GWQ: Group-Wise Quantization Framework for Neural Networks

Jiaming Yang YANGJIAMING1@STU.SCU.EDU.CN
Chenwei Tang* TANGCHENWEI@SCU.EDU.CN
Caiyang Yu YUCY324@GMAIL.COM
Jiancheng Lv* LVJIANCHENG@SCU.EDU.CN
College of Computer Science, Sichuan University, Chengdu 610065, P. R. China
Engineering Research Center of Machine Learning and Industry Intelligence, Ministry of Education, Chengdu 610065, P. R. China

Editors: Berrin Yanıkoğlu and Wray Buntine

Abstract

As the most commonly used quantization techniques for deep neural networks, the int-only quantization methods use scale factor to linearly approximate the weights or activation of each layer. However, when passing activation data between layers, such int-only quantization methods require extra Scale Factor Conversion (SFC) operations, resulting in computational overhead. In this paper, we propose a Group-Wise Quantization framework, called GWQ, to reduce computational consumption during the activation data pass process by allowing multiple layers share one scale factor in SFC operations. Specifically, in the GWQ framework, we propose two algorithms for network layers grouping and model training. For the grouping of network layers, we propose a grouping algorithm based on the similarity of data numerical distribution. Then, the network layers divided into the same group will be quantified using the same common scale factor to reduce the computational consumption. Considering the additional performance loss caused by sharing scale factors among multiple layers, we propose a training algorithm to optimize these shared scale factors and model parameters, by designing a learnable power-of-two scaling parameter for each layer. Extensive experiments demonstrate that the proposed GWQ framework is able to effectively reduce the computational burden during inference, while maintaining model performance with negligible impact.

Keywords: Int-Only Quantization; Scale Factor Conversion; Quantization-Aware-Training; Deep Neural Networks; Grouping-Wise Algorithm

1. Introdution

Benefiting from the rapid development of computing power and storage technology in recent years, deep learning models with a vast number of parameters are designed to handle increasingly complex tasks. However, running these models often consumes significant computational resources. Those expensive costs hinder the deployment of deep learning in industry, agriculture, and vehicle areas. How to design high-efficiency deep learning

* corresponding authors

models that satisfy hardware limitations has become the most critical step in deployment. Current model compression techniques are mainly classified into quantization (Zafrir et al., 2019), pruning (Hoefler et al., 2021; Vadera and Ameen, 2022), Neural Architecture Search (NAS) (Sun et al., 2019), and knowledge distillation (Gou et al., 2021; Menghani, 2021).

Quantization is a model compression technique tightly integrated with hardware by compressing the precision stored in the weight and the data representation (e.g., from 32-bit floating-point numbers to 4-bit integers). Quantized models can be computed in Digital Signal Processor (DSP), Field Programmable Gate Arrays (FPGA) as well as other hardware platforms optimized for integer operations (Qiu et al., 2016; Courbariaux et al., 2016). Compared to the full precision model, the quantized model is faster and consumes less energy during inference.

The most common quantization technique currently used is the int-only quantization (Gholami et al., 2021). Int-only quantization computes a scale factor vector for the entire layer, and then affinely maps the float-point data to a finite range of integers (Jacob et al., 2018; Nagel et al., 2021). The mapped values can be accelerated in hardware. However, this approach requires an SFC operation for each layer to be designed in order to transfer activations between layers. SFC operation involves element-wise INT32 multiplication and bit-shifting, resulting in significant additional computational overhead.

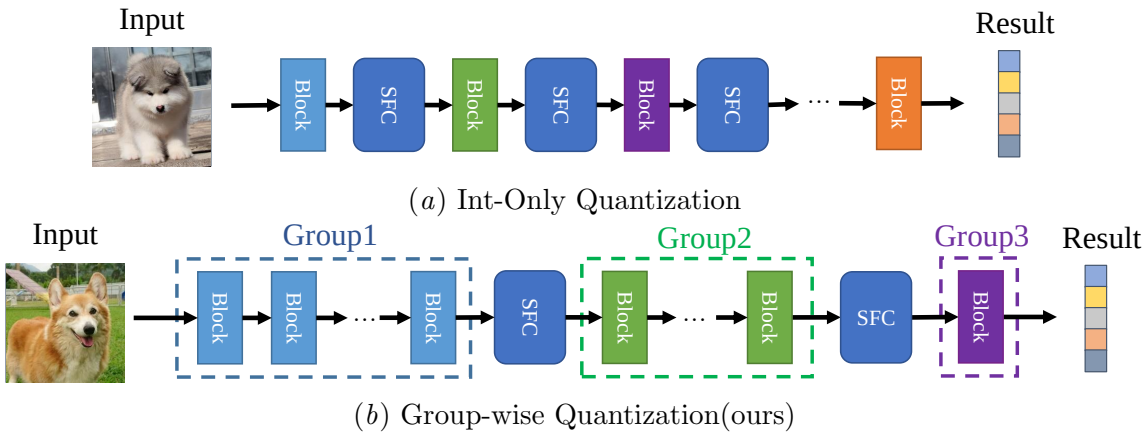
The computational cost of model inference can be reduced by reducing the frequency of SFC usage. To achieve this, our key idea is to group the layers in a network, and multiple layers that are assigned to the same group share a common scale factor. Thus the frequency of SFC operations will be reduced and the computational complexity is reduced. In this paper, we propose a Group-Wise Quantization framework, called GWQ. The GWQ framework allows multiple layers share one scale factor, thus reducing the frequency of SFC operation usage.

