Hessian-Aware Pruning and Optimal Neural Implant

Shixing Yu*,[1], Zhewei Yao*,[2] Amir Gholami*,†,[2], Zhen Dong*,[2], Michael W. Mahoney[2], Kurt Keutzer[2]
[1]Peking University, [2]University of California, Berkeley
yushixing@pku.edu.cn, {zheweiy, amirgh, zhendong, mahoneymw, keutzer}@berkeley.edu

***Abstract*—Pruning is an effective method to reduce the memory footprint and FLOPs associated with neural network models. However, existing structured-pruning methods often result in significant accuracy degradation for moderate pruning levels. To address this problem, we introduce a new Hessian Aware Pruning (HAP) method coupled with a Neural Implant approach that uses second-order sensitivity as a metric for structured pruning. The basic idea is to prune insensitive components and to use a Neural Implant for moderately sensitive components, instead of completely pruning them. For the latter approach, the moderately sensitive components are replaced with with a low rank implant that is smaller and less computationally expensive than the original component. We use the relative Hessian trace to measure sensitivity, as opposed to the magnitude based sensitivity metric commonly used in the literature. We test HAP on multiple models on CIFAR-10/ImageNet, and we achieve new state-of-the-art results. Specifically, HAP achieves 94.3% accuracy ($< 0.1\%$ degradation) on PreResNet29 (CIFAR-10), with more than 70% of parameters pruned. Moreover, for ResNet50 HAP achieves 75.1% top-1 accuracy (0.5% degradation) on ImageNet, after pruning more than half of the parameters. The framework has been open sourced and available online [1].**

I. Introduction

There has been a significant increase in the computational resources required for Neural Network (NN) training and inference. This is in part due to larger input sizes (e.g., higher image resolution) as well as larger NN models requiring more computation with a significantly larger memory footprint. The slowing down of Moore's law, along with challenges associated with increasing memory bandwidth, has made it difficult to deploy these models in practice. Often, the inference time and associated power consumption is orders of magnitude higher than acceptable ranges. This has become a challenge for many applications, e.g., health care and personalized medicine, which have restrictions on uploading data to cloud servers, and which have to rely on local servers with limited resources. Other applications include inference on edge devices such as mobile processors, security cameras, and intelligent traffic control systems, all of which require real-time inference. Importantly, these problems are not limited to edge devices, and state-of-the-art models for applications such as speech recognition, natural language processing, and recommendation systems often cannot

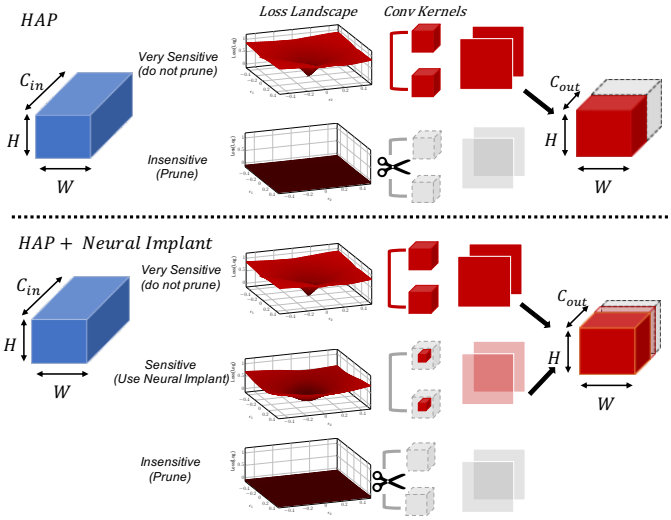

Fig. 1: *(Top) The* HAP *method is a structured pruning method that prunes channels based on their second-order sensitivity, which measured flatness/sharpness of loss landscape. Channels are sorted based on this metric, and only insensitive channels are pruned. (Bottom) Similar to other structured-pruning methods,* HAP *at large pruning ratios results in accuracy degradation. This because one has to inevitably prune moderately sensitive channels at high pruning ratios, which may contain individual neurons that are very sensitive. The removal of the entire channel, along with the sensitive neurons results in accuracy degradation. To address this, we propose* HAP+IMPLANT *method, where such channels are replaced with a light-weight, low-rank Neural Implant, and the network with the implant is then fine-tuned to recover performance. This enables higher accuracy with structured-pruning.*

*Equal contribution.
†Correspondence to: Amir Gholami: amirgh@berkeley.edu

be efficiently performed even on high-end servers.

A promising approach to address this is pruning. However, an important challenge is determining which parameters are insensitive to the pruning process. A brute-force method is not feasible since one has to test each parameter in the network separately and measure its sensitivity. The seminal work of [25] proposed Optimal Brain Damage (OBD), a second-order based method to determine insensitive parameters. However, this approach requires pruning the parameters one at a time, which is time-consuming. To address this problem, we propose a simple, yet effective, modification of OBD by using the Hessian trace to prune a group of parameters along with a low rank Neural Implant. In more detail, our contributions are as follows:

- We propose HAP, a Hessian Aware Pruning method that uses a fast second-order metric to find insensitive parameters in a NN model. In particular, we use the average Hessian trace to weight the magnitude of the parameters in the NN. Parameters with large second-order sensitivity remain unpruned, and those with relatively small sensitivity are pruned. In contrast to OBD [25], HAP finds groups of insensitive parameters, which is faster than pruning a single parameter at a time. Details of the HAP method are discussed in Section III.
- We propose a novel Neural Implant (denoted by HAP+IMPLANT) technique to alleviate accuracy degradation. In this approach, we replace moderately sensitive model components with a low rank implant. The model along with the implant is then fine-tuned. We find that this approach helps boost the accuracy. For details, see Section III-C.
- We perform detailed empirical testing and show that HAP achieves 94.3% accuracy ($< 0.1\%$ degradation) on PreResNet29 (CIFAR-10), with only 31% parameters left (Figure 3). In comparison to EigenDamage, a recent second-order pruning method, we achieve up to 1.2% higher accuracy with fewer parameters and FLOPs (Figure 3). Moreover, for ResNet50, HAP achieves 75.1% top-1 accuracy (0.5% degradation) on ImageNet, with only half of the parameters left (Table II). In comparison to prior state-of-the-art HRank [28], HAP achieves up to 2% higher accuracy with fewer parameters and FLOPs (Table II).
- We perform detailed ablation experiments to illustrate the efficacy of the second-order sensitivity metric. In particular, we compare the second-order sensitivity with a random method, and a reverse-order in which the opposite order sensitivity order given by HAP is used. In all cases, HAP achieves higher accuracy (Table IV).

II. RELATED WORK

Several different approaches have been proposed to make NN models more efficient by making them more compact, faster, and more energy efficient. These efforts could be generally categorized as follows: (i) efficient NN design [18, 19, 22, 34, 39, 50]; (ii) hardware-aware NN design [5, 10, 33, 40, 42, 45]; (iii) quantization [7, 8, 21, 23, 24, 43]; (iv) distillation [17, 36, 38, 48]; and (v) pruning.

