Generalization of Teacher-Student Network and CNN Pruning

Hui Guan, Lin Ning, Xipeng Shen
North Carolina State University
Raleigh, NC, USA
{hguan2, lning, xshen5}@ncsu.edu

Seung-Hwan Lim
Oak Ridge National Laboratory
Oak Ridge, TN, USA
lims1@ornl.gov

Timothy Menzies
North Carolina State University
Raleigh, NC, USA
tjmenzie@ncsu.edu

***Abstract*—Convolutional neural network (CNN) pruning is an important way to adapt a large CNN model attained on general datasets to a more specialized task. It is time-consuming, for its needs for exploring many possible pruned variants and for the long time needed to train and test each pruned variant. *Block-wise pruning* is an idea to accelerate the process: If each block of layers in a CNN can be pruned and trained well, putting them together could compose many almost-ready variants of pruned CNN and hence save much time for training pruned CNN. Materializing the idea, however, faces a major barrier: how to effectively train an arbitrary block of CNN layers independently. This paper proposes to solve the problem through Generalized Teacher-Student Network (GTSN), a novel technique designed for flexibly passing the knowledge of a large CNN to a pruned CNN at an individual layer level. It uncovers a series of insights on the effects of GTSN on CNN pruning, and integrates them into a block-wise CNN pruning framework, which speeds up CNN pruning by up to 202.8X for ResNet-50 and up to 30.2X for Inception-V3.**

***Index Terms*—CNN, network pruning, teacher-student network**

I. Introduction

Recent years have witnessed some rapid progress in the development of Convolutional Neural Networks (CNN), and its successful applications in various data mining tasks.

Large CNN models can achieve high accuracy but at the cost of immense computation and memory requirements. CNN pruning [1]–[3] is a popular method to reduce the size and complexity of a CNN by removing some parameters (e.g. weights or filters) in the network and then retraining the pruned network. It is especially useful for adapting a large CNN model (e.g., a general image classifier) attained on general datasets to meet the needs of more specialized tasks [4], [5] (e.g., recognition of car models, dog breeds, bird species [3], [5]–[7]). Compared to designing a CNN from scratch for each specific task, CNN pruning is an easier and more effective way to achieve a high-quality network [3], [4], [7]–[9]; it is also an essential method for fitting a CNN model on a device with limited storage or computing power [10], [11].

CNN pruning is however extremely time-consuming. To identify the best-pruned network, a practitioner usually needs to retrain many promising networks pruned with different configurations and test them out. As training one CNN already takes hours or days and the configuration space is combinatorial, it is a long process, delaying the turnaround time of AI products development, especially given that the development of an AI product often involves continuous changes in design and hence repeated runs of the CNN pruning process.

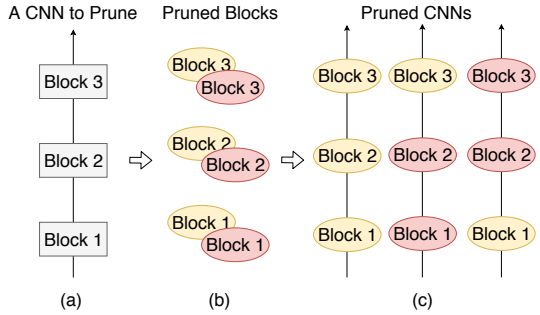


Fig. 1: Illustration of block-wise CNN Pruning. Different colors of pruned blocks correspond to different pruning options.

An idea we propose to accelerate the process is *block-wise CNN pruning*. It comes from the following observation. A CNN can be regarded as a composing result of a set of building blocks, as Figure 1(a) illustrates. Here, a *block* is a sequence of CNN layers. As Figure 1(b,c) depicts, a pruned CNN can be regarded as the result of assembling some pruned variants of the blocks together. A variant of a block may appear in many pruned CNNs. In current CNN pruning, every pruned CNN is trained from scratch, even though some of its building blocks may have already been trained as part of another pruned CNN. The basic idea of *block-wise CNN pruning* is to train each pruned block individually such that the pruned networks can reuse these pre-trained blocks and hence forego a large part of their training process.

Although the idea is intuitive, it has not been explored in prior work. A plausible reason is the lack of effective ways to train an individual block in a CNN. Because a block may appear at an arbitrary point in a CNN, there is typically no ground truth for its output (unless the block includes the top layer of the CNN), which makes standard CNN training hard to apply. Moreover, as training the building blocks is part of the whole pruning process, there is a high requirement on the speed and concurrency (or parallelism) of the training method.

The goal of this work is to remove that key barrier, and establish *block-wise CNN pruning* as a fast reliable method to prune CNN. At the center of our solution is a new concept called Generalized Teacher-Student Network (GTSN).

GTSN uses the original full network as a teacher to train

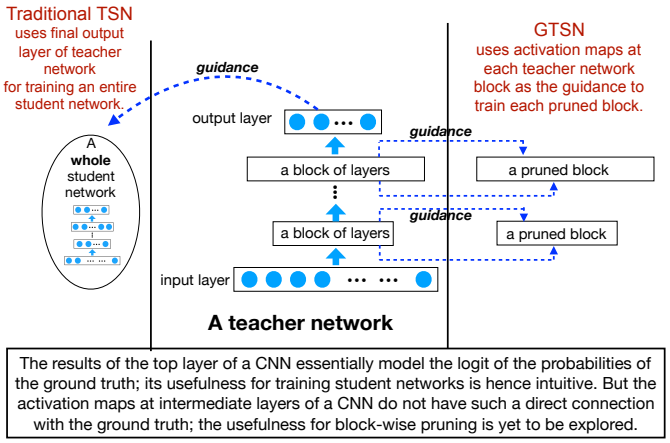


Fig. 2: Illustration of the high-level difference between GTSN and traditional TSN.

each pruned CNN block. As the right side in Figure 2 illustrates, GTSN runs the teacher network and the training of CNN blocks together; it forwards the layer-wise input and output activation maps from the teacher network to the CNN blocks to concurrently train the many blocks at the same time.

The spirit of leveraging the original network as a teacher network in GTSN is inspired by Teacher-Student Network (TSN). GTSN departs from TSN on some important aspects:

(1) As Figure 2 shows, TSN uses the *final output layer results* of the teacher network as the guidance for training a student network, whereas GTSN uses the *activation maps at each teacher network block* as the guidance;

(2) TSN passes the guidance to the student networks from the top to bottom *sequentially*, whereas GTSN allows the student network blocks to *concurrently* leverage the activation maps of the teacher blocks as soon as they are ready;

(3) TSN trains the *entire student network*, whereas GTSN trains *each block*. The result of a block trained by TSN is strongly affected by what configurations the other blocks of the student network have, and is hence hard to be reused in another pruned network. In contrast, in GTSN, the trainings of different building blocks are independent, which makes the trained blocks ready for reuse.

These distinctive features are essential for GTSN to address the needs of *block-wise CNN pruning*. They make block training fast and highly parallelizable, and at the same time, produce trained blocks amenable for composition and reuse.

But for GTSN to serve as an effective solution for *block-wise CNN pruning*, a key question must be answered first:

> Can GTSN produce blocks that, when put together, give a pruned network of good quality?

Even though TSN has shown successes in prior work for whole network training [12]–[18], the successes do not immediately entail effectiveness of GTSN: The results of the top layer of a CNN essentially model the logit of the probabilities of the ground truth; its usefulness for training student networks is hence intuitive; but the output of intermediate layers of a CNN does not have such a direct connection with the ground truth. There are hence open questions how useful such information can be for training CNN blocks, and how much time can the trained blocks help save CNN pruning time.

