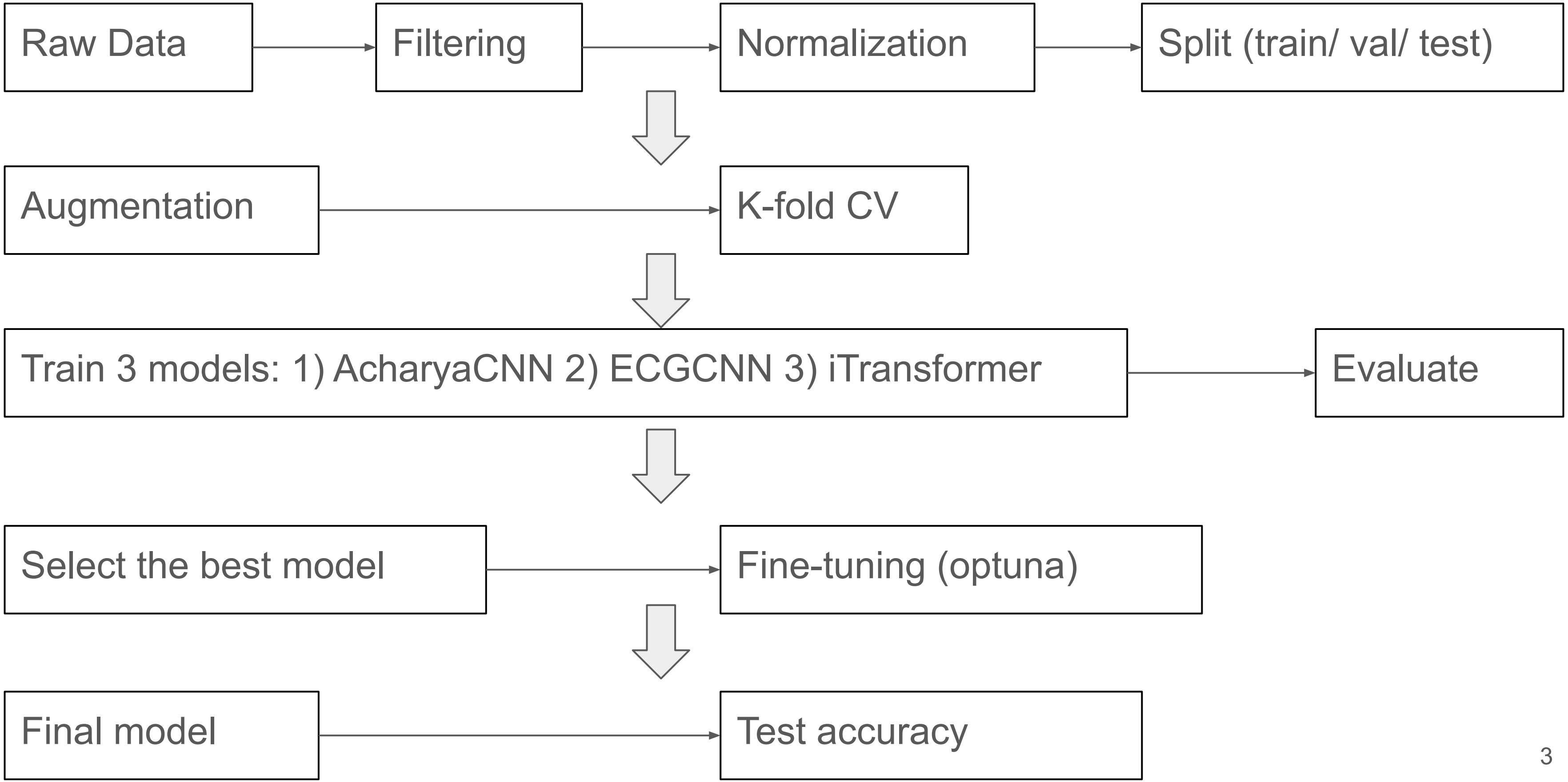
Outline of the MIT BIH Arrhythmia Classification project

- Summary of results
- Analysis Pipeline Overview
- Dataset Summary and Class Imbalance
  - Filtering and normalization
  - Data Split and Augmentation
- Deep Learning Models
  - 3 models before fine-tuning
  - The final model after fine-tuning
- Prediction Example: True vs. Predicted Labels

Summary of results

- Goal: Develop a deep learning model achieving 98.90% accuracy for classifying 5 heartbeat types, surpassing the literature benchmark of 94.03%.
    - Reference: “A deep convolutional neural network model to classify heartbeats” (Acharya et. al., 2017)
- Method: Use data augmentation and 5-fold cross-validation to address severe class imbalance.
- Outcome: An enhanced CNN model with fine-tuned hyperparameters for optimal performance.

