Learning Efficient Convolutional Networks through Network Slimming

Zhuang Liu[1*] Jianguo Li[2] Zhiqiang Shen[3] Gao Huang[4] Shoumeng Yan[2] Changshui Zhang[1]
[1]CSAI, TNList, Tsinghua University [2]Intel Labs China [3]Fudan University [4]Cornell University
{liuzhuangthu, zhiqiangshen0214}@gmail.com, {jianguo.li, shoumeng.yan}@intel.com,
gh349@cornell.edu, zcs@mail.tsinghua.edu.cn

Abstract

The deployment of deep convolutional neural networks (CNNs) in many real world applications is largely hindered by their high computational cost. In this paper, we propose a novel learning scheme for CNNs to simultaneously 1) reduce the model size; 2) decrease the run-time memory footprint; and 3) lower the number of computing operations, without compromising accuracy. This is achieved by enforcing channel-level sparsity in the network in a simple but effective way. Different from many existing approaches, the proposed method directly applies to modern CNN architectures, introduces minimum overhead to the training process, and requires no special software/hardware accelerators for the resulting models. We call our approach network slimming*, which takes wide and large networks as input models, but during training insignificant channels are automatically identified and pruned afterwards, yielding thin and compact models with comparable accuracy. We empirically demonstrate the effectiveness of our approach with several state-of-the-art CNN models, including VGGNet, ResNet and DenseNet, on various image classification datasets. For VGGNet, a multi-pass version of network slimming gives a 20× reduction in model size and a 5× reduction in computing operations.*

1. Introduction

In recent years, convolutional neural networks (CNNs) have become the dominant approach for a variety of computer vision tasks, e.g., image classification [22], object detection [8], semantic segmentation [26]. Large-scale datasets, high-end modern GPUs and new network architectures allow the development of unprecedented large CNN models. For instance, from AlexNet [22], VGGNet [31] and GoogleNet [34] to ResNets [14], the ImageNet Classification Challenge winner models have evolved from 8 layers to more than 100 layers.

However, larger CNNs, although with stronger representation power, are more resource-hungry. For instance, a 152-layer ResNet [14] has more than 60 million parameters and requires more than 20 Giga float-point-operations (FLOPs) when inferencing an image with resolution 224× 224. This is unlikely to be affordable on resource constrained platforms such as mobile devices, wearables or Internet of Things (IoT) devices.

The deployment of CNNs in real world applications are mostly constrained by *1) Model size*: CNNs' strong representation power comes from their millions of trainable parameters. Those parameters, along with network structure information, need to be stored on disk and loaded into memory during inference time. As an example, storing a typical CNN trained on ImageNet consumes more than 300MB space, which is a big resource burden to embedded devices. *2) Run-time memory*: During inference time, the intermediate activations/responses of CNNs could even take more memory space than storing the model parameters, even with batch size 1. This is not a problem for high-end GPUs, but unaffordable for many applications with low computational power. *3) Number of computing operations:* The convolution operations are computationally intensive on high resolution images. A large CNN may take several minutes to process one single image on a mobile device, making it unrealistic to be adopted for real applications.

Many works have been proposed to compress large CNNs or directly learn more efficient CNN models for fast inference. These include low-rank approximation [7], network quantization [3, 12] and binarization [28, 6], weight pruning [12], dynamic inference [16], etc. However, most of these methods can only address one or two challenges mentioned above. Moreover, some of the techniques require specially designed software/hardware accelerators for execution speedup [28, 6, 12].

Another direction to reduce the resource consumption of large CNNs is to sparsify the network. Sparsity can be imposed on different level of structures [2, 37, 35, 29, 25], which yields considerable model-size compression and inference speedup. However, these approaches generally re-

*This work was done when Zhuang Liu and Zhiqiang Shen were interns at Intel Labs China. Jianguo Li is the corresponding author.

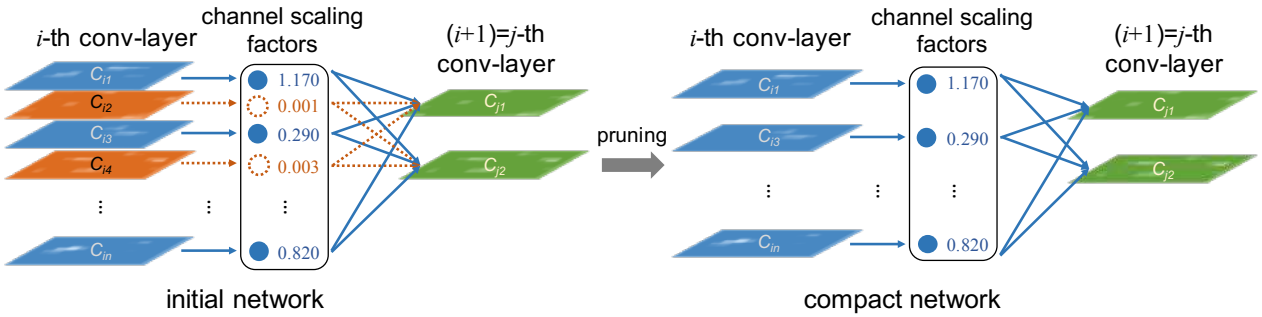

Figure 1: We associate a scaling factor (reused from a batch normalization layer) with each channel in convolutional layers. Sparsity regularization is imposed on these scaling factors during training to automatically identify unimportant channels. The channels with small scaling factor values (in orange color) will be pruned (left side). After pruning, we obtain compact models (right side), which are then fine-tuned to achieve comparable (or even higher) accuracy as normally trained full network.

quire special software/hardware accelerators to harvest the gain in memory or time savings, though it is easier than non-structured sparse weight matrix as in [12].

