Unsupervised Anomaly Detection on ECG Data

James Li and Jeff Zhao
Department of Computer Science
University of Texas at Austin
Austin, TX 78712
lijmr889@gmail.com
jeffzhao100@gmail.com

April 29, 2024

GitHub Repository

https://github.com/lijmr/ecg-autoencoder

1 Background

Our project is focused on building an autoencoder neural network to detect anomalies from electrocardiography (ECG) readings. ECG readings are commonly used to see if there is any abnormal heart activity, which aids medical professionals in identifying any issues with a patient's health. Our aim is to construct a model that will precisely indicate which parts of an ECG reading have a high chance to be considered an anomaly, with the hope of helping doctors diagnose different heart diseases more effectively.

2 Dataset

The dataset that we used to train and test our model is the PTB-XL ECG dataset (4). This dataset contains ECG waveform data from 21,799 patients as well as metadata from each patient. Each record contains 10 seconds long waveform data composed of 12 different ECG leads, with a sampling frequency of 100 Hz. The patient metadata contains 28 features, some of which are patient demographics such as age and sex, as well as others, such as the nurse that took the ECG reading.

In our data cleaning stage, we narrowed down our metadata to contain four features: age, sex, device type, and whether the record was validated by someone. The decision to remove the other features was due to high class imbalances as well as high counts of null values. The labels initially included various the types of abnormal heart conditions, but we changed these to be labeled "ABNORMAL" to create the binary supervised classification problem in our transfer learning experiment. However, this left us with a class imbalance of the labels; this was fixed by undersampling the abnormal records. We ended up with a total of 18,128 records, half of which are normal ECG records and the other half of which are abnormal ECG records.

3 Autoencoder Models

For our autoencoder, we decided to experiment and analyze the differences between two distinct model architectures: JeffNet and JamesNet. JeffNet is constructed of a combination of a CNN autoencoder and a LSTM, and JamesNet composes of a CNN autoencoder and a MLP.

3.1 Design

The fundamental building blocks for our autoencoder model were a CNN autoencoder and a separate metadata processing model. The CNN autoencoder is used to learn the patterns from each ECG waveform and reconstruct the waveforms based off of the patterns that it learns. The metadata processing model is used to learn from each ECG waveform's corresponding patient metadata. The CNN autoencoder and metadata processing model were then concatenated and attached to a fully connected layer to form our final autoencoder model that takes in one record of ECG data and its respective patient metadata as inputs and outputs a reconstructed waveform.

The decision to use a CNN for learning from ECG waveforms largely lies in the fact that each 1000 x 12 waveform has a grid-like structure. Similar to how CNNs are effective in learning from image or sound data, the grid-like structure of ECG readings is perfect for CNNs in detecting important features throughout the waveform.

The LSTM model that we used in JeffNet to process metadata was mostly an arbitrary decision, though we recognized that there was also potential for our model to relate patterns seen in one record to another record seen earlier in training. The MLP model in JamesNet was chosen due to the relatively simplistic nature of MLPs; we believed that the simplicity could potentially capture more complexities in the metadata than the LSTM could.

3.2 Training & Evaluation

The train-test split we chose was 80% training data and 20% testing data. The training set that we used to train our autoencoder was purely composed of normal ECG waveform data and their respective metadata records. This was done in order to teach our autoencoder how to most accurately reconstruct a normal waveform so that the most anomalies would be detected on abnormal waveforms. The testing set was split into a normal ECG dataset and an abnormal

ECG dataset to compare the differences in how many anomalies were detected each kind of data. The ECG data and patient metadata were both normalized and one-hot encoded. We also removed the labels since anomaly detection is an unsupervised task.

In both JeffNet and JamesNet, the loss function used was MSE and the Adam optimizer was used at a learning rate of 5×10^{-3} Both models were also trained for 40 epochs with batch sizes of 32. During our training of JeffNet, the loss per epoch gradually decreased until it converged at a final training loss of 1.35×10^{-4} In comparison, we trained JamesNet until it reached a loss of 7.47×10^{-5} , with the loss also gradually decreasing over epochs.

