MuonGang – Convolutional Neural Networks

Alessandro Valente, Andrea Paccagnella, and Rocco Ardino
(Dated: January 25, 2021)

We discuss the application of Convolutional Neural Networks to bump recognition inside monodimensional time-series. We start from the generation of the input data with a given signal-to-noise ratio and propose different architectures of a neural network for classification. After selecting the best performing one, its hyperparameters are optimized through parallelized grid searches and the performances are evaluated on new time-series data. We then discuss the consequences of regularization techniques and of training on more noisy data, in order to evaluate the generalization power of the optimal trained network. Lastly, we propose a training strategy through which we achieve the best results of our work.

INTRODUCTION

Deep Learning sits at the forefront of many important advances in Machine Learning. Its use of artificial neural networks, resembling the human brain structure, has been proven to be extremely successful. In particular, Convolutional Neural Networks (CNNs) are one of the state-of-art tools of this ML field and are capable of processing data with peculiar spatial and/or temporal patterns. In this work we consider the common monodimensional case of time-series data, where a positive or negative bump should be detected. This example covers a particular importance in many research fields, such as Particle Physics and Electronics.

Before going on, we provide a basic introduction on the working principles of CNNs. The key mathematical operation employed in their layers is the convolution, used in place of the general matrix multiplication. These specialized Convolutional Layers are used along with Pooling Layers, which reduce the dimensionality of a layer output through operations in blocks like min/max or average. Moreover, they can even be connected after shape flattening to a classical DNN. An idea of this structure is given in FIG. **??**.

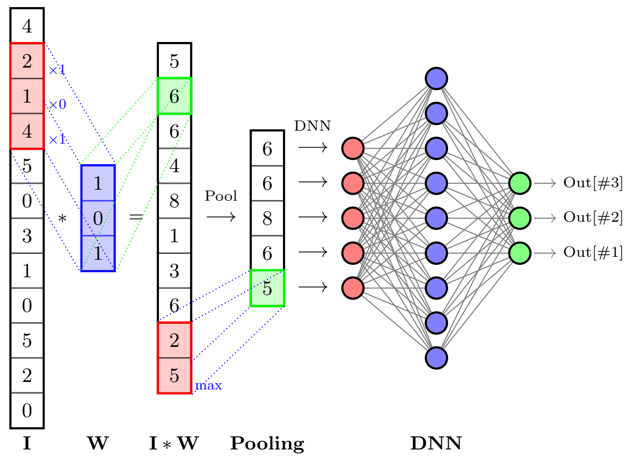

FIG. 1. 1D basic CNN convolution and pooling operations. The flattened pooling output is then connected to a basic DNN.

METHODS

Data generation: First of all, we introduce some notations for the sake of simplicity. We denote with

$$\tilde{x}_j = \{\tilde{x}(t_1), \dots, \tilde{x}(t_L)\} \tag{1}$$

$$\hat{x}_j = \{\hat{x}(t_{L-i+1}), \dots, \hat{x}(t_{L-i+Z})\} \tag{2}$$

respectively the j^{th} input time-series without the bump and the bump itself. The parameters L and Z are respectively the time-series and bump lengths, fixed to 60 and 12, and i ranges from Z to L.

$\tilde{x}_j(t)$ signals are generated by sampling the amplitude difference $\mathrm{d}x$ between $\tilde{x}_j(t_i)$ and $\tilde{x}_j(t_{i+1})$, from an exponential probability distribution:

$$\mathbb{P}(\mathrm{d}x) \sim \exp\left(-\frac{|\mathrm{d}x - b|}{\Delta x}\right) \tag{3}$$

where b is a parameter called bias and Δx is the step typical size of the jump process. In our case, they are fixed respectively to 5 and 50. By applying the inverse transform of Eq. **??**, we get the sampling rule:

$$\mathrm{d}x = \lfloor \log(p)\Delta x \rfloor \cdot 2\,\mathrm{sign}(q - 0.5) + b \tag{4}$$

where p is a random number sampled from a uniform distribution in $[0, 1]$, q is chosen in $\{0, 1\}$ with equal probability assigned to each element of the set.

$\hat{x}_j(t)$ signals are sine bumps generated through:

$$\hat{x}_j(t_{i+k}) = \left\lfloor A \cdot \sin\left(\frac{\pi \cdot k}{Z}\right)\right\rfloor \tag{5}$$

where A is the amplitude of the signal, initially fixed to 500. As we can deduce from Eq. **??**, they are added randomly inside the time-series.