In this paper, we propose *network slimming*, a simple yet effective network training scheme, which addresses all the aforementioned challenges when deploying large CNNs under limited resources. Our approach imposes L1 regularization on the scaling factors in batch normalization (BN) layers, thus it is easy to implement without introducing any change to existing CNN architectures. Pushing the values of BN scaling factors towards zero with L1 regularization enables us to identify insignificant channels (or neurons), as each scaling factor corresponds to a specific convolutional channel (or a neuron in a fully-connected layer). This facilitates the channel-level pruning at the followed step. The additional regularization term rarely hurt the performance. In fact, in some cases it leads to higher generalization accuracy. Pruning unimportant channels may sometimes temporarily degrade the performance, but this effect can be compensated by the followed fine-tuning of the pruned network. After pruning, the resulting narrower network is much more compact in terms of model size, runtime memory, and computing operations compared to the initial wide network. The above process can be repeated for several times, yielding a multi-pass network slimming scheme which leads to even more compact network.

Experiments on several benchmark datasets and different network architectures show that we can obtain CNN models with up to 20x mode-size compression and 5x reduction in computing operations of the original ones, while achieving the same or even higher accuracy. Moreover, our method achieves model compression and inference speedup with conventional hardware and deep learning software packages, since the resulting narrower model is free of any sparse storing format or computing operations.

2. Related Work

In this section, we discuss related work from five aspects.

Low-rank Decomposition approximates weight matrix in neural networks with low-rank matrix using techniques like Singular Value Decomposition (SVD) [7]. This method works especially well on fully-connected layers, yielding $\sim$3x model-size compression however without notable speed acceleration, since computing operations in CNN mainly come from convolutional layers.

Weight Quantization. HashNet [3] proposes to quantize the network weights. Before training, network weights are hashed to different groups and within each group weight the value is shared. In this way only the shared weights and hash indices need to be stored, thus a large amount of storage space could be saved. [12] uses a improved quantization technique in a deep compression pipeline and achieves 35x to 49x compression rates on AlexNet and VGGNet. However, these techniques can neither save run-time memory nor inference time, since during inference shared weights need to be restored to their original positions.

[28, 6] quantize real-valued weights into binary/ternary weights (weight values restricted to $\{-1, 1\}$ or $\{-1, 0, 1\}$). This yields a large amount of model-size saving, and significant speedup could also be obtained given bitwise operation libraries. However, this aggressive low-bit approximation method usually comes with a moderate accuracy loss.

Weight Pruning / Sparsifying. [12] proposes to prune the unimportant connections with small weights in trained neural networks. The resulting network's weights are mostly zeros thus the storage space can be reduced by storing the model in a sparse format. However, these methods can only achieve speedup with dedicated sparse matrix operation libraries and/or hardware. The run-time memory saving is also very limited since most memory space is consumed by the activation maps (still dense) instead of the weights.

In [12], there is no guidance for sparsity during training. [32] overcomes this limitation by explicitly imposing sparse constraint over each weight with additional gate variables, and achieve high compression rates by pruning connections with zero gate values. This method achieves better com-

pression rate than [12], but suffers from the same drawback.

Structured Pruning / Sparsifying. Recently, [23] proposes to prune channels with small incoming weights in trained CNNs, and then fine-tune the network to regain accuracy. [2] introduces sparsity by random deactivating input-output channel-wise connections in convolutional layers before training, which also yields smaller networks with moderate accuracy loss. Compared with these works, we explicitly impose channel-wise sparsity in the optimization objective during training, leading to smoother channel pruning process and little accuracy loss.

[37] imposes neuron-level sparsity during training thus some neurons could be pruned to obtain compact networks. [35] proposes a Structured Sparsity Learning (SSL) method to sparsify different level of structures (e.g. filters, channels or layers) in CNNs. Both methods utilize group sparsity regualarization during training to obtain structured sparsity. Instead of resorting to group sparsity on convolutional weights, our approach imposes simple L1 sparsity on channel-wise scaling factors, thus the optimization objective is much simpler.

Since these methods prune or sparsify part of the network structures (e.g., neurons, channels) instead of individual weights, they usually require less specialized libraries (e.g. for sparse computing operation) to achieve inference speedup and run-time memory saving. Our network slimming also falls into this category, with absolutely no special libraries needed to obtain the benefits.

Neural Architecture Learning. While state-of-the-art CNNs are typically designed by experts [22, 31, 14], there are also some explorations on automatically learning network architectures. [20] introduces sub-modular/super-modular optimization for network architecture search with a given resource budget. Some recent works [38, 1] propose to learn neural architecture automatically with reinforcement learning. The searching space of these methods are extremely large, thus one needs to train hundreds of models to distinguish good from bad ones. Network slimming can also be treated as an approach for architecture learning, despite the choices are limited to the width of each layer. However, in contrast to the aforementioned methods, network slimming learns network architecture through only a single training process, which is in line with our goal of efficiency.

