An Energy-and-Area-Efficient CNN Accelerator for Universal Powers-of-Two Quantization

Tian Xia, *Member, IEEE*, Boran Zhao, Jian Ma, Gelin Fu, Wenzhe Zhao, Nanning Zheng, *Fellow, IEEE*, and Pengju Ren, *Member, IEEE*

***Abstract*— CNN model computation on edge devices is tightly restricted to the limited resource and power budgets, which motivates the low-bit quantization technology to compress CNN models into 4-bit or lower format to reduce the model size and increase hardware efficiency. Most current low-bit quantization methods use uniform quantization that maps weight and activation values onto evenly-distributed levels, which usually results in accuracy loss due to distribution mismatch. Meanwhile, some non-uniform quantization methods propose specialized representation that can better match various distribution shapes but are usually difficult to be efficiently accelerated on hardware. In order to achieve low-bit quantization with high accuracy and hardware efficiency, this paper proposes Universal Power-of-Two (UPoT), a novel low-bit quantization method that represents values as the addition of multiple power-of-two values selected from a series of subsets. By updating the subset contents, UPoT can provide adaptive quantization levels for various distributions. For each CNN model layer, UPoT automatically searches for the optimized distribution that minimizes the quantization error. Moreover, we design an efficient accelerator system with specifically optimized power-of-two multipliers and requantization units. Evaluations show that the proposed architecture can provide high-performance CNN inference with reduced circuit area and energy, and outperforms several mainstream CNN accelerators with higher (8×–65×) area efficiency and (2×–19×) energy efficiency. Further experiments of 4/3/2-bit quantization on ResNet18/50, MobileNet_V2 and EfficientNet models show that our UPoT can achieve high model accuracy which greatly outperform other state-of-the-art low-bit quantization methods by 0.3%–6%. The results indicate that our approach provides a highly-efficient accelerator for low-bit CNN model quantization with low hardware overheads and good model accuracy.**

***Index Terms*— CNN accelerator, power-of-two quantization, circuit design, ASIC implementation, hardware system.**

I. Introduction

CONVOLUTION Neural Networks (CNNs) have achieved remarkable performance in many computer vision tasks. However, the booming CNN models also bring a huge amount of parameters and computational costs, making the practical deployment very challenging for edge devices with limited resources and power budgets. Therefore, many state-of-the-art CNN accelerators deploy quantization technique to compress floating-point CNN models into low-bit representation [1] in order to reduce memory footprint and resource consumption.

To mitigate the accuracy degradation caused by quantization, many methods use the Quantization-Aware Training (QAT) scheme [2] which can recover the model accuracy by retraining the model weights with consideration of quantization errors. QAT is especially useful for lower bit-width setting and quantization-sensitive lightweight models such as MobileNet [3]. Most current QAT methods use uniform quantization that converts the full-precision floating-point tensors of CNN model weights and activation values into low-bit integer tensors [4]. By doing so, the CNN computation can be fulfilled with integer adders and multipliers (such as INT8 or INT4 MAC unit) so that the overhead for computation and circuit implementation can be reduced [5]. Therefore uniform quantization is supported by most existing accelerators, like NVDLA [6] and Eyeriss [7]. However, this method has fixed resolution as its quantization levels are in uniform distribution, i.e. the intervals among quantization levels are equal, making it hard to efficiently represent CNN weight and activation values that are usually not evenly distributed [8], [9]. Fig 1 shows the effect of a well-known uniform quantization method LSQ [2] on lightweight model MobileNet_V2 [3], where the quantization errors are high because the distribution of original values is not uniform. Such *Distribution-Mismatch* problem is the main cause of accuracy degradation after quantization. Therefore, some recent studies propose non-uniform quantization [10], [11] [9], which aims to specifically design the low-bit representation in order to reach finer resolutions in the dense area of weights and activation distributions.

Meanwhile, hardware acceleration is a critical bottleneck [12], [13] for CNN model inference on edge devices where hardware resources are very limited. Pursuing high energy and area efficiency circuit system is a key factor for CNN accelerators [14], [15]. However, it is usually hard to map non-uniform quantization methods onto current CNN accelerators as they require special computation units. Among the current non-uniform approaches, logarithmic quantization [16] gains the most attention for hardware acceleration as it represent numbers in power-of-two values so that the multiplication can be fulfilled with shift-and-add operations, which has great potentials in saving the circuit energy and area. Studies have shown that logarithmic quantization can better represent the single-peak bell-shaped distribution in CNN models. For example, the state-of-the-art logarithmic method APOT [8] builds special power-of-two value sets to

Manuscript received 22 July 2022; revised 9 October 2022 and 10 November 2022; accepted 5 December 2022. Date of publication 27 December 2022; date of current version 27 February 2023. This work was supported in part by the National Key Research and Development Program of China under Grant 2021YFB0300900, in part by the National Natural Science Foundation of China under Grant 62088102, and in part by the Key Research and Development Program of Shaanxi under Grant 2022ZDLGY01-08. This article was recommended by Associate Editor W. Liu. *(Tian Xia and Boran Zhao contributed equally to this work.) (Corresponding author: Pengju Ren.)*

The authors are with the Institute of Artificial Intelligence and Robotics, Xi'an Jiaotong University, Xi'an, Shaanxi 710049, China (e-mail: pengjuren@xjtu.edu.cn).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TCSI.2022.3227608.

