Gong C, Lu Y, Dai SR *et al.* AutoQNN: An end-to-end framework for automatically quantizing neural networks. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 39(2): 401-420 Mar. 2024. DOI: 10.1007/s11390-022-1632-9

AutoQNN: An End-to-End Framework for Automatically Quantizing Neural Networks

Cheng Gong[1] (龚　成), *Member, CCF*, Ye Lu[2, 3] (卢　冶), *Senior Member, CCF*
Su-Rong Dai[2] (代素蓉), *Student Member, CCF*, Qian Deng[2] (邓　倩)
Cheng-Kun Du[2] (杜承昆), *Student Member, CCF*
and Tao Li[2, 3, *] (李　涛), *Distinguished Member, CCF, Member, ACM*

[1] *College of Software, Nankai University, Tianjin 300350, China*

[2] *College of Computer Science, Nankai University, Tianjin 300350, China*

[3] *State Key Laboratory of Processors, Institute of Computing Technology, Chinese Academy of Sciences Beijing 100190, China*

E-mail: cheng-gong@nankai.edu.cn; luye@nankai.edu.cn; daisurong@mail.nankai.edu.cn; dengqian@mail.nankai.edu.cn dck@mail.nankai.edu.cn; litao@nankai.edu.cn

Received May 30, 2021; accepted December 26, 2022.

Abstract Exploring the expected quantizing scheme with suitable mixed-precision policy is the key to compress deep neural networks (DNNs) in high efficiency and accuracy. This exploration implies heavy workloads for domain experts, and an automatic compression method is needed. However, the huge search space of the automatic method introduces plenty of computing budgets that make the automatic process challenging to be applied in real scenarios. In this paper, we propose an end-to-end framework named AutoQNN, for automatically quantizing different layers utilizing different schemes and bitwidths without any human labor. AutoQNN can seek desirable quantizing schemes and mixed-precision policies for mainstream DNN models efficiently by involving three techniques: quantizing scheme search (QSS), quantizing precision learning (QPL), and quantized architecture generation (QAG). QSS introduces five quantizing schemes and defines three new schemes as a candidate set for scheme search, and then uses the Differentiable Neural Architecture Search (DNAS) algorithm to seek the layer- or model-desired scheme from the set. QPL is the first method to learn mixed-precision policies by reparameterizing the bitwidths of quantizing schemes, to the best of our knowledge. QPL optimizes both classification loss and precision loss of DNNs efficiently and obtains the relatively optimal mixed-precision model within limited model size and memory footprint. QAG is designed to convert arbitrary architectures into corresponding quantized ones without manual intervention, to facilitate end-to-end neural network quantization. We have implemented AutoQNN and integrated it into Keras. Extensive experiments demonstrate that AutoQNN can consistently outperform state-of-the-art quantization. For 2-bit weight and activation of AlexNet and ResNet18, AutoQNN can achieve the accuracy results of 59.75% and 68.86%, respectively, and obtain accuracy improvements by up to 1.65% and 1.74%, respectively, compared with state-of-the-art methods. Especially, compared with the full-precision AlexNet and ResNet18, the 2-bit models only slightly incur accuracy degradation by 0.26% and 0.76%, respectively, which can fulfill practical application demands.

Keywords automatic quantization, mixed precision, quantizing scheme search, quantizing precision learning, quantized architecture generation

---

Regular Paper

This work is partially supported by the China Postdoctoral Science Foundation under Grant No. 2022M721707, the National Natural Science Foundation of China under Grant Nos. 62002175 and 62272248, the Special Funding for Excellent Enterprise Technology Correspondent of Tianjin under Grant No. 21YDTPJC00380, and the Open Project Foundation of Information Security Evaluation Center of Civil Aviation, Civil Aviation University of China, under Grant No. ISECCA-202102.

*Corresponding Author

1 Introduction

The heavy computational burden immensely hinders the deployment of deep neural networks (DNNs) on resource-limited devices in real application scenarios. Quantization is a technique which compresses DNN weight and activation values from high-precision to low-precision. The low-precision weights and activation occupy smaller memory bandwidth, registers, and computing units, thus significantly improving the computing performance. For example, AlexNet with 32-bit floating-point (FP32) can only achieve the performance of 1.93 TFLOPS on RTX2080Ti due to the bandwidth constraints. However, the low-precision AlexNet with 16-bit floating-point (FP16) can achieve the performance of 7.74 TFLOPS on the same device because the required bandwidth is halved while the available computing units are doubled①. The FP16 AlexNet is four times faster than the FP32 one on RTX2080Ti GPU. Therefore, quantization can reduce the computing budgets in the DNN inference phases and enable the DNN model deployment on resource-limited devices.

However, unreasonable quantizing strategies, such as binary[2] and ternary[3], tend to seriously affect DNN model accuracy[4, 5] and lead to customer frustration. Lower bitwidth usually leads to higher computing performance but larger accuracy degradation. The quantization strategy selection dominates the computing performance and inference accuracy of models. In order to balance the computing performance and inference accuracy, many previous investigations have tried to select a unified bitwidth for all layers in DNNs[6, 7] cautiously. However, many studies show that different layers of DNNs have different sensitivities[8, 9] and computing budgets[10]. Using the same bitwidth for different layers is hard to obtain superior speed-up and accuracy. To strike a fine-grained balance between efficiency and accuracy, it is strongly demanded to explore desirable quantizing schemes[11] and reasonable mixed-precision policies[10, 12] for various neural architectures. The brute force approaches are not feasible for this exploration since the search space grows exponentially with the number of layers[8]. Heuristic explorations work in some scenarios but greatly rely on the heavy workloads of domain experts[11]. Besides, it is impractical to find desirable strategies for various neural architectures through manual-participated heuristic explorations because there are a large number of different neural architectures, and the number of neural architectures is still explosively increasing.

Therefore, it is expected to propose an automatic DNN quantization without manual intervention. The challenge of automatic quantization lies in efficiently exploring the large search space exponentially increasing with the number of layers in DNNs. Many studies made substantial efforts and gained tremendous advances in this area, such as mixed-precision (MixedP)[12], HAQ[10], AutoQB[13], and HAWQ[8]. Nevertheless, there are still some issues that have not been resolved well yet, as follows. Firstly, it is widely acknowledged that different quantizing schemes can impose an impact on the accuracy of quantized DNNs even with the same quantizing bitwidth[4, 14, 15]. However, few studies investigate seeking quantizing schemes for specific architectures, to the best of our knowledge. Secondly, mixed-precision quantization is an efficient way to improve the accuracy of quantized DNNs without increasing the average bitwidth[10, 12, 13]. The presented algorithms include reinforcement learning[10, 13], evolution-based search[16], and hessian-aware methods[8, 9]. However, these algorithms for mixed-precision search are highly complex and inefficient in exploring the exponential search space. The algorithms usually require lots of computing resources, making it challenging to deploy them in online learning scenarios. Besides, these algorithms can easily fall into sub-optimal solutions because they usually skip the search steps of unusual bitwidths, such as a bitwidth of 5[12], to reduce search time.

To address the above issues and challenges, we propose AutoQNN, an end-to-end framework for automatically quantizing neural networks without manual intervention. AutoQNN seeks desirable quantizing strategies for arbitrary architectures by involving three techniques: quantizing scheme search (QSS), quantizing precision learning (QPL), and quantized architecture generation (QAG). The extensive experiments demonstrate that AutoQNN has the ability to find expected quantizing strategies within finite time and consistently outperforms the state-of-the-art quantization methods for various models on ImageNet. Our contributions are summarised as follows.

①The results are computed with the Roofline algorithm[1] using the memory access quantity and operations of an official AlexNet model from the PyTorch community (https://github.com/pytorch/vision) and the peak performance and memory bandwidth of RTX2080Ti GPU.

• We propose QSS to automatically find desirable quantizing schemes for weights and activation with various distributions.

• We present QPL to efficiently learn the relatively optimal mixed-precision policies by minimizing the classification and precision losses.

• We design QAG to automatically convert the computing graphs of arbitrary DNNs into quantized architectures without manual intervention.

• Extensive experiments highlight that AutoQNN can find the expected quantizing strategies to reduce accuracy degradation with low bitwidth.

The rest of this paper is organized as follows. We first discuss related work in Section 2 and then elaborate QSS, QOL, and QAG in Section 3. Next we introduce the end-to-end framework in Section 4, and evaluate AutoQNN in Section 5 and Section 6. Finally, we conclude this paper in Section 7.

2 Related Work

Quantization has been deeply investigated as an efficient approach to boosting DNN computing efficiency. In this section, we first introduce the related quantizing schemes, and then describe the recent mixed-precision quantizing strategies.

2.1 Quantizing Schemes

Binary methods with only one bit, such as BC[17], BNN[2], and Xnor-Net[18], prefer noticeable efficiency improvement. They can compress DNN memory footprint by up to 32x and replace expensive multiplication with cheap bit-operations. However, the binary methods significantly degrade accuracy, since the low bitwidth loses much information. Ternary methods[3, 5, 19, 20] quantize the weights or activation of DNNs into ternaries of $\{-1, 0, 1\}$, aiming at remedying the accuracy degradation of the binary methods without introducing additional overheads. Quaternary methods[6, 7] quantize model weights into four values of $\{-2, -1, 0, 1\}$. They reduce the model accuracy degradation using the same two bits as the ternary methods.

Despite the advanced computing efficiency of the 1-bit or 2-bit methods above, the low bitwidth can significantly affect the model accuracy. This motivates researchers to investigate high bitwidth fixed-point quantization. The proposed method in [21] abandons the last bits of the binary strings of values and keeps the remaining bits as quantized fixed-point values. T-DLA[22] quantizes values into low-precision fixed-point values by reserving the first several significant bits of the binary string of values and dropping the others. However, it is challenging for fixed-point quantization to handle the weights with a high dynamic range.

Zoom quantization methods handle the weights with a high dynamic range by multiplying a full-precision scaling factor. For example, Dorefa-Net[4] and STC[20] first zoom the weights into the range of $[-1, 1]$ by dividing the weights according to their maximum value, and then uniformly map them into continuous integers. Zoom quantization does not deal with the outliers in weights, which increases the quantization loss.

Clip quantization eliminates the impact of outliers on zoom methods by estimating a reasonable range. It truncates weights into the estimated range. The key point of clip quantization is how to balance clip-errors and project-errors[23–25]. PACT[26] reparameterizes the clip threshold to learn a reasonable quantizing range. HAQ[10] minimizes the KL-divergence between the original weight distribution and the quantized weight distribution to seek the optimal clip threshold. μL2Q[7] seeks the optimal quantizing stepsize by minimizing the L2 distance between the original values and the quantized ones.

Besides, there are many studies concerning non-uniform quantization, such as power-of-two (PoT) quantization and residual quantization. PoT methods[15, 27, 28] quantize values into the form of PoT, thus converting the multiplication into the addition[27]. Residual methods[29–32] quantize the residual errors, which are produced by the last quantizing process, into binaries iteratively. Similar methods, such as LQ-Nets[33], ABC-Net[34], and AutoQ[35], quantize the weights/activation into the sum of several binary results.

2.2 Mixed-Precision Strategies

Mixed precision is well known as an efficient quantizing strategy, which quantizes different layers with different bitwidths, thus achieving high accuracy with low average quantizing bitwidth. HAQ[10] leverages reinforcement learning (RL) to automatically determine the bitwidth of layers of DNNs by receiving the hardware accelerator's feedback in the design loop. The mixed-precision method (MixedP)[12]

formulates mixed-precision quantization as a neural architecture search problem. AutoQB[13] introduces deep reinforcement learning (DRL) to automatically explore the space of fine-grained channel-level network quantization. The proposed method in [16] converts each depth-wise convolution layer in MobileNet to several group convolution layers with binarized weights, and employs the evolution-based search to explore the number of group convolution layers. HAWQ[8] and HAWQ-V2[9] employ the second-order information, i.e., top Hessian eigenvalue and Hessian trace of weights/activation, to compute the sensitivities of layers and then design a mixed-precision policy based on these sensitivities. BSQ[36] considers each bit of the quantized weights as an independent trainable variable and introduces a differentiable bit-sparsity regularizer for reducing precision.

3 AutoQNN

In this section, we present three techniques involved in AutoQNN, including quantizing scheme search (QSS), quantizing precision learning (QPL), and quantized architecture generation (QAG).