For testing our autoencoder models, we also chose to use MSE to calculate the reconstruction errors between the input test data and the resulting reconstructed outputs. We recorded the average reconstruction errors across normal and abnormal test data; the threshold for detecting anomalies was chosen from the 99th percentile of reconstruction errors obtained from the normal test dataset. This same threshold is used to count the number of anomalies in both the normal and abnormal test datasets. Using this threshold, we also recorded the mean, minimum, maximum, lower quartile, and upper quartile anomaly counts as metrics for comparing JeffNet and JamesNet.

3.3 Results

Metric	**JeffNet**	**JamesNet**
99th percentile threshold	7.05×10^{-4}	7.26×10^{-4}
Normal Test Data		
Avg. reconstruction error	1.0×10^{-4}	1.1×10^{-4}
Anomalies		
Mean	120.0	120.0
Max	3683.0	6127.0
Upper quartile	127.0	122.0
Median	90.0	82.0
Lower quartile	68.0	63.0
Min	31.0	33.0
Abnormal Test Data		
Avg. reconstruction error	1.9×10^{-4}	1.9×10^{-4}
Anomalies		
Mean	374.04	368.05
Max	5656.0	5517.0
Upper quartile	389.75	401.5
Median	170.5	169.5
Lower quartile	98.0	92.25
Min	29.0	32.0

Table 1: Comparison of JeffNet and JamesNet

From what we observed from the results, all of the test metrics for JeffNet and JamesNet were very similar across the board. The main difference that we noticed was that it was much easier to lower the training loss for JamesNet than JeffNet; JamesNet would consistently reach a final loss of lower than 1.0×10^{-4}, but JeffNet would consistently reach a final loss of around 1.3×10^{-4}. JeffNet's difficulties in converging to a lower training loss may indicate that the LSTM in JeffNet is not very good at learning the patterns found in patient metadata. This is not entirely surprising to us; the patient metadata has features that are all unrelated, which pose a greater challenge for LSTMs to track dependencies in patterns across records. Additionally, the independent features of the metadata may better suit the simplistic natures of MLPs, allowing JamesNet to better understand the patterns in normal ECG records.

Despite these differences in training results, both models were exceptional in detecting more anomalies in abnormal ECG records. For JeffNet, the median was 89.4% greater and the mean was 211.7% greater in the number of anomalies detected for abnormal ECG data than normal ECG data. JamesNet performed slightly better with the median being 106.7% greater for abnormal data while the mean was slightly worse being 206.7% greater for abnormal data. With how similar the mean is for both models, the significant percent difference for the median shows that JamesNet is better at detecting anomalies than JeffNet. Furthermore, the better training performance of JamesNet may indicate a higher ceiling for JamesNet in fitting better to normal data and detecting more anomalies in abnormal ECG records.

4 Experiment #1: Transfer Learning

4.1 Purpose

Our first computational experiment involves transfer learning our autoencoder to a supervised classification problem. The purpose of this experiment is to assess how effective a model trained to detect anomalies from unsupervised data performs on a classification problem that classifies normal/abnormal ECG readings. If the supervised classifier performs well on the test set, the unsupervised model's learning could be useful in specifically identifying whether or not ECG readings indicate an abnormal heart condition.

Hypothesis Will transfer learning JamesNet to a supervised classifier result in greater than 50% accuracy for detecting abnormal ECG data?

4.2 Classification Model

The transfer learning classification model is built upon the frozen layers of JamesNet, which are then attached to a classification head that determines if the inputs are normal or abnormal. We chose to use JamesNet for this experiment due to it performing marginally better than JeffNet. Since the classification task doesn't require reconstruction of ECG waveform data, the only parts of JamesNet that we decided to transfer over was the encoder layer of the CNN autoencoder and the MLP used to process patient metadata. We concatenated the CNN encoder and MLP, and passed the output from the concatenated models into our classification head. The classification head is built with 6 linear layers with dropout, batch normalization, and ReLU activation used between each linear layer.

Unlike our original autoencoder datasets, our classification training and testing sets includes both normal and abnormal ECG data with their labels attached; we do this so that the classifier is able to learn how to distinguish the differences between normal and abnormal records. We kept our train-test split at 80/20 and chose to not include a validation set.

