RCT: Resource Constrained Training for Edge AI

Tian Huang, Tao Luo, Ming Yan, Joey Tianyi Zhou, Senior Member, IEEE, and Rick Goh

Abstract—Efficient neural network training is essential for in situ training of edge artificial intelligence (AI) and carbon footprint reduction in general. Train neural network on the edge is challenging because there is a large gap between limited resources on edge and the resource requirement of current training methods. Existing training methods are based on the assumption that the underlying computing infrastructure has sufficient memory and energy supplies. These methods involve two copies of the model parameters, which is usually beyond the capacity of on-chip memory in processors. The data movement between off-chip and on-chip memory incurs large amounts of energy. We propose resource constrained training (RCT) to realize resource-efficient training for edge devices and servers. RCT only keeps a quantized model throughout the training so that the memory requirement for model parameters in training is reduced. It adjusts per-layer bitwidth dynamically to save energy when a model can learn effectively with lower precision. We carry out experiments with representative models and tasks in image classification, natural language processing, and crowd counting applications. Experiments show that on average, 8–15-bit weight update is sufficient for achieving SOTA performance in these applications. RCT saves 63.5%–80% memory for model parameters and saves more energy for communications. Through experiments, we observe that the common practice on the first/last layer in model compression does not apply to efficient training. Also, interestingly, the more challenging a dataset is, the lower bitwidth is required for efficient training.

Index Terms—Efficient training, memory efficient, neural network, quantization, weight update.

I. INTRODUCTION

EDGE computing emerges as an attractive alternative to cloud computing, for its advantages in data privacy, response latency, and energy saving for data transmissions [1]. Neural networks have profound effects on human lives. Many research efforts have pushed neural models toward record-breaking predictive performance [2], [3], [4]. There is also a rising requirement in the capability of learning on edge.

Fig. 1. (a) QAT-based methods involve two copies of model parameters. (b) Our method only keeps a quantized model.

RCT: Resource Constrained Training for Edge AI

For example, an edge artificial intelligence (AI) may have to learn continuously to adapt to the evolving environment. Sometimes, data protection regulation also requires the edge AI to consume the data in the field, instead of sending it to remote premises. Traditionally, neural network training happens in data centers or labs. To train a neural network on the edge, we have to face with the limitation of resources on edge devices.

There are extensive research efforts on reducing the size of a model. Quantization-aware training (QAT) [5] is one of the few training methods that can train a model after it is compressed. However, the existing QAT-based methods are based on the assumption that resources such as memory and energy are abundant.

QAT-based methods involve two copies of the model parameters during training, as shown in Fig. 1(a). At the first stage, a master copy of the model parameters in floating-point format travels from memory to the processor for quantization. Then, the quantized model takes part in inference and generates a loss. At the stage of model update, the master copy travels from memory to the processor again such that the gradients can be added to the master copy. The update is usually followed by another write-back to the memory, which costs even more time and energy than a read does. In short, each training iteration involves three more times memory communications to train a quantized model. The training method is not friendly for edge AI from two aspects. First, on-chip memory of edge devices (e.g., cache or buffer in a processor) is usually limited in size such that the master copy has to stay in off-chip memory (e.g., DRAM), which is much slower than on-chip memory. Second, data movement is energy-consuming. Battery-powered edge devices may not have enough energy to support two copies of a model and frequent data movement between off-chip memory and processors.

As shown in Fig. 1(b), if we only keep a quantized model, we save the memory for the master copy. Since the quantized model is usually small enough to stay in on-chip memory
throughout training, we can also save energy for related data movement. This benefits training on edge devices and servers.

The challenge is that with quantized parameters alone, the model does not always learn effectively. A model with low-precision parameters may learn without difficulty at the beginning of the training process. As the training loss and gradients decrease along with the training progress, quantization underflow happens more often in the model update step and slows down or even puts a stall to the training progress. Allocating more bitwidth could mitigate the underflow issue and sustain the overall training efficiency.

In this article, we propose a heuristic method called resource constrained training (RCT), which does not keep floating-point master copy during training. It consumes less memory space and memory operations for model parameters during training. It applies to both training from scratch and fine-tuning a quantized model. We propose three techniques to mitigate the abovementioned challenges. First, we apply variable bitwidth for parameters of different layers and adjust them dynamically to balance memory saving and model accuracy. Second, we introduce MeanOut to drop extreme values in a tensor, to make good use of the dynamic range of quantized numbers. Third, we introduce gradient dithering for bitwidth demanding layers. It adds noise to the gradient to improve the training process.

We evaluate RCT on representative applications: image classification, text classification, question answering, and crowd counting application. Experiments show that given 8–12 bits for weight update, RCT can train a network from scratch with limited accuracy loss on image-related applications. Given 14–15 bits, RCT can fine-tune a pretrained network with limited accuracy loss on natural language processing (NLP)-related applications. Compared with traditional 8-bit QAT, RCT saves 63.5%–80% memory space and more energy for memory communication.

In the experiments, we discovered that the bitwidth for effective training depends on the architecture of a neural network. This is counterintuitive and contradictory to the current practice in model compression, in which the first and last layers have to be in full precision to prevent large accuracy losses. This is because the bitwidth of weight update depends on the magnitude of the gradients and in turn depends on the architecture of the model. It does not depend on its distance to the input and output layer. Furthermore, we discovered that the bitwidth is also related to the challenge in a dataset. The more challenging the dataset is, the lower bitwidth is required for effective training.

Our contribution can be summarized as follows.

1) The master copy is eliminated in QAT to save memory and energy for communication.
2) Three techniques include variable bitwidth, MeanOut, and gradient dithering that are proposed to save memory and energy for communication at the cost of limited accuracy loss.
3) We investigate the distribution of bitwidth in the experiment and draw insightful conclusions for efficient training.

The rest of this article is organized as follows. Section II reviews previous work on memory-constrained training and energy-constrained training for edge devices. Section III describes the proposed method. Section IV presents the experimental results. Section V concludes this article.

II. RELATED WORK

There are two kinds of popular quantization methods, QAT and post-training quantization [5]. QAT emulates inference-time quantization during training, which helps in maintaining accuracy compared to 16- or 32-bit floating-point baselines. The recent progress in QAT has allowed the realization of results comparable to a baseline for as low as 2–4 bits per parameter [6], [7], [8], [9], [10], [11], [12] and show decent performance even for single-bit (binary) parameters [13], [14], [15], [16], [17], [18]. There are QAT-based methods that efficiently exploit various processing units [19], [20], [21], [22], [23], [24], as well as the ones that allocate per-layer bitwidth [25], [26], [27], [28] to improve the performance of a quantized model. Apart from image-related applications, QAT-based methods have also been applied to NLP [13], [23], [24], [27]. QAT-based methods can train a quantized model. However, these QAT-based methods involve two copies of the model parameters, as they are based on the assumption that the machine for training has abundant memory and electricity supply. This is usually not true for edge devices. In contrast, we propose RCT, which only has one copy of the model parameters. RCT saves memory, as well as the energy for related data movement.