Digital Object Identifier 10.1109/TCSI.2022.3227608

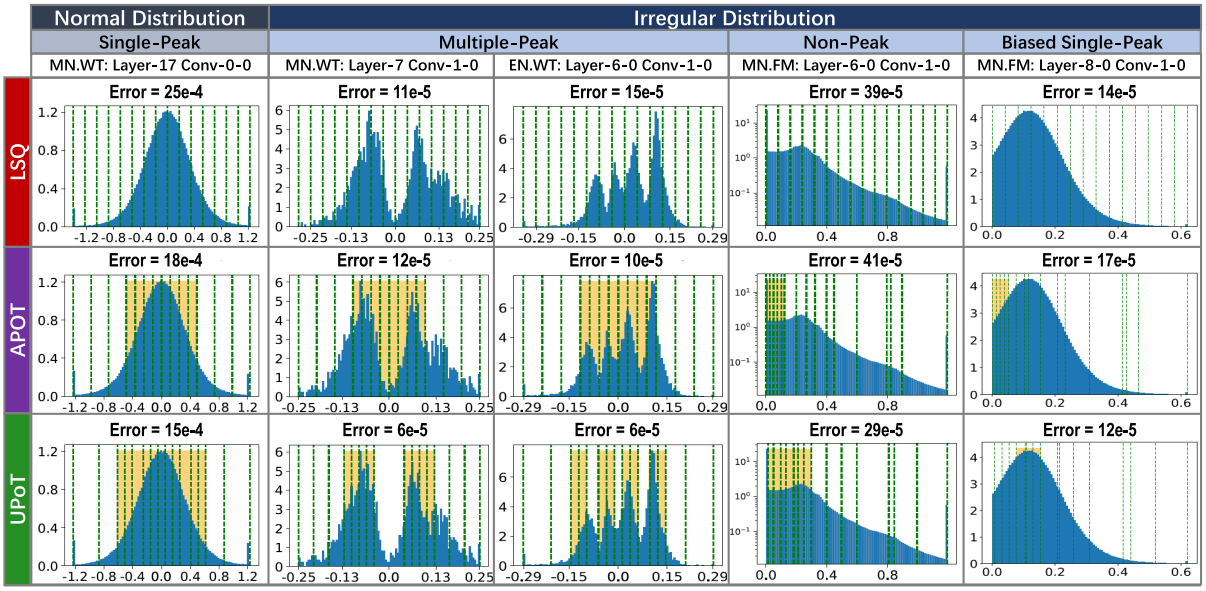

Fig. 1. Example distributions of weights (WT) and feature map values (FM) from MobileNet_V2 (MN) and EfficientNet-B0 (EN) layers. Green dotted lines depict the distribution of quantization levels provided by multiple low-bit quantization methods including uniform quantization (LSQ), state-of-the-art logarithmic quantization APOT and ours quantization method (UPoT). Highlight zones mark high-resolution area of different quantization methods. Quantization errors of each method are measured by MSE. APOT is good on single-peak distribution but poor on other shapes. UPoT flexibly adjusts low-bit values allocation according to distributions and achieve the lowest errors.

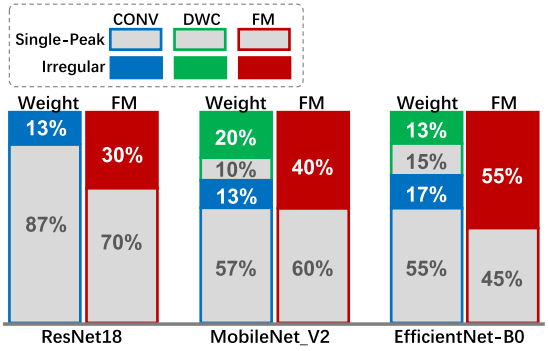

Fig. 2. Proportions of irregularly-distributed convolution weights (CONV), depth-wise convolution weights (DWC) and feature maps (FM) in ResNet18, MobileNet_V2 and EfficientNet-B0 models.

assign more quantization levels around the center but less levels on the tails. As shown in Fig 1, APOT mitigates the quantization errors for the single-peak distributions. However, modern lightweight models commonly have layers with more irregular distributions that are difficult to be represented with APOT or other existing logarithmic methods [9]. As depicted in Fig 1, such irregular distribution shapes include multi-peaks, biased single peaks and even non-peak shape with irregular shapes. On these layers, APOT may have even higher errors than uniform quantization. Moreover, Fig 2 lists the proportion of data distribution types in three mainstream lightweight models including ResNet18, MobileNet_V2 and EfficientNet [17]. It can be observed that a large part of weights and activation values are in irregular distributions, which leads to accuracy degradation [9]. Actually, the ability to represent irregular distributions remains a critical challenge for logarithmic quantization techniques.

