HAWQ: Hessian AWare Quantization of Neural Networks with Mixed-Precision

Zhen Dong*, Zhewei Yao*, Amir Gholami*, Michael W. Mahoney, Kurt Keutzer
University of California, Berkeley
{zhendong, zheweiy, amirgh, mahoneymw, and keutzer}@berkeley.edu

Abstract

*Model size and inference speed/power have become a major challenge in the deployment of neural networks for many applications. A promising approach to address these problems is quantization. However, uniformly quantizing a model to ultra low precision leads to significant accuracy degradation. A novel solution for this is to use mixed-precision quantization, as some parts of the network may allow lower precision as compared to other layers. However, there is no systematic way to determine the precision of different layers. A brute force approach is not feasible for deep networks, as the search space for mixed-precision is exponential in the number of layers. Another challenge is a similar factorial complexity for determining block-wise fine-tuning order when quantizing the model to a target precision. Here, we introduce Hessian AWare Quantization (*HAWQ*), a novel second-order quantization method to address these problems.* HAWQ *allows for the automatic selection of the relative quantization precision of each layer, based on the layer's Hessian spectrum. Moreover,* HAWQ *provides a deterministic fine-tuning order for quantizing layers. We show the results of our method on Cifar-10 using ResNet20, and on ImageNet using Inception-V3, ResNet50 and SqueezeNext models. Comparing* HAWQ *with state-of-the-art shows that we can achieve similar/better accuracy with* $8\times$ *activation compression ratio on ResNet20, as compared to DNAS [39], and up to* 1% *higher accuracy with up to* 14% *smaller models on ResNet50 and Inception-V3, compared to recently proposed methods of RVQuant [27] and HAQ [38]. Furthermore, we show that we can quantize SqueezeNext to just 1MB model size while achieving above* 68% *top1 accuracy on ImageNet.*

1. Introduction

There has been a significant increase in the computational resources required for Neural Network (NN) training and inference. This is mainly due to larger input sizes (*e.g.*, higher image resolution) as well as larger NN models requiring more FLOPs and significantly larger memory footprint. For example, in 1998 the state-of-the-art NN was LeNet-5 [19] applied to MNIST dataset with an input image size of $1 \times 28 \times 28$. Twenty years later, a common benchmark dataset is ImageNet, with an input resolution that is $200\times$ larger than MNIST, and with NN models that have orders of magnitude higher memory footprint.

In fact, ImageNet resolution is now considered "small" for many applications such as autonomous driving where input resolutions are significantly larger (more than $40\times$ in certain cases).

This combination of larger models and higher resolution images has created a major challenge in the deployment of NNs in application environments with computationally constrained resources such as surveillance systems or ADAS systems in passenger cars. This trend is going to accelerate further in the near future.

There has been a significant effort taken by many researchers to address these issues. These could be broadly categorized as follows. (i) Finding NNs that provide the required accuracy, while remaining compact by design (*i.e.*, with small memory footprint) and requiring relatively small FLOPs. SqueezeNet [15] was an early effort here, followed by more efficient NNs such as [32, 22]. (ii) Co-designing NN architecture and hardware together. This can allow significant speed-ups as well as savings in power consumption of the hardware without losing accuracy. SqueezeNext [7] is an example work here where the neural network and associated accelerator are co-designed. (iii) Pruning redundant filters of NN layers. Seminal works here are [9, 26, 21, 23]. (iv) Using quantization (reduced precision) instead of float or double precision, which can significantly speed up inference time and reduce power consumption. (v) Applying AutoML for both hardware aware NN design as well as quantization. Notable works here are DNAS [39] and HAQ [38]. This paper exclusively focuses on quantization, but other approaches could be used in conjunction of our method to allow for further possible reduction on the model size.

Quantization needs to be performed for both NN pa-

*Equal contribution.

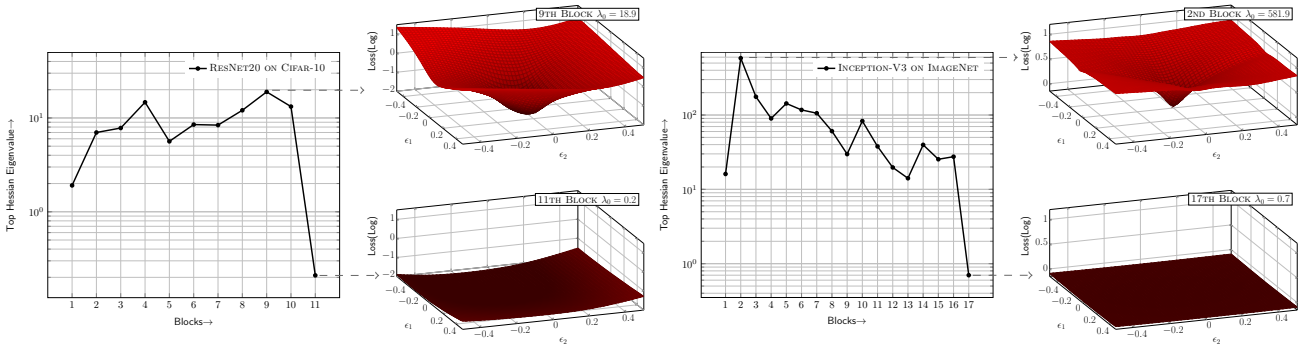

Figure 1. Top eigenvalue of each individual block of pre-trained ResNet20 on Cifar-10 (Left), and Inception-V3 on ImageNet (Right). Note that the magnitudes of eigenvalues of different blocks varies by orders of magnitude. See Figure 6 and 7 in appendix for the 3D loss landscape of other blocks.

rameters (*i.e.*, weights) as well as the activations to reduce the total memory footprint of the model during inference. However, the main challenge here is that a naïve quantization can lead to significant loss in accuracy. In particular, it is not possible to reduce the number of bits of all weights/activations of a general convolutional network to ultra low-precision without significant accuracy loss. This is because not all the layers of a convolutional network allow the same quantization level. A possible approach to address this is to use mixed-precision quantization, where higher precision is used for certain "sensitive" layers of the network, and lower precision for "non-sensitive" layers. However, the search space for finding the right precision for each layer is exponential in the number of layers. Moreover, to avoid accuracy loss we need to perform fine-tuning (*i.e.* re-training) of the model. As we will discuss below, quantizing the whole model at once and then fine-tuning is not optimal. Instead, we need to perform multi-stage quantization, where at each stage parts of the network are quantized to low-precision followed by quantization-aware fine-tuning to recover accuracy. However, the search space to determine which layers to quantize first is factorial in the number of layers. In this paper, we propose a Hessian guided approach to address these challenges. In particular, our contributions are the following.

1. The search space for choosing mixed-precision quantization is exponential in the number of layers. Thus, we present a novel, deterministic method for determining the relative quantization level of layers based on the Hessian spectrum of each layer.

2. The search space for quantization-aware fine-tuning of the model is factorial in the number of blocks/layers. Thus, we propose a Hessian based method to determine fine-tuning order for different NN blocks.