The post-training quantization [29], [30], [31], [32], [33], [34] converts a well-trained model into quantized one. By definition, post-training quantization methods convert models but do not train them. It does not meet the requirements of in situ learning, in which quantized models continue to evolve on edge devices.

Imposing sparsity in neural network is an effective way of compression [35], [36], [37], [38], [39]. In general, sparsity comes in the cost of irregular memory access patterns and unbalanced computation workload [40], [41], which raise resource utilization issues on a resource constrained devices. Other techniques, e.g., knowledge distillation [42], low-rank factorization [43], and fast convolution [44], save memory and/or energy significantly for inference. These methods only conduct compression but do not train the model after it is compressed.

This work has a few differences compared to our previous works [45], [46]. In [45], we demonstrate the adaptive precision training (APT) method that trains an image classification model without a master copy. In [46], we demonstrate that APT also generalizes well to NLP applications. In this work, we propose RCT, which also incorporates MeanOut, gradient dithering, and activation quantization. We discuss a compact memory format for efficient memory storage and communication. We include a real-world application, i.e., crowd counting, in the experiment to demonstrate the ability of generalization. We also include an in-depth analysis of the distribution of training bitwidth, which helps us understand...
the relation between a model architecture and the bitwidth required for effective learning.

III. RESOURCE CONSTRAINED TRAINING

In this section, we detail RCT, which involves three major techniques, including variable bitwidth, MeanOut, and gradient dithering.

A. Variable Bitwidth

The master copy of the model parameters provides the necessary precision for all layers during training. Eliminating the master copy in QAT usually leads to poor performance in terms of accuracy [22]. We look into the training dynamics and find that one of the main reasons is arithmetic underflow. With only quantized parameters, some layers cannot learn effectively. This is because the quantized number has a limited dynamic range compared to a floating-point number and more often suffers from overflow and underflow issues in the weight update process [5]. The parameter update in the fp32 format is given by the following equation:

\[ w_{ij} := w_{ij} - lr \cdot g_{ij} \]

where \( w, lr, \) and \( g \) are the weight, learning rate, and gradient of a layer, respectively. With the fp32 format, the numerical error in the weight update calculation becomes nontrivial. When a layer is too small, it would be overwhelmed by the quantization underflow such that the weight remains unchanged. The threshold of the gradient, or minimum resolution, is given by

\[ \epsilon_i = \frac{\max(W_i) - \min(W_i)}{2^k - 1}. \]

\( W_i \) is a weight tensor of the \( i \)-th layer, \( k \) is the precision or the bitwidth of the tensor, and \( \epsilon_i \) regulates weight update, which is described as follows:

\[ w_{ij} := w_{ij} - \left\lfloor \frac{lr \cdot g_{ij}}{\epsilon_i} \right\rfloor \cdot \epsilon_i. \]

\( \left\lfloor \cdot \right\rfloor \) represents a floor operation, which rounds an operand to the nearest integer below its current value. The regulation quantifies \( lr \cdot g_{ij} \) to the discrete states of \( w_{ij} \). If \( lr \cdot g_{ij} \) is larger than \( \epsilon_i \), the weight does change; otherwise, quantization underflow happens and the weight remains unchanged. When the weight update works with a higher bitwidth, say \( k = 32 \), the weight changes most of the time. With a low bitwidth, e.g., \( k = 2 \), \( \epsilon_i \) is mostly larger than the gradient. The weight virtually “freezes.”

We refer to the effectiveness of weight update as “learning capability.” We observe that some layers usually have smaller gradients compared to others and require higher bitwidth to learn effectively. Besides, the gradients also decrease as the training loss decreases, which suggests that more bitwidth is required at the late stage of training.

1) Precision Metric: We aim to find low-precision configurations that save memory and energy for communication and facilitate learning effectively. The key is to quantify the learning capability of a layer. We introduce a metric, \( G_{avg} \), which is given by the following equation:

\[ G_{avg} = \frac{1}{N_i} \sum_{j=0}^{N_i} \frac{g_{ij}}{\epsilon_i} \]

\( G_{avg} \) is a measurement of the relative magnitude of gradients, compared with \( \epsilon \), i.e., the minimum resolution of the quantized weights. \( G_{avg} \gg 1 \) suggests that on average, the gradients are much larger than \( \epsilon \). Thus, the layer learns effectively. \( G_{avg} \to 0 \) suggests that the layer suffers from a serious quantization underflow problem. The weights freeze most of the time. In order to prevent weights from freezing, we have to increase bitwidth \( k \). A larger \( k \) leads to lower \( \epsilon \) and higher \( G_{avg} \), and a problem-free weight updates eventually.

Apart from “weight,” the concept of \( G_{avg} \) also applies to other learnable parameters in neural network training, such as bias. We design the metric to be decoupled with other training hyperparameters, e.g., momentum or hessian. One can combine our training method with other optimization algorithms, such as stochastic gradient descent (SGD) or Adam.

Fig. 2 shows the distribution of \( G_{avg} \) in the training of MobileNet-V2 on CIFAR-10. Fig. 2(a) shows the histogram across layers at the end of training. \( G_{avg} \) of layers differs by more than 30 times. Fig. 2(b) shows \( G_{avg} \) of all layers over time. \( G_{avg} \) changes along the x-axis, which means that the necessary bitwidth for effective training changes over time. The shadow area, which represents the range of standard deviation, is widening. This suggests the requirement of bitwidth of layers diverges over time. Please find more related information in the appendix of the supplementary material.

Observations in Fig. 2 suggest that some layers can learn with lower bitwidth in a certain training period. There is a great potential in the metric \( G_{avg} \) for saving memory and energy for training.

2) Bitwidth Adjustment: The metric \( G_{avg} \) indicates the learning capability of a layer, which is layer-specific and time-varying. We adjust the bitwidth dynamically to make sure that every layer can learn effectively. A pseudocode of the adjustment policy is given in Algorithm 1.
Algorithm 1 Bitwidth Adjustment

| input: | $k_0, \ldots, k_{M-1}$, $Ga_{g0}, \ldots, Ga_{gM-1}$, $T_{min}$, $T_{max}$ |
| output: | $k_0, \ldots, k_{M-1}$ |
| for $k_i$ in $(k_0, \ldots, k_{M-1})$ |
| if $Ga_{g_i} < T_{min}$ and $k_i < 16$ |
| $k_i := k_i + 1$ |
| if $Ga_{g_i} > T_{max}$ and $k_i > 2$ |
| $k_i := k_i - 1$ |
| return $k_0, \ldots, k_{M-1}$ |

$k_i$ represents the bitwidth for the weight update of the $i$th layer. $Ga_{g_i}$ is the metric of the $i$th layer. There are $M$ layers in the model. $T_{min}$ and $T_{max}$ represent the lower and upper limits of the metric, respectively. Algorithm 1 increases the bitwidth of a layer when $Ga_{g_i} < T_{min}$ and decreases the bitwidth when $Ga_{g_i} > T_{max}$. The lower limit ensures that all layers learn effectively, whereas the upper limit is for saving memory and energy on those parameters that are very easy to update.

