# Torque based Structured Pruning for Deep Neural Network

Arshita Gupta[*], Tien Bau[*], Joonsoo Kim, Zhe Zhu, Sumit Jha
Samsung Research America
{a7.gupta, t.bau, joonsoo.k, zhe.zhu, sumit.jha}@samsung.com

Hrishikesh Garud[†]
Google
hgarud@google.com

## Abstract

*Structured pruning is a popular way of convolutional neural network (CNN) acceleration. However, current state of the art pruning techniques require modifications to the network architecture, implementation of complex gradient update rules or repetitive training and long fine-tuning stages. Our novel physics-inspired approach for structured pruning aims to solve these issues. Analogous to 'Torque' we apply a force that consolidates the weights of a convolutional layer around a selected pivot point during training. Using the distance-dependency nature of torque, we can encourage high density of weights in filters around this point and increase filter sparsity as we move away. Filters away from the pivot point can be pruned, resulting in a minimum loss of information. We can control the tightness of the weights by varying the hyper-parameters, thus assisting us in creating a more compact network. Our proposed technique is jointly able to perform both filter learning and filter importance sorting. Additionally, our method is easy to implement, requires no change to model architecture and needs very little to no fine-tuning. We show that our approach reaches competitive results with previous state-of-the-art by evaluating popular networks such as VGGNet and ResNet on multiple image classification tasks. Notably, our method can reduce the parameter count of VGGNet by* 96% *and still maintain the accuracy achieved by the full-size model without any fine-tuning. This makes our method both latency and memory efficient for hardware deployment.*

## 1. Introduction

Recent years have seen a huge success of *Deep Neural Networks* (DNN) on a variety of computer vision tasks. Most of the research community favors accuracy more than efficiency which has led to dramatic increase in both dataset size and model complexity. From the application perspective, since a large part of the application scenarios for DNNs are hardware-constrained such as on smartphones, these models should be compact in terms of both power (number of floating point operations per second (FLOPs)) and memory (number of parameters) with minimum loss of performance. For ImageNet [7] task, early networks like AlexNet [27] and VGG-19 [46] have approximately $60M$ and $140M$ parameters and 0.72 and 19.6 billion FLOPs respectively, which are too large for practical applications. Currently, the most impressive Generative AI models [43] contains billions of parameters. How to smartly choose a good balance between effectiveness and efficiency is an active research direction.

There are several popular ways in model compression, such as quantization [4, 5, 49], low rank approximation [8, 52], distillation [21] and pruning [9, 10, 13, 30, 34]. More specifically, unlike unstructured pruning methods [9, 10, 13] that remove connections or neurons, our approach removes entire filters inside convolutional layers and can be categorized as *structured pruning* (SP) similar to [17, 30, 34].

Previous SP techniques like Network Slimming (NS) [34] requires additional layers like Batch Normalization added to their original model in order to implement their pruning process. However, many image based deep learning tasks, such as super-resolution, don't require or discourage the use of additional layers like normalizing layers during training and inference due to either added complexity or instability during inference. CNN-FCF [31], jointly learns and selects important filters. Although this technique is similar to our proposed work, it also requires an additional binary scalar. ThiNet [36] proposes selecting filters of $i^{th}$ layer based on the $(i+1)^{th}$ layer. While this method needs no architectural modifications, it requires step by step pruning of each layer followed by fine-tuning. More recent

[*]indicates equal first author contribution
[†]contributed during his time at Samsung Research America

pruning techniques [47, 48] either use complex gradient update rule or require in-depth understanding of second order derivatives which makes their work difficult to implement in one's pruning environment. State of art techniques like RL-MCTS [50] although exhibit plausible performance, come at the cost of integration of complex paradigms like reinforcement learning(RL) [50]. Some SP techniques [34, 36] require a trained model as a starting point. Most of the SP implementations consist of three phases [1] starting from training an over-parameterized model followed by pruning based on some criteria and finally fine-tuning the pruned model. This three step process is often time consuming, especially when a large percentage of the model has to be pruned in a multi-step manner using a combination of the above three steps.

To address the above mentioned limitations, we propose a novel approach for *structured pruning* inspired by a concept in physics called *torque*, which moves objects around a fixed axis. A key property is that increasing the radius from the axis to where the force is applied results in the increment of the torque. We borrow this idea and define torque as the constraint applied on the weights of the output filters of a convolutional layer during training. The further away a filter is from a selected fixed axis, the more constraint (torque) will be applied. This external force encourages a handful of filters around the fixed point of the convolution layer to do most of the heavy lifting, resulting in a high concentration of weights in these filters. Once a stable condition is reached during training, we can prune the less dense filters. Our proposed pruning method can reduce total parameters and still maintain good accuracy of the network. Additionally, it helps to keep the structural integrity of the network by maintaining the dense connections, leading to ease in implementation on hardware. An optional fine-tuning step after pruning can be applied to further boost the performance. Moreover, most of the current SP techniques [3, 33, 50] are trained for a specific model sparsity and have to be retrained if this requirement changes. In contrast, our method has no such sparsity requirement during the torque training. Once the model is trained, the re-aligned compressed weights allow pruning at various sparsities.

In summary we have made the following contributions: (1) We propose torque-based structured pruning which can effectively reduce FLOPS while maintaining high performance of the model. We compare our method with current state-of-the-art SP techniques on multiple datasets and show superior performance when compared with many recent approaches. Specifically on CIFAR datatset, we are able to achieve a high speedup of 2.6x with an insignificant drop in original accuracy. (2) The models pruned by our method tend to have smaller number of parameters under the same FLOPs reduction rate. This property is very useful in application scenarios where storage/memory resources are limited.(3) Our method is easy to implement and requires no alteration to original model architecture. Furthermore, it can be applied to either a randomly-initialized model or a pre-trained models. (4) Our torque trained model can be pruned at different levels of sparsity without the need to retrain. (5) Till a certain sparsity level, the compact model created by our method after the training shows high accuracy even without any additional fine-tuning step.

