Importance Estimation for Neural Network Pruning

Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, Jan Kautz
NVIDIA
{pmolchanov, amallya, styree, ifrosio, jkautz}@nvidia.com

Abstract

Structural pruning of neural network parameters reduces computation, energy, and memory transfer costs during inference. We propose a novel method that estimates the contribution of a neuron (filter) to the final loss and iteratively removes those with smaller scores. We describe two variations of our method using the first and second-order Taylor expansions to approximate a filter's contribution. Both methods scale consistently across any network layer without requiring per-layer sensitivity analysis and can be applied to any kind of layer, including skip connections. For modern networks trained on ImageNet, we measured experimentally a high (>93%) correlation between the contribution computed by our methods and a reliable estimate of the true importance. Pruning with the proposed methods leads to an improvement over state-of-the-art in terms of accuracy, FLOPs, and parameter reduction. On ResNet-101, we achieve a 40% FLOPS reduction by removing 30% of the parameters, with a loss of 0.02% in the top-1 accuracy on ImageNet. Code is available at `https://github.com/NVlabs/Taylor_pruning`.

1. Introduction

Convolutional neural networks (CNNs) are widely used in today's computer vision applications. Scaling up the size of datasets as well as the models trained on them has been responsible for the successes of deep learning. The dramatic increase in number of layers, from 8 in AlexNet [20], to over 100 in ResNet-152 [11], has enabled deep networks to achieve better-than-human performance on the ImageNet [31] classification task. Empirically, while larger networks have exhibited better performance, possibly due to the lottery ticket hypothesis [5], they have also been known to be heavily over-parameterized [34].

The growing size of CNNs may be incompatible with their deployment on mobile or embedded devices, with limited computational resources. Even in the case of cloud services, prediction latency and energy consumption are important considerations. All of these use cases will benefit greatly from the availability of more compact networks. Pruning is a common method to derive a compact network – after training, some structural portion of the parameters is removed, along with its associated computations.

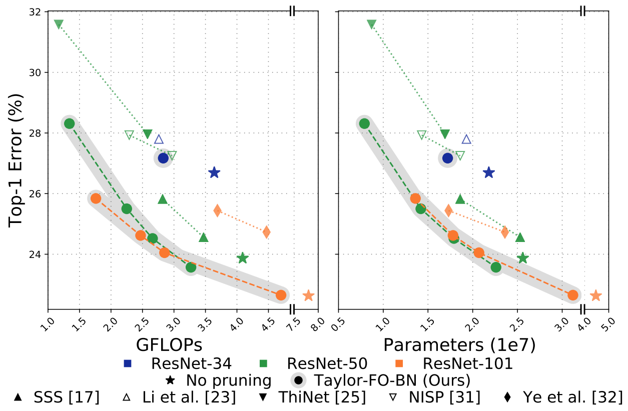

Figure 1: Pruning ResNets on the ImageNet dataset. The proposed method is highlighted in gray. Bottom-left is better.

A variety of pruning methods have been proposed, based on greedy algorithms [26, 33], sparse regularization [21, 23, 32], and reinforcement learning [13]. Many of them rely on the belief that the magnitude of a weight and its importance are strongly correlated. We question this belief and observe a significant gap in correlation between weight-based pruning decisions and empirically optimal one-step decisions – a gap which our greedy criterion aims to fill.

We focus our attention on extending previously proposed methods [2, 22, 27] with a new pruning criterion and a method that iteratively removes the least important set of neurons (typically filters) from the trained model. We define the importance as the squared change in loss induced by removing a specific filter from the network. Since computing the exact importance is extremely expensive for large networks, we approximate it with a Taylor expansion (akin to [27]), resulting in a criterion computed from parameter gradients readily available during standard training. Our method is easy to implement in existing frameworks with minimal overhead.

Additional benefits of our novel criterion include: a) no hyperparameters to set, other than providing the desired number of neurons to prune; b) globally consistent scale of our criterion across network layers without the need for per-layer sensitivity analysis; c) a simple way of computing the criterion in parallel for all neurons without greedy layer-by-layer computation; and d) the ability to apply the method to any layer in the network, including skip connections. We highlight our main contributions below:

- We propose a new method for estimating, with a little computational overhead over training, the contribution of a neuron (filter) to the final loss. To do so, we use averaged gradients and weight values that are readily available during training.
- We compare two variants of our method using the first and second-order Taylor expansions, respectively, against a greedy search ("oracle"), and show that both variants achieve state-of-the-art results, with our first-order criteria being significantly faster to compute with slightly worse accuracy. We also find that using a squared loss as a measure for contribution leads to better correlations with the oracle and better accuracy when compared to signed difference [22]. Estimated Spearman correlation with the oracle on ResNets and DenseNets trained on ImageNet show significant agreement ($>93\%$), a large improvement over previous methods [17, 22, 23, 27, 32], leading to improved pruning.
- Pruning results on a wide variety of networks trained on CIFAR-10 and ImageNet, including those with skip connections, show improvement over state-of-the-art.

2. Related work

One of the ways to reduce the computational complexity of a neural network is to train a smaller model that can mimic the output of the larger model. Such an approach, termed network distillation, was proposed by Hinton *et al.* [15]. The biggest drawback of this approach is the need to define the architecture of the smaller distilled model beforehand.

Pruning – which removes entire filters, or neurons, that make little or no contribution to the output of a trained network – is another way to make a network smaller and faster. There are two forms in which structural pruning is commonly applied: a) with a predefined per-layer pruning ratio, or b) simultaneously over all layers. The second form allows pruning to automatically find a better architecture, as demonstrated in [1]. An exact solution for pruning will be to minimize the ℓ_0 norm of all neurons and remove those that are zeroed-out. However, ℓ_0 minimization is impractical as it is non-convex, NP-hard, and requires combinatorial search. Therefore, prior work has tried to relax the optimization using Bayesian methods [25, 29] or regularization terms.

