**IV. Part 2: Neural Network**

(In this section, you can discuss the problem/topic of interest that you chose and how you used deep learning/machine learning methods to solve it.)

**Introduction**

Cardiovascular diseases are a significant health concern due to their potential severity and high prevalence. Among these, arrhythmias—which denote irregular heartbeats—can often be elusive to detect and diagnose. The accurate detection of these anomalies in electrocardiogram (ECG) data is therefore a critical task with substantial implications for patient outcomes.

This report focuses on utilizing the power of machine learning, particularly deep learning, to detect such anomalies in ECG data. We employ Long Short-Term Memory (LSTM) networks, a type of Recurrent Neural Network (RNN), as the cornerstone of our anomaly detection system. We also demonstrate a popular anomaly detection algorithm called Isolation Forrest, to see how it can compare to the LSTM network.

**Motivation and Significance**

Cardiovascular diseases (CVDs) are the number one cause of death globally, taking an estimated 17.9 million lives each year according to the World Health Organization (WHO) [1]. A large number of these deaths can be prevented through early detection and management of cardiac arrhythmias, anomalies in heart rhythms that may be indicative of an underlying heart condition.

The Electrocardiogram (ECG) is a fundamental tool used in the medical field to monitor the electrical heart activity and detect such anomalies. However, the manual interpretation of ECG recordings can be time-consuming, requiring expertise that may not be readily available, especially in regions that have fewer medically trained professionals on hand. Additionally, the sheer volume of data that needs to be processed presents another challenge. Heart diseases like Myocardial Infarction, AV Block, Ventricular Tachycardia and Atrial Fibrillation could be diagnosed from ECG signals, in which nearly 300 million ECG’s being recorded annually (Hedèn et al., 1996)[3]. Also, with 24-hour monitor tests now becoming more common, more and more data is being recorded and it is becoming exceedingly difficult to manually process these recordings.

Automated anomaly detection in ECG data has the potential to revolutionize healthcare outcomes. Early detection of cardiac arrhythmias could allow for prompt and proactive management, potentially reducing the incidence of severe cardiac events and associated mortality. This not only would result in better patient outcomes but could also substantially reduce the healthcare costs associated with the treatment of advanced CVDs.

**Methodology**

# **Data Acquisition and Preprocessing**

Data serves as the foundation of any machine learning project. In this study, we utilized the MIT-BIH Arrhythmia Database—a widely used resource in ECG studies. The database contains 48 half-hour excerpts of two-channel ambulatory ECG recordings, each providing an abundance of ECG signal data. The recordings were digitized at 360 samples per second per channel with 11-bit resolution over a 10-mV range. Two or more cardiologists independently annotated each record, with approximately 110,000 annotations in total (Moody and Mark)[4].

The ECG signals are preprocessed by reshaping them into windows of 3600 samples each. Subsequently, we annotate each of these windows based on the presence of any abnormal beats as indicated by the expert annotations. The processed ECG data and their corresponding labels are then compiled into two large NumPy arrays for further analysis.

**Data Normalization**

Before feeding this data to the model, we performed data normalization. This process involved standardizing the ECG signals to have zero mean and unit variance. The normalization helps make the model more efficient and ensures that the range of the values in the data does not negatively impact the training process.

mean = np.mean(data, axis=(0, 1))

std = np.std(data, axis=(0, 1))

data = (data - mean) / std

**Data split**

Class imbalance, where one class of data vastly outnumbers the other, is a common issue in machine learning tasks. In our case, the majority of the ECG segments are normal, with only a small portion showing anomalies. This imbalance can bias the model towards the majority class, reducing its ability to detect anomalies.

To address this, we employed Synthetic Minority Over-sampling Technique (SMOTE) to oversample the minority class in the training data, resulting in a balanced dataset. It's important to note that SMOTE was only applied to the training data to prevent information leak from the validation/test sets to the training set.