Hence, we have three possibilities for the input signals, properly labeled by a target variable y:

$$x_j = \begin{cases} \tilde{x}_j + 0 \cdot \hat{x}_j & y_j = 0 \\ \tilde{x}_j + 1 \cdot \hat{x}_j & y_j = 1 \\ \tilde{x}_j - 1 \cdot \hat{x}_j & y_j = 2 \end{cases} \tag{6}$$

These data are normalized before feeding them to the CNN. This is done by subtracting the mean of the time-series and by dividing by the standard deviation.

Layer type	Output shape	Parameters
Conv1D	(50,5)	60
Av. Pool.	(16, 5)	0
Conv1D	(10,5)	180
Av. Pool.	(5, 5)	0
Conv1D	(1,5)	130
Flatten	(5)	0
Dense	(10)	60
Dense	(10)	110
Dropout(0.2)	(10)	0
Dense	(3)	33

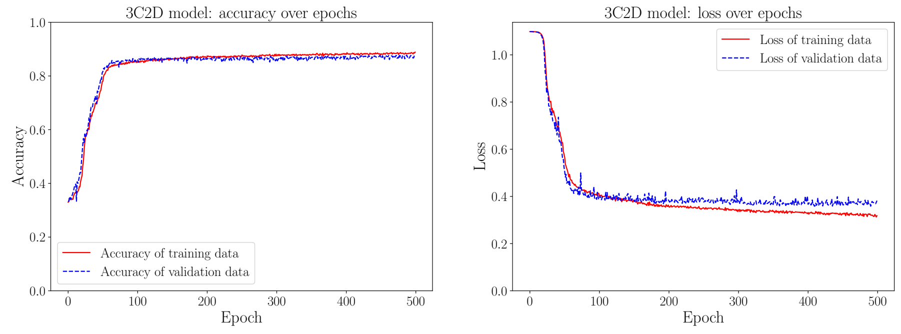

FIG. 2. Detailed architecture of the best performing model (3C2D) on the left. Results for this architecture on the right.

CNN implementation: The networks are built through the powerful python libraries of TensorFlow and Keras, which allow to implement even complex CNNs and DNNs in few lines of code. One of our aims is to find the best performing architecture; to do this we have divided the first part of the work into two steps:

- in the first one we compare various CNN architectures, created from a reference model ("2C1D") by varying the number of Convolutional or Dense layers. A constraint on the number of parameters is set in order to avoid too complex models. After training them with the same conditions, we select the one with the highest maximum accuracy;
- in the second step, we analyze how the best architecture can be tuned by optimizing the hyperparameters through grid searches with a 3-fold cross validation method. More deeply, we focus on the tuning of the optimizer and the regularizer.

After these steps we are able to define the best model for data classification. So, in the last part of our work we try to train the model using datasets where signal-to-noise ratio is reduced progressively. This operation translates into generating datasets with a lower signal amplitude A, allowing us to estimate a threshold under which the model is not capable of effectively classifying the data.

Moreover, we try to maximize the stability and capability of the model of making correct predictions. For this last run we generate a dataset with more samples in comparison to the previous ones, but with a signal amplitude randomly chosen between the starting value and the previously defined threshold. After training and test phases, we exploit the confusion matrix of the predictions to better understand the behaviour of the network for each data category.

RESULTS

The different architectures are presented in TABLE **??** alongside with their score, namely the maximum accuracy on the validation set obtained during training. As we can see. the most important factor affecting the results seems to be the number of layers, rather than the number of parameters. In fact, we can observe that the best architectures are the ones with more layers despite having less parameters respect to the others. In FIG. **??**, the plots of accuracy and loss function over the training epochs, are showed for the best architecture, alongside with a detailed description of the architecture of that model.

Model	**Parameters**	**Score**
1C1D	587	88.4%
2C1D	579	88.8%
2C2D	593	89.1%
3C1D	598	88.7%
3C2D	**573**	**89.3%**

TABLE I. Training results on different CNN architectures, training set of 8000 events with signal amplitude of 500, training performed with a batch size of 250 for 500 epochs. The name of the models describes their structure: "C" indicates a Convolutional layer, "D" a hidden Dense one and the number beside how many per type are present.

The best model is then tuned by performing several grid searches to find the best configuration. In TABLE **??** and TABLE **??** are respectively presented some of the most significant results obtained during this phase, focusing particularly on the optimization algorithms and the regularization functions. For the sake of simplicity, we report a summary of the parameters of the best setup:

- **Optimizer**: Nesterov-SGD with learning rate 0.02, momentum 0.9 and decay 10^{-6}.
- **Activation functions**: "relu" activations for all the layers expect the last one which uses the "softmax", since we want every output entry in $[0, 1]$.
- **Dropout rate**: $p = 0.2$.
- **Regularizer**: L1 regularization with $\lambda_{\mathrm{L1}} = 5 \cdot 10^{-6}$.

With this configuration our model can reach a test accuracy of 90.6%.

