Spectral pruning of fully connected layers

Buffoni, Lorenzo; Civitelli, Enrico; Giambagli, Lorenzo; Chicchi, Lorenzo; Fanelli, Duccio

Publication date:
2021

Spectral Pruning of Fully Connected Layers: Ranking the Nodes Based on the Eigenvalues

Lorenzo Buffoni,[1,2] Enrico Civitelli,[3] Lorenzo Giambagli,[2] Lorenzo Chicchi,[2] and Duccio Fanelli[2]

[1] *Physics of Information and Quantum Technologies Group, Instituto de Telecomunicações, Lisbon, Portugal*

[2] *CSDC, Department of Physics and Astronomy, University of Florence, Sesto Fiorentino, Italy*

[3] *LabGOL, Department of Information Engineering, University of Florence, Florence, Italy*

Training of neural networks can be reformulated in spectral space, by allowing eigenvalues and eigenvectors of the network to act as target of the optimization instead of the individual weights. Working in this setting, we show that the eigenvalues can be used to rank the nodes' importance within the ensemble. Indeed, we will prove that sorting the nodes based on their associated eigenvalues, enables effective pre- and post-processing pruning strategies to yield massively compacted networks (in terms of the number of composing neurons) with virtually unchanged performance. The proposed methods are tested for different architectures, with just a single or multiple hidden layers, and against distinct classification tasks of general interest.

I. INTRODUCTION

Automated learning via deep neural networks is gaining increasing popularity, as a ductile procedure to address a widespread plethora of interdisciplinary applications [1–3]. In standard neural network training one seeks to optimise the weights that link pairs of neurons belonging to adjacent layers of the selected architecture [4]. This is achieved by computing the gradient of the loss with respect to the sought weights, a procedure which amounts to operate in the so called direct space of the network [5]. Alternatively, the learning can be carried out in reciprocal space: the spectral attributes (eigenvalues and eigenvectors) of the transfer operators that underlie information handling across layers define the actual target of the optimisation. This procedure, first introduced in [5] and further refined in [6], enables a substantial compression of the space of trainable parameters. The spectral method leverages on a limited subset of key parameters which impact on the whole set of weights in direct space. Particularly relevant, in this respect, is the setting where the eigenmodes of the inter-layer transfer operators align along random directions. In this case, the associated eigenvalues constitute the sole trainable parameters. When employed for classifications tasks, the accuracy displayed by the spectral scheme restricted to operate with eigenvalues is slightly worse than that reported when the learning is carried in direct space, for an identical architecture and by employing the full set of trainable parameters. To bridge the gap between conventional and spectral methods in terms of measured performances, one can also train the elements that populate the non trivial block of the eigenvectors matrix [5]. By resorting to apt decomposition schemes, it is still possible to contain the total number of trainable parameters, while reaching stunning performances in terms of classification outcomes [6].

In this paper we will discuss a relevant byproduct of the spectral learning scheme. More specifically, we will argue that the eigenvalues do provide a reliable ranking of the nodes, in terms of their associated contribution to the overall performance of the trained network. Working along these lines, we will empirically prove that the absolute value of the eigenvalues is an excellent marker of the node's significance in carrying out the assigned discrimination task. This observation can be effectively exploited, downstream of training, to filter the nodes in terms of their relative importance and prune the unessential units so as to yield a more compact model, with almost identical classifi-

cation abilities. The effectiveness of the proposed method has been tested for different feed-forward architectures, with just a single or multiple hidden layers, by invoking several activation functions, and against distinct datasets for image recognition, with various levels of inherent complexity. Building on these findings, we will also propose a two stages training protocol to generate minimal networks (in terms of allowed computing neurons) which outperform those obtained by hacking off dispensable units from a large, fully trained, apparatus. This is a viable strategy to discover a "winning ticket" [7]: dense (randomly-initialized) feed-forward networks contain sub-networks (aka winning tickets) with recorded performance comparable to those displayed by their unaltered homologues, after a proper round of training.

The paper is organized as follows. In the next section we will discuss the mathematical foundation and set the notation of the spectral learning scheme. We will then move on to illustrating the results of the proposed spectral pruning strategy, after a short account of the alternative methods available in the literature. Finally, we will sum up and draw our conclusions. The details about the proposed schemes are discussed in the Methods Section.

II. SPECTRAL APPROACH TO LEARNING

This Section is devoted to reviewing the spectral approach to the training of deep neural networks. The discussion will follow mainly [6], where an extension of the method originally introduced in [5] is handed over.

Consider a deep feed-forward network made of ℓ distinct layers. Each layer is labelled with a discrete index i $(= 1, ..., \ell)$. Denote by N_i the number of the neurons, the individual computing units, that pertain to layer i. Then, we posit $N = \sum_{i=1}^{\ell} N_i$ and introduce a column vector $\vec{x}^{(1)}$, of size N, the first N_1 entries referring to the supplied input signal. As anticipated, we will be mainly concerned with datasets for image recognition, so we will use this specific case to illustrate the more general approach of spectral learning. This means that, the first N_1 elements of $\vec{x}^{(1)}$ are the intensities (from the top-left to the bottom-right, moving horizontally) as displayed on the pixels of the image presented as an input. All other entries of $\vec{x}^{(1)}$ are identically equal to zero.

The aim of the procedure is to map $\vec{x}^{(1)}$ into an output vector $\vec{x}^{(\ell)}$, still of size N: the last N_ℓ elements are the intensities displayed at the output nodes, where reading is eventually performed. The applied transformation is composed by a suite of linear operations, interposed to non linear filters. To exemplify the overall strategy, consider the generic vector $\vec{x}^{(k)}$, with $k = 1, ..., \ell - 1$, as obtained after k execution of the above procedure. At the successive iteration, one gets $\vec{x}^{(k+1)} = \mathbf{A}^{(k)}\vec{x}_{(k)}$, where $\mathbf{A}^{(k)}$ is a $N \times N$ matrix with a rather specific structure, as elucidated in the following and schematically depicted in Fig. 1. Further, a suitably defined non-linear function $f(\cdot, \beta_k)$ is applied to $\vec{x}^{(k+1)}$, where β_k identifies an optional bias. To proceed in the analysis, we cast $\mathbf{A}^{(k)} = \mathbf{\Phi}^{(k)}\mathbf{\Lambda}^{(k)}\left(\mathbf{\Phi}^{(k)}\right)^{-1}$ by invoking spectral decomposition. Here, $\mathbf{\Lambda}^{(k)}$ denotes the diagonal matrix of the eigenvalues of $\mathbf{A}^{(k)}$. Following [6], we set $\left(\mathbf{\Lambda}^{(k)}\right)_{jj} = 1$ for $j < \sum_{i=1}^{k-1} N_i$ and $j > \sum_{i=1}^{k+1} N_i$. The remaining $N_k + N_{k+1}$ elements are initially assigned to random entries, as e.g. extracted from a uniform distribution, and define a first basin of target variables for the spectral learning scheme. Then, $\mathbf{\Phi}^{(k)}$ is the identity matrix $\mathbb{I}_{N\times N}$, with the inclusion of a sub-diagonal $N_{k+1} \times N_k$ block, denoted by $\boldsymbol{\phi}^{(k)}$, see Fig. 2. This choice amounts to assume a feed-forward architecture. It can be easily shown that $\left(\mathbf{\Phi}^{(k)}\right)^{-1} = 2\mathbb{I}_{N\times N} - \mathbf{\Phi}^{(k)}$, which readily yields $\mathbf{A}^{(k)} = \mathbf{\Phi}^{(k)}\mathbf{\Lambda}^{(k)}\left(2\mathbb{I}_{N\times N} - \mathbf{\Phi}^{(k)}\right)$. The off-diagonal elements of $\mathbf{\Phi}^{(k)}$ define a second set of adjustable parameters to be self-consistently modulated during active training. To implement the learning scheme on these basis, we consider $\vec{x}^{(\ell)}$, the image on the output layer of the input vector $\vec{x}^{(1)}$:

$$\vec{x}^{(\ell)} = f\left(\mathbf{A}^{(\ell-1)}...f\left(\mathbf{A}^{(1)}\vec{x}^{(1)},\beta_1\right),\beta_{\ell-1}\right) \quad (1)$$

Since we are dealing with image classification, we can calculate $\vec{z} = softmax(\vec{x}^{(\ell)})$. We will then use $\vec{z}$ to compute the categorical cross-entropy loss function $\mathrm{CCE}(l(\vec{x}^{(1)}),\vec{z})$, where $l(\vec{x}^{(1)})$ is the label which identifies the category to which $\vec{x}^{(1)}$ belongs, via one-hot encoding [8].

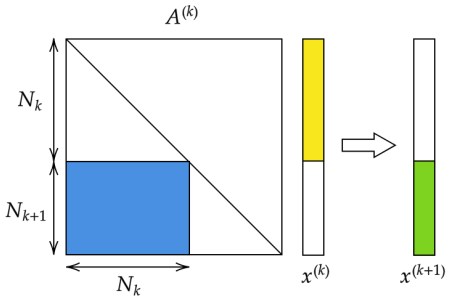


FIG. 1: A schematic outline of the structure of transfer matrix $\mathbf{A}^{(k)}$, bridging layer k to layer $k+1$. The action of $\mathbf{A}^{(k)}$ on $\vec{x}^{(k)}$ is also graphically illustrated.

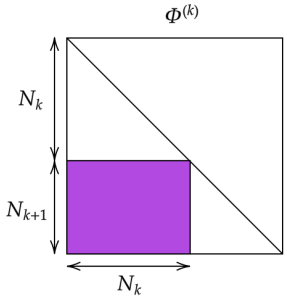


FIG. 2: The structure of matrix $\mathbf{\Phi}^{(k)}$ is schematically displayed.

The loss function can thus be minimized by acting on the spectral parameters, i.e. the ensemble made of non trivial eigenvalues and/or the associated eigendirections. A straightforward calculation, carried out in the annexed supplementary information, allows one to derive a closed analytical expression for $w_{ij}^{(k)}$, the weights of the edges linking nodes i (belonging to layer $k+1$) and j (sitting on layer k) in direct space, as a function of the underlying spectral quantities. In formulae, one gets:

$$w_{ij}^{(k)} = \left(\lambda_{m(j)}^{(k)} - \lambda_{l(i)}^{(k)}\right)\Phi_{l(i),m(j)}^{(k)} \quad (2)$$

where $l(i) = \sum_{s=1}^{k} N_s + i$ and $m(j) = \sum_{s=1}^{k-1} N_s + j$, with $i \in (1,...,N_{k+1})$ and $j \in (1,...,N_k)$. In the above expression, $\lambda_{m(j)}^{(k)}$ stand for the first N_k eigenvalues of $\mathbf{\Lambda}^{(k)}$. The remaining N_{k+1} eigenvalues are labelled $\lambda_{l(i)}^{(k)}$.

To help comprehension denote by $x_j^{(k)}$ the activity on nodes j. Then, the activity $x_i^{(k)}$ on node i reads:

$$x_i^{(k+1)} = \sum_{j=1}^{N_k}\left(\lambda_{m(j)}^{(k)}\Phi_{l(i),m(j)}^{(k)}x_j^{(k)}\right) - \lambda_{l(i)}^{(k)}\sum_{j=1}^{N_k}\left(\Phi_{l(i),m(j)}^{(k)}x_j^{(k)}\right) \quad (3)$$

The eigenvalues $\lambda_{m(j)}^{(k)}$ modulate the density at the origin, while $\lambda_{l(i)}^{(k)}$ set the excitability of the receiver nodes, weighting the network activity in its immediate neighbourhood. As remarked in [6], this can be rationalized as the artificial analogue of the *homeostatic plasticity*, the strategy used by living neurons to maintain the synaptic basis for learning, respiration, and locomotion [9].

Starting from this background, we shall hereafter operate within a simplified setting which is obtained by imposing $\lambda_{m(j)}^{(k)} = 0$. This implies that $\lambda_{l(i)}^{(k)}$ are the sole eigenvalues to be actively involved in the training. As we shall prove, these latter eigenvalues provide an effective criterion to rank a posteriori, i.e. upon training being completed, the relative importance of the nodes belonging to the examined network. Stated differently, nodes can be sorted according to their relevance in carrying out the assigned task. This motivates us to introduce, and thoroughly test, an effective spectral pruning strategy which seeks at removing the nodes deemed unessential, while preserving the overall

network classification score. The Methods Section is entirely devoted to explain in detail the proposed strategy, that we shall contextualize with reference to other existing methodologies.

III. CONVENTIONAL PRUNING TECHNIQUES

Generally speaking, it is possible to ideally group various approaches for network compression into five different categories: Weights Sharing, Network Pruning, Knowledge Distillation, Matrix Decomposition and Quantization [10, 11].

Weights Sharing defines one of the simplest strategies to reduce the number of parameters, while allowing for a robust feature detection. The key idea is to have a shared set of model parameters between layers, a choice which reflects back in an effective model compression. An immediate example of this methodology are the convolutional neural networks [12]. A refined approach is proposed in Bat et al. [13] where a virtual infinitely deep neural network is considered. Further, in Zhang et al. [14] an ℓ_1 group regularizer is exploited to induce sparsity and, simultaneously, identify the subset of weights which can share the same features.

Network Pruning is arguably one of the most common technique to compress Neural Network: in a nutshell it aims at removing a set of weights according to a certain criterion (magnitude, importance, etc). Chang et al. [15] proposed an iterative pruning algorithm that exploits a continuously differentiable version of the $\ell_{\frac{1}{2}}$ norm, as a penalty term. Molchanov et al. [16] focused on pruning convolutional filters, so as to achieve better inference performances (with a modest impact on the recorded accuracy) in a transfer leaning scenario. Starting from a network fine-tuned on the target task, they proposed an iterative algorithm made up of three main parts: (i) assessing the importance of each convolutional filter on the final performance via a Taylor expansion, (ii) removing the less informative filters and (iii) re-training the remaining filters, on the target task. Inspired by the pioneering work in [7], Pau de Jorge et al. [17] proved that pruning at initialization leads to a significant performance degradation, after a certain pruning threshold. In order to overcome this limitation they proposed two different methods that enable an initially trimmed weight to be reconsidered during the subsequent training stages.

