loss	Validation Loss

Train Loss	Validation Loss
	
Train Accuracy	**Validation Accuracy**
	

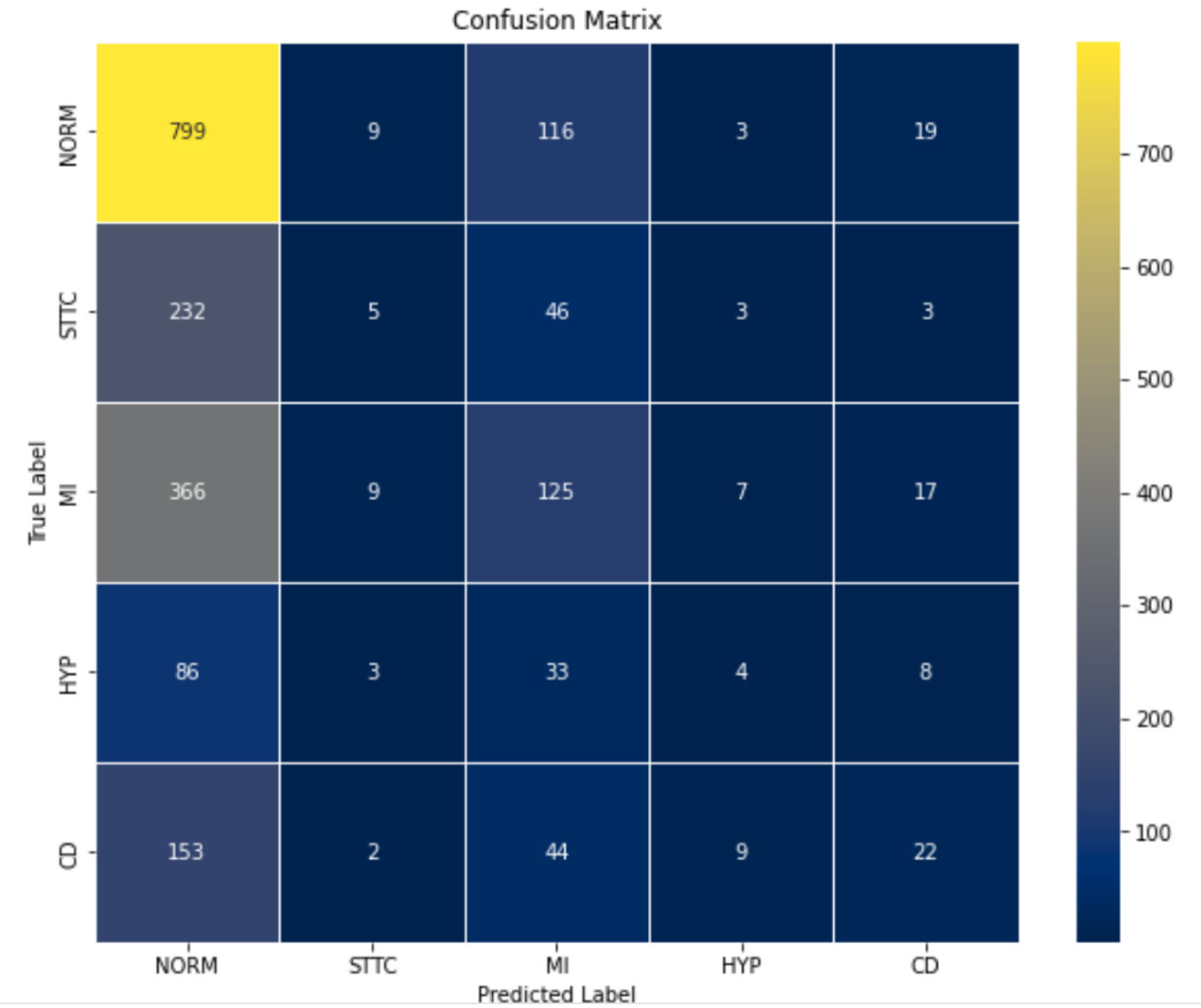
In all, we were able to achieve the following with our best performing FCN architecture and hyperparameters:

Training Accuracy	Validation Accuracy	F1 Score	Weighted Average AUC Score
0.448	0.458	0.292	0.317

We were able to mitigate overfitting by simplifying the architecture, reducing the number of parameters in our fully connected layers, increasing our dropout probability, and implementing early stopping. The plots above are very different from the plots seen during our experimentation as the validation loss trends in the same direction as our training loss with both accuracy of our training and validation increasing over epochs. We can see the comparison with a representative sample shown below:

Train Loss	Validation Loss
	
Train Accuracy	**Validation Accuracy**
	

Our confusion matrix for the different diagnostic classes can be seen below:



Further, the metrics for each diagnostic class are shown in the table below:

Diagnostic Class	Precision	Recall	F1-Score
Normal	0.48	0.84	0.62
STTC	0.17	0.02	0.03
MI	0.41	0.31	0.36
HYP	0.12	0.04	0.06
CD	0.22	0.11	0.15

The model performs very well for the “Normal” class in terms of precision and recall. It’s F1-score is also satistfactory. The “STCC” and “HYP” classes have low precision and recall, indicating poor performance for these classes. The “CD” class has moderate precision but low recall. The “MI” class has reasonable precision but lower recall.

Discussion

In our investigation of the Fully Connected Network architecture on the PTB-XL dataset, we see a nuanced performance across the different diagnostic classes. Overall, the model achieved an acceptable accuracy on the validation set (0.458), which is much better than chance (0.167); however, the F1-score suggested challenges in correctly classifying instances across all diagnostic classes. Digging in deeper via the confusion matrix, we see this is confirmed as “STCC” and “HYP” in particular are not classified well by our model.

