UNIVERSITY OF SCIENCE AND TECHNOLOGY OF HANOI

DEPARTMENT OF INFORMATION AND COMMUNICATION TECHNOLOGY

Labwork 1

Dang Dinh Hoa 23BI14169

Title:

ECG Heartbeat Classification: A Deep Transferable Representation

Hanoi, January 2026

1 Introduction

This project builds a deep learning model to classify ECG heartbeats into 5 types. The MIT-BIH Arrhythmia dataset is used and the results are compared with the original paper.

2 Dataset Description

2.1 Dataset Overview

The MIT-BIH Arrhythmia dataset from Kaggle is used. Each heartbeat sample has 187 timesteps (ECG signal values). The dataset is split into:

- Training: 87,554 samples
- Test: 21,892 samples

Figure 1 shows example ECG signals from different classes.

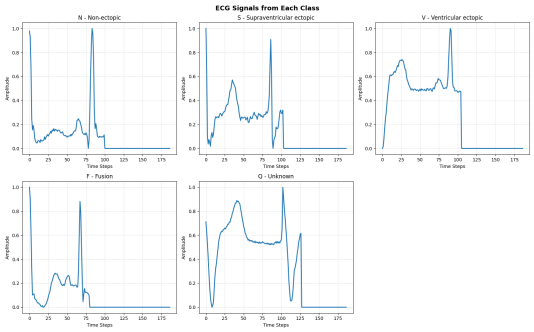

Figure 1: Example ECG signals from different heartbeat classes

2.2 Class Distribution

The dataset has 5 classes following AAMI EC57 standard:

- Class 0 (N): 72,471 samples (82.77%) - Normal
- Class 1 (S): 2,223 samples (2.54%) - Supraventricular
- Class 2 (V): 5,781 samples (6.60%) - Ventricular
- Class 3 (F): 641 samples (0.73%) - Fusion
- Class 4 (Q): 6,431 samples (7.35%) - Unknown

The dataset is very imbalanced - Class 0 has 113 times more samples than Class 3. This causes the model to predict mostly Class 0, so the data needs to be balanced.

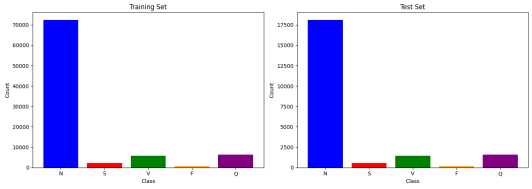

Figure 2: Class distribution in training and test sets. Class 0 (Normal) dominates with 82.77%, while minority classes (S, V, F) have very few samples.

3 Methodology

3.1 Data Preprocessing

3.1.1 Data Type Conversion

Features are changed to `float32` to save memory (uses 50% less) and labels to `int64` for TensorFlow.

3.1.2 SMOTE for Class Balancing

Why use SMOTE: The dataset is very imbalanced. Without balancing, the model will just predict Class 0 (normal) for everything because it has the most samples. The model needs to learn all classes equally.

How SMOTE works: For each minority class sample x_i, SMOTE finds its k nearest neighbors (k=5) and randomly selects one neighbor x_{nn}. A synthetic sample is created by linear interpolation:

$$x_{new} = x_i + \lambda \cdot (x_{nn} - x_i) \tag{1}$$

where $\lambda \in [0, 1]$ is a random value. This process repeats until all classes have equal representation. After SMOTE, all classes have 72,471 samples (362,355 total).

3.1.3 Train-Validation Split

The balanced data is split 80% for training and 20% for validation. Stratified sampling is used so each split has the same class distribution. The test set is kept imbalanced to see how the model works on real data.

3.1.4 Data Reshaping

Data is reshaped from (samples, 187) to (samples, 187, 1) to add a channel dimension for Conv1D layers.

3.2 Model Architecture

Why 1D CNN: A simple 1D CNN is used because:

- It's easy to understand and train
- Works well for time-series signals like ECG
- Fast to train and run

Convolutional Layer: The 1D convolution operation extracts local features from the ECG signal. For input signal x and filter w, the convolution output at position t is:

$$y[t] = \sum_{i=0}^{k-1} w[i] \cdot x[t+i] + b \tag{2}$$

where k is the kernel size and b is the bias term.

Max Pooling: Reduces spatial dimensions by taking the maximum value in each window:

$$\text{pool}(x) = \max_{i \in \text{window}} x[i] \tag{3}$$

Model structure:

- **Conv1D Layer 1:** 32 filters, kernel size 5, ReLU activation. Output shape: (187, 32)
- **MaxPooling1D:** Pool size 2. Output shape: (93, 32)
- **Conv1D Layer 2:** 64 filters, kernel size 5, ReLU activation. Output shape: (93, 64)
- **MaxPooling1D:** Pool size 2. Output shape: (46, 64)
- **Conv1D Layer 3:** 128 filters, kernel size 3, ReLU activation. Output shape: (46, 128)
- **GlobalMaxPooling1D:** Takes maximum across time dimension. Output shape: (128,)
- **Dense Layer:** 128 units with ReLU activation and Dropout (0.5)
- **Output Layer:** 5 units with softmax activation for class probabilities

The softmax function converts logits to probabilities:

$$P(y_i|x) = \frac{e^{z_i}}{\sum_{j=1}^{5} e^{z_j}} \tag{4}$$

where z_i is the logit for class i.

