Challenge 2

Gonzalez-Espitia, Juan Carlos* (10779264)
Petkovic, Nicola† (10821894)
Tonnarelli, Marco‡ (10802888)

2022/2023

POLITECNICO
MILANO 1863

CONTENTS

1 INTRODUCTION

The second challenge of the Artificial Neutral Networks and Deep Learning course aims to the classification of multivariate time series, which is a problem that involves building classification models that are able to classify an ordered sequence of real-valued attributes. The difference between time series classification and the multivariate case resides in the fact than in the latter we are dealing with a multi-dimensional time series: in this case, the dataset provided is made up by 2429 samples, and each sample has 36×6 values, which means that there are 6 features with 36 values each. Overall, the series belong to 12 different classes.

As soon as the challenge has opened, what we did first was reviewing the literature on the topic of the challenge and understanding the dataset; the latter task turned out to be more difficult than expected, since little or no information about data has been provided.

It was immediately clear to us that Recurrent Neural Networks would have played a crucial role for solving our problem, since they are capable of implementing a memory thanks to the presence of cycles between the layers, which is fundamental when processing variable-length input sequences.

In general, time-series classification is a well-studied task [1] and many algorithms are available for solving this kind of problems; however, most of the studies we found focus on specific problems, in particular audio sequences classification. This is a problem, as we did not know what kind of time series we were dealing with: initially, we spent a lot of time trying to understand what kind of data the available sequences represented and, given that the labels for each class recalled Pink Floyd's songs titles, we thought that we were dealing with 36 sampling-time steps long audio sequences. However, we could not be sure of the meaning of the 6 features. They could represent the same sequence filtered is 6 different ways, obtained by applying a filter bank to the original audio signal, or they may also represent the Mel-Frequency Cepstral Coefficients. Nonetheless, all of this assumptions on the nature of the dataset were useless, since we could not prove them. Instead, if we were able to prove what kind of sequence we were dealing with, we might have been able to exploit different characteristics of data: for instance, if we were actually dealing with a 36 samples-long part of audio track, applying the Fast Fourier Transform algorithm to them could have been useful to reveal hidden features in the frequency domain. Unfortunately, this was not possible.

So, our research focused on finding procedures which have been shown to work well with time series in general [2, 1]. In particular, Ref. [2] provides a novel approach which, instead of relying on recurrent layers, it entirely relies on attention layers that allow for great parallelisation and create global dependencies between the input and the output. On the other hand, Ref. [1] provides a deep insight on time series classification, describing how CNNs, in particular 1 dimensional ones, can effectively deal with this kind of problems.

2 DATA PRE-PROCESSING

The preliminary inspection of the dataset showed us how unbalanced the dataset is: most of the samples belong to one class, while other classes as almost no data; for instance, the class labeled with "0" is represented by just 34 samples with respect to the class "9" which is just shy of 800 samples. This problem was even worse than the first challenge.

One of the reasons why we used much of our time understanding the meaning of the available data is that we wanted to perform some specific augmentation for audio signals [3, 4]. In particular, the most promising ways to augment data consist of adding noise on the signal, shifting time, changing the pitch and the speed of the signal, window warping and slicing. Many methods exist for audio signals data augmentation, but since we were not sure about the nature of our data, we avoided wasting time on developing functions that implement such techniques, since they might have worsen the conditions of the dataset.

Nevertheless, during our research we found a Python package, namely tsaug, that is specifically developed time series augmentation and it implements many of the methods discussed before. Actually, we tried to augment data using this package, by making for each sample belonging to a desired class a random augmentation such as resizing, reversing and addition of noise, trying to make it as mild as possible to avoid ruining the dataset. However, we quickly discovered that using the augmented dataset produced worse results -

*juancarlos.gonzalez@mail.polimi.it
†nicola.petkovic@mail.polimi.it
‡marco.tonnarelli@mail.polimi.it

independently from the model used - than using the original one, so we discarded this option.

