Acknowledgments

The UC Berkeley team acknowledges gracious support from Berkeley Deep Drive, Intel corporation, Facebook Reality Labs, Alibaba, Intel VLAB team, Google Cloud, Google TPU Research Cloud team, Amazon, and Nvidia. Amir Gholami was supported through a gracious fund from Samsung SAIT. Michael W. Mahoney would also like to acknowledge the UC Berkeley CLTC, ARO, NSF, and ONR. Our conclusions do not necessarily reflect the position or the policy of our sponsors, and no official endorsement should be inferred.

A. Solving Eq. 3

Eq. 3 can be solved by forming the corresponding Lagrangian and finding its saddle points:

$$\begin{aligned} \mathcal{L} &= \frac{1}{2}\Delta w^T H \Delta w + \lambda^T(\Delta w_p + w_p), \\ \frac{\partial \mathcal{L}}{\partial \Delta w} &= H\Delta w + \begin{pmatrix}\lambda \\ 0\end{pmatrix} = 0, \\ \begin{pmatrix} H_{p,p} & H_{p,l} \\ H_{l,p} & H_{l,l}\end{pmatrix} &\begin{pmatrix}\Delta w_p \\ \Delta w_l\end{pmatrix} + \begin{pmatrix}\lambda \\ 0\end{pmatrix} = 0, \end{aligned} \tag{7}$$

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

$$H_{p,p}\Delta w_p + H_{p,l}\Delta w_l + \lambda = 0, \tag{8}$$

$$H_{l,p}\Delta w_p + H_{l,l}\Delta w_l = 0. \tag{9}$$

Using the constraint in Eq. 3 and adding it to Eq. 9, we have:

$$\begin{aligned} -H_{l,p}w_p + H_{l,l}\Delta w_l &= 0, \\ \Delta w_l &= H_{l,l}^{-1}H_{l,p}w_p. \end{aligned} \tag{10}$$

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. 3, results in the following:

$$\frac{1}{2}\Delta w^T H \Delta w = \frac{1}{2} w_p^T (H_{p,p} - H_{p,l}H_{l,l}^{-1}H_{l,p})w_p. \tag{11}$$

B. Detailed Experimental Setup

Here we present the details of the experiments performed in the paper.

Computer Vision For model pretraining on CIFAR-10 [33], we use the same setting as EigenDamage [64]. 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 [33] and 120 epochs for ImageNet[34]. The initial learning rate is set as 2e-2 for CIFAR-10 [33], 1e-3 for ImageNet, and reduce by one-tenth twice at half and 3/4 of the full epoch. For CIFAR-10 [33], 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 [64].

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.

Natural Language Understanding In order to compute the Hessian sensitivity of attention heads, we assign all weight matrices in a single head (i.e., query, key, value, and output matrices) into a group of parameters. Although we prune the least sensitive heads *globally* across all layer, we retain at least on head in each layer as we empirically find that removing all heads from a single layer can result in a large accuracy drop. We evaluate our method on MRPC [6] and QNLI [58] of the GLUE tasks [63]. We compare our method to the gradient-based heads pruning method of [53]. For both methods, we first finetune the pretained model on downstream tasks until it achieves the best accuracy, apply heads pruning, and perform additional finetuning of 10 epochs to recover accuracy degradation. We follow the same learning rate and optimizer in RoBERTa [44] for all the experiments.

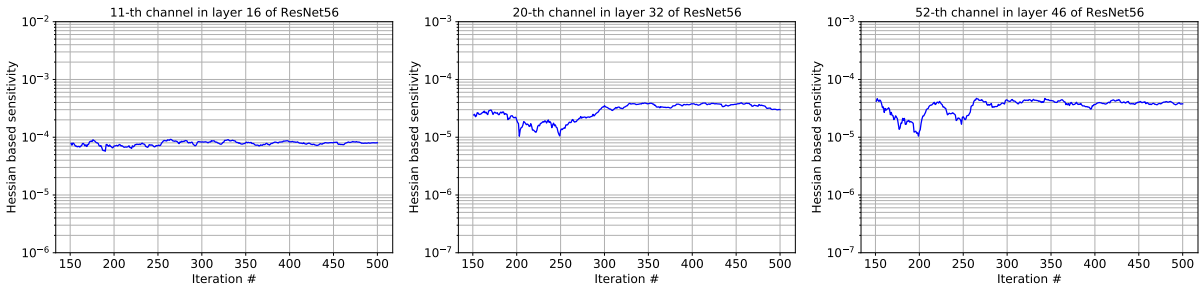

Figure 3: *The convergence of Hessian-based sensitivity throughout the Hutchinson iterations, for different channels of ResNet56. Here, the x-axis is the Hutchinson iteration, and the y-axis is the approximation for the sensitivity corresponding to Eq. 6. As one can see, the approximation converges after about 300 iterations.*

C. Sensitivity Convergence Results

As discussed in Section 3, we compute the sensitivity based on the trace of the Hessian as presented in Eq. 6. This approximation can be computed without explicitly forming the Hessian operator, by using the Hutchinson method [1, 2]. In this approach, the application of the Hessian to a random vector is calculated through backpropogation (similar to how gradient is backpropagated) [71]. 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 [71, 72] for details and discussion. We can directly use this identity to compute the sensitivity in Eq. 6:

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

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 Fig. 3. As one can see, after roughly 300 iterations we get a very good approximation.

