AMLS SoSe 2025 - Alternative Exercise Report

Jakub Wladyslaw Sliwa (468617), Filip Piotr Matysik (510235)

July 15, 2025

1 Introduction

The primary objective of our participation in the alternative exercise of the *Architecture of Machine Learning Systems* module during the Summer Semester 2025 was to develop a classifier for univariate electrocardiogram (ECG) time series data. At the beginning of the semester, we agreed to divide the workload such that each team member would independently implement his own classifier. This approach enabled a comparative analysis of two distinct model architectures. Both implementations were developed using the Python programming language.

This report documents our approach to solving the assigned task and presents the results obtained. Section 1 introduces the purpose and organization of the report. Section 2 provides concise instructions for running our code, including both training and inference. In Section 3, we describe the dataset, including summary statistics, visualizations of the ECG time series, and the rationale for our custom validation split. Section 4 presents both model architectures, outlines the hyperparameter tuning process, and explains the final model selection based on the chosen evaluation metric. Section 5 discusses different data augmentation methods and shows the impact of utilizing data augmentation during training on the quality of a model. Finally, Section 6 summarizes our work within the scope of the entire project.

2 How to run our code

To successfully execute our implementation, make sure that all packages listed in the `requirements.txt` file are successfully installed in your environment. After unpacking the provided ZIP archive, please navigate to the `amls-exercise-sose-2025` directory using the terminal. From this location, the pipeline for 1D convolution model can be executed by running the following command:

```
python ./src/main.py [arg]
```

Depending on whether `[arg]` is set to `train` or `test`, the corresponding training or inference pipeline will be launched. Please note that only these two options are supported.

The `config.py` script defines all parameters that control the behavior and structure of the model, including model architecture, data augmentation settings, optimization parameters, the number of training epochs, early stopping criterion and device. It also specifies paths to input datasets and the output location for result files. By modifying the relevant fields in this file, one can easily tailor both training and inference to one's specific requirements.

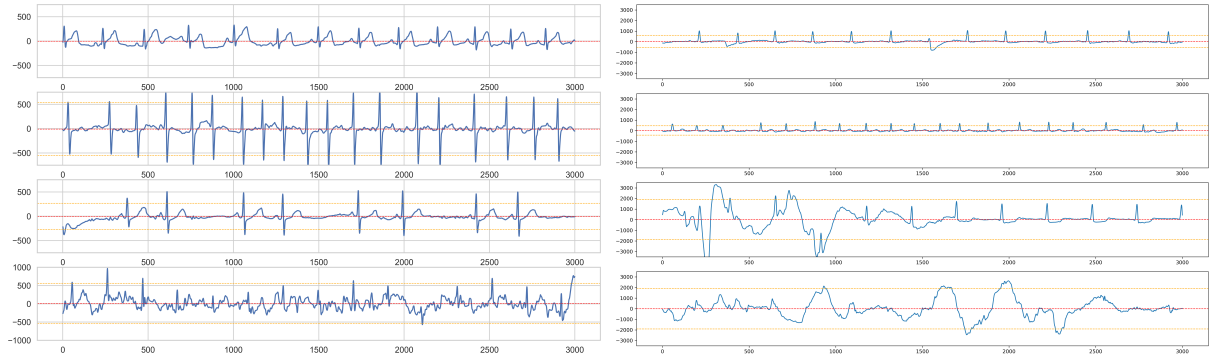
The provided ZIP archive also includes a `predictions` folder, which contains the `base.csv` and `augment.csv` files as specified in the task description. The `models` folder stores the saved states of trained models, which can be used for inference. Additionally, various results folders contain outputs from our hyperparameter tuning and data augmentation experiments. The source code is located in the `src` directory, which is organized into subfolders corresponding to different subtasks of the project.

3 Dataset Exploration

3.1 General Information

The dataset used in this project is closely related to the one originally released for the PhysioNet/Cin Challenge 2017 Clifford et al. (2017). The training set consists of 6,179 samples, while the test set comprises 2,649 observations. Labels were provided for all training samples, whereas the test set labels remained unavailable to enable final evaluation. Each observation is a time series of ECG signal measurements sampled at 300 Hz and labeled according to cardiac rhythm into one of four categories: (0) normal, (1) atrial fibrillation (AF), (2) other (neither normal nor AF) and (3) too noisy to process.

Fig. 1a displays examples of time series from each class, arranged from normal (top) to noisy (bottom). Each signal is a carefully selected segment of length 3,000, which corresponds to 10 seconds of recording. The red dashed line indicates the mean value of the entire signal, while the orange dashed line denotes its standard deviation. Notably, the class (3) example is plotted on a different scale to ensure it fits within the figure. Additionally, the class (0) example contains outlier values in the truncated portion of the signal, which causes the standard deviation line to fall outside the visible plotting area. Selected ECG time series containing such rhythm-breaking outliers are depicted in Fig. 1b. Plots are best viewed when zoomed in.



(a) Relatively clean segments

(b) Segments containing outliers, e.g. for (2)

Figure 1: Visualization of ECG time series for different classes: (0) normal, (1) AF, (2) other, (3) noisy.

3.2 Basic Summary Statistics

To gain a better understanding of the dataset and to investigate potential differences between rhythm classes, we computed several descriptive statistics for each class. This process consisted of two steps. First, we extracted the relevant statistics from each individual ECG time series. Next, we calculated the mean and standard deviation of these statistics within each rhythm category. The results are summarized in Table 1.