Here we briefly discuss the related work on pruning, which can be broadly categorized into: unstructured pruning [6, 26, 37, 44]; and structured pruning [14, 20, 29, 32, 49, 51]. Unstructured pruning prunes out neurons without any structure. However, this leads to sparse matrix operations which are hard to accelerate and are typically memory-bounded [4, 9]. This can be addressed with structured pruning, where an entire matrix operation (e.g., an output channel) is removed. However, the challenge here is that high degrees of structured pruning often leads to significant accuracy degradation.

In both approaches, the key question is to find which parameters to prune. A simple and popular approach is magnitude-based pruning. In this approach, the magnitude of parameters is used as the pruning metric. The assumption here is that small parameters are not important and can be removed. A variant of this approach was used in [31], where the scaling factor of the batch normalization layer is used as the sensitivity metric. In particular, channels with smaller scaling factors (or output values) are considered less important and got pruned. Another variation is proposed by [27], where channel-wise summation over weights is used as the metric. Other methods have been proposed as alternative sensitivity metrics. For instance, [28] uses channel rank as sensitivity metric; [16] uses a LASSO regression based channel selection criteria; and [15] uses the geometric median of the convolutional filters. An important problem with magnitude-based pruning methods is that parameters with small magnitudes can actually be quite sensitive. It is easy to see this through a second-order Taylor series expansion, where the perturbation is dependent on not just the weight magnitude but also the Hessian [25]. In particular, small parameters with large Hessian could in fact be very sensitive, as opposed to large parameters with small Hessian (here, we are using small/large Hessian loosely; the exact metric to measure is given by the second-order perturbation in Eq. 4). For this reason, OBD [25] proposes to use the Hessian diagonal as

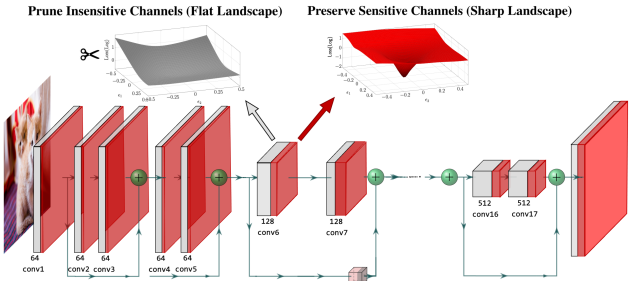

Fig. 2: *Illustration of Hessian-Aware Pruning (*HAP*). Channels are sorted based on their second-order sensitivity (Eq. 11). Insensitive channels are pruned (shown in gray), while sensitive channels are preserved (shown in red).*

the sensitivity metric. The follow up work of Optimal Brain Surgeon (OBS) [11, 12] used a similar method, but considered off-diagonal Hessian components, and showed a correlation with inverse Hessian. One important challenge with these methods is that pruning has to be performed one parameter at a time. The recent work of [6] extends this to layer-wise pruning in order to reduce the cost of computing Hessian information for one parameter at a time. However, this method can result in unstructured pruning. Another second-order pruning method is EigenDamage [41], where the Gauss-Newton operator is used instead of Hessian. In particular, the authors use Kronecker products to approximate the GN operator. Our findings below show that using the average Hessian trace method significantly outperforms EigenDamage. We also find that it is very helpful to replace moderately sensitive layers with a low rank Neural Implant, instead of completely pruning them, as discussed next.

III. Methodology

A. Background

Here, we focus on supervised learning tasks, where the nominal goal is to minimize the empirical risk by solving the following optimization problem:

$$L(w) = \frac{1}{N}\sum_{i=1}^{N} l(x_i, y_i, w), \tag{1}$$

where $w \in \mathbb{R}^n$ is the trainable model parameters, $l(x_i, y_i, w)$ is the loss for the input datum x_i, where y_i is the corresponding label, and N is the training set cardinality. For pruning, we assume that the model is already trained and converged to a local minima which satisfies the first and second-order optimality conditions (that is, the gradient $\nabla_w L(w) = 0$, and the Hessian is Positive Semi-Definite (PSD), $\nabla_w^2 L(w) \succcurlyeq 0$). The problem statement is to prune (remove) as many parameters as possible to reduce the model size and FLOPs to a target threshold with minimal accuracy degradation.

We first start with a general description of the problem and then derive our method. Let $\Delta w \in \mathbb{R}^n$ denote the pruning perturbation such that the corresponding weights become zero (that is $w + \Delta w = 0$). We denote the corresponding change of loss as ΔL:

$$\Delta L = L(w + \Delta w) - L(w). \tag{2}$$

From a Taylor series expansion, we have:

$$\Delta L = g^T \Delta w + \frac{1}{2}\Delta w^T H \Delta w + O(||\Delta w||^3), \tag{3}$$

where $g \in \mathbb{R}^n$ denotes the gradient of loss function L w.r.t. weights w, and $H \in \mathbb{R}^{n\times n}$ is the corresponding Hessian operator (i.e. second-order derivative). For a pretrained neural network that has already converged to a local minimum, we have $g = 0$, and the Hessian is a PSD matrix. As in prior work [12], we assume higher-order terms, e.g., $O(||\Delta w||^3)$, in Eq. 3 can be ignored.

The pruning problem is to find the set of weights that result in minimum perturbation to the loss (ΔL). This leads to the following constrained optimization problem:

$$\min_{\Delta w} \frac{1}{2}\Delta w^T H \Delta w = \frac{1}{2}\begin{pmatrix}\Delta w_p \\ \Delta w_l\end{pmatrix}^T \begin{pmatrix}H_{p,p} & H_{p,l} \\ H_{l,p} & H_{l,l}\end{pmatrix}\begin{pmatrix}\Delta w_p \\ \Delta w_l\end{pmatrix}, \quad \text{s.t.} \quad \Delta w_p + w_p = 0. \tag{4}$$

Here, we denote the channels that are pruned with p as the subscript (e.g. $w_p \in \mathbb{R}^p$), and denote the remaining parameters with l as the subscript (e.g. $w_l \in \mathbb{R}^{n-p}$). Similarly, we use Δw_p and Δw_l to denote the corresponding perturbations. Note that $\Delta w_p = -w_p$ since p-channels are pruned. Moreover, $H_{l,p}$ denotes the cross Hessian w.r.t. l-channels and p-channels (and similarly $H_{p,p}$ and $H_{l,l}$ are Hessian w.r.t. pruned and unpruned parameters). This optimization problem can be solved by forming the corresponding Lagrangian and finding its saddle points:

$$\begin{aligned}\mathcal{L} &= \frac{1}{2}\Delta w^T H \Delta w + \lambda^T(\Delta w_p + w_p), \\ \frac{\partial \mathcal{L}}{\partial \Delta w} &= H\Delta w + \begin{pmatrix}\lambda \\ 0\end{pmatrix} = 0, \\ \begin{pmatrix}H_{p,p} & H_{p,l} \\ H_{l,p} & H_{l,l}\end{pmatrix}&\begin{pmatrix}\Delta w_p \\ \Delta w_l\end{pmatrix} + \begin{pmatrix}\lambda \\ 0\end{pmatrix} = 0,\end{aligned} \tag{5}$$

where $\lambda \in \mathbb{R}^p$ is the Lagrange multiplier. By expanding this equation, we get:

$$H_{p,p}\Delta w_p + H_{p,l}\Delta w_l + \lambda = 0, \tag{6}$$

$$H_{l,p}\Delta w_p + H_{l,l}\Delta w_l = 0. \tag{7}$$

Using the constraint in Eq. 4 and adding it to Eq. 7, we have:

$$\begin{aligned} -H_{l,p}w_p + H_{l,l}\Delta w_l = 0, \\ \Delta w_l = H_{l,l}^{-1} H_{l,p} w_p. \end{aligned} \tag{8}$$

