Artificial Neural Networks and Deep Learning Time Series Classification Homework

Viola Renne, Alessandro Rossi, Lea Zancani

April 4, 2023

Initial phase We started inspecting the dataset we received and trying to figure out the meaning of the three dimensional data. This first approach to the dataset was not as trivial as we expected it, also because in the laboratories we always started with a csv format while in this case we were dealing with numpy objects. We started with a qualitative analysis of the six features of the time series we had to predict by portraying them all on a graph. Unfortunately, since the names of the features were unknown to us, we were not able to make a priori guesses about the patterns of the time series or their correlation. The graph allowed us to observe that the time series showed an almost constant behaviour in most of the points, except for some peaks with a particularly high value compared to the other points. We tried to keep this into consideration while prepossessing the data.

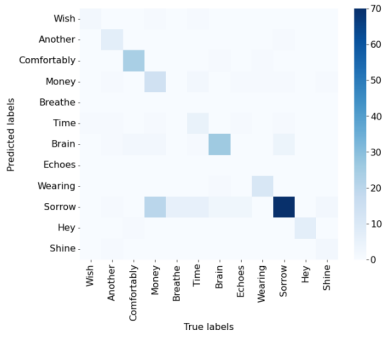
Splitting the training, validation and test set We split the data into training, validation and test with a partition of 80%, 10%, 10% respectively. We did this with a function we wrote in order to make sure the partitions were balanced in the number of points taken for each feature.

Pre-processing We tried different approaches to pre-process the data, starting from the MinMaxScaler, which did not give us good results. So we switched to Standard and finally Robust scaler. The latter gave us the best results, which was consistent with the observation of the behaviour of the time series we previously made, that highlighted the few peaks with high values that made the MinMaxScaler technique not suitable to our project due to his sensitiveness to outliers. We also tried to normalize the data, but immediately discarded this option since the performance was drastically decreasing. Since in the first competition we noticed that our score would heavily increase by adding batch normalization layers, we tried also to do that, but again, we found out that this time the model only performed in a worse way in all our experiments and so we decided to abandon the idea.

First experiments The different tries for the pre-processing were made with the model with 2 LSTM and dropout layer we used in the laboratory, and a even more basic model with just one LSTM and one dropout layer. With the first one, using the RobustScaler, we reached the best local accuracy: 0.694; this became our reference for following models.

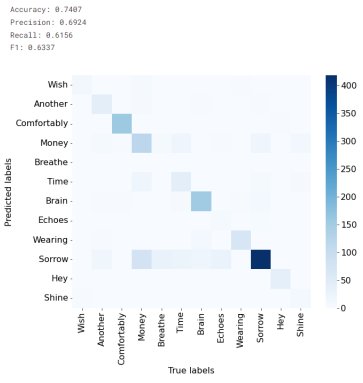
Data Augmentation Based on the work done in the first challenge, where the data augmentation played a key role to improve models' performances, we used the tsaug library to perform data augmentation in order to generate new samples by adding a scaled random noise to the existing samples, which led to some improvement of the performance of the model; we tried different value for the standard deviation of the random noise but the best one remained the default value (0.1).

FCN and bilateral LSTM We tried to develop a Fully Convolutional Network with 1D Convolutional layers and a Global Average Pooling Layer, reaching our best result with 4 conv1D layers with a local accuracy of 0.7029 (0.67 on codalab). We tried to modify this model (by adding other conv1D layers, applying kernel regularizer etc.), but couldn't reach better results, so we went back to the LSTM model and tried to switch from LSTMs to bidirectional LSTMs, reaching a first local accuracy of 0.7071 with the basic model of 2 BiLSTMs and a dropout layer. To further enhance this architecture we added a 1D Convolutional layers in order to build an "hybrid" model, which led us to a local accuracy of 0.7159 and an accuracy of 0.7183 on both phases.

