GU J Sci, Part C, 10(3): 650-665 (2022)

Gazi University

**Journal of Science**

PART C: DESIGN AND TECHNOLOGY

http://dergipark.gov.tr/gujsc

# Improved Knowledge Distillation with Dynamic Network Pruning

Eren ŞENER[1] Emre AKBAŞ[1,*]

[1]*Middle East Technical University, Faculty of Engineering, Department of Computer Engineering, 06800, Çankaya/ANKARA*

***Article Info***

*Research article*
*Received: 06.07.2022*
*Revision: 20.08.2022*
*Accepted: 31.08.2022*

**Keywords**

*Knowledge Distillation*
*Neural Network*
*Compression*
*Image Classification*
*Deep Neural Networks*

**Abstract**

Deploying convolutional neural networks to mobile or embedded devices is often prohibited by limited memory and computational resources. This is particularly problematic for the most successful networks, which tend to be very large and require long inference times. Many alternative approaches have been developed for compressing neural networks based on pruning, regularization, quantization or distillation. In this paper, we propose the "Knowledge Distillation with Dynamic Pruning" (KDDP), which trains a dynamically pruned compact student network under the guidance of a large teacher network. In KDDP, we train the student network with supervision from the teacher network, while applying L1 regularization on the neuron activations in a fully-connected layer. Subsequently, we prune inactive neurons. Our method automatically determines the final size of the student model. We evaluate the compression rate and accuracy of the resulting networks on an image classification dataset, and compare them to results obtained by Knowledge Distillation (KD). Compared to KD, our method produces better accuracy and more compact models.

## 1. INTRODUCTION

Deep neural networks have enabled many applications in a diverse set of domains including vision, language, medicine and robotics. However, these models require large amounts of processing power and memory, which severely limits their deployability in limited-resource computers. New possibilities would emerge if such models can be deployed in embedded platforms, mobile and edge devices. Therefore, research on "neural network compression", that is reducing the processing and memory requirements of neural networks, is important.

Early work on neural network compression aimed to make a large network smaller by removing redundant structures. These can be weights, neurons, blocks, etc. LeCun et al. proposed one of the pioneering network compression methods, the Optimal Brain Damage method [1], which was followed by many magnitude-based network pruning methods [2, 3]. These approaches work by removing weights that are close to zero. To prune more structures and make the models smaller, regularization can be used to enforce sparsity. Han et al. proposed one of the first regularization-based model compression methods. Following this work, some other methods [5-7] that use regularization on different structures were also proposed. For convolutional networks, researchers have designed novel convolutional filters to save parameters which decrease redundancy [8-10]. Research on low-rank factorization methods [11-13] tries to find informative parameters by using matrix or tensor decomposition. Another major body of work [14-16] reduces the number of bits that represent each weight.

A prominent approach to network compression is the "knowledge distillation" (KD) method [17], where a large, cumbersome model called the teacher guides the training of a much smaller model called the student (Figure 1). The student network is trained with two losses: (i) the usual cross-entropy loss coming from the training set, (ii) the "softened" class probabilities output by the teacher, computed via a hyper-parameter

*Corresponding author, e-mail: emre@ceng.metu.edu.tr
DOI: 10.29109/gujsc.1141648

called temperature. The aim of "softened" probabilities is to increase the information about the target class by introducing uncertainty into the probability distribution. Softened probabilities contain similarity information on different classes, which is absent in the one-hot labels coming from the training set.

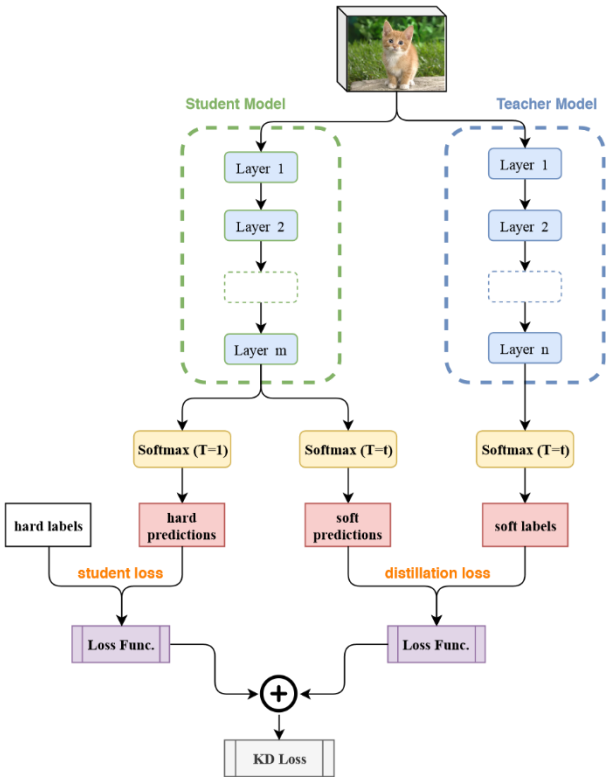


***Figure 1.*** *Illustration of the standard Knowledge Distillation method [17]. In KD, student model is trained with a linear combination of two losses. One loss comes from the one-hot (or hard) labels and the other from the "softened" labels. The architecture of the student model in KD is pre-determined and does not change during or after training. However, in our method, we prune fully-connected layers in the student network based on neuron activations to get a more compact model whose size is determined dynamically and automatically.*

KD based methods [18, 19] yield good performance on computer vision tasks and have had a significant impact on model compression. However, a major disadvantage of KD is that the user has to specify the student newtork architecture and this architecture do not change during or after training. While KD can succesfully distil the knowledge of the teacher into the student, we do not know whether the student model is unnecessarily large or smaller than it should be. In this paper, we address this disadvantage by dynamically pruning the student network based on neuron activations, to obtain a more compact student model. For pruning, we target the largest fully-connected layer of the student model, which typically contain the largest percentage of neurons in the student model. Specifically, during the KD training, we apply L1 regularization on the activations of the neurons in a selected fully-connected layer of the student model to impose sparsity. Then, we calculate the average activation over training examples, of each neuron in this layer. We prune those neurons having an average activation below a certain threshold by directly removing them from the network. To the best of our knowledge, our compression technique is the first method that combines KD and L1 regularization in this way. We name our method as Knowledge Distillation with Dynamic Pruning, or KDDP for short. Since our method can only prune fully-connected layers, its typical targets are Multilayer Perceptrons (MLP) or CNNs with large fully-connected (fc) layers.