X\_train, X\_test, y\_train, y\_test = train\_test\_split(data, labels, test\_size=0.2, random\_state=1)

X\_train, X\_val, y\_train, y\_val = train\_test\_split(X\_train, y\_train, test\_size=0.25, random\_state=1)

smote = SMOTE()

X\_train\_resampled, y\_train\_resampled = smote.fit\_resample(X\_train.reshape(X\_train.shape[0], -1), y\_train)

X\_train\_resampled = X\_train\_resampled.reshape(-1, X\_train.shape[1], X\_train.shape[2])

**Why LSTMs for ECG Anomaly Detection?**

ECG data is essentially a time series, with each heartbeat following the previous one, influenced by its past states and influencing the ones yet to come. LSTM networks are designed to handle such time series data effectively by considering the temporal sequence of the data. They achieve this by leveraging internal states and 'gates' to control the flow of information, thus capturing long-term dependencies that traditional RNNs might miss.

LSTMs can learn to recognize complex patterns over time, which means it can potentially learn the patterns associated with normal heartbeats and thus become proficient in identifying heartbeats that deviate from the norm.

This is a model that consists of stacked bidirectional long short-term memory (LSTM) layers, dropout layers for regularization, and a time-distributed dense layer.

model = Sequential([

    Bidirectional(LSTM(64, return\_sequences=True), input\_shape=(None, 2)),

    Dropout(0.2),

    Bidirectional(LSTM(64, return\_sequences=True)),

    Dropout(0.2),

    TimeDistributed(Dense(1))

])

model.compile(optimizer='adam', loss='mse')

Model: "sequential\_3"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

bidirectional\_6 (Bidirectional) (None, None, 128) 34304

dropout\_6 (Dropout) (None, None, 128) 0

bidirectional\_7 (Bidirectional) (None, None, 128) 98816

dropout\_7 (Dropout) (None, None, 128) 0

time\_distributed\_3 (TimeDistributed) (None, None, 1) 129

=================================================================

Total params: 133,249

Trainable params: 133,249

Non-trainable params: 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

* Bidirectional wrapper is used to apply the LSTM layer forwards and then backwards, helping the LSTM to learn long-range dependencies in the sequence data.
  + The LSTM layer itself has 64 units
  + input\_shape=(None, 2) argument indicates that the model can take sequences of arbitrary length (None) with two features per time-step (2)
* Dropout 20% to prevent overfitting
* Another bidirectional LSTM layer, again with 64 units and returning sequences
* Another dropout layer, again dropping out 20% of its inputs
* The output layer of the model. The TimeDistributed wrapper is used to apply the Dense layer to every time-step of the input sequence independently. The Dense layer itself has a single unit and, by default, no activation function (which means it's a linear activation).
* The model is compiled with the Adam optimizer and mean squared error loss

**Training method**

The model was then trained on the training data for 10 epochs, with a batch size of 32, and validated using the validation data. The model's performance was evaluated on the test set, providing an unbiased assessment of the model's ability to detect anomalies in ECG data.

This methodology was chosen due to LSTM's effectiveness in handling time-series data. LSTMs are capable of learning and remembering over long sequences, which makes them a good decision for sequence prediction, such as this ECG anomaly detection problem.

callbacks = [

    EarlyStopping(patience=5),

    ModelCheckpoint('model.h5', save\_best\_only=True)

]

history = model.fit(X\_train\_resampled, y\_train\_resampled, validation\_data=(X\_val, y\_val), epochs=50, batch\_size=32, callbacks=callbacks)

This methodology, combining effective data preprocessing, addressing class imbalance, using bidirectional LSTM layers, and optimizing the training process should provide a decent model for anomaly detection.

**Other Techniques Utilized**

**Isolation Forrest**

The second methodology for the ECG anomaly detection project employed an unsupervised learning approach using the Isolation Forest algorithm. This algorithm works by isolating observations by randomly selecting a feature and then randomly selecting a split value between the maximum and minimum values of the selected feature.

