EigenDamage: Structured Pruning in the Kronecker-Factored Eigenbasis

Chaoqi Wang [1 2] **Roger Grosse** [1 2] **Sanja Fidler** [1 2 3] **Guodong Zhang** [1 2]

Abstract

Reducing the test time resource requirements of a neural network while preserving test accuracy is crucial for running inference on resource-constrained devices. To achieve this goal, we introduce a novel network reparameterization based on the Kronecker-factored eigenbasis (KFE), and then apply Hessian-based structured pruning methods in this basis. As opposed to existing Hessian-based pruning algorithms which do pruning in parameter coordinates, our method works in the KFE where different weights are approximately independent, enabling accurate pruning and fast computation. We demonstrate empirically the effectiveness of the proposed method through extensive experiments. In particular, we highlight that the improvements are especially significant for more challenging datasets and networks. With negligible loss of accuracy, an iterative-pruning version gives a 10× reduction in model size and a 8× reduction in FLOPs on wide ResNet32. Our code is available at here.

1. Introduction

Deep neural networks exhibit good generalization behavior in the over-parameterized regime (Zhang et al., 2016; Neyshabur et al., 2018), where the number of network parameters exceeds the number of training samples. However, over-parameterization leads to high computational cost and memory overhead at test time, making it hard to deploy deep neural networks on a resource-limited device.

Network pruning (LeCun et al., 1990; Hassibi et al., 1993; Han et al., 2015b; Dong et al., 2017; Zeng & Urtasun, 2019) has been identified as an effective technique to improve the efficiency of deep networks for applications with limited test-time computation and memory budgets. Without much loss in accuracy, classification networks can be compressed by a factor of 10 or even more (Han et al., 2015b; Zeng & Urtasun, 2019) on ImageNet (Deng et al., 2009). A typical pruning procedure consists of three stages: 1) train a large, over-parameterized model, 2) prune the trained model according to a certain criterion, and 3) fine-tune the pruned model to regain the lost performance.

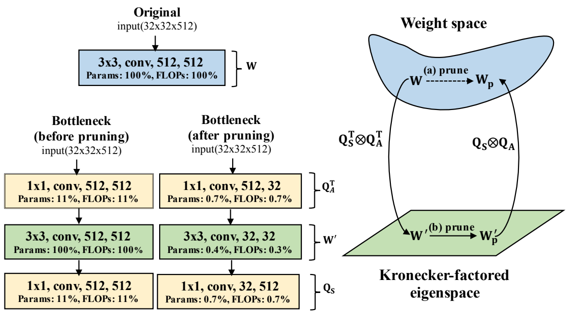


Figure 1. On the left-hand side, the proposed bottleneck structure before and after pruning. The number after 'Params' and 'FLOPs' indicates the remaining portion compared to the original one. On the right-hand side, we highlight the differences of the pruning procedure between traditional methods (a) and our method (b).

Most existing work on network pruning focuses on the second stage. A common idea is to select parameters for pruning based on weight magnitudes (Hanson & Pratt, 1989; Han et al., 2015b). However, weights with small magnitude are not necessarily unimportant (LeCun et al., 1990). As a consequence, magnitude-based pruning might delete important parameters, or preserve unimportant ones. By contrast, Optimal Brain Damage (OBD) (LeCun et al., 1990) and Optimal Brain Surgeon (OBS) (Hassibi et al., 1993) prune weights based on the Hessian of the loss function; the advantage is that both criteria reflect the sensitivity of the cost to the weight. Though OBD and OBS have proven to be effective for shallow neural networks, it remains challenging to extend them for deep networks because of the high computational cost of computing second derivatives. To solve this issue, several approximations to the Hessian have been proposed recently which assume layerwise independence (Dong et al., 2017) or Kronecker structure (Zeng & Urtasun, 2019).

All of the aforementioned methods prune individual weights, leading to non-structured architectures which do not enjoy

[1]Department of Computer Science, University of Toronto, Toronto, Canada [2]Vector Institute, Toronto, Canada [3]NVIDIA. Correspondence to: Chaoqi Wang <cqwang@cs.toronto.edu>, Guodong Zhang <gdzhang@cs.toronto.edu>.

Proceedings of the 36th International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019. Copyright 2019 by the author(s).

computational speedups unless one employs dedicated hardware (Han et al., 2016) and software, which is difficult and expensive in real-world applications (Liu et al., 2018). In contrast, structured pruning methods such as channel pruning (Liu et al., 2017; Li et al., 2016b) aim to preserve the convolutional structure by pruning at the level of channels or even layers, thus automatically enjoy computational gains even with standard software frameowrks and hardware.

Our Contributions. In this work, we focus on structured pruning. We first extend OBD and OBS to channel pruning, showing that they can match the performance of a state-of-the-art channel pruning algorithm (Liu et al., 2017). We then interpret them from the Bayesian perspective, showing that OBD and OBS each approximate the full-covariance Gaussian posterior with factorized Gaussians, but minimizing different variational objectives. However, different weights can be highly coupled in Bayesian neural network posteriors (e.g., see Figure 2), suggesting that full-factorization assumptions may hurt the pruning performance.

Based on this insight, we prune in a different coordinate system in which the posterior is closer to factorial. Specifically, we consider the Kronecker-factored eigenbasis (KFE) (George et al., 2018; Bae et al., 2018), in which the Hessian for a given layer is closer to diagonal. We propose a novel network reparameterization inspired by Desjardins et al. (2015) which explicitly parameterizes each layer in terms of the KFE. Because the Hessian matrix is closer to diagonal in the KFE, we can apply OBD with less cost to prediction accuracy; we call this method EigenDamage.

Instead of sparse weight matrices, pruning in the KFE leads to a low-rank approximation, or bottleneck structure, in each layer (see Figure 1). While most existing structured pruning methods (He et al., 2017; Li et al., 2016b; Liu et al., 2017; Luo et al., 2017) require specialized network architectures, EigenDamage can be applied to any fully connected or convolution layers without modifications. Furthermore, in contrast to traditional low-rank approximations (Denton et al., 2014; Lebedev et al., 2014; Jaderberg et al., 2014) which minimize the Frobenius norm of the weight space error, EigenDamage is *loss aware*. As a consequence, the user need only choose a single compression ratio parameter, and EigenDamage can automatically determine an appropriate rank for each layer, and thus it is calibrated across layers. Empirically, EigenDamage outperforms strong baselines which do pruning in parameter coordinates, especially in more challenging datasets and networks.

2. Background

In this section, we first introduce some background for understanding and reinterpreting Hessian-based weight pruning algorithms, and then briefly review structured pruning to provide context for the task that we will deal with.

Laplace Approximation. In general, we can obtain the Laplace approximation (MacKay, 1992) by simply taking the second-order Taylor expansion around a local mode. For neural networks, we can find such modes with SGD. Given a neural network with local MAP parameters $\boldsymbol{\theta}^*$ after training on a dataset $\mathcal{D}$, we can obtain the Laplace approximation over the weights around $\boldsymbol{\theta}^*$ by:

$$\log p(\boldsymbol{\theta}|\mathcal{D}) \approx \log p(\boldsymbol{\theta}^*|\mathcal{D}) - \frac{1}{2}(\boldsymbol{\theta}-\boldsymbol{\theta}^*)^\top \mathbf{H}(\boldsymbol{\theta}-\boldsymbol{\theta}^*) \quad (1)$$

where $\boldsymbol{\theta} = [\mathrm{vec}(\mathbf{W}_1), ..., \mathrm{vec}(\mathbf{W}_L)]$, and $\mathbf{H}$ is the Hessian matrix of the negative log posterior evaluated at $\boldsymbol{\theta}^*$. Assuming $\mathbf{H}$ is p.s.d., the Laplace approximation is equivalent to approximating the posterior over weights as a Gaussian distribution with $\boldsymbol{\theta}^*$ and $\mathbf{H}$ as the mean and precision, respectively. In practice, we can use the Fisher information matrix $\mathbf{F}$ to approximate $\mathbf{H}$, as done in Graves (2011); Zhang et al. (2017); Ritter et al. (2018). This ensures a p.s.d. matrix and allows efficient approximation (Martens, 2014).

Forward and reverse KL divergence (Murphy, 2012). Suppose the true distribution is $p(\boldsymbol{\theta})$, and the approximate distribution is $q_\phi(\boldsymbol{\theta})$, the forward and reverse KL divergence are $\mathrm{D_{KL}}(p(\boldsymbol{\theta})||q_\phi(\boldsymbol{\theta}))$ and $\mathrm{D_{KL}}(q_\phi(\boldsymbol{\theta})||p(\boldsymbol{\theta}))$ respectively. In general, minimizing the forward KL will arise the mass-covering behavior, and minimizing the reverse KL will arise the zero-forcing/mode-seeking behavior (Minka et al., 2005). When we use a factorized Gaussian distribution $q_\phi(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta}|\mathbf{0}, \boldsymbol{\Sigma})$ to approximate multivariate Gaussian distribution $p(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta}|\mathbf{0}, \boldsymbol{\Sigma}^*)$, the solutions to minimizing the forward KL and reverse KL are

$$(a)\ \boldsymbol{\Sigma} = \mathrm{diag}(\boldsymbol{\Sigma}^*) \qquad (b)\ \boldsymbol{\Sigma} = \mathrm{diag}(\boldsymbol{\Lambda}^*)^{-1}$$

where the precision matrix $\boldsymbol{\Lambda}^* = \boldsymbol{\Sigma}^{*-1}$. For the Laplace approximation, the true posterior variance $\boldsymbol{\Sigma}^*$ is $\mathbf{H}^{-1}$.

K-FAC. Kronecker-factored approximate curvature (K-FAC) (Martens & Grosse, 2015) uses a Kronecker-factored approximation to the Fisher matrix of fully connected layers, *i.e.* no weight sharing. Considering l-th layer in a neural network whose input activations are $\mathbf{a} \in \mathbb{R}^n$, weight matrix $\mathbf{W} \in \mathbb{R}^{n\times m}$, and output $\mathbf{s} \in \mathbb{R}^m$, we have $\mathbf{s} = \mathbf{W}^\top\mathbf{a}$. Therefore, the weight gradient is $\nabla_{\mathbf{W}}\mathcal{L} = \mathbf{a}(\nabla_{\mathbf{s}}\mathcal{L})^\top$. With this formula, K-FAC decomposes this layer's Fisher matrix $\mathbf{F}$ with an independence assumption:

$$\begin{aligned}\mathbf{F} &= \mathbb{E}[\mathrm{vec}\{\nabla_{\mathbf{W}}\mathcal{L}\}\mathrm{vec}\{\nabla_{\mathbf{W}}\mathcal{L}\}^\top] \\ &= \mathbb{E}[\{\nabla_{\mathbf{s}}\mathcal{L}\}\{\nabla_{\mathbf{s}}\mathcal{L}\}^\top \otimes \mathbf{a}\mathbf{a}^\top] \qquad (2) \\ &\approx \mathbb{E}[\{\nabla_{\mathbf{s}}\mathcal{L}\}\{\nabla_{\mathbf{s}}\mathcal{L}\}^\top] \otimes \mathbb{E}[\mathbf{a}\mathbf{a}^\top] = \mathbf{S}\otimes\mathbf{A},\end{aligned}$$

where $\mathbf{A} = \mathbb{E}[\mathbf{a}\mathbf{a}^\top]$ and $\mathbf{S} = \mathbb{E}[\{\nabla_{\mathbf{s}}\mathcal{L}\}\{\nabla_{\mathbf{s}}\mathcal{L}\}^\top]$.