One of the first works that used regularization, by Hanson and Pratt [9], used weight decay along with other energy minimization functions to reduce the complexity of the neural network. At the same time, Chauvin [2] discovered that augmenting the loss with a positive monotonic function of the energy term can lead to learning a sparse solution.

Motivated by the success of sparse coding, several methods relax ℓ_0 minimization with ℓ_1 or ℓ_2 regularization, followed by soft thresholding of parameters with a predefined threshold. These methods belong to the family of Iterative Shrinkage and Thresholding Algorithms (ISTA) [4]. Han *et al.* [8] applied a similar approach for removing individual weights of a neural network to obtain sparse non-regular convolutional kernels. Li *et al.* [23] extended this approach to remove filters with small ℓ_1 norms.

Due to the popularity of batch-normalization [18] layers in recent networks [11, 16], several approaches have been proposed for filter pruning based on batch-norm parameters [24, 32]. These works regularize the scaling term (γ) of batch-norm layers and apply soft thresholding when value fell below a predefined threshold. Further, FLOPS-based penalties can also be included to directly reduce computational costs [6]. A more general scheme that uses an ISTA-like method on scaling factors was proposed by [17] and can be applied to any layer.

All of the above methods explicitly rely on the belief that the magnitude of the weight or neuron is strongly correlated with its importance. This belief was investigated as early as 1988 by Mozer [28] who proposed adding a gating function after each layer to be pruned. With gate values initialized to 1, the expectation of the negative gradient is used as an approximation for importance. Mozer noted that weights magnitude merely reflect the statistics of importance. LeCun *et al.* [22] also questioned whether magnitude is a reasonable measure of neuron importance. The authors suggested using a product of the Hessian's diagonal and the squared weight as a measure of individual parameter importance, and demonstrated improvement over magnitude-only pruning. This approach assumes that after convergence, the Hessian is a positive definite matrix, meaning that removing any neuron will only increase the loss. However, due to stochasticity in training with minibatches under a limited observation set and in the presence of saddle points, there do exist neurons whose removal will decrease the loss. Our method does not assume that the contribution of all neurons is strictly positive. Therefore, we approximate the squared difference of the loss when a neuron is removed and can do so with a first-order or second-order approximation, if the Hessian is available.

A few works have estimated neuron importance empirically. Luo *et al.* [26] propose to use a greedy per-layer procedure to find the subset of neurons that minimize a reconstruction loss, at a significant computational cost. Yu *et al.* [33] estimate the importance of input features to a linear classifier and propagate their importance assuming Lipschitz continuity, requiring additional computational costs and nontrivial implementation of the feature score computation. Our proposed method is able to outperform these methods while requiring little additional computation and engineering.

Pruning methods such as [13, 14, 23, 26, 33] require sensitivity analysis in order to estimate the pruning ratio that should be applied to particular layers. Molchanov *et al.* [27] assumed all layers have the same importance in feed-forward networks and proposed a normalization heuristic for global scaling. However, this method fails in networks with skip connections. Further, it computes the criterion using network activations, which increases memory requirements. Conversely, pruning methods operating on batch-normalization [6, 17, 24, 32] do not require sensitivity analysis and can be applied globally. Our criterion has globally-comparable scaling by design and does not require sensitivity analysis. It can be efficiently applied to any layer in the network, including skip connections, and not only to batch-norm layers.

A few prior works have utilized pruning as a network training regularizer. Han *et al.* [7] re-initialize weights after pruning and finetune them to achieve even better accuracy than the initial model. He *et al.* [12] extend this idea by training filters even after they were zeroed-out. While our work focuses only on removing filters from networks, it might be possible to extend it as a regularizer.

3. Method

Given neural network parameters $\mathbf{W} = \{w_0, w_1, ..., w_M\}$ and a dataset $\mathcal{D} = \{(x_0, y_0), (x_1, y_1), ..., (x_K, y_K)\}$ composed of input (x_i) and output (y_i) pairs, the task of training is to minimize error E by solving:

$$\min_{\mathbf{W}} E(\mathcal{D}, \mathbf{W}) = \min_{\mathbf{W}} E(y|x, \mathbf{W}). \tag{1}$$

In the case of pruning we can include a sparsification term in the cost function to minimize the size of the model:

$$\min_{\mathbf{W}} E(\mathcal{D}, \mathbf{W}) + \lambda ||\mathbf{W}||_0, \tag{2}$$

where λ is a scaling coefficient and $||\cdot||_0$ is the ℓ_0 norm which represents the number of non-zero elements. Unfortunately there is no efficient way to minimize the ℓ_0 norm as it is non-convex, NP-hard, and requires combinatorial search.

An alternative approach starts with the full set of parameters $\mathbf{W}$ upon convergence of the original optimization (1) and gradually reduces this set by a few parameters at a time. In this incremental setting, the decision of which parameters to remove can be made by considering the importance of each parameter individually, assuming independence of parameters. We refer to this simplified approximation to full combinatorial search as *greedy first-order search.*

The importance of a parameter can be quantified by the error induced by removing it. Under an *i.i.d.* assumption, this induced error can be measured as a squared difference of prediction errors with and without the parameter (w_m):

$$\mathcal{I}_m = \Big(E(\mathcal{D}, \mathbf{W}) - E(\mathcal{D}, \mathbf{W}|w_m = 0)\Big)^2. \tag{3}$$

Computing $\mathcal{I}_m$ for each parameter, as in (3), is computationally expensive since it requires evaluating M versions of the network, one for each removed parameter.