Solving data unbalance remained our priority, since was the main issue our models that, as we discuss in the next sections, performed poorly. After discarding the aforementioned options, we went for a custom solution. We implemented a function that adds noise to each feature of given sample, based on a percentage of the covariance of the whole 36×6 matrix. The reason why we decided to chose this technique is that we wanted to be sure that the noise added to the samples was coherent to the class they belong to: so, the augmented sample is similar to the trend of the belonging class.

The remaining problem was that augmenting too much the dataset worsen the accuracy of all our models and after tuning some hyper-parameters of our augmenting function, we found that the best was to augment a subset of the classes, i.e., the one with the least number of samples (classes numbered 0, 11, 4, 7, 10 have less than 80 samples each), by making an augmented copy for each sample available. For instance, augmented class number 0 grew from 34 samples to 68 samples.

The last approach we tried on the dataset, in order to achieve better accuracy scores, was to perform Principal Component Analysis to remove some features. However, PCA did not discover any hidden information on the features since all of them cover most of the covariance.

Eventually, another pre-processing method we performed on data was scaling the dataset, i.e., applying some transformations such that, for instance, mean is removed and the variance is set to 1: this is what the Standard Scaler module of SciPy package does, but it was not the best performing one. Instead, after trying to apply MinMaxScaler with worse results, in the end we assessed that the best scaler was Robust Scaler, which is more robust in presence of outliers in the dataset.

3 MODEL CHOICES

Given the time-series nature of the data, we wanted to test the properties and benefits of sequential data by implementing recurrent neural networks (RNN). We started with the vanilla version and encountered the well-studied gradient exploding problem, i.e., the network couldn't find the global minimum of the loss function. Consequently, the maximum accuracy achieved for this NN class was around 50% for two consecutive 'simple-RNN' layers connected to a dropout and a dense layer.

Afterward, we implemented Long-Short-Term Memories (LSTM) in unidirectional and bidirectional configurations. This implementation effectively solved the gradient exploding problem. Moreover, the long-term memory improved the precision slightly, reaching around 65% (on Codalab's test set) of accuracy for two consecutive unidirectional LSTM layers of 128 and 128 units connected to a 0.5 dropout layer and a Dense layer with 128 neurons. On the other hand, we achieved 63% accuracy for two consecutive BiLSTM layers of 256 and 256 units connected to a 0.5 dropout layer and a Dense layer with 128 neurons.

Even though BiLSTMs didn't perform better than unidirectional LSTMs, we combined both architectures in multiple configurations looking for further accuracy improvements. For example, we tested 2 BiLSTM + 2 LSTM, 2 LSTM + 2 BiLSTM, and two consecutive blocks of 1 BiLSTM + 1 LSTM, only to name a few. All the tested configurations never performed above 60% accuracy, which is no better than in the non-stacked models, probably due to the lack of persistent long-term memory.

To overcome the limited long-term memory capabilities of the LSTMs, we incorporated attention mechanisms into our models. Indeed, we implemented a time-series classification version of the famous paper Attention is all you need (AAYN) [2]. By using this architecture, we wanted to improve the accuracy of our network by taking advantage of the long-recall capabilities of the transformer. Unfortunately, even though the convergence of the AAYN network is faster, the improvement in accuracy was minor.

Throughout all the ahead mentioned architectures, there is a transversal problem: the networks do not over-fit. Indeed, this fact is a direct consequence of the lack complexity on the training dataset. As we previously discussed in Section 2, the class unbalancing introduces a bias towards class 9 that impacts the overall prediction accuracy. Since further augmentations seems prejudicial for our metrics, we implemented a 'step-wise' classifier approach.

