ORNL/TM-2018/777

Composability-Centered Convolutional Neural Network Pruning

Hui Guan
Xipeng Shen
Seung-Hwan Lim
Robert M. Patton

Feb 15, 2018

OAK RIDGE NATIONAL LABORATORY
MANAGED BY UT-BATTELLE FOR THE US DEPARTMENT OF ENERGY

ORNL/TM-2018/777

Computer Science and Mathematics Division

Composability-Centered Convolutional Neural Network Pruning

Hui Guan[1], Xipeng Shen[2], Seung-Hwan Lim, Robert M. Patton

Date Published: Feb, 2018

Prepared by
OAK RIDGE NATIONAL LABORATORY
Oak Ridge, TN 37831-6283
managed by
UT-Battelle, LLC
for the
US DEPARTMENT OF ENERGY
under contract DE-AC05-00OR22725

[1]North Carolina State University, hguan2@ncsu.edu
[2]North Carolina State University, xshen5@ncsu.edu

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

ABSTRACT

This work studies the composability of the building blocks of structural CNN models (e.g., GoogleLeNet and Residual Networks) in the context of network pruning. We empirically validate that a network composed of pre-trained building blocks (e.g. residual blocks and Inception modules) not only gives a better initial setting for training, but also allows the training process to converge at a significantly higher accuracy in much less time. Based on that insight, we propose a *composability-centered* design for CNN network pruning. Experiments show that this new scheme shortens the configuration process in CNN network pruning by up to 186.8X for ResNet-50 and up to 30.2X for Inception-V3, and meanwhile, the models it finds that meet the accuracy requirement are significantly more compact than those found by default schemes.

1. Introduction

Network pruning is a common approach for compressing CNN models to fit into resource-constrained devices that have limited storage and computing power. Recent studies have focused on filter-level pruning, which removes a set of unimportant filters from each convolutional layer. Various efficient heuristic criteria on how to evaluate the importance of a filter have been proposed Li et al. [2016], Hu et al. [2016], Molchanov et al. [2016], Luo et al. [2017].

In spite of various pruning criteria, how to determine the appropriate number of filters to prune for each convolutional layer is yet a largely unexplored algorithm configuration problem, which is called *filter number configuration*. The tuning objective is to find the smallest model that meets a given accuracy requirement. The configuration space is exponential to the depth of a deep network: For a k-layer CNN, the space is m^k if each layer has m possible filter numbers to choose from. As training and testing one single configuration often takes quite some time, *filter number configuration* could be a very slow process. Previous studies have to either simply fix the number of filters to prune as a constant for all layers Luo et al. [2017], or limit the exploration to a small set of choices Li et al. [2016], imposing substantial risks of the quality of the final configurations. It remains an open question how to enable fast explorations of a large set of filter number configurations effectively.

In this work, we propose *composability-centered CNN pruning* to address the problem, particularly for recently proposed structural CNN models such as GoogleLeNet Szegedy et al. [2015], Ioffe and Szegedy [2015], Szegedy et al. [2016] and ResNet He et al. [2016]. Structural CNN models are models built on some building blocks of a predefined structure (e.g. Inception module and residual block). Since they were proposed, they have quickly drawn broad interest for their high flexibility and network quality. The basic idea of *composability-centered CNN pruning* is to first train each pruned building block, then assemble them together to form CNNs of various configurations, and finally runs a short training process on the resulting CNNs.

A fundamental hypothesis underlying the design is that:

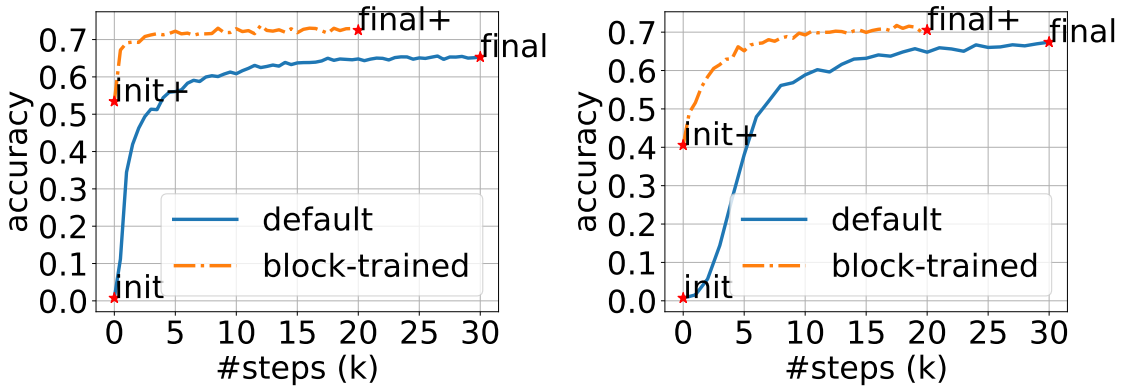
> Pre-training the building blocks of a structural CNN helps the training of that CNN reach a given accuracy sooner.

We call it *the hypothesis on the composability of CNN block-wise training quality*, or *composability hypothesis* in short. The validity of the hypothesis could potentially shorten the network pruning process. Pre-training each block takes time, but because the pre-trained result of a block benefits all the CNNs that include that block, the gains from the many reuses of the block can potentially outweigh the cost of pre-training significantly.

At the first glance, the hypothesis may look like *transfer learning*, where, a large portion (e.g., all except the top layer) of the CNN remains unchanged when a trained CNN is migrated to a new dataset. It is important to notice that the *composability hypothesis* is a hypothesis much stronger than the condition *transfer learning* builds on. Rather than a large portion is reused as a whole, the reuse of pre-training results is at the level of each piece (building block) of the CNN. The successes of *transfer learning* are hence insufficient to validate the *composability hypothesis*.

