2021 International Conference on Signal Processing and Machine Learning (CONF-SPML) | 978-1-6654-1734-1/21/$31.00 ©2021 IEEE | DOI: 10.1109/CONF-SPML54095.2021.00059

Efficient Quantization Techniques for Deep Neural Networks

Chutian Jiang
Department of Computer Science
Florida State University
Tallahassee, United States
cj20cn@my.fsu.edu

***Abstract*—As model prediction becomes more and more accurate and the network becomes deeper and deeper, the amount of memory consumed by the neural network becomes a problem, especially on mobile devices. It is also very difficult to balance the tradeoff between computational cost and battery life, which makes mobile devices very hard as well to become smarter. Model quantification techniques provide the opportunity to tackle this tradeoff by reducing the memory bandwidth and storage and improving the system throughput and latency. This paper discusses and compares the state-of-the-art methods of neural network quantification methodologies including Post Training Quantization (PTQ) and Quantization Aware Training (QAT). PTQ directly quantizes the trained floating-point model. The implementation process is simple and does not require quantization during the training phase. QAT requires us to use simulated quantization operations to model the effect of the quantization, and forward and backward passes are usually performed in the floating-point model. Finally, as discussed in the experiments in this paper, we conclude that with the evolution of the quantization techniques, the accuracy gap between PTQ and QAT is shrinking.**

Keywords—deep neural network, quantification, Post Training Quantization, Quantization Aware Training

I. Introduction

As model prediction becomes more and more accurate and the network becomes deeper and deeper, the amount of memory consumed by the neural network becomes a problem, especially on mobile devices. Generally, current mobile phones are equipped with 4GB memory to support the simultaneous operations of multiple applications. If three deep neural network models are running at the same time (such as VGG-16), it usually takes up 1GB of memory in total. What's more, image-related applications usually need to process data in a short time, which needs at least 30 FPS (Frame per Second) to satisfy the real-time requirement. Hence, if the ResNet-50 network is deployed, 3GB/s memory bandwidth is required to run the network model. During inference, it will devour a large amount of power because CPU and memory consumption will both work at full utilization to support the frequent communication and computation demand. It is also very difficult to balance the tradeoff between computational cost and battery life, which makes mobile devices very hard as well to become smarter.

Model quantification techniques provide the opportunity to tackle this tradeoff by reducing the memory bandwidth and storage and improving the system throughput and latency. However, there is a lot of research in literature about effective quantitative methods in the fast-developing deep learning area. This paper discusses and compares the state-of-the-art methods of neural network quantification including Post Training Quantization (PTQ) and Quantization Aware Training (QAT). PTQ methods, discussed in section 2, and QAT methods, discussed in section 3, are compared in depth. Other quantization methods are discussed in section 4.

II. PTQ

PTQ stands for Post Training Quantization. PTQ directly quantizes the trained floating-point model. The implementation process is simple and does not require quantization during the training phase. The weight is quantified in advance. Then, activation needs to be quantized based on the fixed scale and zero point during the calibration process, and there is no recalculation of the quantization parameters (scale and zero point) in the process.

A. *Quantization method*

The quantization method is used to convert a floating-point tensor (FP32) into a low-bit tensor (int 8). PTQ mainly has two quantization methods that are asymmetric and symmetric. In this section, we mainly introduce the details of these two algorithms and their differences.

1) Asymmetric. Asymmetric quantization is also called uniform affine quantization. The asymmetric quantization can be formalized as follows:

$$x_q = \text{round}\left(\left(x_f - min_{x_f}\right)\frac{2^b - 1}{max_{x_f} - min_{x_f}}\right) = round\left(s * x_f - min_{x_f} * s\right) = round\left(s * x_f - z\right) \quad (1)$$

In this equation [11], x_q is the quantized tensor. x_f is the tensor of the original floating-point value. It contains three main variables. s is the scale factor (a floating-point value), z is the zero point (an integer value), and b denotes the bit-width. s and z are used to map the floating-point values to integers, and the size of the value depends on b. z is to ensure that 0 values can be quantized.[3]

$$z = \text{round}\left(s * min_{x_f}\right) \quad (2)$$

In Figure 1, we can find that $|min_{x_f}| \neq max_{x_f}$, the range is not symmetric with the origin point. So, this method is called asymmetric quantization.