Grosse & Martens (2016) further extended K-FAC to convolutional layers under additional assumptions of spa-

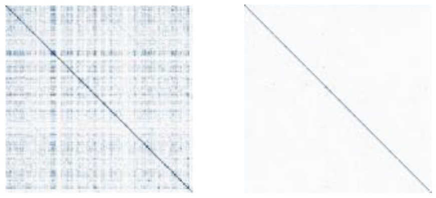


Figure 2. Fisher information matrices measured in initial parameter basis and in the KFE, computed from a small 3-layer ReLU MLP trained on MNIST. We only plot the block for the second layer. Note that we normalize the diagonal elements for visualization.

tial homogeneity (**SH**) and spatially uncorrelated derivatives (**SUD**). Suppose the input $\mathbf{a} \in \mathbb{R}^{c_{\text{in}} \times h \times w}$ and the output $\mathbf{s} \in \mathbb{R}^{c_{\text{out}} \times h \times w}$, then the gradient of the reshaped weight $\mathbf{W} \in \mathbb{R}^{c_{\text{out}} \times c_{\text{in}} k^2}$ is $\nabla_{\mathbf{W}} \mathcal{L} = \sum \mathbf{a}_i \nabla_{\mathbf{s}_i} \mathcal{L}^\top$, and the corresponding Fisher matrix is:

$$\begin{aligned}\mathbf{F} &\approx \sum \mathbb{E}\left[\{\nabla_{\mathbf{s}_i}\mathcal{L}\}\{\nabla_{\mathbf{s}_{i'}}\mathcal{L}\}^\top\right] \otimes \mathbb{E}\left[\mathbf{a}_i \mathbf{a}_{i'}^\top\right] \\ &\approx \underbrace{\left(\frac{1}{|\mathcal{I}|}\sum \mathbb{E}\left[\{\nabla_{\mathbf{s}_i}\mathcal{L}\}\{\nabla_{\mathbf{s}_i}\mathcal{L}\}^\top\right]\right)}_{\mathbf{S},\text{size}=(c_{\text{out}})^2} \otimes \underbrace{\left(\sum \mathbb{E}\left[\mathbf{a}_i \mathbf{a}_i^\top\right]\right)}_{\mathbf{A},\text{size}=(c_{\text{in}} \times k^2)^2}\end{aligned} \tag{3}$$

where $\mathcal{I} = [h] \times [w]$ is the set of spatial locations, $\mathbf{a}_i \in \mathbb{R}^{c_{\text{in}} k^2}$ is the patch extracted from $\mathbf{a}$, $\nabla_{\mathbf{s}_i} \mathcal{L} \in \mathbb{R}^{c_{\text{out}}}$ is the gradient to each spatial location in $\mathbf{s}$ and $i, i' \in \mathcal{I}$. Decomposing $\mathbf{F}$ into $\mathbf{A}$ and $\mathbf{S}$ not only avoids the quadratic storage cost of the exact Fisher, but also enables efficient computation of the Fisher vector product:

$$\mathbf{F}\text{vec}\{\mathbf{X}\} = \mathbf{S} \otimes \mathbf{A}\text{vec}\{\mathbf{X}\} = \text{vec}\{\mathbf{A}\mathbf{X}\mathbf{S}^\top\} \tag{4}$$

and fast computation of inverse and eigen-decomposition:

$$\begin{aligned}\mathbf{F}^{-1} &= (\mathbf{S} \otimes \mathbf{A})^{-1} = \mathbf{S}^{-1} \otimes \mathbf{A}^{-1} \\ \mathbf{F} &= (\mathbf{Q}_\mathbf{S} \otimes \mathbf{Q}_\mathbf{A})(\mathbf{\Lambda}_\mathbf{S} \otimes \mathbf{\Lambda}_\mathbf{A})(\mathbf{Q}_\mathbf{S} \otimes \mathbf{Q}_\mathbf{A})^\top\end{aligned} \tag{5}$$

where $\mathbf{Q}$ and $\mathbf{\Lambda}$ are eigenvectors and eigenvalues. Since $\mathbf{Q}_\mathbf{S} \otimes \mathbf{Q}_\mathbf{A}$ gives the eigenbasis of the Kronecker product, we call it the Kronecker-factored Eigenbasis (KFE).

Structured Pruning. Structured network pruning (He et al., 2017; Liu et al., 2017; Li et al., 2016b; Luo et al., 2017) is a technique to reduce the size of a network while retaining the original convolutional structure. Among structured pruning methods, channel/filter pruning is the most popular. Let c_{in} denote the number of input channels for the l-th convolutional layer and h/w be the height/width of the input feature maps. The conv layer transforms the input $\mathbf{a} \in \mathbb{R}^{c_{\text{in}} \times h \times w}$ with c_{out} filters $\mathcal{F}_i$. All the filters constitute the kernel matrix $\mathcal{F} \in \mathbb{R}^{c_{\text{in}} \times c_{\text{out}} \times k \times k}$. When a filter $\mathcal{F}_i$ is pruned, its corresponding feature map in the next layer $\mathbf{a}_i$ is removed, so channel and filter pruning are typically referred to as the same thing. However, most current channel pruning methods either require predefined target models (Li et al., 2016b; Luo et al., 2017) or specialized network architectures (Liu et al., 2018), making them hard to use.

3. Revisiting OBD and OBS

OBD and OBS share the same basic pruning pipeline: first training a network to (local) minimum in error at weight $\boldsymbol{\theta}^*$, and then pruning a weight that leads to the smallest increase in the training error. The predicted increase in the error for a change in full weight vector $\Delta\boldsymbol{\theta}$ is:

$$\Delta\mathcal{L} = \underbrace{\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}^\top}_{\approx 0} \Delta\boldsymbol{\theta} + \frac{1}{2}\Delta\boldsymbol{\theta}^\top \mathbf{H} \Delta\boldsymbol{\theta} + \mathcal{O}(||\Delta\boldsymbol{\theta}||^3) \tag{6}$$

Eqn. (6) is a simple second order Taylor expansion around the local mode, which is essentially the Laplace approximation. According to Eqn. (1), we can reinterpret the above cost function from a probabilistic perspective:

$$\begin{aligned}\Delta\mathcal{L} &= -\log p_{\text{LA}}(\boldsymbol{\theta}^* + \Delta\boldsymbol{\theta}|\mathcal{D}) + \text{const} \\ p_{\text{LA}}(\boldsymbol{\theta}^* + \Delta\boldsymbol{\theta}|\mathcal{D}) &= \mathcal{N}(\Delta\boldsymbol{\theta}|\mathbf{0},\ \mathbf{H}^{-1})\end{aligned} \tag{7}$$

where LA denotes Laplace approximation.

OBD. Due to the intractability of computing full Hessian in deep networks, the Hessian matrix $\mathbf{H}$ is approximated by a diagonal matrix in OBD. If we prune a weight $\boldsymbol{\theta}_q$, then the corresponding change in weights as well as the cost are:

$$\Delta\boldsymbol{\theta}_q = -\boldsymbol{\theta}_q^* \ \text{ and } \ \Delta\mathcal{L}_{\text{OBD}} = \frac{1}{2}\left(\boldsymbol{\theta}_q^*\right)^2 \mathbf{H}_{qq} \tag{8}$$

It regards all the weights as uncorrelated, such that removing one will not affect the others. This treatment can be problematic if the weights are correlated in the posterior.

OBS. In OBS, the importance of each weight is calculated by solving the following constrained optimization problem:

$$\min_q \{\min_{\Delta\boldsymbol{\theta}} \frac{1}{2}\Delta\boldsymbol{\theta}^\top \mathbf{H} \Delta\boldsymbol{\theta} \ \ s.t. \ \ \mathbf{e}_q^\top \Delta\boldsymbol{\theta} + \boldsymbol{\theta}_q^* = 0\} \tag{9}$$

for considering the correlations among weights, where $\mathbf{e}_q$ is the unit selecting vector whose q-th element is 1 and 0 otherwise. Solving Eqn. (9) yields the optimal weight change and the corresponding change in error:

$$\Delta\boldsymbol{\theta} = -\frac{\boldsymbol{\theta}_q^*}{[\mathbf{H}^{-1}]_{qq}} \mathbf{H}^{-1} \mathbf{e}_q \ \text{ and } \ \Delta\mathcal{L}_{\text{OBS}} = \frac{1}{2}\frac{(\boldsymbol{\theta}_q^*)^2}{[\mathbf{H}^{-1}]_{qq}} \tag{10}$$

The main difference is that OBS not only prunes a single weight but takes into account the correlation between weights and updates the rest of the weights to compensate.

3.1. New Insights and Perspectives

A common belief is that OBS is superior to OBD, though it is only feasible for shallow networks. In the following paragraphs, we will show that this may not be the case in practice even when we can compute exact Hessian inverse.

From Eqn. (8), we can see that OBD can be seen as OBS with off-diagonal entries of the Hessian ignored. If we prune only one weight each time, OBS is advantageous in the sense that it takes into account the off-diagonal entries. However, pruning weights one by one is time consuming and typically infeasible for modern neural networks. It is more common to prune many weights at a time (Zeng & Urtasun, 2019; Dong et al., 2017; Han et al., 2015b), especially in structured pruning (Liu et al., 2017; Luo et al., 2017; Li et al., 2016b).

We note that, when pruning multiple weights simultaneously, both OBD and OBS can be interpreted as using a factorized Gaussian $\mathcal{N}(\Delta\boldsymbol{\theta}|\mathbf{0},\ \boldsymbol{\Sigma})$ to approximate the true posterior over weights, but with different objectives. Specifically, OBD can be obtained by minimizing the reverse KL divergence ($\boldsymbol{\Sigma} = \mathrm{diag}(\mathbf{H})^{-1}$), whereas OBS is using the forward KL divergence ($\boldsymbol{\Sigma} = \mathrm{diag}(\mathbf{H}^{-1})$). Reverse KL underestimates the variance of the true distribution and overestimates the importance of each weight. By contrast, forward KL overestimates the variance and prunes more aggressively. The following example illustrates that while OBS outperforms OBD when pruning only a single weight, there is no guarantee that OBS is better than OBD when pruning multiple weights simultaneously since OBS may prune highly correlated weights all together.

Example 1. *Suppose a neural network converged to a local minima with weight* $\boldsymbol{\theta}^* = [1, 1, 1]^\top$*, and the associated Hessian* $\mathbf{H} = \begin{pmatrix} 1 & 0.99 & 0 \\ 0.99 & 1 & 0.01 \\ 0 & 0.01 & 0.5 \end{pmatrix}$*. Compute the resulting weight and increase in loss of OBD and OBS for the following cases.*

Case 1: Prune one weight (OBS is better).

- *OBD:* $\Delta\boldsymbol{\theta} = [0, 0, -1]^\top$, $\Delta\mathcal{L} = \Delta\mathcal{L}_{OBD} = 0.25$
- *OBS:* $\Delta\boldsymbol{\theta} = [-1, 0.99, 0.02]^\top$, $\Delta\mathcal{L} = \Delta\mathcal{L}_{OBS} = 0.01$

Case 2: Prune two weights simultaneously (OBD is better).

- *OBD:* $\Delta\boldsymbol{\theta} = [0, -1, -1]^\top$, $\Delta\mathcal{L} = 0.76(\Delta\mathcal{L}_{OBD} = 0.75)$
- *OBS:* $\Delta\boldsymbol{\theta} = [-1, -1, 0]^\top$, $\Delta\mathcal{L} = 1.99(\Delta\mathcal{L}_{OBS} = 0.02)$

OBD and OBS are equivalent when the true posterior distribution is fully factorized. It has been observed that different weights are highly coupled (Zhang et al., 2017) and diagonal approximation is too crude. However, the correlations are small in the KFE (see Figure 2). This motivates us to consider applying OBD in the KFE, where the diagonal approximation is more reasonable.

4. Methods

4.1. Approximating the Hessian with the Fisher Matrix

We use the Fisher matrix to approximate the Hessian. In the following, we briefly discuss the relationship between these matrices. For more detailed discussion, we refer readers to Martens (2014); Pascanu & Bengio (2013a).

Suppose the function $\mathbf{z} = f(\mathbf{x}, \boldsymbol{\theta})$ is parameterized by $\boldsymbol{\theta}$, and the loss function is $\ell(y, \mathbf{z}) = -\log p(y|\mathbf{z})$. Then the Hessian $\mathbf{H}$ at (local) minimum is equivalent to the generalized Gauss-Newton matrix $\mathbf{G}$:

$$\begin{aligned}\mathbf{H} &= \mathbb{E}\Big[\mathbf{J}_f^\top \mathbf{H}_\ell \mathbf{J}_f + \underbrace{\sum_{j=1}^{m} [\nabla_{\mathbf{z}} \ell(y, \mathbf{z})|_{\mathbf{z}=f(\mathbf{x},\boldsymbol{\theta})}]_j \mathbf{H}_{[f]_j}}_{\approx 0}\Big] \\ &= \mathbb{E}\Big[\mathbf{J}_f^\top \mathbf{H}_\ell \mathbf{J}_f\Big] = \mathbf{G}\end{aligned} \tag{11}$$

where $\nabla_{\mathbf{z}} \ell(y, \mathbf{z})|_{\mathbf{z}=f(\mathbf{x},\boldsymbol{\theta})}$ is the gradient of $\ell(y, \mathbf{z})$ evaluated at $z = f(\mathbf{x}, \boldsymbol{\theta})$, $\mathbf{H}_\ell$ is the Hessian of $\ell(y, \mathbf{z})$ w.r.t. $\mathbf{z}$, and $\mathbf{H}_{[f]_j}$ is the Hessian of j-th component of $f(\mathbf{x}, \boldsymbol{\theta})$.

Pascanu & Bengio (2013b) showed that the Fisher matrix $\mathbf{F}$ and generalized Gauss-Newton matrix are identical when the model predictive distribution is in the exponential family, such as categorical distribution (for classification) and Gaussian distribution (for regression), justifying the use of the Fisher to approximate the Hessian.

4.2. Extending OBD and OBS to Structured Pruning

OBD and OBS were originally used for weight-level pruning. Before introducing our main contributions, we first extend OBD and OBS to structured (channel/filter-level) pruning. The most naïve approach is to first compute the importance of every weight, *i.e.*, Eqn. (8) for OBD and Eqn. (10) for OBS, then sum together the importances within each filter. We use this approach as a baseline, and denote it C-OBD and C-OBS. For C-OBS, because inverting the Hessian/Fisher matrix is computationally intractable, we adopt the K-FAC approximation for efficient inversion, as first proposed by Zeng & Urtasun (2019) for weight-level pruning.

