Black Body Radiation – Low complexity CNN for time series classification

Pietro Miglioranza, Andrea Scanu, Giuseppe Simionato, and Nicholas Sinigaglia
(Dated: May 15, 2021)

Time series analysis and classification are common tasks in many scientific fields. Nowadays, the main tools used for their classification are the Convolutional Neural Networks: powerful machine learning algorithms with a high number of parameters very suitable for pattern recognition.
The main focus of our work is to find a CNN architecture able to classify as well as possible simple univariate time series with the constraint of having at most 600 learnable parameters. In the following paper, after a brief introduction about the general framework of time series analysis, we present our self-designed model and its learning performance, comparing its classification ability with the results from a 2250-parameters CNN and from a 9968-parameters one showing how a few parameters CNN is a more suitable choice in the case of relatively small training set.

INTRODUCTION

Time series analysis in our society is of paramount importance in many fields. Time series are a series of data points where the set of observations are in a chronological order. We can find some examples in human activity recognition [1], weather analysis [2] and astrophysics [3].

Current approaches to time series classification are based on the Convolutional Neural Networks (CNNs). These types of algorithms are a very powerful tool in the task of finding the underlying structure in datasets composed by time series [4]. However, the CNN needs the proper tuning of its hyper-parameters and often requires a large amount of layers and neurons to obtain good performances. In their work *H. I. Fawaz at al.* use the AlexNet architecture composed by an high number of hyper-parameters ($O(10^6)$) to archieve good performances [5]. *M. Ibrahim et al.* build a CNN with two convolutional layers and two fully connected ones, the performances of their algorithm are excellent, in particular for real-time processing, but the number of parameters is very large ($O(10^5)$) [6].

In the literature, there are a wide variety of approaches to overtake the problem of tuning in the proper way the hyper-parameters [7], but nowadays the most common solution is to set them via grid search or human experience, as we do in this work.

The main focus of this paper is to study the effects of the different arguments of the CNN and to reduce the complexity of the network, limiting the number of learnable parameters. We also focus on the dependence of the algorithm on its variables to see how the performances scale with them and if there are some that are more determinant than others.

In this work, we study the classification of an univariate time series dataset with 3 different classes. Univariate time series are a type of series which have a single variable observed at each time, they are also considered as one-dimensional time series. We evaluate our architecture on a simulated dataset composed by 10000 1-dimensional time series with a specific pattern combined with some randomly generated noise. Finally, we compare our solution with more complex architectures.

METHODS

Our designed CNN is represented in Figure 1. It cannot be a completely general architecture since the size and the number of the filters, together with the size of the output layer depend on the input shape and the number of classes. Nevertheless, the structure of our CNN can be easily adapted to any one-dimensional time series, tuning only the three features mentioned above. To operate a supervised learning, we also provide the label with the corresponding type of pattern for every data.

The CNN is structured to have as fewer parameters as possible and it has only three convolutional layers:

1. The first uses a small filter size referred to input size. The idea is to use it as "input data cleaner", helping next layer in finding patterns.

2. The second is the "pattern-finder" layer. It performs convolution with a large number of filters and big filter size referred to input size. The main idea behind this layer is to use it as a sort of "yes/no decision", searching only for specific macro-pattern.

3. The third has same features of the first one and should highlight even more if a pattern has been found by the second layer.

We implement and train our CNN using Keras software.

Global pooling and number of parameters

Classes	Input shape	Pooling stride	Np
3	(60,1)	global	579
15	(60,1)	global	639
15	(120,1)	global	639
3	(60,1)	4	711
15	(60,1)	4	1299
15	(120,1)	4	2199

Table I: In this table is visible how the number of parameters (**Np**) scales with: number of classes, input shape and usage of max pooling stride. The first row refers to the model presented in this paper. Filters size and number are fixed.

The last convolutional layer is followed by a global pooling. It is used because if the "pattern-finder"

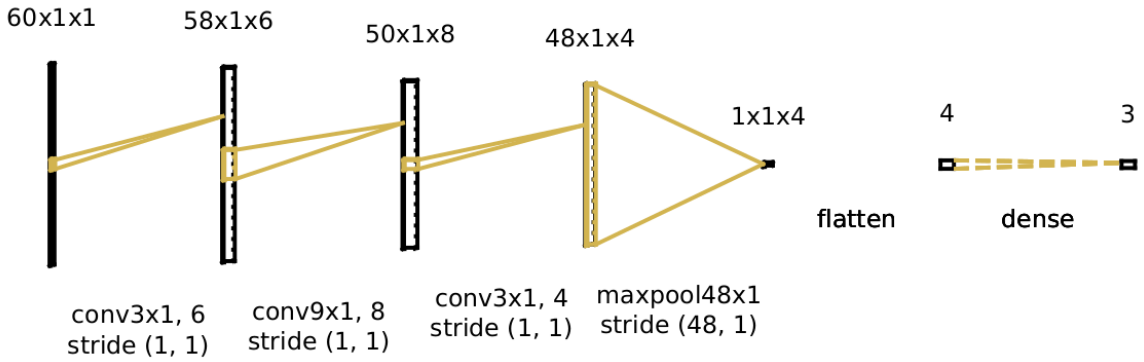

Figure 1: Architecture of the CNN designed to perform time series classification. It is made of three 1D convolutional layers followed by a global max pooling, a flatten and only one fully connected layer. The number of parameters of this architecture is just 579 but it is tuned for the analyzed time series, where input shape is (60,1) and number of classes is equal to 3.

