AN2DL - First Challenge Report - Velocity Raptors

Tommaso Felice Banfi, Francesco Seracini, Matteo Lombardi, Riccardo Ferro

210296, 277999, 300303, 258676

November 17, 2025

1 Introduction

The problem is a ***multi-class time series classification problem***. After receiving a collection of multivariate time series, where each sequence consists of 160 time steps and contains measurements from several input channels, our goal is to train a model, using Neural Network and Deep Learning techniques, that can correctly predict the class of a new unseen sample.

2 Problem Analysis

The data set contains multivariate time series data, captured from ordinary people and pirates through repeated observations over time. Each sample collects the temporal dynamics of the body joints and pain perception, with the goal of predicting the true pain status: (***"no_pain"***), (***"low_pain"***), (***"high_pain"***).

Each record represents a time step within a subject's recording, identified by (*"sample_index"*) and (*"time"*). The data set includes several groups of features: (*"pain_survey_1"–"pain_survey_4"*), simple rule-based sensor aggregations estimating perceived pain; (*"n_legs"), ("n_hands"), ("n_eyes"*), subject characteristics; (*"joint_00"–"joint_30"*), continuous measurements of body joint angles (neck, elbow, knee, etc.) over time.

The main challenges in the problem include:

- ***Class Imbalance***: classes *("low_pain")* and *("high_pain")* are underrepresented, leading to potential biases in model training and prediction;
- ***Data Cleaning***: identify and correct errors or inconsistencies within the provided data set;
- ***Embedding***: re-map categories into a continuous space where they could capture the true meaning and relationships between data points.

After a first look at the training data we saw that the values for parameter (*"joint_30"*) were constant, so we dropped it because it would be uninformative for the task at hand.
After analysing the values for the other parameters, we divide them according to feature distribution types.

3 Method

3.1 Data Cleaning and Preprocessing

All samples were verified for invalid values (NaNs and $\pm\infty$), but no anomalies were detected. Constant features were removed as non-informative. The feature set showed heterogeneous distributions (Gaussian-like, exponential). Several transformations were evaluated (Gaussian filter, min-shift, min+log, min+asinh, Box–Cox) together with different scalers (Standard, MinMax, Robust). Gaussian-like features were left untransformed and standardized, which preserved variance and improved stability. Exponential features were best handled using a min+asinh transformation followed by MinMax scaling, effectively reducing heavy tails.

Data augmentation techniques (jittering, scaling, time-warping, random masking) were tested, but none improved generalization; the dataset size remained the dominant limitation. Additional feature engineering (rolling statistics, derivatives, signal descriptors and cyclic feature with autocorrelation) did not yield performance gains. Sequences were processed in full without windowing, a more detailed discussion about autocorrelation in Appendix A. Categorical features were encoded via one-hot encoding, while labels were mapped to {0,1,2} to preserve the inherent ordinal structure of pain levels.

3.2 Model Architecture

Time series classification involves learning patterns from sequential data to assign each sequence to predefined categories. Temporal dependencies and varying sequence lengths require specialized architectures such as recurrent networks, convolutional models, attention-based models, and hybrid approaches. We explored a variety of models from classical RNNs to modern Transformer-based and ensemble architectures.

LSTM: Bidirectional LSTM with 2-3 layers and hidden size 128-192. Dropout applied between layers. Output taken from the last timestep via fully connected layer.

GRU: Bidirectional GRU with 2-4 layers and hidden size 128-192. Layer normalization and AttentivePooling1D aggregate temporal information. Dropout applied for regularization.

Transformer: Linear projection to hidden dimension (96-192) with learnable positional encoding. 2-3 TransformerEncoder layers with multi-head attention (2-4 heads). LayerNorm and AttentivePooling1D before the output layer.

TimesNet: Stack of TimesBlocks with periodic convolutions (periods=[4,8,16,32]). Residual connections, LayerNorm, and dropout for stability. AttentivePooling1D used before output [6].