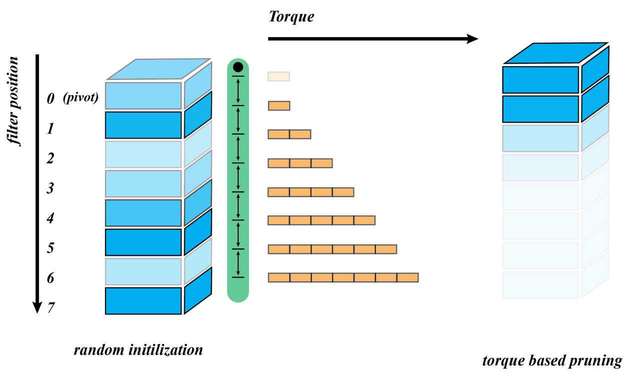


Figure 1. An illustration of torque based pruning method. Here a convolutional layer with 8 filters is used as an example (pivot marked). Vertical direction indicates the position of filters(order increases from top to bottom). The shade of the color indicates the magnitude of the weights. Filters on the left are randomly initialized. Then torque is enforced on each filter to push it towards zero. After training only a few filters have non-zero weights.

The rest of the paper is organized as follows. In Section 2, we discuss previous related work and in Section 3 we explain our proposed approach. We provide the experimental setup in section 4. This is followed by Section 5 where we elucidate the effects of hyper parameters in our method and discuss our results. Finally we conclude in Section 6.

## 2. Related Work

Neural network pruning has been an active area of research since 1980s. Early error sensitivity based methods calculated the correlation of change in the magnitude of each parameter to the error function. The intuition comes from the brute force pruning technique, also dubbed the "oracle pruning" method [39]. The idea of oracle pruning is to ablate each non-zero parameter and record the difference in error. The parameters with the least impact on the error can be removed because, evidently, they don't contribute substantially to the network's output. In order to determine this type of sensitivity of parameters in a more tractable manner, LeCun *et al*. [28] suggested calculating second derivative of the loss function with respect to each parameter. However, calculating the Hessian matrix is computationally expensive, especially for modern deep learning architectures.

**During training**: The unimportant parameters are penalized and eventually pruned during the process of training. We discuss some prior works in this section. In **Regularization-based pruning**, the idea is to induce sparsity using regularization based techniques. Some studies [35, 41] use an approximation of the $L0$ norm to penalize gating variables in order to learn a sparse dropout probability distribution function. Similarly, the $L1$ norm [34, 51] has also been explored. Another popular method is **Variational-based pruning** which utilizes powerful variational bayes approximation methods to learn the dropout probability distribution [6, 25, 37, 40]. Wen *et al.* [52] encourage filters within clusters that are coordinated to be closer during training based on the prior knowledge that correlation exists among trained filters in DNNs and those filters lie within several clusters. **Dropout sparsity learning**: The above mentioned studies can be classified as early attempts in this category. Newer research [11, 22, 29, 48, 53] show that use of regularization in pruning is still evolving and there is scope for more research.

**After training**: This method takes an over-parameterized network which is trained to completion and prunes it using certain heuristics. In **Magnitude-based pruning**, the parameters are pruned based on their absolute magnitude as there is a strong correlation between the sensitivity of the error function with respect to parameters and their absolute magnitudes. Parameters with smallest magnitude tend to have least impact on the network's performance. Another way to measure importance is using activations instead of weights [23, 42]. **Importance based pruning** methods calculate an importance metric for either individual parameters (unstructured pruning) or each channel (structured pruning) using certain heuristics. Aketi *et al.* [1] aim at pruning a network during the training process using a layer-wise relevance metric proposed by bach *et al.* [2]. Guo *et al.* [12] use a feature importance metric calculated on the training data to rank and prune features and corresponding channels. Authors of AMC [18] employ reinforcement learning to provide better layer-wise pruning ratios and prune channels according to the magnitude [14].

Unlike methods that use local threshold per layer [18, 20, 30], our proposed method is a combination of global-local structured pruning. Similar to magnitude based pruning, we prune an entire filter by comparing the sum of absolute weights of each filter (further normalized by the shape of filter) globally across all the filters throughout the network. In addition to the global thresholding, we set a small local constraint (Minimum Filter (MF) rule, explained in section 5.1) on each layer to control the amount of pruning in the layers. As far as we know, this type of pruning methodology is rare and hasn't been implemented in prior works. Previous structured pruning papers have a localized field of view for pruning where they compare filter magnitudes for each layer separately. Our method has the advantage of selecting which filters to remove in what part of the network. This is more flexible and lends perfectly to removing redundant filters in a denser part of the network.

## 3. Torque Methodology

### 3.1. Motivation

Neuropsychological studies has lead to the theory of 'engrams', a group of neurons that serve as 'physical representation of memory' in the brain. Richard Semon, a German zoologist who coined this term in early 1900s , believed in a physical location in the brain where the traces of memory were located [45]. Although, the existence of 'engrams' is an active area of research, we believe that creating such localized memory 'engrams' in neural network can help in concentrating most of the weights around a particular location. By applying a non-uniform force that grows with increasing distance from a fixed point, we can create a highly compressed network with increasing sparsity in the neurons as we move away from this fixed point. Such a network can recover faster when high sparsity part of the network (which is away from the fixed point) is damaged.

To achieve such a distance varying force, we exploit the concept of Torque from physics whose strength is dependent on the distance from the axis.

### 3.2. Proposed Approach

In physics, torque is a measure of force that causes an object to move about a given pivot point. The torque vector $T$ produced by any given force $F$ is

$$T = F \times r \tag{1}$$

where $r$ is the position vector and $F$ is the force vector.

Analogous to the above concept, we define torque $T$ in a convolutional neural network(CNN) as a force applied to each filter in the layer about a pivot point such that it drives the weights of filter with large torque towards zero. The strength of this force is proportional to both the magnitude of the filter weight and the relative distance between the filter and the pivot point.