We extensively analyze and compare standard training from scratch, knowledge distillation and our proposed method, KDDP, on the CIFAR10 dataset [20]. Experiments show that our method performs better than the baselines (standard training and KD), while also sigfinicantly compressing the student model. Furthermore, we find that setting hyper-parameters is crucial for KD based methods. Temperature, T,

distillation weight, α, and L1 regularization penalty should be tuned to find a good balance between the model size and the classification performance. In summary, when the hyper-parameters are chosen carefully, our method works well.

Our contributions in this paper can be summarized as follows.

- We propose a new dynamic compression method based on KD. It dynamically prunes inactive neurons in selected fully-connected layers from the student network. Unlike KD, our method does not require the final size of the compressed model as input; it is determined dynamically.
- We experimentally analyze our method and compare against standard training from scractch and KD. We make extensive experiments on our hyper-parameters to find meaningful relations with the accuracy of the compressed model.
- We test our method on the CIFAR10 dataset. We get better accuracies than both standard training and KD methods with much fewer parameters.

In the rest of the paper, we first summarize the neural network model compression literature in Section 2. We describe our proposed method and its implementation details in Section 3. Then, we analyze the effectiveness of our approach with experiments performed on the CIFAR10 dataset in Section 4, and finally conclude in Section 5.

## 2. BACKGROUND AND RELATED WORK

Here we review the literature on model compression in deep neural networks in two main categories: (i) parameter pruning and sharing, (ii) knowledge distillation. We give more detailed information about the parameter pruning and sharing due to its direct relation to our method.

### 2.1. Parameter Pruning And Sharing

Parameter pruning attracted many researchers since the early development of neural networks due to its effectiveness on reducing model complexity and over-fitting. It is also shown that pruning redundant parameters from the network improves generalization, which is an important side-effect.

Early works to prune parameters are Optimal Brain Damage [1] (OBD) and Optimal Brain Surgeon [2]. In their work, the authors remove redundant paramaters after sorting them by their saliencies. Saliency is measured based on the Hessian of the objective function. Recently, Srinivas and Babu also showed how similar neurons, which have similar weight sets, are redundant [3]. Since Hessian computation is heavy, they propose a more systematic way than OBD and data-free method to remove them.

Most of the follow-up works use sparsity constraints (L0, L1-norm, etc.) in the optimization problems to obtain redundancy. Researchers use these constraints on different elements (e.g. weights, blocks, etc.). Han et al. are one of the first to propose a regularization-based method on model compression [4]. They apply L2 regularization during the training phase in order to have near zero-valued parameters. Then they prune all lowweight connections from the network. The deep compression method [15] uses the same procedure as [4] for removing redundant connections. The authors also add quantization and Huffman coding on top of the pruned network to have a more compact one.

Recently, redundancy in convolutional networks also has been explored. Lebedev and Lempitsky [21] apply the idea of Optimal Brain Damage [1] to convolutional filters. They remove entries of L2,1-norm regularization applied convolution filters, which are below a threshold, in a group-wise fashion. Similarly, work by Zhou et al. [5] enforce low-rank constraints on tensors and L2,1-norm regularization on the objective function during the training stage to achieve compact CNNs with reduced neurons. Another study which uses L2,1-norm is Wen et al.'s work [6]. They apply regularization to big baseline models to learn more compact CNNs. With their structured sparsity method, they regularize filters, channels, filter shapes and layer depth of CNNs. Huang and Wang [22] improve the method of Wen et al. [6] and propose a more general end-to-end method for network pruning. Their method contains a factor to scale the output of a

specific neurons, groups or blocks. They apply L1-norm sparsity regularization to the scaling factors. Structures having scaling factors below a threshold are removed from the network while training. Unlike the previous works, Ullrich et al. [23] base their regularization on the soft weight-sharing method [24]. They compress weights of the pre-trained model into clusters by fitting mixtures of Gaussian models. After retraining the model with new weights concentrated on the cluster means, they obtain a layer-wise-pruned compact network.

There are also methods which focus on sparsity in batch-normalization (BN) layers. Liu et al. [7] add a scaling factor after BN layers. L1 regularization is applied on these scaling factors during training for the purpose of identifying redundant filters. Then, they prune channels with near-zero scaling factors. Another recent study [25] uses the method proposed by Beck and Teboulle [26] to enforce sparsity on the γ-parameter in BN operator. During training, this method makes some γ values zero and helps these channels to block sample-wise (for each sample in the training set) information flow. After the training is completed, they remove these constant-valued channels from the original network. The study MorphNet [27] uses a combination of three ideas above: first, an L1-norm-based regularization of the neurons, second, the idea of multipliers of Howard et al. [28] for reducing the floating point operations and model size, and third, the paradigm introduced by Han et al. [4] for retraining of the pruned network.

There has also been some research for measuring the redundancy in the networks. Guo et al. present a feedback mechanism named splicing which re-establishes mistakenly removed parameters after the pruning operation [29]. With this work, they show that measuring the redundancy of the parameters is an extremely difficult task. Researchers use different techniques for measuring redundancy. In [30], L1-norm of kernels are calculated. After sorting kernels by their L1norm values, small valued kernels and corresponding feature maps were pruned. ThiNet does filter-level pruning based on filter statistics computed from the following layer, not the current layer [31]. In spite of their success, the compression rate of the filters had to be predefined, which is another difficult problem for pruning methods. Moreover, He et al. exploit feature maps for redundancy [32]. The authors select the most representative channels of the feature maps and prune the redundant ones. After pruning, in order not to damage accuracy, they reconstruct the outputs with the remaining channels using linear least squares.

Recently, several methods proposed to measure the importance of structures. Yu et al. propose that layer-by-layer network pruning leads to significant reconstruction error propagation [33]. They introduce a global neuron importance measuring algorithm which uses information at the Final Response Layer (FRL, the second-to-last layer before classification). The algorithm obtains the importance of all neurons in the network with a single backward pass after a feature ranking operation on the FRL. Subsequently, the trimming of the whole network is performed considering the pruning ratio per layer as a pre-defined hyper-parameter. Prakash et al. propose a novel inter-filter orthogonality metric for ranking filter importance and a new training strategy [34]. Their method consists of temporarily dropping (some) of the least important convolutional filters (ranked by their metric), and reintroducing dropped filters with new weights. They repeat this process cyclically. With this strategy, they improve generalization and reduced overlap of learned features. Unlike the traditional deterministic methods, Wang et al. approach pruning weights of convolutional layers in a probabilistic manner [35]. They specify a pruning probability for each weight group. At each iteration, these probabilities are updated with the L1 norm as an importance criterion of each weight group. The pruning is guided by sampling from the pruning probabilities. He et al. use a novel pruning method instead of norm-based pruning approaches [36]. They calculate the geometric median Fletcher et al. of the filters within the same layer, and prune the filter(s) near to the geometric median [37]. In addition to above works, Dong et al. [38] improve the idea in previous works [1, 2]. Their pruning method is based on second order derivatives of a layer-wise error function.