Meanwhile, though logarithmic quantization is theoretically favorable for hardware implementation, prior studies [8], [16] mostly focused on the representation method and training algorithms but overlooked the hardware design. Therefore, it is still an open challenge to build an efficient power-of-two accelerator that can process real CNN models. Specifically, the computation unit should be able to process power-of-two multiplications in low hardware cost. Meanwhile, embedding this novel multiplier into CNN accelerator would cause additional problems. For example, there are operations that still use integer format such as accumulation, pooling and biasing, which requires the accelerator to efficiently re-quantize integer values into power-of-two format for successive layers. The on-chip buffers should also be updated to fit with the low-bit exponent values. Moreover, the accelerator should be general enough to support more flexible logarithmic representations in order to maintain its accuracy on the irregular distributions of lightweight models as in Fig 2. Therefore, the accelerator design should meet the demands for high accuracy, inference performance and hardware efficiency.

In this paper, we address the above challenges by proposing an flexible low-bit logarithmic quantization method named Universal Power-of-Two (UPoT), which quantizes original values as the addition of multiple flexible sets of power-of-two numbers. By automatically adjusting the included numbers of each set, UPoT matches the various distributions of CNN models with much higher resolutions. An adaptive learnable quantizer is proposed to guarantee the efficient training of UPoT parameters. As shown in Fig 1, UPoT quantization achieves the minimal quantization errors for all kinds of distributions. More importantly, we design a complete accelerator system that can efficiently run full inference of UPoT-quantized CNN models with good accuracy and low energy-and-area cost. Meanwhile, as UPoT is flexible enough to be compatible with various kinds of logarithmic representations, the proposed architecture can actually support some state-of-the-art power-of-two methods including Log2 [16] and APOT [8], making our architecture an universal solution for the general methods of logarithmic quantization. To the best of our knowledge,

this is the first accelerator system design dedicated for general logarithmic quantization. Evaluation results indicate that UPoT achieves outstanding performance on ImageNet dataset for both regular models (i.e. ResNet50) and lightweight models (i.e. MobileNet_V2, EfficientNet-B0), while the hardware implementation of ASIC accelerator prototype reports much lower circuit area and power consumption compared with some mainstream CNN accelerators, including Eyeriss [7], NVDLA [6] and PermCNN [18]. Our results prove that the proposed UPoT algorithm and accelerator are potential to be applied in real-world systems as they can fully exploit the benefits of logarithmic quantization to achieve high hardware efficiency without sacrificing the model accuracy.

II. Backgrounds and Related Works

A. Uniform Quantization

Many current CNN model compression methods use the uniform quantization to convert full-precision floating-point CNN model parameters and activation values to 4-bit or lower-bit data. It usually takes complex and expensive retraining process to fine-tune the quantized parameters so that the quantized model's accuracy can be guaranteed. For example, LSQ [2] and LSQ+ [19] proposed to train the step size in conjunction with network parameters and used learnable offsets and parameter initialization to improve the accuracy. More recent works try to build new quantizers whose gradients are friendly for retrain. DAQ [20] introduced a distance-aware quantizer to alleviate the gradient mismatch problem. EWGS [21] scales the gradients of quantizer depending on the quantization errors to improve the learning speed. Generally, the quantization process can be formulated as:

$$V_q = quantize(V_f, s, Q) \tag{1}$$

where V_f is the original full-precision values, and s is the scale factor. The *quantize* function converts $\frac{V_f}{s}$ to a low precision value V_q by using the discrete quantization levels provided by $Q = \{q_0, q_1, \ldots, q_{n-1}\}$. For the uniform quantization, V_f is mapped to b-bit integers as:

$$Q^u = \{-2^{b-1}, -2^{b-1}+1, \ldots, 2^{b-1}-1\} \tag{2}$$

Generally, uniform quantization is widely used to accelerate CNN, as low-bit integer operations are widely available in current commercial GPU and CPU processors. For example, staring from Turing architecture, NVIDIA GPU provides high-throughput INT8/INT4 inference to provide general support for wide-spectrum CNN models. Meanwhile, integer formats are also favored by both general-purpose NPUs (e.g. Eyeriss [7] and NVDLA [6]) and specially-designed accelerators such like PermCNN [18] as they can simplify use standard computation units in their designs. However, the main weakness of uniform quantization is the *Distribution-Mismatch* problem that makes it difficult to efficiently represent non-uniform distributions which are common for CNN models [9]. Furthermore, the heavy overhead of integer multiplier circuits still remain a critical bottleneck for edge or endpoint devices.

B. Logarithmic Quantization

As a potential method to provide more efficient representation with less hardware overhead, logarithmic quantization receives increasing attention recently. Such works assume that most CNN model weight and activation values are concentrated around zero, and thus can be represented as power-of-two values using low-bit formats. A typical power-of-two method was proposed by Log2 [16] that used b bits to represent the following quantization levels:

$$Q^p = s \times \{0, \pm 2^0, \pm 2^1, \ldots, \pm 2^{2^{b-2}-1}\} \tag{3}$$

where s is the scale factor. State-of-the-art method APOT [8] improves its representation ability by using multiple subsets of power-of-two values as:

$$Q^{ap} = s \times \left\{ \sum_{i=0}^{n-1} p_i \right\}, \text{ and } p_i \in \left\{0, 2^i, 2^{i+n}, \ldots, 2^{i+(2^k-2)n}\right\} \tag{4}$$