Through experiment, we find empirical values for the lower limit $T_{min} = 1.0$, which means that the magnitude of the gradient is approximately equal to the minimum resolution $\epsilon$. Empirically, $T_{max} \geq 10$ works fine for most of the situations. We set the upper bound of bitwidth to 16 because we do not observe extra merit for bitwidth beyond 16 in the improvement of accuracy. Please find the detailed discussion in Section IV.

Algorithm 2 describes the workflow of variable bitwidth. The training starts with low-precision weight update on a quantized model. The metrics evaluation requires gradient information, which is available between backpropagation and parameter update. It is followed by MeanOut and gradient dithering, which will be described in Sections III-B and III-C. Before parameter update, RCT performs the bitwidth adjustment. The evaluation and adjustment do not happen in every training iteration. A few times, say 10, in each epoch suffice to help the quantized training catch up with the progress of full-precision training.

As a side product, RCT also works with quantized errors and gradients. Line 11 in Algorithm 2 adds gradients into quantized weights and keeps weights in the quantized format. Full- and low-precision gradients are equivalent here because extra information in full-precision gradients will be lost due to quantization overflow. We can calculate errors and gradients in reduced precision in the first place to save energy and memory.

3) Variable Bitwidth Format: The goal of RCT is to reduce memory space and communication energy. Parameters with variable bitwidth travel and store in a compact format. Fig. 3 shows an example.

As shown in Fig. 3(b), parameters in compact format consist of a number specifying the bitwidth of parameters (with black frames) and the payloads (without black frames). The boundary of each parameter might not be byte-aligned. When a tensor is read into the register file, each parameter is expended or decoded into a byte-aligned format for calculation, which is shown in Fig. 3(a). When the parameters have to be written back to off-chip memory, they are encoded into a compact format, which is shown in Fig. 3(c).

Decoding and encoding of the compact format only involve simple operations such as integer addition and shift arithmetic [48]. The adjustment of bitwidth does not involve any additional calculation. RCT performs the adjustment in the next weight update operation. Therefore, the overhead of variable bitwidth is negligible.

The proposed variable bitwidth is for weight update in training. We use fixed 8-bit weights for inference. Training and inference share the same copy of the quantized model parameters, except that they are using different parts of it. In Fig. 4(a), the dashed boxes indicate the part for weight update in training. The dashed boxes in Fig. 4(b) indicate the part of weight for inference. In comparison, a traditional QAT method uses separate parameters, i.e., floating-point parameters for training and quantized parameters for inference.

When the bitwidth for weight update in training is not greater than 8 bit, the acquisition of weight for inference does not require any additional calculation. When variable bitwidth is beyond 8 bit, a shift or truncation operation can be used to extract the weight for inference. In comparison, a traditional QAT method has to convert the floating-point model into
Fig. 4. Training and inference share the same copy of model parameters. (a) Bitwidth for weight update is indicated by the dashed box. (b) Inference uses 8-bit weight, which is indicated by the dashed boxes.

Fig. 5. Extreme value incurs large quantization error. (a) Histogram of $n_{std}$. (b) Weight in a layer. (c) Quantization w/o MeanOut. (d) Quantization with MeanOut.

Fig. 6. Quantization error versus $n_{std}$ for BERT base uncased on SQuAD1.0.

B. MeanOut

We observe that extreme values in tensors, which could incur large quantization errors. Fig. 5 shows an example of extreme values.

Fig. 5(b) shows the histogram of the weight of layer “encoder.layer.4.output.dense” in BERT. Some elements with extreme values are far away from zero. In order to demonstrate the existence of extreme values in a weight tensor, we define

$$n_{std} = \frac{\max(w) - \min(w)}{\text{std}(w)}.$$  \hspace{1cm} (5)

$w$ is a weight tensor and std stands for standard deviation. For the layer shown in Fig. 5(b), $n_{std}$ is over 200. We find that the existence of extreme values is very common in neural network. The histogram of $n_{std}$ in BERT for different layers is shown in Fig. 5(a). There are a few layers with $n_{std}$ larger than 50. Please refer to the appendix of the supplementary material for examples on other models and datasets. Since linear quantization involves maximum and minimum values of a tensor, extreme values in these layers will increase quantization error considerably.

Fig. 5(c) shows the histogram of the weights after applying 8-bit linear quantization. Please note that we zoom into $[-1.0, 1.0]$ to have a close look at the bins with most of the occurrences. We discover that the representation of quantized weight is relatively sparse and does not cover the original parameters very well. If we eliminate the extreme values before quantization, the quantized weight, as shown in Fig. 5(d), would be covering the original parameters much better.

To understand the relation between $n_{std}$ and quantization error, we quantize the whole model with 8-bit linear quantization. Fig. 6 shows that $n_{std}$ is roughly proportional to quantization error. A weight tensor with $n_{std}$ over 200 has a quantization error of 0.351. If we were able to reduce $n_{std}$ down to 20, the quantization error would be ten times lower.

To reduce the quantization error and facilitate the training of the model with low bitwidth weight update, we propose MeanOut to eliminate extreme values. The MeanOut of a weight tensor $w$ is calculated by

$$w_i = \begin{cases} w_i, & \text{if } \text{abs}(w_i - \text{mean}(w)) < 6 \times \text{std}(w) \\
\text{mean}(w), & \text{else} \end{cases} \hspace{1cm} (6)$$

$w_i$ is the $i$th element of the tensor $w$. The basic idea is that when the value of an element is too far away from the mean, it is set to the mean of the tensor. MeanOut is a technique similar to dropout [49], but with a different purpose. It is a heuristic regularization method that reduces the extremity in parameters. With fewer extreme values, the quantization error would also be reduced. RCT performs MeanOut every few hundred training iterations. The overhead of MeanOut is negligible.

Please note that the value 6 in (6) is an empirical value. This empirical threshold corresponds to $n_{std} = 12$ in training. A threshold lower than $6 \times \text{std}(w)$ affects more parameters and reset parameters more often and leads to poor accuracy. Larger

$1$The calculation of quantization error is given by:

$$((||q||_\infty - ||r||_\infty)/(||r||_\infty)),$$ in which $r$ is a floating-point tensor and $q$ is a quantized tensor.
threshold gradually loses its utility. We have tested MeanOut on image classification, text classification, question answering, and crowd counting. We found that there are no prominent accuracy drops with a threshold of 6–10 times std\((w)\).

Another strategy to eliminate the extreme value in weight is to clip them at 6 × std\((w)\). We have compared Clipping with MeanOut and discovered that the former affects more parameters and happens more often and usually leads to accuracy not as good as MeanOut. Fig. 7 shows the percentage of parameters being affected by MeanOut and Clipping, during the training of ResNet-18 on CIFAR-100.

Fig. 7 suggests that clipping leads to more extreme values in model parameters. With clipping, the extreme values are clamped and stay at the threshold or close to the threshold. In comparison, MeanOut sets the extreme value to mean, making it less likely to go beyond the threshold within the next few hundred training iterations. We also check the \(n_{std}\) metric of the parameters in the models trained by both methods, which is shown in Table I.