3. Network slimming

We aim to provide a simple scheme to achieve channel-level sparsity in deep CNNs. In this section, we first discuss the advantages and challenges of channel-level sparsity, and introduce how we leverage the scaling layers in batch normalization to effectively identify and prune unimportant channels in the network.

Advantages of Channel-level Sparsity. As discussed in prior works [35, 23, 11], sparsity can be realized at different levels, e.g., weight-level, kernel-level, channel-level or layer-level. Fine-grained level (e.g., weight-level) sparsity gives the highest flexibility and generality leads to higher compression rate, but it usually requires special software or hardware accelerators to do fast inference on the sparsified model [11]. On the contrary, the coarsest layer-level sparsity does not require special packages to harvest the inference speedup, while it is less flexible as some whole layers need to be pruned. In fact, removing layers is only effective when the depth is sufficiently large, e.g., more than 50 layers [35, 18]. In comparison, channel-level sparsity provides a nice tradeoff between flexibility and ease of implementation. It can be applied to any typical CNNs or fully-connected networks (treat each neuron as a channel), and the resulting network is essentially a "thinned" version of the unpruned network, which can be efficiently inferenced on conventional CNN platforms.

Challenges. Achieving channel-level sparsity requires pruning all the incoming and outgoing connections associated with a channel. This renders the method of directly pruning weights on a pre-trained model ineffective, as it is unlikely that all the weights at the input or output end of a channel happen to have near zero values. As reported in [23], pruning channels on pre-trained ResNets can only lead to a reduction of $\sim$10% in the number of parameters without suffering from accuracy loss. [35] addresses this problem by enforcing sparsity regularization into the training objective. Specifically, they adopt *group LASSO* to push all the filter weights corresponds to the same channel towards zero simultaneously during training. However, this approach requires computing the gradients of the additional regularization term with respect to all the filter weights, which is non-trivial. We introduce a simple idea to address the above challenges, and the details are presented below.

Scaling Factors and Sparsity-induced Penalty. Our idea is introducing a scaling factor γ for each channel, which is multiplied to the output of that channel. Then we jointly train the network weights and these scaling factors, with sparsity regularization imposed on the latter. Finally we prune those channels with small factors, and fine-tune the pruned network. Specifically, the training objective of our approach is given by

$$L = \sum_{(x,y)} l(f(x, W), y) + \lambda \sum_{\gamma \in \Gamma} g(\gamma) \qquad (1)$$

where (x, y) denote the train input and target, W denotes the trainable weights, the first sum-term corresponds to the normal training loss of a CNN, $g(\cdot)$ is a sparsity-induced penalty on the scaling factors, and λ balances the two terms. In our experiment, we choose $g(s) = |s|$, which is known as

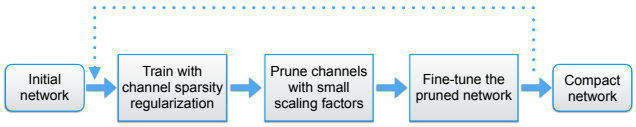

Figure 2: Flow-chart of network slimming procedure. The dotted-line is for the multi-pass/iterative scheme.

L1-norm and widely used to achieve sparsity. Subgradient descent is adopted as the optimization method for the non-smooth L1 penalty term. An alternative option is to replace the L1 penalty with the smooth-L1 penalty [30] to avoid using sub-gradient at non-smooth point.

As pruning a channel essentially corresponds to removing all the incoming and outgoing connections of that channel, we can directly obtain a narrow network (see Figure 1) without resorting to any special sparse computation packages. The scaling factors act as the agents for channel selection. As they are jointly optimized with the network weights, the network can automatically identity insignificant channels, which can be safely removed without greatly affecting the generalization performance.

Leveraging the Scaling Factors in BN Layers. Batch normalization [19] has been adopted by most modern CNNs as a standard approach to achieve fast convergence and better generalization performance. The way BN normalizes the activations motivates us to design a simple and efficient method to incorporates the channel-wise scaling factors. Particularly, BN layer normalizes the internal activations using mini-batch statistics. Let z_{in} and z_{out} be the input and output of a BN layer, $\mathcal{B}$ denotes the current mini-batch, BN layer performs the following transformation:

$$\hat{z} = \frac{z_{in} - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}; \quad z_{out} = \gamma \hat{z} + \beta \tag{2}$$

where $\mu_{\mathcal{B}}$ and $\sigma_{\mathcal{B}}$ are the mean and standard deviation values of input activations over $\mathcal{B}$, γ and β are trainable affine transformation parameters (scale and shift) which provides the possibility of linearly transforming normalized activations back to any scales.