Our so called 'step-wise' classifier consists of 4 different models: the first one classifies between classes 9 and the rest, the second classifies classes 2, 3, 6 and the rest, the third classifies classes 1, 5, 8, 10 and the rest, and the fourth classifies between classes 0, 4, 7 and 11. We built these fourth classifiers so that they group (and classify) classes with a similar number of training samples. The motivation to do so is that we eliminate the unbalancing bias from each sub-classifier and could potentially achieve better results. The pseudo-code for the final classifier is the following:

Algorithm 1. Custom Step-wise Classifier

Require: $X \leftarrow$ features to predict
  $y \leftarrow predict(model\_1, X)$
  if $class(y) = 9$ **then**
    return 9
  end if
  $y \leftarrow predict(model\_2, X)$
  if $class(y)$ in [2, 3, 6] **then**
    return $class(y)$
  end if
  $y \leftarrow predict(model\_3, X)$
  if $class(y)$ in [1, 5, 8, 10] **then**
    return $class(y)$
  end if
  $y \leftarrow predict(model\_4, X)$
  if $class(y)$ in [0, 4, 7, 11] **then**
    return $class(y)$
  end if

This model resulted in an overall accuracy of around 64%. However, it improved the precision and recall metrics up to 69%.

Finally, we assembled and tested a 1D CNN which was the best-performing model and the chosen one. This model is discussed carefully in Section 5.

4 HYPERPARAMETERS TUNING

Since the inspection of the dataset did not bring much insights on the problem and considering that in time-series classification research data augmentation techniques are designed for a particular shape of the signal , it was exploited a more classical approach, i.e., oversampling the minority class [5].

However, this oversampling strategy may change the distribution of raw data and cause over-fitting, which as occurred in many models.

Hence, at this point the choice fell on the correct combination of hyper-parameters, optimizer and activation to make the model exploit all the information on the limited dataset and at the same time avoiding to interpolate it. In order to carry out this experiment it was considered as reference the biLSTM model.

4.1 Optimiser

According to literature, classification tasks have the form of a minimization problem and, since our use case is highly non linear, the choice of the optimizer to find a better local optima was another problem that arose accordingly. In Keras documentation different optimisers are available, but the choice of Adamax was the most consistent with respect to the given data and what it is known, i.e., a series of labels that remind to some Pink Floyd's tracks.

Another common issue in machine learning is the choice of the activation function and it is proved that a wrong choice can make the training slower or prevent the updating of the weights. Hence, the ReLU function has been changed in favor of the sigmoid, that in general has worst performance compared to the previous one. Despite this, it is well known that sigmoid work well with classification [6].

Hence this other possibility has also been explored. These two changes have allowed the considered model to improve itself of the two percent approximately, which is a coherent result with state of the art [7].

4.2 Regularization

At this point of the changes was added a non complex regularization strategy to make the model avoid to interpolate the given dataset. It consisted of different training iterations for different values of the weight decay parameter [8].

This approach did not provide any improvement, indeed the accuracy on the validation set used to be the same. The only difference was that the required iteration were less, most probably due to the fact that the gradient in such cases isn't allowed to follow some direction that bring to a certain local optima.

5 FINAL MODEL

After discussing all the different models we tried and what we achieved by training them, in this section we provide a detailed description of the final model, i.e., the one which got the best accuracy score on the Codalab's test set. The proposed model is based on this paper [1], which is rather simple, with less than 30.000 parameters, but yet effective. As depicted in Fig. 1, the model represents a 1-dimensional CNN, which is composed by a sequence of 1-D convolutional layers (with 64 filters), for a total of 3, all of them followed by Batch Normalization and ReLU activation layers. In the end, a Global Average Pooling (1 dimensional) is present before the final dense layer with the Softmax activation function for classifying the 12 classes.

6 PERFORMANCE COMPARISON AND MODEL SELECTION

In the final section of our resort, we compare the performance obtained by the best models we obtained. In particular, in Fig 2 are depicted the accuracy scores - global accuracy and accuracy for each class - of the following models: LSTM, BiLSTM, Transformer, the custom approach previously described and the final model, i.e., 1D-CNN. It is clear how the performance we obtained were not ideal but it is possible to see how the 1D-CNN is more effective for this specific problem. These results are evaluated over our test set, since we split the dataset to have 0.2 test set and 0.8 training set, which 0.2 was used for validation. The best model scored 0.67 accuracy on Codalab's test set.