3. We perform ablation study of HAWQ, and we present novel quantization results using ResNet20 on Cifar10, as well as Inception-V3/ResNet50/SqueezeNext on ImageNet. Comparison with state-of-the-art shows that our method achieves higher precision (up to 1%), smaller model size (up to 20%), and smaller activation size (up to 8×).

The paper is organized as follows. First, in § 2, we will discuss related works on model compression. This is followed by describing our method in § 3, and our results in § 4. Finally, we present ablation study in § 5, followed by conclusions.

2. Related work

Recently, significant efforts have been spent on developing new model compression solutions to reduce the parameter size as well as computational complexity of NNs [4, 8, 11, 29, 5, 46, 35, 17, 13, 3, 45]. In [9, 23, 21], pruning is used to reduce the number of non-zero weights in NN models. This approach is very useful for models that have very large fully connected layers (such as AlexNet [18] or VGG [34]). For instance, the first fully-connected layer in VGG-16 occupies 408MB alone, which is 77.3% of total model size. Large fully-connected layers have been removed in later convolutional neural networks such as ResNet [10], and Inception family [37, 36].

Knowledge distillation introduced in [11] is another direction for compressing NNs. The main idea is to distill information from a pre-trained, large model into a smaller model. For instance, it was shown that with knowledge distillation it is possible to reduce model size by a factor of 3.6 with an accuracy of 91.61% on Cifar-10 [31].

Another fundamental approach has been to architect models which are, by design, both small and hardware-efficient. An initial effort here was SqueezeNet [15] which could achieve AlexNet level accuracy with 50× smaller footprint through network design, and additional 10× reduction through quantization [8], resulting in a NN with 500× smaller memory footprint. Other notable works here

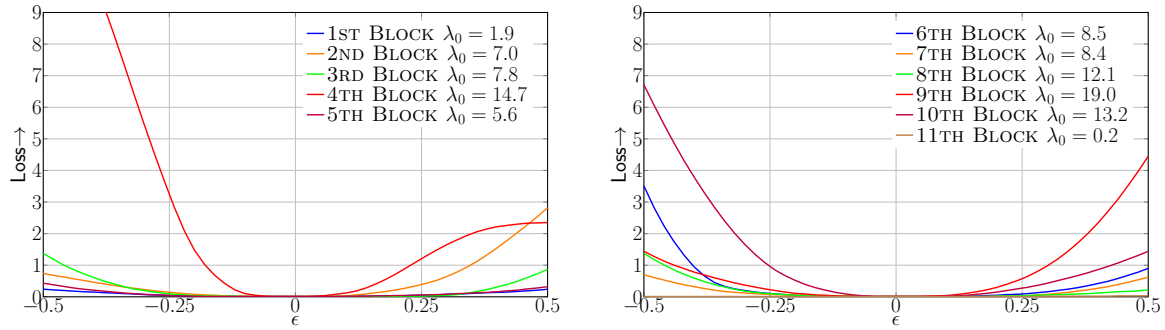

Figure 2. 1-D loss landscape for different blocks of ResNet20 on Cifar-10. The landscape is plotted by perturbing model weights along the top Hessian eigenvector of each block, with a magnitude of ϵ (*i.e.*, $\epsilon = 0$ corresponds to no perturbation).

are [13, 32, 45, 22, 3], where more accurate networks are presented. Another work here is SqueezeNext [7], where a similar approach is taken, but with co-design of both hardware architecture along with a compact NN model.

Quantization [1, 4, 29, 40, 20, 49, 46, 47, 2, 44] is another orthogonal approach for model compression, where lower bit representation are used instead of redesigning the NN. One of the major benefits of quantization is that it increases a NN's arithmetic intensity (which is the ratio of FLOPs to memory accesses). This is particularly helpful for layers that are memory bound and have low arithmetic intensity. After quantization, the volume of memory accesses reduces, which can alleviate/remove the memory bottleneck.

However, directly quantizing NNs to ultra low precision may cause significant accuracy degradation. One possibility to address this is to use Mixed-Precision Quantization [39, 48]. A second possibility, Multi-Stage Quantization, is proposed by [46, 6]. Both mixed-precision and multi-stage quantization can improve the accuracy of quantized NNs, but face an exponentially large search space. Applying existing methods often require huge computational resources or ad-hoc rules to choose precision of different layers which are problem/model specific and do not generalize. The goal of our work here is to address this challenge using second-order information.

3. Methodology

Assume that the NN is partitioned into m blocks denoted by $\{B_1,\ B_2\ \ldots,\ B_m\}$, with learnable parameters $\{W_1,\ W_2,\ \ldots,\ W_m\}$. A block can be a single/multiple layer(s) (or a single/multiple residual block(s) for the case of residual networks). For a supervised learning framework, the loss function $L(\theta)$ is:

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

where $\theta \in \mathbb{R}^d$ is the combination of $\{W_1,\ W_2,\ \ldots,\ W_m\}$, and $l(x, y, \theta)$ is the loss for a datum $(x, y) \in (X, Y)$. Here, X is the input set, Y is the corresponding label set, and $N = |X|$ is the size of the training set.

The training is performed by solving an Empirical Risk Minimization problem, to find the optimal model parameters. This process is typically performed in single precision, where both the weights and activations are stored with 32-bit precision.

After the training is finished, each of these blocks will have a specific distribution of floating point numbers for both the parameters, θ, as well as input/output activations. For quantization, we need to restrict these floating numbers to a finite set of values, defined by the following function:

$$Q(z) = q_j, \quad \text{for } z \in (t_j, t_{j+1}], \tag{2}$$

where $(t_j, t_{j+1}]$ denotes an interval in the real numbers $(j = 0,\ \ldots\ , 2^k - 1)$, k is the quantization bits, and z is either an activation or the weights. This means that all the values in the range of $(t_j, t_{j+1}]$ are mapped to q_j. In the extreme case of binary quantization ($k = 1$), $Q(z)$ is basically the sign function. For cases other than binary quantization, the choice of these intervals can be important. One popular option is to use a uniform quantization function, where the above range is equally split [47, 14]. However, it has been argued that (i) not all layers have the same distribution of floating point values, and (ii) the network can have significantly different *sensitivity* to quantization of each layer. To address the first issue, different quantization schemes such as uniformly discretizing logarithmic-domain have been proposed [25]. However, this does not completely address the sensitivity problem. A sensitive layer cannot be quantized to the same level as a non-sensitive layer.

