Network Pruning via Annealing and Direct Sparsity Control

Yangzi Guo
Department of Mathematics
Florida State University
Tallahassee, Florida, USA
yguo@math.fsu.edu

Yiyuan She
Department of Statistics
Florida State University
Tallahassee, Florida, USA
yshe@stat.fsu.edu

Adrian Barbu
Department of Statistics
Florida State University
Tallahassee, Florida, USA
abarbu@stat.fsu.edu

***Abstract*—Artificial neural networks (ANNs) especially deep convolutional neural networks are very popular these days and have been proved to successfully offer quite reliable solutions to many vision problems. However, the use of deep neural networks is widely impeded by their intensive computational and memory cost. In this paper, we propose a novel efficient network pruning framework that is suitable for both non-structured and structured channel-level pruning. Our proposed method tightens a sparsity constraint by gradually removing network parameters or filter channels based on a criterion and a schedule. The attractive fact that the network size keeps dropping throughout the iterations makes it suitable for the pruning of any untrained or pre-trained network. Because our method uses a L_0 constraint instead of the L_1 penalty, it does not introduce any bias in the training parameters or filter channels. Furthermore, the L_0 constraint makes it easy to directly specify the desired sparsity level during the network pruning process. Finally, experimental validation on extensive synthetic and real vision datasets show that the proposed method obtains better or competitive performance compared to other states of art network pruning methods.**

I. Introduction

In recent years, artificial neural networks (ANNs) especially deep convolutional neural networks (DCNNs) are widely applied and have become the dominant approach in many computer vision tasks. These tasks include image classification [1]–[4], object detection [5], [6], semantic segmentation [7], 3D reconstruction [8], etc. The quick development in the deep learning field leads to network architectures that can go nowadays as deep as 100 layers and contain millions or even billions of parameters. Along with that, more and more computation resources must be utilized to successfully train such a deep modern neural network.

The deployment of DCNNs in real applications is largely impeded by their intensive computational and memory cost. With this observation, the study of network pruning methods that learn a smaller sub-network from a large original network without losing much accuracy has attracted a lot of attention. Network pruning algorithms can be divided into two groups: non-structured pruning and structured pruning. The earliest work for non-structured pruning is conducted by [9], the most recent work is done by [10], [11]. The non-structured pruning aims at directly pruning parameters regardless of the consistent structure for each network layer. This renders modern GPU acceleration technique unable to obtain computational benefits from the irregular sparse distribution of parameters in the network, only specialized software or hardware accelerators can gain memory and time savings. The advantage of non-structured pruning is that it can obtain high network sparsity and at the same time preserve the network performance as much as possible. On the other side, structured pruning aims at directly removing entire convolutional filers or filter channels. Li *et al.* [12] determines the importance of a convolutional filter by measuring the sum of its absolute weights. Liu *et al.* [13] introduces a L_1-norm constraint in the batch normalization layer to remove filter channels associated with smaller γ. Although structured pruning cannot obtain the same level of sparsity as non-structured pruning, it is more friendly to modern GPU acceleration techniques and independent of any specialized software or hardware accelerators.

Unfortunately, many of the existing non-structured and structured pruning techniques are conducted in a layer-wise way, requiring a sophisticated procedure for determining the hyperparameters of each layer in order to obtain a desired number of weights or filters/channels in the end. This kind of pruning manner is not effective nor efficient.

We combine regularization techniques with sequential algorithm design and direct sparsity level control to bring forward a novel network pruning scheme that could be suitable for either non-structured pruning or structured pruning (particular for filter channel-wise pruning of DCNNs with Batch Normalization layers). We investigate a parameter estimation optimization problem with a L_0-norm constraint in the parameter space, together with the use of annealing to lessen the greediness of the pruning process and a general metric to rank the importance of the weights or filter channels. An attractive property is that parameters or filter channels are removed while the model is updated at each iteration, which makes the problem size decrease during the iteration process. Experiments on extensive real vision data, including the MNIST, CIFAR, and SVHN provide empirical evidence that the proposed network pruning scheme obtains a performance comparable to or better than other state of art pruning methods.

II. Related Work

Network pruning is a very active research area nowadays, it provides a powerful tool to accelerate the network inference by having a much smaller sub-network without too much loss in accuracy. The earliest work about network pruning can be dated back to 1990s, when [9] and [14] proposed a weight pruning method that uses the Hessian matrix of the loss function to determine the unimportant weights. Recently, [11] used a quality parameter multiplied by the standard deviation of a layer's weights to determine the pruning threshold. A weight in a layer will be pruned if its absolute value is below that threshold. [15] proposed a pruning method that can properly incorporate connection slicing into the pruning process to avoid incorrect pruning. These pruning schemes mentioned above are all non-structured pruning, needing specialized hardware or software to gain computation and time savings.

For structured pruning, there are also quite a few works in the literature. [12] determine the importance of a convolutional filter by measuring the sum of its absolute weights. [16] compute the average percentage of zero activations after the ReLu function and determine to prune the corresponding filter if its this percentage score is high. [17] propose an iterative two-step channel pruning method by a LASSO regression based channel selection and least square reconstruction. [13] introduce a L_1-norm constraint in the batch normalization layer to remove filter channels associated with smaller $|\gamma|$. [18] impose an extra cluster loss term in the loss function that forces filters in each cluster to be similar and only keep one filter in each cluster after training. [19] utilize a greedy algorithm to perform channel selection in a layer-wise way by constructing a specific optimization problem.