Table I suggests that MeanOut leads to lower \(n_{std}\) compared with Clipping. As we demonstrated in Fig. 6, larger \(n_{std}\) is associated with higher quantization error. Therefore, MeanOut is more friendly to QAT tasks.

### C. Gradient Dithering

We observe that some particular layers of an architecture tend to have smaller \(G_{avg}\) and this feature is independent of the dataset. This is not an issue if you use fp32 parameters. However, with quantized parameters only, these layers cannot learn effectively. Fig. 8 shows the histogram of the weight and gradient of transpose\(_{4}\_1\_a\) layer in the LSC-CNN network at the early stage of training. Given 8-bit quantization, the averaged magnitude of the gradient is more than 200 times smaller than \(\epsilon\), i.e., \(G_{avg} < 0.005\). As the training advances, the gradient will be even smaller. Even if we increase the bitwidth to 16, there would still be quantization underflow for most of the time.

This observation suggests that some layers in an architecture tend to freeze with quantized parameters only. When the gradient is too small, noise in the gradient cannot help the model escape local minima. To mitigate this issue, we add noise to the gradient, namely, gradient dithering. The noise is specified by the following equation:

\[
g_i := g_i + \left(Be(0.5) \ast 2 - 1\right) \ast Be(max(1 - G_{avg}, 0)) \ast \epsilon.
\]  

\(Be(\rho)\) is the Bernoulli random variable with probability \(\rho\) of being one. \((Be(0.5) \ast 2 - 1)\) means that the noise has half the chance being positive or negative. \(Be(max(1 - G_{avg}, 0))\) means that the amount of noise is proportional to \(1 - G_{avg}\). When \(G_{avg} > 1\), there is no noise added to the gradients. The magnitude of noise is set to \(\epsilon\), i.e., the minimum resolution of quantized parameters.

With gradient dithering, the training can escape from the local minima and push toward a lower energy state. RCT performs gradient dithering after the evaluation of \(G_{avg}\). The overhead of gradient dithering is negligible.

### IV. EXPERIMENTS

#### A. Experiment Setup

1) Quantization Scheme: We apply a linear quantization scheme [50] to quantize fp32 numbers into integers. Linear quantization maps real numbers \(r\) in a continuous space to integer \(q\) in a discrete space. The formalized equation is given by

\[
r = S(q - Z).
\]

\(S\) represents the scale of a group of values or a tensor and \(Z\) represents the zero point. All values in a tensor share one \(S\) and \(Z\). Tensors have their own \(S\) and \(Z\). For \(k\)-bit quantization, \(q\) has \(2^k\) possible discrete states. In addition, we apply stochastic rounding [51] to mitigate the errors introduced by quantization. Unless specified, we use \(T_{min} = 1.0\) and \(T_{max} = 100\) for bitwidth adjustment policy.
2) Energy Estimation: According to [48], for a 32-bit addition instruction in 45-nm CPU, only 0.14% energy is spent on the actual addition arithmetic. The rest of the energy is for accessing the cache and the register file. This is based on the assumption that all data live in the L1 cache, which is usually a few tens of kilobytes. Accessing L2 cache or off-chip memory will incur more energy consumption. Since most of the neural network is at the scale of megabytes, energy for memory access dominates the training process. We assume that the L1 cache is big enough to hold all data for training and assume that the energy for communication is proportional to the averaged bitwidth of the parameters. The assumptions are conservative because accessing L1 cache is 2–3 orders of magnitude cheaper than off-chip memory, according to Table II [48].

3) Computing Infrastructure: We carry out all experiments on a general-purpose workstation, which is equipped with one Intel Core i9-10900X CPU, 128-GB DDR4 memory, and four NVIDIA Geforce RTX 2080 Ti graphics cards.

B. Image Classification

We evaluate the RCT on three representative datasets, CIFAR-10, CIFAR-100 [52], and ImageNet [2]. Three popular backbones, ResNet-18 [53], MobileNetV2 [54], and ShuffleNetV2 [55], are included in the experiments. We replace all convolution and fully connected layers with our quantized version. Activation is quantized to 8 bit. This choice allows us to produce good energy savings with negligible accuracy loss. The bias in these layers is not quantized as they only make up for a very small fraction of the amount of memory and energy for training. We use random seeds so that all the results presented in this article have good reproducibility. The quantized layers in these models are all initialized with 8 bit at the beginning of training.

For training on CIFAR-10 and CIFAR-100, we follow the settings given next. We use SGD with a mini-batch size of 128. We use a weight decay of 0.0001 and a momentum of 0.9. These models are trained with a mini-batch size of 128 on one GPU. We set the learning rate to 0.1 with a one-epoch warm-up and decay by a factor of 10 every 30 epochs and terminate training at 90 epochs.

Table III shows the performance of RCT on image classification tasks. The metrics for CIFAR-10/CIFAR-100 and ImageNet are accuracy and top1 accuracy, respectively. The difference column shows the accuracy difference between fp32 training and RCT. According to the results, RCT achieves accuracy results that are very close to those of a floating-point training method. The bitwidth column represents the bitwidth for weight update, averaged across layers at the end of training. This column suggests that 10–13 bits would be enough for training image classification tasks.

Fig. 9 shows the distribution of bitwidth in training. For ResNet-18 on ImageNet, the bitwidth does not have significant changes across layers and over time. The last layer requires more bitwidth than other layers do. However, for MobileNet-V2 with CIFAR-100, some layers in the middle require more bitwidth. This result suggests that the first and last layers do not always require more bitwidth than other layers do. Actually, the bitwidth-demanding layers in Fig. 9(a) remain demanding in the experiments of ResNet-18 on CIFAR-10 and CIFAR-100. The observation holds for other image classification, NLP, and crowd counting tasks. (Please find the details in the appendix of the supplementary material.) This suggests that the bitwidth necessary for effective training is closely related to the architecture. In other words, some particular layers in an architecture inherently have small gradients and require more bitwidth for effective learning. This characteristic is consistent across the dataset.

Please note that the bitwidth in Fig. 9 is for weight update in training, which is different from the bitwidth in model compression techniques, which is used for inference. There
is a well-known practice for efficient inference in previous works, in which researchers usually do not quantize the weight of the first and last layer, to avoid a large accuracy drop. This practice is orthogonal to the variable bitwidth in RCT because the latter quantifies the model’s learning capability with a low-precision weight update, while the former tweaks the bitwidth for inference, given a full-precision weight update.

We also investigate the threshold \( T_{\text{min}} \) in the bitwidth adjustment policy, which balances memory requirement and accuracy. In Fig. 10, the \( X \)-axis represents memory requirement for weight update, normalized to its 32-bit counterpart. The \( Y \)-axis stands for accuracy. A dashed line represents the accuracy of its fp32 baseline. Each point in this figure is the result of 200-epoch training. We find that for ResNet-18 on CIFAR-10, RCT consumes 34% of memory for weight update, while the former tweaks the bitwidth for inference, given a full-precision weight update.

