Understanding Electrocardiograms (ECGs) and ECG Forecasting Tasks

February 13, 2025

1 Data-Centric Workflow for Model Development

Below are the basic steps typically followed in deep learning models like CNN, LSTM, and CNN-LSTM for classification or prediction tasks:

1. **Data Collection:** Collect and gather the relevant data for the task.
2. **Data Preprocessing:** Preprocess the data to make it suitable for input into the model.
   - Remove noise and irrelevant data.
   - Handle missing values (e.g., imputation or deletion).
   - Rescale or normalize features.
   - Data augmentation (especially in CNNs for image data).
3. **Data Splitting:** Split the data into training, validation, and test sets.
4. **Model Initialization:** Initialize the model architecture.
   - For CNN: Initialize convolutional layers, pooling layers, fully connected layers, etc.
   - For LSTM: Initialize LSTM layers, input-output layers, and regularization (e.g., dropout).
   - For CNN-LSTM: Combine CNN for feature extraction followed by LSTM layers for sequence modeling.
5. **Model Compilation:** Choose the appropriate loss function, optimizer, and evaluation metrics.
   - Loss function: E.g., categorical cross-entropy for classification.
   - Optimizer: E.g., Adam, SGD, RMSProp.
   - Metrics: Accuracy, precision, recall, etc.
6. **Model Training:** Train the model using the training dataset.
   - Feed data through the model.
   - Perform forward propagation and compute the loss.
   - Perform backpropagation to adjust weights using gradient descent or variants.
   - Monitor the model's performance using validation data.
7. **Model Evaluation:** Evaluate the trained model on the test set.
   - Calculate performance metrics (accuracy, F1-score, etc.).
   - Adjust hyperparameters or architecture if needed.
8. **Prediction/Classification:** Once the model is trained, make predictions on new, unseen data.

- Pass the data through the trained model.
- Get the output and interpret the results (e.g., class labels or continuous values).

9. **Model Deployment:** Deploy the model into production for real-time predictions or batch processing.

2 Convolutional Neural Networks (CNN)

Core Idea

Convolutional Neural Networks (CNNs) are designed to automatically and adaptively learn spatial hierarchies of features from input data. They are particularly powerful for tasks involving grid-like data such as images, time-series data like ECG signals, and more. CNNs operate through a series of layers that capture increasingly complex features of the input data.

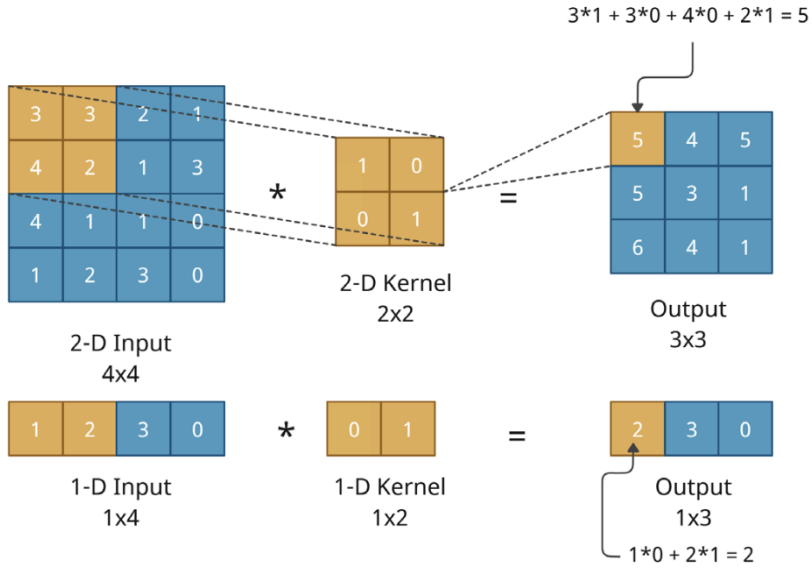

Figure 1: A convolution operation on 2D and 1D inputs [7].

2.1 Inputs to a CNN

A CNN takes in the following inputs:

1. **Current Input (x_t):** This is the raw data, such as an ECG signal or an image. The input data can have multiple dimensions (e.g., 2D for images or 1D for ECG signals).

2. **Filters/Kernels (w):** These are small learnable weights that slide over the input data to extract features such as edges or patterns. The filters are shared across the entire input, which reduces the number of parameters.

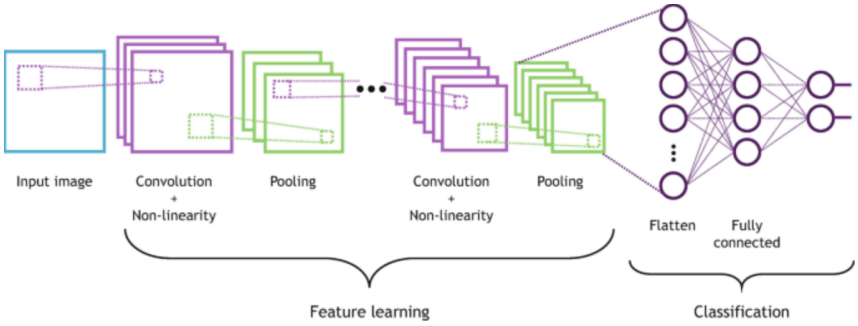

Figure 2: Inputs to CNN.

2.2 What Happens to the Input

Once the input data is fed into the CNN, the following steps occur at each layer:

1. **Convolution Operation:** The input data is convolved with the filters (kernels) to produce feature maps. This operation detects basic features in the data, like edges, corners, or other patterns. The formula for the convolution operation is:

$$y_i = f\left(\sum_{j=0}^{k-1} w_j x_{i+j} + b\right)$$

   where:

   - x represents the input sequence (e.g., ECG signal),
   - w denotes the kernel weights (filters),
   - b is the bias term,
   - f is the activation function (such as ReLU),
   - k is the kernel size.

2. **Activation Function:** After convolution, the output is passed through an activation function like ReLU to introduce non-linearity. This enables the network to learn more complex patterns.

3. **Pooling Layer:** Pooling operations like max pooling or average pooling are applied to reduce the spatial dimensions of the feature maps, thereby reducing computational complexity while retaining important features.

4. **Fully Connected Layer:** The pooled feature maps are flattened and passed through fully connected layers, where the final output is generated (classification or regression).

2.3 Components of a CNN

A CNN typically consists of the following layers:

- **Convolutional Layers:** Extract spatial features from the input (e.g., ECG signals) using filters (kernels). These layers detect patterns like edges, textures, and other important features.
- **Activation Functions:** Non-linear functions such as ReLU (Rectified Linear Unit) are used to introduce non-linearity into the model, allowing it to learn complex patterns.
- **Pooling Layers:** Reduce the spatial dimensions (downsampling), decreasing the computational cost and the risk of overfitting while maintaining important information.
- **Fully Connected Layers:** These layers make final predictions based on the learned features, typically leading to classification or regression outcomes.

2.4 Advantages of CNNs

CNNs offer several advantages, particularly for tasks involving spatial or temporal data:

- **Efficient Feature Extraction:** CNNs automatically learn spatial patterns from raw input data without manual feature engineering.
- **Reduced Number of Parameters:** By using shared weights in convolutional filters, CNNs significantly reduce the number of parameters compared to fully connected networks, leading to lower computational complexity.
- **Robustness:** CNNs are highly robust to noise and distortions, making them suitable for real-world data with imperfections.
- **Scalability:** CNNs can scale well to large datasets, enabling effective learning from massive amounts of data.

2.5 Limitations of CNNs

Despite their strengths, CNNs come with certain limitations:

- **Large Datasets Required:** CNNs typically require large datasets for training to avoid overfitting and achieve good generalization.
- **Computational Intensity:** CNNs are computationally expensive, requiring significant resources, including GPUs, for efficient training and inference.
- **Complex Hyperparameter Tuning:** The process of tuning CNN hyperparameters (such as kernel size, number of layers, and learning rate) can be time-consuming and requires extensive experimentation.
- **Interpretability:** CNNs lack the interpretability of traditional machine learning models, making it harder to understand why a model makes certain predictions.

2.6 Applications of CNNs

CNNs are widely used in a variety of domains, particularly in the analysis of sequential or image data:

- **Arrhythmia Detection:** CNNs can be used to detect and classify arrhythmias in ECG signals by learning features that correspond to abnormal heart patterns.
- **Heart Condition Prediction:** CNNs can predict future heart conditions based on historical ECG data, enabling early intervention.
- **Wearable Health Monitoring:** CNNs are applied in real-time ECG monitoring systems integrated into wearable devices, providing continuous health tracking.
- **Noise and Artifact Removal:** CNNs are effective in removing noise and artifacts from ECG signals, improving the quality of the input data for further analysis.

3 Long Short-Term Memory (LSTM) Networks

Core Idea

Long Short-Term Memory (LSTM) networks are an advanced version of Recurrent Neural Networks (RNNs) introduced by Hochreiter and Schmidhuber in 1997. LSTMs solve key issues of vanishing and exploding gradients in vanilla RNNs. Inspired by logic gates in computers, LSTMs incorporate a special component called a **memory cell**, which acts as long-term memory, in addition to the hidden state used in classical RNNs.

3.1 Inputs to an LSTM

An LSTM network processes the following inputs:

1. **Current Input (X_t):** This is the raw data (e.g., ECG signal, time-series data, or text) at the current time step.
2. **Previous Hidden State (H_{t-1}):** This is the output of the previous LSTM cell at the previous time step, which carries information from past time steps.
3. **Previous Cell State (C_{t-1}):** This is the long-term memory from the previous time step, which stores important past information.

