AN2DL Homework 2

Valeria Amato
10641790

Emanuele Paci
10681377

Andrea Riboni
10699906

1 Introduction

The project's goal was to develop a time series classification model that categorizes samples into 12 groups. The dataset's actual meaning is unknown. In the following report, we will detail how we dealt with this situation in three stages:

1. Dataset analysis: focusing on its structure and its distribution
2. Testing of three base models (pure convolutional, LSTM and bidirectional LSTM)
3. Trial and error to discover the most fitting preprocessing function/algorithm and the corresponding best model

We chose accuracy as our primary metric, especially by keeping an eye on its trend on the validation set, in order to make things straightforward.

2 Approaches

2.1 Analyzing the dataset

The dataset is made up of 2429 sequences with 36 timestamps and 6 features each. The distribution of the eight classes is not uniform, as seen by the histogram below:

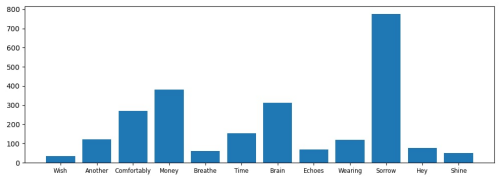

Figure 1: Class distribution

We calculated the dataset's class weights to penalize over-represented classes and thereby balance the distribution because our initial networks were unable to detect three separate classes (Accuracy: 0). Class weights allowed us to distinguish all classes, although at the cost of significantly lower overall accuracy. As a result, we decided to try out several networks in order to enhance overall accuracy and recognize all classes. We determined that class weights hampered model performance, so we opted to omit them. As a consequence, even though the overall accuracy on the final concealed testset was 71.02%, our final model could never detect the ”Breathe” class.

Since a bigger training set could end up in an increased chance of overfitting, we adopted the early stopping approach when training the networks.

2.2 Splitting the dataset

At the beginning, we split our full dataset into training, validation and test set, by dividing using the following ratio: 0.80 – 0.10 – 0.10. After analyzing the confusion matrix on our test-set, we realized that our model was overfitting due to the lack of data. The accuracy metrics from our test-set and the testset available on Codalab differed by 0.08/0.10. As a result, we deleted the test-set and adopted a 0.85 - 0.15 ratio to get additional data in the training-set.

Furthermore, to increase low accuracy and decrease overfitting, we used the ”Reduction On Plateau” learning rate reduction approach as well as the early-stopping strategy.

2.3 Models performances

Our initial attempt was a pure convolutional neural network, with a succession of Conv1D, MaxPooling, and BatchNormalization layers scoring up to 0.55 local accuracy. Next, we evaluated both LSTM layers and bidirectional LSTM layers: to our surprise, the bidirectional version performed worse than the unidirectional one and is more prone to overfitting, thus we opted to design networks with LSTM blocks rather than bidirectional ones.

2.4 Preprocessing

Many different preprocessing techniques have been tried, such as:

2.4.1 Feature transformations

Since the dataset contains a large number of features, one intriguing strategy is to assess whether each feature is relevant to the classification: for this reason, we attempted eliminating each feature one by one. When the fourth feature (index 3) was removed, strong results were achieved locally, but they did not reflect well on CodaLab's dataset.
The "inverse" has also been attempted by introducing additional functionality. We included the correlation coefficient, standard deviation, and mean determined on the attributes of each timestamp for each sample. This strategy once again provided the maximum local accuracy (0.76); however, the model that performed better in CodaLab's final phase did not incorporate this transformation.
Then, by plotting some samples, we were able to detect certain patterns with our eyes: we attempted to increase our network's identification capacity by rolling the data and changing features such that the maximum / minimum is at the beginning.