It is worth noting that in the proposed GWQ framework, there are three issues that need to be addressed, i.e., 1) how to group the network layers? 2) how to choose the number of groupings? 3) how to training the model of group-wise quantization is better? For the first problem, we propose a grouping algorithm for layers based on similarity evaluation of data numerical distribution. For the second problem, we propose an algorithm for determining the optimal number of groupings using the elbow method. The combination of these two algorithms can provide a reference optimal grouping plan for a particular network. For the third problem, we provide a training method to optimize these shared scale factors as well as the model parameters. We design a learnable power-of-two scaling parameter for each layer. This parameter can reduce the discrepancy in quantized values due to the shared scale factor in one group. Our method can improve the convergence speed and stability for group quantization training without introducing additional multiplication operations. Our contributions can be summarized as follows:

1. We extend the granularity of int-only quantization from the traditional layer-wise or channel-wise to group-wise. The reduced granularity of quantization reduces the additional computational overhead due to SFC operations. We also demonstrate through experiments that well-designed group quantization has almost no effect on the performance of the model.

2. We propose a novel method to evaluate the similarity of the numerical distribution between two network layer weights or activations. In addition, we propose a method for the determination of the optimal number of groups based on the elbow method. These two methods can provide a reference grouping plan in group-wise quantization.
3. We propose a novel training method for grouping quantization. Since multiple layers in a group share a common scale factor, this leads to a reduction in training stability. We design a learnable power-of-2 scaling parameter for each layer. This method can increase the speed of model training convergence and stability without introducing additional multiplication computations.

2. Related Works



(*a*) Int-Only Quantization

(*b*) Group-wise Quantization(ours)

Figure 1: Comparison of two different quantization methods in inference process. *Block* is layer with weights, e.g., fully-connected layer, convolutional layer, block connected by skip connections from multiple layers. *SFC* is Scale Factor Conversion.

Retrain or not Retrain Depending on whether an additional retraining step is required, current quantization methods can be categorized into Post-Training Quantization (PTQ) and Quantization-Aware training (QAT) (Gholami et al., 2021). PTQ is directly quantize a trained model without retraining (Chikin and Antiukh, 2022; Nagel et al., 2019). The research of PTQ focuses on how to adjust the distribution of weights and activation values in the trained model so that the model can better tolerate the numerical errors caused by quantization. In contrast, QAT will Fine-tune the quantized model using Straight Through Estimator (STE) (Bengio et al., 2013) Or other gradient estimators (J. Lee, 2021). In most cases, QAT will obtain a higher performance than PTQ because of the extra fine-tuning procedure. Our work belongs to the category of QAT.

Int-only Quantization Int-only quantification is the most widely used quantization method today. Part of the recent work in int-only quantization has focused on how to obtain better Scale Factor parameters. For example, LSQ (Esser et al., 2019), TQT (Jain et al., 2020), CSQ (Asim et al.), these methods obtain performance close to that of a full precision model at 4bits by considering the Scale factor in integer-only as a trainable parameter in the

QAT process, so that the Scale Factor and model parameters can be trained simultaneously by gradient descent methods. Other works, Alizadeh et al. (2020) and Han et al. (2021) improve the performance of the quantized model by introducing additional constraints in QAT to make the model weights more adaptive to the adverse effects of quantization errors. There is also work on introducing new activation functions, such as PACT (Choi et al., 2018), where the authors add a maximum pass-through threshold to the Rectified Linear Unit (ReLU) activation function, which reduces the range of values represented and effectively improves the performance of the low bit-width model. However, an unavoidable disadvantage of aforementioned int-only quantization technologies is that it involves the SFC operations for each layers, which results in extra energy cost and higher latency. Our work is dedicated to eliminating unnecessary SFC operations.

3. Methodology

In this section, we first review the basic process of common neural network inference and describe our motivation. Second, we detail the three methods included in our proposed GWQ. Finally, we will detail the operational steps of GWQ.

3.1. Preliminaries

The in-layer operations of common DNN can be summarized as a series of matrix addition and multiplication, for example, the operations of a fully connected layer can be written as:

$$\alpha = \sigma(wx + b), \tag{1}$$

where w and b represent the weight and bias respectively, σ is an element-wise nonlinear activation function (e.g., ReLU), x represents the input data, and y represents the output result. The step wx, which occupies the main computing time in the inference phase, can be expanded as:

$$y^{i,k} = \sum_{j=1}^{N} w^{i,j} x^{j,i}, \tag{2}$$

where y represents the intermediate result of matrix multiplication and wx has a time complexity of $O(n^3)$. which means that a large number of floating-point multiplication and addition operations are invoked. The quantization technique strives to transform this process into a low bit-width integer operation.