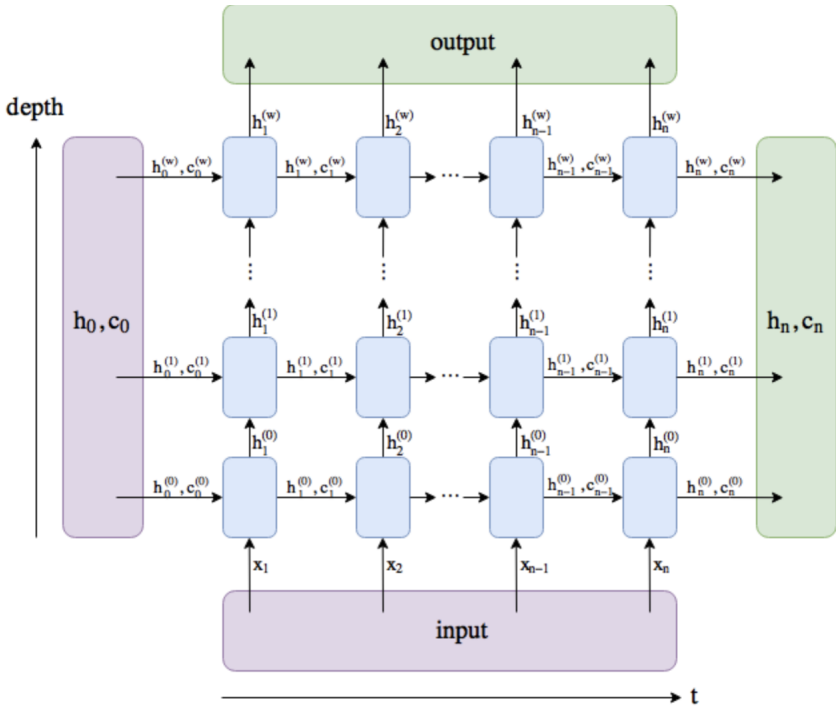

Figure 3: Inputs to LSTM.

3.2 What Happens to the Input

At each time step, the LSTM processes the input data through its various gates and updates the hidden and cell states:

1. **Input Gate:** This gate decides how much of the new information from the current input and previous hidden state should be stored in the memory cell. The input gate's formula is:

$$I_t = \sigma(W_{xi} \cdot X_t + W_{hi} \cdot H_{t-1} + b_i)$$

2. **Forget Gate:** This gate determines how much of the previous memory (C_{t-1}) should be forgotten. The forget gate's formula is:

$$F_t = \sigma(W_{xf} \cdot X_t + W_{hf} \cdot H_{t-1} + b_f)$$

3. **Output Gate:** This gate decides what part of the memory should be output at the current time step. The output gate's formula is:

$$O_t = \sigma(W_{xo} \cdot X_t + W_{ho} \cdot H_{t-1} + b_o)$$

4. **Cell State Update:** This step updates the memory cell. The candidate cell state ($\tilde{C}_t$) is first calculated, and then it is combined with the forget gate to form the updated cell state (C_t):

$$\tilde{C}_t = \tanh(W_{xc} \cdot X_t + W_{hc} \cdot H_{t-1} + b_c)$$

$$C_t = F_t \odot C_{t-1} + I_t \odot \tilde{C}_t$$

5. **Hidden State Update:** The updated memory cell is then passed through the output gate, and the new hidden state (H_t) is calculated:

$$H_t = O_t \odot \tanh(C_t)$$

In the above equations: - σ is the sigmoid activation function. - $\odot$ denotes element-wise multiplication. - $W_{xi}, W_{hi}, W_{xf}, W_{hf}, W_{xo}, W_{ho}, W_{xc}, W_{hc}$ represent weight matrices for the respective gates and the cell state update. - b_i, b_f, b_o, b_c are bias terms for the respective gates.

3.3 Key Features of LSTMs

LSTMs are specifically designed to handle long-term dependencies in sequential data. The main features of LSTMs include:

- **Memory Cell:** Acts like long-term memory, storing crucial information over long periods.
- **Gates:** LSTMs utilize three main gates (Input, Forget, and Output gates) to control how information flows through the network and how the memory is updated.
  - **Input Gate:** Decides how much new information from the input should be stored.
  - **Forget Gate:** Decides how much information from the previous time step should be discarded.
  - **Output Gate:** Determines what information from the memory cell should be output at each time step.

3.4 Why LSTMs are Important

LSTMs are crucial for tasks that involve sequential data, such as time-series data, text, and speech, due to their ability to capture long-term dependencies. Their importance lies in the following areas:

- **Language Modeling and Translation:** LSTMs can predict the next word in a sentence or translate text from one language to another by remembering long-range context.
- **Speech Recognition:** LSTMs excel at handling speech signals where context over time is essential for accurate recognition.
- **Stock Price Prediction:** LSTMs can model time-series data like stock prices, making them valuable for financial forecasting.
- **General Sequence Prediction:** LSTMs are broadly applicable to any domain that involves sequential data, such as ECG signals, video frames, and more.

3.5 Architecture of LSTMs

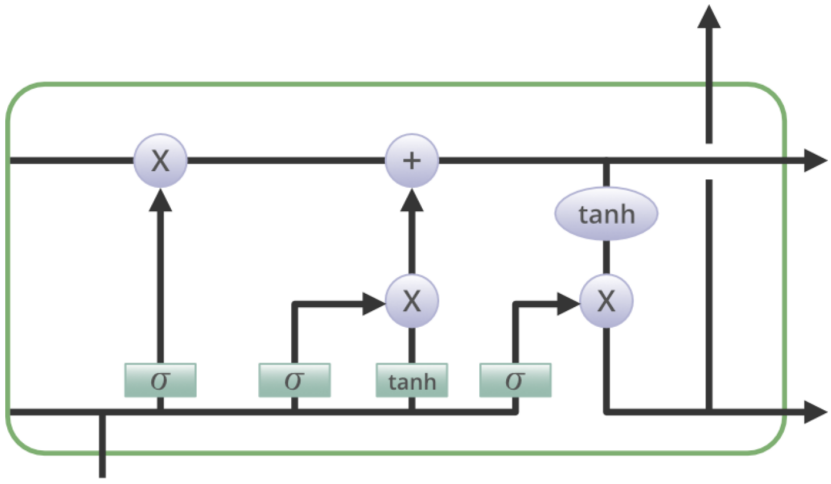

Figure 4: Diagram of a Long Short-Term Memory (LSTM) cell architecture [6].

3.6 Advantages of LSTMs

LSTMs have several advantages that make them suitable for a wide range of applications:

- **Long-Term Dependencies:** LSTMs are designed to capture long-term dependencies in time-series and sequential data.
- **Vanishing Gradient Problem:** LSTMs address the vanishing gradient problem, which occurs in traditional RNNs, allowing them to learn from long sequences.
- **Versatile Applications:** LSTMs are effective for a variety of tasks, including speech, language, and time-series data analysis.
- **Variable-Length Sequences:** LSTMs can handle variable-length sequences, which makes them adaptable to different types of input data.

3.7 Limitations of LSTMs

Despite their advantages, LSTMs have certain limitations:

- **Computational Complexity:** LSTMs are computationally expensive due to their complex gate mechanisms.
- **Training Time:** Training LSTM models requires significant time and resources compared to simpler models.
- **Memory Requirements:** LSTMs require substantial memory for storing parameters, which can be a limitation for large-scale applications.
- **Struggles with Very Long Sequences:** While LSTMs address some issues of traditional RNNs, they may still face challenges when dealing with extremely long sequences.

3.8 Applications of LSTMs in ECG Analysis

LSTMs have demonstrated their usefulness in the analysis of ECG signals for various medical and healthcare applications:

- **ECG Signal Forecasting:** LSTMs can be used to predict future ECG signals, providing insights into the patient's condition over time.

- **Arrhythmia Classification:** LSTMs are effective in detecting and classifying arrhythmias from ECG signals, helping in the diagnosis of heart conditions.
- **Real-Time Monitoring:** LSTMs are applied in wearable devices for real-time monitoring of ECG signals, offering predictive diagnostics.
- **Feature Extraction:** LSTMs are capable of extracting important features from ECG signals, improving the accuracy of cardiovascular analysis.

4 Hybrid Model CNN+LSTM

The CNN-LSTM hybrid model combines the strengths of Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) networks to improve ECG signal forecasting. CNNs are effective in extracting spatial features, while LSTMs are powerful in capturing temporal dependencies. By integrating both architectures, the CNN-LSTM model enhances the performance of ECG signal prediction.

4.1 Inputs to the CNN-LSTM Hybrid Model

The input to the hybrid model consists of multi-lead ECG signals, where each lead captures the heart's electrical activity over time. These raw signals represent time-series data and are passed through the model for further processing.

ECG Input Data: The multi-lead ECG signals are provided as input to the model. Each lead records the electrical activity of the heart from a different angle, resulting in multiple time-series data streams.