There are also some recent and novel compression techniques used for pruning. In SplitNet [39], the goal is to find a tree-structured network that contains a set or a hierarchy of subnetworks, where the leaf-level subnetworks are associated with a specific group of classes. Since each group uses a subset of features that are completely disjoint from the ones used by other groups, the splitting algorithm prunes out inter-group

connections while optimizing the cross entropy loss and the group regularization. At the end, the weight matrix can be explicitly split into block diagonal matrices to reduce the number of parameters. Similarly, Yang et al. approach the network compression from energy consumption of the network [40]. They sort layers by their energy consumption, and pruned weights, which have small magnitudes, of the layers that consume the most energy first. Similar to the idea of "network slimming" [7], Zhao et al. modify the BN layer and add a new parameter called channel saliency to the BN layer [41]. They try to find approximate gamma distributions over these channel saliency parameters. They then remove redundant channels with mean and variance of their gamma distributions less than predefined thresholds.

### 2.2. Knowledge Distillation

Knowledge Distillation is a simple way to have compact deep learning models. In this method, a large (i.e. cumbersome) network or an ensemble model is trained, first. This model is called the "teacher", which typically produces accurate predictions. Then, a smaller network, called the "student", is trained using the guidance coming from the teacher model. This guidance is obtained using "temperature softmax" applied on the logits of the teacher. The goal is to provide a better training for the student model than using only the labels from the dataset. Trained in this way, the final student network was shown to produce comparable results to the teacher's [18].

Similar ideas to knowledge distillation has been explored before. Bucilua et al. approach the idea of knowledge transfer from a different point of view [42]. Instead of training a neural network on an original small set, they use an ensemble of base-level classifiers to label a large unlabeled dataset and then train the network on this much larger dataset. Ba and Caruana propose using L2 loss on the logits to mimic the teacher network [43].

FitNets [18] use knowledge distillation to yield deep and thin student networks that perform on par with or better than the teacher. They achieve this by training some student layers using the teacher's supervision for better initialization. Luo et al. show that using L2 loss to match the features of top hidden layers from both teacher and student is effective [19]. Yim et al. distill knowledge from the teacher by generating a matrix from feature maps at each layer [44]. Then, they transfer the knowledge from teacher to student, which has the same depth as the teacher, by applying L2 loss to these matrices.

### 2.3. Other Approaches

There are also other approaches to neural network compression and pruning that are orthogonal to our method. These include low-rank factorization methods, quantization and binarization methods and methods that aim to obtain compact convolutional filters.

### 2.4. Summary

Given the context of existing work, although $L_1$ regularization to enforce sparsity is commonly used for the purposes of pruning/compression, it has not been applied in the context of KD to obtain a student model whose size is determined dynamically and automatically.

## 3. METHOD

Before we present our method in detail, we first describe the knowledge distillation (KD) method [17] for completeness. In KD, there are two models: teacher and student. Given a supervised dataset, the teacher model is trained first. Then, the student model is trained using a linear combination of two losses: (i) the regular crossentropy loss coming from the supervised dataset, and (ii) "softened" cross-entropy loss coming from the teacher's prediction. To better explain these two losses, let us consider an example input image x with its ground-truth label y, which is a one-hot vector. A neural network outputs a raw score, or logit, zi for quantifying the degree that the input x belongs to class i. These logits are normalized using the "softmax" function so that the resulting vector can be considered as a probability distribution:

$$q_i = \frac{\exp(z_i/T)}{\sum_{c=1}^{C} \exp(z_c/T)} \quad (1)$$

where C is the number of classes and T is called the "temperature" parameter, which by default equals to 1. Then, cross-entropy between $\mathbf{q} = [q_1, q_2, \ldots, q_C]$ and y is computed as

$$\mathrm{CE}(\mathbf{y}, \mathbf{q}) = \sum_{c=1}^{C} y_c \log(q_c). \quad (2)$$

When T > 1, we call the loss as "softened" cross-entropy, as it softens the effect of the exponential function in softmax as T gets larger.

Let $\mathbf{q}^{\mathrm{tch,T}}$ denote the softmax output of the teacher model with temperature T. Similarly, let $\mathbf{q}^{\mathrm{std,T}}$ be the student's softmax output with temperature T. First, the teacher model is trained to minimize the regular cross-entropy loss:

$$L^{\mathrm{tch}} = \sum_{(\mathbf{x},\mathbf{y})} \mathrm{CE}(\mathbf{y}, \mathbf{q}^{\mathrm{tch,1}}). \quad (3)$$

And, the knowledge-distillation loss, by which the student model is trained, can be written as

$$L^{\mathrm{std}} = \sum_{(\mathbf{x},\mathbf{y})} (1-\alpha)\mathrm{CE}\left(\mathbf{y}, \mathbf{q}^{\mathrm{std,1}}\right) + \alpha \mathrm{CE}\left(\mathbf{q}^{\mathrm{tch,T}}, \mathbf{q}^{\mathrm{std,T}}\right). \quad (4)$$

The first term is the regular cross-entropy loss with one-hot ground-truth labels. The second term is the cross-entropy between temperature-softmax outputs. It is this second term, which brings in new information about class similarities predicted by the teacher. α is an hyperparameter to adjust the contribution of the two terms. Figure 1 illustrates the KD method.

In KD, both the teacher and the student model architectures are determined before training and are fixed during and after training. So, essentially, one hast to decide on the size of the student beforehand and KD attempts to distill the knowledge of the teacher into this student. However, there is no way of knowing the optimal size for the student architecture beforehand. Our method addresses this problem by dynamically pruning (removing) neurons from the student. By doing so, our method both finds an optimal size for the student model and slightly improves the final accuracy of the student model. In the following we describe our method.

### 3.1. Knowledge Distillation With Dynamic Pruning (KDDP)

As done in standard KD, we first train the teacher model, or it is provided as an already trained model. Then, we add L1 regularization to the largest fullyconnected layer of the student — let us call this layer fc1. The rationale behind this choice is that this layer typically contains a large percentage (up to 83% in our experiments) of all parameters in the model (Table 1). Next, we train the student using the KD loss defined in Equation (4). After the student is trained, we run it on the training set to calculate the average activation (i.e. output) of each neuron at fc1. If the activation of a neuron is below 10−6, we prune (i.e. kill or remove) that neuron and delete the corresponding weight set in its next layer. After testing all neurons at fc1, we re-train the pruned student network using Equation (4), this time without any L1 regularization on fc1.

