RepQ: Generalizing Quantization-Aware Training for Re-Parametrized Architectures

Anastasiia Prutianova¹
auprutyanova@gmail.com
Alexey Zaytsev²
Likzel@gmail.com
Chung-Kuei Lee¹
lee.chung.kuei@hisilicon.com
Fengyu Sun¹
sunfengyu@hisilicon.com
Ivan Koryakovskiy¹
i.koryakovskiy@gmail.com

¹ Huawei Technologies Co. Ltd.
Shanghai, China
² Beijing Institute of Mathematical Sciences and Applications
Beijing, China

Abstract

Existing neural networks are memory-consuming and computationally intensive, making deploying them challenging in resource-constrained environments. However, there are various methods to improve their efficiency. Two such methods are quantization, a well-known approach for network compression, and re-parametrization, an emerging technique designed to improve model performance. Although both techniques have been studied individually, there has been limited research on their simultaneous application. To address this gap, we propose a novel approach called RepQ, which applies quantization to re-parametrized networks. Our method is based on the insight that the test stage weights of an arbitrary re-parametrized layer can be presented as a differentiable function of trainable parameters. We enable quantization-aware training by applying quantization on top of this function. RepQ generalizes well to various re-parametrized models and outperforms the baseline method LSQ quantization scheme in all experiments.

1 Introduction

The number of parameters in Neural Networks (NN) has been growing over the years. This substantial computational complexity precludes the deployment of real-life NN-based applications to resource-constraint devices, e.g., mobile phones. Many research works aim at designing computationally efficient NN. A non-exhaustive list of ideas in this area includes knowledge distillation [24], model pruning [22], matrix factorization [26, 31], neural architecture search [37, 57], quantization [11, 15, 18], and re-parametrization [16, 52]. Here, we focus on re-parametrization and quantization as our main research fields.

Re-parametrization is an emerging technique that recently allowed the training of a plain non-residual VGG [11] model to achieve the remarkable accuracy of 80% on ImageNet [44].
while being faster than ResNet-101 \cite{25}. Moreover, re-parametrization has set a new state-of-the-art (SOTA) in channel pruning \cite{14} and became a part of YOLO-7 \cite{48}. The idea behind re-parametrization is that a neural architecture can be represented in different mathematical forms. Similar to how 2-D vectors on a plane can be defined by their Cartesian or polar coordinates, the same architecture can be expressed using various algebraic representations. An alternative representation helps gradient descent to reach a better local minimum, resulting in improved performance. In practical applications, re-parametrization involves replacing linear layers, such as fully-connected and convolutional layers, with a combination of linear layers. For example, several papers \cite{13, 16, 23, 27, 52} have proposed replacing each convolution with a block consisting of multiple convolutional layers with various kernel sizes, channel numbers, residual connections, and batch normalization layers. This re-parametrization block is used during training but is equivalently converted back into a single convolution during inference. To sum up, boosting model quality without additional computational burden at inference makes re-parametrization an important method for increasing NN efficiency. Efficiency is a feature of another major compression technique, namely quantization. Basic quantization algorithms can achieve more than a 75\% reduction in model size while maintaining a performance comparable to uncompressed models. This makes quantization a key technique for real-life model deployment.

We aim to bring the advances in re-parametrization research to practical applications by providing a suitable approach to quantizing re-parametrized NN. Currently, only two research works consider quantizing re-parametrized NN \cite{11, 17}, and they both target a particular architecture, RepVGG. Although novel, the obtained results show a considerable quality drop due to quantization which eliminates the improvements introduced by the re-parametrization.

To solve the problem of insufficient quality and generalization, we introduce a Quantization-Aware Training (QAT) strategy for re-parametrized NN. The challenge is that the training stage weights differ from the testing stage weights, prohibiting standard QAT application \cite{21}. In particular, applying a regular quantization independently to each convolution of a re-parametrized block prevents its merging during inference without increasing the quantizer bit width. For instance, two sequential convolutions of bit-width two will merge into a single convolution of bit-width four. Instead, we propose to compute the inference stage parameters of a convolution as a differentiable function of the trainable parameters of the re-parametrized block and then apply a pseudo-quantization function on top of it. This way, our RepQ approach enables end-to-end quantization-aware re-parametrized training.

To sum up, our main contributions are the following.