The process of integer-only quantization can be expressed mathematically as shown in Equation 3, wherein the weights w or inputs x are symbolized as r. The quantized value is represented by q, while s and z respectively indicate the scale factor and the zero point. Additionally, the subscripts min and max specify the minimum and maximum values that the parameter is required to represent.

$$\begin{cases} q &= clip\left(round(r/S), q_{min}, q_{max}\right), \\ S &= (r_{max} - r_{min})/(q_{max} - q_{min}), \\ z &= round(q_{max} - r_{max}/S). \end{cases} \tag{3}$$

Transforming the input x and weight w into quantized values q_x and q_w, respectively, through Equation 3. The computation process of the quantized value q_y of output y can be expressed as follows:

$$q_y^{i,k} = \frac{S_x S_w}{S_y} \left[\sum_{j=1}^{N} q_x^{i,j} q_w^{j,k} - \sum_{j=1}^{N} q_x^{i,j} z_w - \sum_{j=1}^{N} q_w^{j,k} z_x + \sum_{j=1}^{N} z_x z_w \right] + z_y. \tag{4}$$

Symmetric quantization is commonly used in engineering to reduce the complexity of inference hardware, by setting the values of z_x, z_w, and z_y to 0. As a result, the computation process of the quantized value q_y can be simplified as follows:

$$q_y^{i,k} = \frac{S_x S_w}{S_y} \left[\sum_{j=1}^{N} q_x^{i,j} q_w^{j,k} \right]. \tag{5}$$

In Equation 5, all operations within the parentheses can be performed on low bit-width integers. The step of $S_x S_w / S_y$ can be decomposed into a 32-bit fixed-point multiplication followed by an integer shift operation, as shown in Equation 6, where M represents a 32-bit fixed-point number and 2^d represents the shift operation with d denoting the bit-shift step. The bit-shift operation consumes minimal computational resources, but the fixed-point multiplication involved is an element-wise FP32 operation, which greatly slows down the inference speed. We refer to this operation as the Scale Factor Conversion (SFC).

$$S_x S_w / S_y = M \cdot 2^d. \tag{6}$$

Our motivation and goal is to eliminate unnecessary SFC operations from the computational process and further accelerate quantized inference. The difference in quantization methods during inference can be seen in Figure 1(a) and Figure 1(b).

3.2. Block similarity evaluation and grouping plan generation

Our first task is to group the connected blocks that have similar weight or activation value distributions. A variety of previous research has pointed out that the numerical distribution of the quantized model data and the numerical distribution of the full-precision model data is strongly correlated (Jin et al., 2022; Hou et al., 2019; Zhou et al., 2017). It means that the more similar the numerical distribution of two blocks in the full-precision model is, the less it affects the numerical representation of both blocks even if they use the same scale factor in the quantization. We directly used the weights of the full-precision model and the activation values as the basis for the evaluation of our grouping algorithm.

To quantify the similarity of weight values for each block, we can assume that the weights of any two blocks are represented as sets W_1 and W_2. Based on previous research (Huang et al., 2021), W_1 and W_2 can be treated as two approximate Gaussian distributions. Therefore, we can use the Gaussian distribution representation of the numerical values of the two blocks to calculate the KL-Divergence (KLD). The magnitude of KLD can measure whether two blocks should be grouped together. The formula for computing the similarity of weight value distributions between two layers is:

$$\mathrm{KLD}(B_i \| B_j) = \log \frac{\sigma_j}{\sigma_i} + \frac{\sigma_i^2 + (\mu_i - \mu_j)^2}{2\sigma_j^2} - \frac{1}{2}, \tag{7}$$

where σ_i, σ_j, μ_i^2, and μ_j^2 represent the mean and variance of the weight or activation values of the i-th and j-th blocks, respectively. It should be noted that the values of i and j should be adjacent, i.e., $j = i + 1$ or $j = i - 1$. As the KLD is asymmetric, i.e., $KLD(B_i|B_j) \neq KLD(B_j|B_i)$, where $i \neq j$, this results in some unnecessary recalculations in the subsequent steps. Therefore, we can replace KLD with symmetric JS-Divergence (JSD). Its formula is expressed as:

$$\mathrm{JSD}_{pair}(B_i \| B_j) = \frac{1}{2}(\mathrm{KLD}(B_i \| B_j) + \mathrm{KLD}(B_j \| B_i)). \tag{8}$$

This formula can be extended to evaluate the similarity among N blocks:

$$\mathrm{JSD}_{group}(B_1, B_2, \cdots, B_n) = \begin{cases} \sum_{i=1}^{N-1} \mathrm{JSD}_{pair}(B_i \| B_{i+1}), & N >= 2, \\ 0, & N < 2. \end{cases} \tag{9}$$

A smaller value of the formula indicates a higher degree of similarity in weight or activation value distribution among the layers within a group. The grouping plan for a neural network layer consists of multiple groups. To obtain an overall evaluation score for the plan, the JSD of each group is calculated separately and then summed as follows:

$$\text{Group Score}(T) = \sum_{i=1}^{G} \mathrm{JSD}_{group}(T_i), \tag{10}$$

where T is the overall plan, T_i is the i -th group in T, which contains several adjacent blocks. The parameter G represents the number of groups. With the above tools, we can evaluate different sub-graph partitioning results given the number of groups. Figure 2 is an example of how to determine the optimal grouping plan by calculating the group scores.

