Optimal Convolutional Architectures Via Structured Sparsity Regularizers

Anonymous CVPR submission

Paper ID ****

Abstract

The architectures of Convolutional Neural Network (CNN) are determined by a large number of hyper-parameters including (i) the number of layers (or architecture depth), (ii) the number of convolutional kernels in each layer (or layer width), and (iii) the spatial size of each convolutional kernel (or kernel size). Nearly all of the recent advances in CNN methods have been a consequence of a worldwide heuristic search for the right mix of depth, width and size, with extreme examples producing architectures of more than 1000 *layers, widths of* 2048 *kernels, and kernel sizes as small as* 1×1*. The large number of possible architectures has further made it necessary to constrain the search to layers with kernels of the same size, and widths that vary in a regular manner across the depth of the architecture.*

In this work we show how structured sparsity techniques can be used to build a convex regularizer that independently chooses the size of each kernel in each layer. Increasing the weight of this penalty can force kernels (and eventually layers PP: Only for ResNet JZ: We'll know hopefully tomorrow...) to dissapear independently, hence providing an elegant solution to simultaneous selection of architecture depth, layer widths and kernel sizes. We further show that, by varying the penalty weight, one can obtain an efficient frontier PP: Not clear, what is "efficient" JZ: I'll remove efficient... of architectures each displaying an optimal tradeoff between performance and structural sparsity. As we discuss, the structural sparsity regularizer can be interpreted as a measure of complexity, in effect casting the problem of architecture selection as a tradeoff between complexity and performance where complexity is embodied by a regularizer perfectly compatible with stochastic gradient descent solvers.

1. Introduction

depth, kernel size, layer width

Notation: We denote scalars using standard typeface (*e.g.* scalar a), and we denote vectors, matrices and three-dimensional tensors using bold typeface (*e.g.* a tensor $\boldsymbol{M} \in \mathbb{R}^{a \times b \times c}$). Given a vector, tensor or matrix $\boldsymbol{M}$, we define its ℓ_p norm for all $p > 0$ as $|\boldsymbol{M}|_p \triangleq \left(\sum_{m \in \boldsymbol{M}} |m|^p\right)^{\frac{1}{p}}$; we also use $|\boldsymbol{M}|_0$ to denote the ℓ_0 pseudo-norm which produces the number of non-zero elements in $\boldsymbol{M}$ (equal to $|\boldsymbol{M}|_p$ as $p \to 0$).[1]

2. Background

2.1. Recent CNN architectures

While Convolutional Neural Networks have existed for several decades, it was the 2012 ImageNet submission of Krizhevsky *et al.* [8] that shifted vast amounts of academic and industrial research effort towards deep learning. Krizhevsky's architecture consists of several main operations and layer types: spatial convolutions, spatial (per-channel) max-pooling, fully-connected layers, activation by means of Rectified Linear Units (ReLUs), and local response normalization. The final layer of the network is a classifier taking the form a fully-connected layer with softmax activations. The network consisted of 60 million parameters and was trained using dropout regularization. Measured in terms of the number of layers with trainable parameters (*i.e.* the convolutional or fully connected *weight layers*), this network consisted of 8 layers.

Following the success of Krizhevsky's architecture, Simonyan and Zisserman proposed increasing the depth of Krizhevsky's architecture to 11, 13, 16 and 19 weight layers by increasing the number of convolutional layers. They also dispensed of local response normalization, which was found to increase the error rate of the 11-layer network. A variant of their proposed 16-layer network employed size-1 convolutional kernels, which were used together with size-3 kernels in a common layer. Using kernels of multiple sizes in the same layer relates to the work we present herein – we learn the optimal size of each kernel in all layers independently by means of a kernel-size regularizer.

[1] Strictly speaking, $|\boldsymbol{M}|_p$ is a norm only for $p \geqslant 1$.

	Top-1 err.	Top-5 err.	D	Layer widths	Kernel sizes	Num. params	Mult.-adds
AlexNet[8]	– / 40.7%	– / 18.2%	7	$96, 256, 384$	$3, 5, 11$	60M	??
VGG-16 [13]	27.0% / –	8.8% / –	16	$64, 128, 256, 512$	3	138M	15.3B
GoogLeNet[15]	– / –	10.07% / 9.15%	22	$16, 24, 32, 48, 64, 96, 112, 128, 160, 192, 256, 288, 320, 384$	$1, 3, 5$	6.7M	1.5B
ResNet-152 [4]	– / 19.38%	– / 4.49%	152	$64, 128, 256, 512, 1024, 2048$	$1, 3$	??	11.3B

Table 1. Recent architectures in terms of depth D (excluding the soft-max layer), top-1 and top-5 error rates (1-crop / 10-crop) testing, and number of learned parameters.JZ: Need to complete and/or prune this table

The GoogLeNet architecture [15] – consisting of 21 convolutional layers and a single fully-connected layer – was a bigger change from the original architecture of [8]. The motivation behind the model is to obtain weights that are sparse, while at the same time limiting the feed-forward computational complexity of the network. The authors implemented the desired sparsity by means of a Network-in-Network (NiN) mechanism [10] that can be seen as employing non-linear convolutional kernels that, similarly to the models of Simonyan and Zisserman, used layers with multi-sized kernels. Their multi-sized kernels also included 1×1 kernels, but designed as dimensionality reducing operators that keep the computational complexity of the network in check. Two other interesting characteristics of the GoogLeNet network are *(i)* their use of untrained average pooling for dimenstionality reduction and *(ii)* their use of bypass connections that concatenate some feature maps to the output feature maps of the next layer instead of the current layer.