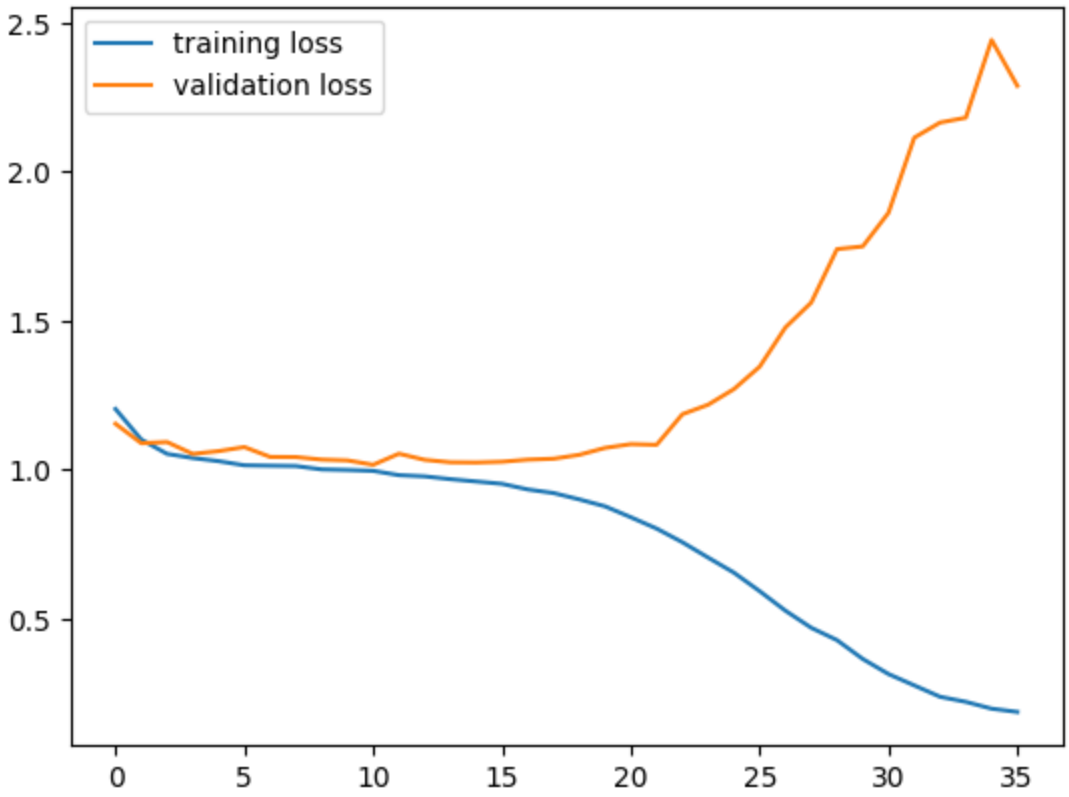
Another issue we saw in implementing and training a FCN was overfitting. As mentioned before, we were able to mitigate overfitting by simplifying the architecture, reducing the number of parameters in fully connected layers, increasing dropout probability, and implementing early stopping. We recommend future implementations of FCNs on the PTB-XL dataset for take these into considerations.

In all, FCN is a viable way to predict ECG diagnostic classes given adequate training and model simplification to prevent overfitting. Nonetheless, some of the other methods used in this study have a much better accuracy rate with the trade-off being higher levels of complexity.

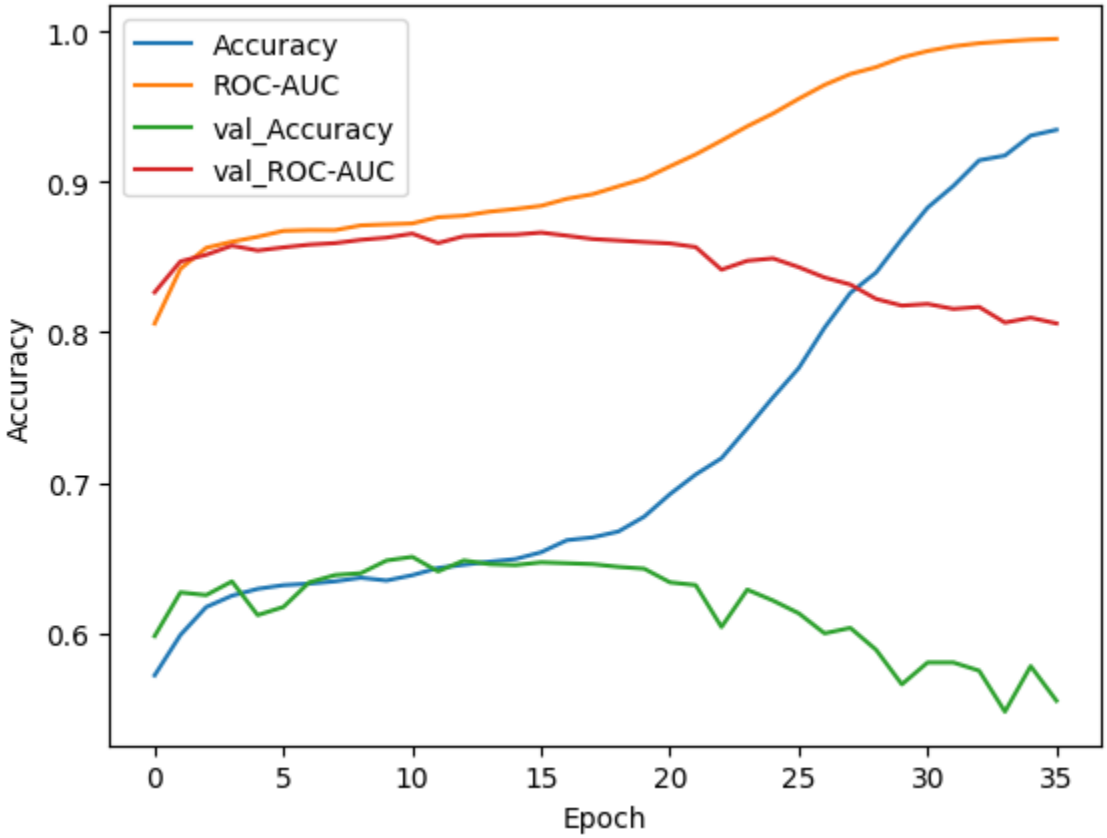
LSTM

As previously mentioned, the model starts to overfit around the 15-25 epoch mark, which is clearly visible in the metrics being captured:

- Training loss goes close to 0, while the validation loss shoots up.