We also investigate the threshold \( T_{\text{min}} \) in the bitwidth adjustment policy, which balances memory requirement and accuracy. In Fig. 10, the \( X \)-axis represents memory requirement for weight update, normalized to its 32-bit counterpart. The \( Y \)-axis stands for accuracy. A dashed line represents the accuracy of its fp32 baseline. Each point in this figure is the result of 200-epoch training. We find that for ResNet-18 on CIFAR-10, RCT consumes 34% of memory for weight update with \( T_{\text{min}} = 1.0 \) and with a limited accuracy loss. This is similar to that for WideResNet-28-10 on CIFAR-100, except that \( T_{\text{min}} \) can be smaller before large accuracy loss occurs. This experiment suggests that the weight update in training works effectively when \( G_{\text{avg}} > 1.0 \), i.e., the gradient on average is larger than the minimum resolution \( \epsilon \) in (2). There is more tolerance for low \( G_{\text{avg}} \) in WideResNet-28-10, which has 36.48M parameters, three times the size of ResNet-18.

C. NLP

We evaluate the RCT on a few NLP fine-tuning tasks. Pretrained transformer-based [56] language models, such as BERT [57], have shown great improvement in many NLP tasks. In this experiment, we include the text-classification task, the general language understanding evaluation (GLUE) benchmark [58], and question-answering task, SQuAD1.0 [59]. We employ BERT (bert-base-uncased) as the backbone. Before starting the fine-tuning tasks, we apply MeanOut on the pretrained weight and quantize the weights as 8 bit. Unless specified, \( (T_{\text{min}}, T_{\text{max}}) \) is set to \((0.1, 10.0)\) for all experiments. Activation is quantized to 8 bit. The bias is not quantized as they only make up for a small fraction of the amount of memory and energy for the tasks.

We fine-tune a BERT with GLUE and SQuAD1.0 on one GPU and follow the experimental settings\(^5\) given in the following. For GLUE, we set the batch size as 32 and set the \( \text{max sequence length} \) to 128. We initialize the learning rate as 0.00002 and fine-tune the network with five epochs for MRPC and three epochs for the rest of the dataset. For SQuAD1.0, we set the batch size to 12 and set the \text{max sequence length} to 384. We initialize the learning rate as 0.00003 and fine-tune the network with two epochs. We adopt random seeds in all experiments such that the reproducibility of the results is guaranteed.

The performance of RCT on NLP applications is shown in Table IV. The first seven rows are the tasks of GLUE. Each task has its metric. For most of the tasks, RCT can fine-tune BERT with 14–15 averaged bitwidth with limited accuracy loss compared to the floating-point training method.

We fine-tune a BERT with GLUE and SQuAD1.0 on one GPU and follow the experimental settings\(^5\) given in the following. For GLUE, we set the batch size as 32 and set the \( \text{max sequence length} \) to 128. We initialize the learning rate as 0.00002 and fine-tune the network with five epochs for MRPC and three epochs for the rest of the dataset. For SQuAD1.0, we set the batch size to 12 and set the \text{max sequence length} to 384. We initialize the learning rate as 0.00003 and fine-tune the network with two epochs. We adopt random seeds in all experiments such that the reproducibility of the results is guaranteed.

The performance of RCT on NLP applications is shown in Table IV. The first seven rows are the tasks of GLUE. Each task has its metric. For most of the tasks, RCT can fine-tune BERT with 14–15 averaged bitwidth with limited accuracy loss compared to the floating-point training method.

Fig. 11(a) shows the bitwidth distribution of BERT on

\[^{5}\text{https://github.com/ianmalcolm/transformers/tree/v4.1.1}\]
gradients and in turn small $G_{avg}$ in most of the layers. This is distinguished with image classification tasks, in which all models are trained from scratch. Please find the details of the distribution of $G_{avg}$ in the appendix of the supplementary material.

D. Crowd Counting

The image classification and NLP tasks are generic benchmarks in computer vision and NLP. In this section, we evaluate RCT with a real-life use case, crowd counting, which is an important application in automated public monitoring.

CNN-based neural network [60] has recently achieved good progress in crowd counting. It has been deployed on edge devices, e.g., robots and drones, to carry out crowd analysis [61], [62] and video surveillance [63]. The environment of the tasks changes. For example, the COVID-19 pandemic affects the density of population. The weather affects the outlook of people. These situations require the edge devices to learn in the field to adapt to the changes and balance between efficiency and accuracy. Fig. 12 shows a representative architecture, LSC-CNN. It consists of a VGG-based encoder for extracting features and a multichannel decoder for the localization of the object, which is the head of a human.

We evaluate RCT with a representative crowd counting model, LSC-CNN [64] on ShanghaiTech [65] and UCF-QNRF [66] dataset. The training hyperparameters follow the experimental settings in [64], except that we replace all trainable linear, conv2d, and conv2dtranspose layers with quantized ones. Weight updates are carried out with variable bitwidth format. $(T_{min}, T_{max})$ is set to $(0.1, 10.0)$ for the experiments. All activations are quantized to a fixed 8 bit. The result of training is presented in Table V.

From Table V, we know that on average, RCT requires 8–10 bitwidth in order to train LSC-CNN to the SOTA accuracy. Fig. 13(b) shows the distribution of bitwidth for LSC-CNN on ShanghaiTech Part B. The light color rows in the figure represent the transpose convolution layers. The brightest part at the bottom is for transpose convolution layers in the 1/2 scale channel. They imply that quantization underflow happens more often in this layer and therefore requires higher bitwidth for weight update. There are similar patterns in Fig. 13(a) as well. The observation agrees with those in image classification and NLP.

E. Comparing With Other Training Methods

We compare RCT with other training methods. Table VI shows a list of methods that can also train a quantized model. We use the averaged bitwidth for the model parameters to indicate the requirement for memory space and energy for memory communication.

Most of the methods in this table use fp32 for weight update. For example, XNOR-Net [17] managed to produce binary parameters for inference, but the weight update happens in the fp32 format. FP4 [68] managed to perform weight update effectively with fp16 format. WAGEUBN [22] further proved that weight update works well with the quantized 24-bit format. We quantify the learning capability of a layer by introducing the metric $G_{avg}$, which enables variable bitwidth. MeanOut eliminates the extreme value of parameters such that RCT can train effectively with lower bitwidth. Gradient dithering helps a layer with weak learning ability escape from local minima. Through these techniques, we managed to push the limit of bitwidth down to 8–15 bits.