III. Network Pruning via Annealing and Direct Sparsity Control

Given a set of training examples $\mathcal{D} = \{(\mathbf{x_i}, y_i), i = 1, ..., N\}$ where $\mathbf{x}$ is an input and y is a corresponding target output, with a differentiable loss function $L(\cdot)$ we can formulate the pruning problem for a neural network with parameters $\mathcal{W} = \{(\mathbf{W_j}, \mathbf{b_j}), j = 1, ..., L\}$ as following **constrained** problem

$$\min_{\mathcal{W}} \quad L(\mathcal{W}) \qquad \text{s.t.} \quad ||\mathcal{W}||_0 \leq K \tag{1}$$

where the L_0 norm bounds the number of non-zero parameters in $\mathcal{W}$ to be less than or equal to a specific positive integer K.

For non-structured pruning, we directly address the pruning problem in the whole $\mathcal{W}$ space. The final $\mathcal{W}$ will have an irregular distribution pattern of the zero-value parameters across all layers.

For structured pruning, suppose the DCNN is with convolutional filters or channels $\mathcal{C} = \{C_j, j = 1, ..., M\}$, we can replace the constrained problem (1) by

$$\min_{\mathcal{W}} \quad L(\mathcal{W}) \qquad \text{s.t.} \quad ||\mathcal{C}||_0 \leq K \tag{2}$$

By solving the problem (2), we will obtain the $\mathcal{W}$ on the convolutional layers having more uniform zero-value parameter distribution, specialized in some filters or filter channels.

These constrained optimization problems (1) and (2) facilitate parameter tuning because our sparsity parameter K is much more intuitive and easier to specify in comparison to penalty parameters such as λ in $\lambda||\mathcal{W}||_1$ and $\lambda||\mathcal{C}||_1$.

In this work, we will focus on the study of the weight-level pruning (non-structured pruning) for all neural networks and channel-level pruning (structured pruning) particularly for neural networks with Batch Normalization layers.

IV. Basic Algorithm Description

Some key ideas in our algorithm design are: a) We conduct our pruning procedures in the specified parameter spaces; b) We use an annealing plan to directly control the sparsity level in each parameter space; c) We gradually remove the most "unimportant" parameters or channels to facilitate computation. The prototype algorithms, summarized in Algorithm 1 and 2, show our ideas. It starts with either an untrained or pre-trained model and alternates two basic steps: one step of parameter updates towards minimizing the loss $L(\cdot)$ by gradient descent and one step that removes some parameters or channels according to a ranking metric $\mathcal{R}$.

Algorithm 1 Network Pruning via Direct Sparsity Control - Weight-Level (DSC-1)

Input: Training set $T = \{(\mathbf{x_i}, y_i)\}_{i=1}^n$, desired parameter space $\{\mathcal{W}_j | \cup \mathcal{W}_j = \mathcal{W} \ \& \cap \mathcal{W}_j = \varnothing\}_{j=1}^B$, desired number $\{K_j\}_{j=1}^B$ of parameters, desired annealing schedule $\{M_j^e, e = 1, .., N^{iter}\}_{j=1}^B$, an ANN model.

Output: Pruned ANN depending on exactly $\{K_j\}_{j=1}^B$ parameters in each parameter space $\{\mathcal{W}_j\}_{j=1}^B$.

1: If the ANN is not pre-trained, train it to a satisfying level.
2: **for** $e = 1$ to N^{iter} **do**
3: Sequentially update $\mathcal{W} \leftarrow \mathcal{W} - \eta \frac{\partial L(\mathcal{W})}{\partial \mathcal{W}}$ via backpropagation.
4: **for** $j = 1$ to B **do**
5: Keep the M_j^e most important parameters in $\mathcal{W}_j$ based on ranking metric $\mathcal{R}$.
6: **end for**
7: **end for**
8: Fine-tune the pruned ANN with exactly $\{K_j\}_{j=1}^B$ parameters in each parameter space $\{\mathcal{W}_j\}_{j=1}^B$.

The intuition behind our **DSC** algorithms is that during the pruning process, each time we remove a certain number of the most unimportant parameters/channels in each parameter/channel space based on an annealing schedule. This ensures that we do not inject too much noise in the parameter/channel dropping step so that the pruning procedure can be conducted smoothly. Our method directly controls the sparsity level obtained at each parameter/channel space, unlike many layer-wise pruning methods where a sophisticated procedure has to

be used to control how many parameters are kept, because pruning the weights or channels in all layers simultaneously can be very time-consuming.

Algorithm 2 Network Pruning via Direct Sparsity Control - Channel-Level (DSC-2)

Input: Training set $T = \{(\mathbf{x_i}, y_i)\}_{i=1}^{n}$, desired channel space $\{\mathcal{C}_j | \cup \mathcal{C}_j = \mathcal{C}\ \&\ \cap \mathcal{C}_j = \varnothing\}_{j=1}^{B}$, desired number $\{K_j\}_{j=1}^{B}$ of channels, desired annealing schedule $\{M_j^e, e = 1, .., N^{iter}\}_{j=1}^{B}$, a DCNN model.

Output: Pruned DCNN depending on exactly $\{K_j\}_{j=1}^{B}$ channels in each channel space $\{\mathcal{C}_j\}_{j=1}^{B}$.