An approach similar to bypass connections – *shortcut* connections – is employed by the more recent ResNet model [4]. Shortcut connections are applied across two contiguous convolutional layers (with ReLU activations) to form a *residual unit*. Instead of concatenating feature maps, the shortcut connections add the feature maps at the input of the two-layer unit to the output feature maps. This approach was motivated by empirical evidence indicating that learning algorithms became too slow when the number of layers increased. Indeed, the experiments in [4] established that overfitting was not at fault for receding performance gains when the network depth became too large. Extensions of this work explored increasing the depth to as much as 1000 layers [5], or increasing the width of the layers to up to 640 convolutional kernels [16]. The ResNet network is fully-convolutional, relying also on untrained average pooling layers as a dimensionality reduction mechanism.

In Table 1 we list various characteristics of the main architectures discussed above. Note that the depth, widths and kernel sizes has varied greatly across the various network incarnations. Yet discussions concerning the selection of these parameters is lacking in the literature as most of these architectures have been found by trial and error.

More formal architecture selection methods exist in the literature, and we next provide an overview of several of these.

2.2. Architecture learning methods

PP: Note: A number of papers have explored greedy or ad-hoc ways to prune the set of convolutional filters on the fly, *e.g.* trying to reduce the redundancy among filters, with the main goal to promote compact networks rather than really exploring different architectures.

PP: Note: In [**?**], a specific CNN architecture is proposed, whose width can be jointly learned with a simple regularizer (simply penalizing the width!) that is claimed to derive from Indian Buffet Process (I though it was call the Chinese Buffet...). This approach is thus not amenable to other existing SoA CNNs. On ImageNet they use 5 of their convolutional leyers and 2 fully connected. Also the optimization is not embedded in the gradient descent, they must rely on a greedy mechanism to learn the width.

PP: Note: In [**?**], group sparsity regularization is used in MLP, not CNN, in order to prune individual neurons. The means are strongly related to what we propose here, but the goal is very different and relates to our BMVC approach[9] JZ: They only do fully-connected layer size selection using group sparsity not structured sparsity.

Several authors have also explored learning factorizations of the convolutional layers in order to reduce computational complexity. Liu *et al.* [11] propose representing the operations in a convolutional layer as a large matrix-matrix product, imposing sparsity and group sparsity constrains on one of the matrix factors while learning the factorization as a part of supervised fine-tuning stage. Jaderberg *et al.* [**?**] follow a related approach wherein the convolutional kernels are constrained to be low-rank tensors JZ: (Term correct? They're matrices or vectors but can be oriented depth-wise) with only one or two dimensions. Sequential application of such low rank tensors is less computationally demanding than full-rank tensors, yet results in comparable accuracy.

2.3. Structured sparsity

The group sparsity penalty terms employed by several of the above referenced works are used to simultaneously activate or deactivate (zero) groups of variables [1]. Given a vector of variables $\boldsymbol{s} = [s_i]_{i=1}^N$ and user selected groups $\mathcal{G}_j \subset \{1, \ldots, N\}$, the group sparsity penalty can be expressed as follows, where we let $\boldsymbol{s}(\mathcal{G}_j) \triangleq [s_i]_{i \in \mathcal{G}_n}$:

$$\sum_n w_n |\boldsymbol{s}(\mathcal{G}_n)|_2 \qquad (1)$$

When the $\mathcal{G}_n$ form a partition of the support $\{1, \ldots, N\}$ of $\boldsymbol{s}$, the above expression is called the ℓ_{21} norm. Otherwise, by judiciously selecting the groups (*e.g.* to be overlapping or nested), one can enforce structure in the sparsity patterns induced in $\boldsymbol{s}$. Examples of this include the work of Jenatton *et al.* on structured sparse principal component analysis [7] or on hierarchical dictionaries for sparse coding [6].

The work we introduce in this paper employs a nested structured sparsity regularizer to individually restrict the spatial sizes of convolutional kernels.

3. CNN Architecture Selection

In this section we first review a general deep learning formulation and discuss the choice of architecture – the depth, layer widths, and kernel sizes– and how this choice affects the resulting network complexity. We then propose using structured sparsity methods to choose the architecture as part of the learning process. We present two variants of our approach. The first one penalizes the total number of kernels in each layer, where each kernel is assumed to have a fixed size and each layer a per-layer-fixed width. By zeroing out all kernels in a layer, this approach can succeed in removing layers completely PP: Only with ResNet? , thus learning the architecture depth as a consequence. The second variant instead penalizes the size of each kernel independently and is accordingly able to remove complete kernels and eventually layers.