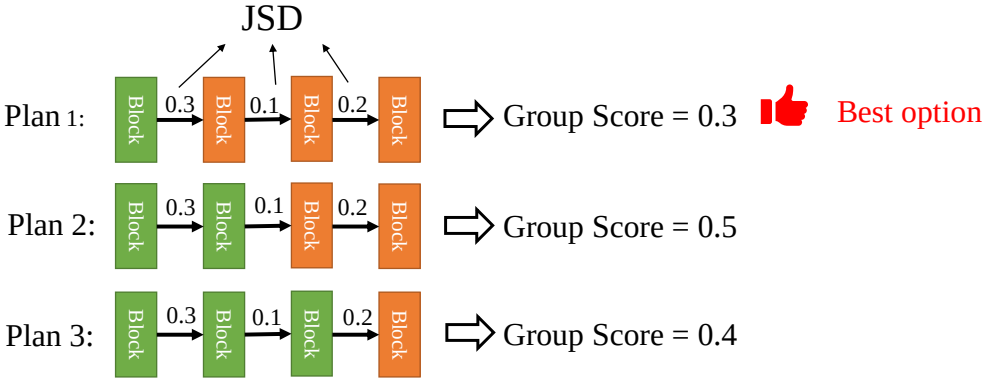


Figure 2: Example of grouping strategy: In a neural network with 4 blocks, setting the number of groups $G = 2$. There are three possible grouping results: Plan1 has the lowest Group Score, thus it is selected as the grouping plan for $G = 2$.

Algorithm 1 Generate optimal Group Plans

Input: the number of layers N and the number of groups G

Output: A list representing the optimal group plan and the corresponding Group Score.

```
// Assign Block IDs from 0 to N-1.  For instance, if there are 4 Blocks, they will be
encoded as 0 to 3.  Suppose Block 0 and Block 1 are grouped together, as well as Block 2
and Block 3, then the final output result will be [[0,1],[2,3]].
group_plans ← [];
stack ← [([x | x ∈ range(N)], G, [])];
while stack is not empty do
    l, g, pre ← stack.pop();
    if g == 1 then
        pre.append(l);
        group_plans.append(pre);
    end
    else
        for i ← 1 to len(l) − 1 do
            pre.append(l[: i]);
            if len(l) ≥ 2 then
                stack.append((l[i :], G − 1, pre[:]));
            end
            pre.pop();
        end
    end
end
// Compute the Group Score for each group plan using Equation 10 and identify the minimum
value among them.
group_scores ← GroupScore(group_plans);
i ← argmin(group_scores);
return group_plans[i], group_scores[i];
```

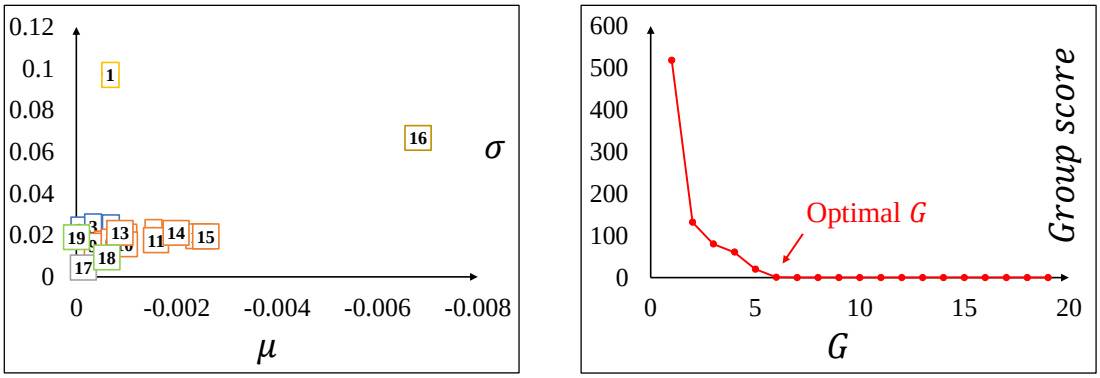
The grouping plan can be generated by a Depth-First-Search (DFS) algorithm, as shown in Algorithm 1. All the grouping plans are evaluated by Equation 10. The optimal grouping plan is the one with the smallest group score.

3.3. Optimal number of groups

By the method introduced in the previous section, we can obtain the optimal grouping result under the condition of manually setting the parameter G. However, how to choose a better parameter G is a noteworthy issue. If G is set too small, the gradients of layers with many parameters will dominate the scale factor during the tuning process, making it difficult to train, and the network's accuracy cannot be guaranteed due to large differences between different blocks being forced to use the same scale factor. When G is too large, for example, when G is set to be equal to the number of layers in the network, each group contains only one layer, and GWQ degenerates into Int-only quantization, facing the problem of a significant increase in computational complexity.

When using K-means clustering, the number of clusters, K, needs to be specified manually. The elbow method is a technique used to determine the optimal number of clusters in a K-means (Sammouda et al., 2021; Nainggolan et al., 2019; Syakur et al., 2018). This method works by plotting the Sum of the Squared Error (SSE) against the number of clusters and identifying the "elbow" point, which is the point of inflection in the curve. The idea is that the elbow point represents the point at which the addition of more clusters does not significantly improve the model's performance.

Our problem is similar to the K-means described above. So, we can use the group score in Equation 10 as a metric for evaluating the within-group similarity instead of SSE. As the value of G increases, the group score will continuously decrease, and when it decreases to a certain value, the rate of decrease will slow down and eventually converge to 0. The curve drawn by the elbow method can guide us in selecting an appropriate value for parameter G. As shown in Figure 3(a), we first plot the mean and variance of each layer's parameters of a trained VGG-19BN network. We then use our proposed group score as the distance metric replace for SSE and plot the resulting group scores under multiple values of G. As illustrated in Figure 3(b), we can find the elbow point in the chart, which is G=6.