To understand our torque method, let us first consider a CNN layer $l$ containing $N$ filters. Next, we define a pivot point which in our CNN implementation is filter $p$ where $p \in \{0, 1...N-1\}$. To define the torque for each filter $n$, we reshape the $n^{th}$ weight filter of the layer and denote it as $W_{ln} \in \mathbb{R}^{1 \times MHB}$ where $M$ is the number of channels and $H$ and $B$ are the spatial height and width of the filter respectively for $\forall n \in [0, N-1]$. As our torque method works for each layer independently, we will drop the $l$ from $W_{ln}$ and refer to it as $W_n$ for simplicity. Similarly, let us denote the weight of pivot filter as $W_p$.

Similar to Equation 1, we define the position vector $r'_n$ between any $n^{th}$ filter and $p$ in a CNN layer as a combination of two vectors $r_n$ and $d_n$ as shown:

$$r'_n = [w_r \cdot r_n, w_d \cdot d_n] \tag{2}$$

where $r'_n \in \mathbb{R}^{1\times(MHB+z)}$. Here both $r_n$ and $d_n$ are the relative distance vector between $n$ and $p$. $r_n$ is dependent on the weights of the filter and can be defined as

$$r_n = W_n - W_p \tag{3}$$

$d_n$ is defined as a monotonically increasing function of the filter index difference between $n$ and $p$. $w_r$ and $w_d$ are simple constants for $r_n$ and $d_n$ respectively.

In order to apply torque, we make both our position vector $r'_n$ and weight vector $W_n$ of same length by padding $W_n$ with zero vector. The new weight tensor, $W'_n$ is defined as:

$$W'_n = [W_n, \vec{0}] \tag{4}$$

where $\vec{0} \in \mathbb{R}^{1\times z}$ is a zero vector with $z$ length . We now define our Torque $T_n$ for the $n^{th}$ filter as:

$$T_n = W'_n \times r'_n \tag{5}$$

Since we only care about the magnitude of torque $T_n$ such that the weight filter with larger magnitude of torque is forced toward zero, we compute the magnitude of $T_n$ as:

$$||T_n||_2 = ||W'_n||_2 \cdot ||r'_n||_2 \cdot |\sin\theta_n| \tag{6}$$

where $\theta_n$ is the angle between the vectors $W'_n$ and $r'_n$. By adopting torque into the loss function as a regularization term, our new loss function can be defined as:

$$L = \Sigma_{n,l}\Sigma_{x,y}L_x(F(x, W_{ln}), y) + \lambda_T||T_{ln}||_2 \tag{7}$$

where $W_{ln}$ and $T_{ln}$ are the $n^{th}$ weight filter and its torque in the $l^{th}$ layer in the network. $L_x()$ is the original loss function defined on each training sample ($x$) and $y$ is the label of the sample. $\lambda_T$ is defined as a torque rate controlling the contribution of the torque term. Once again, we drop the $l$ from the above equation for simplicity,

$$L = \Sigma_n\Sigma_{x,y}L_x(F(x, W_n), y) + \lambda_T||T_n||_2 \tag{8}$$

Minimizing the above loss will force the weight filter with larger magnitude of torque toward zero. In other words, we can compress all the weights to a small number of filters near the pivot filter in the layer. From Equations 5, 4 and 3, we know that $||T_n||_2$ constitutes of both $W_n$ and $r_n$ terms. Thus, we observe that minimizing $||T_n||_2$ not only minimizes the weight filters but also reduces $r_n$. Minimizing $r_n$ can cause $W_n$ and $W_p$ to be pulled closer together, which results in two filters learning almost identical weights ($W_n \approx W_p$). This can lead to a decrease in learning diverse features and furthermore impact the overall accuracy of the network. To solve this problem, we decide to ignore $r_n$ by setting constants $w_r$ = 0 and $w_d$ = 1 such that the weights are compressed near the pivot filter as well as each dense filter is learning diverse features. Then, Equation 6 can be rewritten as:

$$||T_n||_2 = ||W'_n||_2 \cdot ||r'_n||_2 \cdot |\sin\theta_n| = ||W_n||_2 \cdot ||d_n||_2 \tag{9}$$

From Equation 9, it is evident that the weight filters close to the pivot filter would be more dense while the filters far from the pivot filter would be sparse or zero. We adopt the above method into our training environment by modifying the gradient update rule. Since the effectiveness of $||T_n||_2$ and $||T_n||_1$ are similar but the $||T_n||_2$ is more computationally expensive, we choose $||T_n||_1 = ||W_n||_1 \cdot ||d_n||_1$ in the gradient update rule. Typically, the updating of weights by gradient descent for each filter in a given layer is given as:

$$W_n \leftarrow W_n - \eta \cdot (\frac{\partial L(W)}{\partial W_n}) \tag{10}$$

where $\eta$ is the learning rate, $L(W)$ is the data loss. We introduce our torque method by adding an extra penalty term to the gradient update rule.

$$W_n \leftarrow W_n - \eta \cdot \left(\frac{\partial L(W)}{\partial W_n} + \lambda_T \cdot \frac{\partial ||T_n||_1}{\partial W_n}\right) \tag{11}$$

Inserting $||T_n||_1 = ||W_n||_1 \cdot ||d_n||_1$ in the above equation, we get

$$W_n \leftarrow W_n - \eta \cdot \left(\frac{\partial L(W)}{\partial W_n} + \lambda_T \cdot \frac{\partial(||W_n||_1 \cdot ||d_n||_1)}{\partial W_n}\right) \tag{12}$$

$$W_n \leftarrow W_n - \eta \cdot \left(\frac{\partial L(W)}{\partial W_n} + \lambda_T \cdot (||d_n||_1 \cdot (W_n)\right) \tag{13}$$

where () is the sign function. The effect of this torque modified updated rule pushes the majority of the filters that are away from the pivot filter to be zero and forces the filters near the pivot filter to be dense as also illustrated in Figure 1. A combination of both these values decides the degree of compression in our models. More discussions on the $d_n$ and $\lambda_T$ and their implications are given in section 4.3.1.