It is common practice to insert a BN layer after a convolutional layer, with channel-wise scaling/shifting parameters. Therefore, we can directly leverage the γ parameters in BN layers as the scaling factors we need for network slimming. It has the great advantage of introducing no overhead to the network. In fact, this is perhaps also the most effective way we can learn meaningful scaling factors for channel pruning. *1)*, if we add scaling layers to a CNN without BN layer, the value of the scaling factors are not meaningful for evaluating the importance of a channel, because both convolution layers and scaling layers are linear transformations. One can obtain the same results by decreasing the scaling factor values while amplifying the weights in the convolution layers. *2)*, if we insert a scaling layer before a BN layer, the scaling effect of the scaling layer will be completely canceled by the normalization process in BN. *3)*, if we insert scaling layer after BN layer, there are two consecutive scaling factors for each channel.

Channel Pruning and Fine-tuning. After training under channel-level sparsity-induced regularization, we obtain a model in which many scaling factors are near zero (see Figure 1). Then we can prune channels with near-zero scaling factors, by removing all their incoming and outgoing connections and corresponding weights. We prune channels with a global threshold across all layers, which is defined as a certain percentile of all the scaling factor values. For instance, we prune 70% channels with lower scaling factors by choosing the percentile threshold as 70%. By doing so, we obtain a more compact network with less parameters and run-time memory, as well as less computing operations.

Pruning may temporarily lead to some accuracy loss, when the pruning ratio is high. But this can be largely compensated by the followed fine-tuning process on the pruned network. In our experiments, the fine-tuned narrow network can even achieve higher accuracy than the original unpruned network in many cases.

Multi-pass Scheme. We can also extend the proposed method from single-pass learning scheme (training with sparsity regularization, pruning, and fine-tuning) to a multi-pass scheme. Specifically, a network slimming procedure results in a narrow network, on which we could again apply the whole training procedure to learn an even more compact model. This is illustrated by the dotted-line in Figure 2. Experimental results show that this multi-pass scheme can lead to even better results in terms of compression rate.

Handling Cross Layer Connections and Pre-activation Structure. The network slimming process introduced above can be directly applied to most plain CNN architectures such as AlexNet [22] and VGGNet [31]. While some adaptations are required when it is applied to modern networks with *cross layer connections* and the *pre-activation* design such as ResNet [15] and DenseNet [17]. For these networks, the output of a layer may be treated as the input of multiple subsequent layers, in which a BN layer is placed before the convolutional layer. In this case, the sparsity is achieved at the incoming end of a layer, i.e., the layer selectively uses a subset of channels it received. To harvest the parameter and computation savings at test time, we need to place a *channel selection* layer to mask out insignificant channels we have identified.

4. Experiments

We empirically demonstrate the effectiveness of network slimming on several benchmark datasets. We implement

(a) Test Errors on CIFAR-10

Model	Test error (%)	Parameters	Pruned	FLOPs	Pruned
VGGNet (Baseline)	6.34	20.04M	-	7.97×10^8	-
VGGNet (70% Pruned)	**6.20**	2.30M	88.5%	3.91×10^8	51.0%
DenseNet-40 (Baseline)	6.11	1.02M	-	5.33×10^8	-
DenseNet-40 (40% Pruned)	**5.19**	0.66M	35.7%	3.81×10^8	28.4%
DenseNet-40 (70% Pruned)	5.65	0.35M	65.2%	2.40×10^8	55.0%
ResNet-164 (Baseline)	5.42	1.70M	-	4.99×10^8	-
ResNet-164 (40% Pruned)	**5.08**	1.44M	14.9%	3.81×10^8	23.7%
ResNet-164 (60% Pruned)	5.27	1.10M	35.2%	2.75×10^8	44.9%

(b) Test Errors on CIFAR-100

Model	Test error (%)	Parameters	Pruned	FLOPs	Pruned
VGGNet (Baseline)	26.74	20.08M	-	7.97×10^8	-
VGGNet (50% Pruned)	**26.52**	5.00M	75.1%	5.01×10^8	37.1%
DenseNet-40 (Baseline)	25.36	1.06M	-	5.33×10^8	-
DenseNet-40 (40% Pruned)	**25.28**	0.66M	37.5%	3.71×10^8	30.3%
DenseNet-40 (60% Pruned)	25.72	0.46M	54.6%	2.81×10^8	47.1%
ResNet-164 (Baseline)	23.37	1.73M	-	5.00×10^8	-
ResNet-164 (40% Pruned)	**22.87**	1.46M	15.5%	3.33×10^8	33.3%
ResNet-164 (60% Pruned)	23.91	1.21M	29.7%	2.47×10^8	50.6%

(c) Test Errors on SVHN

Model	Test Error (%)	Parameters	Pruned	FLOPs	Pruned
VGGNet (Baseline)	2.17	20.04M	-	7.97×10^8	-
VGGNet (60% Pruned)	**2.06**	3.04M	84.8%	3.98×10^8	50.1%
DenseNet-40 (Baseline)	1.89	1.02M	-	5.33×10^8	-
DenseNet-40 (40% Pruned)	**1.79**	0.65M	36.3%	3.69×10^8	30.8%
DenseNet-40 (60% Pruned)	1.81	0.44M	56.6%	2.67×10^8	49.8%
ResNet-164 (Baseline)	**1.78**	1.70M	-	4.99×10^8	-
ResNet-164 (40% Pruned)	1.85	1.46M	14.5%	3.44×10^8	31.1%
ResNet-164 (60% Pruned)	1.81	1.12M	34.3%	2.25×10^8	54.9%

Table 1: Results on CIFAR and SVHN datasets. "Baseline" denotes normal training without sparsity regularization. In column-1, "60% pruned" denotes the fine-tuned model with 60% channels pruned from the model trained with sparsity, etc. The pruned ratio of parameters and FLOPs are also shown in column-4&6. Pruning a moderate amount (40%) of channels can mostly lower the test errors. The accuracy could typically be maintained with $\geq 60\%$ channels pruned.

our method based on the publicly available Torch [5] implementation for ResNets by [10]. The code is available at `https://github.com/liuzhuang13/slimming`.