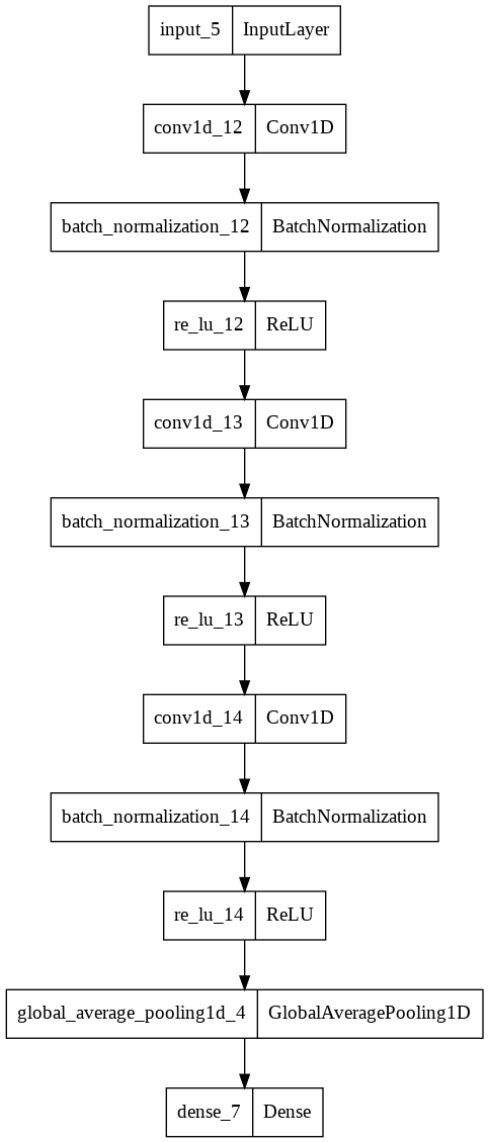

Figure 1. 1DCNN final model layers.

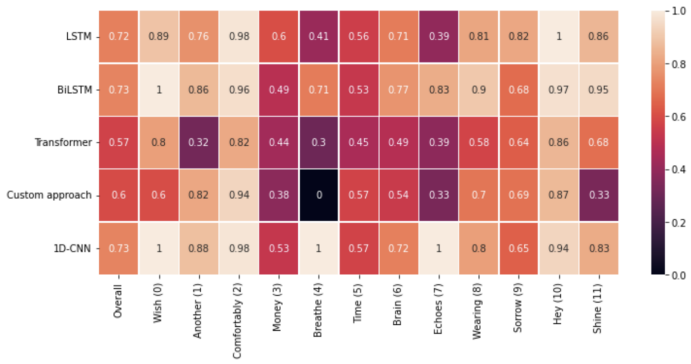

Figure 2. Accuracy score comparison between the models. Results obtained on our test set.

REFERENCES

[1] Zhiguang Wang, Weizhong Yan, and Tim Oates. Time series classification from scratch with deep neural networks: A strong baseline, 2016.

[2] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[3] Edward Ma. Data augmentation for audio, 2019.

[4] Brian Kenji Iwana and Seiichi Uchida. An empirical survey of data augmentation for time series classification with neural networks. *PLOS ONE*, 16:1–32, 07 2021.

[5] Qingsong Wen, Liang Sun, Fan Yang, Xiaomin Song, Jingkun Gao, Xue Wang, and Huan Xu. Time series data augmentation for deep learning: A survey. *arXiv preprint arXiv:2002.12478*, 2020.

[6] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions: Comparison of trends in practice and research for deep learning. *arXiv preprint arXiv:1811.03378*, 2018.

[7] Zachary Kirori. Performance analysis of stochastic gradient descent-based algorithms for time series sequence modeling. 2019.

[8] Amin Ghiasi, Ali Shafahi, and Reza Ardekani. Adaptive weight decay: On the fly weight decay tuning for improving robustness. *arXiv preprint arXiv:2210.00094*, 2022.