3.1 Quantizing Scheme Search

We elaborate the automatic quantizing scheme search in this subsection. We first summarize five classical quantizing schemes, then propose three new quantizing schemes, and finally take the eight schemes as a candidate set. We will seek desirable schemes from the candidate set for arbitrary architectures. For ease of notation, we define $\boldsymbol{d}_{\mathrm{f}} \in \mathbb{R}^d$ and $\boldsymbol{d}_{\mathrm{q}} \in \mathbb{Q}^d$ as the original and quantized vectors, respectively. $\mathbb{R}$ is the real space and $\mathbb{Q}$ is the set of quantized values.

3.1.1 *Candidate Schemes*

We divide related quantization methods into eight categories based on their quantizing processes and the format of quantized values: binary, ternary, quaternary, fixed quantization (FixedQ), residual quantization (ResQ), zoom quantization (ZoomQ), clip quantization (ClipQ), and PoT quantization (PotQ).

The first five schemes can be realized easily, and thus we just refer to the existing methods.

Binary. The binary in [18] is summarized as:

$$\boldsymbol{d}_{\mathrm{q}} = \alpha \cdot \mathrm{sign}(\boldsymbol{d}_{\mathrm{f}}) \ \text{ s.t. } \alpha = E(\mathrm{abs}(\boldsymbol{d}_{\mathrm{f}})).$$

Here $\mathrm{sign}(\boldsymbol{d}_{\mathrm{f}}) = (-1)^{\mathbb{I}_{\boldsymbol{d}_i<0}}$ maps the positive elements of $\boldsymbol{d}_{\mathrm{f}}$ to 1 and non-positive ones to -1. $\mathrm{abs}(\boldsymbol{d}_{\mathrm{f}}) = \mathrm{sign}(\boldsymbol{d}_{\mathrm{f}}) \cdot \boldsymbol{d}_{\mathrm{f}}$ converts the elements of $\boldsymbol{d}_{\mathrm{f}}$ into their absolute values. $E(\cdot)$ computes the expectation of input.

Ternary. The ternary proposed in TWN[3] is reorganized as follows:

$$\begin{aligned}\boldsymbol{d}_{\mathrm{q}} &= \alpha \cdot \left\lfloor \left\lfloor \frac{\boldsymbol{d}_{\mathrm{f}}}{2\beta} + \frac{1}{2} \right\rfloor \right\rceil_{-1}^{1} \\ \text{s.t. } \alpha &= E(\mathrm{abs}(\{x | x \in \boldsymbol{d}_{\mathrm{f}}, x > \beta\})), \\ \beta &= 0.7E(\mathrm{abs}(\boldsymbol{d}_{\mathrm{f}})).\end{aligned}$$

Here $\lfloor \cdot \rfloor$ is the floor function, and $\lfloor \cdot \rceil_{-1}^{1}$ truncates the elements of a vector to the range of $[-1, 1]$.

Quaternary. Quaternary quantizes weights into four values of $\{-2, -1, 0, 1\}$. The quaternary definition in [6] is summarized as follows:

$$\begin{aligned}\boldsymbol{d}_{\mathrm{q}} &= \alpha \cdot \left(\left\lfloor \left\lfloor \frac{\boldsymbol{d}_{\mathrm{f}}}{\alpha} \right\rfloor \right\rceil_{-2}^{1} + \frac{1}{2} \right) \\ \text{s.t. } \alpha &= \sqrt{D(\boldsymbol{d}_{\mathrm{f}})}.\end{aligned}$$

Here $D(\cdot)$ computes the variance of input.

FixedQ. FixedQ quantizes values into low-precision fixed-point formats by dropping several bits of the binary strings of values. The FixedQ in [22] is summarized as follows:

$$\begin{aligned}\boldsymbol{d}_{\mathrm{q}} &= \alpha \cdot \left\lfloor \frac{\boldsymbol{d}_{\mathrm{f}}}{\alpha} + \frac{1}{2} \right\rfloor \\ \text{s.t. } \alpha = 2^p, p &= \lfloor \log_2 (\max(\mathrm{abs}(\boldsymbol{d}_{\mathrm{f}}))) \rfloor - (b-2).\end{aligned}$$

Here b is bitwidth. $\max(\cdot)$ finds the maximum element of an input vector. $\log_2(\cdot)$ calculates the logarithm of a scalar with the base of 2.

ResQ. ResQ quantizes the residual errors, which are the quantization errors produced by the last quantizing process, into binaries iteratively. ResQ can be defined as follows:

$$\begin{aligned}\boldsymbol{d}_{\mathrm{q}} &= \sum_{i=1}^{b} B(\boldsymbol{v}_i) \\ \text{s.t. } B(\boldsymbol{v}) &= E(\mathrm{abs}(\boldsymbol{v})) \cdot \mathrm{sign}(\boldsymbol{v}), \\ \boldsymbol{v}_i &= \begin{cases} \boldsymbol{d}_{\mathrm{f}}, & \text{if } i = 1, \\ \boldsymbol{v}_{i-1} - B(\boldsymbol{v}_{i-1}), & \text{if } i = 2, 3, \ldots, b. \end{cases}\end{aligned}$$

The remaining schemes, including ZoomQ, ClipQ, and PotQ, are widely applied and have been realized in a variety of ways. Here we propose three new definitions of them below.

ZoomQ. ZoomQ indicates a group of schemes that uniformly map full-precision values into integers. It is

usually realized by zooming and rounding operations. The new definition of ZoomQ is given below.

$$\boldsymbol{d}_{\mathrm{q}} = \alpha \cdot \left\lfloor \left\lfloor \frac{\boldsymbol{d}_{\mathrm{f}} - \beta}{\alpha} \right\rfloor \right\rceil_{0}^{2^{b}-1} + \beta + \frac{\alpha}{2}$$
$$\text{s.t. } \alpha = \frac{\max(\boldsymbol{d}_{\mathrm{f}}) - \min(\boldsymbol{d}_{\mathrm{f}})}{2^{b}},\ \beta = \min(\boldsymbol{d}_{\mathrm{f}}).$$

Here $\min(\cdot)$ returns the minimum element of an input vector. The quantizing process of ZoomQ is shown in Fig.1(a). We first compute a quantizing interval width α, and then map the full precision $\boldsymbol{d}_{\mathrm{f}}$ into the integer vector based on the obtained α.

ClipQ. ClipQ first truncates values into a target range, and then uniformly maps the values to low-precision representations. We define ClipQ as follows:

$$\boldsymbol{d}_{\mathrm{q}} = \alpha \cdot \left(\left\lfloor \left\lfloor \frac{\boldsymbol{d}_{\mathrm{f}}}{\alpha} \right\rfloor \right\rceil_{-2^{b-1}}^{2^{b-1}-1} + \frac{1}{2} \right)$$
$$\text{s.t. } \alpha = \arg\min_{\alpha} ||\boldsymbol{d}_{\mathrm{f}} - \boldsymbol{d}_{\mathrm{q}}||_2^2.$$

The quantizing process of ClipQ is drawn in Fig.1(b). For simplicity, we compute optimal α for each quantizing bitwidth offline by assuming that $\boldsymbol{d}_{\mathrm{f}}$ satisfies normal distribution[6, 7]. The optimal solutions under different bitwidths are $\alpha \in \{1.283\,2,\ 0.669\,4,\ 0.357\,0,\ 0.193\,9,\ 0.105\,6,\ 0.057\,3,\ 0.030\,8\}$ for $b \in \{2,\ 3,\ 4,\ 5,\ 6,\ 7,\ 8\}$.

PotQ. PotQ is a non-uniform quantizing scheme that quantizes values into the form of PoT. We define PotQ as follows:

$$\boldsymbol{d}_{\mathrm{q}} = \alpha \cdot \mathrm{sign}(\boldsymbol{d}_{\mathrm{f}}) \cdot 2^{e}$$
$$\text{s.t. } e = v - \mathbb{I}_{v=0},$$
$$v = \left\lfloor \left\lfloor \log_2 \left(\frac{\mathrm{abs}\,(\boldsymbol{d}_{\mathrm{f}})}{\alpha} \right) + \frac{1}{2} \right\rfloor \right\rceil_{0}^{2^{b-1}-1}.$$

Here $\mathbb{I}_{v=0}$ maps all the zeros in v into ones and the non-zero values into zeros. The quantized values and the quantizing process of PotQ are shown in Fig.1(c). Assuming that the inputs are normally distributed, we can obtain the optimal solutions offline that are $\alpha \in \{1.224\,0,\ 0.518\,1,\ 0.038\,1\}$ for $b \in \{2,\ 3,\ 4\}$[②].

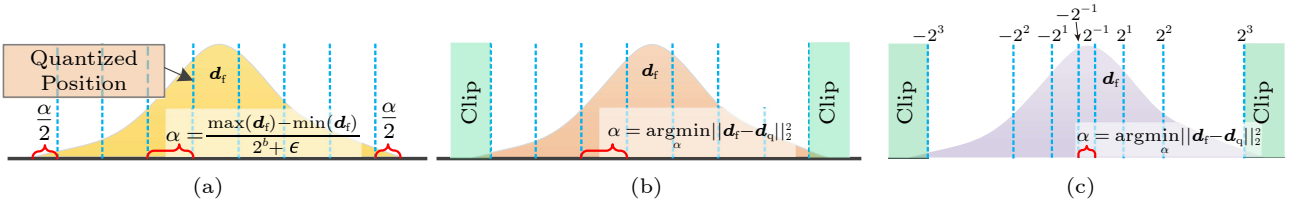
The quantization loss comparisons of the above candidate schemes handling different distributions are shown in Fig.2, and the detailed comparisons of these schemes are presented in Table 1. The results show that the schemes perform widely divergent on different distributions. No scheme can always achieve the minimum quantization loss across various distributions. However, the weights and activation of each layer in DNNs tend to distribute differently. This observation implies that employing an undesirable quantizing scheme will make it difficult to fit the distributions and degrade the accuracy of DNNs. Therefore, seeking desirable quantization for specific layers or models is strongly demanded.

3.1.2 *Seeking Scheme*

We denote the computing graph of a neural architecture as $G(V, E)$ and the corresponding quantized graph as $G'(V', E')$. G' is generated from G by adding the quantizing vertices, that is, $V \subset V'$. $V' - V$ is the set that consists of the quantizing vertices. $N = |V' - V|$ is the number of the added quantizing vertices. We seek suitable schemes for these quantizing vertices to reduce the accuracy degradation in quantization.

We denote all the schemes introduced in Subsection 3.1.1 as a candidate set $\{Q_1^l,\ Q_2^l,\ Q_3^l, \ldots,\ Q_n^l\}$. Let $v_i, v_j \in V'$, $v_q^l \in V' - V$ be the l-th quantizing vertex, and $\langle v_i, v_q^l \rangle, \langle v_q^l, v_j \rangle \in E'$. The implementation of v_q^l should be a scheme Q_k^l selected from the candidate set. Let $\boldsymbol{d}_{\mathrm{f}}^{\,l}$ and $\boldsymbol{d}_{\mathrm{q}}^{l}$ denote the original vector and quantized vector, respectively. A quantizing process of the scheme Q_k^l is denoted as $\boldsymbol{d}_{\mathrm{q}}^{l} = Q_k^l(\boldsymbol{d}_{\mathrm{f}}^{\,l})$.

Quantizing scheme search (QSS) is proposed to seek a desirable quantizing scheme Q_k^l from the candidate set and quantize $\boldsymbol{d}_{\mathrm{f}}^{\,l}$ into $\boldsymbol{d}_{\mathrm{q}}^{l}$ with small accuracy degradation. We employ the sampling way as de-



(a) (b) (c)

Fig.1. Quantizing process of our proposed new schemes, including (a) zoom quantization, (b) clip quantization, and (c) PoT quantization.

[②]Since the intervals of quantized values in PotQ grow exponentially with the bitwidth increasing, the maximum bitwidth should be less than or equal to 4[23].