Metric	**Normal (0)**	**AF (1)**	**Other (2)**	**Noisy (3)**
Mean	7.58 ± 16.65	8.91 ± 19.66	6.90 ± 28.32	5.28 ± 37.29
Std dev	205.06 ± 111.38	**186.76 ± 88.24**	204.35 ± 118.67	**448.94 ± 271.88**
Median	-9.94 ± 20.57	-7.07 ± 17.79	-9.13 ± 25.76	-0.52 ± 26.01
Min	-1069.66 ± 1044.86	-913.85 ± 877.01	-1042.66 ± 1021.29	**-2304.37 ± 1581.59**
Max	1289.28 ± 1005.28	1098.65 ± 755.52	1242.79 ± 943.63	**2392.21 ± 1598.52**
Range	2358.95 ± 1785.90	**2012.49 ± 1422.67**	2285.45 ± 1716.36	**4696.58 ± 2755.19**
Skew	0.94 ± 2.36	0.99 ± 2.32	0.99 ± 2.31	**0.18 ± 2.04**
Kurtosis	14.36 ± 20.09	13.27 ± 21.97	13.77 ± 18.45	12.52 ± 20.48

Table 1: Basic summary statistics of the ECG time series.

After analysis of the descriptive statistics, it becomes evident that ECG time series labeled as (3) noisy exhibit, on average, a significantly higher standard deviation compared to the other rhythm classes. This observation is expected: while signals in classes (0), (1), and (2) typically oscillate near zero and have periodic high-amplitude spikes corresponding to heartbeats, the noisy signals also average around zero but are characterized by frequent, irregular fluctuations (see Fig. 1). As a result, at most time steps, the signal values in the noisy class deviate substantially from the near-zero mean, leading to an increased standard deviation. Consequently, the minimum and maximum values (and therefore range as well) are also typically more extreme in this class than in the others. Additionally, the noisy class is relatively easy to distinguish based on skewness. Its distribution is closer to symmetrical (skewness near zero), probably due to the presence of bidirectional high-amplitude noise. In contrast, the remaining classes generally exhibit positive skewness (around 1), as the upward spikes during the heartbeat tend to have a greater magnitude than the subsequent drop. However, it is challenging to reliably separate the remaining three classes solely on the basis of summary statistics alone. While (1) AF class shows a slightly lower standard deviation and range compared to the others, classes (0) normal and (2) other are nearly indistinguishable in terms of most statistical features. Fig. 2 presents the key insights discussed above in the form of boxplots, illustrating the distributions of the selected statistics across rhythm classes.

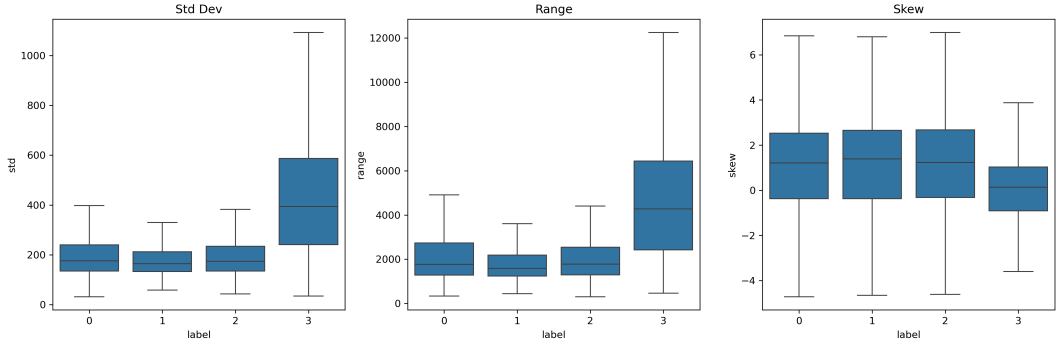


Figure 2: Boxplots of basic summary statistics - std dev, range and skewness - across rhythm classes.

3.3 Constructing Validation Split

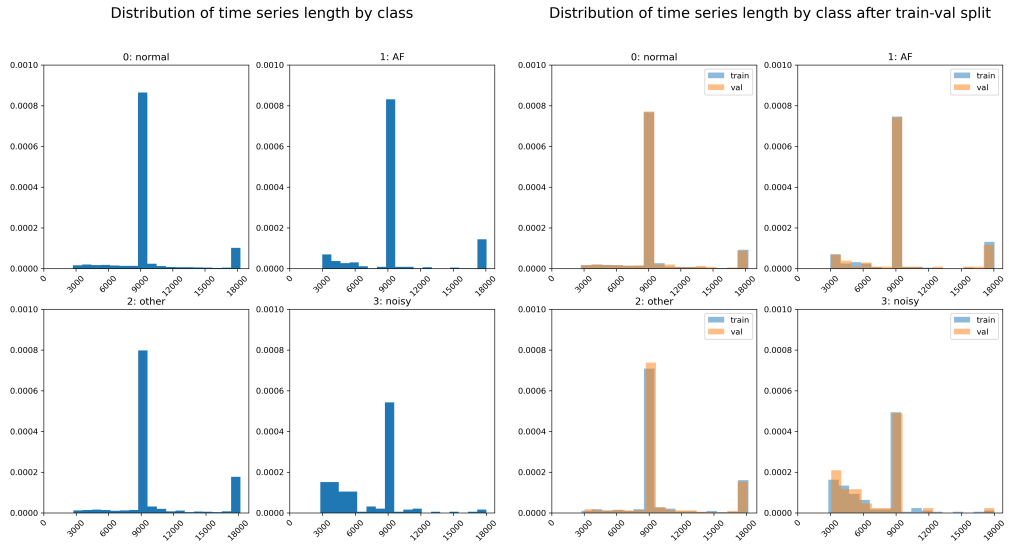
Further analysis of the training dataset revealed substantial class imbalance in the number of training samples across rhythm categories. Specifically, the number of such observations in classes (0) normal, (1) AF, (2) other, and (3) noisy was 3638, 549, 1765, and 227, respectively. Such a pronounced imbalance had to be addressed to ensure that the classifier would generalize well and not disproportionately favor the majority classes (normal and other) while neglecting those that are underrepresented (AF and noisy).

