Time Series Classification

I. Introduction and data exploration

The challenge consisted of a Time Series Classification problem. The provided dataset included 2429 observations grouped into 12 classes, each containing 6 windows of 36 samples and associated with a label.

The first step was to examine the dataset. Observations from the 12 different classes were analysed and compared, both keeping the original structure and attempting multiple unfolding approaches. Despite this, no relevant features that could distinguish between the classes emerged. As a result, we used deep learning techniques to extract relevant information from data.

II. Pre-processing

The first hurdle of the challenge was found when we inspected the dataset's target distribution for class imbalances, since we had been provided with a highly imbalanced dataset. To leave no stone unturned, we set up the work to assess performance using both oversampling techniques (with and without augmentation) and a weighted loss function. As we already had a limited amount of data and some classes were very small, we just excluded down sampling beforehand.

Additionally, we discovered shortly thereafter that also the augmentation issue is not trivial when dealing with time series and this was especially true in our case since we did not know where the data came from. Indeed, since the nature of the observations was hidden, it was challenging to design a meaningful augmentation flow. Moreover, not knowing what data were representing, it was also tricky to estimate what transformation would affect or not the label. As a result, we tried to be as cautious as possible, and we decided to work on a metric that was reasonable for any kind of signal: frequency. In fact, we analysed data in terms of harmonics components, and we tried to exploit both the Fourier Transform and the Spectrogram of signals to perform the classification. We will address in detail this topic in the following sections.

Right after we started working on data processing. We checked for missing values, and we analysed the performance of both normalization and standardization with multiple approaches, including scaling data to unit norm, also limiting the range only to positive values, traditional standardization and the standardization technique based on quantiles. In the end, we decided not to force the dataset to a pre-determined range, but to standardize using the latter standardization technique.

Consequently, we split the dataset into train, validation and test sets, we fit a Robust Scaler on training data and then we applied it to transform all datasets. Once we had prepared our dataset, we moved to model design.

III. Handling the unbalance problem

Since our dataset was severely unbalanced, we tried several techniques to handle this problem. First, we tried using a weighted loss function to give more relevance to the less represented classes during training. Then we attempted to balance the training dataset by replicating the less numerous classes. However, both techniques seemed to be not useful to solve our problem since we obtained worse results. Finally, we tried to generate new data to enlarge the dataset using a Generative Adversarial Network. By training it on the original time series data, we were able to generate additional synthetic samples that we could use to augment the training set for our classification model. By doing so, we were able to significantly reduce overfitting on the training set. Nevertheless, in terms of classification results, we didn’t get a significant improvement overall, but it might be the case that we were not able to fully exploit the capabilities of GANs.

IV. Models

Having tried to address the problem from different perspectives using several models implementing completely different techniques, we cannot define a general approach that was applied to all cases.

However, as a general principle we went through each path at first designing the model and a suitable data structure, then performing an hyperparameter tuning using a Bayesian search, whose result were used as starting point for training and finetuning the model using an adaptive learning rate. Regarding the number of epochs and general training management, we used to start with short training sessions and let training continue for many epochs once the best parameter configuration had already been determined.

a. 1D Convolutional Neural Networks, PCA and ICA

Our first approach was to interpret the data as signals of an unknown type. So, we designed a 1D CNN built from scratch composed of 4 blocks of convolutional layers with increasing spatial extent, a Global Average Pooling layer and two dense layers before the 12-neurons output layer. Each convolutional block is made of three convolutional layers in cascade, using ReLU activation function, L1L2 regularization and a filter size equal to 4. Between each of them a Dropout layer is added to reduce overfitting.

We also decided to give the model the possibility to learn to reduce its own complexity, if needed. Therefore, we designed skip connections between the first and the third convolutional layer, so that the output of the block is the sum of latent representation found after the first layer and the one computed after the third. Then we applied to the output a MaxPooling layer.

This kind of model performed well both on our local test and on CodaLab, having an accuracy of respectively 72.1 and 70.2.

Considering the set as composed of signals we thought that a higher accuracy could be achieved working on the signals rather than on the model due to the possible presence of noise for instance. So, we tried to get cleaner data without losing information applying PCA and ICA, two methods used to reduce and eliminate the noise and to extract synthetic features as informative as the original ones but in a lower dimension space.

Unfortunately, despite the reasonable theoretical base, they didn't lead to a significant improvement. Hence, we moved to other model architectures to address the problem.