1: If the DCNN is not pre-trained, train it to a satisfying level.
2: **for** $e = 1$ to N^{iter} **do**
3: Sequentially update $\mathcal{W} \leftarrow \mathcal{W} - \eta \frac{\partial L(\mathcal{W})}{\partial \mathcal{W}}$ via backpropagation
4: **for** $j = 1$ to B **do**
5: Keep the M_j^e most important channels in $\mathcal{C}_j$ based on ranking metric $\mathcal{R}$.
6: **end for**
7: **end for**
8: Fine-tune the pruned DCNN with exactly $\{K_j\}_{j=1}^{B}$ channels in each parameter space $\{\mathcal{C}_j\}_{j=1}^{B}$.

Through the annealing schedule, the support set of the network parameters or channels is gradually shrunken until we reach $||\mathcal{W}||_0 \leq K$ or $||\mathcal{C}||_0 \leq K$. The keep-or-kill rule is based on the ranking metric $\mathcal{R}$ and does not involve any information of the objective function L. This is in contrast to many ad-hoc networking pruning approaches that have to modify the loss function and can not easily be scaled up to many existing pre-trained models.

V. Implementation Details

In this part, we provide implementation details of our proposed **DSC** algorithms.

First, the annealing schedule M_e is determined empirically. Our experimental experience shows that the following annealing plans can perform well to balance the efficiency and accuracy:

$$M_e = \begin{cases} (1-p_0) + p_0(\frac{N_1 - e}{\mu e + N_1}))M, & 1 \leq e < N_1 \\ (1 - \min(p, p_0 + \left\lfloor \frac{e - N_1}{N^c} \right\rfloor \nu))M, & N_1 < e \leq N^{iter} \end{cases}$$

Here M is the total number of parameters or channels in the neural network. Our M_e consists two parts. The first part can be used to quickly prune the unimportant parameters with a reasonable value of μ down to a percentage p_0 of the parameters. The second part can further refine our pruned sub-network to a more compact model. μ is the pruning rate and we will set it to $\mu = 10$ for all experiments. $p_0 \in [0, 1]$ denotes the percentage of parameters or channels to be pruned in the first part. $p \in [0, 1]$ denotes the final pruning percentage

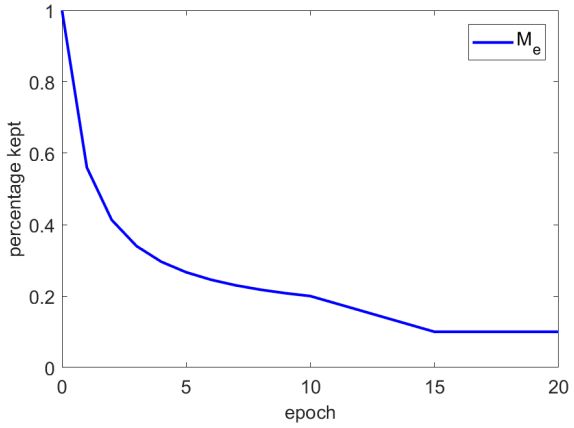


Fig. 1. Annealing schedule with $N_1 = 10, N^{iter} = 20, N^c = 1$, $p_0 = 0.8, p = 0.9, \nu = 0.02$.

goal at the end of the pruning procedure, thus the number of remaining parameters is $K = M(1-p)$. The parameter N^c specifies how many epochs to train before performing another pruning. We will select $N^c \in \{1, 2\}$. ν denotes the incremental pruning percentage as the annealing continues and will be set to $\nu \in \{0.005, 0.01, 0.02\}$. An example of an annealing schedule M_e (with $M = 1$ for clarity) with $N_1 = 10, N^{iter} = 20, N^c = 1$, $p_0 = 0.8, p = 0.9$, and $\nu = 0.02$ is shown in Figure 1.

Second, as the convolutional layers and fully connected layers have very different behavior in a DCNN, we will prune them separately during the structured and non-structured pruning process, i.e. we will fix the convolutional layer parameters while pruning the fully connected layer, and vice versa.

Third, the ranking metric $\mathcal{R}$ we select for structured and non-structured pruning is different. For non-structured pruning, the parameter dropping procedure based on the magnitude of the parameter yields quite good pruning results in our experiments. Therefore we will select it as our metric to rank the importance of parameters for all our non-structured pruning experiments:

$$\mathcal{R}(w) = |w|, w \in \mathcal{W} \tag{3}$$

For structured channel pruning, various dropping criteria are proposed. One family of channel pruning metrics are based on the value of the channel weights. Li [20] uses the L_1-norm by summing up the magnitude of all channel weights to rank the importance of the metric in a channel space; Wen [21] suggests the use of the L_2-norm. Another family of channel pruning metrics [13] lies in the absolute value of the Batch Normalization scales, as Batch normalization [22] has been widely adopted by most modern DCNNs to accelerate the training speed and convergence. Assume z_{in} and z_{out} to be the input and output of a Batch Normalization layer, we can formulate the transformation of that BN layer performs as:

$$BN(z_{in}) = \frac{z_{in} - \mu_{\mathbf{B}}}{\sqrt{\sigma_{\mathbf{B}}^2 + \epsilon}}; z_{out} = \gamma \cdot BN(z_{in}) + \beta$$