This work strives to answer the following three main research questions:

1. Does the *composability hypothesis* hold empirically?



(a) ResNet-50 **(b) Inception-V3**

Figure 1. Accuracy curves of the *default* and *block-trained* networks on dataset CUB200.

Table 1. Accuracies of *Default Networks* (init, final) and *Block-trained Networks* (init+, final+)

Blocks	Accuracy Type	Flowers102		CUB200		Cars		Dogs	
		Mean	Median	Mean	Median	Mean	Median	Mean	Median
residual (ResNet-50)	init	0.065	0.035	0.018	0.012	0.015	0.012	0.014	0.010
	init+	0.924	0.926	0.658	0.662	0.682	0.690	0.729	0.735
	final	0.962	0.962	0.706	0.707	0.799	0.800	0.753	0.754
	final+	0.970	0.970	0.746	0.746	0.820	0.821	0.789	0.791
inception (Inception-V3)	init	0.047	0.029	0.015	0.011	0.011	0.009	0.014	0.012
	init+	0.861	0.866	0.569	0.571	0.531	0.542	0.561	0.563
	final	0.959	0.959	0.711	0.711	0.794	0.796	0.726	0.728
	final+	0.965	0.965	0.735	0.735	0.811	0.811	0.755	0.755

2. If it does, how to design a mechanism to effectively materialize its potential for optimizing the speed and quality of CNN pruning?

3. How much can that help?

Section 2. answers the first question through a set of experiments. The results on different networks and various datasets all provide strong positive support to the hypothesis. Sections 3. and 4. describe the design of *composability-centered pruning* and its implementation on TensorFlow Abadi et al. [2016]. Section 5. reports the experimental results. For ResNet-50 and Inception-V3, *composability-centered pruning* shortens the pruning process by up to 186.7X and 30.2X respectively. Meanwhile, the models it finds are significantly more compact than those by the default pruning scheme.

2. Empirical Validation of the Composability Hypothesis

To examine the *composability hypothesis*, we experiment with four different CNN models: ResNet-50 and ResNet-101, as representatives of the Residual Network family, and Inception-V2 and Inception-V3, as representatives of the Inception family. We adapt the four CNN models pre-trained on ILSVRC 2012 Russakovsky et al. [2015] to each of the four image datasets (listed in Table 2), resulting in our unpruned models.

For each unpruned model, we derive 500 pruned networks by pruning each building block of the model by r fraction, where r is randomly chosen among {30%, 50%, 70%}. For each of the networks, we get a second version by training each of its building blocks individually; the data used to train are the inputs and outputs of that building block in the unpruned model. (More details are in Section 4.1.) We call the second version of the network the *block-trained network*, and the first version the *default network*.

Figure 1 shows the accuracy curves of the two versions of a ResNet-50 network and an Inception-V3 network on dataset CUB200. The initial accuracies (*init*) are close to zero for the *default version*, while 53.4% and 40.5% for the *block-trained version*. Moreover, the *default version* gets only 65.3% and 67.3% final accuracies (*final*) respectively, while the *block-trained version* achieves 72.5% and 70.5% after only two-thirds of the training time.

Table 1 reports the mean and median of the initial and final accuracies of all 500 *block-trained networks* and their *default* counterparts for each of the models on every dataset. Only results for ResNet-50 and Inception-V3 are listed due to the space limit. Overall, *block-trained networks* yield better final accuracies than *default networks* do with one-third less training time.

The results offer strong evidence supporting the *composability hypothesis*. Next, we describe how we leverage the composability for effectively optimizing the speed and quality of CNN pruning.

3. Problem Statement

We start by giving a formal definition of the basic configuration problem in CNN pruning, and then explain the *filter number configuration problem* for structural CNN model pruning.

3.1 Algorithm Configuration in Network Pruning

Consider a set of training examples $\mathcal{D} = \{(\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_{|\mathcal{D}|}, y_{|\mathcal{D}|})\}$, where $\mathbf{x}$ and y represent an input and a target output respectively. The training of a CNN model is to learn a set of parameters $\mathcal{W}$ such that the accuracy of the network $f(\mathcal{W}, \mathcal{D})$ is maximized, $\max_{\mathcal{W}} f(\mathcal{W}, \mathcal{D})$, through minimizing some cost functions (e.g. cross entropy). Note that a parameter might represent an individual weight, a convolution kernel, or the set of kernels (i.e. a filter) that compute a feature map. In this work, a parameter refers to a filter.

Network pruning reduces the number of parameters $\mathcal{W}$ by keeping only a subset of parameters $\mathcal{W}' \subset \mathcal{W}$. Network pruning tries to reduce the model size as much as possible while keeping the accuracy of the network still meeting some requirement. This is a combinatorial optimization problem:

$$\begin{aligned} &\min_{\mathcal{W}'} \|\mathcal{W}'\|_0 \\ &s.t. \quad \frac{\max_{\mathcal{W}} f(\mathcal{W}, \mathcal{D}) - \max_{\mathcal{W}'} f(\mathcal{W}', \mathcal{D})}{\max_{\mathcal{W}} f(\mathcal{W}, \mathcal{D})} <= \alpha, \end{aligned} \tag{1}$$

where α is the accuracy drop ratio for the pruned network. For example, if α is 0.01, only one percentage accuracy drop is allowed. After pruning, the pruned network inherits the value of $\mathcal{W}'$ and is re-trained to recover the accuracy.

Finding a good subset of parameters to optimize the objective defined in Equation 1 will require $2^{|\mathcal{W}|}$ evaluations for a given dataset. Since it is impractical to solve this optimization exactly, previous work in

network pruning simplifies the problem as identifying and removing the least important filters. Many efficient heuristic criteria on how to evaluate the importance of a filter have been proposed, such as L_1/L_2 norm of neuron weights Li et al. [2016], mean activations, average percentage of zeros (APoZ) Hu et al. [2016], and Taylor expansion Molchanov et al. [2016].

