Dataflow-based Joint Quantization for Deep Neural Networks

Xue Geng[†], Jie Fu[‡], Bin Zhao[♯], Jie Lin[†], Mohamed M. Sabry Aly[§], Christopher Pal[‡], Vijay Chandrasekhar[†]

[†]I^{2}R, A⋆STAR
{geng_xue,lin-j,vijay}@i2r.a-star.edu.sg

[‡]MILA, IVADO
{jie.fu,christopher.pal}@polymtl.ca

[♯]IME, A⋆STAR
zhaobin@ime.a-star.edu.sg

[§]School of CSE, NTU
msabry@ntu.edu.sg

Abstract

This paper addresses a challenging problem – how to reduce energy consumption without incurring performance drop when deploying deep neural networks (DNNs) at the inference stage. In order to alleviate the computation and storage burdens, we propose a novel dataflow-based joint quantization approach with the hypothesis that a fewer number of quantization operations would incur less information loss and thus improve the final performance. It first introduces a quantization scheme with efficient bit-shifting and rounding operations to represent network parameters and activations in low precision. Then it restructures the network architectures to form unified modules for optimization on the quantized model. Extensive experiments on ImageNet and KITTI validate the effectiveness of our model, demonstrating that state-of-the-art results for various tasks can be achieved by this quantized model. Besides, we designed and synthesized an RTL model to measure the hardware costs among various quantization methods. For each quantization operation[1], it reduces area cost by ∼15× and energy consumption by ∼9×, compared to a strong baseline.

Introduction

DNNs have been increasingly deployed at the edge due to its remarkable performance. In particular, energy becomes an import factor when processing DNNs at the edge in embedded devices with limited battery capacity (*e.g.*, smartphones, smart sensors, UAVs, and wearables) [1]. Hence, there is an increasing demand for reducing energy consumption and increasing throughput without incurring a significant drop in accuracy. Extensive research has focused on reducing the precision of operations and operands to address the challenge (*e.g.*, DoReFa-Net [2], BinaryNet [3], XNOR-Net [4] and ternary quantization [5]).

Although these quantization methods remarkably reduce the model complexity, they are limited in two key ways. Firstly, a noticeable performance drop exists. For example, DoReFa-Net has a 2.9% performance loss for AlexNet on ImageNet with 8-bit weights and activations. Secondly, additional hardware costs are introduced due to extra inefficient operations such as codebooks in [6] and scaling factors in DoReFa-Net and IOA [7].

To address the above issues, we introduce an energy-efficiency quantization scheme which represents both network parameters and activations in low precision. It only

[1]Here, we choose to measure the energy consumption for various quantization methods on individual operations for simplicity. The overall energy consumption is in proportion to this measurement.

contains efficient bit-shifting and rounding-to-nearest behaviors. Then, based on this scheme, we re-structure the network to form unified modules, which reduce the number of quantization operations. Here we hypothesize that fewer number of quantization operations would incur less information loss and thus boost the final performance accuracy. Finally, a joint reconstruction error loss function is set up on these unified modules to do optimization on the quantized model.

We evaluate our proposed method on ImageNet [8] with ResNet [9] and KITTI [10] with Faster R-CNN [11]. Furthermore, we designed and synthesized a hardware unit to measure the hardware costs among various quantization operations. We show that our proposed approach performs comparably or even better in terms of accuracy and computational costs over existing state-of-the-art approaches on various tasks. In summary, our key contributions are as follows:

1. A purely quantized network is introduced to represent network parameters and activations into low precision with energy-efficient operations. As compared to floating-point representation, the 8-bit quantized model leads to less computation and memory accesses by $\sim 4\times$ without significant performance drop.

2. A novel joint quantization approach is introduced by taking the network dataflow into consideration to define unified modules and then set up joint reconstruction error objectives to learn the optimal parameters for the quantization model. Besides, our approach does not have the time-consuming fine-tuning stage.

3. Extensive experiments on various benchmark datasets with state-of-the-art deep models demonstrate that our proposed method reduces computational costs while still achieving decent accuracy.

