Investigating Similarity Metrics for Convolutional Neural Networks in the Case of Unstructured Pruning

Alessio Ansuini[1][0000−0002−3117−3532], Eric Medvet[2][0000−0001−5652−2113], Felice Andrea Pellegrino[2][0000−0002−4423−1666], and Marco Zullich[2][0000−0002−9920−9095]

[1] Research and Technologies Institute, AREA Science Park, Trieste, Italy
alessio.ansuini@areasciencepark.it
[2] Department of Engineering and Architecture, University of Trieste, Trieste, Italy
{emedvet, fapellegrino}@units.it; marco.zullich@phd.units.it

Keywords: Machine Learning · Pruning · Convolutional Neural Networks · Lottery Ticket Hypothesis · Canonical Correlation Analysis · Centered Kernel Alignment · Network similarity · Explainable AI

Abstract. Deep Neural Networks (DNNs) are essential tools of modern science and technology. The current lack of explainability of their inner workings and of principled ways to tame their architectural complexity triggered a lot of research in recent years. There is hope that, by making sense of representations in their hidden layers, we could collect insights on how to reduce model complexity—without performance degradation—by *pruning* useless connections. It is natural then to ask the following question: how similar are representations in pruned and unpruned models? Even small insights could help in finding principled ways to design good lightweight models, enabling significant savings of computation, memory, time and energy. In this work, we investigate empirically this problem on a wide spectrum of similarity measures, network architectures and datasets. We find that the results depend critically on the similarity measure used and we discuss briefly the origin of these differences, concluding that further investigations are required in order to make substantial advances.

1 Introduction

It is not fully understood why Deep Neural Networks (DNNs) generalize well to new data also in conditions of severe overparametrization—in which the network capacity would be enough to memorize large datasets—and what is the class of functions that these networks are able to learn [37,1]. In order to make sense of inner workings of DNNs, in recent years many new methods have been introduced with the purpose of comparing representations in trained networks and during training [28,20,32]. On the other hand, recent pruning techniques proved very effective in reducing big models to a tiny fraction of their original size, without

substantial degradation in performance [12,29,8,11]. These findings suggest that DNNs are able to represent data in effective ways also when they are forced to get rid of most of their parameters (up to more than 99 %).

In the present exploratory work, we address, empirically, the following question: "How similar are representations in pruned and unpruned models?" We focus on Convolutional Neural Networks (CNNs) engaged in different Computer Vision tasks, training each model from scratch and pruning it with Iterative Magnitude Pruning (IMP) [16,29]. We then extract internal representations of the CNN layers and compare them by means of several recently introduced similarity measures.

The present work is an extension of [3], in which we trained multiple CNNs (from a single architectural family) on five subsets of increasing complexity of the dataset CIFAR-10 [21], pruning these networks using IMP. There, we observed a peculiar trend in the SVCCA similarity, particularly when we considered the complete dataset, from which we concluded that intermediate layers differ the most between pruned and unpruned networks; the following questions, however, remained unanswered:

- are these trends shared across datasets and architectures?
- how these trends depend on the particular notion of similarity used?

In the present work, we address these research questions by performing a more thorough experimental analysis w.r.t. [3] in terms of architectures, datasets, and similarity measures. We show that some of the most recently introduced similarity criteria, result in a remarkable stability of similarity across layers, thus suggesting that the action of IMP pruning (even when the resulting sub-network is much smaller than the original architecture) do not change substantially the representations. In order to make our findings reproducible and more accessible, we made the code publicly available at https://github.com/marcozullich/pruned_layer_similarity.

The remainder of the paper is organized as follows: in Section 2 we describe pruning techniques, stressing the innovations introduced in our investigation. In Section 3 we present in detail the employed techniques, namely, IMP and the similarity measures. In Section 4 we describe how these techniques are applied in this work, and provide a description of datasets, architectures, and hyperparameters. In Section 5 we describe our results on performance and layer-wise similarities between pruned and unpruned networks. Finally, in Section 6 we discuss critically our results, putting an accent on the difference of the results obtained with different similarity measures.

2 Related Work

2.1 Techniques for DNN pruning

DNN pruning techniques may be split into two macro-categories [4]: unstructured and structured. Unstructured pruning acts without following "a specific

geometry or constraint, [...] it leads to irregular sparsity". Structured pruning, on the contrary, acts on well-defined substructures of the DNN, such as a single neuron or a whole convolutional filter; IMP belongs to the first category [16]. In [29] various pruning techniques, both unstructured and structured, are compared on Computer Vision tasks; the authors show that unstructured pruning leads to the best outcomes, in particular IMP with *Learning Rate Rewind* (LRR), that we describe in Section 3.1 and use throughout the present work.

2.2 Comparisons between pruned and unpruned DNNs

There exist several works analyzing analogies between pruned and unpruned DNNs. Apart from *performance* comparisons (test-set accuracy, time efficiency, energy consumption, *etc.*) which are the staple of the majority of such works, there is a minority of studies concentrating on other aspects.

Other types of comparison include, for instance, *calibration* [34] and *robustness* [35]; in [11] the *interpretability* of pruned CNNs w.r.t. their unpruned counterparts is considered.

Interpretability is related to the number of "convolutional units that recognize particular human-interpretable concepts", and it has been shown that pruning with IMP do not substantially alter interpretability. A more recent work compared *pruning masks* obtained from various pruning techniques, including IMP [26]. A pruning mask is a binary structure identifying with a 0 the parameters of the original DNN that have been pruned, and with a 1 the surviving ones. The authors compared masks using the Jaccard similarity (intersection-over-union). Although the relationship between representations and pruning masks is not immediate, the work provides interesting insights about the parallelisms between structured and unstructured pruning; moreover, the authors empirically show that, given an unpruned DNN, there exist multiple sub-networks that can match its test accuracy.

In our previous work [3], we addressed layer-wise similarities between pruned and unpruned DNNs using a similarity metric specifically designed for neural networks, Mean CCA Similarity [28] (see also Section 3.2). We are not aware of other works dedicated to layer-by-layer comparisons of pruned and unpruned DNNs.

3 Tools

In this section we present the tools we used to perform our experiments: IMP—together with the main strategies for its application—and the similarity metrics used to compare representations.

3.1 IMP

IMP, first proposed in [16], is a pruning technique that iteratively performs the following three steps: (1) start from a fully trained neural network (*complete* or

unpruned network), (2) prune parameters with magnitude falling below a given quantile, (3) re-train the network with the remaining parameters for a given number of epochs. At each iteration, IMP produces a network with increased sparsity, i.e., with a larger portion of weights set to 0.

Several variants on this theme are possible, based on how to choose the parameters to be removed:

- *Global pruning* [12]: all the parameters are pooled together and a given proportion of them is pruned, regardless of the layer they belong to.
- *Local pruning* [12]: a fixed proportion of low-magnitude parameters is pruned from each layer; the threshold is determined layer-by-layer.
- *Mixed pruning* [38]: a hybrid between global and local pruning—parameters may be pooled in separate structures depending on the layer they belong to, and pruning may be applied separately for each pool. For instance, in [38] the authors experiment with pooling separately parameters from convolutional layers and fully-connected layers.

In [24] it was shown that global pruning outperforms, in terms of test-set accuracy, the other two strategies; hence, in this work, we will use the former.

When IMP was introduced, it was customary to re-train the pruned neural network for a small number of iterations (*fine-tuning*, FT) while keeping the Learning Rate (LR) fixed at the same value as the last epoch of training of the complete network. Since then, this procedure has been shown [29] to produce subpar results as far as accuracy is concerned. Additional heuristics for re-training were introduced in subsequent works:

- *Weight Rewind* (WR) [12]: before re-training, *rewind* the surviving parameters to their configuration at initialization. This strategy showed promising results, albeit only on relatively small architectures; it was later found in [13] that rewinding at a configuration of a training iteration *close* to the initial one (*late resetting* strategy) allowed for the successful application of WR also on larger networks, such as VGG19 [30].
- *Learning Rate Rewind* (LRR) [29]: before re-training, do not modify the value of the surviving parameters, but re-use the same LR schedule that was used in the complete network. The authors also brought empirical evidence that LRR produces networks having better accuracy than WR and FT, especially at higher pruning rates.