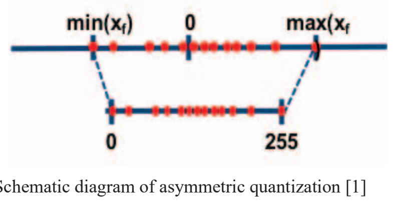

Fig. 1. Schematic diagram of asymmetric quantization [1]

2) Symmetric. Contrasted with asymmetric quantification, there is also symmetric quantification. The range is symmetric with respect to the origin point. The basic idea of the symmetric algorithm is to select the max absolute value between the min/max value in the floating-point range. In Figure 2, max_{x_f} is the max absolute value, which is then mapped to the maximum value of 8-bit data, that is 127. Compared with asymmetric quantization, symmetric algorithms do not use zero_point, and the quantization equation is as follows [3]:

$$x_q = \text{round}(s * x_f)(3) \tag{3}$$

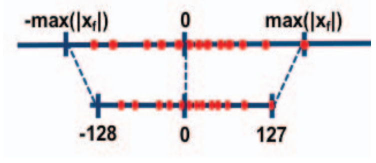

Fig. 2. Schematic diagram of symmetric quantization [1]

There are two different ways for calculating scale factor. One is called "full range" [3]. The "full range "quantized range is set as $\left[-\frac{b}{2}, \frac{b}{2} - 1\right]$. For INT8 quantification, use the range of $[-128,127]$. So scale factor can be calculated as:

$$s = \frac{(2^b-1)/2}{max_{|x_f|}} \tag{4}$$

The other is called "restricted range" [3]. However, quantized range is set as $\left[-\frac{b}{2}+1, \frac{b}{2} - 1\right]$, that is [-128,127] in int8 quantization. And the scale factor is

$$s = \frac{(2^b-1)}{max_{|x_f|}}) \tag{5}$$

3) Symmetric vs Asymmetric. Through sections 2.1.1 and 2.1.2, the advantage of asymmetric quantization is that the utilization rate of the quantized range is maximized. In Figure 1, the maximum and minimum values of the float range can be mapped to the maximum and minimum values of the quantized range. For symmetric quantization, if the data distribution is concentrated somewhere, it is not the best choice. For example, in Figure 2, most of the data values are located to the right of 0, Therefore, most of the quantized data generated are in the interval $[0,127]$, and there is no data distribution near the minimum value -128. This means that this part of the interval is useless. This reduces the capacity to express quantitative data and the accuracy of the quantitative model.

In general, since symmetric quantization does not require zero-point, it is easier to implement, and the cost of calculation is low. If the float range is biased towards one side, asymmetric quantization is better, for it has better accuracy. In PTQ, the range for activation map needs to collect and calculate signal statistics, such as feature map, min, and max, and this process is also called calibration. Because each input sample obtains a different activation map, it is also called dynamic quantization. FP32 weights can usually be quantized without any need for calibration because weights are a constant tensor.

The reference to "dynamic quantization" in this context likely describes how activation ranges are recalculated for each input sample during inference, not that asymmetric quantization itself is dynamic.

B. Floating point dynamic range

In section 2.1, scale factor and zero point can affect the accuracy of the mapping and be calculated from the range of FP32. The value range can be selected by adjusting min and max, and the data outside of the range is mapped to min/max. For example, the range [-1,1] is adjusted to [-0.9,0.8], that is $min_{x_f} = -0.9, max_{x_f} = 0.8$. The value in $[-0.9,0.8]$ will be mapped to (-127,127).

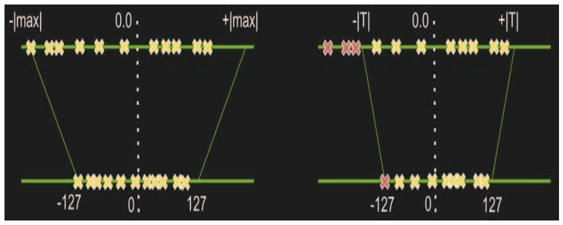

Fig. 3. Adjust the minimum/maximum value when quantizing floating point to fixed point [2]

If every FP32 value in the quantization process is in the actual dynamic range, we generally call this "No saturation" (Figure 3, left). Conversely, if some FP32 values are not within the actual dynamic range, we call it saturation (Figure 3, left). The main deep learning frameworks mainly use three methods: MinMax, mean squared error (MSE), and KL distance Kullback–Leibler divergence to set the range.