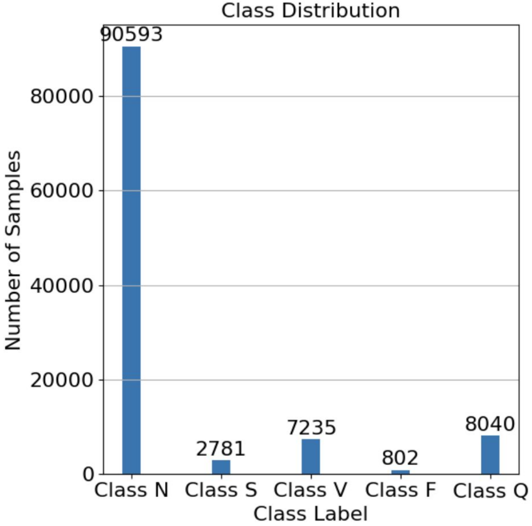
Analysis Pipeline Overview

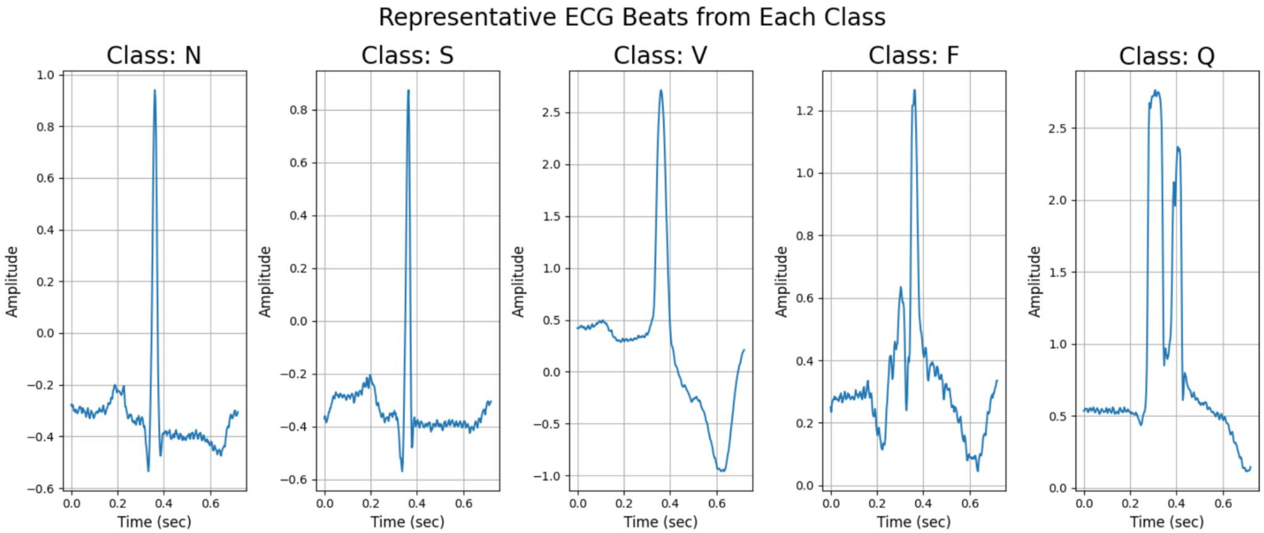


Dataset Summary and Class Imbalance

MIT-BIH Arrhythmia Database

- 109,451 heartbeats across 5 classes
- Severe imbalance: N class makes up 82.8%
- Each heartbeat: 260 samples (sampling rate: 360 Hz)
- Data source: https://www.physionet.org/content/mitdb/1.0.0/