Another important challenge was the varying lengths of the individual ECG time series. Since many ML models require input data to have a fixed dimensionality, it was necessary to standardize the length of each signal. To better understand this issue and to investigate, whether time series length might serve as a discriminative feature, we computed basic summary statistics for signal length within each class, as presented in Table 2. The corresponding distributions are visualized in Fig. 3a.

Class	Mean	Std dev	Median	Min	Max
Normal (0)	9662.82	3020.08	9000	2714	18286
AF (1)	9510.86	3623.82	9000	2738	18062
Other (2)	**10366.38**	3573.79	9000	2738	18188
Noisy (3)	**7210.22**	2883.54	9000	2808	18000

Table 2: Basic summary statistics of the signal length across classes.

Analysis of the statistics above reveals that ECG time series in class (2) other tend to be longer on average, while those in class (3) noisy are generally shorter compared to the other classes. This pattern is also apparent in the corresponding histograms, where a noticeable spike on the far right indicates the longer length of the class (2) signals, and the increased density on the left reflects the shorter duration of the class (3) signals. However, most samples in all classes have a length of 9,000, corresponding to a 30-second signal duration. To address the issue of varying sequence lengths, we therefore standardized all ECG signals to a fixed length of 9,000. Time series originally shorter than this were padded with zeros, while longer sequences were truncated. Both padding and truncation were applied at the end (right side) of each time series.



(a) Before validation split (b) After validation split

Figure 3: Distribution of ECG signal length across classes.

Given the strong class imbalance in the dataset and the varying ECG signal lengths, we opted for a stratified validation split that preserves both class proportions and length distributions within each class. This approach ensures that all rhythm classes are proportionally represented in both the training and validation sets, while also maintaining the original signal length characteristics across classes.

There are two important arguments in favor of such a choice. Firstly, it helps prevent validation metrics from being biased by the dominance of majority classes, thereby providing a more accurate reflection of the model's generalization performance across all rhythm types. Secondly, it avoids introducing artificial bias by ensuring that certain signal lengths are not disproportionately represented in either the training or validation set, which could otherwise lead to spurious correlations. By stratifying the split on both class and signal length, we can more confidently assume that the training and validation sets are sampled from the same underlying data distribution. Fig. 3b illustrates the signal length distributions across classes after the stratified validation split. As shown, the distributions in both the training and validation sets closely resemble those of the original dataset.

4 Modeling and Tuning

In this section, we present two deep neural network architectures that we considered well-suited for the classification of ECG time series. The first architecture is a slight modification of the baseline model proposed in the task description, which was shown to perform effectively on this task by Zihlmann et al. (2018). The second architecture was inspired by the work of Kachuee et al. (2018), even though they originally used it on other ECG datasets: MIT-BIH and PTB Diagnostics. A detailed description of both architectures is provided below.

4.1 First Architecture - 2D Convolutions

This architecture was designed to handle 2D input in the form of the original ECG signals transformed with STFT (Short Time Fourier Transform), followed by the log-transform, into grayscale spectrograms. For such type of input different architectures were tested, starting from the one depicted in Fig. 4, which consists of four convolution blocks, each comprising six 2D convolution layers. All convolution layers first apply a set of 5×5 convolution filters, followed by batch normalization and ReLU activation. After each block, max-pooling is applied and the number of convolution filters is increased by 32. Inside the convolution blocks the spatial dimensions remain fixed, which is taken care of by padding adjustments. The output from convolution blocks is then flattened across the last two dimensions and serves as an input to the LSTM layer, followed by the fully-connected layer. Therefore, this architecture is a slight modification of the baseline proposed in the task description.

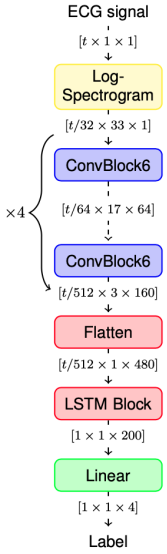


Figure 4: Zihlmann et al.

However, after observing that the model by Zihlmann et al. (2018) is too complex for the task at hand and the amount of data available, the number of convolution layers inside each block has been reduced to one, and the number of blocks itself has become a hyperparameter. Additionally, a downsampling linear layer has been introduced before the LSTM layer in order to reduce the dimensionality of the input RNN embeddings. Lastly, both 3x3 and 5x5 convolution filters have been tested, where the first one has yielded slightly better performance. Therefore, the final configuration of every convolution layer included a 3x3 filter with both padding and stride set to 1. The LSTM part consisted of two stacked layers, each with 128 hidden units. The input to the first LSTM layer comprised token embeddings of size 400.