## 4. Experimental Setup

In this section, we discuss the different experiments we run to show the efficacy of our proposed approach. We assess our approach by training various models on image classification task which is a standard computer vision task.

### 4.1. Dataset

In order to compare our approach with recent pruning approaches, we train our models on standard benchmark datasets for classification: **CIFAR [26]:** CIFAR-10 and CIFAR-100 datasets contain $32 \times 32$ color images divided into 10 and 100 classes respectively. Each dataset contains a training set of $50,000$ and a test set of $10,000$. **ImageNet [44]:** The ILSVRC is a large scale image dataset which consists of 1.2 million images divided in 1000 classes. We adopt data augmentation methods like random flipping, random crop on the images. We also normalize the RGB values using channel means and standard deviations. For ImageNet, we randomly crop the images to $224 \times 224$.

## 4.2. Network Architectures

To study the process of torque based training on different architectures, we consider family of CNN architectures with linear connections only (VGG [46]), with residual connections (ResNet [15]) as well as architectures with Dense connections (DenseNet [24]). Specifically we experiment with VGG-19 [46], the bottleneck structure of ResNet [15] with 164 layers and a 40 layer DenseNet model (used with a growth rate of 12) all trained on both CIFAR-10 and CIFAR-100 datasets. Moreover, we implement torque methodology on ResNet-56 model on CIFAR-10 and bottleneck structure of ResNet-50 on ImageNet and compare our results with the current most popular structured pruning technique in academia. These studies help us demonstrate that the approach works in wide range of architectures.

## 4.3. Implementation

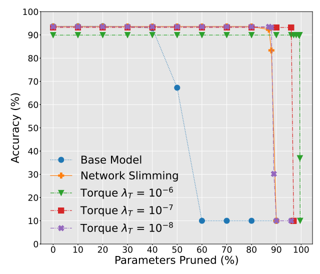
In this section, we discuss each of the steps in our implementation pipeline of obtaining the final pruned model. First, we train our model with torque to encourage sparsity in filters followed by pruning the sparse filters. Optionally, we can finetune our pruned model.

### 4.3.1 Training

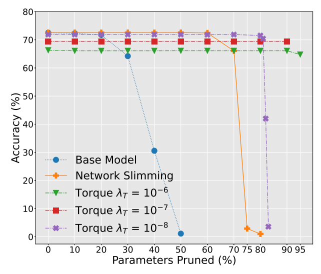
As observed from Equation 13, a combination of both $d_n$ and $\lambda_T$ can help in deciding the compactness of our layer. For simplicity and without loss of generality, we choose our pivot channel $p$ as the first output filter, that is $p = 0$ for all the layers. For $d_n$, we pick a monotonically increasing linear ramp function, that is $|d_n| = n, \forall n \in \{0, 1, ..N-1\}$. We experimented with logarithmic and exponential functions, but the resultant torque force seemed too high. This caused the training to stall and the parameters getting stranded in a local minima. In the following experiments, we fix $d_n$ and experiment with different values of $\lambda_T$. The effects of different $\lambda_T$ are studied in detail in Section 5.2.

Furthermore, we experiment with two different initialization points for our network. We investigate the implications of torque on both random initialization (r) and initialization with a pre-trained (p) model weights.

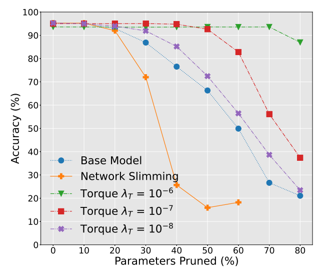
All models are trained with stochastic gradient descent with learning rate (lr) of 0.1 for randomly initialization models and 0.01 for models initialized with pre-trained weights. The lr is reduced by 0.1 after 100 and 150 epochs. The model is trained with torque for a total of 200 epochs and an additional weight decay of $10^{-4}$ is used. No pruning (sparsity) ratio is required at the time of training.



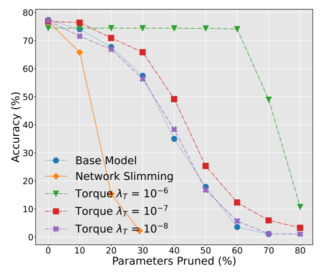
(a) VGG-19 on CIFAR-10



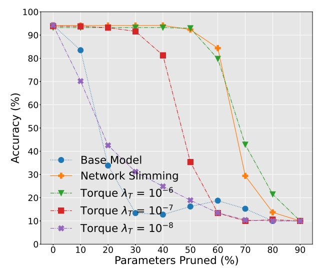
(b) VGG-19 on CIFAR-100



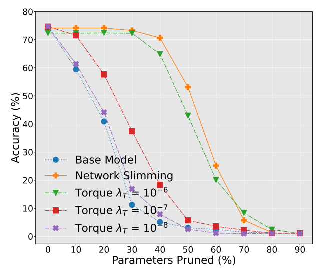
(c) ResNet-164 on CIFAR-10



(d) ResNet-164 on CIFAR-100



(e) DenseNet-40 on CIFAR-10



(f) DenseNet-40 on CIFAR-100

Figure 2. Test accuracy drop with respect to different amounts of pruning without any fine-tuning. For all the network architectures, the Torque method applied is with random initialization and MF value for pruning is set to 5 filters.

### 4.3.2 Pruning

As described earlier in Section 2, we use a 'global-local' structured pruning technique. In our experiments we noticed that at high pruning percentage, it is possible that an entire layer is removed if the sum of absolute weights of all filters in that layer is below a set global threshold. For maintaining the structure of our networks, we impose the minimum filter (MF) rule, which ensures that every layer has at least the minimum number of filters. One single MF value is applied to all the layers of the network.

### 4.3.3 Fine-tuning

While our model shows good performance without fine-tuning, the resultant pruned model can be subjected to an optional fine-tuning step. For fair comparison with baselines, we fine-tune our pruned model and remove the torque constraint as it was a training technique required only for pruning and is not mandatory anymore.

