INTL JOURNAL OF ELECTRONICS AND TELECOMMUNICATIONS, 2025, VOL. 71, NO. 3, PP. 1–7
Manuscript received January 12, 2025; revised June, 2025. doi: 10.24425/ijet.2025.153629

Smoothed per-tensor weight quantization: a robust solution for neural network deployment

Xin Chang

***Abstract*—This paper introduces a novel method to improve quantization outcomes for per-tensor weight quantization, focusing on enhancing computational efficiency and compatibility with resource-constrained hardware. Addressing the inherent challenges of depth-wise convolutions, the proposed smooth quantization technique redistributes weight magnitude disparities to pre-activation data, thereby equalizing channel-wise weight magnitudes. This adjustment enables more effective application of uniform quantization schemes. Experimental evaluations on the ImageNet classification benchmark demonstrate substantial performance gains across modern architectures and training strategies. The proposed method achieves improved accuracy to per-tensor quantization without noticeable computational overhead, making it a practical solution for edge-device deployments.**

***Keywords*—Per-tensor quantization, edge device, neural network compression**

I. Introduction

QUANTIZATION is a fundamental technique in deep learning, widely used to compress models by reducing the bit-width of weights and activations (also referred to as ”data”). This approach enables faster inference and reduced memory consumption, making it particularly valuable for deploying neural networks on resource-constrained devices. Two of the most commonly employed quantization methods are per-tensor quantization and per-channel quantization.

In *per-tensor quantization*, a single scaling factor s is applied globally to all weights W or activations in a tensor. The quantized weights $\hat{W}$ are computed as:

$$\hat{W} = round\left(\frac{W}{s}\right) \cdot s, \tag{1}$$

where the scaling factor s is defined as

$$s = \frac{\max(|W|)}{2^n - 1}, \tag{2}$$

with n denoting the bit-width. This approach is computationally efficient due to the uniform scaling factor, making it well-suited for hardware implementations. However, it can suffer from precision loss, particularly when the range of values within the tensor varies significantly across channels.

In contrast, *per-channel quantization*, assigns a unique scaling factor s_i to each channel i. allowing finer-grained adaptation to the value distribution within individual channels. The quantized weights $\hat{W}_{i,j}$ for channel i and element j are computed as:

$$\hat{W}_{i,j} = round\left(\frac{W_{i,j}}{s_i}\right) \cdot s_i, \tag{3}$$

with the channel-specific scaling factor s_i defined as:

$$s_i = \frac{\max(|W_i|)}{2^n - 1}. \tag{4}$$

While per-channel quantization improves precision by accounting for the variability in value ranges across channels, it introduces additional computational overhead due to the use of multiple scaling factors and poses greater challenges for hardware implementation.

The distinction between these quantization approaches becomes particularly evident in operations such as depth-wise convolutions. In standard convolutions, each output channel is computed as a weighted sum of all input channels. This design enables cross-channel interactions, allowing the network to combine spatial and cross-channel information effectively. In contrast, depth-wise convolutions take a more channel-isolated approach, where separate kernels are applied independently to each input channel. As a result, each output channel is derived solely from its corresponding input channel, with no cross-channel interaction [1], [2].

The unique challenges posed by depth-wise convolution highlight the limitations of per-tensor quantization and underscore the need for innovative techniques to achieve efficient and accurate quantization in such scenarios. This paper addresses this challenge by introducing a novel smooth quantization approach designed to optimize per-tensor quantization for depth-wise convolution operations.

II. Literature review

Several approaches have been proposed to improve per-tensor weight quantization, focusing on addressing its inherent challenges. One prominent direction involves equalizing the magnitude of weights across channels to mitigate the limitations of global scaling factors in per-tensor operations. Cross-Layer Equalization (CLE) [3] introduces a method to scale the weights within each channel and distribute the magnitudes to neighboring layers. This process equalizes the weight magnitudes across channels, allowing the neighboring layers to absorb the adjustments. However, the applicability

This work was supported by TCL-Research Europe for the experiments.
Author is with Warsaw University of Technology, Poland (e-mail: xin.chang@pw.edu.pl).

of CLE is restricted by its reliance on strict linearity within layers, due to the requirements of the magnitude distributed cross layers linearly, which limits its use to activation functions like ReLU [4] and ReLU6 [5]. Advanced activation functions such as Swish [6] and Leaky ReLU [7] are incompatible with this approach due to their non-linear behavior. Furthermore, while CLE effectively redistributes magnitude across layers, it does not ensure absolute consistency in the magnitudes post-equalization, which can still lead to quantization inefficiencies in certain cases.