This equation gives us the optimal change to the unpruned parameters (w_l), if a pre-selected set of weights is pruned (w_p). Inserting this into Eq. 4, results in the following:

$$\frac{1}{2}\Delta w^T H \Delta w = \frac{1}{2} w_p^T (H_{p,p} - H_{p,l} H_{l,l}^{-1} H_{l,p}) w_p. \tag{9}$$

Eq. 9 gives us the perturbation to the loss when a set of parameters w_p is removed. It should be noted that OBS [12] and L-OBS [6], where OBS is applied for each layer under the assumption of cross-layer independence, is a degenerate case of Eq. 9 for the special case of $w_p \in \mathbb{R}^1$. Next we discuss how this general formulation can be simplified.

B. Hessian-aware Pruning

There are three major disadvantages with OBS. First, computing Eq. 9 requires computing (implicitly) information from the inverse Hessian, $H_{l,l}^{-1}$. This can be costly, both in terms of computations and memory (even when using matrix-free randomized methods). The work of L-OBS [6] attempted to address this challenge by ignoring cross-layer dependencies, but it still requires computing block-diagonal inverse Hessian information, which can be costly. Second, in both OBS and L-OBS, one has to measure this perturbation for all the parameters separately, and then prune those parameters that result in the smallest perturbation. This can have a high computational cost, especially for deep models with many parameters. Third, this pruning method results in unstructured pruning, which is difficult to accelerate with current hardware architectures.

In the OBD [25] method the first problem does not exist as the the Hessian is approximated as a diagonal operator, without the need to compute inverse Hessian:

$$\frac{1}{2}\Delta w^T H \Delta w \approx \frac{1}{2} w_p^T Diag(H_{p,p}) w_p. \tag{10}$$

Here $Diag(H_{p,p})$ denotes the diagonal elements of $H_{p,p}$. However, the second and third of these disadvantages still remain with OBD.

To address the second and third of these disadvantages, we propose to group the parameters and to compute the corresponding perturbation when that group is pruned, rather than computing the perturbation for every single parameter separately. Note that this can also address the third disadvantage, since pruning a group of parameters (for example parameters in a convolution channel) results in structured pruning. This can be achieved by considering the Hessian as a block diagonal operator, and then approximating each block with a diagonal operator, with Hessian trace as the diagonal entries. In particular, we use the following approximation:

$$\begin{aligned} \frac{1}{2}\Delta w^T H \Delta w &= \frac{1}{2} w_p^T [H^{-1}]_{p,p}^{-1} w_p \\ &\approx \frac{1}{2} w_p^T \frac{Trace(H_{p,p})}{p} w_p \\ &= \frac{Trace(H_{p,p})}{2p} \|w_p\|_2^2, \end{aligned} \tag{11}$$

where $Trace(H_{p,p})$ denotes the trace of the block diagonal Hessian (the corresponding Hessian block for pruned parameters $H_{p,p}$). The Hessian can be computed very efficiently with randomized numerical linear algebra methods, in particular Hutchinson's method [2, 3, 46, 47]. Importantly, this approach requires computing only the application of the Hessian to a random input vector. This has the same cost as back-propagating the gradient [46, 47]. (Empirically, in our experiments corresponding to ResNet50 on ImageNet, the longest time for computing this trace was three minutes.) A similar approach was proposed by [7] in the context of quantization.

Figure 2 shows a schematic illustration of HAP, where only sensitive layers are pruned, based on their second-order perturbation. In more detail, HAP performs structured pruning by grouping the parameters and approximating the corresponding Hessian as a diagonal operator, with the average Hessian trace of that group as its entries. For a convolutional network, this group can be an output channel. We found that this simple modification results in a fast and efficient pruning method that when combined with the Neural Implant approach exceeds state-of-the-art. This is discussed next.

C. Hessian-aware Neural Implant

In HAP, we sort the channels from most sensitive to least sensitive (based on Eq. 11). For a target model size or FLOPs budget, one has then to prune by starting from insensitive channels. This approach works well, as long as all these channels are extremely insensitive. However, in practice, some of the sorted channels will exhibit some level of sensitivity. Entirely pruning these channels, and leaving the rest of the sensitive ones unpruned, can result in significant accuracy degradation. This is one of the major problems with structured pruning methods, as very few groups of parameters are completely insensitive. As soon as those are pruned, the remaining groups/set of parameters will always include some subset of highly

