Time Series Classification
Artificial Neural Networks and Deep Learning – A.Y. 2022/2023

Paolo Botta*, Teo Bucci† and Silvia Caresana‡
M.Sc. Mathematical Engineering, Politecnico di Milano - Milan, Italy
*Email: *paolo.botta@mail.polimi.it, †teo.bucci@mail.polimi.it, ‡silvia.caresana@mail.polimi.it*
*Student ID: *10612869, †10621873, ‡10630163*
Codalab Group: "Just3Neurons"

1. Introduction

The given dataset consists of 2429 samples with 6 features each, where every feature is a time series of $T = 36$ steps. Every sample belongs to one of the 12 different classes.

The task suggests to use Categorical Cross-Entropy as loss function. If y_{ij} is the output probability of class j for observation i, then:

$$\text{Loss}(y_{\text{pred}}, y_{\text{true}}) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{J} y_{\text{true},ij}\log(y_{\text{pred},ij})$$

where N is the number of samples and $J = 12$ is the number of classes.

As metric we used Accuracy as requested:

$$\text{Accuracy} = \frac{1}{N}\sum_{i=1}^{N}\mathbb{1}_{[\text{argmax}_j(y_{\text{pred},ij})=\text{argmax}_j(y_{\text{true},ij})]}$$

In addition, we also considered the F1–score per class to better understand the weaknesses of the model.

The dataset was split according to an 85:15 train-validation ratio and the results of this report are the ones we obtained on the validation set.

We started tackling the problem building different models and comparing their performances. Finally, based on the accuracy score and the F1–scores we selected the best performing one.

2. Data Pipeline

2.1. Pre-Processing

For a better modelling of the data we chose to apply a standardization. Since the different features span very different value ranges (see Figure 1), we normalized with respect to them by subtracting the mean and dividing by the standard deviation. The same transformation was saved and then applied on the validation and remote test set.

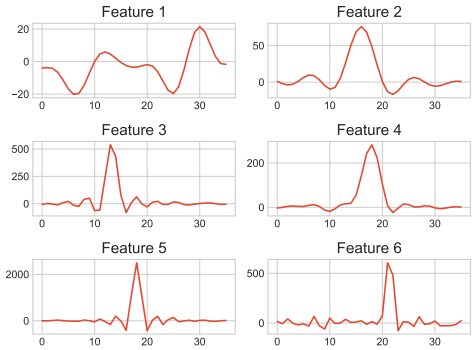

Figure 1. Plot of the 6 features from a random sample of the original dataset across time.

2.2. Data Augmentation

We were a bit reluctant to use data augmentation in this case, not knowing the underlying meaning of the features. Nevertheless we considered some transformations[1] as can be seen in Figure 2. After some trials, we kept **Scaling** and **Permutation**, since the others were either too computationally intensive or deteriorated too much the data.

1. Brian Kenji Iwana and Seiichi Uchida. "An empirical survey of data augmentation for time series classification with neural networks". In: *PLOS ONE* (July 2021). DOI: 10.1371/journal.pone.0254841.

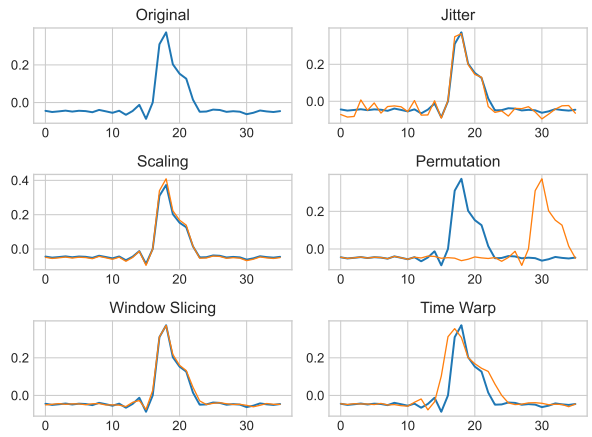

Figure 2. Examples of the different augmentations we explored. This example shows feature 4 from a random sample from the standardized dataset.

3. Models

In the following models the `Dropout` rate was always set to 0.5.

3.1. Vanilla LSTM

LSTM(128) (x2)
Dropout
Dense(128)

3.2. Bidirectional LSTM

BidirectionalLSTM(128) (x2)
Dropout
Dense(128)

3.3. 1D CNN

Conv1D + MaxPooling1D (x4)
GlobalAveragePooling1D
Dropout
Dense(512)
Dropout
Dense(256)

3.4. Keras tuner

After having an initial idea of the potential of the previous models, we turned to the Keras tuner, allowing it to explore different configurations in terms of `LSTM`, `Conv1D` and `Dense` layers. We used the Bayesian Optimization tuner for a maximum of 30 trials.