1 Proposed Method

In this section, we first describe our quantization scheme in DNNs and then provide a detailed induction on the integer-only inference. In the end, a joint quantization approach of network parameters and activations is presented.

1.1 Quantization Scheme

A common practice in reducing the precision of DNNs is to introduce a quantization function [2, 7]. A basic requirement of our quantization scheme is that it is hardware-friendly, *i.e.*, without incurring intensive hardware operations and has equal conversion between integer and floating-point representation. The quantization scheme is defined as: Given a floating-point value r, we use a quantization function, $Q(\cdot)$, to approximate it:

$$r^q = Q(r; N_r, n_{bits}) = \underbrace{min(2^{n_{bits}-1} - 1, max(-2^{n_{bits}-1}, round(r \times 2^{N_r})))}_{r^I} \times 2^{-N_r} \tag{1}$$

where r^q is the quantized floating value, r^I is the integer value and N_r is the fractional bit which is the only parameter to set. n_{bits} is the bit-width including 1 sign bit. For example, when $n_{bits} = 8$, r^I is a 8-bit integer within the range of $[-128, 127]$. When N_r is negative, only the data before the decimal point is selected. For instance, if N_r is -3 with 8-bit bit-width, we select the 3 to 10 digits before the decimal point as the low-precision data. Our quantization scheme only contains bit-shifting, which is different from previous works [2, 7] with multiplication-based scaling factors or [12] weight loading from a codebook.

In DNNs, a separate fractional bit is selected for weights, bias, and activations of each layer. For instance, a convolution layer would have a set of quantization parameters – N_w, N_b, N_x, N_o for weights, bias, input activation and output activation, respectively. N_x comes from the output activation of the previous layer.

1.2 Integer-arithmetic-only Operations

The above quantization scheme is in floating-point arithmetic and thus supposed to be deployed to general hardware platforms such as GPUs and CPUs. In this section, we describe an integer-only arithmetic inference in DNNs by adopting the proposed scheme. The inference provides a step-by-step solution on how custom hardware units compute on DNNs using fixed-point representations.

Following the notations in [1], we define the 2-D convolution operation to be:

$$\mathbf{O}_{l,m,n} = \mathbf{B}_l + \sum_{k=0}^{C-1} \sum_{i=0}^{H-1} \sum_{j=0}^{W-1} \mathbf{X}_{k,Sm+i,Sn+j} \mathbf{W}_{l,k,i,j} \tag{2}$$

where $\mathbf{O}$, $\mathbf{X}$, $\mathbf{W}$, and $\mathbf{B}$ are the matrices of the output feature maps (ofmaps), input feature maps (ifmaps), filters, and biases, respectively. S is a provided stride while C, H and W are ifmaps channels, height and width, respectively.

After quantizing weights, biases and activations, convolution operation becomes:

$$\begin{aligned}
&CONV(\mathbf{X}, \mathbf{W}, \mathbf{B}; N_x, N_w, N_b) \\
&= \mathbf{B}_l^I \times 2^{-N_b} + \sum_{k=0}^{C-1} \sum_{i=0}^{H-1} \sum_{j=0}^{W-1} \mathbf{X}^I_{k,Sm+i,Sn+j} \times 2^{-N_x} \times \mathbf{W}^I_{l,k,i,j} \times 2^{-N_w} \\
&= 2^{-N_w - N_x} (\mathbf{B}_l^I \times 2^{(N_x+N_w)-N_b} + \sum_{k=0}^{C-1} \sum_{i=0}^{H-1} \sum_{j=0}^{W-1} \mathbf{X}^I_{k,Sm+i,Sn+j} \mathbf{W}^I_{l,k,i,j}) \\
&= 2^{-N_w - N_x} \times \mathbf{O\_int32}_{l,m,n}
\end{aligned} \tag{3}$$

where $\mathbf{X}^I$, $\mathbf{W}^I$ and $\mathbf{B}^I$ are the integer matrices of the ofmaps, ifmaps, filters, and bias, respectively. N_x, N_w and N_b are the fractional bit of ifmaps, filters, and biases, respectively. We carefully align biases with the convolution output by sacrificing smaller values achieving less information loss. The intermediate result of convolution is 32-bit integer to handle the overflow of accumulation.