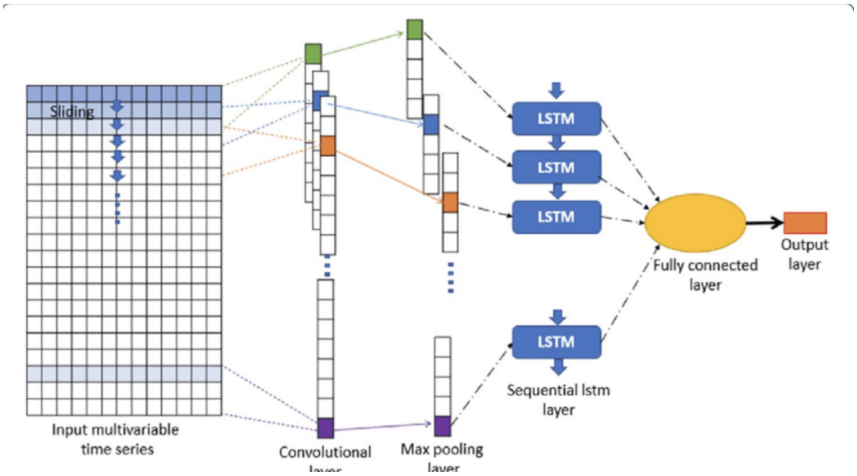

Figure 5: Inputs to CNN-LSTM Hybrid Model.

4.2 What Happens to the Input in the CNN Layers

The input ECG signal undergoes several transformations in the CNN part of the hybrid model, which processes spatial features. The CNN layers perform the following operations:

Convolutional Layers: The multi-lead ECG signals are passed through convolutional layers, which apply filters (kernels) to extract spatial features. These filters detect patterns such as peaks, QRS complexes, or ST segments from the raw ECG signals.

The 1D convolution operation is represented mathematically as:

$$y_i = f\left(\sum_{j=0}^{k-1} w_j x_{i+j} + b\right)$$

where:

- x is the ECG input signal,

- w are the kernel weights,
- b is the bias term,
- f is an activation function applied to introduce non-linearity (e.g., ReLU).

Pooling Layers: After the convolutional layers, pooling layers (typically max pooling) are applied. Pooling reduces the spatial dimensions of the output feature maps, which helps in reducing the computational cost and the number of parameters. It also makes the model more robust by retaining important spatial features while discarding less significant ones.

4.3 What Happens After the CNN Layers

Once the input has passed through the convolutional and pooling layers, the output is a set of features that represent the spatial patterns in the ECG signal. These features are then prepared for the next step in the model:

Flattening: The output of the CNN layers, which is typically in the form of a multi-dimensional tensor, is flattened into a one-dimensional vector so that it can be fed into the LSTM layers.

Fully Connected Layers: The flattened feature vector is passed to fully connected layers for final decision-making. These layers combine the extracted features to generate predictions or classifications, depending on the task.

4.4 Key Components of the CNN-LSTM Hybrid Model

The CNN-LSTM hybrid model consists of the following key components:

- **Convolutional Layers**: Extract spatial features from multi-lead ECG signals by applying convolution operations.
- **Pooling Layers**: Reduce the dimensionality of the extracted features, thereby making the model more efficient and less prone to overfitting.
- **Fully Connected Layers**: These layers use the processed features to make final predictions about the ECG signal.

4.5 Architecture of CNN-LSTM Model

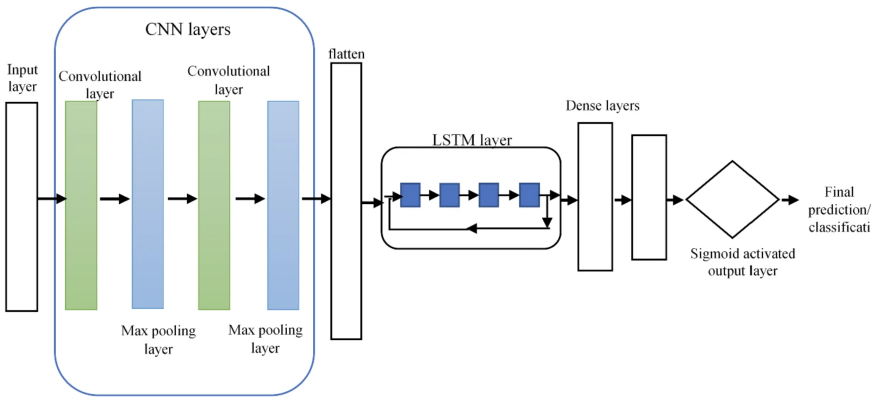

Figure 6: Structure of Proposed Hybrid CNN-LSTM model [8].

4.6 Advantages of CNN-LSTM Hybrid Model

The combination of CNNs and LSTMs offers several advantages:

- **Combines Spatial and Temporal Feature Extraction**: The CNN part extracts important spatial features, while the LSTM part (not detailed here) handles temporal dependencies in the data.

- **Reduces Computational Cost**: The use of CNNs to extract features reduces the complexity and computational cost compared to using only LSTMs.
- **Effective for Multi-Lead ECG Signals**: The model handles complex, multi-lead ECG data well, capturing rich spatial information across different leads.
- **Suitable for Real-Time Analysis**: The hybrid architecture enables real-time ECG analysis and forecasting by extracting meaningful features efficiently.

4.7 Disadvantages of CNN-LSTM Hybrid Model

While the CNN-LSTM model offers great potential, there are challenges to consider:

- **Increased Model Complexity**: The dual architecture (CNN + LSTM) adds complexity, which can make the model harder to train and tune.
- **Hyperparameter Tuning**: The interaction between CNN and LSTM components requires careful tuning of hyperparameters for optimal performance.
- **Higher Training Time**: The hybrid model requires more training time than individual models due to its increased complexity.
- **Large ECG Datasets Required**: The model needs a substantial amount of labeled ECG data to perform optimally and avoid overfitting.

4.8 Applications of CNN-LSTM in ECG Analysis

The CNN-LSTM hybrid model is highly effective for various ECG analysis tasks:

- **Advanced ECG Signal Forecasting and Anomaly Detection**: The model can predict future ECG signals and identify abnormal patterns indicative of heart disease or other conditions.
- **Detection of Arrhythmias and Cardiovascular Abnormalities**: The model classifies different arrhythmias and abnormalities in the ECG, helping in diagnosis.
- **Wearable Device Monitoring**: The model can be integrated into wearable health devices for continuous, real-time monitoring of the heart's electrical activity.
- **Automated Feature Extraction for ECG Classification**: The model automatically extracts important features, improving the accuracy of ECG classification tasks for various cardiac conditions.

4.9 Conclusion

The CNN-LSTM hybrid model effectively captures both spatial and temporal dependencies in ECG data, making it a powerful tool for ECG signal forecasting. By combining the strengths of CNNs for feature extraction and LSTMs for sequence modeling, the model enhances prediction accuracy in healthcare applications. Despite its computational intensity, the CNN-LSTM hybrid model offers promising potential for real-time ECG analysis and healthcare monitoring solutions.

ECG Signal Classification Using Convolutional Neural Networks (CNN + LSTM)

Harnessing deep learning for ECG-based classification and forecasting using 12-lead signals. For classification, a robust 1D CNN model was constructed, leveraging multiple convolutional layers with batch normalization, dropout regularization, and L2 weight penalties. Training employed Adam optimization with learning rate scheduling, early stopping, and model checkpointing. The CNN model demonstrated promising accuracy, precision, recall, and AUC scores, validated through rigorous evaluation metrics.

For time-series forecasting, ECG signals were reshaped into input windows and target sequences. A hybrid CNN-LSTM model was developed, integrating convolutional layers for feature extraction and an LSTM layer to capture temporal dependencies. The forecasting model successfully predicted future ECG trends, with performance tracked through MSE loss curves.

Comparative analysis revealed the CNN model's superior performance for classification, while the CNN-LSTM model effectively generated plausible ECG forecasts across all 12 leads. The results indicate deep learning's potential in both diagnostic classification and predictive cardiology applications.

4.10 Model 1: Data Preprocessing Steps

1. **Load Metadata:** Read the CSV file containing ECG metadata and extract the relevant MI-related cases using scp_codes.
2. **Stratified Sampling:** Use StratifiedShuffleSplit to select a balanced 10% subset of the dataset while maintaining class proportions.
3. **Extract ECG Signal Data:** Read ECG signals using wfdb. Extract signals up to max_len = 1000 samples. Ensure each signal is 1D or reshaped correctly.
4. **Standardization (Feature Scaling):** Apply StandardScaler (Z-score normalization) to each ECG signal.
5. **Padding and Reshaping for CNN & LSTM:** Use sequence.pad_sequences() to ensure all signals have the same length (1000 time steps). Reshape input to (samples, time steps, features) for LSTM compatibility.
6. **Handle Class Imbalance using SMOTE:** Apply Synthetic Minority Oversampling Technique (SMOTE) to increase underrepresented class samples. Reshape data back into (samples, 1000, num_features).
7. **One-Hot Encode Labels:** Convert class labels into categorical format using to_categorical().
8. **Data Augmentation:** Generate **noisy and scaled versions** of ECG signals to improve generalization.
9. **Train-Test Split:** Split data into **80% training and 20% testing** using train_test_split(), ensuring stratification.