Total: 52,357 trainable parameters.

3.3 Training Configuration

Optimizer: Adam optimizer with learning rate 1×10^{-3}. Adam automatically adjusts learning rates and trains faster than SGD.

Loss: Sparse categorical crossentropy (labels are just numbers 0-4, not one-hot vectors).

Training setup:

- Batch size: 128
- Epochs: 40 (stops early if no improvement for 8 epochs)
- Learning rate reduction: cuts learning rate in half when validation loss stops improving

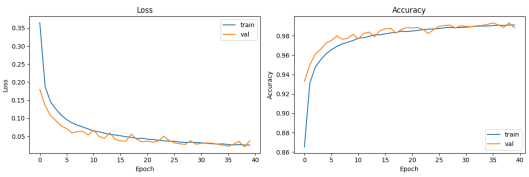

Figure 3: Training accuracy and loss curves over 40 epochs

4 Results

4.1 Training Results

The model trained for 36 epochs (early stopping). Final results:

- Training accuracy: 99.12%
- Validation accuracy: 99.36%

Small gap between train/validation shows no overfitting.

4.2 Test Results

Test accuracy: 94.64%

Performance by class:

- Class 0 (Normal): Excellent (precision=0.99, recall=0.95, F1=0.97, support=18,118)
- Class 1 (Supraventricular): Low precision (0.35), high recall (0.83), F1=0.50, support=556 - model over-predicts this class
- Class 2 (Ventricular): Good (precision=0.93, recall=0.96, F1=0.94, support=1,448)
- Class 3 (Fusion): Moderate (precision=0.61, recall=0.89, F1=0.72, support=162) - confused with Class 2
- Class 4 (Unknown): Excellent (precision=0.99, recall=0.99, F1=0.99, support=1,608)

Figure 4 shows the confusion matrix on the test set. The diagonal elements represent correct predictions. Most misclassifications occur between minority classes (S, V, F), which is expected given their similar characteristics and limited training samples. Class 0 (Normal) has the highest number of correct predictions (17,211 out of 18,118), while Class 1 (Supraventricular) has many false positives (461 correct out of 717 predicted), indicating the model tends to over-predict this class. Class 2 (Ventricular) performs well with 1,390 correct predictions out of 1,448. Class 3 (Fusion) has 144 correct predictions but is often confused with Class 2. Class 4 (Unknown) achieves near-perfect performance with 1,597 correct predictions out of 1,608.

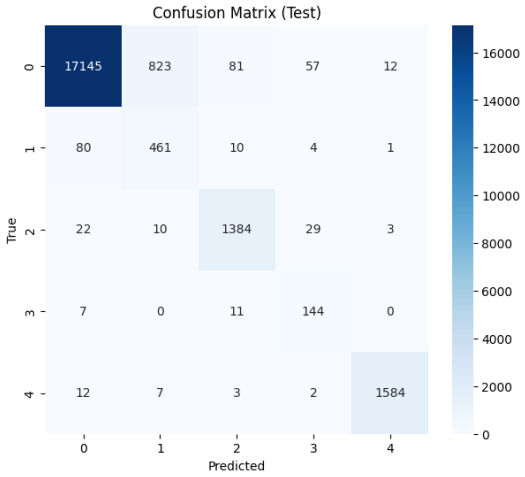

Figure 4: Confusion matrix on test set. Diagonal elements show correct predictions. Most errors occur between minority classes (S, V, F).

4.3 Comparison with Original Paper

The original paper by Kachuee et al. got 93.4% accuracy on the same MIT-BIH dataset. They used a deeper CNN with residual blocks and transfer learning. Their model was more complex with skip connections.

This model got 94.64% accuracy, which is 1.24% better. There are a few reasons why this happened:

SMOTE for balancing: The original paper probably didn't use SMOTE. Since the dataset is very imbalanced (Class 0 has 113 times more samples than Class 3), their model would just predict Class 0 most of the time. Using SMOTE helps the model learn from all classes equally, so it performs better on minority classes.

Better hyperparameters: Different learning rates, batch sizes, and dropout rates were tested to find the best settings. The original paper might have used default values.

The best settings found were learning rate 1×10^{-3}, batch size 128, and dropout 0.5.

Adam optimizer: Adam optimizer was used instead of older optimizers. Adam automatically adjusts the learning rate for each parameter, which makes training faster and more stable.

Regularization: Dropout 0.5 helps prevent overfitting. Early stopping and learning rate reduction also help the model generalize better to new data.

Even though the overall accuracy is better, the model still has trouble with minority classes (S and F). The original paper's deeper model with residual connections might be better at extracting features for these hard classes. To improve further, we could try:

- Using class-weighted loss to focus more on minority classes
- Trying focal loss
- Combining multiple models (ensemble)
- Using a deeper model with residual connections like the original paper

5 Conclusion

A simple 1D CNN model was built for ECG heartbeat classification and got 94.64% accuracy, which is better than the paper's 93.4%. Using SMOTE to balance the dataset helped a lot, and the simple model works well when the data is preprocessed correctly. The model still needs improvement on minority classes (S and F), which could be done by using class-weighted loss or trying deeper architectures.