- We are the first to propose a method that allows QAT of models with arbitrary re-parametrization schemes. We provide extensive experiments showing that our RepQ method of joint quantization with re-parametrization leads to consistent quality enhancement on SOTA architectures. In particular, for the first time, we achieved lossless 8-bit quantization of re-parametrized models.

- Batch Normalization (BN) contained in the re-parametrized blocks creates a challenge for differentiable merging and hence for QAT. We show how to compute differentiable weight functions via BN-folding. Lastly, we enhance BN-folding by introducing BN statistics estimation, which reduces the computational complexity in theory and practice.
2 Related works

Re-parametrization. Early re-parametrizations appeared as a result of batch normalization \([29]\) and residual connection \([25]\) research. Inspired by batch normalization, the authors of \([45]\) proposed a weights normalization technique. This technique can be viewed as a re-parametrization that decouples the direction and length of the weights vector by introducing an additional parameter responsible for the weight norm. The authors of DiracNets \([52]\) looked for a way to train deep networks without explicit residual connections. They proposed re-parametrizing a convolution with a convolution combined with a skip connection. It allowed very deep single-branch architectures to reach a decent quality. ACNets \([13]\) and RepVGG \([16]\) brought re-parametrization to a new level by introducing multiple convolutions and batch normalization to re-parametrization blocks, revisiting well-known architectures like VGG \([46]\) and ResNet \([25]\) and significantly enhancing their quality. More advanced \([15, 27]\) re-parametrization strategies tend to use a larger number of convolutions and more diverse branches in their blocks. Besides its extensive application to classification tasks, re-parametrization recently helped to reduce computational burden in other computer vision tasks like object detection \([34, 48]\) and super-resolution \([49, 54]\). Re-parametrization receives theoretical justification in \([3]\); the authors prove that under certain theoretical assumptions in a simple convex problem, re-parametrization leads to faster convergence. The authors of \([17]\) show that in some cases, re-parametrization can be equivalent to regular network training with a certain gradient scaling strategy.

Quantization. Regularly, NN parameters are stored as floating point numbers. Yet, a 32-bit parameter representation is redundant to maintaining the network’s quality. The research field aiming to find the optimal low-bit integer parameters’ representation is known as neural network quantization. Theoretically, quantization involves rounding, leading to zero gradients in the network almost everywhere. QAT algorithms are used to address this issue by simulating quantization in a differentiable manner, allowing the network to adapt for subsequent quantization. For instance, \([5, 20]\) suggested injecting pseudo-quantization noise into full-precision network training. Alternatively, \([10, 21, 33]\) uses a straight-through estimator \([6]\) to approximate the gradients of the discontinuous quantization function on the backward pass. \([20, 22, 43]\) proposes a smooth approximation of this stair-step function. In addition, the listed base methods could be improved by using knowledge distillation \([42]\), progressive quantization \([56]\), stochastic precision \([18]\), Batch Normalization re-estimation \([53]\), additional regularization \([10, 50]\), and various non-uniform quantization extensions \([19, 38, 51]\).

3 Background

Quantization allows for reducing inference time and power consumption by decreasing the precision of matrix multiplication factors in convolutions or fully-connected layers. Quantization can result in quality degradation, so QAT is applied to recover the quality and ensure model resilience. During QAT, the original convolution operation, denoted as \(X \ast W\), with input \(X\) and weight \(W\), is transformed into \(Q(X) \ast Q(W)\), where \(Q\) is a pseudo-quantization function that allows back-propagation and \(\ast\) represents a convolutional operator. LSQ \([21]\) is the current SOTA pseudo-quantization function, so we also use it in our experiments.

When quantizing re-parameterized models, there are several options available.

1. Apply re-parametrization and Post-Training Quantization (PTQ) \([8, 10, 20]\) successively. Research works \([10, 21]\) show that re-parametrization can lead to PTQ-un-
friendly distributions, resulting in a significant quality drop after PTQ application.

2. Apply re-parametrization and QAT successively. Trained re-parametrized full-precision blocks are converted into single layers, and standard QAT is applied to each of them.