where exists 1-bit sign and (b-1) bits are divided as n parts of k-width bits. The combination of subsets generates more quantization levels near zero, making it efficient for zero-central single-peak distributions. However, as the contents of subsets are predetermined, APOT lacks capability to represent other irregular distributions such as multi-peaks [9]. Meanwhile, most existing studies have no or little discussions on using power-of-two format to build an efficient acceleration system, leaving many unclear problems, e.g. the power-of-two processing unit, the re-quantization between CNN layers, the system scale, etc. One FPGA accelerator design is proposed by MSQ [22], which also leverages the addition of power-of-two format to represent Gaussian-like distributions, so the problem of irregular distribution still remains. And it only quantize weights and uses integer to represent activation values. Meanwhile, ILM [23] proposes an optimized integer multiplier which first maps input operands into logarithmic formats and then use shift-and-add manner to efficiently compute the results. However, ILM is mostly focusing on the design of a multiplier circuit instead of a quantization method. As a result, the claimed advantages of logarithmic quantization are still an open challenge for current CNN accelerator designs.

C. Other Non-Uniform Quantization

Besides of logarithmic quantization, there are other non-uniform quantization methods to handle the *Distribution-Mismatch* problem: DMBQ [24] uses Laplace distribution to generate denser quantization levels around the selected area; QIL [10] learns better quantization intervals with a customized non-linear transformer; LCQ [11] uses learnable companding to adjust its quantized distribution; LQ-Nets [25] and DDQ [9] propose to automatically learn low-bit data representation for multi-peak distributions. Generally, though their representations are more flexible and diverse, using highly-customized formats makes them difficult to be efficiently computed with standard computation units. Instead they require dedicated complex circuits for acceleration, which undermines the benefits of low-bit quantization. Moreover, as the customized formats usually can't be directly used in the biasing, pooling or normalization operations, some methods such as LQ-Nets and DDQ, still uses floating-point values in their inference process, making their circuit implementation even more complex. DRQ [26] improves DNN accelerator

efficiency by selecting appropriate precision (16-bit, 8-bit or 4-bit) according to the sensitive of different image regions. But the use of flexible precision will inevitably lead to the inefficiency of hardware usage. Therefore, non-uniform quantization techniques are rarely used in real accelerator systems.

This paper shows that by extending logarithmic quantization with more adaptive quantization levels, UPoT can have equivalent or even better model accuracy than multiple state-of-the-art non-uniform quantization approaches. Meanwhile, by fully exploiting the advantage of power-of-two formats, UPoT implements a hardware design that outperform multiple state-of-the-art CNN accelerators with much less area and energy costs. With this outcome, we prove that power-of-two format is applicable to accelerate real-world CNN models.

III. Universal Power-of-Two Quantization

A. Representation Method

The proposed Universal Power-of-Two Quantization (UPoT) is a logarithmic quantization method that maps floating-point values to low-bit data representation. Consider the total bit-width of representation is b, we can divide b bits (or $b-1$ bits for signed values) into n segments that each has $\{b_i\}$-width. Then we generate the set of available quantization levels as:

$$\begin{aligned} Q^{up} &= s \times \left\{ \sum_{i=0}^{n-1} p_i \right\}, \text{ and} \\ p_i &\in \left\{ \mathbf{E}_i[0], \mathbf{E}_i[1], \ldots, \mathbf{E}_i[2^{b_i}-1] \right\} \\ &= \left\{ q_0, q_1, \ldots, q_{(2^{(\Sigma b_i)}-1)} \right\} \end{aligned} \tag{5}$$

where the quantization values of Q^{up} set are the combination of n subsets of values $(\mathbf{E}_0, \mathbf{E}_1, \ldots, \mathbf{E}_{n-1})$ that are indexed via the $\{b_i\}$-width segments. To determine the contents in $\mathbf{E}_i$, we first build a ***Candidate Set*** as $\mathcal{C}=\{0, 2^0, 2^1, 2^2, .., 2^{K-2}\}$ that contains K values including zero and K-1 power-of-two values. For each set of $\mathbf{E}_i$, we select 2^{b_i} values from $\mathcal{C}$ as:

$$\mathbf{E}_i = \{e_i^0, e_i^1, \ldots, e_i^{2^{b_i}-1}\} \subset \{0, 2^0, 2^1, 2^2, .., 2^{K-2}\} \tag{6}$$

Note that different settings of $\mathbf{E}_i$ contents will generate different Q^{up} set. In our approach, UPoT automatically adapts the elements of each $\mathbf{E}_i$ set so that the generated quantization levels of Q^{up} can better match the distribution of original values, as to be explained in the following sections.