### 3.2. Teacher And Student Models

As the teacher, we use a ResNet model [45]. ResNet and its variants proved their success on many computer vision tasks. Specifically, our teacher model is a ResNet-56 which achieves 6.97% error rate on CIFAR10 and has 850K learnable parameters. Details of ResNet-56 can be found in the original ResNet paper [45].

***Table 1.*** *Student networks differ only in the number of neurons in the fc1 layer. Percentages in parenthesis indicate the ratio of the parameters in fc1 to the total number of parameters.*

| Model | # of neurons in fc1 | # of parameters at fc1 | total # of parameters |
|---|---|---|---|
| $SN_{50}$ | 50 | 29k (34%) | 85k |
| $SN_{100}$ | 100 | 58k (50%) | 114k |
| $SN_{500}$ | 500 | 289k (83%) | 349k |

We choose our student network to have a very simple architecture in order to efficiently analyze the performance of our method. Our student network architecture starts with an input layer for 32×32×3 sized images. It is followed by a convolutional layer with a kernel size of 7×7 with a stride of 1, with 64 convolution filters. This result in an output of size 16 × 16 × 64. The convolutional layer is followed by a Batch Normalization (BN) layer and a ReLU non-linear activation function. We later use a max-pooling layer which has a window size of 3×3 with stride 2 that produces an output of size 7×7×64. This layer followed by an identity-block of the ResNet architecture [45]. ResNet's identity block is composed of 3 convolutional layers, each followed by a BN and a ReLU layer. The first and third convolutional layers have a kernel size of 1×1, and the middle layer has a kernel size of 3 × 3. The stride of all convolutions of the identity blocks is 1 and the number of filters used in each layer is 64. Before the ReLU layer of the last convolutional layer inside the residual block, there is a skip connection that allows the flow of information from the initial layers to the last layers by adding the input of the identity block and the output of the ReLU layer. The identity-block is followed by an average pooling layer which outputs a 3×3×64-dimensional tensor. This layer is followed by a fully-connected layer, fc1, and a ReLU layer. Finally, the ReLU layer is followed by another fully connected layer, fc2, as a bridge to a softmax layer at the end.

In our experiments, we create three different variant of this student model. The only difference between these student models are the neuron counts in the first fully-connected layer, fc1. We use 50, 100 and 500 neurons for this layer to explore the effect of the increasing number of neurons. We set the number of neurons in the second fully-connected layer, fc2, to the number of classes in the classification task at hand. The total number of parameters and percentage of parameters in fc1 for these networks are presented in Table 1. Figure 2 illustrates the architecture of our student network.

### 3.3. Baseline Methods

We compare our method with the following models.

**Vanilla SN:** We train the student network from scratch without any teacher guidance or regularization penalty. We use this model to find out the baseline performance of our student networks.

**Vanilla-KD SN:** We train the student network with standard Knowledge Distillation [17] at different temperature values ($T$) but without regularization penalty.

### 3.4. Implementation Details

**Teacher Network (TN):** We train a ResNet-56 model from scratch. The learning rate is $10^{-4}$, the mini-batch size is 64, and the optimization algorithm is Adam [46].

**Student Networks (SN):** We use the same hyper-parameters while training all student models. All models are trained from scratch. Weights and biases are initialized with Xavier's initialization [47]. Network architectures are implemented using the Keras framework [48]. Adam [46] is used for training. The learning rate is set to $10^{-4}$, and the mini-batch size is 64. An $L_1$ regularization penalty is applied on fc1 during the training of the KDDP student networks. The training is stopped early if there is no improvement in the accuracy on the validation set for 50 epochs.

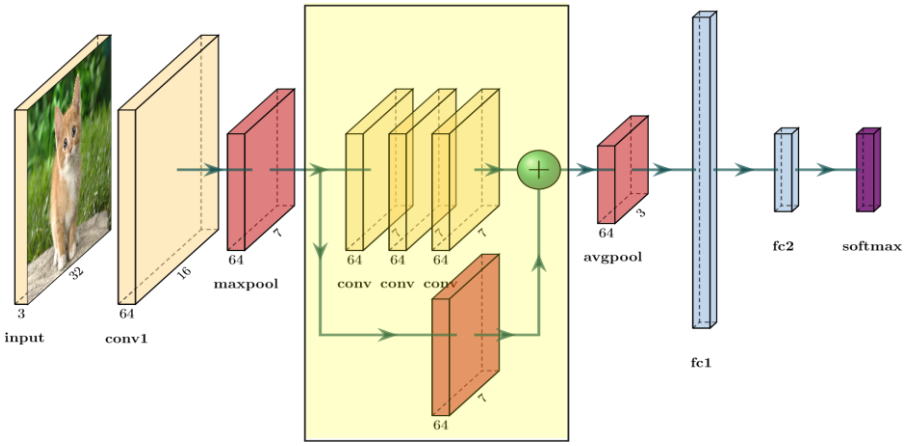


***Figure 2.*** *Student Network overview. The network takes an image and outputs a class label. It is composed of an input layer followed by a convolutional layer, max pooling, an identity-block of ResNet(He et al., 2016), average pooling, two fully connected layers, and a softmax layer. ResNet's identity block is highlighted with the yellow rectangle. This layer is composed of three convolutional layers and a skip connection which adds the input of the identity block and the output of the last convolutional layer in the identity block.*

## 4. EXPERIMENTS

In this section, we describe the experimental evaluation and validation of our method. We evaluate it by comparing against the two baselines and then, provide extensive experiments on hyper-parameters in Section 4.3.


***Figure 3.*** *Example images from the CIFAR10 dataset.*

### 4.1. Dataset

We use the CIFAR10 dataset [20] in our experiments. It contains 60000 32x32 color images in 10 classes, with 6000 images per class. Example images can be seen in Figure 3). There are 50000 training images and 10000 test images. We randomly sample (using stratified sampling, i.e., by preserving class frequencies)10000 images from the training set to form a validation set. We report our results after observing no improvements on the validation set for 50 epochs during training. As data augmentation, we only use horizontal flip. We use this setup in all experiments.

### 4.2. Analysis of the Proposed Method

We present our main results in Table 2, where we compare the performances and parameter counts of the teacher model, vanilla SN model, vanilla KD SN model and our KDDP model.