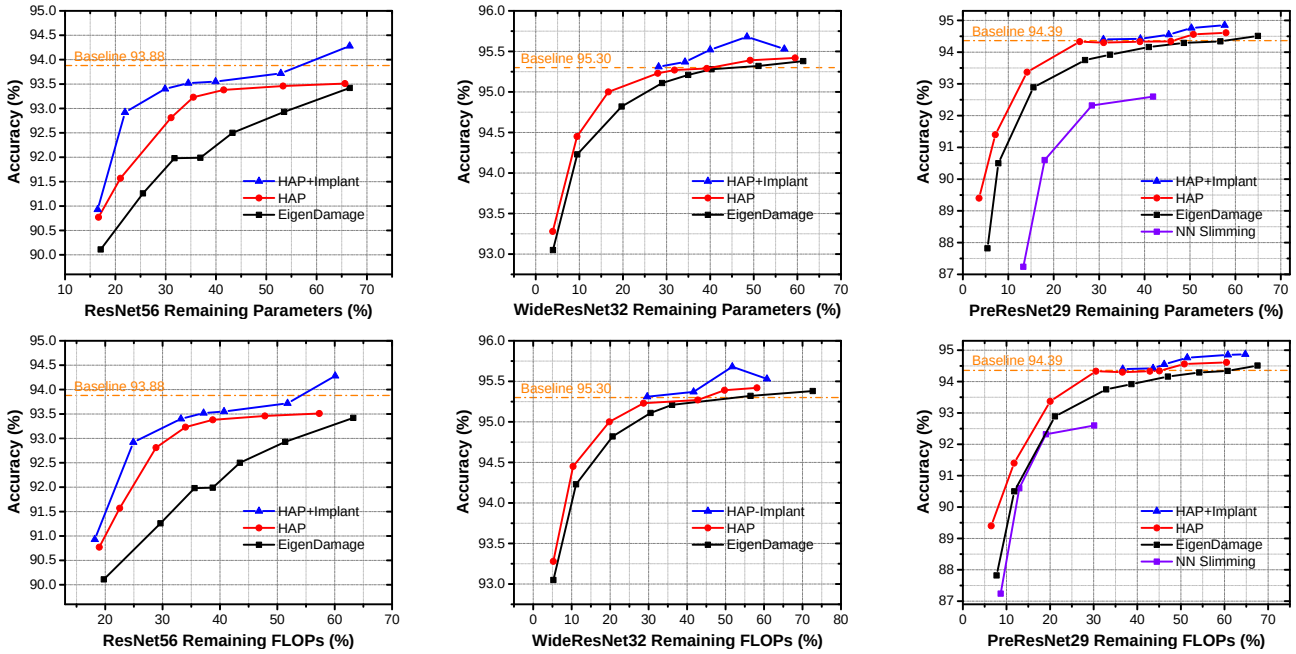

Fig. 3: *Comparison of accuracy with different pruning ratios among* HAP+IMPLANT, HAP, *NN Slimming, EigenDamage, and DCP, on the CIFAR-10 dataset, for ResNet56, WideResNet32, and PreResNet29. (Top) Remaining parameters in the network after pruning is used for x-axis. (Bottom) Remaining FLOPs in the network after pruning is used for x-axis.* HAP *consistently outperforms EigenDamage and NN Slimming, and* HAP+IMPLANT *boosts performance for moderate pruning ratios and surpasses DCP.*

sensitive neurons that if pruned, would result in high accuracy loss.

Here, we propose an alternative strategy to replace moderately sensitive parameter groups with a low rank *Neural Implant*. The basic idea is to prune insensitive layers completely, but detect the moderately sensitive layers, and instead of completely removing all of its parameters (which can contain some sensitive ones as discuss above), replace them with a low rank decomposition. As an example, a spatial convolution could be replaced with a new point-wise convolution that has smaller parameters and flops. One could also consider other types of low rank decomposition (e.g. CP/Tucker decomposition, depth-wise/separable convolution,etc). However, for simplicity we only use a pointwise convolution implant in this paper.

After the implant, the model is fine-tuned to recover accuracy. We denote this approach as HAP+IMPLANT, which is schematically illustrated in Figure 1. In summary, we use the Hessian sensitivity metric in Eq. 11, and then we apply a Neural Implant to the most sensitive channels to be pruned.

We have to emphasize that many prior works have investigated low-rank matrix approximation [35]. However, existing methods for NN pruning replace all or part of the model, irrespective of their sensitivity, whereas in our approach we perform a targeted low-rank approximation, and only replace the sensitive parts of the model, quantified through the Hessian in Eq. 11. We empirically found that this approach is quite effective, especially for moderate pruning ratios, as discussed next.

IV. RESULTS

A. Experimental Settings

For evaluating the performance of HAP, we conduct experiments for image classification on CIFAR-10 (using ResNet56/WideResNet32/PreResNet29/VGG16) and ImageNet (using ResNet50). Our main target comparison for HAP (without Implant) is EigenDamage, a recent second-order pruning method. For fair comparison, we use the same pretrained model used by EigenDamage when available (WideResNet32 on Cifar-10), and otherwise train the model from scratch (ResNet56, VGG16, and PreResNet29 on Cifar-10). For all cases, we ensure comparable baseline accuracy, and when not possible, we report the baseline used by other methods. For comparison we consider a wide range of pruning ratios, and consider validation accuracy, FLOPs, and parameter size as the

metrics. The goal is to achieve higher accuracy with lower FLOPs/parameter size.

B. HAP Results on CIFAR-10

We first start with evaluating HAP without Neural Implant, and then discuss the specific improvement of using Neural Implant. The results on CIFAR-10 for different pruning ratios and various models are presented in Figure 3. In particular, we report both the validation accuracy versus remaining parameters after pruning, as well as validation accuracy versus the FLOPs. For comparison, we also plot the performance of NN slimming [31], EigenDamage [41] and DCP [52] for different pruning ratios. For all the points that we compare, HAP achieves higher accuracy than EigenDamage, even for cases with fewer parameters/FLOPs. We generally observe that the difference between HAP and EigenDamage is more noticeable for higher pruning ratios (i.e., fewer remaining parameter). This is expected, since small amounts of pruning does not lead to significant accuracy degradation, while higher pruning ratios are more challenging. In particular, when the parameter remaining percentage is around 35% (i.e., 65% of the parameters are pruned), HAP achieves 93.2% accuracy, which is 1.24% higher than EigenDamage, with fewer FLOPs (34.0% versus 38.7% for EigenDamage). We observe a similar trend on WideResNet32, where HAP consistently outperforms EigenDamage.

We also plot the performance of DCP, which is not a second-order method, but is known to achieve good pruning accuracy. HAP achieves higher accuracy as compared to NN slimming, and comparable accuracy to DCP. As for the latter, the benefit of HAP is that we do not need to perform any greedy channel selection and the entire Hessian calculation and channel selection is performed in one pass.[1]

For PreResNet29, we also compare with NN Slimming [31], to compare with prior reported results on this model. We observe that HAP achieves up to 6% higher accuracy as compared to NN Slimming method, and slightly higher accuracy as compared to EigenDamage. It is interesting to note that HAP can keep the accuracy the same as baseline, up to pruning 70% of the parameters (corresponding to 30% remaining parameters in Figure 3).

We also present results on VGG16 and compare with other works in the literature, including GAL [30], HRank [28], and VarP [51], as reported in Table I. Here, we consistently achieve higher accuracy. In particular, HAP with 29.7% FLOPs and 10.1% parameters achieves the highest accuracy (despite using a pretrained model with lower baseline accuracy). Similarly, HAP with 20.3% FLOPs and 5.1% parameters achieves 93.37% accuracy, with less than 2× FLOPs and 3× fewer parameters as compared to HRank in the same block. For extreme pruning, HAP achieves 91.22% accuracy with only 1.6% of the parameters remaining. To the best of our knowledge, this level of aggressive pruning, while maintaining such high accuracy, has not been reported in the literature.

Table I: *Comparison between* HAP *and other pruning methods on CIFAR-10. Here, VGG16 denotes the baseline used in HRank [28], and VGG16-*HAP *denotes the baseline used by* HAP *method. As one can see,* HAP *consistently outperforms other pruning methods, even though its pruned models have fewer parameters (Param.) and FLOPs.*

Method	Acc.(%)	Param.(%)	FLOPs(%)
VGG16	93.96	100.0	100.0
VGG16-HAP	93.88	100.0	100.0
L1[27]	93.40	36.0	65.7
SSS[20]	93.02	26.2	58.4
VarP[51]	93.18	26.7	60.9
HRank[28]	93.43	17.1	46.5
GAL-0.05[30]	92.03	22.4	60.4
HRank[28]	92.34	17.9	34.7
GAL-0.1[30]	90.73	17.8	54.8
HAP	**93.66**	**10.1**	**29.7**
HRank[28]	91.23	8.0	23.5
HAP	**93.37**	**5.1**	**20.3**
HAP	**91.22**	**1.6**	**7.5**

Visualizing HAP Channel Selection: It is interesting to visualize how HAP performs channel selection using the second-order sensitivity discussed in Sec. III-B. To this end, we plot the second-order sensitivity of different channels in the sixth convolution layer of WideResNet32 in Figure 4 (blue line). For each channel, we add a binary bar chart which is shown if the channel is present only if the corresponding channel is not pruned. We can clearly see that layers with lower sensitivity (lower values in the blue line) are pruned, and vice versa. Similar results can be seen for other models (see Appendix A).