Another promising direction is the simultaneous adjustment of model weights and scaling parameters using regularization techniques. For example, Solodskikh et al. [8] propose a method where both weights and scale parameters are optimized under a uniform quantization scheme. The goal is to minimize the quantization error by ensuring that the quantized weights $\hat{W}$ in Equation 1 closely align with the discrete values available at lower precision. While this approach reduces quantization error, it introduces a critical limitation. When the regularization loss approaches zero, the kernel weights converge towards periodic signals, a condition that is nearly impossible to achieve without compromising the original task performance. This inherent trade-off restricts the effectiveness of such regularization-based methods, especially for complex model architectures and diverse datasets.

This paper builds upon the direction established by SmoothQuant [9], leveraging per-channel quantization to shift activation quantization challenges to weight quantization. Conversely, the proposed solution tackles per-tensor quantization by transferring weight quantization challenges to activation quantization, mitigating weight magnitude disparities across channels.

III. Analysis

Quantizing weights in depth-wise convolution presents unique challenges, particularly when working with lower-precision representations 2^B, where B is the bit width, and the kernel values $M \times M \times N$ exceed the available discrete representation levels. The inherent non-periodicity and variability in kernel values ensure quantization errors are unavoidable, leading to potential degradation in model performance.

Figure 1 illustrates the significant magnitude differences across channels in a 64-output channel layer of MobileNetV4-Small [10]. When applying the per-tensor quantization scale derived from Equation 2, the resultant quantization errors exceed the value range of most smaller-magnitude channels, causing a significant drop in task performance introduced by this layer.

A deeper analysis of channel-wise weight magnitudes reveals that Batch Normalization (BN) folding—a widely used optimization technique for quantization—is the primary source of this issue. BN is employed during training to stabilize and accelerate convergence by normalizing activations across batches [11]. During inference, BN folding is applied to improve computational efficiency by embedding the BN parameters (mean, variance, scale, and shift) into the preceding convolutional layer. This process eliminates the need for separate BN operations, reducing latency and memory usage [12].

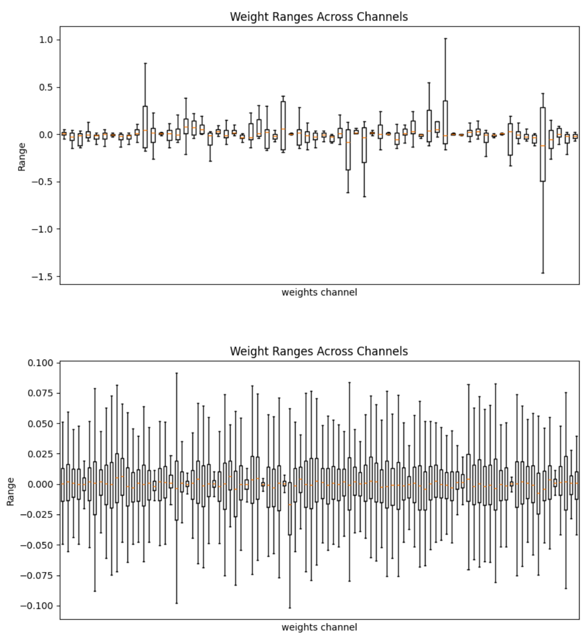

Fig. 1. Per (output) channel weight ranges of a depth-wise convolution layer in MobileNetV4-Small. The top figure shows the weight ranges after batch normalization folding, while the bottom figure shows the weight ranges before batch normalization folding. Each boxplot depicts the minimum, maximum, second quartile, third quartile, and median for each channel. These figures highlight the significant differences in channel weight ranges introduced by batch normalization folding.