In the scenario of structured pruning, a more sophisticated approach is to take into account the correlation of the weights within the same filter. For example, we can compute the importance of each filter as follows:

$$\Delta\mathcal{L}_i = \frac{1}{2}\boldsymbol{\theta}_i^{*\top}\mathbf{F}(i)\boldsymbol{\theta}_i^* \tag{12}$$

where $\boldsymbol{\theta}_i^* \in \mathbb{R}^{c_{\mathrm{in}}k^2}$ and $\mathbf{F}(i) \in \mathbb{R}^{c_{\mathrm{in}}k^2 \times c_{\mathrm{in}}k^2}$ are the parameters vector and Fisher matrix of i-th filter $\mathcal{F}_i$, respectively. To do this, we would need to store the Fisher matrix for each filter, which is intractable for large convolutional layers. To overcome this problem, we adopt the K-FAC approximation $\mathbf{F} = \mathbf{S} \otimes \mathbf{A}$, and compute the change in weights as well as the importance in the following way:

$$\Delta\boldsymbol{\theta}_i = -\boldsymbol{\theta}_i^* \quad \text{and} \quad \Delta\mathcal{L}_i = \frac{1}{2}\mathbf{S}_{ii}\boldsymbol{\theta}_i^{*\top}\mathbf{A}\boldsymbol{\theta}_i^* \tag{13}$$

Unlike Eqn. (12), the input factor $\mathbf{A}$ is shared between different filters, and therefore cheap to store. By analogy,

Algorithm 1 Structured pruning algorithms Kron-OBD and Kron-OBS. For simplicity, we focus on a single layer. $\boldsymbol{\theta}_i$ below denotes the parameters of filter $\mathcal{F}_i$, which is a vector.

```
Require: pruning ratio $p$ and training data $\mathcal{D}$
Require: model parameters (pretrained) $\boldsymbol{\theta} = \mathrm{vec}(\mathbf{W})$
 1: Compute Kronecker factors $\mathbf{A} = \mathbb{E}[\mathbf{a}\mathbf{a}^\top]$ and $\mathbf{S} = \mathbb{E}[\{\nabla_\mathbf{s}\mathcal{L}\}\{\nabla_\mathbf{s}\mathcal{L}\}^\top]$
 2: for all filter $i$ do
 3:     $\Delta\mathcal{L}_i = \frac{1}{2}\mathbf{S}_{ii}\boldsymbol{\theta}_i^\top\mathbf{A}\boldsymbol{\theta}_i$ or $\Delta\mathcal{L}_i = \frac{1}{2}\frac{\boldsymbol{\theta}_i^\top\mathbf{A}\boldsymbol{\theta}_i}{[\mathbf{S}^{-1}]_{ii}}$
 4: end for
 5: Compute $p_{\text{th}}$ percentile of $\Delta\mathcal{L}$ as $\tau$
 6: for all filter $i$ do
 7:     if $\Delta\mathcal{L}_i \leq \tau$ then
 8:         $\boldsymbol{\theta}_i \leftarrow \mathbf{0}$ or $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \frac{\mathbf{S}^{-1}\mathbf{e}_i\otimes\boldsymbol{\theta}_i^*}{[\mathbf{S}^{-1}]_{ii}}$
 9:     end if
10: end for
11: Finetune the network on $\mathcal{D}$ until converge
```

we can compute the change in weights and importance of each filter for Kron-OBS as:

$$\Delta\boldsymbol{\theta} = -\frac{\mathbf{S}^{-1}\mathbf{e}_i\otimes\boldsymbol{\theta}_i^*}{[\mathbf{S}^{-1}]_{ii}} \quad \text{and} \quad \Delta\mathcal{L}_i = \frac{1}{2}\frac{\boldsymbol{\theta}_i^{*\top}\mathbf{A}\boldsymbol{\theta}_i^*}{[\mathbf{S}^{-1}]_{ii}}, \tag{14}$$

where $\mathbf{e}_i$ is the selecting vector with 1 for elements of $\mathcal{F}_i$ and 0 elsewhere. We refer to Eqn. (13) and Eqn. (14) as Kron-OBD and Kron-OBS (See Algorithm 1). See Appendix A for derivations.

4.3. EigenDamage: Structured Pruning in a KFE

As argued in Section 3, weight-level OBD and OBS approximate the posterior distribution with a factorized Gaussian around the mode, which is overly restrictive and cannot capture the correlation between weights. Although we just extended them to filter/channel pruning, which captures correlations of weights within the same filter, the interactions between filters are ignored. In this section, we propose to decorrelate the weights before pruning. In particular, we introduce a novel network reparameterization by breaking each linear operation into three stages. Intuitively, the role of the first and third stages is to rotate to the KFE.

Considering a single layer with weight $\mathbf{W}$ with K-FAC Fisher $\mathbf{S}\otimes\mathbf{A}$ (see Section 2), we can decompose the weight matrix $\mathbf{W}$ as the following form:

$$\mathrm{vec}\{\mathbf{W}\} = (\mathbf{Q}_\mathbf{S}\otimes\mathbf{Q}_\mathbf{A})\mathrm{vec}\{\mathbf{W}\} = \mathrm{vec}\{\mathbf{Q}_\mathbf{A}\mathbf{W}'\mathbf{Q}_\mathbf{S}^\top\} \tag{15}$$

where $\mathrm{vec}\{\mathbf{W}'\} = (\mathbf{Q}_\mathbf{S}\otimes\mathbf{Q}_\mathbf{A})^\top\mathrm{vec}\{\mathbf{W}\}$. It is easy to show that the Fisher matrix for $\mathbf{W}'$ is diagonal if the assumptions of K-FAC are satisfied (George et al., 2018). We then apply C-OBD (or equivalently C-OBS since the Fisher is close to diagonal) on $\mathbf{W}'$ for *both* input and output channels. This way, each layer has a bottleneck structure which is a low-rank approximation, which we term *eigenpruning*. (Note that C-OBD and Kron-OBD only prune the output channels, since it automatically results in removal of corresponding input channel in the next layer.) We refer to our proposed method as EigenDamage (See Algorithm 2).

Algorithm 2 Pruning in the Kronecker-factored eigenbasis, *i.e.*, EigenDamage. For simplicity, we focus on a single layer. $\odot$ denotes elementwise mutliplication.

```
Require: pruning ratio $p$ and training data $\mathcal{D}$
Require: model parameters (pretrained) $\boldsymbol{\theta} = \mathrm{vec}(\mathbf{W})$
 1: Compute Kronecker factors $\mathbf{A} = \mathbb{E}[\mathbf{a}\mathbf{a}^\top]$ and $\mathbf{S} = \mathbb{E}[\{\nabla_\mathbf{s}\mathcal{L}\}\{\nabla_\mathbf{s}\mathcal{L}\}^\top]$
 2: $\mathbf{Q}_\mathbf{S}, \boldsymbol{\Lambda}_\mathbf{S} = \mathrm{Eigen}(\mathbf{S})$ and $\mathbf{Q}_\mathbf{A}, \boldsymbol{\Lambda}_\mathbf{A} = \mathrm{Eigen}(\mathbf{A})$
 3: Decompose weight $\mathbf{W}$ according by Eqn. (15)
 4: $\boldsymbol{\Theta} = \mathbf{W}' \odot \mathrm{diag}(\boldsymbol{\Lambda}_\mathbf{A})\mathrm{diag}(\boldsymbol{\Lambda}_\mathbf{S})^\top \odot \mathbf{W}'$
 5: for all row $r$ or column $c$ in $\boldsymbol{\Theta}$ do
 6:     $\Delta\mathcal{L}_r = \boldsymbol{\Theta}_{r,\cdot}\mathbf{1}$ and $\Delta\mathcal{L}_c = \mathbf{1}^\top\boldsymbol{\Theta}_{\cdot,c}$
 7: end for
 8: Compute $p_{\text{th}}$ percentile of $\Delta\mathcal{L}$ as $\tau$
 9: Remove $r_{\text{th}}$ row (or $c_{\text{th}}$ column) in $\mathbf{W}'$ and $r_{\text{th}}$ (or $c_{\text{th}}$) eigenbasis in $\mathbf{Q}_\mathbf{A}$ (or $\mathbf{Q}_\mathbf{S}$) if $\Delta\mathcal{L}_r$ (or $\Delta\mathcal{L}_c$) $\leq \tau$
10: Finetune the network on $\mathcal{D}$ until convergence
```

EigenDamage preserves the input and output shape, and thus can be applied to any convolutional or fully connected architecture without modification, in contrast with Liu et al. (2017), which requires adaptions for networks with cross-layer connections. Furthermore, like all Hessian-based pruning methods, our criterion allows us to set one global compression ratio for the whole network, making it easy to use. Moreover, the introduced eigen-basis $\mathbf{Q}_\mathbf{A}$ can be further compressed by the "doubly factored" Kronecker approximation (Ba et al., 2016), and $\mathbf{W}'$ can be also compressed by depth-wise separable decomposition, as detailed in Sections 4.5 and 4.6.

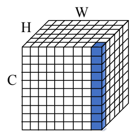
4.4. Iterative Pruning

The above method relies heavily on the Taylor expansion (6), which may be accurate if we prune only a few filters. Unfortunately, the approximation will break down if we prune a large number of filters. In order to handle this issue, we can conduct the pruning process iteratively and only prune a few filters each iteration. Specifically, once we finish pruning the network for the first time, each layer has a bottleneck structure (*i.e.*, $\mathbf{Q}_\mathbf{A}$, $\mathbf{W}'$, and $\mathbf{Q}_\mathbf{S}$). We can then conduct the next pruning iteration (after finetuning) on $\mathbf{W}'$ in the same manner. This will result in two new eigenbases associated with $\mathbf{W}'$. Conveniently, we can always merge these two new eigenbases (*i.e.*, $\mathbf{Q}'_\mathbf{A}$, $\mathbf{Q}'_\mathbf{S}$) into old ones so as to reduce the model size as well as FLOPs by:

$$\mathbf{Q}_\mathbf{A} \leftarrow \mathbf{Q}_\mathbf{A}\mathbf{Q}'_\mathbf{A} \quad \text{and} \quad \mathbf{Q}_\mathbf{S} \leftarrow \mathbf{Q}_\mathbf{S}\mathbf{Q}'_\mathbf{S} \tag{16}$$

This procedure may take several iterations until it reaches desirable compression ratio.

4.5. Reducing the Parameter Count of the Eigenbasis



Since the eigenbasis $\mathbf{Q_A}$ can take up a large chunk of memory for convolutional networks[1], we further leverage the internal structure to reduce the model size. Inspired by Ba et al. (2016)'s "doubly factored" Kronecker approximation for layers whose input feature maps are too large, we ignore the correlation among the spatial locations within the same input channel. In that case, $\mathbf{A} \in \mathbb{R}^{c_{\text{in}} \times c_{\text{in}}}$ only captures the correlation between different channels. Here we abuse the notation slightly and let $\mathbf{A}$ denote the covariance matrix along the channel dimension and $\mathbf{a} \in \mathbb{R}^{c_{\text{in}}}$ (see blue cubes) the activation of each spatial location:

$$\mathbf{A} = \mathbb{E}\left[\mathbf{a}\mathbf{a}^\top\right] = \frac{1}{N|\mathcal{T}|} \sum_{\mathbf{x}} \sum_{t \in \mathcal{T}} \mathbf{a}_t(\mathbf{x}) \mathbf{a}_t(\mathbf{x})^\top \qquad (17)$$

The expectation in Eqn. (17) is taken over training examples $\mathbf{x}$ and spatial locations $\mathcal{T}$. We note that with such approximation, $\mathbf{Q_A}$ can be efficiently implemented by 1×1 conv, resulting in compact bottleneck structures like ResNet (He et al., 2016a), as shown in Figure 1. This will greatly reduce the size of eigen-basis to be $1/k^4$ of the original one.

4.6. Depthwise Separable Decomposition

Depthwise separable convolution has been proven to be effective in designing lightweight models (Howard et al., 2017; Chollet, 2017; Zhang et al., 2018b; Ma et al., 2018). The idea of separable convolution can be naturally incorporated in our method to further reduce the computational cost and model size. For convolution filters $\mathbf{W}' \in \mathbb{R}^{c_{\text{in}} \times c_{\text{out}} \times k \times k}$, we perform the singular value decomposition (SVD) for every slice $\mathbf{W}'_{:,:,i} \in \mathbb{R}^{c_{\text{in}} \times c_{\text{out}}}$; then we can get a diagonal matrix as well as two new bases, as shown in Figure 3 (a). However, such a decomposition will result in more than twice the original parameters due to the two new bases. Therefore, we again ignore the correlation along the spatial dimension of filters, *i.e.* sharing the basis for each spatial dimension (see Figure 3 (b)). In particular, we solve the following problem:

$$\min_{\mathbf{U}, \{\mathbf{D}_i\}_{i=1}^{k^2}, \mathbf{V}} \frac{1}{2} \sum_{i=1}^{k^2} ||\mathbf{U}\mathbf{D}_i\mathbf{V}^\top - \mathbf{W}'_{:,:,i}||^2_{\text{Frob}} \qquad (18)$$

where $\mathbf{U} \in \mathbb{R}^{c_{\text{in}} \times c_{\text{in}}}$, $\mathbf{D}_i \in \mathbb{D}^{c_{\text{in}} \times c_{\text{out}}}$[2] and $\mathbf{V} \in \mathbb{R}^{c_{\text{out}} \times c_{\text{out}}}$. We can merge $\mathbf{U}$ and $\mathbf{V}$ into $\mathbf{Q}_{\text{A}}$ and $\mathbf{Q}_{\text{S}}$ respectively, and

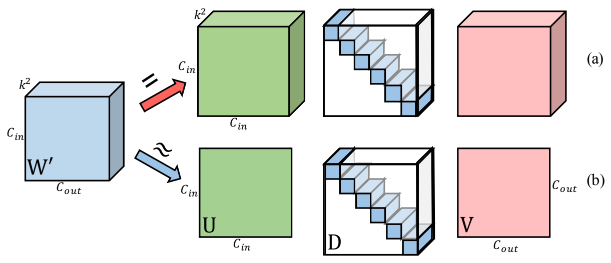


Figure 3. Two schemes for depthwise separable decomposition of the convolution layer. The parameters in the blank region are zeros.

then replace $\mathbf{W}'$ with $\mathbf{D}$, which can be implemented with a depthwise convolution. By doing so, we are able to further reduce the size of the filter to be $1/c_{\text{in}}$ of the original one.