1) min/max. This is one of the most commonly used and easiest methods to implement. This method directly selects the maximum and minimum values from the FP32 tensor, just like the practice in Figure 1. Using this maximum and minimum value to determine the dynamic range, the equation of the min/max method is as follows:

$$min_{x_f} = \min(X) \tag{6}$$

$$max_{x_f} = \max(X) \tag{7}$$

where X represents the tensor to be quantified. Although this method is easy to implement, this method cannot handle subnormal values. For Activation, subnormal values in the data will cause the dynamic range to be expanded. For example, if 99% of the data is distributed among [-50,50] but one data is 1000, the result is that the dynamic range becomes [-50,1000]. This event leads to a decrease in accuracy.

2) Mean squared error (MSE). The loss-aware weight quantization is proposed by Lu [30]. The authors apply proximal Newton algorithm to solve the loss caused by quantization. In ternary weight network (TWN), weight is denoted as $w_l = a_l b_l$ where $a_l > 0$ and $b_l \in \{-1,0,1\}$. Given loss function ℓ, the second order expansion of the object can be written by using proximal Newton method as follows:

This method is to find the mean square error (MSE) between the corresponding quantized values, which can reduce the influence of subnormal values on the dynamic range.

We can try to find the smallest MSE to help us find min_{x_f} and max_{x_f}. The following is the equation of MSE [12]:

$$min = \| X - s * \hat{X} \|_F^2 \tag{8}$$

where $\|.\|_F$ is the Frobenius norm. $\hat{X}$ is used to express approximate linearly as a quantized tensor. z_p is the quantization range. P is the required integer precision. t is quantization offset [3].

$$\hat{X} = s\left[\frac{X-t}{s}\right]_{z_p} \tag{9}$$

$$t = min_i(X_i) \tag{10}$$

$$s = \frac{max_i(X_i)-t}{2^p-1} \tag{11}$$

3) *Kullback–Leibler divergence.* In the quantification process, we need to minimize the loss of accuracy of the data. The more similar the quantized data distribution is to the original data distribution, the smaller the data loss caused by quantization will be. Kullback–Leibler (KL) divergence can be used to measure the similarity between the two. First, collect histograms of activations and transform it into a quantitative distribution, and choose the KL divergence with the smallest threshold. Its equation is as follows [1]

$$D(P\|Q) = \sum(\mathrm{P[i]} * \log(\frac{\mathrm{P[i]}}{\mathrm{Q[i]}})) \tag{12}$$

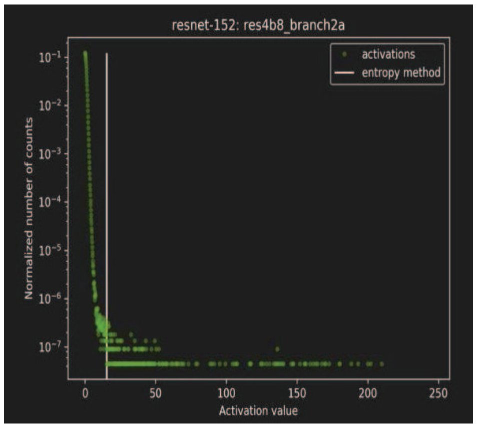

Fig. 4. KL distance dynamic range selection [2]

P and Q are two probability distributions. The distribution of data is determined by the size of the dynamic range. Data outside the range will be mapped to the boundary.

As shown in Figure 4, the ordinate represents the normalized number of counts, and the abscissa represents the Activation value. Through KL-divergence, most of the data can be concentrated on the left side of the white line. The data in the right part will be mapped to the maximum value (boundary) of the quantized data. But KL divergence also has limitations. First of all, it requires a lot of calculations. Secondly, the sample first determines the boundary of the quantization range from the left. For some cases in which the data distributions with inconspicuous distribution characteristics, it is hard to determine whether to determine the boundary of the dynamic range from the left or the right.

C. Quantization Granularity

The quantization granularity is divided into per-tensor quantization and per-channel quantization.

Per-tensor quantization is also called tensor-wise or per-layer quantization. In per-tensor quantization, one tensor sets one scale factor, which means that all values in the tensor are scaled in the same way. Both activation and weights can be regarded as a tensor. So, their quantification is the same. Per-tensor quantization is a coarse-grained quantization method, which is easier to achieve [8].