[1]We also tried to test DCP on other models but the code base is old and we were not able to use it for WideResNet32 or PreResNet29. As such we considered other pruning methods, besides EigenDamage, for comparison with those models.

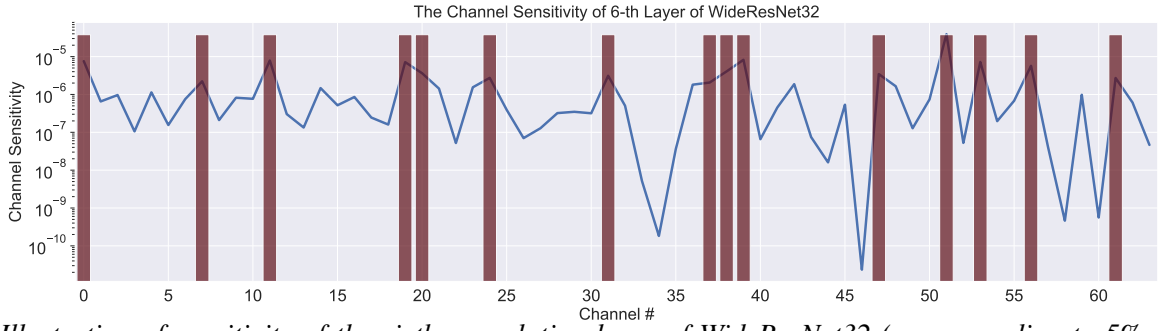

Fig. 4: *Illustration of sensitivity of the sixth convolution layer of WideResNet32 (corresponding to 5% parameter remaining in Figure 3). The x-axis denotes the channel index, and the blue line denotes the corresponding second-order sensitivity computed using Eq. 11. The red bar is added to channels that remain unpruned with the* HAP *method. These correspond to sensitive channels that have large values on the blue line. The corresponding results for ResNet56 and PreResNet29 are presented in Appendix A.*

C. Neural Implant Results on CIFAR-10

Despite HAP's competitive results as compared to prior pruning methods, it still has lower accuracy as compared to baseline. This is known problem and shortcoming of structured pruning methods. We propose to use a low rank Neural Implant to address this problem, and find it particularly helpful for moderate levels of structured pruning. In particular, for the CNNs tested in this paper we replace sensitive 3×3 spatial convolutions with a pointwise convolution. This replacement still reduces the number of parameters for the 3×3 convolution by a factor of $9\times$.

We repeated the previous experiments with this approach, and report the results in Figure 3 (blue line). We observe that HAP+IMPLANT consistently achieves better performance than HAP for both the same parameter size (first row) and the same FLOPs (second row), which also surpasses the performance of DCP [52] that has a competitive result with HAP.

For some cases, the performance of the pruned network slightly exceeds the baseline accuracy. In particular, for ResNet56, we observe up to 1.5% higher accuracy as compared to HAP, and up to 2% higher accuracy as compared to EigenDamage. We observe a similar trend for both WideResNet32/PreResNet29, where HAP+IMPLANT consistently performs better than both HAP and EigenDamage.

It should be noted that the gains from HAP+IMPLANT diminish for higher pruning ratios (around 20% remaining parameters for ResNet56, and around 30% remaining for WideResNet32/PreResNet29). This is expected, since there is a trade-off associated with adding the Neural Implant. While the implant helps reduce the information loss from completely removing sensitive channels, it does so by adding additional parameters. As such, we actually have to enforce a larger pruning ratio to meet a target model size. As the channels are sorted based on their sensitivity (from Eq. 11), this means that we have to prune the next set of more sensitive channels to satisfy the target. However, if such channels have much higher sensitivity, then that can actually degrade the performance. This is what happens for extreme pruning cases, since most of the remaining parameters will be highly sensitive; and, as such, the gains achieved by the Neural Implant will not be enough.

In addition to parameter percentage, we also compare HAP+IMPLANT results based on remaining FLOPs with other methods reported in the literature. This is shown in Table II. As one can see, with a high remaining FLOPs percentage, HAP+IMPLANT can reach 93.55% accuracy with only 0.33% degradation as compared with the corresponding pretrained baseline model. It should be noted that state-of-the-art methods such as FPGM [15] and LFPC [13] requires 6.4% more FLOPs to reach comparable performance. Moreover, when the target percentage of remaining FLOPs is small, HAP+IMPLANT only incurs 0.96% accuracy degradation as compared with 2.31% for HAP and 2.54% for HRank [28] (with comparable FLOPs and baseline accuracy).

D. HAP Results on ImageNet

We also test HAP on ImageNet using ResNet50, and report the results in Table III. We compare with several previous structured pruning methods including SSS [20], CP [16], ThiNet [32], and HRank [28]. It should be noted that the accuracy of our pretrained baseline is slightly lower than HRank, yet our HAP method still achieves higher accuracy. For instance, in all cases, HAP achieves higher accuracy with smaller

Table II: *Comparison of FLOPs and accuracy on CIFAR-10 using ResNet56 for different pruning methods. We report the baseline accuracy used in each work, as well as the corresponding final accuracy after pruning. For ease of comparison, we also report the accuracy drop (Acc. ↓) w.r.t. each baseline. As one can see,* HAP *and* HAP+IMPLANT *consistently outperform other work reported in the literature.*

Method	Baseline acc.	Final acc.	Acc. ↓	FLOPs (%)
CP[16]	92.80	91.80	1.00	50.0
AMC[14]	92.80	91.90	0.90	50.0
FPGM[15]	93.59	93.26	0.33	47.4
LFPC[13]	93.59	93.24	0.35	47.1
HAP+IMPLANT	**93.88**	**93.55**	**0.33**	**40.7**
GAL-0.8[30]	93.26	90.36	2.90	39.8
HRank[28]	93.26	90.72	2.54	25.9
HAP	93.88	91.57	2.31	**21.0**
HAP+IMPLANT	**93.88**	**92.92**	**0.96**	23.9

number of parameters as compared to all prior work reported on ResNet50. The highest difference corresponds to 34.74% remaining parameters (i.e., pruning 65.26% of parameters), where HAP has 2% higher Top-1 accuracy with 19.26% fewer parameters as compared to HRank (although for fairness our FLOPs are slightly larger). We observe a consistent trend even for high pruning ratios. For example, with 20.47% remaining parameters, HAP still has more than 2% higher accuracy as compared to HRank. We should also note that despite using second-order information, HAP is quite efficient, and the end-to-end Hessian calculations were completed three minutes on a single RTX-6000 GPU.