In [3], we employed IMP with WR and late resetting. In the present work, we decided to switch to IMP with LRR, due to its better empirical results. As we will later note in Section 5, if we restrict our considerations on SVCCA, the results we get are comparable to the ones obtained in [3], suggesting that, as far as the representations of the layers of the neural networks are concerned, there is not much difference between IMP with WR and IMP with LRR.

3.2 Similarity metrics for neural networks data representations

In this section, we present the various metrics to compare representations in the (hidden) layers of neural networks which we employed in this work.

These metrics allow for the comparison between generic layers of neural networks (possibly belonging to different architectures) through the *representations*, i.e., the *activation matrices* obtained as output of such layers in response to a fixed dataset [28]. For a dataset composed by n data points, a generic fully-connected layer of p neurons is hence represented by an $n \times p$ matrix.

Comparing layers with *convolutional structure* We say that a layer has a convolutional structure when its representation is not two-dimensional, as is the case of fully-connected layers, but four-dimensional. For instance, convolutional layers and two-dimensional max-pooling layers fall under this category.

Generally, we can represent such layers with four-dimensional tensors having shape $n \times c \times h \times w$, where n is the number of data points through which we obtain the representation, c is the number of channels in the layer, and h and w are the *spatial dimensions* of the data point at the layer[3].

Since the metrics that we are going to present work with matrices, additional strategies for treating these tensors are needed in order to produce a two-dimensional matrix out of the original four-dimensional tensor. These strategies might differ from metric to metric.

The present work is focused on CNNs operating on two-dimensional images, thus we will not consider strategies to compare layers whose representations have larger dimensions (e.g., three-dimensional convolutions).

Metrics based on Canonical Correlation Analysis (CCA) CCA [18] is a multivariate statistical technique to compare two representations of phenomena described starting from the same dataset of n units. CCA finds two linear transforms such that, when applied to the two representations, produce a set of orthonormal vectors in a common space, and these vectors have maximum pair-wise Pearson correlation. In our case, the phenomena are the two layers we wish to compare.

Formally, given two matrices L_1, L_2 such that $L_1 \in \mathbb{R}^{n \times p_1}$ and $L_2 \in \mathbb{R}^{n \times p_2}$, let $\tilde{p}$ be the minimum between p_1 and p_2. The two sought transforms are called $W_1 \in \mathbb{R}^{p_1 \times \tilde{p}}$ and $W_2 \in \mathbb{R}^{p_2 \times \tilde{p}}$.

By applying them to $L1$ and L_2, we get

$$Z_1 = L_1 W_1 \tag{1}$$
$$Z_2 = L_2 W_2, \tag{2}$$

with $Z_1, Z_2 \in \mathbb{R}^{n \times \tilde{p}}$. Denoting by $z_i^{(1)}$ the i-th column of Z_1 (also called the i-th *canonical vector*), we wish that:

a) $z_i^{(k)} \perp z_j^{(k)}$, with $k \in \{1, 2\}, i, j \in \{1, \ldots, \tilde{p}\}, i \neq j$, and

[3] For *spatial dimension* we mean the size of each single channel of the image (or analogous two-dimensional structure) after the application of the convolutions operated by the given layer.

b) the pair $z_i^{(1)}, z_i^{(2)}$, with $i \in \{1, \dots, \tilde{p}\}$ maximizes the residual Pearson correlation, called *canonical correlation* (CC), identified by ρ_i.

Hence, we obtain a sequence of column vectors pairs $(z_i^{(1)}, z_i^{(2)})_{i \in \{1,\dots,\tilde{p}\}}$ exhibiting decreasing CCs.

Mean CCA Similarity These values of CC may be averaged [28] to obtain a similarity metric called **Mean CCA Similarity**:

$$\text{Mean CCA Similarity}(L_1, L_2) \doteq \frac{\sum_{i=1}^{\tilde{p}} \rho_i}{\tilde{p}} \tag{3}$$

The CCs may actually be obtained [33] in a one-shot fashion from the Singular Value Decomposition (SVD), applied to a matrix derived from the variance-covariance matrices of the two layers L_1, L_2.

SVCCA The Singular Vector Canonical Correlation Analysis (SVCCA), was proposed in [28]. The idea behind this technique is to perform a SVD for dimensionality reduction of the representation of the layers, then operate the CCA as explained in the previous section. The authors of [28] recommend to apply SVD such that only the singular values accounting for the 99 % of variance are retained.

Such a dimensionality reduction is motivated by the observation that, in layers representations, "many low variance directions (neurons) are primarily noise". Moreover, the SVD is expected to reduce overestimation of the similarity on some degenerate configurations for the layers.

The Mean CCA Similarity calculated when using SVCCA will be referred to as **Mean SVCCA Similarity**.

PWCCA In [23], it was argued that the Mean CCA Similarity may misrepresent the similarity between two layers, given the fact that all the correlation coefficients in Equation (3) are weighted equally. It was then proposed to weigh the CCs according to the contribution of the corresponding canonical vectors in determining the representation of the original layer.

We recall that $L_1 \in \mathbb{R}^{n \times p_1}$ and $L_2 \in \mathbb{R}^{n \times p_2}$ are the layers representation. We call $l_m^{(1)} \in \mathbb{R}^n$ the m-th column vector of L_1. It contains the activations of the m-th neuron to all the n data points. We suppose, without loss of generality, that $p_1 \leq p_2$, so that $\tilde{p} = \min(p_1, p_2) = p_1$. The canonical vectors of L_1, $z_i^{(1)} \in \mathbb{R}^n$, are the column vectors of $Z_1 = L_1 W_1$. Then, the contribution α_i of $z_i^{(1)}$ in determining L_1 is:

$$\alpha_i \doteq \sum_{j=1}^{p_1} |l_j^T z_i^{(1)}| \tag{4}$$

where ^{T} denotes transposition. In case $p_2 < p_1$, Equation (4) must be formulated using the corresponding indices and vectors from L_2 instead of L_1.

Plugging the contributions as weights of the mean in Equation (3), we get the target similarity metric, called PWCCA (Projection Weighted Canonical Correlation Analysis):

$$\text{PWCCA}(L_1, L_2) \doteq \frac{\sum_{i=1}^{\tilde{p}} \alpha_i \rho_i}{\sum_{i=1}^{\tilde{p}} \alpha_i} \tag{5}$$

Handling of layers with convolutional structure When comparing layers with convolutional structure, whose representation is a 4D tensor, the tensors are reshaped as in [28]. Precisely, given $L_1 \in \mathbb{R}^{n \times c \times h \times w}$, the spatial dimensions h and w are *merged* into the first dimension (the one concerning the data points), thus yielding a matrix $\tilde{L}_1 \in \mathbb{R}^{nhw \times c}$. The reason for this reshaping is that, as noted in [28], the *degrees of freedom* of neurons in layers with convolutional structures are limited, due to the spatial structure of the filters, to the channel only. Incidentally, the reshaping tackles the issue raised in [20] that, when the number of data points is smaller than the number of neurons, CCA-based similarity metrics produce inappropriate results. Indeed, usually, despite $n < chw$, it also happens that $nhw > c$: thus, in such case, the reshaping produces representations where the number of data points is larger than the number of neurons.

This procedure has a major limitation concerning the comparison between layers having different spatial dimensions. Let's suppose that the two layers have spatial dimensions $h \times w$ and $h' \times w'$ respectively (with c and c' channels respectively), the corresponding representations, reshaped as above, have size $nhw \times c$ and $nh'w' \times c'$. Since a requirement for CCA is that the number of rows must be the same between the two representations (i.e., the number of data points in the representations of the two layer must be the same), it follows that the metrics based on CCA cannot be obtained in this case. Similarly, it is also impossible to compare, e.g., a convolutional layer with a fully-connected layer.

Though, to our knowledge, the existing literature has not proposed solutions to this problem, the authors of [28] and [23] have rendered their code and consequent considerations public in [14], while proposing two techniques to enable the comparison: (a) interpolation, to upsample the smaller image such that it matches the larger image spatial dimensions, and (b) average pooling along the spatial dimensions, to *average out* the two spatial dimensions and obtain a matrix of shape $n \times c$. We remark that we will not be needing these metrics, since we will always be comparing convolutional and max-pooling layers having the same number of data points and the same spatial dimensions.