Hyperparameter	**Value**
Learning Rate	0.0005
Dropout Rate	0.5
L2 Regularization	0.01
Batch Size	16
Epochs	30
Filters (CNN Layers)	[32, 64]
Kernel Size	3
Max Sequence Length (max_len)	1000
Sampling Frequency (fs)	100 Hz
LSTM Units	64
LSTM Return Sequences	False
Train-Test Split	80% train, 20% test
SMOTE Used	Yes
Padding Strategy	'post'
Scaling Method	StandardScaler()
Optimizer	Adam
Loss Function	Categorical Crossentropy
Callbacks Used	ReduceLROnPlateau, EarlyStopping, ModelCheckpoint

Table 1: Hyperparameter Settings for Model 1

4.11 Model 1: Performance Analysis

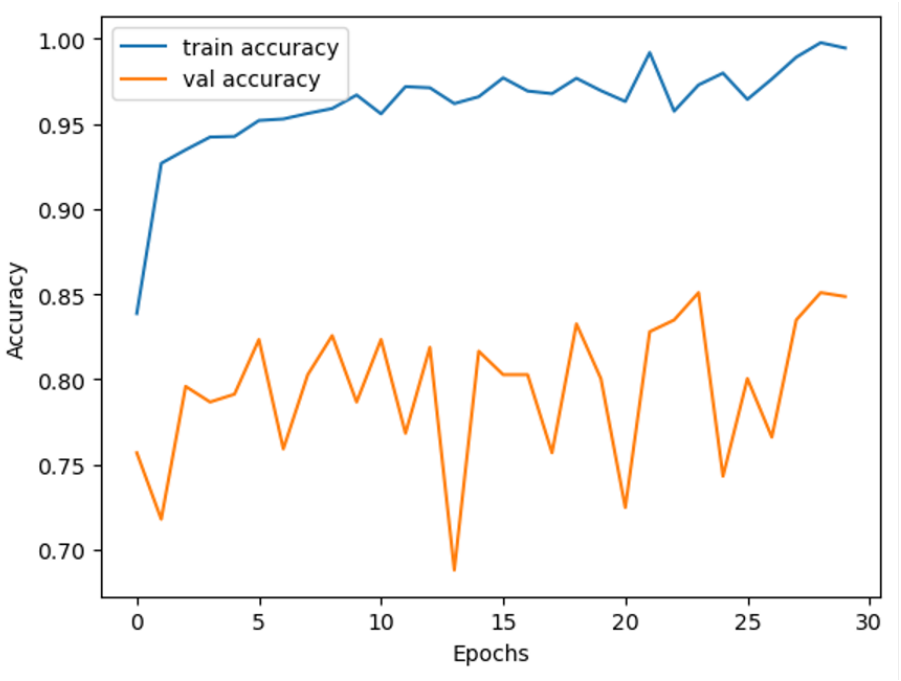

Figure 7: Training and validation accuracy graph Model 1

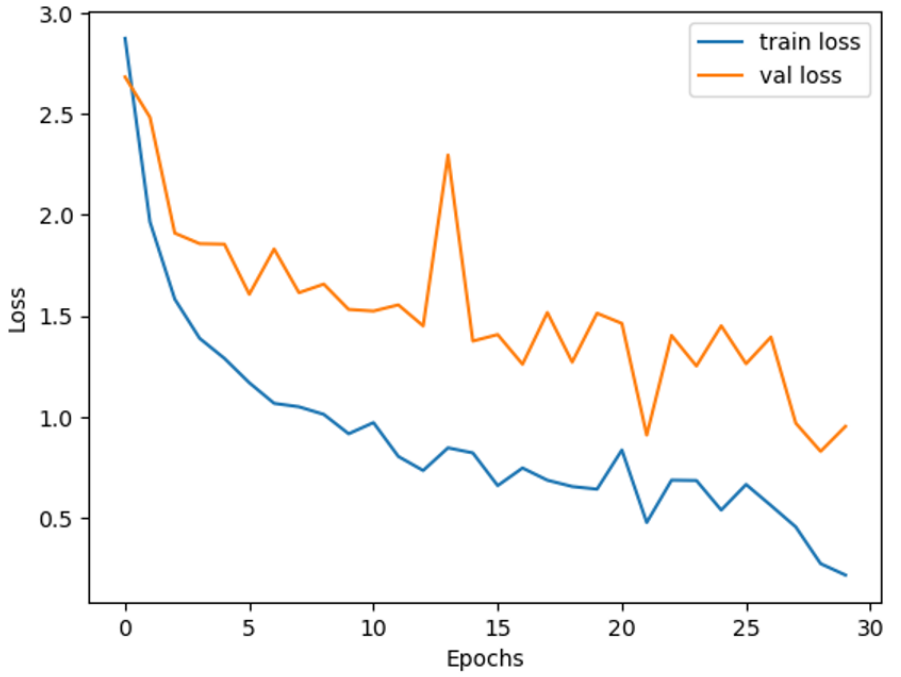

Figure 8: Training and validation loss graph 1

The graphs show that the model is overfitting. While the training accuracy increases and the training loss decreases steadily, the validation accuracy fluctuates and the validation loss is inconsistent. This indicates that the model is performing well on the training data but struggling to generalize to new, unseen data.

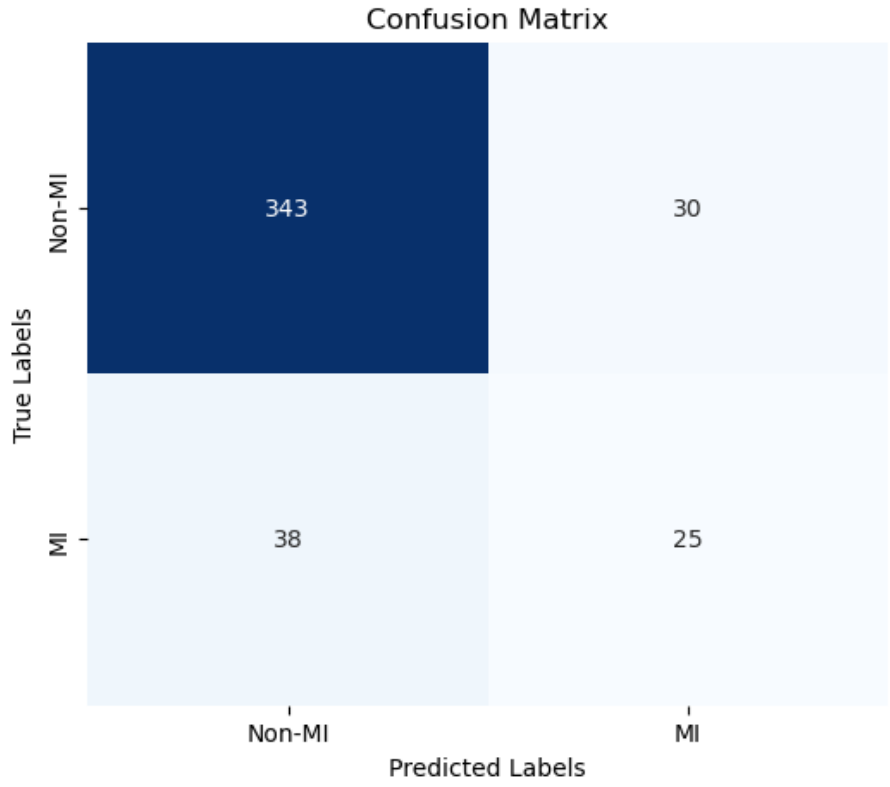

Figure 9: Confusion Matrix for Model 1

This confusion matrix shows a model classifying Non-MI and MI cases. It's good at identifying Non-MI (348 correct), but struggles with MI: low precision (48%) and recall (37%). While accuracy is 85%, the low MI recall (missing many actual MI cases) is a serious problem, making the model unreliable for MI detection in its current state.

4.12 Model 2: Improvements and Evaluation (continued)

The addition of metrics like Precision, Recall, F1-Score, and Specificity provides a more thorough evaluation, particularly when dealing with imbalanced datasets. These metrics offer a fuller picture of the model's performance, especially in terms of its ability to handle the imbalance between classes.

Hyperparameter	**Model 1**	**Model 2**
Learning Rate	0.0005	0.0005
Dropout Rate	0.5	0.5
L2 Regularization	0.01	0.01
Batch Size	16	32
Epochs	30	50
Filters	[32, 64]	[32, 64, 128]
Kernel Size	3	3
Max Length	1000	1000
Sampling Rate (fs)	100	100
Stratified Sampling Test Size	90%	80%
Data Augmentation	Noise & Scaling	Noise, Scaling & Shifting
LSTM Units	LSTM Units	LSTM Units
LSTM Return Sequences	False	False
SMOTE Used?	Yes	Yes
Early Stopping Patience	10	7
Dropout Layers	Only on Dense Layer	Applied in Conv Layers too
Dense Layer Units	64	128