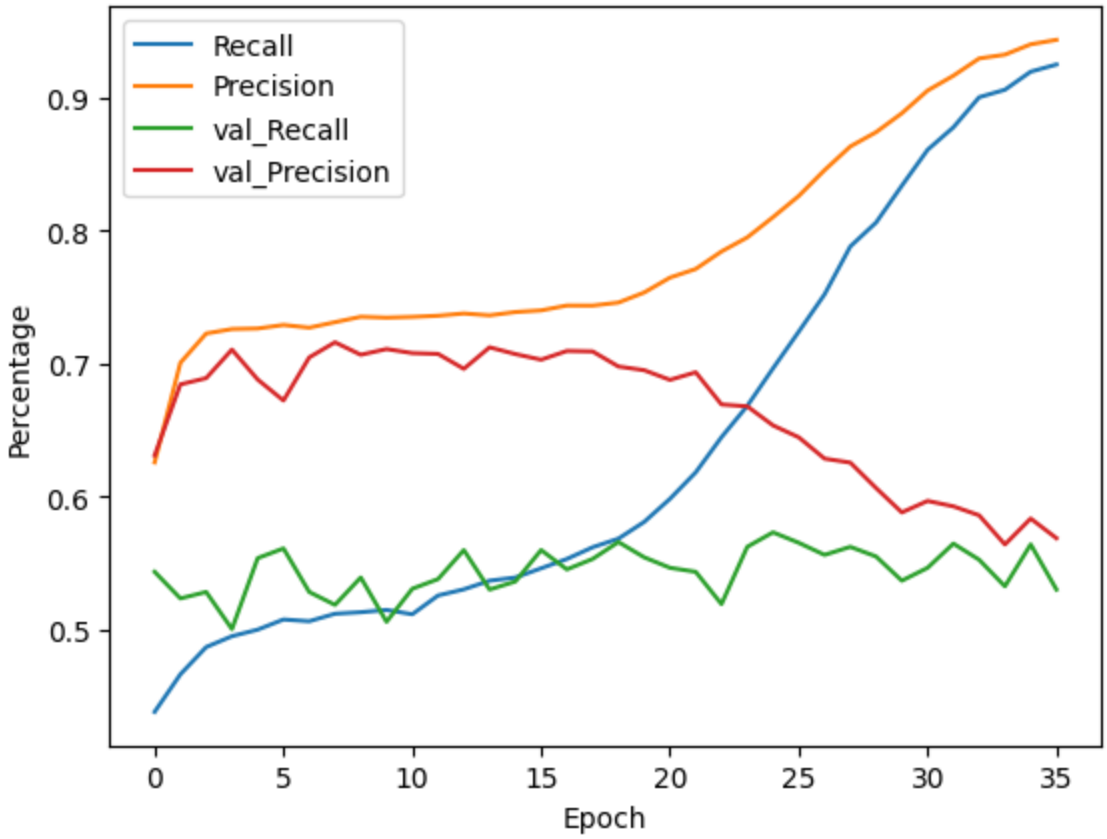


- Accuracy goes over 90% and ROC-AUC goes to 0.9998, while the validation counterparts peak around epoch 15 and decline.



- Similarly, Recall and Precision training values rocket towards 100% while validation metrics decline after peaking around epoch 15. Curiously, Recall validation values still hover

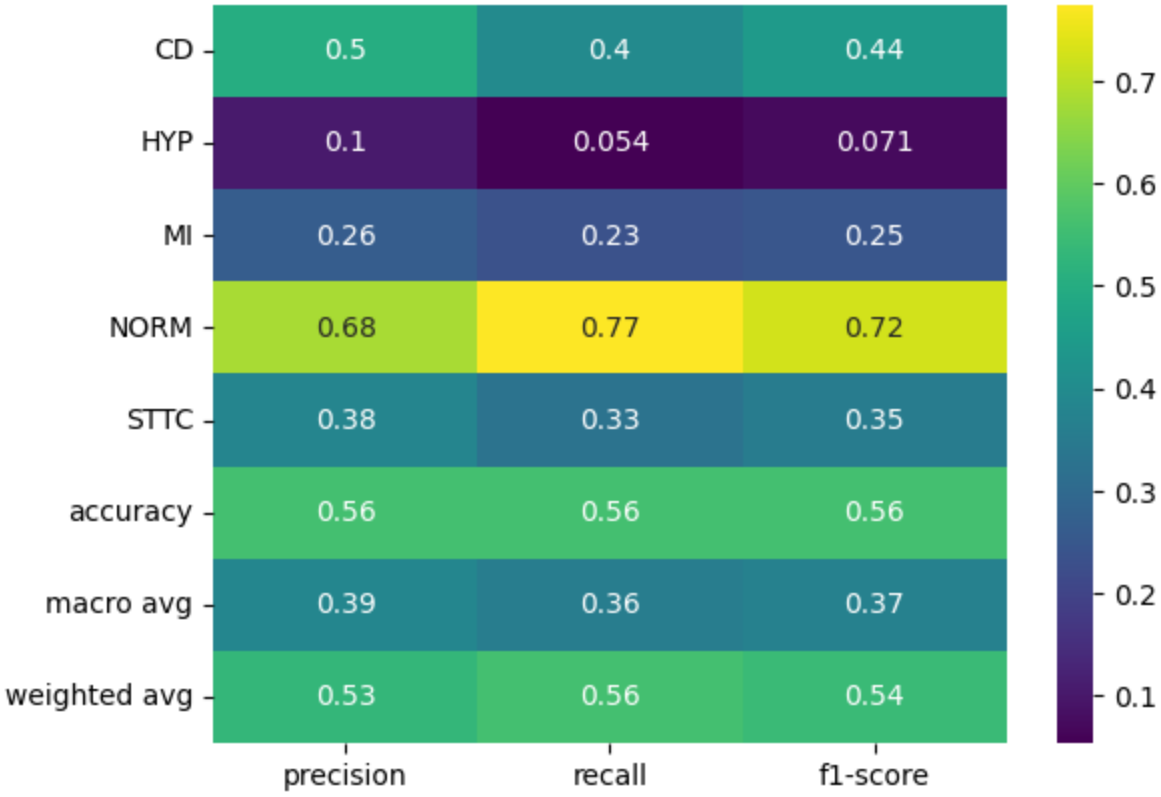
around the same 55% mark even after overfitting.



