Students: AZIMI Arya, BELOTTI Ottavia, IZZO Riccardo
Student IDs: 10824423, 10657411, 10599996

Homework 2 Report
Artificial Neural Networks and Deep Learning - A.Y. 2022/23

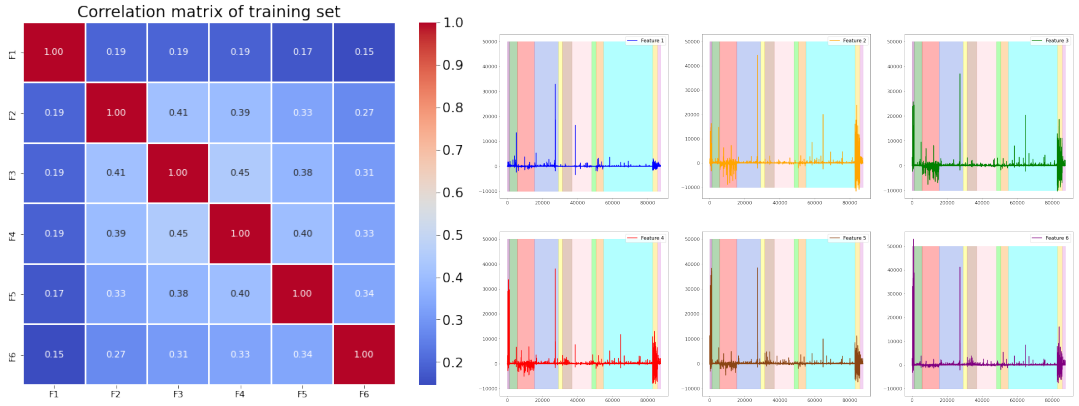
1 Definition of the task

The second homework consisted of a time series classification task over a dataset of 2429 time series with a window of 36 units, composed of 6 features each, and distributed over 12 classes.

2 Data Analysis

We started inspecting the data set before building the model to get a deeper understanding of the data at hand. We discovered that features were not correlated among each other (*fig.* 1a). Furthermore, we looked at the distribution of the features: no regularities have been found, instead we noticed some spikes in values for certain classes.

The data set is not uniformly spread among all the classes either, as we got more samples representing the class *Sorrow* and very few for classes *Breathe* and *Wish*, this explains why all of our models have learnt pretty well *Sorrow*, while are slacking on others, especially on *Breathe*.



(a) Correlation matrix of the data set (b) Features and classes distribution over the data set

Figure 1: Data Analysis and Exploration

3 Models

3.1 BiLSTM

As the number of data is not relatively big, we figured that using a simpler model will end up with a better result, therefore, we tried a model featuring LSTM layers or simple RNN/GRU layers, but we soon discovered that using a simple architecture composed of 2 Bidirectional LSTM layers and a dropout layer for feature extraction, and 2 dense layers for the classifier worked better. For the hyperparameters tested through the validation set, we got the best results for this specific model using a 0.5 dropout, test_size=0.2, and validation_split=0.1.

For this model, we tried a couple of data preprocessing methods, namely deleting and adding features as described in *sec.* 5.1, but still the version without any preprocessing performed better.

3.2 1D CNNs

First CNN We tried implementing different 1D CNNs, reaching almost 70% accuracy as the best performance. The best one is composed of 4 Conv1D layers (128,128,256,256 filters) interleaved with MaxPooling1D layers for the feed-forward part; a GlobalAveragePooling1D and 0.2 rate Dropout layer preparing the data for the classification part which consists of 3 fully connected layers (128, 64, 12 as output). The input has been standardized, allowing us to gain up to +7% in accuracy with respect to the non-standardized input fed to the model. While we noticed that standardization led to a slight decrease in accuracy on LSTM-based models.

Second CNN Another configuration tested was the following one: 1 Conv1D layer (64 filters) with a MaxPooling1D layer, 2 Conv1D layers (128 filters) with a MaxPooling1D layer, 2 Conv1D layers (256 filters) with a GlobalAveragePooling1D layer and a Dropout layer (0.25 rate). Finally the classifier is made up of 3 dense layers (128, 64, 12 as output) with a Dropout layer (0.5 rate) after the first dense layer. This reached almost 69% accuracy, with no preprocessing as well besides data standardization.

3.3 Hybrids

We tried several combinations alternating LSTM and CNN layers in both directions, some configurations allowed us to reach an accuracy of 65% more or less. Another tested configuration consists of a network with two sides: the first side is made up of 3 Conv1D layers with MaxMaxPooling1D and Dropout layers, on the other way the second side is composed only of 2 Bidirectional LSTM layers. At the end we merged the two sides with an Add layer, proceed with a final Conv1D layer and a GlobalMaxPooling1D layer, finally for the classifier we have 2 dense layers interleaved with BatchNormalization and Dropout layers. With this hybrid we reached a decent accuracy of 68%.