Knowledge Distillation is yet another technique, firstly proposed by Hinton et al. [18]. In its simplest version Knowledge Distillation is implemented by combining two objective functions. The first accounts for the discrepancy between the predicted and true labels. The second is the cross-entropy between the output produced by the examined network and that obtained by running a (generally more powerful) trained model. In [19] Polino et al. proposed two approaches to mix distillation and quantization (see below): the first method uses the distillation during the training of the so called student network under a fixed quantization scheme while the second exploits a network (termed the teacher network) to directly optimize the quantization. Mirzadeh et al. [20] analyzed the regime in which knowledge distillation can be properly leveraged. They discovered that the representation power gap of the two networks (teacher and student) should be bounded for the method to yield beneficial effects. To resolve this problem, they inserted an intermediate network (the assistant), which sits in between the teacher and the student, when their associated gap is too large.

Matrix Decomposition is a technique that remove redundancies in the parameters by the means of a tensor/matrix decomposition. Masana et al. [21] proposed a matrix decomposition method for transfer learning scenario. They showed that decomposing a matrix taking into account the activation outperforms the approaches that solely rely on the weights. In [22], Novikov et al. proposed to replace the dense layer with its Tensor-Train representation [23]. Yu et al. [24] introduced a unified framework, integrating the low-rank and sparse decomposition of weight matrices with the feature map reconstructions.

Quantization, as also mentioned above, aims at lowering the number of bits used to represent any given parameter of the network. Stock et al. [25] defined an algorithm that quantize the model by minimizing the reconstruction error for inputs sampled from the training set distribution. The same authors also claimed that their proposed method is particularly suited for compressing residual network architectures and that the compressed model proves very efficient when run on CPU. In Banner et al. [26] a practical 4-bit post-training quantization approach was introduced and tested. Moreover, a method to reduce network complexity based on node-pruning was presented by He et al. in [27]. Once the network has been trained, nodes are classified by means of a node importance function and then removed or retained depending on their score. The authors proposed three different node ranking functions: entropy, output-weights norm (onorm) and input-weights norm (inorm). In particular, the input-weights norm function is defined as the sum of the absolute values of the incoming connections weights. As we will see this latter defines the benchmark model that we shall employ to challenge the performance of the trimming strategy here proposed. Finally, it is worth mentioning the Conditional Computation methods [28–30]: the aim is to dynamically skip part of the network according to the provided input so as to reduce the computational burden.

Summing up, pruning techniques exist which primarily pursue the goal of enforcing a sparsification by cutting links from the trained neural network and have been reviewed above. In contrast with them, the idea of our method is to a posteriori identify the nodes of the trained network which prove unessential for a proper functioning of the device and cut them out from ensemble made of active units. This yields a more compact neural network, in terms of composing neurons, with unaltered classification performance. The method relies on the spectral learning [5, 6] and exploits the fact that eigenvalues are credible parameters to gauge the importance of a given node among those composing the destination layer. In short, our aim is to make the network more compact by removing nodes classified as unimportant, according to a suitable spectral rating.

IV. RESULTS

In order to assess the effectiveness of the eigenvalues as a marker of the node's importance (and hence as a potential target for a cogent pruning procedure) we will consider a fully connected feed-forward architecture. Applications of the explored methods will be reported for $\ell = 3$ and $\ell > 3$ configurations. The nodes that compose the hidden layers are the target of the implemented pruning strategies. As we shall prove, it is possible to get rid of the vast majority of nodes without reflecting in a sensible decrease in the test accuracy, if the filter, either in its pre- or post-training versions, relies on the eigenvalues ranking.

For our test, we used three different datasets of images. The first is the renowned MNIST database of handwritten digits [31], the second is Fashion-MNIST (F-MNIST) [32] (an image dataset of Zalando's items) and the last one is CIFAR-10 [33]. In the main text we report our findings for Fashion-MNIST. Analogous investigations carried out for MNIST and CIFAR10 will be reported as supplementary information. Further, different activation functions have been employed to evaluate the performance of the methods. In the main body of the paper, we will show the results obtained for the ELU. The conclusion obtained when operating with the ReLU and tanh are discussed in the annexed supplementary material. In the following we will report into two separate subsections the results pertaining to either the single or multiple hidden layers settings.

A. Single hidden layer ($\ell = 3$)

In Figure 3 the performance of the inspected methods are compared for the minimal case study of a three layers network. The intermediate layer, the sole hidden layer in this config-

uration, is set to $N_2 = 500$ neurons. The accuracy of the different methods are compared, upon cutting at different percentile, following the strategies discussed in the Methods. The orange profile is the benchmark model: the neural network is trained in direct space, by adjusting the weights of each individual inter-nodes connection. Then, the absolute value of the incoming connectivity is computed and used as an importance rank of the nodes' influence on the test accuracy. Such a model has been presented and discussed by He et al. in [27]. Following this assessment, nodes are progressively removed from the trained network, depending on the imposed percentile, and the ability of the trimmed network to perform the sought classification (with no further training) tested. The same procedure is repeated 5 times and the mean value of the accuracy plotted. The shaded region stands for the semi dispersion of the measurements. A significant drop of the network performance is found when removing a fraction of nodes larger than 60 % from the second layer.

The blue curve Figure 3 refers instead to the post-processing spectral pruning based on the eigenvalues and identified, as method (ii), in the Methods Section. More precisely, the three layers network is trained by simultaneously acting on the eigenvectors and the eigenvalues of the associated transfer operators, as illustrated above. The accuracy displayed by the network trained according to this procedure is virtually identical to that reported when the learning is carried out in direct space, as one can clearly appreciate by eye inspection of Figure 3. Removing the nodes based on the magnitude their associated eigenvalues, allows one to keep stable (practically unchanged) classification performance for an intermediate layer that is compressed of about 70% of its original size. In this case the spectral pruning is operated as a post-processing filter, meaning that the neural network is only trained once, before the nodes' removal takes eventually place.

At variance, the green curve in Figure 3 is obtained following method (i) from the Methods Section, which can be conceptualized as a pre-training manipulation. Based on this strategy, we first train the network on the set of tunable eigenvalues, than reduce its size by performing a compression that reflects the ranking of the optimized eigenvalues and then train again the obtained network by acting uniquely on the ensemble of residual eigenvectors. The results reported in Figure 3 indicate that, following this procedure, it is indeed possible to attain astoundingly compact networks with unaltered classification abilities. Moreover, the total number of parameters that need to be tuned following this latter procedure is considerably smaller than that on which the other methods rely. This is due to the fact that only the random directions (the eigenvectors) that prove relevant for discrimination purposes (as signaled by the magnitude of their associated eigenvalues) undergoes the second step of the optimization. This method can also be seen as a similar kind of [7]. As a matter of fact, the initial training of the eigenvalues uncovers a sub-network that, once trained, obtains performances comparable to the original model. More specifically, the uncovered network can be seen as a *winning ticket* [7]. That is, a sub-network with an initialization particularly suitable for carrying out a successful training.

