BRECQ: PUSHING THE LIMIT OF POST-TRAINING QUANTIZATION BY BLOCK RECONSTRUCTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Quantization of neural networks appears in many real-world systems and application to speed up the inference and downscale the memory footprint. However, quantization without end-to-end finetuning (which is called Post-training quantization) still remains a prohibitive task because there is insufficient data and computing resource to do backpropagation, especially when bitwidth is low. In this work, we push the limit of the bitwidth in PTQ down to INT2 for the first time. By approximating the asymptotic loss function, we derive a quantization framework, dubbed BRECQ, to refine the quadratic optimization objective and construct the quantization neural network with block reconstruction. To further employ the power of quantization, mixed precision technique is incorporated in our framework by approximating the inter-layer and intra-layer sensitivity. Extensive experiments on a variety number of models haven been conducted, including classification and detection tasks. Moreover, we show that post-training quantization can access comparable performance with quantization-aware training models without bells and whistles.

1 INTRODUCTION

The past decade has seen the rapid development of deep learning in many tasks, such as computer vision, autonomous driving, etc. However, the issue of huge computation cost and memory footprint requirements in deep learning has received considerable critical attention. Some works like neural architectures search (Zoph & Le, 2016) aims to design and search a tiny network, while others, like quantization (Hubara et al., 2017) and network pruning (Han et al., 2015) are designed to compress and accelerate off-the-shelf well-trained redundant networks.

Many quantization and network pruning methods follows a simple pipeline: training the original model and then finetune the quantized/pruned model. However, this pipeline requires full dataset and many computation resources to perform end-to-end backpropagation, and therefore is not acceptable on many IoT devices or in online learning situations (Cai et al., 2020). A very popular approach is quantizing the neural networks without retraining, which is called Post-training Quantization. Although PTQ is fast and light, it suffers form severe accuracy degeneration. For example, DFQ (Nagel et al., 2019) can quantize ResNet-18 to 8-bit without accuracy loss (69.7 top-1 accuracy) but in 4-bit quantization, it can only achieve 39% top-1 accuracy. The primary reason is the approximation in the parameter space is not equivalent to the approximation in model space thus we cannot assure the optimal minimization on the final task loss.

Recent works like (Nagel et al., 2020) recognize the problem and analyze the loss degradation by Taylor series expansion. Analysis on the second-order error term indicates we can reconstruct each layer output to approximate the task loss degeneration. However, their work cannot further quantize the weights into INT2 because the cross-layer dependence in Hessian matrix cannot be ignored when the perturbation on weight is not small enough. In this work, we formulate quantization as an optimization problem based on the Hessian matrix. We show that the second-order error can be transformed to network final outputs. Moreover, our contribution is threefold:

1. We show that different diagonal approximation of Hessian matrix leads to different scales of optimization. And neither layer-by-layer nor network-wise optimization is optimal for PTQ. We find an intermediate choice, block-wise optimization is generally suitable. We also incorporate mixed precision technique by the well-defined intra-block sensitivity measure.

2. To approximate the pre-activation Hessian, we use Fisher Information Matrix and thus assign each pre-activation with a importance measure during optimization.
3. We conduct extensive experiments to verify our proposed methods. We find that our method is applicable for a large variety of tasks and models. Moreover, we show that post-training quantization can quantize weights to INT2 without significant accuracy loss for the first time.

2 PRELIMINARIES

Notations Vectors are denoted by small bold letters and matrices (or tensors) are denoted by capital bold letters, for instance, $\mathbf{W}$ and $\mathbf{w}$ represent the weight tensor and its flattend version. Bracketed superscript $\mathbf{w}^{(\ell)}$ indicates the layer index. For a convolutional or a fully-connected layer, we mark its input and output vectors by $\mathbf{x}$ and $\mathbf{z}$. Thus given a feedforward neural network with n layers, we can denote the forward process by

$$\mathbf{x}^{(\ell+1)} = h(\mathbf{z}^{(\ell)}) = h(\mathbf{W}^{(\ell)}\mathbf{x}^{(\ell)} + \mathbf{b}^{(\ell)}), \quad 1 \leq \ell \leq n, \tag{1}$$

where $h(\cdot)$ indicates the activation function (ReLU in this paper). For simplicity, we omit the analysis of bias $\mathbf{b}^{(\ell)}$ as it can be merged into activation. $||\cdot||_F$ denotes the Frobenius norm.

Quantization Background Uniform symmetric quantization maps the floating-point numbers to several fixed-points. These points (or girds) has same interval and are symmetrically distributed. We denote the set that contains these girds as $\mathcal{Q}_b^{\text{u,sym}} = s \times \{-2^{b-1}, \ldots, 0, \ldots, 2^{b-1}-1\}$. Here, s is the step size between two grids and b is the bit-width. Quantization function, denoted by $q(\cdot) : \mathcal{R} \to \mathcal{Q}_b^{\text{u,sym}}$, is generally designed to minimize the quantization error:

$$\min_{\hat{\mathbf{w}}} ||\hat{\mathbf{w}} - \mathbf{w}||_F^2. \text{ s.t. } \hat{\mathbf{w}} \in \mathcal{Q}_b^{\text{u,sym}} \tag{2}$$

Solving this minimization problem, one can easily get the $q(\cdot)$ by leveraging the rounding-to-nearest operation $\lfloor \cdot \rceil$. Rounding-to-nearest is a prevalent method to perform quantization, e.g. PACT (Choi et al., 2018). However, recently many empirical or theoretical evidence support that simply minimizing the quantization error in parameter space does not bring optimal task performances. Specifically, Esser et al. (2020) propose to learn the step size s by gradient descent in quantization-aware training (QAT). Loss-aware Post-Quantization (Nahshan et al., 2019) finds the optimal step size when loss function is minimized without re-training the weights. Their motivation all towards a final objective, which is the task loss, i.e.,

$$\min L(\hat{\mathbf{w}}), \text{ s.t. } \hat{\mathbf{w}} \in \mathcal{Q}_b^{\text{u,sym}}. \tag{3}$$

While this optimization objective is simple and can be well-optimized in QAT scenarios, it is not easy to learn the quantized weight without end-to-end finetuning and sufficient training data. In post-training quantization settings, we only have full precision weights that $\mathbf{w}^\star = \arg\min_{\mathbf{w}} L(\mathbf{w})$ where $\mathbf{w} \in \mathcal{R}$ and a small subset of training data to do calibration.

Taylor Expansion It turns out that quantization imposed on weights could be viewed as a special case of weight perturbation. To quantitatively analyze the loss degradation caused by quantization, Nagel et al. (2020) use Taylor series expansions and approximates the loss degradation by

$$L(\mathbf{w} + \Delta\mathbf{w}) - L(\mathbf{w}) \approx \Delta\mathbf{w}^\mathsf{T}\mathbf{g}^{(\mathbf{w})} + \frac{1}{2}\Delta\mathbf{w}^\mathsf{T}\mathbf{H}^{(\mathbf{w})}\Delta\mathbf{w}, \tag{4}$$

where $\mathbf{g}^{(\mathbf{w})} = \mathbb{E}[\nabla_{\mathbf{w}} L]$ and $\mathbf{H}^{(\mathbf{w})} = \mathbb{E}[\nabla_{\mathbf{w}}^2 L]$ are the gradients and the Hessian matrix and $\Delta\mathbf{w}$ is the weight perturbation. Given the model is converged to a minimum, the gradients can be safely thought to be close to $\mathbf{0}$, however, optimizing with the large-scale full Hessian is memory-infeasible on many devices as the full Hessian requires terabytes memory space. To tackle this problem, they make two assumptions:

1. Layers are mutal-independent, thus the Hessian is in the form of layer-diagonal[1] and Kronecker-factored, i.e., $\mathbf{H}^{(\mathbf{w}^{(\ell)})} = \mathbb{E}[\mathbf{x}^{(\ell)}\mathbf{x}^{(\ell),\mathsf{T}} \otimes \mathbf{H}^{(\mathbf{z}^{(\ell)})}]$, where $\otimes$ is the Kronecker product.
2. The second-order derivatives of pre-activations are constant diagonal matrix ($\mathbf{H}^{(\mathbf{z}^{(\ell)})} = \text{diag}(c, \ldots, c)$) which is independent of data points.

[1]To prevent ambiguity, the layer-diagonal Hessian, in this paper, refers to a layer-wise block-diagonal matrix, while block-diagonal Hessian means the block size of the matrix is the size of a building block of CNNs.

At last, the objective is transformed to a practical but somehow robust proxy signal, *the change in feature-maps* ($\mathbf{z} = \mathbf{W}\mathbf{x}$), and the quantized model can be obtained by a layer-by-layer featuremap reconstruction algorithm (with few training images). Recent works, like Bit-Split (Wang et al., 2020) and AdaQuant (Hubara et al., 2020), also takes this layer-wise objective to improve the post-training quantization. However, they did not succeed to quantize weights to INT2. We think when $\Delta\mathbf{w}$ grows higher, the former assumptions does not hold and an accurate signal is required.