4.2 Second Architecture - Kachuee et al.

The original architecture, depicted in Fig. 5, takes ECG signals that are normalized to zero mean and unit variance and either adjusted to a length of 9000 with padding or truncation, or left unchanged if already of the correct length. These preprocessed signals are passed through an initial convolution layer, followed by five residual blocks. Each residual block consists of two 1D convolution layers, two ReLU activations, a residual (skip) connection, and a max-pooling layer. This feature extractor is then followed by two fully connected layers and a softmax to produce class probabilities. All convolution layers apply 1D convolutions with 32 filters of size 5 and padding of 2 to preserve temporal dimensionality. The pooling layers perform max-pooling with a kernel size of 5 and a stride of 2 to progressively reduce the time dimension. The rationale behind this architecture is that the convolution layers learn shape-based features from the ECG signal, while the pooling layers reduce the temporal dimension. The fully connected layers serve as the classification head, operating on the extracted feature representations.

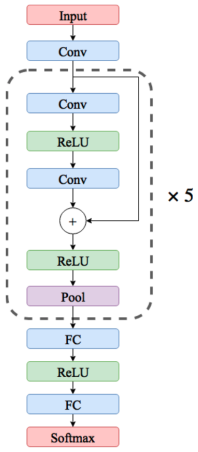


Figure 5: Kachuee et al.

In our implementation, we introduced several modifications to the original model. Firstly, we inserted an LSTM layer between the last residual block and the classification head, aiming to capture temporal dependencies that may not be fully modeled by convolutions alone. Secondly, we added a dropout layer before the last fully connected layer to mitigate overfitting. Finally, we treated several architectural components, such as the number of residual blocks (originally 5), the number of convolution filters (originally 32), and the size of the LSTM's hidden dimension, as hyperparameters to be optimized. Our goal was to explore a wider range of architectures in the hope of identifying a better classifier.

4.3 Loss Function and Evaluation Metric

For the loss function, we selected weighted cross-entropy loss, due to the highly imbalanced nature of the dataset. Ignoring this imbalance could lead the model to overfit to the majority classes ((0) normal and (2) other), resulting in poor performance on underrepresented rhythm types ((1) AF and (3) noisy). The class weights were set inversely proportional to the class frequencies in the training set, thereby penalizing misclassifications of minority classes more heavily. This encourages the model to focus adequately on all classes during training.

For model evaluation, we used the macro-averaged F1-score, which aligns with the official metric used in the PhysioNet/CinC 2017 Challenge Clifford et al. (2017). This metric is particularly well-suited for our imbalanced classification task, as it calculates the F1-score independently for each class and then averages them, giving equal weight to each class regardless of its frequency. As a result, this evaluation metric ensures that the model performs well across all rhythm types, not just the dominant ones, by equally considering both precision and recall for each class.

4.4 Hyperparameter Tuning – 2D Convolutions

Due to limited time and computational resources, a simplified hyperparameter tuning strategy was employed. Instead of training each model configuration multiple times with different random initializations, each configuration was trained only once with a unique random seed. As a result, the experiments did not share a common seed, and potential variance in performance due to initialization was not fully captured. This limitation introduces the risk of attributing strong performance to favorable initial weights rather than the configuration itself. Nevertheless, given the breadth of the hyperparameter search (approximately 70 configurations for the 2D convolution model alone), this approach was deemed sufficient to capture general performance trends. Many configurations were closely related, which increases the likelihood that any bias introduced by random initialization would be detectable by comparing similar configurations.

Under this framework, final model selection should focus not on a single configuration but rather on a set of models exhibiting consistently strong performance under similar hyperparameter settings. The objective of this tuning process was therefore to identify an optimal region of the hyperparameter space, rather than a single best configuration.

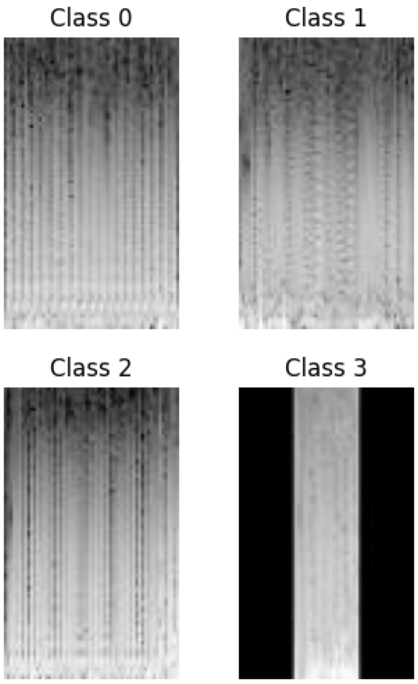
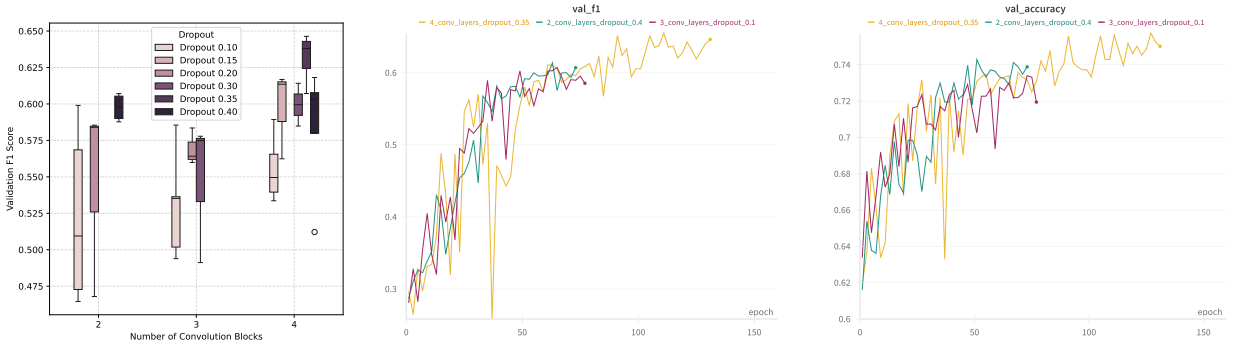