## 5. Analysis and Result

In the following subsections, we will analyze and discuss our results on different architectures and datasets. We compare our method with other structured pruning techniques with and without fine-tuning in Section 5.1 and Section 5.3 respectively. Additionally, we study the implications of using different $\lambda_T$ values during training in Section 5.2.

### 5.1. Effect of Pruning (Before Fine-tuning)

In this section we discuss the effects of different amounts of pruning on the network's performance. Specifically, we analyze the top 1 accuracy drop as we increase the total percentage of parameters pruned for different network architectures and datasets 'before' any fine-tuning step. Here MF is set to 5 filters.

In Figure 2, we consider 3 different models namely VGG-19, ResNet-164 and DensetNet-40, each of which is trained on both CIFAR-10 and CIFAR-100. Each of our models is trained with torque at 3 different $\lambda_T$ values, $\lambda_T \in \{10^{-6}, 10^{-7}, 10^{-8}\}$. Additionally, to show the significance of our Torque method, we compare our results with a Base Model and a model trained (and pruned) using NS [34] technique. 'Base model' means that we train a model in the same training environment but with no torque. However, we apply the same pruning procedure mentioned in Section 5.1 for fair comparison.

From Figure 2, we observe that in most cases, the 'base model' is the first to show a drop in accuracy as the number of pruned parameters are increased. The performance of NS is better than base for VGG-19 and DenseNet-40 but proves to be a bad choice for ResNet type architectures. In comparison, our torque trained networks are able to achieve and maintain a higher accuracy even after substantial parameters are pruned for all architectures. Specifically, for VGG-19 on CIFAR10, **without any fine-tuning**, all our torque trained networks show almost no loss in accuracy when subjected to a reduction in the total parameters by 96%.

To understand our results theoretically we refer to Figure 3. In the left figure, when no torque during training, the weights are distributed randomly throughout the layers. If magnitude based structured pruning is used to remove least important filters, we might lose substantial information and furthermore it will result in long fine-tuning period as the rest of the filters have to compensate for the lost details.

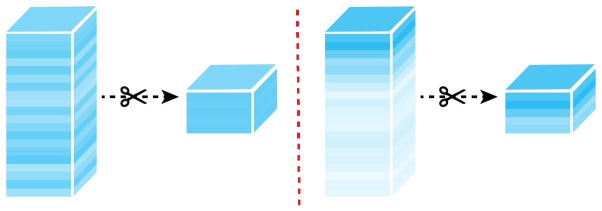


Figure 3. An illustration of weight matrix of trained layer N before and after pruning trained without (left) and with (right) torque.

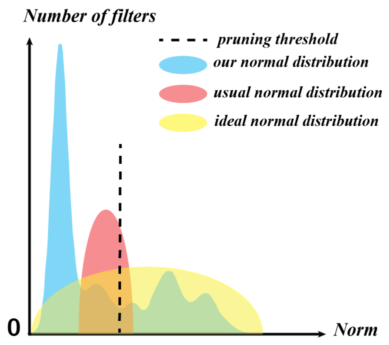


Figure 4. Usual (red region), Ideal (yellow region) and Our norm distribution (blue region). The black dashed line is the threshold used for pruning. This figure clearly proves that it is not hard to choose a threshold from our norm distribution.

In contrast, when trained with torque as shown in the same figure on the left, we observe weights compressed around a few filters, that is, only a few filters in each layer are doing all the learning. In this case, if least important filters are removed, we lose very less information and the time to fine-tune is either completely removed or reduced.

Moreover, FPGM [19] laid out two important points where most 'smaller-norm less-information' pruning techniques fail. The first argument is that, in practice, the filter norm distribution might have small standard deviation, making it difficult to accurately set a pruning threshold. The second argument is that weights with the smallest norm might still have significant impact on the network performance because their actual magnitudes are still large. On further analysis, our method results in a filter norm distribution with the largest peak corresponding to filters with smallest norm as shown in 'blue region' in Figure 4. Additionally, the large deviation makes it easy to set a threshold for pruning (black dotted line in the Figure 4). Consequently, our method also ensures that the filter weights with minimum norm actually are small in magnitude.

**Above experiments show that our methodology gives noteworthy performance even on heavily pruned models without the need of any fine-tuning.** The implications of using different $\lambda_T$ values is discussed in the next section.

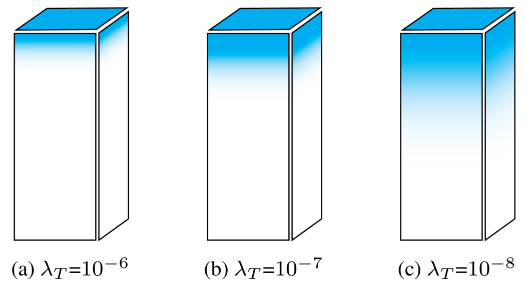

(a) $\lambda_T$=$10^{-6}$
(b) $\lambda_T$=$10^{-7}$
(c) $\lambda_T$=$10^{-8}$

Figure 5. An illustration of the weights distribution of the $14^{th}$ layer of VGG-19 network trained with different Torque Rate $\lambda_T$. Blue indicates non-zeros weights (filters) while white indicates zero weights.

### 5.2. Effect of Torque rate $\lambda_T$

One important hyper-parameter in our method is the Torque rate $\lambda_T$ which controls the amount of extra gradient that is added during back propagation as shown in Equation 13. In short, it defines the strength of force with which the weights are pushed towards the pivot point and compressed.

Figure 5 shows the effect of different $\lambda_T$ on layer 14 of VGG19 network after training is completed. Higher the torque rate, the more they are forced to concentrate near the pivot point.