Then, we need to quantize the output of the convolution layer as:

$$\begin{aligned}
\mathbf{O}^q_{l,m,n} &= Q(CONV(\mathbf{X}, \mathbf{W}, \mathbf{B}; N_x, N_w, N_b), N_o, n_{bits}) \\
&= \mathbf{O}^I_{l,m,n} \times 2^{-N_o}
\end{aligned} \tag{4}$$

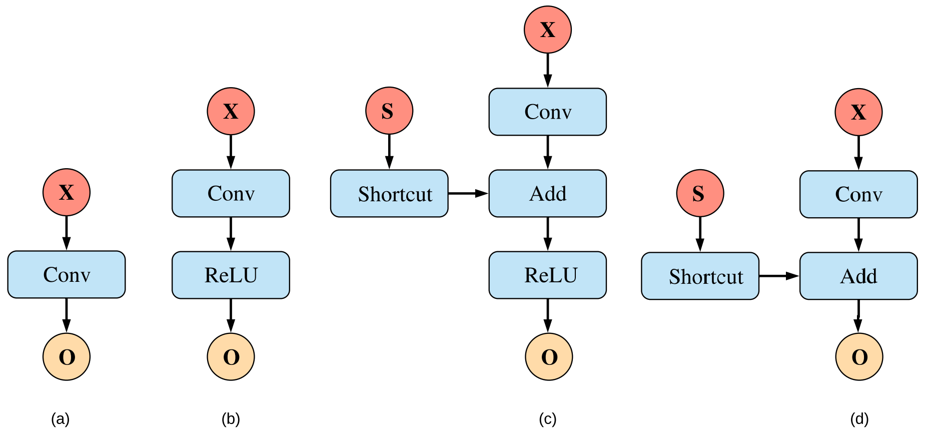


Figure 1: In general, four common cases are considered here: a) convolution layer; b) a convolution layer followed by a ReLU layer; c) a residual connection with a ReLU layer; and d) a residual connection without a ReLU layer.

where $\mathbf{O}^I$ is the integer matrices of the ofmaps. In custom hardware units, two sets of data are stored: one is the integer matrices of ifmaps $\mathbf{X}^I$, ofmaps $\mathbf{O}^I$, filters $\mathbf{W}^I$, and bias $\mathbf{B}^I$; and the other is the bit-shifting values for data alignment in biases and activations such as $N_x + N_W - N_b$ in Equation 3 but not the fractional bits.

1.2.1 Dataflow-based Joint Quantization

The impact of activations on storage capacity depends on the network architecture and dataflow [1]. Hence, the dataflow of network architecture becomes an important factor to quantize the activations. Here we hypothesize that if more quantization operations are applied, it is likely to incur more measurement noise which might lead to information loss. To reduce the number of quantization operations, we propose to combine several basic layers into a unified module based on network architectures. We only consider ReLU as activation functions because it is the de facto choice for DNNs due to its effectiveness. For instance, the quantization is conducted after a ReLU layer in Figure 1(b) because that 1) the negative part can be skipped from computing the quantization error as only non-negative values exist after ReLU layer; 2) the cost of memory accesses is reduced dramatically without writing the convolution output back to memory. This is different from existing methods such as [2], which quantizes activations instantly after convolution. Furthermore, the batch normalization layer is merged into the weights and biases of the next convolution layer at inference stage.

Figure 1 shows 4 cases to conduct joint quantization. In particular, Equation 3 is the simplest case in Figure 1(a). Figure 1(b) talks about the scenario of a convolution layer followed by ReLU, the activations are quantized after the ReLU layer instead of convolution layer. Thus the outputs of the ReLU layer is in the range $[0, 255]$ if the bit-width is 8-bit. Figure 1 (c) and (d) provide two situations of quantization on the residual connection, respectively. As Figure 1(c) has a residual connection from previous block, it is necessary to consider the alignment of shortcut outputs and convolution outputs. Furthermore, if the shortcut is a convolution layer, more