This paper presents two-fold empirical explorations to answer these questions. Through a series of experiments, Section III systematically examines the effects of using activation maps as the guidance for training CNN blocks, the influence of block sizes on the effects, and the importance of some conditions (top-layer freezing) for GTSN to work effectively. It produces four observations, which confirm the positive effects of GTSN on block training, and reveal several insights on the properties of GTSN and how it should be used effectively for *block-wise CNN pruning*.

Section IV examines the ultimate benefits that GTSN can bring to CNN pruning. Analytically, it discusses the aspects on which GTSN helps CNN pruning and presents several formulas for quantitative speedup analysis. Empirically, it presents the implementation of a GTSN-based block-wise CNN pruning system on top of TensorFlow [19] and measures the actual speedups on ResNet-50 and Inception-V3. The pruning system first uses GTSN to train each pruned block, then assembles them together to form some pruned CNNs, and finally runs a short fine-tuning process on the resulting CNNs to identify the best-pruned network. GTSN is applied to train pruned blocks so that all the CNNs that include those trained blocks could start with a better initialization and achieve a given accuracy sooner. For ResNet-50 and Inception-V3, GTSN-based CNN pruning shortens the pruning process by up to 202.8X and 30.2X respectively.

Overall, this work makes the following major contributions:

- It introduces GTSN as a method for efficiently training individual CNN blocks.
- It empirically confirms the usefulness of activation maps of teacher networks for training pruned CNN blocks, laying the foundation for GTSN.
- It presents the design and implementation of the first *block-wise CNN pruning*.
- It provides analytical results on the benefits of GTSN-based CNN pruning, as well as empirical results confirming the orders of magnitude speedups the new method produces for CNN pruning.

II. Related Work

A. Teacher-Student Network

The concept of using TSN for model compression has been around for a while. The basic idea is to pass some dark knowledge contained in a teacher network to the student network often through a certain type of regularization. Early work done by Bucila et al. [20] shows that one can compress the information in an ensemble into a single network without significant loss in performance by training the network on ensemble-labeled data set. Ba et al. [14] extended this work to enable training of a shallower student network via regressing logits with ℓ_2 loss. Hinton et al. [12] proposed knowledge distillation as a more general case of [14], where a student network is trained by the softmax values of the output from a teacher network. Besides the output layer results, some work explores amending them with extra information, such as flow

of solution procedure matrix, which is calculated by computing the inner product between activation maps from two layers [16], attention maps [15], and middle layer results [13]. Chen et al. [18] adapted knowledge distillation [12] and hint learning [13] to train compact object detection networks. Ashok et al. [17] proposed a reinforcement learning approach to identify a compact student architecture, employing a TSN training method similar to what Ba et al. [14] used.

All these prior studies, including those with amended information, rely on the usefulness of the output layer results of teacher networks for training student networks. One earlier work [13] uses intermediate layer output rather than the output layer results. However, the focus of that work is pre-training the bottom part of a deep network rather than systematically explore the effects of intermediate layer outputs on CNN block training. The work limits TSN-like training of intermediate layers only to the first several layers of a CNN, and still resorts to the output layers of the teacher network for the training of the student network. So it answers none of the key questions this current paper attempts to answer for GTSN.

B. Network Pruning

Recent studies on network pruning have focused on filter-level pruning, which removes a set of unimportant filters from each convolutional layer. The problem is challenging because the configuration space is combinatorial in the number of filters in a CNN model. Most prior work has been focused on reducing the search space, by designs of heuristic criteria to effectively determine the importance of a filter [2], [3], [5], [6], [21], [22], or by reinforcement learning-based search algorithms to efficiently sample the design space [17], [23]. Although these methods help heuristically narrow the search space, in practice, to prevent missing good configurations, practitioners often still end up with training and evaluating a large remaining configuration space, suffering from a long exploration process.

III. GTSN for CNNs

In this section, we focus on GTSN, its effects on CNN block training, and its properties. We leave the enabled *block-wise CNN pruning* and the yielded large time savings to later sections.

We start this section with a formal description of the existing TSN training scheme. It provides the necessary background knowledge for the rest of the paper. We then describe the series of experiments we have designed for examining the effects of GTSN. After that, We report the experimental results, list several important observations, and discuss effective deployment of GTSN for CNN pruning.

A. TSN Background

CNN is a sequence of convolutional layers and every layer transforms one volume of activation maps to another. Other kinds of layers such as pooling layer or fully-connected layer may be also included. The number of activation maps from a convolutional layer is the same as the number of filters in that layer. The results from the output layer of a CNN are called *logits*, which is un-normalized log probability values.

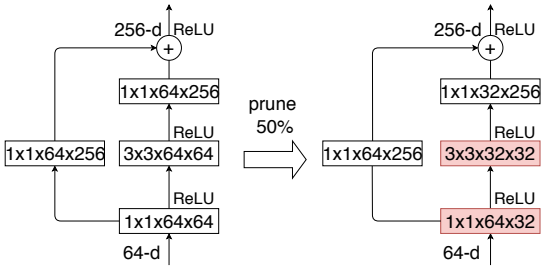


Fig. 3: Illustration of pruning a residual module with $kp = 0.5$. We only prune the first two convolutional layers so that the block output dimension does not change.

When a CNN is trained, for example, for a classification task, the logits from the output layer is typically normalized with softmax function and then used for calculating cross entropy loss. Some regularizations on CNN parameters are commonly added to avoid over-fitting and improve training quality.

Previous studies on TSN usually use the dark knowledge-based loss as a type of regularization during the training of a student network. Let W be the parameters of a student network, $\{(x^i, y^i)\}_{i=1}^N$ be the training data samples and $f(x; W)$ be the results of the output layer (i.e. logits) on the training data. The loss function in TSN is formulated as:

$$\mathcal{L}(W) = \mathcal{L}_{hard}(f(x; W), y) + \lambda * \mathcal{L}_{KD}, \tag{1}$$

where $\mathcal{L}_{hard}$ is the cross entropy loss function used for training with hard ground truth labels, λ is a hyper-parameter, and $\mathcal{L}_{KD}$ is the dark knowledge-based loss. The definition of $\mathcal{L}_{KD}$ varies in previous works [12]–[18]. Despite of various definitions of dark knowledge, the formulation of the loss function in TSN as Eq. 1 requires the results from the output layer of both the teacher network and the student network (i.e. logits and cross entropy).

B. Designed Experiments

As Section I mentions, one of the key questions on GTSN is whether activation maps at intermediate layers of a teacher network can serve as effective guidance for training pruned CNN blocks. we design two experiments to examine it: 1) training a single convolutional layer in a student network; and 2) training each block in a student network. We first explain how student networks are created.

Student Networks Creation: In our experiments, we use well-trained CNN models as teacher networks. We create student networks through network pruning.

Network pruning reduces the number of parameters in a CNN by keeping only a subset of parameters. We focus on filter pruning since it is a popular approach to reduce model size in a structured way [24]. The goal of filter pruning is to remove the least important filters of a DNN as many as possible while keeping the accuracy of the network still meeting some requirement. Many pruning criteria [2], [3], [5], [6], [21], [22], [25] have been proposed to evaluate the importance of a filter. We use the ℓ_1 norm of a filter proposed by [2] for its simplicity and effectiveness. After pruning, the

pruned network inherits the value of remaining parameters and is retrained to recover the accuracy.

When a well-trained CNN is pruned to produce a student network, each convolution layer in the network is pruned by keeping only kp fraction of the most important filters, $kp \in (0, 1]$. $kp = 1$ means that the layer is not pruned at all. We call kp the *keeping rate*. Unless noted otherwise, the default value of kp is set to 0.5 in the experiments. An illustration of pruning a residual block with $kp = 0.5$ is shown in Figure 3.

We next describe the designs of the two experiments for examining the usefulness of activation maps of teacher layers for block training of student networks.