where $\mathbf{B}$ denotes the mini-batch statistic of input activations, $\mu_{\mathbf{B}}$ and $\sigma_{\mathbf{B}}$ are the mean and standard deviation over $\mathbf{B}$, γ and β are trainable scale and shift parameters of the affine transformation. Liu [13] directly leverages the parameters γ in the Batch Normalization layers as the scaling factors they need for channel pruning. They impose a L_1 norm on each Batch Normalization layer for the γ to reformulate the training loss function. Here we combine the metrics of these two families to enjoy wider flexibility on the DCNNs and define our ranking metrics as follows:

$$\begin{aligned} \mathcal{R}_{\mathbf{B}}(\mathcal{C}) &= |\gamma_{\mathcal{C}}| \\ \mathcal{R}_{\mathbf{L}}(\mathcal{C}) &= (||\mathcal{C}||_{L_1} + ||\mathcal{C}||_{L_2})/2 \\ \mathcal{R}(\mathcal{C}) &= \alpha \cdot \frac{\mathcal{R}_{\mathbf{B}}(\mathcal{C})}{\mathcal{R}_{\mathbf{B}}^{max}(\mathcal{C})} + (1-\alpha) \cdot \frac{\mathcal{R}_{\mathbf{L}}(\mathcal{C})}{\mathcal{R}_{\mathbf{L}}^{max}(\mathcal{C})} \end{aligned} \tag{4}$$

where $\gamma_{\mathcal{C}}$ is the scale parameter of the BN for channel $\mathcal{C}$, $\alpha \in [0,1]$ is a hyper-parameter that needs to be specified to balance the two ranking terms $\mathcal{R}_{\mathbf{B}}$ and $\mathcal{R}_{\mathbf{L}}$. The main differences from the other pruning methods are that we do not make any modifications to the loss function, but utilize a L_0 norm constraint and we use an annealing schedule to gradually eliminate channels and lessen the greediness.

Fourth, after the pruning process, we will conduct a fine-tuning procedure to gain back the performance lost during the pruning period. Before we start the fine-tuning, we can remove for non-structured pruning the neurons that have zero incoming or outgoing degree and structured-pruning the convolution filter channels with all zero parameters to form a more compact network for later inference use.

VI. Experiments

In this section, we first present a simulation on a synthetic dataset named parity dataset [23] to demonstrate the effectiveness of our **DSC** algorithm with selected M_e annealing plan. Then we conduct non-structured pruning with Lenet-300-100 and LeNet-5 [24] on MNIST [25] dataset. Finally we conduct our experiments with VGG-16 [2] and DenseNet-40 [4] on CIFAR [26] and SVHN [27] dataset for structured channel pruning.

A. Synthetic Parity Dataset

The parity data with noise is a classical problem in computational learning theory [23]. The data has feature vector $\mathbf{x} \in \mathbb{R}^p$ which is uniformly drawn from $\{-1,+1\}^p$. The label is generated follow the XOR logic: for some unknown subset of k indices $1 \leq i_1 < ... < i_k$, the label value is set as

$$y = \begin{cases} x_{i_1} x_{i_2} ... x_{i_k} & \text{with probablity } 0.9 \\ -x_{i_1} x_{i_2} ... x_{i_k} & \text{with probablity } 0.1 \end{cases}$$

That is this dataset cannot be perfectly separated and the best classifier would have a prediction error of 0.1.

This kind of dataset is frequently used to test different optimizers and regularization techniques on the neural network (NN) model. We perform the experiment in $p = 50$ dimensional data with parities $k = 5$. The training set, valid set, and testing set contain respectively 15K, 5K and 5K data points. We train a one hidden layer neural network with default stochastic gradient descent (SGD) optimizer, Adam [28] optimizer and Adam + **DSC-1**. For NN with Adam + **DSC-1**, we start with 256 hidden nodes, and down to a hidden node number B in the range $B \in [1,16]$ using annealing schedule M_e. We report the best result out of 10 independent random initializations. Recently, a neural network based boosting method named BoostNet [23] significantly outperformed a normal NN on this data. As Zhang's [23] experiment setting is very similar to us but with more training data, so we directly extract their results and report together with our experimental outcomes. The comparison of the test errors is shown in Figure 2.

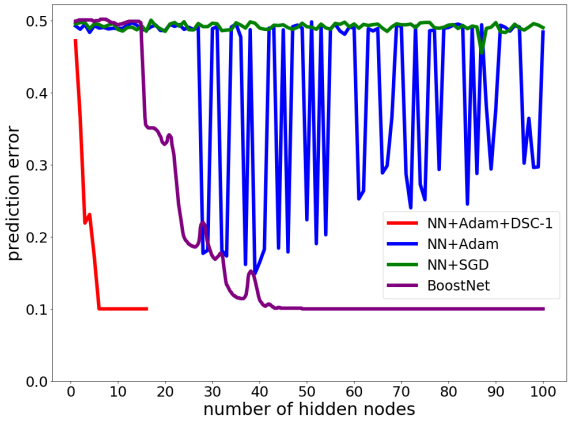


Fig. 2. Test error vs number of hidden nodes. Comparison between single hidden layer neural networks trained by NN + Adam + **DSC-1** starting with 256 hidden nodes, NN + Adam, NN + SGD and BoostNet.

