Report 2: Modeling Report: Heartbeat Classification on MIT-BIH and PTB-DB data

This report presents the modeling phase of the heartbeat classification project, covering the complete machine learning pipeline from baseline models to advanced deep learning (DL) approaches. The project is structured in five phases: baseline model selection, model optimization, DL implementation, optimization, and interpretation.
As this project includes two datasets (A) MIT-BIH Arrhythmia Database for arrhythmia classification and B) PTB Diagnostic ECG Database for MI detection), we structured the modeling phase stepwise. We began with the MIT-BIH Arrhythmia Database, first applying classical baseline models, then progressing to DL approaches. We followed the same progression with the PTB Diagnostic ECG Database, starting with baseline models before implementing DL methods.

Classification of the problem

- **What task does your project relate to? (fraud detection, facial recognition, sentiment analysis, etc)?**

The main goal of this project was to develop reliable machine learning models for automated arrhythmia classification and myocardial infarction (MI) detection based on ECG data. These models could help healthcare professionals diagnose cardiac arrhythmias and MI more efficiently.

- **What kind of machine learning problem is your project like? (classification, regression, clustering, etc.)**

It is a supervised classification problem as labeled training data was provided.

A) In the MIT-BIH Arrhythmia Dataset, this task involves a multiclass classification of five heartbeat types: (1) Normal, (2) Supraventricular, (3) Ventricular, (4) Fusion, and (5) Unknown. The first class corresponds to normal heartbeats, the others to different types of arrhythmias.

B) In the PTB Diagnostic ECG Dataset, the task is a binary classification problem to distinguish between normal and abnormal heartbeats. Here, abnormal corresponds to myocardial infarction (MI).

In both datasets, severe class imbalance had to be handled to achieve high performance while maintaining interpretability.

- **What is the main performance metric used to compare your models? Why this one?**

The F1-score (macro-averaged) was chosen as the main performance metric for all steps due to the strong class imbalance in both datasets. This score balances precision (proportion of correct positive predictions) and recall (proportion of actual positives correctly identified):

$$F1 = 2\,x\,\frac{Precision\,x\,Recall}{Precision + Recall}$$

A high F1-score indicates that the model achieves both high precision (few false positives) and high recall (few false negatives).

- **Did you use other qualitative or quantitative performance metrics? If yes, detail it.**

Sensitivity (Recall) - the ability to correctly identify positive cases (here: ability to correctly classify arrhythmia or detect MI):

$$Sensitivity = \frac{TP}{TP + FN}$$

Accuracy - proportion of all correct predictions: mainly used as a reference and a comparison with results reported in the literature, because alone it can be misleading with imbalanced classes.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

We also inspected confusion matrices to understand misclassification patterns.

Model choice and optimization

- **What algorithms have you tried? Describe which one(s) you selected and why?**

We experimented with a broad range of algorithms to understand both classical and more advanced approaches to arrhythmia classification and MI detection:

Baseline Models:

- Logistic Regression (*LR*) - fast and simple, useful for workflow testing
- Linear Discriminant (*LDA*) - historically popular for arrhythmia classification [1]
- K nearest neighbours (*KNN*) - non-parametric, also used in the past for arrhythmia classification [1]
- Decision Tree (*DT*) - interpretable, useful for quick insights into feature importance, and also used in the past for arrhythmia classification [1]
- Random Forest (*RF*) - an ensemble extension of DT, usually a strong baseline for structured tabular data
- Support Vector Machine (*SVM*) - popular for arrhythmia classification [1]
- Extreme Gradient Boosting (*XGB*) - a high-performance ensemble method that tends to generalize well and handle imbalance better.
- Artificial Neural Network (*ANN*) - a simple feed-forward architecture to test basic DL capability, popular for arrhythmia classification [1]

All baseline models were trained on the MIT-BIH training data using

1. Raw training data
2. RandomOversampler training data: samples in underrepresented classes are randomly duplicated to balance class distribution
3. SMOTE training data: new synthetic samples are generated from existing samples in underrepresented classes to balance class distribution
4. ADASYN: SMOTE extension, focus on the generation of synthetic samples that are harder to learn
5. SMOTETomek: a combination of SMOTE oversampling and Tomek links removal
6. SMOTEENN: a combination of SMOTE oversampling and Edited Nearest Neighbor cleaning

All baseline models were trained on the PTB-DB training data using

1. SMOTE training data

DL Models:

- Dense Neural Networks (DNN) [6]
- Convolutional Neural Networks (CNN) [6] - the most modern approach, leveraging spatial and temporal structure in ECG signals [4]. These models can learn features automatically from raw signals, reducing the need for handcrafted feature extraction
    - Rebuilt the CNN architecture used in [2, 4]
- Long Short-Term Memory (LSTM) [2, 6]

All DL models were trained on the MIT-BIH training data using

1. SMOTE training data

All DL models were trained on the PTB-DB training data using

1. SMOTE training data

- **Did you use parameter optimization techniques such as Grid Search and Cross Validation?**

Parameter optimization and model selection were conducted in several structured phases. All cross-validations have been undertaken with StratifiedKFold (5-fold). For reproducibility, each phase and sub-phase corresponds to a Jupyter notebook.

Workflow Overview: baseline models

A) MIT-BIH Arrhythmia Dataset (MIT dataset)

1. Train/test split: 80%, 20% -> as defined at the beginning of the project to ensure result reproducibility, no duplicates or missing values present. Data is already scaled, so no additional scaling was applied.
2. Hyperparameter tuning using RandomizedSearch with cross-validation for the mentioned baseline models and oversampling techniques.
    a. Randomized Search:
        i. Goal: for each model, test randomly chosen different hyperparameter combinations to find a model achieving the best performance on test data with the most appropriate sampling technique
        ii. Reason for choosing RandomizedSearch: Test random numbers of hyperparameter combinations and not all possible combinations to reduce runtime. This is enough to get a first impression of the performances of the different models.
    b. Cross-validation (here: 5 splits, see Figure 1):
        i. Goal: prevent overfitting
        ii. During the model training, the training set is split into 5 subsets (use of StratifiedKFold to ensure the same class distribution in every subset)
        iii. 4 subsets used for training, 1 subset used for validation (model performance is evaluated during the training procedure to optimize performance)
        iv. If an oversampling method is used, it is applied only to the training subsets in the cross-validation, not to the validation subset, to prevent data leakage (realization using a pipeline)
3. Hyperparameter tuning using GridSearch:
    a. Goal: optimize the performance of the best model found in 2b -> test of every hyperparameter combination
4. Model performance evaluation

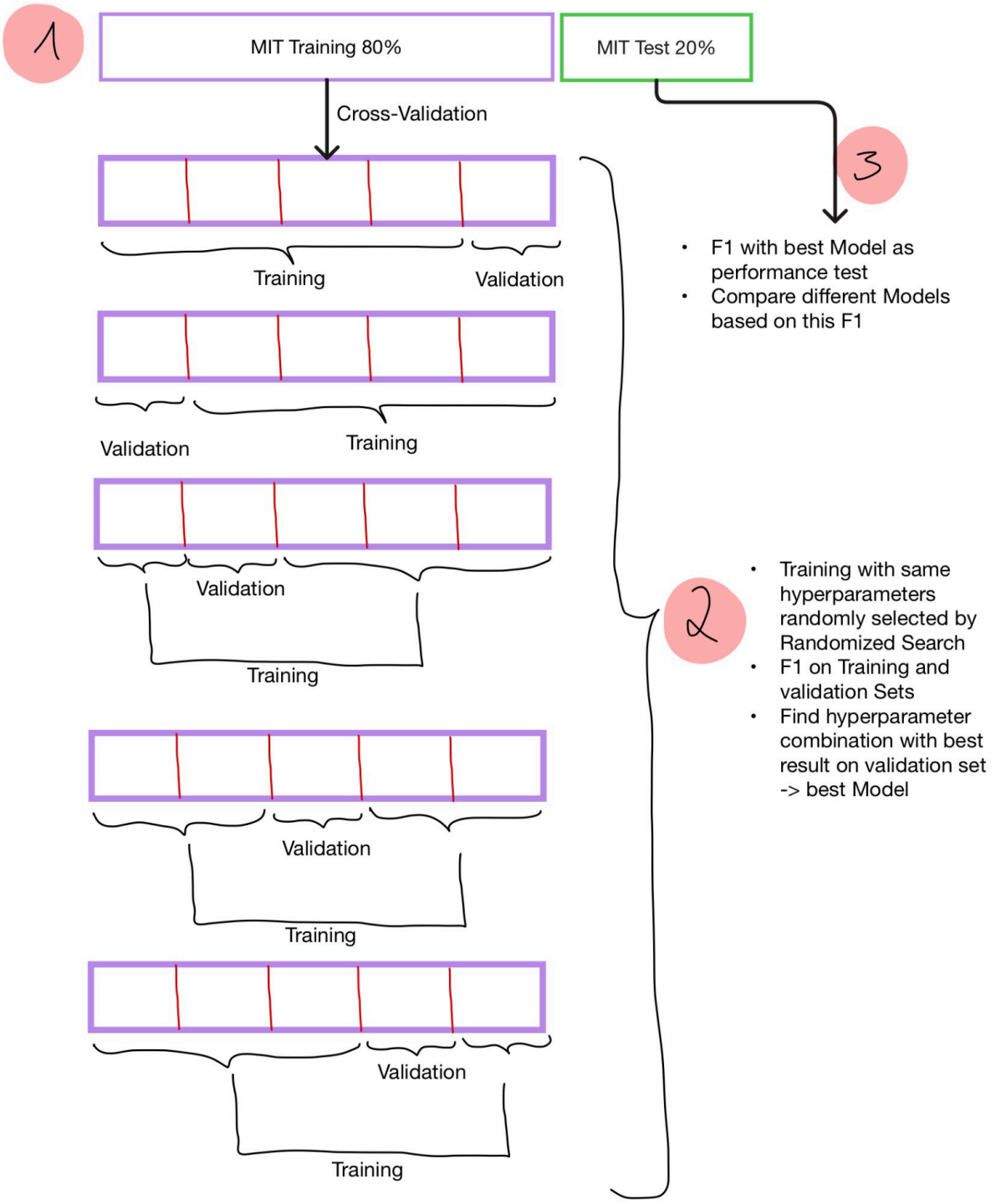


Figure 1 - Training with cross-validation workflow applied to all baseline models.

Workflow Overview: baseline models

B) PTB Diagnostic ECG Database (PTB dataset)

1. Train/test split: 80%, 20% -> after removing duplicates
2. Model Selection using LazyClassifier
    a. Goal: find baseline model with best performance on test data for default parameters
3. Hyperparameter tuning using GridSearch with cross-validation
    a. Goal: optimize model performance found in 2a (test every hyperparameter combination)
4. Model performance evaluation

A2 I Hyperparameter tuning for baseline models using RandomizedSearch without sampling

Establish baseline models via RandomizedSearch without any feature engineering or resampling of the training data. These baseline models will be used to compare the results against the DL models.

Models included: LR, LDA, KNN, DT, RF, SVM, XGBoost, and a simple ANN.

Results: XGBoost (F1=0.9098), SVM (F1=0.9066), KNN(F1=0.8873) (see Table 1)

The difference between the F1-Scores of the training and validation sets within the cross-validation was relatively small for XGB (0.09) and SVM (0.07), while higher for KNN (0.11). The smallest differences achieved were for LDA (0.004), LR (0.005), and ANN (0.04), hinting at slight overfitting in the 3 top models.

A2 II Hyperparameter tuning for baseline models using RandomizedSearch testing sampling methods

Performed systematic hyperparameter tuning via RandomizedSearch on all baseline models. The goal was to identify the best-performing sampling method and model. Sampling methods were tested only on baseline models to determine the “best” sampling method, which will also be applied to the training data of DL model(s).

Sampling methods tested: (1) RandomOverSampler, (2) SMOTE, (3) ADASYN, (4) SMOTETomek, and (5) SMOTEENN.

Results: Using RandomOversampler gave us the best performance on the test data. However, we decided it is not suitable for our use case, since the strong class imbalance would require generating a very large number of random samples. In addition, because we used a RandomizedSearch, we could not be certain at the point that we had found the true optimal combination of parameters and sampling methods.
Comparing the results of all models and sampling strategies, SMOTE turned out to be the most consistent and balanced approach. Therefore, we decided to discard RandomOversampler and use SMOTE for all further modeling.

Best model performance with SMOTE: **XGBoost (F1=0.91)**, ANN (F1=0.89), SVM (F1=0.87) (see Table 2).

	CV						F1 per Class - Test				
Model	**Avg F1 Train**	**Avg F1 Val**	**Diff F1 Train - F1 Val**	**Avg balanced accuracy Val**	**F1 Test**	**Accuracy Test**	**1**	**2**	**3**	**4**	**5**
XGB	1.00	0.91	0.09	0.86	0.91	0.98	0.99	0.82	0.95	0.80	0.98
SVM	0.98	0.91	0.07	0.88	0.91	0.98	0.99	0.82	0.95	0.79	0.99
KNN	1.00	0.89	0.11	0.85	0.89	0.98	0.99	0.78	0.94	0.75	0.98
ANN	0.93	0.89	0.04	0.86	0.88	0.98	0.99	0.75	0.93	0.75	0.97
RF	1.00	0.89	0.11	0.84	0.88	0.98	0.99	0.77	0.94	0.73	0.97
DT	0.88	0.83	0.05	0.79	0.83	0.96	0.98	0.70	0.87	0.66	0.94
LR	0.67	0.66	0.005	0.59	0.67	0.91	0.95	0.56	0.44	0.46	0.92
LDA	0.64	0.63	0.004	0.66	0.63	0.89	0.94	0.54	0.53	0.24	0.91

Table 1 - Baseline model performance with RandomizedSearch (no sampling of training data) for the best model per classifier. F1-Scores achieved by evaluating the hold-out test set. Values within the CV section are the averaged values over all CV-folds. XGB = XGBoost. SVM = Support Vector Machine. KNN = K-Nearest Neighbour. ANN = Artificial Neural Network. RF = Random Forest. DT = Decision Tree. LR = Logistic Regression. LDA = Linear Discriminant Analysis. Values have been rounded to two decimal places for readability.

		CV						F1 per Class - Test				
Model	Sampler	Avg F1 Train	Avg F1 Val	Diff F1 Train - F1 Val	Avg balanced accuracy Val	F1 Test	Accuracy Test	1	2	3	4	5
XGB	ROS	1,0000	0.9201	0.0799	0.8921	0.9236	0.9849	0.9922	0.8332	0.9601	0.8471	0.9856
XGB	ADASYN	1,0000	0.9142	0.0858	0.9115	0.9118	0.9823	0.9908	0.8289	0.9523	0.8036	0.9835
XGB	SMOTETomek	0.9993	0.9107	0.0886	0.9109	0.9112	0.9818	0.9906	0.8198	0.9507	0.8133	0.9816
XGB	SMOTE	0.9994	0.9147	0.0847	0.9117	0.9093	0.9816	0.9904	0.8085	0.9513	0.8121	0.9844
XGB	SMOTEENN	0.9601	0.8933	0.0667	0.9168	0.8924	0.9765	0.9873	0.7604	0.9476	0.7839	0.9829
ANN	SMOTE	0.9714	0.8818	0.0896	0.8942	0.8846	0.9773	0.9885	0.7843	0.9374	0.7309	0.9818
RF	SMOTEENN	0.9606	0.8869	0.0736	0.9023	0.8839	0.9752	0.9863	0.7749	0.9426	0.738	0.9775
KNN	ROS	1,0000	0.8827	0.1173	0.8673	0.8835	0.9762	0.9876	0.7727	0.9319	0.7492	0.9759
ANN	SMOTETomek	0.9714	0.88	0.0914	0.8951	0.8818	0.9767	0.9881	0.7667	0.9425	0.7331	0.9787
RF	ADASYN	0.9745	0.8813	0.0931	0.9066	0.8816	0.9727	0.9852	0.7515	0.9314	0.7656	0.9744
SVM	SMOTE	0.9121	0.8648	0.0473	0.8866	0.8761	0.9711	0.8612	0.7401	0.8991	0.7211	0.9787
RF	ROS	0.9706	0.8799	0.0907	0.8652	0.8752	0.9761	0.9866	0.7881	0.9458	0.6795	0.975
RF	SMOTE	0.9607	0.8789	0.0818	0.8963	0.8743	0.9763	0.987	0.789	0.9526	0.6633	0.9796
RF	SMOTETomek	0.9532	0.8734	0.0797	0.8994	0.8697	0.9751	0.9864	0.7873	0.9473	0.6488	0.9787
KNN	SMOTETomek	1,0000	0.8629	0.137	0.8963	0.8645	0.9724	0.9858	0.7429	0.9304	0.6845	0.9788
ANN	ADASYN	0.9696	0.8718	0.0979	0.8858	0.8632	0.9699	0.9842	0.7464	0.9121	0.6959	0.9774
SVM	SMOTETomek	0.9015	0.8545	0,047	0.8865	0.8623	0.9689	0.9799	0.7296	0.8901	0.7169	0.9779
SVM	ADASYN	0.8963	0.8862	0,010	0.8955	0.8605	0.9678	0.9763	0.7275	0.8896	0.6993	0.9778

ANN	ROS	0.9623	0.8861	0.0762	0.9097	0.8603	0.9667	0.9819	0.6833	0.9277	0.7332	0.9757
ANN	SMOTEENN	0.9312	0.8567	0.0745	0.8948	0.8542	0.966	0.9814	0.7003	0.927	0.6943	0.9682
SVM	SMOTEENN	0.88	0.83	0,05	0.8911	0.8513	0.9632	0.98	0.7	0.89	0.6965	0.9788
KNN	SMOTEENN	0.9172	0.8402	0.077	0.9035	0.844	0.9648	0.9813	0.6928	0.9183	0.6549	0.9728
KNN	ADASYN	0.9314	0.8332	0.0981	0.9066	0.8316	0.9608	0.9789	0.6706	0.9116	0.6256	0.9713
KNN	SMOTE	1,0000	0.8166	0.1834	0.9059	0.8205	0.9566	0.9767	0.6339	0.9077	0.6154	0.9688
DT	ROS	0.9943	0.8194	0.1749	0.8213	0.8184	0.9579	0.9768	0.6981	0.8761	0.5959	0.9452
DT	SMOTEENN	0.9364	0.7812	0.1552	0.8693	0.7825	0.9419	0.9674	0.5736	0.8756	0.5507	0.9453
DT	SMOTE	0.9796	0.7874	0.1922	0.8539	0.7822	0.9444	0.9695	0.6072	0.8567	0.534	0.9439
DT	SMOTETomek	0.9228	0.7741	0.1487	0.8608	0.7736	0.9377	0.9652	0.5793	0.8437	0.5404	0.9392
DT	ADASYN	0.9892	0.7704	0.2188	0.8474	0.7629	0.9375	0.9651	0.5853	0.8469	0.477	0.9402
LDA	ROS	0.5085	0.5047	0.0038	0.7601	0.5058	0.7066	0.8062	0.2346	0.4362	0.1487	0.9034
LDA	SMOTE	0.509	0.5047	0.0043	0.762	0.5038	0.7022	0.8024	0.2332	0.4307	0.1483	0.9042
LDA	SMOTETomek	0.509	0.5048	0.0043	0.762	0.5038	0.7023	0.8025	0.2333	0.4307	0.1483	0.9042
LDA	SMOTEENN	0.5053	0.5008	0.0044	0.7611	0.5003	0.6953	0.7966	0.2309	0.4255	0.1462	0.9025
LR	ROS	0.4891	0.4843	0.0048	0.7753	0.479	0.6754	0.7786	0.2363	0.4203	0.1506	0.8095
LR	SMOTETomek	0.4899	0.4838	0.0061	0.7732	0.4786	0.6734	0.7761	0.2404	0.4142	0.1545	0.8079
LR	SMOTE	0.49	0.4837	0.0063	0.773	0.4774	0.6719	0.7747	0.2393	0.4118	0.1556	0.8058
LR	SMOTEENN	0.4838	0.4779	0.0059	0.7731	0.4717	0.6617	0.7657	0.2338	0.4073	0.1541	0.7977
SVM	ROS	0.92	0.85	0.07	0.86	0.4521	0.65	0.9157	0.3096	0.2766	0.4344	0.3196
LR	ADASYN	0.3457	0.3398	0.0059	0.6449	0.3323	0.4586	0.5746	0.1246	0.2829	0.1031	0.5763

LDA	ADASYN	0.3441	0.3384	0.0058	0.6512	0.3282	0.4373	0.5469	0.1151	0.2715	0.1129	0.5945

Table 2 - RandomizedSearch on all models with different sampling methods. Sorted by F1 macro scores descending. F1-Scores achieved by evaluating the hold-out test set. ROS = RandomOverSampler. SVM = Support Vector Machine. KNN = K-Nearest Neighbour. RF = Random Forest. ANN = Artificial Neural Network. DT = Decision Tree. LR = Logistic Regression. LD = Linear Discriminant Analysis.

Sampler	Extreme Values Removed	Model	Accuracy	F1-Macro	Best CV-SCore	F1 Class 1	F1 Class 2	F1 Class 3	F1 Class 4	F1 Class 5
SMOTE	FALSE	XGBoost	0.98	0.91	0.91	0.99	0.85	0.96	0.82	0.99
-	FALSE	XGBoost	0.98	0.91	0.90	0.99	0.82	0.95	0.8	0.99
SMOTE	TRUE	XGBoost	0.98	0.90	0.91	0.99	0.81	0.95	0.79	0.99
-	TRUE	XGBoost	0.98	0.90	0.91	0.99	0.8	0.95	0.79	0.99
-	FALSE	ANN	0.98	0.90	0.89	0.99	0.79	0.94	0.78	0.98
-	TRUE	ANN	0.97	0.88	0.89	0.99	0.74	0.92	0.78	0.98
SMOTE	FALSE	ANN	0.98	0.88	88.10	0.99	0.79	0.94	0.69	0.98
SMOTE	TRUE	ANN	0.97	0.86	0.89	0.98	0.7	0.91	0.72	0.97

Table 3 - GridSearch Results with combined sampling (SMOTE) and R distance extreme value removal for class 0. Values have been rounded to two decimal places for readability.

Sampler	Extreme Values Removed	Model	Accuracy	F1 Macro	F1 Class 1	F1 Class 2	F1 Class 3	F1 Class 4	F1 Class 5
SMOTE	FALSE	XGBoost	9.837	9.108	0.99	0.86	0.95	0.76	0.99
SMOTE	FALSE	ANN	9.773	8.854	0.99	0.8	0.93	0.73	0.98
SMOTE	TRUE	XGBoost	9.812	9.004	0.99	0.83	0.95	0.75	0.98
SMOTE	TRUE	ANN	9.746	8.718	0.99	0.78	0.94	0.67	0.99
SMOTE	FALSE	SVM	9.797	8.941	0.99	0.8	0.95	0.74	0.99
SMOTE	TRUE	SVM	9.784	8.902	0.99	0.78	0.95	0.74	0.98

Table 4 - Test of the 3 best models with preprocessing (denoising, baseline wander removal) and R distance extreme value removal for class 0. ANN = Artificial Neural Network. SVM = Support Vector Machine. Values have been rounded to two decimal places for readability.

A3 I Signal Denoising & Baseline Wander Removal

We implemented a signal preprocessing based on the literature [4] to test whether denoising improves performance for the best models from A3. However, this step did not increase the average F1-score, possibly because already cleaned, filtered inputs from the Kaggle dataset (Table 4) were used.

A3 II Hyperparameter tuning for best baseline models found in A2 II using GridSearch

Evaluated top models (XGBoost, ANN, SVM) with combinations of sampling (with/without) and R distance extreme value removal for class 0 (with/without, see Rendering 1) and a leak-free sampling Pipeline (see Table 3 for results).

Best model found: **XGBoost** trained on SMOTE-generated data, without R-distance extreme value removal for class 0 (**F1=0.9096**).

Second-best model: ANN trained with no resampled training data and without R-distance extreme value removal for class 0 (F1=0.896).

Parameters tested:

N-Estimators:	150, 200, 250, 350, 500
Max-Depth:	8,9
Learning-Rate:	0.1, 0.2, 0.3
Subsample:	0.6, 0.7, 0.8
Colsample_Bytree:	0.9
reg_alpha:	0.1, 0.2
reg_lambda:	0.0, 0.05
min_child_weight:	5
gamma:	0.0, 0.05

Best parameters found:

SAMPLER:	SMOTE
N-Estimators:	500
Max-Depth:	9
Learning-Rate:	0.2
Subsample:	0.7
Colsample_Bytree:	0.9
reg_alpha:	0.2
reg_lambda:	0.05
min_child_weight:	5
gamma:	0.0

Therefore, the XGBoost model was trained on SMOTE-generated data without R-distance extreme value removal for class 0 and the best hyperparameter combination found. The model is evaluated in detail in A4.

A4 Model evaluation - Best model on MIT test data

A **XGBoost** classifier was trained with the SMOTE MIT training dataset and tested against unseen test data. The same dataset was later used for the DL evaluation to generate comparable results. Figure 3 shows the evolution of the training and validation loss during training. It can be seen that both converged. The gap between the training and validation loss indicates slight overfitting, but performance on unseen test data remained good, as shown in the classification report and confusion matrix (Figure 2). The largest misclassification was for true class 1, which was classified as class 0.