***Table 2.*** *Main results on the CIFAR10 test set.*

| **Model** | **Acc.** | **# Params.** |
|---|---|---|
| Teacher | 0.8808 | 1,673,738 |
| Vanilla $SN_{50}$ | 0.8048 | 85,104 |
| Vanilla KD $SN_{50}$ | 0.8141 | 85,104 |
| KDDP $SN_{50}$ (final \|fc1\|=45) | **0.8193** | **82,169** |
| Vanilla KD $SN_{45}$ | 0.8145 | **82,169** |
| Vanilla $SN_{100}$ | 0.8075 | 114,454 |
| Vanilla KD $SN_{100}$ | 0.8197 | 114,454 |
| KDDP $SN_{100}$ (\|fc1\|=83) | **0.8219** | **104,475** |
| Vanilla KD $SN_{83}$ | 0.8146 | **104,475** |
| Vanilla $SN_{500}$ | 0.8128 | 349,254 |
| Vanilla KD $SN_{500}$ | 0.8264 | 349,254 |
| KDDP $SN_{500}$ (\|fc1\|=264) | **0.8281** | **210,722** |
| Vanilla KD $SN_{264}$ | 0.8267 | **210,722** |

We use the same teacher logits in for all experiments (i.e. $z$ in Eq. (1)). We train our teacher once. We use the same initial weights for all student network trainings with hyper-parameters: L1 = 1e−4 and $\alpha = 0.5$. We also train Vanilla SNs and Vanilla-KD SNs for each model to explore the capacity of these networks and compare with our model.

The teacher network has 1.7M parameters and yields an accuracy of 88.08% on the test set. This score is lower than other ResNet results on the same set, e.g. Cai et al. achieve 97.92% [49]. This is because we hold out 10K examples from the training set as validation data to have a solid early stopping criterion. Also, we only use horizontal-flip augmentation.

The "Vanilla SN" (student network), which is the SN trained from scratch without any teacher guidance or regularization, has three versions. These versions differ only in the number of neurons in the fc1 fully connected layer. For 50, 100 and 500 neurons, Vanilla SN achieves 80.48%, 80.75%, 81.28% test set accuracy, respectively. Increasing the number of neurons in fc1 has a positive effect on model performance. However, this causes an increase in the number of parameters, as well. When the student model is trained using standard knowledge distillation method, we obtain the "Vanilla KD SN" models. Compared to the Vanilla SNs, they achieve around 1% better accuracy for all models.

Our method, KDDP, achieves 81.93%, 82.19%, 82.81% accuracies on the test set for student networks SN50, SN100, SN500, respectively. We observe that our method works better than both Vanilla SN and Vanilla KD SN, and improves the accuracies around 0.5% for SN50, SN100, SN500 with 3%, 9%, 40% fewer parameters than their original networks. It dynamically removes 10%, 17%, 48% of the parameters at fc1.

We conduct further experiments to compare our method against KD to provide fair comparisons based on the total number of neurons in the network. We record the number of final neurons in KDDP and we train smaller Vanilla KD SNs that have the same neuron counts at fc1 with the final KDDP SNs. We denote these models with "Vanilla KD SNn" where n is equal to 45, 83 or 264. We use the same softmax temperature for both bases. We observe that with the same fc1 size, KDDP outperforms Vanilla KD.

From these results, we conclude that our dynamic pruning method both improves accuracy and reduces computational cost of inference. In the following, we analyze the sensitivity of our method to its hyper-parameters, and also conduct statistical significance analysis.

### 4.3. Hyper-parameter Analysis

#### 4.3.1. L1 Regularization Penalty Analysis

We use $L_1$ regularization on the activations of fc1 layer neurons to increase sparsity. $L_1$ regularization penalizes the absolute value of the activations of the neurons. We present results for different $L_1$ penalties in Table 3. We set $\alpha$ to 0.5 in these experiments.

We observe that larger values of $L_1$ penalty result in fewer active neurons at the fc1 layer and therefore decreases the performance of the models. For example, when $L_1$ is $1e^{-3}$ and $T$ = 32 at KDDP $SN_{100}$ experiment, the model gets stuck at some local minima and cannot even reach the vanilla model's performance. However, when there are fewer parameters it helps the model to get acceptable performances. For example, for hyperparameters $L = 1e^{-3}$, $T = 2$, our KDDP $SN_{50}$ model achieves better performance than the other $SN_{50}$ models. We also observe that using smaller values for $L_1$, e.g. $L_1 = 1e^{-5}$, does not work for our pruning method in all student models. Therefore, $L_1$ penalty should be tuned to strike a good balance between the model size and the classification performance. In our experiments, we set the $L_1$ regularization to $1e^{-4}$.

***Table 3.*** *Effect of $L_1$ regularization penalty. Results for models $SN_{50}$, $SN_{100}$, $SN_{500}$. Hyper-parameter $\alpha$ = 0.5.*