One possible approach that can be used to measure quantization sensitivity is to use first-order information, based on the gradient vector. However, the gradient can be very misleading. This can be easily illustrated by considering a simple 1-d parabolic function of the form $y = \frac{1}{2}ax^2$ at origin (*i.e.*, $x = 0$). The gradient signal at the origin is zero,

Algorithm 1: Power Iteration for Hessian Eigenvalue Computation

Input: Block Parameter: W_i.
Compute the gradient of W_i by backpropagation, *i.e.*, $g_i = \frac{dL}{dW_i}$.
Draw a random vector v (same dimension as W_i).
Normalize v, $v = \frac{v}{\|v\|}$
for $i = 1, 2, \ldots, n$ **do** // Power Iteration
  Compute $gv = g_i^T v$ // Inner product
  Compute Hv by backpropagation, $Hv = \frac{d(gv)}{dW_i}$ // Get Hessian vector product
  Normalize and reset v, $v = \frac{Hv}{\|Hv\|}$

Algorithm 2: Hessian AWare Quantization

Input: Block-wise Hessian eigenvalues λ_i (computed from Algorithm 1), and block parameter size n_i for $i = 1, \cdots, m$.
for $i = 1, 2, \ldots, m$ **do** // Quantization Precision
  $S_i = \lambda_i / n_i$ // See Eq. 5
Order S_i in descending order and determine relative quantization precision for each block.
Compute ΔW_i based on Eq. 2.
for $i = 1, 2, \ldots, m$ **do** // Fine-Tuning Order
  $\Omega_i = \lambda_i \|\Delta W_i\|^2$ // See Eq. 6
Order Ω_i in descending order and perform block-wise fine-tuning

irrespective of the value of a. However, this does not mean that the function is not sensitive to perturbation in x. We can get a better metrics for sensitivity by using second-order information, based on the Hessian matrix. This clearly shows that higher values of a result in more sensitivity to input perturbations.

For the case of high dimensions, the second order information is stored in the Hessian matrix, of size $n_i \times n_i$ for each block. For this case, we can compute the eigenvalues of the Hessian to measure sensitivity, as described next.

3.1. Second-Order Information

We compute the eigenvalues of the Hessian (*i.e.*, the second-order operator) of each block in the network. Note that it is not possible to explicitly form the Hessian since the size of a block (denoted by n_i for ith block) can be quite large. However, it is possible to compute the Hessian eigenvalues without explicitly forming it, using a matrix-free power iteration algorithm [43, 24, 42]. This method requires computation of the so-called Hessian *matvec*, which is the result of multiplication of the Hessian matrix with a given (possibly random) vector v. To illustrate how this can be done for a deep network, let us first denote g_i as the gradient of loss L with respect to the i^{th} block parameters,

$$g_i = \frac{\partial L}{\partial W_i}. \tag{3}$$

For a random vector v (which has the same dimension as g_i), we have:

$$\frac{\partial (g_i^T v)}{\partial W_i} = \frac{\partial g_i^T}{\partial W_i} v + g_i^T \frac{\partial v}{\partial W_i} = \frac{\partial g_i^T}{\partial W_i} v = H_i v, \tag{4}$$

where H_i is the Hessian matrix of L with respect to W_i. We can then use power-iteration method to compute the top eigenvalue of H_i, as shown in Algorithm 1. Intuitively the algorithm requires multiple evaluations of the Hessian matvec, which can be computed using Eq. 4.

It is well known, based on the theory of Minimum Description Length (MDL), that fewer bits are required to specify a flat region up to a given threshold, and vice versa for a region with sharp curvature [30, 12]. The intuition for this is that the noise created by imprecise location of a flat region is not magnified for a flat region, making it more amenable to aggressive quantization. The opposite is true for sharp regions, in that even small round off errors may be amplified. Therefore, it is expected that layers with higher Hessian spectrum (*i.e.*, larger eigenvalues) are more sensitive to quantization. The distribution of these eigenvalues for different blocks are shown in Figure 1 for ResNet20 on CIFAR-10 and Inception-V3 on ImageNet. As one can see, different blocks exhibit orders of magnitude difference in the Hessian spectrum. For instance, ResNet20 is an order of magnitude more sensitive to perturbations to its 9^{th} block, than its last block.

To further illustrate this, we provide 1D visualizations of the loss landscape as well. To this end, we first compute the Hessian eigenvector of each block, and we perturb each block individually along the eigenvector and compute how the loss changes. This is illustrated in Figure 2 and 3 for ResNet20 (on Cifar-10) and Inception-V3 (on ImageNet), respectively. It can be clearly seen that blocks with larger Hessian eigenvalue (*i.e.*, sharper curvature) exhibit larger fluctuations in the loss, as compared to those with smaller Hessian eigenvalue (*i.e.*, flatter curvature). A corresponding 3D plot is also shown in Figure 1, where instead of just considering the top eigenvector, we also compute the second top eigenvector and visualize the loss by perturbing the weights along these two directions. These surface plots are computed for the 9^{th} and last blocks of ResNet20, as well as 2nd and last blocks of Inception-V3 (the loss landscape for other blocks is shown in Figure 6 and Figure 7 in the Appendix).

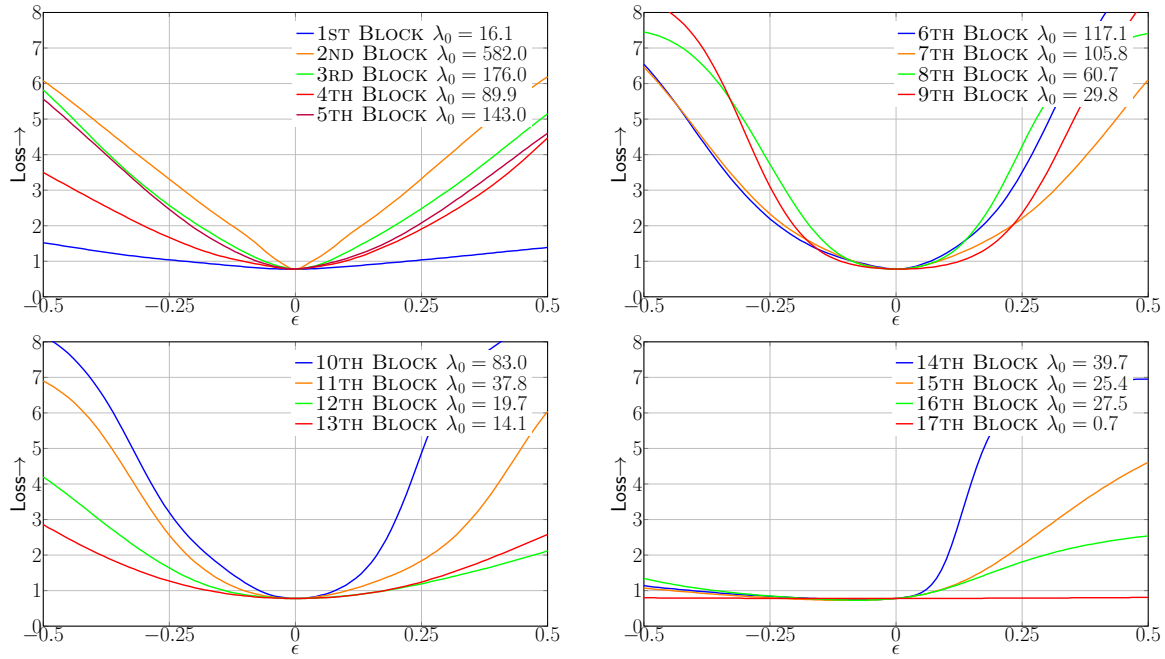

Figure 3. 1-D loss landscape of all blocks of Inception-V3 on ImageNet along the first dominant eigenvector of the Hessian. Here ϵ is the scalar that perturbs the parameters of the corresponding block along the first dominant eigenvectors.