Algorithm 1: Quantization on weights, biases and activations of convolution layer

```
1  Initialization: $\hat{N}_w \leftarrow 0$, $\hat{N}_o \leftarrow 0$, $\hat{N}_b \leftarrow 0$, $err\hat{o}r \leftarrow +\infty$, $n\_bits \leftarrow 8$, $\tau \leftarrow 4$.
2  Input: $\mathbf{W}$, $\mathbf{B}$, $\mathbf{O}$.
3  $N_w^{max} \leftarrow ceiling\left(\log_2\left(\max\left(abs(\mathbf{W})\right)+1\right)\right)+1, N_w^{min} \leftarrow N_w^{max}-\tau$
4  $N_b^{max} \leftarrow ceiling\left(\log_2\left(\max\left(abs(\mathbf{B})\right)+1\right)\right)+1, N_b^{min} \leftarrow N_b^{max}-\tau$
5  $N_o^{max} \leftarrow ceiling\left(\log_2\left(\max\left(abs(\mathbf{O})\right)+1\right)\right)+1, N_o^{min} \leftarrow N_o^{max}-\tau$
6  foreach $i \in \left[N_w^{min}, N_w^{max}\right]$ do
7      $N_w = (n\_bits-1)-i$, $\mathbf{W}^q \leftarrow Q\left(\mathbf{W}, N_w, n\_bits\right)$
8      foreach $j \in \left[N_b^{min}, N_b^{max}\right]$ do
9          $N_b = (n\_bits-1)-j$, $\mathbf{B}^q \leftarrow Q(\mathbf{B}, N_b, n\_bits)$
10         foreach $k \in [N_o^{min}, N_o^{max}]$ do
11             $N_o = (n\_bits-1)-k$, $\mathbf{O}^q \leftarrow Q(CONV(\mathbf{W}^q, \mathbf{B}^q, \mathbf{X}^q), N_o, n\_bits)$
12             $error \leftarrow ||\mathbf{O}-\mathbf{O}^q||_2$
13             if $err\hat{o}r > error$ then
14                 $\hat{N}_w \leftarrow N_w$, $\hat{N}_b \leftarrow N_b$, $\hat{N}_o \leftarrow N_o$, $err\hat{o}r \leftarrow error$
15             end
16         end
17     end
18 end
19 return $\hat{N}_w, \hat{N}_b, \hat{N}_o$
```

complex alignment is done on two convolution layers. As compared to Figure 1(c), Figure 1(d) does not have a ReLU layer after addition.

1.2.2 Optimization

Following [13], we assume that the quantization error in each layer is positive relevant to the final performance accuracy. Hence it is reasonable to minimize each layer's reconstruction error to boost the performance. Our target is to minimize the reconstruction error between the outputs and the quantized outputs as follows:

$$\hat{N}_w, \hat{N}_b, \hat{N}_o = \underset{N_w, N_b, N_o}{\arg\min} ||\mathbf{O} - Q(CONV(\mathbf{X}, \mathbf{W}, \mathbf{B}; N_x, N_w, N_b), N_o, n_{bits})||_2 \tag{5}$$

Where $\hat{N}_w$, $\hat{N}_b$, $\hat{N}_o$ are optimal fractional bits of weights, biases and output activations separately. N_x is from the optimal bit of activations of the previous layer.

Our proposed method is based on the pre-trained model without fine-tuning and the optimization is to learn the optimal fractional bits. To speed up the search, we first narrow down the search space and then apply a simple grid search approach. This is motivated by the observation [14] that larger weights play a more important role than smaller ones. Hence, we hypothesize that the optimal fractional bit should be located in the upper bits. Algorithm 1 provides a detailed solution to the optimization. First, the largest fractional bit to represent the maximum value of parameter $\mathbf{W}$ is:

$$N_w^{max} = ceiling\left(\log_2\left(\max\left(abs(\mathbf{W})\right)+1\right)\right)+1 \tag{6}$$

The search space would be $[N_w^{max}, N_w^{max} - \tau]$ where τ is a hyper-parameter to be set empirically. Biases and activations have a similar search strategy. Now, we are

Table 1: ResNet on ImageNet: Floating-point (FP) versus 8-bit quantized network for various network depths.