| | | $L_1 = 1e^{-3}$ | | $L_1 = 1e^{-4}$ | | $L_1 = 1e^{-5}$ | |
|---|---|---|---|---|---|---|---|
| | T | **Acc.** | **fc1 size** | **Acc.** | **fc1 size** | **Acc.** | **fc1 size** |
| 50 neurons | 2 | **0.8198** | **15** | **0.8193** | **45** | **0.8162** | **49** |
| | 4 | 0.8054 | 10 | 0.8123 | 40 | 0.8146 | 48 |
| | 8 | 0.7638 | 7 | 0.7891 | 41 | 0.7968 | 47 |
| | 12 | 0.7707 | 7 | 0.7913 | 38 | 0.7918 | 47 |
| | 16 | 0.7761 | 8 | 0.7902 | 33 | 0.7948 | 46 |
| | 20 | 0.7802 | 8 | 0.8011 | 37 | 0.7961 | 47 |
| | 32 | 0.7607 | 5 | 0.8010 | 37 | 0.8049 | 48 |
| | 64 | 0.7776 | 7 | 0.8078 | 37 | 0.8034 | 44 |
| | 100 | 0.7871 | 6 | 0.8088 | 32 | 0.8038 | 49 |
| | 200 | 0.7966 | 7 | 0.8053 | 41 | 0.8077 | 47 |
| | 1000 | 0.7877 | 8 | 0.8065 | 35 | 0.8078 | 46 |
| | 5000 | 0.7616 | 5 | 0.8032 | 40 | 0.8069 | 43 |
| 100 neurons | 2 | **0.8132** | **18** | **0.8219** | **83** | **0.8219** | **97** |
| | 4 | 0.7913 | 9 | 0.8156 | 79 | 0.8192 | 93 |
| | 8 | 0.7655 | 7 | 0.8037 | 61 | 0.8099 | 92 |
| | 12 | 0.7931 | 10 | 0.7952 | 59 | 0.7941 | 92 |
| | 16 | 0.7970 | 76 | 0.7922 | 59 | 0.7931 | 92 |
| | 20 | 0.7802 | 7 | 0.7978 | 59 | 0.7935 | 89 |
| | 32 | 0.6900 | 3 | 0.8085 | 61 | 0.8001 | 93 |
| | 64 | 0.7779 | 6 | 0.8095 | 59 | 0.8012 | 91 |
| | 100 | 0.7594 | 4 | 0.8077 | 54 | 0.8121 | 91 |
| | 200 | 0.7800 | 6 | 0.8003 | 54 | 0.8055 | 90 |
| | 1000 | 0.7609 | 4 | 0.8171 | 53 | 0.8058 | 95 |
| | 5000 | 0.8063 | 89 | 0.8087 | 54 | 0.8010 | 88 |
| 500 neurons | 2 | 0.8211 | 103 | **0.8281** | **264** | 0.8185 | 467 |
| | 4 | **0.8213** | **330** | 0.8186 | 156 | **0.8196** | **453** |
| | 8 | 0.8175 | 400 | 0.8106 | 113 | 0.8101 | 441 |
| | 12 | 0.8017 | 140 | 0.8079 | 118 | 0.8110 | 432 |
| | 16 | 0.8010 | 235 | 0.7952 | 109 | 0.8089 | 429 |
| | 20 | 0.7991 | 151 | 0.8003 | 91 | 0.8075 | 425 |
| | 32 | 0.8047 | 98 | 0.8147 | 111 | 0.8063 | 425 |
| | 64 | 0.8050 | 17 | 0.8121 | 109 | 0.8025 | 417 |
| | 100 | 0.8040 | 90 | 0.8114 | 101 | 0.8030 | 431 |
| | 200 | 0.8062 | 185 | 0.8146 | 112 | 0.8076 | 423 |
| | 1000 | 0.8059 | 15 | 0.8140 | 116 | 0.8040 | 425 |
| | 5000 | 0.8123 | 17 | 0.8093 | 105 | 0.8070 | 423 |

### 4.3.2. $\alpha$ Analysis

$\alpha$ in Eq. (4) sets the contribution of the two objective functions (i.e. the weight of distillation). In other words, using bigger $\alpha$ values means giving more importance to soft targets in the objective function. We

present results for different $\alpha$ values in Table 4. We can see that too small and large $\alpha$ values don't lead to good performances. $\alpha$ also should be tuned carefully to have a good balance between the model size and classification performance. In our experiments, we observe that setting $\alpha$ to 0.5 gives the best results.

### 4.3.3. T Analysis

Setting the vaue of *T*, which "softens" the softmax output, is not a trivial task. To find its optimal value, we did a grid search over the temperatures values of [2, 4, 8, 12, 16, 20, 32, 64, 100, 200, 1000, 5000]. If we keep increasing *T*, at some point, logits will be saturated and no information will flow from the teacher to the student network. We present our results in Table 4. We can see that when we train the network with 100 neurons solely with the loss coming from the soft targets (the second term in Eq. (4)) with a temperature of 5000, we get an accuracy of 10%, which is equal to the random guess for CIFAR10. For all models, we observe that the accuracy fluctuates depending on *T*. Therefore, we conclude that the temperature parameter should also be tuned carefully.

***Table 4.*** *Distillation loss weight (α in Eq. (4)) and temperature analysis. Results for models $SN_{50}$,$SN_{100}$,$SN_{500}$. Hyperparameters: $L_1 = 1e^{-4}$.*

| | | $\alpha = 0.2$ | | $\alpha = 0.5$ | | $\alpha = 0.8$ | | $\alpha = 1$ | |
|---|---|---|---|---|---|---|---|---|---|
| | **T** | **Acc** | **fc1 size** | **Acc** | **fc1 size** | **Acc** | **fc1 size** | **Acc** | **fc1 size** |
| 50 neurons | 2 | 0.8046 | 48 | **0.8193** | **45** | 0.8093 | 49 | 0.7955 | 45 |
| | 4 | 0.8003 | 44 | 0.8123 | 40 | **0.8147** | **39** | **0.8034** | **32** |
| | 8 | 0.7987 | 42 | 0.7891 | 41 | 0.8031 | 26 | 0.7967 | 8 |
| | 12 | 0.7945 | 43 | 0.7913 | 38 | 0.7843 | 28 | 0.5061 | 25 |
| | 16 | 0.8093 | 43 | 0.7902 | 33 | 0.7726 | 23 | 0.5844 | 46 |
| | 20 | 0.8081 | 45 | 0.8011 | 37 | 0.7761 | 25 | 0.5882 | 45 |
| | 32 | 0.8058 | 39 | 0.8010 | 37 | 0.7872 | 23 | 0.5101 | 42 |
| | 64 | 0.8060 | 42 | 0.8078 | 37 | 0.7961 | 25 | 0.5850 | 42 |
| | 100 | 0.8021 | 44 | 0.8088 | 32 | 0.8069 | 24 | 0.5835 | 42 |
| | 200 | 0.8057 | 42 | 0.8053 | 41 | 0.8018 | 22 | 0.5526 | 39 |
| | 1000 | 0.8072 | 44 | 0.8065 | 35 | 0.8088 | 26 | 0.6277 | 40 |
| | 5000 | **0.8106** | **42** | 0.8032 | 40 | 0.8040 | 26 | 0.1000 | 12 |
| 100 neurons | 2 | 0.8077 | 76 | **0.8219** | **83** | **0.8239** | **87** | 0.8078 | 66 |
| | 4 | 0.8082 | 75 | 0.8156 | 79 | 0.8177 | 69 | **0.8108** | **49** |
| | 8 | 0.8042 | 66 | 0.8037 | 61 | 0.8041 | 33 | 0.7948 | 8 |
| | 12 | 0.8041 | 75 | 0.7952 | 59 | 0.7880 | 30 | 0.6685 | 88 |
| | 16 | 0.8005 | 69 | 0.7922 | 59 | 0.7733 | 26 | 0.5931 | 63 |
| | 20 | 0.8010 | 79 | 0.7978 | 59 | 0.7689 | 31 | 0.5886 | 40 |
| | 32 | 0.8075 | 69 | 0.8085 | 61 | 0.7889 | 25 | 0.5764 | 49 |
| | 64 | 0.8065 | 79 | 0.8095 | 59 | 0.7962 | 33 | 0.5835 | 51 |
| | 100 | 0.7963 | 72 | 0.8077 | 54 | 0.7985 | 33 | 0.5890 | 85 |
| | 200 | 0.8047 | 67 | 0.8003 | 54 | 0.8037 | 29 | 0.5748 | 85 |
| | 1000 | 0.8039 | 71 | 0.8171 | 53 | 0.8000 | 34 | 0.5864 | 86 |
| | 5000 | **0.8096** | **75** | 0.8087 | 54 | 0.8073 | 31 | 0.1000 | 14 |
| 500 neurons | 2 | **0.8170** | **247** | **0.8281** | **264** | **0.8255** | **262** | **0.8158** | **212** |
| | 4 | 0.8138 | 205 | 0.8186 | 156 | 0.8215 | 126 | 0.8157 | 57 |
| | 8 | 0.8100 | 204 | 0.8106 | 113 | 0.8133 | 44 | 0.6642 | 407 |
| | 12 | 0.8065 | 177 | 0.8079 | 118 | 0.7863 | 28 | 0.5768 | 256 |
| | 16 | 0.8092 | 181 | 0.7952 | 109 | 0.7872 | 27 | 0.5890 | 319 |
| | 20 | 0.8070 | 187 | 0.8003 | 91 | 0.7734 | 24 | 0.5843 | 397 |
| | 32 | 0.8112 | 184 | 0.8147 | 111 | 0.7925 | 29 | 0.6397 | 123 |
| | 64 | 0.8087 | 178 | 0.8121 | 109 | 0.8017 | 28 | 0.6458 | 394 |
| | 100 | 0.8072 | 183 | 0.8114 | 101 | 0.8068 | 27 | 0.7257 | 389 |
| | 200 | 0.8110 | 189 | 0.8146 | 112 | 0.8085 | 36 | 0.6410 | 135 |
| | 1000 | 0.8069 | 189 | 0.8140 | 116 | 0.8070 | 36 | 0.6374 | 395 |
| | 5000 | 0.8116 | 176 | 0.8093 | 105 | 0.8039 | 33 | 0.1001 | 120 |