5. Experiments

In this section, we aim to verify the effectiveness of EigenDamage in reducing the test-time resource requirements of a network without significantly sacrificing accuracy. We compare EigenDamage with other compression methods in terms of test accuracy, reduction in weights, reduction in FLOPs, and inference wall-clock time speedup. Wherever possible, we analyze the tradeoff curves involving test accuracy and resource requirements. We find that EigenDamage gives a significantly more favorable tradeoff curve, especially on larger architectures and more difficult datasets.

5.1. Experimental Setup

We test our methods on two network architectures: VGGNet (Simonyan & Zisserman, 2014) and (Pre)ResNet[3] (He et al., 2016b;a). We make use of three standard benchmark datasets: CIFAR10, CIFAR100 (Krizhevsky, 2009) and Tiny-ImageNet[4]. We compare EigenDamage to the extended versions C-OBD/OBS and Kron-OBD/OBS as well as one state-of-the-art channel-level pruning algorithm, NN Slimming (Liu et al., 2017; 2018), and a low-rank approximation algorithm, CP-Decomposition (Jaderberg et al., 2014). Note that because NN Slimming requires imposing L_1 loss on the scaling weights of BatchNorm (Ioffe & Szegedy, 2015), we train the networks with two different settings, *i.e.*, with and without L_1 loss, for fair comparison.

For networks with skip connections, NN Slimming can only be applied to specially designed network architectures. Therefore, in addition to ResNet32, we also test on PreResNet-29 (He et al., 2016b), which is in the same family of architectures considered by Liu et al. (2017). In our experiments, all the baseline (i.e. unpruned) networks are trained from scratch with SGD. We train the networks

[1]$\mathbf{Q_A}$ has the shape of $c_{\text{in}}k^2 \times c_{\text{in}}k^2$.
[2]$\mathbb{D}^{c_{\text{in}} \times c_{\text{out}}}$ is the domain of diagonal matrices.

[3]For ResNet, we widen the network by a factor of 4, as done in Zhang et al. (2018a)
[4]https://tiny-imagenet.herokuapp.com

Table 1. One-pass pruning on CIFAR10 and CIFAR100 with VGG19, ResNet32 and PreResNet29. To be noted, we cannot control the pruned ratio of parameters since we prune the whole filter and different filters are not of the same size. We run each experiment five times, and present the mean and standard variance.

Dataset	**CIFAR10**						**CIFAR100**					
Prune Ratio (%)	60%			90%			60%			90%		
Method	Test acc (%)	Reduction in weights (%)	Reduction in FLOPs (%)	Test acc (%)	Reduction in weights (%)	Reduction in FLOPs (%)	Test acc (%)	Reduction in weights (%)	Reduction in FLOPs (%)	Test acc (%)	Reduction in weights (%)	Reduction in FLOPs (%)
VGG19(Baseline)	94.17	-	-	-	-	-	73.34	-	-	-	-	-
NN Slimming (Liu et al., 2017)	92.84 ± -	80.07 ± -	42.65 ± -	85.01 ± -	97.85 ± -	97.89 ± -	71.89 ± -	74.60 ± -	38.33 ± -	58.69 ± -	97.76 ± -	94.09 ± -
C-OBD	94.04 ± 0.12	82.01 ± 0.44	38.18 ± 0.45	92.34 ± 0.18	97.68 ± 0.02	77.39 ± 0.36	72.23 ± 0.15	77.03 ± 0.05	33.70 ± 0.04	58.07 ± 0.60	97.97 ± 0.04	77.55 ± 0.25
C-OBS	94.08 ± 0.07	76.96 ± 0.14	34.73 ± 0.11	91.92 ± 0.16	97.27 ± 0.04	87.53 ± 0.41	72.27 ± 0.13	73.83 ± 0.03	38.09 ± 0.06	58.87 ± 1.34	97.61 ± 0.01	91.94 ± 0.26
Kron-OBD	94.00 ± 0.11	80.40 ± 0.26	38.19 ± 0.55	**92.92 ± 0.26**	97.47 ± 0.02	81.44 ± 0.68	72.29 ± 0.11	77.24 ± 0.10	37.90 ± 0.24	60.70 ± 0.51	97.56 ± 0.08	82.55 ± 0.39
Kron-OBS	**94.09 ± 0.12**	79.71 ± 0.26	36.93 ± 0.15	92.56 ± 0.21	97.32 ± 0.02	80.39 ± 0.21	72.12 ± 0.14	74.18 ± 0.04	36.59 ± 0.11	60.66 ± 0.35	97.48 ± 0.03	83.57 ± 0.27
EigenDamage	93.98 ± 0.06	78.18 ± 0.12	37.13 ± 0.41	92.29 ± 0.21	97.15 ± 0.04	86.51 ± 0.26	**72.90 ± 0.06**	76.64 ± 0.12	37.40 ± 0.11	**65.18 ± 0.10**	97.31 ± 0.01	88.63 ± 0.12
VGG19+L_1 (Baseline)	93.71	-	-	-	-	-	73.08	-	-	-	-	-
NN Slimming (Liu et al., 2017)	93.79 ± -	83.45 ± -	49.23 ± -	**91.99 ± -**	97.93 ± -	86.00 ± -	72.78 ± -	76.53 ± -	39.92 ± -	57.07 ± -	97.59 ± -	93.86± -
C-OBD	93.84 ± 0.04	84.19 ± 0.01	47.34 ± 0.02	91.29 ± 0.30	97.88 ± 0.02	81.22 ± 0.38	72.73 ± 0.09	79.47 ± 0.02	39.04 ± 0.02	56.49 ± 0.06	97.96 ± 0.03	80.91 ± 0.16
C-OBS	93.85 ± 0.01	82.88 ± 0.02	44.58 ± 0.10	91.14 ± 0.13	97.31 ± 0.03	88.18 ± 0.27	72.58 ± 0.09	76.17 ± 0.01	41.61 ± 0.06	44.18 ± 0.87	97.31 ± 0.02	91.90 ± 0.07
Kron-OBD	93.86 ± 0.06	84.78 ± 0.00	50.10 ± 0.00	91.14 ± 0.26	97.74 ± 0.02	83.09 ± 0.33	72.44 ± 0.03	79.99 ± 0.02	43.46 ± 0.02	57.59 ± 0.21	97.53 ± 0.02	85.04 ± 0.07
Kron-OBS	93.84 ± 0.04	84.33 ± 0.03	48.01 ± 0.13	91.13 ± 0.17	97.37 ± 0.01	81.52 ± 0.18	72.61 ± 0.15	77.27 ± 0.03	40.89 ± 0.59	57.61 ± 0.67	97.51 ± 0.02	86.60 ± 0.14
EigenDamage	**93.88 ± 0.04**	79.50 ± 0.02	39.84 ± 0.11	91.79 ± 0.16	96.84 ± 0.02	84.82 ± 0.21	**73.01 ± 0.05**	75.41 ± 0.03	37.46 ± 0.06	**64.91 ± 0.23**	97.28 ± 0.04	88.65 ± 0.06
ResNet32(Baseline)	95.30	-	-	-	-	-	78.17	-	-	-	-	-
NN Slimming (Liu et al., 2017)	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A
C-OBD	95.11 ± 0.10	70.36 ± 0.39	66.18 ± 0.46	91.75 ± 0.42	97.30 ± 0.06	93.50 ± 0.37	75.70 ± 0.31	66.68 ± 0.25	67.53 ± 0.25	59.52 ± 0.24	97.74 ± 0.08	94.88 ± 0.08
C-OBS	95.04 ± 0.07	67.90 ± 0.25	76.75 ± 0.36	90.04 ± 0.21	95.49 ± 0.22	97.39 ± 0.04	75.16 ± 0.32	66.83 ± 0.03	76.59 ± 0.34	58.20 ± 0.56	91.99 ± 0.07	96.27 ± 0.02
Kron-OBD	95.11 ± 0.09	63.97 ± 0.22	63.41 ± 0.42	92.57 ± 0.09	96.11 ± 0.12	94.18 ± 0.17	75.86 ± 0.37	63.92 ± 0.23	62.97 ± 0.17	62.42 ± 0.41	96.42 ± 0.05	95.85 ± 0.08
Kron-OBS	95.14 ± 0.07	64.21 ± 0.31	61.89 ± 0.79	92.76 ± 0.12	96.14 ± 0.27	94.37 ± 0.54	**75.98 ± 0.33**	62.36 ± 0.40	60.41 ± 1.02	63.62 ± 0.50	93.56 ± 0.14	95.65 ± 0.13
EigenDamage	**95.17 ± 0.12**	71.99 ± 0.13	70.25 ± 0.24	**93.05 ± 0.23**	96.05 ± 0.03	94.74 ± 0.02	75.51 ± 0.11	69.80 ± 0.11	71.62 ± 0.21	**65.72 ± 0.04**	95.21 ± 0.04	94.62 ± 0.06
PreResNet29+L_1 (Baseline)	94.42	-	-	-	-	-	75.70	-	-	-	-	-
NN Slimming (Liu et al., 2017)	92.32 ± -	71.60 ± -	80.95 ± -	82.50 ± -	93.49 ± -	95.88 ± -	68.87 ± -	61.68 ± -	82.03 ± -	49.48 ± -	93.70 ± -	96.33 ± -
C-OBD	91.17 ± 0.16	87.48 ± 0.23	78.14 ± 0.70	80.03 ± 0.21	98.45 ± 0.02	96.03 ± 0.10	62.19 ± 0.18	89.72 ± 0.01	82.24 ± 0.16	36.44 ± 0.90	98.65 ± 0.00	96.81 ± 0.02
C-OBS	91.64 ± 0.22	83.52 ± 0.12	76.33 ± 0.21	76.59 ± 0.69	98.34 ± 0.02	98.47 ± 0.02	68.10 ± 0.29	81.26 ± 0.10	89.47 ± 0.04	32.77 ± 0.89	97.89 ± 0.01	98.73 ± 0.00
Kron-OBD	90.22 ± 0.43	74.84 ± 0.20	67.83 ± 0.33	82.68 ± 0.20	98.18 ± 0.04	94.90 ± 0.13	57.76 ± 0.28	76.85 ± 0.06	72.38 ± 0.02	34.26 ± 1.12	98.62 ± 0.00	96.09 ± 0.00
Kron-OBS	89.02 ± 0.17	72.96 ± 0.20	70.14 ± 0.18	81.77 ± 0.59	98.44 ± 0.01	96.85 ± 0.09	60.28 ± 0.37	70.53 ± 0.11	76.60 ± 0.14	33.45 ± 0.96	98.31 ± 0.00	97.15 ± 0.01
EigenDamage	**93.80 ± 0.05**	70.09 ± 0.12	63.13 ± 0.26	**89.10 ± 0.13**	93.45 ± 0.04	90.67 ± 0.06	**73.62 ± 0.16**	66.73 ± 0.17	62.86 ± 0.12	**65.11 ± 0.15**	92.33 ± 0.02	90.52 ± 0.02

for 150 epochs for CIFAR datasets and 300 epochs for Tiny-ImageNet with an initial learning rate of 0.1 and weight decay of $2e^{-4}$. The learning rate is decayed by a factor of 10 at $\frac{1}{2}$ and $\frac{3}{4}$ of the total number of training epochs. For the networks trained with L_1 sparsity on BatchNorm, we followed the same settings as in Liu et al. (2017).

5.2. One-pass Pruning Results

We first consider the single-pass setting, where we perform a single round of pruning, and then fine-tune the network. Specifically, we compare eigenpruning[5] (EigenDamage) against our proposed baselines C-OBD, C-OBS, Kron-OBD, Kron-OBS and a state-of-the-art channel-level pruning method, NN Slimming, on CIFAR10 and CIFAR100 with VGGNet and (Pre)ResNet. For all methods, we test a variety of pruning ratios, ranging from 0.5 to 0.9. Due to the space limit, please refer to Appendix C for the full results. In order to avoiding pruning all the channels in some layers, we constrain that at most 95% of the channels can be pruned at each layer. After pruning, the network is finetuned for 150 epochs with an initial learning rate of $1e^{-3}$ and weight decay of $1e^{-4}$. The learning rate decay follows the same scheme as in training. We run each experiment 5 times in order to reduce the variance of the results.