Next, we shall generalize the analysis to the a multi-layer setting ($\ell > 3$), reaching analogous conclusions.

B. Multiple hidden layers ($\ell > 3$)

Quite remarkably, the results achieved in the simplified context of a single hidden layer network also apply within the framework of a multi-layers setting.
To prove this statement we set to consider a $\ell = 5$ feedforward neural network with ELU activation. Here, $N_1 = 784$ and $N_5 = 10$ as reflecting the specificity of the employed dataset. The performed tests follows closely those reported above, with the notable difference that now the ranking of the eigenvalues is operated on the pool of $N_2 + N_3 + N_4$ neurons that compose the hidden bulk of the trained network. In other words, the selection of the neuron to be

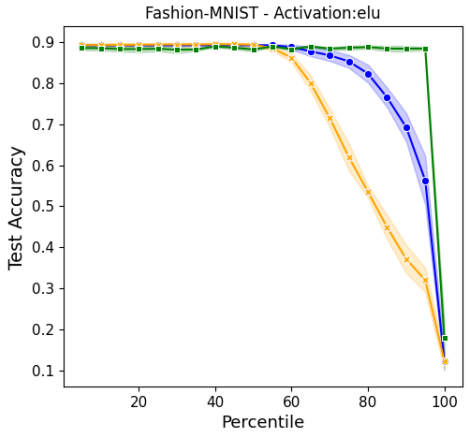


FIG. 3: Accuracy on the Fashion-MNIST database with respect to the percentage of trimmed nodes (from the hidden layer), in a three layers feedforward architecture. Here, $N_2 = 500$, while $N_1 = 784$ and $N_3 = 10$, as reflecting the structural characteristics of the data. In orange the results obtained by pruning the network trained in direct space, based on the absolute value of the incoming connectivity (see main text). In blue, the results obtained when filtering the nodes after a full spectral training (post-training). The curve in green reports the accuracy of the trimmed networks generated upon application of the pre-training filter. Symbols stand for the averaged accuracy computed over 5 independent realizations. The shadowed region is traced after the associated semi-dispersion.

removed is operated after a global assessment, i.e. scanning across the full set of nodes, without any specific reference to an a priori chosen layer.

In Figure 4 the results of the analysis are reported, assuming $N_2 = N_3 = N_4 = 500$. The conclusions are perfectly in line with those reported above for the one layer setting, except for the fact that now the improvement of the spectral pruning over the benchmark reference are even superior. The orange curve drops at percentile 20, while the blue begins its descent at about 60 %. The green curve, relative to the sequential two steps training, stays stably horizontal up to about 90 %.

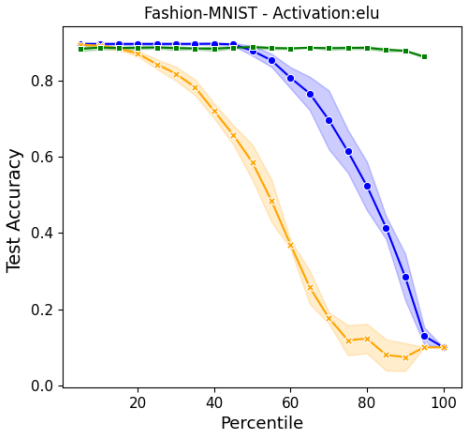


FIG. 4: Accuracy on the Fashion-MNIST database with respect to the percentage of pruned nodes (from the hidden layers), in a five layers feedforward architecture. Here, $N_2 = N_3 = N_4 = 500$, while $N_1 = 784$ and $N_5 = 10$, as reflecting the structural characteristics of the data. Symbols and colors are chosen as in Figure 3.

V. CONCLUSIONS

In this paper we have discussed a relevant byproduct of a spectral approach to the learning of deep neural networks. The eigenvalues of the transfer operator that connects adjacent stacks in a multi-layered architecture provide an effective measure of the nodes importance in handling the information processing. By exploiting this fact we have introduced and successfully tested two distinct procedures to yield compact networks –in terms of number of computing neurons– which perform equally well than their untrimmed original homologous. One procedure (referred as (ii) in the description) is acknowledged as a post processing method, in that it acts on a multi-layered network down-

stream of training. The other (referred as (i)) is based on a sequence of two nested operations. First the eigenvalues are solely trained. After the spectral pruning took place, a second step in the optimization path seeks to adjust the entries of the eigenvectors that populate a trimmed space of reduced dimensionality. The total number of trained parameters is small as compared to that involved when the pruning acts as a post processing filter. Despite that, the two steps pre-processing protocol yields compact devices which outperform those obtained with a single post-processing removal of the unessential nodes.

As a benchmark model, and for a neural network trained in direct space, we decided to rank the nodes importance based on the absolute value of the incoming connectivity. This latter appeared as the obvious choice, when aiming at gauging the local information flow in the space of the nodes, see also [27]. In principle, one could consider to diagonalizing the transfer operators as obtained after a standard approach to the training and make use of the computed eigenvalues to a posteriori sort the nodes relevance. This is however not possible as the transfer operator that links a generic layer k to its adjacent counterpart $k+1$, as follows the training performed in direct space, is populated only below the diagonal, with all diagonal entries identically equal zero. All associated eigenvalues are hence are zero and they provide no information on the relative importance of the nodes of layer $k+1$, at variance with what happens when the learning is carried out in the reciprocal domain.

Summing up, by reformulating the training of neural networks in spectral space, we identified a set of sensible scalars, the eigenvalues of suitable operators, that unequivocally correlate with the influence of the nodes within the collection. This observation translates in straightforward procedures to generate efficient networks that exploit a reduced number of computing units. Tests performed on different settings corroborate this conclusions. As an interesting extension, we will show in the supplementary information that a suitable regularization of the eigenvalues yields a general improvement of the proposed method.

VI. METHODS

We detail here the spectral procedure to make a trained network smaller, while preserving its ability to perform classification.