Classification report:

Results:

Accuracy:	0.9823
F1-Macro:	0.9096
Precision-Macro:	0.9223
Recall-Macro:	0.8981
Per-Class Precision:	
Class 0:	0.9880
Class 1:	0.8727
Class 2:	0.9627
Class 3:	0.8012
Class 4:	0.9868
Per-Class Recalls:	
Class 0:	0.9935
Class 1:	0.7770
Class 2:	0.9441
Class 3:	0.7963
Class 4:	0.9795
Per-Class F1-Scores:	
Class 0:	0.9908
Class 1:	0.8221
Class 2:	0.9533
Class 3:	0.7988
Class 4:	0.9831

Confusion matrix:

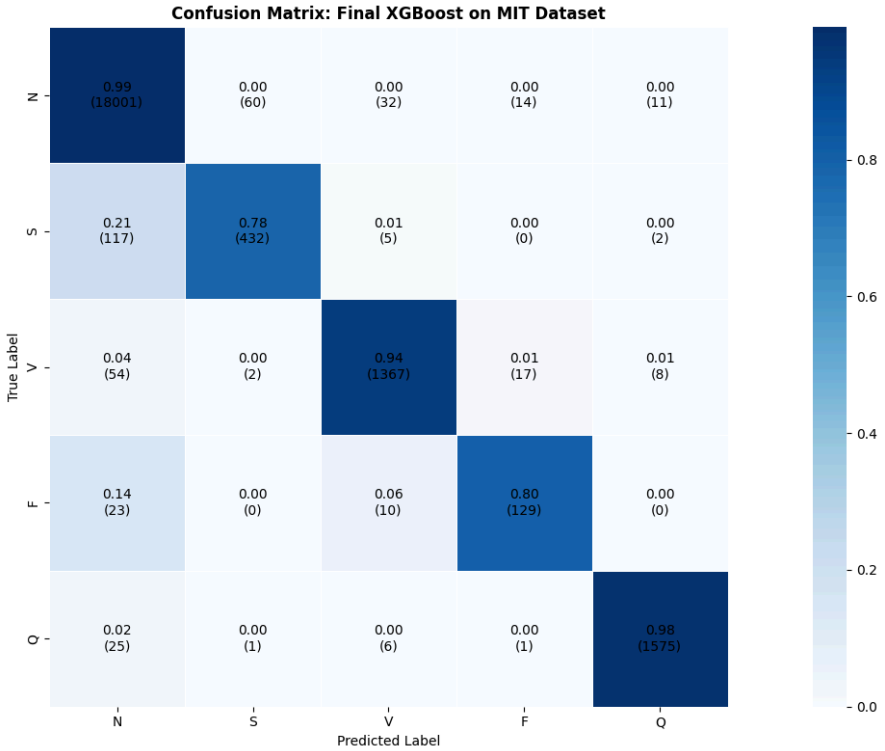


Figure 2 - Normalized Confusion Matrix of the final tuned XGBoost model on the MIT-BIH dataset. N = Norma (0), S = Supraventricular (1), V = Ventricular (2), F = Fusion (3), and Q = Unknown (4).

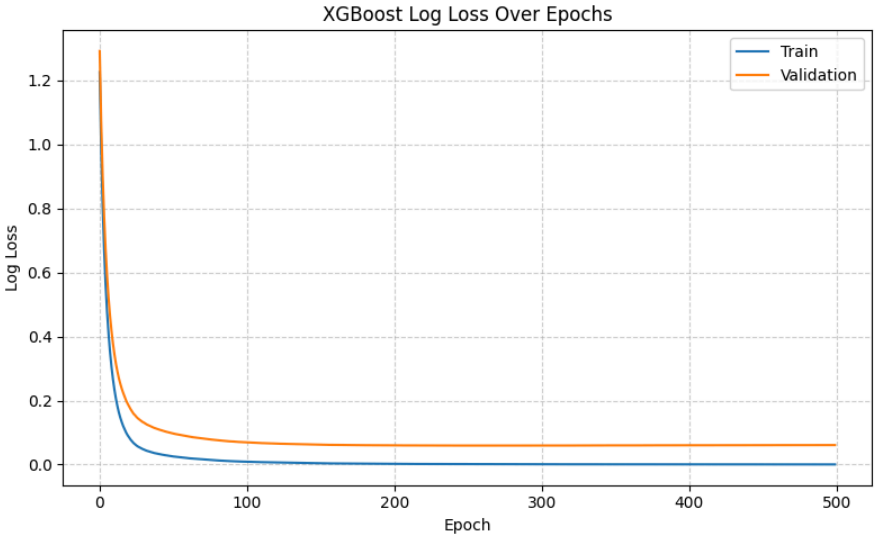


Figure 3 - Evolution of training and validation loss across boosting iterations. The plot shows multiclass log loss for the training and validation sets during XGBoost training. The steady decrease in training loss indicates the model improvement on the training data, while the validation curve reflects how well the model generalizes to unseen data. The divergence between the two curves suggests slight overfitting. Also, this graph suggests early stopping could be reasonable for XGBoost after around 200 epochs.

A4 Model Evaluation - Test on PTB data

As a test, the XGBoost model trained with MIT data (multiclass) was also applied to PTB data to explore transfer potential. For this, the prediction results were combined for classes 1-4 to generate an 'abnormal' class. As shown in the classification report, the results of the F1 score achieved for class 1 are very low. The confusion matrix (Figure 4) also shows significant misclassifications of true class 1, classified as class 0 by the model. Therefore it was decided to train a classifier on the PTB dataset.

Classification Report:

Results:

Accuracy:	0.35
F1-Macro:	0.33
Precision-Macro:	0.59
Recall-Macro:	0.54
Per-Class Precision:	
Class 0:	0.30
Class 1:	0.89
Per-Class Recall:	
Class 0:	0.97
Class 1:	0.12
Per-Class F1-Scores:	
Class 0:	0.46
Class 1:	0.21

Confusion matrix:

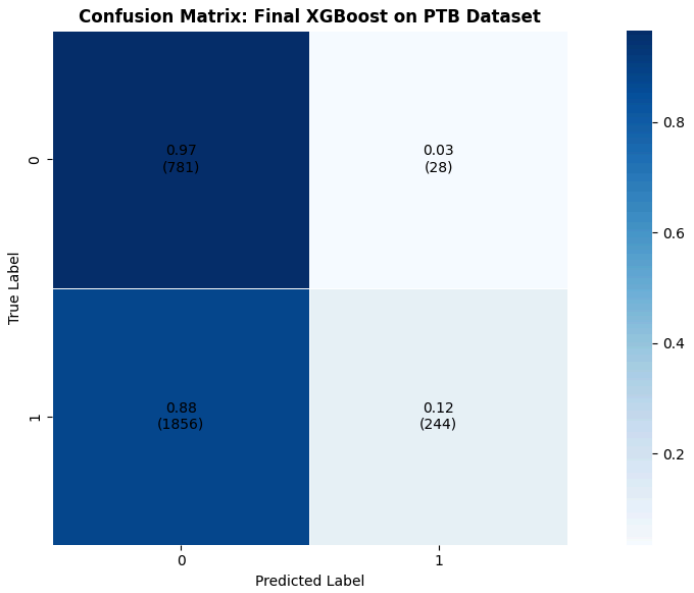


Figure 4 - Normalized Confusion Matrix of final tuned XGBoost (SMOTE), fitted on the MIT dataset, applied on the test data of the PTB dataset. 0 = Normal, 1 = Abnormal.

B2 Model selection using Lazy Classifier on PTB Dataset

For the baseline model training procedure based on the PTB dataset, LazyClassifier was used to select the best-performing model. LazyClassifier tests all models with default parameters. Here, **XGBoost** was most suitable based on F1 score and accuracy, as shown in Figure 4. This model was used for further investigations.

Model	Accuracy	Balanced Accuracy	ROC AUC	F1 Score	Time Taken
XGBClassifier	0.9763	0.9718	0.9718	0.9763	7.6307
LGBMClassifier	0.9735	0.9703	0.9703	0.9736	4.1613
ExtraTreesClassifier	0.9766	0.9694	0.9694	0.9766	4.4082
RandomForestClassifier	0.9701	0.9652	0.9652	0.9702	22.7159
BaggingClassifier	0.9433	0.9459	0.9459	0.9441	38.2723
SVC	0.9123	0.9199	0.9199	0.9145	19.2112
KNeighborsClassifier	0.8965	0.9139	0.9139	0.8999	1.5818
DecisionTreeClassifier	0.9082	0.8939	0.8939	0.9089	5.1366
LabelPropagation	0.8663	0.8903	0.8903	0.8716	8.0470
LabelSpreading	0.8663	0.8903	0.8903	0.8716	13.4411

Figure 5 - Top 10 models for training with PTB data (result of LazyClassifier)

B3 Hyperparameter tuning using GridSearch - XGBoost trained on PTB dataset (SMOTE training data)

The best hyperparameter combination found for **XGBoost** achieved an **F1 score of 0.9730**. As shown in the confusion matrix (Figure 6), the largest misclassification occurred for true class 0, which the model classified as class 1. In Figure 7, the evolution of the training and validation loss during the training is shown. Both converged and slight overfitting is observed, indicated by the gap between the training and validation loss. The achieved performance on unseen test data is still good.

Parameters tested:

N-Estimators:	150, 200, 250, 350, 500
Max-Depth:	8,9
Learning-Rate:	0.1, 0.2, 0.3
Subsample:	0.6, 0.7, 0.8
Colsample_Bytree:	0.9
reg_alpha:	0.1, 0.2
reg_lambda:	0.0, 0.05
min_child_weight:	5
gamma:	0.0, 0.05

Best parameters found:

Sampler:	SMOTE
N-Estimators:	500
Max-Depth:	9
Learning-Rate:	0.2
Subsample:	0.7
Colsample_Bytree:	0.9
reg_alpha:	0.2
reg_lambda:	0.05
min_child_weight:	5
gamma:	0.0

Classification report:

Results:

Accuracy:	0.9739
F1-Macro:	0.9730
Precision-Macro:	0.9739
Recall-Macro:	0.9721
Per-Class Precision:	
Class 0:	0.9639
Class 1:	0.9838
Per-Class Recall:	
Class 0:	0.9580
Class 1:	0.9862
Per-Class F1-Scores:	
Class 0:	0.9591
Class 1:	0.9843

Confusion matrix:

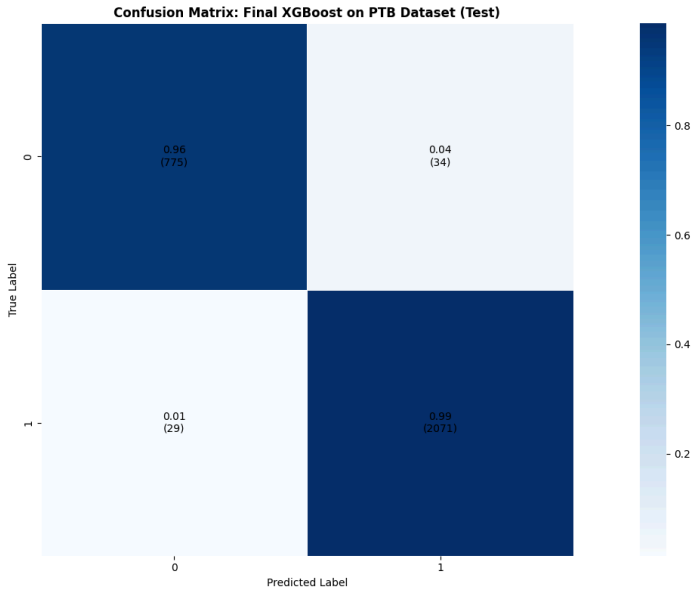


Figure 6 - Normalized Confusion Matrix of final tuned XGBoost (SMOTE), fitted on the PTB dataset, applied on the test data of the PTB-DB dataset. 0 = Normal, 1 = Abnormal.

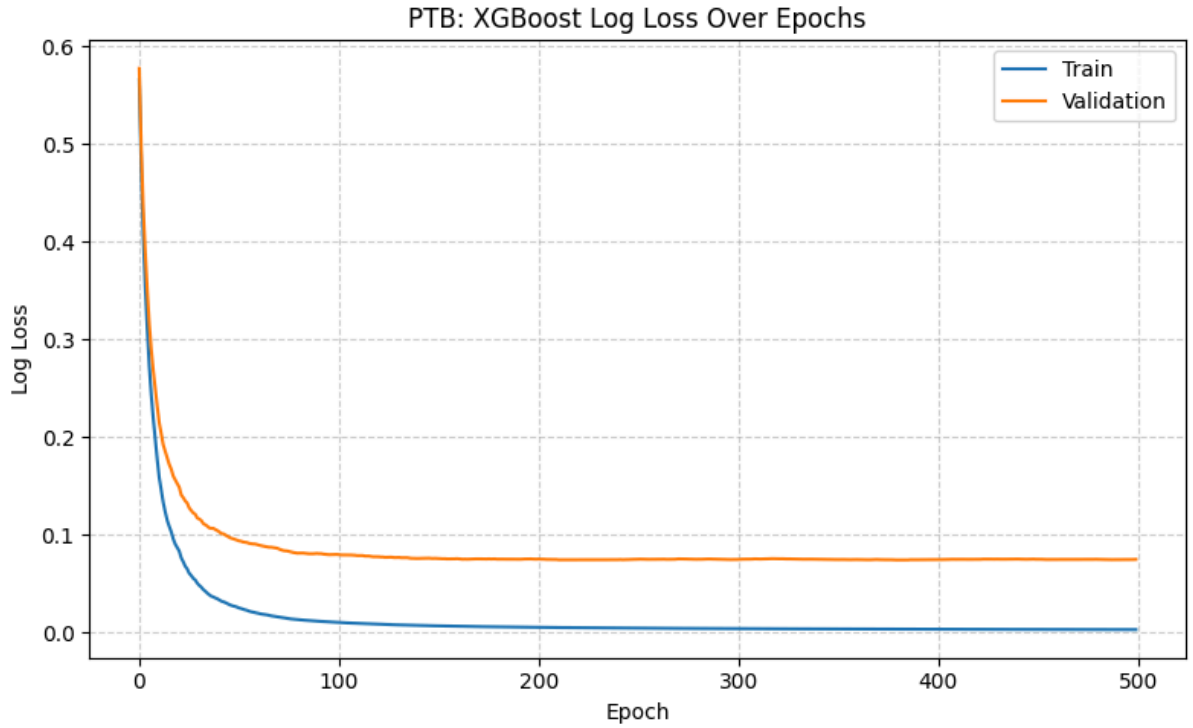


Figure 7 - Evolution of training and validation loss across boosting iterations. The plot shows binary log loss for the training and validation sets during XGBoost training on PTB test data. The steady decrease in training loss indicates the model improvement on the training data, while the validation curve reflects how well the model generalizes to unseen data. The divergence between the two curves suggests slight overfitting.

- **Have you tested advanced models? Bagging, Boosting, Deep Learning… Why?**

We implemented DL models because they have shown superior performance for arrhythmia classification in the literature [2, 4, 6]. It was demonstrated that DL models can automatically learn from raw or minimally processed ECG signals, potentially capturing complex temporal patterns that traditional methods might miss.

Furthermore, we applied transfer learning by pretraining a model on the MIT dataset and fine-tuning it on the PTB dataset. This approach was motivated by [4], which demonstrated its suitability for MI detection. Here, transfer learning offers two key advantages: It uses learned representations from the larger MIT dataset to improve performance on the smaller PTB dataset. In addition, it significantly reduces training time and computational requirements.

For all training procedures in this section, SMOTE-processed data were used, as this oversampling technique is most appropriate for the highly class-imbalanced MIT and PTB datasets.

<u>For all DL models:</u>

- Training over epochs:
    - Training and validation data used
        - Training data: SMOTE applied
        - Validation data: without SMOTE, check the current status of model performance during the training procedure
    - Automated optimization of models by minimizing the loss function
        - An indicator of how wrong the model predicts
        - If the prediction is bad, e.g., for one class, it will result in a high loss -> class sensitive (optimal for the present datasets)
        - Also possible: maximizing accuracy -> not optimal for unbalanced class distribution, therefore not applied here
    - Training was automatically stopped when training loss did not further decrease to ensure convergence and prevent overfitting (early stopping was added)

A) MIT-BIH Arrhythmia Dataset (MIT dataset)
Workflow DL models:

1. Train/validation and test data split
    a. 80% training data, 20% test data
    b. 20% of training data was selected as validation data, 80% as training data
2. Definition of model architecture and training procedure to compare the performance of different models on test data
    a. Goal: find the best-suited model architecture
3. Optimization of the training procedure for the best model found
    a. Goal: find optimal training setup
    b. Testing different batch sizes and learning rates
4. Model performance evaluation

B) PTB Diagnostic ECG Database (PTB dataset)
Workflow transfer learning:

1. Train/validation and test data split
    a. 80% training data, 20% test data
    b. 20% of training data was selected as validation data, 80% as training data
2. Transfer learning based on the best model from A)
    a. Load pretrained DL model
        i. Cut after the last convolutional layer -> freeze those layers
        ii. Add new classifier layers after pretrained model -> adapted for two-class problem (normal, MI)
    b. Train a new model
        i. Training data: PTB SMOTE training data
        ii. Validation data: PTB data without SMOTE
    c. Optimization of the training procedure
        i. Goal: find optimal training setup
        ii. Testing different batch sizes and learning rates
3. Model performance evaluation

A2 Training - Results for different tested DL models

Three different model types were tested: DNN, CNN, and LSTM. All tested architectures are shown on the next page (Figs. 5-14). For the CNNs, the architectures of CNN2, CNN3, and CNN4 were inspired by the architecture of CNN4 in [2]. The architectures of CNN7, CNN8, and CNN9 were inspired by the architecture of the CNN presented in [4]. Testing LSTMs was also inspired by [2]. As the training procedure of the LSTMs was extremely long compared to the other tested models, especially when adding more layers (LSTM1: 2 layers, LSTM: 6 layers), a combination of CNN4 and LSTM was not performed as in [2] because the performance of CNN4 in [2] could not be reproduced. Also, it would have been necessary to add more layers to the LSTM to reproduce the performance achieved in [2], which would have led to a significantly longer training procedure.

All training procedures were performed with a batch size of 512, as this batch size led to convergence of validation loss, especially with starting learning rates of 0.0005 or 0.001, as indicated in Table 5. The training rate was exponentially reduced during the training.

						F1 per Class - Test				
Model	**lr start**	**lr red**	**batch size**	**F1 Test**	**Accuracy Test**	**0**	**1**	**2**	**3**	**4**
DNN3	0.0005	exp. decay	512	0.8240	0.9543	0.9743	0.5882	0.9079	0.6786	0.9706
CNN1	0.0005	exp. decay	512	0.8834	0.9749	0.9863	0.7340	0.9552	0.7513	0.9901
CNN2	0.0005	exp. decay	512	0.8683	0.9736	0.9857	0.7417	0.9586	0.6651	0.9904
CNN3	0.0005	exp. decay	512	0.8376	0.9647	0.9803	0.6855	0.9572	0.5787	0.9861
CNN4	0.0005	exp. decay	512	0.8672	0.9745	0.9862	0.7676	0.9584	0.6356	0.9870
CNN7	0.0005	exp. decay	512	0.9117	0.9828	0.9910	0.8157	0.9525	0.8068	0.9925
CNN8	0.0005	exp. decay	512	0.8996	0.9799	0.9893	0.7800	0.9628	0.7745	0.9913
CNN9	0.0005	exp. decay	512	0.8626	0.9701	0.9833	0.7293	0.9481	0.6651	0.9870
LSTM1	0.001	exp. decay	512	0.7134	0.9058	0.9450	0.4635	0.8385	0.4012	0.9188
LSTM2	0.001	exp. decay	512	0.8208	0.9547	0.9757	0.6480	0.8906	0.6355	0.9541

Table 5 - DL model performance for different types (DNN: Dense Neural Network, CNN: Convolutional Neural Network, LSTM: Long Short-Term Memory) and architectures (indicated by different numbers). The learning rate (lr) was reduced during training via exponential decay (exp. decay) with the starting lr (lr start). The batch size was 512 for all training procedures. F1-scores and accuracies were achieved by applying the trained model on the hold-out test set.

The used parameters and the performance on unseen test data (F1 score and accuracy) for each tested model architecture are shown in Table 5.

The best performance on MIT test data was achieved for CNN7 (model architecture from [4], with added batch normalization layers). The second-best performance on MIT test data was achieved with CNN8 (the model architecture from [4] with added dropout layers). The validation and training loss/accuracy over the training epochs are shown in Fig. 15 and Fig. 16, respectively. In both cases, the validation loss converged over the training epochs. For CNN7, this was achieved within the first epochs, resulting in a very steep first decrease of validation loss. For CNN8, convergence of the validation loss was achieved around epoch 30. It was decided to further improve the performance of these two CNN models, as the F1 scores on unseen MIT test data seemed to be promising (CNN7 F1 on test data: 0.9117; CNN8 F1 on test data: 0.8996). Both models need further improvement at this point, and early stopping of the training procedure needs to be optimized.

Tested DL model architectures:

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 64)	12,032
batch_normalization (BatchNormalization)	(None, 64)	256
activation (Activation)	(None, 64)	0
dropout (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 32)	2,080
batch_normalization_1 (BatchNormalization)	(None, 32)	128
activation_1 (Activation)	(None, 32)	0
dropout_1 (Dropout)	(None, 32)	0
dense_2 (Dense)	(None, 16)	528
batch_normalization_2 (BatchNormalization)	(None, 16)	64
activation_2 (Activation)	(None, 16)	0
dropout_2 (Dropout)	(None, 16)	0
dense_3 (Dense)	(None, 5)	85

Total params: 15,173 (59.27 KB)
Trainable params: 14,949 (58.39 KB)
Non-trainable params: 224 (896.00 B)

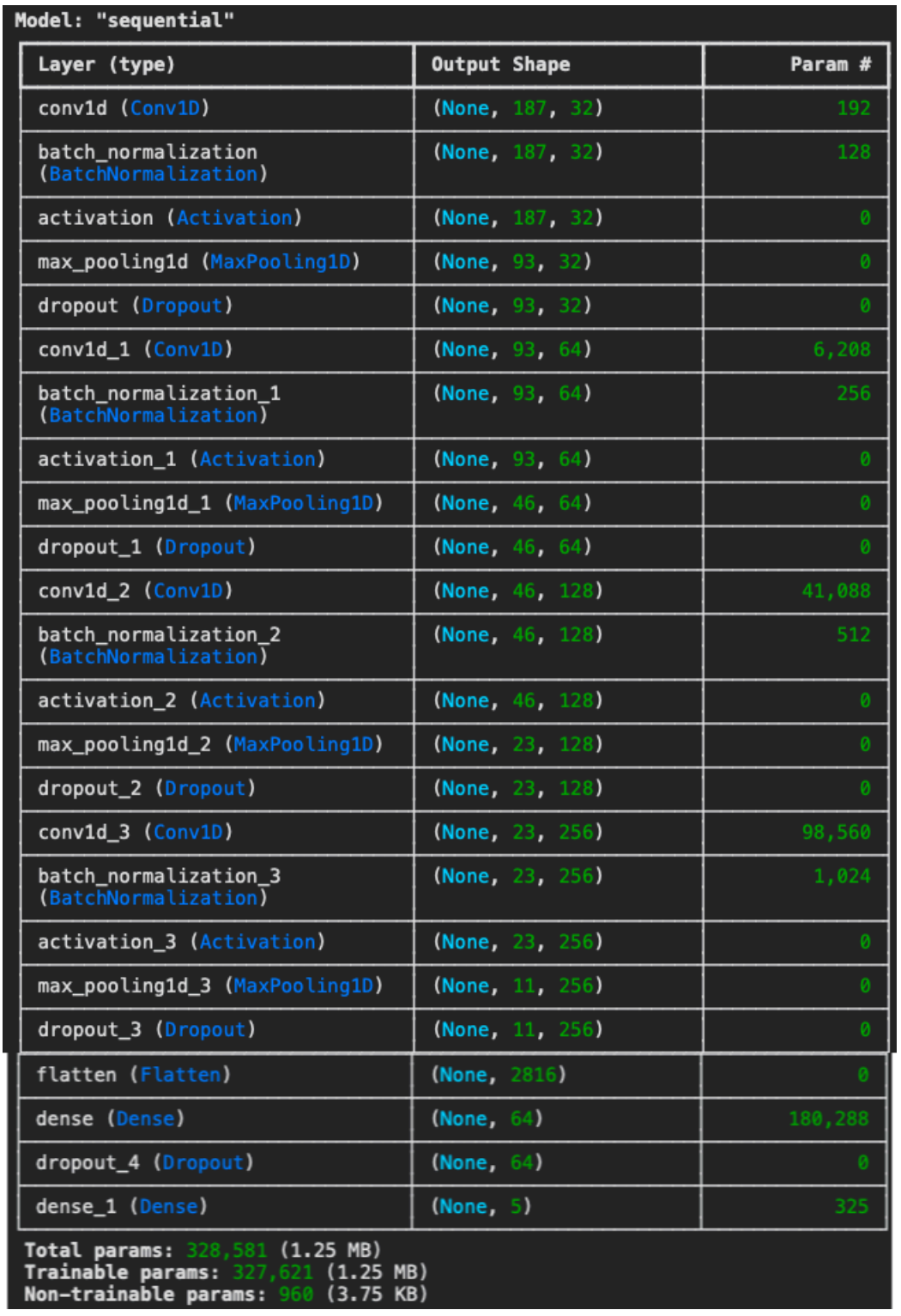
Figure 5 - DNN3 architecture

Model: "sequential"