3.2. Algorithm

We approximate the Hessian as a block diagonal matrix, scaled by its top eigenvalue λ as $\{H_i \approx \lambda_i I\}_{i=1}^{m}$, where m is the number of blocks in the network. Based on the MDL theory, layers with large λ cannot be quantized to ultra low precision without significant perturbation to the model. Thus we can use the Hessian spectrum of each block to sort the different blocks and perform less aggressive quantization to layers with large Hessian spectrum. However, some of these blocks may contain very large number of parameters, and using higher bits here would lead to large memory footprint of the quantized network. Therefore, as a compromise, we weight the Hessian spectrum with the block's memory footprint and use the following metric for sorting the blocks:

$$S_i = \lambda_i / n_i, \tag{5}$$

where λ_i is the top eigenvalue of H_i. Based on this sorting, layers that have large number of parameters and have small eigenvalue would be quantized to lower bits, and vice versa. That is, after S_i is computed, we sort S_i in descending order and use it as a metric to determine the quantization precision.[1]

Quantization-aware re-training of the neural network is necessary to recover performance which can sharply drop due to ultra-low precision quantization. A straightforward way to do this is to re-train (hereafter referred to as fine-tune) the whole quantized network at once. However, as we will discuss in §4, this can lead to sub-optimal results. A better strategy is to perform multi-stage fine-tuning. However, the order in multi-stage tuning is important and different ordering could lead to very different accuracies.

We sort different blocks for fine-tuning based on the following metric:

$$\Omega_i = \lambda_i \|Q(W_i) - W_i\|_2^2, \tag{6}$$

where i refers to i^{th} block, λ_i is the Hessian eigenvalue, and $\|Q(W_i) - W_i\|_2$ is the L_2 norm of quantization perturbation. The intuition here is to first fine-tune layers that have high curvature as well as large number of parameters which cause more perturbations after quantization. Note that the latter metric depends on the bits used for quantization and thus is not a fixed metric. (See Table 5 in the Appendix, where we show how this metric changes for different quantization precision.) The motivation for choosing this order is that fine-tuning blocks with large Ω_i can significantly affect other blocks, thus making prior fine-tuning of layers with small Ω_i futile.

4. Results

In this section, we first present our quantization results for ResNet20 on Cifar-10, and then we present our results for Inception-V3, ResNet50, and SqueezeNext quantization on ImageNet. See the Appendix for details regarding the training procedure and hyper-parameters used.

[1]Note that, as mentioned in the limitations section, S_i does not give us the exact bit precision but a relative ordering for the bits of different blocks.

Table 1. Quantization results of ResNet20 on Cifar-10. We abbreviate quantization bits used for weights as "w-bits," activations as "a-bits," testing accuracy as "Acc," and compression ratio of weights/activations as "W-Comp/A-Comp." Furthermore, we show results without using Hessian information ("Direct"), as well as other state-of-the-art methods [47, 2, 44]. In particular, we compare with the recent proposed DNAS approach of [39]. Our method achieves similar testing performance with significantly higher compression ratio (especially in activations). Here "MP" refers to mixed-precision quantization, and the lowest bits used for weights and activations are reported. Also note that [47, 2, 44] use 8-bit for first and last layers. The exact per-layer configuration for mixed-precision quantized ResNet20 is presented in appendix.

Quantization	w-bits	a-bits	Acc	W-Comp	A-Comp
Baseline	32	32	92.37	1.00×	1.00×
Dorefa [47]	2	2	88.20	16.00×	16.00×
Dorefa [47]	3	3	89.90	10.67×	10.67×
PACT [2]	2	2	89.70	16.00×	16.00×
PACT [2]	3	3	91.10	10.67×	10.67×
LQ-Nets [44]	2	2	90.20	16.00×	16.00×
LQ-Nets [44]	3	3	91.60	10.67×	10.67×
LQ-Nets [44]	2	32	91.80	16.00×	1.00×
LQ-Nets [44]	3	32	92.00	10.67×	1.00×
DNAS [39]	1 MP	32	92.00	16.60×	1.00×
DNAS [39]	1 MP	32	**92.72**	11.60×	1.00×
Direct	2 MP	4	90.34	16.00×	8.00×
HAWQ	2 MP	4	**92.22**	13.11×	8.00×

Cifar-10 After computing the eigenvalues of block Hessian (shown in Figure 1), we compute the weighted sensitivity metric of Eq. 5, along with Ω_i based on Eq. 6. We then perform the quantization based on HAWQ algorithm. Results are shown in Table 1.

For comparison, we test the quantization performance without using the Hessian information, which we refer to as "Direct" method, as well as other methods in the literature including Dorefa [47], PACT [2], LQ-Net [44], and DNAS [39], as shown in Table 1.

For methods that use Mixed-Precision (MP) quantization, the lowest bits used for weights ("w-bits"), and activations ("a-bits") are reported.

The Direct method achieves good compression, but it results in 2.03% accuracy drop, as shown in Table 1.Furthermore, comparison with other state-of-the-art shows a similar trend. There have been several methods proposed in the literature to address this reduction, with the latest method introduced in [44], where a learnable quantization method is used. As one can see, LQ-Nets results in 0.77% accuracy degradation with $10.67\times$ compression ratio, whereas HAWQ has only 0.15% accuracy drop with $13.11\times$ compression. Moreover, HAWQ achieves similar accuracy as compared to DNAS [39] but with $8\times$ higher compression ratio for activations.

ImageNet Here, we test the HAWQ method for quantizing Inception-V3 [37] on ImageNet. Inception-V3 is appealing for efficient hardware implementation, as it does not use any residual connections. Such non-linear structures create dependencies that may be very difficult to optimize for fast inference [41]. As before, we first compute the block Hessian eigenvalues, which are reported in Figure 1, and then compute the corresponding weighted sensitivity metric. We also plot the 1D loss landscape of all Inception-V3 blocks in Figure 3.

We report the quantization results in Table 2, where as before we compare with a direct quantization, as well as recently proposed "Integer-Only" [16], and RVQuant methods [27]. Direct quantization of Inception-V3 (*i.e.*, without use of second-order information), results in 7.69% accuracy degradation. Using the approach proposed in [16] results in more than 2% accuracy drop, even though it uses higher bit precision. However, HAWQ results in an accuracy gap of 2% with a compression ratio of $12.04\times$, both of which are better than previous work [16, 27].[2]

We also compare with Deep Compression [8] and the AutoML based method of HAQ, which has been recently introduced [38]. We compare our HAWQ results with their ResNet50 quantization results, as shown in Table 3. HAWQ achieves higher top-1 accuracy of 75.48% with a model size of 7.96MB, whereas the AutoML based HAQ method has a top-1 of 75.30% even with 16% larger model size of 9.22MB.