To introduce the main idea of the proposed method, we make reference to formula (2) and assume the setting where $\lambda^{(k)}_{m(j)} = 0$. The information travelling from layer k to layer $k+1$ gets hence processed as follows: first, the activity on the departure node j is modulated by a multiplicative scaling factor $\Phi^{(k)}_{l(i),m(j)}$, specifically linked to the selected (i,j) pair. Then, all incoming (and rescaled) activities reaching the destination node i are summed together and further weighted via the scalar quantity $\lambda^{(k)}_{l(i)}$. This latter eigenvalue, downstream of the training, can be hence conceived as a distinguishing feature of node i of layer $k+1$. Assume for the moment that $\Phi^{(k)}_{l(i),m(j)}$ are drawn from a given distribution and stay put during optimization. Then, every individual neuron bound to layer $k+1$ is statistically equivalent (in terms of incoming weights) to all other nodes, belonging to the very same layer. The eigenvalues $\lambda^{(k)}_{l(i)}$ gauge therefore the relative importance of the nodes, within a given stack, and as reflecting the (randomly generated) web of local interlayer connections (though statistically comparable). Large values of $|\lambda^{(k)}_{l(i)}|$ suggest that node i on layer $k+1$ plays a central role in the economy of the neural network functioning. This is opposed to the setting when $|\lambda^{(k)}_{l(i)}|$ is found to be small. Stated differently, the subset of trained eigenvalues provide a viable tool to rank the nodes according to their degree of importance. As such, they can be used as reference labels to make decision on the nodes that should be retained in a compressed analogue of the trained neural network, with unaltered classification performance. As empirically shown in the Results section with reference to a variegated set of applications, the sorting of the

nodes based on the optimized eigenvalues turns out effective also when the eigenvectors get simultaneously trained, thus breaking, at least in principle, statistical invariance across nodes.

As we will clarify, the latter setting translates in a post-training spectral pruning strategy, whereas the former materializes in a rather efficient pre-training procedure. The non linear activation function as employed in the training scheme leaves a non trivial imprint, which has to be critically assessed.

More specifically, in carrying out the numerical experiments here reported we considered two distinct settings, as listed below:

- (i) As a first step, we will begin by considering a deep neural network made of N neurons organized in ℓ layers. The network will be initially trained by solely leveraging on the set of tunable eigenvalues. Then, we will proceed by progressively removing the neurons depending on their associated eigenvalues (as in the spirit discussed above). The trimmed network, composed by a total of $M < N$ units, still distributed in ℓ distinct layers, can be again trained acting now on the eigenvectors, while keeping the eigenvalues frozen to the earlier determined values. This combination of steps, which we categorize as pre-training, yields a rather compact neural network (M can be very small) which performs equally well than its fully trained analogue made of N computing nodes.

- (ii) We begin by constructing a deep neural network made of N neurons organized in ℓ layers. This latter undergoes a full spectral training, which optimizes simultaneously eigenvectors and the eigenvalues. The trained network can be compressed, by pruning the nodes which are associated to eigenvalues (see above) with magnitude smaller that a given threshold. This is indeed a post-training pruning strategy, as it acts *ex post* on a fully trained device.

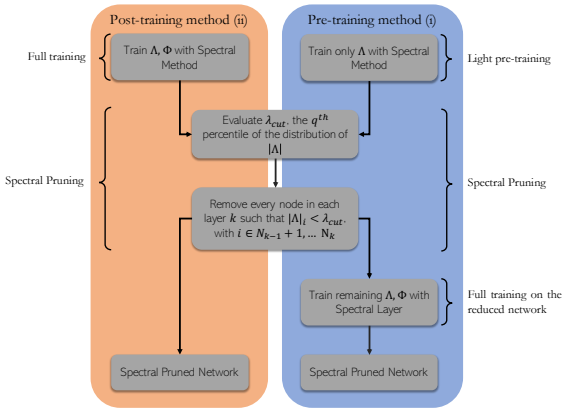


FIG. 5: Flowchart of the pre- and post-training pruning strategies as presented in section VI.

To evaluate the performance of the proposed spectral pruning strategies (schematically represented in the flowchart of Figure 5), we also introduced a reference benchmark model. This latter can be conceptualized as an immediate overturning of the methods in direct space. Simply stated, we train the neural network in the space of the nodes, by using standard approaches to the learning. Then, we classify the nodes in terms of their relevance using a proper metric to which shall make reference below, and consequently trim the nodes identified as less important. When adopting the spectral viewpoint, one can rely on the eigenvalues to rank the nodes importance. As remarked above, in fact, the eigenvalues at the receiver nodes set a local scale for the incoming activity, the larger the eigenvalue (in terms of magnitude) the more important the role played by the processing unit. As a surrogate of the eigenvalues, when anchoring the train in direct space, we can consider the quantity $\sum_{j=1}^{N_k} |w_{ij}|$, for each neuron i belonging to layer $k+1$, see also [27]. The absolute value prevents mutual cancellations of sensible contributions bearing opposite signs, which could incidentally hide the actual importance of the examined node.

In all explored cases, the pruning is realized by imposing a threshold on the reference indi-

cator (be it the magnitude of the eigenvalues or the cumulated flux of incoming –and made positive– weights). Pointedly, the respective indicator is extracted for every node in the arrival layer. Then a percentile q is chosen and the threshold fixed to the q-th percentile. Nodes displaying an indicator below the chosen threshold are removed and the accuracy of the obtained (trimmed) neural network assessed on the test-set. The codes employed, as well as a notebook to reproduce our results, can be found in the public repository of this project [34].

[1] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European conference on computer vision (ECCV)*, pages 784–800, 2018.

[2] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[3] Sorin Grigorescu, Bogdan Trasnea, Tiberiu Cocias, and Gigel Macesanu. A survey of deep learning techniques for autonomous driving. *Journal of Field Robotics*, 37(3):362–386, 2020.

[4] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[5] Lorenzo Giambagli, Lorenzo Buffoni, Timoteo Carletti, Walter Nocentini, and Duccio Fanelli. Machine learning in spectral domain. *Nature communications*, 12(1):1–9, 2021.

[6] Lorenzo Chicchi, Lorenzo Giambagli, Lorenzo Buffoni, Timoteo Carletti, Marco Ciavarella, and Duccio Fanelli. Training of sparse and dense deep neural networks: Fewer parameters, same performance. *Physical Review E*, 104(5):054312, 2021.

[7] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

[8] Charu C Aggarwal et al. Neural networks and deep learning. *Springer*, 10:978–3, 2018.

[9] D James Surmeier and Robert Foehring. A mechanism for homeostatic plasticity. *Nature neuroscience*, 7(7):691–692, 2004.

[10] James O' Neill. An overview of neural network compression. *arXiv preprint arXiv:2006.03669*, 2020.

[11] Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. A survey of model compression and acceleration for deep neural networks. *arXiv preprint arXiv:1710.09282*, 2017.

[12] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[13] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. Deep equilibrium models. *arXiv preprint arXiv:1909.01377*, 2019.

[14] Dejiao Zhang, Haozhu Wang, Mario Figueiredo, and Laura Balzano. Learning to share: Simultaneous parameter tying and sparsification in deep learning. In *International Conference on Learning Representations*, 2018.

[15] Jing Chang and Jin Sha. Prune deep neural networks with the modified $l\_\{1/2\}$ penalty. *IEEE Access*, 7:2273–2280, 2018.

[16] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. *arXiv preprint arXiv:1611.06440*, 2016.

[17] Pau de Jorge, Amartya Sanyal, Harkirat S Behl, Philip HS Torr, Gregory Rogez, and Puneet K Dokania. Progressive skeletonization: Trimming more fat from a network at initialization. *arXiv preprint arXiv:2006.09081*, 2020.