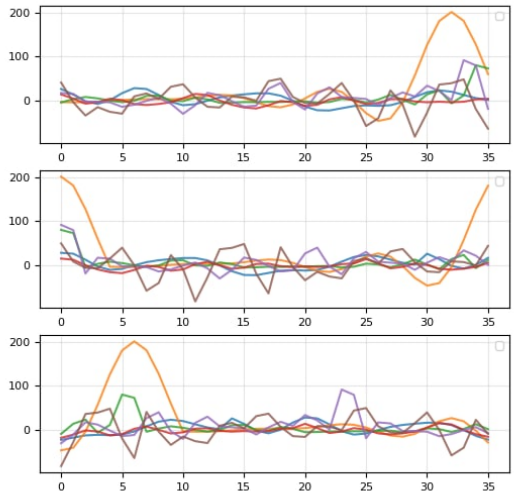

Figure 2: Top: Original, Center: Roll to max, Bottom: Roll to Min

The "Roll-to-max" version was the only one that offered us somewhat better results, allowing us to gain up to 0.77 accuracy in the local computation.

2.4.2 Scalers

Three alternative Keras scalers were tested, but no actual improvements were found:

- MinMaxScaler: scales and translates each feature individually such that it is in the given range on the training set, e.g. between zero and one
- RobustScaler: removes the median and scales the data according to the quantile range
- StandardScaler: standardizes features by removing the mean and scaling to unit variance

2.5 Model identification

We rationally chose to combine the two most accurate networks, pure convolutional and LSTM-based:

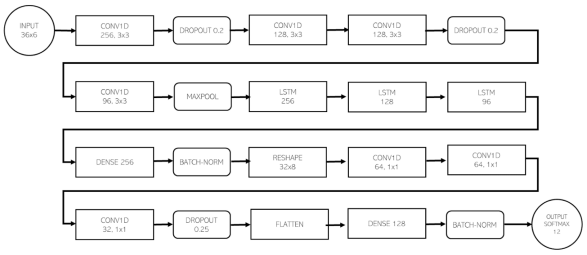

Figure 3: Model

The basic concept is to extract the first features using 1d convolutions, then layer on top of them a series of three LSTMs, and then categorize these high-level features using a series of dense layers and convolutions.

2.6 Other models

We tried a transfer-learning strategy utilizing the ResNet network, as proposed by other course attendees. The reasoning behind this was to treat the time-series as if it were a picture: the samples were reshaped and upscaled to be more network friendly. However, it performed worse than our greatest outcomes, so we decided to drop the concept.
Transformers are deep learning models that may assign different weights to different sections of the input data. We chose to test this concept because there are numerous applications of Transformers for Time Series. Initially, the results were clearly poor, but after a lengthy process of hyperparameter tuning, we were able to get up to 0.65 accuracy. Of course, this is insufficient, and because we were unable to strengthen the network further, we decided to abandon this concept as well.
In the first challenge our best result has been obtained through the ensemble of two different models. As we were able to obtain a high-accuracy model that was not able to correctly classify two classes and another model that, instead, had a slightly lower accuracy but had a non-zero accuracy over every class, the idea was to merge their predictions into a single one. We then tried to combine the classifications in two different ways: average and sum. While the sum performed better, the results were not as impressive and the obtained model was still not able to perform better than our best.

2.7 Hyperparameters tuning

During the development phase, we examined how changing different hyperparameters affected overall performance. Multiple testing have been performed manually as well as with keras-tuner. We concentrated on the following hyperparameters:

- Batch size: we noticed that using large batch sizes, such as 128 or 256, increases instabilities during training. We opted for a batch size of 64, which increased the accuracy of the gradient. A lower batch size did not allow the model to generalize correctly, as a consequence the general accuracy was lower.
- SGD versus Adam: the stochastic gradient descent created instabilities in our model and did not allow it to converge perfectly, so we decided to use to use Adam optimizer.
- Learning Rate: we chose a decreasing learning rate instead of a fixed one due to low accuracy of our model. Reduce the learning rate on the plateau we increased the overall overfitting, but the model could learn a little bit more with the original dataset

3 Conclusions

Although our results are similar to those of the other groups, we are not totally happy with them. Given the varying performances of each attempted model, the model ensemble may have gotten extra testing.