1) Activation-Maps–Based Training on a Single Layer: This experiment uses activation-maps of a teacher network to train a single convolution layer in a CNN, and studies its effects on the quality of the entire network. A positive result would indicate the usability of activation maps in training an arbitrary layer of CNNs.

When some filters at an arbitrary layer is pruned, that layer produces fewer activation maps, causing a mismatch with the numbers of activation maps by that layer in the teacher network. To go around the problem for training, we use the minimization of the ℓ_2 loss of the output activation maps from the next convolutional layer to train this layer.

Mathematically, let $\mathcal{O}_{l+1}$ and $\mathcal{O}'_{l+1}$ be the vectorized outputs activation maps from the $l+1$-th convolutional layer before and after the l-th layer is pruned respectively. Let W_l be the parameters in the l-th convolutional layer. The layer-wise activation map reconstruction loss is:

$$\mathcal{L}_l(W_l, W_{l+1}) = \frac{1}{|\mathcal{O}_{l+1}|}\|\mathcal{O}_{l+1} - \mathcal{O}'_{l+1}\|_2^2. \quad (2)$$

Let $\mathcal{I}_l$ be the inputs activation maps to the i-th convolutional layer. The training data samples are $\{(\mathcal{I}_l^i, \mathcal{O}_{l+1}^i)\}_{i=1}^N$. The difference between Eq. 1 and Eq. 2 is that the former one trains all the parameters in a student network while the latter one trains only parameters in the l-th and $l+1$-th convolutional layers.

2) Activation-Maps–Based TSN on Every Block: Our second experiment studies what happens to the quality of the pruned network if we train *every* pruned block in the CNN independently. We call it *block-wise training*.

A CNN can be treated as a composing result of a set of blocks; each block represents a sequence of convolutional layers. The definition of a block depends on the network structure. CNN architectures largely fall into two broad groups: the traditional models represented by AlexNet [26] or VGGNet [27], and recent structural variants like GoogLeNet [28] and ResNet [29]. For the traditional ones, as the basic building units are mostly convolutional layers, we define a block as a sequence of two or more convolutional layers. For the structural variants, as they adopt some novel network structures like Inception modules in GoogLeNet or residual modules in ResNet, which naturally form a block, we define a block as a sequence of one or more modules. The size of a block refers to the number of basic building units it has. The smallest size of block for traditional models are two and for the structural models are one. We use the smallest block size by default in the experiments unless noted differently.

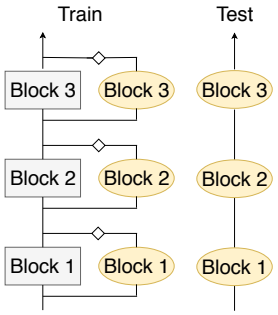


Fig. 4: Illustration of block-wise training. Eclipses are pruned blocks; rectangles are original blocks; diamonds refer to the activation map reconstruction error to minimize. During training, each pruned block is trained with the inputs and outputs activation maps from the original block; during testing, the network composed of the pruned blocks is tested.

Experiments on block-wise training prune each block in a CNN by removing $(1 - kp)$ fraction of filters in the block and trains each pruned block individually using activation-maps–based TSN. Figure 4 illustrates the block-wise training for a student network. This experiment helps examine the ultimate effects of activation-maps–based TSN if it is used for searching for the best-pruned network, as every block of such a network is subject to pruning. In addition, this experiment further examines how the effects of TSN change when the size or other configurations (esp. the top layer) of a block changes.

These experiments together give a comprehensive examination of activation-maps–based block training, and provide four major observations on its effects, which will be described in Section III-D.

C. Experiment Settings

Our experiments use five popular CNN models: VGG-16 [27], ResNet-50/ResNet-101 [29] and Inception-V2/Inception-V3 [30], [31]. The five models are pre-trained on ILSVRC 2012 [32], a large comprehensive image dataset consisting of millions of images.

The goal of CNN pruning is to build more compact CNN models for *fine-grained recognition*—that is, to recognize different subcategories within one kind of objects, such as different types of cars or birds. Fine-grained recognition is a typical usage scenario of CNN pruning [3]. The datasets used for student networks are Flowers102 [33], CUB200 [34], Cars [35], and Dogs [36]. These datasets are commonly used in fine-grained recognition [3], [37]–[40]. Table I reports the statistics of the four datasets, including the size of training dataset (*Train*), the size of testing dataset (*Test*), and the number of classes (*Classes*). For all experiments, network training is performed on the training sets while accuracy results are reported on the testing sets.

To get well-trained models (i.e. teacher network) on the four datasets, we trained the five models for 40,000 steps with batch size 32, learning rate 0.001, weight decay 0.00004 for each of

TABLE I: Dataset statistics.

Dataset	Size			Classes	Accuracy				
	Total	Training	Testing		ResNet-50	ResNet-101	Inception-v2	Inception-V3	VGG-16
Flowers102	8,189	6,149	2,040	102	0.973	0.975	0.972	0.968	0.966
CUB200	11,788	5,994	5,794	200	0.770	0.789	0.746	0.760	0.739
Cars	16,185	8,144	8,041	196	0.822	0.845	0.789	0.801	0.832
Dogs	20,580	12,000	8,580	120	0.850	0.864	0.841	0.835	0.808

the datasets. Additional annotations including bounding boxes and part labels are not used in the training. The accuracy of the trained models on the test datasets are listed in the column *Accuracy* in Table I. A pruned network is trained for 30,000 steps with batch size 32. The training time for recovering the accuracy of a pruned network is usually smaller than the training time for adapting a pre-trained model to new datasets due to weight inheritance in network pruning, hence the fewer number of training steps.

All the experiments are performed using TensorFlow 1.3.0 on Titan[1]. 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.

D. Observations

In this part, we report the experimental results. We summarize them into four observations. Observations 1-2 confirm the positive effects of activation-maps–based block training. Observations 3-4 unveil the effects of the size of blocks on block training, and the importance of freezing the top layer in a block in this context.

Observation 1. *Activation maps from teacher networks are helpful for training an arbitrary convolutional layer in a pruned network.*

We attain this observation through the experiment on applying activation maps as the guidance to train a single layer (Section III-B1) in student networks. In the experiment, we pruned only one convolutional layer by removing 50% of least important filters and trained the pruned layer (with the activation map loss function of its next layer as defined in Eq. 2.) Even though only two convolutional layers are involved in the training, we observe that the pruned network benefits from the training process.

Figure 5 shows the accuracy curves of training convolutional layers in VGG-16 on Flowers102 and CUB200. The curves in each graph correspond to the experiments in which one of the CNN layers is pruned and trained with the activation maps from the teacher network. The accuracies are those of the entire network after that layer is trained for a number of steps. The results demonstrate that minimizing the ℓ_2 loss of the output activation maps of a single pruned layer can directly yield accuracy improvement of the entire student network.

The next question is what happens to the quality of the pruned network if not one but every block in a CNN is pruned and trained independently. Our second experiment of block-wise training gives the following observation:

Observation 2. *Activation maps of teacher networks are useful for block-wise training to improve the quality of a pruned CNN.*

The design of the experiment has been described in Section III-B2. Three different student networks from each teacher network are derived through block-wise pruning, with the keeping rate (kp) being 0.3, 0.5, and 0.7 respectively. Each pruned block is trained with the corresponding activation maps in the teacher network; the accuracy of student networks is measured after training each block for a certain number of steps (without whole network-fine tuning).

Figure 6 shows the accuracy curves of block-wise training (shown as *block-wise*) when the pruning happens to each basic block (e.g., a single residual module in ResNet). It indicates that although the rising rates of the curves vary across kp values and networks, the block-wise training consistently improves the quality of the pruned student network. Similar positive results are observed on other models and datasets. It shows the potential of GTSN using activation maps to guide block-wise CNN pruning (elaborated in Section IV).