In int8 quantification, it is difficult to find a compromise between clipping and rounding error [9]. The output feature weights in depth-wise separable layers are few, which will increase the variability of the weights. The batch normalized folding will enhance this feature and make the weight between the output channels imbalanced. In deep learning, each channel of the tensor represents a feature. So, for different channels, their data distribution will be quite different.

As shown in Figure 5, each (output) channel weight range of the first deep separable layer in MobileNetV2 after BN is folded. We can find that the change between the weights is huge. This is

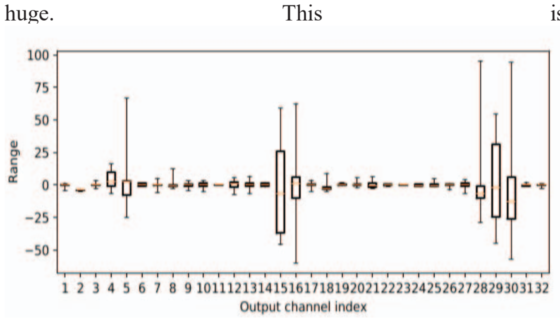

Fig. 5. Per (output) channel weight ranges of the first depthwise-separable layer in MobileNetV2 after BN folding [9]

But there is no such problem with per-channel quantization. Per-channel quantization belongs to the finer granularity, and each channel has different scale factors. Per-channel quantization of the weights can have better accuracy, but therefore it is more difficult to implement and takes longer.

TABLE I. QUANTIZATION ALGORITHMS ON THE ACCURACY OF WEIGHTS [8]

Network	Asymmetric Per-layer	Symmetrix Per-channel	Asymmetric Per-channel	Activation Only	Floating Point
Mobilenet-v_1_224	0.001	0.591	0.703	0.708	0.709
Mobilenet-v_2_224	0.001	0.698	0.697	0.7	0.719
Nasnet-Mobile	0.772	0.721	0.74	0.74	0.74
Mobilenet-v_1.4_224	0.004	0.74	0.74	0.742	0.749
Inception-v3	0.78	0.78	0.78	0.78	0.78
Resnet-v1_50	0.75	0.751	0.751	0.751	0.752
Resnet-v2_50	0.75	0.75	0.75	0.75	0.756
Resnet-v1_152	0.766	0.762	0.767	0.761	0.768
Resnet-v2_152	0.761	0.76	0.76	0.76	0.778

Table 1 shows the experimental results of some popular neural networks. Per-tensor quantization is selected for activation. The experiment compares the accuracy of the neural network when using different quantization methods. From the comparison results, it can be seen that the weights can obtain a lower accuracy loss when using asymmetric per-channel quantization.

D. Cross-Layer Equalization

Per-tensor quantization is not suitable for dealing with the weight imbalance between output channels. But batch normalization folding can exasperate this problem. And Nagel [9] proposed a solution, which can be called "Scaling equivariance". Suppose there are two layers: $h = f(W^{(1)}x + b^{(1)})$ and $y = f(W^{(2)}h + b^{(2)})$. The equation of Scaling equivariance is [9]:

$$\begin{aligned} y &= f\left(W^{(2)} f\left(W^{(1)}x + b^{(1)}\right) + b^{(2)}\right) \\ &= f\left(W^{(2)} S \hat{f}\left(S^{-1}W^{(1)}x + S^{-1}b^{(1)}\right) + b^{(2)}\right) \\ &= f\left(\widehat{W}^{(2)} \hat{f}\left(\widehat{W}^{(1)}x + \hat{b}^{(1)}\right) + b^{(2)}\right) \end{aligned} \quad (12)$$

where S is a scaling diagonal matrix, W is weight, and b is bias, and $\widehat{W}^{(1)} = S^{-1}W^{(1)}$, $\hat{b}^{(1)} = S^{-1}b^{(1)}$, $\widehat{W}^{(2)} = W^{(2)}S$. Scaling will process per channel and broadcast over the spatial dimension. Figure 6 shows a scale factor s_i scales

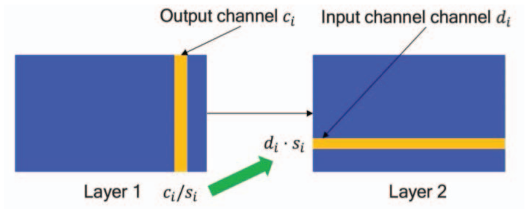

Fig. 6. Rescaling procedure [5]

channel c_i in layer 1, then leads a reparameterization in layer 2 by multiply d_i.