4.1. Datasets

CIFAR. The two CIFAR datasets [21] consist of natural images with resolution 32×32. CIFAR-10 is drawn from 10 and CIFAR-100 from 100 classes. The train and test sets contain 50,000 and 10,000 images respectively. On CIFAR-10, a validation set of 5,000 images is split from the training set for the search of λ (in Equation 1) on each model. We report the final test errors after training or fine-tuning on all training images. A standard data augmentation scheme (shifting/mirroring) [14, 18, 24] is adopted. The input data is normalized using channel means and standard deviations. We also compare our method with [23] on CIFAR datasets.

SVHN. The Street View House Number (SVHN) dataset [27] consists of 32x32 colored digit images. Following common practice [9, 18, 24] we use all the 604,388 training images, from which we split a validation set of 6,000 images for model selection during training. The test set contains 26,032 images. During training, we select the model with the lowest validation error as the model to be pruned (or the baseline model). We also report the test errors of the models with lowest validation errors during fine-tuning.

ImageNet. The ImageNet dataset contains 1.2 million training images and 50,000 validation images of 1000 classes. We adopt the data augmentation scheme as in [10]. We report the single-center-crop validation error of the final model.

MNIST. MNIST is a handwritten digit dataset containing 60,000 training images and 10,000 test images. To test the effectiveness of our method on a fully-connected network (treating each neuron as a channel with 1×1 spatial size), we compare our method with [35] on this dataset.

4.2. Network Models

On CIFAR and SVHN dataset, we evaluate our method on three popular network architectures: VGGNet[31], ResNet [14] and DenseNet [17]. The VGGNet is originally designed for ImageNet classification. For our experiment a variation of the original VGGNet for CIFAR dataset is taken from [36]. For ResNet, a 164-layer pre-activation ResNet with bottleneck structure (ResNet-164) [15] is used. For DenseNet, we use a 40-layer DenseNet with growth rate 12 (DenseNet-40).

On ImageNet dataset, we adopt the 11-layer (8-conv + 3 FC) "VGG-A" network [31] model with batch normalization from [4]. We remove the dropout layers since we use relatively heavy data augmentation. To prune the neurons in fully-connected layers, we treat them as convolutional channels with 1×1 spatial size.

On MNIST dataset, we evaluate our method on the same 3-layer fully-connected network as in [35].

4.3. Training, Pruning and Fine-tuning

Normal Training. We train all the networks normally from scratch as baselines. All the networks are trained using SGD. On CIFAR and SVHN datasets we train using mini-batch size 64 for 160 and 20 epochs, respectively. The initial learning rate is set to 0.1, and is divided by 10 at 50% and 75% of the total number of training epochs. On ImageNet and MNIST datasets, we train our models for 60 and 30 epochs respectively, with a batch size of 256, and an initial learning rate of 0.1 which is divided by 10 after 1/3 and 2/3 fraction of training epochs. We use a weight decay of 10^{-4} and a Nesterov momentum [33] of 0.9 without dampening. The weight initialization introduced by [13] is adopted. Our optimization settings closely follow the original implementation at [10]. In all our experiments, we initialize all channel scaling factors to be 0.5, since this gives higher accuracy for the baseline models compared with default setting (all initialized to be 1) from [10].

Training with Sparsity. For CIFAR and SVHN datasets, when training with channel sparse regularization, the hyper-parameteer λ, which controls the tradeoff between empirical loss and sparsity, is determined by a grid search over 10^{-3}, 10^{-4}, 10^{-5} on CIFAR-10 validation set. For VGGNet we choose λ=10^{-4} and for ResNet and DenseNet λ=10^{-5}. For VGG-A on ImageNet, we set λ=10^{-5}. All other settings are kept the same as in normal training.

Pruning. When we prune the channels of models trained with sparsity, a pruning threshold on the scaling factors needs to be determined. Unlike in [23] where different layers are pruned by different ratios, we use a global pruning threshold for simplicity. The pruning threshold is determined by a percentile among all scaling factors , e.g., 40% or 60% channels are pruned. The pruning process is implemented by building a new narrower model and copying the corresponding weights from the model trained with sparsity.

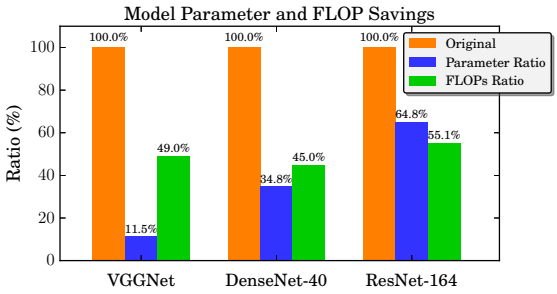

Figure 3: Comparison of pruned models with **lower** test errors on CIFAR-10 than the original models. The blue and green bars are parameter and FLOP ratios between pruned and original models.