3 Proposed Method

3.1 Full Hessian Matrix

Denote the neural network output $\mathbf{z}^{(n)} = f(\theta)$, the loss function can be represented by $L(f(\theta))$ where $\theta = \text{vec}[\mathbf{w}^{(1),\mathsf{T}}, \dots, \mathbf{w}^{(n),\mathsf{T}}]^\mathsf{T}$ is the stacked vector of weights in all n layers. The Hessian matrix can be computed by

$$\frac{\partial^2 L}{\partial\theta_i \partial\theta_j} = \sum_a^m \frac{\partial L}{\partial \mathbf{z}_a^{(n)}} \frac{\partial^2 \mathbf{z}_a^{(n)}}{\partial\theta_i \partial\theta_j} + \sum_{a,b}^m \frac{\partial \mathbf{z}_a^{(n)}}{\partial\theta_i} \frac{\partial^2 L}{\partial \mathbf{z}_a^{(n)} \partial \mathbf{z}_b^{(n)}} \frac{\partial \mathbf{z}_b^{(n)}}{\partial\theta_j}, \tag{5}$$

where $\mathbf{z}^{(a)} \in \mathcal{R}^a$. Since the pretrained full preicison model is converged to a local minimum, we can assume the Hessian is postive-semidefinite (PSD),

Assumption 1. *The Hessian is converged to be PSD. Specifically, the converged model has $\nabla_{\mathbf{z}^{(n)}} L$ close to $\mathbf{0}$ so the first term in Eq. (5) is neglected and Hessian becoms the Gauss-Newton Matrix.*

In the rest of this section, we will focus on the property of Gauss-Newton (GN) matrix. GN matrix can be written in matrix form (Botev et al., 2017) as

$$\mathbf{H}^{(\theta)} = \mathbf{J}[\frac{\mathbf{z}^{(n)}}{\theta}]^\mathsf{T} \mathbf{H}^{\mathbf{z}^{(n)}} \mathbf{J}[\frac{\mathbf{z}^{(n)}}{\theta}], \tag{6}$$

where $\mathbf{J}[\frac{\mathbf{z}^{(n)}}{\theta}]$ is the Jacobian matrix of the network output with respect to the network parameters. Using the above equation, we transform the second-order error to the network output, as shown in the following theorem.

Theorem 3.1. *Consider an n-layer feedforward neural network with ReLU activation function. Assuming all weights are quantized, the second error optimization can be transformed by:*

$$\min_{\hat{\theta}} \Delta\theta^\mathsf{T} \mathbf{H}^{(\theta)} \Delta\theta \approx \min_{\hat{\theta}} \mathbb{E}\left[\Delta\mathbf{z}^{(n),\mathsf{T}} \mathbf{H}^{(\mathbf{z}^{(n)})} \Delta\mathbf{z}^{(n)}\right]. \tag{7}$$

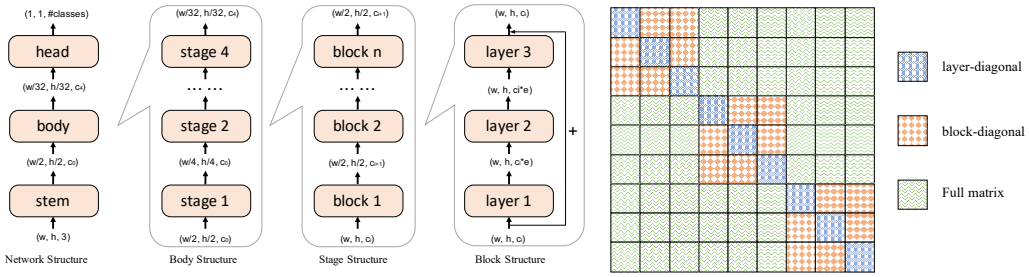
Remark 3.1. The same transformation is also applicable for activation quantization. The quadratic loss is defined as $\Delta\gamma^\mathsf{T} \mathbf{H}^{(\gamma)} \Delta\gamma$ where $\Delta\gamma = \text{vec}[\Delta\mathbf{x}^{(1),\mathsf{T}}, \dots, \Delta\mathbf{x}^{(n),\mathsf{T}}]^\mathsf{T}$.

We prove the theorem using the quadratic form, details can be found in Appendix A.2. Here we provide a sketch of the proof by matrix form. The change in network output $\Delta\mathbf{z}^{(n)}$ can be approximated using first-order Taylor expansion:

$$\Delta\mathbf{z}^{(n)} = \hat{\mathbf{z}}^{(n)} - \mathbf{z}^{(n)} \approx \mathbf{J}[\frac{\mathbf{z}^{(n)}}{\theta}] \Delta\mathbf{w}^{(\ell)}. \tag{8}$$

Therefore, combining Eq. (8) and Eq. (6) we can transform the large-scale second-order error into the change in network outputs. The theorem indicates a simple observation, suppose a well-trained teacher model and an initialized student model, we can minimize their discrepancy by reconstruct the network's final output $\mathbf{z}^{(n)}$. This is exactly the same with distillation (Hinton et al., 2015; Polino et al., 2018).

However, we should emphasize that distillation requires end-to-end backpropagation as in normal training procedure, which is impractical in edged IoT devices. Even if we can do distillation using a small subset of training data, we find the final results is even worse than the layer-by-layer optimization. The primary reason for this is approximating the first order error with final output could cost significant approximation error (cf. Eq. (8)) if we consider all layers in the neural networks, as we show in the following theorem



(a) A typical structure of CNN, (taken from Radosavovic et al. (2020)). Network are composed of a stem layer (first convolution on input images), a body and a head layer (average pooling with a fully connected layer). A body contain several stages, and a stage contain several blocks. A representative block is the bottleneck block with residual path.

(b) An example illustration of Hessian (or Fisher) matrix. Blue sub-block means the layer-diagonal and each layer are mutual-independent; orange sub-block consider the dependence inside a building block and green parts measure all dependencies.

Figure 1: We define 4 kinds of reconstruction granularity, namely net-wise, stage-wise, block-wise and layer-wise optimization, each of them corresponds an essential component of CNN.

Theorem 3.2. *Consider the above feedforward neural network network we analyzed, if all the layers are quantized, then we have*

$$|f(\hat{\theta}) - f(\theta) - \frac{\partial f(\theta)}{\partial \theta}(\hat{\theta} - \theta)| \leq (n-1)|f(\hat{\theta}) - f(\theta)| + |\hat{\theta}(\frac{\partial f(\theta)}{\partial \theta} - \frac{\partial f(\hat{\theta})}{\partial \hat{\theta}})|, \tag{9}$$

where $f(\theta)$ means the output of the network. $\hat{\theta}$ is the quantized weights. $\frac{\partial f(\theta)}{\partial \theta}$ is the Jacobian matrix $\mathbf{J}[\frac{f(\theta)}{\theta}]$.

Proof is available in Appendix A.2. It can be seen that the bound of difference between first-order error and the final output will scale up with number of layers we considered. In other words, there is a trade-off problem. For one thing, we should square up the off-diagonal loss in the Hessian, therefore according to Theorem 3.1, we should optimize the whole quantized network to minimize the final output of the network. And for another, using the final output of the whole network to replace the first-order error will lead to biased estimation, as descripted in Theorem 3.2. As a result, we should use layer-wise optimization as mentioned in Sec. 2 to eliminate the biased estimation.

3.2 Block Reconstruction versus Others

The above analysis clearly demonstrates network-wise as well as layer-wise calibration have their own drawbacks. The layer-wise optimization corresponds to layer-diagonal Hessian (Fig. 1b blue parts) and the network-wise optimization corresponds to full Hessian (Fig. 1b green parts). Similarly, we can define an intermediate block-diagonal Hessian. Such block-diagonal Hessian ignores the inter-block dependency and consider the intra-block dependency.

To this end, we define 2 kinds of intermediate *reconstruction granularity*: *1. Stage-wise reconstruction*: consider every intra-stage loss and reconstruct each stage's output by optimizing all parameters in that stage and *2. Block-wise reconstruction*: consider every intra-block loss, ignore inter-block loss and optimize each block with its output[2]. The relationship between network, stage, block and layer is illustrated in Fig. 1a. **We test these 4 kinds of reconstruction granularity and find that block-wise optimization outperform others**. We think this is because the main off-diagonal loss in the Hessian are concentrated in each block, as Fig. 1b orange part illustrated, while the inter-block loss is small and can be ignored in the optimization. Also, the stage-wise or net-wise optimization suffer from the bad estimation of the first-order error and degenerate the final performances. We report the quantitative comparison in Sec. 4.1.

We name our algorithm BRECQ, because we choose block as our base reconstruction unit. It is necessary to point out that our analysis does not give the optimal configuration of the reconstruction

[2]A stage is where the featuremaps will be downsampled and generate more channels, which is believed to produce higher-level features while a block is the basic building block of the CNN, which is the core component in the modern neural architectures (e.g. Residual Bottleneck Block as shown in Fig. 1a).

Algorithm 1: BRECQ optimization

Input: Pretrained FP model; Calibration dataset, iteration T
for *all* $i = 1, 2, \ldots, N$*-th block in the FP model* **do**
  Collect input data to the block $\mathbf{x}^{(i)}$, the FP output $\mathbf{z}^{(i)}$ and its gradient $\mathbf{g}^{(\mathbf{z}^{(i)})}$;
  for *all* $j = 1, 2, \ldots, T'$*-iteration* **do**
    Get quantized output $\hat{\mathbf{z}}^{(i)}$;
    Descend $\Delta\mathbf{z}^{(k),\mathsf{T}}\mathbf{F}^{(\mathbf{z}^{(k)})}\Delta\mathbf{z}^{(k)}$ and update the rounding of all the weights in this block;
    if *Activation Quantization is triggered* **then**
      Update the activation quantization step size.
  After optimization, compute the sensitivity for each layer and between layers (2-bit only);
return Quantized model, Sensitivities for mixed precision;

granularity. The choice of block-wise optimization comes from our experiments and we find this choice has two merits. (1) No hyper-parameters included and (2) applicable for all models and all tasks we tested.

3.3 FISHER INFORMATION MATRIX

With block-diagonal approximated Hessian matrix, we can transform any sub-block's second-order error to the output of this block $\Delta\mathbf{z}^{(\ell),\mathsf{T}}\mathbf{H}^{(\mathbf{z}^{(\ell)})}\Delta\mathbf{z}^{(\ell)}$, which requires the further computation of the pre-activation Hessian $\mathbf{H}^{(\mathbf{z}^{(\ell)})}$. One way is to follow Nagel et al. (2020) and assume $\mathbf{H}^{(\mathbf{z}^{(\ell)})} = \mathrm{diag}(c, \ldots, c))$. Therefore the quadratic loss becomes $||\Delta\mathbf{z}^{(\ell)}||^2$. This method might be easy to implement but lose too much information.