Table 2: Hyper-parameters comparison for Model 1 and Model 2

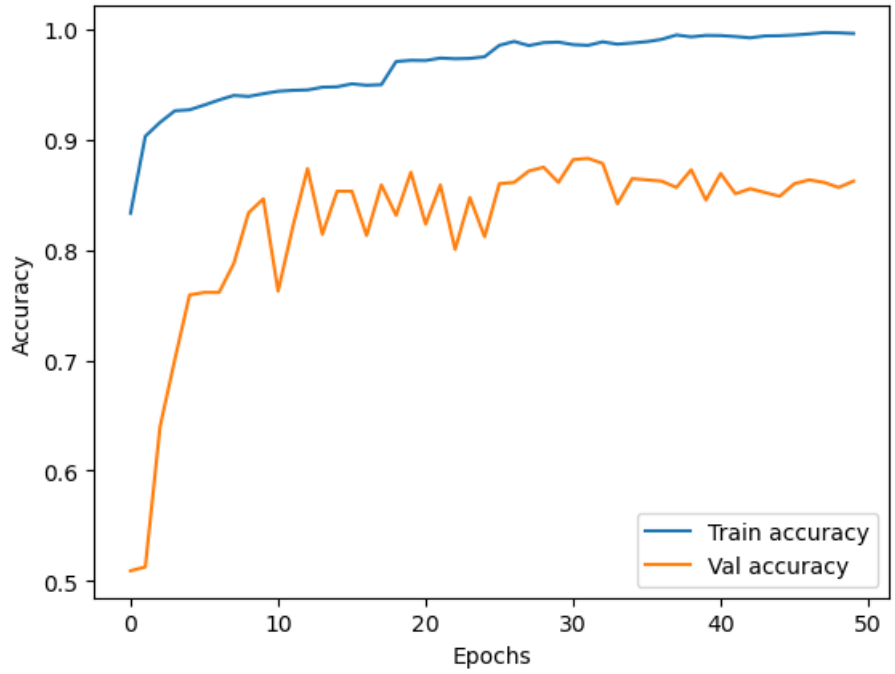

Figure 10: Training and validation accuracy graph for Model 2

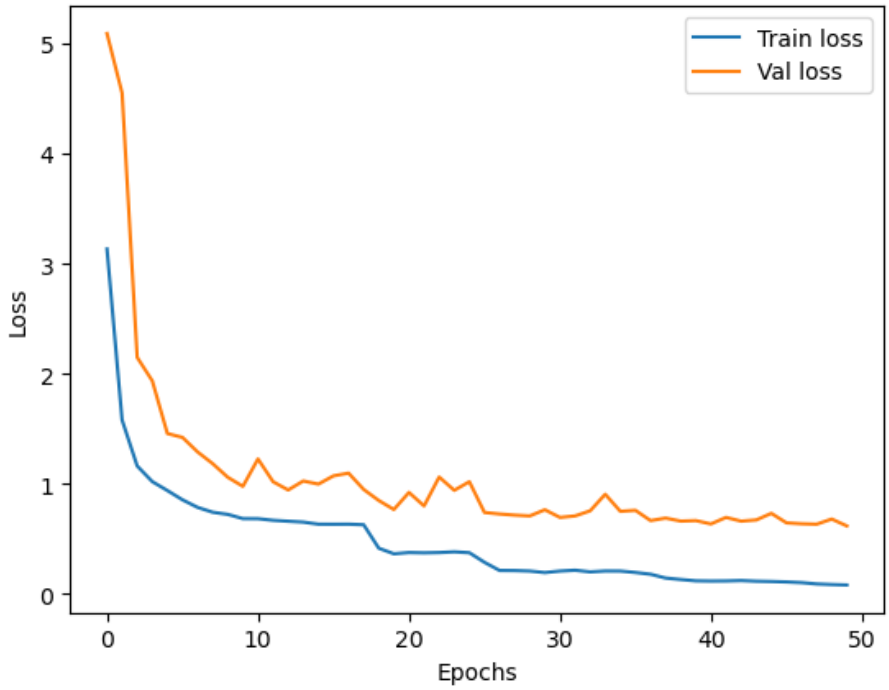

Figure 11: Training and validation loss graph for Model 2

The **Accuracy Graph** (Figure 6):

- The training accuracy (blue line) starts low and increases steadily, reaching close to 100%, indicating that the model is **learning well on the training data.**
- The validation accuracy (orange line) also improves initially but shows fluctuations and stabilizes at a lower level compared to training accuracy.
- The increasing gap between training and validation accuracy suggests overfitting, meaning the model is memorizing the training data rather than generalizing well to unseen data.

The **Loss Graph** (Figure 7):

- The training loss (blue line) decreases consistently, indicating that **the model is optimizing well on the training dataset.**
- The validation loss (orange line) also decreases at first but later fluctuates and remains higher than the training loss, confirming signs of overfitting.
- The fluctuations in validation loss suggest that the model might be struggling to generalize, possibly due to an insufficient amount of data, high model complexity, or lack of regularization.

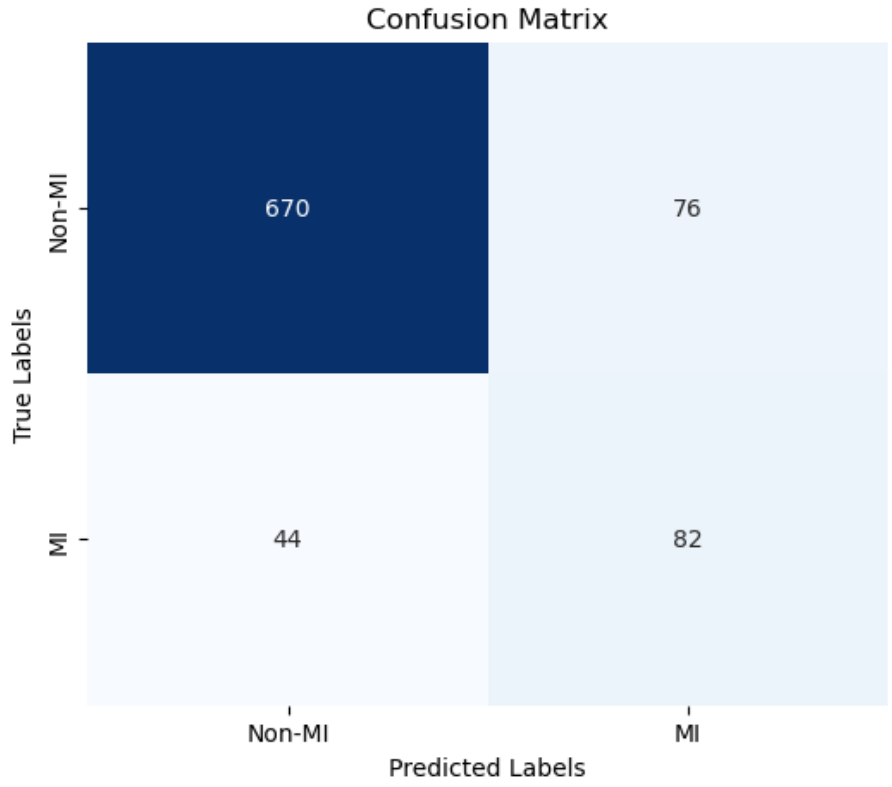

Figure 12: Confusion Matrix for Model 2

Overall Performance:

- The model performs **better at recognizing Non-MI cases** than MI cases.
- Its **low accuracy in detecting MI cases is a major limitation**, as missing heart attack cases can have serious medical implications.
- The **false positives (wrongly diagnosing MI) are less critical** than false negatives (missing MI patients), but still problematic.

5 ECG Signal Forecasting Using Convolutional Neural Networks (LSTM+CNN)

5.1 Data Preprocessing Steps

1. **Load ECG Data:** Load metadata from the CSV file and ECG signals using `wfdb.rdrecord`.
2. **Extract ECG Signals:** Extract the first 12 leads from each ECG signal for the specified patient.
3. **Handle Missing Files:** Skip any missing ECG files using a `try-except` block.
4. **Reshape Signals:** Ensure the ECG signal has the shape (samples, 12) for each patient.
5. **Create Sliding Windows:** Create input-output pairs using windows of size 800 to forecast the next 100 samples (forecast horizon).
6. **Split Data:** Split data into training (80%) and testing (20%) sets using `train_test_split`.
7. **Reshape Data for Model:** Reshape input (X) to (samples, window_size, 12) and output (y) to (samples, forecast_horizon * 12).

Hyperparameter	1st Model
Conv1D Filters (1st layer)	32
Conv1D Filters (2nd layer)	64
Conv1D Kernel Size (1st layer)	3
Conv1D Kernel Size (2nd layer)	3
MaxPooling1D Pool Size (1st layer)	2
MaxPooling1D Pool Size (2nd layer)	2
LSTM Units	64
LSTM Activation	ReLU
LSTM Return Sequences	False
Dense Units (1st layer)	128
Dense Units (2nd layer)	64
Dense Units (Output layer)	forecast_horizon * 12 (1200)
Dense Activation (1st, 2nd layers)	ReLU
Dense Activation (Output layer)	Linear
Optimizer	Adam
Loss Function	MSE (Mean Squared Error)
Epochs	50
Batch Size	32
Test Size	0.2
Forecast Horizon	100
Window Size	800

Table 3: Hyper-parameters for 1st Model.

5.2 Result

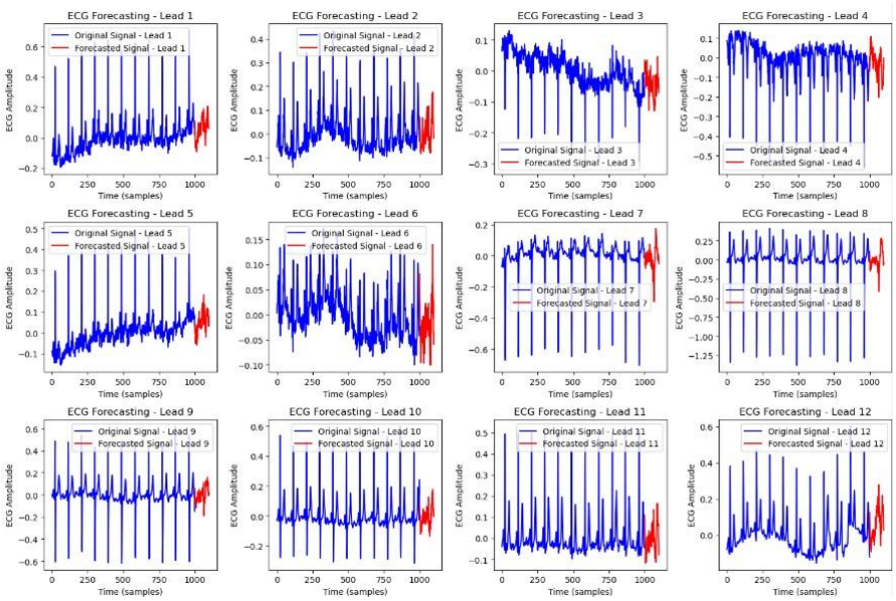

Figure 13: 12 ECG Lead Signal Forecasts for 1st Model.