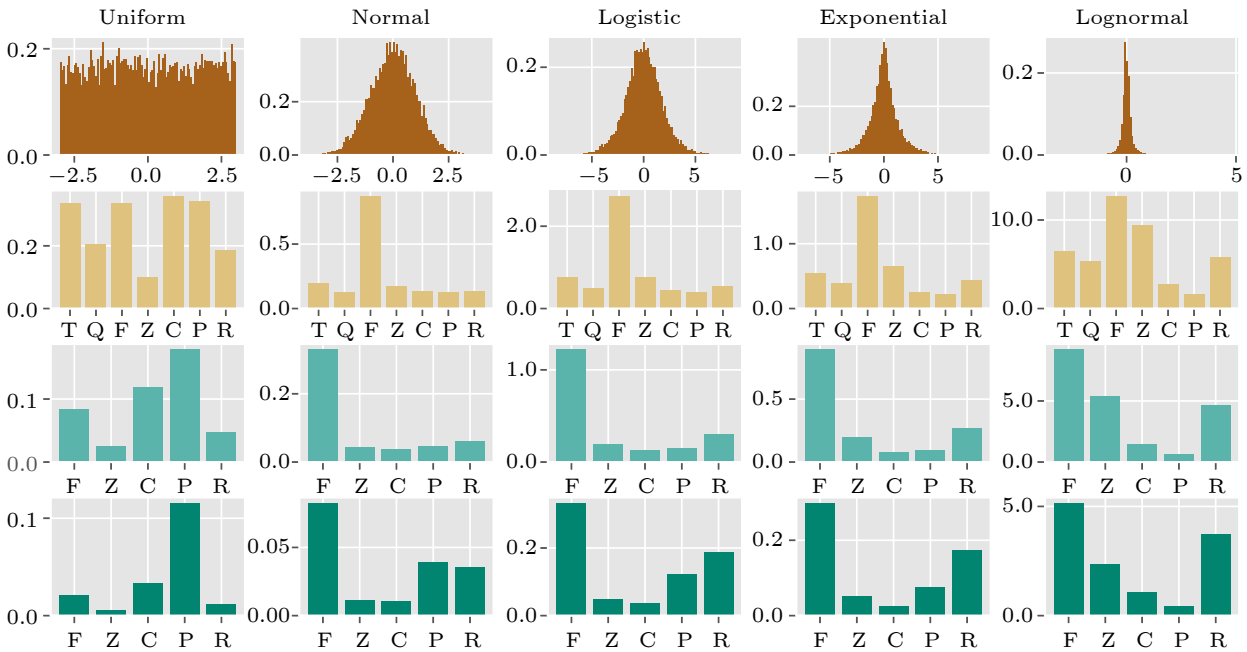


Fig.2. Quantization loss comparisons of candidate schemes across various data distributions and quantizing bits. The candidate schemes include ternary (T), quaternary (Q), FixedQ (F), ZoomQ (Z), ClipQ (C), PotQ (P) and ResQ (R). Quantization loss is the L2 distance between values before and after quantization. The first row shows different data distributions. The second line presents 2-bit quantization losses. The third line is 3-bit quantization losses, and the last line is 4-bit quantization losses.

Table 1. Detailed Information of the Eight Candidate Schemes

Scheme	Number of Bits	TC	SC	Distribution	Format	Reference
Binary	1	$O(n)$	$O(n)$	–	Binary	[2, 17, 18]
Ternary	2	$O(n)$	$O(n)$	Normal	Ternary	[3, 5, 20]
Quaternary	2	$O(n)$	$O(n)$	Normal	Quaternary	[6, 7]
FixedQ	2–8	$O(n)$	$O(n)$	Uniform	Fixed-point	[21, 22]
ZoomQ	2–8	$O(n)$	$O(n)$	Uniform	Integer	[4]
				Normal		
ClipQ	2–8	$O(n)$	$O(n)$	Normal	Integer	[25, 26]
				Logistic		[10, 24]
				Exponential		[7]
PotQ	2–4	$O(n)$	$O(n)$	Log-normal	PoT	[15, 27, 28]
ResQ	2–8	$O(nb)$	$O(nb)$	Uniform	Sum of binaries	[29, 33, 34]

Note: TC is the time complexity of quantizing scheme. n is the scale of input and b is the number of quantizing bits. SC is the space complexity of quantizing scheme. Distributions denote the distributions wanted by quantizing schemes as shown in Fig.2.

scribed in DNAS[12] to find desirable schemes. As shown in Fig.3, we construct state parameters $\boldsymbol{\theta}^l = (\theta_1^l,\ \theta_2^l, \ldots, \theta_n^l)^{\mathrm{T}}$ that correspond to the quantizing schemes, and sample a quantizing scheme with the probabilities $P_{\boldsymbol{\theta}}$ before each training phase. The sampling process can be defined as follows:

$$\begin{aligned} &\boldsymbol{d}_{\mathrm{q}}^l = Q_k^l(\boldsymbol{d}_{\mathrm{f}}^{\,l}) \\ &\quad \text{s.t. } k \sim P_{\boldsymbol{\theta}^l},\ P_{\boldsymbol{\theta}^l} = \mathrm{softmax}(\boldsymbol{\theta}^l). \end{aligned}$$

Here $\mathrm{softmax}(\cdot)$ maps values into probabilities.

Unfortunately, the conventional sampling process $k \sim P_{\boldsymbol{\theta}^l}$ is non-differentiable, which means the sampling result is hard to guide the optimization of state parameters $\boldsymbol{\theta}^l$. Therefore, we employ the Gumbel-Softmax[37] to realize a differentiable sampling process:

$$\begin{aligned} &\boldsymbol{d}_{\mathrm{q}}^l = Q_k^l(\boldsymbol{d}_{\mathrm{f}}^{\,l}) \\ &\quad \text{s.t. } k = \arg\max_k (P_{\boldsymbol{\theta}^l + \boldsymbol{g}^l}), \\ &\qquad P_{\boldsymbol{\theta}^l + \boldsymbol{g}^l} = \mathrm{softmax}(\boldsymbol{\theta}^l + \boldsymbol{g}^l),\ \boldsymbol{g}^l \sim \mathrm{Gumbel}(0,1)^n. \end{aligned} \tag{1}$$

Here $g_k^l \in \boldsymbol{g}^l$ is a value drawn from the Gumbel distri-

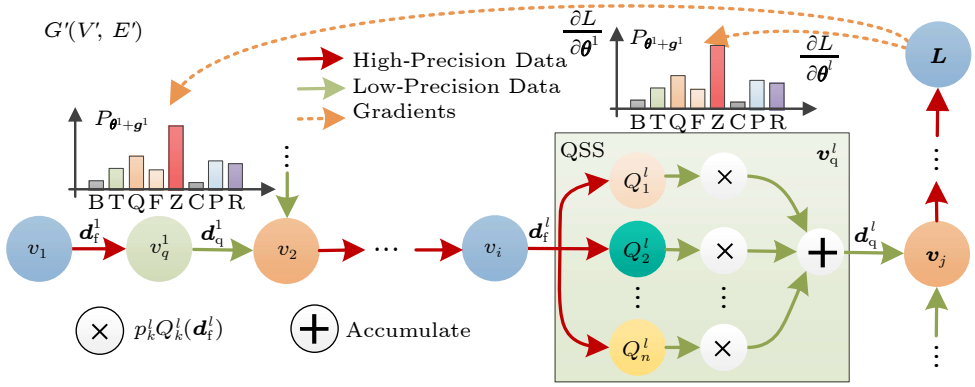


Fig.3. Quantizing scheme search process. Reparameterization is adopted and we optimize the state parameters to seek desirable quantizing schemes.

bution. The sampling process in (1) is differentiable and the gradients of $\boldsymbol{d}_{\mathrm{q}}^{l}$ with respect to $\theta_k^l \in \boldsymbol{\theta}^l$ can be derived. However, the gradient of the "hard" sampling process of $\arg\max(\cdot)$ is zero everywhere, and the state parameters cannot be optimized. To solve this problem, we introduce a "soft" sampling process that adopts the expectation of $Q_k^l(\boldsymbol{d}_{\mathrm{f}}^{l})$. Finally, the seeking scheme of QSS is defined as follows:

$$\begin{aligned}\boldsymbol{d}_{\mathrm{q}}^{l} &= \sum_{k=1}^{n} p_k^l Q_k^l(\boldsymbol{d}_{\mathrm{f}}^{l}) \\ \text{s.t. } p_k^l &\in \operatorname{softmax}\left(\frac{\boldsymbol{\theta}^l + \boldsymbol{g}^l}{\tau}\right),\ \boldsymbol{g}^l \sim \operatorname{Gumbel}(0,1)^n.\end{aligned} \tag{2}$$

Here τ is a temperature coefficient. As $\tau \to \infty$, each output of $Q_k^l(\boldsymbol{d}_{\mathrm{f}}^{l})$ has the same weight p_k^l and $\boldsymbol{d}_{\mathrm{q}}^{l}$ equals the averaged outputs. Thus all of the state parameters can be optimized simultaneously. As $\tau \to 0$, $\boldsymbol{d}_{\mathrm{q}}^{l}$ is equivalent to the sampling result in (1). We smoothly decay τ from a large value to 0 to select a desirable quantizing scheme for v_q^l. The decaying strategy of τ is defined as follows:

$$\tau = \tau_0 \times (1 - \delta/\Delta)^p. \tag{3}$$

τ_0 is the initial temperature. δ is the current training epoch and Δ is the number of total training epochs. p is the exponential coefficient.

Let the final loss of $G'(V', E')$ be L. Since (2) is differentiable, the gradients of L with respect to $\boldsymbol{\theta}^l$ can be computed as follows:

$$\frac{\partial L}{\partial \boldsymbol{\theta}^l} = \frac{\partial L}{\partial \boldsymbol{d}_{\mathrm{q}}^{l}} \frac{\partial \boldsymbol{d}_{\mathrm{q}}^{l}}{\partial \boldsymbol{\theta}^l}.$$

It means that a gradient-descent algorithm can be employed to optimize the state parameters $\boldsymbol{\theta}^l$. Therefore, we can seek a quantizing scheme for each quantizing vertex v_q^l to minimize the loss of neural architecture, so as to improve accuracy.

In addition, different quantizing schemes require distinct accelerating implementations, such as the bit-operations for binaries/ternaries, low-bit multiplication for fixed-point values, and shift operations for PoTs. Implementing all of these operations is unreasonable for resource-constrained devices. To simplify the hardware implementation of accelerators, searching for one shared quantizing scheme for all the vertices in one neural architecture is still demanded. We realize the coarse-grained QSS by maintaining and sharing one group of state parameters $\boldsymbol{\theta}$ for all quantizing vertices in $G'(V', E')$. In coarse-grained QSS, the gradients of L with respect to $\boldsymbol{\theta}$ are computed as:

$$\frac{\partial L}{\partial \boldsymbol{\theta}} = \sum_{l=1}^{N} \frac{\partial L}{\partial \boldsymbol{d}_{\mathrm{q}}^{l}} \frac{\partial \boldsymbol{d}_{\mathrm{q}}^{l}}{\partial \boldsymbol{\theta}}.$$

3.2 Quantizing Precision Learning

Quantizing precision, i.e., the bitwidth, is an essential attribute of quantizing schemes and determines the number of quantized values. Selecting an optimal precision for a specific quantizing scheme is vital for balancing the efficiency and accuracy of quantized DNNs. In this subsection, we reparameterize the quantizing precision and learn the relatively optimal mixed-precision model within a limited model size and memory footprint.

3.2.1 *Bitwidth Reparameterization*

Let b be the number of the quantizing bits and $\mathbb{Q}$ be the quantized value set, and we have $|\mathbb{Q}| = 2^b$. A general quantizing process of $\boldsymbol{d}_{\mathrm{q}} = Q_k(\boldsymbol{d}_{\mathrm{f}})$ can be represented as two stages: 1) mapping the full-precision weights or activation to the quantized values that be-

long to $\mathbb{Q}$, and then 2) scaling the quantized values into a target range, as shown in Fig.4. Here we expand the general quantizing process as following formulations.