3. Apply re-parametrization and QAT simultaneously as follows. Quantize each layer inside a re-parametrized block independently and then merge those layers into a single quantized layer only after QAT. This option is impractical as merging quantized convolutions results in a convolution with a higher bit width while binary and lower bit quantization is much more challenging than four or eight-bit quantization [50]. Consider a simple multiplication function $f(x) = xw$, which is re-parametrized as follows $R(x) = xw_1w_2$, with the resulting merged weight $w = w_1w_2$. When $w_1$ and $w_2$ are in FP32, their multiplication usually incurs a negligible loss of precision. However, if $w_1$ and $w_2$ are quantized to 2 bits, represented by integers in the range [0, 3], their multiplication results in an integer in the range [0, 9], requiring at least 4 bits for storage.

4. We introduce a novel approach where QAT and re-parametrization are applied simultaneously by performing pseudo-quantization on top of the merged re-parametrized block.

Options 1-3 are based on well-known approaches to regular model quantization, while option 4 is our novel approach designed specifically for the quantization of re-parametrized models. We exclude options 1 and 3 as they are either not generic or known to produce unsatisfactory results for the re-parametrized models. In our experiments, we compare option 2 to our proposed RepQ method (option 4) to demonstrate its effectiveness.

4 Proposed Methods

In Section 4.1, we describe the quantization strategy of re-parameterized blocks without BN layers and introduce a general RepQ training framework. For blocks with BN layers, we provide two alternative extensions in Sections 4.2 and 4.3.

4.1 RepQ: Quantization-aware training with re-parametrization

Let us first consider the scenario in which BN is not used in a re-parametrized block. The authors of [27] notice that it is possible to reduce training time by merging the re-parametrized blocks without BN into a single convolution while still optimizing the extended set of weights introduced by re-parametrization. This section shows how this merged training benefits QAT.

To illustrate the concept of re-parametrization, we use the simple example shown in Figure 1: $R(X, W) = X \ast W_1 \ast W_2 + X \ast W_3$. $R(X, W)$ denotes a re-parametrized block that replaces a single convolution during training. We can simplify this block to a single convolution with weight $M$ by deducing that $R(X, W) = X \ast (W_1 \ast W_2 + W_3) = X \ast M(W_1, W_2, W_3)$. In a broader sense, $M$ is a differentiable function that maps the block’s trainable parameters to the weight of the final converted convolution. This example easily generalizes to the other re-parametrization strategies, where the re-parametrized block has a form of $R(X, W) = X \ast M(W_1, \ldots, W_n)$ or, roughly speaking, re-parametrizations without BN. The articles introducing novel re-parametrized blocks regularly provide the formulas necessary to compute $M$, so we do not repeat them here.
Figure 1: This picture illustrates the application of a re-parametrized quantization-aware training (RepQ) to a single layer with convolutions with input $X$ and parameters $W_1, W_2, W_3$. The left plot illustrates the re-parametrization block substituting a single convolutional layer used as an example. The middle plot is an equivalent transformation (in terms of the gradient flow) to the left plot. Note that the computation order is different. The right plot shows how the pseudo-quantization functions $Q$ are inserted to perform RepQ.

Notably, merged training does not affect the gradient flow, as the two sides of the equation are numerically equivalent during both forward and backward passes. However, the merged weight $M$ is explicitly computed on the right side.

Now it is easy to see that the pseudo-quantization function can be applied on top of $M$:

$$X \ast M(W_1, \ldots, W_n) \rightarrow Q(X) \ast Q(M(W_1, \ldots, W_n)).$$

As a result, function $Q(M(W_1, \ldots, W_n))$ will equal the quantized weight used on the inference. Since $Q$ and $M$ are differentiable functions, the gradient propagates smoothly to the weights $W_1, \ldots, W_n$. The combination of re-parametrization with introducing $M$ and pseudo-quantization function $Q$ constitutes the RepQ approach and enables end-to-end quantization-aware training. This is the main algorithm used in all RepQ experiments.

### 4.2 RepQ-BN: Merging batch normalization

Many SOTA re-parametrizations use batch normalization. Several papers show that BN is an essential component of their blocks. BN’s removal leads to a significant performance drop. Since we aim to provide a quantization strategy that generalizes well to diverse re-parametrizations, we study how to handle BN in QAT.