We can see that the NN with the SGD optimizer cannot learn any good model with less than 100 hidden nodes on this data, while a NN with the Adam optimizer can learn some pattern when the number of hidden nodes is greater than 25, but still mostly cases are trapped in shallow local optima. The BoostNet can learn well if the hidden node number is greater than about 45 hidden nodes. The best performance is achieved by NN with Adam + **DSC-1**, with 256 starting hidden nodes. After applying **DSC-1** during the NN training, we only needed to keep as few as 6 hidden nodes to get the best possible prediction error. This observation implies: The **DSC-1** algorithm has a good capability to find a global or deep enough local optimum by gradually removing unimportant connections; The direct sparsity control design can help the final NN model reach very close to the most compact model achievable.

B. Non-structured Pruning on MNIST

The MINIST dataset provided by [25] is a handwritten digits dataset that is widely used in evaluating machine learning algorithms. It contains 50K training observations, 10K validation and 10K testing observations respectively. In this section, we will test our non-structured pruning method **DSC-1** on two network models: LeNet-300-100 and LeNet-5.

LeNet-300-100 [24] is a classical fully connected neural network with two hidden layers. The first hidden layer has 300 neurons and the second has 100. The LeNet-5 is a conventional convolutional neural network that has two convolution layers and two fully connected layers. LeNet-300-100 consists of 267K learnable parameters and LeNet-5 consists of 431K. To have a fair comparison with [11], we follow the same experimental setting by using the default SGD method, training batch size and initial learning rate to train the two models from scratch. After a model with similar performance was obtained, we stop the training and directly apply our **DSC-1** pruning algorithm to compress the model. During the pruning and retraining procedure, a learning rate with 1/10 of the original network's learning rate is adopted. A momentum with value of 0.9 is used to speed up the model retraining.

Model	Error	Params	Prune Rate
Lenet-300-100 (Baseline)	1.64%	267k	-
Lenet-300-100 (Han *et al.*)	1.59%	22K	91.8%
Lenet-300-100 (Ours)	**1.57%**	**17.4K**	**93.5%**
Lenet-5 (Baseline)	0.8%	431K	-
Lenet-5 (Han *et al.*)	0.77%	36k	91.6%
Lenet-5 (Ours)	0.77%	**15.8k**	**96.4%**

TABLE I
NON-STRUCTURED PRUNING COMPARISON. OUR **DSC-1** PRUNING METHOD CAN LEARN A MORE COMPACT SUB-NETWORK.

Model	Layer	Params.	Han %	Ours %
Lenet-300-100	fc1	236K	8%	4.6%
	fc2	30K	9%	20.1%
	fc3	1K	26%	68.5%
	Total	267K	8.2%	6.5%
Lenet-5	conv1	0.5K	66%	75%
	conv2	25K	12%	29.1%
	fc1	400K	8%	1.8%
	fc2	5K	19%	17.2%
	Total	431K	8.4%	3.6%

TABLE II
LAYER BY LAYER COMPRESSION COMPARISONS ON LENET-300-100 AND LENET-5. THE PERCENTAGE OF REMAINING PARAMETERS OF [11]'S PRUNING METHOD IS DISPLAYED IN THE THIRD COLUMN, OUR **DSC-1** PRUNING IS DISPLAYED IN THE LAST COLUMN.

In LeNet-300-100, a total of 20 epochs were used for both pruning and fine-tuning. For the annealing schedule, p_0 is directly set to 0.85 without using any annealing schedule. Then we follow the fine-grain pruning annealing schedule which $N^c = 1$ and $\nu = 0.05$ to reach at the final percentage goal $p = 0.935$.

The remaining epochs are used for fine-tuning purposes. In LeNet-5, the pruning for fully connected layers and convolutional layers are treated separately. For pruning on fully connected layers, we directly set at $p_0 = 0.9$ and then reach $p = 0.98$ with $N^c = 1, \nu = 0.05$. For the convolutional layers we start with $p_0 = 0$, $N^c = 1$ and $\nu = 0.05$ to reach at $p = 0.7$. The total number of pruning and retraining epochs for LeNet-5 is 40 epochs. After several experimental trials, we output our best result in Table I .

From the result table shown above, one can observe that our proposed non-structured pruning algorithm can learn a more compact sub-network for both LeNet300-100 and LeNet-5 with comparable performance with [11].

By using a hyperparameter we can directly control the sparsity level to get close to the most compact model achievable. It is not hard to conjecture that using a quality factor times the variance as a pruning threshold in each layer as proposed by [11] cannot exactly determine how many parameters should be kept. Our method can directly control the sparsity level and therefore enjoy a higher possibility to reach the position of the most compact sub-network.

Table II shows the layer-by-layer compression comparisons between ours and [11]. It is interesting to see that although two different pruning algorithms yield a similar performance result, the network architecture is quite different. Our **DSC-1** algorithm controls the directly specified sparsity level in the parameter space with an annealing schedule, this ensures the target sub-network can learn its pattern in an automatic way. For LeNet300-100, the most parameter killing comes from the first layer, which is quite reasonable as the images in the MNIST dataset are grayscale containing a large portion of pure black pixels. This large portion of black pixels almost has nothing to contribute to the neural network learning of useful information. The least parameter percentage dropping comes from the output layer, preserved as high as 68.5%. We can conjecture the reason for this behavior could be that as the most unrelated features are removed from the first fully connected layer, the output layer should remain a considerable number of parameters to bear the weight of those kept and useful features. For LeNet-5, the most parameter preservation occurs in the first convolutional layer. This is again really very reasonable, as indeed the first layer should be the most important layer that directly extracts relevant features from the raw input image pattern. Our direct sparsity control strategy lets the network itself decide which part is more important, and which part contains most irrelevant or junk connections that could be removed safely. The parameter percentage distribution of the two fully connected layers in LeNet-5 has a similar behavior as in LeNet-300-100.