Furthermore, we apply HAWQ to quantize SqueezeNext [7] on ImageNet. We choose the wider SqueezeNext model which has a baseline accuracy of 69.38% with 2.5 million parameters (10.1MB in single precision). We are able to quantize this model to uniform 8-bit precision, with just 0.04% top-1 accuracy drop. Direct quantization of SqueezeNext (*i.e.*, without use of second-order information), results in 3.98% accuracy degradation. HAWQ results in an unprecedented 1MB model size, with only 1.36% top-1 accuracy drop. The significance of this result is that it allows deployment of the whole model on-chip or on hardwares with very limited memory and power constraints.

5. Ablation Study

Here we discuss the ablation study for the HAWQ. The HAWQ method has two main steps: (i) relative precision order for different blocks using second-order information, and (ii) relative order for fine-tuning these blocks. Below we discuss the ablation study for each step separately.

[2]We should emphasize here that the work of [16] uses integer arithmetic, and it is not completely fair to compare their results with ours.

Table 2. Quantization results of Inception-V3 on ImageNet. We abbreviate quantization bits used for weights as "w-bits," activations as "a-bits," top-1 testing accuracy as "Top-1," and weight compression ratio as "W-Comp." Furthermore, we compare HAWQ with direct quantization method without using Hessian ("Direct") and Integer-Only method [16]. Here "MP" refers to mixed-precision quantization. We report the exact per-layer configuration for mixed-precision quantization in appendix. Compared to [16, 27], we achieve higher compression ratio with higher testing accuracy.

Method	w-bits	a-bits	Top-1	W-Comp	Size(MB)
Baseline	32	32	77.45	1.00×	91.2
Integer-Only [16]	8	8	75.40	4.00×	22.8
Integer-Only [16]	7	7	75.00	4.57×	20.0
RVQuant [27]	3 MP	3 MP	74.14	10.67×	8.55
Direct	2 MP	4 MP	69.76	15.88×	5.74
HAWQ	2 MP	4 MP	**75.52**	12.04×	**7.57**

Table 3. Quantization results of ResNet50 on ImageNet. We show results of state-of-the-art methods [47, 2, 44, 8]. In particular, we also compare with the recent proposed AutoML approach of [38]. We achieve higher compression ratio with higher testing accuracy compared to [38]. Also note that [47, 2, 44] use 8-bit for first and last layers.

Method	w-bits	a-bits	Top-1	W-Comp	Size(MB)
Baseline	32	32	77.39	1.00×	97.8
Dorefa [47]	2	2	67.10	16.00×	6.11
Dorefa [47]	3	3	69.90	10.67×	9.17
PACT [2]	2	2	72.20	16.00×	6.11
PACT [2]	3	3	75.30	10.67×	9.17
LQ-Nets [44]	3	3	74.20	10.67×	9.17
Deep Comp. [8]	3	MP	75.10	10.41×	9.36
HAQ [38]	MP	MP	75.30	10.57×	9.22
HAWQ	2 MP	4 MP	**75.48**	12.28×	**7.96**

Table 4. Quantization results of SqueezeNext on ImageNet. We show a case where HAWQ is used to achieved uniform quantization to 8 bits for both weights and activations, with an accuracy similar to ResNet18. We also show a case with mixed precision, where we compress SqueezeNext to a model with just 1MB size with only 1.36% accuracy degradataion. Furthermore, we compare HAWQ with direct quantization method without using Hessian ("Direct").

Method	w-bits	a-bits	Top-1	W-Comp	Size(MB)
Baseline	32	32	69.38	1.00×	10.1
ResNet18 [28]	32	32	69.76	1.00×	44.7
HAWQ	8	8	**69.34**	4.00×	2.53
Direct	3 MP	8	65.39	9.04×	1.12
HAWQ	3 MP	8	**68.02**	9.25×	**1.09**

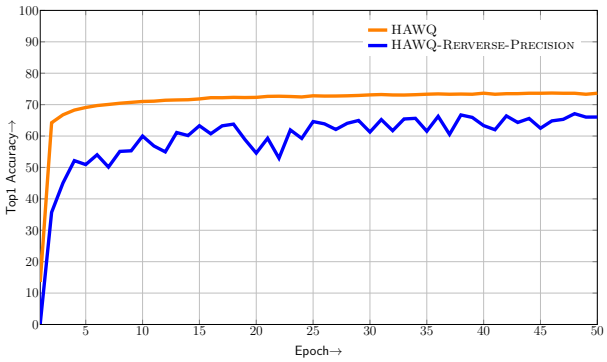

Figure 4. Accuracy recovery from Hessian aware mixed-precision quantization versus HAWQ-Reverse-Precision quantization. Here, we show top-1 accuracy of quantized Inception-V3 on ImageNet. HAWQ-Reverse-Precision achieves 66.72% (compression-ratio 7.2) top-1 accuracy, while our HAWQ method achieves 74.36% (compression-ratio 12.0) top-1 accuracy (7.64% better) with a higher convergence speed (30 epochs v.s. 50 epochs of HAWQ-Reverse-Precision).

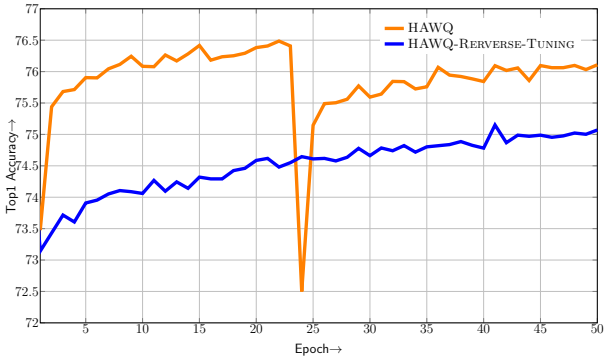

Figure 5. Effectiveness of Hessian aware block-wise fine-tuning. Here, HAWQ shows the quantization process based on the descending order of Ω_i for Inception-V3 with Hessian aware quantization order. HAWQ-Reverse-Tuning shows the quantization process of Inception-V3 with a reverse order. Note that HAWQ finishes the fine-tuning of this block in just 25 epochs and switches to fine-tuning another block, whereas HAWQ-Reverse-Tuning takes 50 epochs for this block, before converging to sub-optimal top-1.

5.1. Hessian AWare Mixed Precision Quantization

We first discuss the ablation study for step (i), where the quantization precision is chosen based on Eq. 5. As discussed above, blocks with higher values of S_i are assigned higher quantization precision, and vice versa for layers with relatively lower values of S_i. For the ablation study we reverse this order and avoid performing the block-wise fine-tuning of step (ii) so we can isolate step (i). Instead of the fine-tuning phase, we re-train the whole network at once after the quantization is performed. The results are shown in Figure 4, where we perform 50 epochs of fine-tuning using Inception-V3 on ImageNet. As one can see, HAWQ re-

sults in significantly better accuracy (74.26% as compared to 66.72%) than the reverse method (labeled as "HAWQ-Reverse-Precision"). This is despite the fact that the latter approach only has a compression ratio of $7.2\times$, whereas HAWQ has a compression ratio of $12.0\times$.