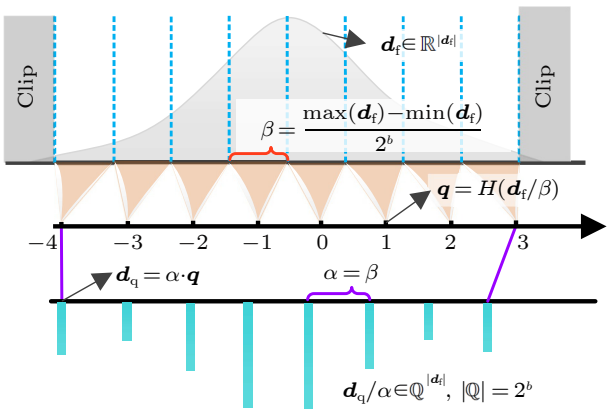


Fig.4. General quantizing process: mapping full-precision values to integers and then scaling these integers into a target range.

$$\boldsymbol{d}_{\mathrm{q}} = Q_k(\boldsymbol{d}_{\mathrm{f}}) = \alpha H\left(\frac{\boldsymbol{d}_{\mathrm{f}}}{\beta}\right)$$
$$\text{s.t. } H : \mathbb{R}^{|\boldsymbol{d}_{\mathrm{f}}|} \to \mathbb{Q}^{|\boldsymbol{d}_{\mathrm{f}}|}, \ |\mathbb{Q}| = 2^b. \tag{4}$$

Here $|\boldsymbol{d}_{\mathrm{f}}|$ and $|\boldsymbol{d}_{\mathrm{q}}|$ are the number of elements of $\boldsymbol{d}_{\mathrm{f}}$ and $\boldsymbol{d}_{\mathrm{q}}$, respectively, and $|\boldsymbol{d}_{\mathrm{f}}| = |\boldsymbol{d}_{\mathrm{q}}|$. $\mathbb{R}$ is the set of real numbers. H is a function that projects the vector with real values into that with quantized values. α is the scaling factor and β is the average step size of the quantizing scheme Q_k. In general, α varies linearly with β to maintain a similar range of $\boldsymbol{d}_{\mathrm{f}}$ and $\boldsymbol{d}_{\mathrm{q}}$, thereby reducing quantization loss.

$$\alpha = \lambda \times \beta.$$

Typically, $\lambda = 1$ and $\alpha = \beta$[4, 7, 10, 38]. The average step size β can be calculated using the quantizing precision b and the range of $\boldsymbol{d}_{\mathrm{f}}$ as follows:

$$\beta = \frac{\max(\boldsymbol{d}_{\mathrm{f}}) - \min(\boldsymbol{d}_{\mathrm{f}})}{2^b}.$$

Therefore, the loss L of $G'(V', E')$ is differentiable with respect to the quantizing precision b. The gradient $(\partial L)/(\partial b)$ can be calculated as follows:

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial \boldsymbol{d}_{\mathrm{q}}} \frac{\partial \boldsymbol{d}_{\mathrm{q}}}{\partial \alpha} \frac{\partial \alpha}{\partial \beta} \frac{\partial \beta}{\partial b}$$
$$= -\frac{\partial L}{\partial \boldsymbol{d}_{\mathrm{q}}} \frac{\max(\boldsymbol{d}_{\mathrm{f}}) - \min(\boldsymbol{d}_{\mathrm{f}})}{2^b (\ln 2)^{-1}} \left(H\left(\frac{\boldsymbol{d}_{\mathrm{f}}}{\lambda\alpha}\right) - \frac{\boldsymbol{d}_{\mathrm{f}}}{\lambda\alpha} \right). \tag{5}$$

We employ the straight through estimator (STE)[4] to compute the differential of $H(\cdot)$, i.e., $H'(\cdot) = 1$. Based on (5), we get the gradient of L with respect to the quantizing precision b, which means that we can reparameterize b and employ the gradient-descent algorithm to learn a reasonable quantizing precision.

3.2.2 *Precision Loss*

Generally, DNNs tend to learn high-precision weights or activation by minimizing L. It implies that the values of the quantizing precision b for DNNs will be optimized to be great ones, such as 32 bits or 64 bits, instead of small ones. To learn a policy with low average quantizing precision, we propose precision loss $\overline{L}$, which measures the distance between the average quantizing precision and a target precision $\overline{b}$.

$$\overline{L} = (E(\mathbb{B}) - \overline{b})^2$$
$$\text{s.t. } \mathbb{B} = \left\{ b_i | i = 1, 2, \ldots, \sum_{l=1}^{N} |\boldsymbol{d}_{\mathrm{q}}(l)| \right\}. \tag{6}$$

$\overline{b}$ is the expected quantizing bitwidth of a neural architecture, and preset before a precision learning phase. b_i is the bitwidth of one weight or activation value in neural architecture $G'(V', E')$. $\mathbb{B}$ is the set consisting of b_i. $N = |V' - V|$ is the number of quantizing vertices as described in Subsection 3.1. The gradient of $L + \overline{L}$ with respect to b is defined as follows:

$$g_b = \frac{\partial L}{\partial b} + \frac{\partial \overline{L}}{\partial b}.$$

3.3 Quantized Architecture Generation

We denote the computing graph of a neural architecture as $G(V, E)$. V is the set of the vertices and $v_i \in V$ indicates a vertex. E is the set of edges and $\langle v_i, v_j \rangle \in E$ indicates that there is a tensor $\boldsymbol{d}^{i,j}$ flowing from v_i to v_j. $Id(v_i)$ and $Od(v_i)$ compute the in-degree and out-degree of v_i, respectively. v_i is a data vertex when $Id(v_i) = 0$, such as input image, feature map, and weight. v_i is an operation vertex when $Id(v_i) > 0$, such as convolution, batch normalization, full-connection, and so on.

Let $\boldsymbol{d}^{i,j}$ denote a tensor that flows into a time-consuming and/or memory-consuming vertex, such as the weight tensor of convolutional (Conv) vertices and full-connected (FC) vertices. The target of quantization is to reduce the precision of $\boldsymbol{d}^{i,j}$ and shrink the computing consumption of DNNs. These time-consuming and/or memory-consuming vertices are called expensive vertices. $V_{\mathrm{e}} \subset V$ is the expensive ver-

tex set to be quantized. The Conv and FC vertices occupy more than 99% of operations and memory footprint in mainstream DNN architectures and applications. According to the above observations, we set the Conv and FC vertices as the default expensive vertices, that is, $V_e = \{v_i | v_i \in V, v_i \text{ is Conv or FC}\}$.

We design the quantized architecture generation (QAG) algorithm to automatically reconstruct the computing graph $G(V, E)$ into its quantized counterpart $G'(V', E')$ requiring less memory footprint and few computing overheads. As shown in Algorithm 1, for a specific DNN with computing graph $G(V, E)$ and the expensive vertex set V_e, QAG first collects the vertices with 0 in-degree as the initial vertex set V' and initializes the edge set $E' = \{\}$. Let $I = \{v_j | v_i \in V', v_j \in V - (V' \cap V), \langle v_i, v_j \rangle \in E\}$, and we have $I \neq \phi$ when $V \neq V' \cap V$, since $G = (V, E)$ is a connected graph. Then we move the vertices from I to V' iteratively. During this process, we embed quantizing vertices into the edges flowing into expensive vertices.

The original $G(V, E)$ can be automatically reconstructed as $G'(V', E')$ within $|V|$ iterations, and the tensors flowing into the expensive vertices of V_e are replaced with low-precision representations by embedding quantizing vertices. Therefore, the new architecture $G'(V', E')$ requires less computing budget than $G(V, E)$. The average time and space complexity of Algorithm 1 are $O(|V||E|)$ and $O(|V| + |E|)$, respectively.

4 End-to-End Framework

The proposed QAG in Algorithm 1 can significantly reduce the quantization workload and avoid prone errors, so as to provide fast, cheap, and reliable quantized architecture generation. We finally implement an end-to-end framework based on the three techniques in AutoQNN, i.e., QAG, QSS, and QPL. As shown in Fig.5 and Algorithm 2, AutoQNN is constructed with three stages corresponding to the three techniques and takes a full-precision architecture as input. AutoQNN first reconstructs the input architecture into a quantized one through QAG. Then it trains total Δ epochs to seek desirable quantizing schemes through QSS automatically. Finally, AutoQNN retrains/fine-tunes the quantized architecture with QPL to converge.

Algorithm 1. Quantized Architecture Generation (QAG)

Input: computing graph $G(V, E)$, expensive vertex set $V_e \subset V$
Output: quantized architecture $G'(V', E')$

$V' \leftarrow \{v_i | v_i \in V, Id(v_i) = 0\}$
$E' \leftarrow \{\}$
$I \leftarrow \{v_j | v_i \in V', v_j \in V - (V' \cap V), \langle v_i, v_j \rangle \in E\}$
while $I \neq \phi$ **do**
  for $v_j \in I$ **do**
    $I' \leftarrow \{v_k | v_k \in V, \langle v_k, v_j \rangle \in E\}$
    if $I' \subseteq V'$ **then**
      $V' \leftarrow V' \cup \{v_j\}$
      for $v_k \in I'$ **do**
        if $v_j \in V_e$ **then**
          create quantizing vertex v_{q_k}
          $V' \leftarrow V' \cup \{v_{q_k}\}$
          $E' \leftarrow E' \cup \{\langle v_k, v_{q_k} \rangle, \langle v_{q_k}, v_j \rangle\}$
        else
          $E' \leftarrow E' \cup \{\langle v_k, v_j \rangle\}$
        end if
      end for
    end if
  end for
  $I \leftarrow \{v_j | v_i \in V', v_j \in V - (V' \cap V), \langle v_i, v_j \rangle \in E\}$
end while

Algorithm 2. AutoQNN Framework

Input: computing architecture $G(V, E)$, expensive vertices $V_e \subset V$, candidate quantizing schemes Q, dataset $\mathcal{X}$, training epochs Δ
Output: quantized architecture $G'(V', E')$

Generating $G'(V', E')$ with Algorithm 1
for $\delta = 1$ to Δ **do**
  Decaying τ with (3)
  Training G' on $\mathcal{X}$ with respect to weights and $\boldsymbol{\theta}$
end for
Sampling the quantizing schemes
Training the weights and b of G' on $\mathcal{X}$ to converge

5 Evaluation

In order to train DNN models and evaluate their performance conveniently, we implement AutoQNN and integrate it into Keras[③] (v2.2.4). Furthermore, we implement the eight candidate quantizing schemes presented in Subsection 3.1. For ease of notation, we use the combination of the scheme name and bitwidth to denote one quantizing strategy. For example, "P-3" indicates a PotQ quantizer with a bitwidth of 3. All quantized architectures are automatically constructed by AutoQNN in our experiments. In addition, respecting that most DNNs have employed ReLU[39] to eliminate the negative elements of activa-

[③] https://github.com/fchollet/keras, Mar. 2024.

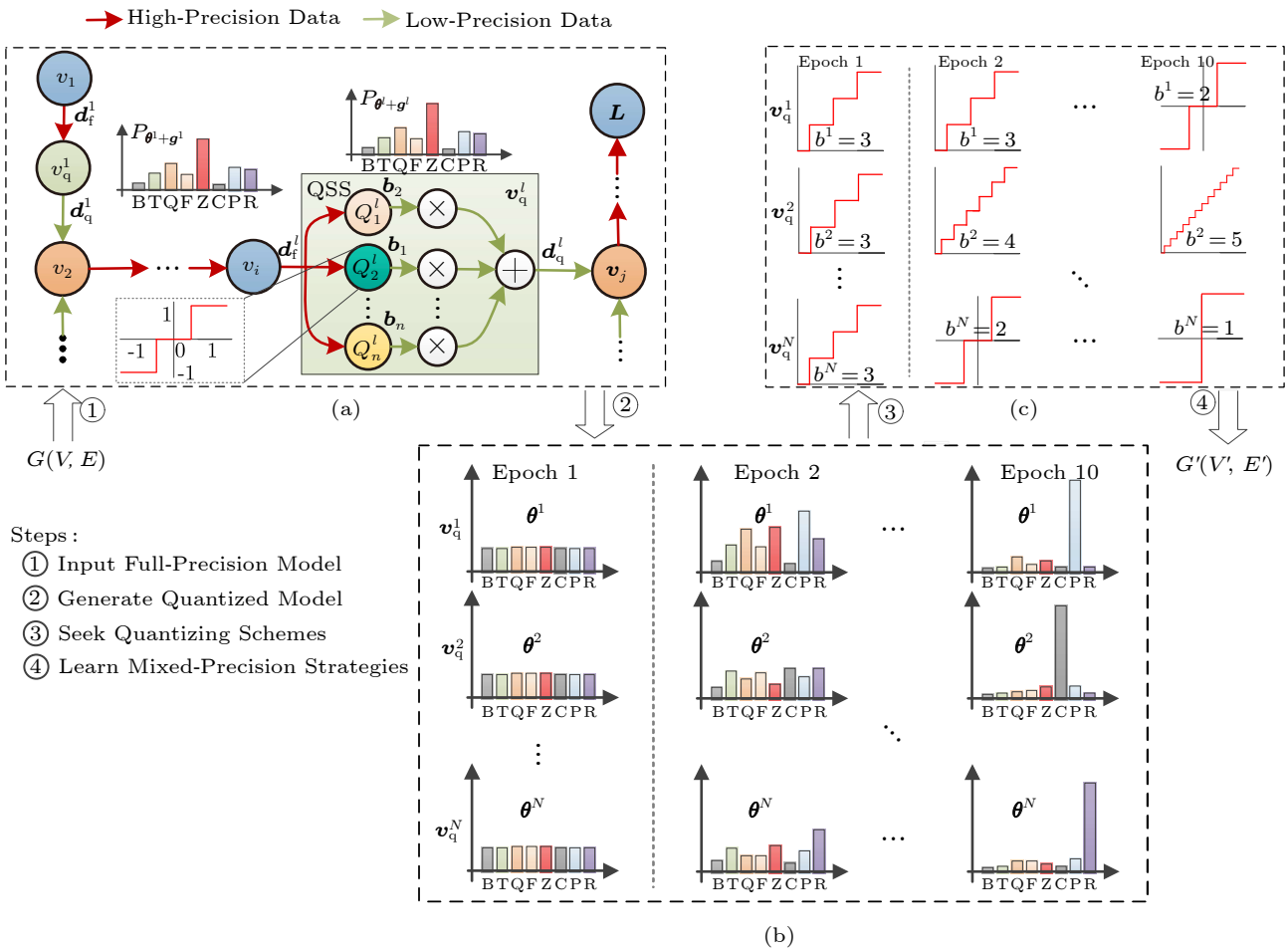


Fig.5. End-to-end neural network quantization. Input is a full-precision architecture $G(V, E)$ and output is the efficient quantized architecture $G'(V', E')$ with desirable quantizing strategies. (a) Generating a quantized architecture with QAG. (b) Seeking quantizing schemes with QSS. (c) Learning precision with QPL.

tion, we do not use binary, ternary, or quaternary schemes in activation quantization.

