AN2DL - First Challenge Report
Team Name: iFormaggini

Federico Pinto, Mattia Gotti, Michele Milani, Stefano Pedretti

federicopinto02, stealthygotti, michelemilani2, stefanopedretti

273427, 271034, 276185, 273086

November 17, 2025

1 Introduction

In this first challenge, we were given a collection of multivariate time series. Each sequence consists of 160 time steps and contains measurements from several input channels. The goal of this task is to perform time series classification, which means assigning each entire sequence to one of three possible classes. To be evaluated we are expected to submit:

1. `.csv` file with test predictions, over which the F1-score will be evaluated and posted on Kaggle leaderboard
2. This work report

2 Problem Analysis

The given dataset contains multivariate time series data, captured from both ordinary folk and pirates over repeated observations in time. Each sample collects temporal dynamics of body joints and pain perception, with the goal of predicting the subject's 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 dataset includes several groups of features:

- `pain_survey_1` to `pain_survey_4` is a simple rule-based sensor aggregations estimating perceived pain
- `n_legs`, `n_hands`, `n_eyes` are subject characteristics. These were initially tricky because they were written as `one+peg_leg`, `one+hook_hand`, `one+eye_patch` and `two`.
- `joint_00`–`joint_30` are continuous measurements of body joint angles (neck, elbow, knee, etc.) across time.

The main challenge here is to develop a model that captures meaningful spatio-temporal features[2]. This means capturing long range dependencies, learn how the multivariate features interact and last but not least **prevent overfitting** as much as possible. On top of that, no pre-trained models are allowed so models are developed from scratch. Initial assumptions we have made are mainly two:

- **Normalization** of the input data it's necessary to ensure all channels contribute equally to the learning process and to stabilize the network training.
- **Evaluation optimization**, the model will be trained using a loss function like Categorical Cross-Entropy. Model selection and hyperparameter tuning will prioritize the F1-Score, official evaluation metric for the competition.

3 Method

Our final method is defined by three key areas: the model architecture, the data preprocessing pipeline that supports it, and the training strategy.

Model Architecture: Our model is a hybrid network[2]. The **static branch**[2] uses `nn.Embedding`[2] layers to learn dense vector representations for the categorical subject features, the **dynamic branch** is a four-stage pipeline:

1. A **CNN** (with parallel `Conv1d` branches and `BatchNormalization`)[2] extracts local patterns at different temporal scales (short, medium, long) simultaneously.
2. **Squeeze-and-Excitation blocks** applied after each CNN branch. This is a channel-wise attention mechanism that learns to assign an importance score to each feature map (filter). It adaptively boosts informative patterns and suppresses noise, preventing the model from focusing on irrelevant features and thus acting as an regularizer.
3. A **bidirectional GRU**[1] `'GRU'` with Bi flag=`True` then models the sequential dependencies between these extracted features.
4. A **transformer encoder** uses self-attention to weigh the relative importance of each timestep before classification. The *classifier head* concatenates the static vector and the mean-pooled dynamic vector (from the Transformer) and passes them to a final `nn.Linear` layer.

Preprocessing and Feature Engineering: We implemented a 5-Fold *stratified KFold*[1] (grouped by `sample_index`) to prevent data leakage. Based on autocorrelation analysis, we used a **short** and **overlapping** sliding window [2](`WINDOW=20`, `STRIDE=10`). This served as a crucial data augmentation step, tripling the training data. We also added cyclical **time features** (`sin/cos`) to provide explicit temporal context to the model.

Training and Regularization Strategy: We used *CrossEntropyLoss*[1] combined with class weights (to address data imbalance) and label smoothing (to prevent overconfidence). The model was trained with the `AdamW`[1] optimizer for L2 regularization. We applied **gradient clipping** to stabilize the BiGRU and used a *ReduceLROnPlateau* scheduler to adaptively lower the learning rate based on the validation F1-Score, allowing for a finer convergence.

Development Process: Our approach was a systematic process of stabilizing and enhancing a baseline hybrid architecture. We began with a simple CNN-BiGRU model, which proved **unstable** low performing and prone to **rapid overfitting**. To solve this, we first introduced *gradient clipping*, which was essential for clipping the recurrent component. To improve feature extraction and capture patterns at different temporal scales, we then replaced the simple CNN stack with a more complex one. We integrated *squeeze-and-excitation blocks* to provide channel-wise attention. To improve the temporal analysis, we added the *transformer encoder* that applies self-attention over the BiGRU's full output sequence; We also implement during the days the professor suggestions and this helped us a lot to reach a stable model.