The first trial was to keep the structure of the 1D-CNN model adding a recurrent layer after the convolutional blocks. The idea behind this is to make the model learn recursively not just the input itself but a latent representation of it. Another approach was to insert some recurrent layers at the beginning, fed with the input, and then some convolutional blocks with the same structure as before to include skip connections. In fact, we thought that we could extract relevant features from the recurrent layers' outputs. Both these attempts were however not satisfying in terms of accuracy. Thus, we moved to different approaches.

b. Mapping the task to an image classification problem

Starting with the assumption that everything neural networks do is to find the most appropriate parameters to correctly classify an input thanks to an optimization problem, regardless of the intrinsic meaning of that input, we decided to also try to handle our time series as images. Hence, we designed a 2D CNN starting from pre-trained architectures to classify images obtained from the original signals. We began by simply mapping the 36x6 signal to a 2D image, but soon we increased the complexity of the processing.

We had the inkling that such a simple mapping wasn't enough, and we knew we could exploit multiple channels to increase the overall information. Thus, we attempted to map the signals to 3 channel images, by stacking different 2D representations of the original signal. This included augmented versions of the simple mapping in terms of brightness, contrast, and flip. We also applied some transformations to switch to frequency domain to generate images. Indeed, we tried to apply the Fourier Transform to the entire sample and the Spectrogram, building a 6-channel image to be used as input for our CNN.

As for the architectures used at this point, we included in our models some pre-trained networks like VGG16 and EfficientNet. Using interpolated images, starting from the samples, the results were in line with the 1D-CNN model considering a

0.708% of accuracy on CodaLab. Unfortunately, the results were not very satisfactory with images coming from frequency domain transformations probably due to the too short sample period.

c. Recurrent Neural Networks

Motivated by the ambition to improve our results, we decided to switch from CNN to RNN, hoping that these new models will fit better the data. We started by trying Long Short-Term Memory (LSTM) models, since they have been developed to overcome the vanishing gradient problem in the standard RNN. The initial configuration was simple, but the results looked promising, so we were encouraged to follow this lead. We switched to bidirectional RNN, to exploit the information flow in both directions, designing a model consisting in two bidirectional LSTM layers and one dropout layer as feature extractor and a fully connected top for final classification. By doing so, we had a substantial improvement in performance. However, we were still not completely satisfied since we had low accuracy on some classes. To get more out of recurrent models we decided to exploit the Attention mechanism.

d. Attention Mechanism

The promising results Attention is bringing to the state of the art in sequential data problems led us to give it a try, despite the lack of previous experience in the field. Our attention mechanism consisted in a weighted sum of the encoded states the RNN outputs. The attention function was defined as a tanh activation function following a dense layer. The architecture, instead, consisted in a fully connected network on top of the source states, used to compute the attention weights, namely the set of parameters to used to compute the weighted average to define the context vector.

To assess the performance of this handcrafted mechanism on our problem, we included it in a RNN. The first attempt was performed on a network consisting in two Bidirectional LSTMs, the attention layer, and a fully connected top for classification. The results obtained were quite satisfactory still having low accuracy on less represented classes.

Therefore, we went ahead trying to change the first LSTM layer into a GRU layer, since it is known to perform better when dealing with limited data available, as in our case. By doing so we obtained a 2% increase in accuracy.

e. Ensemble methods

After testing many model architectures, we selected our best ones to build an ensemble model to further improve our performance. The ensemble architecture included the chosen models and a final layer combining the output vector of each model, usually by average. We tried many configurations and combinations of models, some performed better than others, but all of them showed an increase of performance over single models.

V. Final Model and conclusions

To define our final model, we combined all our best solutions, namely:

- the recurrent model with attention and a GRU bidirectional layer,
- the model with the attention layers,
- the fine-tuned EfficientNet.

This allowed us to have a significant increase in accuracy as in the first challenge. In fact, we went from having an accuracy of 70.8% on CodaLab with the EfficientNet in the first phase to 74.0% in the first phase and 75.25% in the second one.

The increase observed in the latter phase with the larger dataset can be explained by a different distribution of the labels and a great generalization capability of our model.

We are delighted with the performance of our deep learning model in the competition. It exceeded our expectations and demonstrated a strong ability to understand and solve the proposed problem. Also considering our final placement on the leader board, we can say to be very proud of our dedicated efforts. These challenges have been an exciting experience for us, and we appreciate having the chance to work on them.