Layer (type)	Output Shape	Param #
conv1d (Conv1D)	(None, 187, 64)	384
batch_normalization (BatchNormalization)	(None, 187, 64)	256
activation (Activation)	(None, 187, 64)	0
max_pooling1d (MaxPooling1D)	(None, 93, 64)	0
dropout (Dropout)	(None, 93, 64)	0
conv1d_1 (Conv1D)	(None, 93, 64)	20,544
batch_normalization_1 (BatchNormalization)	(None, 93, 64)	256
activation_1 (Activation)	(None, 93, 64)	0
max_pooling1d_1 (MaxPooling1D)	(None, 46, 64)	0
dropout_1 (Dropout)	(None, 46, 64)	0
conv1d_2 (Conv1D)	(None, 46, 32)	6,176
batch_normalization_2 (BatchNormalization)	(None, 46, 32)	128
activation_2 (Activation)	(None, 46, 32)	0
max_pooling1d_2 (MaxPooling1D)	(None, 23, 32)	0
dropout_2 (Dropout)	(None, 23, 32)	0
flatten (Flatten)	(None, 736)	0
dense (Dense)	(None, 32)	23,584
dropout_3 (Dropout)	(None, 32)	0
dense_1 (Dense)	(None, 5)	165

Total params: 51,493 (201.14 KB)
Trainable params: 51,173 (199.89 KB)
Non-trainable params: 320 (1.25 KB)

Figure 6 - CNN1 architecture

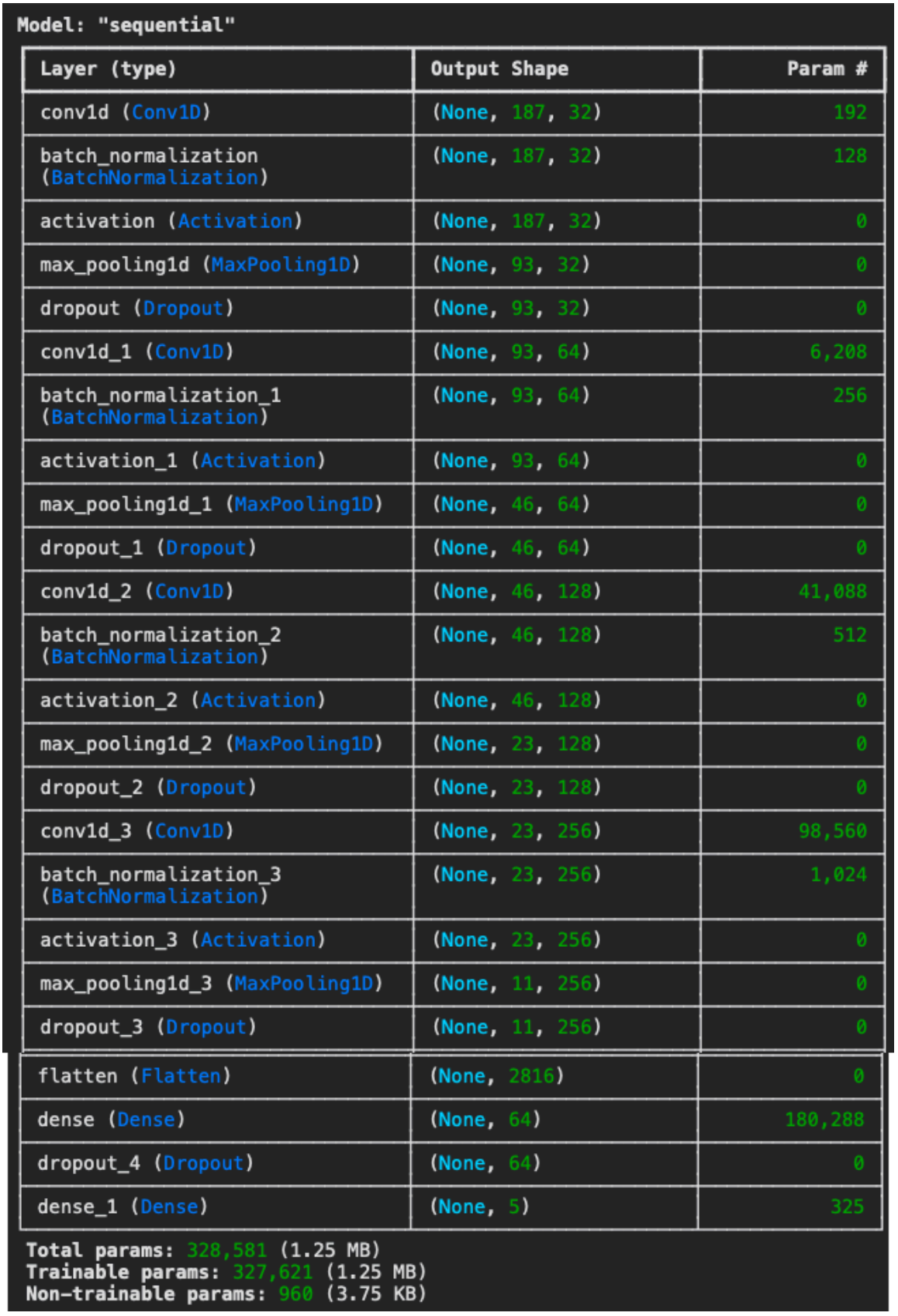


Model: "sequential"

Layer (type)	Output Shape	Param #
conv1d (Conv1D)	(None, 187, 32)	192
batch_normalization (BatchNormalization)	(None, 187, 32)	128
activation (Activation)	(None, 187, 32)	0
max_pooling1d (MaxPooling1D)	(None, 93, 32)	0
dropout (Dropout)	(None, 93, 32)	0
conv1d_1 (Conv1D)	(None, 93, 64)	6,208
batch_normalization_1 (BatchNormalization)	(None, 93, 64)	256
activation_1 (Activation)	(None, 93, 64)	0
max_pooling1d_1 (MaxPooling1D)	(None, 46, 64)	0
dropout_1 (Dropout)	(None, 46, 64)	0
conv1d_2 (Conv1D)	(None, 46, 128)	41,088
batch_normalization_2 (BatchNormalization)	(None, 46, 128)	512
activation_2 (Activation)	(None, 46, 128)	0
max_pooling1d_2 (MaxPooling1D)	(None, 23, 128)	0
dropout_2 (Dropout)	(None, 23, 128)	0
conv1d_3 (Conv1D)	(None, 23, 256)	98,560
batch_normalization_3 (BatchNormalization)	(None, 23, 256)	1,024
activation_3 (Activation)	(None, 23, 256)	0
max_pooling1d_3 (MaxPooling1D)	(None, 11, 256)	0
dropout_3 (Dropout)	(None, 11, 256)	0
flatten (Flatten)	(None, 2816)	0
dense (Dense)	(None, 64)	180,288
dropout_4 (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 5)	325

Total params: 328,581 (1.25 MB)
Trainable params: 327,621 (1.25 MB)
Non-trainable params: 960 (3.75 KB)

Figure 7 - CNN2 architecture



Model: "sequential"

Layer (type)	Output Shape	Param #
conv1d (Conv1D)	(None, 187, 32)	192
batch_normalization (BatchNormalization)	(None, 187, 32)	128
activation (Activation)	(None, 187, 32)	0
max_pooling1d (MaxPooling1D)	(None, 93, 32)	0
dropout (Dropout)	(None, 93, 32)	0
conv1d_1 (Conv1D)	(None, 93, 64)	6,208
batch_normalization_1 (BatchNormalization)	(None, 93, 64)	256
activation_1 (Activation)	(None, 93, 64)	0
max_pooling1d_1 (MaxPooling1D)	(None, 46, 64)	0
dropout_1 (Dropout)	(None, 46, 64)	0
conv1d_2 (Conv1D)	(None, 46, 128)	41,088
batch_normalization_2 (BatchNormalization)	(None, 46, 128)	512
activation_2 (Activation)	(None, 46, 128)	0
max_pooling1d_2 (MaxPooling1D)	(None, 23, 128)	0
dropout_2 (Dropout)	(None, 23, 128)	0
conv1d_3 (Conv1D)	(None, 23, 256)	98,560
batch_normalization_3 (BatchNormalization)	(None, 23, 256)	1,024
activation_3 (Activation)	(None, 23, 256)	0
max_pooling1d_3 (MaxPooling1D)	(None, 11, 256)	0
dropout_3 (Dropout)	(None, 11, 256)	0
flatten (Flatten)	(None, 2816)	0
dense (Dense)	(None, 64)	180,288
dropout_4 (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 5)	325

Total params: 328,581 (1.25 MB)
Trainable params: 327,621 (1.25 MB)
Non-trainable params: 960 (3.75 KB)

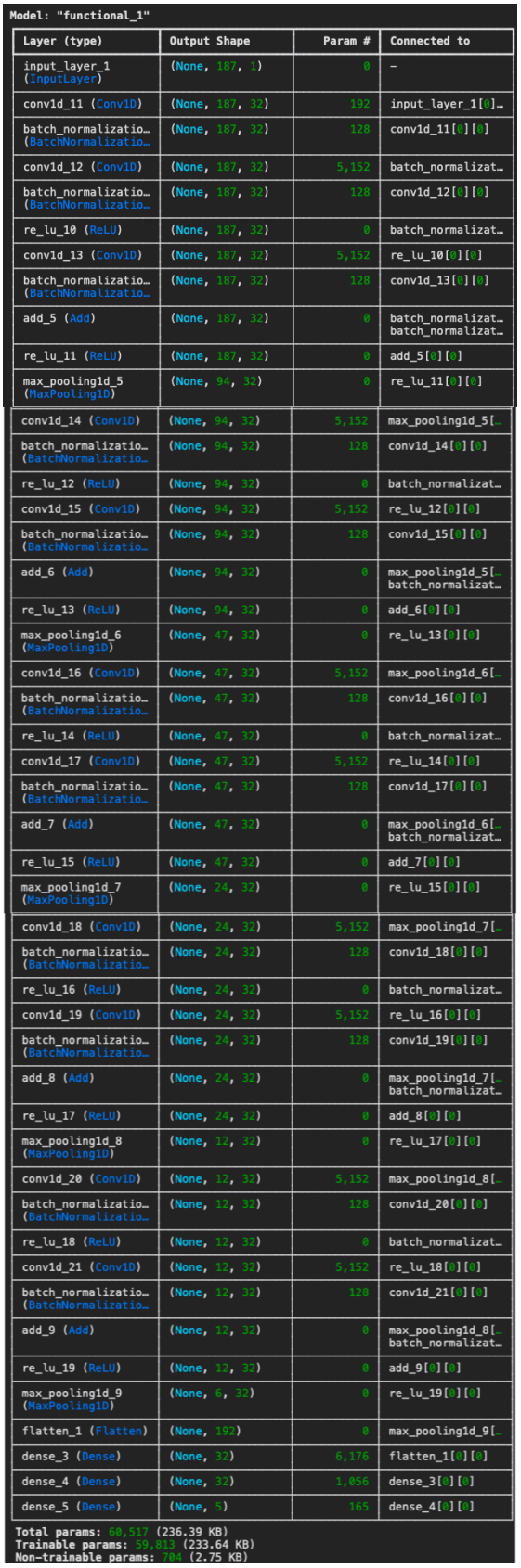
Figure 8 - CNN3 architecture

Model: "sequential"

Layer (type)	Output Shape	Param #
conv1d (Conv1D)	(None, 187, 32)	192
batch_normalization (BatchNormalization)	(None, 187, 32)	128
activation (Activation)	(None, 187, 32)	0
max_pooling1d (MaxPooling1D)	(None, 93, 32)	0
dropout (Dropout)	(None, 93, 32)	0
conv1d_1 (Conv1D)	(None, 93, 64)	6,208
batch_normalization_1 (BatchNormalization)	(None, 93, 64)	256
activation_1 (Activation)	(None, 93, 64)	0
max_pooling1d_1 (MaxPooling1D)	(None, 46, 64)	0
dropout_1 (Dropout)	(None, 46, 64)	0
conv1d_2 (Conv1D)	(None, 46, 128)	41,088
batch_normalization_2 (BatchNormalization)	(None, 46, 128)	512
activation_2 (Activation)	(None, 46, 128)	0
max_pooling1d_2 (MaxPooling1D)	(None, 23, 128)	0
dropout_2 (Dropout)	(None, 23, 128)	0
conv1d_3 (Conv1D)	(None, 23, 256)	98,560
batch_normalization_3 (BatchNormalization)	(None, 23, 256)	1,024
activation_3 (Activation)	(None, 23, 256)	0
max_pooling1d_3 (MaxPooling1D)	(None, 11, 256)	0
dropout_3 (Dropout)	(None, 11, 256)	0
flatten (Flatten)	(None, 2816)	0
dense (Dense)	(None, 64)	180,288
dropout_4 (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 5)	325

Total params: 328,581 (1.25 MB)
Trainable params: 327,621 (1.25 MB)
Non-trainable params: 960 (3.75 KB)

Figure 9 - CNN4 architecture

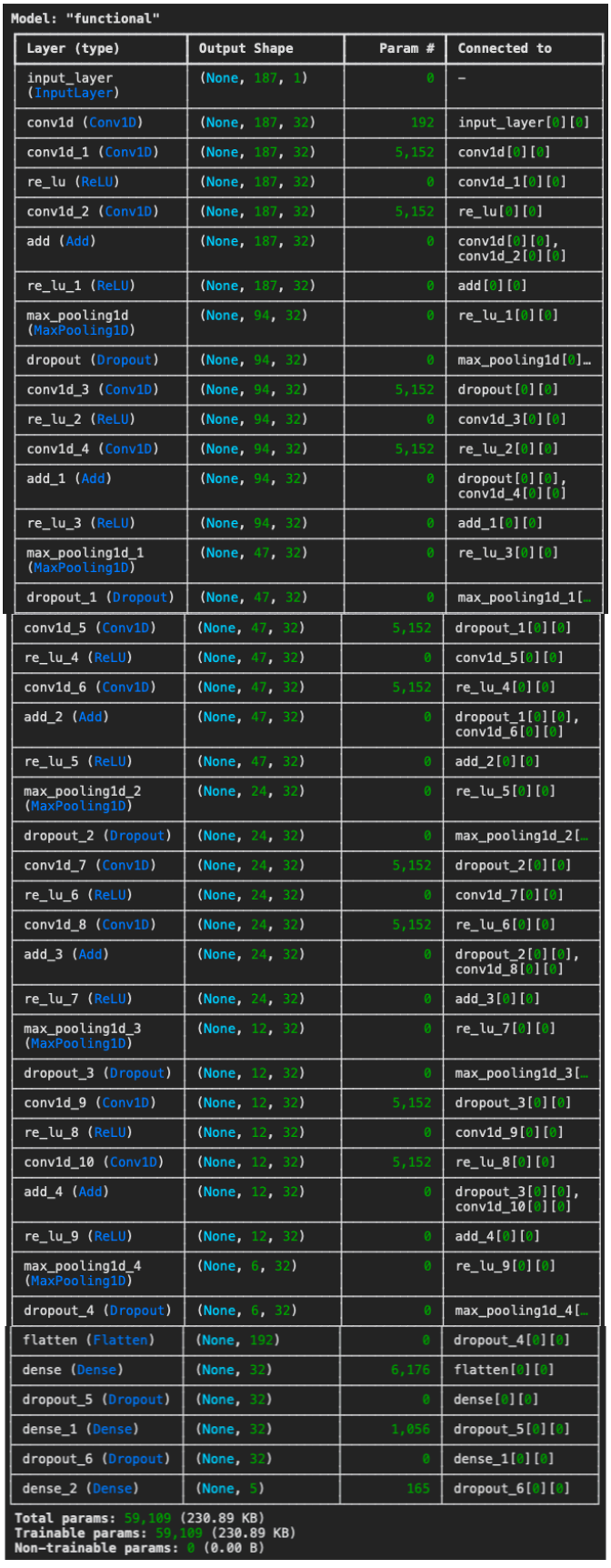


Model: "functional_1"

Layer (type)	Output Shape	Param #	Connected to
input_layer_1 (InputLayer)	(None, 187, 1)	0	–
conv1d_11 (Conv1D)	(None, 187, 32)	192	input_layer_1[0]…
batch_normalizatio… (BatchNormalizatio…	(None, 187, 32)	128	conv1d_11[0][0]
conv1d_12 (Conv1D)	(None, 187, 32)	5,152	batch_normalizat…
batch_normalizatio… (BatchNormalizatio…	(None, 187, 32)	128	conv1d_12[0][0]
re_lu_10 (ReLU)	(None, 187, 32)	0	batch_normalizat…
conv1d_13 (Conv1D)	(None, 187, 32)	5,152	re_lu_10[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 187, 32)	128	conv1d_13[0][0]
add_5 (Add)	(None, 187, 32)	0	batch_normalizat… batch_normalizat…
re_lu_11 (ReLU)	(None, 187, 32)	0	add_5[0][0]
max_pooling1d_5 (MaxPooling1D)	(None, 94, 32)	0	re_lu_11[0][0]
conv1d_14 (Conv1D)	(None, 94, 32)	5,152	max_pooling1d_5[…
batch_normalizatio… (BatchNormalizatio…	(None, 94, 32)	128	conv1d_14[0][0]
re_lu_12 (ReLU)	(None, 94, 32)	0	batch_normalizat…
conv1d_15 (Conv1D)	(None, 94, 32)	5,152	re_lu_12[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 94, 32)	128	conv1d_15[0][0]
add_6 (Add)	(None, 94, 32)	0	max_pooling1d_5[… batch_normalizat…
re_lu_13 (ReLU)	(None, 94, 32)	0	add_6[0][0]
max_pooling1d_6 (MaxPooling1D)	(None, 47, 32)	0	re_lu_13[0][0]
conv1d_16 (Conv1D)	(None, 47, 32)	5,152	max_pooling1d_6[…
batch_normalizatio… (BatchNormalizatio…	(None, 47, 32)	128	conv1d_16[0][0]
re_lu_14 (ReLU)	(None, 47, 32)	0	batch_normalizat…
conv1d_17 (Conv1D)	(None, 47, 32)	5,152	re_lu_14[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 47, 32)	128	conv1d_17[0][0]
add_7 (Add)	(None, 47, 32)	0	max_pooling1d_6[… batch_normalizat…
re_lu_15 (ReLU)	(None, 47, 32)	0	add_7[0][0]
max_pooling1d_7 (MaxPooling1D)	(None, 24, 32)	0	re_lu_15[0][0]
conv1d_18 (Conv1D)	(None, 24, 32)	5,152	max_pooling1d_7[…
batch_normalizatio… (BatchNormalizatio…	(None, 24, 32)	128	conv1d_18[0][0]
re_lu_16 (ReLU)	(None, 24, 32)	0	batch_normalizat…
conv1d_19 (Conv1D)	(None, 24, 32)	5,152	re_lu_16[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 24, 32)	128	conv1d_19[0][0]
add_8 (Add)	(None, 24, 32)	0	max_pooling1d_7[… batch_normalizat…
re_lu_17 (ReLU)	(None, 24, 32)	0	add_8[0][0]
max_pooling1d_8 (MaxPooling1D)	(None, 12, 32)	0	re_lu_17[0][0]
conv1d_20 (Conv1D)	(None, 12, 32)	5,152	max_pooling1d_8[…
batch_normalizatio… (BatchNormalizatio…	(None, 12, 32)	128	conv1d_20[0][0]
re_lu_18 (ReLU)	(None, 12, 32)	0	batch_normalizat…
conv1d_21 (Conv1D)	(None, 12, 32)	5,152	re_lu_18[0][0]
batch_normalizatio… (BatchNormalizatio…	(None, 12, 32)	128	conv1d_21[0][0]
add_9 (Add)	(None, 12, 32)	0	max_pooling1d_8[… batch_normalizat…
re_lu_19 (ReLU)	(None, 12, 32)	0	add_9[0][0]
max_pooling1d_9 (MaxPooling1D)	(None, 6, 32)	0	re_lu_19[0][0]
flatten_1 (Flatten)	(None, 192)	0	max_pooling1d_9[…
dense_3 (Dense)	(None, 32)	6,176	flatten_1[0][0]
dense_4 (Dense)	(None, 32)	1,056	dense_3[0][0]
dense_5 (Dense)	(None, 5)	165	dense_4[0][0]

Total params: 60,517 (236.39 KB)
Trainable params: 59,813 (233.64 KB)
Non-trainable params: 704 (2.75 KB)

Figure 10 - CNN7 architecture

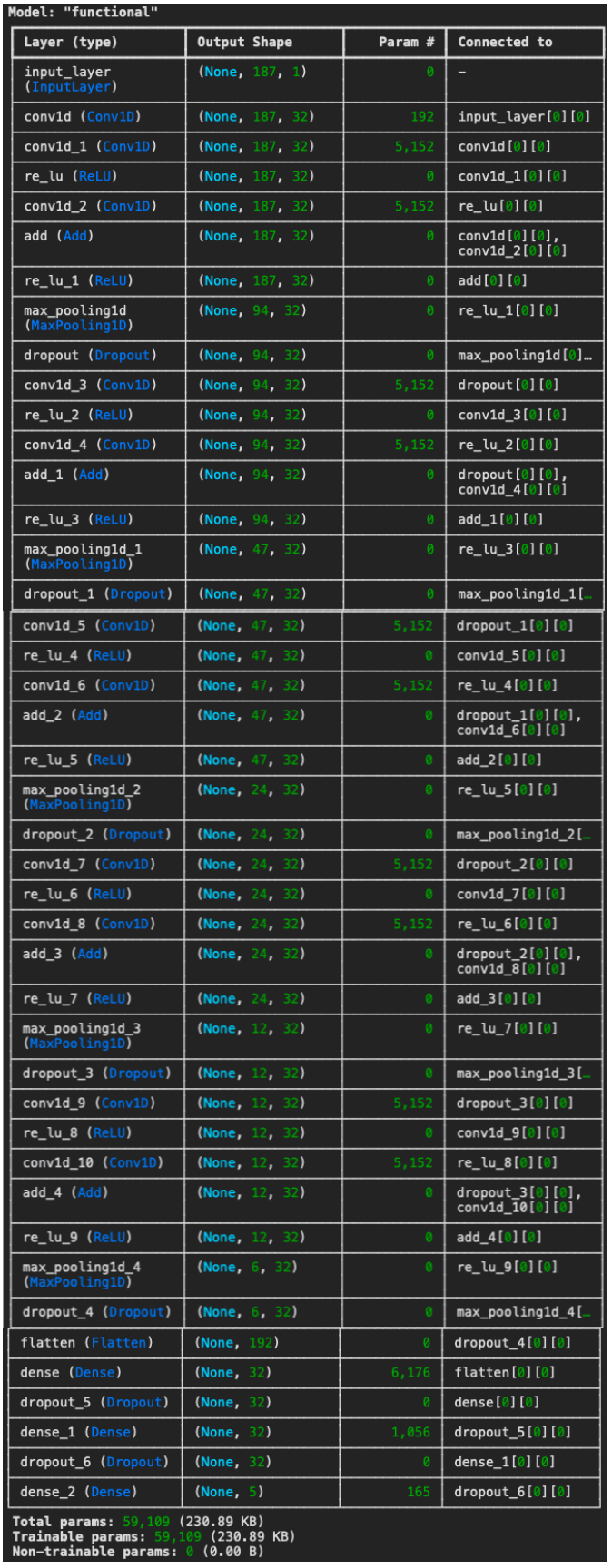


Model: "functional"

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, 187, 1)	0	–
conv1d (Conv1D)	(None, 187, 32)	192	input_layer[0][0]
conv1d_1 (Conv1D)	(None, 187, 32)	5,152	conv1d[0][0]
re_lu (ReLU)	(None, 187, 32)	0	conv1d_1[0][0]
conv1d_2 (Conv1D)	(None, 187, 32)	5,152	re_lu[0][0]
add (Add)	(None, 187, 32)	0	conv1d[0][0], conv1d_2[0][0]
re_lu_1 (ReLU)	(None, 187, 32)	0	add[0][0]
max_pooling1d (MaxPooling1D)	(None, 94, 32)	0	re_lu_1[0][0]
dropout (Dropout)	(None, 94, 32)	0	max_pooling1d[0]…
conv1d_3 (Conv1D)	(None, 94, 32)	5,152	dropout[0][0]
re_lu_2 (ReLU)	(None, 94, 32)	0	conv1d_3[0][0]
conv1d_4 (Conv1D)	(None, 94, 32)	5,152	re_lu_2[0][0]
add_1 (Add)	(None, 94, 32)	0	dropout[0][0], conv1d_4[0][0]
re_lu_3 (ReLU)	(None, 94, 32)	0	add_1[0][0]
max_pooling1d_1 (MaxPooling1D)	(None, 47, 32)	0	re_lu_3[0][0]
dropout_1 (Dropout)	(None, 47, 32)	0	max_pooling1d_1[…
conv1d_5 (Conv1D)	(None, 47, 32)	5,152	dropout_1[0][0]
re_lu_4 (ReLU)	(None, 47, 32)	0	conv1d_5[0][0]
conv1d_6 (Conv1D)	(None, 47, 32)	5,152	re_lu_4[0][0]
add_2 (Add)	(None, 47, 32)	0	dropout_1[0][0], conv1d_6[0][0]
re_lu_5 (ReLU)	(None, 47, 32)	0	add_2[0][0]
max_pooling1d_2 (MaxPooling1D)	(None, 24, 32)	0	re_lu_5[0][0]
dropout_2 (Dropout)	(None, 24, 32)	0	max_pooling1d_2[…
conv1d_7 (Conv1D)	(None, 24, 32)	5,152	dropout_2[0][0]
re_lu_6 (ReLU)	(None, 24, 32)	0	conv1d_7[0][0]
conv1d_8 (Conv1D)	(None, 24, 32)	5,152	re_lu_6[0][0]
add_3 (Add)	(None, 24, 32)	0	dropout_2[0][0], conv1d_8[0][0]
re_lu_7 (ReLU)	(None, 24, 32)	0	add_3[0][0]
max_pooling1d_3 (MaxPooling1D)	(None, 12, 32)	0	re_lu_7[0][0]
dropout_3 (Dropout)	(None, 12, 32)	0	max_pooling1d_3[…
conv1d_9 (Conv1D)	(None, 12, 32)	5,152	dropout_3[0][0]
re_lu_8 (ReLU)	(None, 12, 32)	0	conv1d_9[0][0]
conv1d_10 (Conv1D)	(None, 12, 32)	5,152	re_lu_8[0][0]
add_4 (Add)	(None, 12, 32)	0	dropout_3[0][0], conv1d_10[0][0]
re_lu_9 (ReLU)	(None, 12, 32)	0	add_4[0][0]
max_pooling1d_4 (MaxPooling1D)	(None, 6, 32)	0	re_lu_9[0][0]
dropout_4 (Dropout)	(None, 6, 32)	0	max_pooling1d_4[…
flatten (Flatten)	(None, 192)	0	dropout_4[0][0]
dense (Dense)	(None, 32)	6,176	flatten[0][0]
dropout_5 (Dropout)	(None, 32)	0	dense[0][0]
dense_1 (Dense)	(None, 32)	1,056	dropout_5[0][0]
dropout_6 (Dropout)	(None, 32)	0	dense_1[0][0]
dense_2 (Dense)	(None, 5)	165	dropout_6[0][0]