In training our transfer learning classifier, we used binary cross entropy with logits as the loss function and Adam for the optimizer at a learning rate of 2×10^{-5}. We also used L2 regularization to add onto our loss per epoch. Throughout training, we found that getting the loss to converge was very difficult; it would consistently be fluctuating between 0.47 and 0.63. We also chose to use batch sizes of 64 to combat the training instability that we were facing. At the end of 10 epochs, our final loss was 0.519.

4.3 Results

The transfer learning classification model produced a test accuracy of 68.4%, and our confusion matrix produced 1,209 true positives, 1,271 true negatives, 543 false positives, and 605 false negatives (we are using "normal" as positive and "abnormal" as negative). Even though our model was eventually able to have an accuracy of greater than 50%, we had a lot of difficulty in actually creating a model that would perform better than a coin flip. We consistently produced test accuracies of around 50%, and some of the test results we had would classify all of the ECG test data as normal. Additionally, around 14.96% of our results were false positives; while this is not bad, we ideally would want this to be minimized as the worst case scenario is incorrectly diagnosing patients with having normal heart conditions when they actually have problems.

From this experiment, we learned that transfer learning is actually very difficult. We were hoping that using a model that already knows what normal ECG data should look like would help the classifier with identifying differences between normal and abnormal data. However, this seemed to produce the opposite effect, with some of our previous classifiers attempting to classify every input as normal ECG records. Only when we added more layers and neurons to the classification head were we able to achieve higher test accuracy.

5 Experiment #2: TCN Autoencoder

5.1 Purpose

The second computational experiment involves building a temporal convolutional network (TCN) autoencoder. We based this experiment on papers such as Thill et al., 2021 (3) and Bai et al., 2018 (1), and we want to compare how well an autoencoder developed by researchers compares with our own model.

Hypothesis The custom TCN, with its ability to handle longer and shorter-term ECG information through dilations, will lead to better performance than our custom-made models.

5.2 TCN Autoencoder

Experiment 2 features the use of a specialized temporal convolutional network (TCN). Various papers, along with Large Language Models, suggested that these types of models have numerous benefits and may lead to better results (3).

We use an external library (2) to implement TCN blocks due to the repetitive and complex nature of manual implementation. This TCN that I am using is much more sophisticated than a Conv1D with dilation with multiple available features. Each layer consists The current configuration used was 3 residual blocks with 32, 64, and 128 channels respectively. Kernel size was 4 and epochs trained was 15. This is an example of a residual block:

1. Dilated Causal Convolutional Layer (dilation = $2^{(n-1)}$)
2. Weight Normalization
3. ReLU Activation
4. Dropout (p = 0.2)
5. Dilated Causal Convolutional Layer (dilation = $2^{(n-1)}$)
6. Weight Normalization
7. ReLU Activation
8. Dropout (p = 0.2)

There is also an output_projection parameter to project the output to the desired dimensionality, in this case, the number of inputs.

Metadata is handled with linear layers, which are concatenated, unsqueezed, and expanded along the time dimension to match the shape of the encoded output, then sent to the decoder.

Given that this was likely more complex than the other models, we decided 3 residual blocks were enough to handle the ECG data and utilize the varying dilations. We also chose to use linear layers instead of embedding layers since they are not too sparse.

5.2.1 Results

The TCN showed similar results to JeffNet, displaying similar 99th percentile thresholds and reconstruction errors. What was notable was that the TCN had far lower anomaly counts for both anomalous and regular data. From analyzing a few graphs I believe the TCN had much greater success replicating the original waveform, but there may be a risk of memorizing it. This is particularly evident in the abnormal test loader's median, which may suggest a skewed dataset However, this also could be because of model differences and potentially a lower threshold might be more optimal for detecting anomalies.

99th percentile threshold: 9.73×10^{-4}

Normal test data	
Avg. reconstruction error:	1.0×10^{-4}
Anomalies:	
Mean:	2.533
Max:	152.2
Upper quantile:	0.0
Median:	0.0
Lower quantile:	0.0
Min:	0.0
Abnormal test data	
Avg. reconstruction error:	2.3×10^{-4}
Anomalies:	
Mean:	7.532
Max:	428.0
Upper quantile:	7.0
Median:	0.0
Lower quantile:	0.0
Min:	0.0

It was also very quick to train compared to the other models. Along with this, time spent tuning hyperparameters was far less so it is more than likely to achieve much higher performance. Overall, this model is very promising but definitely needs more work.