Most of the baseline methods in this table maintain two copies of the model parameters, i.e., the compact copy for inference and an fp32 master copy for weight update. These two copies occupy separate memory spaces, i.e., they do not share a memory space. The total memory (indicated by total bitwidth) required for model parameters would be the sum of two parts. However, this is different for RCT because it only keeps one copy of quantized parameters. The weight update requires 8–15 bits on average, whereas inference only uses the

---

**TABLE V**

| Model      | Dataset         | Metric | fp32 | RCT          | Bitwidth |
|------------|-----------------|--------|------|--------------|----------|
| LSC-CNN    | ShanghaiTech    |        |      |              |          |
|            | Part A          | MAE    | 67.808 | 69.275 | 8.81      |
|            |                  | MSE    | 114.414 | 117.765 |          |
| LSC-CNN    | ShanghaiTech    |        |      |              |          |
|            | Part B          | MAE    | 9.585  | 9.573   | 9.469    |
|            |                  | MSE    | 15.55  | 14.515  |          |
| LSC-CNN    | UCF-QNRF        |        |      |              |          |
|            |                  | MAE    | 228.9  | 227.6   | 9.789    |
|            |                  | MSE    | 303.1  | 304.8   |          |

---

https://github.com/val-iisc/lsc-cnn
most significant 8-bit part of the copy, just like we have shown in Fig. 4. Therefore, the total bitwidth for RCT is 8–15 bits, not 16–23 bits.

Every time after the update of the master copy, one has to convert the master copy into a compact one for inference. The conversion of the weight usually requires linear quantization, which involves a few integer and floating-point arithmetic for each element of a tensor. For training ultracompact models, e.g., binary or ternary networks, the conversion even involves quadratic optimization, which is more expensive than linear quantization. In comparison, RCT only requires a truncation or shift for conversion. When the bitwidth for weight update is not greater than 8 bit, no conversion is required.

Industrial implementation of QAT, such as Pytorch or Tensorflow or Q8BERT [23] in this table, employs fp32 master copy and 8 bit for inference. Compared with these baselines, RCT can train effectively with an average of 8–15 bits, which is 20%–37.5% of the baselines. It saves 63.5%–80% memory space and at least the same amount of energy in communication. Lower memory requirement translates to lower cache miss rate and less off-chip communication. As shown in Table II, off-chip communication is three orders of magnitude expensive than computation [48]. The potential saving in energy for communication could be more. This fact holds not only for edge devices but also for servers and workstations.

The training method WAGEUBN [22] quantizes errors and gradients. RCT can also work with quantized errors and gradients. In fact, all experiments of RCT in this article are based on quantized gradients. As we mentioned in Section III-A2, full- and low-precision gradients are equivalent in the weight update stage because extra information in full-precision gradients will be lost due to quantization overflow. We can calculate errors and gradients in reduced precision in the first place to save memory and energy. However, we are not going to have a discussion on the quantization of errors and gradients because our focus is on the quantization of model parameters, which has a higher impact on memory efficiency. A weight tensor has a longer life cycle than that of an error or gradient tensor. The memory space for errors and gradients of one layer can be released once the calculation for that layer is done, as the errors and gradients are not required elsewhere. The model parameters are required in the forward/backward pass and the weight update stage. One has to either keep the model parameters in on-chip memory all the time or load them from

### Table VI

| Method      | Weight for inference | Weight update | Total bitwidth | Shared weight | Weight conversion |
|-------------|----------------------|---------------|----------------|--------------|-----------------|
| XNOR-Net [17] | 1-bit                | fp32          | 33 bits        | No           | Quadratic Optimisation |
| TWN [12]    | 3-bit                | fp32          | 35 bits        | No           | Quadratic Optimisation |
| QNN [20]    | 8-bit                | fp32          | 40 bits        | No           | Linear Quantisation |
| WAGEUBN [22] | 8-bit                | 24-bit        | 32 bits        | No           | Linear Quantisation |
| MP [19]     | fp16                 | fp32          | 48 bits        | No           | Truncation       |
| PG [67]     | fp32                 | fp32          | 32 bits *      | Yes          | None             |
| Q8BERT [23] | 8-bit                | fp32          | 40 bits        | No           | Linear Quantisation |
| Q-BERT [27] | 3.7-bit              | fp32          | 35.7 bits      | No           | Linear Quantisation |
| FF4 [68]    | 4-bit                | fp16          | 20 bits        | No           | Linear Quantisation |
| RCT         | 8-bit                | 8-15 bits     | 8-15 bits *    | Yes          | Truncation or None |

* Weight is shared by training and inference, therefore the total bitwidth is the sum of the two parts, not the sum.

### Table VII

| Init Bitwidth | Averaged Bitwidth | Accuracy |
|---------------|-------------------|----------|
| 4             | Failed            | Failed   |
| 6             | 8.370643          | 0.7141   |
| 8             | 8.557228          | 0.7253   |
| 10            | 9.533863          | 0.721    |
| 12            | 11.37384          | 0.7228   |
| float         | 32                | 0.7239   |

### Table VIII

| Model     | Dataset                | Averaged bitwidth |
|-----------|------------------------|-------------------|
| ResNet-18 | CIFAR-10               | 12.397            |
| ResNet-18 | CIFAR-100              | 10                |
| ResNet-18 | ImageNet               | 11.943            |
| MobileNet-V2 | CIFAR-10               | 11.723           |
| MobileNet-V2 | CIFAR-100              | 10.944           |
| ShuffleNet-V2 | CIFAR-10              | 11.688           |
| ShuffleNet-V2 | CIFAR-100              | 10.576           |
| LSC-CNN   | ShanghaiTech Part A    | 8.81              |
| LSC-CNN   | ShanghaiTech Part B    | 9.469             |

### F. Sensitivity to Initial Bitwidth

We evaluate the sensitivity of RCT with respect to the initial bitwidth for weight update. Table VII shows the training results of VGG16 with the CIFAR-100 dataset [52]. The training settings are exactly the same as that in Section IV-B, except that we alter the initial bitwidth. The initial bitwidth of 4 leads to NAN number in training. This is because the weight in this setting fluctuates largely and runs into infinity very quickly before RCT allocates more bitwidth to the layer. The initial bitwidth of 6 and 8 leads to a very similar averaged bitwidth, which implies that 6–8 bitwidth is essential for training without a master copy. All settings except 4 bit achieved similar accuracy compared to floating-point training. We use 8 bit as the initial setting for RCT. All other experiments are following this setting.

### G. Bitwidth Versus Batch Size

A larger batch size leads to larger gradients and less frequent quantization underflow. This means that a model can learn...
effectively with lower bitwidth. In many edge applications, the batch size of training data is usually quite limited. Micro-batching [69] is one of the methods that can mitigate the limitation in batch size. It is also known as gradient accumulation, which is widely adopted by the NLP research community. This method accumulates gradients from multiple batches before carrying out the model update. With this technique, RCT can have larger gradients and lower quantization underflow, which leads to models with lower bitwidth.

H. Bitwidth Versus Challenge

In the experiment, we observe that the bitwidth required for effective learning is related to the level of challenge of the dataset. The observation is summarized in Table VIII. In the image classification, we train ResNet-18 with CIFAR-10/100 and ImageNet. CIFAR-10 is less challenging than CIFAR-100 and ImageNet. The resulting bitwidth for CIFAR-10 is higher than that for CIFAR-100 and ImageNet. This holds for the experiments on MobileNet-V2 and ShuffleNet-V2. A similar observation is also available from the crowd counting application in Section IV-D. ShanghaiTech Part A has more objects in the dataset and has higher variation in the scale of the objects, which makes it more challenging than Part B. LSC-CNN requires less bitwidth on Part A than on Part B.