3.1. Learning convolutional architectures

We consider convolutional architectures with layers numbered $i = 1, \ldots, D$. Layer i takes as input a tensor $\boldsymbol{M}_{i-1}$ and produces a tensor $\boldsymbol{M}_i$, where

$$\boldsymbol{M}_i \in \mathbb{R}^{r_i \times c_i \times d_i}. \qquad (2)$$

Each convolutional layer consists of a set of d_i convolutional kernels

$$\mathcal{K}_i = \{\boldsymbol{k}_{ij} \in \mathbb{R}^{s_{ij} \times s_{ij} \times d_{i-1}}\}_{j=1}^{d_i}, \qquad (3)$$

where s_{ij} is the size of each kernel. We note that this notation subsumes fully connected layers, where the input tensor $\boldsymbol{M}_{i-1}$ is of dimensions $1 \times 1 \times d_{i-1}$ and the d_i kernels are of dimensions $1 \times 1 \times d_i$. The kernels are all applied by means of spatial convolution to the input tensor $\boldsymbol{M}_{i-1}$ and the output subsequently fed through an activation function $a_i(\cdot)$ to produce the layer output

$$f_i(\boldsymbol{M}_{i-1}; \mathcal{K}_i) = a_i(\{\boldsymbol{k}_{ij} * \boldsymbol{M}_{i-1}\}_j) \quad (\triangleq \boldsymbol{M}_i)\,. \qquad (4)$$

The layer-dependent activation functions $a_i(\cdot)$ can operate element-wise, such as the ReLU non-linearity, or across multiple elements, such as a spatial max-pooling operator, maxout operator [3], local response normalization operator, downsampling operator, and softmax. The $a_i(\cdot)$ can even be a combination of multiple non-linearities.

Acordingly, letting $\boldsymbol{J}$ be an input image and

$$\boldsymbol{\Theta} \triangleq \{\mathcal{K}_1, \ldots, \mathcal{K}_D\} \qquad (5)$$

contain the kernels for all layers, an entire CNN architecture can be written as

$$f(\boldsymbol{J}; \boldsymbol{\Theta}) \triangleq f_D \circ \ldots \circ f_1(\boldsymbol{J}; \boldsymbol{\Theta}). \qquad (6)$$

Given the above described network, the kernel parameters can be learned for a specific task by defining an adequate objective taking usually the form

$$\operatorname*{argmin}_{\boldsymbol{\Theta}} \frac{1}{N} \sum_{n=1}^{N} \ell(f(\boldsymbol{J}_n; \boldsymbol{\Theta}), y_n) + \alpha \Omega(\boldsymbol{\Theta}), \qquad (7)$$

where $\ell(\cdot)$ is a task-dependent loss function, $\Omega(\cdot)$ is a regularizer, and y_n is an annotation related to the n-th example $\boldsymbol{J}_n$. For example, for the image classification task with C classes, $y \in \{1, \ldots, C\}$ is the ground-truth class label for a given image, $\boldsymbol{x} \in \mathbb{R}^C$ is the probability estimate produced by a soft-max layer at the end of the network, and $\ell(\cdot)$ is usually the cross-entropy loss

$$\ell(\boldsymbol{x}, y) = -\log(\boldsymbol{x}[y]). \qquad (8)$$

A commonly used regularizer $\Omega(\cdot)$ that leads to *weight decay* learning steps [8] is the squared ℓ_2 norm:

$$\Omega(\boldsymbol{\Theta}) = \sum_{ij} |\boldsymbol{k}_{ij}|_2^2, \qquad (9)$$

where we define the ℓ_2 norm of $\boldsymbol{k}$ as the sum of squares of all its coefficients.

3.2. Heuristic selection of the architecture

Choosing a CNN architecture amounts to selecting both $\boldsymbol{\Theta}$ and the activations $a_i(\cdot)$ of each layer. Concerning the activation function, the element-wise Rectfied Linear Unit (ReLU) activation $\max(0, x)$ has proven to be a very good choice that leads to increased learning speed over alternatives such as the sigmoid or hyperbolic tangent functions [8, ?].

Indeed, most of the focus of recent research efforts has been on selecting the set of kernels $\boldsymbol{\Theta} = \{\mathcal{K}_i\}_i$. This choice implies selecting

- the depth D or number of layers $\mathcal{K}$,
- the number d_i of kernels in each layer $\mathcal{K}_i$,
- and the size s_{ij} of each kernel $\boldsymbol{k}_{ij}$.

We refer collectively to these parameters as the *architecture* of the network.

A vast research effort has been undertaken in the past several years that has drastically improved the state-of-the-art in several large scale datasets. Most of this effort has amounted to choosing a different architecture by means of a heuristic search. Indeed, the choice of architecture often has an underlying motivation, including a quest for sparsity together with low complexity [**?**], or an empirical observation of deficiencies in learning algorithms for very large depths [**?**].

Yet the specific choice of the parameters of the architecture is hardly discussed in the literature. One important reason for this is the very large number of possible architectures which, coupled together with the long CNN training times (days or even weeks), makes it impossible to cross-validate the architecture. This difficulty is addressed by restricting the architecture search space using, for example a fixed kernel size $s_{ij} = s_i$ for a given layer i, or by using layer sizes that are the same for groups of layers and change only a few times, further increasing monotonically with the depth across the network. While based on different insights, such restrictions on the architecture's search space are clearly a matter of convenience for heuristic purposes and are not optimal.

Indeed the choice of architecture has two important consequences that ideally need to be balanced against each other. The first is a consequence on the accuracy produced by the network: larger architectures have better accuracy, but only up to the point where generalization begins to suffer. The second consequence is on the networks computational complexity which is given in terms of feed-forward multiply-adds for a single image by

$$\sum_{i=1}^{D}\sum_{j=1}^{d_i} r_i \cdot c_i \cdot d_{i-1} \cdot s_{ij}^2. \tag{10}$$

Note, for illustrative purposes, that this reduces to $D \cdot r \cdot c \cdot d^2 \cdot s^2$ if we assume that all dimension values are the same for all layers and kernels. Hence the importance of choosing architectures that are very compact: the complexity grows as the square of the number of kernels and the square of the kernel sizes.