[18] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[19] Antonio Polino, Razvan Pascanu, and Dan Alistarh. Model compression via distillation and quantization. *arXiv preprint arXiv:1802.05668*, 2018.

[20] Seyed Iman Mirzadeh, Mehrdad Farajtabar, Ang Li, Nir Levine, Akihiro Matsukawa, and Hassan Ghasemzadeh. Improved knowledge distillation via teacher assistant. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 5191–5198, 2020.

[21] Marc Masana, Joost van de Weijer, Luis Herranz, Andrew D Bagdanov, and Jose M Alvarez. Domain-adaptive deep network com-

pression. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4289–4297, 2017.

[22] Alexander Novikov, Dmitry Podoprikhin, Anton Osokin, and Dmitry Vetrov. Tensorizing neural networks. *arXiv preprint arXiv:1509.06569*, 2015.

[23] Ivan V Oseledets. Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33(5):2295–2317, 2011.

[24] Xiyu Yu, Tongliang Liu, Xinchao Wang, and Dacheng Tao. On compressing deep models by low rank and sparse decomposition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7370–7379, 2017.

[25] Pierre Stock, Armand Joulin, Rémi Gribonval, Benjamin Graham, and Hervé Jégou. And the bit goes down: Revisiting the quantization of neural networks. *arXiv preprint arXiv:1907.05686*, 2019.

[26] Ron Banner, Yury Nahshan, Elad Hoffer, and Daniel Soudry. Post-training 4-bit quantization of convolution networks for rapid-deployment. *arXiv preprint arXiv:1810.05723*, 2018.

[27] Tianxing He, Yuchen Fan, Yanmin Qian, Tian Tan, and Kai Yu. Reshaping deep neural network for fast decoding by node-pruning. In *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 245–249, 2014.

[28] Xin Wang, Fisher Yu, Lisa Dunlap, Yi-An Ma, Ruth Wang, Azalia Mirhoseini, Trevor Darrell, and Joseph E Gonzalez. Deep mixture of experts via shallow embedding. In *Uncertainty in Artificial Intelligence*, pages 552–562. PMLR, 2020.

[29] Xin Wang, Fisher Yu, Zi-Yi Dou, Trevor Darrell, and Joseph E. Gonzalez. Skipnet: Learning dynamic routing in convolutional networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

[30] Emmanuel Bengio, Pierre-Luc Bacon, Joelle Pineau, and Doina Precup. Conditional computation in neural networks for faster models. *arXiv preprint arXiv:1511.06297*, 2015.

[31] Yann LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

[32] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

[33] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[34] `https://github.com/Buffoni/spectral_learning`.

[35] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[36] Francis Bach, Rodolphe Jenatton, Julien Mairal, and Guillaume Obozinski. Optimization with sparsity-inducing penalties. *Foundations and Trends® in Machine Learning*, 4(1):1–106, 2012.

AUTHORS CONTRIBUTIONS

LB and EC conceived the idea of the work. All authors participated in writing the code. EC and LG performed the experiments on the various datasets. DF supervised the project. All authors contributed to the writing of the manuscript.

COMPETING INTERESTS

The authors declare no competing interests.

Appendix A: Analytical characterisation of inter-nodes weights in direct space

In the following, we will derive Eq. 2 in the main body of the paper. We begin by recalling that $\mathbf{A}^{(k)}$ is a $N \times N$ matrix. From $\mathbf{A}^{(k)}$ we select a square sub-block of size $(N_k + N_{k+1}) \times (N_k + N_{k+1})$, formed by the elements $\mathbf{A}^{(k)}_{i',j'}$ with $i' = \sum_{s=1}^{k-1} N_s + i$ and $j' = \sum_{s=1}^{k-1} N_s + j$, with $i = 1, ..., N_k + N_{k+1}$, $j = 1, ..., N_k + N_{k+1}$. We use $\mathbf{A}^{(k)}$ to identify the obtained matrix and

proceed in analogy for $\mathbf{\Lambda}^{(k)}$ and $\mathbf{\Phi}^{(k)}$. Then:

$$\begin{aligned} A_{ij}^{(k)} &= \left[\mathbf{\Phi}^{(k)}\mathbf{\Lambda}^{(k)}\left(2I-\mathbf{\Phi}^{(k)}\right)\right]_{ij} \\ &= \left[2\mathbf{\Phi}^{(k)}\mathbf{\Lambda}^{(k)}\right]_{ij} - \left[\mathbf{\Phi}^{(k)}\mathbf{\Lambda}^{(k)}\mathbf{\Phi}^{(k)}\right]_{ij} \\ &= \alpha_{ij}^{(k)} - \beta_{ij}^{(k)} \end{aligned} \tag{A1}$$

From hereon, we will omit the apex (k). Assume $\lambda_1 \dots \lambda_{N_k+N_{k+1}}$ to identify the eigenvalues of the transfer operator $\mathbf{A}$, namely the diagonal entries of Λ. Hence, $\Lambda_{ij} = \sum_{j=1}^{N_k+N_{k+1}} \delta_{ij}\lambda_j$. The quantities α_{ij} and β_{ij} read:

$$\begin{aligned} \alpha_{ij} &= 2\sum_{k=1}^{N_k+N_{k+1}} \Phi_{ik}\lambda_k\delta_{kj} = 2\Phi_{ij}\lambda_j \\ \beta_{ij} &= \sum_{k,m=1}^{N_k+N_{k+1}} \Phi_{ik}\lambda_k\delta_{km}\Phi_{mj} \\ &= \sum_{m\in\mathcal{I}\cup\mathcal{J}} \delta_{im}\lambda_m\Phi_{mj} \end{aligned}$$

where $j \in \mathcal{J} = (1, ..., N_k)$ refer to the nodes at the departure layer (k), whereas $i \in \mathcal{I} = (N_k + 1, ..., N_k + N_{k+1})$ stand for those at arrival. Hence, $\mathcal{I} \cup \mathcal{J} = [1, ..., N_k + N_{k+1}]$. The above expression for β_{ij} can be further manipulated to eventually yield

$$\begin{aligned} \beta_{ij} &= \sum_{m\in J} \Phi_{im}\lambda_m\Phi_{mj} + \sum_{m\in I} \Phi_{im}\lambda_m\Phi_{mj} \\ &= \Phi_{ij}\lambda_j + \lambda_i\Phi_{ij} \end{aligned}$$

and therefore: (A1) as

$$\begin{aligned} \alpha_{ij} - \beta_{ij} &= 2\Phi_{ij}\lambda_j - \Phi_{ij}\lambda_j - \lambda_i\Phi_{ij} \\ &= (\lambda_j - \lambda_i)\phi_{ij} \end{aligned} \tag{A2}$$

From the above expression, one obtains the sought equation, after redefining the index i to have it confined in the interval $[1, ..., N_{k+1}]$. By definition, the matrix of the weights, $\mathbf{w}$, is in fact a $N_k \times N_{k+1}$ matrix.