Let L be the number of layers in a CNN network and $\mathcal{W}_1, \cdots \mathcal{W}_L$ be the parameters in each layer, $\mathcal{W} = \cup_{i=1}^{L} \mathcal{W}_i$. Each layer contains a set of filters $\mathcal{W}_i = \{W_i^1, \cdots, W_i^{N_i}\}$. Given a pruning criterion, the importance score of each filter $W_i^n \in \mathcal{W}_i$ can be calculated with some formula. A subset of filters with the lowest importance scores are discarded for each layer. Let γ_i be the percentage of filters kept for the i-th layer in a pruned CNN network and $\boldsymbol{\gamma} = (\gamma_1, \cdots, \gamma_L)$. After pruning, the set of filters in the i-th layer becomes $\mathcal{W}_i' = \{W_i'^1, \cdots, W_i'^{N_i'}\}$, where $N_i' = \gamma_i \times N_i$. The parameters for the pruned network become $\mathcal{W}' = \cup_{i=1}^{L} \mathcal{W}_i'$.

A problem orthogonal to identifying filter importance is how many least important filters should be removed from each convolution layer, i.e, how to set the values for $\boldsymbol{\gamma}$. We call the problem *filter number configuration problem*. The size of the configuration space is $\prod_{i=1}^{L} |\Gamma_i|$, where $\gamma_i \in \Gamma_i, i = 1, \cdots, L$. For example, ResNet-50 He et al. [2016] contains 54 convolution layers. If the pruning options are fixed to $\Gamma = \{0.3, 0.5\}$ for all layers, there are totally 2^{54} configurations to evaluate. We are not aware of any prior work that focuses on this configuration problem.

Some general methods proposed for speeding up the configuration process of algorithms Hoos [2011] could be applied to reduce the number configurations to evaluate. This work focuses on a complementary direction, i.e., how to accelerate the examination of the remaining configurations. Reducing the evaluation time for each configuration could further largely shorten the configuration process. Particularly, we focus on speeding up the pruning of structural CNN models. we next describe the specific filter number configuration problem for structural CNN models.

3.2 Filter Number Configuration for Structural CNN Models

CNN architectures largely fall into two broad groups: the traditional models represented by AlexNet Krizhevsky et al. [2012] or VGGNet Simonyan and Zisserman [2014], and recent structural variants like GoogLeNet Szegedy et al. [2015] and ResNet He et al. [2016]. We refer to the recent structural variants as *structural CNN models* as they adopt some novel network structures like Inception in GoogLeNet or residual blocks in ResNet. These models are able to achieve state-of-the-art performance on many computer vision tasks. Being able to compress such models is of prominent importance for embedded deployment.

For structural CNN models, we consider network pruning at the granularity of building blocks composed of multiple convolutional layers instead of a single layer. Suppose a subset of unimportant filters from a convolutional layer are removed based on a certain criterion, its outputs (i.e. activation maps) have a different size from the outputs from the original layer. However, for a building block, as long as the last convolutional layer is not pruned, the outputs of the block will be of the same size as the original block. Thus we could train the pruned blocks by reconstructing activation maps from the original block in the unpruned network.

Let B be the number of blocks in a CNN network, $\mathcal{W}_b$ be the parameters in the b-th original block, $\mathcal{W}_b'$ be the parameters in the b-th pruned block, and δ_b be the percentage of filters kept for convolutional layers within the b-th block. For structural models, instead of optimizing the configuration of $\boldsymbol{\gamma}$, we optimize $\boldsymbol{\delta} = (\delta_1, \cdots, \delta_B)$. After $\boldsymbol{\delta}$ is determined, the pruning would be just prune the $1 - \delta_b$ fraction of the filters that

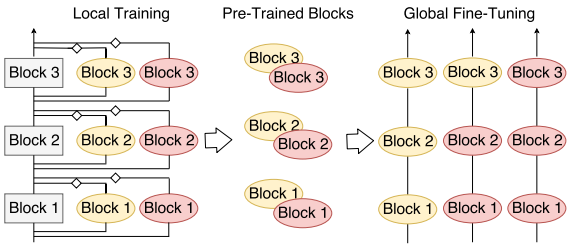


Figure 2. Illustration of composability-centered network pruning. Eclipses are pruned building blocks; rectangles are original blocks; diamonds refer to the activation map reconstruction error. Different colors of pruned blocks correspond to different pruning options.

are the least important from convolutional layers in the b-th block; the importance of the filers are determined through existing techniques. In the experiments, we adopt the pruning criterion used in Li et al. [2016].

Mathematically, the configuration problem for structural models can be defined as:

$$\begin{aligned} &\min_{\boldsymbol{\delta}} \|\mathcal{W}'\|_0, \\ s.t. \quad &\frac{\max_{\mathcal{W}} f(\mathcal{W}, \mathcal{D}) - \max_{\mathcal{W}'} f(\mathcal{W}', \mathcal{D})}{\max_{\mathcal{W}} f(\mathcal{W}, \mathcal{D})} <= \alpha, \\ &\delta_b \in \Delta_b, b = 1, \cdots, B, \end{aligned} \tag{2}$$

where Δ_b is the available percentages of filters kept for the b-th block. The size of the configuration space is $\prod_{b=1}^{B} |\Delta_b|$.

We propose composability-centered network pruning to reduce evaluation time of trial configurations when solving the configuration problem defined in Equation 2. Details of this method are described in the next section.

4. Method

This section describes our composability-centered network pruning algorithm. It has two phases: the first phase locally trains each pruned block and the second phase assembles the blocks together and then conducts a global fine-tuning for each block-trained network. Figures 2 illustrates the two phases. We next explain the two phases in detail.