The first option is fusing the BN with the preceding convolution during training, described in this section. A similar procedure was proposed in [30, 32] to achieve integer-arithmetic-only quantization. In our case, folding BN reduces the task to the no-BN case described in the previous section, 4.1.

$$BN(X \ast W) = \frac{X \ast W - \mathbb{E}[X \ast W]}{\sqrt{\mathbb{V}[X \ast W] + \varepsilon}} \gamma + \beta =$$

$$= X \ast \frac{W}{\sqrt{\mathbb{V}[X \ast W] + \varepsilon}} \gamma - \frac{\mathbb{E}[X \ast W]}{\sqrt{\mathbb{V}[X \ast W] + \varepsilon}} \gamma + \beta = X \ast M(X, W) + b(X, W).$$

(2)
The equation (2) shows that a convolution followed by BN is equivalent to a single convolutional operator. However, its parameters are dependent on the input $X$ through the batch statistics, mean and variance. Formally, re-parametrization with BN has the form of $R(X, W) = X \ast M(X, W_1, \ldots, W_n)$. Algorithm 1 shows how to compute $M$ and apply quantization in practice for a simple case of $R(X, W) = BN(X \ast W)$. By fusing BN with preceding convolutions, we reduce the task of merging weights to the no-BN case described in Subsection 4.1. We call this variant of our approach RepQ-BN.

Algorithm 1 Fusing BN with Convolution for Quantization

Given: $R(X, W) = BN(X \ast W)$ ▷ a simple re-parametrization of BN after a convolution
1: $Y = X \ast W$
2: $\mu, \Sigma = E[Y], \sqrt{\Sigma}$ ▷ computing BatchNorm statistics
3: $\hat{\mu} = (1 - m) \cdot \hat{\mu} + m \cdot \mu$ ▷ updating cumulative moving mean ($m$ denotes momentum)
4: $\hat{\Sigma} = (1 - m) \cdot \hat{\Sigma} + m \cdot \Sigma$ ▷ updating cumulative moving variance
5: $M(X, W) = \frac{W}{\sqrt{\Sigma + \epsilon}} \cdot \gamma$ ▷ computing merged weight
6: $b(X, W) = -\frac{\mu}{\sqrt{\Sigma + \epsilon}} \cdot \gamma + \beta$ ▷ computing merged bias
7: $R_q(X, W) = Q(X) \ast Q(M(X, W)) + b(X, W)$ ▷ quantized re-parametrized layer

4.3 RepQ-BNEst: Batch normalization estimation

An observant reader may have noticed that in Algorithm 1, the convolutional operator is computed twice, first in line 1 and then again in line 7. While the additional computation in line 1 is used to calculate the BN statistics $\mu$ and $\Sigma$, the question arises whether it is necessary to perform such a computationally expensive convolution to determine the mean and variance of the output. Here we propose a novel method of estimating BN running statistics based on inputs and weights without computing the convolution.

For simplicity, let us consider the case of a $1 \times 1$ convolution. The $1 \times 1$ convolution can be viewed as a matrix multiplication of the input $X$ of the shape $[B \cdot H \cdot D \cdot IN]$ and weight $W$ of shape $[IN, OUT]$, where $B$ is the batch size, $H$ is the image height, $D$ is the image width, $IN$ is the number of input channels, and $OUT$ is the number of output channels. Consider computing BN mean statistics,

$$\mathbb{E}[XW] = \mathbb{E}[X]W. \quad (3)$$

The equation suggests we can first calculate the per-channel mean over the batch, height, and width dimensions and then multiply the result by the weight matrix. This approach has a computational complexity of $O(B \cdot H \cdot D \cdot IN)$, making it more favourable than the naive solution, which has a complexity of $O(B \cdot H \cdot D \cdot IN \cdot OUT)$. Additionally, BN estimation allows us to avoid storing the feature map $XW$ on the GPU.

For exact variance computation, a similar reduction is not possible due to the need to calculate the input covariance matrix. As a solution, we propose to approximate the covariance matrix with a diagonal form,


where $W^2$ is an element-wise square of $W$, Cov is the sample covariance matrix, and $D$ is the diagonalizing operator that leaves only diagonal matrix elements non-zero. As a result of BN
estimation, the variance is substituted with another quadratic statistic of the weight and the output that estimates variance but is computationally more efficient for quantization-aware re-parametrization. The above formulas generalize to arbitrary weight shapes as follows (for more details, we refer to supplementary materials),

\[
E[X \ast W] \approx \tilde{E}[X \ast W] = E[X] \cdot \sum_{h,d} W_{h,d},
\]

(5)

\[
\nabla[X \ast W] \approx \tilde{\nabla}[X \ast W] = \nabla[X] \cdot \sum_{h,d} W^2_{h,d}.
\]

(6)

To sum up, for BN Estimation, we modify Algorithm 1 by replacing the calculation of mean \( \mu \) and variance \( V \) with \( \tilde{E} \) and \( \tilde{V} \), respectively, in line 2. In addition, we skip computing \( Y \) in line 1. We call this variant of our approach RepQ-BNEst.