Figure 6: Visualized STFT input

Fig. 6 presents the outputs of the STFT after log-transform for each class in the dataset. The Short-Time Fourier Transform (STFT) was computed using a sampling frequency of 300 Hz, which corresponds to the original signal's rate. A Hann window of 300 samples (equivalent to 1 second) was applied to segment the signal, balancing time and frequency resolution while reducing spectral leakage. To enhance temporal continuity in the resulting spectrogram, a 50% overlap between windows (150 samples) was used. The FFT size was set to 512, providing sufficient frequency resolution for the analysis. Frequencies above 100 Hz were discarded, as they were considered less informative or potentially noisy. Additionally, the input time series data were normalized to have zero mean and unit variance, improving stability during further processing.

For most experiments, learning rate and weight decay have been set to 0.001 and 0.0001 respectively, since no large influence on the performance has been observed after changing the values of either of them. All models were trained for 200 epochs with the early stopping callback patience parameter set to 20, monitoring the validation loss. In order to find the best subsets of parameters for the 2D Convolution model, a few parameters have been compared against each other: the number of convolution blocks (2-4), number of convolution layers inside each block (1-4), probability of dropout after each convolution layer (0.1-0.4) and batch size.

Initial experiments have shown that the models were heavily overfitting (increase of validation loss past a certain epoch and F1-score in training much higher than in validation). In order to address this issue, probability of dropout has been increased and the number of convolution layers inside each block has been set to 1. Thus, from this point on, instead of referring to the number of convolution blocks, we will refer to the number of convolution layers, since it is equivalent to a convolution block with only one layer. Additionally, a learning rate scheduler has been added to decrease the learning rate when the validation loss fails to keep decreasing. To additionally avoid the model learning to predict the majority class only, a batch size of 8 has been selected as the best one. Thus, the most important parameters remaining to be tuned, were the number of convolution layers and the dropout rate. As described in section 4.1, the number of channels in the convolution layers is always increased by 32 after each layer.



(a) Num Blocks vs Dropout (b) Best Val F1 run per configuration (c) Best Val Acc run per configuration

Figure 7: Validation performance for different number of Convolution layers

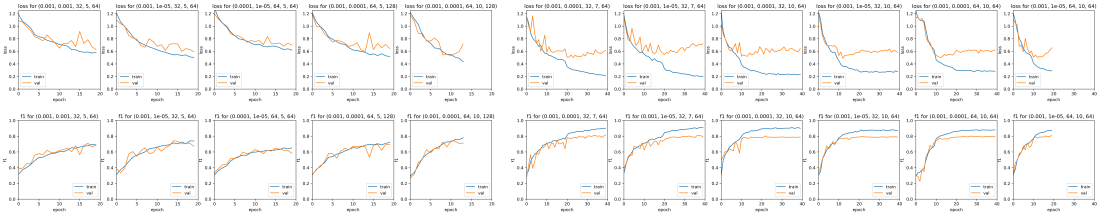
Fig. 7a presents a comparison of the F1 scores achieved on the validation set by various model configurations with respect to the different number of convolution layers and dropout probabilities. It can be observed that the models with 4 convolution layers were able to achieve generally higher F1 scores than the ones with 2 or 3 convolution layers. The highest F1 score among all models was 0.64 (4 convolution layers), and the difference between it and the best scores for 2 and 3 convolution layers setup is around 0.05 and 0.07, respectively. Moreover, the models with higher dropout probabilities tend to perform slightly better across all groups. However, most importantly, even the best results are not satisfying and strongly suggest that the model is unable to distinguish between the 4 classes well enough. As depicted in Fig. 7c, the best accuracy scores are around 0.10 higher, on average, than the best F1 score, which indicates that the model learns to classify most of the data points as the majority class.

For a clear comparison, Fig. 7b and 7c present the performance of the three best models (ranked by their validation F1 score) from each group on both the F1 score and accuracy metrics. Varying training durations are the effect of utilizing early stopping. Metrics of the models with smaller number of convolution layers start to saturate after 40 epochs, whereas for the models with 4 convolution layers 60 epochs are needed. Provided that no augmentation is applied, one should aim for the model with a dropout rate above 0.2 and 4 convolution layers.

4.5 Hyperparameter Tuning - Kachuee et al.

For the second architecture, we performed a grid search over the following hyperparameters: learning rate (`0.0001`, `0.001`, `0.01`), weight decay (`0.00001`, `0.0001`, `0.001`), number of convolution filters (`32`, `64`), number of residual blocks (`5`, `10`) and the hidden dimension size of the LSTM (`64`, `128`). Some hyperparameters were arbitrarily fixed due to limited access to computational resources, specifically the number of neurons in the fully connected layers and the dropout probability, which were set to `64` and `0.3`, respectively. Fig. 8a shows the average loss (top row) and performance metric (bottom row) after each epoch, for both training and validation sets. In this setup, all models were trained for 20 epochs, with early stopping triggered if the average validation loss increased for three consecutive epochs[1].