Methods	FP	TensorRT[15]	IOA[7]	Ours
ResNet-50	75.2%	73.1%	73.8%	73.6%
ResNet-101	76.4%	74.4%	74.6%	74.6%
ResNet-152	76.8%	74.7%	75.1%	75.0%
Quantization type	N/A	scaling factor	scaling factor	bit-shifting

Table 2: Training time for joint quantization for various network depths of ResNet.

Methods	ResNet-50	ResNet-101	ResNet-152
Training time (mins)	5.6	7.1	8.5

ready to traverse all the solutions in a narrowed-down search space to get the optimal fractional bits. Obviously, the time complexity of Algorithm 1 is $O(\tau^3\Gamma)$ where Γ comes from the convolution operation. In our experiments, we empirically set τ as 4.

2 Experiments

We conduct experiments on ImageNet [8], KITTI [10] datasets with two widely used deep models – ResNet [9] and Faster R-CNN [11] to investigate the performance of our proposed method. Besides, we conduct hardware measurements on various approaches to verify the computational efficiency of our proposed method.

2.1 Implementation Settings

Our optimization is conducted on a single image as the number of weights, bias and activations for each layer are enough to bring little biases on the optimization. Meanwhile, the baselines are implemented according to their default settings. The CPU platform is AMD 1950X 16-Core CPU with 64GB RAM while the GPU platform is NVIDIA Tesla V100.

To evaluate the hardware cost among various methods, we have created an RTL model for each method and conducted synthesis using UMC 40nm library, the area and power are then estimated at 500MHz clock Synthesizers.

2.2 Evaluation on ImageNet

For training, all images are resized preserving aspect ratio so that the smallest side is 256. Then the center 224×224 patch is cropped and each of the RGB channels subtracts the means. While for inference, we apply a single-crop testing for standard evaluation. We apply our method to ResNet with different layer depths on ImageNet. The pre-trained floating-point model is from the TensorFlow's official repository [2]. Method IOA [7] is fine-tuned on the pre-trained floating-point model.

[2]https://github.com/tensorflow/models/tree/master/research/slim

Table 3: ResNet50 on ImageNet: Accuracy under various approaches with different bit-widths.

Methods	CLIP-Q[16]	INQ[17]	ABC-net[18]	FGQ[19]	Ours
Weight bits	4	5	5	2	8
Activation bits	32	32	5	8	8
Quantization type	codebook	N/A	scaling factor	scaling factor	bit-shifting
Accuracy	73.7%	74.8%	70.1%	70.8%	73.6%

Table 4: Object detection performance on KITTI dataset for various data precision.

	FP	8-bit	7-bit	6-bit
Car	94.35%	93.98%	92.32%	75.30%
Pedestrian	80.78%	81.43%	79.06%	53.85%
Cyclist	87.63%	86.87%	85.34%	48.05%

Table 1 demonstrates the performance of ResNet-50, ResNet-101 and ResNet-152 on ImageNet. It can be seen that 1) our method is robust with various network depths with only about 1.8% drop; 2) as compared to other baselines, our proposed approach achieves competitive performance. Although baseline IOA has similar results, it contains scaling factors and 32-bit biases. Besides, it has extra addition operations on the "zero-point" values. While our method saves more hardware cost with bit-shifting and 8-bit biases; and 3) the quantized model with optimal fractional bits is probably a great starting point to continue fine-tuning. It may further improve the performance and speed up the training as the search space of parameters becomes smaller.

Table 2 lists the training time of our proposed method on ResNet with various network depths. We observe that in comparison with several-days fine-tuning on a pre-trained model, our approach has a dramatically fast speed within several minutes. Table 3 reports the ImageNet accuracy for ResNet-50 on various approaches with various bit-widths. As expected, the integer-only work performs better than ABC-net, which uses 5-bit for weights and activations. INQ and CLIP-Q perform better than ours as they represent activations in 32-bit.