While BN folding simplifies the deployment of quantized models in resource-constrained environments, such as edge devices, it introduces unintended side effects. Specifically, the folding process incorporates the scaling and shifting behavior of BN into the convolutional weights and biases, amplifying the channel-wise magnitude differences. This disparity complicates per-tensor quantization, as a single scaling factor fails to account for the wider dynamic range introduced by BN folding.

Mathematically, BN normalizes the activations of a layer using the channel-wise mean μ and variance σ^2, with a small constant ϵ for numerical stability:

$$\hat{x} = \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}}, \tag{5}$$

The normalized activation $\hat{x}$ is then transformed using learnable parameters γ (scale) and β (shift).

$$y = \gamma \hat{x} + \beta. \tag{6}$$

During training, the convolutional layer output is expressed as:

$$y = \gamma \frac{(W * x + b) - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta, \tag{7}$$

where W and b represent the weights and biases of the convolutional layer, and $*$ denotes the convolution operation. BN folding effectively integrates γ, β, μ and σ^2 into W

and b. However, this integration often amplifies discrepancies in weight magnitudes across channels, making global per-tensor quantization unsuitable and necessitating alternative approaches.

After folding, the output of the convolutional layer becomes:

$$W_{folded} = \frac{\gamma W}{\sqrt{\sigma^2 + \epsilon}}, b_{folded} = \frac{\gamma(b - \mu)}{\sqrt{\sigma^2 + \epsilon}} + \beta. \tag{8}$$

The output of the convolutional layer then becomes:

$$y = W_{folded} * x + b_{folded}, \tag{9}$$

effectively combining the effects of both the convolution and the batch normalization into a single operation. This integration eliminates the need for separate normalization steps during inference, reducing computational overhead while preserving the performance benefits of BN.

However, as illustrated in Figure 1, BN folding inadvertently introduces significant magnitude differences across weights in different channels. These disparities exacerbate the challenges of per-tensor quantization by increasing the dynamic range within the tensor.

The quantization error $e(w)$ for a weight $w \in W$ arises due to the rounding operation during quantization. The maximum possible quantization error, e_{max}, is directly proportional to the maximum value M in the weights and inversely proportional to the precision determined by the bit-width B. Mathematically:

$$e_{max} = \frac{M}{2 \cdot (2^{B-1} - 1)} \tag{10}$$

This relationship highlights that larger weight values result in higher quantization errors, underscoring the importance of managing weight magnitudes during quantization effectively.

The challenges associated with quantization become particularly apparent when comparing normal convolution layers to depth-wise convolution layers. Depth-wise convolutions, commonly used in lightweight models like MobileNet, differ fundamentally from normal convolutions in their operation. These differences in computation also affect how weight magnitudes are distributed and, consequently, how quantization errors propagate. To understand the root cause of these challenges, we analyze the forward and backward passes for both types of layers.

Normal Convolution

In standard convolution, each kernel K with weights w_{ij} contributes to all output channels by convolving across multiple input channels.

a) Forward Pass:: The output at location (m, n) in channel o is computed as:

$$Y_{o,m,n} = \sum_{c=1}^{C_{in}} \sum_{i=1}^{k_h} \sum_{j=1}^{k_w} w_{ij,c,o} \cdot X_{c,m+i,n+j}.$$

b) Backward Pass:: The gradient with respect to a single weight w_{ij} is:

$$\frac{\partial L}{\partial w_{ij}} = \sum_{o=1}^{C_{out}} \sum_{m=1}^{H'} \sum_{n=1}^{W'} \delta_{o,m,n} \cdot X_{c,m+i,n+j}, \tag{11}$$

This formulation ensures that gradients are accumulated from all input and output channels, resulting in robust gradient signals during optimization.