E. Bias

Bias is a parameter in DNN, which can adjust the activation function so that the triggering of the activation function will change. Each input in DNN is associated with a weight. As shown in Figure 7, when there is no bias, the input of the activation function is "x" multiplied by the weight "w_0".

No Bias
Input
x
Output
f(w0* x)
w0

Fig. 7. No bias [13]

When there is bias in Figure 8, the input of the activation function is the connection weight of "x" multiplied by the connection weight "w0" plus the bias multiplied by the bias "w1". This has the effect of moving the activation function.

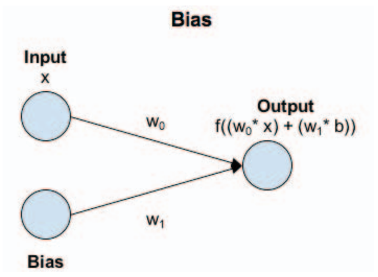

Fig. 8. With bias[13]

1) *Absorbing high bias.* High bias will cause the dynamic ranges of the activation to change. Nagel et al. proposed a solution called "Absorbing high biases". Its basic idea is to transfer high biases to the next layer. Its equation is as follows [5]:

$$y = W^{(2)} h + b^{(2)} \quad (13)$$

$$y = W^{(2)}\left(r\left(W^{(1)}x + b^{(1)}\right) + c - c\right) + b^{(2)} \quad (14)$$

$$y = W^{(2)}\left(r\left(W^{(1)}x + \hat{b}^{(1)}\right) + c\right) + b^{(2)} \quad (15)$$

$$y = W^{(2)}\hat{h} + \hat{b}^{(2)} \quad (16)$$

where $\hat{b}^{(2)} = cW^{(2)} + b^{(2)}$, $\hat{h} = h - c$, $\hat{b}^{(1)} = b^{(1)} - c$. r is the activation function, c is a non-negative vector that $r(Wx + b - c) = r(Wx + b) - c$

2) *Quantization error biased.* Quantization error is often biased, and this is a common problem. The output of the quantized layer will be shifted, thereby changing the input distribution of the next layer. Figure 9 shows the bias error of each channel of the convolutional layer in the MobileNetV2 model. As you can see in the Figure, in the layer's output, the quantization errors of many channels are biased. Depthwise-separable layers are particularly vulnerable to biased errors. So bias correction is needed.

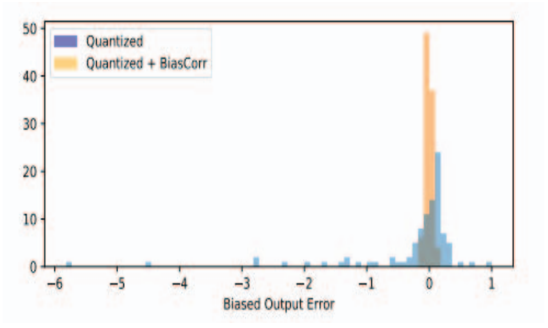

Fig. 9. Per-channel biased output error [5]

3) *Bias correction.* The two most common methods for calculating bias correction terms are empirical bias correction and analytic bias correction.

- Bias-correction: [9]: Let W be a weight tensor, and Wˆ be quantized weights, then quantization is $\Delta W = \widehat{W} - W$. So, the expected output distribution can be written as:

$$E[\hat{y}] = E[\widehat{W}] - E[\Delta Wx] \quad (17)$$

Where $E[\Delta Wx]$ is a biased error. If we want to eliminate the shift in the output, we can do this [9]:

$$E[y_{corr}] = E[\widehat{W}] - E[\Delta Wx] = E[y] \quad (18)$$

The "empirical" means that we can use calibration dataset and calculate the bias correction by comparing the quantized model and the full precision model. Its equation is as follows:

$$E[\hat{y}] = E[\widehat{W}] - E[\Delta Wx] \quad (19)$$

- Analytic bias correction[9]: This method does not require a calibration dataset to calculate the biased error. It requires the network to contain batch normalization, activation function belonging to clipped linear activation functions, and the input value satisfying a normal distribution.