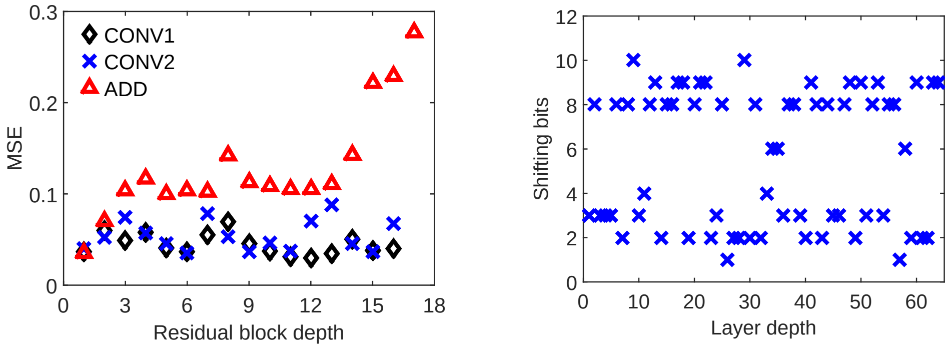
2.3 Evaluation on KITTI

For object detection, we evaluate our method on the autonomous driving benchmark dataset – KITTI, using Faster R-CNN (F-RCNN) with ResNet-152 as the backbone network. Since the ground-truth for test set is not publicly available, we randomly sample 80% of training images for training F-RCNN and the remaining for validation on the quantized model. All images are resized into HD resolution (1280×720).

Table 4 lists the object detection performance of our proposed method in terms of various bit-widths on KITTI dataset. We observe that our method in 8-bit achieves almost and even better performance as compared to full precision models. Our method in 7-bit precision also has a competitive performance. However, 6-bit representation has a dramatic performance drop.

Table 5: Hardware cost of various methods with 32-bit inputs and 8-bit outputs.

Operation types	scaling factor	k-means	bit-shifting
Power (mW)	30.6	228.8	15.5
Area (µm^2)	502.7	1787.6	198.2



(a) MSE with residual block depth. (b) Shifting bits with layer depth.

Figure 2: Statistics on ResNet-50.

2.4 *Hardware Experiments*

We measure the energy and area costs in Table 5 on various operations. To do comparison fairly, all implementations are constrained to 32-bit input and 8-bit output. In particular, scaling factor operation is implemented by a 32-bit multiplier and the output is then clipped to the rightmost 8-bit integer value; k-means has a 4-bit codebook with each entry a 8-bit value. The selected entry is multiplied with input data and then clipped to the rightmost 8-bit; while for bit-shifting, the input data is shifted right by the range in [1,10] and then clipped the rightmost 8-bit.

It shows that bit-shifting saves the power and area the most. The scaling factor operation consumes ~2× more energy and area of bit-shifting operation. The codebook consumes the power and area most as the codebook contains intensive encoding-decoding operations. Following [20], if in floating-point precision, the computational cost of the quantization for activations is about $\frac{1}{\text{filter size}}$ of the standard convolution layer, only occupying about $1-2\%$ of the whole network computation. However, in fixed-point precision, the computational cost of quantization cannot be ignored. For example, for the scaling factor experiment, the convolution layer is implemented by 8-bit multipliers while the quantization is 32-bit multipliers, the computational cost of quantization would be increased by ~16× and should not be ignored.

2.5 *Discussion*

Figure 2 shows the statistics on mean squared error (MSE) between quantized activations and floating-point activations with residual block depth and the shifting bits with layer depth for ResNet-50. From Figure 2a, we observe that 1) the MSE of residual addition is larger than the first two convolution layers as it integrates last convolution layer and shortcut connection; and 2) when the layer goes deeper, the

MSE error of the first two convolutions in a residual block is stable while the addition units increase. From Figure 2b, it can be seen that 1) the bit-shifting operation operates in a range [1,10]; and 2) the bit-shifting values often revolves around 3 and 8, respectively.

3 Related Work

Recent work on compressing DNNs while accelerating inference can be classified as: a) adopting low-precision of both operands and operations; and b) reducing the number of operations by designing compact architectures or pruning.

Lower precision Some works have been focused on reducing the precision of weights for efficient on-chip memory storage [17, 5]. For instance, [17] proposes an incremental network quantization (INQ) to convert pre-trained full-precision models into low-precision ones by quantizing weights into power of two or zero values. To further save the computing cost and memory storage, recent works also consider the quantization of activation [2, 3, 4, 21]. For instance, binarized neural networks (BNNs) [3] uses binary weights and activations and thus reduces the MAC to an XNOR operation. Different from these methods, a novel dataflow-based joint quantization approach is proposed to quantize the weights and activations.