3.3 Kernel-based metrics

Metrics based on kernels are computed from the Gram matrices obtained from the representations of the layers. In a sense, these techniques consider an additional step from those based on CCA, viewing the representation of the layer not as the activation matrix L, but as the Gram matrix obtained from applying a kernel function κ to the rows of L itself.

Centered Kernel Alignment (CKA) CKA [6,7] is a similarity metric for kernels computed from representations obtained from the same number of data points n. [20] makes the case for CKA as a similarity index particularly suitable for comparing DNNs layers representations. The authors cite the invariance to orthogonal transformations and isotropic scalings of the space as desirable properties enjoyed by CKA.

We consider once again our two layers representations $L_1 \in \mathbb{R}^{n \times p_1}$ and $L_2 \in \mathbb{R}^{n \times p_2}$, with corresponding Gram matrices $K_1 \in \mathbb{R}^{n \times n}$ and $K_2 \in \mathbb{R}^{n \times n}$, both positive semi-definite and obtained out of universal kernel functions κ_1 and κ_2. We constrain L_1, L_2 to be centered w.r.t. their column means[4] or, alternatively, their Gram matrices K_1, K_2 must be centered by subtracting both the column and row means[5].

CKA is calculated as:

$$\mathrm{CKA}(K_1, K_2) \doteq \frac{\langle K_1, K_2 \rangle_F}{\|K_1\|_F \|K_2\|_F} \tag{6}$$

where $\|K\|_F = \sqrt{\mathrm{tr}(KK^T)}$ is the Frobenius norm, and $\langle K_1, K_2 \rangle_F = \mathrm{tr}(K_1 K_2^T)$ is the Frobenius inner product.

CKA itself is a normalization[6] of the Hilbert-Schmidt Independence Criterion [15] to test the independence between sets of random variables.

In [20], it is argued that CKA is a metric which is fitter than CCA-based similarities to compare DNN representations. Besides the aforementioned invariances, the authors base this claim upon two experiments in which they record much greater success by CKA in recognizing similarity patterns in architecturally equivalent DNNs trained from different random initializations.

Normalized Bures Similarity (NBS) A more recent work [32], proposes, in addition to CKA, the use of another metric based on Gram matrices, NBS [5]. Still, the requirement is that L_1, L_2, or K_1, K_2, must be centered as explained for CKA.

It is calculated as:

$$\mathrm{NBS}(K_1, K_2) = \frac{\sqrt{\mathrm{tr}(K_1^{1/2} K_2 K_1^{1/2})}}{\sqrt{\mathrm{tr}\, K_1 \,\mathrm{tr}\, K_2}} \tag{7}$$

Also this quantity is comprised between 0 and 1. [32] refers to it as an alternative to CKA, sharing all of its invariances. It is to be noted, though, that the authors argue, on limited settings, that this metric may be inferior w.r.t. CKA.

Considerations on kernels Both CKA and NBS are, in their respective works, calculated on Gram matrices obtained with linear kernels or variations of them.

[4] I.e., for each column, its mean across the instances must be 0.
[5] I.e., for each row/column, its mean across the columns/rows must be 0.
[6] Such that its value lies between 0 and 1.

In [20], some experiments are conducted with Radial Basis Function kernels, finding that results obtained with these kernels are similar with respect to linear kernels, preferring the latter due to their simplicity and absence of tunable parameters. Taking this into account, we will also calculate CKA and NBS using linear kernels only.

In [32], only a variation of a linear kernel is used. This kernel is modified in order to incorporate the gradient of the DNN. They do so by calculating Gram matrices on (a) the values of the parameters of the layer, (b) the values of the gradient of the loss function w.r.t. the parameters of the layer, and combining the two via the Hadamard product (matrix element-by-element product) obtaining a new positive semidefinite Gram matrix. While we believe that this technique may be well suited to compute similarities of DNNs in their training steps, we think that gradients are not needed in order to compare feature maps of fully-trained networks, although it may be interesting to incorporate this metric in future analyses. We will hence conform to work with Gram matrices obtained strictly on the parameter space.

4 Methods

In order to carry out our exploration, we operated following this scheme:

1. We trained a *complete* (unpruned) CNN on a given dataset and a given optimizer until convergence.
2. We pruned it using IMP with LRR for 20 iterations with a pruning rate of 20 %. This value is suggested as a default in [29]. The pruning was operated for P runs, each time re-starting from the same complete CNN. We averaged the similarities over these runs in order to get more robust results. After the execution of IMP for 20 iterations, the pruned CNNs have a sparsity level of around 98.5 %[7] (or, alternatively, around 1.5 % of the parameters survive the pruning operation).
3. We obtained the representations for the hidden layers of all the CNNs, both unpruned and pruned.
4. We compared, using multiple similarity metrics, each layer of the pruned CNNs with the corresponding layer of the complete CNN.

We repeated this process over different datasets and CNN architectures:

- CNN on the dataset CIFAR-10 [21]; architecture based on VGG16 [30].
- CNN on the dataset CIFAR-10; architecture based on ResNet [17].
- CNN on the dataset SVHN [25]; architecture based on VGG [30].

The details about the datasets, the CNN architectures, and the optimizers used are presented in Sections 4.1 and 4.2. The metrics employed in the comparison are Mean SVCCA Similarity, PWCCA, CKA, and NBS.

[7] The exact value depends on the presence of layers or parameters not affected by pruning, such as batch normalization.

4.1 Datasets

CIFAR-10 CIFAR-10 [21] is a dataset for image classification composed of 60 000 color images having size 32×32 pixels. The images belong to 10 classes, of which 6 are animals and the other 4 are means of transportation. The dataset, which is available on https://www.cs.toronto.edu/~kriz/cifar.html, is already split into a training set composed of 50 000 images and a test-set of 10 000 images.

In our experiments, the dataset was augmented by applying random data augmentation schemes (cropping, horizontal flipping, affine transformation, cut-out).

SVHN SVHN [25] is a dataset composed of 99 289 color images of size 32×32 pixels, each exhibiting a caption of a house number extracted from the online service Google Street View of Google Maps[8]. Each image is centered into a single digit, which is the target of the classification. Hence, there are 10 classes (digits from 0 to 9). Some images may exhibit more than one digit: in this case, only the central digit must be considered and the other digits act as distractors. The dataset, which is available on http://ufldl.stanford.edu/housenumbers/, is already split into a training set of 73 257 images and a test-set of 26 032 images. Moreover, an additional dataset of 531 131 images may be used for training purposes (e.g., as a validation dataset) although it is specified in the hosting site that these examples are "somewhat less difficult".

SVHN is currently available both as an image classification dataset and as an object recognition dataset. In this second case, the images are larger than 32×32 and the target is to recognize all of the digits present inside the images. In this work, we refer to SVHN solely as an image classification dataset.

4.2 CNN architectures and optimizers used

VGG [30] is a relatively simple family of CNN architectures characterized by a cascade of *convolutional blocks* followed by fully-connected layers and the output layer, having as many neurons as the number of classes. Each convolutional block is composed of 2, 3, or 4 convolutional layers having the same number of filters (usually doubling the number of filters of the convolutional layers of the previous block), followed by a final max-pooling layer.

VGG16 for CIFAR-10 VGG16 is a specification of VGG with 5 convolutional blocks.

The original implementation uses the ReLU activation function for all the hidden layers and has 2 fully-connected hidden layers at the end; moreover, no dropout [31] or batch normalization (BN) [19] are employed. Our implementation differs from the original one, tracing the recent work on IMP (see [29]):

– we employ BN before the activation function in each convolutional layer;

[8] https://maps.google.com

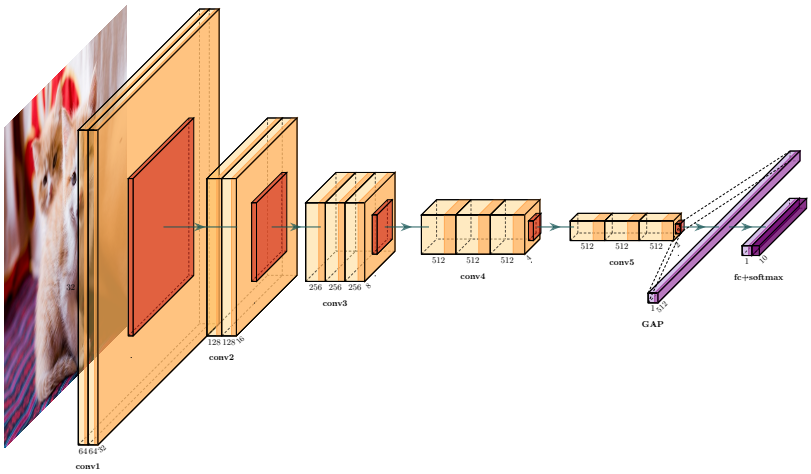

Fig. 1. Graphical schematization of **VGG16_BN**. Yellow blocks represent convolutional layers, red blocks represent max-pooling layers, and teal blocks represent fully-connected layers.