4.1 Phase One: Local Training

Local training works at building blocks instead of individual convolutional layers. Figure 3 gives an illustration of the pruning strategy for a residual block. Each rectangle represents a convolutional layer. Colored convolutional layers are pruned with a given pruning option (e.g. 50%). The last convolutional layer from each branch is modified accordingly but not pruned, keeping the block output dimension unchanged. Such restriction is also applied in Luo et al. [2017]. According to the results in Li et al. [2016], pruning the last layer in a branch could cause more severe performance degradation compared with pruning other layers. We leave a rigorous analysis about the restriction to future work.

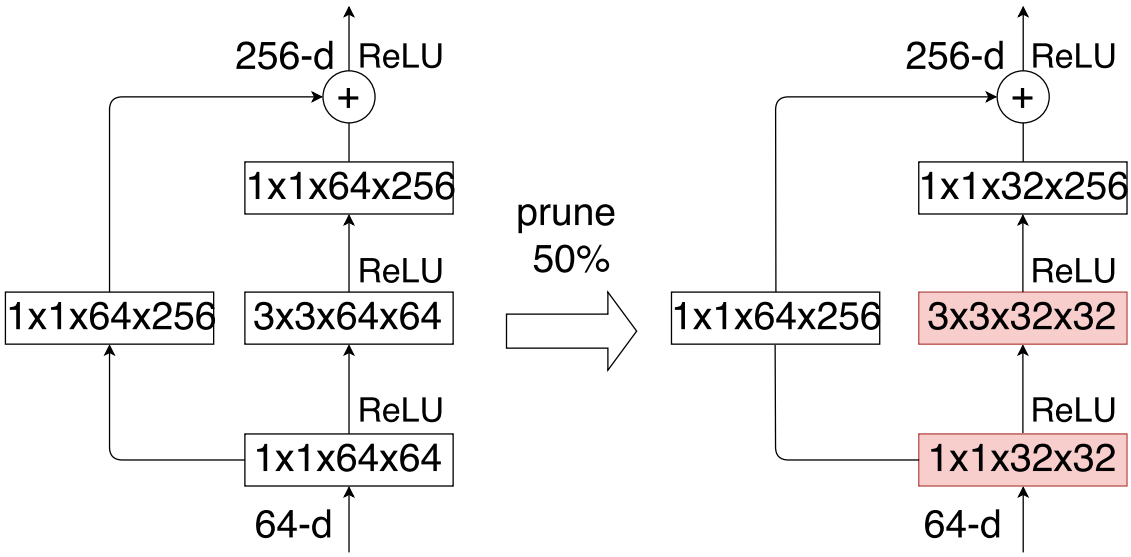


Figure 3. Illustration of the residual block pruning strategy. We only prune the first two convolutional layers so that the block output dimension does not change.

Each pruned block is trained by minimizing the reconstruction error between the output activation maps from the pruned block and the ones from the original block. Let O_b be the output activation maps from the b-th original block and the O'_b be the ones from the b-th pruned block, the optimization objective is:

$$\min_{W'_b} \frac{1}{|O_b|} \|O_b - O'_b\|_2^2 \tag{3}$$

Local training is illustrated in the left side of Figure 2. Each original block and the corresponding pruned block share the same input activation maps and yield output activation maps with the same dimension. The parameters in each pruned block are trained with the objective defined in Equation 3 through Stochastic Gradient Descent (SGD). Pruned blocks with more pruning options can be attached to the original network and concurrently trained by sharing the intermediate inference results (i.e. activation maps) from the original network.

4.2 Phase Two: Composing and Global Fine-Tuning

The local training phase outputs a bag of pre-trained pruned blocks, as shown in Figure 2 (blocks in the original network could also be included). These blocks could be selected and stacked together to construct a block-trained network. Note that the order of pre-trained blocks in a block-trained network is the same as the order of their corresponding original blocks in the original network: a pruned block, locally trained with the outputs from the i-th original block, should also be the i-th block in a block-trained network. Three networks assembled with three different sets of pre-trained blocks are shown on the right side of Figure 2.

Ideally, if the output activation maps from original blocks can be reconstructed without error (i.e. $O_b = O'_b, b = 1, \cdots, B$), a block-trained network should be able to produce the same accuracy performance as the original network. In practice, however, a pruned block with only a subset of parameters has a smaller

Table 2. Dataset Statistics.

Dataset	Size			Classes	Accuracy	
	Total	Training	Testing		ResNet	Inception
Flowers102	8,189	6,149	2,040	102	0.973	0.968
CUB200	11,788	5,994	5,794	200	0.770	0.760
Cars	16,185	8,144	8,041	196	0.822	0.801
Dogs	20,580	12,000	8,580	120	0.850	0.835

model capacity compared with the corresponding original block. Also, finding the global optimum for Equation 3 is difficult due to its non-convex property. Therefore, a global fine-tuning is required to further recover the accuracy performance of a block-trained network. Compared with training a default network, which is built with pruned blocks that are not locally trained, fine-tuning a block-trained network requires less training time.

4.3 Implementation

Our implementation is based on TensorFlow Abadi et al. [2016]. We built an integrated framework that allows the input activation maps and output activation maps for each block in the original network to be fed on the fly to serve as the input and ground truth in the training of the blocks. This scheme circumvents the difficulty of saving the intermediate activation maps of the original network.

5. Experiments

We empirically evaluate the effectiveness and efficiency of composability-centered network pruning in this section, reporting its benefits on both the speed and the quality of network pruning. We first explain the experiment settings in Section 5.1, describe the configuration optimization strategy in Section 5.2, and then report the experiment results in Sections 5.3 with both single node and multiple nodes.