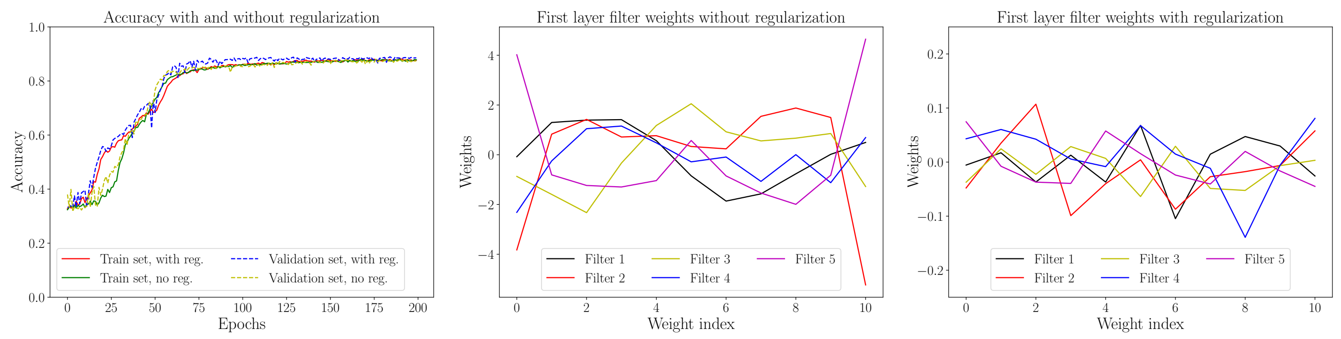

FIG. 3. Left: accuracy of the model during training with and without regularization. Center and right: weights of the filters of the first layer with and without regularization.

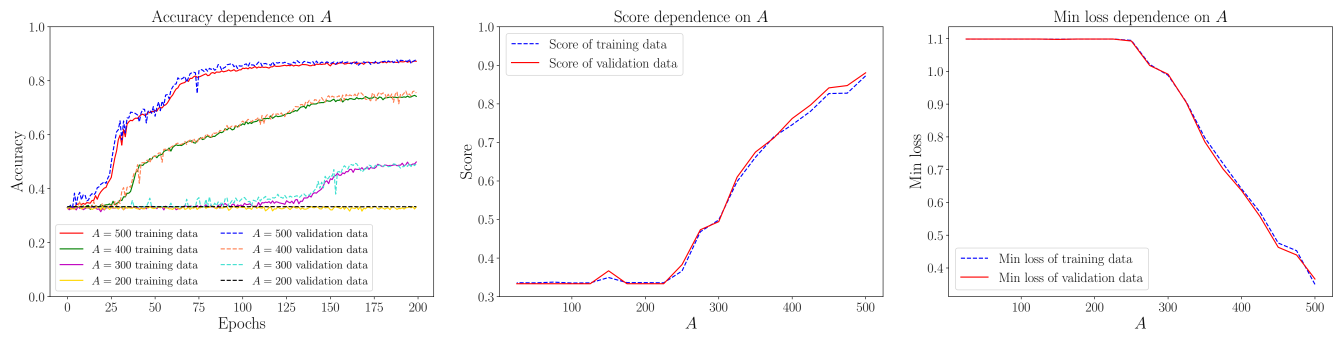

FIG. 4. Left: accuracy of the model during training for datasets with different signal amplitudes. Center and right: score and loss dependence on A.

Optimizer: NAdam									
η	0.003	0.005	0.007	0.003	**0.005**	0.007	0.003	0.005	0.007
β_1	0.85	0.85	0.85	0.875	**0.875**	0.875	0.9	0.9	0.9
Score	0.868	0.875	0.852	0.693	**0.878**	0.874	0.866	0.869	0.691
Optimizer: RMSprop									
η	0.01	**0.01**	0.01	0.015	0.015	0.015	0.02	0.02	0.02
ρ	0.87	**0.9**	0.93	0.87	0.9	0.93	0.87	0.9	0.93
Score	0.861	**0.869**	0.682	0.672	0.507	0.683	0.513	0.854	0.639
Optimizer: Nesterov-SGD									
η	0.01	0.01	0.01	0.02	0.02	**0.02**	0.03	0.03	0.03
m	0.875	0.9	0.925	0.875	0.9	**0.925**	0.875	0.9	0.925
Score	0.733	0.660	0.675	0.688	0.875	**0.882**	0.333	0.10	0.860

TABLE II. Cross validated grid search results in the tuning of the regualarization for the model 3C2D.