Fine-tuning. After the pruning we obtain a narrower and more compact model, which is then fine-tuned. On CIFAR, SVHN and MNIST datasets, the fine-tuning uses the same optimization setting as in training. For ImageNet dataset, due to time constraint, we fine-tune the pruned VGG-A with a learning rate of 10^{-3} for only 5 epochs.

4.4. Results

CIFAR and SVHN The results on CIFAR and SVHN are shown in Table 1. We mark all lowest test errors of a model in **boldface**.

Parameter and FLOP reductions. The purpose of network slimming is to reduce the amount of computing resources needed. The last row of each model has $\geq$ 60% channels pruned while still maintaining similar accuracy to the baseline. The parameter saving can be up to $10\times$. The FLOP reductions are typically around 50%. To highlight network slimming's efficiency, we plot the resource savings in Figure 3. It can be observed that VGGNet has a large amount of redundant parameters that can be pruned. On ResNet-164 the parameter and FLOP savings are relatively insignificant, we conjecture this is due to its "bottleneck" structure has already functioned as selecting channels. Also, on CIFAR-100 the reduction rate is typically slightly lower than CIFAR-10 and SVHN, which is possibly due to the fact that CIFAR-100 contains more classes.

Regularization Effect. From Table 1, we can observe that, on ResNet and DenseNet, typically when 40% channels are pruned, the fine-tuned network can achieve a lower test error than the original models. For example, DenseNet-40 with 40% channels pruned achieve a test error of 5.19% on CIFAR-10, which is almost 1% lower than the original model. We hypothesize this is due to the regularization effect of L1 sparsity on channels, which naturally provides feature selection in intermediate layers of a network. We will analyze this effect in the next section.

VGG-A	Baseline	50% Pruned
Params	132.9M	23.2M
Params Pruned	-	82.5%
FLOPs	4.57×10^{10}	3.18×10^{10}
FLOPs Pruned	-	30.4%
Validation Error (%)	36.69	36.66

Table 2: Results on ImageNet.

Model	Test Error (%)	Params Pruned	#Neurons
Baseline	1.43	-	784-500-300-10
Pruned [35]	1.53	83.5%	434-174-78-10
Pruned (ours)	1.49	84.4%	784-100-60-10

Table 3: Results on MNIST.

ImageNet. The results for ImageNet dataset are summarized in Table 2. When 50% channels are pruned, the parameter saving is more than 5×, while the FLOP saving is only 30.4%. This is due to the fact that only 378 (out of 2752) channels from all the computation-intensive convolutional layers are pruned, while 5094 neurons (out of 8192) from the parameter-intensive fully-connected layers are pruned. It is worth noting that our method can achieve the savings with no accuracy loss on the 1000-class ImageNet dataset, where other methods for efficient CNNs [2, 23, 35, 28] mostly report accuracy loss.

MNIST. On MNIST dataset, we compare our method with the Structured Sparsity Learning (SSL) method [35] in Table 3. Despite our method is mainly designed to prune channels in convolutional layers, it also works well in pruning neurons in fully-connected layers. In this experiment, we observe that pruning with a global threshold sometimes completely removes a layer, thus we prune 80% of the neurons in each of the two intermediate layers. Our method slightly outperforms [35], in that a slightly lower test error is achieved while pruning more parameters.

We provide some additional experimental results in the supplementary materials, including (1) detailed structure of a compact VGGNet on CIFAR-10; (2) wall-clock time and run-time memory savings in practice. (3) comparison with a previous channel pruning method [23];

4.5. Results for Multi-pass Scheme

We employ the multi-pass scheme on CIFAR datasets using VGGNet. Since there are no skip-connections, pruning away a whole layer will completely destroy the models. Thus, besides setting the percentile threshold as 50%, we also put a constraint that at each layer, at most 50% of channels can be pruned.

The test errors of models in each iteration are shown in Table 4. As the pruning process goes, we obtain more and more compact models. On CIFAR-10, the trained model achieves the lowest test error in iteration 5. This model achieves 20× parameter reduction and 5× FLOP reduction, while still achieving **lower** test error. On CIFAR-100, after iteration 3, the test error begins to increase. This is possibly due to that it contains more classes than CIFAR-10, so pruning channels too agressively will inevitably hurt the performance. However, we can still prune near 90% parameters and near 70% FLOPs without notable accuracy loss.