5.1 Methodology

We pruned ResNet-50 as the representative of the Residual Network family and Inception-V3 as the representative of the Inception family. The two models pre-trained on ILSVRC 2012 Russakovsky et al. [2015] are adapted to four image classification tasks with the datasets Flowers102 Nilsback and Zisserman [2008], CUB200 Welinder et al. [2010], Cars Krause et al. [2013], and Dogs Khosla et al. [2011]. The statistics of the four datasets including the data size for training (*Train*), the data size for testing (*Test*), and the number of classes (*Classes*) are reported in Table 2.

To get well-trained models for the four datasets, we trained ResNet-50 and Inception-V3 for 40,000 steps with batch size 32, fixed learning rate 0.001, weight decay 0.00004 for all the datasets. Additional annotations including bounding boxes and part labels are not used in the training. The preprocessing for Resnet-50 is the same as Simonyan and Zisserman [2014] and the preprocessing for Inception-V3 follows Szegedy et al. [2015]. The accuracy of the trained ResNet-50 and Inception-V3 models on the test datasets are listed in columns *ResNet* and *Inception* in Table 2.

5.1.0.1 Network Pruning We use the $L1$ norm of a filter proposed by Li et al. [2016] as the pruning criterion. An illustrations of pruning a residual block is shown in Figure 3. For all experiments, network training is performed on the training sets while accuracy results are reported on the testing sets. ResNet-50 contains totally 16 residual blocks ($B = 16$) and Inception-V3 contains 11 inception modules ($B = 11$). The set of pruning options for each block are fixed as $\Delta = \{0.3, 0.5, 0.7\}$. Thus the configuration space for ResNet-50 is 3^{16} and for Inception-V3 is 3^{11}.

5.1.0.2 Baseline In the baseline approach, when filters are pruned, a new model with fewer filters is created and the remaining parameters of the modified layers, as well as the unaffected layers, are copied into the new model. Given a set of configurations to evaluate, the baseline approach trains networks pruned according to each configuration for 30,000 steps with batch size 32, fixed learning rate 0.001, and weight decay 0.00001. We assume that the training time for recovering the accuracy of a pruned network could be smaller than the training time for adapting a pre-trained model to new datasets due to weight inheritance in network pruning. Thus we use a smaller number of steps for training.

5.1.0.3 Composability-Centered Network Pruning Our proposed approach first locally trains each pruned block, assembles the trained blocks according to each configuration and conducts a global fine-tuning on each block-trained network. The number of pruned blocks to train for ResNet-50 and Inception-V3 are 48 (i.e., 3×16) and 33 (i.e., 3×11) respectively. Local training takes 10,000 steps for ResNet-50 with batch size 32, fixed learning rate 0.2, and weight decay 0.0001, and takes 20,000 steps for Inception-V3 with batch size 32 and fixed learning rate 0.08, and weight decay 0.0001. The second phase fine-tuning uses the same hyper-parameters as the baseline approach does.

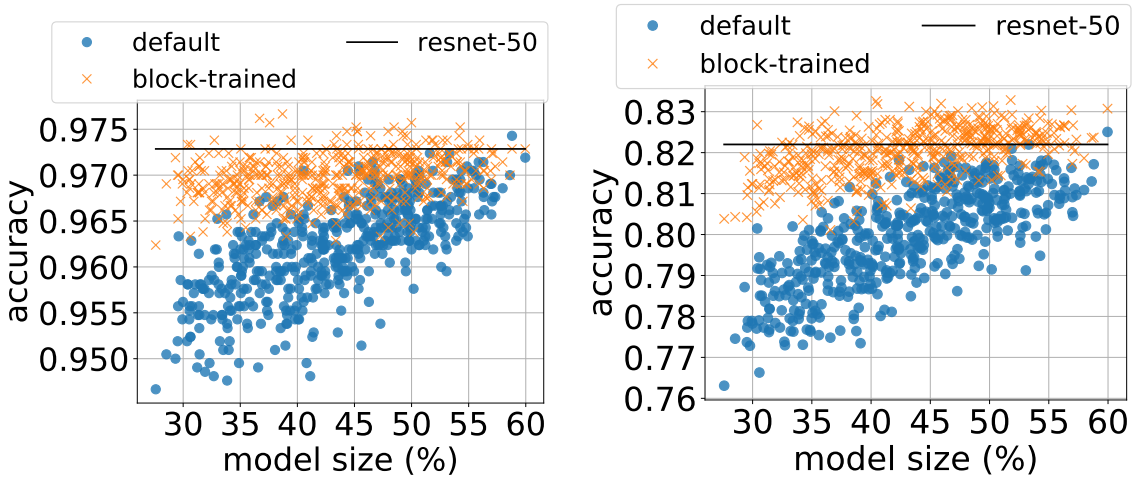
5.1.0.4 Configuration Objective According to the objective function defined in Equation 2, the best configuration is the one that has the smallest model size but maintains a satisfying accuracy (i.e., with an accuracy drop rate less than α.) We experiment with a spectrum of α values, ranging from -2% to 8%. A negative drop rate (e.g. -2%) means that the accuracy of the pruned model should be higher than the original model (e.g. two percentages higher). We include negative drop rates because it is possible for the pruned networks to outperform the original as shown in Figure 4. The model size refers to the number of model variables. The model sizes for the original ResNet-50 and Inception-V3 are around 25.6 million and 27.2 million.

All the experiments are performed using TensorFlow 1.3.0 on Titan. Titan is a Cray XK7 supercomputer located at the Oak Ridge Leadership Computing Facility. Each compute node contains a 16-core 2.2GHz AMD Opteron 6274 (Interlagos) processor, 32 GB of RAM and an NVIDIA Kepler GPU with 6 GB of DDR5 memory. In the experiments, one network is trained using one node with one GPU.