L1 regularization									
λ_{L1}	5e-7	1e-6	3e-6	**5e-6**	8e-6	1e-5	5e-5	1e-4	5e-4
Score	0.692	0.871	0.868	**0.881**	0.870	0.873	0.687	0.689	0.506
L2 regularization									
λ_{L2}	5e-6	1e-5	5e-5	1e-4	**3e-4**	5e-4	8e-4	1e-3	5e-3
Score	0.775	0.694	0.698	0.697	**0.875**	0.866	0.696	0.467	0.638
L1-L2 regularization									
λ_{L1}	1e-5	1e-5	1e-5	5e-5	**5e-5**	5e-5	1e-4	1e-4	1e-4
λ_{L2}	1e-5	5e-4	1e-4	1e-5	**5e-4**	1e-4	1e-5	5e-4	1e-4
Score	0.873	0.689	0.872	0.877	**0.878**	0.689	0.516	0.872	0.693

TABLE III. Cross validated grid search results in the tuning of the regularization for the model 3C2D.

The new best score obtained is a visible enhancement, but there are other improvements hidden in the core of the network. For this reason, we deepen the consequences that the introduction of a regularizer has on the model. As we can see in FIG. **??**, there is a very small improvement in the accuracy. On the other hand, we observe a significant reduction of the magnitude of the filters weights. This evidence translates into a more stable and reliable predictor after the training is completed.

Now that the best architecture is defined, the next step is finding a threshold for the signal amplitude such that the trained model is still reliable in its predictions. Therefore, we generate multiple training and test datasets, with a progressively decreasing signal-to-noise ratio, thus with a decreasing value of A.

We then evaluate the performances of the model on each test set after training on the corresponding training set. Some of the results of this operation are showed in TABLE **??** and FIG. **??**.

A	**500**	**450**	**400**	**350**	**300**	**250**	**200**
Score	0.891	0.841	0.762	0.675	0.495	0.383	0.333
Min loss	0.366	0.463	0.636	0.785	0.991	1.093	1.099

TABLE IV. Performances of the best architecture on datasets with different signal amplitude.

As highlighted by the results, the model is capable of making good predictions down to a signal amplitude of approximately 350. In fact, for an amplitude of about 300 the score is already at 49.5%, which does not translate into a completely reliable prediction. When going down to 200, the score is about 33%, which is equivalent to a random guess, since we have 3 possible equal-probable categories.

The final step is then to experiment if we can further improve the classification power of the best model up to now. The idea is to train it on a bigger dataset of about 10^6 samples with a variable signal amplitude, chosen randomly between 350 and 500, to improve the stability and adaptability of the network. We then test the trained model on another test dataset, achieving the best accuracy of our work: 91.5%.

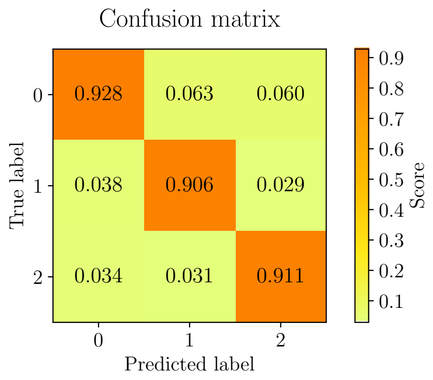

FIG. 5. Confusion matrix of the best model prediction over a test set with random signal amplitude.

A more precise analysis of this result can be extrapolated by the normalized confusion matrix in FIG. **??**. As we can see, most of the errors are represented by "false positives", namely cases where the CNN classifies an input sample as signal despite it being only noise.

CONCLUSIONS

In this work we highlight the importance of choosing the correct architecture of a CNN for the task. In fact, despite being the one with the lowest amount of parameters, we find the "3C2D" model to be the best choice among the others.

The study of the stability and the tuning of the network prove to be really important factors in the selection of the architecture. In particular, we show that the most important hyperparameters to tune, in order to achieve a higher score and stability, are the optimizer function, the regularizer and their respective parameters.

The other key aspect of this work is the dataset itself. We see that the best model selected is stable and able to make good predictions in a certain interval of the signal amplitude value A. Therefore, the best strategy to achieve a better generalization power is to train the network on a dataset of time-series generated without fixing, but only constraining, this parameter. In fact, during the tuning of the hyperparameters, we employ a dataset of 10^4 samples with fixed signal amplitude. This improves the CNN performances from 89.3% to 90.6%, but the further improvement up to 91.5% score, is achieved through the use of a bigger dataset, composed of 10^6 samples with random signal amplitude.

A way to reach even higher scores would be to tune every hyperparameter of the CNN on the previous bigger dataset, but the computational cost would be very high and not worth for a small improvement.

[1] I. Goodfellow, Y. Bengio, A. Courville, *Deep Learning* **9**, 300–371 (2015)

[2] P. Mehta, M. Bukov, et al., *A high-bias, low-variance introduction to Machine Learning for physicists* **10**, 61–64 (2019)