Depth-Wise Convolution

In contrast, depth-wise convolution operates differently. Each kernel K_c works independently on a single input channel X_c, producing the corresponding output channel Y_c. In contrast, depth-wise convolution operates differently. Each kernel K_c works independently on a single input channel X_c, producing the corresponding output channel Y_c.

c) Forward Pass:: The output at location (m, n) for input channel c is:

$$Y_{c,m,n} = \sum_{i=1}^{k_h} \sum_{j=1}^{k_w} w_{ij,c} \cdot X_{c,m+i,n+j}.$$

d) Backward Pass:: The gradient with respect to a single weight w_{ij} is:

$$\frac{\partial L}{\partial w_{ij}} = \sum_{m=1}^{H'} \sum_{n=1}^{W'} \delta_{c,m,n} \cdot X_{c,m+i,n+j}. \tag{12}$$

Here, gradients are accumulated only from the corresponding input channel c, leading to a sparser and weaker gradient signal compared to standard convolution.

This difference not only introduces magnitude variations during float32 training but also complicates Quantization-Aware Training (QAT). In standard convolution, the gradients for a single weight are computed by aggregating contributions across all input and output channels. This cross-channel dependency enables a more robust gradient flow, allowing the model to partially recover even when the quantization scale is suboptimal. Conversely, in depth-wise convolution, the gradient for a weight is computed solely within its corresponding single channel, as can be seen from Equation 11 and 12. This lack of cross-channel gradient sharing amplifies the impact of an unsuitable quantization scale, making QAT significantly more challenging for depth-wise convolution layers.

As shown in Figure 2, depth-wise convolution exhibits a broader spread in kernel weight distribution compared to standard convolution. According to Equation 10, this broader distribution leads to higher quantization errors per weight, further emphasizing the challenges associated with quantizing depth-wise convolution layers.

IV. Proposal

Our proposed solution addresses the quantization challenges in weights by redistributing the burden of quantization errors from weights to activations. This is based on the observation that the outputs of a quantized layer are influenced by two quantization processes: weight quantization and activation quantization, as shown in Equation 13.

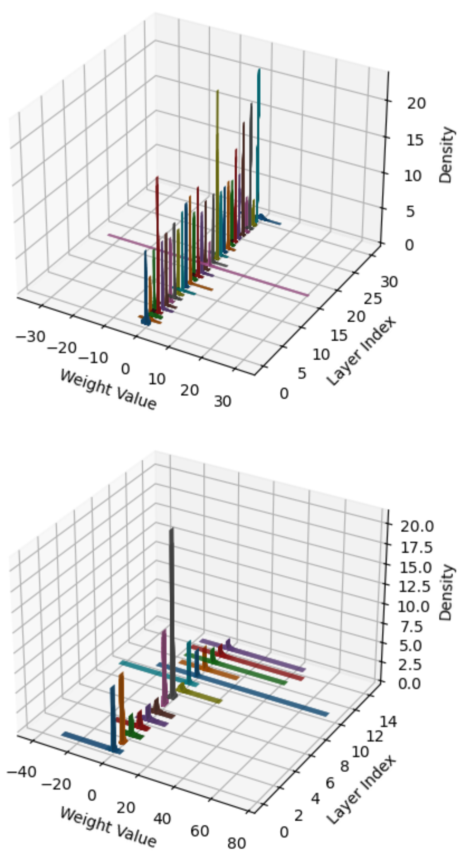

Fig. 2. Histogram of convolution weights for certain layers after BN folding in MobileNetV4-Small. Top: normal convolution layers (36 layers total). Bottom: depth-wise convolution layers (17 layers total).

$$y_q = Clamp\left(\left\lfloor \frac{1}{s_y} \cdot \left(s_w \cdot w_q\right) * \left(s_x \cdot x_q\right) + z_y \right\rfloor, q_{min}, q_{max}\right) \tag{13}$$

Where:

- $w_q = Quantize(w, s_w, z_w)$: Quantized weights.
  $$w_q = Clamp\left(\left\lfloor \frac{w}{s_w} \right\rceil + z_w, q_{min}, q_{max}\right)$$
- $x_q = Quantize(x, s_x, z_x)$: Quantized activations.
  $$x_q = Clamp\left(\left\lfloor \frac{x}{s_x} \right\rceil + z_x, q_{min}, q_{max}\right)$$
- $*$: Represents the convolution or matrix multiplication operation.
- $s_y = s_w \cdot s_x$: The scale factor for the output.
- z_y: The zero-point for the quantized output.
- $Clamp$: Ensures that the result stays within $[q_{min}, q_{max}]$.