Another interesting observation is that the convergence speed of the Hessian aware approach is significantly faster than the reverse method. Here, HAWQ converges in about 30 epochs, whereas the HAWQ-Reverse-Precision case takes 50 epochs before converging to a sub-optimal value (Figure 4).

5.2. Block-Wise Fine-Tuning

Here we perform the ablation study for the Hessian based fine-tuning part of HAWQ. The block-wise fine tuning is performed based on Ω_i (Eq. 6) of each block. The blocks are fine-tuned based on the descending order of Ω_i. Similar to the above, we compare the quantization performance when a reverse ordering is used (*i.e.*, we use the ascending order of Ω_i and refer to this as "HAWQ-Reverse-Tuning").

We test this ablation study using Inception-V3 on ImageNet, as shown in Figure 5. As one can see, the fine-tuning for HAWQ method quickly converges in just 25 epochs, allowing it to switch to fine-tuning the next block. However, "HAWQ-Reverse-Tuning" takes more than 50 epochs to converge for this block.

6. Conclusions

We have introduced HAWQ, a new quantization method for neural network training. Our method is based on exploiting second-order (Hessian) information to systematically select both quantization precision as well as the order for block-wise fine-tuning. We performed an ablation study for both the relative quantization bit-order for different blocks, as well as the fine-tuning order. We showed that HAWQ can achieve good testing performance with high compression-ratio, as compared to state-of-the-art. In particular, we showed results for ResNet20 on Cifar-10, where we can achieve similar testing performance as [39], but with $8\times$ higher compression ratio for activations. We also showed results for Inception-V3 on ImageNet, for which we showed ultra low precision quantization results with 2-bit for weights and 4-bit for activations, with only 1.93% accuracy drop. For ResNet50 model, our approach results in higher accuracy of 75.48% with smaller model size of 7.96MB, as compared to HAQ method with top-1 of 75.30% and 9.22MB [38]. Furthermore, our method applied to SqueezeNext can result in an unprecedented 1MB model size with 68.02% top-1 accuracy on ImageNet.

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, we only need to compute the top eigenvalue of the Hessian, which can be found using the matrix-free method presented in Algorithm 1. (The total computational overhead is equivalent to about 20 gradient back-propogations to compute top Hessian eigenvalue of each block). Another limitation is that in this work we solely focused on image classification, but it would be interesting to see how HAWQ would perform for more complex tasks such as segmentation, object detection, or natural language processing. Furthermore, one has to consider that implementation of a NN with mixed-precision inference for embedded processors is not as straightforward as the case with uniform quantization precision. Practical solutions have been proposed in recent works [33]. Another limitation is that we can only determine the relative ordering for quantization precision, and not the absolute value of the bits. However, the search space for this is significantly smaller than the original exponential complexity. Finally, even though we showed benefits of HAWQ as compared to DNAS [39] or HAQ [38], it may be possible to combine these methods for more efficient AutoML search. We leave this as part of future work.

Acknowledgments

This work was supported by a gracious fund from Intel corporation, Berkeley Deep Drive (BDD), and Berkeley AI Research (BAIR) sponsors. We would like to thank the Intel VLAB team for providing us with access to their computing cluster. We also gratefully acknowledge the support of NVIDIA Corporation for their donation of two Titan Xp GPU used for this research. We would also like to acknowledge ARO, DARPA, NSF, and ONR for providing partial support of this work.

References

[1] Krste Asanovic and Nelson Morgan. *Experimental determination of precision requirements for back-propagation training of artificial neural networks*. International Computer Science Institute, 1991.

[2] Jungwook Choi, Zhuo Wang, Swagath Venkataramani, Pierce I-Jen Chuang, Vijayalakshmi Srinivasan, and Kailash Gopalakrishnan. Pact: Parameterized clipping activation for quantized neural networks. *arXiv preprint arXiv:1805.06085*, 2018.

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

[4] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in neural information processing systems*, pages 3123–3131, 2015.

[5] Emily L Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in neural information processing systems*, pages 1269–1277, 2014.

[6] Yinpeng Dong, Renkun Ni, Jianguo Li, Yurong Chen, Jun Zhu, and Hang Su. Learning accurate low-bit deep neural networks with stochastic quantization. *British Machine Vision Conference*, 2017.

[7] 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.

[8] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *International Conference on Learning Representations*, 2016.

[9] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pages 1135–1143, 2015.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

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

[12] Sepp Hochreiter and Jürgen Schmidhuber. Flat minima. *Neural Computation*, 9(1):1–42, 1997.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

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

[18] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[19] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[20] Fengfu Li, Bo Zhang, and Bin Liu. Ternary weight networks. *arXiv preprint arXiv:1605.04711*, 2016.

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

[22] 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.

[23] Huizi Mao, Song Han, Jeff Pool, Wenshuo Li, Xingyu Liu, Yu Wang, and William J Dally. Exploring the regularity of sparse structure in convolutional neural networks. *Workshop paper in CVPR*, 2017.

[24] James Martens. Deep learning via hessian-free optimization. In *ICML*, volume 27, pages 735–742, 2010.

[25] Daisuke Miyashita, Edward H Lee, and Boris Murmann. Convolutional neural networks using logarithmic data representation. *arXiv preprint arXiv:1603.01025*.

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

[27] Eunhyeok Park, Sungjoo Yoo, and Peter Vajda. Value-aware quantization for training and inference of neural networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 580–595, 2018.

[28] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[29] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pages 525–542. Springer, 2016.

[30] Jorma Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465–471, 1978.

[31] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550*, 2014.

[32] 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.

[33] Hardik Sharma, Jongse Park, Naveen Suda, Liangzhen Lai, Benson Chau, Vikas Chandra, and Hadi Esmaeilzadeh. Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural networks. In *Proceedings of the 45th Annual International Symposium on Computer Architecture*, pages 764–775. IEEE Press, 2018.

[34] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

[35] Sanghyun Son, Seungjun Nah, and Kyoung Mu Lee. Clustering convolutional kernels to compress deep neural networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 216–232, 2018.

[36] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[37] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[38] Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. HAQ: Hardware-aware automated quantization. *In Proceedings of the IEEE conference on computer vision and pattern recognition*, 2019.

[39] Bichen Wu, Yanghan Wang, Peizhao Zhang, Yuandong Tian, Peter Vajda, and Kurt Keutzer. Mixed precision quantization of convnets via differentiable neural architecture search. *arXiv preprint arXiv:1812.00090*, 2018.

[40] 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.

[41] Yifan Yang, Qijing Huang, Bichen Wu, Tianjun Zhang, Liang Ma, Giulio Gambardella, Michaela Blott, Luciano Lavagno, Kees Vissers, John Wawrzynek, et al. Synetgy: Algorithm-hardware co-design for convnet accelerators on embedded fpgas. In *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 23–32. ACM, 2019.

[42] Zhewei Yao, Amir Gholami, Kurt Keutzer, and Michael Mahoney. Large batch size training of neural networks with adversarial training and second-order information. *arXiv preprint arXiv:1810.01021*, 2018.