Total params: 59,109 (230.89 KB)
Trainable params: 59,109 (230.89 KB)
Non-trainable params: 0 (0.00 B)

Figure 11 - CNN8 architecture



Model: "functional"

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, 187, 1)	0	–
conv1d (Conv1D)	(None, 187, 32)	192	input_layer[0][0]
conv1d_1 (Conv1D)	(None, 187, 32)	5,152	conv1d[0][0]
re_lu (ReLU)	(None, 187, 32)	0	conv1d_1[0][0]
conv1d_2 (Conv1D)	(None, 187, 32)	5,152	re_lu[0][0]
add (Add)	(None, 187, 32)	0	conv1d[0][0], conv1d_2[0][0]
re_lu_1 (ReLU)	(None, 187, 32)	0	add[0][0]
max_pooling1d (MaxPooling1D)	(None, 94, 32)	0	re_lu_1[0][0]
dropout (Dropout)	(None, 94, 32)	0	max_pooling1d[0]…
conv1d_3 (Conv1D)	(None, 94, 32)	5,152	dropout[0][0]
re_lu_2 (ReLU)	(None, 94, 32)	0	conv1d_3[0][0]
conv1d_4 (Conv1D)	(None, 94, 32)	5,152	re_lu_2[0][0]
add_1 (Add)	(None, 94, 32)	0	dropout[0][0], conv1d_4[0][0]
re_lu_3 (ReLU)	(None, 94, 32)	0	add_1[0][0]
max_pooling1d_1 (MaxPooling1D)	(None, 47, 32)	0	re_lu_3[0][0]
dropout_1 (Dropout)	(None, 47, 32)	0	max_pooling1d_1[…
conv1d_5 (Conv1D)	(None, 47, 32)	5,152	dropout_1[0][0]
re_lu_4 (ReLU)	(None, 47, 32)	0	conv1d_5[0][0]
conv1d_6 (Conv1D)	(None, 47, 32)	5,152	re_lu_4[0][0]
add_2 (Add)	(None, 47, 32)	0	dropout_1[0][0], conv1d_6[0][0]
re_lu_5 (ReLU)	(None, 47, 32)	0	add_2[0][0]
max_pooling1d_2 (MaxPooling1D)	(None, 24, 32)	0	re_lu_5[0][0]
dropout_2 (Dropout)	(None, 24, 32)	0	max_pooling1d_2[…
conv1d_7 (Conv1D)	(None, 24, 32)	5,152	dropout_2[0][0]
re_lu_6 (ReLU)	(None, 24, 32)	0	conv1d_7[0][0]
conv1d_8 (Conv1D)	(None, 24, 32)	5,152	re_lu_6[0][0]
add_3 (Add)	(None, 24, 32)	0	dropout_2[0][0], conv1d_8[0][0]
re_lu_7 (ReLU)	(None, 24, 32)	0	add_3[0][0]
max_pooling1d_3 (MaxPooling1D)	(None, 12, 32)	0	re_lu_7[0][0]
dropout_3 (Dropout)	(None, 12, 32)	0	max_pooling1d_3[…
conv1d_9 (Conv1D)	(None, 12, 32)	5,152	dropout_3[0][0]
re_lu_8 (ReLU)	(None, 12, 32)	0	conv1d_9[0][0]
conv1d_10 (Conv1D)	(None, 12, 32)	5,152	re_lu_8[0][0]
add_4 (Add)	(None, 12, 32)	0	dropout_3[0][0], conv1d_10[0][0]
re_lu_9 (ReLU)	(None, 12, 32)	0	add_4[0][0]
max_pooling1d_4 (MaxPooling1D)	(None, 6, 32)	0	re_lu_9[0][0]
dropout_4 (Dropout)	(None, 6, 32)	0	max_pooling1d_4[…
flatten (Flatten)	(None, 192)	0	dropout_4[0][0]
dense (Dense)	(None, 32)	6,176	flatten[0][0]
dropout_5 (Dropout)	(None, 32)	0	dense[0][0]
dense_1 (Dense)	(None, 32)	1,056	dropout_5[0][0]
dropout_6 (Dropout)	(None, 32)	0	dense_1[0][0]
dense_2 (Dense)	(None, 5)	165	dropout_6[0][0]

Total params: 59,109 (230.89 KB)
Trainable params: 59,109 (230.89 KB)
Non-trainable params: 0 (0.00 B)

Figure 12 - CNN8 architecture

Model: "sequential"

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 187, 32)	4,352
lstm_1 (LSTM)	(None, 32)	8,320
dense (Dense)	(None, 32)	1,056
dense_1 (Dense)	(None, 5)	165

Total params: 13,893 (54.27 KB)
Trainable params: 13,893 (54.27 KB)
Non-trainable params: 0 (0.00 B)

Figure 13 - LSTM1 architecture

Model: "sequential"

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 187, 32)	4,352
lstm_1 (LSTM)	(None, 187, 32)	8,320
lstm_2 (LSTM)	(None, 187, 32)	8,320
lstm_3 (LSTM)	(None, 187, 32)	8,320
lstm_4 (LSTM)	(None, 187, 32)	8,320
lstm_5 (LSTM)	(None, 32)	8,320
dense (Dense)	(None, 32)	1,056
dense_1 (Dense)	(None, 5)	165

Total params: 47,173 (184.27 KB)
Trainable params: 47,173 (184.27 KB)
Non-trainable params: 0 (0.00 B)

Figure 14 - LSTM2 architecture

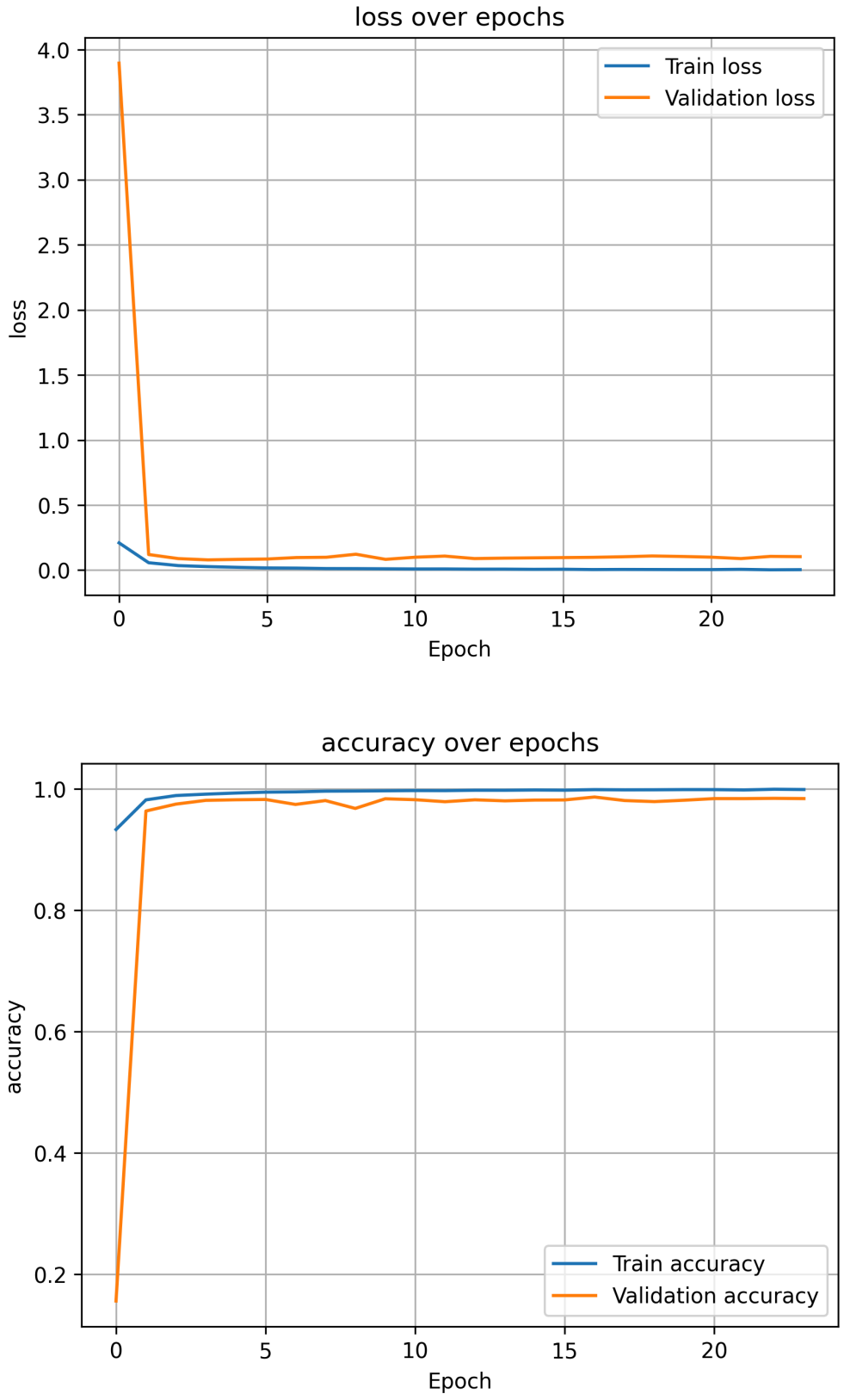


Figure 15 - CNN7: training/validation loss and accuracy over epochs during training procedure

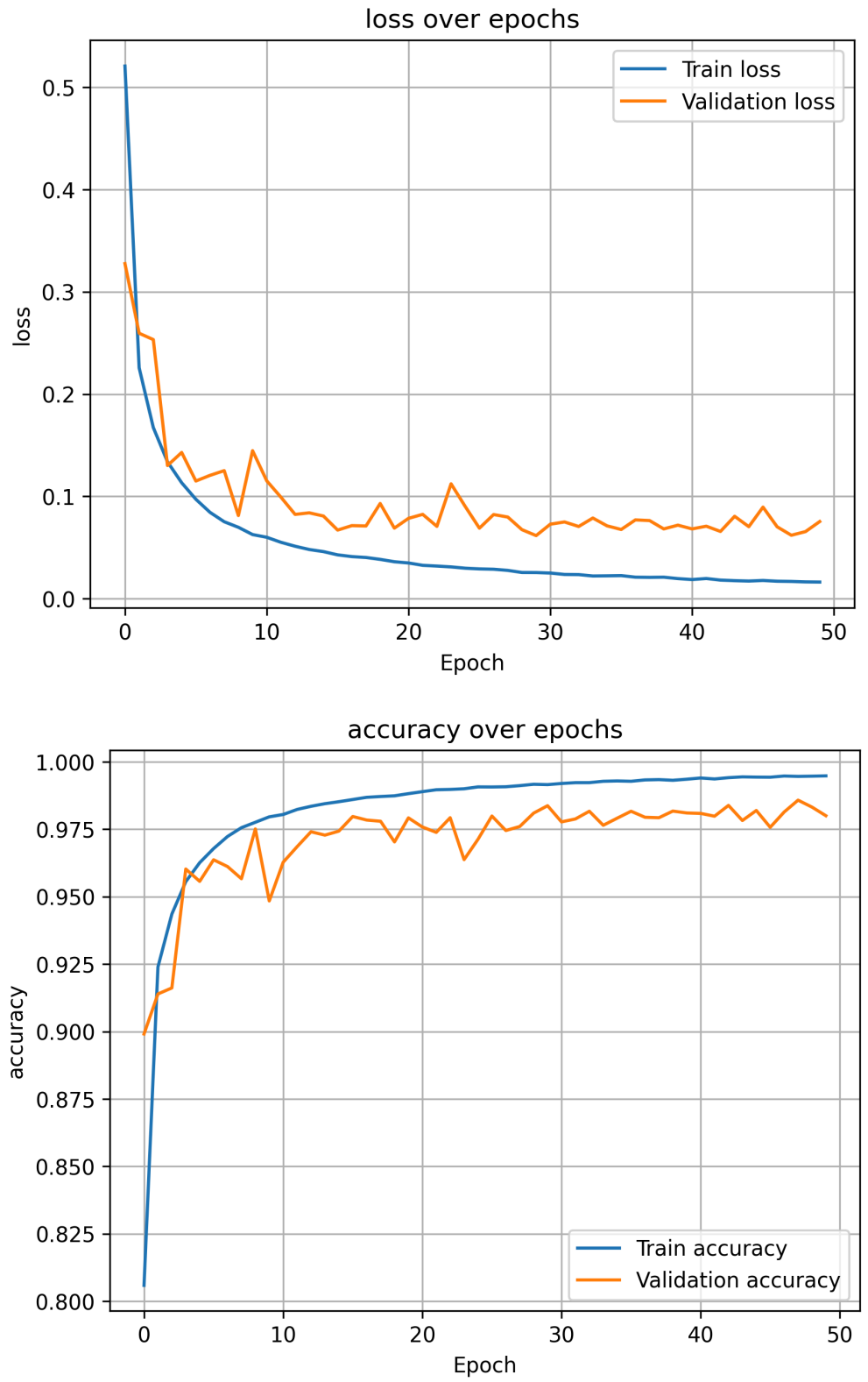


Figure 16 - CNN8: training/validation loss and accuracy over epochs during training procedure

A3 Optimization - Result Table for the best models found in A2, testing different batch sizes and learning rates

The next step was to improve the training procedure of CNN7 and CNN8. The used parameters and the achieved performance on unseen test data (F1 score and accuracy) for the two tested models are shown in Table 6.

						F1 per Class - Test				
Model	**lr start**	**lr red**	**batch size**	**F1 Test**	**Accuracy Test**	**0**	**1**	**2**	**3**	**4**
CNN7	0.0005	exp. decay	512	0.9117	0.9828	0.9910	0.8157	0.9525	0.8068	0.9925
CNN7	0.001	exp. decay	512	0.9108	0.9822	0.9909	0.8171	0.9526	0.8063	0.9871
CNN7	0.0001	exp. decay	512	0.9079	0.9805	0.9895	0.7879	0.9536	0.8193	0.9891
CNN8	0.0005	exp. decay	512	0.8996	0.9799	0.9893	0.7800	0.9628	0.7745	0.9913
CNN8	0.001	exp. decay	512	0.9236	0.9851	0.9924	0.8606	0.9600	0.8171	0.9876

Table 6 - DL model performance for CNN7 and CNN8. The learning rate (lr) was reduced during the training with exponential decay (exp. decay) from the starting lr (lr start). The batch size was 512 for all training procedures, as other batch sizes did not lead to converging validation loss over the epochs. F1-scores and accuracies were achieved by applying the trained model on the hold-out test set.

The best performance on MIT test data was achieved with CNN8 (dropout=0.1 for all added dropout layers), a batch size of 512, and an exponential decay of the learning rate starting at 0.001. For this setup, an F1-score of 0.9236 on MIT test data and an accuracy of 0.9851 were achieved. The validation and training loss/accuracy over the training epochs are shown in Fig. 17. The Validation loss converged over the training epochs. Convergence of the validation loss was achieved around epoch 30. The progression of the validation loss appears a bit noisy, but other batch sizes did not lead to converging loss curves. Furthermore, slight overfitting can be observed from the loss and accuracy curves, indicated by the gap between validation and training data. As the performance on the unseen test data is still good, the overfitting was classified as acceptable

It was decided to try training this setup with MIT training data, with extreme values of the R-distance for class 0 removed. The results can be found in Table 7. The previous results were not improved; therefore, this model was not further used.

						F1 per Class - Test				
Model	**lr start**	**lr red**	**batch size**	**F1 Test**	**Accuracy Test**	**0**	**1**	**2**	**3**	**4**
CNN8	0.001	exp. decay	512	0.8925	0.9770	0.9880	0.7440	0.9576	0.7826	0.9904

Table 7 - DL model performance for CNN8 trained with MIT training data, with removed extreme values of the R distance for class 0. The learning rate (lr) was reduced during the training procedure with exponential decay (exp. decay, starting lr noted as lr start). The batch size was 512. F1-score and accuracy were achieved by applying the trained model on the hold-out test set.

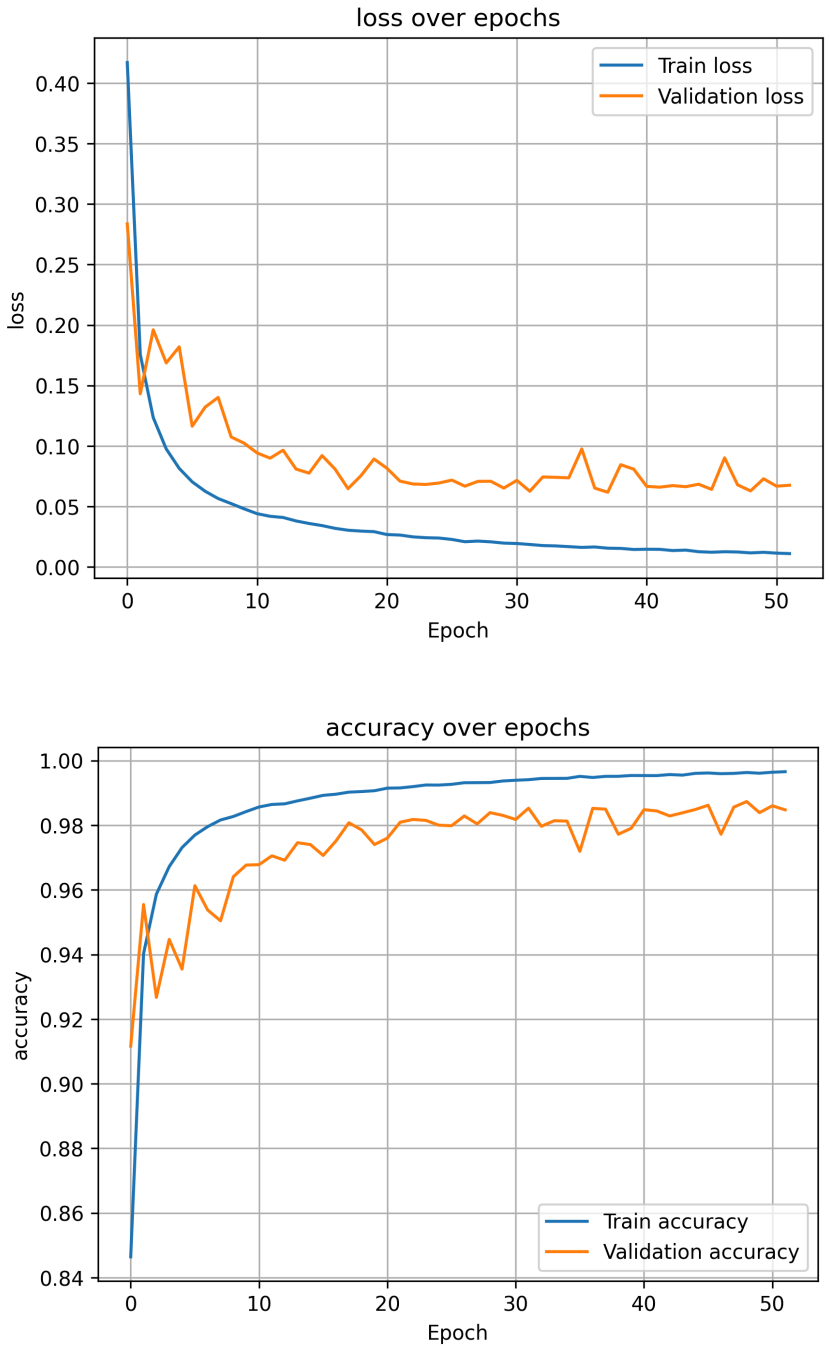


Figure 17 - CNN8: training/validation loss and accuracy over epochs during training procedure

A4 Model Performance Evaluation - Best model found in A3 (CNN8, batch size 512, lr with exp. decay starting at 0.001): confusion matrix, classification report

Confusion matrix:

The confusion matrix (Fig. 18) reflects the good performance of the trained model on unseen MIT test data. Typical misclassifications seem to be samples with true label class 0 classified as class 1 or vice versa. For samples with a true label class 2, the biggest misclassification was in class 3. Samples with true label class 3 were most often misclassified as class 2 or 0. Therefore, neighboring classes are often misclassified.

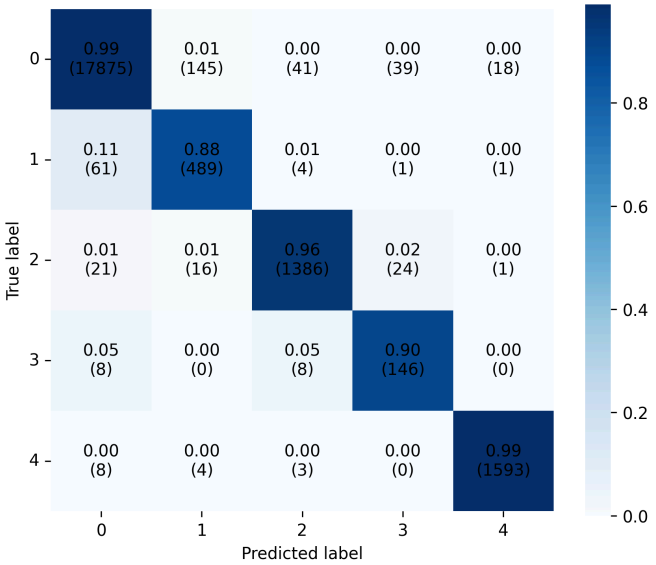


Figure 18 - Confusion Matrix for the best model (CNN8) applied to MIT test data.

Classification report:

Results:

Accuracy:	0.9851
F1 Score:	0.9236
Precision:	0.9062
Recall:	0.9424
Per-Class F1-Scores:	
Class 0:	0.9924
Class 1:	0.8606
Class 2:	0.9600
Class 3:	0.8171
Class 4:	0.9876
Per-Class Precision:	
Class 0:	0.9946
Class 1:	0.8393
Class 2:	0.9580
Class 3:	0.7606
Class 4:	0.9816
Per-Class Recall:	
Class 0:	0.9902
Class 1:	0.8831

Class 2:	0.9620
Class 3:	0.8827
Class 4:	0.9938

According to the F1-score, class 3 is most difficult to predict, which is also reflected in the precision and recall. This is not surprising, as class 3 is the most underrepresented class in the MIT dataset. Therefore, class 3 is the class containing the most SMOTE data in the training dataset to balance the class distribution. Class 1 is the second hardest class to predict, with the same reasons as for class 3. Class 0 is the easiest class to predict, as this class contains most of the data in the MIT dataset. From a medical point of view, it is important to note that only a few arrhythmias are not detected at all (Predicted: 0, True Class: 1, 2, 3, or 4). This means that even if an arrhythmia was misclassified into the wrong label, it was still identified as an arrhythmia, which can then be corrected by a human expert. All in all, the classification report confirms the observations about the class distribution made in Rendering 1.

A4 Model Performance Evaluation - Test of trained CNN8 on PTB data without retraining:

Here, CNN8 trained on MIT training data was applied to PTB data, without retraining, as a first performance test. For this, the PTB data was classified as class 0-4. Then classes 1-4 were combined into an 'abnormal' class (class 1).

Results:

F1 Score:	0.3208
Per-Class F1-Scores:	
Class 0:	0.4507
Class 1:	0.1902

The average F1 score was 0.3208. Especially for class 1, the F1 score was very low (0.1902), indicating that most of the data with a true label of 1 were classified as class 0. The performance was not acceptable; therefore, it was decided to perform transfer learning with CNN8 with PTB training data to improve the performance on PTB test data.