  $E[x]$ can be calculated using the batch normalization parameters of the previous layer. Use γ and β to denote batch normalization scale and shift parameters. Set x^{pre} as pre-activation output. Introduce clipped normal distribution to represent the activation function $f()$ after the batch normalization layer. And P represents Probability Density Function, and C represents Cumulative Distribution Function. The equation is as follows[5]:

$$E[x] = E[f(x^{pre})] = \gamma P\left(\frac{-\beta}{\gamma}\right) + \beta\left[1 - C\left(\frac{-\beta}{\gamma}\right)\right] \quad (20)$$

  We can multiply the weighted error ΔW by $E[x]$ to get the biases correction term.

III. QAT

QAT stands for Quantization Aware Training. If the accuracy of the model decreases severely during the quantization process, QAT is required. QAT needs to model the quantization error during the training phase. This method can reduce the accuracy loss. QAT simulates low precision behavior in the forward pass and remains the same during the back pass. The quantization error generated in this process needs to be modeled during the training phase. Although QAT is more complicated, it can minimize the loss of accuracy.

A. *Forward and backward pass*

QAT requires us to use simulated quantization operations to model the effect of quantization, and forward and backward passes are usually performed in the floating-point quantization model. That is, in the forward pass, the quantized weights and activations are used; but in the backward pass, the float-type weights are still updated with gradients [6]. This step is actually optional in PTQ, but it is needed in QAT.

One difficulty of the back pass is how to deal with zero gradient. As shown in Figure 10, the top is Simulated Quantizer. We can find that its gradient or derivative is either zero or undefined. This will make the uniform quantizer function (Equation 1) always be 0and makes it impossible for us to choose gradient-based training. So, during the backward pass, we hope the image can become the bottom one.

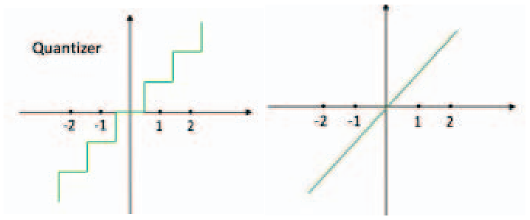

Fig. 10. Quantizer during forward and backward pass [10]

The method "straight-through estimator" (STE) can solve this problem [4][14], STE can change the gradient to 1. As:

$$gradient = \begin{cases} 0, & y < max \\ 1, & min \leq y \leq max \\ 0, & y > min \end{cases} \quad (21)$$

STE is just like its name, as shown in Figure 11. It skips the quantizer and passes the gradient directly to the function

B. *Batch normalization*

Batch normalization is a popular technique. What the "normalization" of the batch normalization method in DNN does is to perform data normalization processing in each layer of the network. But the calculation cost of normalizing all data in each layer is too large. Therefore, just like using the minimum batch gradient descent, the "batch" in batch normalization is actually sampling a small batch of data and then normalizing the output of the batch of data in each layer of the network to reduce the dependence between layers. Batch normalization is to count the mean and variance in a batch, normalize the input, then scale and shift it [8]. The equation for batch normalization has several parts:

$$For\ training: x_{bn} = \gamma\left(\frac{x-\mu_B}{\sigma_B}\right) + \beta \quad (22)$$

$$For\ inference: x_{bn} = \gamma\left(\frac{x-\mu}{\sigma}\right) + \beta \quad (23)$$

where γ and β are batch normalization's scale parameter. μ_B and σ_B are batch mean and standard deviations. μ and σ are the long term mean and standard deviations and are computed as moving averages of batch statistics during training.

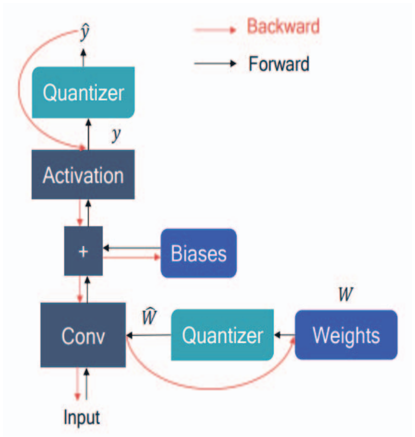

Fig. 11. Forward and backward computation graph with STE for QAT [9]

Another advantage of BN is that it can be folded in a convolutional layer. This means that we can replace convolution and batch normalization with just one convolution and different weights. The equation for weight is as follows [7]:

$$w_{fold} = \left(\frac{\gamma\, w}{\sqrt{EMA(\sigma_B^2)+\varepsilon}}\right) \quad (24)$$

$EMA(\sigma_B^2)$ stands for moving average estimate of the variance of convolution results across the batch. ε is a constant for numerical stability. Figure 12 shows the training graph with BN folding.