[43] Zhewei Yao, Amir Gholami, Qi Lei, Kurt Keutzer, and Michael W Mahoney. Hessian-based analysis of large batch training and robustness to adversaries. *Advances in Neural Information Processing Systems*, 2018.

[44] Dongqing Zhang, Jiaolong Yang, Dongqiangzi Ye, and Gang Hua. LQ-Nets: Learned quantization for highly accurate and compact deep neural networks. In *The European Conference on Computer Vision (ECCV)*, September 2018.

[45] 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.

[46] Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen. Incremental network quantization: Towards lossless cnns with low-precision weights. *International Conference on Learning Representations*, 2017.

[47] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016.

[48] Yiren Zhou, Seyed-Mohsen Moosavi-Dezfooli, Ngai-Man Cheung, and Pascal Frossard. Adaptive quantization for deep neural network. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[49] Chenzhuo Zhu, Song Han, Huizi Mao, and William J Dally. Trained ternary quantization. *International Conference on Learning Representations (ICLR)*, 2017.

7. Appendix

Here, we provide additional experimental results as well as quantization details for the neural networks that we tested.

- In § 7.1 we discuss the fine-tuning details.
- In § 7.2 we present extra results for 3D plots for loss landscape of different blocks of ResNet20 and Inception-V3 as well as exemplary results showing distribution of Ω_i in Eq. 6.
- In § 7.3 we show the exact bit-precision used for different blocks of ResNet20 on Cifar-10 as well as Inception-V3 on ImageNet.

7.1. Fine-tuning details

The results were tested on two classification datasets of Cifar-10 and ImageNet:

Cifar-10 This is a classification dataset with 10 classes consisting of 50,000 training images and 10,000 test images of size $3 \times 32 \times 32$. We used pre-trained ResNet20 model and performed quantization on this model in PyTorch framework. We follow the same learning rate policy as the baseline (*i.e.*, decaying learning rate from 1e-1 to 1e-4).

ImageNet This is a classification problem with 1000 classes consisting of more than 1.2 million training images and 50,000 validation images of size $3 \times 224 \times 224$ on SqueezeNext and ResNet50 , and $3 \times 299 \times 299$ on Inception-V3. (i) We used pre-trained Inception-V3 model and used a fixed learning rate of 2e-4 for fine-tuning of each block. (ii) We used pre-trained ResNet50 model and used a fixed learning rate of 1e-4 for fine-tuning of each block. (iii) We used pre-trained SqueezeNext and used a fixed learning rate of 1e-4 for fine-tuning of each block. All experiments were performed on PyTorch framework. As for data augmentation, we used standard random crop, resizing and horizontal flip in all experiments.

7.2. Extra results

In Table 5, we show how Ω_i changes as a function of quantization precision. In Figure 6, we plot the rest surface visualization of ResNet20 on Cifar-10. And in Figure 7, we plot the rest surface visualization of Inception-V3 on ImageNet.

Table 5. Here we show how Ω_i changes as a function of target weight bit precision. Results are computed for ResNet20 on Cifar-10.

Block \ Precision	8-bit	6-bit	4-bit	3-bit	2-bit
Block 3	0.03	0.52	9.25	41.9	191
Block 5	0.05	0.81	14.0	65.1	309
Block 8	0.29	4.83	84.8	392	2056

7.3. Mixed-precision details

In this section, we give the details about how we separate blocks and details about weight/activation precision of each individual block. We show the exact bit-precision used for different blocks of ResNet20 (Table 6) on Cifar-10 as well as Inception-V3 (Table 7) on ImageNet.

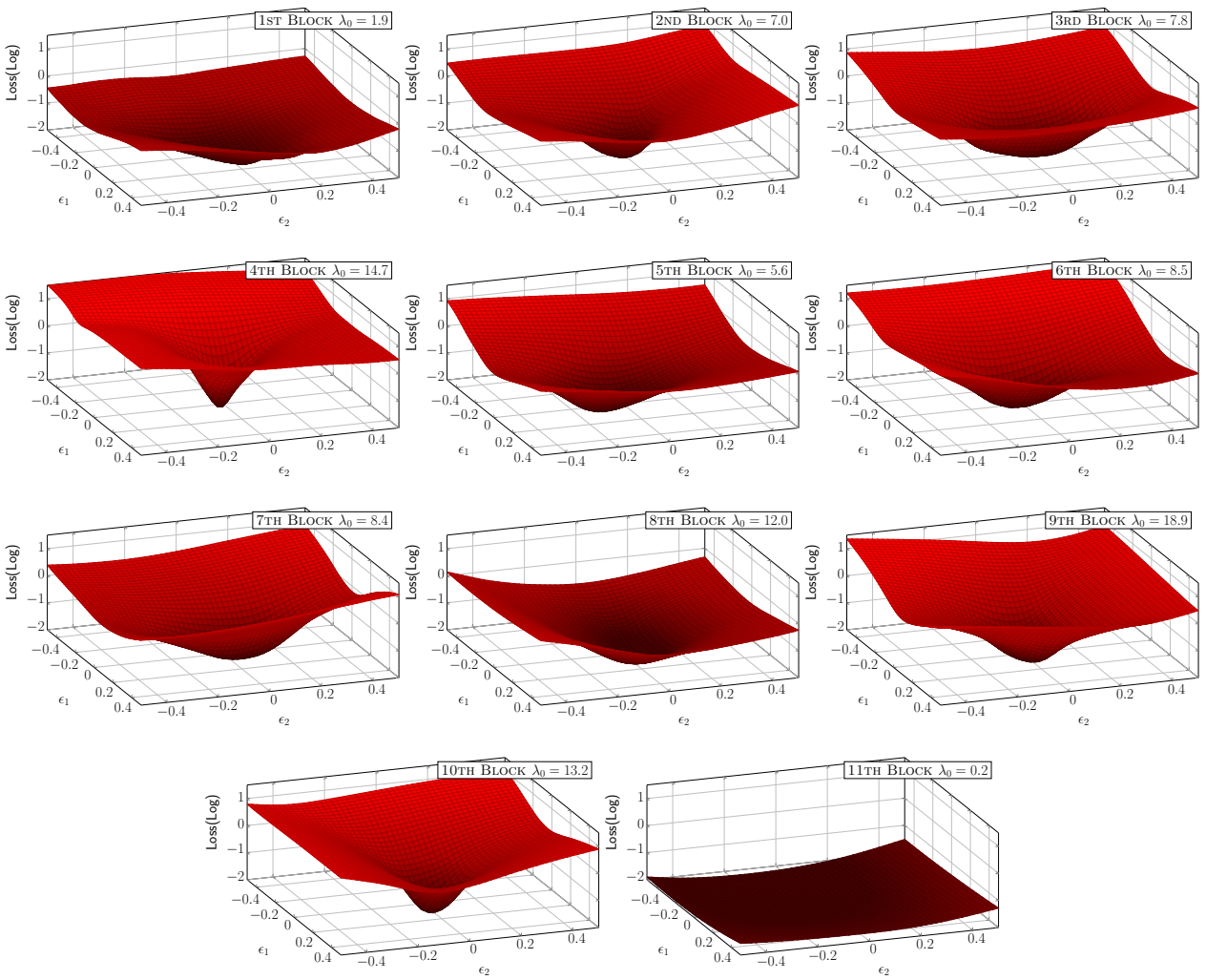

Figure 6. 3-D loss landscape of all blocks of ResNet20 on Cifar-10 along the first two dominant eigenvectors of the Hessian. Here ϵ_1, ϵ_2 are scalars that perturb the parameters of the corresponding block along the first and second dominant eigenvectors. The corresponding eigenvalue distribution for different blocks is also shown in Figure 1.

