Time Series Classification Challenge 2022

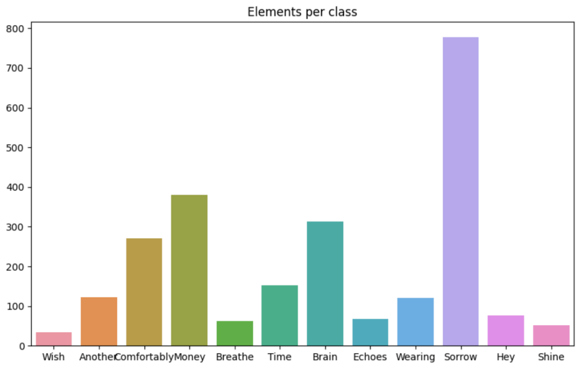
TEAM: ALPOLIZZATI

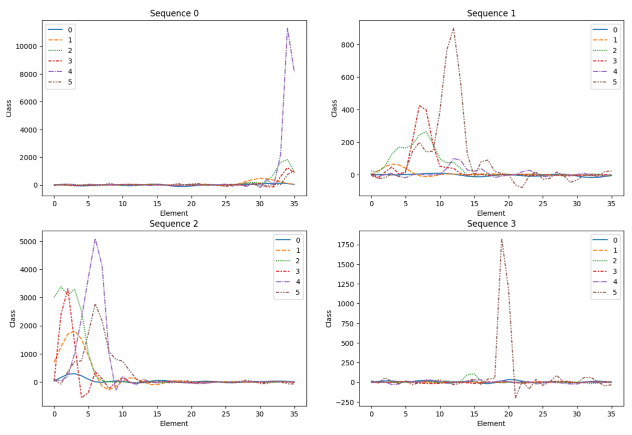
Andrea Aiello, 10863133
Dario Cavalli, 10820532
Claudio Moriello, 10869147
Paolo Basso, 10783951

1 Dataset Specification

The dataset provided consists of a total of 2429 time series belonging to one among 12 classes. Each sample is divided into 36 units, or sequences, characterized by 6 continuous features. No additional details were given about the dataset origin, so some deeper analysis was required. It was found that samples were ordered by class and sequences were neither continuous nor overlapping, as shown on the right image below.

The most relevant finding was the great unbalance between classes, with a third of the total samples belonging to a single one, leaving others severely outnumbered with even less than 100 time series per class, as shown in the left image below. This had a great influence on the classification task and many attempts were made to mitigate this issue, from oversampling of sub represented classes to custom weighting of samples, some of this techniques will be discuss further later.





2 Dataset split, encoding and prepossessing

Given the high imbalance in the dataset, we used the **stratified sampling technique** for splitting into train, validation and testing. We kept 5% of the data for testing and we split the remaining data in validation (15%) and train (85%) data.

The data was **standardized feature-wise**, meaning that each of the 6 features was standardized independently from the other features. We also tried standardizing all the features together, normalizing feature-wise and normalizing all the features together but features-wise standardization yielded the best result.

Finally, the classes were encoded using the **one-hot encoding strategy**.

We tried other preprocessing methods to fight against the high imbalance of the dataset:

- *Oversampling the minority classes*: we tried increasing the samples of the minority classes extracting random samples with replacements

- *Class weighting*: we tried weighting the different samples during training, putting more weight on the minority classes samples

However, these techniques did not bring good results and we ended up not using them in our final model.

3 Experiments

During this challenge, we tried a lot of different models to find the best possible one. We started by creating three simple neural networks, one using CNN layers as feature extractors, the second using bidirectional LSTMs, and the last using a mixture of the previous two. We obtained good results by stacking three CNN layers and two BLSTM layers, both mixed with dropout and batch normalization layers, and finalizing the classification with two fully connected dense layers. Regarding the choice between LSTM and GRU, the first architecture was the one with the best results.

We made another attempt with Transformers, as described in [5]. We began by implementing a base model of Transformers that took advantage of the MultiHeadAttention layers of keras. We tried different variations of its hyperparameters and worked with different numbers of Transformer blocks, although the best results were obtained with only one block and a small number of heads.

In all these experiments, we also varied the number of features used and the class weights. In most cases, we obtained the best results when we used only five of the six features and reduced the weight of the ”Sorrow” class by a factor of 1.5-2. Moreover, we tested the use of a smaller number of time steps for the samples by splitting each sample into different parts and varying the length of each segment from 6 to 18.

We also tried to add some data augmentation by using the library *tsaug* [3]. We experimented different methods proposed by this package such as: Crop, Drift, Reverse, TimeWarp and AddNoise. But even with different combinations and very small parameters the results got worse.