5.1 Image Classification

Image classification provides the basic cues for many computer vision tasks, and thus the results of image classification are representative for the evaluation of quantization. For a fair comparison with the state-of-the-art quantization[23, 25], we quantize the weights and activation of all layers except the first and last layers.

5.1.1 *Dataset and Models*

We conduct experiments with widely used models on the ImageNet[40] dataset, including AlexNet[41], ResNet18[31], ResNet50[31], MobileNets[42], MobileNetV2[43], and InceptionV3[44]. To make a fair comparison and ensure reproducibility, the full-precision models used in our experiments refer to the full-precision pre-trained models obtained from open sources. Specifically, AlexNet refers to [45], and ResNet18 cites from literature [46]. Other models are obtained from official Keras community④. Referenced accuracy results of the full-precision models used in our experiments are also from the open-source results, and no further fine-tuning is made. The data argumentation of ImageNet can be found here⑤.

5.1.2 *Baseline Methods*

We select the state-of-the-art fixed-precision quantizers including TWNs[3], TTQ[14], INQ[28], ENN[15],

④https://github.com/fchollet/keras, Mar. 2024.
⑤https://github.com/tensorflow/models, Mar. 2024.

μL2Q[7], VecQ[6], TSQ[24], DSQ[47], PACT[26], Dorefa-Net[4], LQ-Nets[33], QIL[25], APoT[23], BRQ and TRQ[32], and BCGD[48], and mixed-precision quantization methods including MixedP[12], AutoQB-QBN and AutoQB-BBN[13], HAQ[10], BSQ[36], HAWQ[8] and HAWQ-V2[9], as the baselines for comparison.

5.1.3 *Quantizing Strategy Search*

For quantizing scheme search, we configure all the candidate schemes with the same low bitwidth of 3, since low-bitwidth quantization can highlight the difference among candidate schemes, ensuring the robustness of searching results. We train the models with coarse-grained QSS by 10 epochs to find desirable quantizing schemes. Then, based on the found quantizing schemes, we train 60 epochs to learn a reasonable quantizing precision for each layer.

The quantizing strategies found by QSS and QPL are shown in Table 2. On both AlexNet and ResNet18, PotQ is the desirable scheme for weight quantization, and ClipQ is the scheme for activation quantization. The reason for the results is that most of the weights are Log-Normal distributed, and PotQ is the best one for handling this distribution, as described in Subsection 3.1. The activation distributions of ResNet18 and AlexNet are mostly bell-shaped, and ClipQ performs well on bell-shaped distributions, such as normal, logistic, and exponential distributions described in Subsection 3.1. Therefore, PotQ and ClipQ are selected as the solutions for the weight and activation quantization, respectively.

Besides, QPL learns mixed-precision strategies. For a fair comparison with state-of-the-art fixed-precision schemes, we compute the average bitwidth for representing weights and activation of models as conditions in our experiments, since the models with the same average bitwidth consume the same memory footprint and storage in model inference.

5.1.4 *Comparison with State-of-the-Art Quantization*

We compare AutoQNN with state-of-the-art fixed-precision quantization across various quantizing bits and the results are shown in Table 3. The results show that AutoQNN can consistently outperform the compared methods with much higher inference accuracies. At extreme conditions with only 2 bits for weights and activation, the accuracy of AutoQNN on AlexNet is 59.75%, slightly lower than the accuracy of the referenced full-precision AlexNet by 0.26%. The accuracy also exceeds that of QIL by 1.65%, and outperforms that of VecQ by 1.27%. Besides, AutoQNN achieves an accuracy of 68.84% on ResNet18 using only 2 bits for all weights and activation. The result is slightly less than the accuracy of the referenced full-precision model by 0.76% and exceeds APoT and VecQ by 1.74% and 0.61%, respectively. AutoQNN achieves the highest accuracy results of 61.85% and 62.63% when quantizing AlexNet into a 3-bit model and a 4-bit model correspondingly. AutoQNN also realizes the best results of 69.88% and 70.36% when quantizing the weights and activation of ResNet18 into 3 bits and 4 bits correspondingly. It is worth noting that the results of quantized models may exceed the specific referenced accuracy results by fine-tuning. Still, quantization may harm accuracy, and the quantized models cannot perform better than full-precision models in accuracy theoretically.

The experiments demonstrate that AutoQNN can automatically seek desirable quantizing strategies to reduce accuracy degradation under the different conditions of model size and memory footprint, thus achieving a new balance between accuracy and efficiency in DNN quantization.

Table 2. Optimal Quantizing Strategies for AlexNet and ResNet18

Model	Notation (W/A)	Quantizing Bits of Layer Weights	Quantizing Bits of Layer Activation	Average Bits (W/A)
AlexNet	P-2/C-2	4, 4, 4, 4, 2, 2	2, 2, 2, 2, 3, 3	2.13/2.01
	P-3/C-3	4, 4, 4, 4, 3, 3	3, 3, 4, 3, 4, 4	3.06/3.20
	P-4/C-4	4, 4, 4, 4, 4, 4	3, 3, 3, 5, 5, 5	4.00/4.05
ResNet18	P-2/C-2	4, 3, 4, 4, 3, 3, 4, 4, 2, 2, 4, 2, 2, 2, 2, 2, 2, 4, 2, 2	3, 2, 3, 2, 2, 3, 3, 2, 2, 2, 3, 3, 2, 3, 3, 3, 3, 3, 3, 3	2.13/2.49
	P-3/C-3	4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 2, 4, 3, 2	4, 3, 3, 2, 2, 4, 4, 2, 2, 3, 4, 4, 2, 3, 4, 5, 6, 4, 5, 4	2.94/3.07
	P-4/C-4	4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4	5, 4, 5, 3, 3, 4, 4, 3, 3, 3, 5, 6, 3, 4, 5, 7, 7, 5, 7, 6	4.00/4.08

Note: The first and last layers are not quantized. W/A denotes the results for weights and activation, respectively. The maximum quantizing bitwidth of PotQ is 4.

Table 3. Accuracy Comparison with State-of-the-Art Methods

QB	Method	W/A	AlexNet		ResNet18	
			Top 1	Top 5	Top 1	Top 5
32	Referenced	32/32	60.01	81.90	69.60	89.24
2	TWNs	2/32	57.50	79.80	61.80	84.20
	TTQ	2/32	57.50	79.70	66.60	87.20
	INQ	2/32	–	–	66.02	87.12
	ENN	2/32	58.20	80.60	67.00	87.50
	μL2Q	2/32	–	–	65.60	86.12
	VecQ	2/32	58.48	80.55	68.23	88.10
	TSQ	2/2	58.00	80.50	–	–
	DSQ	2/2	–	–	65.17	–
	PACT	2/2	55.00	77.70	64.40	85.60
	Dorefa-Net	2/2	46.40	76.80	62.60	84.40
	LQ-Nets	2/2	57.40	80.10	64.90	85.90
	QIL	2/2	58.10	–	65.70	–
	APoT	2/2	–	–	67.10	87.20
	BRQ	2/2	–	–	64.40	–
	TRQ	2/2	–	–	63.00	–
	AutoQNN	P-2/C-2	59.75	81.72	68.84	88.50
3	INQ	3/32	–	–	68.08	88.36
	ENN-2	3/32	59.20	81.80	67.50	87.90
	ENN-4	3/32	60.00	82.40	68.00	88.30
	VecQ	3/32	58.71	80.74	68.79	88.45
	DSQ	3/3	–	–	68.66	–
	PACT	3/3	55.60	78.00	68.10	88.20
	Dorefa-Net	3/3	45.00	77.80	67.50	87.60
	ABC-Net	3/3	–	–	61.00	83.20
	LQ-Nets	3/3	–	–	68.20	87.90
	QIL	3/3	61.30	–	69.20	–
	APoT	3/3	–	–	69.70	88.90
	BRQ	3/3	–	–	66.10	–
	AutoQNN	P-3/C-3	61.85	83.47	69.88	89.07
4	INQ	4/32	–	–	68.89	89.01
	μL2Q	4/32	–	–	65.92	86.72
	VecQ	4/32	58.89	80.88	68.96	88.52
	DSQ	4/4	–	–	69.56	–
	PACT	4/4	55.70	78.00	69.20	89.00
	Dorefa-Net	4/4	45.10	77.50	68.10	88.10
	LQ-Nets	4/4	–	–	69.30	88.80
	QIL	4/4	62.00	–	70.10	–
	BCGD	4/4	–	–	67.36	87.76
	TRQ	4/4	–	–	65.50	–
	AutoQNN	P-4/C-4	62.63	83.93	70.36	89.43

Note: QB is the quantizing bitwidth. W/A denotes the weight/activation bitwidth. The sign - indicates that the results cannot be found. The best results are underlined.

5.1.5 *Comparison with Mixed-Precision Schemes*

We verify that the proposed AutoQNN can outperform state-of-the-art mixed-precision schemes and gain higher accuracy in DNN quantization in this experiment. According to the results in Subsection 5.1.4, we employ PotQ for weight quantization and ClipQ for activation quantization. We seek reasonable mixed-precision policies by QPL and compare them with the mixed-precision schemes under the same average bitwidth condition.

The comparison results are shown in Table 4. Compared with MixedP[12] on ResNet18, AutoQNN achieves a higher accuracy of 68.84% with the lower

Table 4. Accuracy Comparison with Mixed-Precision Quantization

Model	Method	W/A	Accuracy (%)
ResNet18	MixedP	>2.00/4.00	68.65
	AutoQNN	2.13/2.49	68.84
ResNet18	AutoQB-QBN	3.12/3.29	67.63
	AutoQB-BBN	3.06/3.27	63.41
	AutoQNN	2.94/3.07	69.88
MobileNets	HAQ	2.16/32.00	57.14
	AutoQNN	2.28/32.00	69.88
MobileNetV2	HAQ	2.27/32.00	66.75
	AutoQNN	2.30/32.00	69.77
ResNet50	HAQ	2.06/32.00	70.63
	BSQ	2.30/4.00	75.16
	AutoQNN	2.27/32.00	74.76
InceptionV3	HAWQ	2.60/4.00	75.52
	HAWQ-V2	2.61/4.00	75.98
	BSQ	2.48/6.00	75.90
	BSQ	2.81/6.00	76.60
	AutoQNN	2.69/4.00	76.57

Note: W/A indicates the average bitwidth for weight and activation, respectively. The best results are underlined.

average bitwidths of 2.16/2.49 for weight and activation quantization. AutoQNN outperforms AutoQB-QBN[13] and AutoQB-BBN[13] by 2.25% and 6.47% improvements in model accuracy, respectively, under even lower average bitwidths of 2.94/3.07. When we employ AutoQNN to quantize models, including MobileNets, MobileNetV2, and ResNet50, into the same size as that in HAQ[10], we can gain accuracy improvements by 12.74%, 3.02%, and 4.13%, respectively. Besides, there are 1.05% and 0.59% accuracy improvements on InceptionV3 by AutoQNN, compared with HAWQ[8] and HAWQ-V2[9], respectively. On InceptionV3, AutoQNN with 2.69/4 achieves higher accuracy (76.57% vs 75.90%) using even lower activation bitwidth compared with BSQ with 2.48/6, and also uses lower weight and activation bitwidth to achieve similar accuracy (76.57% vs 76.60%) to BSQ with 2.81/6. The top 1 accuracy of BSQ[36] with 2.3/4 on ResNet50 over ImageNet achieves 75.16%, which slightly exceeds the accuracy result 74.76% of AutoQNN with 2.27/32. The reason is that the referenced pre-trained ResNet50 in PyTorch has relatively higher accuracy than that in Keras. The ResNet50 from the Keras community has an accuracy of 74.90%, but that in the PyTorch community is up to 76.15%. According to the referenced accuracy, the accuracy drop of AutoQNN is only 0.14%, while that of BSQ is up to 0.99%. These comprehensive comparison results highlight that AutoQNN can obtain better mixed-precision policies for various mainstream architectures and surpass the state-of-the-art methods.