In practice, we set Hutchinson iteration to 300 conform to the result above and we show the average time to calculate the channel-wise sensitivity on one Titan-Xp GPU for HAP in Tab. 7. Our method is fast and effective and runs around 100 seconds.

D. Limitations and Future Work

We believe it is critical for every work to clearly state its limitations, especially in this area. An important limitation is that computing the second order information adds some computational overhead. However, the overhead is actually much lower than expected as shown in Tab. 7. Another limitation is that in this work we only focused on computer vision (image classification) and natural language understanding, but it would be interesting to see how HAP would perform for more complex tasks such as object detection and machine translation. Third, in this paper, we solely work on static pruning (i.e., the model is fixed after pruning). However, for different inputs, the model can be adaptively/dynamically pruned so that we can minimize the accuracy degradation. We leave this as a future work.

E. Additional Results

Here, we show the distribution of pruning for different channels of WideResNet32, ResNet56, and PreResNet29. See Fig. 4. As one can see, HAP only prunes insensitive channels, and keeps channels with high sensitivity (computed based on Eq. 6).

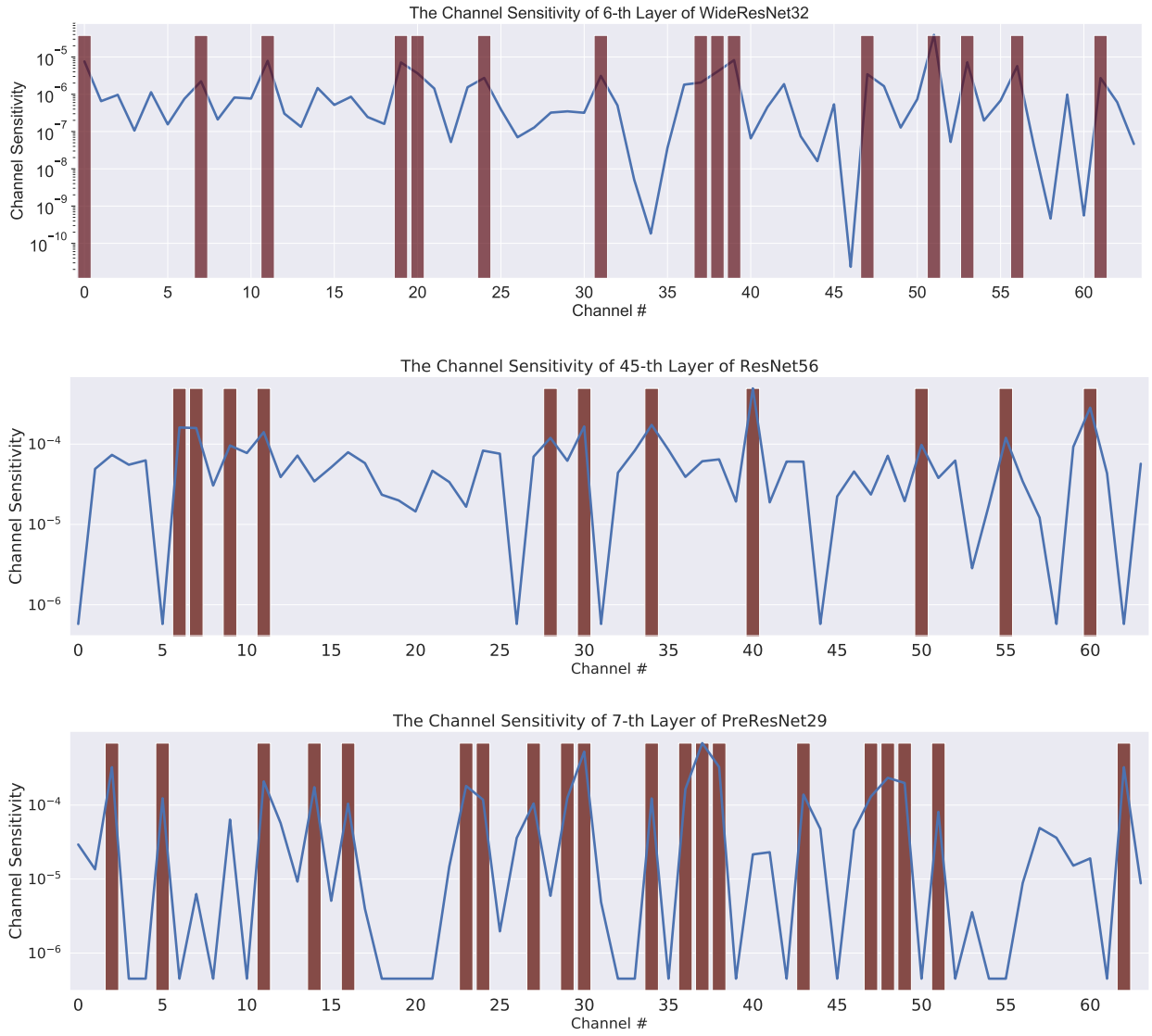

Figure 4: *Illustration of sensitivity of the t6h layer of WideResNet32, 45th convolution layer of ResNet56, and the 7th convolution layer of PreResNet29. The x-axis denotes the channel index, and the blue line denotes the corresponding second-order sensitivity computed using Eq. 6. The red bar is added 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.*

Table 7: *Calculation time for channel wise sensitivity in* HAP.

Method	WideResNet32	ResNet56	PreResNet
HAP	120s	61s	81s