Competition 2 – Artificial Neural Networks and Deep Learning

Tommaso Capacci, Simone Giampà, Gabriele Ginestroni

Models and Ensembles

Our first approach consisted in adapting the handcrafted convolutional network designed in the previous challenge to use only 1D convolutions. Immediately, we noticed higher overfitting with respect to the image classification task.

After that, we tried adding some LSTM layers before and after the convolutions. Models with LSTM layers after the convolutions showed better results. Nevertheless, the validation accuracy was much lower than the one we got with simpler conv-dense models. For this reason, we decided to substitute the LSTMs with GRUs. This brought a good 2-3% increase in the accuracy, especially when using bidirectional GRU layers. This is probably caused by the smaller number of parameters of GRUs.

We also tried using an attention layer, which was effective when placed after the bidirectional GRU layers. This layer has been used to eliminate the temporal dimension from the output of GRU layers (we set return sequence to True in every layer). The output of the attention layer is then fed into the softmax.

Our best model is composed of a series of convolutions, 2 bidirectional GRU layers, and an attention layer. This model has been able to reach a validation accuracy in the range 74-77% with good consistency. Our final submission is an ensemble of our 3 best models (two of them are conv-dense models). With the ensemble, we increased the test accuracy of the best-performing model by about 2% (on Codalab).

Scaling

We applied standard scaling, min-max scaling (between 0 and 1, and between -1 and 1), and robust scaling. We discovered in our tests that these scaling helped increase the accuracy, but there wasn't a specific scaling that worked well for every model. In some models, such as the convolutional networks, the standard scaling was enough, while in models containing recurrent nodes (like LSTM and GRU), standard scaling after a min-max scaler (the range was not so important) gave better results.

We tested 2 scaling strategies: feature-wise and with respect to each pair (timestamp, feature). The first solution provided better results in all our models.

Class weights

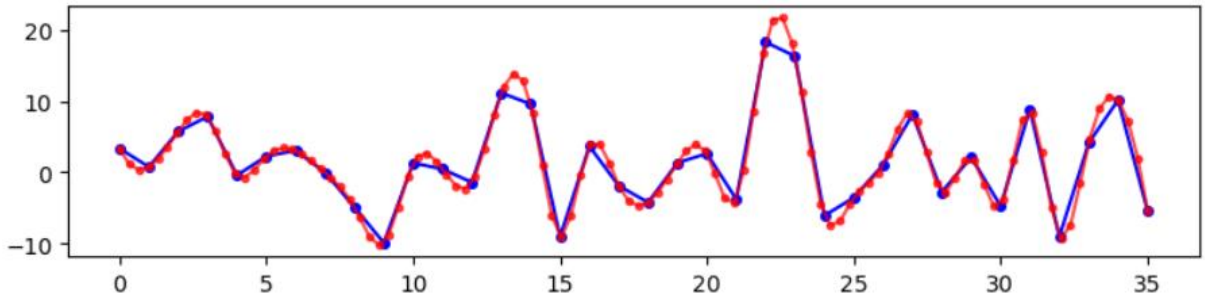
We tried implementing the class weights for the training, but it showed bad results and an accuracy decrease of several percentage points for every model tested. This is probably due to the very strong imbalance of the data samples among the classes, especially the classes that have a very limited number of samples available. We eventually did not use any class weighting in our trainings of the models.

Data augmentation techniques

All the augmentation techniques used were applied to the dataset in an offline fashion. This means that the dataset is augmented before training, and it doesn't change at every epoch (like a training dataset generator would do).

The first method used for data augmentation was **data oversampling** for the classes with the least number of samples available. This method takes every sample and copies it multiple times so that eventually all classes will have more or less the same number of samples. The class with the highest number of samples is not oversampled at all. This approach aims at balancing the dataset as much as possible. When each sample is copied, a random gaussian white noise is added to the time series (feature-wise) so that the samples are not exactly equal but they differ slightly one from the other.