Stopping the model training during the 37th epoch, we proceed to run a prediction test for strat_fold[10] datapoints as split earlier to obtain predictions as one-hot encoded values for 1652 datapoints.

Upon getting predicted results as probabilities, we selected the column with the max probability and compared it to the ground truth labels provided in y_test.

The following accuracy metrics were observed upon creating a confusion matrix for y_test and y_pred:



This suggests that while we can classify normal ECG readings fairly accurately, it is much more challenging to identify other diagnostic superclasses individually and exclusively.

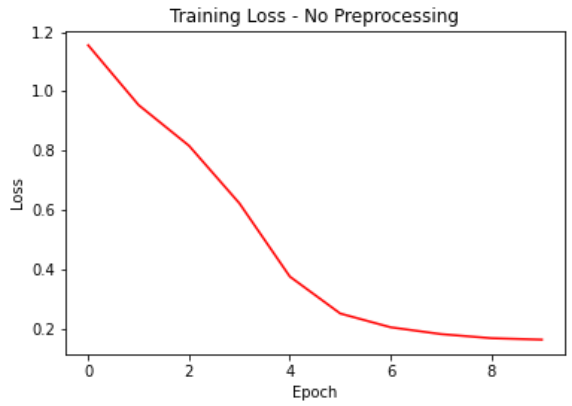
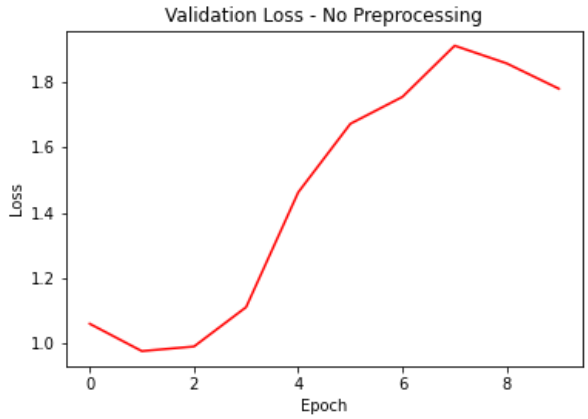
Transformers

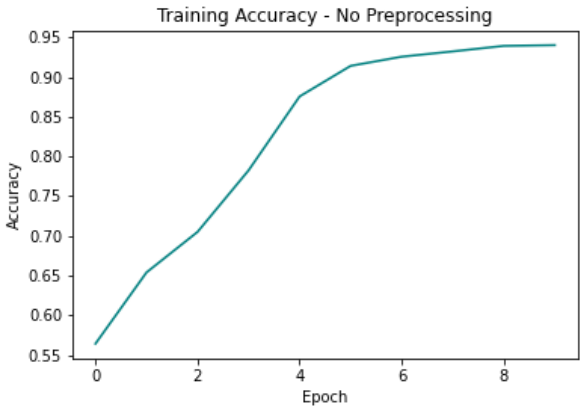
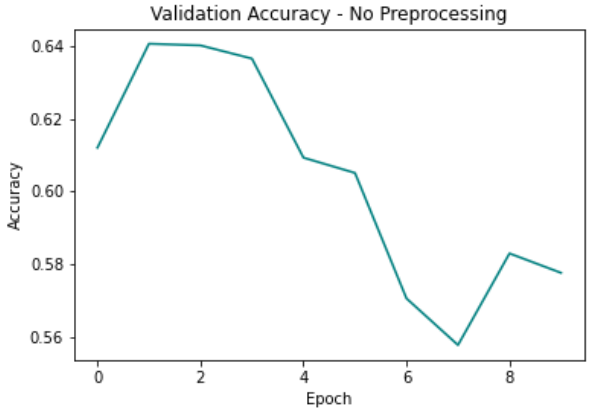
We first split our dataset into training (90%) and testing (10%) sets. We use the version of the dataset which labels diagnostic superclasses. After the preprocessing mentioned above has been applied, we end up with 5 distinct output classes - 'CD', 'HYP', 'MI', 'NORM', 'STTC'. For our training process, we use CrossEntropy Loss with the Adam optimizer. Based on observation, we decide on a learning rate of 0.0001. We plan to more systematically discover an appropriate learning rate before the final report. With a Batch Size of 10, we train our models for 10 epochs each.

Results Without Preprocessing

For the training set, we observe the loss steadily decreasing and accuracy increasing as expected. However, for the test set, we observe that the loss initially decreases but then shoots up in the later epochs. The accuracy follows a similar pattern, increasing initially and then rapidly falling.