(a) The distribution of each layer in a trained VGG19_BN network.

(b) The group score curve of VGG19_BN network for different numbers of groups.

Figure 3: A toy experiment: determining the value of G in GWQ using the elbow method.

In practice, we use the above mentioned grouping method and the optimal number of groups determination algorithm to calculate the optimal grouping plan for the weights and activation separately. The final grouping plan we use for retraining is the intersection of these two optimal grouping plans.

3.4. Retrain Method

The quantization process can lead to imprecise numerical representation, resulting in significant performance drop for models quantized to lower bit-widths. QAT is a common approach to restore model performance. In QAT, the process described in Equation 3 is introduced into forward propagation. Since the numerical perturbations introduced by the quantization process directly affect the magnitude of the loss, we can use the gradient descent algorithm to optimize the parameters of the original model to adapt to the impact

of the quantization process. However, a problem arises in that the *round* function cannot provide effective gradient information for the backward propagation process.

STE (Bengio et al., 2013) is one solution to address the aforementioned issue. It set the derivative of the *round* to 1:

$$\frac{\partial \text{round(x)}}{\partial \text{x}} = 1. \tag{11}$$

This approach enables the gradient to be directly propagated to the FP32 weights during back-propagation, facilitating the training of quantized weights and activations. Although the modified gradient deviates from the true gradient, the practical success of this method demonstrates its effectiveness.

Our proposed GWQ framework can be combined with many QAT methods. In this paper, we have selected Learned Step Size Quantization (LSQ) (Esser et al., 2019), one of the currently most successful methods, as our optimization method.

The LSQ involves incorporating the scale factor S into the training optimization process, instead of computing it based on the statistical values of the data. In LSQ, the scale factor is represented by a learnable scalar parameter and dynamically adjusted during training to fit the data distribution. The gradient calculation process of S can be represented by the following equation:

$$\frac{\partial \hat{r}}{\partial S} = \begin{cases} -r/S + round(r/S), & \text{if } q_{min} < r/S < q_{max}, \\ q_{min}, & \text{if } r/S \leq q_{min}, \\ q_{max}, & \text{if } r/S \geq q_{max}. \end{cases} \tag{12}$$

To ensure the stability of the optimization process, the author scaled the gradient of S based on the number of elements in terms of weight or activation value. The gradient scaling factor g is given by:

$$g = 1/\sqrt{N_w q_{max}}, \tag{13}$$

N_w represents the number of elements in terms of data that need to be quantize.

We found that the training performance was not satisfactory when we used LSQ training directly. Our analysis is attributed to the fact that there is still a small unavoidable difference in the distribution of values between blocks in the same group. This difference, although small, still affects the stability of the training.

To address this issue, we introduce an additional power-of-2 scaling parameter L to LSQ. This parameter provides independent variability to each layer, thereby increasing the stability of the training process. We modify the S parameter in Equation 12 to the following form:

$$\begin{aligned} S &= L \times \hat{S}, \\ L &= sign(\hat{L}) \times 2^{round(\log_2 abs(\hat{L}))}, \end{aligned} \tag{14}$$

where $\hat{S}$ serves as the scale factor for each group, L is the scaling parameter for each layer, and $\hat{L}$ is a floating-point number. L is the most approximate power-of-2 value of $\hat{L}$. In the inference phase, the computation of L is replaced with a cheap shift operation. We term this approach Bit-shift based Layer Adaption (BLA).

3.5. Group-wise Quantization Framework

Combining the methods described in the previous sections, we can obtain a complete framework for group-wise quantization. The running process of this framework is shown in Figure 4.

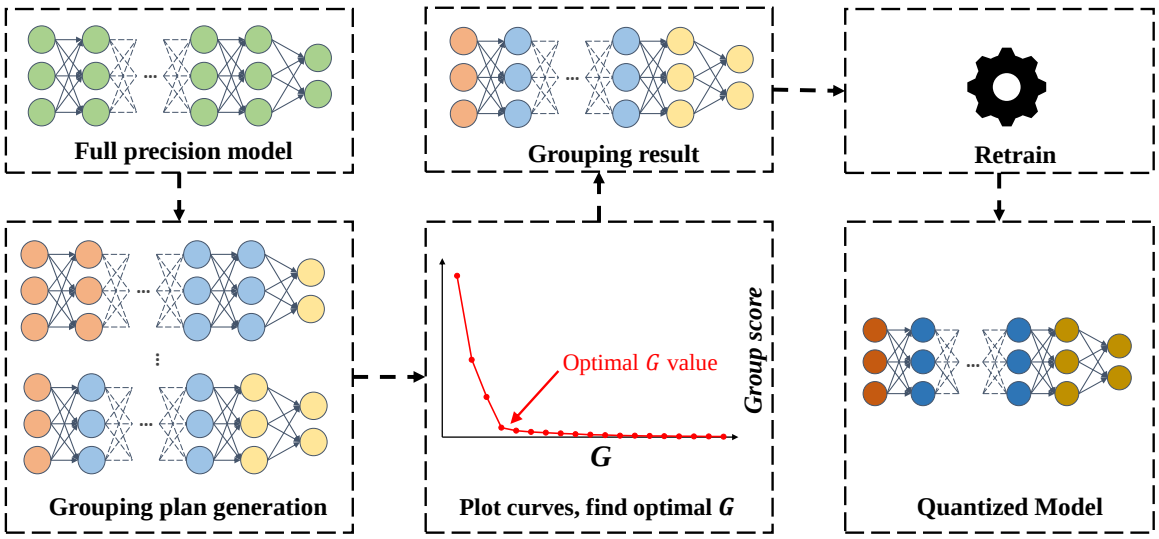


Figure 4: GWQ framework schematic diagram

The steps for running our framework are as follows:

1. The user submits a pre-trained full-precision neural network model and specifies the desired quantization precision. The precision should be determined according to the actual hardware resource requirements.
2. The optimal grouping plan and its corresponding grouping score are calculated for each parameter G using Algorithm 1.
3. The group score curve for each parameter G is plotted, and the optimal grouping size is determined using the elbow method. The reference optimal grouping plan generated in step 2. The user can directly adopt this grouping plan or manually assign a grouping plan.
4. The model is retrained using the BLA method and finally deployed to the inference device.

4. Experiment

In this section, we extensively evaluated the performance of GWQ under multiple experimental settings.

4.1. Experimental Setting

Datasets: The datasets used in our experiments are ImageNet (ILSVRC12) (Russakovsky et al., 2015) and Cifar-100 (Krizhevsky et al., 2009). The ImageNet is a large-scale image classification dataset, consisting of a training set with one million images and a validation set with 50,000 images. The Cifar-100 is a relatively small-scale image classification dataset, consisting of 100 categories, each containing 500 32×32 pixel color images. Among them, 50,000 images are used for training, and 10,000 images are used for testing.

Network Structures: We conducted tests on six commonly used network structures, including the ResNet series (ResNet18, ResNet34, ResNet50) (He et al., 2016), the VGG series (VGG13_BN, VGG16_BN) (Simonyan and Zisserman, 2015), and the MobileNet series with MobileNetV2 (Sandler et al., 2019).

Evaluation Metric: We employ the classification TOP-1 accuracy of the model as the primary evaluation metric. In addition, the size and computational complexity of the model are equally important indicators we pay close attention to. For measuring the computational complexity, we use the number of BOPs (Bit operations)(Baskin et al., 2019) required for one-time inference with $batch\_size = 1$ as our primary indicator. The formula for BOPs can be generalized as the product of the number of computations and the bit width of the parameters involved in the computation. For instance, the formula for calculating BOPs in a convolutional layer is expressed as follows:

$$\text{BOPs} \approx F_h F_w m n k^2 b_{\text{a}} b_{\text{w}}, \tag{15}$$

where F_h and F_w are the height and width of the output feature map, m and n are the number of input and output channels, k is the kernel size, and b_a and b_w are the bit width of input and weight. It should be noted that we ignore the impact of shift operations as they have a negligible effect compared to multiplication and accumulation operations.

4.2. Performance and Computational Complexity

We conducted comparative experiments on performance and computational complexity using the ImageNet dataset across the all six network architectures. We train the full-precision model ourselves from scratch, using Stochastic Gradient Descent (SGD) optimizer, with 0.9 momentum, cosine learning rate decay (Loshchilov and Hutter, 2016) without restarts, and the initial learning rate of 0.1 and $10 - 4$ weight decay. In the retraining process, we use the same setup except the learning rate is modified to 0.01. The models are retrained for 90 epochs. The previous state-of-the art int-only quantization methods we compared are: PACT (Choi et al., 2018), LQ-Net (Zhang et al., 2018), LSQ (Esser et al., 2019), CSQ (Asim et al.). For a fair comparison, same as LSQ, we set the precision of the first and last layers to 8 bits. All our code is implemented using PyTorch [1].

Table 1 shows the comparison of our proposed method with a variety of mainstream int-only quantization methods for Top1-accuracy with both weights and activation values quantized to 4 bits. Our method can meet or exceed the current state-of-the-art int-only quantization methods. We calculate the computational complexity of the int-only quantization and

1. https://pytorch.org/

Table 1: A comparison of ImageNet Top-1 accuracy (%) classification performance for 4-bit quantization (both weights and activation). Marked with ★ are int-only quantization methods. The baseline models are 32-bit full precision.

Network	Baseline	PACT★	LQ-Nets★	LSQ★	CSQ★	GWQ
ResNet-18	70.58	69.20	69.30	70.52	**70.53**	70.52
ResNet-34	73.31	-	-	73.12	73.01	**73.27**
ResNet-50	76.12	74.21	74.72	75.83	-	**75.84**
VGG13_BN	71.58	68.70	69.12	**69.56**	-	**69.56**
VGG16_BN	73.37	71.58	71.60	72.40	-	**72.41**
MobileNet-v2	71.88	-	-	66.82	**66.98**	65.79

Table 2: The computational complexity (GBOPs) comparison of the int-only method and the GWQ method using different bit widths for quantization on ImageNet.