B2 Transfer learning - model architecture, retraining, optimization

The pretrained CNN8 was loaded, and the architecture was cut after the last residual block (in total, 5 residual blocks present). In a first test, all those were frozen and therefore not adapted in an additional training procedure with PTB training and validation data. A new classifier was then added, which was adapted for the binary classification problem of the PTB dataset. The added layers are named 'transfer'. Different options of added transfer

layers were tested (as indicated by the numbers) to find the best option. The architecture of CNN8 + transfer2 can be found in Fig. 19. In principle, all the different tested transfer models have the same architecture, but with added batch normalization layers or increased dropout (for transfer2, dropout=0.1). The batch normalization and dropout values are listed on the next page for the different transfer models.

Retraining was performed with PTB training and validation data. The retrained models were tested on unseen PTB test data. Results of the pretrained CNN8 combined with different transfer models can be found in Table 9. In this table, only the results for the best combination of learning rate and batch size achieved (indicated by the F1 score) are listed (more combinations were tested).

The best performance was achieved by CNN8 + transfer2. As the F1 score was still relatively low, it was decided to test another strategy. For this, only the first four residual blocks were frozen, and the last block was also retrained with the added transfer layers. The results for this approach can be found in Table 10.

The performance was significantly increased. The best performance was achieved for CNN8 + transfer6, reaching an average F1 score of 0.9805 and an accuracy of 0.9842.

Model: "functional_3"

Layer (type)	Output Shape	Param #
input_layer_2 (InputLayer)	(None, 187, 1)	0
functional_1 (Functional)	(None, 6, 32)	51,712
flatten_2 (Flatten)	(None, 192)	0
dense_6 (Dense)	(None, 32)	6,176
dense_7 (Dense)	(None, 32)	1,056
dropout_8 (Dropout)	(None, 32)	0
dense_8 (Dense)	(None, 2)	66

Total params: 59,010 (230.51 KB)
Trainable params: 7,298 (28.51 KB)
Non-trainable params: 51,712 (202.00 KB)

Figure 19 - Model architecture for the transfer learning procedure. Here: CNN8 + transfer2

⋮

transfer Model	batch normalization	dropout
2	no	0.1
3	no	0.4
4	before activation	0.3
5	before activation	0.4
6	no	0.3
7	before activation	0.1
8	before activation	0.2

Table 8 - Setup for the different tested transfer models added after CNN8.

						F1 per Class - Test	
Model	**lr start**	**lr red**	**batch size**	**F1 Test**	**Accuracy Test**	**0**	**1**
CNN8 + transfer2	0.001	exp. decay	512	0.8915	0.9062	0.8517	0.9314
CNN8 + transfer3	0.001	exp. decay	512	0.8689	0.8845	0.8237	0.9141
CNN8 + transfer4	0.001	exp. decay	256	0.8826	0.8986	0.8392	0.9259
CNN8 + transfer5	0.001	exp. decay	512	0.8601	0.8766	0.8119	0.9082
CNN8 + transfer6	0.001	exp. decay	256	0.8715	0.8879	0.8257	0.9174
CNN8 + transfer 7	0.001	exp. decay	128	0.8879	0.9031	0.8467	0.9291
CNN8 + transfer8	0.001	exp. decay	128	0.8850	0.9007	0.8425	0.9274

Table 9 - Performance of different transfer model versions with different added transfer layers. The learning rate (lr) was reduced during the transfer training procedure with exponential decay (exp. decay, starting lr noted as lr start). The batch sizes were chosen to achieve convergence of the validation loss over the epochs. F1 scores and accuracies were achieved by applying the retrained model on the hold-out PTB test set.

						F1 per Class - Test	
Model	**lr start**	**lr red**	**batch size**	**F1 Test**	**Accuracy Test**	**0**	**1**
CNN8 + transfer2	0.001	exp. decay	128	0.9801	0.9838	0.9715	0.9887
CNN8 + transfer3	0.001	exp. decay	128	0.9727	0.9777	0.9611	0.9843
CNN8 + transfer4	0.001	exp. decay	128	0.9752	0.9797	0.9645	0.9858
CNN8 + transfer6	0.001	exp. decay	138	0.9805	0.9842	0.9721	0.9890

Table 10 - Performance of different transfer model versions with different added transfer layers. Here, the last residual block of CNN8 was unfrozen and also retrained with PTB training data. The learning rate (lr) was reduced during the transfer training procedure with exponential decay (exp. decay, starting lr noted as lr start). The batch sizes were chosen to achieve convergence of the validation loss over the epochs. F1 scores and accuracies were achieved by applying the retrained model on the hold-out PTB test set.

B3 Transfer learning - Model performance evaluation

Confusion Matrix:

The confusion matrix (Fig. 20) reflects the good performance of the trained model on unseen PTB test data. The biggest misclassification is present for samples with a true label of 1 classified as class 0, but even this misclassification is very small.

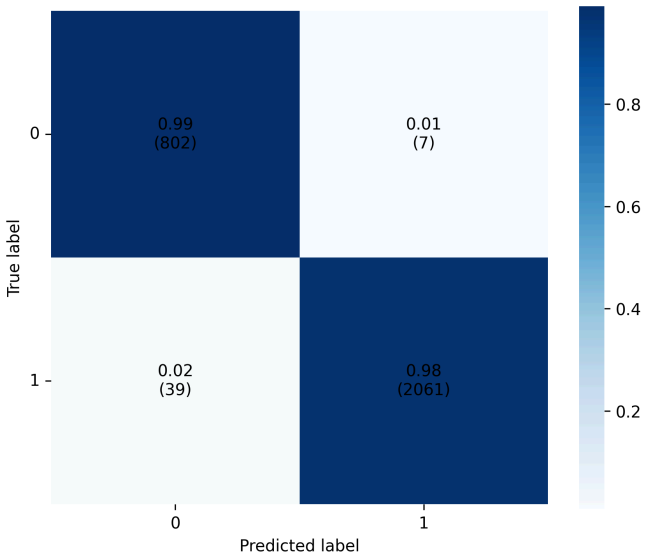


Figure 20 - Confusion Matrix for the best transfer model applied to PTB test data

Classification Report:

Results:

Accuracy:	0.9842
F1 Score:	0.9805
Precision:	0.9751
Recall:	0.9864
Per-Class F1-Scores:	
Class 0:	0.9721
Class 1:	0.9890
Per-Class Precision:	
Class 0:	0.9536
Class 1:	0.9966
Per-Class Recall:	
Class 0:	0.9913
Class 1:	0.9814

According to the F1-score, class 0 is more difficult to predict than class 1. This is not surprising, as class 0 is underrepresented in the PTB dataset. Therefore, class 0 is the class containing the most SMOTE data in the training dataset to balance the class distribution. In general, the performance difference for class 0 and class 1 is small. It can be assumed that transfer learning helped achieve this performance.

All the modeling results for the best-found baseline and DL models are summarised in Table 11.

Result Summary

			Metrics on test data						
Model	**Dataset**	**Sampling**	**Avg accuracy**	**Avg F1 Score**	**F1 Class 0**	**F1 Class 1**	**F1 Class 2**	**F1 Class 3**	**F1 Class 4**
XGB	MIT	SMOTE	0.9823	0.9096	0.9908	0.8221	0.9533	0.7988	0.9831
CNN8	MIT	SMOTE	0.9851	0.9236	0.9924	0.8606	0.9600	0.8171	0.9876
XGB	PTB	SMOTE	0.9739	0.9730	0.9591	0.9843	-	-	-
CNN8 + transfer6	PTB	SMOTE	0.9842	0.9805	0.9721	0.9890	-	-	-

Table 11 - Performance summary for the best baseline and DL models trained with MIT and PTB training data (SMOTE) to solve the multi-class arrhythmia classification problem (MIT dataset) or the binary MI detection problem (PTB dataset).

Interpretation of results

- Have you analyzed the errors in your model?
- Did this contribute to his improvement? If yes, describe.

We faced large class imbalances, and our model's performance was lower (F1-score), especially for the minority classes in the MIT dataset (classes 1 and 3). Also, common misclassifications for both models were analysed using a SHAP analysis.

Additionally, we tried training with MIT data with removed extreme values of the R distance for class 0 (identified in Rendering 1), but the overall performance could not be improved. Therefore, those models were not used for further investigations.

- Have you used interpretability techniques such as SHAP?

SHAP DL - A) MIT dataset:

A SHAP analysis was chosen to gain insights into the extent to which the individual time steps contribute to the classification decision of the model. It is conceivable that this could enable conclusions to be drawn about the extent to which the individual phases of the ECG signal could be significant for the decision-making of the applied model. For this, the ECG signals for a limited number of samples per class will be plotted together with the Shapley values for the individual timesteps in the end.

The analysis was conducted on 200 test samples (40 samples per class) to understand which temporal features are important for the model to predict classes. 200 test samples were chosen to prevent runtime issues. Moreover, this amount of data should be sufficient to achieve meaningful results. Furthermore, a background sample of 100 samples was generated. This acts as a reference distribution, as the SHAP analysis should answer the question of how much a certain feature contributes to the prediction made. For this, SHAP analyzes what the prediction would be without a certain feature. To do this, a random background sample, which acts as a ‘typical’ sample, is used as a replacement for a certain feature looked at in the analysis. The difference in prediction is then translated into a measure for the contribution of a certain feature to the prediction made by the model.

Analysed samples:	200, 40 samples per class
Background samples:	100 to generate the SHAP baseline
Explainer:	DeepExplainer, suitable for neural networks
Features:	187, time steps of original ECG data

In Fig. 21-25, the 20 most important features for each class can be found. The calculation was done by determining the importance regarding the decision made by the model of every feature for every sample in the different classes. In the next step, these results were averaged to generate a single value for each class per feature. Those were ranked.

For classes 0, 1, and 4, feature 3 is the most important. For classes 2 and 3, the most important timestep is feature 2. In general, very early timesteps (especially 1, 2, 3, and 6), early timesteps, or timesteps around the middle of the ECG signal seem to be very important for the decision process of the model. This is not surprising as the data is zero-padded, resulting in a lot of samples with only zeros towards the end of the ECG signal. This part then seems to have low importance for the model.

Feature importance for all classes from Fig. 21 - Fig. 25:

Class 0: see Fig. 21
- 11 out of the 20 most important features are before timestep 20
- 6 out of the 20 most important features are between timestep 90 and 110
- The rest are timesteps 159, 80, and 26
- Very early timesteps and timesteps around the middle of the signal seem to be important for the model to predict class 0

Class 1: see Fig. 22
- 15 out of the 20 most important features are before timestep 20
- 3 out of the 20 most important features are between timestep 90 and 110
- The rest are timesteps 26 and 23
- Very early timesteps of the signal seem to be important for the model to predict class 1

Class 2: see Fig. 23
- 12 out of the 20 most important features are before timestep 20
- 7 out of the 20 most important features are between timestep 20 and 30
- The rest is timestep 95
- Very early and early timesteps of the signal seem to be important for the model to predict class 2

Class 3: see Fig. 24
- 8 out of the 20 most important features are before timestep 20
- 3 out of the 20 most important features are between timestep 60 and 70
- 5 out of the 20 most important features are between timestep 90 and 110
- The rest are timesteps 80, 55, 25, and 37
- Very early timesteps and timesteps around the middle of the signal seem to be important for the model to predict class 3
- The distribution of important timesteps is bigger

Class 4: see Fig. 25

- 7 out of the 20 most important features are before timestep 20
- 10 out of the 20 most important features are between timestep 90 and 110
- The rest are timesteps 159, 43, and 42
- Very early timesteps and timesteps around the middle of the signal seem to be important for the model to predict class 4

This distribution can start to explain the misclassifications of the model, as for all classes, very early timesteps have a high importance.

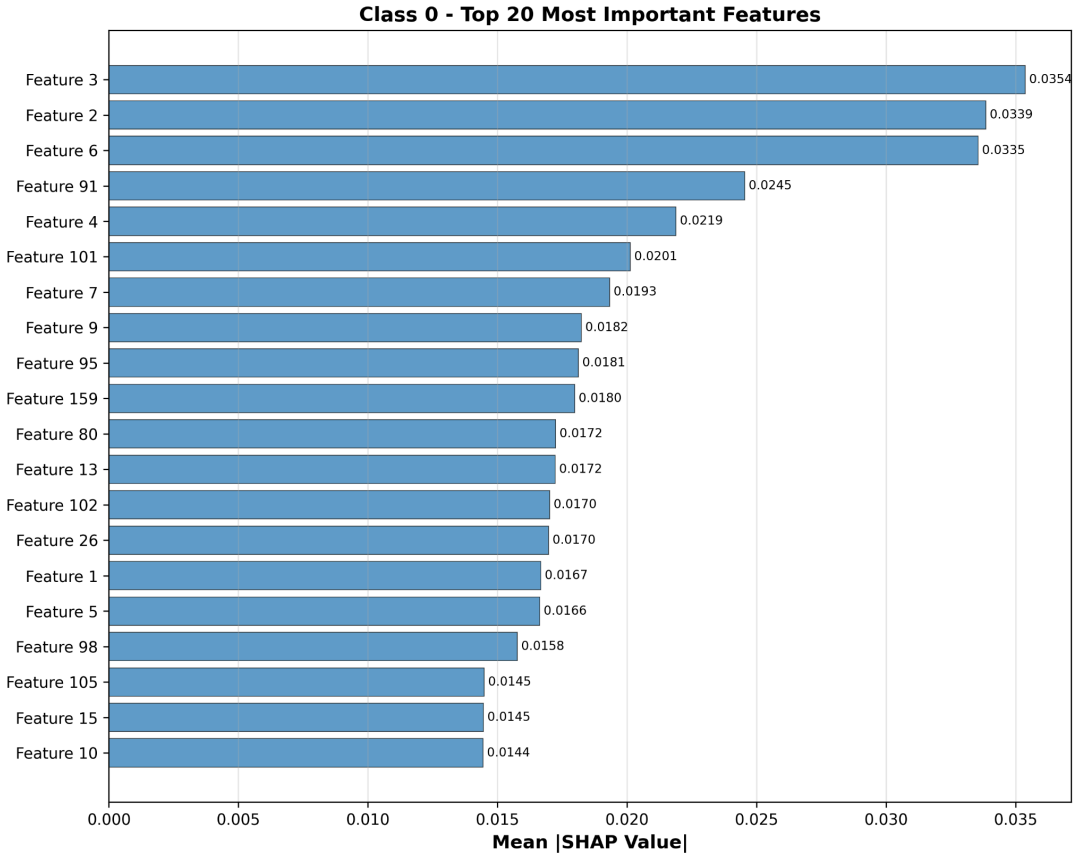


Figure 21 - 20 most important features for class 0

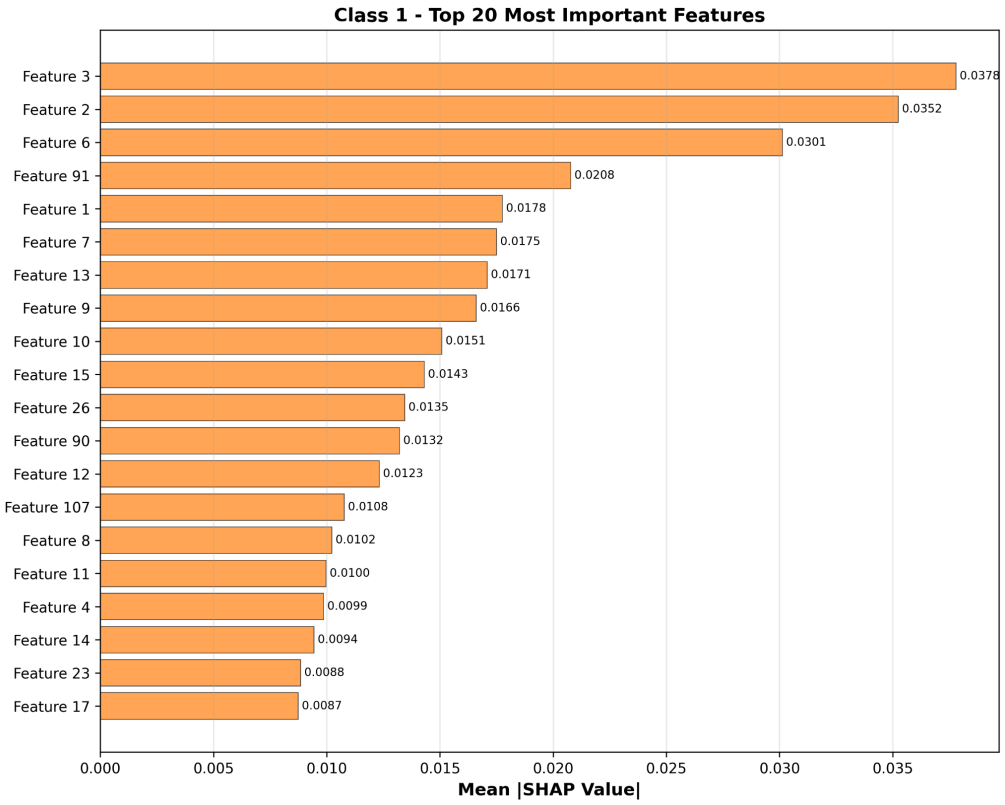


Figure 22 - 20 most important features for class 1

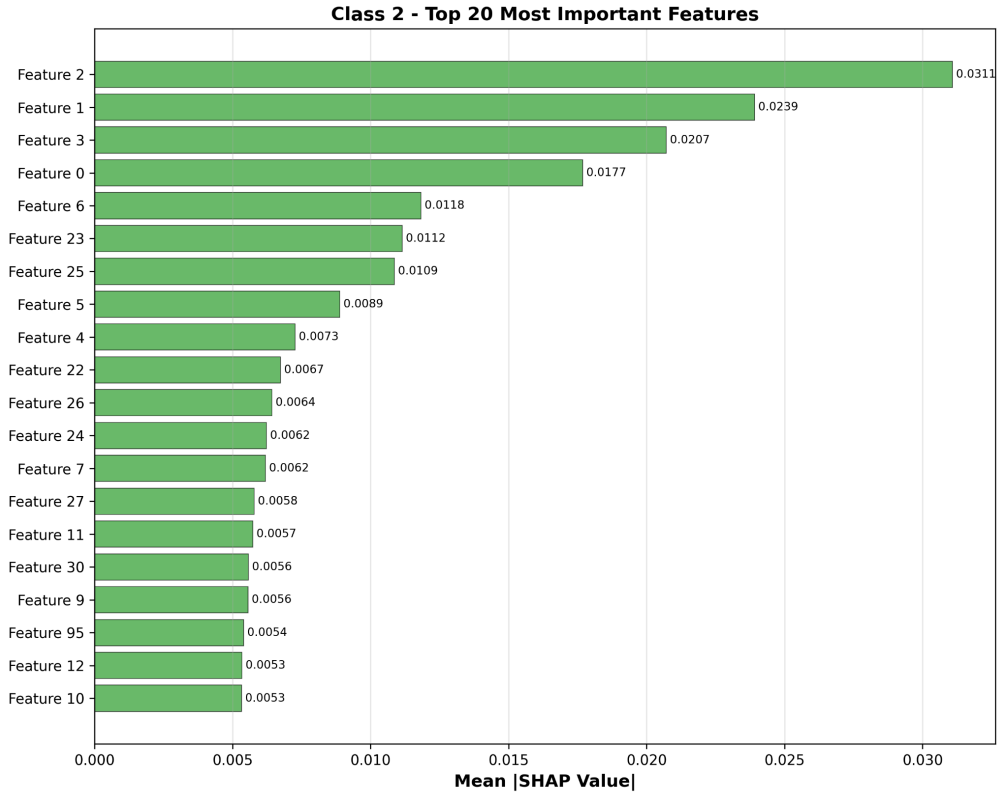


Figure 23 - 20 most important features for class 2

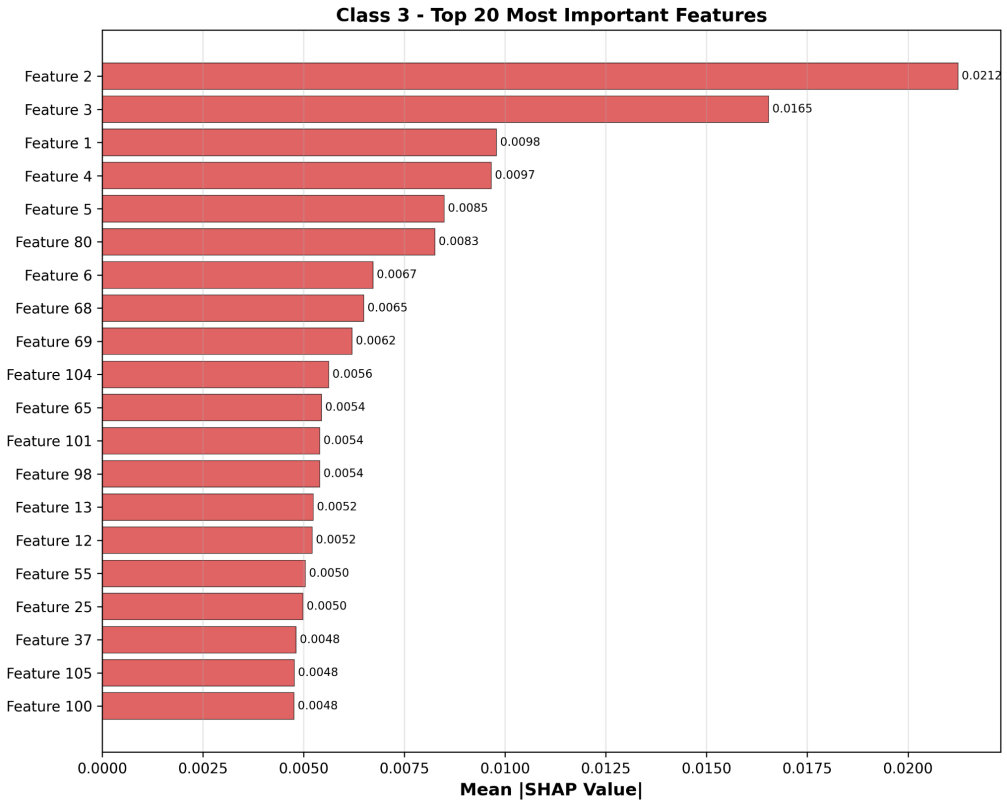


Figure 24 - 20 most important features for class 3

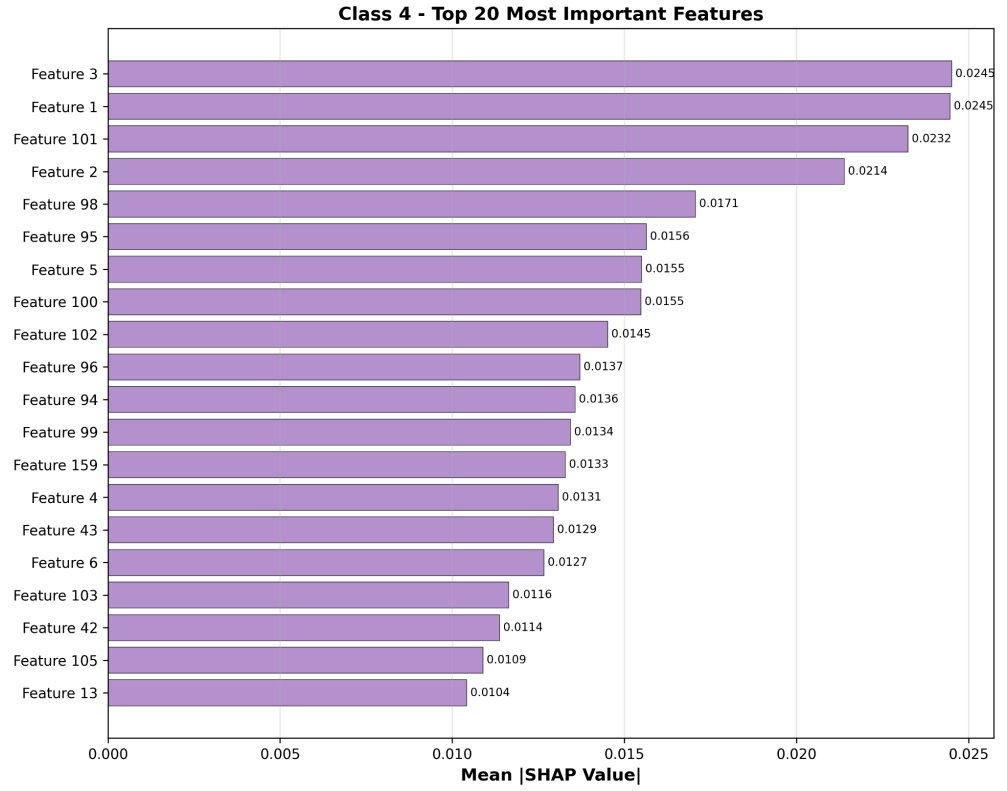


Figure 25 - 20 most important features for class 4

Detailed analysis of SHAP values for different features:

Class 0: see Fig. 26

- SHAP range: -3.2 to 4.2
- SHAP values for all features centered around 0, with extreme values
- Most of the data is spread between -0.1 and 0.1
- Most important feature: 3
    - Single low value at -2 prevents classification as class 0
    - Single low value at 1 pushes classification towards class 0
- Second most important feature: 2
    - Single low to medium value at -2 prevents classification as class 0
    - Single medium value at 0.8 pushes classification towards class 0
- Third most important feature: 6
    - Single low value at 4.2 pushes classification towards class 0
- Feature 91 -> 4th most important feature
    - Single low value at -3.2 prevents classification as class 0