5.3 Interpretation of Forecasting Results

- **Performance Across Leads:**
  - The model generally captures the underlying pattern and periodicity of the ECG waves.
  - In some leads (e.g., Leads 1, 5, 9), the forecasted signal (red) aligns well with the expected waveform.

- Other leads (e.g., Leads 3, 6, 7) show more variability or deviation, suggesting some difficulty in modeling those specific ECG patterns.

- **Noise and Signal Stability:**
  - Some leads exhibit more noise or less periodicity, making them harder to predict accurately (e.g., Lead 6 and Lead 7).
  - The model might struggle with high-frequency components or sudden amplitude variations.

- **Forecast Accuracy:**
  - The shape and trend of the forecasted signals resemble the original ECG, which is a good sign.
  - However, **some leads show noticeable drift or misalignment**, suggesting that further fine-tuning (e.g., adjusting the model architecture, optimizing hyperparameters) may be needed.

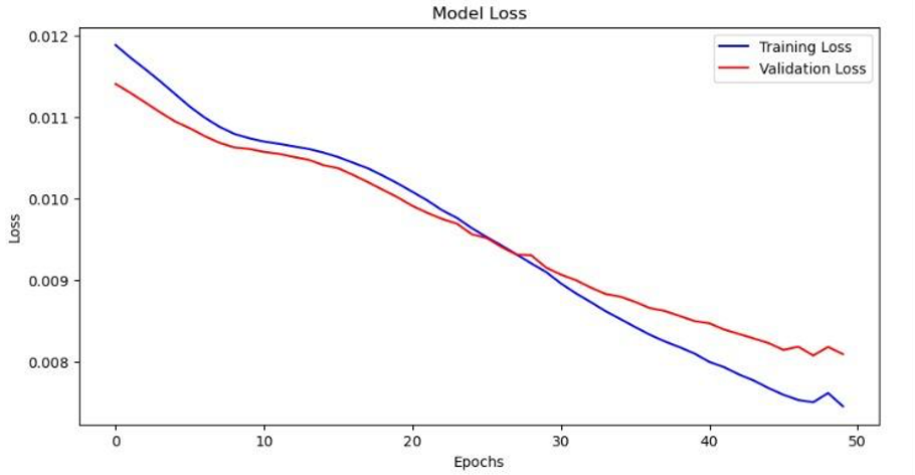

Figure 14: Training and Validation Loss graph for 1st Model.

This graph shows a model training process where both training and validation loss are decreasing, indicating the model is learning. However, the validation loss is consistently slightly higher than the training loss, which could suggest some minor overfitting. Overall, the training appears relatively successful, though further epochs might not yield significant improvement due to the flattening of the curves.

In the updated model, the addition of dropout layers, batch normalization, and L2 regularization aims to address the minor overfitting observed in the previous model. Dropout will help prevent overfitting by randomly ignoring neurons during training, while batch normalization will stabilize the learning process and improve convergence speed. L2 regularization will further help reduce overfitting by penalizing large weights. The introduction of early stopping will allow the model to halt training early if no significant improvement is seen, preventing unnecessary epochs and reducing overfitting. These changes are expected to improve the model's generalization, leading to a better validation loss and potentially less overfitting in the next training cycle.

Hyperparameter	**1st Model**	**2nd Model**
Conv1D Filters (1st layer)	32	32
Conv1D Filters (2nd layer)	64	64
Conv1D Kernel Size (1st layer)	3	5
Conv1D Kernel Size (2nd layer)	3	3
MaxPooling1D Pool Size (1st layer)	2	2
MaxPooling1D Pool Size (2nd layer)	2	2
LSTM Units (1st layer)	64	64
LSTM Units (2nd layer)	32	32
LSTM Activation	ReLU	Tanh
LSTM Return Sequences	False	True
Dense Units (1st layer)	128	128
Dense Units (2nd layer)	64	64
Dense Units (Output layer)	forecast_horizon * 12 (1200)	forecast_horizon * 12 (1200)
Dense Activation (1st, 2nd layers)	ReLU	ReLU
Dense Activation (Output layer)	Linear	Linear
Optimizer	Adam	Adam
Loss Function	MSE	MSE
Epochs	50	300
Batch Size	32	64
Test Size	0.2	0.2
Forecast Horizon	100	100
Window Size	800	800
Dropout Rate	N/A	0.4
Batch Normalization	N/A	Yes
Regularization (L2)	N/A	Yes (L2 regularization on Dense)
Early Stopping Patience	N/A	30 (increased)
Signal Filtering (Median Filter)	N/A	Yes (Median Filter with kernel 3)

Table 4: Hyper-parameter Comparison for 1st and 2nd Models

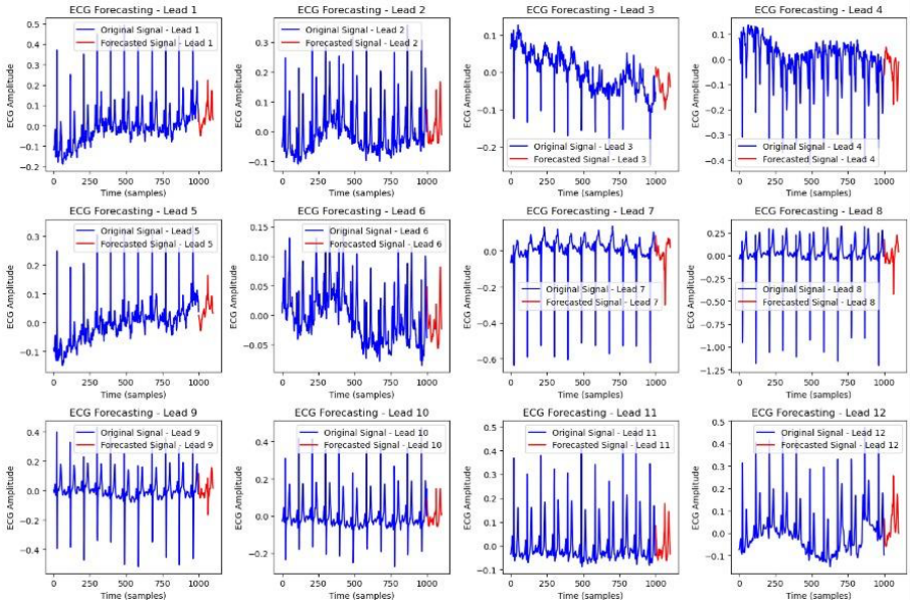

Figure 15: 12 ECG Lead Signal Forecasts for 2nd Model.

5.4 Analysis of Each Lead

- **Leads 1, 2, 5, 6, 9, 10, 11, 12:** These leads show some degree of correspondence between the forecasted (red) and original (blue) signals, particularly in capturing the general trend and some of the peaks. However, the forecasts are not perfect and often miss the finer details and accurate peak amplitudes.
- **Leads 3, 4, 7, 8:** The forecasting in these leads appears less accurate. The red lines often deviate significantly from the blue lines, indicating difficulty in capturing the signal patterns in these specific leads.

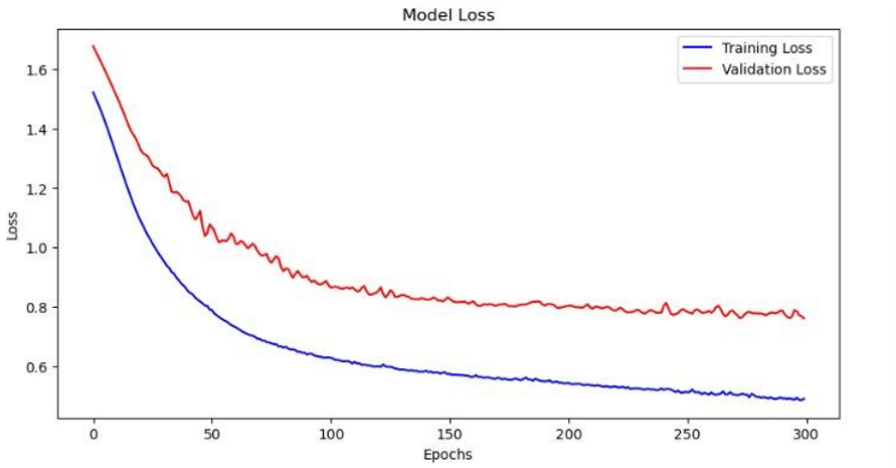

Figure 16: Training and Validation Loss for 2nd Model.

The graph shows that the model's training loss decreases steadily, but the validation loss plateaus at a higher value, indicating overfitting. This suggests the model is memorizing training data rather than generalizing. Compared to a previous run with better performance (smaller gap between losses), the current run's larger gap shows worsened overfitting, likely due to recent changes. To improve, strategies like stronger regularization, data augmentation, or model simplification should be explored to enhance generalization.

In this version, the model complexity has been reduced by using fewer Conv1D filters and LSTM units, making it more efficient while maintaining performance. The dropout rate has been increased to 0.4 to better prevent overfitting, and L2 regularization has been added to improve generalization. Additionally, the number of epochs has been reduced to 100, allowing for faster convergence without compromising accuracy.

