# AN2DL - First Challenge Report
# ANeuronInFour

Carmen Giaccotto, Davide Bertoni, Simone Pio Bottaro, Francesco Lauria

*carmengiaccotto*, *davidebertoni*, *simonepiobottaro*, *fralauria*

286745, 300380, 286706, 252766

November 17, 2025

## 1 Introduction

This project focuses on **multivariate time series classification** and requires to develop a predictive model for pain assessment. The objective was to assign each sequence to one of three possible classes: *no_pain, low_pain*, or *high_pain*. In the following document, we will illustrate the process through which we built, optimized, and validated our model.

## 2 Problem Analysis

### 2.1 Dataset Characteristics

The **Pirate Pain Dataset** consists of multivariate time-series data, provided across three distinct files to facilitate training and inference:

- *pirate_pain_train.csv*: this file contains the primary training set.
- *pirate_pain_train_labels.csv*: this file provides the associated ground truth labels (*no_pain*, *low_pain*, *high_pain*) for the training data.
- *pirate_pain_test.csv*: this file serves as the test set for final inference.

Each patient has a sequence of 160 time steps, identified by a *sample_index*. The dataset includes 31 continuous joint-angle features and several static categorical attributes describing patient characteristics and pain surveys.

### 2.2 Data Cleaning and Feature Selection

We performed an initial **Exploratory Data Analysis** (EDA) using Python visualizations to select the most relevant features. We first discarded `joint_24` and `joint_30` as they were constant. Next, to reduce redundancy, we removed highly correlated variables, specifically discarding one feature from an highly correlated pair, like *joint_10* and *joint_11*. In order to do this we used the **Correlation Matrix** in Figure 1. Additionally, since the number of arms, legs, and eyes were perfectly collinear, we retained only one of these attributes and removed the others.

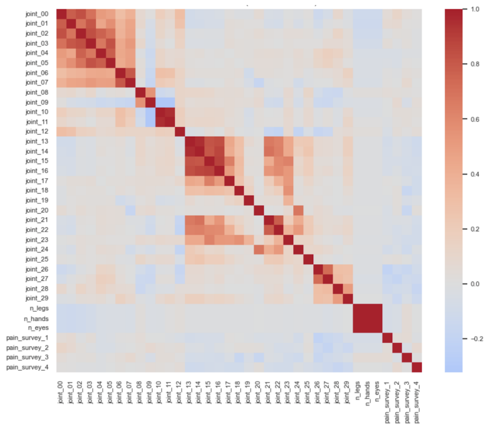

Figure 1: Correlation Matrix

### 2.3 Main Challenge

A primary challenge was the **severe class imbalance** across the three labels. To mitigate this risk, we adopted two main strategies:

- We first implemented **Stratified Cross-Validation** to maintain consistent class ratios in each split.
- Secondly, we utilized a **Weighted Cross-Entropy loss** function by assigning higher penalty weights to the rare classes *low_pain* and *high_pain*.

We ensured the model prioritized learning the clinically critical minority samples, leading to enhanced generalization performance.

## 3 Methods

First, we imported the CSV files into the program and applied type conversions to all column values. Specifically, floating-point numbers were cast to `float32`, and the string values contained in the columns `n_legs`, `n_hands`, and `n_eyes` were mapped to integer indices to enable their use within embedding layers.

Following one of the recommendations, we performed feature engineering on the timestamp column by normalizing it to the range $[0, 1]$.

Based on observations of the dataset, we separated the features into categorical and continuous groups. We defined a set of low-information columns to be removed and adapted the base functions provided in the exercises to separately handle categorical and continuous features. We customized the `RecurrentClassifier` class to process categorical features through embedding vectors, allowing the model to autonomously learn meaningful representations of these attributes.

Gradient clipping was incorporated into the `train_one_epoch` function using the appropriate PyTorch utility.

The loss function was defined as a `CrossEntropyLoss` with manually rescaled class weights and label smoothing.

The file `pirate_pain_train.csv` and the corresponding labels file `pirate_pain_labels.csv` were split for cross-validation according to the following proportions:

- Training: 75%
- Validation: 15%
- Test: 10%

We then customized the k-fold function in order to have a stratified split on the sets, using the scikit-learn function `train_test_split`.