## 4.4. Statistica Analysis of the Results

We use Welch's T-test to measure the significance of our method's results. We train Vanilla SN, Vanilla KD and KDDP models with 100 fc1 neurons, starting with different initial weights for 11 times. We set our

hyper-parameters as $L_1 = 1e^{-4}$, $\alpha = 0.5$. We present these results in Table 5.

**KDDP & Vanilla Analysis:** We start with assuming a null hypothesis that the mean of the results of the KDDP is equal to the mean of the results of the Vanilla network. Then, we calculate the T-score of these sets (classification results) using Eq. 5. We get a T-score of 9.91. For two-tailed hypothesis and 10 degrees of freedom, this T-score corresponds to $p < .00001$., which indicates statistical significant. Therefore, it is safe to reject the null hypothesis that there is no difference between the means of results.

**KDDP & Vanilla-KD Analysis:** We follow the same computations for comparing our KDDP with the Vanilla-KD. We get a T-score of 2.9730, which corresponds to $p = .013974$ for two-tailed hypothesis with 10 degrees of freedom. Since $p < .05$, it is again safe to reject the null hypothesis. We conclude that our KDDP model's performance has intrinsic differences from Vanilla SN and Vanilla KD results, and they are strong and are not by chance.

*Table 5. Results of 11 different trainings for Vanilla SN, Vanilla KD and KDDP models with 100 neurons in fc1. Hyper-parameters for KDDP are $L_1 = 1e^{-4}$, $\alpha = 0.5$. Although the difference in mean accuracies are small, they are statistically significant.*

| Run # | Vanilla Acc. | Vanilla-KD Acc. | KDDP Acc. |
|---|---|---|---|
| **0** | 0.8075 | 0.8197 | 0.8219 |
| **1** | 0.8041 | 0.8156 | 0.8226 |
| **2** | 0.8089 | 0.8182 | 0.8234 |
| **3** | 0.7975 | 0.8125 | 0.8166 |
| **4** | 0.8051 | 0.8172 | 0.8183 |
| **5** | 0.8101 | 0.8167 | 0.8175 |
| **6** | 0.7998 | 0.8210 | 0.8221 |
| **7** | 0.8080 | 0.8193 | 0.8212 |
| **8** | 0.8033 | 0.8163 | 0.8210 |
| **9** | 0.8098 | 0.8139 | 0.8152 |
| **10** | 0.7980 | 0.8169 | 0.8233 |
| **Mean** | 0.8047 | 0.8170 | 0.8203 |
| **Var** | $1.9522e^{-5}$ | $5.6783e^{-6}$ | $7.5033e^{-6}$ |

**KDDP & Vanilla T-score:** 9.9177
**KDDP & Vanilla-KD T-Score:** 2.9730

## 5. CONCLUSION

In this paper, we propose a new method based on Knowledge Distillation (KD) [17]. We use $L_1$ regularization on the activities of the neurons in a fully-connected layer and remove the inactive neurons. There is no need to provide the final size of the student model as input; our method determines it automatically. Our method performs better than the standard KD method with much fewer parameters.

In our extensive experiments, we show that KD based methods including ours are highly hyperparameters dependent. Temperature, $T$, and distillation weight, $\alpha$ selection determine the performance of the trained model. We observe that the accuracy varies significantly between low and high values for different $T$ values. Moreover, $\alpha$ constrains us to decide to what extent we should rely on the teacher network's logits. However, when the hyper-parameters are chosen carefully, our method works well. It performs better than the baselines.

In conclusion, our method can be used when there is a need for a much smaller network that performs comparably. Moreover, considering the benefits such as comparable accuracy with fewer parameters, one should expect that the hyper-parameter selection is vital for the performance.

Although we did not explore the use of our method for convolutional layers, we expect that similar gains (higher accuracy with fewer parameters) would be obtained. We leave this as future work.

## REFERENCES

[1] Y. LeCun, J. S. Denker, S. A. Solla, R. E. Howard, and L. D. Jackel, "Optimal brain damage.," in Advances in Neural Processing Systems (NIPS Conference), vol. 2, pp. 598–605, 1989.

[2] B. Hassibi and D. Stork, "Second order derivatives for network pruning: Optimal brain surgeon," in Advances in Neural Information Processing Systems 5 (NIPS Conference), pp. 164–171, 1992.

[3] S. Srinivas and R. V. Babu, "Data-free parameter pruning for deep neural networks," in Proceedings of the British Machine Vision Conference 2015, BMVC 2015, Swansea, UK, September 7-10, 2015, pp. 31.1–31.12, 2015

[4] 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 (NIPS Conference), pp. 1135–1143, 2015.