4 Experiments

Our experimental process was iterative. We began by establishing baseline performance with simpler architectures, such as basic RNNs and CNNs. Each model was then tested to identify key weaknesses, like rapid overfitting. Our development consisted of a systematically fix of these weaknesses. We experimented with different architectural families and different hyperparameter to see which model perform better for our problem. We then focused on hybrid model by progressively integrating components and combining architecture to find the best solution. This iterative refinement allowed us to find the best compromise between complexity and performance and in a very large set of results that was so useful to take best decision in our development. Our best results are summarized in Table 1.

5 Results

In the table we show our main models journey, the relative performance of the F1-score with standard deviation and the relative Kaggle submission evaluation. The models we've found more interesting are in **bold** style.
There is an unexpected result, especially in the last models. It can be seen from the table that the last models have a F1-score on the validation test that is *lower* than the one on the test set by various percentage points. These models perform better on

Table 1: Best validation F1-score and std. Deviation for various tested models. Where you see Embedding and Window Slicing in the table, it's the first introduction of that technique. All models below the introduction ones have also that feature.

Model	**F1-score**	**Std. Deviation**	**Kaggle**
RNN	0.7571	±0.0278	0.7668
BiRNN	0.7084	±0.0226	0.7102
BiLSTM	0.8228	±0.0231	0.8129
K-Fold+BiGRU	0.9102	±0.0142	0.9025
K-Fold+CNN+RNN	0.9034	±0.0388	0.9133
K-Fold Strat+CNN+GRU+Embedding	0.9238	±0.0130	0.9327
As above but with Window Slicing	0.9305	±0.0124	0.9447
K-Fold strat multiple CNNs	**0.9096**	**±0.0174**	**0.9534**
K-Fold strat CNN+BiGRU	0.9133	±0.0171	0.9560
K-Fold strat CNN+SE+BiGRU+Transformer	**0.9330**	**±0.0143**	**0.9677**

the test set than the validation test. This is a great thing, maybe it is given by the fact that we have a **classes weights correction** and there is one class that is a little bit dominant in the test set.

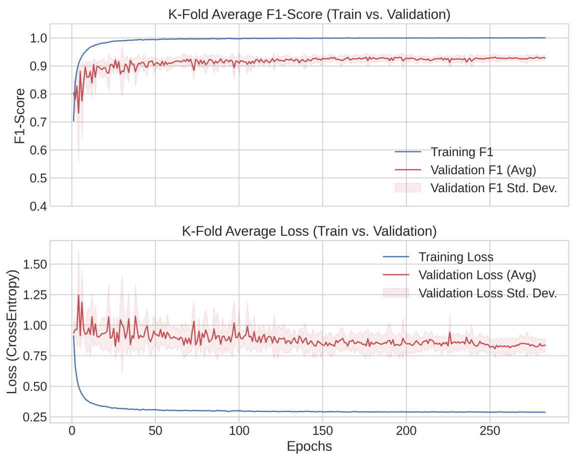

Figure 1: Average F1-Score and Loss (Train vs Validation) in our final model

6 Discussion

From the table we can clearly see an improvement going from the simpler models like RNN to the most complex ones. This can be seen both in terms of *F1-Score* and in terms of its *standard deviation*. It can be seen in the Figure1 how the F1-score in validation gets less fluttuant and more stable, even if the model overfits. This implies also that the last models are more reliable and stable over different folds. We can give the hot state that stratified K-Fold technique is a key factor for the improvement of the models. As well as stratified K-Fold we noticed the importance of the window slicing, which gave us a nice boost to models score.
Yes, the models get better **but** they also get more complicated and they take more time and resources to train. More complex models are also less easy to interpret and distribute.
A wise person we all know would say that, as in life, a trade-off is needed.

7 Conclusions

Our main goal was the design of an effective and stable architecture. The model we've introduced give us some stable and pretty effective scores over the given half of the test set. Although the model performs well, it can still be improved. As our analysis shows, it still tends to overfit the training data. For an advanced improvement we could follow two paths:

- **Hyperparameter Tuning:** A finer-grained search on regularization parameters and architectural parameters could improve the model generalization power. This, in our opinion, could be done both for the multiple CNNs model and the final high-score hybrid model.
- **Architectural Tuning:** The model's complexity (i.e. the hidden size of the GRU or the number of *transformer layers*) could be systematically reduced or *pruned* to find a better balance between model capacity and regularization, potentially reducing overfitting further.

References

[1] Eugenio Lomurno. *AN2DL exercice notebook*.

[2] Eugenio Lomurno. *LogBook challenge AN2DL*.