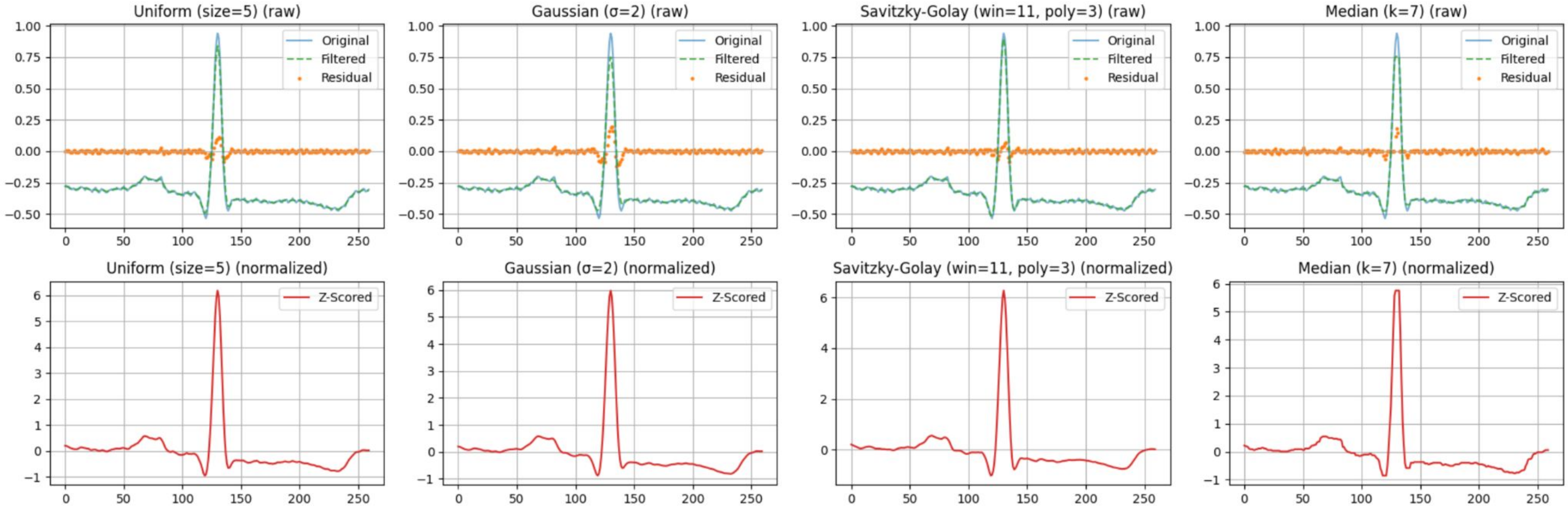




Filtering & Normalization

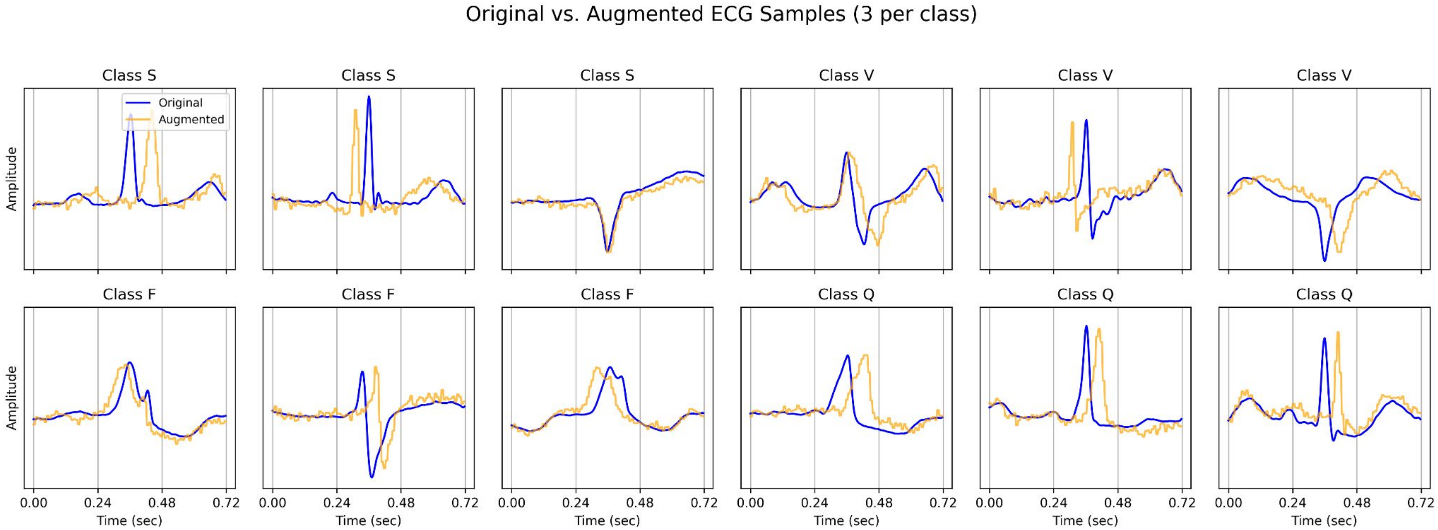
- Filtering: Tested Uniform, Gaussian, Median, and Savitzky-Golay filters.
  - Savitzky-Golay showed the best performance based on residual flatness.