3.3. Structured sparsity regularization

Given that complexity has to be balanced against the accuracy of the network suggests penalizing the network by using (10) as the regularizer $\Omega(\boldsymbol{\Theta})$. Yet the problem here is that (10) cannot be differentiated against $\boldsymbol{\Theta}$ – the sub-gradients of (10) are zero almost everywhere. Hence in this work we propose two alternate regularization schemes that penalize d_i and s_{ij} in (10) and have nonethelesss non-zero sub-gradients that make them immediately compatible with standard stochastic gradient descent solvers.

Kernel-number-reducing regularizer. The first regularization scheme that we propose aims to penalize the number of kernels d_i in each layer. For ease of implementation, we assume that each layer i is constrained to have at most $\hat{d}_i$ kernels, and refer to the architecture having exactly $\hat{d}_i$ kernels in layer i as the *base architecture*.

Noting that

$$d_i = \left| \left[\, |\boldsymbol{k}_{i1}|_2, \ldots, |\boldsymbol{k}_{i\hat{d}_i}|_2 \, \right]^\top \right|_0, \tag{11}$$

PP: what is $\hat{d}_i$ here? JZ: Added text above. suggests using the standard relaxation approach wherein the ℓ_0 norm is substituted by the ℓ_1 norm to derive a convex surrogate d_i' for d_i. This yields:

$$d_i' \triangleq \left| \left[\, |\boldsymbol{k}_{i1}|_2, \ldots, |\boldsymbol{k}_{i\hat{d}_i}|_2 \, \right]^\top \right|_1 = \sum_{j=1}^{\hat{d}_j} |\boldsymbol{k}_{ij}|_2 \tag{12}$$

that has a non-zero gradient almost everywhere. The resulting regularizer takes the form

$$\Omega_1(\boldsymbol{\Theta}) = \sum_{i=1}^{D} d_i' \tag{13}$$

We implement this approach by taking an architecture with a fixed number of kernels $\hat{d}_i$ in layer i, effectively imposing the constraints

$$d_i \leqslant \hat{d}_i \tag{14}$$

on problem (7) while using (13) as the regularizer. Given the resulting architecture with $\hat{d}_i$ kernels in each layer i, we apply a standard stochastic solver to the resulting problem. Following the learning process, none of the kernels will be exactly zero – a consequence of the stochastic update process. Hence, after the learning process is finished, it will be necessary to apply a threshold τ such that all kernels satisfying

$$|\boldsymbol{k}_{ij}|_2 \leqslant \tau \tag{15}$$

are removed from the learned CNN. PP: Is that something classic in other uses of ℓ_1 regularizer? If not, is it only because of the SGD ? JZ: Added the following While the above approach in effect adds an extra hyper-parameter, we found this to be an advantageous approach given that it is

Dataset	D	$(\hat{d}_1, \hat{s}_1), \ldots, (\hat{d}_D, \hat{s}_D)$
ImageNet	8	$(64,7)$, $(64,3) \times 2$, $(128,3) \times 2$, $(256,3) \times 2$
ImageNet	18	$(64,7)$, $(64,3) \times 4$, $(128,3) \times 4$, $(256,3) \times 4$, $(512,3) \times 4$
CIFAR-10	20	$(16,3) \times 7$, $(32,3) \times 6$, $(64,3) \times 6$
CIFAR-10	20	$(64,5) \times 18$

Table 2. Summary of base architectures used in this work. As per [?], ResNet variants of these base architectures use shortcut connections across pairs of layers starting with the second layer. The depth D indicated includes the fully connected layer for the softmax, which is not regularized. JZ: CS: please add all missing base archs. Need to center m column, > {\centering}

very simple to implement. Alternatives for exact ℓ_1 minimization within an SGD framework do exist [2, 9]. However, they lead to extra computational and memory complexity during training due to their need to represent kernels as a difference of non-negative parts $\boldsymbol{k}_{ij} = \boldsymbol{k}_{ij}^{+} - \boldsymbol{k}_{ij}^{-}$.

We note that, in practice, one can further make the constraint in (14) layer-dependent using instead

$$\forall i, d_i \leqslant \hat{d}_i. \tag{16}$$

One advantage of using such layer-dependent constraints is that it allows us to more easily compare with existing architectures by letting the $\{\hat{d}_i\}_i$ (as well as D and the s_{ij}) be given by a state-of-the-art *base architecture*. A second advantage of (16) is that it allows us to reduce learning complexity by incorporating empirical findings about the architecture selection. In Table 2 we present all the base architectures that we use in this work.

While this first regularization scheme (13) only penalizes the number of kernels d_i, it also succeeds in implicitly learning the number of layers D. In effect, by sufficiently increasing the regularization weight α in (7) entire layers can be made to dissapear. This will be possible only for residual architectures where shortcut connections will bypass layers where all kernels have been zeroed out.