Results on CIFAR datasets. The results on CIFAR datasets are presented in Table 1. It shows that even C-OBD and C-OBS can almost match NN slimming on CIFAR10 and CIFAR100 with VGGNet, if trained with L_1 sparsity on BatchNorm, and outperform when trained without it.

[5]For EigenDamage, we count both the parameters of $\mathbf{W}'$ and two eigenbasis.

Table 2. One pass pruning on Tiny-ImageNet with VGG19. To be noted, the network for NN Slimming is pretrained with L_1 loss as required by the method. See Appendix C for the full results.

Prune Ratio (%)	50%		
Method	Test acc (%)	Reduction in weights (%)	Reduction in FLOPs (%)
VGG19(Baseline)	61.56	-	-
VGG19+L_1(Baseline)	60.68	-	-
NN Slimming (Liu et al., 2017)	50.90 ± -	60.14 ± -	85.42 ± -
C-OBD	51.10 ± 0.60	69.27 ± 0.22	63.61 ± 0.19
C-OBS	53.13 ± 0.47	57.99 ± 0.52	78.51 ± 0.56
Kron-OBD	53.82 ± 0.32	67.22 ± 0.19	76.11 ± 0.24
Kron-OBS	53.54 ± 0.32	64.51 ± 0.23	74.57 ± 0.29
EigenDamage	**58.20 ± 0.30**	61.87 ± 0.11	66.21 ± 0.15

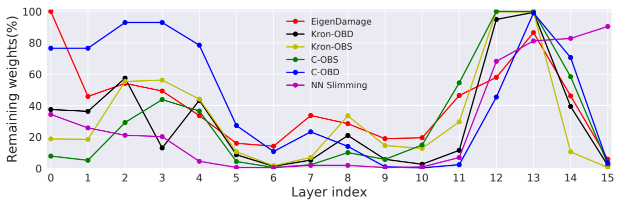


Figure 4. The percentage of remaining weights at each conv layer after one-pass pruning with a ratio of 0.5 on Tiny-ImageNet with VGG19. The legend is sorted in descending order of test accuracy.

Moreover, when the pruning ratio is 90%, two channel-level variants outperform NN Slimming on CIFAR100 with VGGNet by $\sim 2\%$ in terms of test accuracy. For the experiments on ResNet, EigenDamage achieves better performance ($\sim 2\%$) than others when the pruning ratio is 90% on CIFAR-100 dataset. Besides, for the experiments on PreResNet, EigenDamage achieves the best performance in terms of test accuracy on all configurations and outperforms other baselines by a bigger margin.

To summarize, EigenDamage performs the best across almost all the settings, and the improvements become more

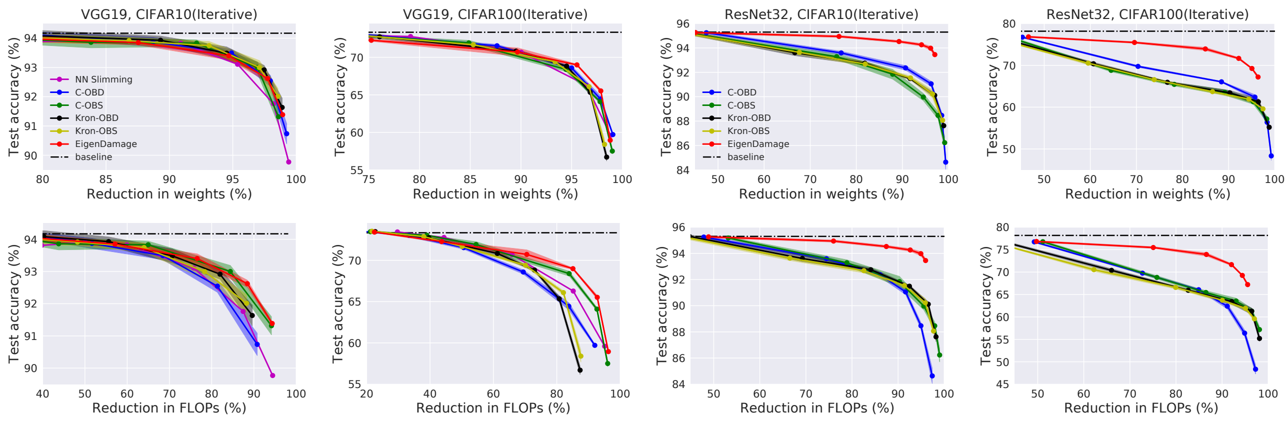


Figure 5. The results of iterative pruning. The first row are the curves of reduction in weights vs. test accuracy, and second row are the curves of pruned FLOPs vs. test accuracy of VGGNet and ResNet trained on CIFAR10 and CIFAR100 datasets. The shaded areas represent the variance over five runs.

significant when the pruning ratio is high, *e.g.* 90%, especially on more complicated networks, *e.g.* (Pre)ResNet, which demonstrates the effectiveness of pruning in the KFE. Moreover, EigenDamage adopts the bottleneck structure, which preserves the input and output dimension, as illustrated in Figure 1, and thus can be trivially applied to any fully connected or convolution layer without modification.

As we mentioned in Section 2, the success of loss-aware pruning algorithms relies on the approximation to the loss function for identifying unimportant weights/filters. Therefore, we visualize the loss on training set after one-pass pruning (without fintuning) in Figure 6. For EigenDamage, we can see that for VGG19 on CIFAR10, when even prune 80% of the weights, the increase in loss is negligible, and for other settings, the loss is also significantly lower than for other methods. For the remaining methods, which conduct pruning in the original weight space, they all result in a large increase in loss, and the resulting network performs similarly to uniform predictions in terms of loss.

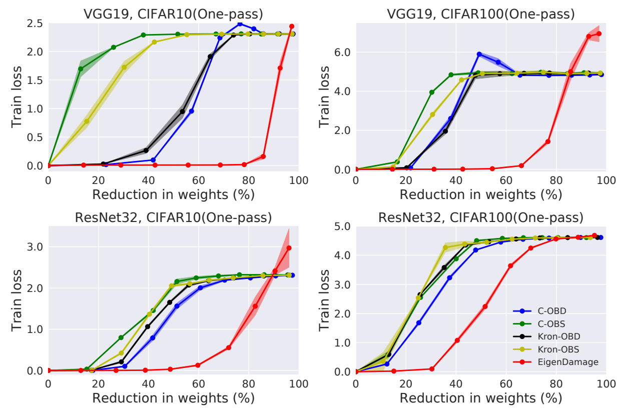


Figure 6. The above four figures show the training loss after one-pass pruning (without finetuning) vs. reduction in weights. The network pruned by EigenDamage achieves significantly lower loss on the training set. This shows that pruning in the KFE is very accurate in reflecting the sensitivity of loss to the weights.

Results on Tiny-ImageNet dataset. Apart from the results on CIFAR datasets, we futher test our methods on a more challenging dataset, Tiny-ImageNet, with VGGNet. Tiny-ImageNet consists of 200 classes and 500 images per class for training, and 10,000 images for testing, which are downsampled from the original ImageNet dataset. The results are in Table 2. Again, EigenDamage outperforms all the baselines by a significant margin.

We further plot the pruning ratio in each convolution layer for a detailed analysis. As shown in Figure 4, NN Slimming tends to prune more in the bottom layers but retain most of the filters in the top layers, which is undesirable since neural networks typically learn compact representations in the top. This may explain why NN Slimming performs worse than other methods in Tiny-ImageNet (see Table 2). By contrast, EigenDamage yields a balanced pruning ratio across different layers (retains most filters in bottom layers while pruning most redundant weights in top layer).

5.3. Iterative Pruning Results

We further experiment with the iterative setting, where the pruning can be conducted iteratively until it reaches a desired model size or FLOPs. Concretely, the iterative pruning is conducted for 6 times with a pruning ratio of 0.5 at each iteration for simulating the process. In order to avoiding pruning the entire layer, we also adopt the same strategy as in Liu et al. (2017), *i.e.* we constrain that at most 50% of the channels can be pruned in each layer for each iteration.

We compare EigenDamage to C-OBD, C-OBS, Kron-OBD and Kron-OBS. The results are summarized in Figure 5. We notice that EigenDamage performs slightly better than

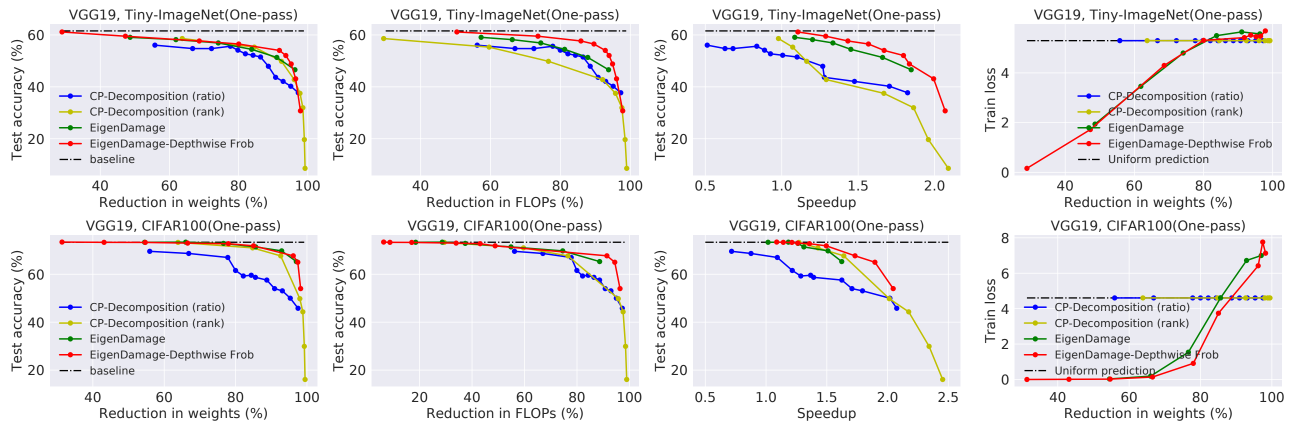


Figure 7. Low-rank approximation results on VGG19 on CIFAR100 and Tiny-ImageNet. The results are obtained by varying either the ranks of approximation or the pruning ratios.

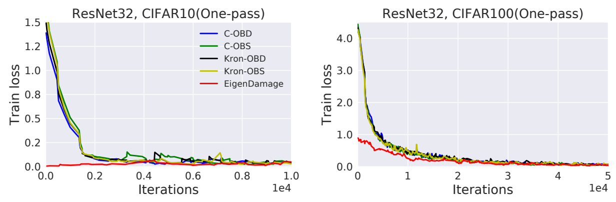


Figure 8. Loss on training set when finetuning the network after pruning (with a ratio of 0.5) with ResNet32 on CIFAR10 and CIFAR100 datasets.

other baselines with VGGNet and achieves significantly higher performance on ResNet. Specifically, for the results on CIFAR10 dataset with VGGNet, nearly all the methods achieved similar results due to the simplicity of CIFAR10 and VGGNet. However, the performance gap is a bit more clear as the dataset becoming more challenging, *e.g.*, CIFAR100. On a more sophisticated network, ResNet, the performance improvements of EigenDamage were especially significant on CIFAR10 or CIFAR100. Furthermore, EigenDamage was especially effective in reducing the number of FLOPs, due to the bottleneck structure.

5.4. Comparisons with Low-rank Approximation

Since EigenDamage can also be viewed as low-rank approximation, we compared it with a state-of-the-art low-rank method, CP-Decomposition (Lebedev et al., 2014), which computes a low-rank decomposition of the filter into a sum of rank-one tensors. We experimented low-rank approximation for VGG19 on CIFAR100 and Tiny-ImageNet. For CP-Decomposition, we tested it under two settings: (1) we varied the ranks from $\{0.1, \ldots, 1.0, 1.25, 1.5, 2.0\}$ times of the original rank at each layer; (2) we varied ranks in $\{4, 8, \ldots, 512\}$ for computing the approximation[6]. For EigenDamage, we chose different pruning ratios in the range of $\{0, 4, \ldots, 0.9\}$, and EigenDamage-Depthwise Frob is obtained by applying depthwise separable decomposition on the network obtained by EigenDamage.

[6]We choose the minimum of the target rank and the original rank of the convolution filter as the rank for approximation.

The results are presented in Figure 7. EigenDamage outperforms CP-Decomposition significantly in terms of speedup and accuracy. Moreover, CP-Decomposition approximates the original weights under the Frobenius norm in the original weight coordinates, which does not precisely reflect the sensitivity to the training loss. In contrast, EigenDamage is loss-aware, and thus the resulting approximation will achieve lower training loss when only pruning is applied, *i.e.* without finetuning, as is shown in the Figure 7. Note that EigenDamage will determine the approximation rank for each layer automatically given a global pruning ratio. However, CP-Decomposition requires pre-determined approximation rank for each layer and thus the search complexity will grow exponentially in the number of layers.