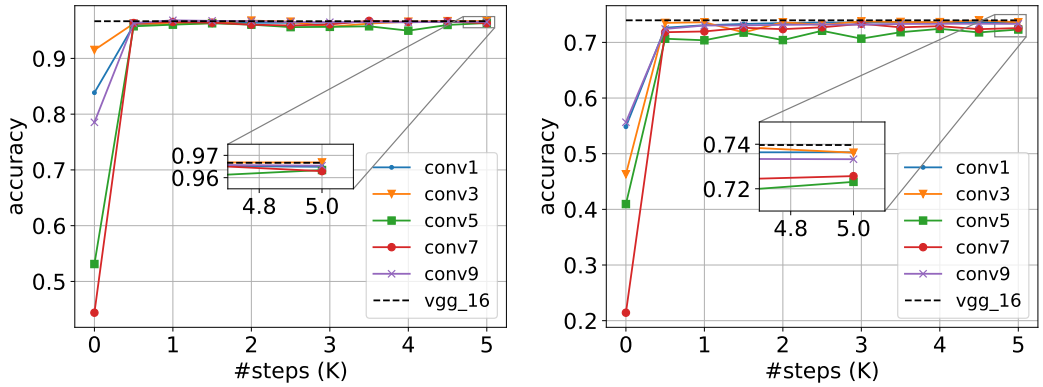
Figure 6 also shows the accuracy curves of training the pruned networks using the standard cross-entropy loss (shown as *default*). Compared with block-wise training, training the entire pruned networks instead of each pruned block individually yields higher accuracies. One of the reasons is that in block-wise training, the inputs to each pruned block in a student network come from the original blocks in a teacher network during training, but during testing, they come from the pruned blocks in the student network, as shown in Figure 4. Despite such a gap, the results do show that networks assembled from blocks trained with activation maps consistently show a much better accuracy than before the block-wise training. Section V will show that the gap disappears quickly (and even reverses) after a short fine-tuning process.

We further study the influence of block size on the effects of activation-map–based block-wise training, and get the following observation:

Observation 3. *In block-wise activation-map–based training, a larger block size improves the training quality.*

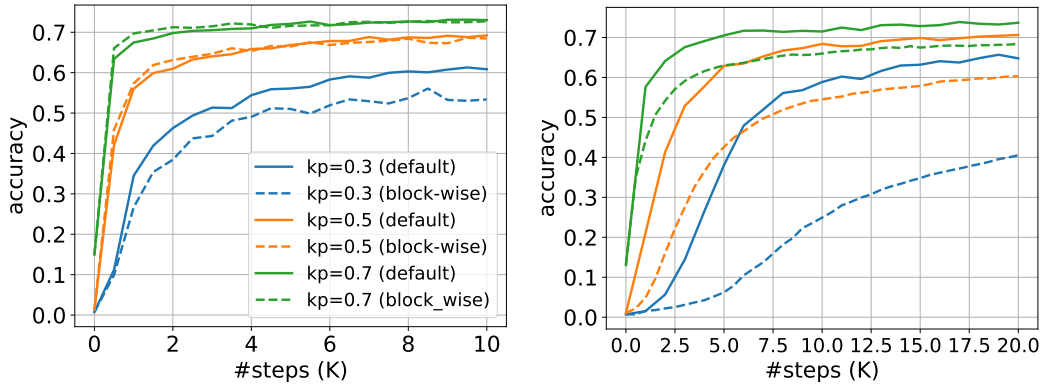
The observation comes from an exploration in which we vary the size of blocks in the block-wise training experiment. Student networks are derived by pruning 50% of least important filters from each block. The numbers of basic building units in ResNet-50 and Inception-V3 are 16 and 11 respectively. Thus we set block size $bs \in \{1, 2, 4, 8, 16\}$ for ResNet-50 and $bs \in \{1, 2, 4, 8, 11\}$ for Inception-V3. Figure

[1]https://www.olcf.ornl.gov/titan/



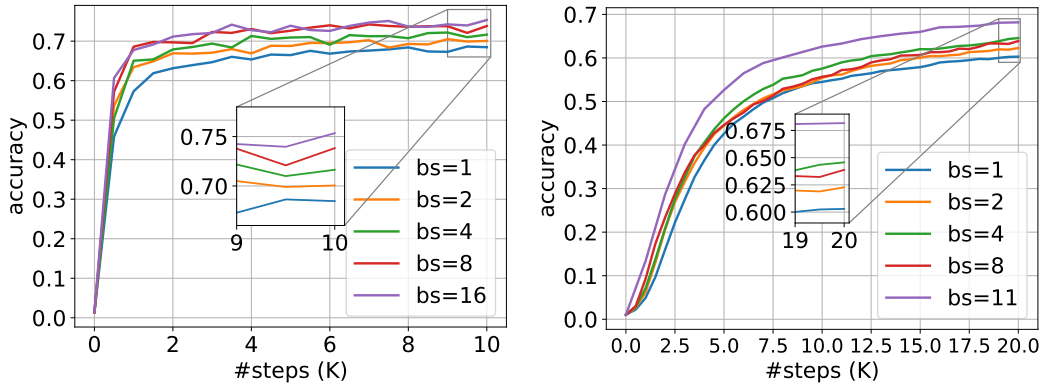
(a) Flowers102 (b) CUB200

Fig. 5: Accuracy curves of training convolutional layers in VGG-16 using layer-wise activation map reconstruction loss on two datasets. Black dashed lines show the accuracies of the original VGG-16 model.



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

Fig. 6: Accuracy curves of block-wise training (*block-wise*) and training using the standard cross-entropy loss (*default*) on CUB200.



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

Fig. 7: Accuracy curves of block-wise training for student networks on CUB200 with different size of blocks.

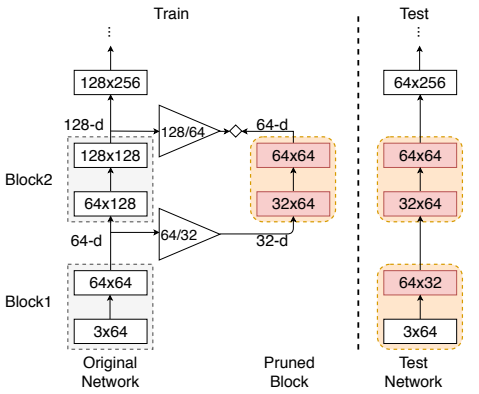


Fig. 8: Illustration of training a pruned block whose last convolutional layer is pruned. Block size is two. Dotted rectangles are blocks while rounded doted ones are pruned blocks; rectangles are convolutional layers; triangles are selectors to down-sample the activation maps from the original block; diamonds refer to the activation map reconstruction error.

7 shows the accuracy curves of two student networks on CUB200 with different block sizes, indicating the trend that a larger block size improves the training quality. A similar trend is observed on other networks and datasets.

In the above experiments, we do not prune the top layer in a block to ensure that the dimension of output activation maps from a pruned block is the same as that from the original block. To check if the constraint can be removed, we prune all the convolutional layers in a block and check the accuracy of the network when the block is trained. We get the following observation:

Observation 4. *It is important to freeze the top layer (e.g., keep it unpruned) in a block for benefiting from block-wise activation-map–based training.*

Specifically, in this experiment, we derived each student network by pruning only one block from VGG-16. We define each block as a sequence of two convolutional layers and thus have totally eight student networks (VGG-16 contains 16 convolutional/fully-connected layers). In block-wise training, only the pruned block in each student network needs to be trained. When the top layer in a block is pruned, the number of output activation maps from the original block and the pruned block does not match. To address the problem, we down-sample the output activation maps from the original block. As each filter in a convolutional layer produces one activation map, we remove activation maps that are produced by the least important filters (per a criterion proposed before [2]). Figure 8 illustrates the training and testing of a pruned block whose last convolutional layer is also pruned.

