Artificial neural network and deep learning - Second Challenge 2022

Time Series Classification

Giulia Forasassi, Manuel Montecalvo, and Diego Martin

Politecnico di Milano

December 23, 2022

1 Introduction

Time series classification is a Machine Learning problem. Given a fixed set of classes, time series classification is the task of predicting which label to assign to an input time series. In this case, we solve the problem using Deep Neural Networks.

In this project, we analyzed the data and experimented with different techniques for training the network, including convolution neural networks, recurrent neural networks, and ensemble techniques.

2 Data analysis

The training dataset that was provided consists of a set of 2429 time series. They belong to 12 different classes (Wish, Another, Comfortably, Money, Breathe, Time, Brain, Echoes, Wearing, Sorrow, Hey, Shine). Each time series has 36 time intervals and for each of them, 6 features are considered. (Figure 1) The distribution of samples in the dataset is very unbalanced since there is a different number of samples for each of the classes. (Figure 2)

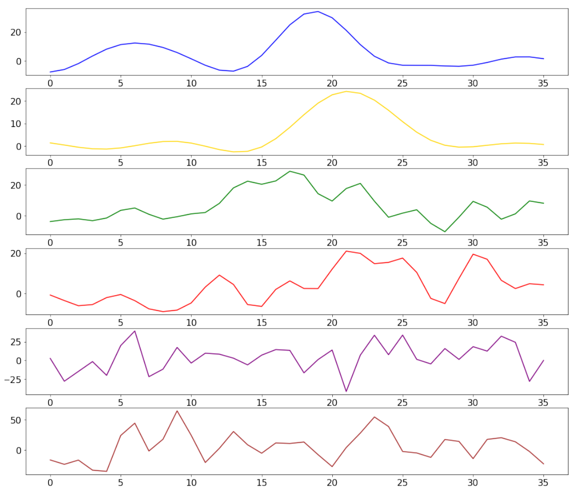


Figure 1: Example of a time series

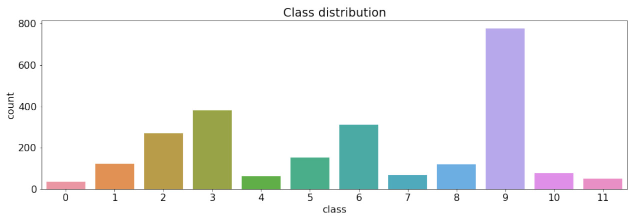


Figure 2: Classes distribution

3 Data preparation

3.1 Data splitting

We started by splitting the dataset into training and validation sets using a fixed ratio of 80 and 20 and taking random elements from the initial dataset.

To overcome the classes imbalance problem we used two alternatives methods: oversampling and class weighting. With oversampling, time series belonging to classes with fewer samples are duplicated an arbitrary number of times to reach a minimum number of samples per class. On the other side, class weight allows giving different importance to prediction errors on a per class basis. By increasing the weight of a class the lower the number of its samples, the class imbalance problem is mitigated. However, being the dataset so unbalanced, we noticed that the oversampling technique worked better than the class weights.

3.2 Data preprocessing

To process the data we chose to apply standardization, as it gave better results than normalization. In particular, the `Robust Scaler` was used, which scales features using statistics that are robust to outliers.

3.3 Data augmentation

To prevent overfitting and give more robustness to the network, we have considered it appropriate to effectively increase the number of training samples provided by introducing data augmentation. Using `tsaug`, a Python package, we have applied random transformations creating new versions of the same time series. We added random noise and dropped out values of some random time points in the time series.

4 Models and techniques

The models selection and construction followed three main approaches: custom deep convolutional neural networks, recurrent neural networks using the LSTM and BiLSTM layers, and ensemble technique on previously trained models.

4.1 Base 1D-CNN model

The first model we tried is a CNN, made by a sequence of 1D-Convolutional and 1D-MaxPooling layers, followed by a Dropout layer to reduce overfitting, with a couple of Dense layers at the end of the network. The network is 7 layers deep with Relu activation function after each convolutional layer and the Softmax activation function in the final layer. High levels of accuracy were not achieved with this basic model so we decided to improve it and implement new models using the recurrent neural networks.

4.2 Recurrent neural networks

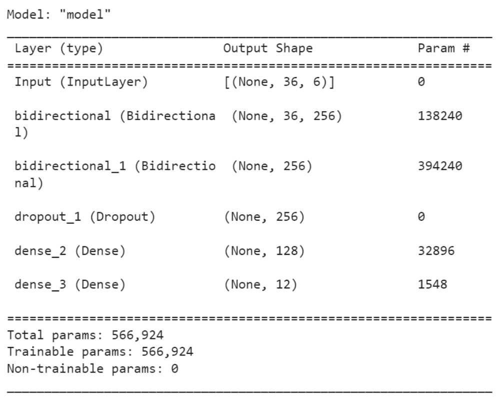
The second approach we tried is the use of recurrent neural networks where connections between nodes can create a cycle, allowing output from some nodes to affect subsequent input to the same nodes. In particular, we worked on LSTM and BiLSTM models.

It's made by a sequence of LSTM layers with a final Dropout as a features extractor. Then the classifier is created by some Dense layer with Relu activation function and the Softmax activation function in the output layer. The structure of the BiLSTM is very similar to the LSTM one, but it achieved better results.

In the Figure 3 the structure of the network with the best model with recurrent structure is reported.

4.3 Resnet

The network structure is made up of 3 macro blocks. Each block consists of a sequence of three 1D-Convolutional layers followed by a Batch Normalization layer and Relu as activation function. To create the skip-connections, each block sum its input to its output: these skip connections enhance the flow of information to deeper layers, and the backward propagation of the gradient. After the three blocks, there is a GlobalAveragePooling layer, followed by a Dense layer with Sotmax as activation function.