The amount of subsets is according to the desired bitwidth b and the distribution of original values. For example, a 6-bit quantization value can be broken into three 2-bit parts so that Q^{up} is composed by the combination of three $\mathbf{E}_i$ subsets. In this paper we mainly focus on 4-bit or lower-bit quantization. With this bitwidth, we found that two subsets ($\mathbf{E}_0$, $\mathbf{E}_1$) are usually adequate to represent irregular distributions for most CNN models. Therefore, without loss of generality, we mainly discuss the case of two subsets in the following paper. In this case, we can divide b bits into two parts:

$$\begin{cases} b_0 = \lceil \frac{b-1}{2} \rceil \text{ and } b_1 = \lfloor \frac{b-1}{2} \rfloor, & \text{if signed} \\ b_0 = \lceil \frac{b}{2} \rceil \text{ and } b_1 = \lfloor \frac{b}{2} \rfloor, & \text{else} \end{cases} \tag{7}$$

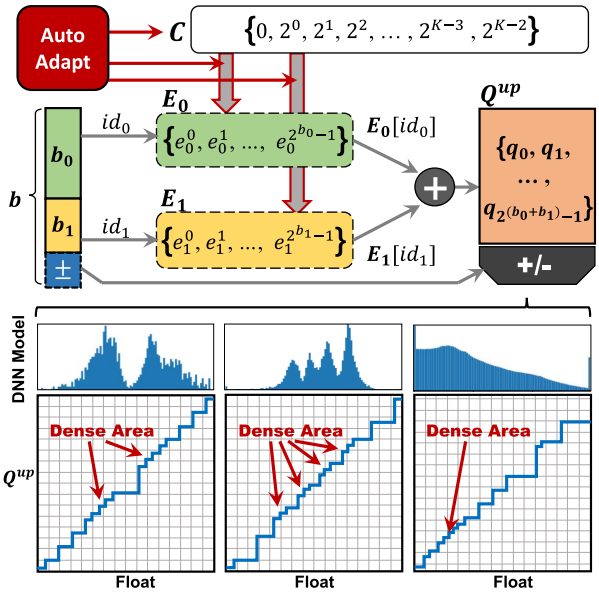

Fig. 3. Example of 4-bit representation of UPoT. The quantization level set Q^{up} is the combination of two subsets $\mathbf{E}_0$ and $\mathbf{E}_1$, whose elements are automatically selected from the $\mathcal{C}$ set that only contains powers-of-two and zero. By adapting the contents of $\mathbf{E}_0$ and $\mathbf{E}_1$, Q^{up} can fit with various distribution.

where $\lceil \cdot \rceil$ and $\lfloor \cdot \rfloor$ are ceil and floor operators. The two parts are used as two integer indices, i.e. id_0 and id_1, that select elements from $\mathbf{E}_0$ and $\mathbf{E}_1$ to add, as follows:

$$Q^{up} = \begin{cases} \pm s \times \{e_0^{id_0} + e_1^{id_1}\}, & \text{if signed} \\ s \times \{e_0^{id_0} + e_1^{id_1}\}, & \text{else,} \end{cases} \quad e_0^{id_0} \in \mathbf{E}_0, e_1^{id_1} \in \mathbf{E}_1 \tag{8}$$

Fig 3 depicts the representation method of UPoT in a case of two subsets. As shown, by selecting different values from the $\mathcal{C}$ set to construct different $\mathbf{E}_0$ and $\mathbf{E}_1$ subsets, we can adjust the available quantization levels in Q^{up} set and form various distributions in order to fit with different CNN layers.

It should be noted that UPoT is flexible enough to support other logarithmic quantization methods such as Log2 [16] and APOT [8]. Log2 uses the entire b-bit value to directly represent the exponent number as $\{0, 2^0, 2^1, \ldots, 2^{2^b-1}\}$, which is similar with UPoT when n=1 and b_0=b. Meanwhile, if UPoT constructs each subset to be $\mathbf{E}_i$=$\{0, 2^i, 2^{i+n}, \ldots, 2^{i+(2^{b_i}-2)n}\}$, it has the equivalent representation[1] of APOT as shown in Eq 4. Therefore, UPoT is an universal representation method for wide-spectrum power-of-two quantization formats. Meanwhile, as UPoT is capable to automatically adjust the content of subsets, it is much more adaptive with different distribution shapes and achieves better performance on CNN models.

B. Training Method

UPoT applies quantization-aware training (QAT) technique to fine-tune trained CNN models by embedding the quantization process into the inference path so that the quantization errors are optimized during the training. Specifically, consider the original forward path of a convolution layer is as Fig 4(a), we insert *Quantizer* nodes to map floating-point values $\mathcal{V}_f$

[1]The value of b_i equals to k in Eq 4, which are both the bitwidth of segment divided from b.

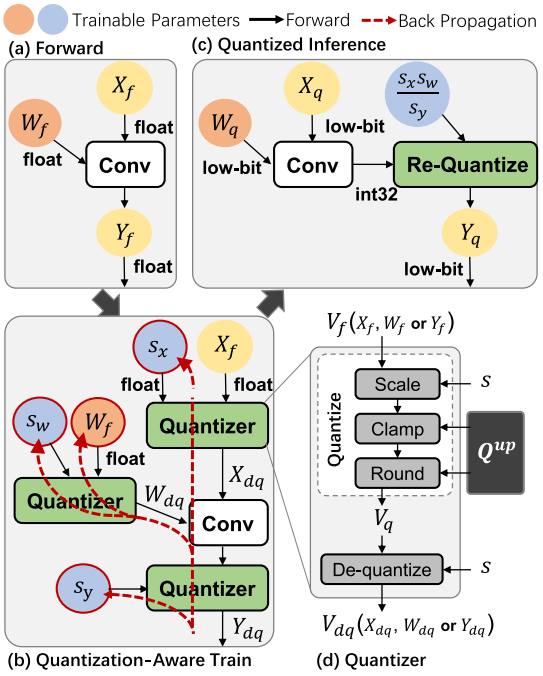

Fig. 4. (a)(b) The schemes of forward and backward propagation of UPoT Quantization-Aware Training where both weights and scale factors are learned. (c) The process of quantized inference using UPoT quantization. (d) The steps of *Quantizer* node is also depicted.

