ANNDL- Challenge 2

Adriano Racano – Mattia Siriani – Marco Tramontini

General considerations on the dataset

The training dataset is composed of a NumPy array representing time series (X) belonging to 12 different classes defined by the labels contained in an array (Y).
The X array is composed by N windows sized 36x6 (36 float values for 6 features)
It is also evident that the dataset is not balanced (classes 1, 5, 8, 11 and 12 have a lower number of samples respect to the others, while class 11 has a very high number of samples respect to the others) and that overall, it is very small.

Our considerations on the dataset

We thought that with a dataset like this (very small), it would be better to use K-fold cross validation but we got better results using the Hold-out technique and so we opted for that.
We split the dataset in training, validation, and test (0.8, 0.1, 0.1) and we used the confusion matrix to evaluate the robustness of our model.
Once we decided all the hyperparameters, we trained with more data and we obtained the best result splitting the dataset as 0.95 for the training set and 0.05 for the validation set, without a test set.
Since we did not know if the time series were sorted or not, we assumed they were not, but we also noticed that this aspect does not affect the result.

Analysis on the dataset

To better understand how the dataset is composed and to graphically visualize the data, we plotted it in 2 different ways:

- The features of each class in a single graph, to visualize the general trend of the time series.
- The features of each class in different graphs, to check if some features have more jitter than the others. We then noticed that overall, the first feature had more jitter than the others and so we decided to remove it from the whole dataset. Doing that, the accuracy slightly increased.

Dataset balancing

Since the given dataset was not balanced, we tried to use different techniques in order to make the model recognize all the classes at the best possible.

- The first approach we tried, was to use class weights. We manually calculated class weights according to the distribution of the samples over the classes, dividing the result by the total samples of the dataset in order to obtain percentage values. The resulting weights slightly influenced the classification of the classes during the training, assigning a higher weight to the smaller classes.
- The second approach was to use the under-sampling technique randomly removing samples from the most populated classes until the dataset was well balanced. Since some classes were very small, we established an upper bound of the number of samples per classes.
- The third approach was to use the over-sampling technique randomly duplicating samples of the less populated classes. We also decided to augment the duplicated samples in order to reduce overfittings.
- The fourth approach was to use a combination of under-sampling and over-sampling until we reached a defined number of samples per each class (discarding samples from the more represented classes and duplicating samples of the less represented classes)
- The last approach was to augment the training set to generate new samples

The first approach gave the best results overall while the second approach gave us the worst results since there were too less samples in the dataset to let the model learn. The third and fourth approaches gave us quite good results, but considering the number of duplicates in some classes, the models overfitted on them and so we decided not to apply them. The last approach gave us bad results in terms of accuracy since the new generated series were too different from the original ones and so the model was unable to learn.

Preprocessing

To better generalize our model, we decided to use a scaler to normalize the series. After some tries of Minmax and Standard scaler, we saw the best scaler was the Robust Scaler and so we opted for it. The reason why this scaler works better, is that it better manages the outliers that are present in our classes as we saw from the graphs. We decided to apply the scaler to the features and not to the windows since we thought that in that way, we were not modifying the overall series trend.
We also tried to reshape the windows so that the new shape could be more useful than the given one. To do that, we tried to:

- Duplicate many times the features of a window, so that we obtained a 36x36 matrix that can be seen as a squared image.
- Sum and multiply the existing features to obtain new features that can possibly be used to arbitrary modify the shape of the windows combining the new features with the old ones.
- Randomly pick existing features from different windows to obtain new windows of different size (even if we expected the model to overfit a little bit).
- Apply a sliding window mechanism so that every 12 time-instants a new window starts. In this way we multiplied by 3 the total number of windows but since the original data were the same, it led us to overfitting.

When we modified the size of the window for the training, we also modified the size of the window in model.py file, duplicating the same feature multiple times to obtain a matching window shape.

Data augmentation

Since the dataset is quite small, we decided to apply data augmentation on our training set to increase the number of samples that we could have used for training and to better generalize the model. Since the transformations made in the augmentation process are pseudo-random, we did not augment the validation and test sets to avoid the model to learn on transformed samples.
To perform data augmentation, we both used custom generated functions and the tsaug library.
The transformation we tried to apply on the series were jittering, scaling, magnitude warping and time warping (according to what we read on this paper: https://arxiv.org/pdf/2206.13508.pdf).

Callbacks

In order to avoid overfitting, we used early stopping monitoring the “validation_accuracy” and after the patience ends, it restores the best weights. We also tried to use a scheduling technique (ReduceLROnPlateau) to dynamically reduce the learning rate of the optimizer (we used Adam as optimizer) when there is no improvement for a defined number of epochs. Doing so, the model takes smaller learning steps to the optimal solution of the cost function.