Then the created sets are normalized using the scikit-learn `StandardScaler` object fitted only on the train set, to prevent any data leakage. The created object is then stored on disk in such a way that it can be retrieved when doing the inference.

As optimizer we used AdamW from pyTorch, with l2 lambda regularization.

We then defined the set of hyperparameters considered optimal after performing multiple Cross-Validation Grid Searches.

The model was trained using the K-Shuffle procedure, and the best-performing model was selected according to the F1 score on the validation set.

Local tests were performed on the corresponding test split, and a dedicated code section was subsequently implemented to generate model inference on the Kaggle test set.

## 4 Failed Experiments

During the model development and hyperparameter tuning phases, we explored various strategies to enhance the recurrent classifier's performance. However, some approaches did not yield the expected results:

- **Principal Component Analysis (PCA)**: applying PCA for dimensionality reduction on the continuous features did not result in a better performance.
- **1D Convolutional Layers**: this did not improve the F1 score and suggests that the prior data cleaning and feature selection were already effective, allowing the LSTM to extract optimal features directly.
- **Attention Layer**: implementing a standard Attention mechanism to selectively weigh the importance of time steps did not provide a measurable benefit.

- **Dynamic Learning Rate Scheduler**: experimenting with various learning rate schedulers failed to outperform the fixed, low *LEARNING_RATE* = 1e-4, which offered more stable convergence in our setup.

- **Outlier Filtering**: identifying and removing outliers offered no significant performance gain, suggesting that existing robust regularization was sufficient to mitigate their negative impact.

## 5 Results

This section presents the main performance metrics achieved by the optimal Bidirectional LSTM configuration across the $K = 5$ cross-validation splits. The performance was rigorously evaluated on the dedicated test sets within each split, ensuring no data leakage from the training or validation sets.

| Split Number | F1 Score On Test |
|---|---|
| 1 | 0.8889 |
| 2 | 0.9197 |
| 3 | **0.9515** |
| 4 | 0.8688 |
| 5 | 0.8822 |

Table 1: Results

The average metrics across all independent runs, including the mean and standard deviation, are presented in Table 2.

| Metric | Value |
|---|---|
| Mean Accuracy | 0.9068 ±0.0283 |
| Mean Precision | 0.9032 ±0.0320 |
| Mean Recall | 0.9068 ±0.0283 |
| Mean F1 Score | 0.9022 ±0.0297 |

Table 2: Average metrics across all splits

On Figure 2 we can see the aggregated confusion matrix on the test accross all splits.

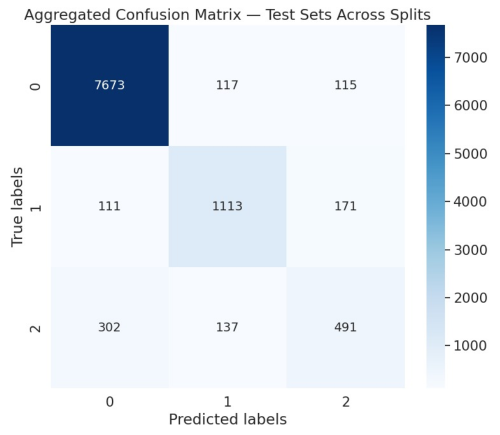

Figure 2: Aggregated Confusion Matrix

The final model configuration (Bidirectional LSTM, HIDDEN_LAYERS = 2, HIDDEN_SIZE = 128) achieved a Mean F1 score of **0.9022** across the test sets, with a low standard deviation of 0.0297. This low variance indicates that the model is robust and performs consistently across different partitions of the data.

## 6 Discussion

The model's performance is subject to several limitations and inherent assumptions. Even with weighted CrossEntropyLoss it was very difficult to well differentiate the less represented classes. The initial assumption of removing some features determined to have less information quantity, was a good choice, improving F1 score by some margin. We made the choice of setting the WINDOW_SIZE = 25 and STRIDE = 5, based on autocorrelation plots, potentially leading to information loss at the boundaries or in sequences significantly shorter than the average.

## 7 Conclusions

Our primary focus throughout the development was on achieving high generalization, reducing model complexity, rather than maximizing the validation score at all costs. We consciously chose a less complex architecture and adopted strict regularization and feature pruning to control complexity. We noticed that adding conv1D and attention layers did nothing but increase overfitting.