5 Experiments

5.1 Experimental setup

Architectures. We evaluate the performance of RepQ on three architectures: ResNet-18, VGG, and ECBSR [54]. For ResNet-18, we employ two re-parametrization techniques, the well-known ACNets [13] and the recently published OREPA [27], which we refer to as AC-ResNet-18 and OREPA-ResNet-18, respectively. For VGG, we use the SOTA RepVGG [16] approach to re-parametrization with two network depth variants, RepVGG-A0 and RepVGG-B0. The ECBSR is a re-parametrization approach developed for the super-resolution problem. Our focus on evaluating RepQ with lightweight versions of SOTA architectures was motivated by their suitability for mobile devices.

Comparisons. We compare our RepVGG quantization results with QARepVGG, a quantization-friendly version of VGG introduced in [11]. To the best of our knowledge, no studies provide quantization results for re-parametrized architectures apart from RepVGG. To further support the main claims of our article, we have designed baselines, which are described in the following paragraph.

Baselines. Quantized model training includes two sequential stages: (1) a regular full-precision (FP) pre-training and (2) QAT. Weights pre-trained on the FP stage are used to initialize the quantized model at the beginning of the second stage. We compare our re-parametrized quantization-aware training RepQ that uses re-parametrization during QAT with the baselines that do not use re-parametrization in the QAT stage. In particular, we compare RepQ to plain and merged baselines schematically described in Table 1:

- **Plain.** Regularly trained model with no re-parametrization in both stages.

- **Merged.** The re-parametrized model is trained in the FP stage. Re-parametrized blocks are merged back into single convolutions, and the merged weights are used to initialize the quantized model. Such initialization can be profitable for subsequent quantization because re-parametrized models usually have better metrics than regular models.
Table 1: Quantization strategies: baselines (Plain, Merged) and our proposed method (RepQ).

Training pipeline. Generally, re-parametrization changes only the model architecture. For example, imagine that all convolutions are replaced by the following re-parametrization blocks \( R(X) = \text{BN}(X \ast W) + X \). The simplified pseudo-code of this block is shown below.

```python
class ReparametrizedBlock:
    def kernel(x, training):
        BN_stats = self.BN_stats(x, self.conv.weight, training)
        weight = fuse_BN(self.conv.weight, BN_stats)
        return fuse_residual(weight)

    def BN_stats(x, weight, training):
        if self.BNEst:
            return BNEst(x, weight, training)  # Use Eq. 5–6.
        else:
            return BN(x, weight, training)  # Use Algorithm 1..

    def call(x, training=False):
        # "training" defines the training or testing mode.
        if self.quantize:
            return conv2d(Q(x), Q(self.kernel(x, training)))  # Quantized training.
        elif self.BNEst:
            return conv2d(x, self.kernel(x, training))  # BN estimation for FP training.
        else:
            return self.bn(self.conv(x)) + x  # Regular FP training with BN.
```

Once full-precision training converges, the "self.quantize" parameters are set to True, and training is repeated with minor changes to hyperparameters, described in the supplementary.

Implementation details. We trained full-precision models in accordance with the re-parametrization articles’ setup using official repositories. That is why the baseline for the same model architecture may slightly differ for various re-parametrization blocks. We trained quantized models with the same hyperparameter setup as the full-precision models, except for learning rate adjustments. All quantized models are initialized with corresponding pre-trained full-precision weights; quantization steps are initialized using MinError [8] initialization for the first batch. For reproducibility, we provide hyperparameters in the supplementary materials.