Model Architecture	**1st Model**	**2nd Model**	**3rd Model**
Conv1D Filters	128, 256	64, 128	32, 64
Kernel Size (Conv1D)	5	3	5, 3
MaxPooling1D Pool Size	2	2	2
Hyperparameter	**Previous Model 1**	**Previous Model 2**	**Current Model**
Dropout Rate	0.2, 0.3	0.2, 0.3	0.4
LSTM Units	128, 64	128, 64	64, 32
Dense Layer Units	128, 64	128, 64	128
Dense Layer Activation	'relu'	'relu'	'relu'
Kernel Regularizer	None	None	'l2' (L2 regularization)
Loss Function	MSE (Mean Squared Error)	MSE (Mean Squared Error)	MSE (Mean Squared Error)
Optimizer	Adam	Adam	Adam
Epochs	50	300	100
Batch Size	64	64	64
Early Stopping Patience	20	30	30
Validation Split	0.2 (or no specific mention)	0.2	0.2 (explicit use of validation data)
Training Data Scaling	StandardScaler (X and y)	StandardScaler (X and y)	StandardScaler (X and y)
Input Shape	(800, 12)	(800, 12)	(800, 12)

Table 5: Model Architecture and Comparison of hyper-parameters settings for 1st, 2nd, and 3rd Models.

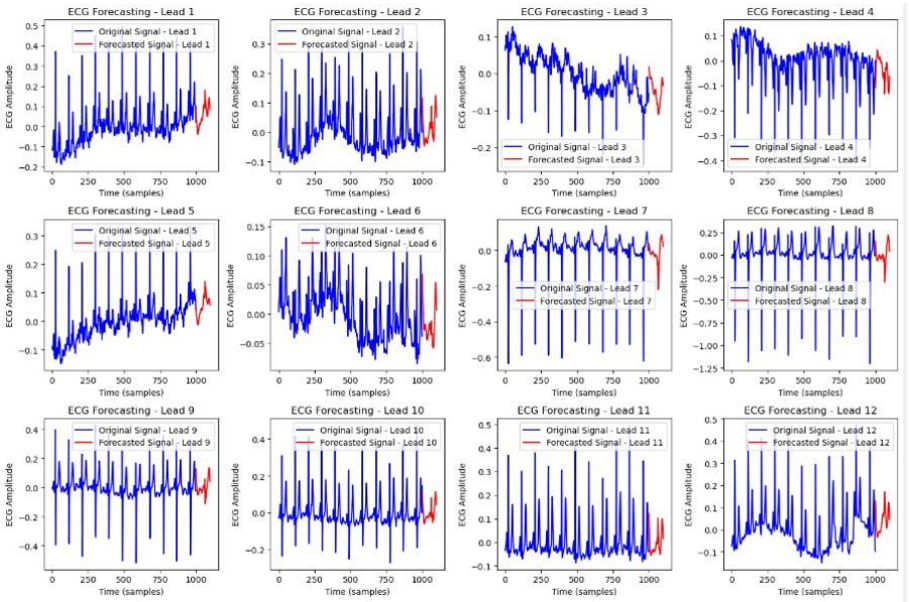

Figure 17: 12 ECG Lead Signal Forecasts for Model 3.

- **Leads 1, 2, 5, 6, 9, 10, 11, 12:** Forecasts capture the general trend and some peaks, but miss finer details and accurate amplitudes.
- **Leads 3, 4, 7, 8:** Forecasts are less accurate, deviating significantly from the original signal.

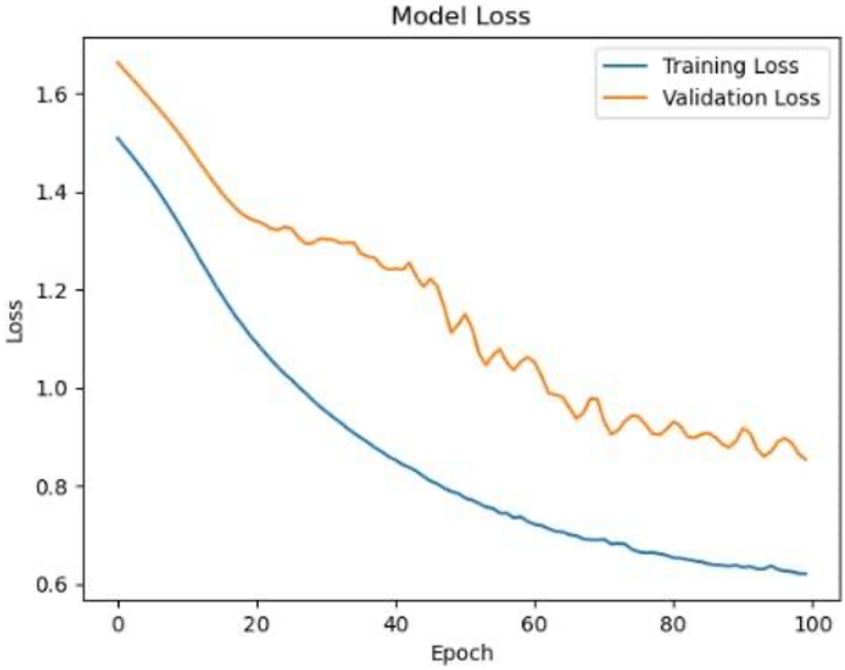

Figure 18: Training and Validation Loss graph for Model 3.

5.5 Loss Graph Analysis

- **Graph 1 (50 epochs, Good Convergence):**
  - **Training Loss:** Decreases steadily and significantly.
  - **Validation Loss:** Closely tracks the training loss, indicating good generalization and minimal overfitting. Achieves the lowest overall loss values.
- **Graph 2 (300 epochs, Overfitting):**
  - **Training Loss:** Decreases significantly, but not as consistently as Graph 1.
  - **Validation Loss:** Plateaus relatively early and shows a clear gap compared to the training loss, indicating overfitting. Higher final loss values compared to Graph 1.
- **Graph 3 (100 epochs, Early Plateau):**
  - **Training Loss:** Decreases steadily.
  - **Validation Loss:** Plateaus very early (around epoch 20), indicating a very early performance limit and potential overfitting. Loss values are somewhere in between Graph 1 and Graph 2.

The current version simplifies the architecture by using a single LSTM layer instead of two, reduces the number of epochs to 50 (compared to 100-300 in previous versions), and removes dropout and batch normalization for a faster, more efficient model. It also retains signal preprocessing (median filtering), focusing on improving model training speed while maintaining performance.

Hyperparameter	**1st Model**	**2nd Model**	**3rd Model**	**4th Model**
Conv1D Filters (1st layer)	32	32	32	32
Conv1D Filters (2nd layer)	64	64	64	64
Conv1D Kernel Size (1st layer)	3	5	5	3
Conv1D Kernel Size (2nd layer)	3	3	3	3
MaxPooling1D Pool Size (1st layer)	2	2	2	2
MaxPooling1D Pool Size (2nd layer)	2	2	2	2
LSTM Units (1st layer)	64	64	64	64
LSTM Units (2nd layer)	64	32	32	None (Only 1 LSTM layer)
LSTM Activation	ReLU	Tanh	ReLU	ReLU
LSTM Return Sequences	False	True	False	False
Dense Units (1st layer)	128	128	128	128
Dense Units (2nd layer)	64	64	64	64
Dense Units (Output layer)	forecast_horizon * 12 (1200)	forecast_horizon * 12 (1200)	forecast_horizon * 12 (1200)	forecast_horizon * 12 (1200)
Dense Activation (1st, 2nd layers)	ReLU	ReLU	ReLU	ReLU
Dense Activation (Output layer)	Linear	Linear	Linear	Linear
Optimizer	Adam	Adam	Adam	Adam
Loss Function	MSE	MSE	MSE	MSE
Epochs	50	300	100	50
Batch Size	32	64	64	32
Early Stopping Patience	N/A	30	30	N/A
Dropout Rate	N/A	0.4	0.2, 0.3	N/A
Batch Normalization	N/A	Yes	Yes	N/A
Regularization (L2)	N/A	Yes (L2 regularization on Dense)	None	None
Signal Filtering (Median Filter)	N/A	Yes (Median Filter with kernel 3)	N/A	Yes (Median Filter with kernel 3)
Validation Split	N/A	0.2	0.2	0.2
Training Data Scaling	StandardScaler (X and y)	StandardScaler (X and y)	StandardScaler (X and y)	StandardScaler (X and y)
Input Shape	(800, 12)	(800, 12)	(800, 12)	(800, 12)
Test Size	0.2	0.2	0.2	0.2
Forecast Horizon	100	100	100	100
Window Size	800	800	800	800

Table 6: Comparison of Hyperparameters for 1st, 2nd, 3rd, and 4th Models.