In Fig. 1 we illustrate this layer-selection effect for the family of architectures obtained by fixing the $\{\hat{d}_i\}_{i=1}^{D}$ constraints and varying the regularizer weight α, where we define this to be one example in one such approach in (**??**)

It is also worth discussing that (13) is very similar to (9). The difference is that our proposed regularizer is a sum of norms, while the original regularizer is a sum of squared norms. The regularizer we are proposing is in effect a group-sparsity regularizer [?] where each group is taken to be one of the kernels, and group sparsity regularizers are known to zero out entire groups of variables.

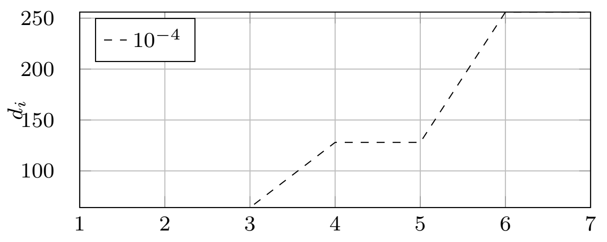

Figure 1. Illustration of the d_i-regularized family of architectures obtained for $\tau = 10^{-3}$ by varying α (value indicated in the legend) when using the seven-layer base architecture in Table 2 for the constraints $\hat{d}_i$ (denoted as the dashed envelope). The learned width d_i of each layer is denoted by the vertical length of the area plots. JZ: Need to stretch curve to occupy all the columns... Can add o markers to indicate active constraints. Need to specify all parameters. Why is this shifted right??PP: I don't understand the vertical symmetry.

Kernel-size-reducing regularizer. The second regularizer we propose reduces the size of each kernel $\boldsymbol{k}_{ij}$ independently. Similarly to the case of the d_i-reducing regularizer discussed above, the approach we follow assumes that all kernels $\boldsymbol{k}_{ij}$ have a size at most $\hat{s}_i$, in effect imposing the following constraint on the learning problem in (7):

$$0 \leqslant s_{ij} \leqslant \hat{s}_i. \tag{17}$$

In order to define a regularizer for this task, we will use structured sparsity methods – an extension of nested sparsity methods wherein the groups of variables that are zeroed together overlap or are nested. We use nested groups built using the *nested half-ring* strategy illustrated on the left in Fig. 2 for the case $\hat{s}_i = 5$. The *nested half-ring* strategy forms groups by recursively appending one top-left or bottom-right half-ring alternatively: Group $\mathcal{G}_n, n = 1, \ldots, \hat{s}_i$, is formed from $\mathcal{G}_{n-1}$ (with $\mathcal{G}_0 \triangleq \varnothing$) by appending, for all kernel channels, the outermost top-left or bottom-right column-row pair not contained in $\mathcal{G}_{n-1}$. This strategy will result in kernel sizes

$$s_{ij} \in \{0, 1, 2, \ldots, \hat{s}_i\} \tag{18}$$

that can be even-valued. For applications requiring precise spatial localization of CNN activations, odd-valued kernel sizes are advantageous. The *nested-ring* strategy illustrated on the right in Fig. 2 can be used in such situations as it results in kernels with sizes

$$s_{ij} \in \{0, 1, 3, \ldots, \hat{s}_i\} \tag{19}$$

that are zero or odd. But for conciseness, in this work we only use nested half-ring groups.

Given a group $\mathcal{G}_n$ indicating a subset of the support of a kernel $\boldsymbol{k}_{ij}$, we let $\boldsymbol{k}_{ij}(\mathcal{G}_n)$ be the vector formed from all entries of $\boldsymbol{k}_{ij}$ indicated by $\mathcal{G}_n$. When using the nested-half

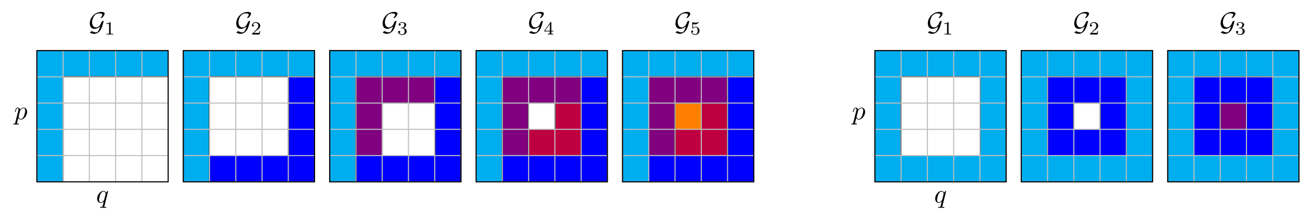

Figure 2. Examples of supports (positions of colored entries) used for groups $\mathcal{G}_n$ for the case $\hat{s} = 5$ using *(left)* a *nested half-ring* strategy with $\hat{s}$ groups resulting in $s_{ij} \in \{0, 1, 2, \ldots, \hat{s}_i\}$; and *(right)* a *nested ring* strategy with $\lceil \frac{\hat{s}}{2} \rceil$ groups resulting in $s_{ij} \in \{0, 1, 3, \ldots, \hat{s}_i\}$ that are zero or odd. For a given kernel $\boldsymbol{k}_{ij}$ with entries k_{pqr}, $1 \leqslant p, q \leqslant \hat{s}_i$, $1 \leqslant r \leqslant \hat{d}_{i-1}$, each group $\mathcal{G}_n$ contains all triplets (p, q, r) corresponding to the shaded area. Note that the shaded supports only depend on the spatial indices p and q – for a given group, the same support is used for all of the kernel's channels.