– we drop the two fully-connected layers, thus rendering the CNN fully convolutional [22]. After the last max-pooling layer, Global Average Pooling (GAP) is applied to flatten the output of said layer and feed it to the output layer.

We will refer to this network architecture as *VGG16_BN* throughout this work: a schematic representation of VGG16_BN is present in Figure 1.

VGG16_BN was trained for 160 epochs on CIFAR-10 using the optimizer Stochastic Gradient Descent with Momentum [27] (SGDM) and employing a step decay LR annealing schedule. The hyperparameters are as of [29], with a batch size of 128 since we trained the models on a single GPU.

VGG for SVHN In order to train a CNN on SVHN, we employed a custom VGG composed of 4 blocks, each one having two convolutional layers, except for the last one. In the first block each convolutional layer has 32 filters. The number of filters double in each convolutional block, so that the convolutional layer in the fourth block has 256 filters. After the last block, the output is pooled with GAP and fed to the output layer, thus rendering also this CNN fully-convolutional. In this work, we will refer to this architecture as *VGG_SVHN*: a schematic representation of VGG_SVHN is shown in Figure 2.

We trained the network for 15 epochs with a batch size of 50, we used the optimizer SGDM with hyperparameters as for VGG16_BN, and a step decay LR schedule, annealing by a factor of 10 at epochs 7 and 12.

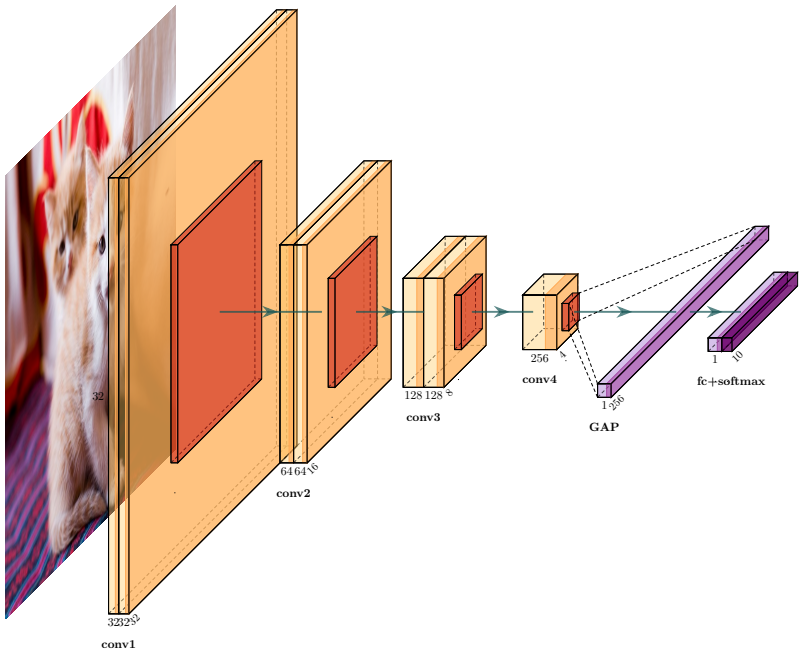

Fig. 2. Graphical schematization of **VGG16_SVHN**. Yellow blocks represent convolutional layers, red blocks represent max-pooling layers, and teal blocks represent fully-connected layers.

ResNet ResNet [17] is a family of CNN architectures based on a cascade of convolutional layers and employing *skip connections* which bypass one or more convolutional layers. Skip connections propagate the received information content directly to the end of the skipped layers. The output of the skipped layers is then summed to the content propagated by the skip connections: for this reason, the skipped layers can be also considered as belonging to a *residual block*, while the corresponding skip connection may also be called *identity shortcut*. The skipped connections allow for the construction of very deep DNNs, tackling the vanishing gradient problem.

We trained a variation of a ResNet on CIFAR-10. The architecture is inspired by a publicly available GitHub implementation (https://github.com/davidcpage/CIFAR-10-fast); a scheme is reported in Figure 3.

The main difference between this architecture and the classic ResNet is that the second convolutional block (conv2 in Figure 3) has no subsequent residual block. According to the author of this architecture, this allows for a better optimization on CIFAR-10; generally, the convergence of this CNN is very fast and

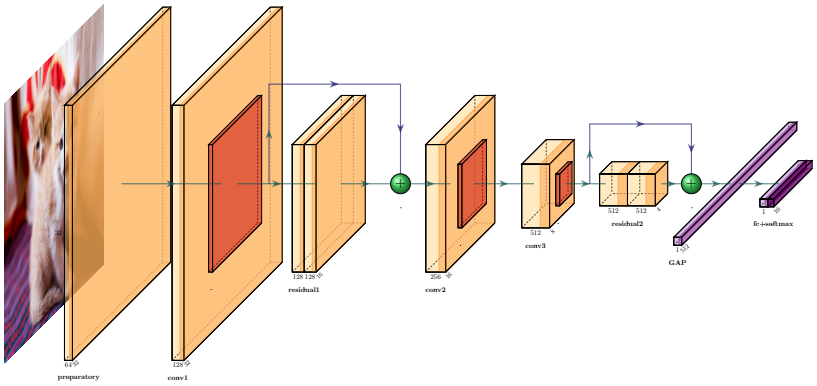

Fig. 3. Graphical schematization of **ResNet_fast**. Yellow blocks represent convolutional layers, red blocks represent max-pooling layers, and teal blocks represent fully-connected layers.

it allows for a full training in less than 10 minutes on a single GPU. Due to this reason, we will refer to this architecture as ***ResNet_fast***.

We trained this CNN for 24 epochs and batch size of 512 using SGDM with LARS updates [36], as in the original implementation of this network.

5 Results

5.1 Test-set accuracy

The performances in terms of test-set accuracy are shown in Figure 4.

We can note how the two models based upon CIFAR-10 (VGG16_BN and ResNet_fast) behave similarly: the first iterations of IMP produce CNNs which outperform the complete model; at around 10 % of parameters remaining, the accuracy spikes, then it slowly decreases, while remaining comparable w.r.t. the unpruned counterpart. This behaviour is expected from a DNN pruned with IMP, and has been known since [12].

Focusing instead on the CNN trained on SVHN (VGG_SVHN), we note a different behaviour: somewhat unexpectedly, the test-set accuracy of all the pruned networks tends to decrease as IMP is performed. This may be an indication that the unpruned structure of the network (VGG_SVHN) is adequate w.r.t. the difficulty of the problem (SVHN) and pruning even a small percentage of the connections is detrimental to its generalization capability. It may also be that the hyperparameters of the optimizer are adequate to train the complete network only, and may be hence tuned during the following IMP iterations to improve accuracy. We are not going to delve into this analysis as we believe it is not relevant to the foundations of this work. Anyway, this trend in test-set accuracy