Models

We noticed that better results came when we ran the models for a lot of epochs.
To perform the best training, we chose an upper bound limit of 1000 epochs with 200 epochs as early stopping, and a batch size of 128, which, after some attempts, turned out to give the best results in terms of accuracy. We also noticed that using a low learning rate, the accuracy slightly increased.
We then tried to apply label smoothing’s (value: 0.2) to the loss function to prevent the model from becoming overconfident and failing to generalize obtaining better results.

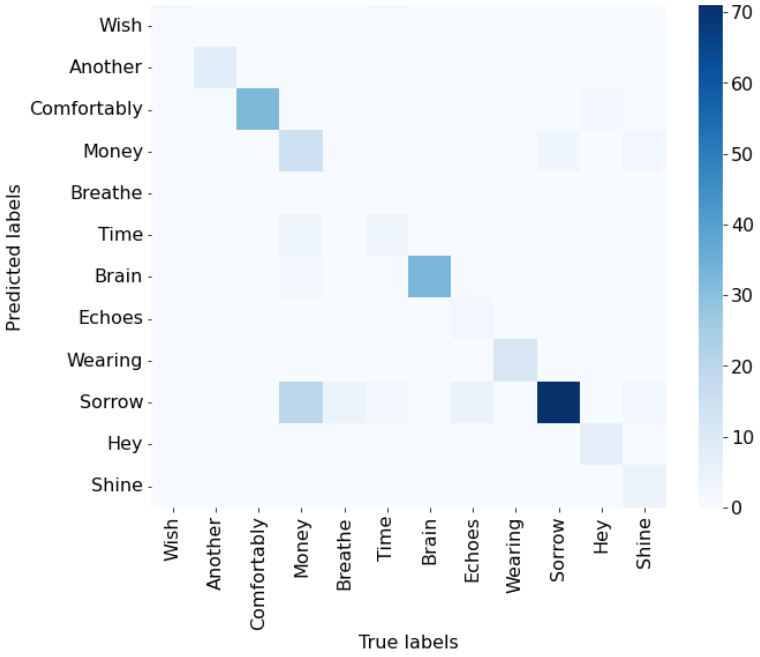
We tried different type of models suited for time series classification:

- LSTM and BiLSTM: they gave us good results, especially when increasing with larger dense layers in the classifier or increasing the number of LSTM/BiLSTM layers with return_sequences set to True.
- 1D CNN: this network was similar to the CNN used to train images, but with Conv1D layers, in order to train time series instead of images.
- ResNet: 1D adaptation of the ResNet model for image classification, which has the advantage of using skip connections between consecutive convolutional blocks. This one gave us the best results overall. To build our model, we started from the GitHub code related to the paper: https://arxiv.org/pdf/1809.04356.pdf. We tried to modify a little bit the classifier to see if we could better adapt it to our dataset but at the end, we obtained the best results leaving it as it was.
- GRU and BiGRU: we tried to substitute the LSTM layers of the previous model with GRU layers, which were supposed to have the same performance of LSTM but being computationally more efficient, in fact they gave very similar results.
- ConvNeXt: our last approach was to replicate the features in each window, in order to move from a shape of 36x6 to a shape of 36x36. Once obtained those matrices, we added a new dimension to turn them into images and lastly, we exploited transfer learning, using the same network of the first challenge, but in the end, it did not give us the expected results and so we discarded it.

GAN

To increase the number of samples of the dataset, we tried to create a generative adversarial network that could create new samples from the existing ones. After some tries, we did not manage to create a time series custom GAN and so we tried to recycle an existing image-based GAN with some adjustments. Since the windows had not a squared size, which was preferable for a GAN built to generate images, we needed to reshape them using the techniques described in the preprocessing paragraph. We also added a padding to fix the output shape according to the requested window size.
At the end, looking at the result graphs, we saw that our generated samples were too far from what we expected and so we decided not to use them.

Results



After training our best model, we analyzed its robustness on the test set.
As shown in the confusion matrix, our model is still not able to generalize so much that it can recognize all classes but given that the diagonal is highlighted, we thought that this was a good compromise. It is also clear that class 10 is the more recognized (it is very dark) but since it is also the most populated, the confusion matrix reflects the dataset distribution. Furthermore, the test set was split taking a fixed percentage of samples from each class and so it was unbalanced too. We tried to balance the test set and that gave us a better confusion matrix but since in this way, some classes would be very low represented, we preferred not to remove too many samples from the training data.