Figure 9 shows the accuracy curves of these student networks. Compared to Figure 5, the curves rise much slower. The contrast suggests that the transferred dark knowledge, which is the down-sampled outputs from a block in a teacher network, is not as beneficial as activation maps from a block with the top layer unpruned.

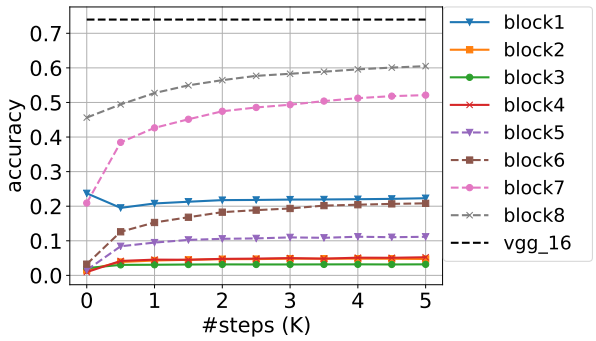


Fig. 9: Accuracy curves of eight student networks with only one block pruned. The last layer in the block is also pruned. The legend *blockx* represents a student network with the x-th block pruned with $kp = 0.5$. Block size is two. Black dashed lines show the accuracies of the original VGG-16 model.(VGG-16, CUB200).

E. Discussions and GTSN for CNNs

The first two observations confirm the value of teacher activation maps in training CNNs at two different levels: an arbitrary layer by the first observation, and every block by the second observation. Together, they confirm the promise of GTSN for training blocks in a pruned CNN. The use of activation maps by GTSN grants it a much larger flexibility than existing TSN as activation maps are available at every convolution layer. The third observation provides insights on the effects of the size of blocks. The fourth observation offers a guideline for effective use of GTSN at block level. These observations provide the foundation for GTSN to effectively support *block-wise CNN pruning*, as described next.

IV. GTSN-based Block-Wise CNN Pruning

This section presents the implementation of *block-wise CNN pruning*, which is powered by GTSN based on the observations made in the previous section. It then analyzes the speed benefits of this new way of CNN pruning compared to the default method.

A. Connections between GTSN and CNN Pruning

CNN pruning is a combinatorial optimization problem. Given that each block could be pruned with m possible options, for a CNN with B basic building blocks, there would be m^B candidate pruned CNNs to choose from. Prior studies have focused on reducing the search space to a set of promising configurations through various heuristic methods [2], [6], [23]. Even though they can often reduce the search space significantly, the assessment of the remaining networks still takes a long time as it requires the training and testing of each of those networks; training one CNN often takes hours.

Mathematically, let C be the collection of *candidate pruned CNNs* selected using any existing method and T_i be the time spent on training the i-th pruned network in the collection C. The total training time required for identifying the best-pruned network is

$$T = \sum_{i=1}^{|C|} T_i. \tag{3}$$

GTSN provides new opportunities for accelerating the assessment of the remaining network configurations. Specifically, Observation 2 in the previous section shows that individually GTSN-trained blocks help improve the quality of the overall network. So if we use GTSN to train a number of pruned blocks, we could improve the quality of all networks containing those blocks. The one-time block-level training result can benefit many pruned networks. As this process puts these networks into a good initial state, they would need only a short fine-tuning to reach a certain accuracy. The total training time of the candidate pruned CNNs can hence be reduced.

B. Mechanism

We materialize the idea by developing a GTSN-based block-wise CNN pruning framework. It is based on TensorFlow version 1.3.0 [19]. It first trains each pruned block, then assembles them together to form CNNs of each candidate configuration, runs a short fine-tuning process on the resulting CNNs, and tests these CNNs to identify the best. The criterion of the best network is the smallest network that reaches a predefined accuracy threshold.

In GTSN-based block-wise CNN pruning, the training of the collection of candidate pruned CNNs contains mainly two stages: *GTSN-based block training* and *network fine-tuning*. As there are m possible options and B basic building blocks, the total number of pruned blocks to train is at most $K = m \times B$. Let $T_k^{(1)}$ be the time spent on training the k-th pruned block and $T_i^{(2)}$ be the time spent on fine-tuning the i-th pruned networks, the total training time of the candidate pruned CNNs becomes:

$$T_{GTSN} = \sum_{k=1}^{K} T_k^{(1)} + \sum_{i=1}^{|C|} T_i^{(2)}, \tag{4}$$

where the first item corresponds to the time for GTSN-based block training and the second item corresponds to the time for fine-tuning each pruned network. According to the experiments in Section V, the time for fine-tuning each pruned network to a given accuracy threshold is largely reduced due to the GTSN-based block training (i.e., $T_i^{(2)} << T_i$), resulting in significant speedups in CNN pruning (i.e., $T_{GTSN} << T$). We next describe the two stages in detail.

GTSN-based block training Figure 10 illustrates how the framework enables GTSN-based training of CNN blocks. It is an online mechanism; the teacher network and a number of pruned blocks are put into a single TensorFlow graph. During GTSN-based block training, the teacher network runs and its activation maps directly flow into the pruned blocks to facilitate their training. Such an online mechanism avoids saving the activation maps which often takes too much space.

In our experiments, we use the basic CNN blocks (i.e. an Inception/residual module) of ResNet and Inception as the blocks for GTSN training. Based on Observation 4 in the previous section, we keep the top layer of a block unpruned.

Network Fine-Tuning The GTSN-based block training stage outputs a bag of pre-trained pruned blocks. At the beginning of the fine-tuning phase is an initialization step that uses the value of parameters in the pre-trained pruned blocks to initialize the blocks in each of the candidate pruned networks. We call the resulting network a *block-trained network* and the network before such an initialization step a *default network*.

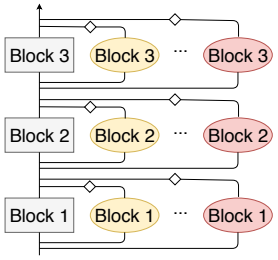


Fig. 10: The online GTSN-based block training in our CNN pruning framework. Eclipses are pruned blocks.

After the initialization, the standard CNN training runs on the block-trained networks. Compared with training a default pruned network, fine-tuning a block-trained network usually takes much less training time to reach a predefined accuracy threshold as the network starts with a much better initialization. In other words, given the same amount of training time, a block-trained network could converge to a higher accuracy compared with a default network.

Figure 11 shows the accuracy curves of two versions of a ResNet-50 network and an Inception-V3 network on CUB200. The *default* network is obtained by pruning each block by 50% and the *block-trained* network is obtained by training the *default* network with GTSN-based training. As block training improves the initial accuracy of the default network, the fine-tuning yields a much better accuracy than training the *default* network with the same loss function after 30,000 training steps.

C. Speedup Analysis

In this part, we theoretically analyze the potential speedup that GTSN-based block-wise CNN pruning may generate. The next section will provide empirical measurement results.

The discussion uses the following denotations:

C: the set of candidate pruned networks to explore;
m: the number of possible values of kp;
s: the number of training steps;
p: the amount of computations in a training step.

If s, p don't carry superscripts, they are about the training of a pruned CNN in the default case. If they carry superscripts, such as $s^{(1)}$ or $p^{(2)}$, they are about the training in the block-wise training phase of GTSN (superscript (1)) or the fine-tuning phase of GTSN (superscript (2)).

Then in Eq 3, the time spent on training the i-th pruned network in the collection C is proportional to the product of the number of training steps and the amount of computations, $T_i \propto s_i \times p_i$. Similarly, $T_i^{(2)} \propto s_i^{(2)} \times p_i$ in Eq 4.