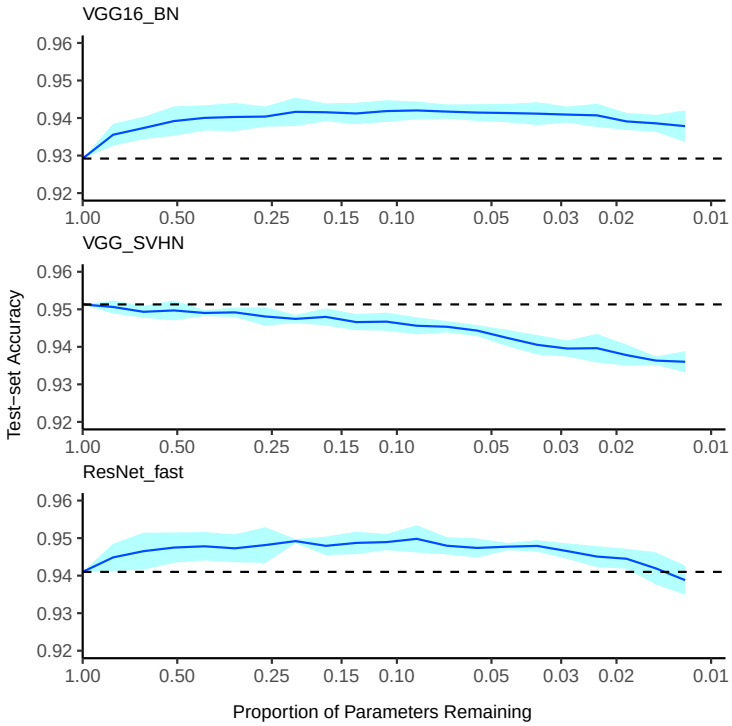

Fig. 4. Test-set accuracy averaged over multiple runs for our three models. Error bands correspond to 2 standard deviations. The dotted lines represent the reference accuracy of the complete model (corresponding to proportion of parameters equals to 1.00). The x-axis is in logarithmic scale.

on SVHN was noticed also in other works, such as [34], albeit on different CNN architecture.

5.2 Layer-wise pruned vs. unpruned similarity

As stated in Section 4, we compared the layers of the pruned CNNs with their unpruned counterparts. Specifically, the comparison was operated on representations of:

- convolutional layers (in short, *conv*; *conv_res* if the layer is part of a residual block of a ResNet), *after* the application of BN and activation function (ReLU);
- max-pooling layers (in short, *pool*);
- for ResNet only, addition nodes (in short, *add*), i.e., the nodes where the residuals are summed to the output of the skipped connections;
- output layers (in short, *out*), *before* the application of the activation function (softmax).

VGG16_BN The values of the metrics for VGG16_BN are shown in Figure 5. We can make the following observations.

Mean SVCCA Similarity and PWCCA show a very specific "U" shape which we also noted in [3][9] with simpler CNN architectures, while the problem was still CIFAR-10. The curves still bottom out around the third convolutional block, with a minimum slightly greater than 0.3. In addition to that, we can tell that the similarities decrease as the IMP iteration is increased, a phenomenon which become slightly less obvious as we near the output layer. Finally, in all iterations, the representations of the output layer have a similarity larger than 0.9.

CKA tells a different story: overall, the similarity is still high for all layers; while in SVCCA and PWCCA we had a similarity index ranging from 0.3 to around 0.95, in CKA we never record a value lower than 0.75. Moreover, the general trend we can observe is that the similarity seems to decrease as we progress from the first layers toward the output layer. Within the first two convolutional blocks, the similarity is rather packed between different iterations, with slightly smaller values as the IMP iteration is increased; after that, the values fan out, and the CNNs with higher sparsity exhibit a smaller similarity w.r.t. the unpruned counterparts. It is also of interest to note that the output layers seem to be not as similar as indicated in SVCCA/PWCCA; moreover, the values seem to be largely more scattered, ranging from around 0.77 to 0.95. Furthermore, starting with the 8-th IMP iteration (around 16 % of parameters remaining in the model), they stack together CKA-wise, exhibiting a similarity of around 79 %.

The values produced by NBS seem to blend characteristics of both CKA and SVCCA/PWCCA: the values range from 0.7 to 1.0, similarly to what we observed in CKA, and there is a "fan-out" effect at the third convolutional block, akin to CKA. On the other hand, we may notice that there does not seem to be the overall decreasing trend produced by CKA; instead, there is a clear bottom, identifiable between the 3-rd pooling layer and the subsequent convolutional layers, after which the similarities increase once again. This parallels what happens with SVCCA and PWCCA, although, in that case, the similarities were much lower, closer to 0.35, while with NBS they rest between 0.7 and 0.85. Finally the output layers exhibit a higher NBS than they do with CKA; on the other hand, there still seems to be a convergence at around 0.9 for the sparser CNNs.

VGG_SVHN Figure 6 shows the results for VGG_SVHN. W.r.t. VGG16_BN, we see more consistency between the four metrics.

Mean SVCCA Similarity and PWCCA do not exhibit a minimum with a clear subsequent recovery; instead, in both cases there seems to be two bottoms in the 3-rd and the 4-th convolutional blocks, after which the similarity spikes at the output; despite this spike, there seems not to be a recovery in the similarity corresponding to subsequent hidden layers, as instead we noticed with VGG16_BN.

[9] In that work, we employed Mean SVCCA Similarity only, but the shape produced by PWCCA is very similar.

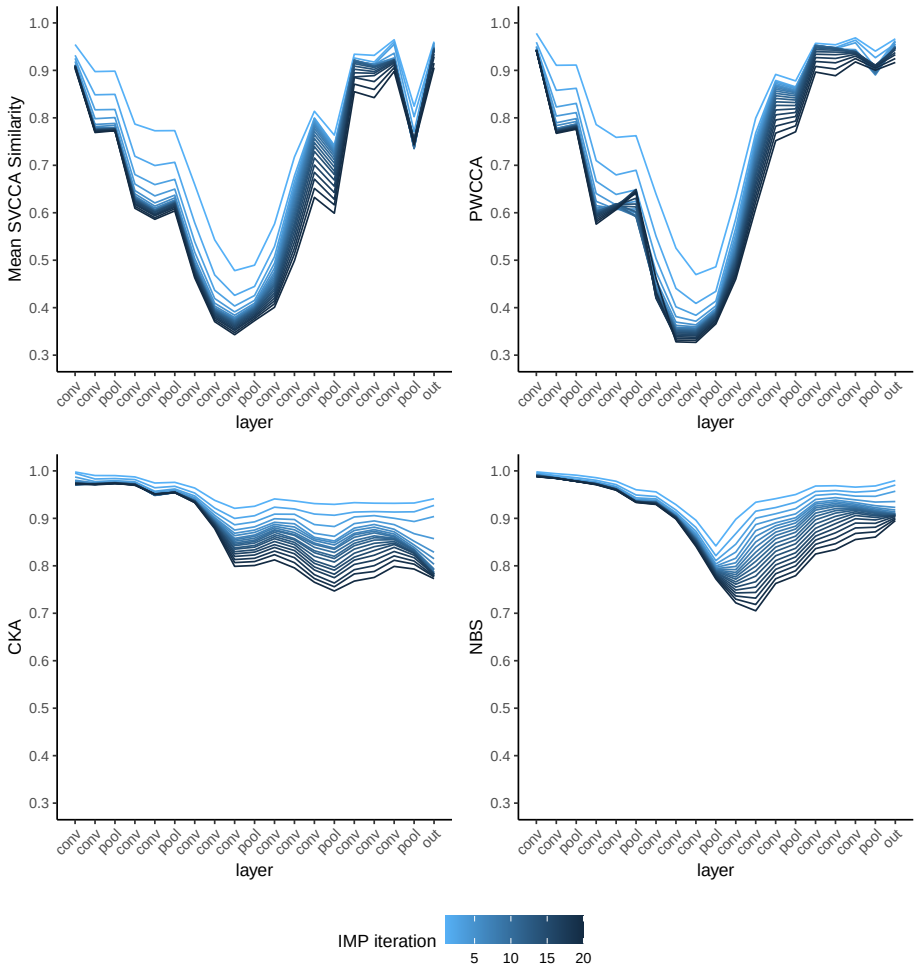

Fig. 5. Average layer-wise similarity between layers of pruned **VGG16_BN** vs. unpruned counterpart for various metrics (indicated on the y-axis). The values are averages over 20 runs of IMP. Error bands omitted for chart readability. Line color (light to dark) identifies the iteration of IMP.

CKA shows a similar landscape w.r.t. VGG16_BN, with an overall decreasing trend leading to the output layer; moreover, there is a noticeable bowl for the last 3 or 4 iterations of IMP in correspondence with the 3-rd pooling layer. Again, we can note somewhat of a large range of values at the output layer, with CKA values ranging from 0.78 to 0.95.