ring groups $\mathcal{G}_n$ on the left in Fig. 2, we will have that

$$s_{ij} = \left| \left[\, |\boldsymbol{k}_{ij}(\mathcal{G}_1)|_2, \ldots, |\boldsymbol{k}_{ij}(\mathcal{G}_{\hat{s}_i})|_2 \,\right]^\top \right|_0 \tag{20}$$

(this is also the case when using the nested-ring groups). Applying the same ℓ_1 relaxation strategy used to obtain (13) from (11) yields

$$s'_{ij} \triangleq \left| \left[\, |\boldsymbol{k}_{ij}(\mathcal{G}_1)|_2, \ldots, |\boldsymbol{k}_{ij}(\mathcal{G}_{\hat{s}_i})|_2 \,\right]^\top \right|_1 = \sum_{n=1}^{\hat{s}_i} |\boldsymbol{k}_{ij}(\mathcal{G}_n)|_2. \tag{21}$$

Subsequently summing over all layers and kernels produces the following s_{ij}-reducing regularization scheme:

$$\Omega_2(\boldsymbol{\Theta}) = \sum_{i=1}^{D} \sum_{j=1}^{\hat{d}_i} s'_{ij}. \tag{22}$$

Similarly to the d_i-reducing regularization scheme, the regularizer in (22) will produce kernels with rings that are only approximately zero. Hence it will be necessary to apply a threshold so that, following the learning process, the entries in all rings satisfying

$$|\boldsymbol{k}_{ij}(\mathcal{G}_n)|_2 \leqslant \tau \tag{23}$$

are zeroed out. The resulting kernel will have a size s_{ij} given by (21).

Given that the resulting kernel sizes s_{ij} can be zero, for sufficiently high regularization weights α, when using ResNet CNNs, entire layers can be made to dissapear. Note that these removed layers can occur in the middle of the network, a consequence of the ResNet shortcut connections. Contrary to the case of d_i-reducing regularization, the s_{ij}-reducing regularization can also remove entire layers from plain (non-ResNet) architectures not having shortcut connections, where a removed kernel would correspond to an identity mapping PP: confusing. In effect, we say that, provided $\hat{d}_{j-1} \leqslant \hat{d}_j$, we have the possibility to learn an identity map, right ? But this is true for non-regularized CNNs, though it never happens in practice. So, does this happen

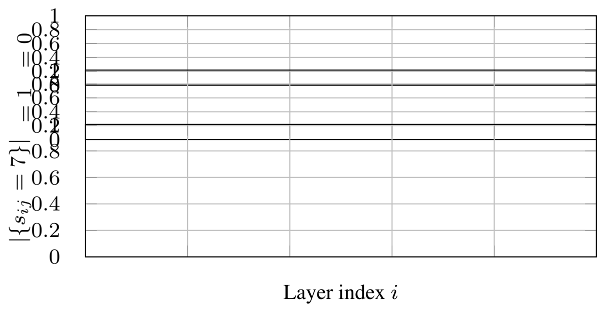

Figure 3. Illustration of the family of s_{ij}-regularized architectures obtained for τ =?? by varying α (value indicated in the legend) when using the seven layer base architecture in Table 2 for the constraints $\hat{s}_{ij}$ (denoted as the dashed envelopes). The number of kernels with a fixed size s_{ij} (one fixed s_{ij} per plot) is denoted by the vertical vertical length of the area plots. JZ: Need to stretch curve to occupy all the column... Can add o markers to indicate active constraints. Need to specify all parameters. Add legend. Add correct/more data in file `data/hist_szs_vs_layer_vs_alpha.dat`, modify for loop in picture code. Can replace the $s_{ij} = 0$ plot by a d_i snake plot. Add $\hat{s}_{ij}$ dashed envelopes

with the help of our regularizer?, as we illustrate in the top of Fig. ??. Letting $\boldsymbol{I}_{r'}$ be a kernel with entries I_{pqr}, this identity mapping has the form

$$I_{pqr} = \begin{cases} 1 & \text{if } p, q, r = 0, 0, r', \\ 0 & \text{otherwise.} \end{cases} \tag{24}$$

PP: Changed p' and q' in 0, to be checked For dissapearing layers, if one assumes that $\hat{d}_i \geqslant \hat{d}_{i-1}$, one would expect $\hat{d}_{i-1}$ such kernels, one for each input channel $r' = 1, \ldots, \hat{d}_{i-1}$, with p' and q' approximately indicating the spatial center. The remaining $\hat{d}_i - \hat{d}_{i-1}$ kernels would be completely zero-valued. JZ: To verify if this is the case... JZ: We illustrate this in the bottom of Fig. 3 .

Many other possible approaches exist to define groups that can result in supports with shapes other than square, including rectangular supports and even (convex) polygonal shapes [1]. This is a direction that needs exploring in future work.

3.4. Variants

3.4.1 Alternative norms

The following norms can be used instead of the ℓ_2 norm to define d'_i and s'_i in (12) and (21).