Conv1D (x3)
BidirectionalLSTM(256)
GlobalAveragePooling1D
Dropout
Dense(128)
BatchNormalization
Dropout

3.5. Inception Time

Inception Time[2] is a model based on Convolutional Neural Networks. It consists of a series of **Inception Modules** followed by a GAP Layer and a Fully Connected one (in our case 2 `Dense` layer with 256 and 128 units). Moreover, a residual connection is added at every third inception module in order to mitigate the vanishing gradient problem. The Inception Module (see Figure 3) consists of 4 layers:

- Bottleneck Layer that reduces the dimensionality and improves generalization;
- Parallel Convolutional Layers of different size acting on the same input feature map;
- MaxPooling that improves the invariance to small perturbation;
- Depth Concatenation Layer that concatenates the output of convolutions and MaxPooling.

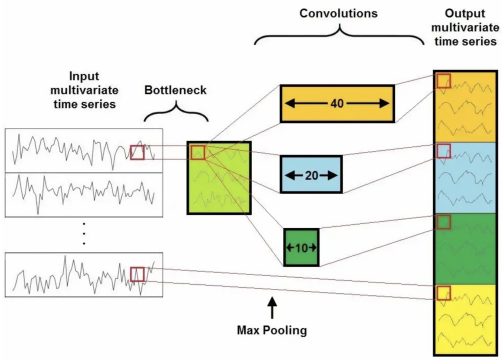

Figure 3. Inception module.

2. Marco Del Pra. *Time Series Classification with Deep Learning*. https://towardsdatascience.com/time-series-classification-with-deep-learning-d238f0147d6f. [Online; accessed 2022-12-22]. Nov. 2020.

4. Training techniques

In terms of training, we used for all models:

- Batch size: 16
- Optimizer: Adam
- Learning rate: 0.001
- Callbacks: **EarlyStopping** with patience 20, **ReduceLROnPlateau** with patience 5 and factor 0.5

4.1. Class imbalance

Class imbalance played an important role, as can be seen in Figure 4.

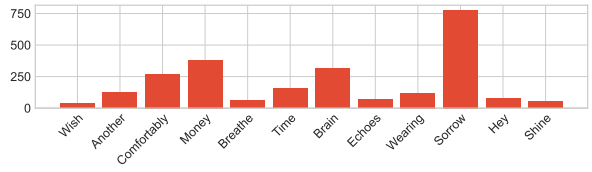

Figure 4. Number of samples in each class.

The approach we followed was to train 12 different Inception models, each one very strong in classifying a single class. To do so, each time we gave weight 10 to the class in consideration and weight 1 to all the others. Finally, these models were put in an ensemble, together with the base Inception, by taking the simple average.

5. Model choice

We compared the different models, both with and without augmentation (except the Keras tuner, which was too computationally intensive to run with augmentation too). Even though augmentation improved the accuracy in almost all cases, this didn't happen with the final Inception of Section 4.1. However, without augmentation it outperformed all the others by 4%. Therefore, that is the final model we choose.

	Std.	Std. + Aug.
Vanilla LSTM	66.30%	68.77%
Bidirectional LSTM	69.59%	74.79%
1D CNN	73.42%	74.52%
Keras tuner	74.52%	-
Inception Time	74.52%	76.16%
Inception Ensemble	**80.55%**	76.99%

6. Conclusion

6.1. Performance

The results of the final model both on the train, the validation and the remote test set are as follows:

Train acc.	Val acc.	Test acc.
95.74%	80.55%	74.00%

The F1–scores are:

Class	0	1	2	3	4	5	6	7	8	9	10	11
F1	1	0.67	0.99	0.65	0	0.67	0.90	0.50	0.97	0.82	1	0.59

The final confusion matrix can be seen in Figure 5.

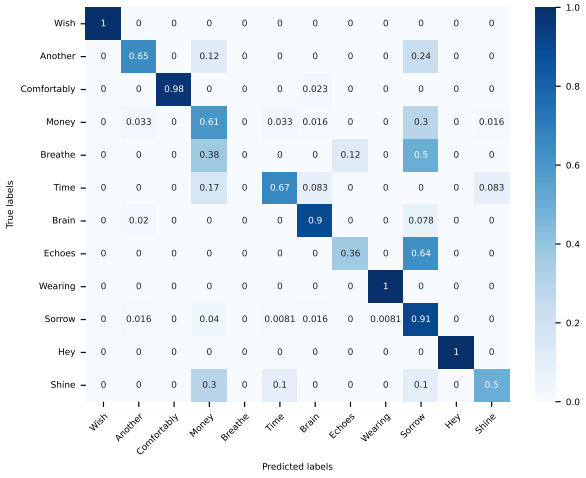

Figure 5. Normalized confusion matrix of the final model.

6.2. Further Developments

A few aspects of our work might use some improvement. We could do some more preliminary analysis on the dataset to understand more suitable preprocessing steps.

Moreover, we are not yet satisfied with how we dealt with class imbalance, specifically because class `Breathe` got an F1–score of 0 in almost all models, except the one obtained with the Keras tuner. This deserves some attention.

Speaking of attention, we quickly tried using `Attention` layers, but we weren't quite used to them and didn't manage to reach results as good as with the classical techniques. Therefore, we sticked to those for the time being, but we're sure that some improvements can be made from the architecture side.