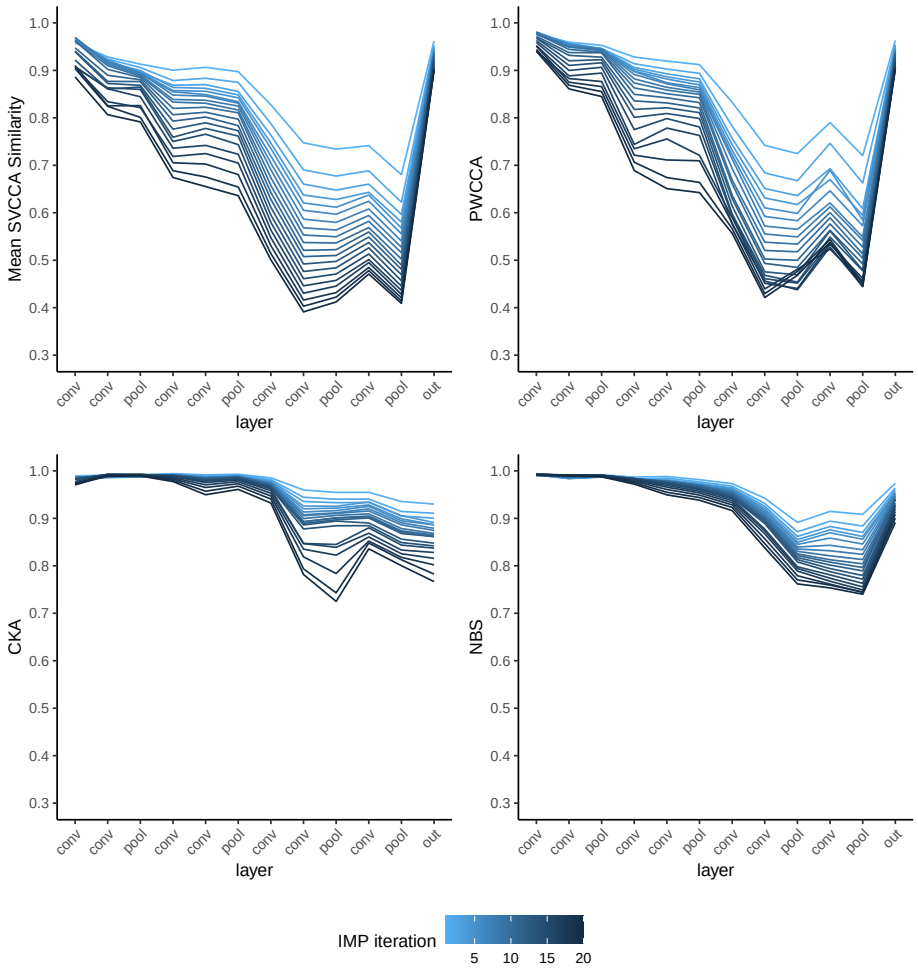

Fig. 6. Average layer-wise similarity between layers of pruned **VGG_SVHN** vs. unpruned counterpart for various metrics (indicated on the y-axis). The values are averages over 5 runs of IMP. Error bands omitted for chart readability. Line color (light to dark) identifies the iteration of IMP.

Finally, the figure traced by NBS is very similar to CKA, barring the "bowl" at the 3-rd convolutional block we saw in CKA: hence we might say that NBS seems a more regular CKA counterpart.

5.3 ResNet_fast

The similarity values for ResNet_fast are shown in Figure 7

CKA and PWCCA show a trend which, at first sight, may recall the one saw in VGG16_BN. It must be noted, though, that, excluding the output layer, the recovery in similarity is very weak after the minimum found at the third convolutional layer. Moreover, the similarity is slightly higher w.r.t. VGG16_BN: for VGG16_BN, the bottom reached as low as 0.35 for SVCCA and 0.32 for PWCCA, while for ResNet_fast the values are always above 0.40, and even above 0.45 for PWCCA. In addition to this, there seems to be a clear convergence of the similarity values as the IMP iteration (and, hence, the sparsity of the CNN) increases, a phenomenon that did not seem as evident in the other two networks.

For CKA and NBS, again the similarity values are all very high, with CKA hardly touching the 0.90 mark, while NBS reaches as low as 0.82. Generally, CKA seems to exhibit no trend at all, with values remaining about constant. NBS, on the other hand, shows a more distinct trend, with a minimum located at the second residual block, with a slight recovery as we head toward the output layer.

6 Discussion

6.1 Takeaways from results

The results presented in Section 5.2 show that the four similarity metrics we used to compare representations in CNNs lead to different outcomes. While CCA-based metrics reveal a "U"-shaped trend, CKA gives constant or slightly decreasing similarities; finally, NBS outputs a "U"-shaped similarity profile with some peculiar traits: (a) the minimum is reached 2–3 layers "later" w.r.t CCA-based metrics and (b) the similarity is much higher w.r.t. SVCCA or PWCCA.

In [3] we conjectured that the trend observed with Mean SVCCA Similarity could be a consequence of the Intrinsic Dimensionality (ID) of representations [2]: since in intermediate layers the ID is typically at its highest (often much higher than the ID of the input), there might be the possibility that in intermediate layers there is more variability and therefore more opportunities for dissimilarities that could be captured by such measures.

In the light of these new results, however, this conjecture seems problematic since (a) CKA suggests a very different picture than SVCCA or PWCCA, and (b) NBS, despite in qualitative agreement with SVCCA and PWCCA, places the minimum further in the progression of the layers, where the ID is typically far from its maximum [2].

In [20,32] it is argued that CKA is to be preferred to other CCA-based measures, since the latter pose too stringent requirements (invariance to invertible linear transformations) that can lead to paradoxical results.

The similarity results obtained with CKA seem to suggest the following:

a) representations in pruned and unpruned networks are remarkably similar;

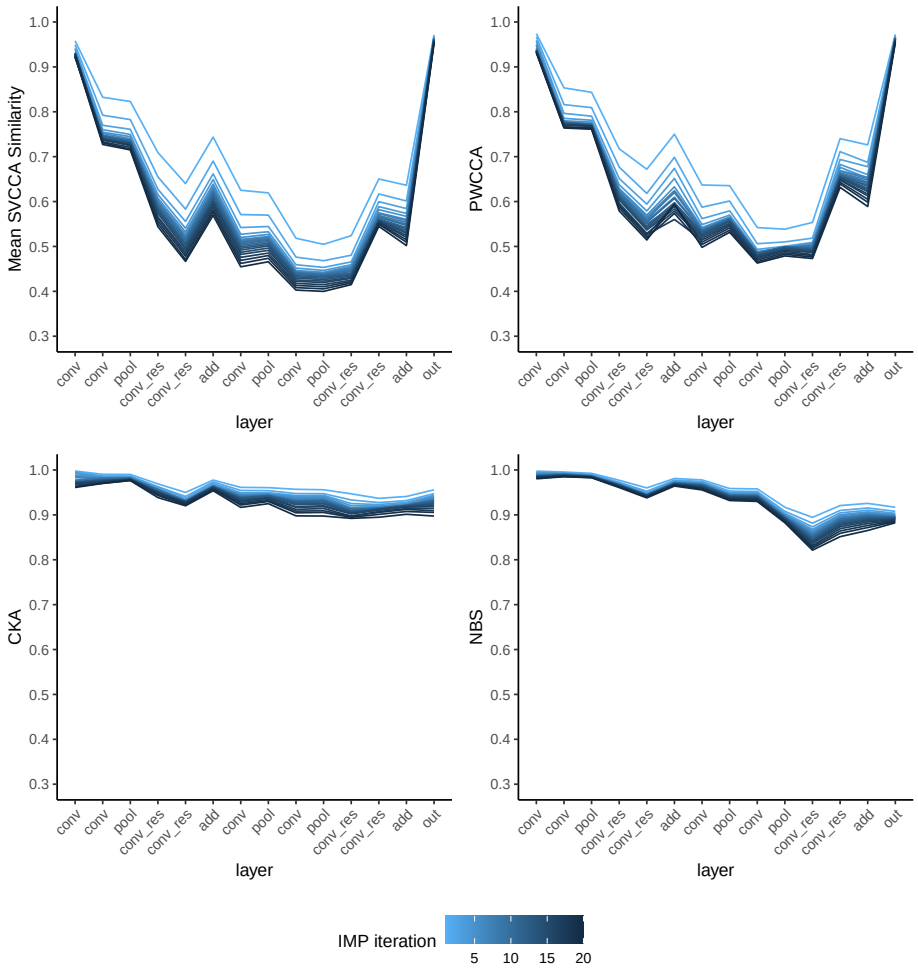

Fig. 7. Average layer-wise similarity between layers of pruned **ResNet_fast** vs. unpruned counterpart for various metrics (indicated on the y-axis). The values are averages over 5 runs of IMP. Error bands omitted for chart readability. Line color (light to dark) identifies the iteration of IMP.

b) there is a weak but progressive decrease of similarity (i) as we increase the sparsity in the CNN, and (ii) as we transverse the network in the forward direction; however, this trend seems not to be very pronounced, and there are typically plateaus in the CKA similarity.