CNN1D (Inception + Dilated + SE): Residual 1D convolutional blocks with multiple kernel sizes and SEBlock1D [3] for channel-wise attention. BatchNorm1d, GELU, and dropout. AdaptiveMaxPool1d for global aggregation.

CNN + GRU: 1D CNN for local feature extraction followed by bidirectional GRU. LayerNorm, attentive pooling, and dropout applied after GRU.

TCN: Temporal convolutional blocks with multiple kernels, SEBlock1D, residual connections, and increasing dilation [4]. Global aggregation via AdaptiveMaxPool1d.

MLP: Sequence-wise mean pooling followed by fully connected layers with BatchNorm1d, ReLU, and Dropout.

GNN: Sequences treated as linear graphs with edges between timesteps. Multi-head attention in GATConv layers and AttentivePooling1D before output [1, 5].

BNN: Bidirectional GRU encoder with AttentivePooling1D. Bayesian linear layers with learnable priors and KL-annealing for uncertainty estimation [2].

Ensemble (trained): Combines CNN1D, GRU, LSTM, Transformer, and MLP models. Learnable weighted ensemble with softmax, supporting both soft predictions and hard majority voting. Provides the most robust performance by leveraging complementary strengths of individual models.

Ensemble (final): the final ensemble is obtained by merging the predictions obtained through the equally weighted majority vote from the following models: BNN, CNN1D, CNN_RNN, GNN, GRU, LSTM, MLP, and TCN.

3.3 Training Settings

Models were trained with the AdamW optimizer (weight decay 1e$-$4) and architecture-specific learning rate schedules: CosineAnnealingWarmRestarts for most models, CosineAnnealingLR for the Ensemble, and ReduceLROnPlateau for Bayesian networks. Cross-entropy loss with label smoothing (0.1) was used. Class weighting was tested but led to unstable training and lower performance. Bayesian networks included an additional KL-divergence term. Gradient clipping (max norm 5.0) stabilized training.

4 Experiments

4.1 Hyperparameter Tuning

Before selecting the final architectures, we first trained all implemented models to establish a global performance baseline. All models were trained from scratch using PyTorch, with the AdamW optimizer, batch size 64, and up to 100 epochs. Early stopping based on validation loss and a ReduceLROnPlateau scheduler were used consistently.

The tuning was performed with a lightweight grid search specific to each model family. For each configuration, we executed a full 10-fold GroupKFold cross-validation to avoid information leakage across time steps of the same sequence. As primary evaluation metric we adopt the **macro F1-score** (see Table 2), since it treats all three classes equally and is more informative in the presence of class imbalance. For completeness, we also calculate accuracy, macro precision and macro recall.

Table 1: Results of macro metrics for the best configuration of parameters across different models

Model	F1-Score	Accuracy	Precision	Recall
MLP	0.8160 ± 0.1182	0.9243 ± 0.0465	0.8545 ± 0.0846	0.8012 ± 0.1398
LSTM	0.7394 ± 0.1093	0.8925 ± 0.0448	0.7581 ± 0.1242	0.7371 ± 0.1078
GRU	0.7631 ± 0.1098	0.9001 ± 0.0436	0.7897 ± 0.0886	0.7625 ± 0.1172
CNN1D	0.7915 ± 0.0963	0.9259 ± 0.0289	0.8355 ± 0.1103	0.7764 ± 0.0912
Ensemble	0.7495 ± 0.0893	0.9031 ± 0.0306	0.7853 ± 0.0960	0.7347 ± 0.0961
GNN	0.7806 ± 0.1462	0.9106 ± 0.0536	0.8105 ± 0.1459	0.7691 ± 0.1532

4.2 Final Training Procedure

Once the optimal configuration for each promising model was identified, we trained a dedicated notebook that uses the best hyperparameters and performs full GroupKFold training with consistent preprocessing. Using the best validated configurations, we built a final **Ensemble model** combining all the other model detailed in Section 3. For each model we calculated the total F1-score, accuracy, precision and recall.

5 Results

In the table, we summarize the results obtained by the best configuration identified for each model through hyperparameter tuning. The results were computed using 7-fold cross-validation.