We can avoid evaluating M different networks by approximating $\mathcal{I}_m$ in the vicinity of $\mathbf{W}$ by its second-order Taylor expansion:

$$\mathcal{I}_m^{(2)}(\mathbf{W}) = \Big(g_m w_m - \frac{1}{2} w_m \mathbf{H}_m \mathbf{W}\Big)^2, \tag{4}$$

where $g_m = \frac{\partial E}{\partial w_m}$ are elements of the gradient $\mathbf{g}$, $H_{i,j} = \frac{\partial^2 E}{\partial w_i \partial w_j}$ are elements of the Hessian $\mathbf{H}$, and $\mathbf{H}_m$ is its *m*-th row. An even more compact approximation is computed using the first-order expansion, which simplifies to:

$$\mathcal{I}_m^{(1)}(\mathbf{W}) = \Big(g_m w_m\Big)^2. \tag{5}$$

The importance in Eq. (5) is easily computed since the gradient $\mathbf{g}$ is already available from backpropagation. For the rest of this section we will primarily use the first-order approximation, however most statements also hold for the second-order approximation. Future reference we denote the set of first-order importance approximations:

$$\mathbf{I}^{(1)}(\mathbf{W}) = \{\mathcal{I}_1^{(1)}(\mathbf{W}), \mathcal{I}_2^{(1)}(\mathbf{W}), ..., \mathcal{I}_M^{(1)}(\mathbf{W})\}. \tag{6}$$

To approximate the joint importance of a structural set of parameters $\mathbf{W}_{\mathcal{S}}$, e.g. a convolutional filter, we have two alternatives. We can define it as *a group contribution*:

$$\mathcal{I}_{\mathcal{S}}^{(1)}(\mathbf{W}) \triangleq \Big(\sum_{s \in S} g_s w_s\Big)^2, \tag{7}$$

or, alternatively, *sum the importance of the individual parameters* in the set,

$$\widehat{\mathcal{I}}_{\mathcal{S}}^{(1)}(\mathbf{W}) \triangleq \sum_{s \in \mathcal{S}} \mathcal{I}_s^{(1)}(\mathbf{W}) = \sum_{s \in \mathcal{S}} (g_s w_s)^2. \tag{8}$$

For insight into these two options, and to simplify calculations, we add "gates" to the network, $\mathbf{z} = \mathbf{1}^M$, with weights equal to 1 and dimensionality equal to the number of neurons (feature maps) M. Gating layers make importance score computation easier, as they: a) are not involved in optimization; b) have a constant value, therefore allowing $\mathbf{W}$ to be omitted from Eq. (4-8); and c) implicitly combine the contributions of filter weights and bias.

If a gate z_m follows a neuron parameterized by weights $\mathbf{W}_{s\in\mathcal{S}_m}$, then the importance approximation $\mathcal{I}^{(1)}_m$ is:

$$\mathcal{I}^{(1)}_m(\mathbf{z}) = \left(\frac{\partial E}{\partial \mathbf{z}_m}\right)^2 = \left(\sum_{s\in\mathcal{S}_m} g_s w_s\right)^2 = \mathcal{I}^{(1)}_{\mathcal{S}_m}(\mathbf{W}), \quad (9)$$

where $\mathcal{S}$ represents the inner dimensions needed to compute the output of the previous layer, *e.g.* input dimension for a linear layer, or spatial and input dimensions for a convolutional layer. We see that gate importance is equivalent to *group contribution* on the parameters of the preceding layer.

Through some manipulation, we can make a connection to information theory from our proposed method. Let's denote $\mathbf{h}_m = \frac{\partial E}{\partial \mathbf{z}_m} = \mathbf{g}^T_{s\in\mathcal{S}_m}\mathbf{W}_{s\in\mathcal{S}_m}$ and observe (under the assumption that, at convergence, $\mathbb{E}(\mathbf{h}_m)^2 = 0$):

$$\mathrm{Var}(\mathbf{h}_m) = \mathbb{E}(\mathbf{h}^2_m) - \mathbb{E}(\mathbf{h}_m)^2 = \mathbf{I}^{(1)}(\mathbf{z}), \quad (10)$$

where the variance is computed across observations.

If the error function $E(\cdot)$ is chosen to be the log-likelihood function, then assuming the gradient is estimated as $\mathbf{h}_x = \frac{\partial \ln p(x;\mathbf{z})}{\partial \mathbf{z}}$, borrowing from concepts in information theory [3], we obtain

$$\mathrm{Var}_x(\mathbf{h}) = \mathop{\mathbb{E}}_x\left\{\mathbf{h}_x\mathbf{h}^T_x\right\} = J(\mathbf{h}), \quad (11)$$

where J is the expected Fisher information matrix. We conclude that the variance of the gradient is the expectation of *the outer product of gradients* and is equal to the *expected Fisher information* matrix. Therefore, the proposed metric, $\mathbf{I}^{(1)}$, can be interpreted as the variance estimate and as the diagonal of the Fisher information matrix.

3.1. Pruning algorithm

Our pruning method takes a *trained network* as input and prunes it during an iterative fine-tuning process with a small learning rate. During each epoch, the following steps are repeated:

1. For each minibatch, we compute parameter gradients and update network weights by gradient descent. We also compute the importance of each neuron (or filter) using the gradient averaged over the minibatch, as described in (7) or (8). (Or, the second-order importance estimate may be computed if the Hessian is available.)
2. After a predefined number of minibatches, we average the importance score of each neuron (or filter) over the of minibatches, and remove the N neurons with the smallest importance scores.

Fine-tuning and pruning continue until the target number of neurons is pruned, or the maximum tolerable loss can no longer be achieved.