Train Loss	Validation Loss

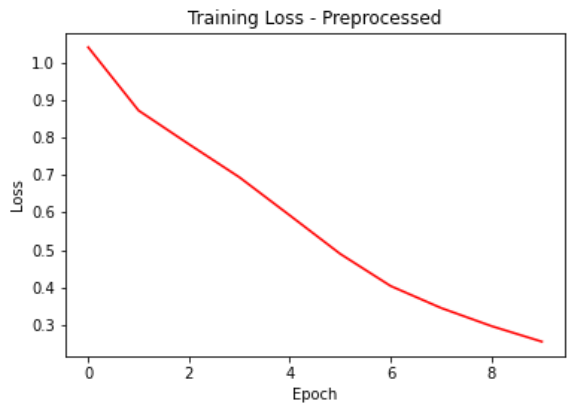
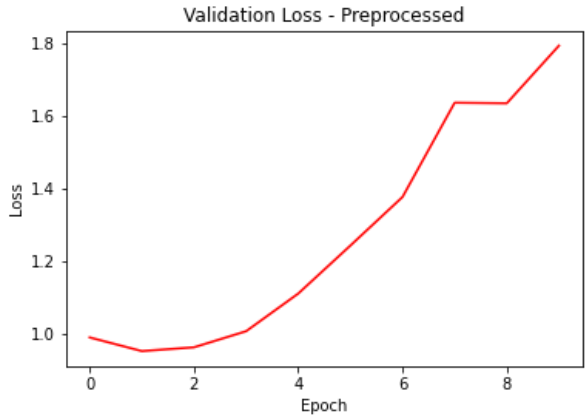
Train Loss	Validation Loss
	

Train Accuracy	Validation Accuracy
	

Results With Preprocessing

We observe similar behavior in the preprocessed set.

Train Loss	Validation Loss

Train Loss	Validation Loss
	

First, let us discuss the relative performance of the two experiments. If we look at the accuracy plots, we see that the model achieves higher training accuracy on the dataset without preprocessing. However, when it comes to testing accuracy, we observe that the preprocessed data leads to a higher accuracy. From this we can deduce that preprocessing reduces overfitting to some extent.

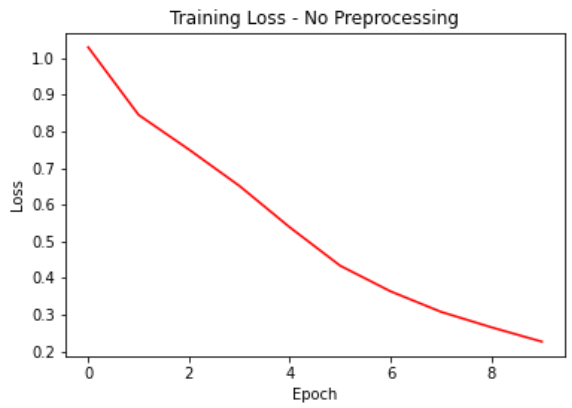
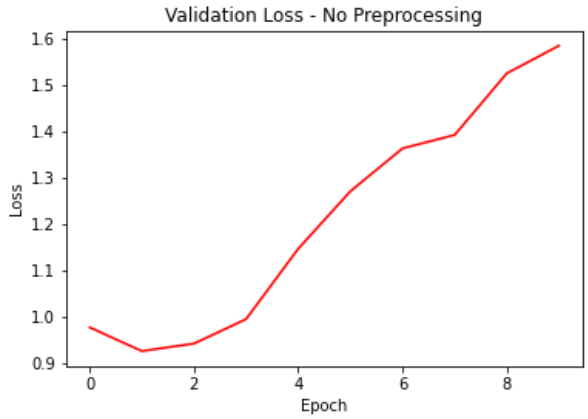
Now, the chief issue with the current training procedure is that there is a great degree of overfitting clearly visible. As training accuracy increases, testing accuracy decreases for both experiments. This is in fact the reason we have chosen to keep both the CNN backbone and Transformer as simple as possible. This is also our reason for converting the task into single class classification; since this limits the degrees of freedom and also provides us with a little more data.

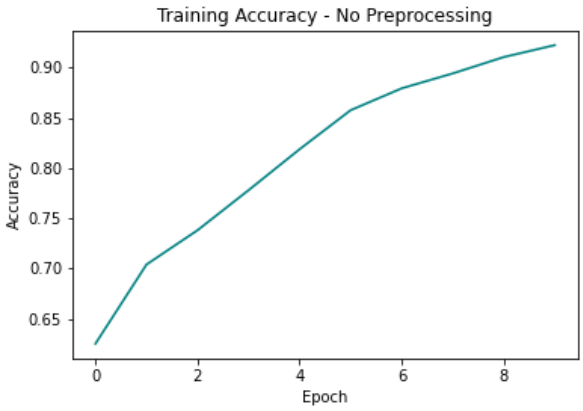
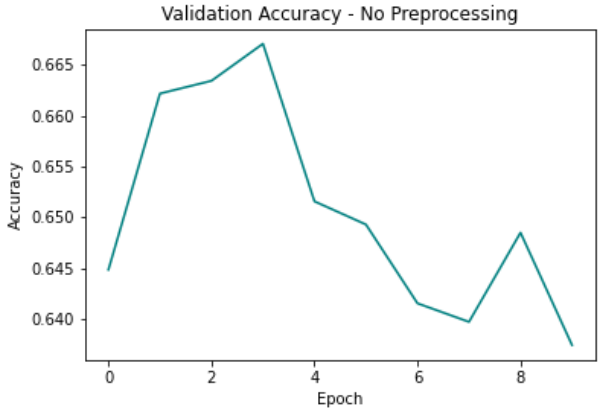
Based on the above reasoning, we performed another set of experiments with regularization added to the model in the form of Dropout layers. The results are elaborated below.

Regularization - Without Preprocessing

For the training set, the behavior in both the loss and accuracy plots remains nearly unchanged, as is expected. For the validation set, however, we see an interesting change. The overall trend remains the same. However, we can see that the absolute values, compared to the previous set of experiments, have improved quite a bit. Validation accuracy, at its maximum, is now 0.665 as opposed to 0.64 without regularization. The overfitting trend is also much less exaggerated - we can see that validation accuracy fluctuates only within the 0.64-0.665 range. Without regularization, this range was 0.56-0.64.