Table 2: Model performance

Model	F1-Score
BNN	0.8887 ± 0.1281
CNN1D	0.9169 ± 0.0621
CNN-RNN	0.9046 ± 0.1080
Ensemble (trained)	0.8837 ± 0.0594
GNN	0.9283 ± 0.0237
GRU	0.8528 ± 0.0402
LSTM	0.8444 ± 0.0882
MLP	$\mathbf{0.9046 \pm 0.0397}$
TCN	0.9283 ± 0.1024
TimesNet	0.8203 ± 0.0830
Transformer	0.8495 ± 0.0609

Using a simple MLP model ([256 $\rightarrow$ 128 $\rightarrow$ 64] hidden sizes, no dropout, and no preprocessing) as our baseline, which achieved an F1-score of 0.6059 on the test set, we obtained an improvement of **33.02%** with the **MLP** model (Kaggle submission 1).

Our Kaggle submission 2 is the Ensemble model. It is important to note that this model is the "Ensemble (final)" detailed in Section 3, which performs the majority vote among the best individually-trained models that we obtained.

6 Discussion

The proposed pipeline is robust thanks to grouped cross-validation, macro evaluation metrics, and the architectural diversity explored during tuning. The final ensemble benefits from complementary strengths of CNNs, RNNs, Transformers, and MLPs, improving stability and performance on minority classes. Several architectures proved sensitive to preprocessing and hyperparameters, and improvements observed in one model often degraded others. Moreover, the ensemble increases computational cost and complexity compared to a single strong baseline. The dataset is relatively small and imbalanced, which restricts generalization and may induce model bias, and traditional data augmentation and feature engineering did not yield improvements.

7 Conclusions

We evaluated a wide range of deep learning models for multivariate time series classification and selected the most promising ones for hyperparameter tuning and ensemble integration. The final weighted ensemble outperforms individual architectures and provides more reliable predictions across pain levels.

Future work may focus on better augmentation strategies, more systematic hyperparameter search, and exploring new architectures, but also leveraging more scientific literature.

References

[1] S. Guan, B. Zhao, Z. Dong, M. Gao, and Z. He. Gtad: Graph and temporal neural network for multivariate time series anomaly detection. *Entropy*, 24(6), 2022.

[2] K. Gundersen, G. Alendal, A. Oleynik, and N. Blaser. Binary time series classification with bayesian convolutional neural networks when monitoring for marine gas discharges. *Algorithms*, 13(6), 2020.

[3] J. Hu, L. Shen, S. Albanie, G. Sun, and E. Wu. Squeeze-and-excitation networks, 2019.

[4] B. H. D. Koh, C. L. P. Lim, H. Rahimi, W. L. Woo, and B. Gao. Deep temporal convolution network for time series classification. *Sensors*, 21(2), 2021.

[5] Y. Wang, Y. Xu, J. Yang, M. Wu, X. Li, L. Xie, and Z. Chen. Fully-connected spatial-temporal graph for multivariate time-series data, 2024.

[6] H. Wu, T. Hu, Y. Liu, H. Zhou, J. Wang, and M. Long. Timesnet: Temporal 2d-variation modeling for general time series analysis, 2023.

A Appendix Overview

This appendix provides additional material supporting the results presented in the main text. The included figures illustrate training logs, model prediction correlations, and autocorrelation analysis of selected features, helping to better understand the behavior of individual models and the ensemble.

A.1 Autocorrelation Analysis of Cyclic Features

Figure 1 presents the autocorrelation of a cyclic feature, exemplified by `joint_06`. Peaks at lag $\approx$ 40 indicate periodic patterns. A mean filter was applied to reduce Gaussian noise, which clarifies the cyclic structure.

Across several features, we identified a few dominant lags indicating potential periodicity. Based on this, we engineered additional cyclic features using sine and cosine transformations to capture continuity and phase information. However, incorporating these additional cyclic features did not lead to measurable improvements in model performance. Despite this, the analysis provided useful insight into temporal dependencies in the data, informing preprocessing decisions and feature understanding.