The real effect of the different $\lambda_T$ is seen during the pruning step. In Figure 2, we are clearly able to distinguish the different networks trained on different $\lambda_T$. Here at $\lambda_T = 10^{-8}$, although had given us the highest accuracy, we observed a drop in accuracy after approximately 90% of the parameters were pruned. On the other hand at $\lambda_T = 10^{-6}$, although the accuracy reached while training was 90%, there was hardly any drop in accuracy event after 98% of the parameters of the model were pruned. This is because using $\lambda_T = 10^{-6}$, in contrast to $10^{-8}$, tries to concentrate the weight density heavily on the first few initial filters, thus leading to a large number of filters being left sparse. This is also demonstrated in Figure 5.

### 5.3. Results with Fine-tuning

In this subsection, we analyse the performance of our method at different speedups (ratio of flops in the original model to number of flops in pruned model) and compare it with previous state-of-the-art (SOTA) SP techniques. Additionally, as mentioned in Section 4.3.1, we use two different initialization points for our torque trained networks. In all our results in Table 1, (r) and (p) corresponds to random initialization and initialization from pre-trained respectively. We provide the performance of our models for different $\lambda_T$ in the supplementary material.

Although our current setting of MF is optimum, we noticed a drastic reduction in parameters as we reduce flops in order to compare with other competitive techniques. In our experiments, we noticed that torque creates more sparse filters as we go deeper in the network which results in higher percentage of filter reduction in these layers. In theory this would work, but in the architectures mentioned in this paper for comparison, the use of 'strided' and 'maxpool' layers reduce the size of the feature map as we go deeper in the network. Thus, pruning filters from deep layers reduces parameter count but does not reduce flop count to the same extent as when pruning filters from a shallower layer. Taking this into consideration, we updated our MF for each layer to be 30% of the total number of filters in that layer.

**ResNet-56 on CIFAR-10:** In Table 1a, we present the different SP techniques on ResNet-56 for CIFAR-10 classification task. For a thorough comparison, we analyze the performance at various different speed-ups. Authors of GReg-1 [48] proposes a variant of L2 regularization with increasing penalty factors. Additionally, they introduce GReg-2 [48] approach which is based on exploiting the Hessian information without dealing with their exact values and thus avoiding the common problems related to Hessian approximation. However, their main focus is mainly in model acceleration rather than compression. We observe here that our torque (both (r) and (p)) outperform both GReg-1 and GReg-2. Specifically, Torque (p) surpasses GReg-1 and GReg-2 by a margin of $\approx 0.5\%$ in drop in accuracy at $2.15\times$ and $\approx 0.6\%$ drop in accuracy at $2.6\times$ with a much larger parameters pruned. ABCPruner [33] proposes to search for the optimal pruned structure by uncovering channel number in each layer. Additionally, they start with shrinking the total combinations of pruned structures in order to efficiently search for their desired structure. However, we believe that such a constraint by the user requires deep knowledge of the network being pruned and changes with different architectures. In torque, because there is no such restriction, the model is free to explore the different combinations of channel per layer throughout its training process. Moreover, our 'global-local' pruning technique ensures optimal number of channels are pruned in each layer. As shown in the table, our approach outperforms ABCPruner by securing $\approx 0.3\%$ lower accuracy drop with a higher speedup of $2.23\times$ and much larger number of parameters pruned. The authors of WHC [3] present a SP technique by considering both magnitude of channels as well as dissimilarity between channel pairs to recognize redundant filters. However, we attribute the superiority of our Torque (p) results over [3] on the fact that we ensure dissimilarity between channels during the training itself as also explained in section . Thus we are able to achieve a 0.2% less accuracy drop as compared to WHC at speed up of $2.23\times$. We also observe that although RL-MCTS [50] performs marginally better than our method, Torque method is much easier to implement and requires no additional RL network. Moreover, our method is faster to implement in

| Model | Base Acc. (%) | Pruned Acc. (%) | Acc. drop (%) | Speed Up | Pruned params (%) |
|---|---|---|---|---|---|
| AMC [18] | 92.80 | 91.90 | 0.90 | 2.00× | - |
| CP [20] | 92.80 | 91.80 | 1.00 | 2.00× | - |
| FPGM [19] | 93.59 | 93.26 | 0.33 | 2.11× | - |
| SFP [17] | 93.59 | 93.36 | 0.23 | 2.11× | - |
| WHC [3] | 93.59 | 93.47 | 0.12 | 2.11× | - |
| LFPC [16] | 93.59 | 93.24 | 0.35 | 2.12× | - |
| GReg-1 [48] | 93.51 | 93.25 | 0.26* | 1.99× | 49.82 |
| GReg-2 [48] | 93.51 | 93.28 | 0.23 | 1.99× | 49.82 |
| Torque (r) (ours) | 93.48 | 93.28 | 0.20 | 2.15× | 63.21 |
| Torque (p) (ours) | 93.48 | 93.76 | **-0.28** | **2.15×** | **65.00** |
| ABC Pruner [33] | 93.26 | 93.23 | 0.03 | 2.18× | 54.20 |
| WHC [3] | 93.59 | 93.66 | -0.07 | 2.21× | - |
| RL-MCTS [50] | 93.2 | 93.56 | **-0.36** | 2.22× | - |
| Torque (r) (ours) | 93.48 | 93.17 | 0.31 | 2.22× | 63.75 |
| Torque (p) (ours) | 93.48 | 93.73 | -0.25 | **2.23×** | 65.64 |
| GReg-1 [48] | 93.51 | 92.92 | 0.59* | 2.55× | 42.76 |
| GReg-2 [48] | 93.51 | 92.85 | 0.66* | 2.55× | 42.76 |
| Torque (r) (ours) | 93.48 | 93.06 | 0.42 | **2.61×** | 66.16 |
| Torque (p) (ours) | 93.48 | 93.40 | **0.08** | **2.60×** | **66.85** |
| WHC [3] | 93.59 | 93.29 | 0.30 | 2.71× | - |
| Torque (r) (ours) | 93.48 | 92.59 | 0.89 | 2.72× | 67.39 |
| Torque (p) (ours) | 93.48 | 93.26 | **0.22** | **2.72×** | **67.44** |