The second method used for data augmentation was the **cubic interpolation with 1d splines** of the time series. This approach serves for getting a higher resolution time series that is as accurate as possible, considering the number of points available for the time series. Our code takes each sample and increases its resolution of the time series by 3x. So we trained models that take an input shape of size (108, 6) instead of (36, 6). We also tried with different resolution multipliers, but the best-performing ones were 3x and 5x.



This plot shows the effect of the cubic interpolation of a randomly drawn time series. The blue dots represent the original time series points, while the red dots the interpolated time series. The blue dots are distanced 1 timestamp between each other, while the red dots are 3 times denser in space.

This increase in the resolution of the time series did not show significant improvements in our models, as we expected, so we used another technique to exploit the cubic splines interpolation differently.

Cubic splines interpolation for the creation of artificial samples

The **final method** for data augmentation is based on the computed interpolated time series of the entire dataset **without increasing the resolution of the samples.** The previous approach created a dataset of 2429 samples with shape (108, 6). Now, instead of training models on time series with higher resolution, we decided to try to train the models on time series with the original resolution, but on a dataset with more samples. So, the final technique creates **artificial samples**, using the interpolated time series, to create a training set with more samples than the original one. The algorithm takes every interpolated time series from every sample. Then, given a time series with 3x resolution, for example, it extracts 3 different time series that are 36 timestamps long (the original time series length). For each extracted time series, a gaussian white noise is added too, so to create samples that differ as much as possible.

The way for which new artificial time series are extracted from an interpolated one is by taking the points that are 3 timestamps apart and saving the sequence in 3 time series. For example, if the resolution multiplier was set to 3, the length of the interpolated time series is 108 and the first sequence is obtained by the points at the timestamps [0 3 6 9 12 … 105]. The second sequence is obtained by the points at [1 4 7 10 13 … 106], and the third sequence is obtained by the points at [2 5 8 11 14 … 107]. This way 3 new sequences with length 36 are created and correspond to the artificial samples of the dataset. The new sequences will be very similar one from each other, and

similar also to the original sample. The resulting dataset will contain a number of samples that is 4 times bigger than the original one, because, for each original sample, the augmented dataset will contain the original sample and the 3 artificial samples. This technique proved to be effective in our tests, providing an accuracy increase of several percentage points.

Feature Selection and PCA

We tried removing 1 feature at-a-time, but it seemed not so beneficial. Better results were achieved with PCA, projecting features over 5 or 6 components after applying the scalers to the original features.

Cross-validation

Given the small training time of our models, we decided to select our models with a cross-validation technique. We used the StratifiedKFold cross-validator from sklearn, with k ranging from 7 to 10. The stratification was important as the original dataset was very imbalanced.

It's important to note that we preferred not to modify the original window provided with the dataset. Indeed, we had no idea whether consecutive time-series samples in the original dataset were in temporal-dependent order. Additionally, we had no clue that at test time (on Codalab) samples were fed in the same order. In this way, we considered every sample independent from each other and were able to implement the cross-validation straightforwardly. Cross-validation has been used to also test the splines augmentation, learning rates related parameters, PCA, features drop, and all combination of scalers.

The use of cross-validation turned out to be very important as the performance of models in this problem setting appeared strongly influenced by the train-test split. In this way, we selected the models that had high average accuracy and low fluctuations among all the splits, discarding models that reached very high accuracy due to a "lucky" split.

For what concerns the splines augmentation, with cross-validation we found reasonable values for the resolution multiplier and the standard deviation of the gaussian noise.

***Tsaug* library for augmentation**

We also tested the *tsaug* library, which provides augmentation for time series data. The results were worse than our splines augmentation implementation. Moreover, it slowed out too much our trainings, as the augmentation is computed by the CPU, so testing many augmentation combinations inside the cross-validation was not affordable. Therefore, we decided not to use it.

Other possible techniques

These are some ideas that haven't been tested but could improve results.

The first one is reshaping the samples as an image and use 2D convolutions with augmentation applied at training time using network layers (for example one could apply random translation to the input signals).

Another option could be reshaping the training set and rearrange it in windows of different sizes. This can be beneficial in case consecutives samples are temporally correlated, better exploiting the capabilities of recurrent networks.