5.2 Experimental results

We present our main results in Tables 2 and 3. They demonstrate that the proposed variants of RepQ consistently outperform the baselines of all the tested architectures and re-parametrized blocks. Table 3 exhibits the benefits of RepQ on a wide range of super-resolution datasets. For all the studied classification models, the 8-bit RepQ performance exceeds the full-precision result, while the QARepVGG [11] exhibits a quality drop. The gap between our RepQ and QARepVGG exceeds 1% for both RepVGG-A0 and RepVGG-B0. For RepVGG and ResNet, the best result is achieved either by RepQ-BN or RepQ-BNEst. In particular, RepQ-BN is slightly better than RepQ-BNEst in eight-bit experiments; nevertheless,
RepQ-BNEst achieves better results in four-bit quantization. Surprisingly, RepQ-BNEst outperforms RepQ-BN by a considerable margin on the 4-bit RepVGG-A0 and RepVGG-B0. In addition, we measured the ResNet-18 training speed on 2 V100 GPUs. RepQ-BNEst allows for a 25% training time reduction compared to RepQ-BN. The merged baseline demonstrates inconsistency on different studied architectures. For RepVGG-B0, the merged results are quite close to RepQ; however, for Resnet-18 and ECBSR, it is inferior to plain models.

<table>
<thead>
<tr>
<th>Model</th>
<th>Method</th>
<th>Precision 32 (FP)</th>
<th>Precision 8</th>
<th>Precision 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>RepVGG-A0 [16]</td>
<td>QARepVGG</td>
<td>72.40</td>
<td>71.90</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Plain</td>
<td>70.91</td>
<td>71.52</td>
<td>69.90</td>
</tr>
<tr>
<td></td>
<td>Merged</td>
<td>-</td>
<td>72.49</td>
<td>69.21</td>
</tr>
<tr>
<td></td>
<td>RepQ-BN</td>
<td>72.25</td>
<td>73.11</td>
<td>70.31</td>
</tr>
<tr>
<td></td>
<td>RepQ-BNEst</td>
<td>72.43</td>
<td>72.94</td>
<td>71.12</td>
</tr>
<tr>
<td>RepVGG-B0 [16]</td>
<td>QARepVGG</td>
<td>75.10</td>
<td>74.60</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Plain</td>
<td>73.48</td>
<td>74.79</td>
<td>72.52</td>
</tr>
<tr>
<td></td>
<td>Merged</td>
<td>-</td>
<td>75.48</td>
<td>73.06</td>
</tr>
<tr>
<td></td>
<td>RepQ-BN</td>
<td>75.27</td>
<td>75.60</td>
<td>73.51</td>
</tr>
<tr>
<td></td>
<td>RepQ-BNEst</td>
<td>75.38</td>
<td>75.53</td>
<td>74.01</td>
</tr>
<tr>
<td>AC-Resnet-18 [13]</td>
<td>Plain</td>
<td>70.24</td>
<td>70.77</td>
<td>70.00</td>
</tr>
<tr>
<td></td>
<td>Merged</td>
<td>-</td>
<td>70.64</td>
<td>68.25</td>
</tr>
<tr>
<td></td>
<td>RepQ-BN</td>
<td>71.01</td>
<td>71.40</td>
<td>70.07</td>
</tr>
<tr>
<td></td>
<td>RepQ-BNEst</td>
<td>70.80</td>
<td>71.23</td>
<td>70.20</td>
</tr>
<tr>
<td>OREPA-Resnet-18 [27]</td>
<td>Plain</td>
<td>71.24</td>
<td>71.76</td>
<td>71.49</td>
</tr>
<tr>
<td></td>
<td>Merged</td>
<td>-</td>
<td>69.58</td>
<td>69.96</td>
</tr>
<tr>
<td></td>
<td>RepQ</td>
<td>72.07</td>
<td>72.32</td>
<td>71.49</td>
</tr>
</tbody>
</table>

Table 2: ImageNet [44] top-1 accuracy for different quantization strategies for reparametrized NNs. The best result is emphasized with bold text, and the second best is underscored.