We use the diagonal Fisher Information Matrix (FIM) to replace the pre-activation Hessian. Formally, given a probabilistic model $p(x|\theta)$, the FIM is defined as the covariance of the gradients:

$$\mathbf{F}^{(\theta)} = \mathbb{E}\left[\nabla_\theta \log p_\theta(y|x) \nabla_\theta \log p_\theta(y|x)^\mathsf{T}\right] = -\mathbb{E}\left[\nabla^2_\theta \log p_\theta(y|x)\right] = -\mathbf{H}^{(\theta)}_{\log p(x|\theta)}. \tag{10}$$

The FIM is equal to the Hessian of the negative log-likelihood function, therefore, a simple corollary is that the Hessian of task loss will become FIM if the model distribution matches the true data distribution (LeCun et al., 2012). Although matching true data distribution seems impossible, this is the best we can do since pretrained model is converged. The FIM is also related to the asymptotic Kullback-Leibler divergence, see Appendix A.1 for more details.

The diagonal of the pre-activation FIM is equal to the squared gradients of each elements $\mathrm{diag}((\frac{\partial L}{\partial \mathbf{z}_1^{(\ell)}})^2, \ldots, (\frac{\partial L}{\partial \mathbf{z}_a^{(\ell)}})^2)$, which is successfully used in Adam (Kingma & Ba, 2014) for the second momentum. Martens & Grosse (2015) uses layer-diagonal FIM to performe Natural Gradient Descent to speed-up the convergence. Here, we use diagonal FIM to replace pre-activation Hessian. The optimization objective becomes

$$\min_{\hat{\mathbf{w}}} \mathbb{E}\left[\Delta\mathbf{z}^{(\ell),\mathsf{T}}\mathbf{H}^{(\mathbf{z}^{(\ell)})}\Delta\mathbf{z}^{(\ell)}\right] = \min_{\hat{\mathbf{w}}} \mathbb{E}\left[\Delta\mathbf{z}^{(\ell),\mathsf{T}}\mathrm{diag}((\frac{\partial L}{\partial \mathbf{z}_1^{(\ell)}})^2, \ldots, (\frac{\partial L}{\partial \mathbf{z}_a^{(\ell)}})^2)\Delta\mathbf{z}^{(\ell)}\right]. \tag{11}$$

Compared with the mean squared error minimization, the above minimization incorporates the squared gradient information. If the output has higher gradients, it will receive more attention when reconstructing it. Similar method for pruning the pre-activation has been proposed in Theis et al. (2018).

In this paper, we choose per-channel weight quantization and per-layer activation quantization. We adopt adaptive rounding for weights and learned step size for activation step size, see details in Appendix C.4.1. We formulate the overall calibration algorithm for a unified precision model in algorithm 1. We should emphasize that we only need a small subset of whole training dataset to calibrate the quantized model, (1024 in our experiments). And we can obtain a quantized ResNet-18 within 20 minutes on a single NVIDIA Tesla V100.

3.4 MIXED PRECISION

To further push the limit of post-training quantization, we employ mixed precision techniques, which can be formulated by

$$\min_{\mathbf{c}} L(\hat{\mathbf{w}}, \mathbf{c}), \quad \text{s.t. } H(\mathbf{c}) \leq \delta, \quad \mathbf{c} \in \{2, 4, 8\}^n. \tag{12}$$

Here, $\mathbf{c}$ is the bit-width vector with the shape of number of layers. $H(\cdot)$ is a hardware performance measurement function, which is used to ensure the mixed precision model has the same or lower hardware performance (e.g., memory and speed) than a predefined threshold δ. We choose 2, 4, 8-bit for mixed precision because they are most common in practical deployment.

Regarding the training loss L, we find that nearly all existing literature (Cai et al., 2020; Hubara et al., 2020; Dong et al., 2019) use layer-wise measurement. They all assume the total loss can be summed together. Therefore, the mixed precision problem becomes an integer programming problem. However, we argue that the loss measurement should contain two parts: diagonal loss and off-diagonal loss, the first is same with previous works and measure the sensitivity of each layer independently, while the off-diagonal loss is used to measure the cross-layer sensitivity. Theoretically, we should examine all permutations, which results in 3^n possibilities and prohibits the search algorithm. Therefore, we should reduce the search space. Our first attempt is to reduce the off-diagonal loss into block-level as we mentioned that the Hessian can be approximated to a block-diagonal matrix. Granted, we still find the search space is large, for example, if a block has four layers, then we have to consider the $3^4 = 81$ permutations for a single block. Based on our preliminary experiments, we find that 4-bit and 8-bit quantization nearly do not drop the final accuracy. Hence we only take 2-bit permutations into consideration and drastically reduce the search space. We use genetic algorithm (Guo et al., 2020) to search the optimal bitwidth configuration with hardware performance threshold, the algorithm is located in algorithm 2. Due to sapce limit, we put related works in Appendix B. Readers can refer to related works for a brief discussion on quantization and second-order analysis.

4 EXPERIMENTS

In this section, we report experimental results tested in ImageNet classification task and MS COCO object detection task. We provide our source codes in Google Drive. The detailed implementation of the experiments can be found in the Appendix. The rest of this section will contain ablation study on reconstruction granularity, classification and detection results, mixed precision results. In Appendix C, we conduct more experiments, including the impact of first and last layer, the impact of data size and data source and comparison with quantization-aware training.

4.1 ABLATION STUDY

We test four kinds of reconstruction granularity: Net-wise, Stage-wise, Block-wise, and Layer-wise reconstruction. We conduct experiments in MobileNet-V2 and ResNet-18 with 2-bit weight quantization for all layers except for the first and the last layer. It can be seen from Table 1 that Block-wise optimization outperform other methods. This result implies that the approximation error in net-wise and stage-wise optimization outweighs their off-diagonal loss. In ResNet-18, we find the difference is not significant, this is potentially because ResNet-18 only has 19 layers in the body and the block size as well as the stage size is 2, therefore leading to indistinct results.

Table 1: Ablation study.

Model	Layer	Block	Stage	Net
ResNet-18	65.19	66.39	66.01	54.15
MobileNet-V2	52.13	59.67	54.23	40.76

4.2 IMAGENET

We conduct experiments on a variety of modern deep learning architectures, including ResNet (He et al., 2016) with normal convolution, MobileNet-V2 (Sandler et al., 2018) with depthwise separable convolution and RegNet (Radosavovic et al., 2020) with group convolution. Last but not least important, we also investigate the neural architecture searched (NAS) models, MNasNet (Tan et al.,

Table 2: Accuracy comparison on *weight-only quantized* post-training models. Activations here are unquantized and kept full precision. We also conduct variance study for our experiments. Bold values indicates best results. * indicates ours implementation based on opensourced codes.

Methods	Bits (W/A)	ResNet-18	ResNet-50	MobileNet-V2	RegNet-600MF	RegNet-3.2GF	MnasNet-2.0
Full Prec.	32/32	71.08	77.00	72.49	73.71	78.36	76.68
Bias Correction*	4/32	50.43	64.64	62.82	67.09	71.73	72.31
OMSE (Choukroun et al., 2019)	4/32	67.12	74.67	-	-	-	-
AdaRound (Nagel et al., 2020)	4/32	68.71	75.23	69.78	71.97*	77.12*	74.87
AdaQuant (Hubara et al., 2020)	4/32	68.82	75.22	44.78	-	-	-
Bit-Split (Wang et al., 2020)	4/32	69.11	75.58	-	-	-	-
Ours	4/32	**70.70±0.07**	**76.29±0.04**	**71.66±0.04**	**73.02±0.09**	**78.04±0.04**	**76.00±0.02**
Bias Correction*	3/32	12.85	7.97	10.89	28.82	17.95	40.72
AdaRound (Nagel et al., 2020)*	3/32	68.07	73.42	64.33	67.71	72.31	69.33
AdaQuant (Hubara et al., 2020)*	3/32	58.12	67.61	12.56	-	-	-
Bit-Split (Wang et al., 2020)	3/32	66.75	73.24	-	-	-	
Ours	3/32	**69.81±0.05**	**75.61±0.09**	**69.50±0.12**	**71.48±0.07**	**77.22±0.04**	**74.58±0.08**
Bias Correction*	2/32	0.13	0.12	0.14	0.18	0.11	0.11
AdaRound (Nagel et al., 2020)*	2/32	55.96	47.95	32.54	25.66	24.70	30.60
AdaQuant (Hubara et al., 2020)*	2/32	0.30	0.49	0.11	-	-	-
Ours	2/32	**66.30±0.12**	**72.40±0.12**	**59.67±0.13**	**65.83±0.13**	**73.88±0.14**	**67.13±0.13**

Table 3: Accuracy comparison on *fully quantized* post-training models. Activations here are quantized to 4-bit. Notations follows the upper table.