Group-normalized ℓ_2 norm. The group-normalized ℓ_2 norm, where $|\mathcal{G}|$ denotes the cardinality of group $\mathcal{G}$ is given by

$$\eta_{\mathcal{G}}\left(\boldsymbol{k}\right) \triangleq \left(\frac{1}{|\mathcal{G}|}\sum_{\mathcal{I}\in\mathcal{G}} |k_{\mathcal{I}}|_2\right)^{\frac{1}{2}}. \tag{25}$$

It has the property that, assuming $\mathrm{E}|k_{\mathcal{I}}|_2^2 = C^2$ for active groups (otherwise $\boldsymbol{k}(\mathcal{G}) = \mathbf{0}$) [2]

$$[?]\mathrm{E}\left[\left(\eta_{\mathcal{G}}\left(\boldsymbol{k}\right)\right)^2\right] = C^2 \tag{26}$$

does not depend on the group size. Accordingly, from Cauchy-Schwartz,

$$\mathrm{E}\left[\eta_{\mathcal{G}}\left(\boldsymbol{k}\right)\right] \leqslant \sqrt{\mathrm{E}\left[\left(\eta_{\mathcal{G}}\left(\boldsymbol{k}\right)\right)^2\right]} = C \tag{27}$$

implies that the expected norm is bounded for any group size.

The potential benefit of using $\eta_{\mathcal{G}}\left(\boldsymbol{k}\right)$ to define, for example d'_i in (12) is that each term corresponding to an active group in the summation will be bounded by C, and hence

$$d'_i \leqslant d_i \cdot C, \tag{28}$$

where d_i is the learned width for layer i.

Group-scaled ℓ_2 norm. The work in [] and [] proposes the opposite approach

$$\eta_{\mathcal{G}}\left(\boldsymbol{k}\right) \triangleq \left(|\mathcal{G}|\sum_{\mathcal{I}\in\mathcal{G}} |k_{\mathcal{I}}|_2\right)^{\frac{1}{2}}. \tag{29}$$

The motivation used in [] is that, by using this normalization, the resulting problem produces the ANOVA solution.

Other approaches. It is also possible to use other norms instead of the ℓ_2 norm, for example, the ℓ_∞ has been proposed elsewhere.

[2]Not valid?...

Batch-normalization variant. One potential problem with the above analysis is that the assumption in (**??**) is not valid. A possible reason for this is that the dynamic range of the layer-input activations can vary widely from layer to layer. The batch normalization technique [] addresses this problem by scaling and shifting the activation coefficients M_i produced by a given layer before feeding them to the next layer. This normalization is done in two steps on a per-feature-map basis. Let $\boldsymbol{m}$ denote a feature map of $\boldsymbol{M}$ and μ and σ denote an estimate of the mean and variance of the entries $\boldsymbol{m}$ derived from the current training batch, the first step

$$\boldsymbol{m}' = \frac{1}{\sigma}(\boldsymbol{m} - \mathbf{1}\mu) \tag{30}$$

ensures that the resulting feature maps (over the batch) have unit variance and zero mean.

The next step gives the system back the liberty to learn an adequate mean and variance by means of the learned parameters a and b:

$$\boldsymbol{m}'' = a\boldsymbol{m}' - b \tag{31}$$

3.5. Complexity regularizers.

The complexity expression (10) has a square depdendence on both the kernel size and the layer width that is not reflected in the regularizers in (13) and (22). This suggests instead using

$$\Omega_1(\boldsymbol{\Theta}) = \sum_{i=1}^{D} {d'_i}^2 \tag{32}$$

and

$$\Omega_2(\boldsymbol{\Theta}) = \sum_{i=1}^{D}\sum_{j=1}^{\hat{d}_i} {s'_{ij}}^2. \tag{33}$$

Alternatively, one can substitue d_i and s_{ij} in (10) by their relaxed versions to obtain

$$\Omega_3(\boldsymbol{\Theta}) = \sum_{i=1}^{D}\sum_{j=1}^{d_i} r_i \cdot c_i \cdot d'_{i-1} \cdot {s'_{ij}}^2. \tag{34}$$

4. Results

JZ: To do in relation to complexity/accuracy curves like Fig. 5 :

- Add baseline results for multiple SOTA archs *** VERY IMPORTANT *** . The more the better, this makes all the difference.
- Add similar curves for all other datasets and architectures
  - ImageNet - `ResNet` 7 and plain equivalent
  - ImageNet - `ResNet` 18