To reduce the quantization error in weights, we introduce an additional scaling matrix s_m before the quantization process. This matrix adjusts w_q by normalizing its magnitude and redistributes the scaling to x_q. The updated quantized output equation is defined in Equation 14:

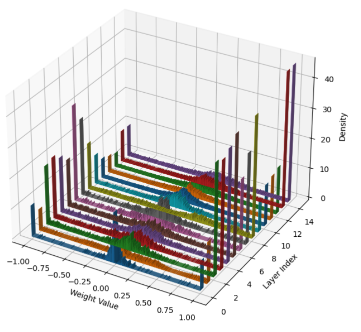

Fig. 3. Histogram of convolution weights for smoothed weight depth-wise convolution layers (17 layers total)

$$y_q = Clamp\Bigg(\left\lfloor \frac{1}{s_y} \cdot \left(s_m \cdot s_w \cdot w_q\right) * \left(s_x \cdot \frac{1}{s_m} \cdot x_q\right) + z_y \right\rceil, q_{min}, q_{max} \tag{14}$$

The scaling matrix s_m redistributes the weight magnitudes across channels to improve quantization accuracy. Specifically, s_m is defined as a diagonal matrix where each element corresponds to the maximum absolute value of weights within a channel:

$$s_m = diag\big(\max(|w_c|)\big), \tag{15}$$

where w_c represents the weights for the c-th channel, and $\max(|w_c|)$ is the maximum absolute value within that channel. We can compare Figure 4 and Figure 5 to see how the weights are normalized after the smoothing. By applying this approach, we migrate the challenges of weight quantization to activation quantization. While this scaling appears to shift the issue linearly, its actual impact is nonlinear.

When quantizing weights w_q, the quantization error introduced is directly propagated to the outputs y_q for any given input x_q. This deterministic behavior means that larger quantization errors in weights consistently result in larger errors in the outputs. In contrast, activation quantization errors are not deterministic, as x_q varies based on the input data. This variability allows for the possibility of reducing overall quantization errors across output samples.

By appropriately scaling the channel magnitudes and combining this approach with effective data clipping solutions, such as those proposed by Nvidia [13], we minimize the occurrence of output samples y_q with significant quantization errors. However, while clipping activations can be effective, applying clipping to weights is not feasible due to their deterministic nature. Clipping weights would introduce a fixed clipping error that propagates consistently to all outputs, exacerbating the quantization issue.

In an ideal scenario, if all quantization errors were confined to weights, the output quantization errors would be deterministic, as weights remain fixed. However, by migrating these errors to activations, we introduce variability, which can reduce the overall quantization error across output samples, even when the maximum possible error due to magnitude differences remains high.

This issue becomes even more critical in the context of Quantization-Aware Training (QAT). When weights have large quantization errors, these errors are deterministic and consistently affect all samples during the training process. As a result, the backpropagation step introduces errors for every training sample, making it challenging for the model to adapt effectively.

In contrast, when the magnitude is migrated linearly to data quantization, the variability of input data allows for a distribution of quantization errors. Some samples will have minimal quantization error, providing cleaner gradients during backpropagation. This variability helps the model adapt to the quantization effects more effectively, as it can leverage the low-error samples to guide training and reduce overall quantization impact.

V. Experiment

To showcase the effect of smoothed weight quantization, we conducted experiments on the ImageNet [14] dataset using several popular edge-device friendly neural network architectures to evaluate its influence on classification accuracy. Additionally, we assessed its impact on processing speed by running measurements on an edge-device-targeted processing chip, the MT9652 from Mediatek. This chip features a 64-bit Arm Cortex-A73 CPU with four cores and a maximum frequency of 1.3 GHz.