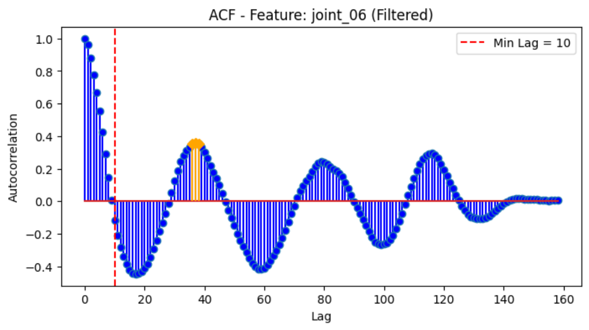

Figure 1: Autocorrelation of the cyclic feature `joint_06`. Peaks show temporal patterns, with a mean filter applied to reduce Gaussian noise. Insights from this analysis informed feature engineering attempts, including additional cyclic features, although no performance gains were observed.

A.2 Training Logs

Figures 2 and 3 show the training performance of the MLP model. Figure 2 visualizes the prediction similarity matrix on fold 7, highlighting patterns and misclassification during training. Figure 3 reports the F1-score across folds, indicating the model's stability and variability under GroupKFold cross-validation.

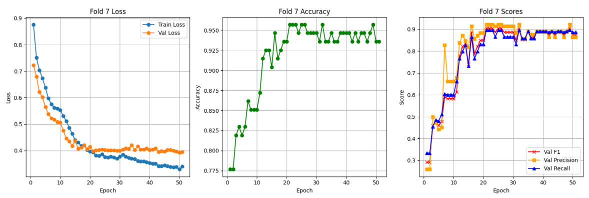

Figure 2: Prediction similarity matrix for the MLP model on fold 7. Darker colours indicate higher similarity between predicted labels, helping to identify clusters of consistent or misclassified samples.

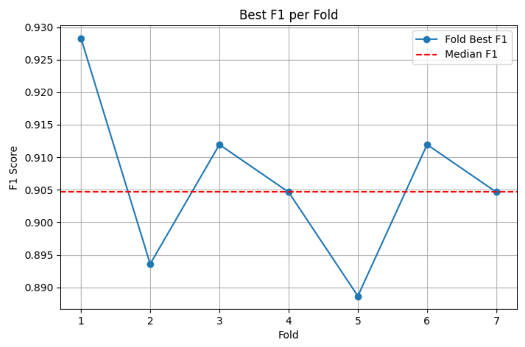

Figure 3: F1-score distribution for the MLP model across 7 cross-validation folds. The median score is highlighted, demonstrating fold-to-fold stability and variance in performance.

A.3 Model Predictions Correlation

Figure 4 shows the prediction similarity matrix across all evaluated models. High agreement regions indicate models making similar predictions, whereas low agreement points highlight complementary behavior. This information was used to design the ensemble, selecting models with complementary strengths to improve robustness.

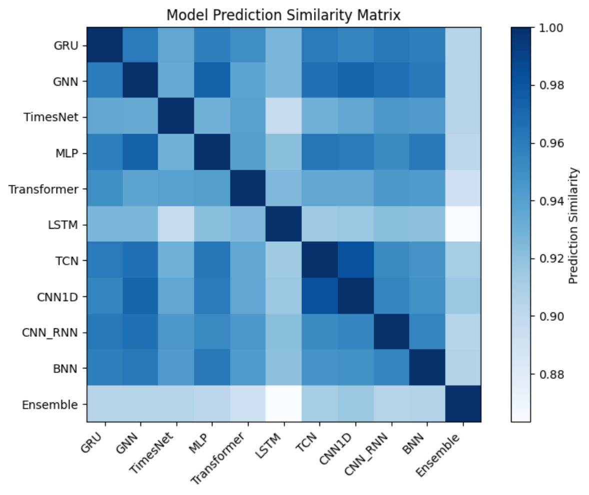

Figure 4: Prediction similarity matrix across all models on the test set. This visualization highlights correlations between models and guides ensemble construction by selecting complementary models.