# Preprocessing - Feature Extraction

# Calculate statistical features (mean and standard deviation) for each window

features = np.array([np.mean(window, axis=0) for window in data])

features = np.hstack((features, np.array([np.std(window, axis=0) for window in data])))

# Standardize the features to have 0 mean and unit variance

scaler = StandardScaler()

features = scaler.fit\_transform(features)

# Create an Isolation Forest model

clf = IsolationForest(contamination=0.01)

# Fit the model on the features

clf.fit(features)

# Predict the anomalies in the data

pred = clf.predict(features)

# Check the anomaly score of each window

score = clf.decision\_function(features)

anomalies = np.argwhere(pred == -1)

This unsupervised learning methodology, using the Isolation Forest algorithm, provides a powerful alternative for ECG anomaly detection. It does not require a balanced dataset like supervised learning methods and can be more robust to changes in the type and structure of the anomalies in the ECG signal.

**Results from the LSTM Model:**

**Model Evaluation**

y\_pred = model.predict(X\_test)

y\_pred = y\_pred.max(axis=1).flatten()

# The model's output is continuous, but we need binary predictions for the metrics.

# We can choose a threshold (e.g., 0.5) and classify all instances with an output above this threshold as anomalies.

y\_pred\_bin = (y\_pred > 0.5).astype(int)

**Confusion Matrix**

# Generate confusion matrix

cm = confusion\_matrix(y\_test, y\_pred\_bin)

# Plot confusion matrix

plt.figure(figsize=(10,7))

sns.heatmap(cm, annot=True, fmt='d')

plt.xlabel('Predicted')

plt.ylabel('Truth')

**A picture containing screenshot, text, design

Description automatically generatedTrue Negatives (0, 0):** These are the cases where the model correctly predicted the negative class (i.e., the absence of an anomaly). Here, this number is 0, which suggests that there were no actual negative cases, or the model failed to correctly predict any negative cases.

**False Positives (0, 1):** These are the cases where the model incorrectly predicted the positive class (i.e., the presence of an anomaly). Here, this number is 751, which is quite high. This means that the model has incorrectly flagged a lot of instances as anomalies.

**False Negatives (1, 0):** These are the cases where the model incorrectly predicted the negative class. Here, this number is 0, which suggests that the model didn't miss any actual anomalies.

**True Positives (1, 1):** These are the cases where the model correctly predicted the positive class. Here, this number is 987, which suggests that the model has done well in correctly identifying the anomalies.

**Accuracy:** This is not applicable here because it assumes an equal cost of false positives and false negatives, which is rarely the case in real-world scenarios. The dataset seems to have too many anomalies per regular heartbeats to train on, which can make accuracy misleading.

**Precision (Positive Predictive Value):** This is the ratio of true positives to the sum of true and false positives. Here, it would be 987 / (987 + 751) = 0.57. This is a relatively low precision, which suggests that when the model predicts an anomaly, it's correct only about 57% of the time.

**F1 Score:** This is the harmonic mean of precision and recall, and it tries to find the balance between them. Here, the F1 Score would be 2 \* (0.57 \* 1) / (0.57 + 1) = 0.72. This is a moderate F1 score.

**ROC Curve:**

# Calculate the ROC curve points

fpr, tpr, thresholds = roc\_curve(y\_test, y\_pred)

# Calculate the AUC (area under the ROC curve)

roc\_auc = auc(fpr, tpr)

plt.figure(figsize=(10,7))

plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc\_auc)

plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('Receiver Operating Characteristic')

plt.legend(loc="lower right")

**A picture containing line, text, plot, screenshot

Description automatically generatedROC-AUC:** This is the area under the Receiver Operating Characteristic (ROC) curve. The ROC curve is created by plotting the true positive rate (recall) against the false positive rate (ratio of false positives to the sum of false positives and true negatives) at various threshold settings. The AUC is the area under this curve, and it ranges from 0 to 1. An AUC of 0.5 suggests no discrimination (i.e., ability to classify correctly), 0.7 to 0.8 is considered acceptable, 0.8 to 0.9 is considered excellent, and more than 0.9 is considered outstanding. Here, an AUC of 0.79 suggests acceptable discrimination.