To test these models, we divided the test samples into segments of the same length as the input of the models and then averaged the prediction for each subsegment for each sample. Although this type of ensemble significantly increased test accuracy, we did not find a sufficiently good model in the training phase.

The best results overall were achieved using another type of ensemble technique, where we simply took some of the best models on the validation set, concatenated their results, and used an additional dense layer to make the final prediction, which increased both accuracy and precision. Good results using ensemble techniques were also obtained with the average of the predictions or majority voting. Unfortunately, we were unable to save the model due to a bug [4] and therefore did not test it on Codalab.

3.1 Keras Tuner

Given the short training time per epoch and the amount of hyperparameters to tune, the use of Keras Tuner [2] was found useful to optimize the selection of most parameters such as: number of units in each dense layer, dimension of convolutional filters and kernel size, learning rate, LSTM units per layer, dropout and its rate.

Tuning was first performed on 100/200 epochs to find the most promising parameters and then the top 5/10 models where retrained until early stopping triggered. The optimal hyperparameters choice will be illustrated in the final model section.

4 Final model

4.1 Hyperparameters

- *BATCH_SIZE*: we used mini-batch gradient descent with a batch size of 32 samples.

- *MAX_EPOCHS*: we used a maximum of 700 epochs, but due to early stopping this maximum was never reached.
- *EARLY_STOPPING_PATIENCE*: we used a patience of 80 epochs because we realized that the model tended to overfit quickly.
- *OPTIMIZER*: we used Adam Optimization because it is faster, requires less memory and achieves good performances.
- *LEARNING_RATE*: we used a learning rate of 1e-4 in order to avoid fast overfitting, so we gave the model the time to adapt itself to the dataset in small steps.
- *LOSS*: considering that it is a multi-class classification problem, we used Categorical Cross Entropy in order to have a probability over all the classes for each time series.

4.2 ResNet

To get our best results we started from a ResNet to which we later added some additional layers. The base network is a relatively deep Residual Network composed of 9 convolutional layers divided into three residual blocks followed by a GAP layer that averages the time series across the time dimension, and a final softmax classifier whose number of neurons is equal to the number of classes in a dataset. The main characteristic is the shortcut connection between consecutive convolutional layers in order to reduce the vanishing gradient effect. The number of filters for all convolutions is fixed to 64 and the filter's length is set to 8, 5 and 3 respectively for the first, second and third convolution. Each convolutional layer is followed by a LayerNormalization and a Relu activation function. [1]

4.3 Model Layers

As previously mentioned, we started from a basic ResNet and subsequently added other layers both at the top and bottom of it.

At the top of it, the first layer is the *Input* layer, to instantiate a Keras tensor. Then, we added two *Dense* layers with 2048 and 1024 units respectively; in both layers we used a "relu" activation function, a "He uniform" variance scaling initializer and in order to limit overfitting, we constrained the model through *L2* regularization. After that, we added two *Bidirectional LSTM* layers by wrapping the LSTM layers into the Bidirectional ones with 256 units for each LSTM. Before the base ResNet we also added a *Dropout* layer with a rate of 0.4 to reduce overfitting.

After the GAP layer and before the final softmax classifier we added other 4 *Dense* layers with 2048, 1024, 512, 128 units respectively and with the same parameters as before.

5 Comparison between models

To conclude, we compare the results obtained by the best models, for each technique used. The validation accuracy is based on the split of the dataset in train and validation set, using as validation a percentage of the dataset varying from 10% to 20%, whereas the test accuracy is taken from the results of the "development phase" of the challenge.

Model	Validation Accuracy (%)	Test Accuracy (%)
Ensemble	79.28	\
ResNet	78.08	73.52
CNN+BLSTM	76.99	71.46
Transformer	74.12	\

Table 1: Validation and Test accuracy comparison

References

[1] Hassan Ismail Fawaz et al. “Deep learning for time series classification: a review”. In: *Data Mining and Knowledge Discovery* 33.4 (Mar. 2019), pp. 917–963. DOI: 10.1007/s10618-019-00619-1. URL: https://doi.org/10.1007%2Fs10618-019-00619-1.

[2] Tom O’Malley et al. *KerasTuner*. https://github.com/keras-team/keras-tuner. 2019.

[3] *Time Series Augmentation API*. URL: https://tsaug.readthedocs.io/en/stable/index.html.

[4] *Unable to save ensembled tf.keras.Model instance 39567*. URL: https://github.com/tensorflow/tensorflow/issues/39567.

[5] Shazeer N. Vaswani A. et al. *Attention Is All You Need*. 2017. DOI: 10.48550/arXiv:1706.03762v5. URL: https://arxiv.org/abs/1706.03762.