GTSN-based block training involves the training of K pruned blocks. Because all blocks of the same keeping rate are trained together, the time of the entire block-wise training phase is proportional to $s^{(1)} \times \sum_{j=1}^{m} p_j^{(1)}$, where m is the number of options for the keeping rate kp.

The speedup brought by GTSN-based block-wise CNN pruning can be calculated as follows ($E[x]$: the average value of x):

$$\begin{aligned}\frac{T}{T_{GTSN}} &= \frac{\sum_{i=1}^{|C|} T_i}{\sum_{k=1}^{K} T_k^{(1)} + \sum_{i=1}^{|C|} T_i^{(2)}} \\ &= \frac{\sum_{i=1}^{|C|} s_i \times p_i}{s^{(1)} \times \sum_{j=1}^{m} p_j^{(1)} + \sum_{i=1}^{|C|} s_i^{(2)} \times p_i} \\ &= \frac{|C| \times E[s] \times E[p]}{s^{(1)} \times m \times E[p^{(1)}] + |C| \times E[s^{(2)}] \times E[p]} \\ &= \frac{|C| \times E[s]}{s^{(1)} \times m + |C| \times E[s^{(2)}]} \\ &= \frac{E[s]}{s^{(1)} \times \frac{m}{|C|} + E[s^{(2)}]}. \end{aligned} \tag{5}$$

The two parts of the denominator both tend to be much smaller than the numerator, hence the speedups. The first part, $s^{(1)} \times \frac{m}{|C|}$, is small as $m << |C|$. The second part, $E[s^{(2)}]$, is typically much smaller than $E[s]$, for two reasons. First, according to Observation 2 in the previous section, the block-wise training in GTSN already prepares the networks with a much better initial accuracy. Second, as we will see in Section V and Figure 11, the fine tuning phase of GTSN yields accuracies higher than what the default training of a pruned CNN gives. Together, they entail that it takes GTSN fine-tuning phase a much shorter time to reach a target accuracy.

Following Eq. 5, the upper-bound of the speedup can be calculated as

$$\min\{\gamma \frac{|C|}{m}, \frac{E[s]}{E[s^{(2)}]}\}, \tag{6}$$

where, $\gamma = \frac{E[s]}{s^{(1)}}$. The first bound, $\gamma \frac{|C|}{m}$, can be reached if GTSN-based block training is powerful enough to make pruned networks reach a target accuracy even without fine-tuning. In this case, a larger number of configurations entails a larger speedup. The second bound, $\frac{E[s]}{E[s^{(2)}]}$, corresponds to the case where the block training time is negligible. The bound is the ratio between the expected number of training steps in the baseline approach and the expected number of fine-tuning steps in the GTSN-based approach. We next present empirical measurements on the speedups achieved in practice.

V. Evaluation of Block-Wise CNN Pruning

This section examines the ultimate benefits of block-wise CNN pruning. A brief summary is that the new training method, with block training and all other overhead counted, consistently speeds up the default CNN pruning by a factor of one or two magnitudes.

Specifically, we evaluate the GTSN-based block-wise CNN pruning framework in two settings. The first one examines the basic time savings that the framework brings to the evaluation of a set of pruned networks. The second one measures the speedups it brings to end-to-end CNN pruning in a more comprehensive manner, with more factors considered, including the effects of GTSN on the accuracy of the trained networks and parallel pruning. We first describe the experiment settings in Section V-A and then report our experiment results in Sections V-B and V-C.

A. Experiment Settings

In both experiments, we use ResNet-50 and Inception-V3 as the CNN models for pruning. Details on the machines, datasets, and how to adapt the well-trained models to these datasets are the same as Section III-C. After adaption, we have 8 trained full CNN models. The dataset statistics and accuracy of the trained ResNet-50 and Inception-V3 are listed in Table I.

Candidate Pruned CNNs: There are many ways to generate a collection of candidate pruned CNNs. As that is orthogonal to the focus of this work, to avoid bias from that factor, our experiments generate a set of promising configurations through random sampling, which has shown to be an effective way to provide a strong baseline for hyperparameter optimization [41]. The set of pruning options for each block are $kp \in \{0.3, 0.5, 0.7\}$. The numbers of blocks in ResNet-50 and Inception-V3 are 16 and 11 respectively, making the search space as large as 3^{16} and 3^{11}. When a model is pruned based on a configuration, a candidate pruned network with fewer filters is created by inheriting the remaining parameters of the model, which is the same way as creating student networks in Section III-B.

Baseline for Comparison and Our Approach: The baseline approach trains candidate networks pruned according to each configuration for 30,000 steps with batch size 32, weight decay 0.00001, and fixed learning rate 0.001 (we tried other learning rates including the scheme with decays, but found 0.001 overall best for the baseline approach.) GTSN-based CNN pruning first trains all the pruned blocks, initializes each candidate pruned network with values of parameters from trained blocks, 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. GTSN-based block training takes 10,000 steps for ResNet-50 (batch size=32, learning rate=0.2, weight decay=0.0001), and takes 20,000 steps for Inception-V3 (batch size=32, learning rate=0.08, weight decay=0.0001). The fine-tuning phase uses the same hyper-parameters as the baseline approach does.

B. Speedup on a set of candidate pruned networks

This experiment compares the time taken by the GTSN-based CNN pruning framework to train a collection of pruned networks (T_{GTSN} in Eq. 4) with the time by the default cross-entropy training on the same set of networks (T in Eq. 3).

We test four sizes of collections, $|C| = 4, 16, 64, 256$. For each size, we repeat the experiments five times, with a random collection of pruned networks created each time. The training of a pruned network stops once a target accuracy is met. The target accuracy is set to $min(a_i)$, where a_i is the accuracy the i-th network in C reaches at the end of the default training. Setting such a target ensures that the default trained networks

can all hit the target, making the performance comparison meaningful.

Table II reports the average training times and speedups achieved by GTSN-based CNN pruning. The time of GTSN-based CNN pruning includes the pre-training time of blocks, which is listed in the column *block*. GTSN helps reduce the training time up to 20.8X for ResNet-50 and 4.4X for Inception-V3. The speedups are larger as the collection of pruned networks increases. It is because the time to train the blocks weighs less as the total time increases for a larger collection. Overall, the block-training time is a small portion of the entire training time in all settings.

C. Speedup in parallel CNN pruning with early stop

This experiment measures the comprehensive effects of GTSN on end-to-end CNN pruning. Comparing to the basic measurement reported in the previous subsection, this study considers two extra factors that are common in CNN pruning. The first is *early stop*. Because the objective of the pruning is to find the smallest network that meets a predefined accuracy target, a strategy for efficiency is to evaluate the networks from the smallest to the largest, and stop as soon as it finds a network that meets the accuracy target. This *early stop* strategy can avoid evaluating some large networks and hence save time. The second factor is *parallel pruning*. It uses multiple nodes to evaluate multiple networks in parallel. Both factors apply to both the default CNN pruning scheme and our GTSN-based scheme. This experiment compares the end-to-end CNN pruning times by the two schemes.

We experiment with a spectrum of target accuracy, represented by accuracy drop rate α, which is defined as the distance from the accuracy offered by the original well-trained large network. The value of α goes from -2% to 4%. 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 randomly sample 500 configurations from the configuration space and start evaluations from the smallest models and proceed to larger ones. We run the experiment in distributed settings with one node evaluating one network configuration independently. The numbers of workers are 1, 2, 4, 8, 16. Each worker corresponds to one node with one GPU.

Table III shows the speedups by GTSN. Speedups are calculated as the ratio of the end-to-end configuration time of the baseline approach and that of the GTSN-based CNN pruning, which includes the time for both block training and fine-tuning. GTSN shows significant speedups, cutting the end-to-end time by up to 202.8X for pruning ResNet-50 and up to 30.2X for Inception-V3.