Methods	Bits (W/A)	ResNet-18	ResNet-50	MobileNet-V2	RegNet-600MF	RegNet-3.2GF	MNasNet-2.0
Full Prec.	32/32	71.08	77.00	72.49	73.71	78.36	76.68
ACIQ-Mix (Banner et al., 2019)	4/4	67.0	73.8	-	-	-	-
ZeroQ (Cai et al., 2020)*	4/4	21.71	2.94	26.24	28.54	12.24	3.89
LAPQ (Nahshan et al., 2019)	4/4	60.3	70.0	49.7	57.71*	55.89*	65.32*
AdaQuant (Hubara et al., 2020)	4/4	67.5	73.7	34.95*	-	-	-
Bit-Split (Wang et al., 2020)	4/4	67.56	73.71	-	-	-	
Ours	4/4	**69.60±0.04**	**75.05**	**64.38**	**68.76**	**74.21**	**73.56**
ZeroQ (Cai et al., 2020)*	2/4	0.08	0.08	0.10	0.10	0.05	0.12
LAPQ (Nahshan et al., 2019)*	2/4	0.18	0.14	0.13	0.17	0.12	0.18
AdaQuant (Hubara et al., 2020)*	2/4	0.21	0.12	0.10	-	-	
Ours	2/4	**64.80±0.04**	**70.29**	**53.34±0.15**	**60.17**	**67.15**	**62.01**

2019). In Table 2, we only quantize weights into low-bit integer and keep activation full precision. We compare Bias Correction (Finkelstein et al., 2019), optimal MSE (Choukroun et al., 2019), AdaRound (Nagel et al., 2020), AdaQuant (Hubara et al., 2020), and Bit-split (Wang et al., 2020) as strong baselines. Note that the first and the last layer are kept with 8-bit. While most of existing methods have good performances in 4-bit quantization, they cannot correctly quantize the model into 2-bit. Our method can quantize ResNet-18 within 5% top-1 accuracy loss. It is worthwhile to point out that ResNets can achieve lowest degradation in 2-bit quantization than other networks, which is generally believed that normal convolution has more redundancy and can be easier to be compressed than compact convolution, such as depthwise separable in MobileNets. We further quantize activations into 4-bit to make quantized model run on integer-arthimetic hardware platform. We find that 4-bit activation quantization can have a huge impact on RegNet and MobileNet. Yet our methods still generates higher performances than other state-of-the-arts.

4.3 MS COCO

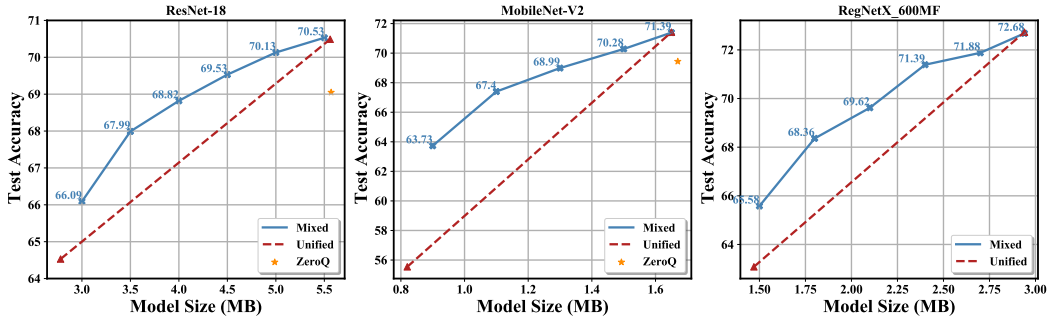
We then conduct experiments on object detection task on the two-stage Faster-RCNN (Ren et al., 2015) and the one-stage RetinaNet (Lin et al., 2017). ResNet-18, 50 as well as MobileNetV2 are adopted as backbones for the detection model. Results in Table 4 demonstrates our method nearly do not drop the performance in 4-bit weight quantization and 8-bit activation. Even when weight bit decreases to 2, the model still achieve near-to-original mAP.

4.4 Mixed Precision

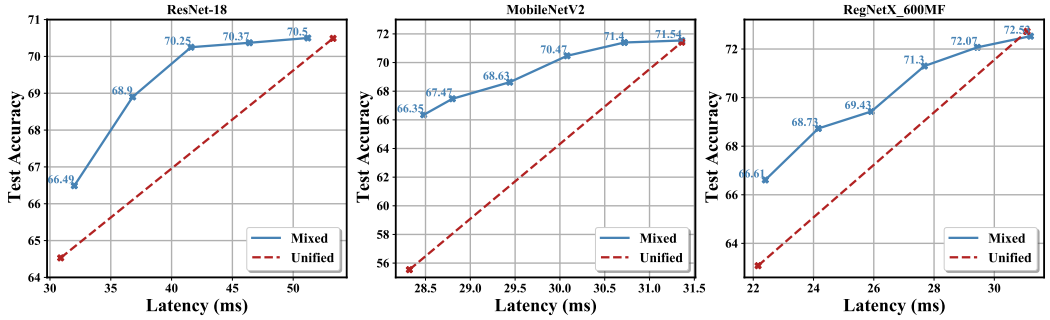
In this section, we incorporate the mixed precision technique. We test *(1) model-size guaranteed mixed precision* and *(2) latency guaranteed mixed precision*. We choose ResNet-18, MobileNet-V2 and RegNetX-600MF to validate the effecicay of our algorithm. Note that in this section, we keep activation in 8-bit because we only compare the discrepance between unified and the mixed precision in weights. We also quantize all layers instead of keep first and the last layer full precision. We omit

Table 4: Objection detection task (MS COCO) comparison on *fully quantized* post-training models. Activations here are quantized to 8-bit. We report mean Average Precision (mAP) value.

Models	**Backbone**	**Full Prec.**	**Bias Correction**		**AdaRound**		**Ours**		
		32/32	8/8	4/8	4/8	2/8	8/8	4/8	2/8
Faster RCNN (Ren et al., 2015)	ResNet-18	34.55	34.30	0.84	33.96	23.01	**34.53**	**34.34**	**31.82**
	ResNet-50	38.55	38.25	0.25	37.58	19.63	**38.54**	**38.29**	**34.23**
	MobileNet-V2	33.44	33.24	18.39	32.77	16.35	**33.40**	**33.18**	**27.54**
RetinaNet (Lin et al., 2017)	ResNet-18	33.20	33.00	0.04	32.59	19.93	**33.14**	**33.01**	**31.42**
	ResNet-50	36.82	36.68	0.07	36.00	19.97	**36.73**	**36.65**	**34.75**
	MobileNet-V2	32.63	**32.60**	18.47	31.89	14.10	32.57	**32.31**	**27.59**



(a) Model size guaranteed mixed precision.



(b) Latency guaranteed mixed precision.

Figure 2: Mixed Precision results.

3-bit weight quantization in unified precision because our mixed precision algorithm only leverages 2, 4, and 8-bit quantization. Latency settings can refer to Appendix. From Fig. 2a and Fig. 2b we find that (1) mixed precision consistently outperform unified precision below 4-bit hardware performance. However, in 4-bit threshold (top right red scatter), the mixed precision does not have more advantages, (2) mixed precision can produce many bit configuration that adapt to plently of hardware requirements while unified precision can only have 2 fixed models.

5 Conclusion

In this paper, we propose a novel quantization framework by analyzing the second-order information upon the case the perturbation.

REFERENCES

Shun-Ichi Amari. Natural gradient works efficiently in learning. *Neural computation*, 10(2):251–276, 1998.

Ron Banner, Yury Nahshan, and Daniel Soudry. Post training 4-bit quantization of convolutional networks for rapid-deployment. In *Advances in Neural Information Processing Systems*, pp. 7950–7958, 2019.

Aleksandar Botev, Hippolyt Ritter, and David Barber. Practical gauss-newton optimisation for deep learning. *arXiv preprint arXiv:1706.03662*, 2017.

Yaohui Cai, Zhewei Yao, Zhen Dong, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. Zeroq: A novel zero shot quantization framework. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 13169–13178, 2020.

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.

Yoni Choukroun, Eli Kravchik, Fan Yang, and Pavel Kisilev. Low-bit quantization of neural networks for efficient inference. In *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, pp. 3009–3018. IEEE, 2019.

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*, pp. 4857–4867, 2017.

Zhen Dong, Zhewei Yao, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. Hawq: Hessian aware quantization of neural networks with mixed-precision. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 293–302, 2019.

Steven K. Esser, Jeffrey L. McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S. Modha. Learned step size quantization. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=rkgO66VKDS.

Alexander Finkelstein, Uri Almog, and Mark Grobman. Fighting quantization bias with bias. *arXiv preprint arXiv:1906.03193*, 2019.

Ruihao Gong, Xianglong Liu, Shenghu Jiang, Tianxiang Li, Peng Hu, Jiazhen Lin, Fengwei Yu, and Junjie Yan. Differentiable soft quantization: Bridging full-precision and low-bit neural networks. *arXiv preprint arXiv:1908.05033*, 2019.

Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. {SINGLE} {path} {one}-{shot} {neural} {architecture} {search} {with} {uniform} {sampling}, 2020. URL https://openreview.net/forum?id=r1gPoCEKvH.

Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

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

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*, pp. 770–778, 2016.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

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.

Itay Hubara, Yury Nahshan, Yair Hanani, Ron Banner, and Daniel Soudry. Improving post training neural quantization: Layer-wise calibration and integer programming. *arXiv preprint arXiv:2006.10518*, 2020.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pp. 9–48. Springer, 2012.

Yuhang Li, Xin Dong, and Wei Wang. Additive powers-of-two quantization: An efficient non-uniform discretization for neural networks. In *International Conference on Learning Representations*, 2019.

Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pp. 2980–2988, 2017.

James Martens. Deep learning via hessian-free optimization. 2010.

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

Markus Nagel, Mart van Baalen, Tijmen Blankevoort, and Max Welling. Data-free quantization through weight equalization and bias correction. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1325–1334, 2019.

Markus Nagel, Rana Ali Amjad, Mart van Baalen, Christos Louizos, and Tijmen Blankevoort. Up or down? adaptive rounding for post-training quantization. *arXiv preprint arXiv:2004.10568*, 2020.

Yury Nahshan, Brian Chmiel, Chaim Baskin, Evgenii Zheltonozhskii, Ron Banner, Alex M Bronstein, and Avi Mendelson. Loss aware post-training quantization. *arXiv preprint arXiv:1911.07190*, 2019.