3.2. Implementation details

Hessian computation. Computing the full Hessian in Eq. (4) is computationally demanding, thus we use a diagonal approximation. During experiments with ImageNet we cannot compute the Hessian because of memory constraints.
Importance score accumulation. During training or fine-tuning with minibatches, observed gradients are combined to compute a single importance score $\hat{\mathbf{I}} = \mathbb{E}\langle\mathbf{I}\rangle$.
Importance score aggregation. In this work, we compute the importance of structured parameters as a sum of individual contributions defined in Eq. (8), unless gates are used automatically compute the group contribution on the parameters from the preceding layer. Second-order methods are always computed on gates. We observed that the "group contribution" criterion in Eq. (7) exhibits very low correlation with the "true" importance (3) if the parameter set $\mathcal{S}$ is too large, due to expectation of gradients tending to zero at convergence.
Gate placement. Unless otherwise stated, gates are placed immediately after a batch normalization layer to capture contributions from scaling and shifting parameters simultaneously. The first-order criterion computed for a feature map m at the gate can be shown to be $\mathbf{I}^{(1)}_m(\gamma_m, \beta_m) = (\gamma_m\frac{\partial E}{\partial\gamma_m} + \beta_m\frac{\partial E}{\partial\beta_m})^2$ with γ and β being the scale and shift parameters of the batch normalization.
Averaging importance scores over pruning iterations. We average importance scores between pruning iterations using an exponential moving average filter (momentum) with coefficient 0.9.
Pruning strategy. We found that the method performs better when we define the number of neurons to be removed, prune them in batches and fine-tune the network after that. An alternative approach is to continuously prune as long as the training or validation loss is below the threshold. The latter approach leads the optimization into local minima and final results are slightly worse.
Number of minibatches between pruning iterations needs be sufficient to capture statistics of the overall data. We use 10 minibatches and a small batch size for CIFAR datasets, but a larger (256) batch size and 30 minibatches for ImageNet pruning, as noted with each experiment.
Number of neurons pruned per iteration needs to be chosen based on how correlated the neurons are to each other. We observed that a filter's contribution changes during pruning and we usually prune around 2% of initial filters per iteration.

4. Experiments

We evaluate our method on a variety of neural network architectures on the CIFAR-10 [19] and ImageNet [31] datasets. We also experiment with variations of our method to understand the best variant. Whenever we refer to *Weight*,

Weight magnitude or *BN scale*, we use ℓ_2 norm.

4.1. Results on CIFAR-10

With the CIFAR-10 dataset, we evaluate "*oracle*" methods and second-order methods by pruning smaller networks, including LeNet3 and variants of ResNets [10] and pre-activation ResNets [11].

4.1.1 LeNet3

We start with a simple network, LeNet3, trained on the CIFAR-10 dataset to achieve 73% test accuracy. The architecture of LeNet consists of 2 convolutional and 3 linear layers arranged in a `C-R-P-C-R-P-L-R-L-R-L` (C: Conv, R: ReLU, P: Pooling, L: Linear) order with 16, 32, 120, 84, and 10 neurons respectively. We prune the first 2 convolutional and first 2 linear layers without changing the output linear layer or finetuning after pruning.

Single layer pruning. In this setup, we only prune the first convolutional layer. This setting allows us to use the *Combinatorial oracle*, the true ℓ_0 minimizer: we compute the loss for all possible combinations of k neurons that can be pruned and pick the best one. Note that this requires an exponential number of feedforward passes to evaluate – $\binom{n}{k}$ per k, where n is the number of filters and k is number of filters to prune, and so is not practical for multiple layers or larger networks. We compare against a greedy search approximation, the *Greedy oracle*, that exhaustively finds the single best neuron to remove at each pruning step, repeated k times. Results shown in Fig. 2 show the loss vs. the number of neurons pruned. We observe that the *Combinatorial oracle* is not significantly better than the *Greedy oracle* when pruning a small number of neurons. Considering that the former has exponential computational complexity, in subsequent experiments we use the *Greedy oracle* (referred to simply as *Oracle*) as a representation of the best possible outcome.

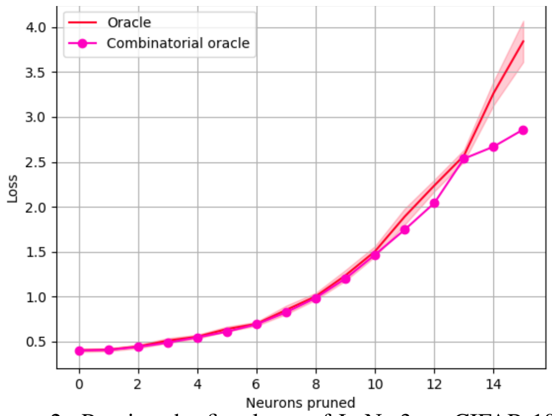

Figure 2: Pruning the first layer of LeNet3 on CIFAR-10 with Combinatorial oracle and Greedy oracle. Networks remain fixed and are not fine-tuned. Results for Greedy oracle are averaged over 30 seeds with mean and standard deviation shown. Best observed results for Combinatorial oracle for every seed are averaged.

All layers pruning. Fig. 3 shows pruning results when all layers are pruned using various criteria. We refer to our methods based on the Taylor expansion as *Taylor FO*/*Taylor SO*, indicating the order of the approximation used, first- and second-order, respectively. We consider both a direct application to convolutional filter weights ("on weight") and the use of gates following each convolutional layer ("on gate").] We treat linear layers as 1×1 convolutions. In all cases, pruning removes the entire filter and its corresponding bias. At each pruning iteration, we remove the neuron with the least importance as measured by the criterion used, and measure the loss on the training set.