[5] H. Zhou, J. M. Alvarez, and F. Porikli, "Less is more: Towards compact cnns," in European Conference on Computer Vision (ECCV), pp. 662–677, Springer, 2016.

[6] 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 (NIPS Conference), pp. 2074–2082, 2016.

[7] 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 (ICCV), pp. 2736–2744, 2017.

[8] J. Jin, A. Dundar, and E. Culurciello, "Flattened convolutional neural networks for feedforward acceleration," in 3rd International Conference on Learning Representations, ICLR, San Diego, CA, USA, May 7-9, 2015, Workshop Track Proceedings, 2015.

[9] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size," CoRR, vol. abs/1602.07360, 2016.

[10] T. Li, B. Wu, Y. Yang, Y. Fan, Y. Zhang, and W. Liu, "Compressing convolutional neural networks via factorized convolutional filters," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 3977–3986, 2019.

[11] E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus, "Exploiting linear structure within convolutional networks for efficient evaluation," in Advances in Neural Information Processing Systems 27 (NIPS Conference), December 8-13, Montreal, Quebec, Canada, pp. 1269–1277, 2014.

[12] H. Kim, M. U. K. Khan, and C.-M. Kyung, "Efficient neural network compression," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 12569–12577, 2019.

[13] B. Minnehan and A. Savakis, "Cascaded projection: End-to-end network compression and acceleration," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 10715–10724, 2019.

[14] M. Courbariaux, Y. Bengio, and J. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," in Advances in Neural Information Processing Systems 28 (NIPS Conference), Montreal, Quebec, Canada, pp. 3123–3131, 2015.

[15] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding," in 4th International Conference on Learning Representations, ICLR, 2016.

[16] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks," in Advances in Neural Information Processing Systems (NIPS Conference), pp. 4107–4115, 2016.

[17] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," in Deep Learning Workshop, Advances in Neural Information Processing Systems (NIPS Conference), 2014.

[18] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio, "Fitnets: Hints for thin deep nets," in 3rd International Conference on Learning Representations, ICLR, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings, 2015.

[19] P. Luo, Z. Zhu, Z. Liu, X. Wang, and X. Tang, "Face model compression by distilling knowledge from neurons," in Thirtieth AAAI Conference on Artificial Intelligence, 2016.

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

[21] V. Lebedev and V. Lempitsky, "Fast convnets using group-wise brain damage," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 2554–2564, 2016.

[22] Z. Huang and N. Wang, "Data-driven sparse structure selection for deep neural networks," in Proceedings of the European Conference on Computer Vision (ECCV), pp. 304–320, 2018.

[23] K. Ullrich, E. Meeds, and M. Welling, "Soft weight-sharing for neural network compression," in 5th International Conference on Learning Representations, ICLR, Toulon, France, April 24-26, 2017, Conference Track Proceedings, 2017.

[24] S. J. Nowlan and G. E. Hinton, "Simplifying neural networks by soft weight-sharing," Neural Computation, vol. 4, no. 4, pp. 473–493, 1992.

[25] J. Ye, X. Lu, Z. Lin, and J. Z. Wang, "Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers," in 6th International Conference on Learning Representations, ICLR, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings, 2018.

[26] A. Beck and M. Teboulle, "A fast iterative shrinkage-thresholding algorithm for linear inverse problems," SIAM journal on imaging sciences, vol. 2, no. 1, pp. 183–202, 2009.

[27] 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 Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 1586–1595, 2018.

[28] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," arXiv preprint arXiv:1704.04861, 2017.

[29] Guo, A. Yao, and Y. Chen, "Dynamic network surgery for efficient dnns," in Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems, Barcelona, Spain, pp. 1379–1387, 2016.

[30] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," in 5th International Conference on Learning Representations, ICLR, Toulon, France, April 24-26, 2017, Conference Track Proceedings, 2017.

[31] J.-H. Luo, J. Wu, and W. Lin, "Thinet: A filter level pruning method for deep neural network compression," in Proceedings of the IEEE international conference on computer vision, pp. 5058–5066, 2017.

[32] 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, pp. 1389–1397, 2017.

[33] 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, pp. 9194–9203, 2018.

[34] A. Prakash, J. Storer, D. Florencio, and C. Zhang, "Repr: Improved training of convolutional filters," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 10666– 10675, 2019.

[35] H. Wang, Q. Zhang, Y. Wang, and H. Hu, "Structured probabilistic pruning for convolutional neural network acceleration," in British Machine Vision Conference 2018, BMVC, Northumbria University, Newcastle, UK, September 3-6, 2018, p. 149, 2018.

[36] Y. He, P. Liu, Z. Wang, Z. Hu, and Y. Yang, "Filter pruning via geometric median for deep convolutional neural networks acceleration," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 4340–4349, 2019.

[37] P. T. Fletcher, S. Venkatasubramanian, and S. Joshi, "Robust statistics on riemannian manifolds via the geometric median," in 2008 IEEE Conference on Computer Vision and Pattern Recognition, pp. 1–8, IEEE, 2008.

[38] X. Dong, S. Chen, and S. Pan, "Learning to prune deep neural networks via layer-wise optimal brain surgeon," in Advances in Neural Information Processing Systems, pp. 4857–4867, 2017.

[39] J. Kim, Y. Park, G. Kim, and S. J. Hwang, "Splitnet: Learning to semantically split deep networks for parameter reduction and model parallelization," in Proceedings of the 34th International Conference on Machine Learning-Volume 70, pp. 1866–1874, JMLR. org, 2017.

[40] T.-J. Yang, Y.-H. Chen, and V. Sze, "Designing energy-efficient convolutional neural networks using energy-aware pruning," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 5687–5695, 2017.

[41] C. Zhao, B. Ni, J. Zhang, Q. Zhao, W. Zhang, and Q. Tian, "Variational convolutional neural network pruning," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2780–2789, 2019.

[42] 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, pp. 535–541, ACM, 2006.

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

[44] J. Yim, D. Joo, J. Bae, and J. Kim, "A gift from knowledge distillation: Fast optimization, network minimization and transfer learning," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 4133–4141, 2017.

[45] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in European Conference on Computer Vision (ECCV), pp. 630–645, Springer, 2016.

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

[47] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in Proceedings of the thirteenth international conference on artificial intelligence and statistics, pp. 249–256, 2010.

[48] F. Chollet et al., "Keras." https://keras.io, 2015.

[49] H. Cai, L. Zhu, and S. Han, "Proxylessnas: Direct neural architecture search on target task and hardware," in 7th International Conference on Learning Representations, ICLR, New Orleans, LA, USA, May 6-9, 2019, 2019.