In general, we observed a consistent decrease in both training and validation loss, suggesting that the models were able to learn relevant patterns from the data. While we could not determine the best hyperparameter configuration at this stage, some patterns emerged. Notably, a learning rate of `0.01` was clearly too high, whereas `0.0001` was slightly too low. A weight decay of `0.001` appeared too restrictive, as shown by spikes in the loss function during later epochs. Furthermore, we observed sudden drops in validation performance when the LSTM hidden dimension was set to `128`. Additionally, there was a marked difference in training time between models with `5` and `10` residual blocks, yet we were unable to determine which configuration was optimal at that time.



(a) First grid search.

(b) Second grid search.

Figure 8: Average value of loss function and evaluation metric on both train and val sets after each epoch.

As a result, we proceeded with a second round of experiments. This time, we employed a learning rate schedule: starting with `0.001` and reducing to `0.0001` after a fixed number of epochs (25 for models with `5` residual blocks and 10 for those with `10` blocks). We also evaluated two values for weight decay (`0.00001`, `0.0001`), different numbers of kernels (`32`, `64`) and residual blocks (`5`, `7`, `10`). All models were trained for 40 epochs, again using the same early stopping criterion. Fig. 8b illustrates training dynamics for a selected subset of hyperparameter combinations.

From these experiments, we observed that models with `10` residual blocks tended to reach the minimum average validation loss more quickly than those with `5` or `7` blocks. They also exhibited more stable performance after around 10 epochs, as measure by macro-average F1-score on validation set. Slightly better results were achieved using `64` kernels and a weight decay of `0.0001`. In all configurations, we observed mild overfitting, typically starting after 20 epochs for models with `7` blocks, and as early as 10 epochs for those with `10` blocks.

4.6 Model Evaluation and Comparison

Since the 2D Convolution model suffered from heavy overfitting, mostly learned to predict the majority class and demonstrated poor generalization ability on the validation set, it has been disregarded in the following comparison. Instead, hoping that data augmentation would mitigate the overfitting problem, a group of most promising models has been selected for further training and evaluation in the augmentation part. A more detailed description of Convolutions 2D model performance can be found in section 5.1.

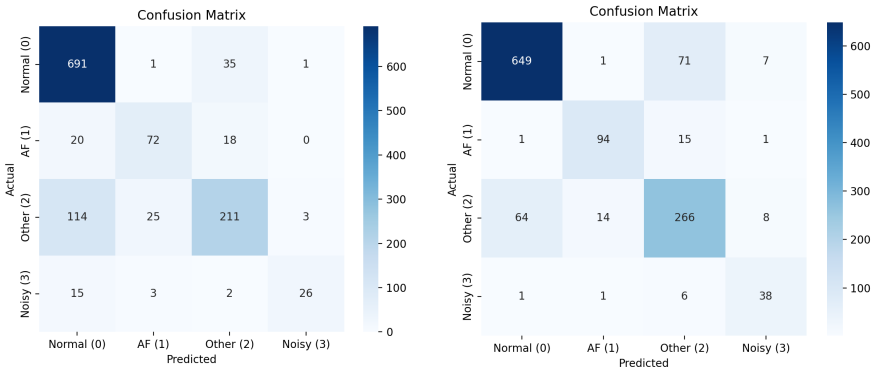
[1]The results of all the experiments can be found in JSON files residing in the `parameter_tuning_results` folder.

Finally, we report the evaluation results for the second architecture on both the training and validation sets. The performance of various hyperparameter combinations is summarized in Table 3. As highlighted, 5 out of the 12 evaluated models achieved an F1-score above 0.8 on the validation set. However, in some cases, these high scores were achieved near the end of training, when overfitting had already become apparent. Taking into account both the quantitative results presented in the table and the corresponding training visualizations in Fig. 8b, we conclude that the best-performing model is the one highlighted in bold.

weight decay, # kernels, # res blocks	**# epoch**	**F1-score (train)**	**F1-score (val)**
0.00001, 32, 5	37	0.8555	0.7849
0.00001, 32, 7	37	0.9080	**0.8077**
0.00001, 32, 10	21	0.8799	0.7951
0.00001, 64, 5	38	0.8386	0.7805
0.00001, 64, 7	33	0.8827	0.7936
0.00001, 64, 10	19	0.8718	**0.8129**
0.0001, 32, 5	28	0.8160	0.7681
0.0001, 32, 7	39	0.9109	**0.8158**
0.0001, 32, 10	22	0.9005	**0.8035**
0.0001, 64, 5	38	0.8084	0.7585
0.0001, 64, 7	37	0.8738	0.7823
0.0001, 64, 10	27	0.8763	**0.8017**

Table 3: Performance metrics for second architecture for various hyperparameter combinations.

Fig. 9 presents confusion matrices for both models. Note that since the 2D Convolution model performed rather poorly, the corresponding confusion matrix was plotted for the model trained on the augmented data.



(a) 2D Convolutions wtih augmentation (b) Kachuee et al.

Figure 9: Confusion matrices for best model of both architectures.

5 Data Augmentation

In an effort to improve our model's performance, we investigated the effect of applying additional data augmentation techniques to raw ECG signals prior to further pre-processing steps such as time series normalization (to zero mean and unit variance) and padding or truncating to a fixed length of 9000 samples. The implemented augmentation techniques included time shifting, which offsets the signal by a specified number of timesteps either to the left or right; time stretching or compression, which alters the frequency of the signal by mapping it to a longer or shorter duration; Gaussian noise injection, where noise sampled from a normal distribution is added to the signal with a standard deviation set as a fraction of the original signal's standard deviation; and random cropping, which selects a continuous segment of the signal with a length equal to a fraction of the original. We decided to refrain from using amplitude scaling, since its effect would be discarded by normalization during pre-processing step.