Appendix B: MNIST and Fashion-MNIST: single hidden layer with different activation functions.

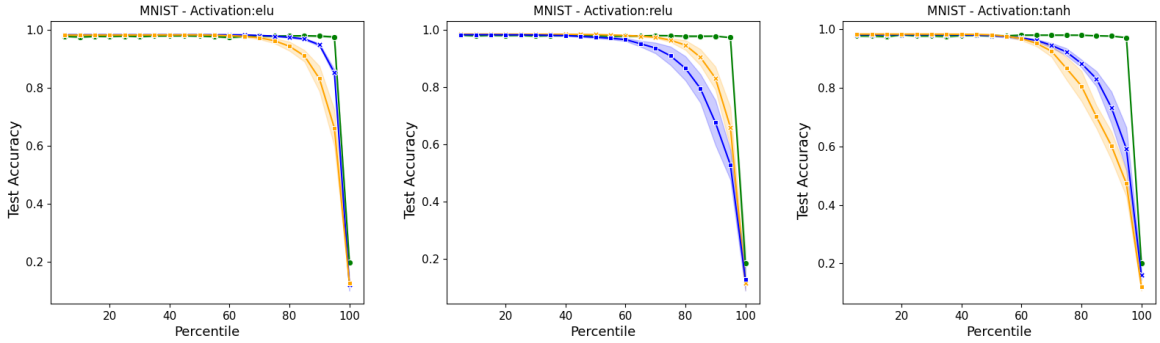
We shall here report (see Figures 6a, 6b, 6c, 7a and 7b) on the performance of the proposed trimming strategies, as applied to MNIST and Fashion-MNIST, for a single hidden layer architecture and beyond the setting reported in the main body of the paper. In particular, we will assume (i) ELU, tanh and ReLU for MNIST (ii) tanh and ReLU activation function for Fashion-MNIST (the ELU activation was employed in the main text). Here, $N_2 = 500$, while $N_1 = 784$ and $N_3 = 10$.

Appendix C: MNIST and Fashion-MNIST: multiple hidden layers with different activation functions.

We will here generalize the analysis carried out in the preceding section to the case of a multilayered ($\ell > 3$) architecture (see Figures 8a, 8b, 8c, 9a and 9b). In line with the choice operated in the main body of the paper, we will assume a five layered deep neural network with $N_2 = N_3 = N_4 = 500$, and $N_1 = 784$ and $N_5 = 10$.

Appendix D: Testing the trimming strategies on CIFAR10 dataset.

To assess the flexibility of the schemes outlined in Section III-B we here consider the CIFAR10 dataset and assume a modified MobileNetV2 [35] adding two dense layer at the end of the network. During training we freeze all the layers, except for the two appended dense layers. These latter are trained in the spectral domain. Working in this setting, the pruning is performed on the first dense layer by using strategies both (i) and (ii), as introduced in the main body of the paper. Here again the results are compared to those obtained when using the absolute value of the incoming connectivity as an alternative trimming criterion (see Figures



(a) (b) (c)

FIG. 6: Accuracy on the MNIST database with respect to the percentage of trimmed nodes (selected from the 500 neurons that compose the sole hidden layer), in a three layers feedforward architecture. The results reported in each panel refer to a different selection of the nonlinear activation functions, respectively ELU (a), ReLU (b) and tanh (c). In orange, the results obtained by using the trimming procedure based on the absolute value of the incoming connectivity. In blue, the results obtained when filtering the nodes after a full spectral training (post-training). The curve in green displays the accuracy of the trimmed networks generated upon application of the pre-training filter. In this case, the examined network is initially trained on the set of eigenvalues, while keeping the eigenvectors frozen. After having removed unessential nodes, based on their associated eigenvalues, the network undergoes another training phase that is solely targeted to adjusting the entries of the residual eigenvectors. The shadowed region represents the semi-dispersion over 5 independent realizations. When using the Relu function, trimming on the absolute value of the incoming connectivity yields slightly better results than what found when using the post-training spectral filter. The two stages spectral trimming proves always more effective.

10a, 10b and 10c). As a further step in the analysis, we also introduce and test a ℓ_1-norm regularization acting on the eigenvalues, so as to induce a sparse solution [36]. All experiments are performed by using a MobileNetV2 based architecture. The first dense layer is made of 512 nodes with an ELU activation function (others activation functions yield analogous results). The following regularization loss functions are considered depending on whether the training takes place in the reciprocal (spectral layer) or direct space:

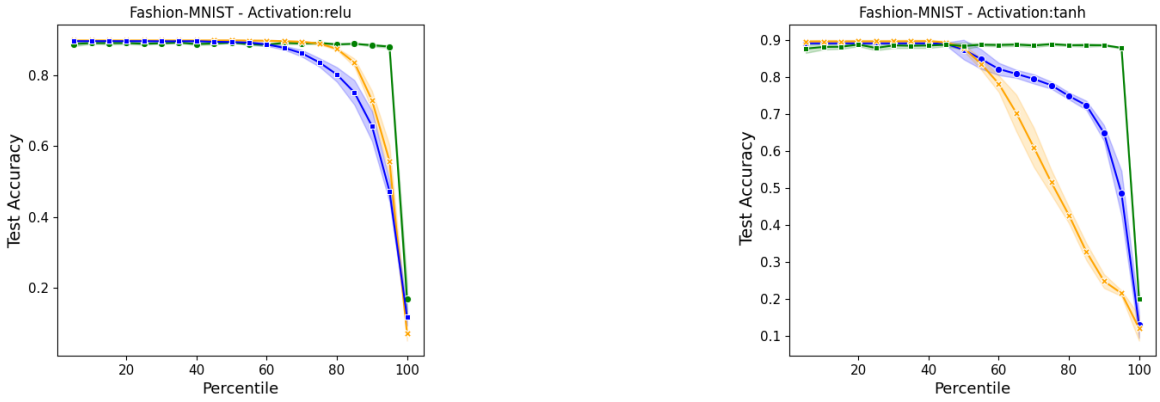
- Spectral regularization

$$L_r^{\text{spec}} = \gamma * \sum_{i=1}^{N_{\ell-1}} |\lambda_i^{(\ell-1)}|$$

- Connectivity regularization

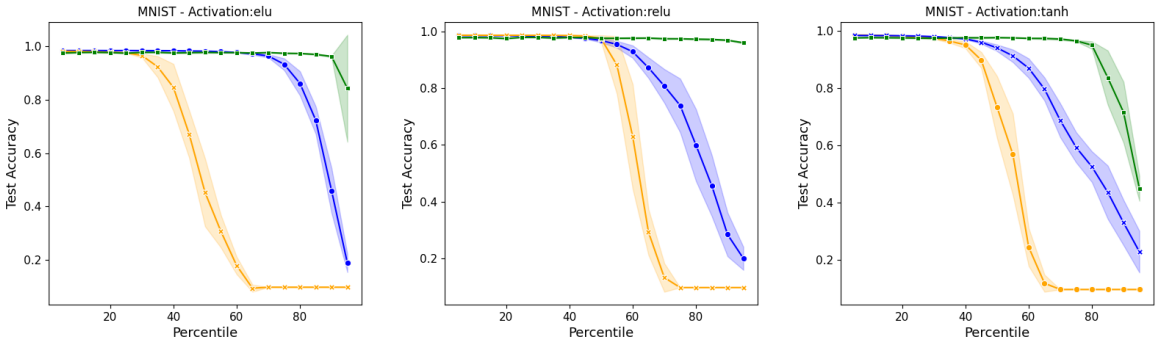
$$L_r^{\text{conn}} = \gamma * \sum_{i,j} |w_{ij}^{(\ell-1)}|$$