C. Structured Channel Pruning on the CIFAR and SVHN Datasets

The CIFAR datasets (CIFAR10 and CIFAR100) provided by [26] are well established computer vision datasets used for image classification and object recognition. Both CIFAR datasets consist of a total of 60K natural color images and are divided into a training dataset with 50K images and a testing dataset with 10K images. The CIFAR-10 dataset is drawn from 10 classes with 6000 images per class. The CIFAR-100 dataset is drawn from 100 classes with 60 images per class. The color images in the CIFAR datasets have resolution 32×32.

The SVHN dataset [27] is a real-world image dataset for developing machine learning classification and object recognition algorithms. Similar to MNIST it consists of cropped digit images, but has as many as 600K training samples and 26K testing images in total. Each digit image is 32×32 and extracted from natural scenes.

DCNN	Model	Error (%)	Channels	Pruned	Params	Pruned
VGG-16	Base-unpruned [13]	6.34	5504	-	20.04M	-
	Pruned [13]	6.20	1651	70%	2.30M	88.5%
	Base-unpruned (Ours)	6.34	4224	-	14.98M	-
	Pruned (Ours)	**6.14**	1689	60%	4.40M	70.6%
	Pruned (Ours)	**6.20**	1267	70%	2.88M	80.7%
DenseNet-40	Base-unpruned [13]	6.11	9360	-	1.02M	-
	Pruned [13]	5.65	2808	70%	0.35M	65.2%
	Pruned (Ours)	**5.48**	3744	60%	0.45M	55.9%
	Pruned (Ours)	**5.57**	2808	70%	0.34M	66.7%

TABLE III
PRUNING PERFORMANCE RESULTS COMPARISON ON CIFAR-10.

DCNN	Model	Error (%)	Channels	Pruned	Params	Pruned
VGG-16	Base-unpruned [13]	26.74	5504	-	20.08M	-
	Pruned [13]	26.52	2752	50%	5.00M	75.1%
	Base-unpruned (Ours)	26.81	4224	-	15.02M	-
	Pruned (Ours)	26.55	2112	50%	6.01M	60.0%
DenseNet-40	Base-unpruned [13]	25.36	9360	-	1.06M	-
	Pruned [13]	25.72	3744	60%	0.46M	54.6%
	Pruned (Ours)	**25.66**	3744	60%	0.47M	55.6%

TABLE IV
PRUNING PERFORMANCE RESULTS COMPARISON ON CIFAR-100.

In this section, we will test our structured channel pruning method **DSC-2** on two network models: VGG-16 [2] and DenseNet40 [4]. The VGG-16 [2] is a deep convolutional neural network containing 16 layers which was mainly designed for the ImageNet dataset. Here we adopt a variation of VGG-16 designed for CIFAR datasets, which was used in [12] and has a smaller number of total parameters compared to Liu's [13], to conduct our experiments and compare with other state of art pruning algorithms. For DenseNet [4] we adopted the DenseNet40 with a total of 40 layers and a growth rate of 12.

We first train all the networks from scratch to obtain similar baseline results compared to [13]. The total epochs for training was set to 250 epochs for CIFAR, 20 epochs for SVHN, for all networks. The batch size used was 128. A Stochastic Gradient Descent (SGD) optimizer with an initial learning rate of 0.1, weight decay of 5×10^{-4} and momentum of 0.9 was adopted. A division of the learning rate by 5 occurs at every $25\%, 50\%, 75\%$ of total training epochs. For these datasets, standard data augmentation techniques like normalization, random flipping, and cropping may be applied.

During the pruning and fine-tuning procedure, the same number of training epochs is adopted in total. We use an SGD optimizer with an initial learning rate of 0.005 and no weight decay or very small weight decay for pruning and fine-tuning purposes. Similarly, a division of the learning rate by 2 occurs at every $25\%, 50\%, 75\%$ of total training epochs. For the annealing schedule, a grid search is utilized here to determine the best p_0, N^c and α for different p. After the first part of the pruning schedule when we reach the pruning target p_0, we conduct the fine-grain pruning for each final pruning rate p. We output our best results in Table III for CIFAR 10, Table IV for CIFAR 100 and Table V for SVHN.

The experimental results displayed in Tables III, IV and V demonstrate the effectiveness of our proposed channel pruning algorithm **DSC-2**. It can be observed that our **DSC-2** method can obtain results competitive with or even better than [13]. What's even better, our **DSC-2** pruning method does not introduce any extra term in the training loss function. By using the annealing schedule to gradually remove the "unimportant" channels based on a specified channel importance ranking metric $\mathcal{R}$, we could successfully find a compact sub-network without losing any model performance. Our **DSC-2** is easy to use and can be easily scaled up to any untrained or existing pre-trained model. The results of the FLOPs ratio between the original DCNNs and pruned sub-networks are shown in Figure 3.