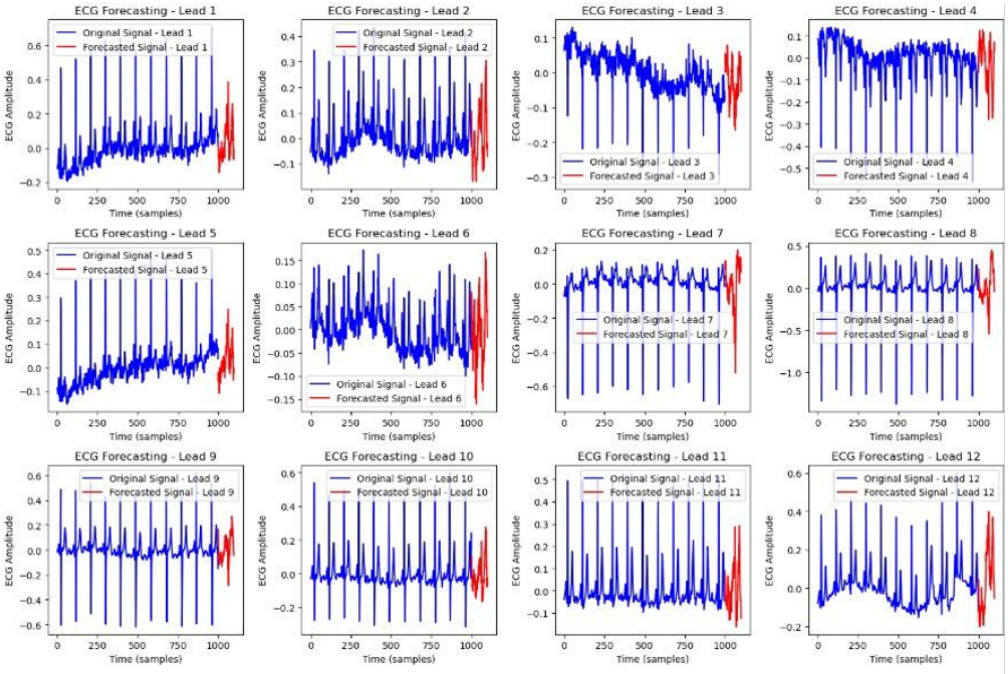

Figure 19: 12 ECG Lead Signal Forecasts for Model 4.

5.6 Specific Lead Observations

- **Leads 1, 2, 5, 6, 9, 10, 11, 12:** These leads generally show better agreement between the forecasted and original signals, although the amplitude and phase issues are still present.
- **Leads 3, 4, 7, 8:** These leads continue to be problematic. The forecasts often deviate significantly from the original signal, suggesting the model struggles to capture the patterns in these specific leads.

5.7 Key Observations

- **Attempt 1 and 4:** Appear to be the best attempts, with Attempt 1 potentially having a slight edge.
- **Attempt 3:** Clearly degraded the performance, likely due to over-smoothing or other changes that negatively impacted the model's ability to capture fine details.
- **Lead Variability:** This is a persistent challenge across all attempts. Further investigation into why some leads are more difficult to forecast than others is needed.
- **Data Limitations:** The limited data from a single patient continues to be a significant constraint.

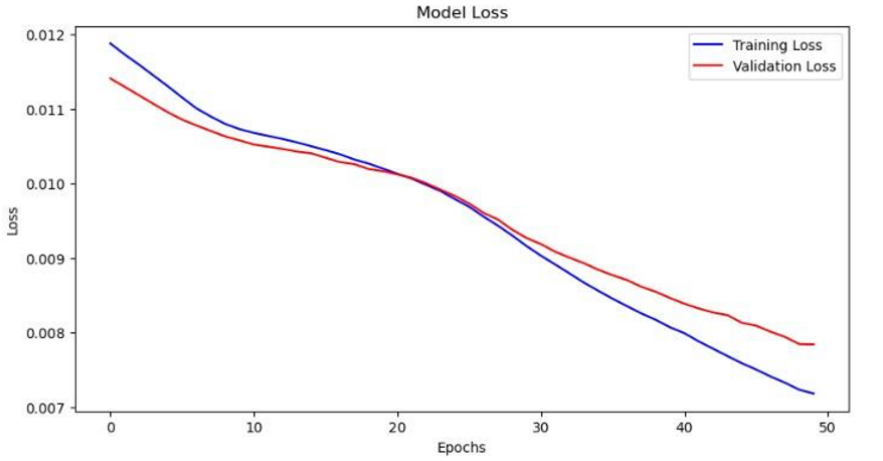

Figure 20: Training and Validation Loss

5.8 Loss Graph Analysis

Graph 1 (50 epochs, Good Convergence):

- **Training Loss:** Decreases steadily and significantly.
- **Validation Loss:** Closely tracks the training loss, indicating good generalization and minimal overfitting. Achieves the lowest overall loss values.

Graph 2 (300 epochs, Overfitting):

- **Training Loss:** Decreases significantly, but not as consistently as Graph 1.
- **Validation Loss:** Plateaus relatively early and shows a clear gap compared to the training loss, indicating overfitting. Higher final loss values compared to Graph 1.

Graph 3 (100 epochs, Early Plateau):

- **Training Loss:** Decreases steadily.
- **Validation Loss:** Plateaus very early (around epoch 20), indicating a very early performance limit and potential overfitting. Loss values are somewhere in between Graph 1 and Graph 2.

Graph 4 (50 epochs, Good Convergence Again):

- **Training Loss:** Decreases steadily and significantly, similar to Graph 1.
- **Validation Loss:** Closely tracks the training loss, again indicating good generalization and minimal overfitting. Loss values are comparable to Graph 1.

6 PTB-XL ECG Dataset: Data Composition & Characteristics

- **Total Records:** 21,799
- **Total Patients:** 18,869
- **Duration:** 10 seconds per ECG
- **Leads:** 12-lead standard ECG

6.1 Annotations & Metadata

- 71 ECG statements (diagnostic, form, and rhythm) based on SCP-ECG standard.
- Metadata includes:
  - Demographics: Age, sex, weight, height.
  - ECG characteristics: Heart axis, infarction stadium.
  - Signal quality metrics: Noise, baseline drift, artifacts.
- Validation: Some records reviewed by a second cardiologist.

6.2 Data Format & Sampling Rates

- Stored in WFDB (WaveForm DataBase) format.
- When you want to load a record using WFDB in Python, you typically specify just the base name of the record (i.e., without the .dat or .hea extension). WFDB will allow you to read and visualize the file.
- Two versions available:
  - 500 Hz (high-resolution data).
  - 100 Hz (downsampled for convenience).

6.3 Diagnosis Distribution

- Normal ECG (NORM): 9,514 records
- Myocardial Infarction (MI): 5,469 records
- ST/T Changes (STTC): 5,235 records
- Conduction Disturbance (CD): 4,898 records
- Hypertrophy (HYP): 2,649 records

	Section	Datatype	Description
2	**ecg_id**	integer	Unique ECG identifier
3	**patient_id**	integer	Unique patient identifier
4	**filename_lr**	string	Path to low-resolution waveform data (100 Hz)
5	**filename_hr**	string	Path to high-resolution waveform data (500 Hz)
6	**age**	integer	Age at recording in years
7	**sex**	categorical	Sex (male 0, female 1)
8	**height**	integer	Height in centimeters
9	**weight**	integer	Weight in kilograms
10	**nurse**	categorical	Involved nurse (pseudonymized)
11	**site**	categorical	Recording site (pseudonymized)
12	**device**	categorical	Recording device
13	**recording_date**	datetime	ECG recording date and time
14	**report**	String	ECG report from diagnosing cardiologist
15	**scp_codes**	dictionary	SCP ECG statements
16	**heart_axis**	categorical	Heart's electrical axis
17	**infarction_stadium1**	categorical	Infarction stadium
18	**infarction_stadium2**	categorical	Second infarction stadium
19	**validated_by**	categorical	Validating cardiologist (pseudonymized)
20	**second_opinion**	Boolean	Flag for second (deviating) opinion
21	**initial_autogenerated_report**	Boolean	Initial autogenerated report by ECG device
22	**validated_by_human**	Boolean	Validated by human
23	**baseline_drift**	string	Baseline drift or jump present
24	**static_noise**	string	Electric hum/static noise present
25	**burst_noise**	string	Burst noise
26	**electrodes_problems**	string	Electrodes problems
27	**extra_beats**	string	Extra beats
28	**pacemaker**	string	Pacemaker
29	**strat_fold**	integer	Suggested stratified folds

Figure 21: PTB-XL Dataset Characteristics

References

1. Makridakis, Spyros and et al. (2018). *Statistical and Machine Learning forecasting methods: Concerns and ways forward. An evaluation of traditional and modern forecasting methods, discussing the limitations of each.* arXiv.

2. Zhang, A., Zheng, Y. (2020). *Multivariate Time Series Forecasting with Missing Values. Focuses on handling incomplete multivariate time series data using imputation techniques.* IEEE Xplore.

3. Gamboa, J. C. B. (2017). *Deep Learning for Time-Series Analysis. A review paper discussing the applications of deep learning in multivariate time series.* ACM Digital Library.

4. Hyndman, R. J., Athanasopoulos, G. (2018). *Forecasting: Principles and Practice. Covers forecasting methods for both univariate and multivariate time series.* OTexts.

5. Salinas, D., Flunkert, V., et al. (2019). *DeepAR: Probabilistic Forecasting with Autoregressive Recurrent Networks. Often cited in newer research on deep learning for time series.* Connected Papers.

6. Talent500. (2023). *Time Series Forecasting with Long Short-Term Memory (LSTM) Networks in TensorFlow.* Retrieved from https://talent500.com/blog/time-series-forecasting-with-long-s

7. Joseph, M. (2022). *Modern Time Series Forecasting with Python: Explore Industry-Ready Time Series Forecasting Using Modern Machine Learning and Deep Learning.* Packt Publishing, Limited. Retrieved from http://ebookcentral.proquest.com/lib/frankfurtmain/detail.action?docID=30259471.

8. Pandey, A., Mannepalli, P.K., Gupta, M., et al. (2024). *A Deep Learning-Based Hybrid CNN-LSTM Model for Location-Aware Web Service Recommendation. Neural Process Letters*, 56, 234. doi: https://doi.org/10.1007/s11063-024-11687-w.