E. Ablation Study

We conducted several different ablation experiments to study the effectiveness of the second-order based metric in HAP. For all the experiments, we use ResNet56 on CIFAR-10.

One of the main components of HAP is the Hessian trace metric used to sort different channels to be pruned. In particular, this ordering sorts the channels from the least sensitive to most sensitive, computed based on Eq. 11. In the first ablation study, we use the reverse order of what HAP recommends, and denote this method as R-HAP. The results are shown in Table IV. It can be clearly observed that for all cases R-HAP achieves lower accuracy as compared to HAP (more than 3% for the case with 35.50% remaining parameters). In the second ablation experiment, we use a random order for pruning the layers, irrespective of their second-order sensitivity,

Table III: *Comparison between* HAP, HAP+IMPLANT, *and other state-of-the-art pruning methods on ImageNet. Here, ResNet50 is the baseline used in HRank [28]'s table, while ResNet50-*HAP *is the baseline used by* HAP.

Method	Top-1	Param.(%)	FLOPs(%)
ResNet50	76.15	100.0	100.0
ResNet50-HAP	75.62	100.0	100.0
SSS-32[20]	74.18	72.94	68.95
CP[16]	72.30	-	66.75
GAL-0.5[30]	71.95	83.14	56.97
HRank[28]	74.98	63.33	**56.23**
HAP	75.12	55.41	66.18
HAP+IMPLANT	**75.36**	**53.74**	**55.49**
GDP-0.6[29]	71.19	-	45.97
GDP-0.5[29]	69.58	-	38.39
SSS-26[20]	71.82	61.18	56.97
GAL-1[30]	69.88	57.53	38.63
GAL-0.5-joint[30]	71.82	75.73	44.99
HRank[28]	71.98	54.00	**37.90**
HAP	**74.00**	**34.74**	40.44
ThiNet-50[32]	68.42	33.96	26.89
GAL-1-joint[30]	69.31	40.04	27.14
HRank[28]	69.10	32.43	**23.96**
HAP	**71.18**	**20.47**	32.85

Table IV: *Ablation study on the sensitivity metric. R-*HAP *denotes pruning by reversely using sensitivity in* HAP. *Random is conducted by randomly allocating channel-wise sensitivity.*

Method	Acc.	Param.(%)	FLOPs(%)	Channel
R-HAP	89.77	47.48	46.98	65
Random	93.12	45.60	46.68	60
Magnitude	93.29	61.82	39.29	55
HAP	**93.38**	**41.53**	**38.74**	50
R-HAP	89.97	42.85	39.99	60
Random	92.21	36.01	33.99	50
Magnitude	92.99	56.15	34.97	50
HAP	**93.23**	**35.50**	**33.99**	45
R-HAP	88.83	**27.15**	30.61	50
Random	90.95	29.64	31.32	40
Magnitude	92.45	47.28	28.97	42.2
HAP	**92.81**	31.08	**28.85**	40
R-HAP	88.18	23.34	28.04	45
Random	90.18	25.33	25.69	30
Magnitude	91.65	38.25	22.93	35
HAP	**92.06**	**22.05**	**22.86**	31

and denote this as Random in Table IV. Similar to the previous case, the random ordering achieves consistently lower accuracy as compared to HAP. In addition, its results exhibit a larger variance.

Another important ablation study is to compare the

performance of the Hessian-based pruning with the commonly used magnitude based methods that use variants of $\|w\|_2^2/p$ (denoted as Magnitude in Table IV). To make a fair comparison, we set the FLOPs of the model after pruning to be the same for HAP and the magnitude based pruning (and slightly higher for the latter to be fair). The results are reported in Table IV. As the results show, HAP achieves the same accuracy as magnitude based pruning but with much fewer parameters (i.e., higher pruning ratio). In particular, for pruning with 22.53% of FLOPs (last row of Table IV), HAP achieves 92.06% which is almost the same as magnitude based pruning (91.65%). However, HAP achieves this accuracy with only 21.01% of the parameters remaining, as compared to 38.25%, which is quite a significant difference. This is expected, as HAP's performance was higher than the different magnitude based results reported in the literature, for both the CIFAR-10 and ImageNet tests of the previous subsections (Sec. IV-B and IV-D, respectively).

V. Conclusion

Existing structured-pruning methods often result in significant accuracy degradation for moderate pruning levels. To address this, we propose HAP, a new second-order structured-pruning method that uses the Hessian trace as the sensitivity metric for pruning a NN model. We also proposed a new Neural Implant approach that uses HAP's sensitivity metric to perform targeted replacement of sensitive neuron's with a light-weight low rank implant. The main intuition is to prune insensitive components and to use the Neural Implant for moderately sensitive components., instead of completely pruning them. We performed extensive empirical tests using multiple NN models. We compared with several prior works, including both the second-order based structured pruning method of EigenDamage, as well as several magnitude-based pruning methods. HAP consistently achieved higher accuracy with fewer parameters. Specifically, HAP achieves 94.3% accuracy ($< 0.1\%$ degradation) on PreResNet29 (CIFAR-10), with more than 70% of parameters pruned. In comparison to EigenDamage, we achieve up to 1.2% higher accuracy with fewer parameters and FLOPs. Moreover, for ResNet50 HAP achieves 75.1% top-1 accuracy (0.5% degradation) on ImageNet, after pruning almost half of the parameters. In comparison to the prior state-of-the-art of HRank, we achieve up to 2% higher accuracy with fewer parameters and FLOPs. We have open sourced our implementation available at [1].

References

[1] https://github.com/yaozhewei/hap.git, Dec. 2021.

[2] Haim Avron and Sivan Toledo. Randomized algorithms for estimating the trace of an implicit symmetric positive semi-definite matrix. *Journal of the ACM (JACM)*, 58(2):8, 2011.

[3] Zhaojun Bai, Gark Fahey, and Gene Golub. Some large-scale matrix computation problems. *Journal of Computational and Applied Mathematics*, 74(1-2):71–89, 1996.

[4] Aydin Buluc and John R Gilbert. Challenges and advances in parallel sparse matrix-matrix multiplication. In *2008 37th International Conference on Parallel Processing*, pages 503–510. IEEE, 2008.

[5] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment. *arXiv preprint arXiv:1908.09791*, 2019.

[6] Xin Dong, Shangyu Chen, and Sinno Pan. Learning to prune deep neural networks via layer-wise optimal brain surgeon. In *Advances in Neural Information Processing Systems*, pages 4857–4867, 2017.

[7] Zhen Dong, Zhewei Yao, Daiyaan Arfeen, Amir Gholami, Michael W. Mahoney, and Kurt Keutzer. HAWQ-V2: Hessian aware trace-weighted quantization of neural networks. *Advances in neural information processing systems*, 2020.