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

Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollár. Designing network design spaces. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10428–10436, 2020.

Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pp. 91–99, 2015.

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*, pp. 4510–4520, 2018.

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*, pp. 2820–2828, 2019.

Lucas Theis, Iryna Korshunova, Alykhan Tejani, and Ferenc Huszár. Faster gaze prediction with dense networks and fisher pruning. *arXiv preprint arXiv:1801.05787*, 2018.

Kuan Wang, Zhijian Liu, Yujun Lin, Ji Lin, and Song Han. Haq: Hardware-aware automated quantization with mixed precision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 8612–8620, 2019.

Peisong Wang, Qiang Chen, Xiangyu He, and Jian Cheng. Towards accurate post-training network quantization via bit-split and stitching. In *Proc. 37nd Int. Conf. Mach. Learn.(ICML)*, pp. 243–252, 2020.

Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

APPENDIX FOR BRECQ

A ADDITIONAL RESULTS AND MAIN PROOFS

A.1 QUADRATIC FORM OF KL DIVERGENCE

In this section, we provide the derivation of quadratic form of the KL divergence. First, we will show that Fisher is the negative expected Hessian of log-likelihood ($\mathbb{E}_{p(x|\theta)}[\mathbf{H}_{\log p(x|\theta)}]$) and then show that the Hessian of KL divergence is equal to $\mathbb{E}_{p(x|\theta)}[\mathbf{H}_{\log p(x|\theta)}]$.

Consider a probabilistic model parameterized by θ with likelihood function $p(x|\theta)$, the FIM is defined as $\mathbf{F}^{(\theta)} = \mathbb{E}_{p(x|\theta)}[\nabla_\theta \log p(x|\theta) \nabla_\theta \log p(x|\theta)^\mathsf{T}]$. Then, we start to derive the Hessian

$$\mathbb{E}_{p(x|\theta)}[\mathbf{H}_{\log p(x|\theta)}] = \mathbb{E}_{p(x|\theta)}\left[\nabla\nabla \log p(x|\theta)\right] = \mathbb{E}_{p(x|\theta)}\left[\nabla(\frac{\nabla p(x|\theta)}{p(x|\theta)})\right] \tag{13a}$$

$$= \mathbb{E}_{p(x|\theta)}\left[\frac{\nabla\nabla p(x|\theta)p(x|\theta) - \nabla p(x|\theta)\nabla p(x|\theta)^\mathsf{T})}{p(x|\theta)p(x|\theta)}\right] \tag{13b}$$

$$= \mathbb{E}_{p(x|\theta)}\left[\frac{\nabla\nabla p(x|\theta)}{p(x|\theta)}\right] - \mathbb{E}_{p(x|\theta)}\left[\left(\frac{\nabla p(x|\theta)}{p(x|\theta)}\right)\left(\frac{\nabla p(x|\theta)}{p(x|\theta)}\right)^\mathsf{T}\right] \tag{13c}$$

$$= \int \frac{\nabla\nabla p(x|\theta)}{p(x|\theta)} p(x|\theta)\mathrm{d}x - \mathbb{E}_{p(x|\theta)}[\log p(x|\theta) \log p(x|\theta)^\mathsf{T}] \tag{13d}$$

$$= \nabla\nabla \int p(x|\theta)\mathrm{d}x - \mathbf{F}^{(\theta)} \tag{13e}$$

$$= \nabla\nabla 1 - \mathbf{F}^{(\theta)} = -\mathbf{F}^{(\theta)} \tag{13f}$$

Note that the KL divergence between two same distributions is equal to 0. Before taking Taylor series expansion, we will show the first- and second-order derivative of the KL divergence.

$$\nabla_{\theta'}\mathcal{D}_{\mathrm{KL}}(p(x|\theta)||p(x|\theta'))\big|_{\theta'=\theta} = \nabla\mathbb{E}_{p(x|\theta)}\left[\log p(x|\theta) - \log p(x|\theta')\right]\big|_{\theta'=\theta} \tag{14a}$$

$$= -\mathbb{E}_{p(x|\theta)}\left[\nabla_{\theta'} \log p(x|\theta')\right]\big|_{\theta'=\theta} \tag{14b}$$

$$= -\int p(x|\theta)\nabla_{\theta'} \log p(x|\theta')\big|_{\theta'=\theta}\mathrm{d}x = 0 \tag{14c}$$

The Hessian of the KL divergence is given by:

$$\nabla\nabla_{\theta'}\mathcal{D}_{\mathrm{KL}}(p(x|\theta)||p(x|\theta'))\big|_{\theta'=\theta} = \nabla\nabla\mathbb{E}_{p(x|\theta)}\left[\log p(x|\theta) - \log p(x|\theta')\right]\big|_{\theta'=\theta} \tag{15a}$$

$$= -\mathbb{E}_{p(x|\theta)}\left[\nabla\nabla_{\theta'} \log p(x|\theta')\right]\big|_{\theta'=\theta} \tag{15b}$$

$$= \mathbf{F}^{(\theta)}, \tag{15c}$$

where the last line is the result of Eq. (13). To this end, we can directly use Taylor series expansion on the KL divergence given a small perturbation $\Delta\theta$

$$\mathcal{D}_{\mathrm{KL}}(p(x|\theta)||p(x|\theta+\Delta\theta)) \approx \mathcal{D}_{\mathrm{KL}}(p(x|\theta)||p(x|\theta)) + \Delta\theta^\mathsf{T}\nabla_{\theta'}\mathcal{D}_{\mathrm{KL}}(p(x|\theta)||p(x|\theta))\big|_{\theta'=\theta}$$
$$+ \frac{1}{2}\Delta\theta^\mathsf{T}\nabla\nabla_{\theta'}\mathcal{D}_{\mathrm{KL}}(p(x|\theta)||p(x|\theta'))\big|_{\theta'=\theta}\Delta\theta \tag{16a}$$

$$= \frac{1}{2}\Delta\theta^\mathsf{T}\mathbf{F}^{(\theta)}\Delta\theta \tag{16b}$$

A.2 PROOFS

proof of Theorem 3.1. We will prove the theorem using quadratic form. Denote the weight vector shape as $\theta \in \mathbb{R}^d$, and the network output vector shape as $\mathbf{z}^{(n)} \in \mathbb{R}^m$. The quadratic form of the $\Delta\theta^\mathsf{T}\mathbf{H}^{(\theta)}\Delta\theta$ can be represented by:

$$\Delta\theta^\mathsf{T}\mathbf{H}^{(\theta)}\Delta\theta = \sum_{i=1}^{d}\Delta\theta_i^2\left(\frac{\partial^2 L}{\partial\theta_i^2}\right) + 2\sum_{i<j}^{d}\Delta\theta_i\Delta\theta_j\frac{\partial L}{\partial\theta_i\theta_j}, \tag{17}$$

where L is the cross-entropy loss. Based on Eq. (5), we have

$$\frac{\partial^2 L}{\partial \theta_i \theta_j} = \sum_{a,b}^{m} \frac{\partial \mathbf{z}_a^{(n)}}{\partial \theta_i} \frac{\partial^2 L}{\partial \mathbf{z}_a^{(n)} \mathbf{z}_b^{(n)}} \frac{\partial \mathbf{z}_b^{(n)}}{\partial \theta_j} \tag{18}$$

Substituting above euqation in Eq. (17), we have

$$\begin{aligned}
\Delta\theta^{\mathsf{T}} \mathbf{H}^{(\theta)} \Delta\theta = & \sum_{i=1}^{d} \Delta\theta_i^2 \left(\sum_{a,b}^{m} \frac{\partial \mathbf{z}_a^{(n)}}{\partial \theta_i} \frac{\partial^2 L}{\partial \mathbf{z}_a^{(n)} \mathbf{z}_b^{(n)}} \frac{\partial \mathbf{z}_b^{(n)}}{\partial \theta_i} \right) \\
& + 2 \sum_{i<j}^{d} \Delta\theta_i \Delta\theta_j \left(\sum_{a,b}^{m} \frac{\partial \mathbf{z}_a^{(n)}}{\partial \theta_i} \frac{\partial^2 L}{\partial \mathbf{z}_a^{(n)} \mathbf{z}_b^{(n)}} \frac{\partial \mathbf{z}_b^{(n)}}{\partial \theta_j} \right) && \text{(19a)} \\
= & A + B && \text{(19b)}
\end{aligned}$$

Here, we divide the diagnoal part and off-diagonal part as A and B. Term A is given by:

$$\begin{aligned}
A &= \sum_{i=1}^{d} \Delta\theta_i^2 \left(\sum_{a,b}^{m} \frac{\partial \mathbf{z}_a^{(n)}}{\partial \theta_i} \frac{\partial^2 L}{\partial \mathbf{z}_a^{(n)} \mathbf{z}_b^{(n)}} \frac{\partial \mathbf{z}_b^{(n)}}{\partial \theta_i} \right) && \text{(20a)} \\
&= \sum_{i=1}^{d} \Delta\theta_i^2 \left(\sum_{a=1}^{m} \frac{\partial^2 L}{\partial \mathbf{z}_a^{(n)2}} (\frac{\partial \mathbf{z}_a^{(n)}}{\partial \theta_i})^2 + 2 \sum_{a<b}^{m} \frac{\partial^2 L}{\partial \mathbf{z}_a^{(n)} \mathbf{z}_b^{(n)}} (\frac{\partial \mathbf{z}_a^{(n)}}{\partial \theta_i})(\frac{\partial \mathbf{z}_b^{(n)}}{\partial \theta_i}) \right) && \text{(20b)} \\
&= \sum_{a=1}^{m} (\frac{\partial^2 L}{\partial \mathbf{z}_a^{(n)2}}) \left(\sum_{i=1}^{d} (\Delta\theta_i \frac{\partial \mathbf{z}_a^{(n)}}{\partial \theta_i})^2 \right) + 2 \sum_{a<b}^{m} (\frac{\partial^2 L}{\partial \mathbf{z}_a^{(n)} \mathbf{z}_b^{(n)}}) \left(\sum_{i=1}^{d} (\Delta\theta_i^2 \frac{\partial \mathbf{z}_a^{(n)}}{\partial \theta_i} \frac{\partial \mathbf{z}_b^{(n)}}{\partial \theta_i}) \right) && \text{(20c)} \\
&= C + D && \text{(20d)}
\end{aligned}$$