Figure 4 displays two 70% channel-pruned network models for the CIFAR-10 dataset. Due to the significant differences in network architecture between the VGG-16 and DenseNet-40, the resulting distribution of the percentage of remaining channels is quite different. For VGG-16, only a very small number of channels are kept in the last five CONV layers. This is reasonable as the last five CONV layers are those layers that initially have 512 input channels. Evidently, we do not need so many channels in each of the last five layers. The high pruning percentage may suggest that the VGG-16 network is over-parameterized in a layer-wise way for the CIFAR 10 dataset. For DenseNet-40 with a growth rate of 12, the kept channel percentage is relatively evenly distributed in each CONV layer except the two transitional layers. This is again very reasonable based on the special architecture of DenseNet. With a growth rate of 12, every 12 consecutive layers are correlated with

DCNN	Model	Error (%)	Channels	Pruned	Params	Pruned
VGG-16	Base-unpruned [13]	2.17	5504	-	20.04M	-
	Pruned [13]	**2.06**	2201	60%	3.04M	84.8%
	Base-unpruned (Ours)	2.18	4224	-	14.98M	-
	Pruned (Ours)	**2.06**	1689	60%	4.31M	71.2%
DenseNet-40	Base-unpruned [13]	1.89	9360	-	1.02M	-
	Pruned [13]	1.81	3744	60%	0.44M	56.6%
	Pruned (Ours)	**1.80**	3744	60%	0.46M	54.9%

TABLE V
PRUNING PERFORMANCE RESULTS COMPARISON ON SVHN.

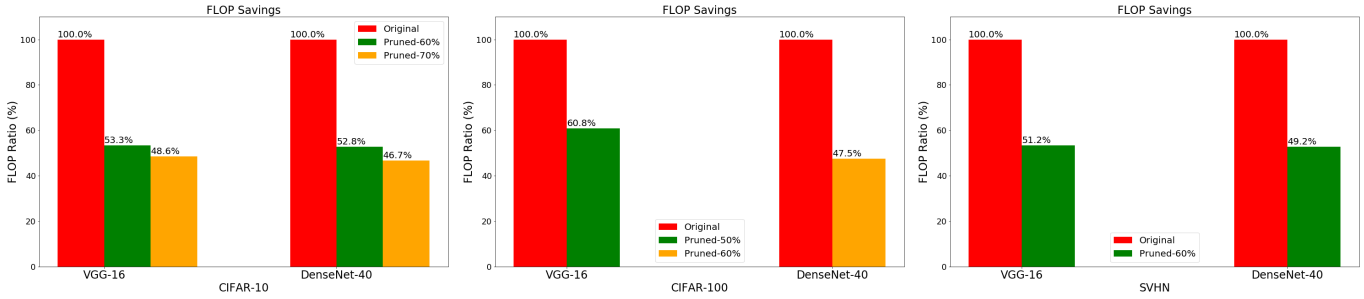
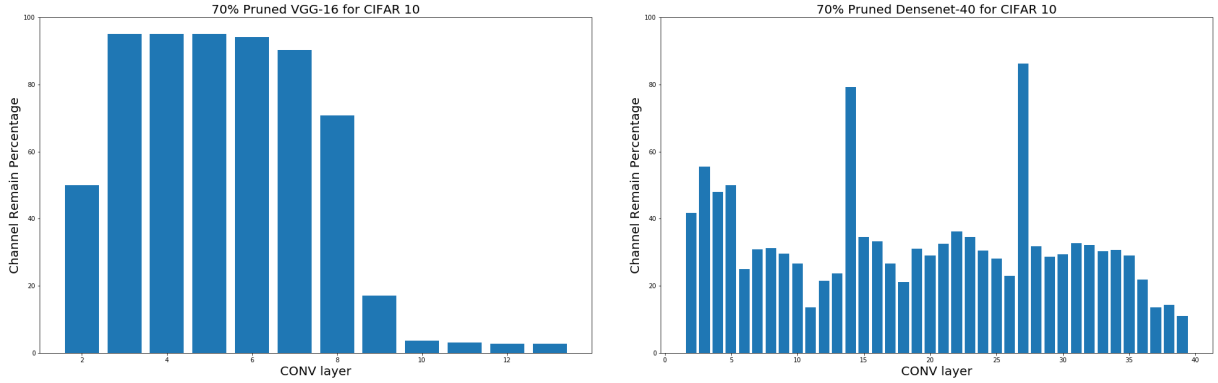


Fig. 3. FLOPs ratio between the original DCNNs and pruned sub-network for VGG-16 and DenseNet-40 on CIFAR and SVHN dataset.



70% channel pruned VGG-16 70% channel pruned DenseNet-40

Fig. 4. The remaining channel distribution for each CONV layer for pruned networks on CIFAR 10. The first CONV layer is not displayed as our channel pruning algorithm **DSC-2** will not act on this layer, thus contain the full percentage of channels.

each other, and outputs of those previous CONV layers will be concatenated to be the inputs of the following CONV layer inside the growth rate period. Only the transitional layers do not hold that property. Overall, our channel-level pruning algorithm **DSC-2** can automatically detect the reasonable sub-network without performance loss for VGG-16 and DenseNet-40 on the CIFAR and SVHN datasets.