(a) Multi-pass Scheme on CIFAR-10

Iter	Trained	Fine-tuned	Params Pruned	FLOPs Pruned
1	6.38	6.51	66.7%	38.6%
2	6.23	6.11	84.7%	52.7%
3	5.87	6.10	91.4%	63.1%
4	6.19	6.59	95.6%	77.2%
5	5.96	7.73	98.3%	88.7%
6	7.79	9.70	99.4%	95.7%

(b) Multi-pass Scheme on CIFAR-100

Iter	Trained	Fine-tuned	Params Pruned	FLOPs Pruned
1	27.72	26.52	59.1%	30.9%
2	26.03	26.52	79.2%	46.1%
3	26.49	29.08	89.8%	67.3%
4	28.17	30.59	95.3%	83.0%
5	30.04	36.35	98.3%	93.5%
6	35.91	46.73	99.4%	97.7%

Table 4: Results for multi-pass scheme on CIFAR-10 and CIFAR-100 datasets, using VGGNet. The baseline model has test errors of 6.34% and 26.74%. "Trained" and "Fine-tuned" columns denote the test errors (%) of the model trained with sparsity, and the fine-tuned model after channel pruning, respectively. The parameter and FLOP pruned ratios correspond to the fine-tuned model in that row and the trained model in the next row.

5. Analysis

There are two crucial hyper-parameters in network slimming, the pruned percentage t and the coefficient of the sparsity regularization term λ (see Equation 1). In this section, we analyze their effects in more detail.

Effect of Pruned Percentage. Once we obtain a model trained with sparsity regularization, we need to decide what percentage of channels to prune from the model. If we prune too few channels, the resource saving can be very limited. However, it could be destructive to the model if we prune too many channels, and it may not be possible to recover the accuracy by fine-tuning. We train a DenseNet-40 model with λ=10^{-5} on CIFAR-10 to show the effect of pruning a varying percentage of channels. The results are summarized in Figure 5.

From Figure 5, it can be concluded that the classification performance of the pruned or fine-tuned models degrade only when the pruning ratio surpasses a threshold. The fine-

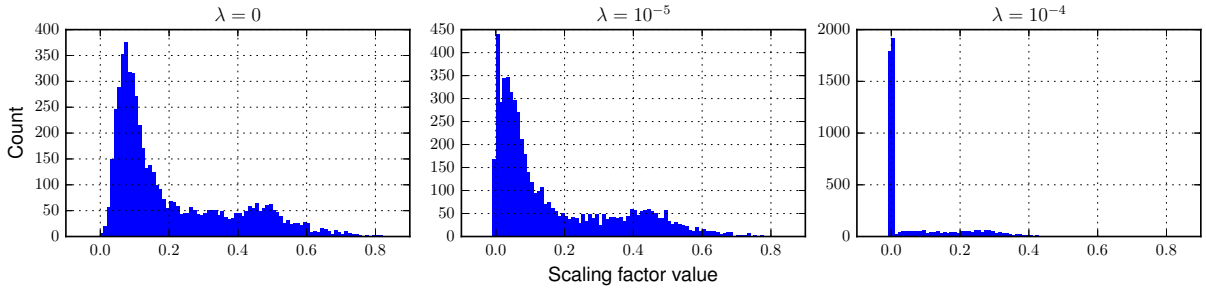

Figure 4: Distributions of scaling factors in a trained VGGNet under various degree of sparsity regularization (controlled by the parameter λ). With the increase of λ, scaling factors become sparser.

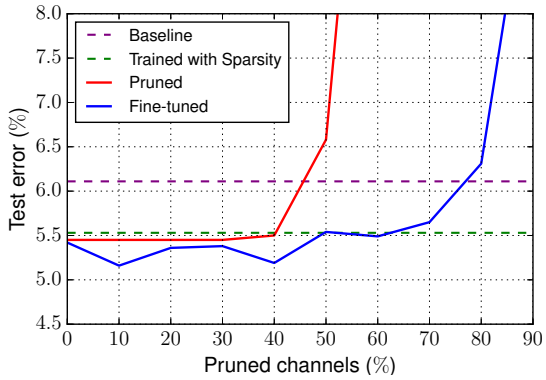

Figure 5: The effect of pruning varying percentages of channels, from DenseNet-40 trained on CIFAR-10 with λ=10^{-5}.

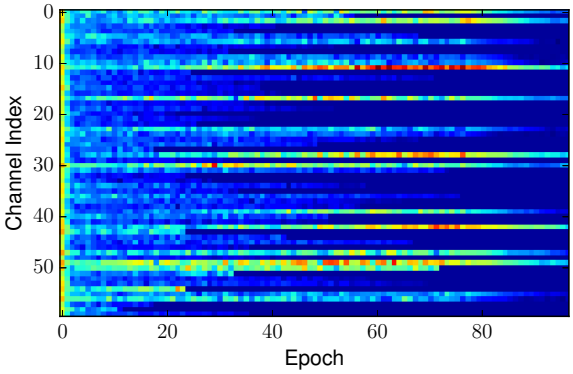

Figure 6: Visulization of channel scaling factors' change in scale along the training process, taken from the 11th conv-layer in VGGNet trained on CIFAR-10. Brighter color corresponds to larger value. The bright lines indicate the "selected" channels, the dark lines indicate channels that can be pruned.

tuning process can typically compensate the possible accuracy loss caused by pruning. Only when the threshold goes beyond 80%, the test error of fine-tuned model falls behind the baseline model. Notably, when trained with sparsity, even without fine-tuning, the model performs better than the original model. This is possibly due the the regularization effect of L1 sparsity on channel scaling factors.