We infer from these interesting observations that the more challenging the dataset is, the lower bitwidth is required for training. A challenging dataset has a higher loss, which leads to a larger gradient and higher $G_{gavg}$ for each parameter. Therefore, RCT can save more bitwidth on the dataset. In practice, an edge device might have to learn in the field to adapt to the changing environment. The new incoming data instance would be more challenging than that repeatedly fetched from a dataset and allows the edge AI to learn effectively with low-precision parameters.

I. Overhead Estimation

$G_{gavg}$ metric introduces computation and memory overhead. Each $G_{gavg}$ metric is a floating-point number. The total number of $G_{gavg}$ metric required by RCT is equal to the total number of weight tensors in a model that applies variable bitwidth. For example, in ResNet-18, we have 21 $G_{gavg}$ Metrics in total, 17 for the conv layers, one for the fc layer, and another three for the shortcut of the residual block. The additional memory for the metrics is trivial compared with the 11.4 million learnable parameters of the model.

Equations (2) and (4) depict the calculation of a $G_{gavg}$ metric. The dot product in (4) dominates the computational overhead of $G_{gavg}$. The computation of a gradient tensor is linearly proportional to the size of a batch, whereas $G_{gavg}$ is based on the aggregated gradient tensor. In addition, as described in Algorithm 2, the calculation of $G_{gavg}$ does not happen in every training iteration. Take the training task of ResNet-18 on CIFAR-100 as an example. We use a batch size of 128 and a $G_{gavg}$ evaluation interval equals 100. The computation overhead of $G_{gavg}$ is lower than 0.0078125% that of a gradient tensor.

MeanOut and gradient dithering do not require additional memory space. As described in lines 7–9 of Algorithm 2, the computation overhead of MeanOut and gradient dithering is similar to that of $G_{gavg}$.

The compact format for variable bitwidth introduces additional computation overhead in encoding and decoding. In a naive encoding/decoding scheme, the load operation of a model parameter from an off-chip memory to an on-chip register is followed by a decoding step, which consists of a bit-wise logic operation and a logical operation. The decoding step distributes the packed and unaligned data into separate registers. The store of a model parameter is preceded by an encoding step, which is the reverse operation of the decoding step. The energy cost for the encoding and decoding is at least 2–3 orders of magnitude lower than that of the communication between off-chip memory and on-chip registers. In practice, the overhead can be further reduced by sophisticated encoding/decoding schemes and advanced hardware architecture.

J. Ablation Study

In this section, we perform an ablation study on RCT. We apply the three parts of RCT, i.e., variable bitwidth, MeanOut, and gradient dithering, step by step, to training and compare the performance and memory requirement for model parameters.

Table IX shows the results of the ablation study. With variable bitwidth only, we are able to achieve metric with 1%–2% drops compared to its fp32 counterpart. MeanOut does not have a large improvement in terms of metrics but reduces the bitwidth for weight update by about 1 bit. Finally, gradient dithering improves the performance metric by a little bit without changing the bitwidth significantly.

V. Conclusion

In this article, we propose an RCT method for efficient training of quantized neural networks. Unlike traditional QAT
methods, RCT does not employ a master copy during training. It applies variable bitwidth, MeanOut, and gradient dithering techniques to train the quantized model efficiently and saves memory space and energy for communication. Experiments suggest that RCT generalizes well to image classification, NLP tasks, and real-life crowd counting applications. The experiments also provide insight into the bitwidth requirement for effective learning.

REFERENCES

[1] R. Krishnamoorthi, “Quantizing deep convolutional networks for efficient inference,” in Proc. 32nd AAAI Conf. Artif. Intell., 2018, pp. 1737–1746.
[2] A. H. Zadeh, I. Edo, O. M. Awad, and A. Moshovos, “QGOB: Quantizing attention-based NLP models for low latency and energy efficient inference,” in Proc. 53rd Ann. IEEE/ACM Int. Symp. Microarchitecture (MICRO), Oct. 2020, pp. 811–824.
[3] J. Fung, A. Shafiee, H. Abdel-Aziz, D. Thorsley, G. Georgiadis, and J. Hassoun, “Post-training piecewise linear quantization for deep neural networks,” 2020, arXiv:2002.00104.
[4] A. H. Zadeh, I. Edo, O. M. Awad, and A. Moshovos, “QGOB: Quantizing attention-based NLP models for low latency and energy efficient inference,” in Proc. 53rd Ann. IEEE/ACM Int. Symp. Microarchitecture (MICRO), Oct. 2020, pp. 811–824.
[5] T. Luo et al., “NC-Net: Efficient neuromorphic computing using aggregated sub-nets on a crossbar-based architecture with non-volatile memory,” IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., early access, Oct. 14, 2021, doi: 10.1109/TCAD.2021.3120068.
[6] Z. Wang, T. Luo, R. S. M. Goh, and J. T. Zhu, “EDCompression: Energy-aware model compression for dataflows,” IEEE Trans. Neural Netw. Learn. Syst., early access, May 13, 2022, doi: 10.1109/TNNLS.2022.3172941.
[7] S. Han, H. Mao, and W. J. Dally, “Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding,” 2015, arXiv:1510.0049.
[8] J. Frankle and M. Carbin, “The lottery ticket hypothesis: Finding sparse, trainable neural networks,” 2018, arXiv:1806.03635.
[9] T. Dettmers and L. Zettlemoyer, “Sparse networks from scratch: Faster training without losing performance,” 2019, arXiv:1907.04840.
[10] Z. Gale, E. Olsen, and S. Hooker, “The state of sparsity in deep neural networks,” Feb. 2019, arXiv:1902.09574.
[11] K. Guo, S. Zeng, J. Yu, Y. Wang, and H. Yang, “A survey of FPGA-based neural network accelerator,” 2017, arXiv:1712.08934.
[12] N. S. Sohoni, C. R. Aberger, M. Leszczynski, J. Zhang, and C. Ré, “Low-memory neural network training: A technical report,” 2019, arXiv:1904.10631.
[13] J. Gou, B. Yu, S. J. Maybank, and D. Tao, “Knowledge distillation: A survey,” 2020, arXiv:2006.05525.
[14] C. Li and C. R. Shi, “Constrained optimization based low-rank approximation of deep neural networks,” in Proc. Eur. Conf. Comput. Vis. (ECCV), 2018, pp. 365–382.
[15] F. Li, B. Zhang, and B. Liu, “Ternary weight networks,” 2016, arXiv:1605.04711.
[16] H. Bai et al., “BinaryBERT: Pushing the limit of BERT quantization,” 2020, arXiv:2012.15701.
[17] H. Kim, K. Kim, J. Kim, and J.-J. Kim, “BinaryDuo: Reducing gradient mismatch in binary activation network by coupling binary activations,” 2020, arXiv:2002.06517.
[18] C. Liu et al., “RBCN: Rectified binary convolutional networks for enhancing the performance of 1-bit DCNNs,” 2019, arXiv:1908.07748.
[19] H. Peng and S. Chen, “BDNN: Binary convolution neural networks for fast object detection,” Pattern Recognit. Lett., vol. 125, pp. 91–97, Jul. 2019.
[20] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, “XNOR-net: ImageNet classification using binary convolutional neural networks,” in Proc. Eur. Conf. Comput. Vis. (ECCV), 2018, pp. 365–382.
[21] F. Li, B. Zhang, and B. Liu, “Ternary weight networks,” 2016, arXiv:1605.04711.
[22] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, “DeReLU-Net: Training low-bitwidth convolutional neural networks with low-bitwidth gradients,” 2016, arXiv:1606.01610.
[23] P. Micikevicius et al., “Mixed precision training,” 2017, arXiv:1710.03740.
[24] R. Banner, I. Hubara, E. Hoffer, and D. Soudry, “Scalable methods for low-rank training of neural networks,” 2018, arXiv:1805.11046.
[25] S. Wu, G. Li, F. Chen, and L. Shi, “Training and inference with integers in deep neural networks,” 2018, arXiv:1802.04680.
[26] Y. Yang, S. Wu, L. Deng, T. Yan, Y. Xie, and G. Li, “Training high-performance and large-scale deep neural networks with full 8-bit integers,” Neural Netw., vol. 125, pp. 70–82, May 2020.
[27] G. Zhou and Z. Li, “Training low-bitwidth deep neural networks using binary representation,” 2020, arXiv:2002.07686.
[28] C. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han, “HAQ: Hardware-aware automated quantization with mixed precision,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., Jun. 2019, pp. 8612–8620.
[53] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., Jun. 2016, pp. 770–778.