to discrete quantized values $\mathcal{V}_{dq}$ before they are computed in convolution, as depicted in Fig 4(b). Fig 4 shows the steps of UPoT *Quantizer* that uses the symmetric scheme of *quantize* and *de-quantize*. The steps of *quantize* is as below:

$$\mathcal{V}_n = \frac{\mathcal{V}_f}{s} \tag{9}$$

$$\mathcal{V}_c = clamp(\mathcal{V}_n, Q_{min}, Q_{max}) \tag{10}$$

$$\mathcal{V}_q = round(\mathcal{V}_c, Q^{up}) \tag{11}$$

where $\mathcal{V}_f$ is first scaled by s and then get truncated by *clamp* using Q_{min} and Q_{max} that are the minimum/maximum values in Q^{up} set. Then we use *round* function to select the nearest value in Q^{up} set and generate $\mathcal{V}_q$, which is the quantized low-bit representation for $\mathcal{V}_f$. Then the *de-quantize* step is used to restore the quantized value to similar scale of $\mathcal{V}_f$ by:

$$\mathcal{V}_{dq} = \mathcal{V}_q \times s \tag{12}$$

The generated $\mathcal{V}_{dq}$ is used for the forward computation of convolution layer, as depicted in Fig 4(b).

Since $\mathcal{V}_{dq}$ inflects UPoT quantization process, we simultaneously learn weights (W_f) and scale factors (s_x, s_w) during back-propagation in order to minimize the quantization errors. We use the Straight Through Estimator to generate the back-propagation gradient of $round$ function as $\frac{\partial \mathcal{V}_q}{\partial \mathcal{V}_c}=1$. Then we specifically design the gradients of output $\mathcal{V}_{dq}$ with respect to $\mathcal{V}_f$ and s as:

$$\frac{\partial \mathcal{V}_{dq}}{\partial \mathcal{V}_f} = \begin{cases} 1, & \text{if } Q_{min} < \mathcal{V}_n < Q_{max} \\ 0, & \text{else} \end{cases} \tag{13}$$

$$\frac{\partial \mathcal{V}_{dq}}{\partial s} = \begin{cases} -\mathcal{V}_n + \mathcal{V}_q, & \text{if } Q_{min} < \mathcal{V}_n < Q_{max} \\ \mathcal{V}_q, & \text{else} \end{cases} \tag{14}$$

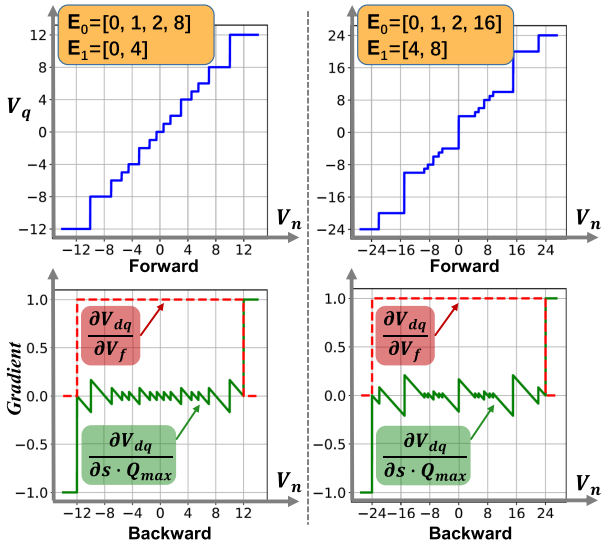

Fig. 5. Examples of forward calculation of V_q in respect of V_n, and corresponding backward propagation gradients of s and V_f under two different configurations of $\{\mathbf{E}_0, \mathbf{E}_1\}$.

We always have $\frac{\partial \mathcal{V}_{dq}}{\partial \mathcal{V}_f}=1$ within the range $[Q_{min}, Q_{max}]$. On the other hand, the gradient value of $\frac{\partial \mathcal{V}_{dq}}{\partial s}$ is determined by the distance between $\mathcal{V}_n$ and $\mathcal{V}_q$, meaning it is highly sensitive to the distribution of quantization levels in Q^{up}. In this way, values that are far from quantization levels of $\mathcal{V}_q$ have more influences during the back-propagation update, driving the *Quantizer* to match various distributions. Fig 5 depicts the *Quantizer*'s forward and backward propagation when Q^{up} is based on different $\mathbf{E}_0$ and $\mathbf{E}_1$ contents. It shows that the proposed gradients are adapted according to Q^{up} so the learning process is efficient to find optimal quantized parameters for different CNN weights and activation distributions.