<table>
<thead>
<tr>
<th>Precision</th>
<th>Method</th>
<th>Set5</th>
<th>Set14</th>
<th>B100</th>
<th>U100</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>Plain</td>
<td>36.90</td>
<td>32.62</td>
<td>31.43</td>
<td>29.66</td>
</tr>
<tr>
<td></td>
<td>Merged</td>
<td>36.81</td>
<td>32.55</td>
<td>31.39</td>
<td>29.57</td>
</tr>
<tr>
<td></td>
<td>RepQ</td>
<td>36.91</td>
<td>32.54</td>
<td>31.39</td>
<td>29.53</td>
</tr>
<tr>
<td>8</td>
<td>Plain</td>
<td>36.20</td>
<td>32.16</td>
<td>31.09</td>
<td>28.91</td>
</tr>
<tr>
<td></td>
<td>Merged</td>
<td>36.16</td>
<td>32.15</td>
<td>31.08</td>
<td>28.95</td>
</tr>
<tr>
<td></td>
<td>RepQ</td>
<td>36.30</td>
<td>32.19</td>
<td>31.09</td>
<td>28.95</td>
</tr>
<tr>
<td>4</td>
<td>Plain</td>
<td>34.82</td>
<td>31.30</td>
<td>30.46</td>
<td>27.92</td>
</tr>
<tr>
<td></td>
<td>Merged</td>
<td>34.74</td>
<td>31.28</td>
<td>30.44</td>
<td>27.91</td>
</tr>
<tr>
<td></td>
<td>RepQ</td>
<td>34.89</td>
<td>31.36</td>
<td>30.50</td>
<td>28.00</td>
</tr>
</tbody>
</table>

Table 3: PSNR metric for ECBSR-m4c8 [54] architecture trained on DIV2K [1] dataset, and x2 scaling.
5.3 Discussion

Our results demonstrate that 8-bit quantization can improve the quality of re-parametrized models. A similar behaviour is also observed in plain models [21, 47, 55] and usually explained by an additional regularization effect of quantization. Four-bit quantization with the RepQ method results in a minor quality drop. At the same time, the number of bit-operations is reduced by four times compared to 8-bit models. Interestingly, 4-bit RepVGG-B0 has two times less bit-operations than 8-bit RepVGG-A0, while its accuracy is higher. This makes 4-bit RepVGG-B0 more favourable for deployment.

Limitations. The main limitation of re-parametrization and RepQ is the increase in training time (TT). Let us take as an example ResNet-18 re-parametrized with two different blocks: ACNets and OREPA. The table provides the relative training time of the re-parametrized models for Plain FP and QAT networks. The TT increase introduced by RepQ is comparable to the one introduced by re-parametrization on full-precision training for both ACNets and OREPA blocks. Although RepQ-BN experiences a longer training due to the twice-forward computation, RepQ-BNEst mitigates this issue. Overall, despite the training time overhead, re-parametrization and RepQ is beneficial when the trade-off between inference time and model quality is the priority.

<table>
<thead>
<tr>
<th>Block</th>
<th>FP</th>
<th>Re-parametrized</th>
<th>QAT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Plain</td>
<td>RepQ</td>
<td>RepQ-BN</td>
</tr>
<tr>
<td>ACNets</td>
<td>100%</td>
<td>200%</td>
<td>100%</td>
</tr>
<tr>
<td>OREPA</td>
<td>100%</td>
<td>210%</td>
<td>100%</td>
</tr>
</tbody>
</table>

Table 4: Training time overhead for different re-parametrization strategies on ResNet-18.

6 Conclusions

This paper introduces RepQ, a QAT strategy specifically designed for re-parametrized models. During training, RepQ merges re-parametrized blocks into a single convolution and applies a pseudo-quantization function on top of the merged weight. To be able to quantize arbitrary re-parametrization blocks, we provide a natural solution for merging non-linear batch normalization layers inside the blocks. Furthermore, we enhance this solution by estimating BN statistics and thus achieve a speed-up. We conduct an extensive experimental evaluation for diverse re-parametrized blocks and model architectures. Results show that RepQ surpasses existing solutions and provides lossless 8-bit quantization for the re-parametrized models. Overall, RepQ expands the applicability of re-parametrization to the field of quantized NN with an easy-to-implement approach.

References


[2] Milad Alizadeh, Arash Behboodi, Mart van Baalen, Christos Louizos, Tijmen Blankevoort, and Max Welling. Gradient l1 Regularization for Quantization Robust-


[34] Chuyi Li, Lulu Li, Hongliang Jiang, Kaiheng Weng, Yifei Geng, Liang Li, Zaidan Ke, Qingyuan Li, Meng Cheng, Weiqiang Nie, and others. YOLOv6: A single-stage object detection framework for industrial applications. arXiv:2209.02976, 2022.