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

 bidirectional (Bidirectiona  (None, 36, 256)          138240
 l)

 bidirectional_1 (Bidirectio  (None, 256)              394240
 nal)

 dropout_1 (Dropout)         (None, 256)               0

 dense_2 (Dense)             (None, 128)               32896

 dense_3 (Dense)             (None, 12)                1548

=================================================================
Total params: 566,924
Trainable params: 566,924
Non-trainable params: 0
_________________________________________________________________
```

Figure 3: network summary

4.4 Ensemble technique

To improve our performance we used the ensemble technique that combines base models to produce a better predictive one. This has been done by sending the input time series, with the corresponding preprocessing, to every single model and then making the average of the output predictions. In the Figure 4 we can see the ensemble model that produced the best results.

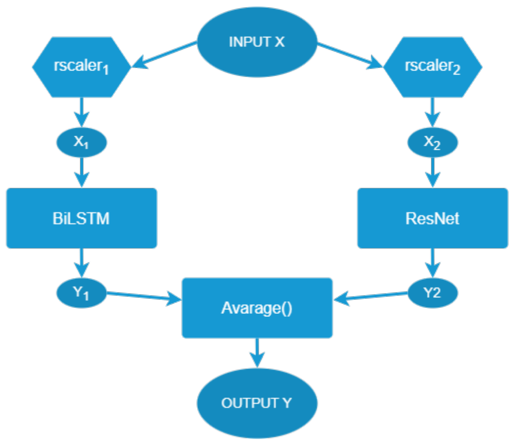


Figure 4: Ensemble model

4.5 Best models

In the end, the model that achieved the best result was the Resnet. While, if we consider also the ensemble technique explained above, the best model was the one combining 2 networks respectively based on BiLSTM and Resnet.

5 Training

The training was done entirely on Google Colab and was performed by measuring the accuracy and loss functions on the validation set and tracking the potential overfitting over the training data.

To measure the performance of the classification models was used the Cross Entropy or Sparse Cross Entropy Loss functions setting a number of epochs between 200 and 500. Then we tested the Focal Loss function which applies a modulating term to the cross entropy loss in order to focus learning on hard misclassified examples but it didn't help to achieve better results.

Also, we experimented with different batch sizes (64 or 128) and a number of patience equal to 20 for the early stopping technique.

In some models, we optimized using the Adam method with the default learning rate, whereas in other models we tested a dynamic learning rate using the `ReduceLROnPlateau` function which starting from an initial value, the learning rate was reduced when a metric has stopped improving.

6 Results

This section shows the results of the most successful models used for the classification task, and the accuracy values are reported in the table below.

Model	Accuracy
(1) LSTM	0.6457
(2) BiLSTM	0.7019
(3) Resnet	0.6927
(4) Conv1D	0.6921
(5) Ensemble model (2 + 3)	0.80

Table 1: Model accuracy on local test data

As we can see in the accuracy data (Table 1), the best results were obtained by combining the BiLSTM and Resnet models.

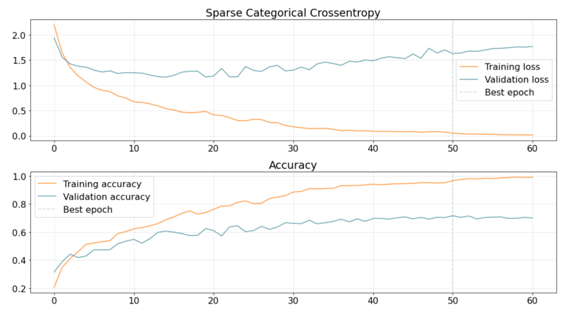


Figure 5: `BiLSTM` cross-entropy plots and accuracy

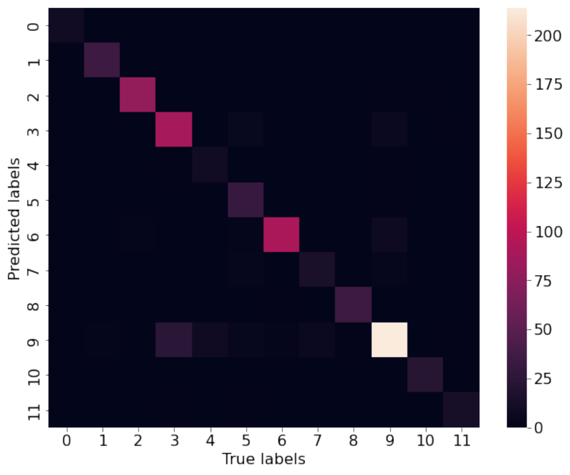


Figure 6: `Ensemble model` confusion matrix

Following, we report additional metrics and results from the best models on local test data. The confusion matrix (Figure 6) shows classification performance on the local validation set performed by the Ensemble model. While the loss and accuracy metrics (Figure 5) show the training speed and metric trends of the BiLSTM model without considering the ensemble technique.

7 Conclusions

Time series classification is a difficult task, since it's usually not possible to take advantage of pretrained models. Our specific dataset has two further challenges: it's highly unbalanced and also quite small, two characteristics that usually lead to overfitting.

We tried several approaches, some based on Recurrent Neural Networks, others on Convolutional Neural Networks, and also ensembles of them. The model that obtained better results is an ensemble of a RNN a CNN, in particular of a BiLSTM and a ResNet: this model was probably able to take the best of both worlds.

Tools

- Tensorflow
- Keras
- Scikit-Learn
- Jupyter notebook