5.2 Evaluation on LSTM

We conduct two long-short-term-memory (LSTM)[49] experiments in this subsection to verify the effectiveness of AutoQNN on natural language processing (NLP) applications. The baseline methods include EffectiveQ[50], QNNs[51], LP-RNNs[52], BalancedQ[53], and HitNet[54]. We employ AutoQNN to search different quantization policies for four weight matrices and one output state in LSTMs, as did in HitNet. The experimental results show that AutoQNN can find the appropriate quantizing strategies for different weights and activation in LSTMs.

5.2.1 *Experiments on Text Classification*

We first evaluate AutoQNN on the text classification task over the subset of the THCUNews dataset[⑥], which contains 50k pieces of news for 10 categories. We use the model with one word embedding layer, one LSTM layer (with 512 hidden units), and two fully-connected layers (with 256 and 128 hidden units, respectively) as an evaluated model, and employ accuracy to measure model performance. We quantize the weights and activation of the embedding, LSTM, and fully-connected layers in the evaluated model.

The results are shown in Table 5. AutoQNN finds that ZoomQ and PotQ are the best quantizing schemes for weight and activation quantization, respectively. The accuracy of the quantized model using 2-bit weights and 2-bit activation achieves 94.53%, slightly lower than the full-precision result by 0.99%. When increasing the quantizing bitwidth to 3, the model accuracy achieves 95.46%, which is very close to the accuracy of the full-precision model. This experiment shows that AutoQNN can be applied to text classification tasks to preserve the accuracy of quantized recurrent neural networks (RNNs).

5.2.2 *Experiments on Penn TreeBank*

We further evaluate AutoQNN on the sequence prediction task over the penn treebank (PTB)

[⑥]https://github.com/thunlp/THUCTC, Mar. 2024.

Table 5. Accuracy (%) of LSTM Model on THCUNews

Method	Weight Quantization	Activation Quantization	Average Bits of Weights	Average Bits of Activation	Accuracy
Full-precision	–	–	32.00	32.00	95.52
AutoQNN	ZoomQ	PotQ	2.00	2.01	94.53
			2.00	3.01	95.03
			3.01	3.02	95.46

dataset[55], which contains 10k unique words. We use an open model implementation for evaluation, and its source codes can be found here[⑦]. For a fair comparison, we modify the open model and use one embedding layer with 300 outputs and one LSTM layer with 300 hidden units, as did in [50]. We train the full-precision model for 100 epochs with the Adam optimizer[56] and a learning rate of 1.0×10^{-3}. Then, we adopt AutoQNN to search for the best quantizing strategy for the full-precision model and quantize the weights and activation of its embedding and LSTM layers. Specifically, we employ the coarse-grained QSS to seek a shared quantizing scheme and use QPL to learn a mixed-precision policy. Model performance is measured in the perplexity per word (PPW) metric, as used in [50–54].

The comparison results are shown in Table 6. AutoQNN employs the 2.48-bit ResQ for weight quantization and the 2.28-bit ClipQ for activation quantization, and achieves a competitive PPW result, outperforming previous methods, including EffectiveQ[50], QNNs[51], LP-RNNs[52], and BalancedQ[53]. The result of AutoQNN is slightly higher than that of HitNet[54], because we use an inaccurate full-precision model with a compressed embedding layer. Another reason is that the candidate schemes in AutoQNN are designed for CNNs and are not adapted to RNNs. For example, the activation function applied in RNNs is tanh, and its output range is $(-1, 1)$. In contrast, the activation range of CNN is usually $[0, +\infty)$. Therefore, applying truncation for RNN's outputs in quantization is harmful to model performance. Nevertheless, AutoQNN can still find the best quantizing strategies for RNNs to preserve PPW.

6 Ablation Study

In this section, we perform two ablation studies including QSS validation and QPL validation.

6.1 QSS Validation

We evaluate the attainable accuracy of quantized models to verify that QSS can find desirable quantizing schemes for DNNs[⑧]. To do this, we take all candidate schemes as baselines and conduct two experiments: layer-wised search and model-wised search.

6.1.1 Settings

The widely verified Cifar10[57] and VGG-like[58] are used in this experiment. The data augmentation of Cifar10 in [59] is employed. To highlight the discrimination of results, we fix the quantizing bitwidth as 3 for the multi-bit schemes such as ClipQ and PotQ. We train VGG-like on Cifar10 for 300 epochs with an

Table 6. PPW Results of Different Quantization Methods on PTB

Method	Weight Quantization	Activation Quantization	Average Bits of Weights	Average Bits of Activation	PPW
EffectiveQ	–	–	2.00	2.00	152.0
	–	–	2.00	3.00	142.0
	–	–	3.00	3.00	120.0
QNNs	–	–	2.00	3.00	220.0
LP-RNNs	–	–	2.00	2.00	152.2
BalancedQ	–	–	2.00	2.00	126.0
	–	–	2.00	3.00	123.0
HitNet	TTQ	BTQ	2.00	2.00	110.3
AutoQNN	ResQ	ClipQ	2.48	2.28	116.7

[⑦] https://github.com/adventuresinML/adventures-in-ml-code, Mar. 2024.

[⑧] More experimental results can be found in our supplementary materials. https://github.com/JCST-supplementary/Paper-Supplementary/blob/main/supplementary-materials.pdf, Mar. 2024.

initial learning rate of 0.1, and decay the learning rate to 0.01 and 0.001 at 250 epochs and 290 epochs, respectively.

6.1.2 Layer-Wised Search

We present layer-wised search processes in Fig.6, which draws the changes of sampling probabilities of different candidate schemes during 150 training epochs. All the schemes have the same sampling probabilities at the beginning of a training phase. The sum of the probabilities at each epoch equals 1. For the weight quantization of the seven layers of VGG-like, the search processes of different layers tend to be similar, i.e., the probability of P-3 gradually grows with the training epoch increasing, while that of other schemes decreases. The results imply that P-3 contributes less training loss during model training than the other schemes. Therefore, QSS finds that P-3 is the desirable quantizing scheme among all candidate schemes. Similarly, for the activation quantization of VGG-like, QSS has found that R-3 owns the highest sampling probability at the first layer because R-3 can reserve the features in the first layer well. Besides, C-3 achieves the highest sampling probabilities at the rest six layers since C-3 can eliminate outliers and contribute robust training results. Finally, QSS finds the desirable quantizing schemes of {P-3, P-3, P-3, P-3, P-3, P-3, P-3} and {R-3, C-3, C-3, C-3, C-3, C-3, C-3} for the weight and activation quantization of VGG-like, respectively.

Next, we generate a quantized architecture with the found schemes above and fine tune it for another 150 epochs to converge, as described in Algorithm 2. The accuracy comparison is shown in Fig.7. QSS is denoted as QSS-F. B-1/3 denotes employing 1-bit binary for weight quantization and 3-bit ClipQ for activation quantization, respectively. The accuracy of QSS-F achieves 93.07%, which constantly outperforms that of candidate schemes by 2.70% on average, and only incurs 0.42% accuracy degradation compared with the full-precision result (denoted as FP32). The result verifies that QSS is able to seek desirable quantizing schemes for different layers to gain high accuracy.

6.1.3 Model-Wised Search

We employ the coarse-grained QSS to find shared quantizing schemes for all layers of VGG-like. Specifically, respecting that the distributions of weights and activation are usually diverse, we seek two shared schemes for the weight and activation quantization in this experiment, respectively. The search process for weight quantization is shown in Fig.8(a), and that for activation quantization is shown in Fig.8(b). Similar to the layer-wised results above, QSS finds that C-3 and P-3 are the desirable quantizing schemes for weight quantization, and C-3 is the desirable one for activation quantization. Therefore, we take C-3/C-3 and P-3/C-3 as two desirable quantizing strategies. C-3/C-3 employs C-3 for both weight and activation quantization. P-3/C-3 adopts P-3 for weight quanti-

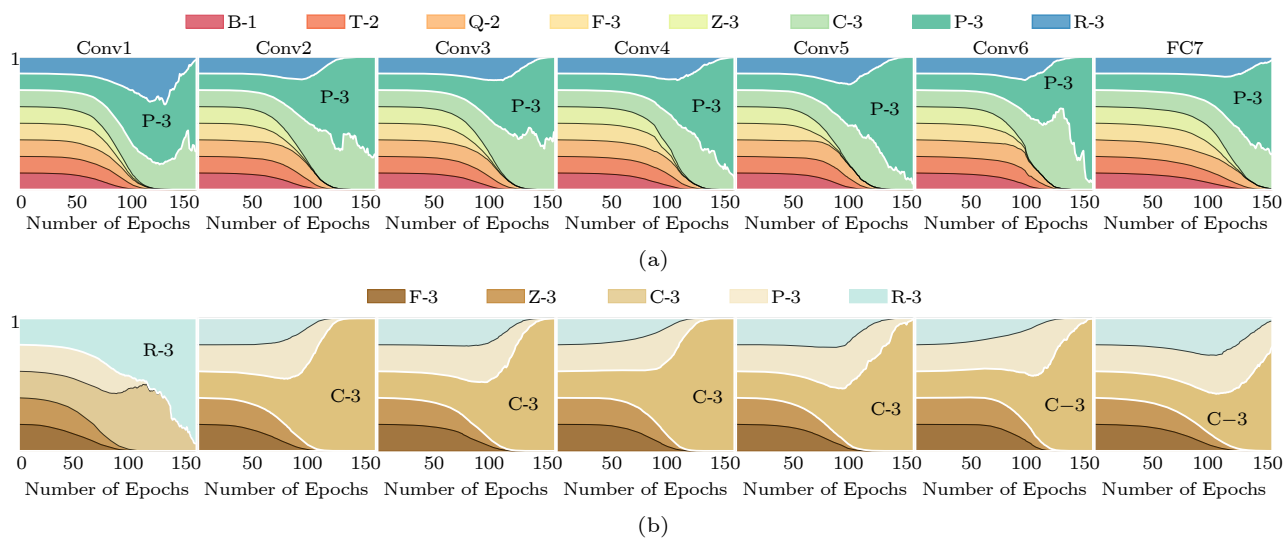


Fig.6. Changes of sampling probabilities of candidate schemes in the training phase of VGG-like. The x-axis is the number of training epochs (total number of 150 epochs) and the y-axis is the sampling probabilities of schemes (the sum of the probabilities is 1). (a) Quantizing scheme search processes for the weights of different layers. (b) Quantizing scheme search processes for the activation of different layers.

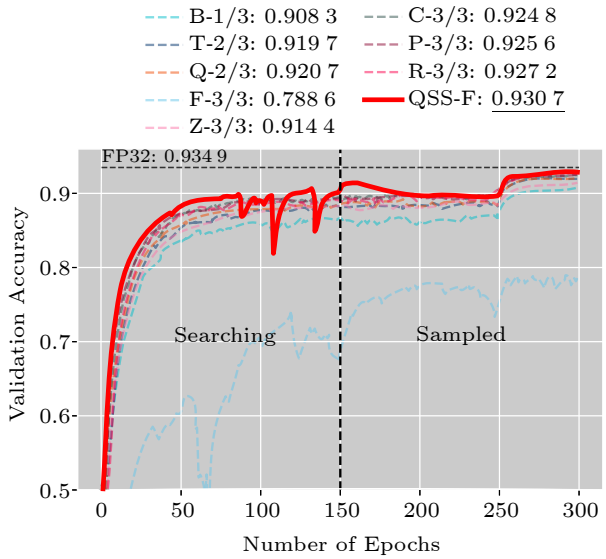


Fig.7. Validation accuracy curves in model training. QSS-F denotes the result of the model quantized with the layer-wised search scheme.

zation and uses C-3 for activation quantization.