Train Loss	Validation Loss

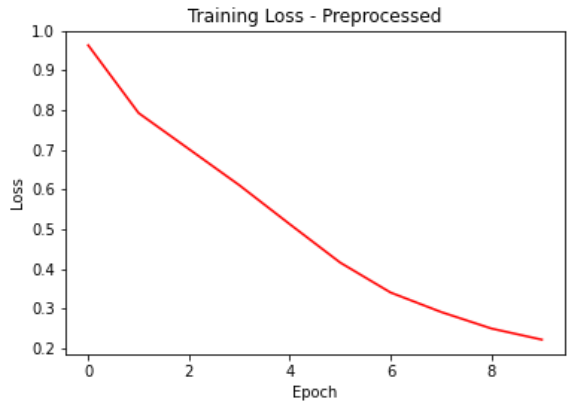
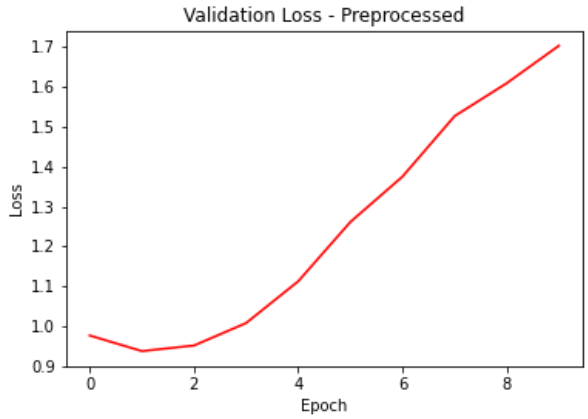
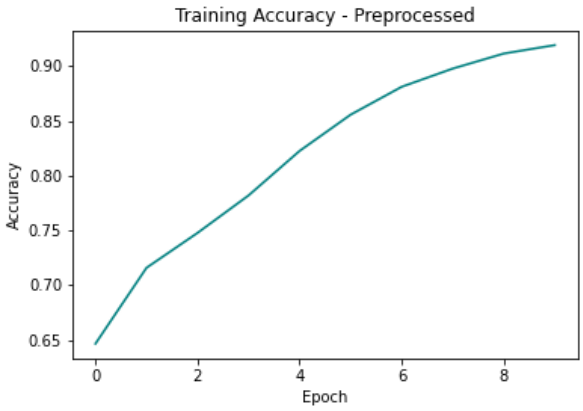
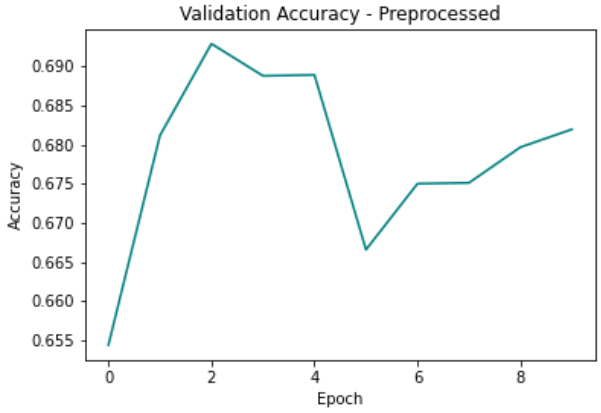
Train Loss	Validation Loss
	

Train Accuracy	Validation Accuracy
	

Regularization - With Preprocessing

The same improvements are observed with preprocessing. In purely numeric terms, this experiment set performs the best, achieving a maximum validation accuracy of more than 0.69.

Train Loss	Validation Loss

Train Loss	Validation Loss
	
Train Accuracy	**Validation Accuracy**
	

Based on these results, we can say that regularization was certainly successful to some extent. However, the problem of overfitting remains - the reason for this is possibly that transformers are simply too elaborate a model for the task in question.

We report here the quantitative metrics from the final (regularized) model:

On the training dataset:

	Accuracy	F1 Score	AUC Score
Without Preprocessing	0.834	0.838	0.971
With Preprocessing	0.833	0.837	0.972

On the validation dataset:

	Accuracy	F1 Score	AUC Score
Without Preprocessing	0.621	0.625	0.869
With Preprocessing	0.641	0.643	0.880

Comparison and Analysis

Unsupervised Methods

In this section, we compare the Unsupervised methods that we implemented for this ECG dataset, namely, KMeans, Gaussian Mixture Model (GMM).

We use several metric to compare the performance of the models like below:

- **Accuracy score** - It represents a percentage of correct predictions made by a model.The higher the better.
- **Precision score** - It is is the fraction of relevant instances among the retrieved instances. The higher the better.
- **Recall Score** - It is the fraction of relevant instances that were retrieved. A score of 1 means perfect clustering.
- **f1-Score** - It is calculated from the precision and recall of the test, where the precision is the number of true positive results divided by the number of all positive results, including those not identified correctly, and the recall is the number of true positive results divided by the number of all samples that should have been identified as positive. A value of 1 means perfect modelling.
- **Rand score** - It has a value between 0 and 1, with 0 indicating that the two data clusterings do not agree on any pair of points and 1 indicating that the data clusterings are exactly the same.
- **Davis-bouldin index** - This is an internal evaluation scheme, where the validation of how well the clustering has been done is made using quantities and features inherent to the dataset. A lower value means better clustering.
- **Calinski-Harabasz Index** - It is an internal evaluation metric, where the assessment of the clustering quality is based solely on the dataset and the clustering results, and not on external, ground-truth labels. Higher value indicates better clustering.

Based on the above metrics, we present an analysis as below:

Metric	KMeans	GMM (Diagonal)	GMM (Spherical)

Metric	KMeans	GMM (Diagonal)	GMM (Spherical)
Accuracy Score	0.39	0.42	0.51
Precision Score	0.38	0.53	0.61
Recall Score	0.31	0.54	0.68
f1-score	0.33	0.54	0.64
Rand Score	0.42	0.51	0.66
Davis-Bouldin Index	65.25	58.88	49.65
Calinski-Harabasz Index	113.72	216.44	312.63

As per the above metric, GMM (Spherical Covariance) is the best performing model on the given ECG Dataset compared to KMeans or GMM (Diagonal Covariance). GMM (Spherical) > GMM (Diagonal) > KMeans

Supervised Methods

Here, we will compare the 3 deep-learning based Supervised Methods: Fully Connected Networks, Long Short Term Memory (LSTM) and Transformers. For these methods, we will focus primarily on 3 metrics - Accuracy, F1 Score, and the AUC Score (that is, the Area Under the ROC Curve).