Network	Int-only (4bits)	GWQ (4bits)	Int-only (8bits)	GWQ (8bits)
ResNet-18	31.57	**29.03 (-8.06%)**	118.65	**116.10 (-2.14%)**
ResNet-34	62.45	**58.62 (-6.13%)**	238.31	**234.48 (-1.16%)**
ResNet-50	76.81	**65.43 (-14.82%)**	273.09	**261.71 (-4.17%)**
VGG13_BN	193.48	**180.94 (-6.48%)**	736.29	**723.74 (-1.70%)**
VGG16_BN	261.41	**247.52 (-5.31%)**	1003.98	**990.10 (-1.38%)**
MobileNet-V2	11.32	**4.42 (-60.42%)**	24.76	**17.92 (-27.62%)**

our proposed method for different network structures in Table 2. Combining Table 1 and Table 2, we can observe that our method is not the best compared to the SOTA quantization methods, but our method outperforms the current int-only quantization in terms of computational complexity across the board. In particular, our method on MobileNet-v2 is almost 2 percentage points lower in performance compared to LSQ (Esser et al., 2019) and CSQ (Asim et al.) quantization methods, but our computational complexity is more than 60 percentage points lower compared to these two methods. In a real scenario, we believe this trade-off is worthwhile.

4.3. Ablation Study

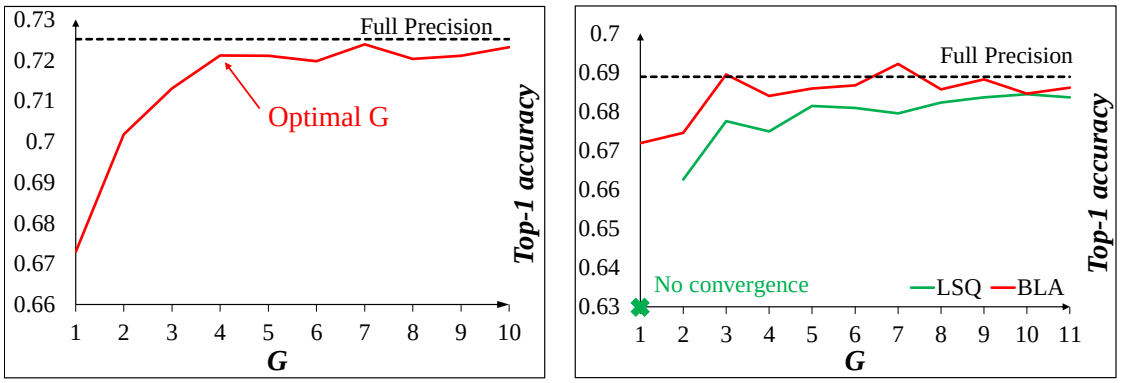
To validate the effectiveness of the group quantization strategy, we conducted experiments on group quantization using the Cifar-100 dataset on the ResNet-18 network. Furthermore, to demonstrate the effectiveness of our proposed training method, we performed stability experiments on training using LSQ and BLA on the VGG13_BN network with the same dataset. In both of these experiments, we used the recipe from this repository [2] to train the FP32 model for these two networks. We use the same hyperparameter settings as for the ImageNet experiments, except that the retrain epoch is decreased to 30.

In the first experiment, we tested the performance of the 4-bit quantized ResNet-18 model after retraining by manually set different number of groups. As shown in Figure 5(a), we

2. https://github.com/weiaicunzai/pytorch-cifar100

observe that when all layers of the entire network share one scale factor, i.e., setting $G = 1$, the performance of the quantized model decreases significantly even after retraining. As the number of groupings increases, the performance of the retrained quantized model gradually increases. When G increases to 4, the performance of the quantized model is close to the performance of the full precision model. When the number of groups is larger than 4, the performance does not improve greatly. This inflection point coincides with the optimal number of groups determined by the grouping algorithm (shown by the red arrow in the figure).

In the second experiment, we tested the performance comparison between using the LSQ method and our proposed BLA method under several G settings. The results are shown in Figure 5(b). We can observe that BLA has more stable training convergence compared to LSQ. For example, when setting $G = 1$, LSQ is unable to converge, while BLA shows better training stability. In addition, we can find that the performance of the model retrained by the BLA method is superior across the board compared to LSQ.



(a) The Top-1 accuracy of the ResNet18 model under different G settings after retraining.

(b) The Top-1 accuracy comparison of the VGG13_BN model under different G settings using LSQ and BLA retraining methods respectively.

Figure 5: Ablation experiment in Cifar-100

5. Conclusion

In this paper, we propose a group-wise quantization framework and demonstrate the effectiveness of the proposed framework through comprehensive experiments. Our work shows that the use of the same quantization scale factor between neighboring neural network layers with similar numerical distributions does not significantly affect the performance of the network, which means that the balance between computational cost and model performance can be achieved by controlling the granularity of quantization reasonably. Our study gives more opportunities for the research of model quantization techniques.

Acknowledgement

This work is supported by the National Science Foundation of China under Grant 62106161, the Key Program of the National Science Foundation of China under Grant 61836006, the Key R&D Program of Sichuan Province under Grant 2022YFN0017, and the Key Project of Science and Technology Department of Sichuan Province under Grant 2023YFG0278·

References

Milad Alizadeh, Arash Behboodi, Mart van Baalen, Christos Louizos, Tijmen Blankevoort, and Max Welling. Gradient ℓ_1 regularization for quantization robustness. *arXiv preprint arXiv:2002.07520*, 2020.

Faaiz Asim, Jaewoo Park, Azat Azamat, and Jongeun Lee. Csq: Centered symmetric quantization for extremely low bit neural networks.