Since the shared quantizing schemes may not meet the requirements of some layers inevitably, model-wised search sacrifices some accuracy to obtain the unified quantizing schemes. However, QSS still presents competitive results compared with baselines. As shown in Table 7, the accuracy of C-3/C-3 achieves 92.48%, which is only slightly lower than the results of P-3/3 and R-3/3. The accuracy of P-3/C-3 achieves 92.90%, which outperforms that of P-3/3 and R-3/3 by 0.34% and 0.18%, respectively. These results demonstrate that QSS is able to find expected quantizing schemes from a candidate set efficiently, thus improving the accuracy of quantized models without requiring more bits and avoiding heavy manual workloads.

6.2 QPL Validation

In this subsection, we verify that QPL can find reasonable mixed-precision policies for VGG-like and reduce the performance degradation of the quantized model within a limited model size and memory footprint.

We take ClipQ as the quantizing scheme for verifying QPL, since clip quantization is the most widely used scheme. We still utilize Cifar10 and VGG-like for evaluations. In addition, we set a target expected bit of 3 for precision loss in this experiment to highlight comparison results.

We train 150 epochs to learn optimal mixed-precision policies for VGG-like. Fig.9(a) presents the quantizing bit changes of weight quantization with the number of epochs. We finally obtain the mixed-precision policy of {7, 8, 6, 6, 7, 5, 2} for the weight quantization of different layers, and the average bitwidth of this policy is 2.82. There are $4\,096 \times 1\,024$ parameters in the full-connected layer FC7, which occupy over 90% of the VGG-like parameters and have a lot

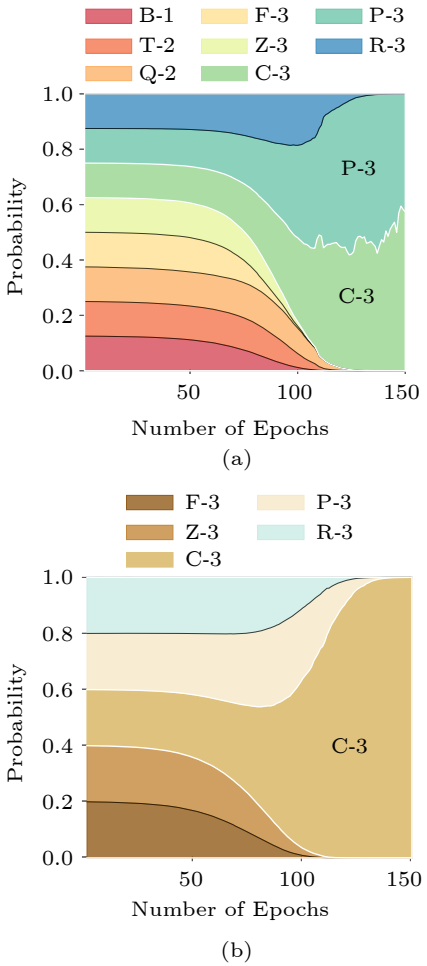


Fig.8. Changes of shared sampling probabilities in model-wised search. (a) Quantizing scheme search for the weights in VGG-like. (b) Quantizing scheme search for the activation in VGG-like.

Table 7. Accuracy Comparison in Model-Wised Search (C-3/C-3 and P-3/C-3)

Name	Method	Accuracy (%)
Normal quantization	B-1/3	90.83
	T-2/3	91.97
	Q-2/3	92.07
	F-3/3	78.86
	Z-3/3	91.44
	P-3/3	92.56
	R-3/3	92.72
Model-wised search	C-3/C-3	92.48
	P-3/C-3	<u>92.90</u>

Note: The best results are underlined.

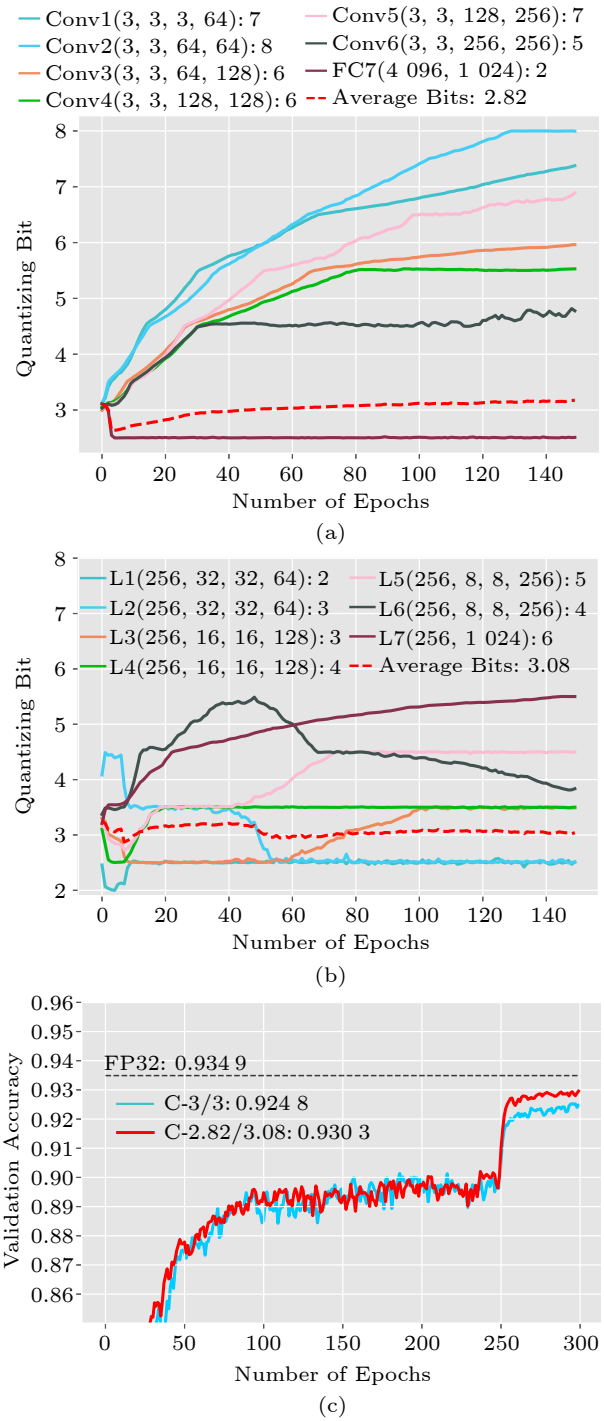


Fig.9. Learning process of QPL and the final validation accuracy comparison of quantized VGG-like. Conv and FC denote the convolutional layers and full-connected layers of the model, respectively, and the content in brackets after them indicate the shape of the layer weight. Li indicates the i-th layer of the model, and the content in brackets after them indicate the shape of the layer activation. (a) Bit changes of weight quantization. (b) Bit changes of activation quantization. (c) Accuracy results.

of redundancy. Therefore, QPL obtains the bitwidth of 2 for FC7 to balance the classification and precision losses. The convolutional layers use a small number of parameters to extract features and have less redundancy. Consequently, QPL learns high bitwidths to reduce the classification loss, such as the 7 and 8 bits for Conv1 and Conv2, respectively, as shown in Fig.9(a). The mixed-precision learning process of activation quantization is shown in Fig.9(b). QPL learns the mixed-precision policy of {2, 3, 3, 4, 5, 4, 6} and the average bitwidth of this policy is 3.08. The accuracy comparison is shown in Fig.9(c). Compared with ClipQ (denoted as C-3/3), the accuracy of the learned mixed-precision policy (denoted as C-2.82/3.08) achieves 93.03%, which outperforms C-3/3 by 0.55%.

The results above demonstrate that QPL is able to learn relatively optimal mixed-precision policies to balance the classification and precision losses of DNNs. QPL employs the classification loss to reduce the model redundancy and proposes the precision loss $\overline{L}$ in (6) to constrain the model size and memory footprint, thus improving the accuracy and efficiency of quantized DNNs.

7 Conclusions

In this paper, we proposed AutoQNN, an end-to-end framework aiming at automatically quantizing neural networks. Differing from manual-participated heuristic explorations with heavy workloads of domain experts, AutoQNN could efficiently explore the search space of automatic quantization and provide appropriate quantizing strategies for arbitrary DNN architectures. It automatically sought desirable quantizing schemes and learned relatively optimal mixed-precision policies for efficiently compressing DNNs. Compared with full-precision models, the quantized models using AutoQNN achieved competitive classification accuracy with a much smaller model size and memory footprint. Compared with state-of-the-art competitors: DSQ[47], QIL[25], BCGD[48], and TRQ[32], the comprehensive evaluations on AlexNet and ResNet18 demonstrated that AutoQNN obtained accuracy improvements by up to 1.65% and 1.74%, respectively.

Conflict of Interest The authors declare that they have no conflict of interest.

References

[1] Williams S, Waterman A, Patterson D. Roofline: An in-

sightful visual performance model for multicore architectures. *Communications of the ACM*, 2009, 52(4): 65–76. DOI: 10.1145/1498765.1498785.

[2] Hubara I, Courbariaux M, Soudry D, El-Yaniv R, Bengio Y. Binarized neural networks. In *Proc. the 30th Int. Conf. Neural Information Processing Systems*, Dec. 2016, pp.4114–4122. DOI: 10.5555/3157382.3157557.

[3] Li F F, Liu B, Wang X X, Zhang B, Yan J C. Ternary weight networks. arXiv: 1605.04711, 2022. https://doi.org/10.48550/arXiv.1605.04711, Mar. 2024.

[4] Zhou S C, Wu Y X, Ni Z K, Zhou X Y, Wen H, Zou Y H. DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients. arXiv: 1606.06160, 2018. https://doi.org/10.48550/arXiv.1606.06160, Mar. 2024.

[5] Lin Z H, Courbariaux M, Memisevic R, Bengio Y. Neural networks with few multiplications. arXiv: 1510.03009, 2016. https://doi.org/10.48550/arXiv.1510.03009, Mar. 2024.

[6] Gong C, Chen Y, Lu Y, Li T, Hao C, Chen D M. VecQ: Minimal loss DNN model compression with vectorized weight quantization. *IEEE Trans. Computers*, 2020, 70(5): 696–710. DOI: 10.1109/TC.2020.2995593.

[7] Gong C, Li T, Lu Y, Hao C, Zhang X F, Chen D M, Chen Y. μL2Q: An ultra-low loss quantization method for DNN compression. In *Proc. the 2019 International Joint Conference on Neural Networks*, Jul. 2019. DOI: 10.1109/ijcnn.2019.8851699.

[8] Dong Z, Yao Z W, Gholami A, Mahoney M, Keutzer K. HAWQ: Hessian aware quantization of neural networks with mixed-precision. In *Proc. the 2019 IEEE/CVF International Conference on Computer Vision*, Oct. 27–Nov. 2, 2019, pp.293–302. DOI: 10.1109/iccv.2019.00038.

[9] Dong Z, Yao Z W, Arfeen D, Gholami A, Mahoney M W, Keutzer K. HAWQ-V2: Hessian aware trace-weighted quantization of neural networks. In *Proc. the 34th Conference on Neural Information Processing Systems*, Dec. 2020.

[10] Wang K, Liu Z J, Lin Y J, Lin J, Han S. HAQ: Hardware-aware automated quantization with mixed precision. In *Proc. the 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Jun. 2019, pp.8604–8612. DOI: 10.1109/cvpr.2019.00881.

[11] Lin D D, Talathi S S, Annapureddy V S. Fixed point quantization of deep convolutional networks. In *Proc. the 33rd International Conference on International Conference on Machine Learning*, Jun. 2016, pp.2849–2858. DOI: 10.5555/3045390.3045690.

[12] Wu B C, Wang Y H, Zhang P Z, Tian Y D, Vajda P, Keutzer K. Mixed precision quantization of convnets via differentiable neural architecture search. arXiv: 1812.00090, 2018. https://doi.org/10.48550/arXiv.1812.00090, Mar. 2024.

[13] Lou Q, Liu L, Kim M, Jiang L. AutoQB: AutoML for network quantization and binarization on mobile devices. arXiv: 1902.05690v1, 2020. https://doi.org/10.48550/arXiv.1902.05690, Mar. 2024.

[14] Zhu C Z, Han S, Mao H Z, Dally W J. Trained ternary quantization. arXiv: 1612.01064, 2017. https://doi.org/10.48550/arXiv.1612.01064, Mar. 2024.

[15] Leng C, Dou Z S, Li H, Zhu S H, Jin R. Extremely low bit neural network: Squeeze the last bit out with ADMM. In *Proc. the 32nd AAAI Conference on Artificial Intelligence*, Feb. 2018, pp.3466–3473. DOI: 10.1609/aaai.v32i1.11713.

[16] Phan H, Liu Z C, Huynh D, Savvides M, Cheng K T, Shen Z Q. Binarizing mobilenet via evolution-based searching. In *Proc. the 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Jun. 2020, pp.13417–13426. DOI: 10.1109/CVPR42600.2020.01343.

