POLITECNICO
MILANO 1863

AN2DL - First Challenge Report
Grissinboys

Francesco Street, Francesco Urbano Sereno, Andrea Venezia,

frastreet, francescosereno, andreavenezia,

279610, 286389, 277027,

November 18, 2025

1 Introduction

This project focuses on *Time Series Classification* task on a multivariate sensor dataset, aiming to predict one of three classes using a **Recurrent Neural Network**. The adopted **LSTM-based architecture** was designed to handle *class imbalance* and *high-dimensional input features*.

2 Problem Analysis

2.1 Dataset characteristics and main challenges

The *Pirate Pain Dataset* consists of multivariate time-series data collected through repeated joint-related pain assessments for both ordinary individuals and pirates. The dataset contains no outliers or missing values and appeared to be **highly imbalanced**, with label *high_pain* appearing far more frequently in pain surveys inputs while the output labels show a predominance of label *no_pain*. Additionally, the joint measures themselves exhibit imbalance: joints 1 to 12 have comparable magnitudes, while joints 13 to 25 have values several orders of magnitude smaller and less variable. Joints 26 onward have means close to zero but show high variability. Joint 30, instead assumes a constant value of 0.5.

The main challenges concerned: (i) the *classes imbalance* of the dataset, affecting model stability; (ii) the imbalance between features scales across joints, requiring normalization; (iii) *constant or near-constant features*, that could effect negatively model performances.

2.2 Initial assumptions

It is assumed removing constant or near-constant features, such as joint 30, will not negatively impact the model's predictive capabilities.

3 Method

3.1 Data pre-processing

Pain-survey time series were merged with their corresponding labels via sample index, and categorical variables (*n_legs, n_hands, n_eyes*) were numerically encoded as `float32`. In order to fix the imbalances in the provided data, different strategies have been adopted: (i) **Constant or Near-Constant** features were removed, both for training and test datasets, by filtering ones whose **variance** wasn't above a fixed *threshold* ($1e^{-4}$). In particular, *joint 30* and *joints 13 to 25* were removed since they wouldn't provide any useful information; (ii) **Min-Max Normalization** it's been applied to *compensate scale disparities* between joints features. Using scaling parameters computed exclusively on the training set prevented data leakage and ensured that all features contributed comparably to the loss function.

The training dataset was then split into training and validation datasets with 80/20 proportionality.

3.2 Static features extraction

Features such as *n_legs, n_hands, n_eyes* remain constant across recordings. So, they were processed separately from the dynamical and used to capture subject-specific characteristics.

3.3 Sampling and loss function

Due to the *high class imbalance*, a *WeightedRandomSampler* was considered so that each mini-batch contained a more balanced distribution of classes. This improves the gradient signal from underrepresented classes and reduces bias toward the majority class.

Regarding the **loss function**, the baseline choice was the standard *Cross-Entropy (CE)*. To better address class imbalance, *Focal Loss (FL)* was also tested. FL down-weights well-classified samples and focuses learning on harder examples through a focusing parameter γ.

3.4 Model Architecture:

The model consists of a **Recurrent Neural Network (RNN)**, in particular a *Bidirectional LSTM*, which classifies the dynamic features and is able to learn temporal dependencies. The outputs of the LSTM and the static features are then processed by a Linear classifier.
Other architectures, such as *GRU (Gated Recurrent Units)* or *simple RNN*, were also tested, but they did not show better generalization or performance than the LSTM.

3.5 Regularization and Optimization

To improve generalization and prevent overfitting, several regularization techniques were employed. **Dropout** was applied to both types of layers, since it allowed to reduce overfitting by randomly deactivating neurons during training. Additionally, **L1 (Lasso) and L2 (Ridge)** penalties were tested, both individually and in combination, to constrain model weights. L1 encourages *sparsity* in the weights, while L2 limits their magnitude, preventing the model from overfitting to the training data. The **Adam optimizer** was used for parameter updates, with **early stopping** based on *validation loss* to stop training when performance plateaued, further reducing the risk of overfitting.

3.6 K-fold cross-validation

After the training, model stability was tested through **K-fold cross-validation** scheme with $K = 5$ was adopted. The dataset was partitioned into five independent folds, such that samples form the same individual would appear only once between training and validation. Each split were trained using the same settings. This approach gave a better estimation of the model's generalization.

