AN2DL - First Challenge Report
GranchiAviatori

Alberto Sposito, Lorenzo Valentini, Pedro Villarinho, Francesco Virgulti

albertosposito, tininini, gustovilla, francescovirgulti02

10808384, 10770773, 11108811, 10802921

November 17, 2025

1 Introduction

The goal of this project is to design a deep learning model capable of classifying human pain levels from biosignal recordings. The task is framed as a supervised sequence-classification problem, and the dataset consists of short temporal sequences collected from multiple sensors.

Our approach focused on understanding the data, exploring several model architectures, and testing different training strategies. The report walks through the problem setup, the methods we used, the experiments we ran, and the final results.

2 Problem Analysis

The dataset consists of short biosignal sequences, where each timestep contains a mix of categorical survey indicators and continuous joint-angle measurements:

$$\text{sample}_t = \{\text{surv}_t,\ \text{n\_legs}_t,\ \text{n\_hands}_t,\ \text{n\_eyes}_t,\ \text{joint}_t\}.$$

A preliminary data analysis revealed two key challenges:

1. **Limited dataset size** – only 661 labeled samples were available.
2. **Severe class imbalance** – with class distributions of 77% (no pain), 14% (low pain), and 9% (high pain).

These constraints heavily influenced our design decisions, motivating the use of data balancing techniques and careful model tuning to ensure generalization despite data scarcity.

3 Method

3.1 Data Preprocessing

To handle these heterogeneous signals and prepare them for sequence modeling, we applied the following preprocessing steps:

1. **Normalization and Data Mapping:** The input combines continuous joint-angle measurements and categorical labels. Continuous values are normalized to ensure numerical stability during training. Categorical variables (survey scores in $\{0, 1, 2\}$ and limb/eye indicators in $\{0, 1\}$) are non-numeric labels and are therefore mapped to dense vectors through learned embeddings:

$$k \mapsto \text{Embedding}(k) \in R^d,$$

enabling the model to capture semantic relationships and similarities between categories.

2. **Sequence Vector Construction and Temporal Modeling.** In this implementation the temporal position is treated explicitly: a `time_idx` value for each timestep is mapped to a learned time embedding time_e$_t$. To preserve the relation between a survey entry and when it was recorded, the survey embedding and the corresponding time embedding are concatenated. After preprocessing, all features at timestep t are concatenated into a single input vector:

$$x_t = \big[\,\text{num}_t,\ \text{s\_e}_t + \text{time\_e}_t,\ \text{l\_e}_t,\ \text{h\_e}_t,\ \text{e\_e}_t\,\big]$$

This explicit time embedding supplements the recurrent backbone by providing a learned positional signal that is not solely dependent on sequential recurrence.

3.2 Model Architecture

At the start of the challenge, we had a basic idea of the models covered in the lectures (RNNs, LSTMs, GRUs), but no clear sense of what kind of performance to expect. So, we wrote our code to be flexible enough to try out lots of architectures and hyperparameter combinations. After a few days of experimenting, we landed on the following setup, which worked best for us during testing:

- **Recurrent feature extraction:** We use a multi-layer bidirectional GRU[2] to process each input sequence. This lets the model capture dependencies in both forward and backward time directions. Layer Normalization and Dropout are applied to the hidden states for regularization and stability.

- **Attention layer[1]:** To help the model focus on the most relevant parts of the sequence, we apply a simple attention mechanism. A linear projection computes attention scores for each timestep, which are normalized with softmax and used to compute a weighted average of the hidden states.

- **Context vector:** The attention-weighted sum of hidden states gives a fixed-size vector that summarizes the entire sequence, emphasizing the most informative parts.

- **Classifier head:** This context vector is passed through a lightweight feed-forward network (Linear–ReLU–Dropout–Linear), which outputs the final class logits used for prediction.

4 Experiments

Throughout the project, we explored a wide range of architectural and training choices to understand what worked best for this problem. This section summarizes the main findings.

4.1 Architecture and Hyperparameters

We began by comparing different recurrent backbones (GRU, LSTM) and testing the impact of adding an attention mechanism. We also varied the number of hidden units and the number of layers.
Table 1 reports the validation F1 score for each configuration. GRU with attention consistently performed well. Surprisingly, increasing the hidden size to 128 did not help and actually hurt generalization. LSTM offered competitive performance but didn't improve over GRU in our setup.

4.2 Training Strategies

To improve robustness, we experimented with multiple training-related strategies. Results are reported in Table 2.

Table 2: Effect of training strategies on validation F1 score.

Training Technique	Val F1 Score
Baseline (CrossEntropy)	0.9271
+ K-Fold Cross Validation	**0.9441**
+ Weighted Random Sampler	**0.9526**
+ Data Augmentation	0.9379
+ Fine-tuning on Train+Val	0.9331
+ Focal Loss	0.9451

Table 2 shows how each training strategy affected validation performance. The baseline model already performed reasonably well, and only a few additions actually pushed the score further. In particular, **K-fold cross validation** and the **Weighted Random Sampler** gave clear improvements when combined with the rest of the

Table 1: Effect of architecture choices on validation F1 score.

Model Variant	Hidden Units	Attention	Val F1 Score
GRU, no attention	64	X	0.9472
GRU + Attention	64	Y	**0.9526**
GRU + Attention	128	Y	0.9347
LSTM + Attention	64	Y	0.9432
GRU + Attention	32	Y	0.9441

pipeline. The other techniques tested (data augmentation, fine-tuning, and Focal Loss) ended up hurting performance once integrated into the full setup.

5 Results

Taking these experimental findings into account, we combined the most effective elements into our final model. Our final solution is an ensemble of five bidirectional GRUs with 64 hidden units and an attention layer, trained using a WeightedRandomSampler to handle class imbalance.
Although more advanced augmentation strategies or larger ensembles could be explored, we focused on a setup that stayed stable, easy to tune, and consistently gave strong generalization.

6 Conclusions

In this project, we explored different ways to model pain intensity from biosignal sequences and built a system that performs reliably on the given dataset. Even though we aimed for something simpler at the beginning, the final solution ended up more complex than we first expected, mainly due to the challenges of class imbalance and subtle differences between pain levels.

There are several interesting directions to improve or simplify the approach. A promising idea is to split the task into two stages: first classify "pain" vs. "no pain," and then distinguish between "low pain" and "high pain." This could help the model focus on the right level of detail at each step. We could also look into smarter training strategies or lightweight forms of data augmentation to boost generalization without adding much complexity.

Overall, the project highlighted both what works well and what could be refined, leaving a clear path for future improvements.

References

[1] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *ICLR*, 2015.

[2] K. Cho et al. Learning phrase representations using rnn encoder–decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.