6 Using ChatGPT

ChatGPT and other LLMs (mainly Opus 3 and Bing Copilot) were used in a variety of applications, from quickly writing up training and testing code to creating visualization and utility functions. Additionally, we used ChatGPT and LLMs to aid us in various debugging scenarios, providing us templates for how to construct our models, and suggestions in improving our models. While they were used for consulting, we were careful to collaborate their results with additional evidence, either internet articles or information obtained from our own analysis. LLMs also helped write the model cards and parts of the data sheet, especially when turning them into Latex.

7 Contributions

Jeff worked primarily on the second experiment, including research, finding libraries, programming, training, and testing. During preprocessing of metadata, Jeff helped with choosing which features to remove, as well as normalization and one-hot encoding the metadata. However, he did utilize preexisting functions and data loaders from James. While not creating JeffNet, Jeff contributed greatly to hyperparameter tuning and provided some of the starter code for training. Jeff also helped in several stages of the classifier (experiment 1), notably hyperparameter tuning, and also contributed to early versions of turning preprocessed data into tensor data that the models would be able to use for training and testing. He also wrote the TCN autoencoder, Using ChatGPT, Model Cards, Datasheet, and References sections of the paper.

James implemented the code for the Cleaning and Analysis notebook for preprocessing data. For the Modeling notebook, he also implemented the train-test split and normalization of the ECG waveform data. James took the starting implementations of JeffNet, created a majority of the final model and training function, and fully worked on the testing functions for the model. JamesNet was also implemented by James, but the code structure was almost completely the same as JeffNet for initialization, training, and testing, aside from the new MLP. The transfer learning classifier experiment was also mostly coded by James, with the exception of hyperparameter tuning and model adjustments made by Jeff. For the paper, James wrote the Background, Dataset, Autoencoder Models, and Transfer Learning sections of the paper.

8 Model Cards

8.1 JeffNet

8.1.1 Model Overview

JeffNet is a model that combines a CNN autoencoder and an LSTM model for processing ECG data and metadata. The outputs of the CNN autoencoder and LSTM model are concatenated and passed through a fully connected layer to generate the final output.

8.1.2 Model Architecture

JeffNet consists of the following components:

- **CNN Autoencoder**: A CNN autoencoder model that processes the ECG data.
- **LSTM Model**: An LSTM model that processes the metadata.
- **Fully Connected Layer**: A fully connected layer that takes the concatenated outputs of the CNN autoencoder and LSTM model and generates the final output.

8.1.3 Training Procedure

JeffNet is trained using the MSE loss and the Adam optimizer. The training loop iterates over the specified number of epochs, and the loss and epoch progress is printed every 5 epochs. The LSTM hidden and cell states are detached from the graph to prevent backpropagation through time.

8.1.4 Hyperparameters

The key hyperparameters of JeffNet include:

- `nepoch`: The number of training epochs.
- `lr`: The learning rate for the Adam optimizer.

8.2 JamesNet

8.2.1 Model Overview

JamesNet is a model that combines a CNN autoencoder and an MLP model for processing ECG data and metadata. The outputs of the CNN autoencoder and MLP model are concatenated and passed through a fully connected layer to generate the final output.

8.2.2 Model Architecture

JamesNet consists of the following components:

- **CNN Autoencoder**: A CNN autoencoder model that processes the ECG data.
- **MLP Model**: An MLP model that processes the metadata.
- **Fully Connected Layer**: A fully connected layer that takes the concatenated outputs of the CNN autoencoder and MLP model and generates the final output.

8.2.3 Training Procedure

JamesNet is trained using the MSE loss and the Adam optimizer. The training loop iterates over the specified number of epochs, and the loss and epoch progress is printed every 5 epochs.

8.2.4 Hyperparameters

The key hyperparameters of JamesNet include:

- `nepoch`: The number of training epochs.
- `lr`: The learning rate for the Adam optimizer.

8.3 TransferLearningClassifier

8.3.1 Model Overview

The TransferLearningClassifier is a model that combines a pre-trained CNN encoder and an MLP model for binary classification tasks. The encoder of the CNN and the MLP model are frozen, and their outputs are concatenated and passed through a fully connected network for classification.

