Time Series Classification - Homework 2

Team: gliANNidoroDLgrandereal

Enrico Sarneri - 10805002
Luigi Umana - 10614228
Giovanni Rizzo - 10681879

December 22, 2022

1 Introduction

This report aims to show the results of the Time Series Classification Project of Artificial Neural Networks and Deep Learning course. The aim of this project is to solve a time series classification problem by designing the most efficient algorithm and exploiting what we have learned during the course. The given time series are characterized by 2429 sequences (samples) composed of 36 elements (timestamps) with 6 attributes (features) each (2429x36x6), divided into 12 classes.

2 Solution

We started by doing exploratory data analysis and, after loading the provided .zip file, we noticed that the sequences were not perfectly balanced and scaled. In particular, class 0 was presenting more clearly fewer occurrences with respect to the other classes. We decided to face all these aspects in a second moment. We initially considered the models faced during the labs i.e. a 6-layer Long Short Term Memory (LSTM) Neural Network, a 6-layer Bidirectional LSTM (BiLSTM) Neural Network, and an 8-layer 1D Convolutional Neural Network (CNN).

Without performing any kind of pre-processing, we tried to find the best combination of parameters and hyperparameters in order to get the best accuracy. In this case, we reached a really low validation accuracy of $\approx$0.15. As a first thing, we decided to implement the train-validation split outside the model.fit using the sklearn train_test_split function with an 80% - 20% split and stratification. After that, the validation accuracy improved by $\approx$15%, and, more importantly, the validation loss started having a decreasing trend during the fit. As a consequence, we decided to keep the splitting in this way from now on.

2.1 Scaling

Since we didn't reach any improvement after several attempts, we thought that it could have been useful to scale the original data and then apply the train-validation splitting. At first, we used the MinMax Scaler, which is able to scale each feature to a given range, scaling on 0-1 range. Then we tried StandardScaler, which standardizes features by removing the mean and scaling to unit variance, and RobustScaler, which removes the median and scales the data according to the quantile range. Unfortunately, none of them gave any improvement to any of the models we tried, both in terms of validation accuracy and validation loss. Consequently, we didn't use scaling in our dataset by keeping it as the original one.

2.2 Class Weights

Analyzing the results by classes, after the first models, we realized that one of the reasons why we struggled to obtain optimal performance, was due to the misclassification of some classes. In order to solve this problem, we thought that it could have been useful to implement the class weights. In this case, the loss function is influenced by assigning relatively higher costs to examples from minority classes. We adopted the compute_class_weight method from sklearn.utils to estimate class weights, using 'balanced' as parameter. We trained all the defined models as well as the following ones both with and without weights, but, unfortunately, the 'with' options didn't give us better results compared to the 'without' ones.

2.3 More Complex Models

Given the scarce effectiveness of the previous networks, in this step, we went on by designing new models, starting by increasing the complexity of the foregoing ones. We initially incremented the size of the filters

(256 before and 512 later) of the LSTM Neural Network, which gave us little improvement in the order of ≈3-4%, and we also tried to add one more Dense Layer in the classifier which didn't help much more.

To further increase the size of the network, we decided to implement a CNN-LSTM model. It basically uses CNN layers for feature extraction on input data combined with LSTMs to support sequence prediction. We started with two Conv1D (with low numbers of filters), a MaxPooling1D, a Flatten, and an LSTM layer, leaving as a classifier the one of the base models presented before. This model started having higher validation accuracy, fluctuating around ≈0.50. Then, we tried to slightly add more complex features, like the number of filters in the Conv1D layers and LSTM layers (64, 128, 256, 512, 1024), as well as in the Dense layers for the classifier. Also, we modified the number of feature extraction layers, trying 2, 3, 4 Conv1D layers both in sequence with a MaxPooling1D at the end, and in couples with a MaxPooling1D after each of them. We then changed the number of sequence prediction layers using 2 and 3 and kept the same number of units (64, 128, 256, 512, 1024). Moreover, we made several trials also using different sequence prediction layers as BiLSTM, GRU, BiGRU. However, they didn't give us better results than the LSTM.

2.4 Parallel Architectures

At this point, we decided to design parallel architectures to explore the results. Firstly, we built two simple parallel branches constituted by two Conv1D layers and one LSTM layer, taking the same input, and merged together before a classifier made of 2 Dense layers. This model reached a validation accuracy of ≈0.60 so we started thinking it could have been improved following the same path. We slightly increased the complexity of this model by adding more Conv1D layers and filters, 'playing' with dropout, and reducing overfitting by decreasing the learning rate but it didn't improve much more than before.