Chaim Baskin, Natan Liss, Eli Schwartz, Evgenii Zheltonozhskii, Raja Giryes, Alex M. Bronstein, and Avi Mendelson. UNIQ. *ACM Transactions on Computer Systems*, 37(1-4):1–15, nov 2019. doi: 10.1145/3444943. URL https://doi.org/10.1145%2F3444943.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

Vladimir Chikin and Mikhail Antiukh. Data-free network compression via parametric non-uniform mixed precision quantization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 450–459, 2022.

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.

Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

Steven K Esser, Jeffrey L McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S Modha. Learned step size quantization. *arXiv preprint arXiv:1902.08153*, 2019.

Amir Gholami, Sehoon Kim, Zhen Dong, Zhewei Yao, Michael W Mahoney, and Kurt Keutzer. A survey of quantization methods for efficient neural network inference. *arXiv preprint arXiv:2103.13630*, 2021.

Jianping Gou, Baosheng Yu, Stephen J Maybank, and Dacheng Tao. Knowledge distillation: A survey. *International Journal of Computer Vision*, 129:1789–1819, 2021.

Tiantian Han, Dong Li, Ji Liu, Lu Tian, and Yi Shan. Improving low-precision network quantization via bin regularization. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5261–5270, 2021.

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.

Torsten Hoefler, Dan Alistarh, Tal Ben-Nun, Nikoli Dryden, and Alexandra Peste. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *The Journal of Machine Learning Research*, 22(1):10882–11005, 2021.

Lu Hou, Ruiliang Zhang, and James T Kwok. Analysis of quantized models. In *International Conference on Learning Representations*, 2019.

Zhongzhan Huang, Wenqi Shao, Xinjiang Wang, Liang Lin, and Ping Luo. Rethinking the pruning criteria for convolutional neural network. *Advances in Neural Information Processing Systems*, 34:16305–16318, 2021.

B. Ham J. Lee, D. Kim. Network quantization with element-wise gradient scaling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021.

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.

Sambhav Jain, Albert Gural, Michael Wu, and Chris Dick. Trained quantization thresholds for accurate and efficient fixed-point inference of deep neural networks. *Proceedings of Machine Learning and Systems*, 2:112–128, 2020.

Qing Jin, Jian Ren, Richard Zhuang, Sumant Hanumante, Zhengang Li, Zhiyu Chen, Yanzhi Wang, Kaiyuan Yang, and Sergey Tulyakov. F8net: Fixed-point 8-bit only multiplication for network quantization. *arXiv preprint arXiv:2202.05239*, 2022.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

Gaurav Menghani. Efficient deep learning: A survey on making deep learning models smaller, faster, and better. *arXiv preprint arXiv:2106.08962*, 2021.

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

Markus Nagel, Marios Fournarakis, Rana Ali Amjad, Yelysei Bondarenko, Mart van Baalen, and Tijmen Blankevoort. A white paper on neural network quantization. *arXiv preprint arXiv:2106.08295*, 2021.

Rena Nainggolan, Resianta Perangin-angin, Emma Simarmata, and Astuti Feriani Tarigan. Improved the performance of the k-means cluster using the sum of squared error (sse) optimized by using the elbow method. In *Journal of Physics: Conference Series*, volume 1361, page 012015. IOP Publishing, 2019.

Jiantao Qiu, Jie Wang, Song Yao, Kaiyuan Guo, Boxun Li, Erjin Zhou, Jincheng Yu, Tianqi Tang, Ningyi Xu, Sen Song, et al. Going deeper with embedded fpga platform for convolutional neural network. In *Proceedings of the 2016 ACM/SIGDA international symposium on field-programmable gate arrays*, pages 26–35, 2016.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115:211–252, 2015.

Rachid Sammouda, Ali El-Zaart, et al. An optimized approach for prostate image segmentation using k-means clustering algorithm with elbow method. *Computational Intelligence and Neuroscience*, 2021, 2021.

Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks, 2019.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

Yanan Sun, Bing Xue, Mengjie Zhang, and Gary G Yen. Completely automated cnn architecture design based on blocks. *IEEE transactions on neural networks and learning systems*, 31(4):1242–1254, 2019.

MA Syakur, BK Khotimah, EMS Rochman, and Budi Dwi Satoto. Integration k-means clustering method and elbow method for identification of the best customer profile cluster. In *IOP conference series: materials science and engineering*, volume 336, page 012017. IOP Publishing, 2018.

Sunil Vadera and Salem Ameen. Methods for pruning deep neural networks. *IEEE Access*, 10:63280–63300, 2022.

Ofir Zafrir, Guy Boudoukh, Peter Izsak, and Moshe Wasserblat. Q8bert: Quantized 8bit bert. In *2019 Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing-NeurIPS Edition (EMC2-NIPS)*, pages 36–39. IEEE, 2019.

Dongqing Zhang, Jiaolong Yang, Dongqiangzi Ye, and Gang Hua. Lq-nets: Learned quantization for highly accurate and compact deep neural networks. *CoRR*, abs/1807.10029, 2018. URL http://arxiv.org/abs/1807.10029.

Shu-Chang Zhou, Yu-Zhi Wang, He Wen, Qin-Yao He, and Yu-Heng Zou. Balanced quantization: An effective and efficient approach to quantized neural networks. *Journal of Computer Science and Technology*, 32:667–682, 2017.