- Normalization: Applied z-score normalization after filtering.



Data Split and Augmentation

- The 109,451 heartbeats are split into 70% training, 20% validation, and 10% testing.
- Data augmentation and k-fold cross-validation are applied to address severe class imbalance.
- Augmentation Techniques
  - TimeWarp: Simulates faster or slower heartbeats.
  - Drift: Mimics baseline shifts in ECG signals.
  - AddNoise: Adds slight noise to reflect real-world conditions.
  - Pooling: Smooths the signal by reducing sharp fluctuations.



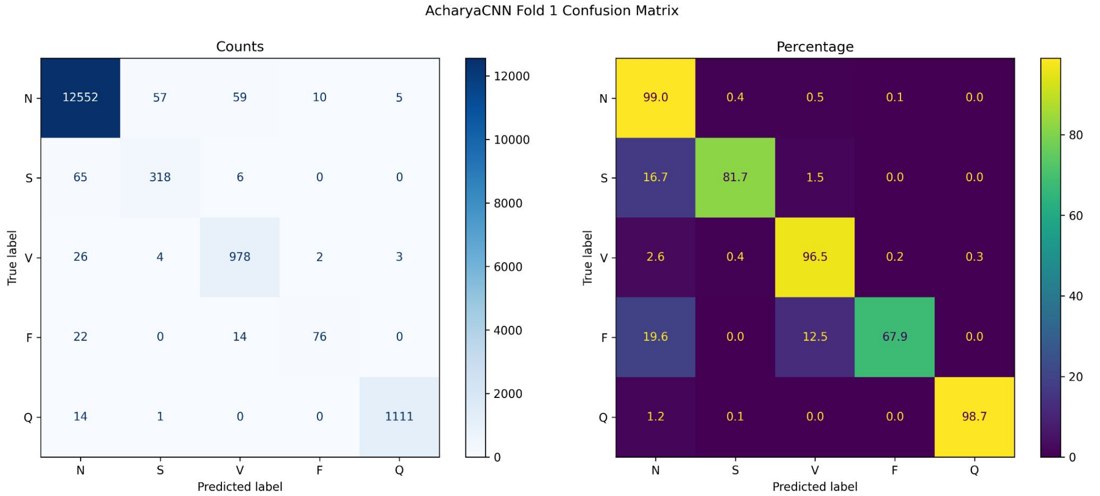
1. AcharyaCNN (baseline from literature before fine-tuning)