TABLE I
Inference speed comparison of the typical and proposed solution.

name	fused BN	smoothed	mean(ms)
mv4s			138.0730
mv4s	X		48.4567
mv4s	X	X	49.9928
mv2			266.3170
mv2	X		83.4738
mv2	X	X	88.1750
mv3l			144.0390
mv3l	X		64.7894
mv3l	X	X	70.8392
mv3s			40.8741
mv3s	X		22.1297
mv3s	X	X	24.0313

Table I illustrates the impact of BN folding and smooth quantization on inference speed. The reported inference times are mean values calculated over 100 iterations for $224\times224\times3$ input data, executed on the device's CPU. It is evident that BN folding plays a critical role, significantly reducing inference time. While the proposed smoothed weight quantization introduces a slight overhead—typically adding only a few milliseconds—this trade-off is justified by the substantial potential improvements in task performance it enables.

Per-channel weight quantization, as described by Equation 4, generally does not pose significant challenges, and many studies, such as Nvidia's work [13], report no accuracy drop, so do we evaluated. However, per-tensor quantization introduces greater difficulties. Table II presents the results of applying standard weight quantization and the proposed smoothed weight quantization method. The results indicate that smoothed quantization consistently achieves better quantization performance across a variety of popular efficient models.

Furthermore, the proposed method, based on Equation 14, only requires linear scaling before activation, making it agnostic to the type of activation function used. This is a notable advantage over methods like CLE, which are restricted to specific activation functions such as ReLU.

TABLE II
Per-tensor weight quantization results with/without smooth weight quantization.

name	smooth	accuracy
mv4s	X	73.39%
mv4s		71.41%
mv2	X	72.08%
mv2		72.02%
mv3l	X	73.77%
mv3l		73.75%
mv3s	X	66.98%
mv3s		65.99%
enet-b0	X	76.73%
enet-b0		74.48%
enet-v2s	X	80.91%
enet-v2s		80.65%

Different training strategies and recipes can significantly influence the performance of the final quantized model. At the same time, they also impact critical factors such as batch normalization statistics and the distribution of weights. As a result, certain model checkpoints may pose greater challenges for quantization, despite their potential to achieve higher float32 performance.

In Table III, we present results using publicly available checkpoints for MobileNetV4-Small and MobileNetV3-Large, sourced from Torch [15] or Timm [16], and trained with different recipes. The results demonstrate that smoothed weight quantization is effective across all training recipes, achieving even better performance when applied to higher-performing float32 model checkpoints.

Upon examining the details of the model checkpoints, we observe in Figure 4 that models trained with different recipes exhibit variations in weight distributions. These differences arise because the magnitude information is carried differently across layers, which can lead to higher quantization errors in certain model checkpoints.

The proposed smoothed weight quantization provides a unified solution for weight quantization. As shown in Figure 5, the same depth-wise convolution layers become almost uniform in their weight distributions after applying smoothed weight quantization.

Additionally, we evaluated full model quantization, incorporating both weight and data quantization. For data quantization, we adopted Nvidia's best data clipping solution [13]. While the approach appears to linearly shift the channel-wise magnitude information (s_w) to data quantization as per

TABLE III
PER-TENSOR WEIGHT QUANTIZATION RESULTS WITH/WITHOUT SMOOTH QUANTIZATION FOR DIFFERENT CHECKPOINTS.

name	quantized	smooth	accuracy
mv4(timm-2400e)			73.52%
mv4(timm-2400e)	X	X	73.39%
mv4(timm-2400e)	X		71.41%
mv4(timm-1200e)			73.36%
mv4(timm-1200e)	X	X	73.04%
mv4(timm-2400e)	X		72.35%
mv3-l(torch-v1)			73.91%
mv3-l(torch-v1)	X	X	73.75%
mv3-l(torch-v1)	X		73.12%
mv3-l(torch-v2)			75.57%
mv3-l(torch-v2)	X	X	75.40%
mv3-l(torch-v2)	X		75.07%
mv3-l(timm)			75.09%
mv3-l(timm)	X	X	75.11%
mv3-l(timm)	X		74.15%