Class 1: see Fig. 27

- SHAP range: -4.2 to 3.1
- SHAP values for all features are centered around 0, with some extreme values
- Most of the data is spread between -0.1 and 0.1
- Most important feature: 3
    - Single low value at 1.6 pushes classification towards class 1
- Second most important feature: 2
    - Single medium value at 1.5 pushes classification towards class 1
- Third most important feature: 6
    - Single low value at -4.2 prevents classification as class 1
- Feature 91 -> 4th most important feature
    - Single high value at 3.1 pushes classification towards class 1

Class 2: see Fig. 28

- SHAP range: -0.06 to 0.2
- SHAP values for all features are centered slightly left of 0, with a few extreme values
- Most of the data is spread between -0.04 and 0.02
- Horizontal spread of points
- No strong relation on individual features (only a few extreme values)
- Most important feature: 2
    - Single high value at 0.2 pushes classification towards class 2
- Second most important feature: 1
    - The accumulation of a few low values at 0.12 pushes classification towards class 2

- Third most important feature: 3
    - Tendency of lower values preventing classification as class 2 and higher values pushing classification towards class 2 -> separation of blue dots on the left and red dots on the right (also present for feature 0, but vice versa: high values preventing classification as class 2, low values pushing classification towards class 2)
- Feature 95 -> 18th most important feature
    - Single high value at -0.06 prevents classification as class 2

Class 3: see Fig. 29

- SHAP range: -0.6 to 1.1
- SHAP values for all features centered around 0
- Most of the data is spread between -0.1 and 0.1
- Most important feature: 2
    - Tendency of lower values preventing classification as class 3 and higher values pushing classification towards class 3 -> separation of blue dots on the left and red dots on the right
    - Single medium value at -0.25 prevents classification as class 3
- Second most important feature: 3
    - Single low value at -0.6 prevents classification as class 3
- Third most important feature: 1
    - Tendency of higher values pushing classification towards class 3
- Feature 80 -> 6th most important feature
    - Single low value at 1.1 pushes classification towards class 3

Class 4: see Fig. 30

- SHAP range: -2.6 to 1.5
- SHAP values for all features centered around 0, with extreme values
- Most of the data is spread between -0.1 and 0.1
- most important feature: 3
    - Single high value at 0.4 pushes classification towards class 4
    - Single low value at -0.4 prevents classification as class 4
    - tendency of higher values pushing classification towards class 4
- Second most important feature: 1
    - Single high value at -0.5 prevents classification as class 4
    - Tendency of higher values preventing classification as class 4
- Third most important feature: 101
    - Single medium value at -0.6 prevents classification as class 4
    - Single low value at 0.8 pushes classification towards class 4
- Feature 159 -> 13th most important feature
    - Single high value at -2.6 prevents classification as class 4
- Feature 42 -> 18th most important feature
    - Single high value at 1.4 pushes classification towards class 4

Misclassification class 0/class 1: possible reasons

- The top 4 most important features are the same
- Very early timesteps are important for both classes -> overlap

Good performance class 2: possible reasons

- Timesteps between 20 and 30 play an important role -> not for other classes, unique characteristic for class 2

Lower performance class 3: possible reasons

- Timesteps between 60 and 70 play a role -> not for other classes, unique characteristic for class 3: maybe those timesteps are less distinctive than early timesteps or timesteps around the middle of the signal
- Only one extreme value in the top 20 features is pushing classification towards class 3

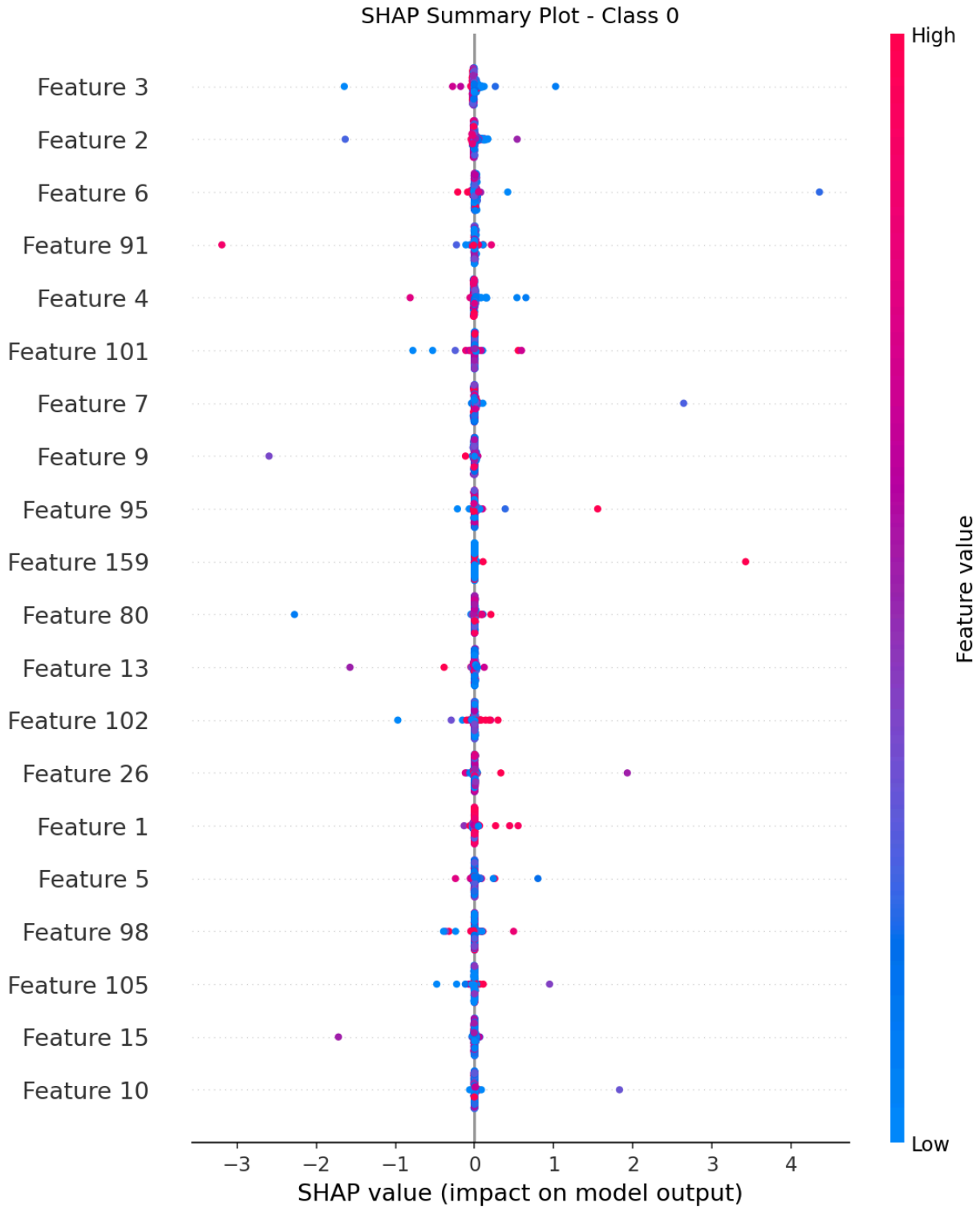


Figure 26 - Distribution of SHAP values for the 20 most important features for class 0

Figure 27 - Distribution of SHAP values for the 20 most important features for class 1

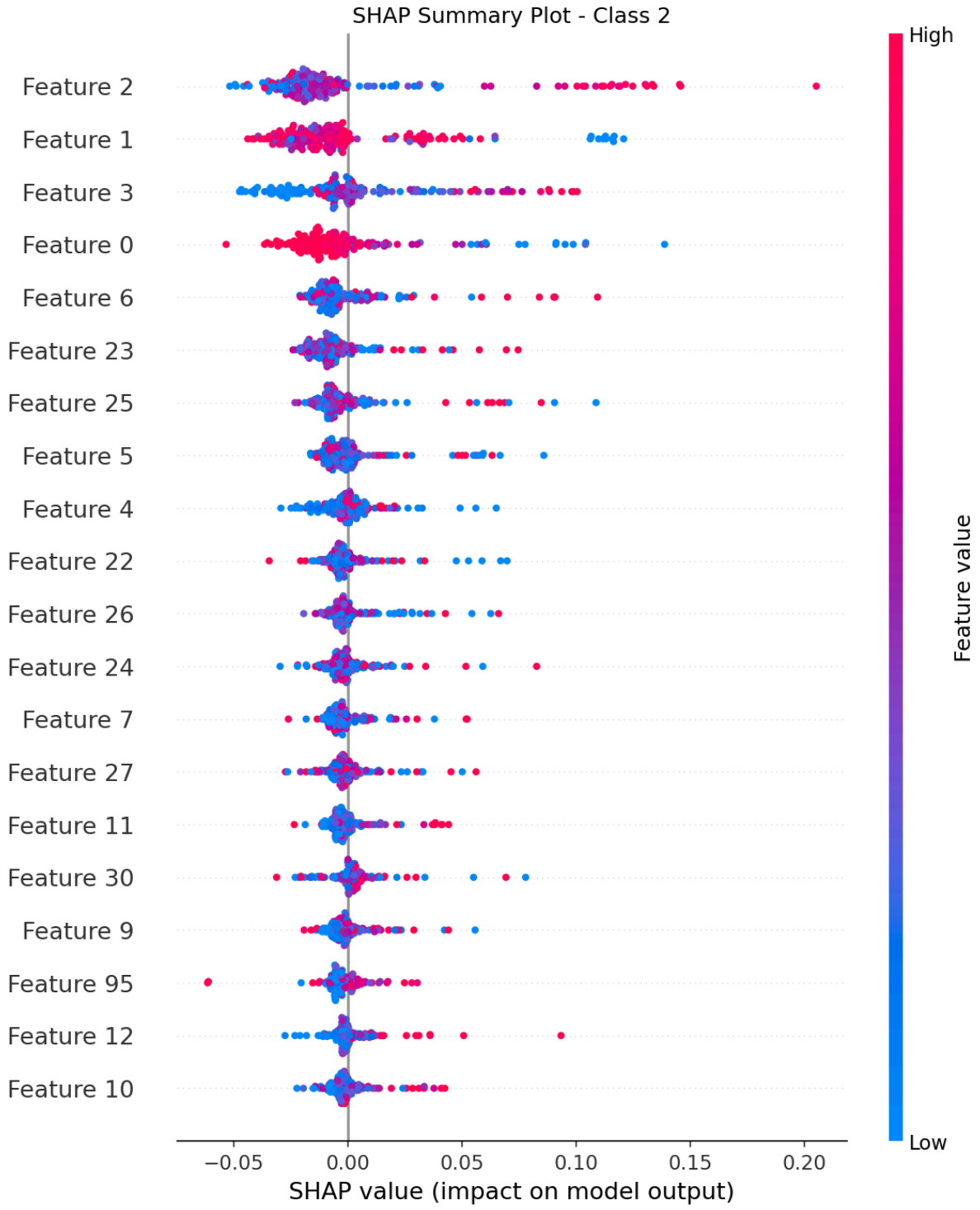


Figure 28 - Distribution of SHAP values for the 20 most important features for class 2

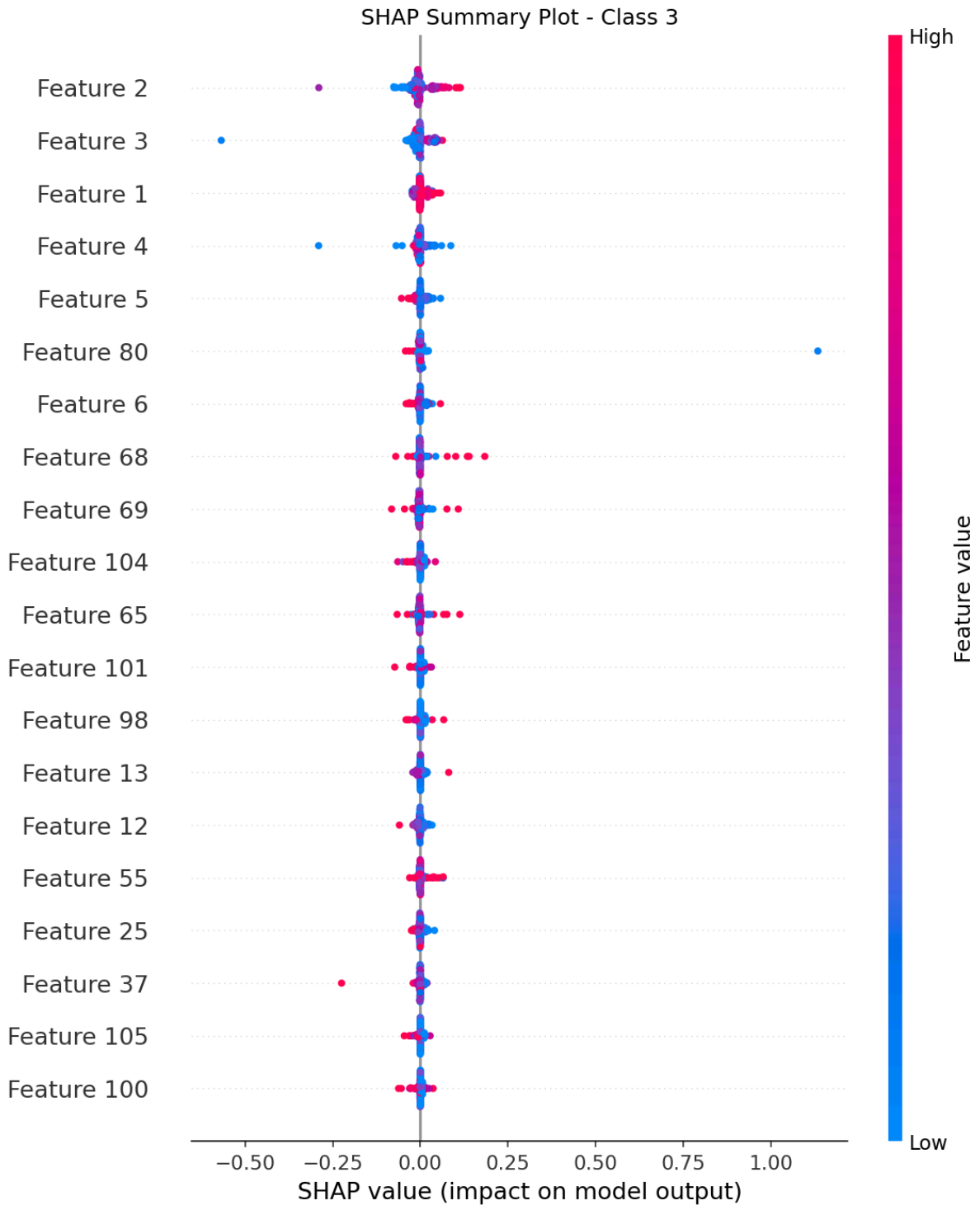


Figure 29 - Distribution of SHAP values for the 20 most important features for class 3

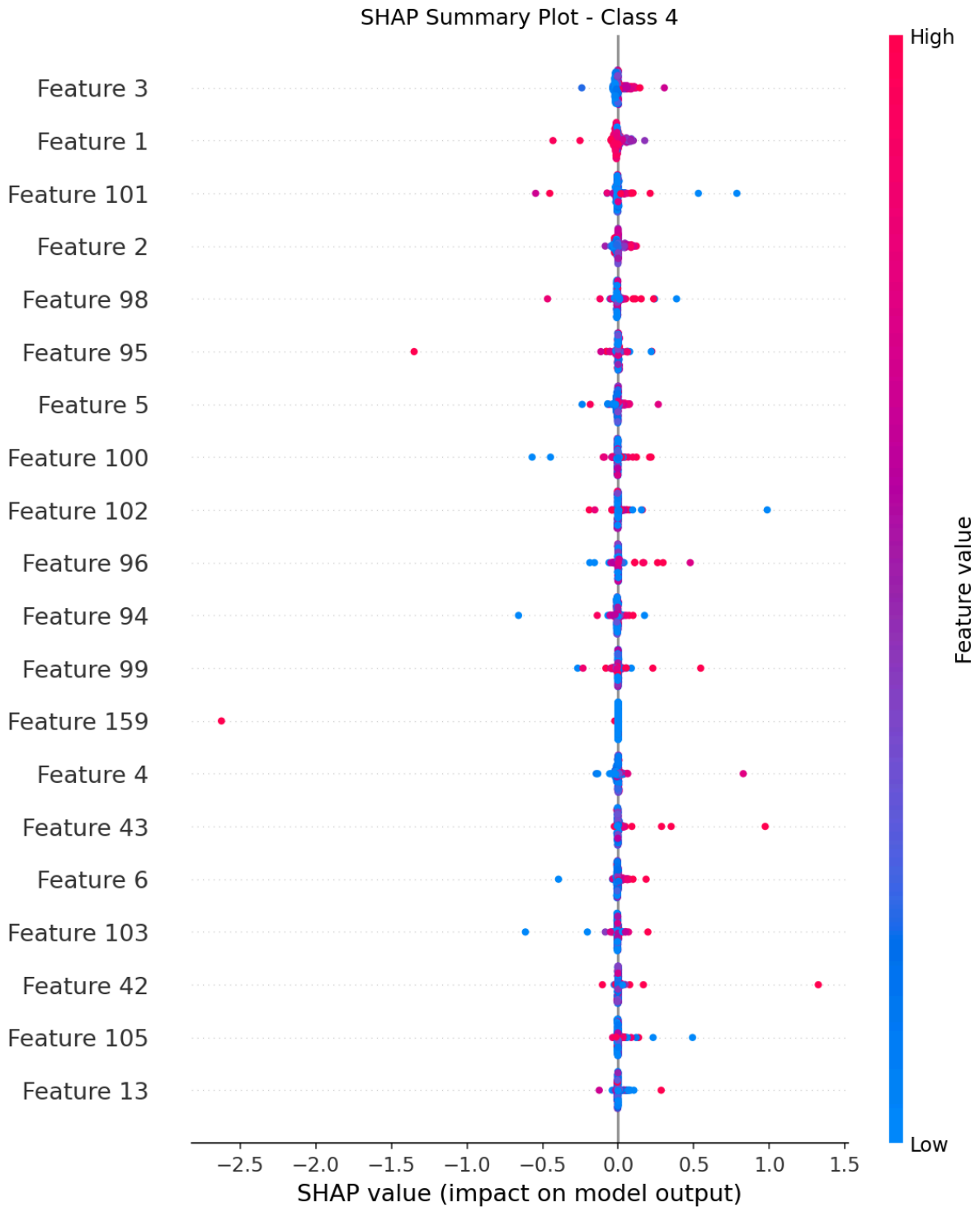


Figure 30 - Distribution of SHAP values for the 20 most important features for class 4

The next step was to have a closer look at example ECG data belonging to all classes in combination with the SHAP value for each feature to define in more detail the parts of the ECG signal that are important for the different classes.

Class 0: see Fig. 31

- For all 3 examples, the R peak around feature 70-90 is of very high importance for the model classifying those samples as class 0
- In very early timesteps, also belonging to an R peak, are very important
- Those examples confirm the previous observations

Class 1: see Fig. 32

- The distribution of high SHAP values is more spread and more complex than for class 0
- 4 important peaks are identifiable: feature 0-10, feature 25-50, 75-110, and the peak after the second R peak
    - 0-10: R peak
    - 25-50: T wave
    - 75-110: R peak
    - peak after second R peak: T wave

Class 2: see Fig. 33

- Very early features important for all 3 examples: normally position of R peak, but no R peak present -> seems to be the most important characteristic for the model
- Confirmation of previous observations

Class 3: see Fig. 34

- R peak seems to be important, but the distribution of high SHAP values is spread, complex, and reveals inconsistent signatures for the different examples
- If present, features between T and P waves are important, but their importance is lower than for the R peak
- The model seems to have exploited the zero-padded values. This suggests that the R-R interval might contribute to the identification.

Class 4: see Fig. 35

- High importance: second R peak (especially example 2 and 3) -> different location as for class 0 (here: around feature 100), important characteristic for the model to predict class 4 and to separate from class 0
- Also important: very early timesteps (especially for example 1)
- The first wave after the R peak has medium importance (more for example 1 than for 2 or 3)

Misclassification class 0/class 1: possible reasons

- For both classes, the R peaks are important, combined with overlapping locations
- But: class 0 -> clean peak, class 1 -> multiple peaks

Good performance class 2: possible reasons

- Characteristics of early timesteps are unique -> no R peak present where usually the R peak is located

Lower performance class 3: possible reasons

- Spread distribution, inconsistent signatures for different examples

With the SHAP analysis, it was possible to gain insights about the features that are important for the trained model to make a certain prediction. For each class, specific parts in the ECG signal could be identified that contribute to the prediction or prevent it. Performance differences could be explained in more detail. The importance of early timesteps and the presence/absence of the R peak were especially outstanding.