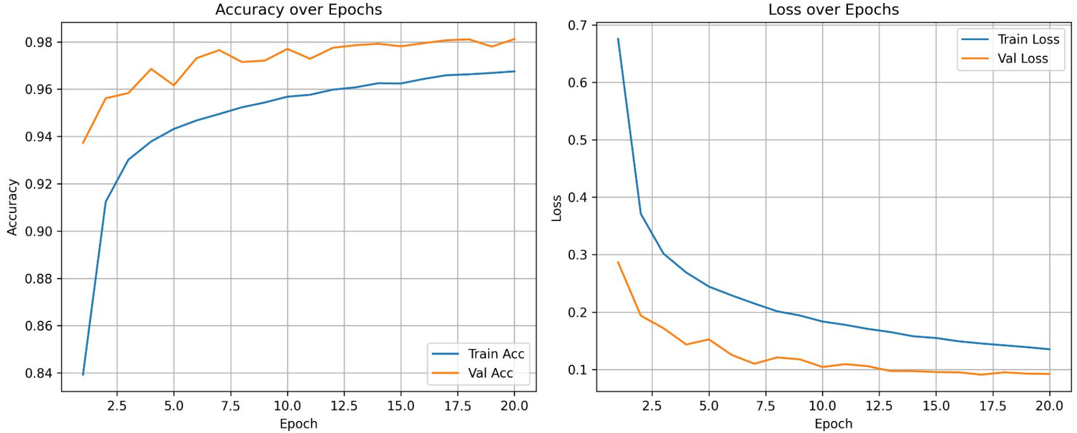
Weighted accuracy: 0.9810

- The median of 5-folds

Architecture

- Input → (1 × 260)
- Conv1D(5 filters, k=3) → ReLU → MaxPool
- Conv1D(10 filters, k=4) → ReLU → MaxPool
- Conv1D(20 filters, k=4) → ReLU → MaxPool
- Flatten
- FC(30) → ReLU → FC(20) → ReLU → FC(5) (Output)





2. iTransformer

Weighted accuracy: 0.9736

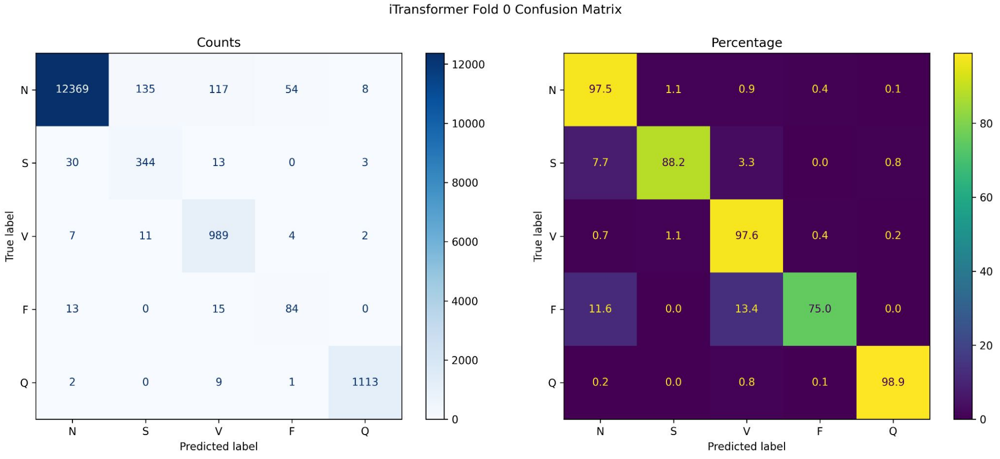
- The median of 5-folds

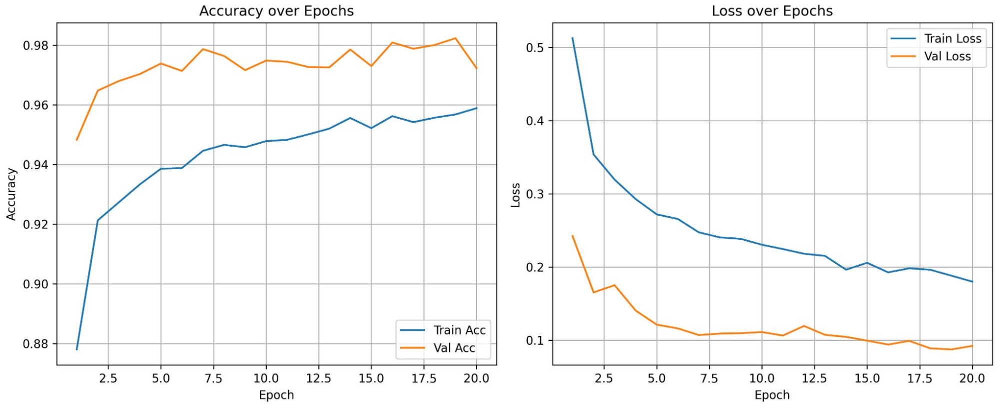
Architecture

- Input → (1 × 260)
- Linear Projection → Embedding (dim=128)
- Add CLS Token
- Add Positional Encoding
- Transformer Encoder Layers (×4, heads=4)
- CLS Token Output
- FC(5) (Output)