5.2 Configuration Optimization Strategy

According to the optimization objective described in Section 3.2, a naive approach to finding the best model is to sort the configurations in an increasing order based on model size and evaluate one by one. It ensures a global optimum if there is at least one satisfying model that can meet the constraints in Equation 2. However,

https://www.olcf.ornl.gov/titan/



(a) Flowers102 **(b) Cars**

Figure 4. Accuracies of sampled configurations for ResNet-50.

as mentioned in Section 5.1, the size of the configuration space for ResNet-50 is 3^{16} and for Inception-V3 is 3^{11} based on our experiment settings. The above strategy is not feasible in practice.

Instead of evaluating the entire set of configurations in the space, we select configurations that are representatives of the space with respect to model sizes through sampling. To make the model size of sampled configurations spread out evenly in the space, we force the model sizes of sampled configurations to follow a close-to-uniform distribution.

Specifically, let N be the number of sampled configurations. We first include the special configurations with $\delta_b = \delta, \forall \delta \in \Delta, b = 1, \cdots, B$. To sample remaining configurations, we limit the probability of the summation of a configuration, $p(\sum_{b=1}^{B} \delta_b) \leq \frac{\lambda}{B \times (|\Delta|-1)+1}$. In the experiments, $|\Delta| = 3$. λ is set to be 1.5.

The strategy to explore the configuration space is as follow:

1. Sample 500 configurations from the space using the above sampling approach.
2. Sort the set of sampled configurations based on corresponding model size in an increasing order.
3. Start evaluations from the smallest models and proceed to larger ones.
4. Stop evaluation of the remaining configurations once a satisfying model that meets the constraints in Equation 2 is found.

Based on the above strategy, the number of trial configurations that are evaluated could be much less than 500 if there is one satisfying model found at the early stage of the exploration. Figure 4 shows the final accuracies of all the 500 ResNet-50 variants trained with or without leveraging composability on the Flower102 and CUB200 datasets. For reference, we also plot the accuracies of the original well-trained ResNet-50 on the two datasets. The exploration is from the smallest model size, which is less than 30% of the original model size, to the largest one, about 60% of the original model size.

According to Figures 4a and 4b, the block-trained network gets a better final accuracy, which echoes the results reported in Section 2.. Since a block-trained network takes less training time and also yields a higher

Table 3. Speedups and Configurations Saving for ResNet-50 in the Single Node Setting.

dataset	α	thr_acc	#configs		saving	model size		diff	time(h)		speedup(X)
			base	comp		base	comp		base	comp	
Flowers102	-2%	0.992	500	500	0.0%	100.0%	100.0%	0.0%	2858.7	1912.7(0.4%)	1.5
	-1%	0.983	500	500	0.0%	100.0%	100.0%	0.0%	2858.7	1912.7(0.4%)	1.5
	0%	0.973	297	3	99.0%	45.4%	29.3%	16.0%	1639.4	16.9(40.4%)	96.8
	1%	0.963	6	1	83.3%	29.6%	27.6%	2.0%	31.0	8.3(82.8%)	3.8
CUB200	3%	0.747	500	6	98.8%	100.0%	29.6%	70.4%	2884.1	27.9(24.5%)	103.4
	4%	0.739	323	2	99.4%	46.6%	28.5%	18.1%	1807.3	12.7(53.7%)	142.1
	5%	0.731	297	1	99.7%	45.4%	27.6%	17.8%	1654.7	8.9(77.1%)	186.8
	6%	0.724	154	1	99.4%	38.8%	27.6%	11.2%	840.1	8.3(82.6%)	101.6
Cars	-1%	0.830	500	100	80.0%	100.0%	35.7%	64.3%	2864.9	362.4(1.9%)	7.9
	0%	0.822	332	11	96.7%	46.9%	30.4%	16.5%	1848.6	44.4(15.4%)	41.6
	1%	0.814	189	2	98.9%	40.4%	28.5%	11.9%	1026.4	12.8(53.4%)	80.2
	2%	0.806	104	1	99.0%	35.8%	27.6%	8.2%	555.6	9.6(71.2%)	57.8
Dogs	5%	0.808	500	243	51.4%	100.0%	43.0%	57.0%	2852.7	885.9(0.8%)	3.2
	6%	0.799	500	123	75.4%	60.0%	36.9%	23.0%	2848.1	441.1(1.6%)	6.5
	7%	0.791	434	70	83.9%	51.9%	34.2%	17.6%	2445.4	251.8(2.7%)	9.7
	8%	0.782	297	11	96.3%	45.4%	30.4%	15.0%	1632.8	42.3(16.2%)	38.6

* *thr_acc* is the threshold accuracy corresponding to an accuracy drop rate α. *base* and *comp* refer to the baseline approach and the our proposed composability-centered approach. *saving* = (#configs in *base* − #configs in *comp*)/(#configs in *base*)×100%. *diff* = model size in *base*− model size in *comp*. The percentage of time spent on the local training for *comp* is in parentheses.

final accuracy, we expect composability-centered network pruning could save some training time of the baseline approach and also be able to find pruned models with smaller sizes.

5.3 Experiment Results

We first evaluate the performance of composability-centered network pruning in the single node scenario and then proceed to the cases with multiple nodes.