The speedups come from two aspects. Firstly, the second step fine-tuning takes less training steps and thus less training time to reach a target accuracy threshold. The block-based training takes some time, but the time is small, easily offset by the time savings in the fine-tuning phase. This reason echoes the result of the basic measurement experiment reported in Section V-B.

The second aspect is the saving of trial configurations. Our experiments show that a network trained with the GTSN-based scheme often reaches a higher accuracy than it gets through the default training. Figure 11 provides the accuracy curves of ResNet-50 and Inception-V3 on dataset CUB200. The GTSN-based training yields a 3-5% higher accuracy than the default training. The better training results hence allow GTSN-based CNN pruning to stop the exploration of the configuration space earlier than the baseline approach does. For example, with a drop rate $\alpha = 0\%$, the baseline approach needs to evaluate 297 configurations before a satisfying model is found on the Flowers102 dataset; the process takes 1639.4 hours. However, by leveraging GTSN, the number of configurations to evaluate is reduced to three and the overall training time including both block training and fine-tuning is only 16.9 hours.

When many nodes are used, the speedups become less significant; it is because the time of GTSN-based block training weights more in those cases. Our current implementation of the GTSN-based block training is largely serial; we expect even more speedups if parallelism in that part is fully leveraged in future implementations.

VI. Discussions

A. Relations with existing techniques for CNN pruning

The GTSN-based block-wise CNN pruning is a technique complementary to existing techniques for speeding up CNN pruning. Most prior work focuses on reducing the enormous configuration space to a smaller set of promising network configurations [17], [41]. GTSN-based pruning is useful for accelerating the evaluations of those remaining promising configurations by effectively leveraging the block-level training results across those networks. It is not a competitor but a complementary technique that can be used together with prior techniques.

B. Possible Extensions

Two possible extensions to GTSN for CNNs and GTSN-based CNN pruning are discussed here for future work.

First, the proposed GTSN-based block-wise CNN pruning uses the basic CNN blocks (i.e. an Inception/residual module). As Observation 3 shows, using larger blocks may lead to even better block-wise training result. As the main goal of this work is to confirm the usefulness of GTSN for block-wise CNN pruning, we leave a systematic exploration of different sizes of blocks for block-wise CNN pruning to the future. It is, however, worth noting a trade-off: Even though a pruned network could benefit more from larger pre-trained blocks and thus require even fewer fine-tuning steps, there are more variants of pruned blocks (with a combinatorial growth) to train, and the time for training each pruned block also becomes longer.

Second, GTSN improves the accuracy of a student CNN when the last convolutional layer in each block of the student network is not changed so that the entire set of activation maps from each block of a teacher network can be used to train the corresponding block of the student network (See Observation 4). In the case of CNN pruning, such restriction slightly limits the exploration space for pruning. However, similar restrictions are also applied in prior works [2] [6] as pruning the last layer in a module could cause more severe

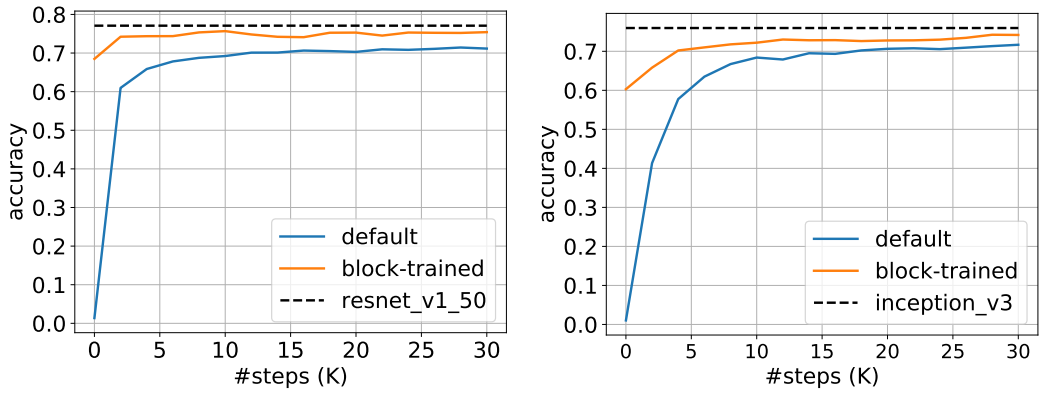
TABLE II: Average training time and speedups achieved by GTSN-based CNN pruning given a set of pruned networks.

Dataset	#configs	ResNet-50					Inception-V3				
		time (h)				speedup (X)	time (h)				speedup (X)
		baseline	GTSN-based				baseline	GTSN-based			
			block	finetune	total			block	finetune	total	
Flowers102	4	14.7	6.8	1.7	8.5	1.7	16.9	11.0	5.4	16.4	1.0
	16	41.3		3.2	10.0	4.1	59.7		17.8	28.8	2.1
	64	134.3		8.5	15.4	8.7	197.2		60.7	71.7	2.7
	256	451.1		27.7	34.6	13.0	696.5		214.7	225.7	3.1
CUB200	4	12.3	6.8	0.4	7.2	1.7	16.3	10.9	4.2	15.1	1.1
	16	40.0		1.3	8.1	4.9	57.4		15.2	26.1	2.2
	64	109.5		4.5	11.3	9.6	189.8		51.5	62.4	3.0
	256	331.1		13.2	20.0	16.5	596.0		158.9	169.8	3.5
Cars	4	17.5	6.8	3.2	10.1	1.7	20.8	11.0	9.0	20.0	1.0
	16	63.7		11.7	18.5	3.4	71.8		30.8	41.8	1.7
	64	205.1		31.3	38.1	5.5	249.2		104.2	115.2	2.2
	256	696.4		87.9	94.7	7.4	868.0		353.0	364.0	2.4
Dogs	4	9.4	6.8	0.2	7.1	1.3	13.1	10.9	2.8	13.7	1.0
	16	30.1		0.8	7.6	3.9	40.5		9.3	20.2	2.0
	64	82.7		1.7	8.6	9.6	138.5		31.6	42.5	3.3
	256	290.7		7.2	14.0	20.8	428.8		88.0	98.9	4.4

* *GTSN-based*: GTSN-based CNN pruning approach; *block*: GTSN-based block training; *finetune*: the fine-tuning step after block training;

TABLE III: End-to-end time for CNN pruning and speedups achieved by GTSN-based CNN pruning.

Dataset	α	#workers	ResNet-50					Inception-V3				
			baseline		GTSN-based		speedup (X)	baseline		GTSN-based		speedup (X)
			#configs	time (h)	#configs	time (h)		#configs	time (h)	#configs	time (h)	
Flowers102	0%	1	297	1639.4	3	16.9	97.0	244	1428.6	10	47.3	30.2
		4	300	412.6	4	5.2	79.3	244	358.2	12	13.9	25.8
		16	304	103.3	16	4.7	22.0	256	84.8	16	6.5	14.6
	0.5%	1	137	736.4	1	9.0	81.8	74	422.6	2	18.3	23.1
		4	140	186.8	4	3.9	47.9	76	106.7	4	6.9	15.5
		16	144	47.5	16	3.4	14.0	80	27.5	16	6.1	4.5
CUB200	3%	1	408	2319.4	2	12.7	182.6	201	1163.8	24	100.3	11.6
		4	408	578.2	4	3.3	175.2	204	294.7	24	25.9	11.4
		16	416	148.2	16	3.3	44.9	208	74.8	32	10.7	7.0
	3.5%	1	323	1804.8	1	8.9	202.8	120	688.0	4	25.6	26.9
		4	324	451.5	4	3.0	150.5	120	173.0	4	7.3	23.7
		16	336	115.3	16	2.9	39.8	128	46.1	16	6.5	7.1
Cars	-0.5%	1	500	2864.9	11	44.6	64.2	129	742.0	8	40.2	18.5
		4	500	720.4	12	12.3	58.6	132	188.8	8	10.8	17.5
		16	500	185.3	16	5.4	34.3	144	51.0	16	7.1	7.2
	0%	1	332	1848.6	11	44.4	41.6	84	480.3	3	21.8	22.0
		4	332	461.4	12	12.1	38.1	84	120.5	4	7.2	16.7
		16	336	115.9	16	5.2	22.3	96	33.8	16	6.7	5.0



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

Fig. 11: *Block-trained* networks, after fine-tuning, reaches a higher accuracy than the *default* training of the networks. Black dashed lines show the accuracies of the original unpruned CNN models. The X-axis is the number of fine-tuning/training steps. (dataset CUB200 is used)

performance degradation compared with pruning other layers. We leave a more systematic analysis of the restriction to future work.