Results in Fig. 3 show that *Oracle* pruning performs best, followed closely by the second- and first-order Taylor expansion criteria, respectively. Both first and second-order Taylor methods prune nearly the same number of neurons as the *Oracle* before exceeding the loss threshold. Weight-based pruning, which removes neurons with the least ℓ_2 norm, performs as poorly as randomly removing neurons. OBD [22] performs similarly to the *Oracle* and Taylor methods.

The experiments on LeNet confirm the following: (1) The greedy *oracle* closely follows the pruning performance of the *Combinatorial oracle* for small changes to the network, while being exponentially faster to compute. (2) Our first-order method (Taylor FO) is comparable to the second-order method (Taylor SO) in this setting.

4.1.2 ResNet-18

Now we compare pruning criteria on the more complex architecture ResNet-18, from the pre-activation family [11]. Each residual block has an architecture of `BN1-ReLU-conv1-BN2-ReLU-conv2`, together with a skip connection from the input to the output, repeated for a total of 8 blocks. Trained on CIFAR-10, ResNet-18 achieves a test accuracy of 94.79%. For pruning, we consider entire

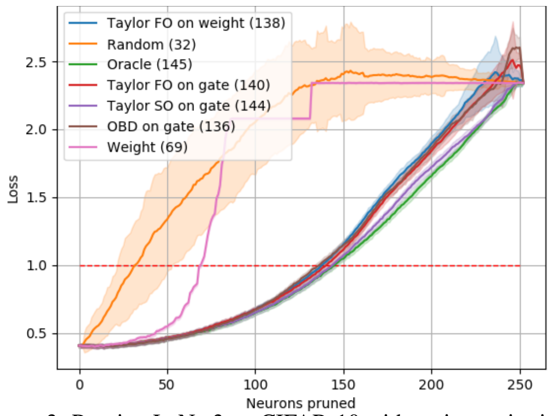

Figure 3: Pruning LeNet3 on CIFAR-10 with various criteria. Network remains fixed and is not fine-tuned. Results are averaged over 50 seeds with mean and standard deviation. The number of pruned neurons when the loss reaches 1.0 is shown in parentheses.

Method	Residual block conv1	Residual block conv2	All layers
Taylor FO on conv weight	0.726	0.766	0.892
Weight magnitude	0.660	0.677	0.861
Gate after BN2			
Taylor FO	0.955	0.811	0.924
Taylor SO	**0.968**	**0.849**	**0.957**
OBD	0.855	0.707	0.901
Taylor FO - FG	0.775	0.746	0.924
Gate before BN2			
Taylor FO	0.275	0.811	0.295
Taylor SO	0.376	0.849	0.286
OBD	0.350	0.707	0.299
Taylor FO - FG	0.642	0.746	0.900

Table 1: Spearman correlation of different criteria with the Oracle on CIFAR-10 with ResNet-18. (FG denotes full gradient, as described in the text).

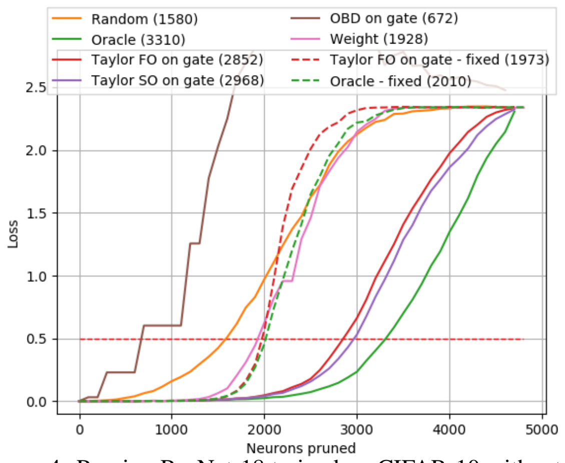

Figure 4: Pruning ResNet-18 trained on CIFAR-10 without fine-tuning. The number of neurons pruned when the loss reaches 0.5 is shown in parentheses.

feature maps in the conv layers as they command the largest share of computational resources.

In these experiments, we examine the following ways of estimating our criterion: (1) Applying it directly on convolutional filter weights, (2) Using gates placed before BN2 and after conv2, and (3) Using gates placed after BN2 and after conv2. We remove 100 neurons every 20 minibatches, and report final results averaged over 10 seeds. We also compare using gradients averaged over a mini-batch and gradients obtained per data sample, the latter denoted by "full grad", or "FG". We should note that using the full gradient changes the gate formulation from computing the group contribution (Eq. 7) to the sum of individual contributions (Eq. 8).

Table 1 presents the Spearman correlation between various pruning criteria and the greedy oracle. Results in the *Residual block* column are averaged over all 8 blocks. The *All layers* column includes additional layers: the first convolutional layer (not part of residual blocks), all convolutions in residual blocks, and all strided convolutions. We observe that placing the gate before BN2 significantly reduces correlation – correlation for conv1 drops from 0.95 to 0.28 for Taylor FO, suggesting that the subsequent batch-normalization layer significantly affects criteria computed from the gate. We observe that the effect is less significant when the full gradient is used, however it shows smaller correlation overall with the oracle. OBD has lower correlation than our Taylor based methods. The highest correlation is observed for Taylor SO, with Taylor FO following right after. As placing gates after BN2 dramatically improves the results, this indicates that the batch-normalization layers play a key role in determining the contribution of the corresponding filter.

Results of pruning ResNet-18 without fine-tuning are shown in Fig. 4. We observe that the oracle achieves the best accuracy for a given number of pruned neurons. All methods, except "-fixed" and Random, recompute the criteria after each iterative step and can adjust to the pruned network. Oracle-fixed and Taylor FO-fixed are computed across the same number of batches as non-fixed criteria. We notice that fixed criteria clearly perform significantly worse than oracle, emphasizing importance of reestimating the criteria after each pruning iteration, allowing the values to adjust to changes in the network architecture.