After the fine-tune process, we get the quantized values of weights ($\mathcal{V}_q$ in Fig 4) and the scale factors of each *Quantizer*, which are saved as the compressed low-bit quantized model.[2] Fig 4(d) shows the inference flow using quantized parameters, where low-bit weights and feature maps are directly computed in convolution, and a *Requantize* node is used to transform the convolution results into quantization as:

$$\begin{aligned} Y_q \times s_y &= ReQuantize\big((X_q \times s_x) * (W_q \times s_w)\big) \\ Y_q &= ReQuantize\Big((X_q * W_q) \times \frac{s_x \times s_w}{s_y}\Big) \end{aligned} \tag{15}$$

where $(*)$ is convolution operator. Therefore, most operations in the convolution layer are based on power-of-two formats.

C. Automatic Adaption

One critical advantage of UPoT is that it can automatically adapt the available quantization levels in Q^{up} to match various distributions of different CNN layers, by searching for the optimized combination of $\{\mathbf{E}_i\}$ subsets. Specifically, for a floating-point tensor $\mathcal{V}_f$ that represents CNN weights or activation values, we search for the collections of $\{\mathbf{E}_i\}$ subsets

[2]Actually, the quantized weight values need to be additionally encoded into b-bitwidth UPoT format as listed in Eq 5. This is explained in next section.

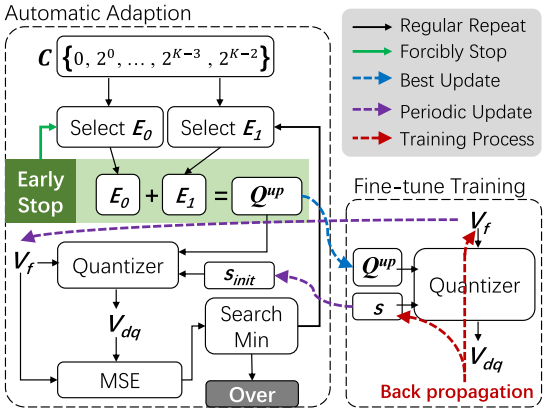

Fig. 6. Automatic adaption of UPoT quantization value set Q^{up}. During the fine-tune training process, periodic adaption is performed to update Q^{up} with the latest values of s and $\mathcal{V}_f$.

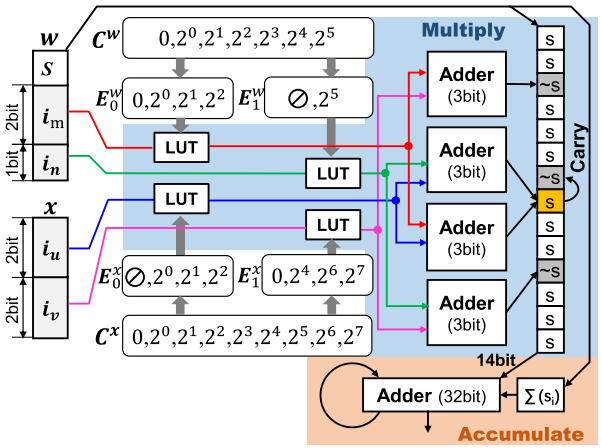

Fig. 7. Design of power-of-two multiply-and-accumulate (MAC) unit where multiply is fulfilled with several low-bit LUTs and adders. The accumulate unit adds the multiplier outputs with the sum of signs to generate the final result.

that minimize the *MSE* between $\mathcal{V}_f$ and $\mathcal{V}_{dq}$ as:

$$\{\mathbf{E}_i\}^{\star} = \arg\min_{\{\mathbf{E}_i\}} \Big(MSE(\mathcal{V}_f, \mathcal{V}_{dq})\Big),$$
$$\text{and } \mathcal{V}_{dq} = s \times quantize\Big(\mathcal{V}_f, Q^{up}\big(\{\mathbf{E}_i\}\big), s\Big). \quad (16)$$

where V_{dq} is the discrete quantized values of *Quantizer*, as defined in Equ 12. We use the original CNN model parameters to search the weight quantization levels of $\{\mathbf{E}_i^w\}$. On the other hand, to generate activation quantization levels of $\{\mathbf{E}_i^x\}$, we feed a batch of calibration dataset into the original CNN model and record the inference activation values as the estimation of $\mathcal{V}_f$. As each layer has different data distributions, we independently search different $\{\mathbf{E}_i^w\}$ and $\{\mathbf{E}_i^x\}$ subsets for each layer's weight and activation values. We still assume the quantization value is divided into two parts (i.e. n=2). Firstly, we set the scale of ***Candidate Set*** $\mathcal{C}$ as:

$$\mathcal{C} = \{0, 2^0, 2^1, \ldots, 2^{K-2}\}, \text{ and } K = 2^{b_0} + 2^{b_1} + 1 \quad (17)$$

which includes K elements with the maximal value 2^{K-2}. In total, there are $C_K^{2^{b_0}}$ possibilities for $\mathbf{E}_0$ and $C_K^{2^{b_1}}$ for $\mathbf{E}_1$. We try all possible combinations of $\{\mathbf{E}_0\} + \{\mathbf{E}_1\}$ to build different Q^{up} and find the best solution as Fig 6 shows. To reduce the search time, we propose *early-stop* mechanism that forcibly dumps unpromising solutions with rules: 1) Dump $\mathbf{E}_0$ and $\mathbf{E}_1$ that have more than two same elements in order to generate more elements for Q^{up} set; 2) Dump Q^{up} if all its elements are the multiple of any previous Q^{up} as their values are identical after scaling; 3) Dump Q^{up} if two adjacent values so close that $\frac{q_{i+1}-q_i}{q_i} < G_{min}$, as close values leads to the waste of quantization levels. Empirically we set $G_{min} = 0.02$.