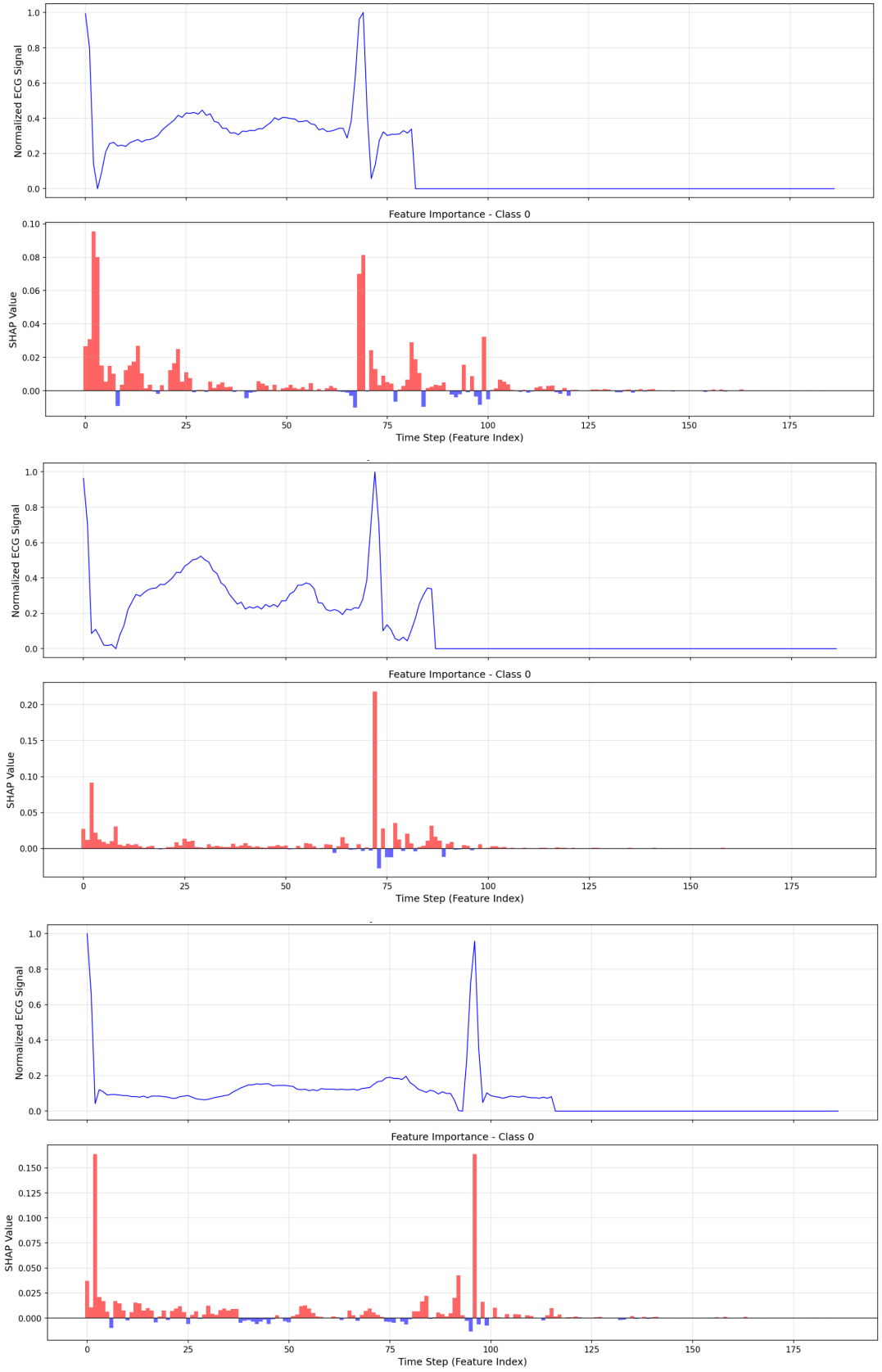


Figure 31 - ECG signal and SHAP values for true class: 0, predicted class: 0

Figure 32 - ECG signal and SHAP values for true class: 1, predicted class: 1

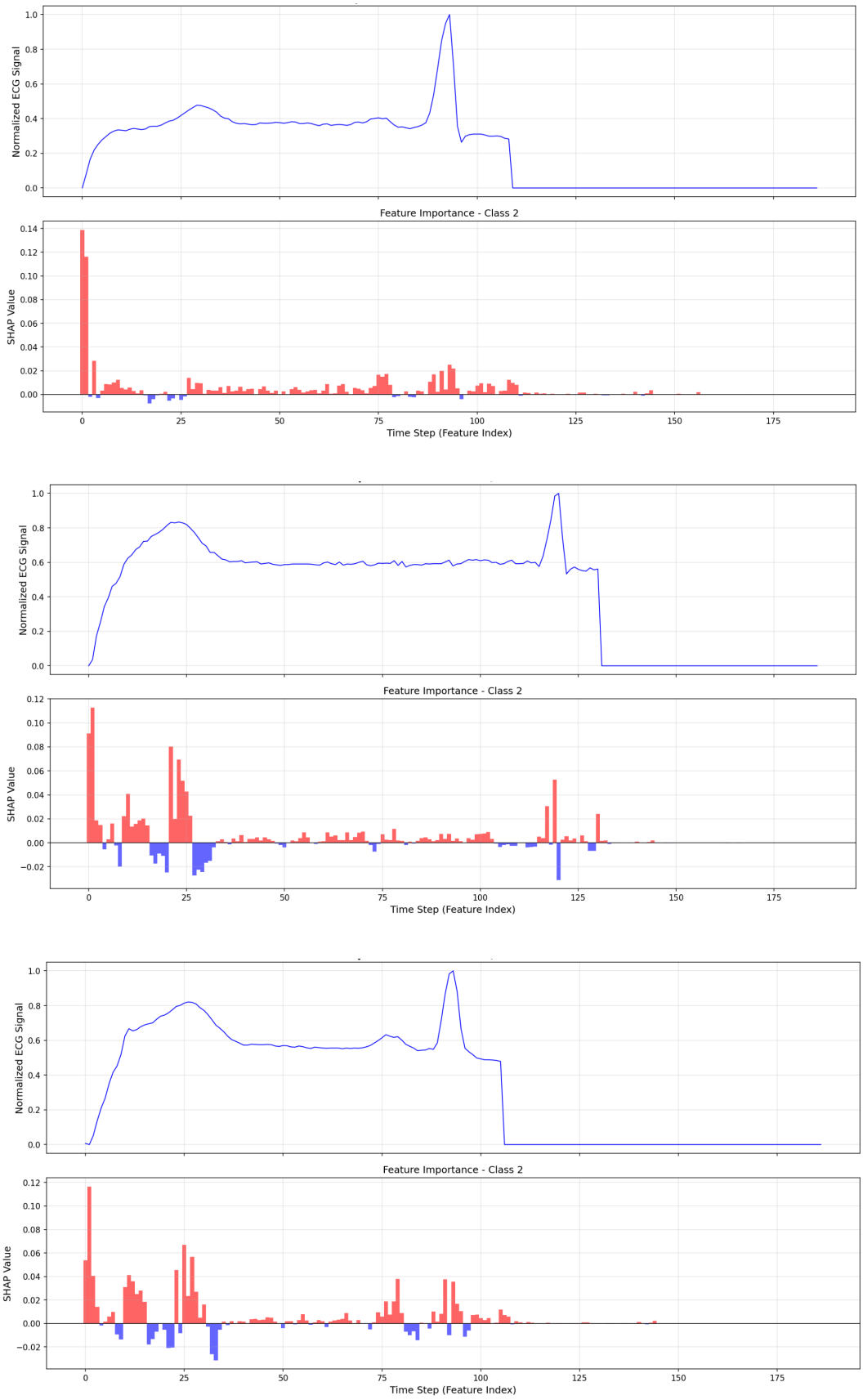


Figure 33 - ECG signal and SHAP values for true class: 2, predicted class: 2

Figure 34 - ECG signal and SHAP values for true class: 3, predicted class: 3

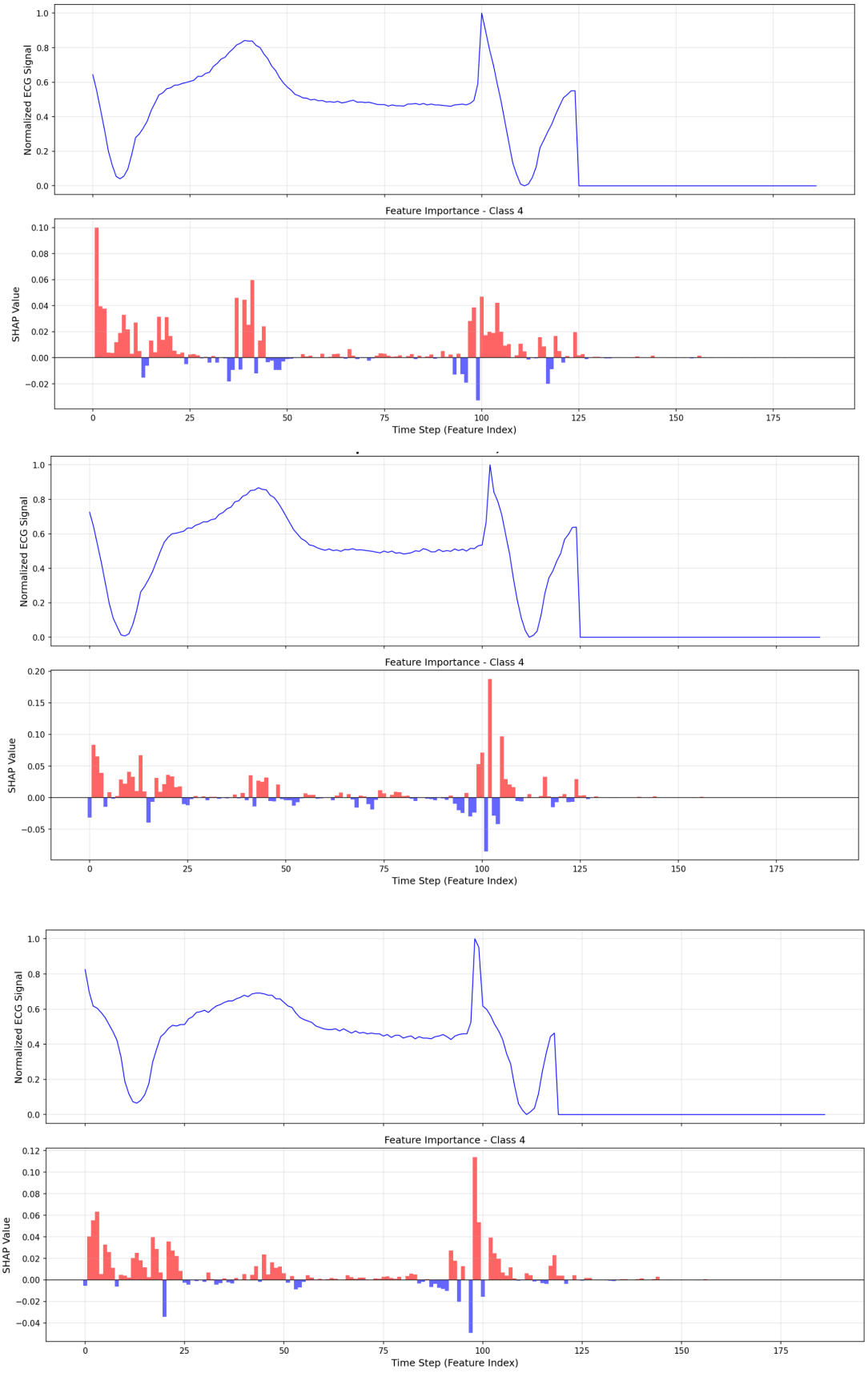


Figure 35 - ECG signal and SHAP values for true class: 4, predicted class: 4

Misclassification analysis:

In the next step, the misclassified samples were investigated using SHAP analysis to gain insights into why the model made false predictions. In general, the trained model only classified 1.49% of the test data incorrectly. Most of the misclassifications were made between class 0 and 1, followed by samples with a true label of 0, classified as class 2 or class 4. In the following, 5 different examples will be analysed (Fig. 36-40).

Examples 1 and 2 (see Fig. 36 and Fig. 37) are samples with a true label of 0, classified by the model as class 1, as this is the most common misclassification. For example 1, the prediction probability for class 1 was very high. Looking at the SHAP analysis for why the model predicted class 1, features around the second R peak strongly pushed the classification towards class 1, likely because the position of the second R peak is relatively late for class 0. Also, features at the beginning of the ECG signal and features between timesteps 50 and 110 contributed to the classification in class 1. As can be seen in the plot of evidence for true class 0, only a few red bars are present, which pushed the classification towards 0. Therefore, it is not surprising that the model predicted class 1.

For example 2 (see Fig. 37), the prediction probability for class 1 was close to 0.8. Especially high positive SHAP values at the beginning of the ECG signal pushed the classification towards class 1, in combination with high positive SHAP values around feature 35, although the present negative SHAP value around feature 110 pushed the classification towards class 0. In the plot of SHAP values for evidence for class 0, it can be observed that only a few features between timestep 90 and 100 pushed the classification towards class 0. Especially the beginning of the ECG signal pushed the classification towards class 1. In general, the ECG signal of this sample seems to be a bit noisy, and therefore, the classification could be more difficult.

Example 3 (see Fig. 38) shows a misclassification of true label class 1 classified as class 0. It can be seen that the especially features around timesteps 5, 30, and 50 pushed the classification strongly towards class 0. Also, features around timestep 100 pushed the classification towards class 0. As can be seen in the plot underneath, high positive SHAP values are mainly present for very early timesteps and around timestep 35. The other positive SHAP values have a lower amplitude. Especially features around timestep 30 and 50 pushed the classification towards class 0.

Example 4 (see Fig. 39) shows a misclassification of true label 0, classified as class 2. This ECG signal shows structures that are untypical for class 0, increasing the prediction difficulty. Especially features around the beginning of the signal pushed the classification

towards class 2. Features around the second R peak pushed the classification towards class 0. In the end, the prediction probability for class 0 was around 0.35, while for class 2 it was around 0.65. So the model only took two classes into consideration.

The same things can be mentioned, for example, 5 (see Fig. 40), showing a misclassification of true label 0 as class 4. The ECG signal does obviously not contain the typical characteristics for class 0; therefore, the misclassification is not surprising. Here, the model decided between classes 0, 2, 3, and 4. Especially strong positive SHAP values around the second R peak pushed the classification towards class 4, in combination with positive SHAP values with lower amplitude at the beginning of the signal. Also, for this example, the evidence for true class 0 is very distributed over the signal, combined with strong negative SHAP values around timesteps 90 and 135, pushing the classification towards class 4.

All in all, this misclassification analysis could give insights into why the model misclassified certain samples. For class 0, unusual ECG shapes, eventually belonging to extreme values of the R distance identified in Rendering 1, seem to make predictions more difficult. But in general, the model shows a high prediction performance on unseen test data. The analysed misclassification characteristics could therefore help to further improve the model performance in the future.

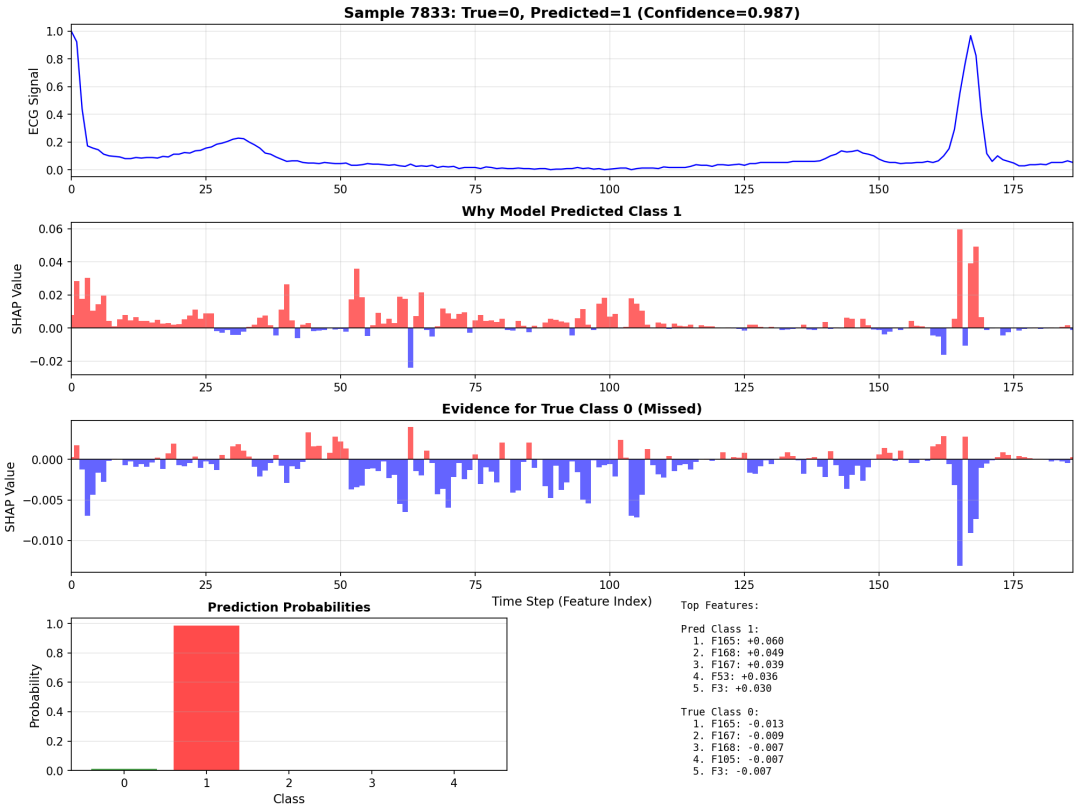


Figure 36 - ECG signal and SHAP values for true class: 0, predicted class: 1 (example 1)

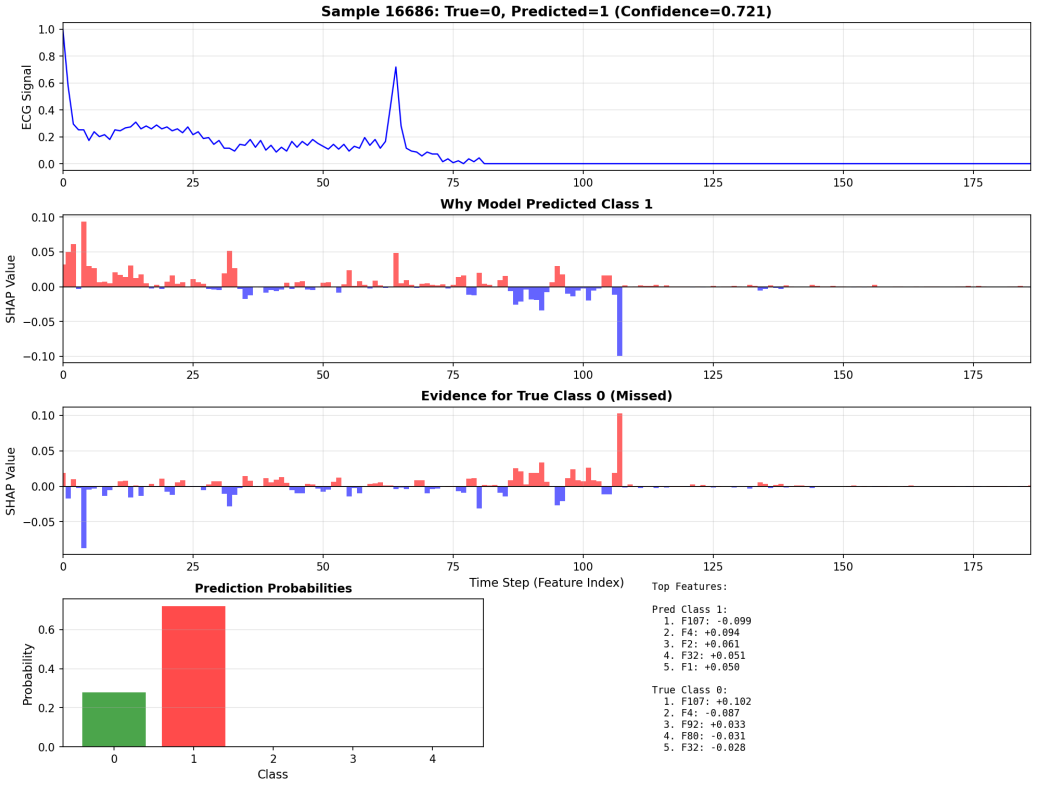


Figure 37 - ECG signal and SHAP values for true class: 0, predicted class: 1 (example 2)

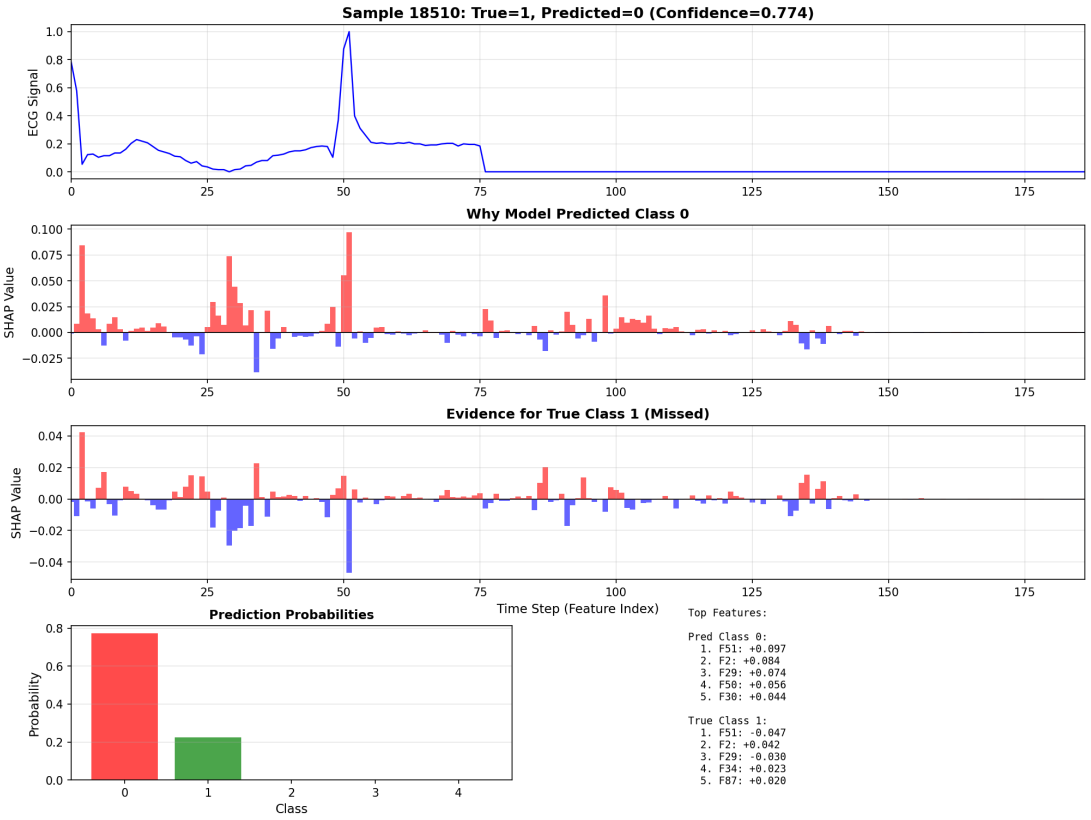


Figure 38 - ECG signal and SHAP values for true class: 1, predicted class: 0 (example 3)

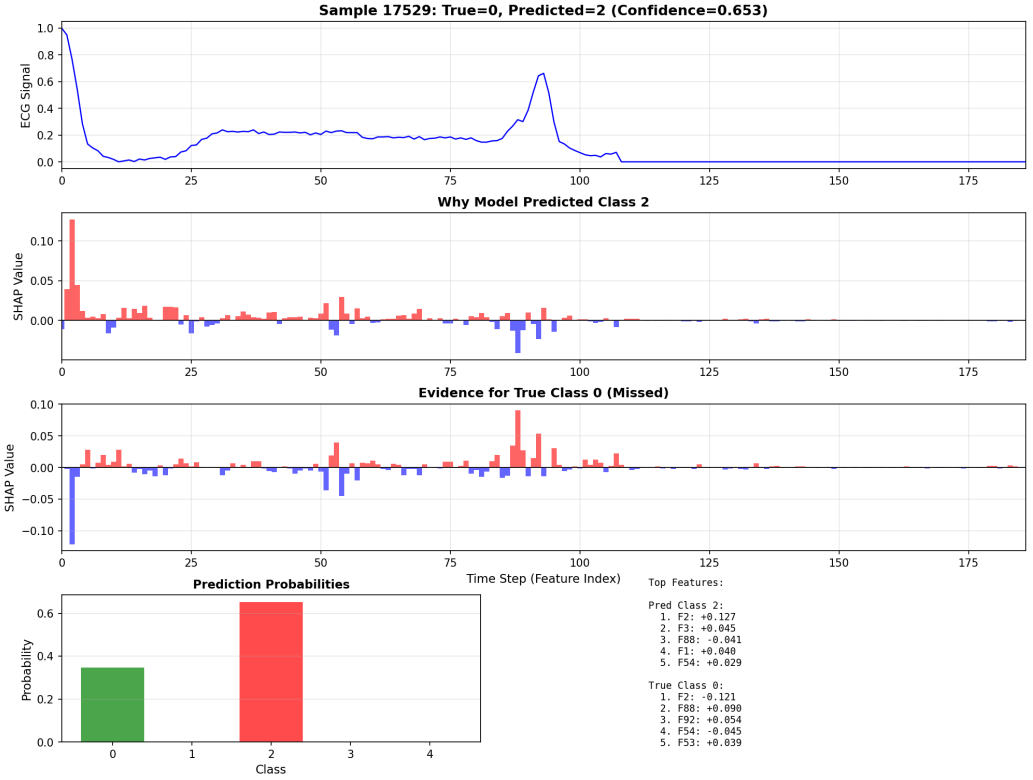


Figure 39 - ECG signal and SHAP values for true class: 0, predicted class: 2 (example 4)

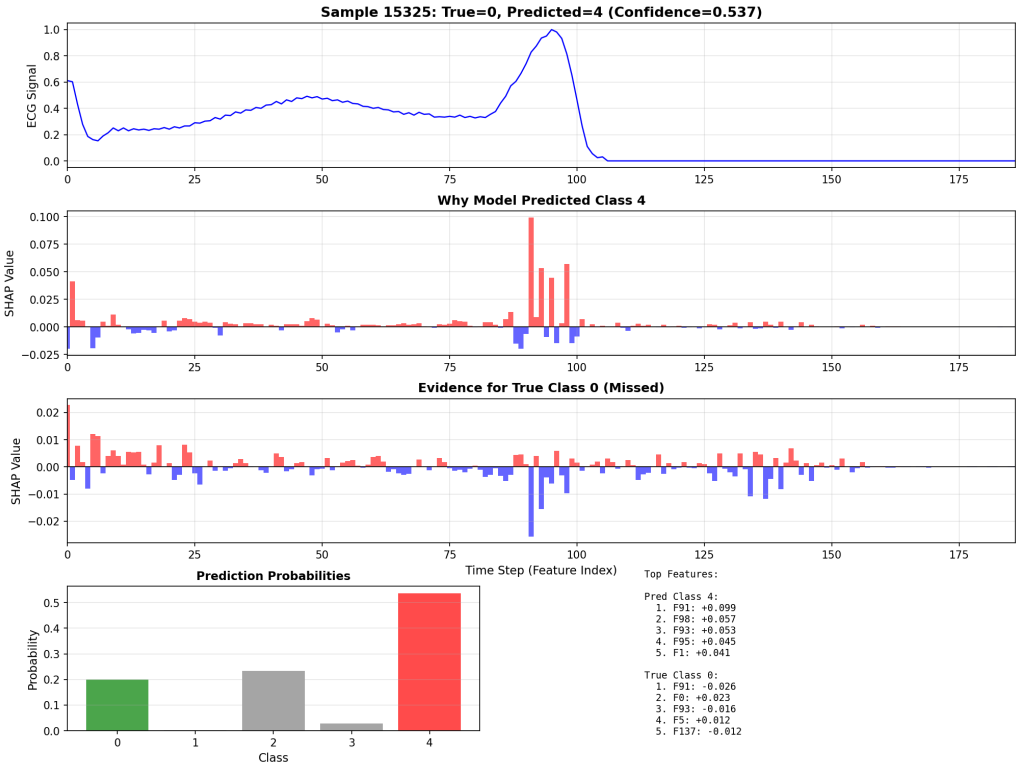


Figure 40 - ECG signal and SHAP values for true class: 0, predicted class: 4 (example 5)

SHAP DL - B) PTB dataset:

For the PTB dataset also a SHAP analysis was also conducted for the model achieving the best performance on PTB test data (CNN8 + transfer6). The analysis was conducted on 100 test samples (50 samples per class) to understand which temporal features are important for the model to predict classes. Moreover, this amount of data should be sufficient to achieve meaningful results. Furthermore, a background sample of 100 samples was generated.

Analysed samples:	100 -> 50 samples per class
Background samples:	100 -> generate SHAP baseline
Explainer:	DeepExplainer -> suitable for neural networks
Features:	187 -> time steps of ECG data

In Fig. 41, the 20 most important features for the present binary classification can be found. Feature 2 is the most important. 6 out of the 20 most important features are before timestep 20. 12 out of the 20 most important features are distributed between 20 to 40. The rest are timesteps 119 and 52. Therefore, very early and early timesteps seem to have the biggest importance for the prediction of class 0.