3.4 Transformer

Our transformer is implemented with the following architecture: 4 chained transformer encoders and, for the classifier, two 128-units dense layers interleaved by 0.5 rate dropout layers, finally a 12-units dense layer for the output. Each transformer encoder consists of a MultiHeadAttention layer featuring 4 heads each of size 256 and 0.25 rate dropout, a normalization layer, while for its feed-forward part, two 128-filters (3x3) Conv1D layers, a 0.25 rate dropout and a last Conv1D layer with 6 filters (1x1) with a normalization layer. Unfortunately, we haven't been able to achieve results up to our expectations, managing to score 56% in accuracy at best.

4 Best Model

Among all the models we have generated, the above mentioned two CNNs (*sec.* 3.2) are the ones that, taken alone, can achieve the best overall prediction accuracy, however both of them severely lack the ability to correctly predict class *Breathe* and class *Echoes* (as can be seen in *fig.* 2a). From our previous data analysis we attribute the source of the problem to scarce representation they have in the data set and to the very small values that characterize those classes under all the six features. This said, our best BiLSTM model doesn't reach the CNNs accuracy, but it manages to score a much higher accuracy in those difficult classes. So we combined the CNNs with the LSTM model by averaging the predictions' probabilities. This allowed us to reach 73% in accuracy.

We tried ensembling the CNNs with other models as well, but they didn't reach the above mentioned result.

5 Further Experiments

Quasi-SVM A RandomFourierFeatures layer has been added but yielded inconsistent results, making the accuracy oscillate too much ($\pm 5\%$). For this reason it was later deleted.

Patience As the fit phase of all the models was limited by the Early Stopping technique, we also tried with different values of patience and we ended up using a patience of 15 which avoided ending the fitting too early and overfitting too much.

5.1 Data Preprocessing

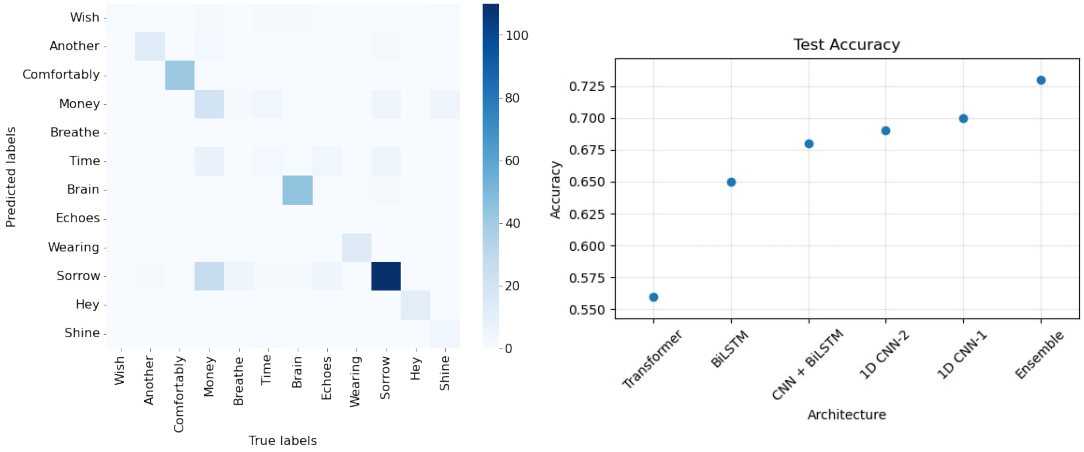
Oversampling and Undersampling We tried both the RandomOverSampler and the RandomUnderSampler to manage the imbalanced dataset. More in detail the RandomOverSampler allowed us to randomly duplicate the samples of the less represented classes such as *Breath*. On the other way, RandomUnderSampler deletes randomly selected samples of the most significant classes from the training dataset. However these didn't increase accuracy by much.

Scalers We tested three different types of scalers which are MaxMinScaler, RobustScaler and StandardScaler. All three scalers transform the features by scaling each of them, given a specific range. This ended up with a slight improvement in accuracy only on LSTM-based models.

Noise functions To add robustness to the nets, we tested three noise functions by adding uniform, Laplace and Gaussian noises, with poor results.

Different class weights We used the parameter class_weights defined by Keras to handle imbalanced classes. This resulted in a +1% accuracy on the BiLSTM models.

Add/delete features We tried adding the mean and the weighted mean of the features as a new feature, deleting the noisiest feature and deleting the feature furthest from the mean, where none of them made a remarkable change and mostly reduced the accuracy. Especially, removing features was expected to reduce the accuracy since our study over the correlation between features proved that they were mostly uncorrelated among each other, so every one of them should bring useful information.



(a) First CNN's predictions confusion matrix (b) Accuracy comparison among the tested models

Figure 2: Results

6 Conclusions

In the end by studying the behavior of the time samples in the time series, we concluded our work, having used the 1DCNN models with standardization and BiLSTM to account for poorly classified classes, which put us in a decent position in terms of accuracy. This challenge gave us the opportunity to experiment with our own models and highlighted how much important is the *a priori* understanding of the problem domain and of the data set.