3.7 Hyperparameter tuning

The best configuration of the model was found by performing a **grid search**, by selecting the model with the best **F1-score** between some chosen configurations. For each of the different configurations chosen, *5-fold cross-validation* was performed using the protocol described above.

4 Experiments

Different experiments were performed on various architectures, as described in table 4. Every model was trained with a *batch size* of 32, the *Adam optimizer*, a *learning rate* of $1e^{-3}$, *dropout regularization*, and *early stopping* based on the validation loss. Each experiment was repeated using the same *sequence length* (60 frames) and *stride* (20).

The first architecture considered is *LSTM64 v1* which is a regular bidirectional LSTM with 2 layers of hidden size equal to **64** units and minimal regularization. *LSTM64 v2* adds *L1 regularization* (10^{-4}) and *label smoothing* ($\alpha = 0.05$).

The "Custom Models" implement static features extraction, separating (n_hands, n_legs, n_eyes) from the dynamical features and concatenating them with the LSTM output. Nearly-static *joints (13–25)* and constant *joint 30* were removed.

Custom Model 1 uses *class weighting* and *dropout* set to **0.3**. *Custom Model 2* applies *gradient clipping*, *Label smoothing* and *class weights* scaled by the square root of their frequency. *Custom Model 3* increases the hidden size to **128** and includes *L2 regularization* ($1e^{-4}$), Random Weighted Sampling, and square-root class weights. *Custom*

Model 4 also uses 128 units and Random Weighted Sampling, combined with a custom class-weighting scheme.

The *Embedding Model* extends the Custom Model by adding an *embedding layer* for the four pain-survey features before feeding them to a 128-unit LSTM

Model	Val F1 Score	Test F1 Score
LSTM64 v1	0.9269	**0.9595**
Custom Model 1	0.9265	0.9565
Custom Model 2	0.9316	0.9534
LSTM64 v2	0.9255	0.9500
Custom Model 3	0.9311	0.9437
Embedding Model	0.9327	0.9354
Custom Model 4	0.9226	0.9338

5 Results and Discussion

Simple architectures like RNN, LSTM and GRU, bidirectional and with two layers of at most 64 neurons, often appeared to achieve the best performances with LSTM outperforming other models, but still showing some problems on the classification of the least frequent classes. As highlighted before, some features were not so informative and close to 0 so **L1-Regularization** was used to introduce *sparsity*, but this often degraded results. So, it was decided to handle these features differently, by simply deleting them through a very low variance threshold. Overfitting was always an issue, especially on class 0, so **L2-Regularization** was introduced to limit it. No negative sides were observed, therefore it was maintained as a default.

Due to the *high imbalance* in the classes distribution, *class weighting* was introduced in order to weights classes by the inverse of the frequency. Together with *label smoothing* generally it helped predicting class 1 and 2 better but overall showed minimal or negative effects. Larger models (128 neurons per layer) benefited from a custom weighting scheme, where the minority classes (1 and 2) were amplified by 1.25–1.5, and the majority class (0) was slightly reduced (factor 0.95). This improved generalization on minority classes but did not increase overall accuracy due to persistent errors in the majority class. Label smoothing of 0.05 was applied as a standard, analogous to L2 regularization, since it helped generalization.

Window size and *stride* were not modified after initial selection and hyperparameter tuning, as autocorrelation analysis confirmed the appropriateness of the chosen parameters.

Weighted Random Sampling was also extensively tested to improve the generalization of the less frequent classes, but combined with class weighting made training much more difficult and produced overall worse models. The same applied with adding an *embedding layer* before the recurrent ones, in an attempt to try to map the pain surveys in a more "readable" continuous space for the model.

6 Conclusions

Possible improvements could have been made through the analysis of the integration between the embedding layer and the LSTMs, possibly also a 1D convolutional layer (which has been implemented in the code but never really integrated in experiments). Especially the latter could have been useful to learn the "shape"/"impact" of features through time. Also, experimenting something more regarding the "timestamp" feature with different techniques (e.g. Attention) could have been useful.

From our observations, simpler models often outperformed more sophisticated and regularized ones. In fact, the highest score was obtained with the simplest model, which is quite surprising and underwhelming. We think that this may be caused by implementation issues encountered throughout our work. This is quite evident in the results obtained with the *embedding layer* on pain surveys: more often than not made actual learning quite impossible, with training F1-Scores never exceeding low values (lower than what could have been made by a model which was instructed to always guess class 0).