6. Conclusion

In this paper, we introduced a novel network reparameterization based on the Kronecker-factored eigenbasis, in which the entrywise independence assumption is approximately satisfied. This lets us prune the weights effectively using Hessian-based pruning methods. The pruned networks give low-rank (bottleneck structure) which allows for fast computation. Empirically, EigenDamage outperforms strong baselines which do pruning in original parameter coordinates, especially on more chanllenging datasets and networks.

Acknowledgements

We thank Shengyang Sun, Ricky Chen, David Duvenaud, Jonathan Lorraine for their feedback on early drafts. GZ was funded by an MRIS Early Researcher Award.

References

Ba, J., Grosse, R., and Martens, J. Distributed second-order optimization using kronecker-factored approximations. 2016.

Bader, B. W. and Kolda, T. G. Efficient matlab computations with sparse and factored tensors. *SIAM Journal on Scientific Computing*, 30(1):205–231, 2007.

Bae, J., Zhang, G., and Grosse, R. Eigenvalue corrected noisy natural gradient. *arXiv preprint arXiv:1811.12565*, 2018.

Chollet, F. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1251–1258, 2017.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pp. 248–255. Ieee, 2009.

Denton, E. L., Zaremba, W., Bruna, J., LeCun, Y., and Fergus, R. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in neural information processing systems*, pp. 1269–1277, 2014.

Desjardins, G., Simonyan, K., Pascanu, R., et al. Natural neural networks. In *Advances in Neural Information Processing Systems*, pp. 2071–2079, 2015.

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

George, T., Laurent, C., Bouthillier, X., Ballas, N., and Vincent, P. Fast approximate natural gradient descent in a kronecker-factored eigenbasis. *arXiv preprint arXiv:1806.03884*, 2018.

Graves, A. Practical variational inference for neural networks. In *Advances in neural information processing systems*, pp. 2348–2356, 2011.

Grosse, R. and Martens, J. A kronecker-factored approximate fisher matrix for convolution layers. In *International Conference on Machine Learning*, pp. 573–582, 2016.

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

Han, S., Pool, J., Tran, J., and Dally, W. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pp. 1135–1143, 2015b.

Han, S., Liu, X., Mao, H., Pu, J., Pedram, A., Horowitz, M. A., and Dally, W. J. Eie: efficient inference engine on compressed deep neural network. In *Computer Architecture (ISCA), 2016 ACM/IEEE 43rd Annual International Symposium on*, pp. 243–254. IEEE, 2016.

Hanson, S. J. and Pratt, L. Y. Comparing biases for minimal network construction with back-propagation. In *Advances in neural information processing systems*, pp. 177–185, 1989.

Hassibi, B., Stork, D. G., and Wolff, G. J. Optimal brain surgeon and general network pruning. In *Neural Networks, 1993., IEEE International Conference on*, pp. 293–299. IEEE, 1993.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016a.

He, K., Zhang, X., Ren, S., and Sun, J. Identity mappings in deep residual networks. In *European conference on computer vision*, pp. 630–645. Springer, 2016b.

He, Y., Zhang, X., and Sun, J. Channel pruning for accelerating very deep neural networks. 2017.

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

Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., and Bengio, Y. Binarized neural networks. In *Advances in neural information processing systems*, pp. 4107–4115, 2016.

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

Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H., and Kalenichenko, D. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2704–2713, 2018.

Jaderberg, M., Vedaldi, A., and Zisserman, A. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*, 2014.

Krizhevsky, A. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

Lebedev, V., Ganin, Y., Rakhuba, M., Oseledets, I., and Lempitsky, V. Speeding-up convolutional neural networks using fine-tuned cp-decomposition. *arXiv preprint arXiv:1412.6553*, 2014.

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

Li, F., Zhang, B., and Liu, B. Ternary weight networks. *arXiv preprint arXiv:1605.04711*, 2016a.

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

Lin, Z., Courbariaux, M., Memisevic, R., and Bengio, Y. Neural networks with few multiplications. *arXiv preprint arXiv:1510.03009*, 2015.

Liu, Z., Li, J., Shen, Z., Huang, G., Yan, S., and Zhang, C. Learning efficient convolutional networks through network slimming. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, pp. 2755–2763. IEEE, 2017.

Liu, Z., Sun, M., Zhou, T., Huang, G., and Darrell, T. Rethinking the value of network pruning. *arXiv preprint arXiv:1810.05270*, 2018.

Luo, J.-H., Wu, J., and Lin, W. Thinet: A filter level pruning method for deep neural network compression. *arXiv preprint arXiv:1707.06342*, 2017.

Ma, N., Zhang, X., Zheng, H.-T., and Sun, J. Shufflenet v2: Practical guidelines for efficient cnn architecture design. *arXiv preprint arXiv:1807.11164*, 1, 2018.

MacKay, D. J. A practical bayesian framework for backpropagation networks. *Neural computation*, 4(3):448–472, 1992.

Martens, J. New insights and perspectives on the natural gradient method. *arXiv preprint arXiv:1412.1193*, 2014.

Martens, J. and Grosse, R. Optimizing neural networks with kronecker-factored approximate curvature. In *International conference on machine learning*, pp. 2408–2417, 2015.

Minka, T. et al. Divergence measures and message passing. Technical report, 2005.

Murphy, K. P. *Machine Learning: A Probabilistic Perspective*. MIT Press, 2012.

Neyshabur, B., Li, Z., Bhojanapalli, S., LeCun, Y., and Srebro, N. Towards understanding the role of overparametrization in generalization of neural networks. *arXiv preprint arXiv:1805.12076*, 2018.

Novikov, A., Podoprikhin, D., Osokin, A., and Vetrov, D. P. Tensorizing neural networks. In *Advances in Neural Information Processing Systems*, pp. 442–450, 2015.

Pascanu, R. and Bengio, Y. Revisiting natural gradient for deep networks. 2013a.

Pascanu, R. and Bengio, Y. Revisiting natural gradient for deep networks. *arXiv preprint arXiv:1301.3584*, 2013b.

Ritter, H., Botev, A., and Barber, D. A scalable laplace approximation for neural networks. 2018.

Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

Zeng, W. and Urtasun, R. MLPrune: Multi-layer pruning for automated neural network compression, 2019. URL https://openreview.net/forum?id=r1g5b2RcKm.

Zhang, C., Bengio, S., Hardt, M., Recht, B., and Vinyals, O. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*, 2016.

Zhang, G., Sun, S., Duvenaud, D., and Grosse, R. Noisy natural gradient as variational inference. *arXiv preprint arXiv:1712.02390*, 2017.

Zhang, G., Wang, C., Xu, B., and Grosse, R. Three mechanisms of weight decay regularization. *arXiv preprint arXiv:1810.12281*, 2018a.

Zhang, X., Zhou, X., Lin, M., and Sun, J. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018b.

Zhu, C., Han, S., Mao, H., and Dally, W. J. Trained ternary quantization. *arXiv preprint arXiv:1612.01064*, 2016.

Supplementary Material

A. Derivation of Kron-OBD and Kron-OBS

Derivation of Kron-OBD. Assuming that the weight of a conv layer is $\boldsymbol{\theta} = \mathrm{vec}\left(\mathbf{W}\right)$, where $\mathbf{W} \in \mathbb{R}^{n\times m}$, $n = c_{\mathrm{in}}k^2$ and $m = c_{\mathrm{out}}$, and the two Kronecker factors are $\mathbf{S} \in \mathbb{R}^{m\times m}$ and $\mathbf{A} \in \mathbb{R}^{n\times n}$. Then the Fisher information matrix of $\boldsymbol{\theta}$ can be approximated by $\mathbf{F} = \mathbf{S} \otimes \mathbf{A}$. Substituting the Hessian with K-FAC Fisher in eqn. (8), we get:

$$\Delta\mathcal{L} = \frac{1}{2}\Delta\boldsymbol{\theta}^\top \left(\mathbf{S}\otimes\mathbf{A}\right)\Delta\boldsymbol{\theta} = \frac{1}{2}\mathrm{Tr}\left(\Delta\mathbf{W}^\top\mathbf{A}\Delta\mathbf{W}\mathbf{S}\right) = \frac{1}{2}\sum_{i,j}\mathbf{S}_{ij}\Delta\boldsymbol{\theta}_i^\top\mathbf{A}\Delta\boldsymbol{\theta}_j \tag{19}$$

where $\Delta\boldsymbol{\theta}_i$ represents the change in $\boldsymbol{\theta}_i^*$, and $\boldsymbol{\theta}_i^* \in \mathbb{R}^n$ is the weight of i-th filter $\mathcal{F}_i$, *i.e.*, i-th column of $\mathbf{W}$. Under the assumption that each filter is independent to each other, and thus $\mathbf{S}$ is diagonal. So, we can get the importance of each filter and the corresponding change in weights are:

$$\Delta\mathcal{L}_i = \frac{1}{2}\mathbf{S}_{ii}\boldsymbol{\theta}_i^{*\top}\mathbf{A}\boldsymbol{\theta}_i^* \ \text{ and } \ \Delta\boldsymbol{\theta}_i = -\boldsymbol{\theta}_i^* \tag{20}$$

Derivation of Kron-OBS. Under the assumption of Kron-OBS that different filters are correlated to each other, $\mathbf{S}$ is no longer diagonal. Then, similar to eqn. (20), the corresponding structured version of eqn.(9) becomes:

$$\min_i\left\{\min_{\Delta\mathbf{W}}\frac{1}{2}\mathrm{Tr}\left(\Delta\mathbf{W}^\top\mathbf{A}\Delta\mathbf{W}\mathbf{S}\right)\right\} \ \text{ s.t. } \ \Delta\mathbf{W}\mathbf{e}_i + \boldsymbol{\theta}_i^* = \mathbf{0} \tag{21}$$

We can solve the above constrained optimization problem with Lagrange multiplier:

$$\min_i\left\{\min_{\Delta\mathbf{W}}\frac{1}{2}\mathrm{Tr}\left(\Delta\mathbf{W}^\top\mathbf{A}\Delta\mathbf{W}\mathbf{S}\right) - \boldsymbol{\lambda}^\top\left(\Delta\mathbf{W}\mathbf{e}_i + \boldsymbol{\theta}_i^*\right)\right\} \tag{22}$$

Taking the derivatives w.r.t to $\Delta\mathbf{W}$ and set it to $\mathbf{0}$, we get:

$$\Delta\mathbf{W} = \mathbf{A}^{-1}\boldsymbol{\lambda}\mathbf{e}_i^\top\mathbf{S}^{-1} \tag{23}$$

Substitute it back to the constrain to solve the equation, we get:

$$\boldsymbol{\lambda} = \frac{-\mathbf{A}\boldsymbol{\theta}_i^*}{[\mathbf{S}^{-1}]_{ii}} \tag{24}$$

Then substitute eqn. (24) back to eqn. (23), we can finally get the optimal change in weights if we remove filter $\mathcal{F}_i$:

$$\Delta\mathbf{W} = -\frac{\boldsymbol{\theta}_i^*}{[\mathbf{S}^{-1}]_{ii}}\mathbf{e}_i^\top\mathbf{S}^{-1} \ \text{ and } \ \Delta\boldsymbol{\theta} = -\frac{\mathbf{S}^{-1}\mathbf{e}_i\otimes\boldsymbol{\theta}_i^*}{[\mathbf{S}^{-1}]_{ii}} \tag{25}$$

In order to evaluating the importance of each filter, we can substitute eqn. (25) back to eqn. (21):

$$\begin{aligned}\Delta\mathcal{L}_i &= \frac{1}{2}\mathrm{Tr}\left(\mathbf{S}^{-1}\mathbf{e}_i\frac{\boldsymbol{\theta}_i^{*\top}}{[\mathbf{S}^{-1}]_{ii}}\mathbf{A}\frac{\boldsymbol{\theta}_i^*}{[\mathbf{S}^{-1}]_{ii}}\mathbf{e}_i^\top\mathbf{S}^{-1}\mathbf{S}\right) = \frac{1}{2}\mathrm{Tr}\left(\frac{\boldsymbol{\theta}_i^{*\top}\mathbf{A}\boldsymbol{\theta}_i^*}{[\mathbf{S}^{-1}]_{ii}^2}\mathbf{S}^{-1}\mathbf{e}_i\mathbf{e}_i^\top\right)\\ &= \frac{1}{2}\mathrm{Tr}\left(\frac{\boldsymbol{\theta}_i^{*\top}\mathbf{A}\boldsymbol{\theta}_i^*}{[\mathbf{S}^{-1}]_{ii}^2}[\mathbf{S}^{-1}]_{ii}\right) = \frac{1}{2}\frac{\boldsymbol{\theta}_i^{*\top}\mathbf{A}\boldsymbol{\theta}_i^*}{[\mathbf{S}^{-1}]_{ii}}\end{aligned} \tag{26}$$

B. Algorithm for Solving eqn. (18)

In this section, we will introduce the algorithm for solving the optimization problem in eqn. (18).