So, we tried to change a bit the architecture by making parallel branches of just Conv1D layers while leaving the LSTM and the classifier after the merging of them. In this case, the accuracy immediately reached higher values moving around ≈0.65 of validation accuracy. We kept trying to optimize this model. We used two LSTM layers, two Dense classification layers, and more filters per layer (512, 1024) and we started having ≈0.70 validation accuracy. In this case, we noticed that the use of MaxPooling1D, GlobalAveragePooling, Flatten and BatchNormalization layers affected negatively the results, so we decided to not use any of them, keeping the model much 'simpler'. Any other attempt to increase the complexity, like adding more parallel branches, more LSTM, Conv1D, and Dense layers, didn't reach better results.

We also thought that adding noise to the samples could have improved the results. We added GaussianNoise layers in the classifier and in the Conv1D branches both together and separately but it didn't help, so we kept the network as before.

2.5 ResNet

After that, we decided to try a ResNet architecture in order to see if it was able to perform better than a parallel CNN-LSTM architecture. We started by using the ResNet Conv1D architecture presented in the paper of Hassan Ismail Fawaz et al. [1], trying both with and without scaling as well as different batch sizes. In this case we never reached more than ≈0.55 validation accuracy.

A second attempt we made, was by using the Keras Applications ResNet50, ResNet50V2, ResNet152, and ResNet152V2. Since the models require a 3D shape in input, we performed initial Reshape and Resize operations from the (36, 6) input in order to get a shape in the form of 32x32x3. We tried both by training the models from zero as well as by performing transfer learning and fine-tuning using 'imagenet' weights and by developing custom classifiers. Unfortunately, in all these cases we reached at most a validation accuracy of ≈0.68, slightly lower than the previous parallel CNN-LSTM model.

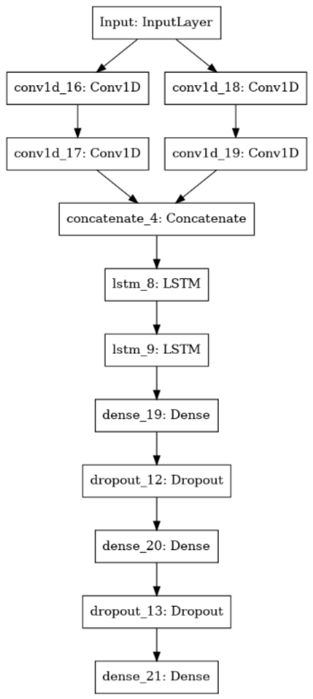
[1] https://link.springer.com/article/10.1007/s10618-019-00619-1

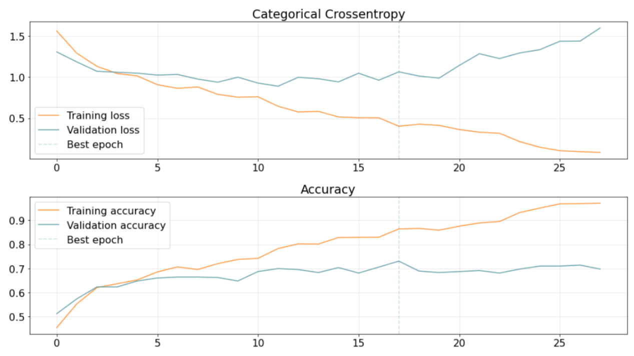
2.6 Feature Selection and PCA

As a possible contribution, we also thought about applying Feature Selection and Program Component Analysis. In the first case, we basically iterated among all the 6 features removing a different one each time. We did this for the model which up to that moment was giving the best results (Parallel CNN-LSTM). The best results were given by removing feature 0 and feature 2, however, the validation accuracy was still lower than the original case. As a sort of counter proof, we also decided to apply Principal Component Analysis (PCA) so as so the extract the most relevant features. We performed it by flattening the first two dimensions of the original dataset (2429, 36) and exploiting the sklearn.decomposition.PCA method. In this case, we standardized the data through StandardScaler and we selected the number of features in order to have a cumulative explained variance between 0.9 and 1.0. Still in this case, the results performed worse than the original one.

3 Final Model: Parallel CNN-LSTM

Here we present and illustrate the skeleton of the model that reached the best results together with its correspondent accuracy and losses.


Input: InputLayer
conv1d_16: Conv1D
conv1d_18: Conv1D
conv1d_17: Conv1D
conv1d_19: Conv1D
concatenate_4: Concatenate
lstm_8: LSTM
lstm_9: LSTM
dense_19: Dense
dropout_12: Dropout
dense_20: Dense
dropout_13: Dropout
dense_21: Dense


Categorical Crossentropy
Training loss
Validation loss
Best epoch
Accuracy
Training accuracy
Validation accuracy
Best epoch