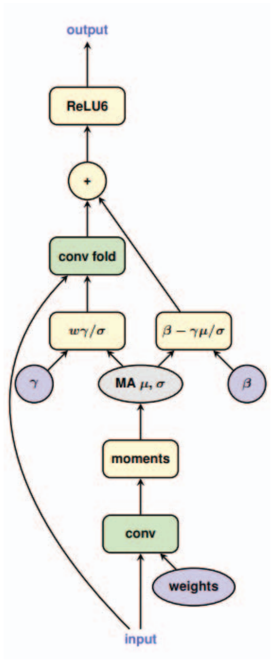

Fig. 12. training graph with BN folding [7]

IV. Comparison and Discussion

Table 2 are Nagel's experimental results [9], 3, and 4 are Wu's experimental results [6]. The models, as shown in table 1, all uses PTQ. In Table 2, W8A8 means 8-bit quantization of weights and activation, the same as W4A8. We can find per-channel quantization has not achieved a significant improvement in accuracy in table 2. But sometimes per-tensor quantization can cause serious loss of accuracy of some models. But as shown in table 2, using per-channel quantization maintain model accuracy and most of the model accuracy is better than Per-tensor quantization. When the weight is reduced to 4 digits (W4A8), EfficientDet-D1 will suffer severe accuracy loss. This may be due to the quantization of the depth separable convolution.

TABLE II. Comparison of the accuracy of per-tensor and per-channel quantization granularities [9]

Model	FP32	Per-tensor		Per-channel	
		W8A8	W4A8	W8A8	W4A8
ResNet18	69.68	69.60	68.62	69.56	68.91
ResNet50	76.07	75.87	75.15	75.88	75.43
MobileNetV2	71.72	70.99	69.21	71.16	69.79
InceptionV3	77.40	77.68	76.48	77.71	76.82
EfficientNet lite	75.52	75.25	71.24	75.39	74.01
DeeplabV3	72.94	72.44	70.80	72.27	71.67
EfficientDet-D1	40.08	38.29	0.31	38.67	35.08

And table 3 shows that when using int8 as the weight, for most models, choosing per-channel quantization will only cause a small amount of accuracy degradation, compared to fp32. This is a very desirable result. Table 4 lists the accuracy comparison between PTQ and QAT, both of which are based on Symmetric, per-channel. We can find that the accuracy loss of QAT is very small compared with FP32. For Mobilenet-v1 and MobileNet-v2_1, QAT has a great accuracy advantage compared to PTQ. However, compared with other models, the accuracy improvement of QAT is very small, and PTQ can also maintain a small accuracy loss.

TABLE III. Accuracy with int8 quantization of weights only: per-channel granularity [6]

Model	fp32	Accuracy	Relative
MobileNet-v1	71.88	71.59	-0.40%
MobileNet-v2	71.88	71.61	-0.38%
ResNet50-v1.5	76.16	76.14	-0.03%
ResNet152-v1.5	78.32	78.28	-0.05%
Inception-v3	77.34	77.44	0.13%
Inception-v4	79.71	79.64	-0.09%
ResNeXt50	77.61	77.62	0.01%
ResNeXt101	79.30	79.29	-0.01%
EfficientNet-b0	76.85	76.72	-0.17%
EfficientNet-b3	81.61	81.55	-0.07%
Faster R-CNN	36.95	36.86	-0.24%
Mask R-CNN	37.89	37.84	-0.13%
Retinanet	39.30	38.20	-0.25%
FCN	63.70	63.70	0.00%
DeepLab-v3	67.40	67.40	0.00%
GNMT	24.27	24.41	0.58%
Transformer	28.27	28.58	1.10%
Jasper	96.09	95.10	0.01%
Bert Large	91.01	90.94	-0.08%