VII. Conclusions

This work proposes GTSN as a way to overcome the key barrier for block-wise CNN pruning. Through empirical studies, it confirms the usefulness of activation maps of teacher networks for training individual pruned CNN blocks, which provides the foundation for GTSN to support block-wise CNN pruning. Based on a series of observations on the properties of GTSN, we implement the first block-wise CNN pruning framework and test it in various settings. The framework consistently shows significant benefits in accelerating CNN pruning, speeding up pruning of ResNet-50 by up to 202.8X and Inception-V3 by 30.2X.

Acknowledgement

This material is based upon work supported by DOE Early Career Award (DE-SC0013700), the National Science Foundation (NSF) under Grant No. CCF-1455404, CCF-1525609, CNS-1717425, CCF-1703487. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DOE or NSF.

References

[1] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Advances in neural information processing systems*, 2015, pp. 1135–1143.

[2] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," *arXiv:1608.08710*, 2016.

[3] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient transfer learning," *arXiv:1611.06440*, 2016.

[4] Q. Tian, T. Arbel, and J. J. Clark, "Deep lda-pruned nets for efficient facial gender classification," in *Computer Vision and Pattern Recognition Workshops (CVPRW), 2017 IEEE Conference on*. IEEE, 2017, pp. 512–521.

[5] J. Ye, X. Lu, Z. Lin, and J. Z. Wang, "Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers," *arXiv:1802.00124*, 2018.

[6] J.-H. Luo, J. Wu, and W. Lin, "Thinet: A filter level pruning method for deep neural network compression," *arXiv:1707.06342*, 2017.

[7] J. Liu, Y. Wang, and Y. Qiao, "Sparse deep transfer learning for convolutional neural network." in *AAAI*, 2017, pp. 2245–2251.

[8] A. Gordon, E. Eban, O. Nachum, B. Chen, H. Wu, T.-J. Yang, and E. Choi, "Morphnet: Fast & simple resource-constrained structure learning of deep networks," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[9] S. O'Keeffe and R. Villing, "Evaluating pruned object detection networks for real-time robot vision," in *Autonomous Robot Systems and Competitions (ICARSC), 2018 IEEE International Conference on*. IEEE, 2018, pp. 91–96.

[10] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv:1510.00149*, 2015.

[11] T.-J. Yang, Y.-H. Chen, and V. Sze, "Designing energy-efficient convolutional neural networks using energy-aware pruning," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[12] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv:1503.02531*, 2015.

[13] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio, "Fitnets: Hints for thin deep nets," *arXiv:1412.6550*, 2014.

[14] J. Ba and R. Caruana, "Do deep nets really need to be deep?" in *Advances in neural information processing systems*, 2014, pp. 2654–2662.

[15] S. Zagoruyko and N. Komodakis, "Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer," *arXiv:1612.03928*, 2016.

[16] J. Yim, D. Joo, J. Bae, and J. Kim, "A gift from knowledge distillation: Fast optimization, network minimization and transfer learning," in *The IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[17] A. Ashok, N. Rhinehart, F. Beainy, and K. M. Kitani, "N2n learning: Network to network compression via policy gradient reinforcement learning," *arXiv:1709.06030*, 2017.

[18] G. Chen, W. Choi, X. Yu, T. Han, and M. Chandraker, "Learning efficient object detection models with knowledge distillation," in *Advances in Neural Information Processing Systems*, 2017, pp. 742–751.

[19] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: A system for large-scale machine learning." in *OSDI*, vol. 16, 2016, pp. 265–283.

[20] C. Bucilua, R. Caruana, and A. Niculescu-Mizil, "Model compression," in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2006, pp. 535–541.

[21] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *International Conference on Computer Vision (ICCV)*, vol. 2, 2017, p. 6.

[22] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning efficient convolutional networks through network slimming," in *2017 IEEE International Conference on Computer Vision (ICCV)*. IEEE, 2017, pp. 2755–2763.

[23] Y. He and S. Han, "Adc: Automated deep compression and acceleration with reinforcement learning," *arXiv:1802.03494*, 2018.

[24] J. Cheng, P.-s. Wang, G. Li, Q.-h. Hu, and H.-q. Lu, "Recent advances in efficient computation of deep convolutional neural networks," *Frontiers of Information Technology & Electronic Engineering*, vol. 19, no. 1, pp. 64–77, 2018.

[25] H. Hu, R. Peng, Y.-W. Tai, and C.-K. Tang, "Network trimming: A data-driven neuron pruning approach towards efficient deep architectures," *arXiv:1607.03250*, 2016.

[26] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[27] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv:1409.1556*, 2014.

[28] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[29] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[30] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv:1502.03167*, 2015.

[31] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 2818–2826.

[32] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.

[33] M.-E. Nilsback and A. Zisserman, "Automated flower classification over a large number of classes," in *Computer Vision, Graphics & Image Processing, 2008. ICVGIP'08. Sixth Indian Conference on*. IEEE, 2008, pp. 722–729.

[34] P. Welinder, S. Branson, T. Mita, C. Wah, F. Schroff, S. Belongie, and P. Perona, "Caltech-ucsd birds 200," 2010.

[35] J. Krause, M. Stark, J. Deng, and L. Fei-Fei, "3d object representations for fine-grained categorization," in *Computer Vision Workshops (ICCVW), 2013 IEEE International Conference on*. IEEE, 2013, pp. 554–561.

[36] A. Khosla, N. Jayadevaprakash, B. Yao, and F.-F. Li, "Novel dataset for fine-grained image categorization: Stanford dogs," in *Proc. CVPR Workshop on Fine-Grained Visual Categorization*, vol. 2, 2011, p. 1.

[37] A. S. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson, "Cnn features off-the-shelf: an astounding baseline for recognition," in *Computer Vision and Pattern Recognition Workshops (CVPRW), 2014 IEEE Conference on*. IEEE, 2014, pp. 512–519.

[38] J. Fu, H. Zheng, and T. Mei, "Look closer to see better: Recurrent attention convolutional neural network for fine-grained image recognition," in *Conf. on Computer Vision and Pattern Recognition*, 2017.
[39] B. Zhao, X. Wu, J. Feng, Q. Peng, and S. Yan, "Diversified visual attention networks for fine-grained object classification," *IEEE Transactions on Multimedia*, vol. 19, no. 6, pp. 1245–1256, 2017.
[40] J. Krause, B. Sapp, A. Howard, H. Zhou, A. Toshev, T. Duerig, J. Philbin, and L. Fei-Fei, "The unreasonable effectiveness of noisy data for fine-grained recognition," in *European Conference on Computer Vision*. Springer, 2016, pp. 301–320.
[41] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of Machine Learning Research*, vol. 13, no. Feb, pp. 281–305, 2012.