Channel Sparsity Regularization. The purpose of the L1 sparsity term is to force many of the scaling factors to be near zero. The parameter λ in Equation 1 controls its significance compared with the normal training loss. In Figure 4 we plot the distributions of scaling factors in the whole network with different λ values. For this experiment we use a VGGNet trained on CIFAR-10 dataset.

It can be observed that with the increase of λ, the scaling factors are more and more concentrated near zero. When λ=0, i.e., there's no sparsity regularization, the distribution is relatively flat. When λ=10^{-4}, almost all scaling factors fall into a small region near zero. This process can be seen as a feature selection happening in intermediate layers of deep networks, where only channels with non-negligible scaling factors are chosen. We further visualize this process by a heatmap. Figure 6 shows the magnitude of scaling factors from one layer in VGGNet, along the training process. Each channel starts with equal weights; as the training progresses, some channels' scaling factors become larger (brighter) while others become smaller (darker).

6. Conclusion

We proposed the network slimming technique to learn more compact CNNs. It directly imposes sparsity-induced regularization on the scaling factors in batch normalization layers, and unimportant channels can thus be automatically identified during training and then pruned. On multiple datasets, we have shown that the proposed method is able to significantly decrease the computational cost (up to 20×) of state-of-the-art networks, with no accuracy loss. More importantly, the proposed method simultaneously reduces the model size, run-time memory, computing operations while introducing minimum overhead to the training process, and the resulting models require no special libraries/hardware for efficient inference.

Acknowledgements. Gao Huang is supported by the International Postdoctoral Exchange Fellowship Program of China Postdoctoral Council (No.20150015). Changshui Zhang is supported by NSFC and DFG joint project NSFC 61621136008/DFG TRR-169.

References

[1] B. Baker, O. Gupta, N. Naik, and R. Raskar. Designing neural network architectures using reinforcement learning. In *ICLR*, 2017.

[2] S. Changpinyo, M. Sandler, and A. Zhmoginov. The power of sparsity in convolutional neural networks. *arXiv preprint arXiv:1702.06257*, 2017.

[3] W. Chen, J. T. Wilson, S. Tyree, K. Q. Weinberger, and Y. Chen. Compressing neural networks with the hashing trick. In *ICML*, 2015.

[4] S. Chintala. Training an object classifier in torch-7 on multiple gpus over imagenet. `https://github.com/soumith/imagenet-multiGPU.torch`.

[5] R. Collobert, K. Kavukcuoglu, and C. Farabet. Torch7: A matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, number EPFL-CONF-192376, 2011.

[6] M. Courbariaux and Y. Bengio. Binarynet: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

[7] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *NIPS*, 2014.

[8] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *CVPR*, pages 580–587, 2014.

[9] I. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio. Maxout networks. In *ICML*, 2013.

[10] S. Gross and M. Wilber. Training and investigating residual nets. `https://github.com/szagoruyko/cifar.torch`.

[11] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In *ICLR*, 2016.

[12] S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. In *NIPS*, pages 1135–1143, 2015.

[13] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*, 2015.

[14] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[15] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In *ECCV*, pages 630–645. Springer, 2016.

[16] G. Huang, D. Chen, T. Li, F. Wu, L. van der Maaten, and K. Q. Weinberger. Multi-scale dense convolutional networks for efficient prediction. *arXiv preprint arXiv:1703.09844*, 2017.

[17] G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten. Densely connected convolutional networks. In *CVPR*, 2017.

[18] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger. Deep networks with stochastic depth. In *ECCV*, 2016.

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

[20] J. Jin, Z. Yan, K. Fu, N. Jiang, and C. Zhang. Neural network architecture optimization through submodularity and supermodularity. *arXiv preprint arXiv:1609.00074*, 2016.

[21] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. In *Tech Report*, 2009.

[22] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, pages 1097–1105, 2012.

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

[24] M. Lin, Q. Chen, and S. Yan. Network in network. In *ICLR*, 2014.

[25] B. Liu, M. Wang, H. Foroosh, M. Tappen, and M. Pensky. Sparse convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 806–814, 2015.

[26] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, pages 3431–3440, 2015.

[27] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng. Reading digits in natural images with unsupervised feature learning, 2011. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.

[28] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *ECCV*, 2016.

[29] S. Scardapane, D. Comminiello, A. Hussain, and A. Uncini. Group sparse regularization for deep neural networks. *arXiv preprint arXiv:1607.00485*, 2016.

[30] M. Schmidt, G. Fung, and R. Rosales. Fast optimization methods for l1 regularization: A comparative study and two new approaches. In *ECML*, pages 286–297, 2007.

[31] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015.

[32] S. Srinivas, A. Subramanya, and R. V. Babu. Training sparse neural networks. *CoRR*, abs/1611.06694, 2016.

[33] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. In *ICML*, 2013.

[34] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, et al. Going deeper with convolutions. In *CVPR*, pages 1–9, 2015.

[35] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li. Learning structured sparsity in deep neural networks. In *NIPS*, 2016.

[36] S. Zagoruyko. 92.5% on cifar-10 in torch. `https://github.com/szagoruyko/cifar.torch`.

[37] H. Zhou, J. M. Alvarez, and F. Porikli. Less is more: Towards compact cnns. In *ECCV*, 2016.

[38] B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. In *ICLR*, 2017.