[54] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, “MobileNetV2: Inverted residuals and linear bottlenecks,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., Jun. 2018, pp. 4510–4520.

[55] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, “ShuffleNet V2: Practical guidelines for efficient CNN architecture design,” in Proc. Eur. Conf. Comput. Vis. (ECCV), Sep. 2018, pp. 116–131.

[56] N. Vaswani et al., “Attention is all you need,” 2017, arXiv:1706.03762.

[57] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of deep bidirectional transformers for language understanding,” 2018, arXiv:1810.04805.

[58] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman, “GLUE: A multi-task benchmark and analysis platform for natural language understanding,” 2018, arXiv:1804.07461.

[59] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, “SQuAD: 100,000+ questions for machine comprehension of text,” 2016, arXiv:1606.05250.

[60] G. Gao, J. Gao, Q. Liu, Q. Wang, and Y. Wang, “CNN-based density estimation and crowd counting: A survey,” 2020, arXiv:2003.12783.

[61] J. Shao, K. Kang, C. C. Loy, and X. Wang, “Deeply learned attributes for crowded scene understanding,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2015, pp. 4657–4666.

[62] B. Zhou, X. Wang, and X. Tang, “Understanding collective crowd behaviors: Learning a mixture model of dynamic pedestrian-agents,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2012, pp. 2871–2878.

[63] A. B. Chan, Z.-S. John Liang, and N. Vasconcelos, “Privacy preserving crowd monitoring: Counting people without people models or tracking,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., Jun. 2008, pp. 1–7.

[64] D. B. Sum, S. V. Peri, M. N. Sundararaman, A. Kamath, and V. B. Radhakrishnan, “Locate, size and count: Accurately resolving people in dense crowds via detection,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 43, no. 8, pp. 2739–2751, Feb. 2020.

[65] H. Idrees et al., “Composition loss for counting, density map estimation and localization in dense crowds,” in Proc. Eur. Conf. Comput. Vis. (ECCV), 2018, pp. 544–559.

[66] D. Lian, J. Li, J. Zheng, W. Luo, and S. Gao, “Density map regression guided detection network for RGB-D crowd counting and localization,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2019, pp. 1821–1830.

[67] Y. Zhang, R. Zhao, W. Hua, N. Xu, G. E. Suh, and Z. Zhang, “Precision gating: Improving neural network efficiency with dynamic dual-precision activations,” in Proc. Int. Conf. Learn. Represent., Feb. 2020, pp. 1–14. [Online]. Available: https://openreview.net/forum?id=S1gVU0EKwS

[68] X. Sun et al., “Ultra-low precision 4-bit training of deep neural networks,” in Proc. Adv. Neural Inf. Process. Syst., vol. 33, 2020, pp. 1–12.

[69] Y. Huang et al., “GPipe: Efficient training of giant neural networks using pipeline parallelism,” 2018, arXiv:1811.06965.

[70] Tian Huang received the B.Sc. and Ph.D. degrees from Shanghai Jiao Tong University, Shanghai, China, in 2008 and 2016, respectively. He is currently a Research Scientist with the Institute of High Performance Computing, Agency for Science Technology and Research, Singapore. His research interests lie in high-performance computing and machine learning algorithms.

[71] Liu Guoh received the Ph.D. degree in electrical and computer engineering from the National University of Singapore, Singapore, in 2006. He is currently the Director of the Computing Science Department, Institute of High Performance Computing (IHPC), Agency for Science Technology and Research, Singapore. The research focus areas include artificial intelligence (AI), high-performance computing (HPC), distributed computing, machine and deep learning, complex systems, human–machine interaction, and modeling and simulation.

[72] Ming Yan graduated from Sichuan University, Chengdu, China. He was a Visiting Ph.D. Student at Georgia State University, Atlanta, GA, USA, in 2018. He is currently a Scientist with the Agency for Science Technology and Research, Singapore. He has published several papers on ACL, KBS, IEEE TRANSACTIONS, AND NEURAL COMPUTING. His research interests include neural networks and natural language processing.

[73] Joeq Tianyi Zhou (Senior Member, IEEE) received the Ph.D. degree in computer science from Nanyang Technological University (NTU), Singapore, in 2015. He is currently a Senior Scientist, an Investigator, and the Group Manager with the Center for Frontier AI Research (CFAR), Agency for Science Technology and Research, Singapore. He is also holding an adjunct faculty position (Adjunct Associate Professor) at the National University of Singapore (NUS), Singapore. Before working at the Institute of High Performance Computing (IHPC), Agency for Science Technology and Research, Singapore, he was a Senior Research Engineer with the SONY U.S. Research Center, San Jose, CA, USA. His current research interests mainly focus on improving the efficiency and robustness of machine learning algorithms.

Dr. Zhou organized the ICDCS’ 2020–2021 Workshop on Efficient AI meets Edge Computing, the ACML’16 Workshop on Learning on Big Data Workshop, and the IJCAI’19 Workshop on Multi-Output Learning. He received the NeurIPS Best Reviewer Award in 2017. He is serving as an Associate Editor for IEEE TRANSACTIONS ON EMERGING TOPICS IN COMPUTATIONAL INTELLIGENCE (TETCI), IEEE ACCESS, and IET Image Processing and the TPC Chair of Mobimedia 2020.