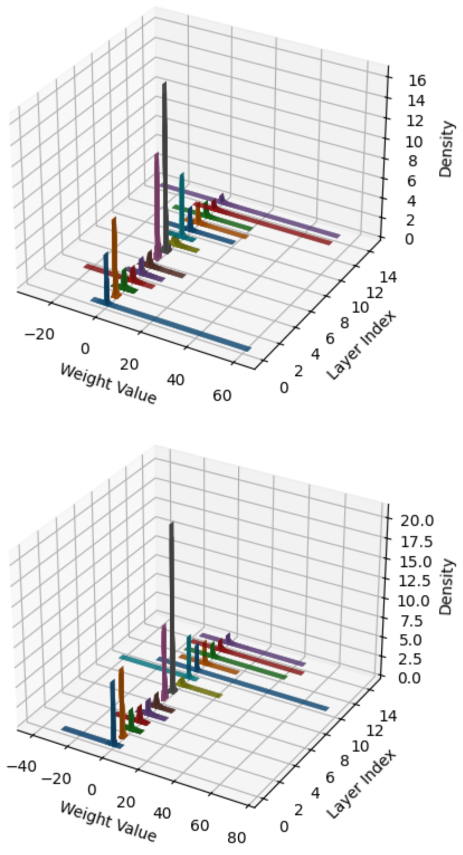

Fig. 4. histogram of convolution weights after batch normalization fusion in MobilenetV4-small. top: normal convolution layers, in total 36. bottom: depth wise convolution layers, in total 17.

Equation 14, the variability of x_q (unlike the fixed values of w_q) allows for better outcomes when combined with data clipping methods. This flexibility of x_q yields improved results compared to native weight quantization combined with data clipping. Moreover, weight clipping is not advisable, as it introduces a fixed clipping error that affects every forward and backward iteration, compounding errors during training.

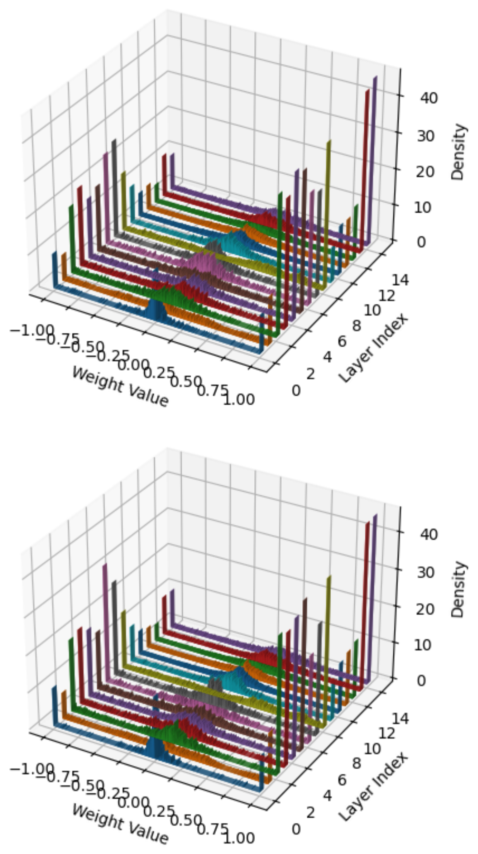

Fig. 5. histogram of convolution weights after batch normalization fusion in MobilenetV4-small. top: normal convolution layers, in total 36. bottom: depth wise convolution layers, in total 17.

The final results are presented in Table IV, showcasing the superior performance of the approach in full model quantization.

TABLE IV
MODEL ACCURACY WITH WEIGHT AND SMOOTH QUANTIZATION FOR THE SAME NETWORK BUT DIFFERENT CHECKPOINTS.

name	quantized	smooth	accuracy
mv4(timm-2400e)			73.52%
mv4(timm-2400e)	X	X	73.05%
mv4(timm-2400e)	X		70.21%
mv4(timm-1200e)			73.36%
mv4(timm-1200e)	X	X	72.84%
mv4(timm-2400e)	X		72.04%
mv3-l(torch-v1)			73.91%
mv3-l(torch-v1)	X	X	73.35%
mv3-l(torch-v1)	X		72.89%
mv3-l(torch-v2)			75.57%
mv3-l(torch-v2)	X	X	75.02%
mv3-l(torch-v2)	X		74.82%
mv3-l(timm)			75.09%
mv3-l(timm)	X	X	74.87%
mv3-l(timm)	X		73.79%
enet-v2s			82.02%
enet-v2s	X	X	81.25%
enet-v2s	X		80.02%

VI. Conclusions

In this paper, we addressed the challenges of weight quantization in deep neural networks, particularly for depth-wise convolution layers, which are integral to efficient models like MobileNet. By introducing the smoothed weight quantization method, we demonstrated its ability to redistribute channel-wise magnitude variations, effectively reducing quantization errors and improving the overall performance of quantized models.