We did not compare explicitly the features learnt by the CNNs, although an analysis may be possible considering, for instance (i) parameters-based similarity measures instead of representation-based, or (ii) quantitative metrics comparing features learnt by the DNNs, like Net2Vec [10].

6.2 Comparing output layers

Conversely to hidden layers, it could be argued that a similarity metric to compare output layers does not need invariances to orthogonal transformation, since the role of the neurons within the output layers are fixed: in a DNN for classification, neuron i of the output layer is proportional to the probability of assignment of a given data point to the i-th class.

The four metrics we used up to now have all been indicated as fit to compare DNNs for a given amount of reasons: invariances to specified transformations, measuring statistical correlations between random variables in different dimensions, *etc.* As an addition to our analysis, we experiment with comparing the output layers with a slight variation of a classical similarity metric for vectors: the cosine similarity. Given vectors $a, b \in \mathbb{R}^p$, the cosine similarity is defined as

$$\text{csim}(a, b) = \frac{a^T b}{\|a\|_2 \|b\|_2} \tag{8}$$

This similarity metric is used to compare generic vectors belonging to the same space. An output layer of c neurons, though, is represented a generic matrix $A \in \mathbb{R}^{n \times c}$ where n is the number of data points employed to obtain the representation. In order to enable the comparison between two output layers A, B via cosine similarity, we simply consider the vectorized representations $\text{vec}(A) = [A_{11}, \ldots, A_{1c}, \ldots, A_{n1}, \ldots, A_{nc}]$ and feed it into Equation (8). Finally, since $\text{csim}(\cdot, \cdot) \in [-1, 1]$, in order to normalize its support w.r.t. the other four metrics used, we consider its magnitude:

$$|\text{csim}(\text{vec}(A), \text{vec}(B))| \in [0, 1]$$

This adjustment reflects already what happens, for instance, in CKA, which, to put it simply, measures the (in)dependence between two sets of variables [20], but does not distinguish whether the possible dependence is positive or negative, being comprised between 0 and 1.

Settings and results We evaluated the network VGG16_BN with the same subset of CIFAR-10 of 5000 data points (see Section 4), keeping only the representations of the output layer. The evaluation was repeated for the complete network and all of the pruned networks (20 runs $\times$20 IMP iterations).

The results of this similarity compared with the previous four are shown in Figure 8. We note that:

a) the vectorized cosine similarity does not exhibit the "stacking" effect we note on the last iterations of IMP; conversely, the values seem rather well distributed along the range $[0.80, 0.97]$;

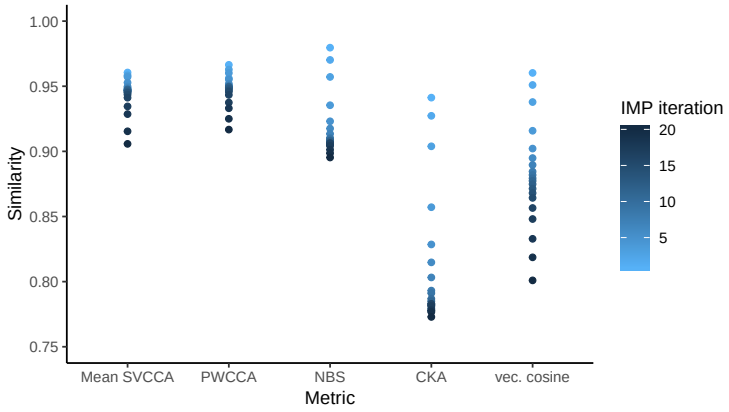

Fig. 8. Average similarity of pruned output layers w.r.t. the unpruned counterparts for the network **VGG16_BN**, for various iterations of IMP and different similarity metrics. Values are averaged over 20 different runs of IMP starting from the same complete network.

b) unlike SVCCA, PWCCA, and NBS, which all record similarities above 0.9, the cosine similarity starts high at the first iterations, before sinking to around 0.8. Under this facet, it seems more akin to CKA, even if the distribution w.r.t. the iteration of IMP is different. Note also that the cosine similarity and CKA are somewhat related as CKA is based upon the Hilbert-Schmidt Independence Criterion, which is itself based upon the cosine similarity between features of a representation [20].

The fact that we notice some differences between output layers may indicate how, despite producing DNNs with comparable test-set accuracy, the probabilities of assignment of the 5000 data points to the various classes in the pruned models may disagree w.r.t. the unpruned model. This may be attributable, for example, to a difference in calibrations, as noted in [34].

6.3 Considerations on the rotational invariance of similarity metrics for convolutional layers

To wrap up the discussions, we believe it is of interest to make a brief digression whether the rotational invariance is really a desirable property that a generic similarity metric must possess. It is undeniable that this property be enjoyed for comparing, for instance, fully-connected layers: if a network is composed only of such layers, it may be possible to carefully permute all of the neurons in its hidden layers to obtain a new DNN producing the exact same output; in this case, we wish that a similarity metric indicates that the layers of the old and the new network are pairwise identical.

This, instead, is not the case for CNNs, because the parameters of convolutional layers possess a spatial constraint w.r.t. the image they see: neurons inside

a channel of a convolutional layer recognize specific features that they would not be able to distinguish as accurately if "moved around" in other positions within the same channel. The same does not hold, instead, when we consider full channels: a careful permutation of the channels in a CNN, along with the neurons in the possible fully-connected layers, can result in a CNN producing the same output.

Hence, we argue that, for the case of convolutional layers (and pooling layers as well, as they share the same spatial structure), a similarity metric should enjoy a rotational invariance only at the level of channels of convolutional layers. In Section 3.2, when talking about the handling of convolutional layers by SVCCA and PWCCA, we reported that, to compute similarities for convolutional layers, these two metrics require a preprocessing step for the layers such that the spatial dimensions are *merged* into the data points dimension, leaving as *proper neurons* in the representation the sole dimension regarding channels. This strategy may also be, in principle, applicable to CKA and NBS and may be analyzed in a future work.

7 Conclusions and future work

In this work, we compared representations in pruned and unpruned networks, trained for Computer Vision task, extending in several directions the range of our earlier work [3], by considering a wider spectrum of similarity measures, new models and datasets.

Specifically, we analyzed layer-wise representation similarities across several CNN architectures trained on CIFAR-10 and SVHN. We considered a representative set of available similarity measures, namely: Mean SVCCA Similarity, PWCCA, CKA, and NBS. These metrics produce, in our settings, contrasting results which do not allow to draw univocal conclusions on the impact of pruning on representations.

While SVCCA and PWCCA yield similar results, suggesting that in pruned network intermediate layers create very dissimilar representations with respect to the complete network (see Figures 5 to 7, panels on the first row), CKA shows a weak, almost monotonic decrease across the network (see Figures 5 to 7, left panel on the second row). NBS, which in principle is similar to CKA, and shares its desirable invariances, is in qualitative agreement with the trends exhibited by SVCCA and PWCCA, on a much smaller scale. All the similarity measures we investigated are in agreement in indicating a progressive departure—as more and more connections are removed during the iterations of IMP—from the original representations. In the case of CKA anyway, this departure is contained (similarity never falling below 0.75 in all our experiments).

We proposed two observations on the rotational-invariance of metrics for comparing output layers and convolutional layers, respectively, hinting at directions for future works (see Section 6.3). Very recently, a modification of existing similarity measures was proposed, integrating the information coming from the gradient [32].