layer finds a significant presence of a pattern, then a global max pooling is the right way to perform the "yes/no decision" mentioned previously.
Moreover, from Table I is visible that as expected using a global pooling the number of parameters does not change depending on the input shape and scales much slowly in comparison with using a non-global max pooling.

Using a global max pooling is not the best choice *a priori*, but for a low complexity problem such the one faced in this work, it could be a winning solution.

Regularization and dropout

Two important strategies used to avoid overfitting in the training process are regularization and dropout. These two techniques are described and compared in detail in [8].

Dropout is not used in our proposal model because it has only one dense layer with an already small number of parameters thanks to global max pooling.

Moreover, due to the low number of used filters in our CNN, the regularization does not help in obtaining better performance. For this reason, it is not implemented either.

Hyperparameters

In the tuning of the activation function and optimizer of each layer, we use a gridsearch: as a result, the better performing configuration is the one with adam as optimizer for every layer, Relu activation function for the three convolutional layers and softmax activation function for the output layer.

In the training of our CNN, we use the 75% of the dataset while the remaining 25% is used for the validation. The loss is computed using categorical cross entropy and a batch size of 300. To avoid overfitting, we use the so-called early stopping which is provided by Keras callbacks method. Its main parameter is called "patience" (see documentation [10]). In our CNN, this parameter is equal to 20 and the training is stopped in any case if a number of 600 epochs is reached.

The only non-reproducible parameters are the gaussian initializations ($\mu = 0, \sigma = 0.1$) of the weights.

Data augmentation

As it is known, the CNNs work better with more available data and for this reason we operate a data augmentation starting from our dataset. When we handle time series, there are a lot of techniques to enlarge our data, as *H.I. Fawaz* presents in his paper [9]. We decide to perform a mirror augmentation over the dataset used in this work. In this augmentation, all the series are mirrored reversing the order of its values and keeping the same label of the original one. With respect to the other techniques, this is the one that brings our CNN to slightly improve its performance.

RESULTS

Since the problem is a deterministic task, there exists a NN with an architecture that recognises all possible patterns. In our case, the main problems are that the network is trained with few data (16000) and the possible presence of wrong labels introduced by the data augmentation.

The analysis of this particular architecture starts with the training: our results can be reproduced by training the network using the same hyper-parameters described in the previous section. In Figure 2, panels (a) and (b) are shown our proposal network accuracy and loss over the epochs of training: at the end, the best result is a validation accuracy of about 85.1% and a validation loss of about 0.37.

We also compare our proposal CNN with two competitor Neural Networks: a CNN with the same architecture with an extra dense layer before the output and a CNN with two dense layers before the output. Their architectures are shown in Tables II and IV. Both models are initialised with the hyper-parameters from Table III and are implemented using Keras software.

The first competitor network has 2250 trainable parameters and its performances are quite the same of

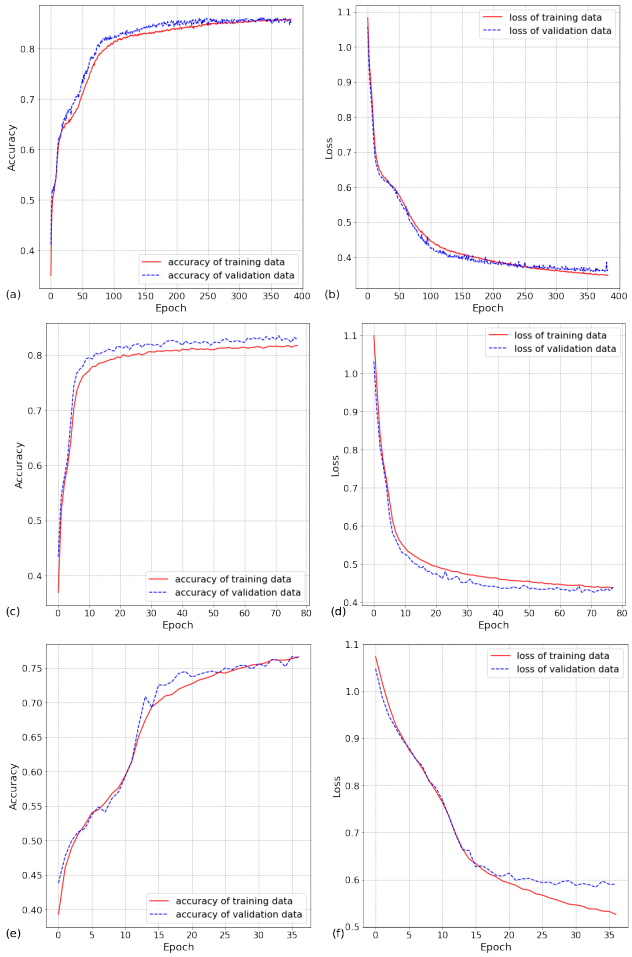

Figure 2: In panel **(a)** and **(b)** CNN accuracy and loss are represented versus the number of epoch of training. In panel **(c)** and **(d)** the same values are shown for the 2250 parameters network. In paneles **(e)** and **(f)** there are the ones of the 9968 parameters one.