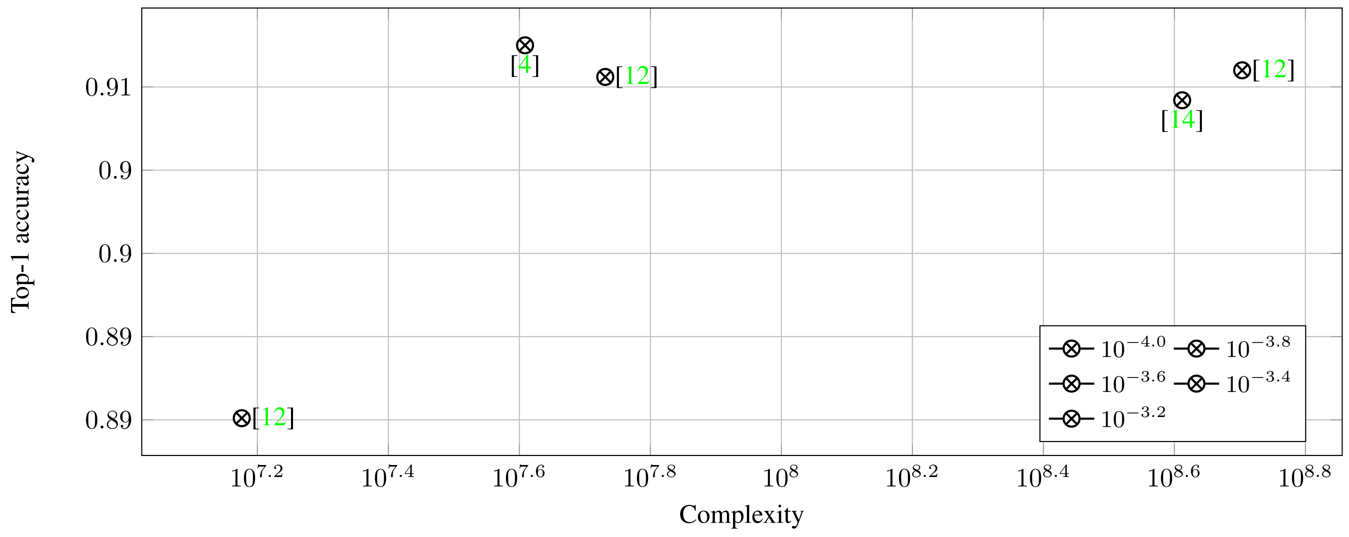

Figure 4. JZ: Better order for the legend values. Add more τ values for $\alpha = 10^{-3.8}, \ldots, 10^{-4.0}$

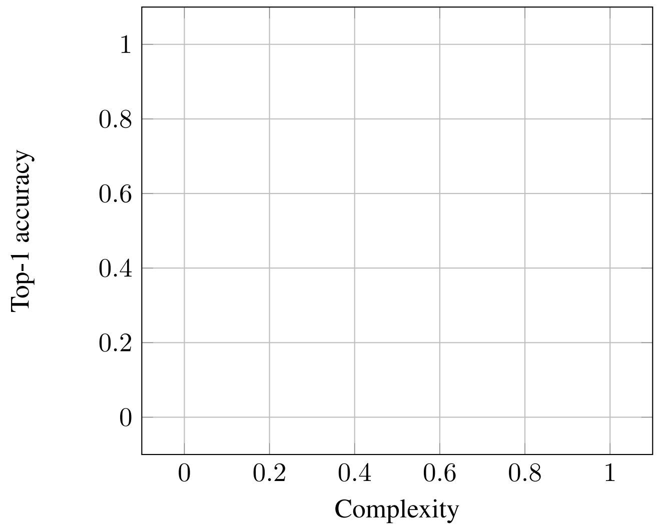

Figure 5. Kernel num. regularization, base arch. JZ: Description, more τ values...

- CIFAR10 - squared constraints ($D = 20$, $\hat{s}_i = 5$, $\hat{d}_i = 64$)
- CIFAR10 - base arch ($D = 20$, $\hat{s}_i = 3$, $\hat{d}_i$ varying).
- MNIST - Square constraints ().

• Add envelope curve (copy-paste vals ok).

JZ: Other results to add:

• Review experiments.txt file for other possible curves.

• Visualizations of kernels when learning is finished.

• Visulaizations of average kernel pairwise correlation across learning iterations and/or across alphas.

References

[1] F. Bach and I. W. Project-team. Convex Optimization with Sparsity-Inducing Norms. pages 1–35. 3, 6

[2] L. Bottou. Stochastic gradient descent tricks. In G. Montavon, G. Orr, and K.-R. Müller, editors, *Neural Networks: Tricks of the Trade*. Springer, 2nd edition, 2012. 5

[3] I. J. Goodfellow, D. Warde-farley, and A. Courville. Maxout Networks. *Journal of Machine Learning Research*, 2013. 3

[4] K. He, X. Zhang, S. Ren, and J. Sun. Deep Residual Learning for Image Recognition. In *Arxiv.Org*, volume 7, pages 171–180, 2015. 2, 8

[5] K. He, X. Zhang, S. Ren, and J. Sun. Identity Mappings in Deep Residual Networks. *arXiv preprint*, pages 1–15, 2016. 2

[6] R. Jenatton, I. Fr, and F. Bach. Proximal Methods for Sparse Hierarchical Dictionary Learning. *Bach*, 2010. 3

[7] R. Jenatton, G. Obozinski, and F. Bach. Structured Sparse Principal Component Analysis. 9:366–373, 2010. 3

[8] A. Krizhevsky, I. Sutskever, and G. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Neural Information Processing Systems*, pages 1–9, 2012. 1, 2, 3

[9] P. Kulkarni, J. Zepeda, F. Jurie, P. Pérez, and L. Chevallier. Learning the Structure of Deep Architectures via L1 Penalization. In *British Machine Vision Conference*, 2015. 2, 5

[10] M. Lin, Q. Chen, and S. Yan. Network In Network. *arXiv preprint*, page 10, 2013. 2

[11] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, and S. Reed. SSD: Single Shot MultiBox Detector. *Arxiv*, pages 1–15, 2015. 2

[12] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio. FitNets: Hints for Thin Deep Nets. In *International Conference on Learning Representations*, 2014. 8

[13] K. Simonyan and A. Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition, sep 2015. 2

[14] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller. Striving for Simplicity: The All Convolutional Net. In *International Conference on Learning and Recognition*, pages 1–14, 2015. 8

[15] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going Deeper with Convolutions. 2014. 2

[16] S. Zagoruyko and N. Komodakis. Wide Residual Networks. *Arxiv*, 2016. 2