Khatri-Rao product. The Khatri-Rao product $\odot$ of two matrices $\mathbf{A} \in \mathbb{R}^{m\times r}$ and $\mathbf{B} \in \mathbb{R}^{n\times r}$ is the column-wise Kronecker product, that is:

$$\mathbf{A}\odot\mathbf{B} = \begin{pmatrix} a_{11}b_{11} & a_{12}b_{12} & \cdots & a_{1r}b_{1r} \\ a_{11}b_{21} & a_{12}b_{22} & \cdots & a_{1r}b_{2r} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}b_{n1} & a_{m2}b_{n2} & \cdots & a_{mr}b_{nr} \end{pmatrix} \in \mathbb{R}^{mn\times r} \tag{27}$$

Kruskal tensor notation. Suppose $\mathbf{T} \in \mathbb{R}^{n_1\times n_2\times\cdots\times n_d}$ has low-rank Canonical Polyadic (CP) structure. Following (Bader & Kolda, 2007), we refer to it as a *Kruskal tensor*. Normally, it can be defined by a collection of factor matrices, $\mathbf{A}_k \in \mathbb{R}^{n_k\times r}$ for $k = 1, ..., d$, such that:

$$\mathbf{T}(i_1, i_2, \cdots, i_d) = \sum_{j=1}^{r} \mathbf{A}_1(i_1, j)\mathbf{A}_2(i_2, j)\cdots\mathbf{A}_d(i_d, j) \quad \text{for all } (i_1, i_2, \cdots, i_d) \in \mathcal{I} \tag{28}$$

where $\mathcal{I} \equiv \{1, \cdots, n_1\}\otimes\{1, \cdots n_2\}\otimes\cdots\otimes\{1, \cdots, n_d\}$. Denote $\mathbf{T}_{(k)} \in \mathbb{R}^{n_k\times(n_d\cdots n_{k-1}n_{k+1}\cdots n_1)}$ is the mode-k unfolding of a Kruskal tensor, which has the following form that depends on the Khatri-Rao products of the factor matrices:

$$\mathbf{T}_{(k)} = \mathbf{A}_k\mathbf{Z}_k^\top \quad \text{where} \;\; \mathbf{Z}_k \equiv \mathbf{A}_d \odot\cdots\odot \mathbf{A}_{k+1}\odot\mathbf{A}_{k-1}\odot\cdots\odot\mathbf{A}_1 \tag{29}$$

Alternating Least Squares (ALS). We can use ALS to solve problems similar to eqn. (18). Suppose we are approximating $\mathbf{T}$ using $\mathbf{A}_1, \cdots, \mathbf{A}_d$. Specifically, for fixed $\mathbf{A}_1, \cdots, \mathbf{A}_{k-1}, \mathbf{A}_{k+1}, \cdots, \mathbf{A}_d$, there is a closed form solution for $\mathbf{A}_k$. Specifically, we can update update $\mathbf{A}_1, \cdots, \mathbf{A}_d$ by the following update rule:

$$\mathbf{A}_k^\top = \mathbf{Z}_k^\dagger \mathbf{T}_{(k)}^\top \quad \text{for } k = 1, \cdots, d \tag{30}$$

alternatively until converge or reach the maximum number of iterations. For the Mahalanobis norm case (with $\mathbf{F}$ as the metric tensor), if we take the derivative with respect to $\mathbf{A}_k$ to be $\mathbf{0}$,

$$\text{unvec}(\mathbf{F}\text{vec}(\mathbf{A}_k\mathbf{Z}_k^\top - \mathbf{T}_{(k)}))\mathbf{Z} = \mathbf{0} \tag{31}$$

we can get the corresponding update rule for $\mathbf{A}_k$:

$$\mathbf{A}_k^\top = \mathbf{Z}_k^\dagger \left(\mathbf{T}_{(k)} + \text{unvec}(\mathbf{F}^{-1}\text{vec}(\mathbf{P}))\right)^\top \tag{32}$$

where unvec and vec are inverse operators to each other, and in our case, unvec operation is to convert the vectorized matrix back to the original matrix form. $\mathbf{Z}^\dagger = (\mathbf{Z}^\top\mathbf{Z})^{-1}\mathbf{Z}^\top$ and $\mathbf{P}$ has the same shape with $\mathbf{T}_k$, and for each column $\mathbf{P}_i \in \text{Null}(\mathbf{Z}_k^\top)$.

C. Additional Results on One-pass Pruning

We present the additional results on one-pass pruning in the following tables. We also present the data in tables as trade-off curves in terms of acc vs. reduction in weight and acc vs. reduction in FLOPs for making it easy to tell the difference in performances of each method.

Table 3. One pass pruning on CIFAR-10 with VGG19

Prune Ratio (%)	50%			70%			80%		
Method	Test acc (%)	Reduction in weights (%)	Reduction in FLOPs (%)	Test acc (%)	Reduction in weights (%)	Reduction in FLOPs (%)	Test acc (%)	Reduction in weights (%)	Reduction in FLOPs (%)
VGG19(Baseline)	94.17	-	-	-	-	-	-	-	-
NN Slimming (Liu et al., 2017)	92.84 ± -	73.84 ± -	38.88 ± -	92.89 ± -	84.30 ± -	54.83 ± -	91.92 ± -	91.77 ± -	76.43 ± -
C-OBD	94.01 ± 0.15	76.84 ± 0.30	35.07 ± 0.38	94.04 ± 0.09	85.88 ± 0.10	41.17 ± 0.23	93.70 ± 0.07	92.17 ± 0.07	56.87 ± 0.33
C-OBS	**94.19 ± 0.10**	66.91 ± 0.08	26.12 ± 0.13	93.97 ± 0.16	84.97 ± 0.02	43.16 ± 0.20	93.77 ± 0.12	91.52 ± 0.09	63.64 ± 0.13
Kron-OBD	93.91 ± 0.16	73.93 ± 0.42	33.71 ± 0.69	93.95 ± 0.12	85.80 ± 0.09	43.78 ± 0.24	93.78 ± 0.17	92.04 ± 0.04	60.81 ± 0.26
Kron-OBS	94.03 ± 0.13	69.17 ± 0.20	28.02 ± 0.26	94.10 ± 0.15	85.83 ± 0.09	42.56 ± 0.14	**93.87 ± 0.14**	92.00 ± 0.04	60.19 ± 0.38
EigenDamage	94.15 ± 0.05	68.64 ± 0.19	28.09 ± 0.21	**94.15 ± 0.14**	85.78 ± 0.06	45.68 ± 0.31	93.68 ± 0.22	92.51 ± 0.05	66.98 ± 0.36
VGG19+L_1 (Baseline)	93.71	-	-	-	-	-	-	-	-
NN Slimming (Liu et al., 2017)	93.79 ± -	77.44 ± -	45.19 ± -	93.74 ± -	88.81 ± -	52.15 ± -	**93.48 ± -**	92.60 ± -	62.23 ± -
C-OBD	93.85 ± 0.03	76.83 ± 0.01	41.14 ± 0.05	93.88 ± 0.03	89.04 ± 0.03	52.73 ± 0.14	93.38 ± 0.04	93.29 ± 0.05	63.74 ± 0.12
C-OBS	**93.88 ± 0.04**	74.95 ± 0.03	37.56 ± 0.06	93.84 ± 0.04	88.53 ± 0.20	51.88 ± 0.00	93.27 ± 0.04	92.01 ± 0.02	63.96 ± 0.10
Kron-OBD	**93.88 ± 0.01**	83.43 ± 0.00	49.58 ± 0.00	**93.89 ± 0.03**	89.02 ± 0.01	53.40 ± 0.09	93.33 ± 0.05	93.55 ± 0.06	67.01 ± 0.30
Kron-OBS	93.85 ± 0.03	76.95 ± 0.01	42.04 ± 0.10	93.88 ± 0.04	88.69 ± 0.02	52.38 ± 0.08	93.44 ± 0.07	92.66 ± 0.05	63.77 ± 0.27
EigenDamage	93.84 ± 0.04	78.14 ± 0.11	39.02 ± 0.30	93.85 ± 0.04	85.71 ± 0.01	46.56 ± 0.03	93.40 ± 0.07	91.48 ± 0.06	62.18 ± 0.29

Table 4. One pass pruning on CIFAR-100 with VGG19

Prune Ratio (%)	50%			70%			80%		
Method	Test acc (%)	Reduction in weights (%)	Reduction in FLOPs (%)	Test acc (%)	Reduction in weights (%)	Reduction in FLOPs (%)	Test acc (%)	Reduction in weights (%)	Reduction in FLOPs (%)
VGG19(Baseline)	73.34	-	-	-	-	-	-	-	-
NN Slimming (Liu et al., 2017)	72.77 ± -	66.50 ± -	30.61 ± -	69.98 ± -	85.56 ± -	54.51 ± -	66.09 ± -	92.33 ± -	76.76 ± -
C-OBD	72.82 ± 0.15	65.47 ± 0.13	24.24 ± 0.10	71.10 ± 0.22	86.06 ± 0.04	41.18 ± 0.04	67.46 ± 0.26	93.31 ± 0.06	60.39 ± 0.25
C-OBS	72.73 ± 0.17	62.31 ± 0.05	25.50 ± 0.06	71.25 ± 0.21	84.49 ± 0.04	49.25 ± 0.52	67.47 ± 0.13	91.04 ± 0.06	68.38 ± 0.26
Kron-OBD	72.88 ± 0.12	67.11 ± 0.21	28.57 ± 0.19	71.16 ± 0.11	85.83 ± 0.10	47.19 ± 0.35	67.70 ± 0.32	92.86 ± 0.05	65.26 ± 0.26
Kron-OBS	72.89 ± 0.12	67.26 ± 0.08	25.80 ± 0.16	71.36 ± 0.17	84.75 ± 0.02	45.74 ± 0.17	68.17 ± 0.34	92.16 ± 0.03	63.95 ± 0.19
EigenDamage	**73.39 ± 0.12**	66.05 ± 0.11	28.55 ± 0.11	**71.62 ± 0.14**	85.69 ± 0.05	54.83 ± 0.79	**69.50 ± 0.22**	92.92 ± 0.03	74.55 ± 0.33
VGG19+L_1 (Baseline)	73.08	-	-	-	-	-	-	-	-
NN Slimming (Liu et al., 2017)	73.24 ± -	72.68 ± -	35.37 ± -	71.55 ± -	84.38 ± -	51.59 ± -	66.55 ± -	92.48 ± -	76.54 ± -
C-OBD	73.39 ± 0.05	74.16 ± 0.01	35.13 ± 0.01	71.40 ± 0.09	86.13 ± 0.01	45.47 ± 0.10	67.56 ± 0.16	93.00 ± 0.01	63.42 ± 0.11
C-OBS	**73.44 ± 0.04**	71.17 ± 0.03	33.77 ± 0.67	71.30 ± 0.12	84.07 ± 0.01	56.74 ± 0.13	66.90 ± 0.23	91.20 ± 0.04	73.39 ± 0.31
Kron-OBD	73.24 ± 0.05	74.00 ± 0.03	36.56 ± 0.03	71.01 ± 0.13	86.66 ± 0.05	52.66 ± 0.21	67.24 ± 0.20	92.90 ± 0.05	68.62 ± 0.21
Kron-OBS	73.20 ± 0.12	72.27 ± 0.03	36.45 ± 0.66	**71.88 ± 0.11**	84.77 ± 0.01	50.53 ± 0.08	67.75 ± 0.14	92.08 ± 0.01	67.39 ± 0.17
EigenDamage	73.23 ± 0.08	66.80 ± 0.02	29.49 ± 0.03	71.81 ± 0.13	84.27 ± 0.04	52.75 ± 0.21	**69.83 ± 0.24**	92.36 ± 0.01	73.68 ± 0.13