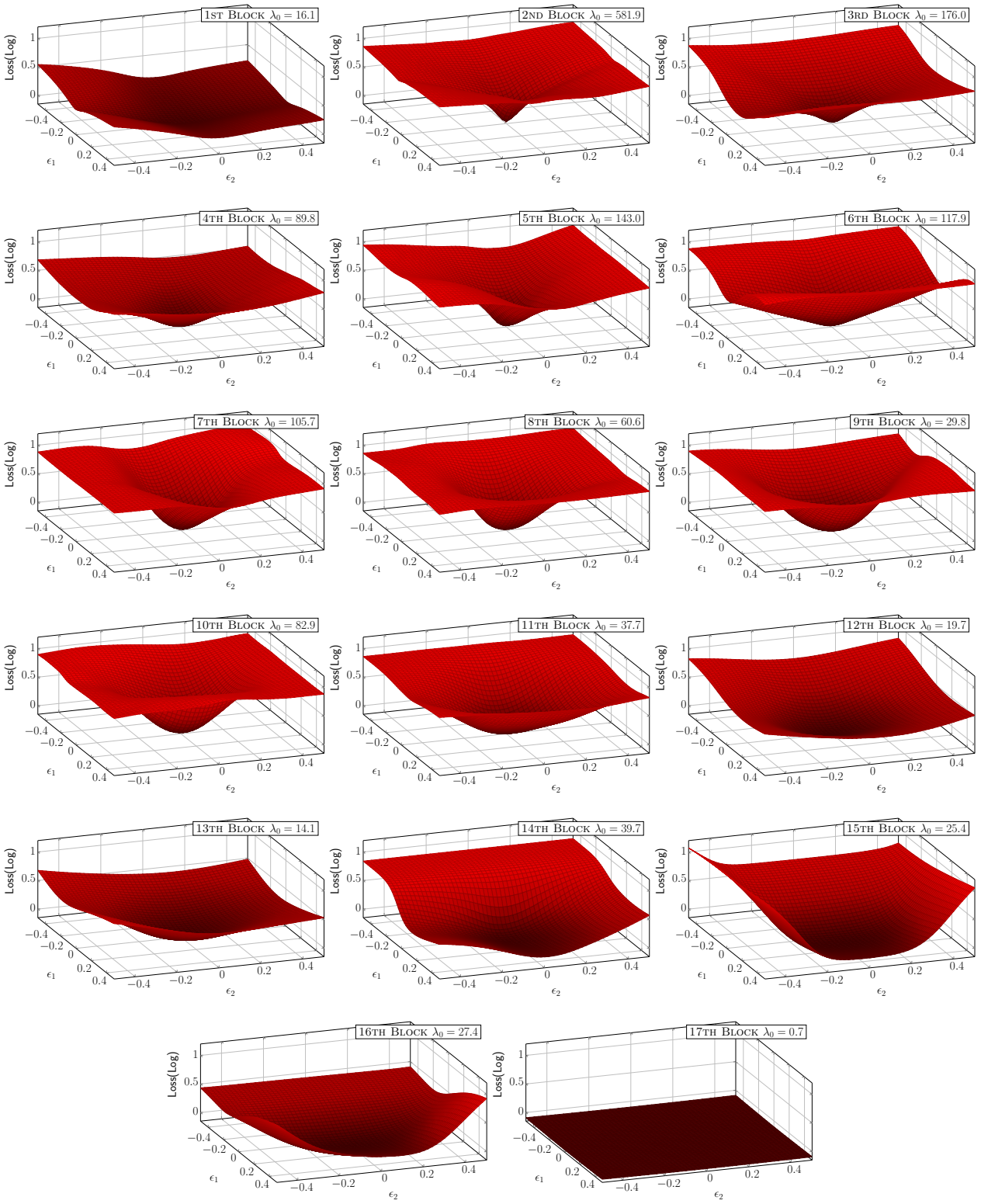

Figure 7. 3-D loss landscape of all blocks of InceptionV3 on ImageNet along the first two dominant eigenvectors of the Hessian. Here ϵ_1, ϵ_2 are scalars that perturb the parameters of the corresponding block along the first and second dominant eigenvectors. The corresponding eigenvalue distribution for different blocks is also shown in Figure 1.

Table 6. Block seperation and final block precision of ResNet20 on Cifar-10. Here we abbreviate convolutional layer as "Conv," fully connected layer as "FC."

Block	Layer(s)	Layer Type	Parameter Size	Weight bit	Activation bit
Block 0	Layer 0	Conv	4.32e2	8	8
Block 1	Layer 1-2	Conv	4.61e3	6	4
Block 2	Layer 3-4	Conv	4.61e3	6	4
Block 3	Layer 5-6	Conv	4.61e3	8	4
Block 4	Layer 7-8	Conv	1.38e4	3	4
Block 5	Layer 9-10	Conv	1.84e4	3	4
Block 6	Layer 11-12	Conv	1.84e4	3	4
Block 7	Layer 13-14	Conv	5.53e4	2	4
Block 8	Layer 15-16	Conv	7.37e4	2	4
Block 9	Layer 17-18	Conv	7.37e4	2	4
Block 10	Layer 19	FC	6.40e2	3	8

Table 7. Block seperation and final block precision of Inception-V3 on ImageNet. Here we abbreviate convolutional layer as "Conv," fully connected layer as "FC."

Block	Layer(s)	Layer Type	Parameter Size(M)	Weight bit	Activation bit
Block 0	Layer 0	Conv	8.64e-4	6	6
Block 1	Layer 1	Conv	9.22e-3	6	6
Block 2	Layer 2	Conv	1.84e-2	4	6
Block 3	Layer 3	Conv	5.12e-3	4	6
Block 4	Layer 4	Conv	0.14	4	6
Block 5	Layer 5-11	Conv	0.25	4	4
Block 6	Layer 12-18	Conv	0.28	4	4
Block 7	Layer 19-25	Conv	0.28	4	4
Block 8	Layer 26-29	Conv	1.15	2	4
Block 9	Layer 30-39	Conv	1.29	4	4
Block 10	Layer 40-49	Conv	1.69	4	4
Block 11	Layer 50-59	Conv	1.69	4	4
Block 12	Layer 60-69	Conv	2.14	4	4
Block 13	Layer 70-75	Conv	1.70	2	4
Block 14	Layer 76-84	Conv	5.04	2	4
Block 15	Layer 85-93	Conv	6.07	2	4
Block 16	Layer 94	FC	2.05	2	4