We report performance on the validation set. For the Transformer model, the variant with preprocessing is used.

Metric	FCN	LSTM	Transformer
Accuracy Score	0.458	0.555	0.641
F1 Score	0.292	0.54	0.643
AUC Score	0.317	0.806	0.880

From these metrics we can conclude for now that (CRNN) Transformers are the best performing model architecture on our dataset, despite the great degree of overfitting observed. Another interesting observation is that GMM Clustering is in fact able to outperform the Fully Connected Network, despite not being well adapted for the task. This indicates that there is still quite some

latent information present in ECG data (that can be picked up by an unsupervised model) that can be exploited to improve the supervised models.

In both the LSTM and Transformer models, we observe a great degree of overfitting. Overfitting is also present in the Fully Connected Network, but to a much lesser extent due to simplification of the model architecture and much heavier dropout. We believe that the reason for high overfitting in LSTM and Transformers is the emphasis on temporal modeling and the complexity of the models themselves. Both models attempt to identify and learn trends across the temporal dimension. However, our dataset only contains 10-second ECG waveform records. While we have a large number of datapoints, our dataset is in fact quite limited temporally.

Future Work

Unsupervised Methods

Although GMM was able to handle the dataset and classify it with a moderate accuracy score, the search for the perfect model or the perfect setup for the current models continue. Not many performance or accuracy benefits were achieved after applying feature reduction techniques either. This is supported by the current research work being not just KMeans or GMM for example, but an enhanced model including KMeans and other models like K-SVD [16] which have a proven accuracy rate of over 90%. So, the current choice of models aren't the way to go but they seem like an excellent starting point to build the final model upon in the future.

- A future work can include optimizing the performance of the various model runs.
- Implementing GMM with full covariance looks like a far shot but exploring it is worth a shot as well. To check if there are any drastic improvements given that it does complete the run on this dataset.
- We also aim to apply other unsupervised learning models like the below to our dataset and compare the results we got from KMeans and GMM to draw better and more accurate conclusions.
    - DBScan
    - Hierarchical Clustering (HC)

FCN

Despite recommendations by the PTB-XL research group to stratify training, testing, and validation sets into 10 groups, it may be useful to investigate techniques that oversample minority classes or adjust class weights during training to help enhance model sensitivity to underrepresented classes. We recommend investigating these as potential solutions that may

also help mitigate overfitting.

Moreover, feature engineering may be a useful technique employed here to capture more nuanced patterns in ECG data, especially for problematic classes like “STTC” and “HYP”. Additional hyperparameter tuning may also help identify settings that can improve generalizability and performance across all classes.

In conclusion, while the FCN shows promise in certain diagnostic classes, addressing specific challenges, particularly in underrepresented classes, may be important for enhancing the model’s clinical applicability and reliability. Nonetheless, we believe that FCN can be a viable method despite concerns regarding overfitting in the classification of ECG data, particularly for those use cases where accuracy can be sacrificed for a more simplistic model.

LSTM

We can clearly observe a very high chance of overfitting in model training over 15 epochs, while the validation metrics stay the same, which suggests that the model architecture needs to be tuned further in order to capture some essential data features we or the model seem to be missing. We can modify the sizes of the kernels, and experiment with larger LSTMs to see if we can gain accuracy at the expense of computational costs.

Additionaly, as mentioned previously, while we can classify normal ECG readings fairly accurately, it is much more challenging to identify other diagnostic superclasses. It might be a worthy experiment to see if we can club the other diagnoses as an ‘abnormal’ label and see if we can achieve better accuracy with binary classifications instead.

It can be a very useful tool as a ‘precursor’ to a cardiovascular checkup, where the physician’s load can be reduced in case a patient gets a ‘normal’ diagnosis. The caveat being, the model accuracy needs to be extremely high, and needs to avoid falsely identifying abnormal readings as normal at all costs. Unless that can be achieved, it is too great a risk in the medical domain.

Transformers

Even with our application of regularization, we observe a great degree of overfitting in the Transformer model. Despite this overfitting, Transformers were observed to be the highest-performing model on the dataset. This indicates that there must be room for improvement. There are some directions of work we could explore to try and mitigate overfitting and make our model more suited to the dataset at hand.

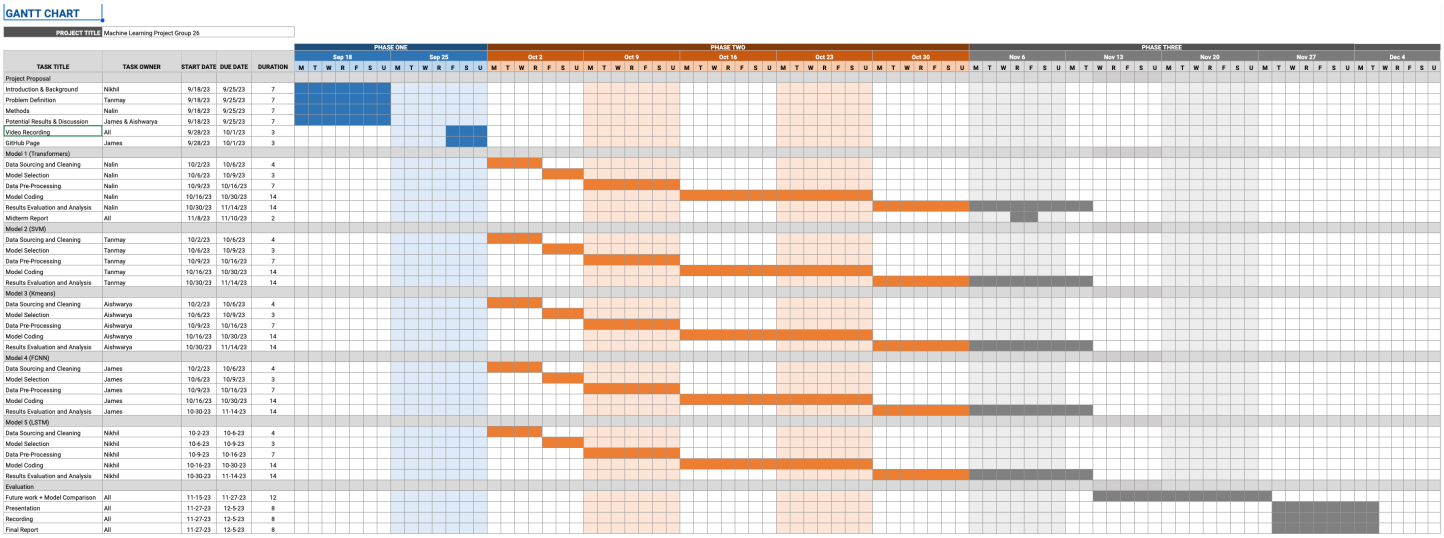
Our first concern would be the preprocessing methods used. Applying Normalization and Bandpass Filtering on the data had a clear positive impact on model performance. Here, we have