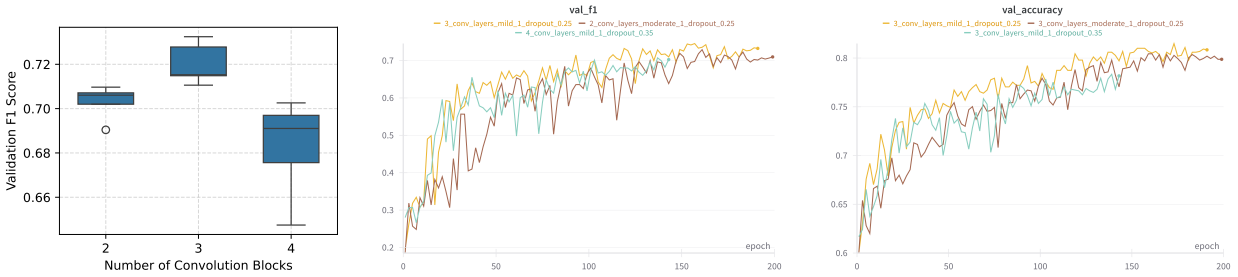
These augmentations were applied independently to each mini-batch element during training. Compared to applying augmentations to the entire dataset as a pre-processing step, this strategy introduces greater variability in the training data, resulting in a more diverse set of observations. The rationale behind this approach is to encourage the model to become invariant to minor transformations, which commonly occur in noisy real-world datasets. As a result, the model is expected to extract more robust and meaningful patterns from the underlying data distribution, ultimately improving its generalization ability.

We evaluated several augmentation strategies, guided by two defined hyperparameters: mode and number of augmentations. The mode hyperparameter defines the range of possible values for the internal parameters of each type of augmentation (e.g., the number of time steps to shift by). For instance, the “mild” mode applies rather conservative transformations that closely resemble the original signals, whereas the “high” mode allows for more pronounced modifications. The number of augmentations specifies how many distinct augmentation techniques are applied to each observation during training. For example, a setting of two may result in both random cropping and time shifting being applied to a sample. Augmentations were applied exclusively to the training data, while validation signals remained unchanged during inference.

5.1 2D convolutions

For this model both ”mild” and ”moderate” augmentation modes have been tested with either 1 or 2 transformations applied to the original signals. It is worth noting that in this case both the augmentation and computation of STFT were applied to the input data ”on the fly”, during the data loading process. Since, as shown in section 4.4, all previously tested configurations of the model performed rather poorly, we thereby explored more than just three best configurations presented in Fig. 7. However, according to the heuristic obtained earlier, we restricted the experiments to models with a dropout rate > 0.2 and three or four convolution layers. However, to fully inspect the effect of employing data augmentation, a single configuration with 2 convolution layers was also taken under consideration in experiment setup.

After observing that the mode of augmentation and number of applied transformations does not have a clear impact on the model’s performance (across the best performing models all 4 combinations of the augmentation parameters have been observed), we once again focused on the number of convolution layers. Generally, augmentation increased the F1 score and accuracy on average by 0.14 and 0.08, respectively, (0.702 vs 0.567 and 0.790 vs 0.717) compared to the models trained on the data without augmentation.



(a) Num Blocks vs Dropout (b) Best Val F1 run per configuration (c) Best Val Acc run per configuration

Figure 10: Validation performance for different number of convolution layers with data augmentation

Fig. 10a clearly demonstrates that, with data augmentation, models with three convolution layers achieve the highest validation F1 scores. However, the difference between the best-performing model in this group and the highest F1 scores from models with other convolution configurations is less than 0.04 — and even smaller between the configurations with four and two convolution layers. This suggests that the model with only two convolution layers still performs competitively well and may serve as a viable alternative to deeper models.

Table 4 presents the best 2D convolution models, both with and without data augmentation. Clearly, models trained on the augmented data performed significantly better. Notably, the best consisted of 3 convolution layers and achieved validation F1 score of 0.72 and accuracy of 0.81. This again suggests that the model classifies the majority class with higher accuracy than minority classes (this has been confirmed by analyzing individual class predictions). Finally, the obtained F1 score is lower than the one achieved by the best 1D convolution model by about 0.1, meaning that the latter is better for the classification problem at hand.

Best Models	**Augmentation**	# Epoch	**F1-score (train)**	**F1-score (val)**
4 conv layers, dropout 0.35	False	131	0.7305	0.6517
2 conv, layers, dropout, 0.4	False	73	0.8207	0.6071
3 conv layers, dropout 0.1	False	77	0.8030	0.5855
3 conv layers, dropout 0.25, mild 1	True	191	0.8234	**0.7325**
2 conv layers, dropout 0.25, moderate 1	True	199	0.7937	0.7180
4 conv layers, dropout 0.35, mild 1	True	144	0.7025	0.7030

Table 4: Performance metrics for best 2D convolution models with and without data augmentation.

5.2 Kachuee et al.

Table 5 presents the results of our experiments, and Fig. 11 illustrates the corresponding training dynamics. All experiments in this section were conducted using the best-performing model identified earlier.

mode, # augmentations	**# epoch**	**F1-score (train)**	**F1-score (val)**
mild, 1	35	0.9088	**0.8208**
mild, 2	**17**	0.8536	**0.8229**
moderate, 1	27	0.8663	0.8075
moderate, 2	23	0.8480	0.8156
high, 1	25	0.8164	0.8054
high, 2	36	0.8438	0.7947

Table 5: Performance metrics for best Kachuee et al. model found for various augmentation strategies.