It can be seen that term A also has diagonal and off-diagnoal parts characterized by $\mathbf{H}^{\mathbf{z}^{(n)}}$, we mark them as C and D, respectively. Similarly, for term B, we have

$$\begin{aligned}
B &= 2 \sum_{i<j}^{d} \Delta\theta_i \Delta\theta_j \left(\sum_{a,b}^{m} \frac{\partial \mathbf{z}_a^{(n)}}{\partial \theta_i} \frac{\partial^2 L}{\partial \mathbf{z}_a^{(n)} \mathbf{z}_b^{(n)}} \frac{\partial \mathbf{z}_b^{(n)}}{\partial \theta_j} \right) && \text{(21a)} \\
&= 2 \sum_{i<j}^{d} \Delta\theta_i \Delta\theta_j \left(\sum_{a=1}^{m} (\frac{\partial^2 L}{\partial \mathbf{z}_a^{(n)2}})(\frac{\partial \mathbf{z}_a^{(n)}}{\partial \theta_i} \frac{\partial \mathbf{z}_a^{(n)}}{\partial \theta_j}) + 2 \sum_{a<b}^{m} (\frac{\partial^2 L}{\partial \mathbf{z}_a^{(n)} \mathbf{z}_b^{(n)}})(\frac{\partial \mathbf{z}_a^{(n)}}{\partial \theta_i} \frac{\partial \mathbf{z}_b^{(n)}}{\partial \theta_j}) \right) && \text{(21b)} \\
&= 2 \sum_{a=1}^{m} (\frac{\partial^2 L}{\partial \mathbf{z}_a^{(n)2}}) \left(\sum_{i<j}^{d} (\Delta\theta_i \Delta\theta_j \frac{\partial \mathbf{z}_a^{(n)}}{\partial \theta_i} \frac{\partial \mathbf{z}_a^{(n)}}{\partial \theta_j}) \right) \\
&\qquad + 4 \sum_{a<b}^{m} (\frac{\partial^2 L}{\partial \mathbf{z}_a^{(n)} \mathbf{z}_b^{(n)}}) \left(\sum_{i<j}^{d} (\Delta\theta_i \Delta\theta_j \frac{\partial \mathbf{z}_a^{(n)}}{\partial \theta_i} \frac{\partial \mathbf{z}_b^{(n)}}{\partial \theta_i}) \right) && \text{(21c)} \\
&= E + F && \text{(21d)}
\end{aligned}$$

Then, we combine $(C + E)$ and get

$$\begin{aligned}
C + E &= \sum_{a=1}^{m} (\frac{\partial^2 L}{\partial \mathbf{z}_a^{(n)2}}) \left(\sum_{i=1}^{d} (\Delta\theta_i \frac{\partial \mathbf{z}_a^{(n)}}{\partial \theta_i})^2 + 2 \sum_{i<j}^{d} (\Delta\theta_i \Delta\theta_j \frac{\partial \mathbf{z}_a^{(n)}}{\partial \theta_i} \frac{\partial \mathbf{z}_a^{(n)}}{\partial \theta_j}) \right) && \text{(22a)} \\
&= \sum_{a=1}^{m} (\frac{\partial^2 L}{\partial \mathbf{z}_a^{(n)2}}) \left(\sum_{i=1}^{d} (\Delta\theta_i \frac{\partial \mathbf{z}_a^{(n)}}{\partial \theta_i}) \right)^2 && \text{(22b)}
\end{aligned}$$

In a similar way, combine $(D+F)$ and we have

$$D+F = 2\sum_{a<b}^{m}(\frac{\partial^2 L}{\partial \mathbf{z}_a^{(n)}\mathbf{z}_b^{(n)}})\left(\sum_{i=1}^{d}(\Delta\theta_i^2\frac{\partial \mathbf{z}_a^{(n)}}{\partial\theta_i}\frac{\partial \mathbf{z}_b^{(n)}}{\partial\theta_i}) + 2\sum_{i<j}^{d}(\Delta\theta_i\Delta\theta_j\frac{\partial \mathbf{z}_a^{(n)}}{\partial\theta_i}\frac{\partial \mathbf{z}_b^{(n)}}{\partial\theta_i})\right) \tag{23a}$$

$$= 2\sum_{a<b}^{m}(\frac{\partial^2 L}{\partial \mathbf{z}_a^{(n)}\mathbf{z}_b^{(n)}})\left(\sum_{i=1}^{d}(\Delta\theta_i\frac{\partial \mathbf{z}_a^{(n)}}{\partial\theta_i})\right)\left(\sum_{i=1}^{d}(\Delta\theta_i\frac{\partial \mathbf{z}_b^{(n)}}{\partial\theta_i})\right) \tag{23b}$$

Finally, we have

$$\Delta\theta^{\mathsf{T}}\mathbf{H}^{(\theta)}\Delta\theta = A+B = (C+E)+(D+F) \tag{24a}$$

$$= \sum_{a=1}^{m}(\frac{\partial^2 L}{\partial \mathbf{z}_a^{(n)2}})\left(\sum_{i=1}^{d}(\Delta\theta_i\frac{\partial \mathbf{z}_a^{(n)}}{\partial\theta_i})\right)^2$$

$$+ 2\sum_{a<b}^{m}(\frac{\partial^2 L}{\partial \mathbf{z}_a^{(n)}\mathbf{z}_b^{(n)}})\left(\sum_{i=1}^{d}(\Delta\theta_i\frac{\partial \mathbf{z}_a^{(n)}}{\partial\theta_i})\right)\left(\sum_{i=1}^{d}(\Delta\theta_i\frac{\partial \mathbf{z}_b^{(n)}}{\partial\theta_i})\right) \tag{24b}$$

$$= (\Delta\theta\mathbf{J}\left[\frac{\mathbf{z}^{(n)}}{\theta}\right])^{\mathsf{T}}\mathbf{H}^{(\mathbf{z}^{(n)})}\Delta\theta\mathbf{J}\left[\frac{\mathbf{z}^{(n)}}{\theta}\right] \tag{24c}$$

To this end, we use the first-order Taylor expansion as we did in Eq. (8) to approximate the change in network output, i.e.,

$$\Delta\mathbf{z}^{(n)} \approx \Delta\theta\mathbf{J}\left[\frac{\mathbf{z}^{(n)}}{\theta}\right] \tag{25}$$

Therefore, the final objective is transformed to $\Delta\mathbf{z}^{(n),\mathsf{T}}\mathbf{H}^{(\mathbf{z}^{(n)})}\Delta\mathbf{z}^{(n)}$. □

In order to prove Theorem 3.2, we first introduce the following lemma:

Lemma A.1. *Given an n-layer feedforward neural network with ReLU activation, and a single input image $\mathbf{x}^{(1)}$, we have:*

$$\theta\mathbf{J}\left[\frac{\mathbf{z}^{(n)}}{\theta}\right] = n\times\mathbf{z}^{(n)}. \tag{26}$$

Remark A.1. The above lemma holds for any activation that satisfies $h(x) = h'(x)x$ where $h'(\cdot)$ is the derivative. For example, leaky ReLU.

Proof. Recall that $\theta = \mathrm{vec}[\mathbf{w}^{(1),\mathsf{T}}, \mathbf{w}^{(2),\mathsf{T}}, \ldots, \mathbf{w}^{(n),\mathsf{T}}]^{\mathsf{T}}$ is the stacked vector of weights in each layer. We first prove that for weights in any layer, $\mathbf{w}^{(i)}\mathbf{J}[\frac{\mathbf{z}^{(n)}}{\mathbf{w}^{(i)}}] = \mathbf{z}^{(n)}$ holds, then we can prove the lemma by adding the equation for each layer.

Consider the Jacobian matrix $\mathbf{J}[\frac{\mathbf{z}^{(i+1)}}{\mathbf{z}^{(i)}}]$, it can computed by

$$\mathbf{J}\left[\frac{\mathbf{z}^{(i+1)}}{\mathbf{z}^{(i)}}\right] = \mathbf{J}\left[\frac{\mathbf{x}^{(i+1)}}{\mathbf{z}^{(i)}}\right]\mathbf{J}\left[\frac{\mathbf{z}^{(i+1)}}{\mathbf{x}^{(i+1)}}\right] = \mathbf{B}^{(i)}\mathbf{W}^{(i+1)} \tag{27}$$

where $\mathbf{B}^{(i)} = \mathrm{diag}(h'(\mathbf{z}^{(i)}))$ is the gradient of activation function. Therefore, according to the property of (leaky) ReLU activation $h(x) = h'(x)x$, we have

$$\mathbf{z}^{(i)}\mathbf{J}\left[\frac{\mathbf{z}^{(i+1)}}{\mathbf{z}^{(i)}}\right] = h(\mathbf{z}^{(i)})\mathbf{W}^{(i+1)} = \mathbf{z}^{(i+1)} \tag{28}$$