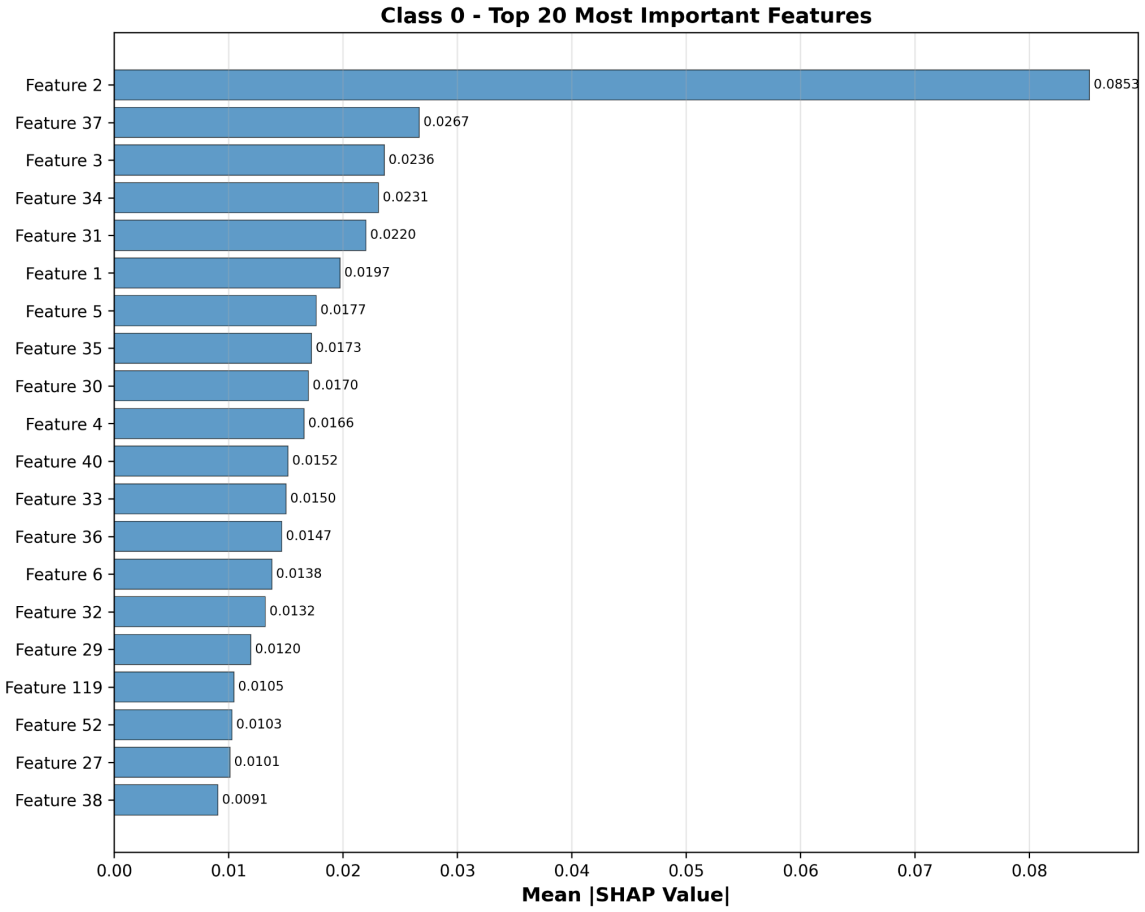


Figure 41 - 20 most important features for binary classification

Detailed analysis of SHAP values for different features (see Fig. 42):

- SHAP range: -2 to 5
- SHAP values for all features centered around 0, with extreme values
- Most of the data is spread between -0.1 and 0.1
- most important feature: 2
    - Single low value at 5 pushes classification towards class 0
- Second most important feature: 37
    - Single low value at -2 prevents classification as class 0
- Third most important feature: 3
    - Single low value at 0.8 pushes classification towards class 0

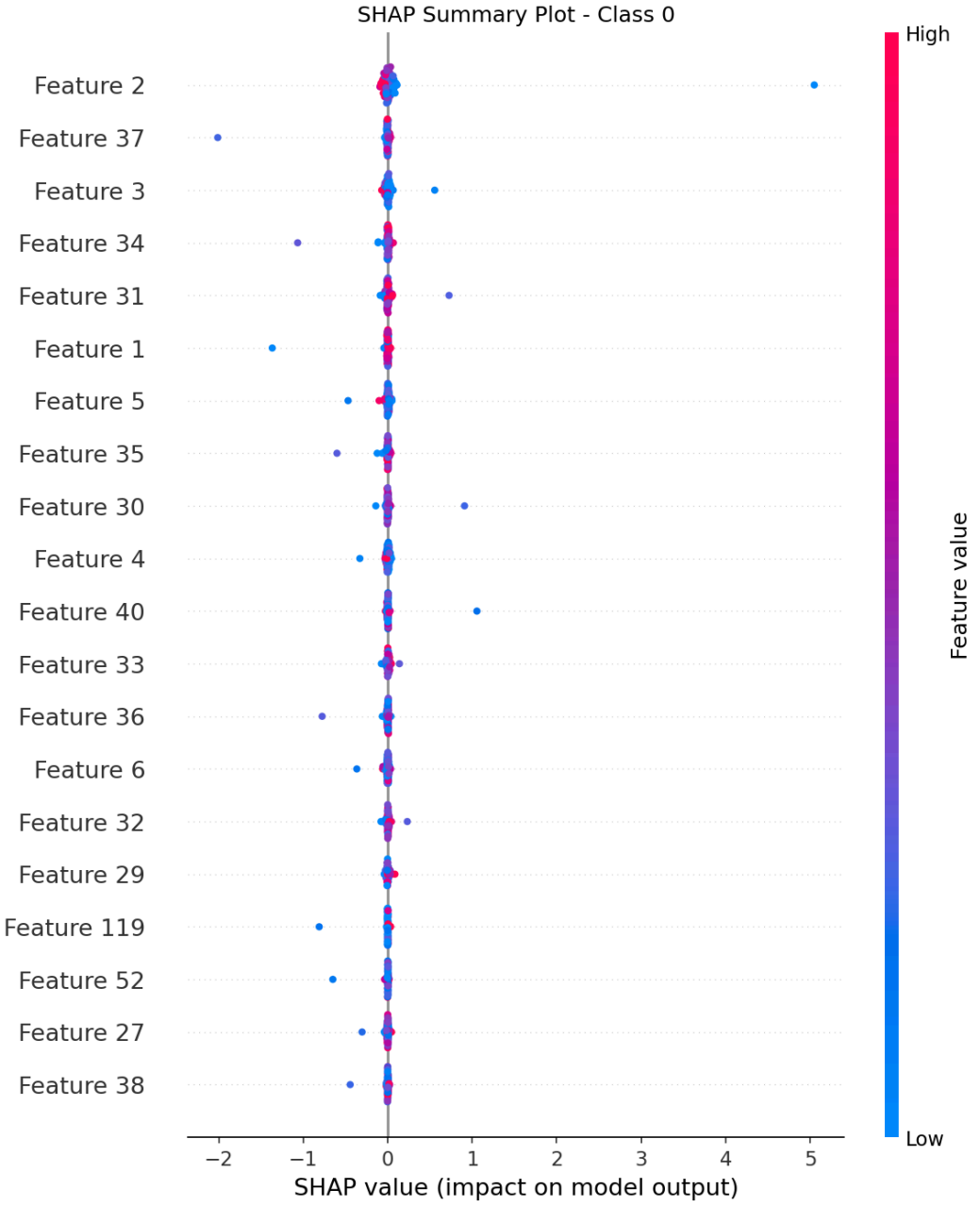


Figure 42 - Distribution of SHAP values for the 20 most important features

The next step was to have a closer look at example ECG data belonging to the two classes in combination with the SHAP value for each feature to define in more detail the parts of the ECG signal that are important for the decision of the model towards the different classes.

Class 0: see Fig. 43

- For both examples, the first and second R peaks are very important for the classification in class 0, as indicated by high SHAP values
- Also, timesteps around feature 30 are important
- For the upper example also a timestep around 155 pushed the classification strongly towards class 0
- For the lower example, high SHAP values are also distributed between the two R peaks of the signal
- Early features and R peaks have significant importance for the decision of the model with but also timesteps in between

Class 1: see Fig. 44

- The region of the second R peak and the beginning of the signal have a strong influence on the decision of the model towards class 1
- For example, high SHAP values are also distributed between the beginning and the second R peak, especially a small wave before the second R peak contributed to the decision towards class 1
- For the lower example, high SHAP values are less distributed -> two regions of high SHAP values are more separated (around timestep 20 and 50)

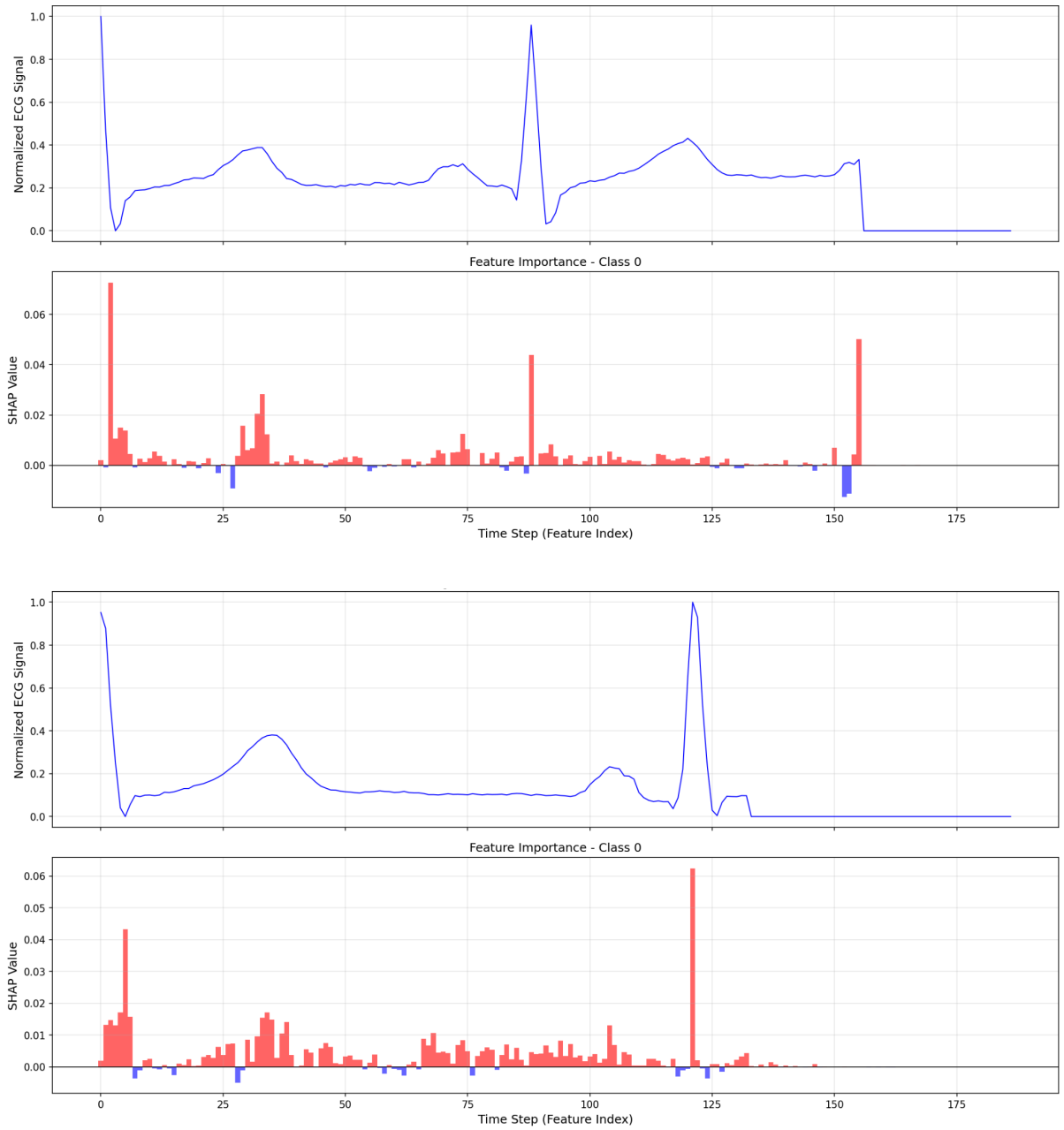
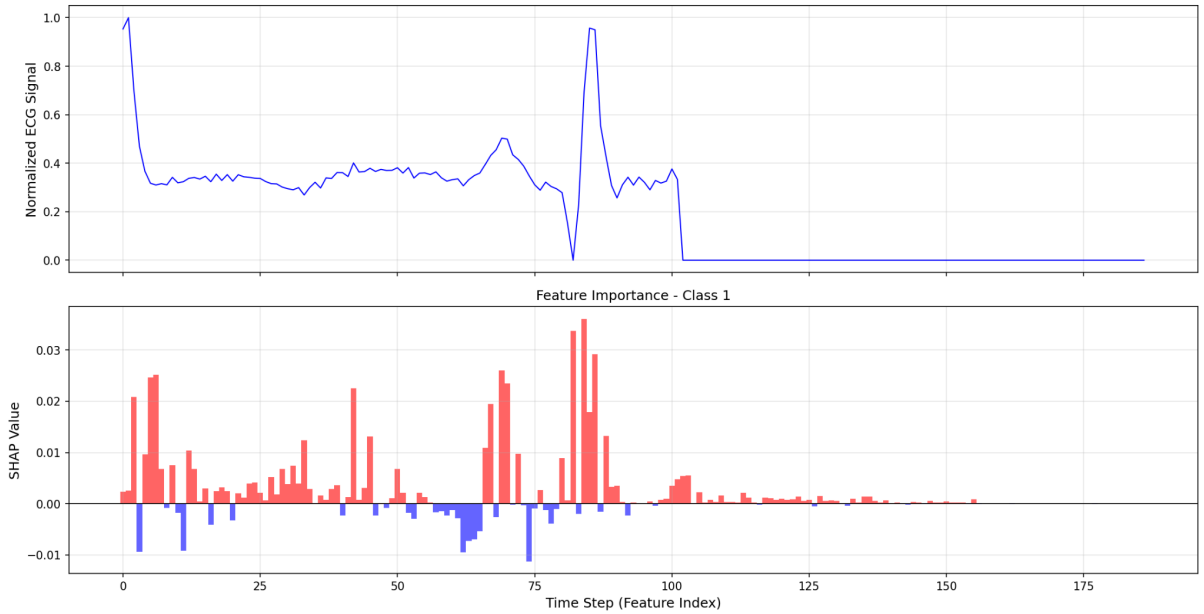


Figure 43 - ECG signal and SHAP values for true class: 0, predicted class: 0



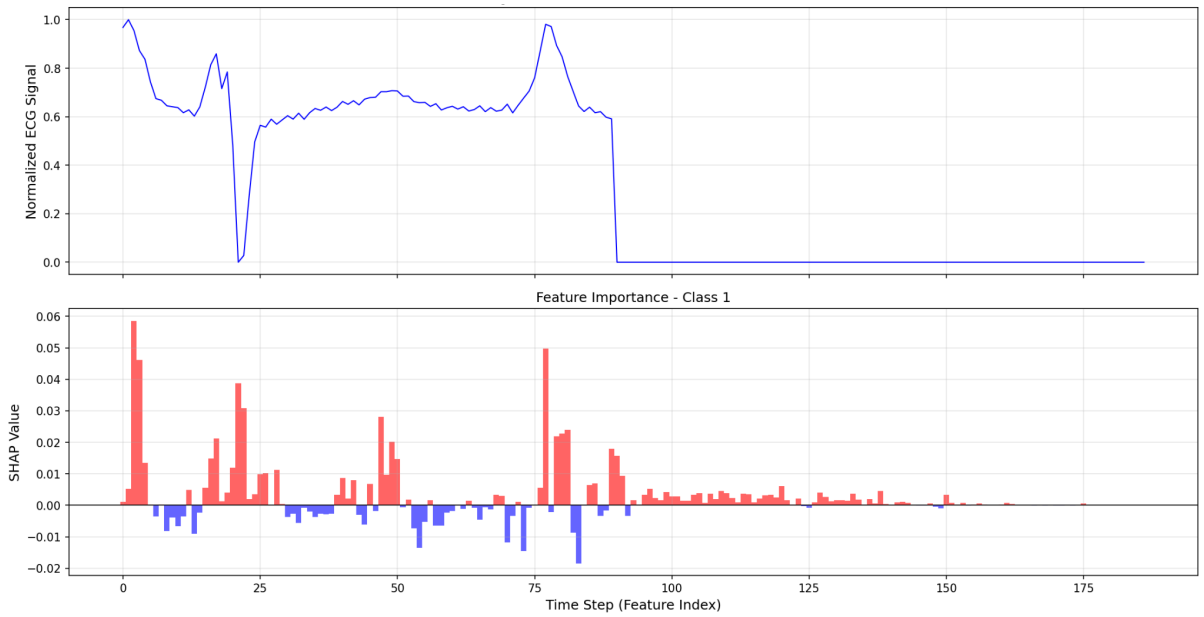


Figure 44 - ECG signal and SHAP values for true class: 1, predicted class: 1

Misclassification analysis:

In the next step, the misclassified samples were investigated using SHAP analysis to gain insights into why the model made false predictions. In general, the trained model only classified 1.6% of the test data incorrectly. Most of the misclassifications were made for samples with a true label of 1, predicted as class 0. In the following, different examples will be analysed (Fig. 45-47).

Examples 1 and 2 (see Fig. 45 and Fig. 46) are samples with true class 1, classified by the model as class 0 because this is the most common misclassification. For example 1, the beginning of the ECG signal pushed the classification strongly towards class 0. Also, features around timestep 35 and 90 pushed the classification towards class 0. For this example, the second R peak around timestep 95 pushed the classification decision towards class 0. Evidence for class 1 was very underrepresented. For the second example, features around timestep 30 pushed the prediction towards class 0 in addition to the features at the beginning of the signal. Again, evidence for class 1 was very limited.

Example 3 (see Fig. 47) shows a sample with true class 0, classified by the model as class 1. Features around timesteps 55 and 75 (second R peak) pushed the classification towards class 1. Especially features at the beginning of the signal had the opposite effect and pushed the classification towards class 0. The prediction probabilities were therefore closer together than for the other two examples of misclassifications.

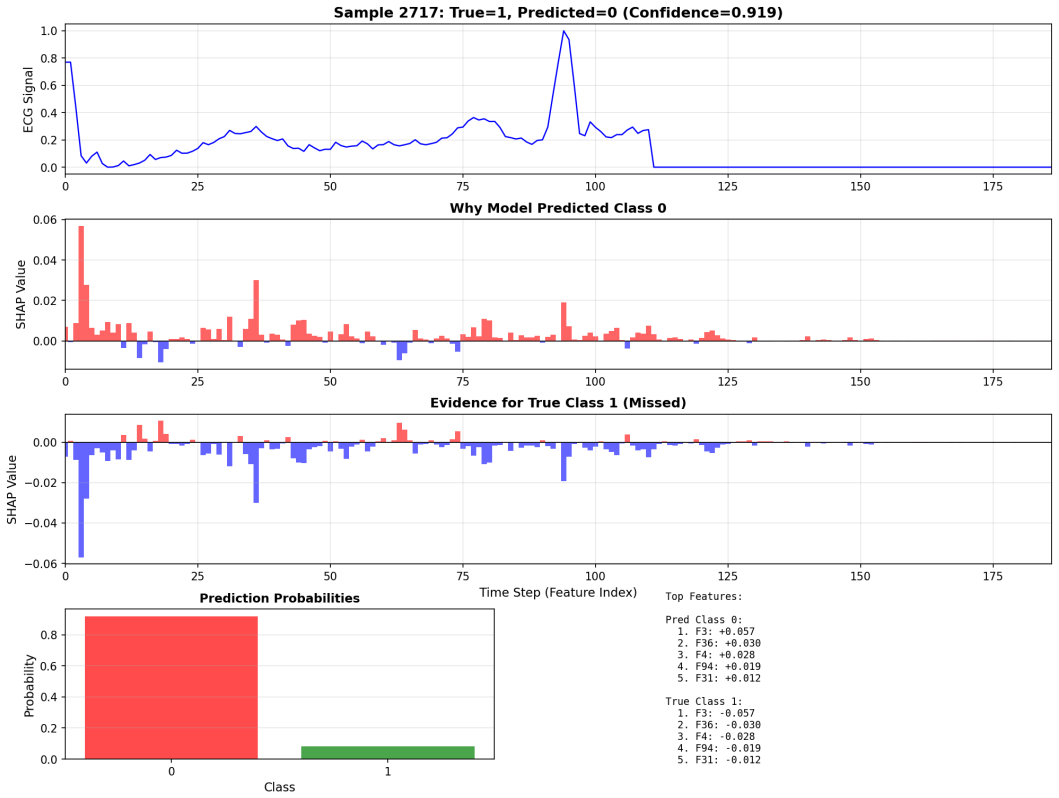


Figure 45 - ECG signal and SHAP values for true class: 1, predicted class: 0 (example 1)

Figure 46 - ECG signal and SHAP values for true class: 1, predicted class: 0 (example 2)

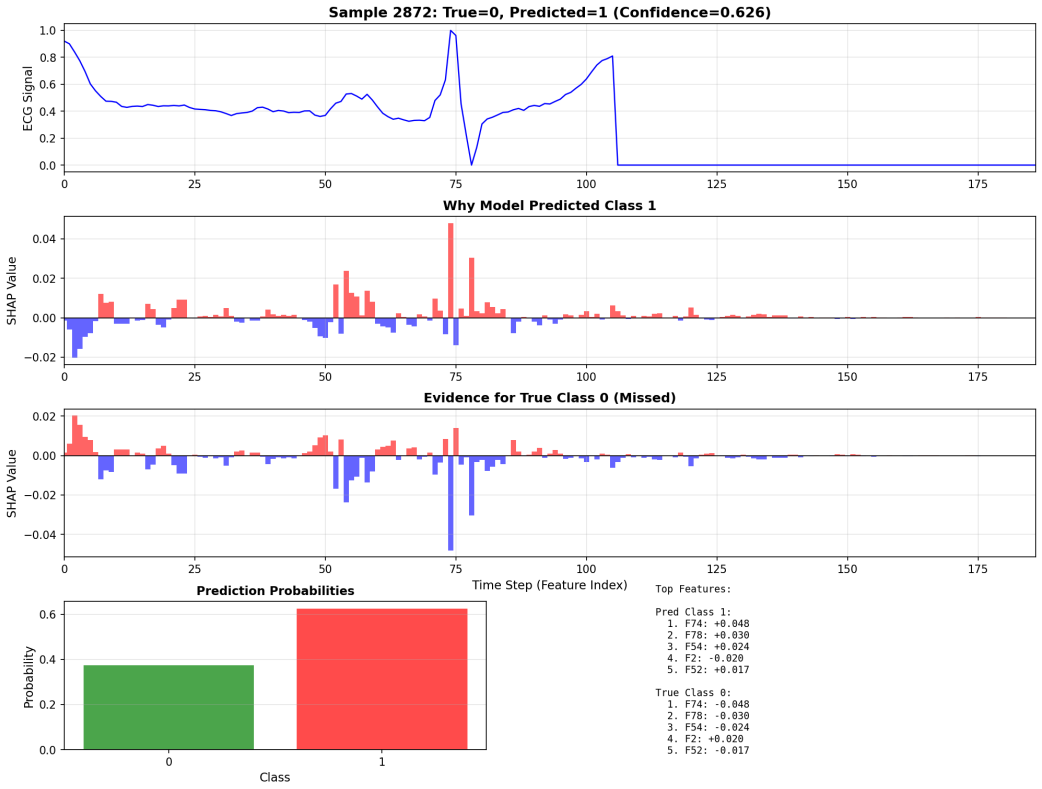


Figure 47 - ECG signal and SHAP values for true class: 0, predicted class: 1 (example 3)

- **What has (or hasn't) generated a significant improvement in your performance?**

- Arrhythmia classification:
  - Extreme value removal did not improve results for baseline or DL models
  - Filter techniques on the signal: denoising and baseline wander removal -> no improvement for baseline models
  - Using CNNs, choosing an appropriate batch size and learning rate schedule improved the performance

- MI detection:
  - Using a pretrained model (trained on MIT training data) and performing transfer learning with the last residual block unfrozen and PTB training data, finding an appropriate batch size and learning rate schedule improved the performance

Citations

[1] ECG-based heartbeat classification for arrhythmia detection: A survey; E. J. da S. Luz, W. R. Schwartz, G. Cámara-Chávez, D. Menotti (2015); *Computer Methods and Programs in Biomedicine*; doi: 10.1016/j.cmpb.2015.12.008

[2] Application of deep learning techniques for heartbeats detection using ECG signals-analysis and review; F. Murat, O. Yildirim, M, Talo, U. B. Baloglu, Y. Demir, U. R. Acharya (2020); *Computers in Biology and Medicine*; doi:10.1016/j.compbiomed.2020.103726

[3] A novel deep neural network heartbeats classifier for heart health monitoring; V. S. Sindhu, K. J. Lakshmi, A. S. Tangellamudi, K. G. Begum (2022); *International Journal of Intelligent Networks*; doi: 10.1016/j.ijin.2022.11.001

[4] ECG Heartbeat Classification: A Deep Transferable Representation; M. Kachuee, S. Fazeli, M. Sarrafzadeh (2018); *CoRR;* doi: 10.48550/arXiv.1805.00794

[5] Interpretable XGBoost-SHAP Model for Arrhythmic Heartbeat Classification; R. Xiao, M. Yang, C. Ma, L. Zhao, J. Li, C. Liu (2024); doi: 10.22489/CinC.2024.186

[6] Deep learning for ECG Arrhythmia detection and classification: an overview of progress for period 2017–2023; Y. Ansari, O. Mourad, K. Qaraqe, E. Serpedin (2023); doi: 10.3389/fphys.2023.1246746