In summary the model seems to perform well at identifying actual anomalies (high recall) but not so good at confirming its predictions (moderate precision). It results in a high number of false positives, which ultimately renders this model useless in a healthcare application. False positives completely defeat the purpose of automatic heart irregularity detection. Professionals need to be able to be confident in the models’ warnings. If there are too many alerts to anomalies, then the alerts will not be acknowledged with any sincerity.

**Results from the Isolated Forrest Model:**

**Model Evaluation**

# Predict the anomalies in the data

pred = clf.predict(features)

# Check the anomaly score of each window

score = clf.decision\_function(features)

anomalies = np.argwhere(pred == -1)

**Confusion Matrix**

**A picture containing text, screenshot, colorfulness, rectangle

Description automatically generated**

* True negatives (TN): 67 (The model predicted normal and it's indeed normal)
* False negatives (FN): 20 (The model predicted normal, but it was actually an anomaly)
* False positives (FP): 5028 (The model predicted anomaly, but it was actually normal)
* True positives (TP): 3573 (The model predicted anomaly and it's indeed an anomaly)

Based on these numbers, the model seems to be making a significant number of false positive errors. This means the model is tending to classify normal data as anomalies quite often.

The model's sensitivity (or True Positive Rate), which measures the proportion of actual positives that are correctly identified, can be calculated as TP / (TP + FN) = 3573 / (3573 + 20) = 0.994, or approximately 99.4%.

The model's specificity (or True Negative Rate), which measures the proportion of actual negatives that are correctly identified, can be calculated as TN / (TN + FP) = 67 / (67 + 5028) = 0.013, or approximately 1.3%.

**ROC Curve:**

**A picture containing text, screenshot, line, plot

Description automatically generated**

The AUC (Area Under the Curve) is another measure of the model's performance. It's the area underneath the ROC curve, and provides an aggregate measure of the model's performance across all possible classification thresholds. An AUC of 1 indicates a perfect classifier, while an AUC of 0.5 implies that the classifier is no better than random chance. AUC values between these two extremes indicate varying degrees of classifier performance.

**5. Discussion**

* Analysis of the strengths of your methodology (e.g., what it does well, how it improves on existing methods).
* Honest discussion of the limitations and potential areas for improvement.
* Thoughts on potential future work in this area, building on what you've done.
  + Could possibly train the model first to learn normal heartbeats on ECG data of regular heartbeats. Then test the model to detect the anomalies in the MIT-BIH Arrhythmia Database.
  + Bigger dataset, larger, better tuned model
  + Perhaps a CNN

**6. Conclusion**

* Recap of the problem, your method, and the results.
* Final thoughts on the importance of this work and its implications for the future.

**Works Cited:**

[1] Rajpurkar, Pranav, et al. “Cardiologist-Level Arrhythmia Detection with Convolutional Neural Networks.” *arXiv.Org*, 6 July 2017, arxiv.org/abs/1707.01836.

[2] “Cardiovascular Diseases (Cvds).” *World Health Organization*, 11 June 2021, www.who.int/news-room/fact-sheets/detail/cardiovascular-diseases-(cvds).

[3] Hedèn, Bo, Ohlsson, Mattias, Holst, Holger, Mj¨oman, Mattias, Rittner, Ralf, Pahlm, Olle, Peterson, Carsten, and Edenbrandt, Lars. Detection of frequently overlooked electrocardiographic lead reversals using artificial neural networks. The American journal of cardiology, 78(5):600–604, 1996.

[4] Moody, George, and Roger Mark. “MIT-BiH Arrhythmia Database.” *MIT-BIH Arrhythmia Database v1.0.0*, 24 Feb. 2005, www.physionet.org/content/mitdb/1.0.0/.