Fewer operations There is a significant amount of research on designing efficient network architectures [22, 20, 23]. SqueezeNet [22] uses 1×1 instead of 3×3 filters, which dramatically increases the depth of DNNs thus reducing the model complexity. Xception [23] and MobileNet [20] rely on depthwise separable convolutions. Yet another way to reduce the number of operations is through network pruning [6, 24, 25]. As an example, [25] prunes the network based on the magnitude of weights. As the model size of a DNN does not directly reflect the hardware energy consumption, [24] proposes an energy-aware method to prune the weights.

Conclusions

In this work, we explored a dataflow-based joint quantization mechanism to lower the precision of network parameters and activations in DNNs and performed extensive experiments to demonstrate the effectiveness of our proposed methods in terms of performance accuracy and hardware costs. The quantized model is a purely low-precision model with weights, biases and activations in low precision. Besides, the batch normalization is merged into the convolution layer.

References

[1] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S Emer, "Efficient processing of deep neural networks: A tutorial and survey," *Proceedings of the IEEE*, 2017.

[2] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou, "Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients," *arXiv preprint arXiv:1606.06160*, 2016.

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

[4] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *ECCV*, 2016.

[5] Chenzhuo Zhu, Song Han, Huizi Mao, and William J Dally, "Trained ternary quantization," in *ICLR*, 2017.

[6] Song Han, Huizi Mao, and William J Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *ICLR*, 2016.

[7] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *CVPR*, 2018.

[8] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *CVPR*, 2009.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Deep residual learning for image recognition," in *CVPR*, 2016.

[10] Andreas Geiger, Philip Lenz, and Raquel Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *CVPR*, 2012.

[11] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *NIPs*, 2015.

[12] Dongqing Zhang, Jiaolong Yang, Dongqiangzi Ye, and Gang Hua, "Lq-nets: Learned quantization for highly accurate and compact deep neural networks," in *ECCV*, 2018.

[13] Yiren Zhou, Seyed-Mohsen Moosavi-Dezfooli, Ngai-Man Cheung, and Pascal Frossard, "Adaptive quantization for deep neural network," in *AAAI*, 2018.

[14] Song Han, Jeff Pool, John Tran, and William Dally, "Learning both weights and connections for efficient neural network," in *NIPs*, 2015.

[15] S Migacz, "8-bit inference with tensorrt," in *GPU Technology Conference*, 2017.

[16] Frederick Tung and Greg Mori, "Clip-q: Deep network compression learning by in-parallel pruning-quantization," in *CVPR*, 2018.

[17] Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen, "Incremental network quantization: Towards lossless cnns with low-precision weights," *ICLR*, 2017.

[18] Xiaofan Lin, Cong Zhao, and Wei Pan, "Towards accurate binary convolutional neural network," in *NIPs*, 2017.

[19] Naveen Mellempudi, Abhisek Kundu, Dheevatsa Mudigere, Dipankar Das, Bharat Kaul, and Pradeep Dubey, "Ternary neural networks with fine-grained quantization," *arXiv preprint arXiv:1705.01462*, 2017.

[20] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," in *CVPR*, 2017.

[21] Chen Xu, Jianqiang Yao, Zhouchen Lin, Wenwu Ou, Yuanbin Cao, Zhirong Wang, and Hongbin Zha, "Alternating multi-bit quantization for recurrent neural networks," *ICLR*, 2018.

[22] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and¡ 0.5 mb model size," in *ICLR*, 2017.

[23] Francois Chollet, "Xception: Deep learning with depthwise separable convolutions," in *CVPR*, 2017.

[24] Tien-Ju Yang, Yu-Hsin Chen, and Vivienne Sze, "Designing energy-efficient convolutional neural networks using energy-aware pruning," in *CVPR*, 2017.

[25] Song Han, Jeff Pool, John Tran, and William Dally, "Learning both weights and connections for efficient neural network," in *NIPs*, 2015.