An interesting observation is that the OBD method performs poorly in spite of having a good correlation with the oracle in Table 1. The reason for this discrepancy is that when we evaluate correlation with the oracle, we square estimates of OBD to make them comparable to the way the oracle was estimated. However, during pruning, we use signed values of OBD, as was prescribed in [22]. As mentioned earlier, for deep networks, the diagonal of the Hessian is not positive for all elements and removing those with negative impact results in increased network instability. Therefore, without fine-tuning, OBD is not well suited for pruning. Another important observation is that if the Hessian is available, using the Taylor SO expansion can get both better pruning and correlation. Surprisingly, we observe no improvement in using the full gradient, probably because of the switch in contributions from group to individual.

At this stage, after experiments with the small LeNet3 network the larger ResNet-18 on the CIFAR-10 dataset, we make the following observations: (1) Our proposed criteria based on the Taylor expansion of the pruning loss have a very high correlation with the neuron ranking produced by the oracle. (2) The first- and second-order Taylor criteria are comparable. As the Taylor FO can be computed much faster with a lower memory footprint, further experiments with larger networks on ImageNet are performed using this criterion only.

4.2. Results on ImageNet

Here, we apply our method on the challenging task of pruning networks trained on ImageNet [31], specifically the ILSVRC2012 version. For all experiments in this section, we use PyTorch [30] and default pretrained models as a starting point for network pruning. We use standard preprocessing and augmentation: re-sizing images to have a smallest dimension of 256, randomly cropping a 224×224 patch, randomly applying horizontal flips, and normalizing images by subtracting a per-dataset mean and dividing by a per-dataset standard deviation. During testing, we use the central crop of size 224×224.

4.2.1 Neuron importance correlation study

Method	Ours	Averaged per layer			All layers		
	Taylor FO	Pearson	Spearman	Kendall	Pearson	Spearman	Kendall
				ResNet-101			
Gate after BN	✓	**0.877**	**0.870**	**0.710**	**0.925**	**0.965**	**0.843**
Gate after BN - FG	✓	0.772	0.817	0.644	0.778	0.944	0.803
Conv weight	✓	0.719	0.740	0.570	0.780	0.874	0.698
BN scale	✓	0.703	0.664	0.501	0.792	0.866	0.681
BN scale		0.371	0.405	0.296	0.632	0.807	0.621
Weight magnitude		0.566	0.651	0.493	0.376	0.587	0.432
Taylor-output [27]		0.520	0.586	0.429	0.381	0.287	0.198
				Including skip connections			
Gate after BN	✓	0.874	0.867	0.707	0.806	0.946	0.809
Gate after BN - FG	✓	0.768	0.814	0.640	0.725	0.873	0.727
				VGG11-BN			
Gate after BN	✓	**0.964**	**0.974**	**0.894**	**0.798**	**0.999**	**0.972**
Conv/Linear weight	✓	0.659	0.627	0.507	0.843	0.983	0.893
Gate after BN - FG	✓	0.825	0.800	0.666	0.812	0.982	0.887
BN scale	✓	0.751	0.718	0.586	0.634	0.968	0.846
BN scale		0.474	0.438	0.351	0.031	0.257	0.213
Weight magnitude		0.604	0.603	0.474	0.537	0.812	0.654
Taylor-output [27]		0.590	0.581	0.468	0.534	0.968	0.876
				DenseNet-201			
Gate after BN	✓	**0.825**	**0.849**	**0.740**	**0.967**	**0.932**	**0.811**
Gate after BN - FG	✓	0.891	0.898	0.742	0.764	0.944	0.798
Conv weight	✓	0.825	0.836	0.659	0.701	0.817	0.627
BN scale	✓	0.645	0.645	0.479	0.677	0.434	0.307
BN scale		0.472	0.471	0.342	0.506	0.597	0.436
Weight magnitude		0.725	0.737	0.558	0.300	0.300	0.208
Taylor-output [27]		0.673	0.699	0.530	0.455	0.472	0.333

Table 2: Correlation study of different criteria and oracle on the ImageNet dataset. Spearman and Kendall measure rank correlations. BN stands for batch-normalization, FG for full gradient.

We compare against pruning methods that use various heuristics, such as *weight magnitude* [21, 23], magnitude of the batch-norm scale, *BN scale* [6, 24, 32], and *output*-based heuristics (Taylor expansion applied to layer outputs) [27].

We estimate the correlation between the "real importance" of a filter and these criteria. Estimating real importance, or the change in loss value upon removing a neuron, requires running inference multiple times while setting each individual filter to 0 in turn. (Note that the oracle ranks neurons based on this value). For ResNet-101, we pruned filters in the first 2 convolutional layers of every residual block. Separately, we add gates to skip connections at the input and output of each block. For the VGG11-BN architecture, we replace drop-out layers with batch-norms (0.5 scale and 0 shift) and fine-tune for 12 epochs until test accuracy reaches 70.8% to be comparable with [1]. For DenseNet201, we considered features after the batch-norm layer that follows the first 1×1 convolution in every dense layer.

The statistical correlation between heuristics and measured importance are summarized in Table 2. Correlations were measured on a subset of ImageNet consisting of a few thousand images. We evaluated various implementations of our method, but always use the first-order Taylor expansion, denoted Taylor FO. As previously discussed, the most promising variation uses a gate after each batch-norm layer. The *All layers* correlation columns show how well the criteria scale across layers. Our method exhibits >93% Spearman correlation for all three networks. *Weight magnitude* and *BN scale* have quite low correlation, suggesting that magnitude is not a good representation of importance. *Output*-based expansion proposed in [27] has high correlation on the VGG11-BN network but fails on ResNet and DenseNet architectures. Surprisingly, we observe >92% Pearson correlation for ResNet and DenseNet, showing we can almost exactly predict the change in loss for every neuron.