We identified several directions for further research on comparing representations in pruned and unpruned networks:

- exploration of new similarity measures that integrate CKA with the information coming from gradients [32];
- modifications of existing metrics, e.g., enforcing channels-only rotational-invariance in convolutional layers;
- extensions to large-scale datasets, like ImageNet [9];
- exploration of other pruning strategies, like structured pruning or Accelerated IMP [39], to examine how representations obtained in this way differ w.r.t. IMP with WR.

A detailed understanding of representations in pruned networks is of primary importance for theoretical and practical reasons. We hope that this exploratory, empirical work will inspire new theoretical investigations on the nature of these representations, enabling a deeper understanding of the inner workings of deep neural networks, and enhancing the security of their applications.

References

1. Allen-Zhu, Z., Li, Y., Liang, Y.: Learning and generalization in overparameterized neural networks, going beyond two layers. In: Advances in neural information processing systems. pp. 6155–6166 (2019)
2. Ansuini, A., Laio, A., Macke, J.H., Zoccolan, D.: Intrinsic dimension of data representations in deep neural networks. In: NIPS 2019 (2019)
3. Ansuini, A., Medvet, E., Pellegrino, F.A., Zullich, M.: On the Similarity between Hidden Layers of Pruned and Unpruned Convolutional Neural Networks. In: De Marsico, M., Sanniti di Baja, G., Fred, A. (eds.) Proceedings of the 9th International Conference on Pattern Recognition Applications and Methods (ICPRAM 2020). pp. 52–59. Scitepress, La Valletta (feb 2020)
4. Anwar, S., Hwang, K., Sung, W.: Structured pruning of deep convolutional neural networks. ACM Journal on Emerging Technologies in Computing Systems (JETC) **13**(3), 1–18 (2017)
5. Bures, D.: An extension of kakutani's theorem on infinite product measures to the tensor product of semifinite w*-algebras. Transactions of the American Mathematical Society **135**, 199–212 (1969)
6. Cortes, C., Mohri, M., Rostamizadeh, A.: Algorithms for learning kernels based on centered alignment. Journal of Machine Learning Research **13**(Mar), 795–828 (2012)
7. Cristianini, N., Shawe-Taylor, J., Elisseeff, A., Kandola, J.S.: On kernel-target alignment. In: Advances in neural information processing systems. pp. 367–373 (2002)
8. Crowley, E.J., Turner, J., Storkey, A., O'Boyle, M.: Pruning neural networks: is it time to nip it in the bud? arXiv preprint arXiv:1810.04622 (2018)
9. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: ImageNet: A Large-Scale Hierarchical Image Database. In: CVPR09 (2009)
10. Fong, R., Vedaldi, A.: Net2vec: Quantifying and explaining how concepts are encoded by filters in deep neural networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 8730–8738 (2018)

11. Frankle, J., Bau, D.: Dissecting pruned neural networks. arXiv preprint arXiv:1907.00262 (2019)
12. Frankle, J., Carbin, M.: The lottery ticket hypothesis: Finding sparse, trainable neural networks. In: International Conference on Learning Representations (2019), https://openreview.net/forum?id=rJl-b3RcF7
13. Frankle, J., Dziugaite, G.K., Roy, D.M., Carbin, M.: Stabilizing the lottery ticket hypothesis. arXiv preprint arXiv:1903.01611 (2019)
14. Google: (sv)cca for representational insights in deep neural networks (2019), https://github.com/google/svcca
15. Gretton, A., Bousquet, O., Smola, A., Schölkopf, B.: Measuring statistical dependence with hilbert-schmidt norms. In: International conference on algorithmic learning theory. pp. 63–77. Springer (2005)
16. Han, S., Pool, J., Tran, J., Dally, W.: Learning both weights and connections for efficient neural network. In: Cortes, C., Lawrence, N.D., Lee, D.D., Sugiyama, M., Garnett, R. (eds.) Advances in Neural Information Processing Systems 28, pp. 1135–1143. Curran Associates, Inc. (2015)
17. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016)
18. Hotelling, H.: Relations between Two Sets of Variates. Biometrika **28**(3-4), 321–377 (12 1936). https://doi.org/10.1093/biomet/28.3-4.321, https://doi.org/10.1093/biomet/28.3-4.321
19. Ioffe, S., Szegedy, C.: Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: ICML. pp. 448–456 (2015), http://proceedings.mlr.press/v37/ioffe15.html
20. Kornblith, S., Norouzi, M., Lee, H., Hinton, G.: Similarity of neural network representations revisited. In: Chaudhuri, K., Salakhutdinov, R. (eds.) Proceedings of the 36th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 97, pp. 3519–3529. PMLR, Long Beach, California, USA (09–15 Jun 2019), http://proceedings.mlr.press/v97/kornblith19a.html
21. Krizhevsky, A., Hinton, G., et al.: Learning multiple layers of features from tiny images (2009)
22. Long, J., Shelhamer, E., Darrell, T.: Fully convolutional networks for semantic segmentation. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 3431–3440 (2015)
23. Morcos, A., Raghu, M., Bengio, S.: Insights on representational similarity in neural networks with canonical correlation. In: Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (eds.) Advances in Neural Information Processing Systems 31, pp. 5732–5741. Curran Associates, Inc. (2018)
24. Morcos, A., Yu, H., Paganini, M., Tian, Y.: One ticket to win them all: generalizing lottery ticket initializations across datasets and optimizers. In: Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché Buc, F., Fox, E., Garnett, R. (eds.) Advances in Neural Information Processing Systems 32, pp. 4932–4942. Curran Associates, Inc. (2019)
25. Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., Ng, A.Y.: Reading digits in natural images with unsupervised feature learning (2011)
26. Paganini, M., Forde, J.: On iterative neural network pruning, reinitialization, and the similarity of masks. arXiv preprint arXiv:2001.05050 (2020)
27. Qian, N.: On the momentum term in gradient descent learning algorithms. Neural networks **12**(1), 145–151 (1999)

28. Raghu, M., Gilmer, J., Yosinski, J., Sohl-Dickstein, J.: Svcca: Singular vector canonical correlation analysis for deep learning dynamics and interpretability. In: Guyon, I., Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R. (eds.) Advances in Neural Information Processing Systems 30, pp. 6076–6085. Curran Associates, Inc. (2017)
29. Renda, A., Frankle, J., Carbin, M.: Comparing fine-tuning and rewinding in neural network pruning. In: International Conference on Learning Representations (2020)
30. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (09 2014)
31. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. The journal of machine learning research **15**(1), 1929–1958 (2014)
32. Tang, S., Maddox, W.J., Dickens, C., Diethe, T., Damianou, A.: Similarity of neural networks with gradients. arXiv preprint arXiv:2003.11498 (2020)
33. Uurtio, V., Monteiro, J.a.M., Kandola, J., Shawe-Taylor, J., Fernandez-Reyes, D., Rousu, J.: A tutorial on canonical correlation methods. ACM Comput. Surv. **50**(6) (Nov 2017). https://doi.org/10.1145/3136624, https://doi.org/10.1145/3136624
34. Venkatesh, B., Thiagarajan, J.J., Thopalli, K., Sattigeri, P.: Calibrate and prune: Improving reliability of lottery tickets through prediction calibration. arXiv preprint arXiv:2002.03875 (2020)
35. Ye, S., Xu, K., Liu, S., Cheng, H., Lambrechts, J.H., Zhang, H., Zhou, A., Ma, K., Wang, Y., Lin, X.: Adversarial robustness vs. model compression, or both. In: The IEEE International Conference on Computer Vision (ICCV). vol. 2 (2019)
36. You, Y., Gitman, I., Ginsburg, B.: Large batch training of convolutional networks. arXiv preprint arXiv:1708.03888 (2017)
37. Zhang, C., Bengio, S., Hardt, M., Recht, B., Vinyals, O.: Understanding deep learning requires rethinking generalization. arXiv preprint arXiv:1611.03530 (2016)
38. Zhou, H., Lan, J., Liu, R., Yosinski, J.: Deconstructing lottery tickets: Zeros, signs, and the supermask. In: Advances in Neural Information Processing Systems. pp. 3597–3607 (2019)
39. Zullich, M., Medvet, E., Pellegrino, F.A., Ansuini, A.: Speeding-up pruning for artificial neural networks: Introducing accelerated iterative magnitude pruning. In: Proceedings of the 25th International Conference on Pattern Recognition (2021)