For any $i < n$, i.e., any layer except the last layer, we have $\mathbf{J}[\frac{\mathbf{z}^{(n)}}{\mathbf{w}^{(i)}}] = \mathbf{J}[\frac{\mathbf{z}^{(i)}}{\mathbf{w}^{(i)}}]\mathbf{J}[\frac{\mathbf{z}^{(n)}}{\mathbf{z}^{(i)}}]$. The first term is equal to $\mathbf{x}^{(i)}$. Hence, we have

$$\mathbf{w}^{(i)}\mathbf{J}\left[\frac{\mathbf{z}^{(n)}}{\mathbf{w}^{(i)}}\right] = \mathbf{z}^{(i)}\mathbf{J}\left[\frac{\mathbf{z}^{(n)}}{\mathbf{z}^{(i)}}\right] = \mathbf{z}^{(i)}\prod_{j=i}^{n-1}\mathbf{J}\left[\frac{\mathbf{z}^{(j+1)}}{\mathbf{z}^{(j)}}\right] = \mathbf{z}^{(n)} \tag{29}$$

For $i = n$, it is easy to get $\mathbf{w}^{(n)}\mathbf{J}[\frac{\mathbf{z}^{(n)}}{\mathbf{w}^{(n)}}] = \mathbf{w}^{(n)}\mathbf{x}^{(n)} = \mathbf{z}^{(n)}$, therefore:

$$\theta \mathbf{J}\left[\frac{\mathbf{z}^{(n)}}{\theta}\right] = \sum_{i=1}^{n} \mathbf{w}^{(i)}\mathbf{J}\left[\frac{\mathbf{z}^{(n)}}{\mathbf{w}^{(i)}}\right] = n \times \mathbf{z}^{(n)} \tag{30}$$

□

proof of Theorem 3.2. Based on the above lemma, we have $\frac{\partial f(\theta)}{\partial \theta}\theta = nf(\theta)$, where f is the neural network function and $f(\theta)$ denotes the final output. Therefore, we have

$$|f(\hat{\theta}) - f(\theta) - \frac{\partial f(\theta)}{\partial \theta}(\hat{\theta} - \theta)| = |f(\hat{\theta}) - f(\theta) - \frac{\partial f(\theta)}{\partial \theta}\hat{\theta} + nf(\theta)| \tag{31a}$$

$$= |f(\hat{\theta}) - \frac{\partial f(\hat{\theta})}{\partial \hat{\theta}}\hat{\theta} + \frac{\partial f(\hat{\theta})}{\partial \hat{\theta}}\hat{\theta} - \frac{\partial f(\theta)}{\partial \theta}\hat{\theta} + (n-1)f(\theta)| \tag{31b}$$

$$= |(n-1)(f(\theta) - f(\hat{\theta})) + \hat{\theta}\left(\frac{\partial f(\hat{\theta})}{\partial \hat{\theta}} - \frac{\partial f(\theta)}{\partial \theta}\right)| \tag{31c}$$

$$\leq (n-1)|f(\theta) - f(\hat{\theta})| + |\hat{\theta}\left(\frac{\partial f(\hat{\theta})}{\partial \hat{\theta}} - \frac{\partial f(\theta)}{\partial \theta}\right)|, \tag{31d}$$

where the last line follows $|x + y| \leq |x| + |y|$. □

B Related Works

B.1 Quantization

Quantization can be classified into two category: Quantization-aware Training (QAT) and Post-training Quantization (PTQ). QAT method usually train a quantized model from scratch or fine-tune it from the pre-trained full precision models. Rounding floating-point numbers to fixed-points numbers will produce 0 gradients almost everywhere. Therefore, most QAT methods employ the Straight-Through Estimator (STE) for gradients approximation. Gong et al. (2019) use differentiable tanh function to gradually approach the step function. Choi et al. (2018) first introduce parameterized clipping thresholds to learn it from task loss. Esser et al. (2020) propose the Learned Step Size Quantization by learning the clipping range with STE. Apart from uniform quantization, some works like Li et al. (2019) argue that non-uniform quantization have better performance than uniform quantization while keeping its efficiency. Despite the promising results given by QAT methods, they usually needs more than 100 GPU hours to get it. Meanwhile, we cannot assure the full training data is ready-to-use. In that case, PTQ plays an important role which is what we focus on in this paper. Generally, most deep learning models can be safely quantized to 8-bit without re-training. Data-Free Quantization Nagel et al. (2019) even do layer-wise 8-bit PTQ without any data. However, in 4-bit quantization, most parameter space based method cannot obtain good performances. Bias Correction Finkelstein et al. (2019) and LAPQ Nahshan et al. (2019) narrow the gap but still cannot completely recover the full precision performances in 4-bit quantization. Recently, Nagel et al. (2020) propose to do layer-wise calibration and made huge progress in 4-bit quantization. Our work continue its analysis on Taylor expansion and consider the off-diagonal loss.

A different axes in quantization is the precision allocation scheme. It is generally believed that each layer has its own redundancy and hardware performances. Hardware-aware Quantization (HAQ Wang et al. (2019)) leverages reinforcement learning to search the optimal bitwidth configuration. Hessian-aware Weight Quantization (HAWQ) (Dong et al., 2019) utilizes the second-order information to decide the bitwidth. Mixed precision also appears in PTQ, such as Pareto frontier method in ZeroQ (Cai et al., 2020) and Integer Programming method in AdaQuant (Hubara et al., 2020). However, apart from end-to-end search like HAQ, the HAWQ, ZeroQ and AdaQuant do not consider the inter-layer sensitivities, i.e., they assume the choice of bit-width is independent from bit-width for other layers. Our work approximate the inter-layer sensitivity by intra-block sensitivity.

B.2 SECOND-ORDER ANALYSIS AND OPTIMIZATION

The history of second-order information in perturbation analysis can be traced to 1990s like Optimal Brain Surgeon (Hassibi & Stork, 1993; Dong et al., 2017). The Hessian matrix is essential for pruning and quantization. As aforementioned, HAWQ use the largest eigenvalue of Hessian to determine the sensitivity. Hessian matrix is also important for second-order optimization like Newton's method as it consists of the curvature information. However, calculating the real full Hessian is prohibitive on today's deep learning architectures because the Hessian will scale up with the number of parameters rapidly. Therefore, approximations are made to simplify the calculation and make storage more flexible. Such as Guass-Newton optimization with Kronecker-factored recursive approximation Botev et al. (2017). Hessian-Free optimization (Martens, 2010) avoid the explicit computation of Hessian matrix by solving the linear system $g = Hv$. Second-order optimization with FIM are called Natural Gradient Descent (Amari, 1998). Adam Kingma & Ba (2014) uses the moving average of the diagonal FIM to represent the curvature information. K-FAC (Martens & Grosse, 2015) utilize the layer-diagonal FIM and the approximation of the expected Kronecker product to compute the curvature information.

C EXPERIMENTS

C.1 EFFECT OF THE FIRST AND THE LAST LAYER

Many papers claim that the first and the last layer can have a huge impact on the final accuracy. In this section, we investigate this phenomenon as well as the impact of the first and the last layer on hardware performances. We test ResNet-18, MobileNet-V2 as well as RegNet-600MF. Our observations includes:

1. In terms of accuracy, the 4-bit quantization is essentially good, both of these two layers won't drop too much accuracy (with 0.2%). But in 2-bit quantization, the last fully connected layer is far more important than the first layer. We also observe that first layer in MobileNet and RegNet (7×7 kernel, 32 channels) is slightly more sensitive than than in ResNet-18 (7×7 kernel, 64 channels).
2. In terms of model size, the first layer merely do not have much impact because the input images only has 3 channels, while the last layer contain many weight parameters and greatly affect the memory footprint. We should point out that just because the model size in the first layer is low doesn't mean the memory burden is low, because input image will cost huge memory space.
3. In terms of latency, the situation depends on architectures. For example, in ResNet-18 the first layer has a huge impact on the latency, while in MobileNet-V2 and RegNet-600MF the last layer is more important than the first layer. This is because the latency is affected by multiple factors, such as input size of the featuremap, the FLOPs and the weight memory size. The arithmetic intensity (OPs/byte) greatly affect the latency. We find that high arithmetic intensity generate less latency difference between differnet bit-width.

Table 5: Impact of the first and the last layer

Models	**No Quantization**		**Precision 4/8**			**Precision 2/8**		
	First	Last	Accuracy	Model Size	Latency	Accuracy	Model Size	Latency
ResNet-18 FP: 71.08	✓	✓	70.76	5.81 MB	70.72 ms	66.30	3.15 MB	59.84 ms
	✗	✓	70.66	5.81 MB	53.76 ms	65.95	3.15 MB	31.20 ms
	✓	✗	70.64	5.57 MB	70.08 ms	64.87	2.79 MB	58.72 ms
	✗	✗	70.58	5.56 MB	53.28 ms	64.53	2.78 MB	30.88 ms
MobileNet-V2 FP: 72.49	✓	✓	71.80	2.26 MB	32.80 ms	59.59	1.74 MB	30.40 ms
	✗	✓	71.69	2.26 MB	32.64 ms	59.13	1.74 MB	30.24 ms
	✓	✗	71.42	1.65 MB	31.52 ms	56.29	0.83 MB	28.48 ms
	✗	✗	71.42	1.65 MB	31.36 ms	55.58	0.82 MB	28.32 ms
RegNet-600MF FP: 73.71	✓	✓	72.98	3.19 MB	31.84 ms	65.66	1.84 MB	23.20 ms
	✗	✓	72.89	3.19 MB	31.68 ms	65.83	1.85 MB	22.88 ms
	✓	✗	72.69	2.94 MB	31.20 ms	62.93	1.47 MB	22.40 ms
	✗	✗	72.73	2.94 MB	31.04 ms	63.08	1.47 MB	22.08 ms

In conclusion, we find that keeping 8-bit first and last layer while remain other layers with 4-bit is unnecessary. Especially in ResNet-18, we find that setting all layers to 4-bit results in 53.3 ms latency and is lower than the 59.8 ms in 2-bit quantization (with first and last layer 8-bit), but the accuracy is even 4% higher. Such phenomenon indicates the potential power of the mixed precision.