(a) Results of ResNet-56 on CIFAR10

| Model | Base Acc (%) | Pruned Acc (%) | Acc. drop (%) | Speed Up | Pruned params (%) |
|---|---|---|---|---|---|
| Kron-OBD [47] | 73.34 | 60.70 | 12.64 | 5.73× | 82.55 |
| Kron-OBS [47] | 73.34 | 60.66 | 12.68 | 6.09× | 83.57 |
| ED [47] | 73.34 | 65.18 | 8.16 | 8.80× | 88.63 |
| GReg-1 [48] | 74.02 | 67.55 | 6.47 | 8.84× | **90.98** |
| Greg-2 [48] | 74.02 | 67.75 | 6.27 | 8.84× | **90.98** |
| Torque (r) (ours) | 73.03 | 65.87 | 7.16 | 8.88× | 90.83 |
| Torque (p) (ours) | 73.03 | 63.91 | **6.12** | **8.89×** | 90.81 |

(b) Results of VGG-19 on CIFAR100

| Model | Base Acc. (%) | Pruned Acc. (%) | Acc. drop (%) | Speed up | Pruned params (%) |
|---|---|---|---|---|---|
| SFP [17] | 76.15 | 74.61 | 1.54 | 1.72× | - |
| HRank [32] | 76.15 | 74.98 | 1.17 | 1.78× | 36.81 |
| TaylorFO [38] | 76.18 | 74.50 | 1.68 | 1.82× | 44.44 |
| RL-MCTS [50] | 77.34 | 76.8 | **0.54** | 1.85× | - |
| Torque (p) (ours) | 76.07 | 75.07 | 1.00 | **2.05×** | **62.41** |
| ThiNet [36] | 75.3 | 72.03 | 3.27 | 2.26× | 51.56 |
| ABC Pruner [33] | 76.01 | 73.52 | 2.49 | 2.30× | 56.01 |
| CNN-FCF [31] | 76.15 | 74.55 | 1.6 | 2.33× | 52.52 |
| Torque (p) (ours) | 76.07 | 74.58 | **1.49** | **2.34×** | **64.50** |

(c) Results of ResNet-50 on ImageNet

Table 1. Results with respect to speedup and pruned parameters. (r) and (p) corresponds to random initialization and initialization from pre-trained models respectively.

terms of time required to train and prune model at different speedups. For even higher speed-ups, Torque initialized with a pre-trained model outperforms all previous techniques by a large amount along with large percentage of parameters pruned. This evidently proves that our torque method is able to compete with state-of-the-art accuracy for a given ResNet model on cifar-10 classification task and is both latency as well as memory efficient.

**VGG-19 on CIFAR-100:** Next, we evaluate torque for VGG-19 trained on CIFAR-100 in Table 1b. Similar to Table 1a, we observe that Torque (p) is able to surpass both GReg-1 and GReg-2 by a margin of $\approx 0.15\%$ accuracy drop at an even higher speed-up.

In most of our experiments, we have observed that starting from pre-trained weights gives better accuracy. Our understanding is that for the same number of epochs, it is advantageous for the model to be at a good minima in order to absorb any damage done during the torque training process.

**ResNet-50 on ImageNet:** To further examine the efficacy of our method, we test our methodology on a larger dataset. In Table 1c, we compare our results with previous SP techniques including HRank [32] which uses the rank of the feature map as a criteria for pruning. Our method surpasses majority of the previous complex methods, all while attaining a much higher speed-up and more parameters pruned. Additionally by using our easily adaptable Torque technique, ResNet-50 trained on ImageNet can be subjected to a pruning percentage of more than 60% of original parameters and a speed up of as high as 2.34x (less than 50% of total FLOPS) and still experience only a marginal drop in accuracy of less than $1.5\%$ of the original model.

## 6. Conclusion

We propose a novel approach for structured pruning without altering the CNN model architecture. We have demonstrated both theoretically from Figure 3 and empirically from Figure 2 that our torque technique helps to shrink the fine-tuning step, by either completely removing or minimizing the time required in the process. Extensive experiments indicate that our approach outperforms majority of the previous structured pruning methods with higher speed-up rate and lower accuracy drop. At a given speed up, our approach prunes more parameters than our competitors which is beneficial for more constrained deployment. We also investigate the distribution of the norm of the weights produced by our approach in Figure 4, which gives an intuition why this simple method works well. In the future, we plan to extend our method to other layers, such as attention, to further expand its application especially in generative AI.

*indicates that these values were obtained from running their open source code and differs from what was reported in their paper.

## References

[1] Sai Aparna Aketi, Sourjya Roy, Anand Raghunathan, and Kaushik Roy. Gradual channel pruning while training using feature relevance scores for convolutional neural networks. *IEEE Access*, 8:171924–171932, 2020. 2, 3

[2] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one*, 10(7):e0130140, 2015. 3

[3] Shaowu Chen, Weize Sun, and Lei Huang. Whc: Weighted hybrid criterion for filter pruning on convolutional neural networks. In *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1–5. IEEE, 2023. 2, 7, 8

[4] Wenlin Chen, James Wilson, Stephen Tyree, Kilian Weinberger, and Yixin Chen. Compressing neural networks with the hashing trick. In *International conference on machine learning*, pages 2285–2294. PMLR, 2015. 1

[5] Brian Chmiel, Liad Ben-Uri, Moran Shkolnik, Elad Hoffer, Ron Banner, and Daniel Soudry. Neural gradients are near-lognormal: improved quantized and sparse training. *arXiv preprint arXiv:2006.08173*, 2020. 1

[6] Bin Dai, Chen Zhu, Baining Guo, and David Wipf. Compressing neural networks using the variational information bottleneck. In *International Conference on Machine Learning*, pages 1135–1144. PMLR, 2018. 3

[7] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009. 1

[8] Emily L Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. *Advances in neural information processing systems*, 27, 2014. 1

[9] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018. 1

[10] Trevor Gale, Erich Elsen, and Sara Hooker. The state of sparsity in deep neural networks. *arXiv preprint arXiv:1902.09574*, 2019. 1