[8] Zhen Dong, Zhewei Yao, Amir Gholami, Michael W. Mahoney, and Kurt Keutzer. HAWQ: Hessian AWare Quantization of neural networks with mixed-precision. In *The IEEE International Conference on Computer Vision (ICCV)*, October 2019.

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

[10] Amir Gholami, Kiseok Kwon, Bichen Wu, Zizheng Tai, Xiangyu Yue, Peter Jin, Sicheng Zhao, and Kurt Keutzer. SqueezeNext: Hardware-aware neural network design. *Workshop paper in CVPR*, 2018.

[11] Babak Hassibi and David G Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in neural information processing systems*, pages 164–171, 1993.

[12] Babak Hassibi, David G Stork, and Gregory J Wolff. Optimal brain surgeon and general network pruning. In *IEEE international conference on neural networks*, pages 293–299. IEEE, 1993.

[13] Yang He, Yuhang Ding, Ping Liu, Linchao Zhu, Hanwang Zhang, and Yi Yang. Learning filter pruning criteria for deep convolutional neural networks acceleration. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2009–2018, 2020.

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

[15] Yang He, Ping Liu, Ziwei Wang, Zhilan Hu, and Yi Yang.

Filter pruning via geometric median for deep convolutional neural networks acceleration. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4340–4349, 2019.

[16] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1389–1397, 2017.

[17] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *Workshop paper in NIPS*, 2014.

[18] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for mobilenetv3. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1314–1324, 2019.

[19] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[20] Zehao Huang and Naiyan Wang. Data-driven sparse structure selection for deep neural networks. In *Proceedings of the European conference on computer vision (ECCV)*, pages 304–320, 2018.

[21] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research*, 18(1):6869–6898, 2017.

[22] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[23] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2704–2713, 2018.

[24] Raghuraman Krishnamoorthi. Quantizing deep convolutional networks for efficient inference: A whitepaper. *arXiv preprint arXiv:1806.08342*, 2018.

[25] Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In *Advances in neural information processing systems*, pages 598–605, 1990.

[26] Namhoon Lee, Thalaiyasingam Ajanthan, and Philip HS Torr. Snip: Single-shot network pruning based on connection sensitivity. *arXiv preprint arXiv:1810.02340*, 2018.

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

[28] Mingbao Lin, Rongrong Ji, Yan Wang, Yichen Zhang, Baochang Zhang, Yonghong Tian, and Ling Shao. Hrank: Filter pruning using high-rank feature map. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1529–1538, 2020.

[29] Shaohui Lin, Rongrong Ji, Yuchao Li, Yongjian Wu, Feiyue Huang, and Baochang Zhang. Accelerating convolutional networks via global & dynamic filter pruning. In *IJCAI*, pages 2425–2432, 2018.

[30] Shaohui Lin, Rongrong Ji, Chenqian Yan, Baochang Zhang, Liujuan Cao, Qixiang Ye, Feiyue Huang, and David Doermann. Towards optimal structured cnn pruning via generative adversarial learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2790–2799, 2019.

[31] Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2736–2744, 2017.

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

[33] Li Lyna Zhang, Yuqing Yang, Yuhang Jiang, Wenwu Zhu, and Yunxin Liu. Fast hardware-aware neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 692–693, 2020.

[34] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 116–131, 2018.

[35] M. W. Mahoney. *Randomized algorithms for matrices and data*. Foundations and Trends in Machine Learning. NOW Publishers, Boston, 2011.

[36] Asit Mishra and Debbie Marr. Apprentice: Using knowledge distillation techniques to improve low-precision network accuracy. *arXiv preprint arXiv:1711.05852*, 2017.

[37] Sejun Park, Jaeho Lee, Sangwoo Mo, and Jinwoo Shin. Lookahead: a far-sighted alternative of magnitude-based pruning. *arXiv preprint arXiv:2002.04809*, 2020.

[38] Antonio Polino, Razvan Pascanu, and Dan Alistarh. Model compression via distillation and quantization. *arXiv preprint arXiv:1802.05668*, 2018.

[39] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. MobileNetV2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018.

[40] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2820–2828, 2019.

[41] Chaoqi Wang, Roger Grosse, Sanja Fidler, and Guodong

Zhang. Eigendamage: Structured pruning in the kronecker-factored eigenbasis. *arXiv preprint arXiv:1905.05934*, 2019.

[42] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. FBNet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 10734–10742, 2019.

[43] Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. Quantized convolutional neural networks for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4820–4828, 2016.

[44] Xia Xiao, Zigeng Wang, and Sanguthevar Rajasekaran. Autoprune: Automatic network pruning by regularizing auxiliary parameters. In *Advances in Neural Information Processing Systems*, pages 13681–13691, 2019.

[45] Tien-Ju Yang, Andrew Howard, Bo Chen, Xiao Zhang, Alec Go, Mark Sandler, Vivienne Sze, and Hartwig Adam. Netadapt: Platform-aware neural network adaptation for mobile applications. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 285–300, 2018.

[46] Zhewei Yao, Amir Gholami, Kurt Keutzer, and Michael W. Mahoney. PyHessian: Neural networks through the lens of the Hessian. *arXiv preprint arXiv:1912.07145*, 2019.

[47] Zhewei Yao, Amir Gholami, Sheng Shen, Kurt Keutzer, and Michael W Mahoney. Adahessian: An adaptive second order optimizer for machine learning. *arXiv preprint arXiv:2006.00719*, 2020.

[48] Hongxu Yin, Pavlo Molchanov, Jose M Alvarez, Zhizhong Li, Arun Mallya, Derek Hoiem, Niraj K Jha, and Jan Kautz. Dreaming to distill: Data-free knowledge transfer via deepinversion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8715–8724, 2020.

[49] Ruichi Yu, Ang Li, Chun-Fu Chen, Jui-Hsin Lai, Vlad I Morariu, Xintong Han, Mingfei Gao, Ching-Yung Lin, and Larry S Davis. Nisp: Pruning networks using neuron importance score propagation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9194–9203, 2018.

[50] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6848–6856, 2018.

[51] Chenglong Zhao, Bingbing Ni, Jian Zhang, Qiwei Zhao, Wenjun Zhang, and Qi Tian. Variational convolutional neural network pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2780–2789, 2019.

[52] Zhuangwei Zhuang, Mingkui Tan, Bohan Zhuang, Jing Liu, Yong Guo, Qingyao Wu, Junzhou Huang, and Jinhui Zhu. Discrimination-aware channel pruning for deep neural networks. In *Advances in Neural Information Processing Systems*, pages 875–886, 2018.