[17] Courbariaux M, Bengio Y, David J P. BinaryConnect: Training deep neural networks with binary weights during propagations. arXiv: 1511.00363, 2016. https://doi.org/10.48550/arXiv.1511.00363, Mar. 2024.

[18] Rastegari M, Ordonez V, Redmon J, Farhadi A. XNOR-Net: ImageNet classification using binary convolutional neural networks. In *Proc. the 14th European Conference on Computer Vision*, Oct. 2016, pp.525–542. DOI: 10.1007/978-3-319-46493-0_32.

[19] Alemdar H, Leroy V, Prost-Boucle A, Pétrot F. Ternary neural networks for resource-efficient AI applications. In *Proc. the 2017 International Joint Conference on Neural Networks*, May 2017, pp.2547–2554. DOI: 10.1109/ijcnn.2017.7966166.

[20] Jin C R, Sun H M, Kimura S. Sparse ternary connect: Convolutional neural networks using ternarized weights with enhanced sparsity. In *Proc. the 23rd Asia and South Pacific Design Automation Conference*, Jan. 2018, pp.190–195. DOI: 10.1109/aspdac.2018.8297304.

[21] Gysel P. Ristretto: Hardware-oriented approximation of convolutional neural networks. arXiv: 1605.06402, 2016. https://doi.org/10.48550/arXiv.1605.06402, Mar. 2024.

[22] Chen Y, Zhang K, Gong C, Hao C, Zhang X F, Li T, Chen D M. T-DLA: An open-source deep learning accelerator for ternarized DNN models on embedded FPGA. In *Proc. the 2019 IEEE Computer Society Annual Symposium on VLSI*, Jul. 2019, pp.13–18. DOI: 10.1109/isvlsi.2019.00012.

[23] Li Y H, Dong X, Wang W. Additive powers-of-two quantization: An efficient non-uniform discretization for neural networks. arXiv: 1909.13144, 2020. https://doi.org/10.48550/arXiv.1909.13144, Mar. 2024.

[24] Wang P S, Hu Q H, Zhang Y F, Zhang C J, Liu Y, Cheng J. Two-step quantization for low-bit neural networks. In *Proc. the 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Jun. 2018, pp.4376–4384. DOI: 10.1109/cvpr.2018.00460.

[25] Jung S, Son C, Lee S, Son J, Han J J, Kwak Y, Hwang S J, Choi C. Learning to quantize deep networks by optimizing quantization intervals with task loss. In *Proc. the 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Jun. 2019, pp.4345–4354. DOI: 10.1109/CVPR.2019.00448.

[26] Choi J, Wang Z, Venkataramani S, Chuang P I J, Srinivasan V, Gopalakrishnan K. PACT: Parameterized clip-

ping activation for quantized neural networks. arXiv: 1805.06085, 2018. https://doi.org/10.48550/arXiv.1805.06085, Mar. 2024.

[27] Miyashita D, Lee E H, Murmann B. Convolutional neural networks using logarithmic data representation. arXiv: 1603.01025, 2016. https://doi.org/10.48550/arXiv.1603.01025, Mar. 2024.

[28] Zhou A J, Yao A B, Guo Y W, Xu L, Chen Y R. Incremental network quantization: Towards lossless CNNs with low-precision weights. arXiv: 1702.03044, 2017. https://doi.org/10.48550/arXiv.1702.03044, Mar. 2024.

[29] Ghasemzadeh M, Samragh M, Koushanfar F. ReBNet: Residual binarized neural network. In *Proc. the 26th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines*, Apr. 29–May 1, 2018, pp.57–64. DOI: 10.1109/fccm.2018.00018.

[30] Li Z F, Ni B B, Zhang W J, Yang X K, Gao W. Performance guaranteed network acceleration via high-order residual quantization. In *Proc. the 2017 IEEE International Conference on Computer Vision*, Oct. 2017, pp.2603–2611. DOI: 10.1109/iccv.2017.282.

[31] He K M, Zhang X Y, Ren S Q, Sun J. Deep residual learning for image recognition. In *Proc. the 2016 IEEE Conference on Computer Vision and Pattern Recognition*, Jun. 2016, pp.770–778. DOI: 10.1109/cvpr.2016.90.

[32] Li Z F, Ni B B, Yang X K, Zhang W J, Gao W. Residual quantization for low bit-width neural networks. *IEEE Trans. Multimedia*, 2023, 25: 214–227. DOI: 10.1109/TMM.2021.3124095.

[33] Zhang D Q, Yang J L, Ye D Q Z, Hua G. LQ-Nets: Learned quantization for highly accurate and compact deep neural networks. In *Proc. the 15th European Conference on Computer Vision*, Sept. 2018, pp.373–390. DOI: 10.1007/978-3-030-01237-3_23.

[34] Lin X F, Zhao C, Pan W. Towards accurate binary convolutional neural network. In *Proc. the 31st Conference on Neural Information Processing Systems*, Dec. 2017, pp.345–353.

[35] Lou Q, Guo F, Kim M, Liu L T, Jiang L. AutoQ: Automated kernel-wise neural network quantization. In *Proc. the 8th Int. Conf. Learning Representations*, Apr. 2020.

[36] Yang H R, Duan L, Chen Y R, Li H. BSQ: Exploring bit-level sparsity for mixed-precision neural network quantization. In *Proc. the 9th International Conference on Learning Representations*, May 2021.

[37] Maddison C J, Mnih A, Teh Y W. The concrete distribution: A continuous relaxation of discrete random variables. arXiv: 1611.00712, 2017. https://doi.org/10.48550/arXiv.1611.00712, Mar. 2024.

[38] Jacob B, Kligys S, Chen B, Zhu M L, Tang M, Howard A, Adam H, Kalenichenko D. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proc. the 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Jun. 2018, pp.2704–2713. DOI: 10.1109/cvpr.2018.00286.

[39] Nair V, Hinton G E. Rectified linear units improve restricted Boltzmann machines. In *Proc. the 27th Int. Conf. Machine Learning*, Jun. 2010, pp.807–814.

[40] Deng J, Dong W, Socher R, Li L J, Li K, Fei-Fei L. ImageNet: A large-scale hierarchical image database. In *Proc. the 2009 IEEE Conference on Computer Vision and Pattern Recognition*, Jun. 2009, pp.248–255. DOI: 10.1109/cvpr.2009.5206848.

[41] Krizhevsky A, Sutskever I, Hinton G E. ImageNet classification with deep convolutional neural networks. In *Proc. the 25th Int. Conf. Neural Information Processing Systems*, Dec. 2012, pp.1097–1105. DOI: 10.5555/2999134.2999257.

[42] Howard A G, Zhu M L, Chen B, Kalenichenko D, Wang W J, Weyand T, Andreetto M, Adam H. MobileNets: Efficient convolutional neural networks for mobile vision applications. arXiv: 1704.04861, 2017. https://doi.org/10.48550/arXiv.1704.04861, Mar. 2024.

[43] Sandler M, Howard A, Zhu M L, Zhmoginov A, Chen L C. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proc. the 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Jun. 2018, pp.4510–4520. DOI: 10.1109/cvpr.2018.00474.

[44] Szegedy C, Vanhoucke V, Ioffe S, Shlens J, Wojna Z. Rethinking the inception architecture for computer vision. In *Proc. the 2016 IEEE Conference on Computer Vision and Pattern Recognition*, Jun. 2016, pp.2818–2826. DOI: 10.1109/CVPR.2016.308.

[45] Simon M, Rodner E, Denzler J. ImageNet pre-trained models with batch normalization. arXiv: 1612.01452, 2016. https://doi.org/10.48550/arXiv.1612.01452, Mar. 2024.

[46] Gross S, Wilber M. Training and investigating residual nets. *Facebook AI Research*, 2016. https://torch.ch/blog/2016/02/04/resnets.html, Mar. 2024.

[47] Gong R H, Liu X L, Jiang S H, Li T X, Hu P, Lin J Z, Yu F W, Yan J J. Differentiable soft quantization: Bridging full-precision and low-bit neural networks. In *Proc. the 2019 IEEE/CVF International Conference on Computer Vision*, Oct. 27–Nov. 2, 2019, pp.4851–4860. DOI: 10.1109/iccv.2019.00495.

[48] Yin P H, Zhang S, Lyu J C, Osher S, Qi Y Y, Xin J. Blended coarse gradient descent for full quantization of deep neural networks. arXiv: 1808.05240, 2019. https://doi.org/10.48550/arXiv.1808.05240, Mar. 2024.

[49] Hochreiter S, Schmidhuber J. Long short-term memory. *Neural Computation*, 1997, 9(8): 1735–1780. DOI: 10.1162/neco.1997.9.8.1735.

[50] He Q Y, Wen H, Zhou S C, Wu Y X, Yao C, Zhou X Y, Zou Y H. Effective quantization methods for recurrent neural networks. arXiv: 1611.10176, 2016. https://doi.org/10.48550/arXiv.1611.10176, Mar. 2024.

[51] Hubara I, Courbariaux M, Soudry D, El-Yaniv R, Bengio Y. Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research*, 2017, 18(1): 6869–6898.

[52] Kapur S, Mishra A, Marr D. Low precision RNNs: Quantizing RNNs without losing accuracy. arXiv: 1710.07706,

2017. https://doi.org/10.48550/arXiv.1710.07706, Mar. 2024.

[53] Zhou S C, Wang Y Z, Wen H, He Q Y, Zou Y H. Balanced quantization: An effective and efficient approach to quantized neural networks. *Journal of Computer Science and Technology*, 2017, 32(4): 667–682. DOI: 10.1007/s11390-017-1750-y.

[54] Wang P Q, Xie X F, Deng L, Li G Q, Wang D S, Xie Y. HitNet: Hybrid ternary recurrent neural network. In *Proc. the 32nd Conference on Neural Information Processing Systems*, Dec. 2018, pp.602–612.

[55] Taylor A, Marcus M, Santorini B. The Penn Treebank: An overview. In *Treebanks*, Abeillé A (ed.), Springer, 2003, pp.5–22. DOI: 10.1007/978-94-010-0201-1_1.

[56] Kingma D P, Ba J. Adam: A method for stochastic optimization. In *Proc. the 3rd International Conference on Learning Representations*, May 2015.

[57] Krizhevsky A, Hinton G. Learning multiple layers of features from tiny images. Technical Report TR-2009, University of Toronto, 2009. https://learning2hash.github.io/publications/cifar2009learning/, Mar. 2024.

[58] Simonyan K, Zisserman A. Very deep convolutional networks for large-scale image recognition. In *Proc. the 3rd Int. Conf. Learning Representations*, May 2015.

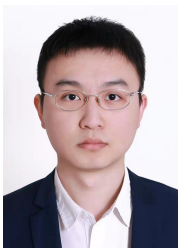
[59] Lee C Y, Xie S N, Gallagher P W, Zhang Z Y, Tu Z W. Deeply-supervised nets. In *Proc. the 18th Int. Conf. Artificial Intelligence and Statistics*, May 2015, pp.562–570.

Cheng Gong received his B.S. and Ph.D. degrees in computer science and technology from Nankai University, Tianjin, in 2016 and 2022, respectively. He is a postdoctoral fellow at the College of Software, Nankai University, Tianjin. His main research interests include neural network compression, heterogeneous computing, and machine learning.

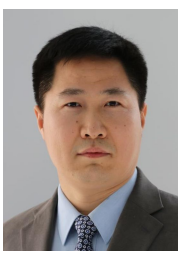
Ye Lu received his B.S. and Ph.D. degrees from Nankai University, Tianjin, in 2010 and 2015, respectively. He is an associate professor at the College of Computer Science of the same university. His main research interests include DNN FPGA accelerator, blockchian virtual machine, embedded system, and Internet of Things.

Su-Rong Dai received her B.S. degree in computer science and technology from Nankai University, Tianjin, in 2020. She is currently working toward her Ph.D. degree in the same university. Her main research interests include computer architecture, compiler design, and blockchain virtual machine.

Qian Deng received her B.S. degree in computer science and technology from Nankai University, Tianjin, in 2020. She is working toward her M.S. degree in the same university. Her main research interests include computer vision and artificial intelligence.



Cheng-Kun Du received his B.S. degree from Nankai University, Tianjin, in 2020. He is currently working toward his M.S. degree in the same university. His main research interests include heterogeneous computing and machine learning.



Tao Li received his Ph.D. degree in computer science from Nankai University, Tianjin, in 2007. He a professor of the College of Computer Science of the same university. His main research interests include heterogeneous computing, machine learning, and blockchain system.