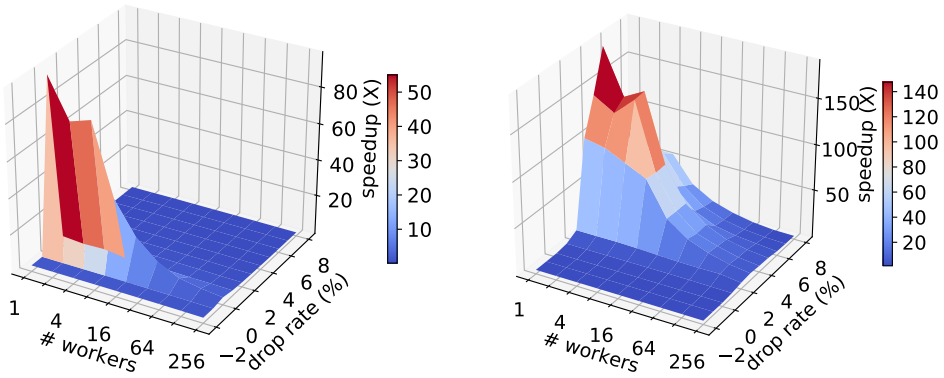
5.3.1 Results on Single Node

In the single node setting, the configurations are evaluated sequentially. The speedups from composability-based network pruning are listed in Tables 3 and 4. The number of configurations to evaluate and the total evaluation time (in hours) for the baseline approach and our approach are listed in the columns $\#configs$ and $time(h)$. The percentage of configurations avoided from the evaluation are listed in the column *saving*. The total evaluation time for our approach includes the time spent on local training, that is, the time on pre-training each building block. Due to space constraints, we only listed the results with four different drop ratios.

According to Table 3, we could avoid up to 99.6% of trial configurations and reduce the evaluation time up to 186.7X by leveraging pre-trained residual blocks for pruning ResNet-50. According to Table 4, we could avoid up to 96.7% of trial configurations and reduce the evaluation time up to 30.2X by leveraging pre-trained inception blocks for pruning ResNet-50.

Table 4. Speedups and Configurations Saving for Inception-V3 in the Single Node Setting.

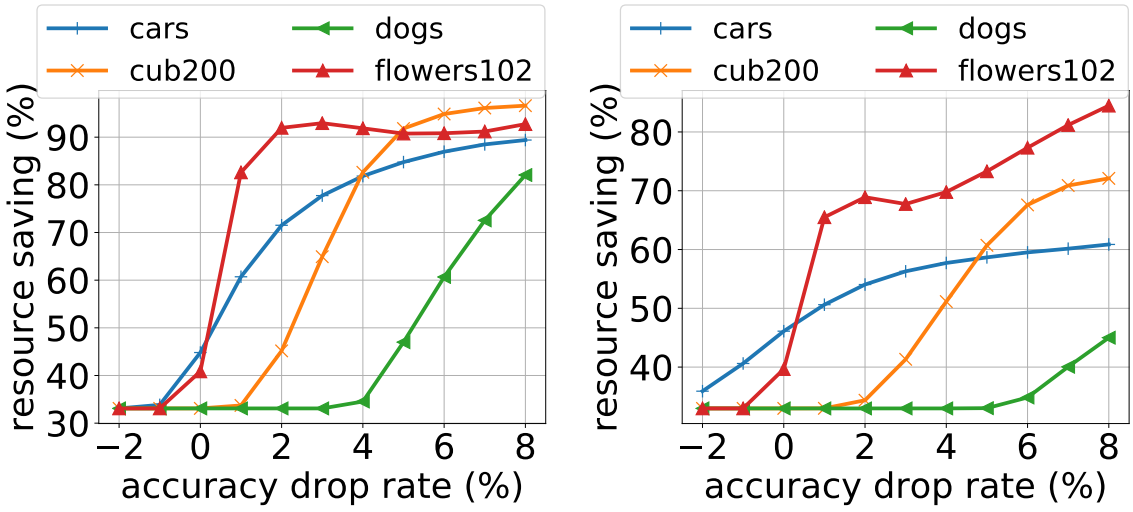
dataset	α	thr_acc	#configs		saving	model size		diff	time(h)		speedup(X)
			base	comp		base	comp		base	comp	
Flowers102	-2%	0.987	500	500	0.0%	100.0%	100.0%	0.0%	3018.8	2023.5(0.5%)	1.5
	-1%	0.978	500	500	0.0%	100.0%	100.0%	0.0%	3018.8	2023.5(0.5%)	1.5
	0%	0.968	244	10	95.9%	43.2%	32.4%	10.8%	1428.6	47.3(23.3%)	30.2
	1%	0.958	27	1	96.3%	33.9%	31.0%	2.9%	152.6	13.9(79.0%)	11.0
CUB200	4%	0.729	201	24	88.1%	41.4%	33.7%	7.7%	1163.8	100.3(10.9%)	11.6
	5%	0.722	120	4	96.7%	38.5%	31.5%	7.0%	688.0	24.9(43.8%)	27.6
	6%	0.714	63	1	98.4%	35.9%	31.0%	4.9%	357.7	14.1(77.5%)	25.4
	7%	0.706	44	1	97.7%	34.8%	31.0%	3.8%	247.8	13.8(79.0%)	17.9
Cars	-2%	0.817	311	63	79.7%	45.8%	35.9%	9.9%	1828.9	250.2(4.4%)	7.3
	-1%	0.809	155	20	87.1%	40.1%	33.5%	6.6%	896.5	85.6(12.8%)	10.5
	0%	0.801	84	3	96.4%	36.9%	31.3%	5.6%	480.3	21.8(50.2%)	22.0
	1%	0.793	33	1	97.0%	34.4%	31.0%	3.4%	186.4	14.2(77.0%)	13.1
Dogs	5%	0.794	500	499	0.2%	100.0%	56.6%	43.4%	3001.5	2007.5(0.5%)	1.5
	6%	0.786	500	356	28.8%	100.0%	47.9%	52.1%	3001.5	1410.5(0.8%)	2.1
	7%	0.777	496	201	59.5%	56.0%	41.4%	14.6%	2974.2	786.0(1.4%)	3.8
	8%	0.769	355	129	63.7%	47.9%	39.0%	8.9%	2094.8	503.6(2.2%)	4.2



(a) Flowers102 **(b) CUB200**

(c) Cars **(d) Dogs**

Figure 5. Speedups on ResNet-50 in distributed settings.