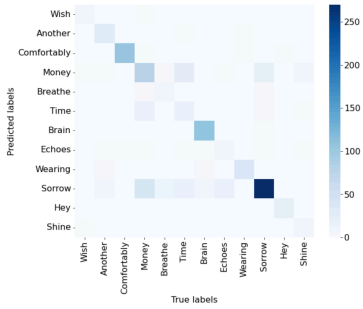


(a) FCN confusion matrix

ResNet inspired model We then tried to reproduce part of a ResNet architecture, with 3 convolutional layers and one skip connection with a keras add layer. We noticed it performed really well, especially at detecting patterns throughout the time series. We thought about adding this layer since big and complex networks are hard to train and easy to overfit, so it could be very useful to explicitly add this term to avoid these problems. By using the skip connection, we provided an alternative path for the gradient (with backpropagation). We also noticed that adding this layer helped reducing the number of epochs needed to reach convergence. With this model we reached an accuracy of 0.741 on the first phase on Codalab and 0.728 in the second one. Starting from this model, we did some tuning, especially on the stride value of the function we used to build sequences (obtaining as best parameters a window size equal to 36 and 6 as stride), and we added an Relu activation function layer, which led us to our best performing model: accuracy of 0.7485 and 0.7351 in the first and second CodaLab phase, respectively [Figure 3a].



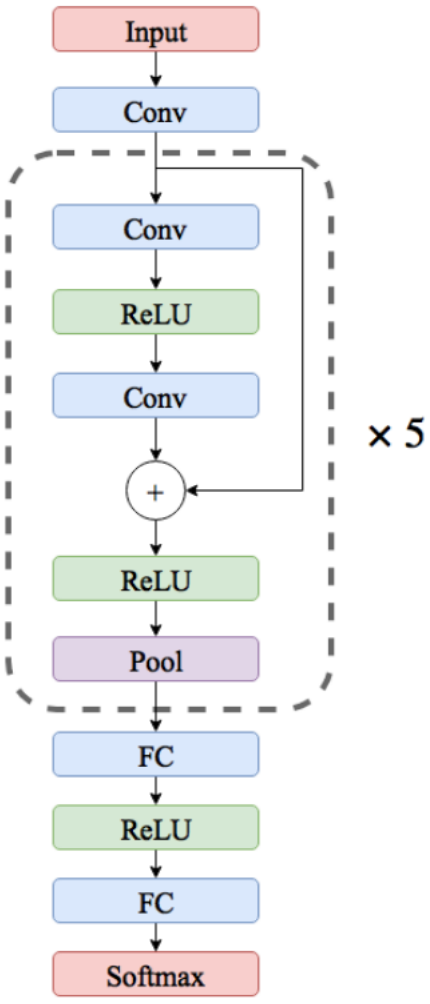
(a) ResNet Conv1D confusion matrix



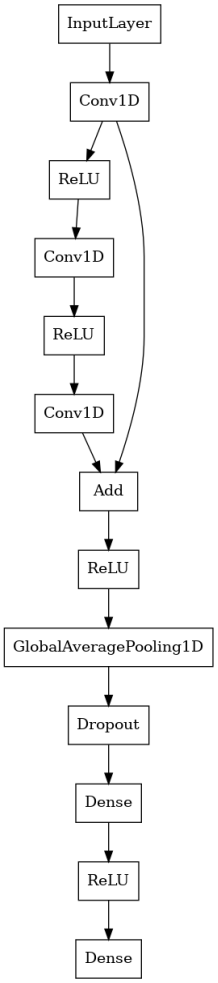
(b) Bidirectional LSTM confusion matrix

Attention We tried to use an attention module to improve the model with BiLSTMs and Convolutional layers, but could not improve the result we already obtained without it.

Removing features At the end, we tried to remove one feature at time and train the model with the five remaining features. In all the attempts the final performance on the validation and test set was slightly worst than the result we obtained with all the features. So, we decided to maintain all the six features in our final model.



(a) ResNet reference model



(b) ResNet Conv1D final model