TABLE IV. Summary of Post Training Quantization and Quantization Aware Training [8]

Model	fp32 Accuracy	QAT best	PTQ best
Mobilenet-v1	70.9	70.7	59.1
MobileNet-v2_1	71.9	71.1	69.8
Nasnet-Mobile	74	73	72.1
Mobilenet-v2_1.4	74.9	74.5	74
Inception-v3	78	78	78
Resnet-v1_50	75.2	75	75.1
Resnet-v2_50	75.6	75	75
Resnet-v1_152	76.8	76.2	76.2
Resnet-v2_152	77.8	76	76

V. Conclusion

This paper introduces and contrasts the state-of-the-art methods of neural network quantification including Post Training Quantization (PTQ) and Quantization Aware Training (QAT). PTQ directly quantizes the trained floating-point model, which is a lightweight quantization method. The computational cost of PTQ is low. And QAT is more complicated. It can minimize the loss of accuracy. QAT needs to model the quantization error during the training phase. During experiments in this paper, we can find that with the advancement of technology, the accuracy gap between PTQ and QAT is very small, which gives us a reason to believe that PTQ is a better choice in most cases.

In the process of research, how to choose dynamic ranges of weights and activations will affect the results of network quantification. All existing methods have limitations. How to use regularization techniques to solve this problem is a valuable research direction in the future [15].

REFERENCES

[1] Deep learning model quantification (low precision reasoning) summary - programmer sought. (n.d.). https://www.programmersought.com/article/20754571637/.

[2] Migacz, S. (2017, May 8). 8-bit Inference with TensorRT. NVIDIA. https://gputechconf2017.smarteventscloud.com/connect/sessionDetail.ww?SESSION_ID=105897.

[3] Quantization algorithms. Quantization - Neural Network Distiller. (n.d.). https://intellabs.github.io/distiller/algo_quantization.html#range-based-linear-quantization.

[4] Intuitive explanation of straight-through estimators with pytorch implementation. Hassan Askary. (2020, September 19). https://www.hassanaskary.com/python/pytorch/deep%20learning/2020/09/19/intuitive-explanation-of-straight-through-estimators.html.

[5] Nagel, M., Baalen, M. V., Blankevoort, T., &amp; Welling, M. (2019). Data-free quantization through weight equalization and bias correction. 2019 IEEE/CVF International Conference on Computer Vision (ICCV). https://doi.org/10.1109/iccv.2019.00141

[6] Wu, H., Judd, P., Zhang, X., Isaev, M., & Micikevicius, P. (2020). Integer Quantization for Deep Learning Inference: Principles and Empirical Evaluation. *ArXiv, abs/2004.09602*.

[7] Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H., &amp; Kalenichenko, D. (2018). Quantization and training of neural networks for efficient integer-arithmetic-only inference. 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition. https://doi.org/10.1109/cvpr.2018.00286

[8] Krishnamoorthi, R. (2018). Quantizing deep convolutional networks for efficient inference: A whitepaper. *ArXiv, abs/1806.08342*.

[9] Nagel, M., Fournarakis, M., Amjad, R.A., Bondarenko, Y., Baalen, M.V., & Blankevoort, T. (2021). A White Paper on Neural Network Quantization. *ArXiv, abs/2106.08295*.

[10] Gholami, A., Kim, S., Dong, Z., Yao, Z., Mahoney, M., & Keutzer, K. (2021). A Survey of Quantization Methods for Efficient Neural Network Inference. *ArXiv, abs/2103.13630*.

[11] Nayak, P., Zhang, D.C., & Chai, S. (2019). Bit Efficient Quantization for Deep Neural Networks. *EMC2@NeurIPS*.

[12] Choukroun, Y., Kravchik, E., & Kisilev, P. (2019). Low-bit Quantization of Neural Networks for Efficient Inference. *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, 3009-3018.

[13] The role of bias in Neural Networks. Pico. (n.d.). https://www.pico.net/kb/the-role-of-bias-in-neural-networks/.

[14] Bengio, Y., Léonard, N., & Courville, A.C. (2013). Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation. *ArXiv, abs/1308.3432*.

[15] Yang, Q., Wen, W., Wang, Z., & Li, H. (2019). Joint Pruning on Activations and Weights for Efficient Neural Networks. *ArXiv, abs/1906.07875*.