We are also able to study the effect of skip connections by adding a gate after the output of each residual block. We add skip connections to the full set of filters and evaluate their correlation, denoted "Including skip connections" in Table 2. We observe high correlation of the criterion with skip connections as well. Given this result, we adopt this methodology for pruning ResNets and remove channels from skip connections and bottleneck layers simultaneously. We refer to this variant of our method as *Taylor-FO-BN*.

4.2.2 Pruning and fine-tuning

We use the following settings: 4 GPUs and a batch size of 256 examples; we optimized using SGD with initial learning rate 0.01 (or 0.001, see Sec. 6.2) decayed a factor 10 every 10 epochs, momentum set to 0.9; pruning and fine-tuning run for 25 epochs total; we report the best validation accuracy observed. Every 30 mini-batches we remove 100 neurons until we reach the predefined number of neurons to be pruned, after which we reset the momentum buffer and continue fine-tuning. By setting the percentage of neurons to remain after pruning to be *X*, we get different versions of the final model and refer to them as *Taylor-FO-BN-X%*.

Comparison of pruning networks on ImageNet by the proposed method and other methods is presented in the Table 3, where we report total number of FLOPs, number of parameters, and the top-1 error rate. Comparison is grouped by network architecture type and the number of parameters. For ResNet-101 we observe smaller error rates and fewer GFLOPs (by at least 1.22 GFLOPs) when compare to BN-ISTA method [32]. Pruning only skip connections shows larger errors however makes the final network faster (see Sec 6.2). By pruning 40% of FLOPs and 30% of parameters from original ResNet-101 we only lose 0.02% in accuracy.

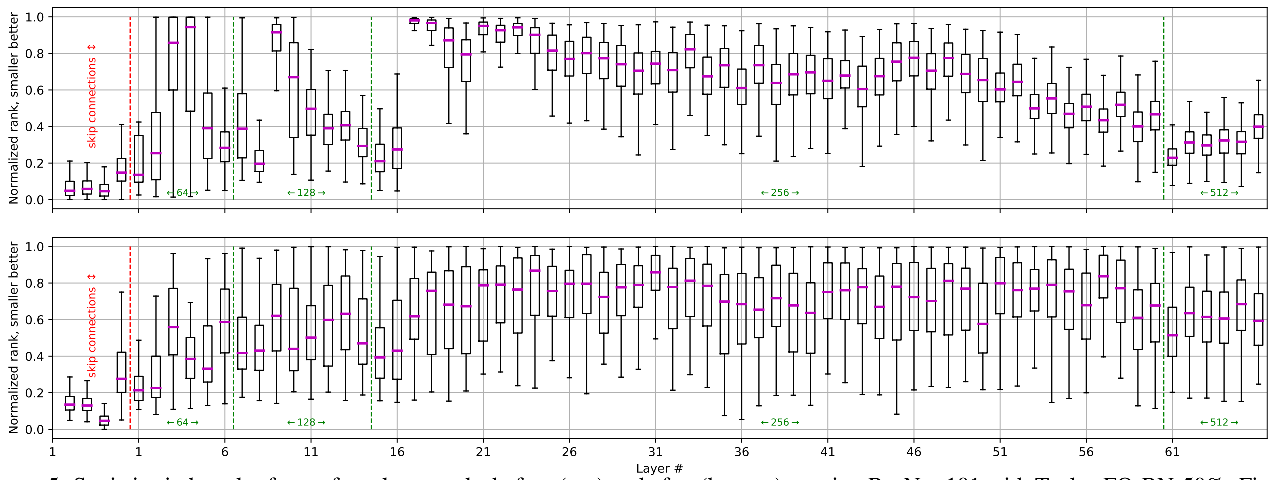

Figure 5: Statistics in boxplot form of per-layer ranks before (top) and after (bottom) pruning ResNet-101 with Taylor-FO-BN-50%. First 4 layers correspond to skip connections, the rest are residual blocks represented by the first 2 convolutional layers per block. We can notice that after pruning most of neurons become more equal than before pruning.

Pruning Method	GFLOPs	Params(10^7)	↓ Error, %
ResNet-101			
Taylor-FO-BN-40% **(Ours)**	1.76	1.36	25.84
Taylor-FO-BN-50% **(Ours)**	**2.47**	1.78	**24.62**
BN-ISTA v2 [32]	3.69	**1.73**	25.44
Taylor-FO-BN-55% **(Ours)**	**2.85**	**2.07**	**24.05**
BN-ISTA v1 [32]	4.47	2.36	24.73
No pruning	7.80	4.47	**22.63**
Taylor-FO-BN-75% **(Ours)**	**4.70**	**3.12**	22.65
pruning only skip connections			
Taylor-FO-BN-52% **(Ours)**	6.57	3.60	22.94
Taylor-FO-BN-22% **(Ours)**	5.19	2.86	24.77
ResNet-50			
Taylor-FO-BN-56% **(Ours)**	1.34	**0.79**	**28.31**
Taylor-FO-BN-56% (No skip)	1.28	0.85	30.74
ThiNet-30 [25]	≈**1.17**	0.87	31.58
Taylor-FO-BN-72% **(Ours)**	**2.25**	**1.42**	**25.50**
NISP-50-B [31]	≈2.29	1.43	27.93
ThiNet-70 [25]	≈2.58	1.69	27.96
Taylor-FO-BN-81% **(Ours)**	**2.66**	**1.79**	**24.52**
SSS [17], ResNet-32	2.82	1.86	25.82
NISP-50-A [31]	≈2.97	1.86	27.25
Taylor-FO-BN-91% **(Ours)**	**3.27**	**2.26**	**23.57**
No pruning	4.09	2.56	23.82
SSS [17], ResNet-41	3.47	2.53	24.56
ResNet-34			
No pruning	3.64	2.18	26.69
Taylor-FO-BN-82% **(Ours)**	2.83	**1.72**	**27.17**
Li *et al.* [23]	**2.76**	1.93	27.80
VGG11-BN			
No pruning	7.61	13.29	**29.16**
Taylor-FO-BN-50% **(Ours)**	**6.93**	**3.18**	29.35
From scratch [1]	≈**6.93**	≈**3.18**	30.00
Slimming [24], from [1]	≈**6.93**	≈**3.18**	31.38
DenseNet-201			
No pruning	4.29	2.20	**23.20**
Taylor-FO-BN-60% **(Ours)**	3.02	1.25	23.49
Taylor-FO-BN-36% **(Ours)**	**2.21**	0.90	24.72
No pruning	2.74	**0.76**	25.57

