NonDormoLucido:
Anastasi Nicola - 10807261
Casciani Daniele - 10638165
Terraneo Leonardo - 10530883

# Homework 2 - Report:

## Time series classification

### Problem description:

The aim of the second homework was to build and train a Neural Network that could perform time series classification on a dataset that contained 2429 sequences formed each by 6 channels with 36 values each and belonging to one of the 12 possible labels.

#### Approaching the problem:

In the early stage of the challenge we performed some data exploration, examining the frequency of the labels as well as plotting some of the training samples. We noticed the presence of class imbalance, with class 9 having more samples than the others and with some classes, such as class 0, having very few samples. Similarly to how we approached the first homework we decided to split the training data into a training set and a local test set, which we used to evaluate the impact of preprocessing techniques and different network architectures on the performance of the models before submitting them on Codalab. The models were evaluated using several metrics, such as accuracy and F1 score, as well as confusion matrices to visualize on what classes the models performed better.

#### Applying preprocessing:

After the first commits, the difficulty in classifying some classes, specifically those with fewer samples, immediately emerged.
To improve the performance of our models we tested several preprocessing techniques, both in isolation and in combination with each other, namely:

- Padding: At first, as we have seen at the laboratory session we performed padding with all zeroes, but after a more accurate analysis of the sequences, we thought that it would have been a good idea to try to ignore them when training our network: we therefore implemented masks that avoided feeding the network with the padding values during training. This approach resulted in an increase in performance.
- Min-max scaling: we adopted three different strategies. First we performed a min-max scaling considering the whole dataset and applying the same scaling for all the input data. The second one took in consideration max and min for each series, while the last attempt tried taking max and min for each class. However this approach did not improve our model's performance and was eventually scrapped.
- Standard scaling: we tried scaling the time series in order to give it zero mean and unit variance, testing how this scaling impacted model accuracy when applied along different dimensions. We tested scaling each of the six channels after concatenating them, as well as scaling each of them along a single time series. This approach gave us mixed results,

causing marginal improvements with some models and drastic reductions in accuracy in others.

- Seasonality decomposition: we decided to exploit knowledge acquired from other courses, after observing the time series shape we thought that a good way to proceed could have been to decompose the signals into three parts: the trend, or the increasing or decreasing aspect in the series; seasonality, or the periodicity component that was extremely visible in our series; and noise. The first two are the two fundamental ones, the ones that we wanted to train the network on (since noise, in theory, should be unpredictable). Our approach at this point was to train our model only on trend and seasonality: hence we took the seasonality component, and we added it to the trend component. We trained our best model at that moment with this, however the performance decreased dramatically. Still considering this technique very valuable for 'cleaning' the signal we believe the main cause of the downgrade is imputable to the small size of the series (36 timestamps for each one), which probably makes it more difficult to correctly distinguish between trend, seasonality and noise, leading to a loss of important information.
- Rolling mean: This technique, as the previous one, is based on the idea of making the input signal smoother. Rolling means creating a moving window with a specified size and performing calculations on the data in this window which, of course, rolls through the data. We tested the effects of applying a rolling mean to smoothen the signal, finding the best results to be when the mean was applied to three consecutive values. It was interesting to note that the models trained with this type of pre-processing were the only ones which obtained statistics greater than 0 for all classes. Nevertheless these models still achieved a lower global accuracy on Codalab and therefore we did not include rolling in our best model.
- Data augmentation: in order to obtain more training data, as well as tackle the issue of class imbalance, we tried generating new samples for the least represented classes by taking some of the time series in the training dataset and adding gaussian noise to them. We tested several parameters for the gaussian noise but ultimately this approach did not improve model accuracy.

In addition to these techniques we also experimented with class weights in order to address the class imbalance, however,after seeing that this resulted in worse accuracy results on Codalab, we assumed that the hidden test set was imbalanced in the same way as the training set and scrapped this idea.

## LSTM Network:

The first networks we tried using were those we had seen in the laboratories, starting with the LSTM network. We used a simple architecture with an LSTM layer to extract features followed by a series of dense layers to perform classification. We experimented with different numbers of dense layers as well as with applying several of the preprocessing techniques we described earlier, but this network never achieved better results than the others.

## BiLSTM Network:

A very similar approach was taken with the BiLSTM network, where we used a bidirectional LSTM layer for feature extraction. We repeated the same experiments we performed with the LSTM network and achieved better results, but the model accuracy still never broke past 60%.

## Convolutional Network:

While testing network formats the convolutional archetype was the next to try out. The first draft consisted of a simple convolutional layer, followed by a max pooling, a global average pooling and finally a dropout layer to perform the classification. Early tests revealed that the model was really inaccurate, yielding locally a maximum of 34% average guess rate on all the classes meaning it wasn't complex enough to correctly classify the series. The next iteration was increasing the number of layers, namely a new convolutional layer, max pooling and dense layer, to be inserted before the last dropout classificator layer. Adding layers returned no substantial increase in performance so we scrapped the idea of adding layers, instead we started increasing the number of parameters the network could work with leaving the model as simple as possible. This was the best result obtained with a convolutional network, having a simple convolution layer that had an output shape of (36, 4096), a max pooling , a global average pooling and a dropout layer to perform the feature extraction followed by a dense layer with 4096 parameters and a last dense layer to perform classification. Locally this model correctly classified 71% of the series given to it, while on codalab it got a score of 64.9%. This result was achieved using both rolling and attention mechanisms but normalization made the result worse. Further iterations were experimented with new layers, parameters and different preprocessing but in the end these didn't yield better results.

## ResNet

Seeing as we were not getting good results with the model we had seen in the laboratories we decided to try a ResNet-style architecture. The network we used was composed of three residual blocks connected in series, followed by a global average pooling layer and a softmax classifier with 12 neurons. Each residual block is in turn composed of three convolutional layers, each one followed by a batch normalization and activation layers. Once again we tried combining this architecture with several of the preprocessing approaches we described earlier, but ended up noticing that all of them impacted the model's performance for the worse. Still, even without scaling or decomposition, this was the network that performed best, averaging around 69% accuracy in our test.

## Complex Model

We also tried a more complex model, composed of some parallel parts of the network. The reason was to merge the output of a CNN network and a bidirectional LSTM network to achieve better performances. It was composed of one input layer, which was the input of three separate parallel branches: two of them were exploiting convolutional, and implementing; the other was only composed of a bidirectional LSTM.
Then, these tree branches were concatenated, and pooled. From our tests, both locally and on Codalab, is carried out that this kind of network works badly with our dataset, therefore we dropped this line.

## Attention Mechanism

After reading some articles about it we understood the potential of this approach. We found that there was a library,  called "keras self attention", ready to be used. Therefore we decided to try it with all the models already described above, adding a self-attention layer to them. The better results we obtained were with bidirectional LSTM network and one layer of Attention with sigmoidal activation, even if the global accuracy was still lower than before, reaching a score of around 65% on Codalad.