Table 5. One pass pruning on CIFAR-10 with ResNet

Prune Ratio (%)	50%			70%			80%		
Method	Test acc (%)	Reduction in weights (%)	Reduction in FLOPs (%)	Test acc (%)	Reduction in weights (%)	Reduction in FLOPs (%)	Test acc (%)	Reduction in weights (%)	Reduction in FLOPs (%)
ResNet32(Baseline)	95.30	-	-	-	-	-	-	-	-
C-OBD	95.27 ± 0.10	60.67 ± 0.44	55.46 ± 0.35	**95.00 ± 0.17**	80.64 ± 0.40	76.78 ± 0.63	94.41 ± 0.10	90.73 ± 0.11	86.66 ± 0.34
C-OBS	95.30 ± 0.15	58.99 ± 0.02	65.54 ± 0.25	94.43 ± 0.17	76.27 ± 0.35	85.89 ± 0.23	93.45 ± 0.25	86.15 ± 0.68	92.77 ± 0.05
Kron-OBD	95.30 ± 0.09	56.05 ± 0.24	52.21 ± 0.36	94.94 ± 0.02	73.98 ± 0.46	74.97 ± 0.63	**94.60 ± 0.14**	85.96 ± 0.41	86.36 ± 0.38
Kron-OBS	**95.46 ± 0.08**	56.48 ± 0.26	50.93 ± 0.46	94.92 ± 0.11	73.77 ± 0.24	74.58 ± 0.44	94.44 ± 0.08	85.65 ± 0.46	86.05 ± 0.55
EigenDamage	95.28 ± 0.16	59.68 ± 0.28	58.32 ± 0.23	94.86 ± 0.11	82.57 ± 0.27	80.88 ± 0.36	94.23 ± 0.13	90.48 ± 0.35	88.86 ± 0.50
PreResNet29+L_1 (Baseline)	94.42	-	-	-	-	-	-	-	-
NN Slimming (Liu et al., 2017)	92.60 ± -	58.12 ± -	69.88 ± -	90.60 ± -	81.99 ± -	87.05 ± -	87.24 ± -	86.68 ± -	91.33 ± -
C-OBD	92.85 ± 0.14	79.67 ± 0.41	68.90 ± 0.49	88.74 ± 0.32	93.22 ± 0.08	86.37 ± 0.32	85.75 ± 0.66	96.14 ± 0.04	91.39 ± 0.16
C-OBS	92.43 ± 0.03	73.81 ± 0.14	76.33 ± 0.21	85.85 ± 0.36	91.74 ± 0.17	92.39 ± 0.07	78.97 ± 0.84	96.55 ± 0.02	96.60 ± 0.06
Kron-OBD	93.19 ± 0.06	59.72 ± 0.31	54.26 ± 0.34	87.99 ± 0.37	86.50 ± 0.05	78.96 ± 0.17	86.40 ± 0.28	95.02 ± 0.04	88.58 ± 0.04
Kron-OBS	92.88 ± 0.08	58.95 ± 0.14	57.61 ± 0.12	87.71 ± 0.22	85.38 ± 0.03	82.00 ± 0.10	85.67 ± 0.31	93.93 ± 0.07	90.74 ± 0.31
EigenDamage	**94.15 ± 0.07**	62.06 ± 0.15	54.40 ± 0.10	**93.33 ± 0.07**	77.71 ± 0.11	71.92 ± 0.15	**92.30 ± 0.15**	86.27 ± 0.04	81.59 ± 0.07

Table 6. One pass pruning on CIFAR-100 with ResNet

Prune Ratio (%)	50%			70%			80%		
Method	Test acc (%)	Reduction in weights (%)	Reduction in FLOPs (%)	Test acc (%)	Reduction in weights (%)	Reduction in FLOPs (%)	Test acc (%)	Reduction in weights (%)	Reduction in FLOPs (%)
ResNet32(Baseline)	78.17	-	-	-	-	-	-	-	-
C-OBD	76.59 ± 0.06	57.82 ± 0.48	56.88 ± 0.37	73.74 ± 0.42	77.08 ± 0.32	78.64 ± 0.32	68.86 ± 0.40	89.67 ± 0.19	88.45 ± 0.19
C-OBS	76.26 ± 0.25	58.47 ± 0.22	63.81 ± 0.26	73.06 ± 0.23	74.33 ± 0.15	86.06 ± 0.03	63.15 ± 0.41	80.61 ± 0.15	91.10 ± 0.06
Kron-OBD	76.41 ± 0.29	53.08 ± 0.35	52.06 ± 0.27	72.82 ± 0.28	73.40 ± 0.11	75.25 ± 0.15	69.62 ± 0.38	84.50 ± 0.16	88.04 ± 0.10
Kron-OBS	**76.74 ± 0.32**	52.21 ± 0.20	49.40 ± 0.13	73.00 ± 0.19	71.60 ± 0.15	73.33 ± 0.48	70.42 ± 0.20	80.34 ± 0.25	86.96 ± 0.27
EigenDamage	76.12 ± 0.12	61.73 ± 0.12	60.87 ± 0.38	**73.82 ± 0.07**	79.85 ± 0.07	81.03 ± 0.11	**70.78 ± 0.08**	88.68 ± 0.08	88.68 ± 0.06
PreResNet29+L_1 (Baseline)	75.70	-	-	-	-	-	-	-	-
NN Slimming (Liu et al., 2017)	71.93 ± -	47.55 ± -	73.44 ± -	63.47 ± -	78.41 ± -	89.92 ± -	61.64 ± -	85.47 ± -	92.38 ± -
C-OBD	67.18 ± 0.10	84.72 ± 0.05	75.48 ± 0.11	55.02 ± 0.50	93.42 ± 0.01	88.12 ± 0.06	47.87 ± 0.57	96.57 ± 0.01	93.48 ± 0.02
C-OBS	71.17 ± 0.11	65.67 ± 0.12	80.77 ± 0.05	51.45 ± 0.74	91.88 ± 0.06	95.00 ± 0.03	41.55 ± 0.91	95.64 ± 0.02	97.20 ± 0.01
Kron-OBD	69.64 ± 0.19	56.63 ± 0.20	58.24 ± 0.05	46.46 ± 0.72	89.50 ± 0.07	82.44 ± 0.05	41.64 ± 0.86	95.83 ± 0.01	89.63 ± 0.06
Kron-OBS	69.87 ± 0.17	49.09 ± 0.13	62.59 ± 0.05	49.00 ± 0.68	87.03 ± 0.05	87.82 ± 0.10	40.51 ± 1.04	94.72 ± 0.01	93.86 ± 0.01
EigenDamage	**74.50 ± 0.13**	57.98 ± 0.15	53.87 ± 0.14	**72.41 ± 0.16**	75.79 ± 0.04	71.92 ± 0.12	**70.09 ± 0.11**	85.34 ± 0.02	81.61 ± 0.06

Table 7. One pass pruning on Tiny-ImageNet with VGG19. N/A denotes the network failed to converge, and achieves random guess performance on the test set.

Prune Ratio (%)	40%			60%			70%			80%		
Method	Test acc (%)	Reduction in weights (%)	Reduction in FLOPs (%)	Test acc (%)	Reduction in weights (%)	Reduction in FLOPs (%)	Test acc (%)	Reduction in weights (%)	Reduction in FLOPs (%)	Test acc (%)	Reduction in weights (%)	Reduction in FLOPs (%)
VGG19(Baseline)	61.56	-	-	-	-	-	-	-	-	-	-	-
NN Slimming (Liu et al., 2017)	57.40 ± -	52.61 ± -	76.99 ± -	40.05 ± -	71.04 ± -	90.04 ± -	N/A ± -	84.63 ± -	94.09 ± -	N/A ± -	93.02 ± -	95.45 ± -
C-OBD	56.87 ± 0.13	58.95 ± 0.06	55.49 ± 0.50	47.36 ± 0.47	79.10 ± 0.32	69.74 ± 0.38	43.61 ± 0.31	88.57 ± 0.14	74.90 ± 0.31	42.29 ± 0.53	95.62 ± 0.13	78.45 ± 0.46
C-OBS	56.72 ± 0.21	46.90 ± 0.18	68.87 ± 0.14	39.80 ± 0.53	67.46 ± 0.34	86.81 ± 0.25	35.30 ± 0.33	76.47 ± 0.07	92.71 ± 0.07	31.52 ± 0.66	86.19 ± 0.07	96.66 ± 0.04
Kron-OBD	56.81 ± 0.22	56.75 ± 0.27	66.96 ± 0.65	44.41 ± 0.82	76.55 ± 0.15	81.27 ± 0.16	41.03 ± 0.50	85.28 ± 0.11	86.12 ± 0.05	38.88 ± 0.43	95.02 ± 0.33	90.98 ± 0.34
Kron-OBS	56.47 ± 0.20	50.67 ± 0.11	62.28 ± 0.66	44.54 ± 0.43	73.88 ± 0.10	83.27 ± 0.13	41.44 ± 0.41	82.61 ± 0.41	87.91 ± 0.22	39.54 ± 0.20	92.77 ± 0.19	91.91 ± 0.33
EigenDamage	**59.09 ± 0.05**	48.62 ± 0.06	57.30 ± 0.12	**56.92 ± 0.23**	74.12 ± 0.15	74.37 ± 0.13	**54.46 ± 0.32**	83.77 ± 0.02	81.19 ± 0.16	**51.34 ± 0.37**	91.05 ± 0.06	87.82 ± 0.16

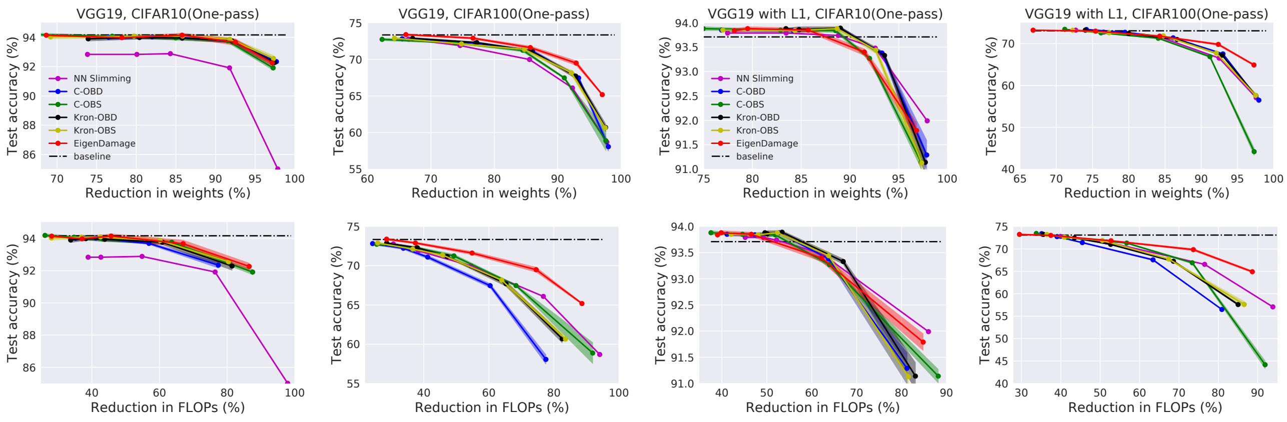


Figure 9. The results of one pass pruning, which are plotted based on the results in Tables. The first row are the curves of reduction in weights vs. test accuracy, and second row are the curves of pruned FLOPs vs. test accuracy of VGGNet trained on CIFAR10 and CIFAR100 dataset under the settings of with and without L_1 sparsity on BatchNorm. The shaded areas represent the standard variance over five runs.

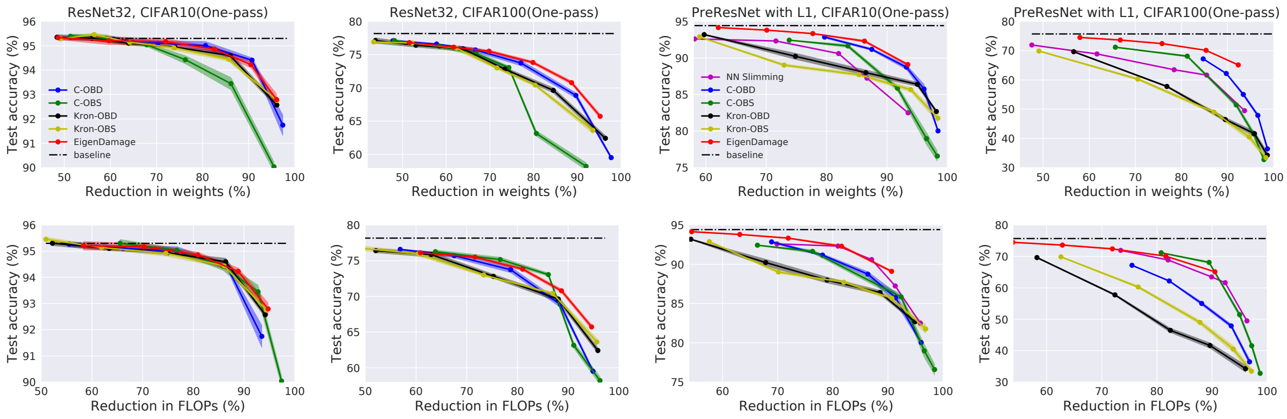


Figure 10. The results of one pass pruning, which are plotted based on the results in Tables. The first row are the curves of reduction in weights vs. test accuracy, and second row are the curves of pruned FLOPs vs. test accuracy of (Pre)ResNet trained on CIFAR10 and CIFAR100 dataset under the settings of with and without L_1 sparsity on BatchNorm. The shaded areas represent the standard variance over five runs.

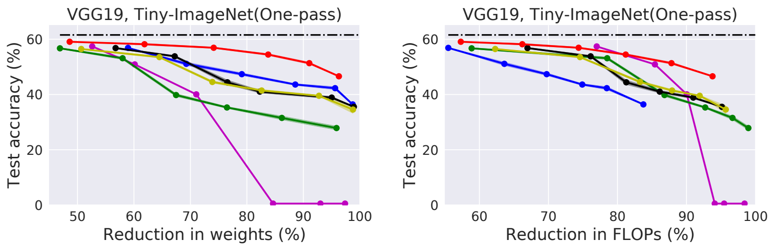


Figure 11. The results of one pass pruning, which are plotted based on the results in Tables. The base network for NN Slimming is pre-trained with L_1 sparsity on BatchNorm as required, and the others are normally pre-trained.