ANNDL 2022 - Time Series Challenge

Team Name	The Convolutional Trio		
Group members	Andreotti Vittorio 10824694	Buoninfante Erika 10636425	Pizzo Luca 10614734

1 Dataset inspection

The dataset provided to us was composed of 2429 samples, each having a shape of 36x6. We had no clue about the nature of this data or where it came from, therefore we started to inspect it in order to understand what to do.
After some reasoning, we reached the following conclusion: the dataset was already divided in windows of 36 samples each and 6 was the number of features describing our data.
The distribution of the classes is the following:

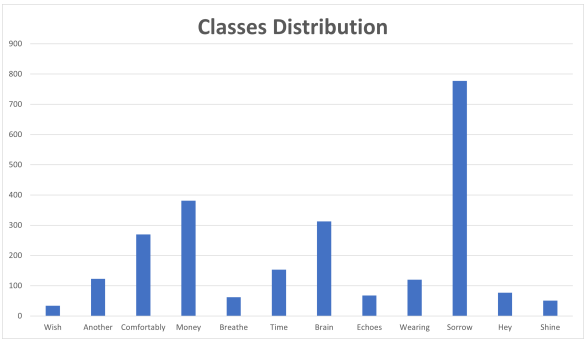

Figure 1: Classes distribution before operating on the dataset

With such a small number of samples, we decided at the beginning to select only 10% of each class for the validation set using the function `train_test_split` and setting the parameter `stratify` in order to have the same distribution of classes for training and validation.

2 Data Preprocessing

2.1 Scalers

To preprocess our data, we used some functions taken from the `sklearn.preprocessing` library:

- MinMaxScaler()
- StandardScaler()
- RobustScaler()

Before applying a scaler, we needed to reshape our training set by removing the second dimension of `X` vector (i.e. 36), and then rebuilding the target vector `y` to associate the right label for each row of X. (The models in which transformed data is used will be described in section 3).

2.1.1 MinMaxScaler

With this estimator we scaled each feature individually such that they were in the range (0,1) on the training set.
This transformation gave us the worst results, in fact our models were unable to classify most of samples, so we discarded it.

2.1.2 StandardScaler

This estimator standardizes features by removing the mean and scaling to unit variance. Mean and standard deviation are computed on the training set and then stored to be used on test data using the `transform` function.
Standardization gave us slightly better results in validation accuracy.

2.1.3 RobustScaler

This estimator scales features using statistics that are robust to outliers. It removes the median and scales the data according to the quantile range (defaults to IQR: Interquartile Range). The IQR is the range between the 25th quantile and the 75th quantile. Among the scalers, this one gave us the better results in validation accuracy, so we kept it in our models.

2.2 Window resize

To learn more from the dataset, we thought about using windows with a modified shape. We glued all the time series of each class to make one single long sequence of values, then cut it in multiple parts to generate data with a (window_size, 6) shape. In order to create more samples, we added a stride parameter in our function and tried various values both for the window_size and for the stride.

Using bigger window sizes gave us some good results in our validation accuracy (80+%): this was expected since we were feeding our model with more informative samples. Unfortunately we didn't find a way to implement this preprocessing in the final test so after several attempts we decided to focus on other methods.

3 Model architectures

3.1 Vanilla LSTM

We wanted to try a LSTM model for its capability of learning long-term dependencies, especially in sequence prediction problems. We started with a vanilla LSTM and tried to tune its hyperparameters but the model overfitted too much every time so we moved on to other models.

3.2 Bidirectional LSTM

In bidirectional our input flows in two directions, making a bi-lstm different from the regular LSTM. We expected this to give us better results compared to a Vanilla LSTM but unfortunately it was not the case. The model overfitted just like the LSTM.

3.3 1D CNN

The model with 1 dimensional convolutional layers was the most promising one because it didn't overfit so much unlike the other ones. When we noticed that it was consistently performing better than other model architectures we decided to focus on this type of models. The final model that we obtained in fact is a 1D CNN and will be described in detail later.

3.4 ResNet-style network

Since we were really curious to understand how the extremely deep neural networks respond to time series data, we investigate the ResNet structure, building a ResNet-Style model. ResNet extends neural networks to very deep network by including a shortcut link in each residual block that allows gradient flow to go directly through the bottom layers. It performs better in object detection and other vision-related activities, indeed it overfits on our training data much more easily because our dataset is very tiny and lacks sufficient variations to learn the complicated structures with such deep networks, but the test was useful to understand how the dataset behaved with such a deep network and to analyze its advantages and disadvantages.

[Our model is based on the model found here]

3.5 Keras Tuner

Since we were trying to find optimal values of some of the hyperparameters with a manual random research, we decided to automate this process with an external library. With `keras_tuner` it was possible to try a big variety of combinations of settings and ultimately find the most promising ones. For example, for the classifier and for the level of regularization to apply (e.g. dropout rate), we were able to try a lot of mixtures of hyperparameters and discard unpromising ones with the HyperBand algorithm. We used Keras Tuner both for Recurrent neural networks and for the simple Convolutional ones.

4 Data Augmentation

4.1 Oversampling with additional noise

Some classes had very few samples so we wanted to create new ones. The idea was to create new windows just like we did with the window resize (2.2), while keeping a window size of 36.
The smaller the stride the more numerous are the windows created, the objective at this point was to find an appropriate stride to reach a desired number of new windows. Since some classes were overrepresented and other underrepresented, we wanted to create more windows for underrepresented ones. This was done by writing a function that would calculate the best stride value from the desired sample size for each class.
While doing so, we added some random noise (with mean = 0 and a standard deviation calculated from each feature of each class) to make our model

more robust and not overfit too much.
We noticed that adding few windows (with a big stride) made the model more accurate in some of the classes, adding too many windows led to worse results and overfitting.

4.2 Undersampling

The "Sorrow" class was extremely overrepresented and we noticed from the confusion matrix that our models were predicting most of the samples as belonging to this class. We thought that a solution could be to undersample the class, so we did it by removing about 400 samples, but we didn't see a real improvement in our performance.

4.3 Weight balance

An alternative approach was to use the parameter `class_weight` in `model.fit()` initialized with the attribute `balanced`, which equated the representation of the whole dataset on all classes. This however did not improve the accuracy of the underrepresented classes, so we abandoned the method.

5 Feature Selection

We thought about selecting only the most useful features and ignore the ones that it seemed like they didn't help the model at classifying correctly. With some combinations of removed features we didn't lose precision but no combination could actually improve it so we decided to keep all features for all classes.

6 Final model

The model that scored better with the hidden test set (about 73%) is a simple 1D CNN with data transformed using the RobustScaler preprocessing. The main parameters are the following:

- number of convolutional layers: 2
- units for convolutional layers: 256 and 512
- number of dense layers: 2
- units per dense layer: 512 and 128
- activation function: Relu
- optimizer: Adam

7 Final comments

The results that we obtained on the validation set and the test set were not so good, we couldn't reach a good precision for underrepresented classes and we also had problems due to a high variance in the final results.
Even models that showed a good accuracy on the validation set didn't guarantee the same performances in the test set and this was probably due to the fact that the number of samples for some classes was extremely low.
This was expected according to what we studied in class since deep learning models need huge datasets to train on but we thought we could still do better.
Further research on other techniques for data augmentation and feature engineering can be done in order to improve performance, especially on the classes that ended up being the main penalizing factors in our model.