The experiments on the ImageNet dataset and edge-device-targeted hardware, such as the Mediatek MT9652 chip, validated the efficacy of the proposed approach across a variety of popular network architectures. Unlike conventional weight quantization techniques, our method achieves superior quantization performance without being constrained by activation functions, a limitation found in methods like Cross-Layer Equalization.

Furthermore, we showed that smoothed weight quantization works consistently across model checkpoints trained with diverse recipes, providing a unified solution for weight quantization. When combined with effective data quantization techniques, such as Nvidia's data clipping method, the approach delivers improved accuracy for full model quantization. Importantly, it avoids the pitfalls of weight clipping, which introduces deterministic errors that degrade model performance.

In conclusion, the smoothed weight quantization method provides a robust, activation-agnostic, and hardware-friendly solution for tackling the challenges of quantization in modern neural networks, paving the way for more efficient and accurate deployment of deep learning models on resource-constrained devices.

References

[1] S. Yun and A. Wong, "Do all mobilenets quantize poorly? gaining insights into the effect of quantization on depthwise separable convolutional networks through the eyes of multi-scale distributional dynamics," 2021. [Online]. Available: https://arxiv.org/abs/2104.11849

[2] T. Dinh, A. Melnikov, V. Daskalopoulos, and S. Chai, "Subtensor quantization for mobilenets," 2020. [Online]. Available: https://arxiv.org/abs/2011.08009

[3] R. Krishnamoorthi, "Quantizing deep convolutional networks for efficient inference: A whitepaper," *CoRR*, vol. abs/1806.08342, 2018. [Online]. Available: http://arxiv.org/abs/1806.08342

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2012.

[5] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L. Chen, "Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation," *CoRR*, vol. abs/1801.04381, 2018. [Online]. Available: http://arxiv.org/abs/1801.04381

[6] P. Ramachandran, B. Zoph, and Q. V. Le, "Swish: a self-gated activation function," *arXiv: Neural and Evolutionary Computing*, 2017. [Online]. Available: https://api.semanticscholar.org/CorpusID:196158220

[7] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proceedings of the International Conference on Machine Learning*, 2013.

[8] K. Solodskikh, V. Chikin, R. Aydarkhanov, D. Song, I. S. Zhelavskaya, and J. Wei, "Towards accurate network quantization with equivalent smooth regularizer," in *European Conference on Computer Vision*, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:253448431

[9] G. Xiao, J. Lin, M. Seznec, H. Wu, J. Demouth, and S. Han, "Smoothquant: Accurate and efficient post-training quantization for large language models," 2024. [Online]. Available: https://arxiv.org/abs/2211.10438

[10] D. Qin, C. Leichner, M. Delakis, M. Fornoni, S. Luo, F. Yang, W. Wang, C. Banbury, C. Ye, B. Akin, V. Aggarwal, T. Zhu, D. Moro, and A. Howard, "Mobilenetv4 – universal models for the mobile ecosystem," 2024. [Online]. Available: https://arxiv.org/abs/2404.10518

[11] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International Conference on Machine Learning (ICML)*, 2015.

[12] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, and H. Adam, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[13] H. Wu, P. Judd, X. Zhang, M. Isaev, and P. Micikevicius, "Integer quantization for deep learning inference: Principles and empirical evaluation," 2020. [Online]. Available: https://arxiv.org/abs/2004.09602

[14] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255.

[15] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett, Eds., vol. 32. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: https://papers.nips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library

[16] R. Wightman, "Pytorch image models," https://github.com/huggingface/pytorch-image-models, 2021.