# ANNDL Homework 2 - Time Series Classification

Gradient Climbers
Federico Caspani, Andrea Cerasani, Matteo Citterio

December 23, 2022

## 1 Introduction and Assumptions

This report describes our work regarding the challenge 2 of the ANNDL course. The goal of this challenge is to model a neural network able to classify time series characterized by sequences of 36 samples and 6 features. Since we don't have any prior knowledge about the application domain of the data we dealt with, the following assumptions were made:

1. The dataset is given in terms of chronologically ordered samples
2. The distribution of the provided dataset and the online hidden dataset are similar
3. Some kind of augmentation can be performed without altering the core structure of the series

The chapters below will reflect our workflow, analyzing and describing our choices.

## 2 Data Preprocessing

Most of our work for this challenge has been dedicated to finding ways to improve our results by exploiting at most our data. To do so, we made use of various techniques that will be described below.

### 2.1 Data Standardization

To improve stability and performance of our model we tried various standardization methods. Among the others we experimented with MinMax Scaler and Standard Scaler but the one that performed best was Robust Scaler. This scaler uses statistics that are robust to outliers, removing the median and scaling each feature according to its Interquantile range, which is the difference between the $75^{th}$ quantile and the $25^{th}$ quantile, as shown in the following formula.

$$X_{scaled} = \frac{X - X_{med}}{X_{75} - X_{25}}$$

Under the assumption of working with datasets having similar distributions, we developed a manual implementation of RobustScaler that allowed us to collect, save and re-use statistics of our training set to standardize the hidden data of the competition. We found necessary to save our statistic indices because reapplying the scaling from scratch on the online test set did not yield the expected results.

### 2.2 Time series Augmentation

We experimented with various types of augmentation, using the library tsaug. We found empirically that the best types of augmentation to apply were:

- AddNoise, which adds random noise to time series
- TimeWarp, which randomly changes the speed of timeline
- Pool, which reduces the temporal resolution without changing the length.

```
heavy_augmenter = (
                  tsaug.AddNoise(scale=(0.5, 0.8)) @ 0.1
                  + tsaug.TimeWarp(50) @ 0.3
                  + tsaug.Pool(size=4) @ 0.3
                 )
light_augmenter = (
                  tsaug.AddNoise(scale=(0.05, 0.2)) @ 0.5
                  + tsaug.TimeWarp(2) @ 0.3
                  + tsaug.Pool(size=4) @ 0.3
                 )
```

Figure 1: The different augmentations applied to our final model

We also found out that the augmentation had different effects on the classes 4, 5, 7, 11 which reacted worse with a strong augmentation while others performed better in terms of F1 score. For this reason we decided to apply two different augmentations: a lighter one for the classes mentioned earlier and a stronger one for the remaining ones, as shown in figure 1.

### 2.3 Oversampling and Undersampling

One of the characteristics of our dataset was the class imbalance, with class 9 (”Sorrow”) being the most frequent class with 777 samples and class 0 (”Wish”) being the least frequent with 34 samples. To make up for this imbalance we tried to apply oversampling and undersampling techniques. In particular, both undersampling and oversampling effects on a class are as accentuated as the quantity of its series distances itself from the quantity defined by a uniform distribution over the totality of samples. The distribution of classes in the original case and after both resampling techniques is shown with the plots in figure 2.

Between the two techniques, we found oversampling to give the best result, probably due to the fact that undersampling removed important information about classes, lowering the performances.

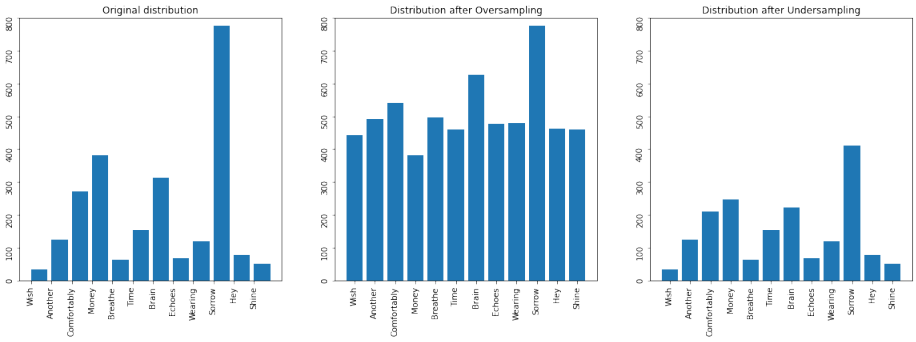



Figure 2: Classes distributions before and after resampling

### 2.4 Sliding Window

Under the assumption that the training dataset contained chronologically ordered samples, as a form of further augmentation, we developed an algorithm to build sliding window with random sampling of validation and test set.

As shown in figure 3, sequences are obtained by sliding a window of width 36 and configurable stride, through all subsequences of the training set obtained by randomly sampling the original dataset. To the obtained sequences we apply the same oversampling described before, in order to balance the classes. Our best model was obtained while using this technique.