[11] Aidan N Gomez, Ivan Zhang, Siddhartha Rao Kamalakara, Divyam Madaan, Kevin Swersky, Yarin Gal, and Geoffrey E Hinton. Learning sparse networks using targeted dropout. *arXiv preprint arXiv:1905.13678*, 2019. 3

[12] Jinyang Guo, Wanli Ouyang, and Dong Xu. Channel pruning guided by classification loss and feature importance. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 10885–10892, 2020. 3

[13] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015. 1

[14] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. *Advances in neural information processing systems*, 28, 2015. 3

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016. 5

[16] Yang He, Yuhang Ding, Ping Liu, Linchao Zhu, Hanwang Zhang, and Yi Yang. Learning filter pruning criteria for deep convolutional neural networks acceleration. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2009–2018, 2020. 8

[17] Yang He, Guoliang Kang, Xuanyi Dong, Yanwei Fu, and Yi Yang. Soft filter pruning for accelerating deep convolutional neural networks. *arXiv preprint arXiv:1808.06866*, 2018. 1, 8

[18] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European conference on computer vision (ECCV)*, pages 784–800, 2018. 3, 8

[19] Yang He, Ping Liu, Ziwei Wang, Zhilan Hu, and Yi Yang. Filter pruning via geometric median for deep convolutional neural networks acceleration. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4340–4349, 2019. 6, 8

[20] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE international conference on computer vision*, pages 1389–1397, 2017. 3, 8

[21] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network, 2015. 1

[22] Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *J. Mach. Learn. Res.*, 22(241):1–124, 2021. 3

[23] Hengyuan Hu, Rui Peng, Yu-Wing Tai, and Chi-Keung Tang. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *arXiv preprint arXiv:1607.03250*, 2016. 3

[24] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017. 5

[25] Durk P Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick. *Advances in neural information processing systems*, 28, 2015. 3

[26] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009. 5

[27] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012. 1

[28] Yann LeCun, John Denker, and Sara Solla. Optimal brain damage. *Advances in neural information processing systems*, 2, 1989. 2

[29] Carl Lemaire, Andrew Achkar, and Pierre-Marc Jodoin. Structured pruning of neural networks with budget-aware regularization. In *Proceedings of the IEEE/CVF Conference*

*on Computer Vision and Pattern Recognition*, pages 9108–9116, 2019. 3

[30] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016. 1, 3

[31] Tuanhui Li, Baoyuan Wu, Yujiu Yang, Yanbo Fan, Yong Zhang, and Wei Liu. Compressing convolutional neural networks via factorized convolutional filters. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3977–3986, 2019. 1, 8

[32] Mingbao Lin, Rongrong Ji, Yan Wang, Yichen Zhang, Baochang Zhang, Yonghong Tian, and Ling Shao. Hrank: Filter pruning using high-rank feature map. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1529–1538, 2020. 8

[33] Mingbao Lin, Rongrong Ji, Yuxin Zhang, Baochang Zhang, Yongjian Wu, and Yonghong Tian. Channel pruning via automatic structure search. *arXiv preprint arXiv:2001.08565*, 2020. 7, 8

[34] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017. 1, 2, 3, 6

[35] Christos Louizos, Max Welling, and Diederik P Kingma. Learning sparse neural networks through $l\_0$ regularization. *arXiv preprint arXiv:1712.01312*, 2017. 3

[36] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. In *Proceedings of the IEEE international conference on computer vision*, pages 5058–5066, 2017. 1, 2, 8

[37] Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov. Variational dropout sparsifies deep neural networks. In *International Conference on Machine Learning*, pages 2498–2507. PMLR, 2017. 3

[38] Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. Importance estimation for neural network pruning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11264–11272, 2019. 8

[39] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. *arXiv preprint arXiv:1611.06440*, 2016. 2

[40] Kirill Neklyudov, Dmitry Molchanov, Arsenii Ashukha, and Dmitry P Vetrov. Structured bayesian pruning via log-normal multiplicative noise. *Advances in Neural Information Processing Systems*, 30, 2017. 3

[41] Wei Pan, Hao Dong, and Yike Guo. Dropneuron: Simplifying the structure of deep neural networks. *arXiv preprint arXiv:1606.07326*, 2016. 3

[42] Adam Polyak and Lior Wolf. Channel-level acceleration of deep face representations. *IEEE Access*, 3:2163–2175, 2015. 3

[43] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10684–10695, 2022. 1

[44] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. 5

[45] R Semon. The mneme london, 1921. 3

[46] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 1, 5

[47] Chaoqi Wang, Roger Grosse, Sanja Fidler, and Guodong Zhang. Eigendamage: Structured pruning in the kronecker-factored eigenbasis. In *International conference on machine learning*, pages 6566–6575. PMLR, 2019. 2, 8

[48] Huan Wang, Can Qin, Yulun Zhang, and Yun Fu. Neural pruning via growing regularization. *arXiv preprint arXiv:2012.09243*, 2020. 2, 3, 7, 8

[49] Naigang Wang, Jungwook Choi, Daniel Brand, Chia-Yu Chen, and Kailash Gopalakrishnan. Training deep neural networks with 8-bit floating point numbers. *Advances in neural information processing systems*, 31, 2018. 1

[50] Zi Wang and Chengcheng Li. Channel pruning via lookahead search guided reinforcement learning. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 2029–2040, 2022. 2, 7, 8

[51] Wei Wen, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Learning structured sparsity in deep neural networks. *Advances in neural information processing systems*, 29, 2016. 3

[52] Wei Wen, Cong Xu, Chunpeng Wu, Yandan Wang, Yiran Chen, and Hai Li. Coordinating filters for faster deep neural networks. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 658–666, 2017. 1, 3

[53] Tao Zhuang, Zhixuan Zhang, Yuheng Huang, Xiaoyi Zeng, Kai Shuang, and Xiang Li. Neuron-level structured pruning using polarization regularizer. *Advances in neural information processing systems*, 33:9865–9877, 2020. 3