Figure 5 displays the best results we obtained for just using one single global pruning rate $p \in \{0.1, 0.3, 0.5, 0.7\}$ to perform the channel pruning in the whole channel space. We can observe that even using a single target pruning rate parameter, we can still obtain very good sub-networks that generalize to CIFAR-10 test data well. A large portion of our best results displayed in Tables III, IV and V are obtained just using a single global pruning rate to guide the network pruning procedure. This observation makes our annealing pruning algorithm very easy to use, without worrying about channel subspace partitions. We also tested three different values for the parameter α of Eq. (4). For DenseNet-40, all of the choices of α can yield satisfactory performance for the pruned sub-network. For VGG-16, when α is set to 0, that is when the magnitude of γ is used as the ranking metric for channel pruning, gives the best results. This implies that different α values may be suitable for different network architectures.

VII. CONCLUSION

This paper presented a neural network pruning framework that is suitable for both structured and non-structured pruning. The method directly imposes a L_0 sparsity constraint on the network parameters, which is gradually tightened to the desired sparsity level. This direct control allows us to obtain

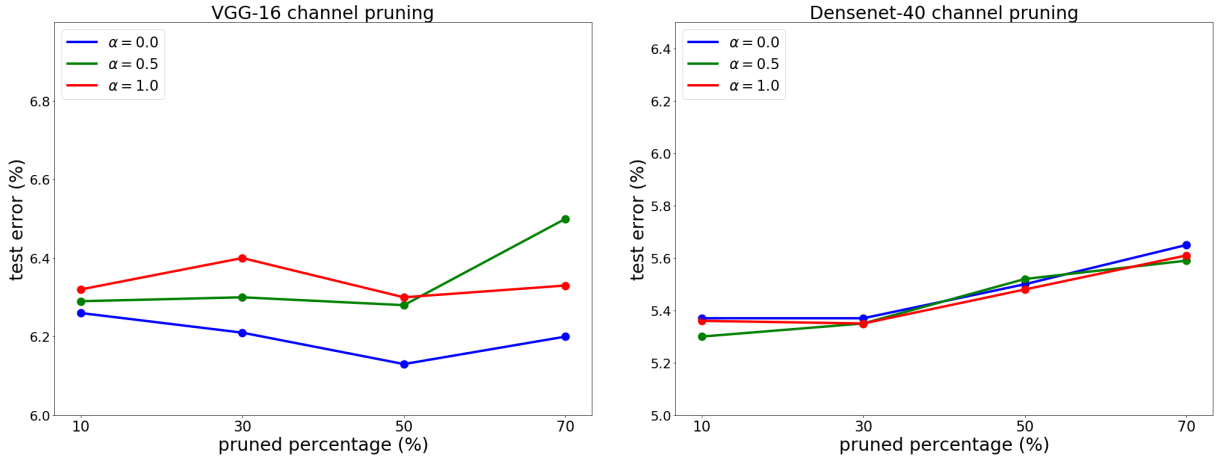


Fig. 5. The test error for various channel pruned percentage using one single global pruning rate for CIFAR-10.

the precise sparsity level desired, as opposed to other methods that obtain the sparsity level indirectly through either a quality factor times the variance or the use of penalty parameters. Experiments on extensive synthetic and real vision data, including the MNIST, CIFAR, and SVHN provide empirical evidence that the proposed network pruning scheme obtains a performance comparable to or better than other state of art pruning methods.

REFERENCES

[1] 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.

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

[3] 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.

[4] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.

[5] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580–587.

[6] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in neural information processing systems*, 2015, pp. 91–99.

[7] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3431–3440.

[8] P. Dou, S. K. Shah, and I. A. Kakadiaris, "End-to-end 3d face reconstruction with deep neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 5908–5917.

[9] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *Advances in neural information processing systems*, 1990, pp. 598–605.

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

[11] 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.

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

[13] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning efficient convolutional networks through network slimming," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 2736–2744.

[14] B. Hassibi and D. G. Stork, "Second order derivatives for network pruning: Optimal brain surgeon," in *Advances in neural information processing systems*, 1993, pp. 164–171.

[15] Y. Guo, A. Yao, and Y. Chen, "Dynamic network surgery for efficient dnns," in *Advances In Neural Information Processing Systems*, 2016, pp. 1379–1387.

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

[17] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 1389–1397.

[18] Z. Zhou, W. Zhou, R. Hong, and H. Li, "Online filter weakening and pruning for efficient convnets," in *2018 IEEE International Conference on Multimedia and Expo (ICME)*. IEEE, 2018, pp. 1–6.

[19] R. Yu, A. Li, C.-F. Chen, J.-H. Lai, V. I. Morariu, X. Han, M. Gao, C.-Y. Lin, and L. S. Davis, "Nisp: Pruning networks using neuron importance score propagation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 9194–9203.

[20] Y. Li, S. Lin, B. Zhang, J. Liu, D. Doermann, Y. Wu, F. Huang, and R. Ji, "Exploiting kernel sparsity and entropy for interpretable cnn compression," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 2800–2809.

[21] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Advances in neural information processing systems*, 2016, pp. 2074–2082.

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

[23] Y. Zhang, J. Lee, M. Wainwright, and M. I. Jordan, "On the learnability of fully-connected neural networks," in *Artificial Intelligence and Statistics*, 2017, pp. 83–91.

[24] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[25] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[26] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[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.

[28] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.