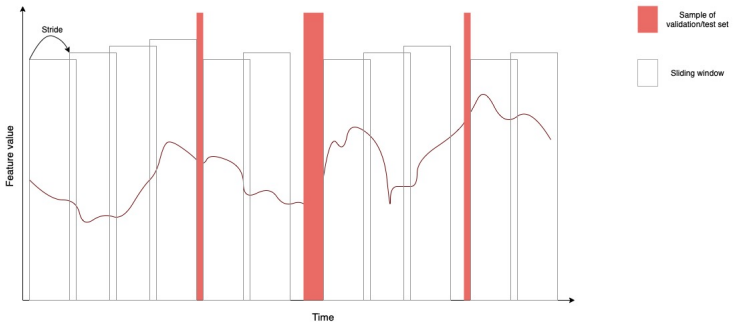



Figure 3: A scheme of our sliding window algorithm

## 3 Models

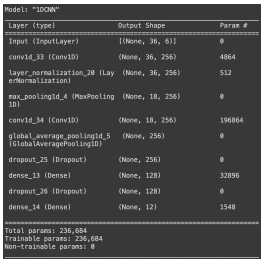
During this challenge various models were implemented and experimented with, which will be described below. The models were validated using Hold Out and in case we wanted a more accurate validation we used Stratified K-Fold with K=10.

### 3.1 Failed attempts

We tried plenty of different models that are in theory suited for Time Series Classification, such as a standard LSTM, a ResNet implementation and a reinterpretation of the Transformer model that is suitable for our problem’s category. Unfortunately, due to a possible misuse on our part or that the models were not suited for our specific problem, these models didn’t produce the results we were expecting, giving about 0.70 in accuracy both LSTM and ResNet and about 0.50 the Transformer model.

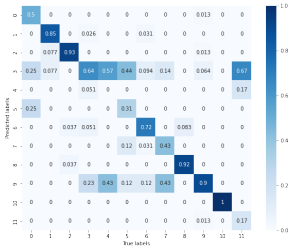
### 3.2 1D Convolution

Our best result has been obtained using a 1D Convolutional model. Its characteristics are summarized in figure 4a. As can be seen in the confusion matrix presented in figure 4b, the model performs very well with the majority of the classes, while struggles predicting classes 4, 5 and 7 due to the behavioral impact of classes 3 and 9.



```
Model: "1DCNN"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 Input (InputLayer)          [(None, 36, 6)]           0

 conv1d_33 (Conv1D)          (None, 36, 256)           4864

 layer_normalization_20 (Lay  (None, 36, 256)          512
 erNormalization)

 max_pooling1d_4 (MaxPooling  (None, 18, 256)          0
 1D)

 conv1d_34 (Conv1D)          (None, 18, 256)           196864

 global_average_pooling1d_5  (None, 256)               0
 (GlobalAveragePooling1D)

 dropout_25 (Dropout)        (None, 256)               0

 dense_13 (Dense)            (None, 128)               32896

 dropout_26 (Dropout)        (None, 128)               0

 dense_14 (Dense)            (None, 12)                1548

=================================================================
Total params: 236,684
Trainable params: 236,684
Non-trainable params: 0
_________________________________________________________________
```

(a) Model summary

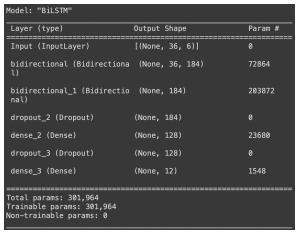


(b) Confusion Matrix

Figure 4: 1D Convolutional model

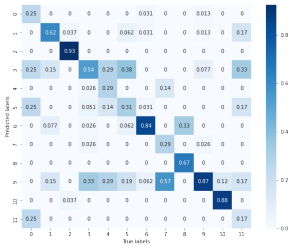
### 3.3 BiLSTM

A Bi-directional LSTM implementation has been tried and we noticed through the matrix in figure 5b that, even if its overall metrics were not the best we obtained, this model presented classification accuracies on the difficult classes that are more balanced with the other ones.



```
Model: "BiLSTM"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 Input (InputLayer)          [(None, 36, 6)]           0

 bidirectional (Bidirectiona  (None, 36, 184)          72864
 l)

 bidirectional_1 (Bidirectio  (None, 184)              203872
 nal)

 dropout_2 (Dropout)         (None, 184)               0

 dense_2 (Dense)             (None, 128)               23680

 dropout_3 (Dropout)         (None, 128)               0

 dense_3 (Dense)             (None, 12)                1548

=================================================================
Total params: 301,964
Trainable params: 301,964
Non-trainable params: 0
_________________________________________________________________
```

(a) Model summary



(b) Confusion Matrix

Figure 5: BiLSTM model

### 3.4 Ensemble

Given the fact that BiLSTM performed slightly better on telling apart the classes on which 1D Conv was struggling, we thought that a good solution could be an ensemble using a technique conceptually similar to Boosting: at first the sample was evaluated using 1D Conv, if the sample was classified as 9 it was evaluated again using a BiLSTM net trained only on the 4 classes 1D Conv was not able to distinguish properly, i.e. 4,5,7,9. Sadly the ensemble did not yield the result we hoped since the worsened performance of class 9 did not correspond to a sufficient improvement in the performance of the problematic classes.

## 4 Conclusions

After all the described attempts, our best model was obtained by feeding our data to a preprocessing pipeline composed of Sliding Window, Oversampling and two different tsaug augmentations and using the 1D Convolutional model explained above.

Moreover, after obtaining our best result using hold-out validation, we tried to further improve it by training this tuned model with all our available dataset by stopping the procedure when we reached the same training accuracy corresponding to the maximum validation accuracy of the hold-out model. This final attempt led us to an improved accuracy equal to 0.7494, our all-time best score.

We believe that thanks to our efforts to improve the F1 score on the most problematic classes we were able to score the $5^{th}$ best result in class 5, $6^{th}$ in class 0, $9^{th}$ in class 4 and $10^{th}$ in class 3.

Of course, further improvements could be obtained by tuning even more the augmentation in order to enhance the classes in which we performed worse.