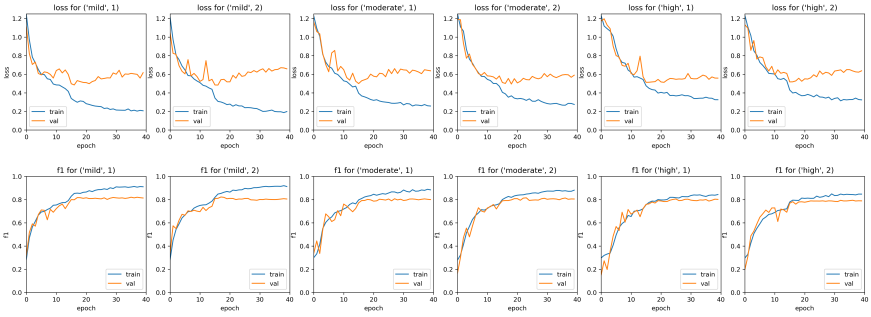


Figure 11: Average loss and performance metric on train and val sets throughout epochs.

The results illustrated in Table 5 and Fig. 11 demonstrate that the effectiveness of data augmentation strongly depends on the chosen hyperparameters. Firstly, we observe that using the 'mild' augmentation mode allows the model to exceed a macro-averaged F1 score of 0.82 on the validation set, a performance level we failed to achieve without augmentation. Secondly, while data augmentation appears to serve as a form of regularization by reducing overfitting, it does not eliminate it entirely. This is evident from the learning curves: stronger augmentation modes result in validation and training loss curves that are more closely aligned, and the same trend is observed in the corresponding F1-score curves. This suggests that more aggressive augmentations lead to models that generalize better. Thirdly, applying two augmentation techniques simultaneously tends to produce slightly better results compared to using only one. Lastly, the best model, saved at epoch 17 before significant overfitting occurred, was indeed obtained with data

augmentation. These findings support the conclusion that the utilization of data augmentation, combined with early stopping, can contribute meaningfully to improving the performance of the model.

6 Summary

Throughout this project, we developed ML pipelines for classifying raw ECG signals into four rhythm classes: (0) normal, (1) AF, (2) other and (3) noisy. We began with exploratory data analysis, during which we identified two major challenges: strong class imbalance and varying signal lengths. To address the latter, we standardized all signals to a fixed length of 9000 observations through padding or truncation. To mitigate the effect of class imbalance and reduce bias towards the majority class, we utilized a weighted cross-entropy loss function, performed a stratified validation split and evaluated model performance using the macro-averaged F1-score. Extensive hyperparameter tuning led us to our best-performing model, which demonstrated that a modified version of the architecture originally proposed by Kachuee et al. (2018) yielded superior results for our task. Furthermore, we explored the effect of data augmentation on model's ability to generalize well and its role in mitigating overfitting.

Naturally, the project has certain limitations. With more time, we would have further explored data preprocessing techniques, such as outlier or anomaly detection and signal smoothing, which could potentially enhance model performance. In terms of augmentation, introducing an additional parameter to control the probability of applying specific transformations to each ECG signal at training might improve robustness Cubuk et al. (2019). Another promising direction would be to retain and utilize information about where padding begins in each ECG time series, and feed this into the LSTM, potentially improving the model's sensitivity to signal boundaries.

Finally, we acknowledge an oversight from the early stages of our project: after initial experiments, we observed that using only the final hidden state of the LSTM resulted in poor performance, particularly within the limited number of training epochs. As a result, we instead opted to use the mean of the LSTM outputs across all time steps, the temporal average pooling, as input to the penultimate fully connected layer. Our choice was motivated by the assumption that relevant information may be distributed throughout the entire signal, rather than concentrated at the end. Although we later spotted our mistake and revisited the final hidden state approach for comparison, its performance turned out to be slightly worse and required more epochs to converge. Given that the temporal average pooling strategy yielded better results and considering the timing of this discovery near the end of the semester, we decided to retain our original approach.

Another limitation of our project, which emerged while polishing this report, is that the 2D Convolution model was trained without incorporating a weight penalty in the cross-entropy loss function, which clearly explains the observed tendency towards the majority class. Including such regularization could have significantly improved the performance of this model. However, in its present configuration, the 2D Convolution model performed significantly worse than the one inspired by Kachuee et al. (2018).

References

G. Clifford, C. Liu, B. Moody, L.-w. Lehman, I. Silva, Q. Li, A. Johnson, and R. Mark. AF classification from a short single lead ECG recording: the physionet computing in cardiology challenge 2017, 09 2017. URL `https://www.cinc.org/archives/2017/pdf/065-469.pdf`.

E. D. Cubuk, B. Zoph, D. Mane, V. Vasudevan, and Q. V. Le. Autoaugment: Learning augmentation policies from data, 2019. URL `https://arxiv.org/abs/1805.09501`.

M. Kachuee, S. Fazeli, and M. Sarrafzadeh. ECG heartbeat classification: A deep transferable representation, 04 2018. URL `https://arxiv.org/pdf/1805.00794`.

M. Zihlmann, D. Perekrestenko, and M. Tschannen. Convolutional recurrent neural networks for electrocardiogram classification, 2018. URL `https://arxiv.org/abs/1710.06122`.