only explored these basic techniques. Future work could see us systematically exploring and comparing a wide variety of common preprocessing techniques in tandem with transformers.

Another area of interest is the Convolutional Network used in the early stages of the model. The CRNN architecture was pioneered primarily when Transformers were in early stages of research. Since then, Transformers have been better adapted for vision processing in the form of Vision Transformers (ViT). If we are to extend the analogy between signal and image processing, exploring the effectiveness of Vision Transformers directly as opposed to CRNNs could be an interesting line of work.

GANTT Chart

Google Sheets Link



Contribution Table

Member Name	Contribution
Nikhil Sachdeva	LSTM, Introduction, Dataset Description
Tanmay Shishodia	SVM, GANTT Chart
Nalin Semwal	Transformers, Supervised Methods Comparison
James Jun	FCN, GitHub Pages Preparation

Member Name	Contribution
Aishwarya Solanki	KMeans, GMM, Introduction, Dataset Description, Unsupervised Methods Comparison

References

[1] S. H. Jambukia, V. K. Dabhi, and H. B. Prajapati, “Classification of ECG signals using machine learning techniques: A survey,” in 2015 International Conference on Advances in Computer Engineering and Applications, Ghaziabad, India, 2015, pp. 714-721, doi: 10.1109/ICACEA.2015.7164783.

[2] P. Wagner, N. Strodthoff, R. D. Bousseljot, et al., “PTB-XL, a large publicly available electrocardiography dataset,” in Sci Data, vol. 7, p. 154, 2020, doi: 10.1038/s41597-020-0495-6.

[3] ISO Central Secretary, “Health informatics – Standard communication protocol – Part 91064: Computer-assisted electrocardiography,” Standard ISO 11073-91064:2009, International Organization for Standardization, Geneva, CH, 2009.

[4] E. J. da S. Luz, W. R. Schwartz, G. Cámara-Chávez, and D. Menotti, “ECG-based heartbeat classification for arrhythmia detection: A survey,” in Computer Methods and Programs in Biomedicine, vol. 127, pp. 144–164, 2016.

[5] S. Hochreiter and J. Schmidhuber, “Long Short-Term Memory,” in Neural Computation, vol. 9, no. 8, pp. 1735–1780, Nov. 01, 1997, doi: 10.1162/neco.1997.9.8.1735.

[6] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, and I. Polosukhin, “Attention is all you need,” in Advances in neural information processing systems, vol. 30, 2017.

[7] S. Aziz, S. Ahmed, and M. S. Alouini, “ECG-based machine-learning algorithms for heartbeat classification,” in Sci Rep, vol. 11, p. 18738, 2021, doi: 10.1038/s41598-021-97118-5.

[8] S. Śmigiel, K. Pałczyński, and D. Ledziński, “ECG Signal Classification Using Deep Learning Techniques Based on the PTB-XL Dataset,” in Entropy, vol. 23, no. 9, p. 1121, Aug. 2021, doi: 10.3390/e23091121.

[9] C. Che, P. Zhang, M. Zhu, Y. Qu, and B. Jin, “Constrained transformer network for ECG signal processing and arrhythmia classification,” in BMC Med Inform Decis Mak, Jun 9, 2021.

[10] P. Wagner, N. Strodthoff, R.-D. Bousseljot, D. Kreiseler, F.I. Lunze, W. Samek, and T. Schaeffter, “PTB-XL: A Large Publicly Available ECG Dataset,” in Scientific Data, 2020, DOI: 10.1038/s41597-020-0495-6.

[11] S. Hong et al., “Combining deep neural networks and engineered features for cardiac arrhythmia detection from ECG recordings,” Physiological Measurement, vol. 40, no. 5. IOP Publishing, p. 054009, Jun. 04, 2019. doi: 10.1088/1361-6579/ab15a2.

[12] J. Hu, L. Shen, S. Albanie, G. Sun, and E. Wu, “Squeeze-and-Excitation Networks.” arXiv, 2017. doi: 10.48550/ARXIV.1709.01507.

[13] Z. Zhao, “Transforming ECG Diagnosis:An In-depth Review of Transformer-based DeepLearning Models in Cardiovascular Disease Detection.” arXiv, 2023. doi: 10.48550/ARXIV.2306.01249.

[14] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.” arXiv, 2018. doi: 10.48550/ARXIV.1810.04805.

[15] D. P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization,” arXiv preprint arXiv:1412.6980, 2014.

[16] YAKUT, Önder & Bolat, Emine & EFE, Hatice. (2021). K-Means Clustering Algorithm Based Arrhythmic Heart Beat Detection in ECG Signal. Balkan Journal of Electrical and Computer Engineering. 9. 10.17694/bajece.814473

[17] Zhan, X. (2023). Analysis of the ECG data based on a Gaussian mixture model. Highlights in Science, Engineering and Technology, 53, 177–184.

This page was generated by GitHub Pages.