Table 3: Pruning results on ImageNet (1-crop validation errors).

Pruning results on ResNet-50 and ResNet-34 demonstrate significant improvements over other methods. Additionally we study our method without pruning skip connections, marked as "No skip" and observe accuracy loss. Comparison per layer ranking of different layers in ResNet-101 before and after pruning is shown in Fig. 5.

Pruning neurons with a single step. As an alternative to iterative pruning, we performed pruning of 10000 neurons with a single step after 3000 mini-batches, followed by fine-tuning. This gave a top-1 error of 25.3% , which is 0.68% higher than *Taylor-FO-BN-50%*, again emphasizing the benefit of re-evaluating the criterion between pruning iterations.

Pruning other networks. We also prune the VGG11-BN and DenseNet networks. The former is a simple feed-forward architecture, without skip connections. We prune 50% of neurons across all layers, as per prior work [1, 24]. Our approach shows only 0.19% loss in accuracy after removing 76% of parameters and improves on the previously reported results by 0.65% [1] and more than 2% [24]. DeseNets reuse feature maps multiple times, potentially making them less amenable to pruning. We prune DenseNet-201 and observe that with the same number of FLOPs (*Taylor-FO-BN-52%*) as DenseNet-121, we have 1.79% lower error.

5. Conclusions

In this work, we have proposed a new method for estimating the contribution of a neuron using the Taylor expansion applied on a squared change in loss induced by removing a chosen neuron. We demonstrated that even the first-order approximation shows significant agreement with true importance, and outperforms prior work on a range of deep networks. After extensive analysis, we showed that applying the first-order criterion after batch-norms yields the best results, under practical computational and memory constraints.

References

[1] Anonymous submission. Rethinking the value of network pruning. *ICLR*, 2019.

[2] Y. Chauvin. A back-propagation algorithm with optimal use of hidden units. In *NIPS*, 1989.

[3] T. M. Cover and J. A. Thomas. *Elements of information theory*. John Wiley & Sons, 2012.

[4] I. Daubechies, M. Defrise, and C. De Mol. An iterative thresholding algorithm for linear inverse problems with a sparsity constraint. *Communications on Pure and Applied Mathematics*, 2004.

[5] J. Frankle and M. Carbin. The lottery ticket hypothesis: Training pruned neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

[6] 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 *CVPR*, 2018.

[7] S. Han, J. Pool, S. Narang, H. Mao, S. Tang, E. Elsen, B. Catanzaro, J. Tran, and W. J. Dally. Dsd: regularizing deep neural networks with dense-sparse-dense training flow. *arXiv preprint arXiv:1607.04381*, 2016.

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

[9] S. J. Hanson and L. Y. Pratt. Comparing biases for minimal network construction with back-propagation. In *NIPS*, 1989.

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

[11] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In *ECCV*, 2016.

[12] Y. He, X. Dong, G. Kang, Y. Fu, and Y. Yang. Progressive deep neural networks acceleration via soft filter pruning. *arXiv preprint arXiv:1808.07471*, 2018.

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

[14] Y. He, X. Zhang, and J. Sun. Channel pruning for accelerating very deep neural networks. In *ICCV*, 2017.

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

[16] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *CVPR*, 2017.

[17] Z. Huang and N. Wang. Data-driven sparse structure selection for deep neural networks. *arXiv preprint arXiv:1707.01213*, 2017.

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

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

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

[21] V. Lebedev and V. Lempitsky. Fast convnets using group-wise brain damage. In *CVPR*, pages 2554–2564, 2016.

[22] Y. LeCun, J. S. Denker, S. Solla, R. E. Howard, and L. D. Jackel. Optimal brain damage. In *NIPS*, 1990.

[23] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. Pruning filters for efficient convnets. *ICLR*, 2017.

[24] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang. Learning efficient convolutional networks through network slimming. In *ICCV*, 2017.

[25] C. Louizos, M. Welling, and D. P. Kingma. Learning sparse neural networks through l_0 regularization. *arXiv preprint arXiv:1712.01312*, 2017.

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

[27] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz. Pruning convolutional neural networks for resource efficient transfer learning. *ICLR*, 2017.

[28] M. C. Mozer and P. Smolensky. Skeletonization: A technique for trimming the fat from a network via relevance assessment. In *NIPS*, 1989.

[29] K. Neklyudov, D. Molchanov, A. Ashukha, and D. P. Vetrov. Structured bayesian pruning via log-normal multiplicative noise. In *Advances in Neural Information Processing Systems*, pages 6775–6784, 2017.

[30] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017.

[31] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *IJCV*, 2015.

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

[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. *CVPR*, 2017.

[34] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals. Understanding deep learning requires rethinking generalization. *ICLR*, 2016.