8.3.2 Model Architecture

The TransferLearningClassifier consists of the following components:

- **Encoder**: The encoder of a pre-trained CNN autoencoder, which is frozen during training.
- **MLP Model**: A pre-trained MLP model, which is also frozen during training.
- **Fully Connected Network**: A sequence of fully connected layers with batch normalization, ReLU activation, and dropout. The network takes the concatenated outputs of the encoder and MLP model and performs binary classification.

The fully connected network architecture consists of a series of linear layers with a decreasing number of neurons, batch normalization, relu activations, and dropout.

8.3.3 Input and Output

The TransferLearningClassifier takes two inputs:

- `ecg_data`: The ECG waveform data with shape (batch_size, num_leads, sequence_length).
- `metadata`: The metadata associated with each ECG record.

The output of the model is a single value representing the binary classification prediction (0 or 1).

8.3.4 Training Procedure

The TransferLearningClassifier is trained using the Binary Cross-Entropy with Logits Loss (BCEWithLogitsLoss) and the Adam optimizer. The encoder and MLP model weights are frozen during training, and only the weights of the fully connected network are updated.

8.3.5 Hyperparameters

The key hyperparameters of the TransferLearningClassifier include:

- `lr`: The learning rate for the Adam optimizer (default: 2e-5).
- Dropout probabilities in the fully connected network (default: 0.5).

8.4 TCN Autoencoder

8.4.1 Model Description

The TCN Autoencoder is an unsupervised learning model that consists of an encoder and a decoder, both implemented using TCN blocks. The encoder compresses the input ECG signal into a lower-dimensional representation, while the decoder reconstructs the original signal from the compressed representation. The model also incorporates metadata information, such as age, sex, device, and validation status, by embedding them and concatenating them with the encoded representation.

8.4.2 Model Architecture

The TCN Autoencoder consists of the following components:

- **Encoder**: A TCN block that takes the input ECG signal and compresses it into a lower-dimensional representation.
- **Metadata Embeddings**: Separate embedding layers for age, sex, device, and validation status. These embeddings are concatenated and expanded to match the temporal dimension of the encoded representation.
- **Decoder**: Another TCN block that takes the concatenated encoded representation and metadata embeddings and reconstructs the original ECG signal.

8.4.3 Training Procedure

The TCN Autoencoder is trained using the mean squared error (MSE) loss between the input ECG signal and the reconstructed signal. The model is trained on normal ECG data with the goal of learning to reconstruct normal patterns accurately.

8.4.4 Evaluation

The model is evaluated on a separate test dataset that includes both normal and abnormal ECG signals. The reconstruction error is used as an anomaly score, and a threshold is determined based on the reconstruction errors of the normal data. The model's performance can be assessed using metrics such as precision, recall, and F1 score, depending on the specific requirements of the application.

9 Datasheet

9.1 Motivation

The PTB-XL ECG dataset from Wagner et al., 2020 (4) is a large dataset of 21799 clinical 12-lead ECGs with comprehensive annotations covering a wide range of cardiac conditions.

9.2 Composition

The cleaned dataset consists of the following:

1. cleaned_ptbxl_metadata.csv: Contains metadata for each ECG record, including age, sex, device, SCP codes, validation status, diagnostic superclass, and file paths to the waveform data.

2. Waveform data: CSV files containing the ECG waveform data for each record, stored in a directory structure similar to the original dataset. When imported with the 100 hz rate, the rate we chose, the dataset consists of 1000 timesteps with 12 ECG channel values.

References

[1] Shaojie Bai, J. Zico Kolter, Vladlen Koltun. (2018). An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling.

[2] Krug, P. (2023). PyTorch-TCN: Streamable (Real-Time) Temporal Convolutional Networks in PyTorch [Computer software]. GitHub. `https://github.com/paul-krug/pytorch-tcn`

[3] Thill, M., Konen, W., Wang, H., Bäck, T. H. W. (2021). Temporal convolutional autoencoder for unsupervised anomaly detection in time series. Applied Soft Computing, 112. 10.1016/j.asoc.2021.107751

[4] Wagner, P., Strodthoff, N., Bousseljot, RD. et al. (2020). PTB-XL, a large publicly available electrocardiography dataset. Scientific Data, 7, 154. 10.1038/s41597-020-0495-6