Reference

“A deep convolutional neural network model to classify heartbeats” (Acharya et. al., 2017)





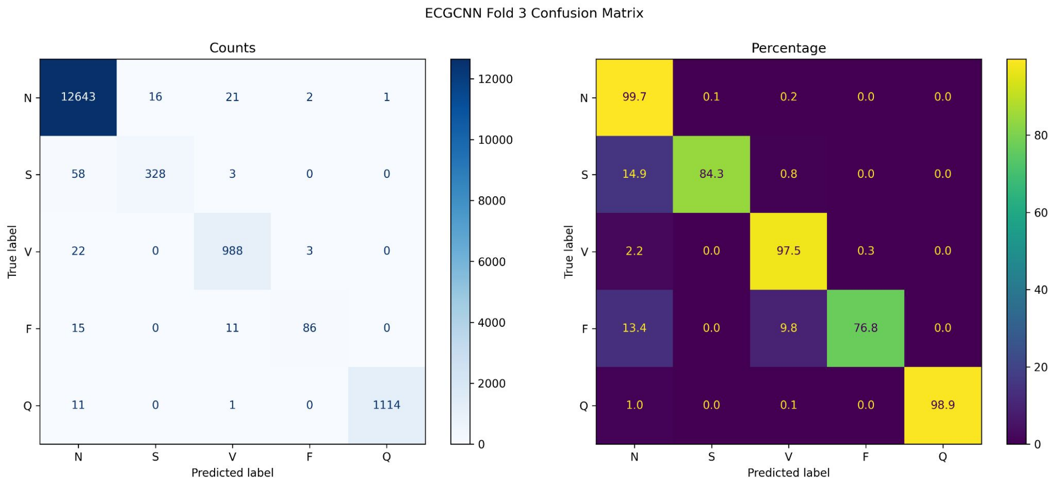
3. ECGCNN (our tunable model, before fine-tuning)

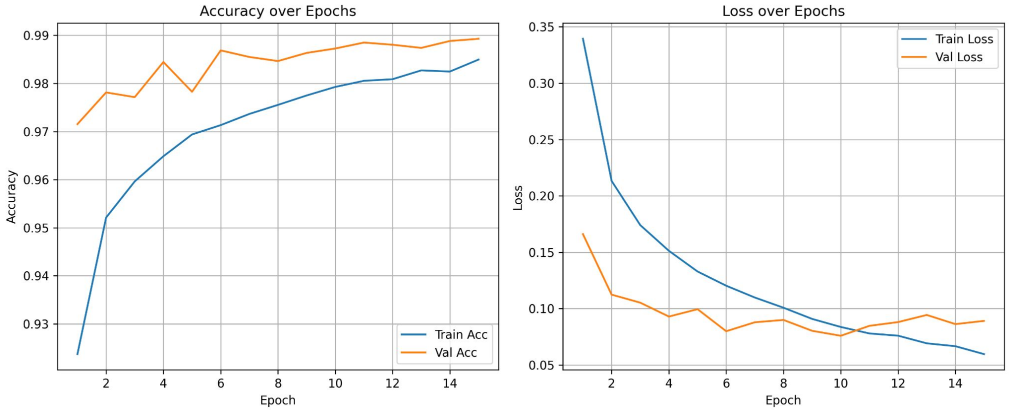
Weighted accuracy: 0.9891

- The median of 5-folds

Architecture

- Input → (1 × 260)
- Conv1D(filters1) → BatchNorm → ReLU → MaxPool
- Conv1D(filters2) → BatchNorm → ReLU → MaxPool
- (Optional) Conv1D(filters3) → BatchNorm → ReLU → MaxPool
- Flatten
- FC(fc1_size) → ReLU → Dropout
- FC(5) (Output)