where γ stands for a suitable regularizer weight. Clearly L_r^{conn} is equivalent to a regularization which acts on the incoming absolute connectivity. In fact, $|\sum_i |x_i|| = \sum_i |x_i|$.
The ℓ_1 regularization impacts significantly on the classification accuracy, as it can be clearly appreciated by direct inspection of Figure 11. Choosing the correct regularizer weight (γ), the performance of the network are stable across various range of pruning thresholds, even at the highest percentile.



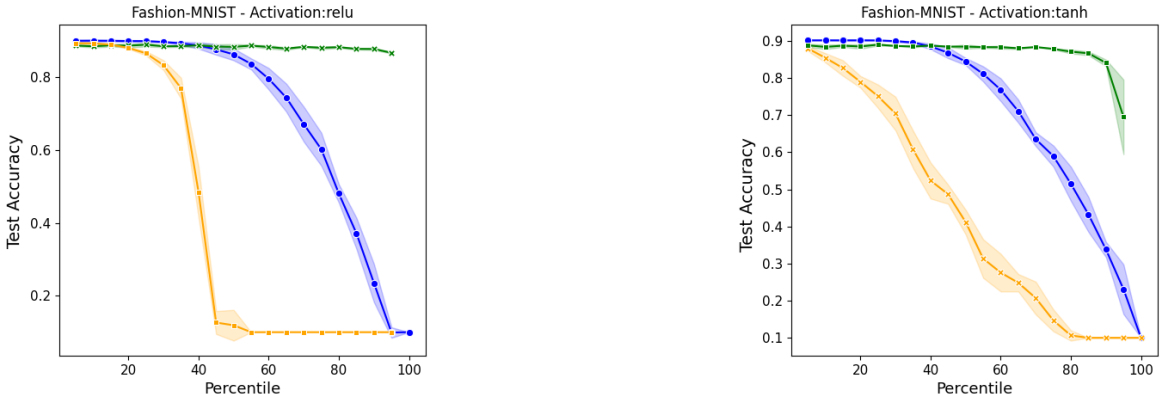
(a) (b)

FIG. 7: Accuracy on the Fashion-MNIST database with respect to the percentage of trimmed nodes (selected from the 500 neurons that compose the sole hidden layer), in a three layers feedforward architecture. The results reported in each panel refer to a different selection of the nonlinear activation functions, respectively ReLU (b) and tanh (c). Symbols and conclusions are in line with those reported for the case of MNIST.



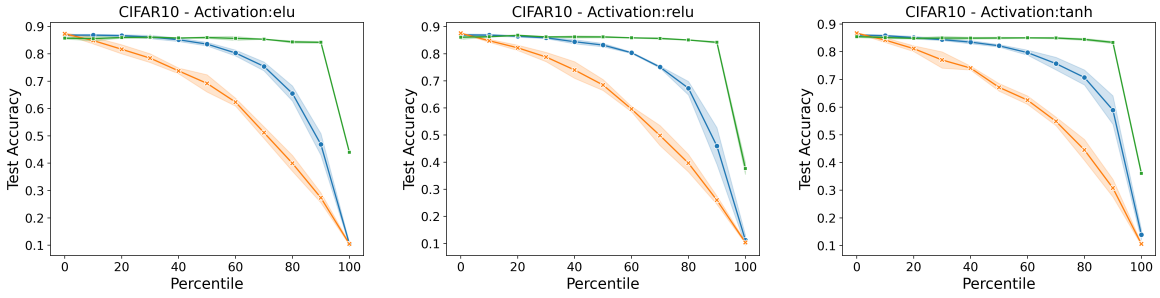
(a) (b) (c)

FIG. 8: Accuracy on the MNIST database with respect to the percentage of trimmed nodes (from the set of $N_2 + N_3 + N_4$ neurons). The results in each panel refer to different choices of the non linear function, ELU (a), ReLU (b) and tanh (c). Symbols are chosen as for the case of the single hidden layer setting. It should be remarked that the spectral trimming strategies proves definitely more effective than the benchmark model anchored to direct space, also when the Relu function is employed, in the case of multiple hidden layers.



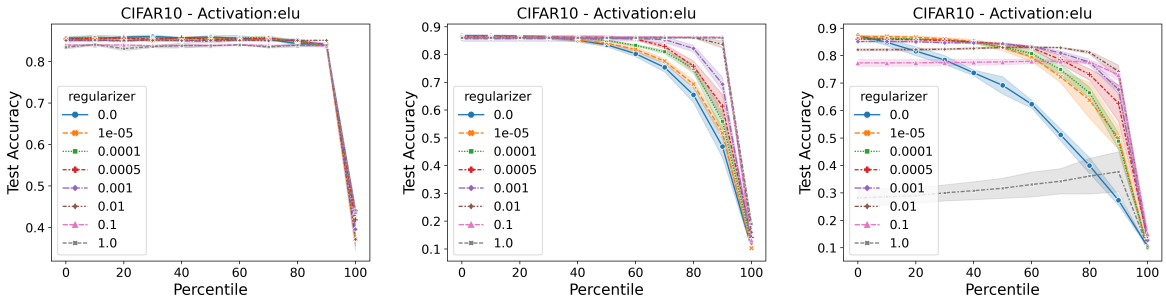
(a) (b)

FIG. 9: Accuracy on the Fashion-MNIST database with respect to the percentage of trimmed nodes (from the set of $N_2 + N_3 + N_4$ neurons). The results in each panel refer to different choices of the non linear activation function, ReLU (a) and tanh (b). For the symbols, see the caption of the Figures above. Also in this case the spectral filters prove always superior.



(a) (b) (c)

FIG. 10: Accuracy on the CIFAR10 database with respect to the percentage of trimmed nodes (from the $\ell - 1$ layer). The results in each panel refer to different non linear functions, respectively ELU (a), ReLU (b) and tanh (c). Symbols are chosen in analogy with the above (the result drawn in green are based on two different runs).



(a) (b) (c)

FIG. 11: Computed accuracy on the CIFAR10 dataset against the percentage of trimmed nodes (from the first of the two dense layers appended to the MobileNet-like architecture). The panels displays the performance of the network as according to each trimming procedure, and using weights (W) for the ℓ_1 regularizer. In panel (a) and (b) pre-training (based on two runs) and post-spectral filter, respectively; in panel (c) the reduction schem based on the absolute connectivity.