(a) ResNet-50 **(b) Inception-V3**

Figure 6. Computation resource saving by composability-centered pruning.

The speedups come from two aspects. Firstly, the second phase fine-tuning takes less training steps and thus less training time. Even without any configuration saving, our proposed approach could bring up to 1.5X speedup as the results on Flower102 show. It is because the training steps for fine-tuning is two-thirds of the training steps in the baseline approach. The local training takes some time, but the time is small, easily offset by the time savings in the fine-tuning phase. For example, the total time spent on local training of residual blocks pruned with the three pruning options of ResNet-50 on Flowers102 is around 410 min (124 min for $\delta = 0.3$, 130 min for $\delta = 0.5$, and 156 min for $\delta = 0.7$). Fine-tuning one configuration in our approach takes at least 200 min while the training time in the baseline is at least 300 min. If totally 500 configurations need to evaluate, the overhead of local training is negligible due to the many reuses of trained building blocks. The results for Flowers102 in Tables 3 and 4 show that the percentage of time spent on local training takes only 0.4% and 0.5% of the end-to-end time in our proposed approach for ResNet-50 and Inception-V3 with $\alpha = -2\%, -1\%$.

The second aspect is the saving of trial configurations. Because a block-trained network could achieve a higher final accuracy, composability-centered network pruning could stop the exploration of configuration space earlier than the baseline approach. For example, in Table 3, with a drop rate $\alpha = 0\%$, the baseline approach needs to evaluate 297 configurations before a satisfying model is found on the Flowers102 dataset. It takes 1639.4 hours of training. However, by leveraging pre-trained residual blocks, the number of configurations to evaluate is reduced to three and the training time including both the first phase local training and the second phase fine-tuning is only 16.9 hours. This second aspect contributes to the most of the speedups.

Due to the saving of trial configurations, the size of a satisfying model found by leveraging composability is also smaller, as evidenced by the results in columns *model size* and *diff* in Tables 3 and 4. For example, for ResNet-50 on Flowers102 with $\alpha = 0\%$, after evaluating 297 configurations, the baseline approach found a satisfying model that is 45.4% of the original model size while our approach found a much smaller satisfying model that is 29.3% of the original model size. Overall, in most cases, satisfying models found by leveraging

composability are much smaller than the models found by the baseline approach.

5.3.2 Results on Multiple Nodes

The results from distributed settings are interesting to examine because it is commonly used to accelerate the end-to-end time of algorithm configurations, especially when the configurations to examine are independent of each other.

We specify the number of workers to be 2, 4, 8, 16, 32, 64, 128, and 256. Each worker corresponds to one node with one GPU. Each worker evaluates the configuration with the current smallest model size that is not evaluated yet. Once a worker found a satisfying model, all the workers stop the training. Speedups are calculated as the ratio of the end-to-end configuration time of the baseline approach and that of the composability-centered network pruning, which includes the time for both local training and fine-tuning.

As mentioned in Section 5.3.1, the speedups of leveraging composability mainly comes from the saving of trial configurations. With an increasing amount of nodes, the end-to-end time saving from the reduced number of trial configurations becomes less significant. It is because even though in the composability-centered case, lots of computing nodes are idle, waiting for the slowest node to finish, the end-to-end time is determined by the finish of the slowest node. In addition, the overhead of local training becomes prominent because the pruned building blocks are trained on a single node in this experiment. The above two reasons explain the speedup trends shown in Figure 5. It is worth mentioning that the overhead of local training can be reduced by pre-training building blocks with multiple nodes since they are independent tasks. We leave this to future work.

Although the speedups become smaller with multiple nodes, the total used computing resource (e.g., the sum of the active computing times of all nodes) is significantly reduced by the computability-centered version. As Figure 6 shows, even in the extreme case with 500 nodes, there are up to 96.6% computation resource saving for pruning ResNet-50 and up to 84.4% resource saving for pruning Inception-V3. Note that the size benefits of the composability-centered network pruning remain the same as in the single node setting; the sizes of the models it finds are significantly smaller than those found by the baseline method.

6. Conclusions

In this work, we have studied composability for pruning structural CNN models. We empirically validated the *composability hypothesis* and proposed a *composability-centered network pruning* method to accelerate the filter number configuration problem. Experimental results show that by leveraging the composability, the method reduces the configuration time by a factor of up to 186.7 for ResNet-50 and up to 30.2 for Inception-V3 and also finds satisfying models of significantly smaller sizes.

References

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Holger H Hoos. Automated algorithm configuration and parameter tuning. In *Autonomous search*, pages 37–71. Springer, 2011.

Hengyuan Hu, Rui Peng, Yu-Wing Tai, and Chi-Keung Tang. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *arXiv preprint arXiv:1607.03250*, 2016.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456, 2015.

Aditya Khosla, Nityananda Jayadevaprakash, Bangpeng Yao, and Fei-Fei Li. Novel dataset for fine-grained image categorization: Stanford dogs. In *Proc. CVPR Workshop on Fine-Grained Visual Categorization (FGVC)*, volume 2, page 1, 2011.

Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3d object representations for fine-grained categorization. In *Computer Vision Workshops (ICCVW), 2013 IEEE International Conference on*, pages 554–561. IEEE, 2013.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. *arXiv preprint arXiv:1707.06342*, 2017.

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

Maria-Elena Nilsback and Andrew Zisserman. Automated flower classification over a large number of classes. In *Computer Vision, Graphics & Image Processing, 2008. ICVGIP'08. Sixth Indian Conference on*, pages 722–729. IEEE, 2008.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2818–2826, 2016.

Peter Welinder, Steve Branson, Takeshi Mita, Catherine Wah, Florian Schroff, Serge Belongie, and Pietro Perona. Caltech-ucsd birds 200. 2010.