Layer	Output shape
Input	(60)
Conv 1	(48,16)
Conv 2	(40,12)
G. Max Pooling	(12)
Dense	(18)
Output	(3)

Table II: 2250 parameters NN architecture

our proposal ones as it is shown in Figure 2 panels (c) and (d). Its validation accuracy and loss at the end of the training are 82.8% and 0.44. The second competitor has 9968 learnable parameters and its results are shown in Figure 2, panels (e) and (f). Its validation accuracy is 76.5 % and its final loss is 0.59.

The graphs in Figure 2 show the dependency of the training with respect to the number of parameters and the architecture of our NNs. The first competitor network (panels (c), (d)) behaves similarly to our: the training starts with a vertical learning curve and then reaches a saturating part. The difference between this two networks is in the number of epochs

Init.	
Max epochs	600
Batch size	300
Activ. fun.	Relu
Activ. fun. out	Softmax
Loss	Categorical cross-entropy
Gradient	Adam
Patience 1	10
Patience 2	3

Table III: hyperparameters of competitor CNNs

Layer	Output shape
Input	(60)
Conv 1	(49,15)
Conv 2	(40,10)
Flatten	(400)
Dense	(20)
Dense	(10)
Output	(3)

Table IV: 9968 parameters NN architecture

that they need to reach the saturating part. In the first, we can use a high patience since it will not easily overfit due to the few learnable parameters (579). Instead, the second NN has a patience of 10 and so it stops earlier in about 80 epochs. Considering the third network, looking at the panels (e) and (f), it is noticeable that the training runs for about 30 epochs. To manage this big NN, we use a very low patience that prevents overfitting and stops the training. This procedure has the side effect of stopping the learning with a lower validation accuracy.

CONCLUSIONS

In this work, we present a CNN with a very low number of trainable parameters designed to solve a classification task over a small dataset.

We also show how the number of trainable parameters plays a fundamental role in the training and how its ideal value is strongly related to data availability and task complexity. For this reason, it is fundamental to design every architecture specifically with respect to the task it has to solve.

Finally, we demonstrate that choosing a complex architecture without really adapting it to the task that it has to solve is not a wise choice.

Our proposal CNN reaches a validation accuracy of 85.6% and outperforms two more complex and less studied CNNs that reach a validation accuracy of 82.8% and 76.5% on the same dataset.

[1] M. Zeng, L. T. Nguyen, B. Yu, O. J. Mengshoel, J. Zhu, P. Wu, J. Zhang, *Convolutional Neural Networks for Human ActivityRecognition using Mobile Sensors*, 10.4108/icst.mobicase.2014.257786

[2] Z. Zhang, Y. Dong, *Temperature Forecasting via Convolutional Recurrent NeuralNetworks Based on Time-Series Data*, https://doi.org/10.1155/2020/3536572
[3] A. Brunel, J. Pasquet, J. Pasquet, N. Rodriguez, F. Comby, D. Fouchez, M. Chaumont, *A CNN adapted to time series for the classification of Supernovae*, arXiv:1901.00461
[4] B. Zhao, H. Lu, S. Chen, J. Liu, D. Wu, *Convolutional neural networks for time series classification*, 10.21629/JSEE.2017.01.18
[5] H.I. Fawaz, B. Lucas, G. Forestier, C. Pelletier, D.F. Schmidt, J. Weber, G.I. Webb, L. Idoumghar, P. Muller, F. Petitjean, *InceptionTime: Finding AlexNet for Time Series Classification*, 10.1007/s10618-020-00710-y
[6] M. Ibrahim, M. Torki, M. ElNainay, *CNN based Indoor Localization using RSSTime-Series*, 10.1109/ISCC.2018.8538530
[7] W. Tang, G. Long, L. Liu, T. Zhou, J. Jiang, M. Blumenstein, *Rethinking 1D-CNN for time series classification: A stronger baseline*, arXiv:2002.10061
[8] Ekachai Phaisangittisagul, *An Analysis of the Regularization Between L2 and Dropout in Single Hidden Layer Neural Network*, https://uksim.info/isms2016/CD/data/0665a174.pdf
[9] H.I. Fawaz, G Forestier, J. Weber, L. Idoumghar, P. Muller, *Data augmentation using synthetic data for time series classification with deep residual networks*, arXiv:1808.02455
[10] *Keras callbaks documentation*, https://keras.io/api/callbacks/