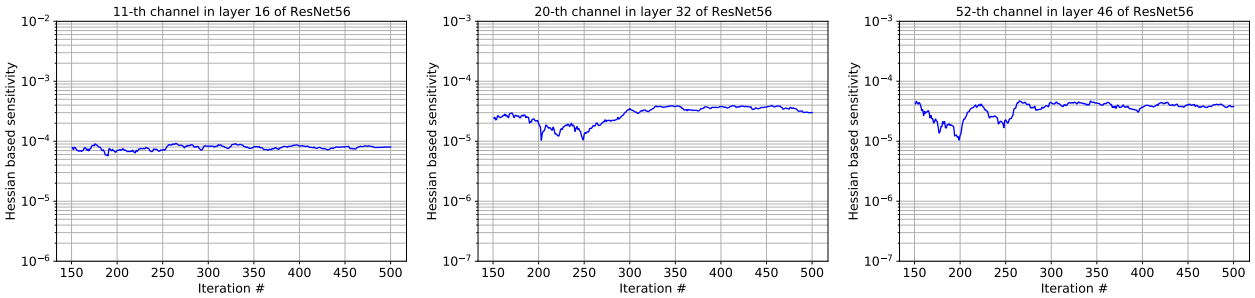

Fig. 5: *The convergence of Hessian-based sensitivity throughout the Hutchinson iterations, for different channels of ResNet56. Here, the x-axis is the Hutchinson iteration, and the y-axis is the approximation for the sensitivity corresponding to Eq. 11. As one can see, the approximation converges after about 300 iterations.*

APPENDIX

Here we present the details of the experiments performed in the paper. For model pretraining on CIFAR-10, we use the same setting as EigenDamage [41]. To finetune the pruned model for performance improvement, we use SGD with momentum 0.9 and train the compressed model for 160 epochs for CIFAR-10 and 120 epochs for ImageNet. The initial learning rate is set as 2e-2 for CIFAR-10, 1e-3 for ImageNet, and reduce by one-tenth twice at half and 3/4 of the full epoch. For CIFAR-10, we use a batch size of 64 and weight decay of 4e-4, and for ImageNet we use a batch size of 128 and weight decay of 1e-4. We also set a pruning ratio limit for each layer, following [41].

As for Neural Implant, we select a fixed neural implant ratio of 0.2, meaning that 20% of the pruned 3x3 convolution kernels are replaced by 1x1 convolution kernels.

We have open sourced our implementation available at [1].

As discussed in Section III, we compute the sensitivity based on the trace of the Hessian as presented in Eq. 11. This approximation can be computed without explicitly forming the Hessian operator, by using the Hutchinson method [2, 3]. In this approach, the application of the Hessian to a random vector is calculated through backpropogation (similar to how gradient is backpropagated) [46]. In particular, for a given random vector $v \in R^p$ with i.i.d. components, we can show:

$$Tr(H) = \mathbb{E}[v^T H v]. \tag{12}$$

See [46, 47] for details and discussion. We can directly use this identity to compute the sensitivity in Eq. 11:

$$\frac{Trace(H_{p,p})}{2p}\|w_p\|_2^2 = \frac{1}{2p}\|w_p\|_2^2 \mathbb{E}[v^T H v]. \tag{13}$$

Here, note that the norm of the parameters is a constant. One can prove that for a PSD operator, this expectation converges to the actual trace. To illustrate this empirically, we have plotted the convergence for this sensitivity metric for different channels of ResNet56. See Figure 5. As one can see, after roughly 300 iterations we get a very good approximation.

Here, we show the distribution of pruning for different channels of ResNet56, and PreResNet29. See Figure 6. As one can see, HAP only prunes insensitive channels, and keeps channels with high sensitivity (computed based on Eq. 11).

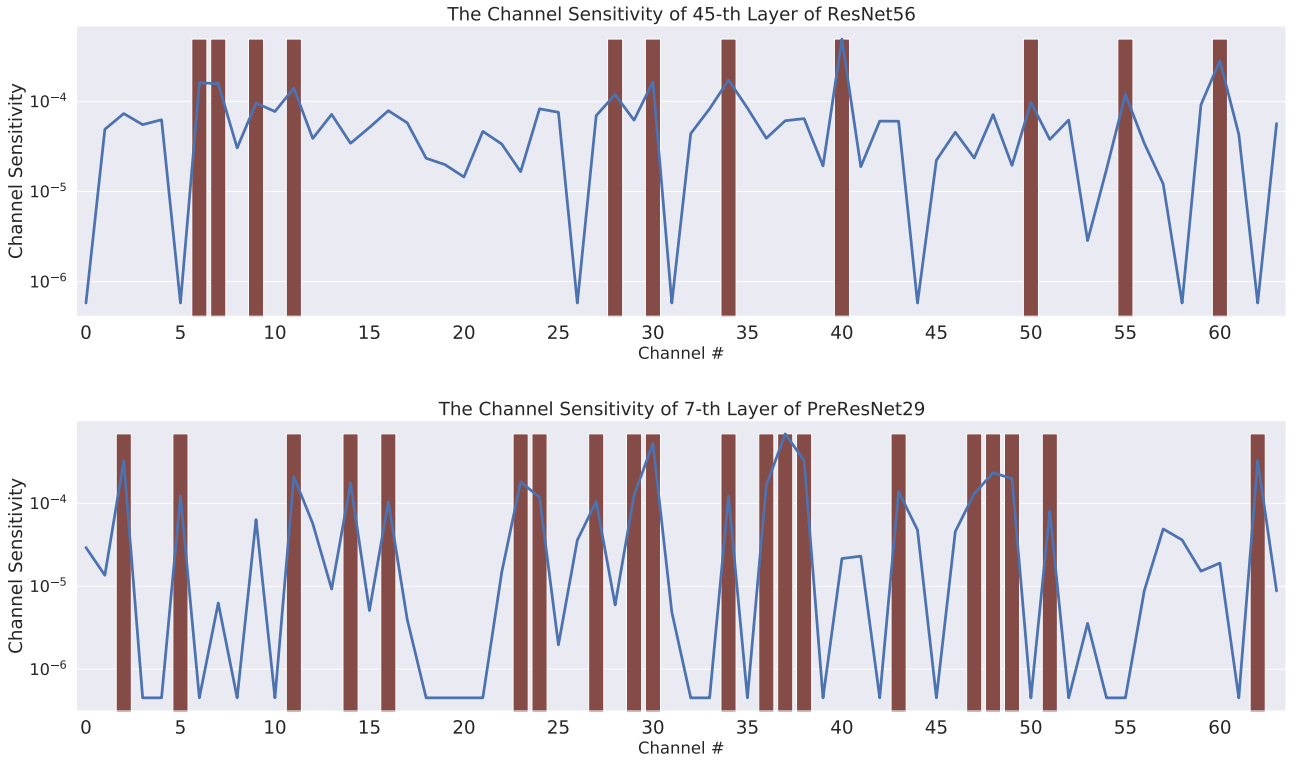

Fig. 6: *Illustration of sensitivity of the 45th convolution layer of ResNet56 and the 7th convolution layer of PreResNet29. The x-axis denotes the channel index, and the blue line denotes the corresponding second-order sensitivity computed using Eq. 11. The red bar is added to channels that remain unpruned with the* HAP *method. As one can see, these correspond to sensitive channels that have large values on the blue line. The corresponding result for WideResNet32 is shown in Figure 4 in main text.*