C.2 Effect of Data

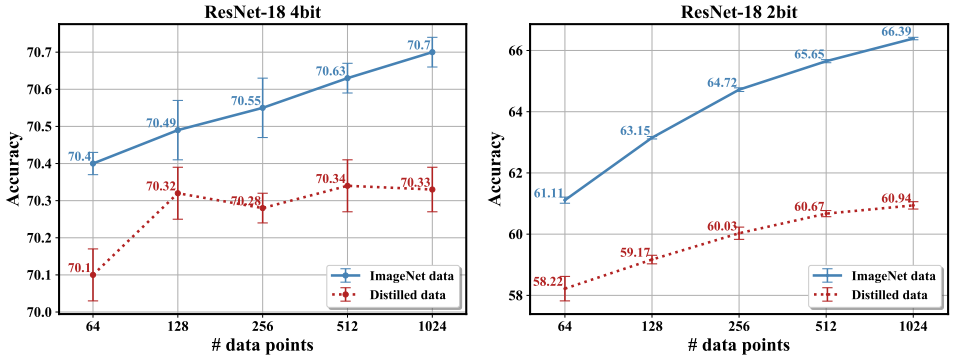


Figure 3: Effect of #data points and data source.

We test the size of calibration dataset and the source of the data on ResNet-18. We test different numbers of input data point and find that the improvement in 4-bit quantization is trivial. Yet in 2-bit quantization we can see that the accuracy increase 5% when #data points increase. We also test the distilled data introduced in ZeroQ (Cai et al., 2020). Distilled data is learned from pretrained models' BN statistics, i.e., $\mathbf{x}^1_{\text{distilled}} = \arg\min_{\mathbf{x}\in\mathcal{R}} \sum_{i=1}^{n}((\mu_i - \hat{\mu}_i)^2 + (\varsigma_i - \hat{\varsigma}_i))$ where μ_i and ς_i is the original mean and variance in BN statistics of the (i)-th layer. We find that distilled data performs good in 4-bit quantization but still has a large margin with the original ImageNet dataset in 2-bit quantization. We also find the final accuracy does not benefit much from the increase of number of distilled data, this might because the distilled data are minimized by a same objective and has low diversity.

C.3 Comparison with Quantization-aware Training

Table 6: Performance as well as training cost comparison with quantization-aware training (QAT).

Models	Methods	Precision	Accuracy	Model Size	Training Data	GPU hours
ResNet-18 FP: 71.08	ZeroQ (Cai et al., 2020))	4/4	21.20	5.81 MB	0	0.001
	BRECQ (Ours)	4/4	69.60	5.81 MB	1024	0.4
	BRECQ (Distilled Data)	4/4	69.32	5.81 MB	0	0.4
	PACT (Choi et al., 2018)	4/4	69.2	5.81 MB	1.2 M	100
	DSQ (Gong et al., 2019)	4/4	69.56	5.81 MB	1.2 M	100
	LSQ (Esser et al., 2020)	4/4	71.1	5.81 MB	1.2 M	100
MobileNet-V2 FP: 72.49	BRECQ (Ours)	4/4	67.50	2.26 MB	1024	0.8
	PACT (Choi et al., 2018)	4/4	61.40	2.26 MB	1.2 M	192
	DSQ (Gong et al., 2019)	4/4	64.80	2.26 MB	1.2 M	192
	BRECQ (Ours)	Mixed/8	70.74	1.38 MB	1024	3.2
	HAQ (Wang et al., 2019)	Mixed/8	70.90	1.38 MB	1024	384

In this section, we compare our algorithm (post-training quantization) with some quantization-aware training methods, including PACT (Choi et al., 2018), DSQ Gong et al. (2019), LSQ Esser et al. (2020), and a mixed precision technique HAQ (Wang et al., 2019). Table 6 shows that BRECQ can achieve comparable accuracy results with existing quantization-aware training models. In addition, our method can surpass them in 4-bit MobileNet-V2 while keeping the training GPU hours in less than 1. Our method also has higher accuracy than HAQ, which is training-based mixed precision. Note that our GPU hours includes 3 unified precision training (2-, 4-, 8-bit respectively) and further mixed precision training only needs to check the look-up table. In stead, HAQ would start the end-to-end search for each hardware performance threshold.

C.4 IMPLEMENTATION

C.4.1 LEARNING STRATEGIES

In this work, we mainly focus on developing optimization objective rather than optimization strategies. We observe adaptive rounding performs well in post-training quantization. A brief introduction on AdaRound is given below, the detailed algorithm can be found in Nagel et al. (2020).

Predominant quantization function is performed by rounding-to-nearest operation: $\hat{\mathbf{w}} = s \times \text{clip}(\lfloor \mathbf{w}/s \rceil, n, p)$. AdaRound optimizes the rounding policy in post-training quantization. Specifically, all weights are default rounded by floor operation, and a learnable variable $\mathbf{v}$ determines the final rounding result is flooring or ceiling. A sigmoid-like function $\sigma(\cdot)$ keeps the learnable variable $\mathbf{v}$ moving between 0 and 1 and a regularization term assures the $\sigma(\mathbf{v})$ can converged to either 0 or 1. The formulation is given by

$$\hat{\mathbf{w}} = s \times \text{clip}\left(\lfloor \frac{\mathbf{w}}{s} \rfloor + \sigma(\mathbf{v}), n, p\right) \tag{32}$$

The minimization problem together with the regularization is given by

$$\min_{\mathbf{v}} \mathbb{E}\left[\Delta \mathbf{z}^{(\ell),\mathsf{T}} \text{diag}((\frac{\partial L}{\partial \mathbf{z}_1^{(\ell)}})^2, \ldots, (\frac{\partial L}{\partial \mathbf{z}_a^{(\ell)}})^2) \Delta \mathbf{z}^{(\ell)}\right] + \lambda \sum_i \left(1 - |2\sigma(\mathbf{v}_i) - 1|^{\beta}\right), \tag{33}$$

where progressively decreasing β in the calibration ensures the $\sigma(\mathbf{v})$ converged to binary values. The activations cannot be quantized using adaptive rounding because they vary with different input data points. Thus, we can only adjust it quantization step size Esser et al. (2020). Denote the quadratic loss in above equation as L_q, the gradients of step size is given by

$$\frac{\partial L_q}{\partial s} = \begin{cases} \frac{\partial L_q}{\partial \hat{\mathbf{x}}} & \text{if } \mathbf{x} > n \\ \frac{\partial L_q}{\partial \hat{\mathbf{x}}} \left(\frac{\hat{\mathbf{x}}}{s} - \frac{\mathbf{x}}{s}\right) & \text{if } 0 \leq \mathbf{x} < \alpha \\ 0 & \text{if } \mathbf{x} \leq 0 \end{cases}, \tag{34}$$

where all step size in the block will be optimized.

C.4.2 GENETIC ALGORITHM FOR MIXED PRECISION

Algorithm 2: Genetic algorithm

Input: Random initialized population P_0 with population size S; iteration T, mutation probability p; Hardware performance threshold δ; Hardware measurement function $H(\cdot)$

$TopK = \varnothing$;
for *all* $t = 1, 2, \ldots, T$*-th iteration* **do**
  Evaluate fitness value for each individual ;
  Update and sort $TopK$ based on fitness function;
  repeat
    New bitwidth configuration by crossover $\mathbf{c}_{cross} = \text{Crossover}(TopK)$;
    $P_{crossover} += \mathbf{c}_{cross}$ if $H(\mathbf{c}_{cross}) < \delta$;
  until *Size of* $P_{crossover}$ *equal to* $S/2$;
  repeat
    New bitwidth configuration by mutation $\mathbf{c}_{mutate} = \text{Mutate}(TopK, \text{probability} = p)$;
    $P_{mutate} += \mathbf{c}_{mutate}$ if $H(\mathbf{c}_{mutate}) < \delta$;
  until *Size of* P_{mutate} *equal to* $S/2$;
  $P_t = P_{crossover} \cup P_{mutate}$;
  $P_{mutate} = \varnothing$;
  $P_{crossover} = \varnothing$;
return entry with the highest fitness value;

C.4.3 LATENCY ACQUISITION

C.4.4 IMPLEMENTATION DETAILS

We run baseline methods from opensourced codes, such as ZeroQ (Cai et al., 2020)[3], AdaQuant (Hubara et al., 2020)[4], and LAPQ (Nahshan et al., 2019)[5].

The ImageNet dataset consists of 1.2M training images and 50K test images. We follows standard pre-process (He et al., 2016) to get 1024 224×224 input images as calibration dataset. We fold batch normalization layer into convolution and freeze the BN statistics before post-training quantization. We use Adam optimizer (Kingma & Ba, 2014) to learn the weight rounding and activation range to reconstruct the block output. Note that some layers are not a component of any block, such as the first convolutional layer and the last fully connected layer. These layers use the naive layer reconstruction. The batch size of learning is set to 32 and each block will be optimized for 2×10^4 iterations. Learning rate is set to 10^{-3} during the whole learning process. Other hyper-parameters such as the temperature β is kept the same with Nagel et al. (2020). After reconstruction, we will store the sensitivity measured on calibration dataset. Note that we will store intra-block sensitivity in 2-bit quantization. The sensitivity, as well as hardware performances for each layer will be stored in a look-up table. When calculating the fitness value and determining the hardware performances in genetic algorithm, we will check the look-up table.

For object detection task, we use 256 training images taken from MS COCO dataset for calibration. The image resolution is set to 800 (max size 1333) for ResNet-18 and ResNet-50, while the image resolution for MobileNet-V2 is set to 600 (max size 1000).

[3] https://github.com/amirgholami/ZeroQ
[4] https://github.com/itayhubara/CalibTIP
[5] https://github.com/ynahshan/nn-quantization-pytorch/tree/master/lapq