4. ECGCNN (after fine-tuning)

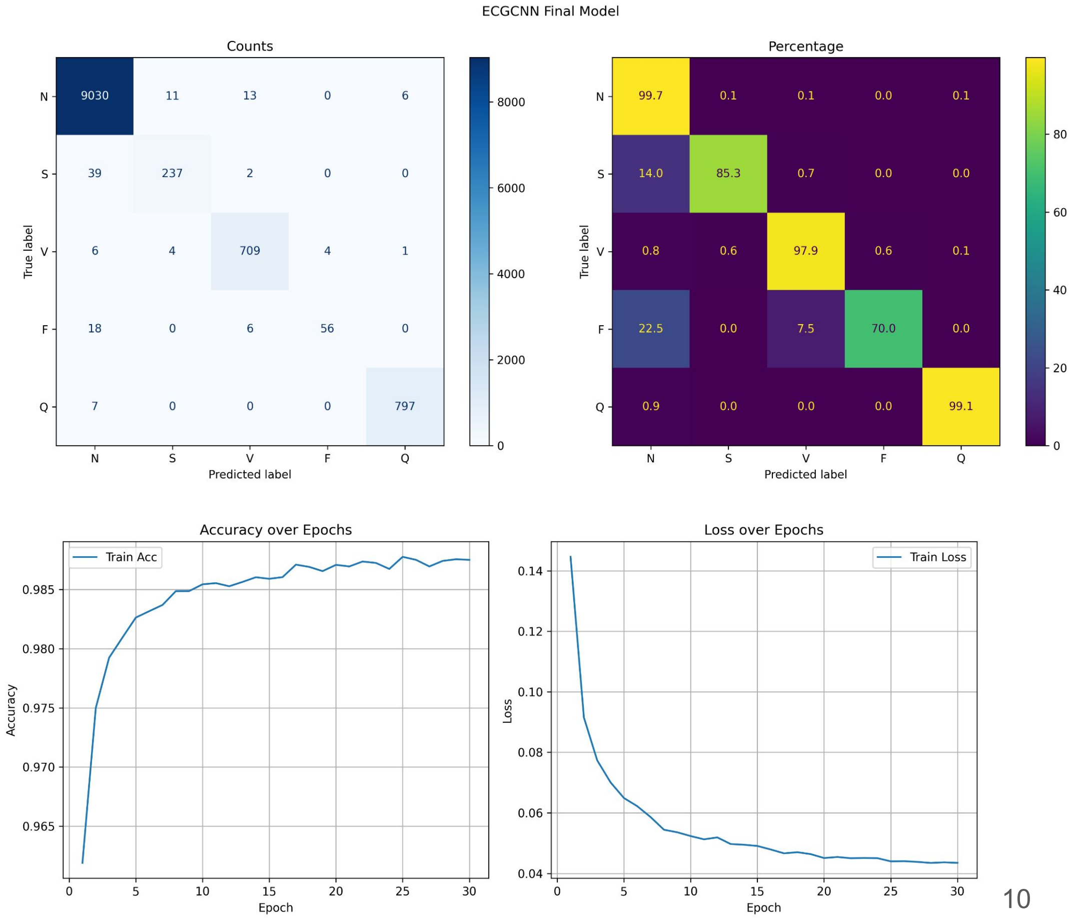
Weighted accuracy: 0.9890

Hyperparameter Optimization (Optuna)

- For each fold:
  - Run an Optuna study (20 trials) to search for best hyperparameters.
  - Retrain best trial model on combined train+val data.

- After all folds:
  - Identify best overall hyperparameters across folds.
  - Retrain final model on full training+validation set.

- Fine-tuning parameters
  - 'kernel_size': 7
  - 'dropout': 0.35248649322853515
  - 'filters1': 64
  - 'filters2': 128
  - 'fc1_size': 256
  - 'use_third_conv': False
  - 'lr': 0.0021064202284029935
  - 'weight_decay': 0.00019002863867097 03
  - 'Class_weight_alpha': 0.8138978456439353



Example: True vs. Predicted Labels

- Example heartbeat signals with predicted vs true labels.

- Generated by running: "python run_demo.py"