For the initial value of scale factor s_{init}, we use the pre-trained model weights and calibration dataset to set it as:

$$s_{init} = \begin{cases} \dfrac{\sqrt{\mathbb{E}(\mathcal{V}_f^2)}}{Q_{max}}, & \text{if } \mathcal{V}_f > 0 \\ \dfrac{|clamp(\mathcal{V}_f, \mu - 3\delta, \mu + 3\delta)|_{max}}{Q_{max}}, & \text{else} \end{cases} \quad (18)$$

where μ and δ are the mean and standard deviation of V_f, and Q_{max} is the maximal value of Q^{up} for each search iteration. Meanwhile, because the values of s factors and weights keeps changing during training, we periodically use the latest s and weights from the training to re-adapt the Q^{up} so that the Q^{up} is guaranteed to match the latest distribution, as shown in Fig 6. Empirically, we update Q^{up} for every 30 training epochs.

IV. UPoT Accelerator Design

Theoretically, logarithmic quantization is friendly for hardware implementations as the multiplication of power-of-two values can be performed by simply adding their exponents. However, it is still an open and unclear problem about how to rationally design logarithmic quantization accelerator systems. In this section, we proposed our accelerator design of UPoT inference scheme that has high resource efficiency and computation performance with feasible integration with current mainstream accelerator architectures.

A. Multiply-and-Accumulate Unit

UPoT uses dedicated Multiply-and-Accumulate (MAC) unit to compute convolution between input feature map and weight values in UPoT quantization format. We assume the representation bitwidth is 4-bit. The signed weight values are quantized to be $w = \pm(2^{\epsilon_0^w} + 2^{\epsilon_1^w})$ being encoded into 4-bit which includes: 1-bit sign, 2-bit i_m to index $2^{\epsilon_0^w}$ in $\mathbf{E}_0^w$, and 1-bit i_n to index $2^{\epsilon_1^w}$ in $\mathbf{E}_1^w$. Similarly, the unsigned input feature map values are quantized as $x = (2^{\epsilon_0^x} + 2^{\epsilon_1^x})$ which is encoded into two 2-bit indices i_u and i_v. As depicted in Fig 7, UPoT uses local look-up tables (LUT) to store the exponents of $\mathbf{E}_i$ with an additional magic number (noted as $\oslash$) to represent the value of zero, i.e. $2^{\oslash}$=0. Each LUT corresponds to a specific $\mathbf{E}_i$ set. As UPoT provides optimized Q^{up} for each CNN model layer, the contents in each LUT should be updated with corresponding layer quantization levels. This is fulfilled by loading new exponent values from main memory at the beginning of each convolution layer.

To perform multiplication, MAC unit first loads and extracts the indices of $\{i_m, i_n, i_u, i_v\}$ from the encoded 4-bit weight and feature map data, and then uses them to find the exponents $\{\epsilon_0^w, \epsilon_1^w, \epsilon_0^x, \epsilon_1^x\}$ stored in LUTs, as depicted in Fig 7. Then

the product of $w \times x$ is calculated as:

$$\begin{aligned} y &= \pm(\mathbf{E}_0^w[i_m] + \mathbf{E}_0^w[i_n]) \times (\mathbf{E}_0^x[i_u] + \mathbf{E}_0^x[i_v]) \\ &= \pm(2^{\epsilon_0^w} + 2^{\epsilon_1^w}) \times (2^{\epsilon_0^x} + 2^{\epsilon_1^x}) \\ &= \pm(2^{\epsilon_0^w+\epsilon_0^x} + 2^{\epsilon_1^w+\epsilon_0^x} + 2^{\epsilon_0^w+\epsilon_1^x} + 2^{\epsilon_1^w+\epsilon_1^x}) \end{aligned} \tag{19}$$

Therefore, the product is sum of four power-of-two numbers, whose exponents can be generated with four low-bit adders. In case of 4-bit quantization, INT3 adders are adequate as the maximal exponent value in the $\mathcal{C}$ set is 7 according to Eq 17. In order to efficiently get y in hardware, we define an alternate representation of y using two's complement notation:

$$y = c(y) + sign(y) \tag{20}$$

$$c(y) = \begin{cases} |y| & \text{if } y \geq 0 \\ \sim (|y|) & \text{if } y < 0 \end{cases} \tag{21}$$

where *sign(y)* is 0 for positive value and 1 for negative, $|\cdot|$ is the absolute value, and $\sim(\cdot)$ is to invert every bit of a given data. Using this representation, the accumulation of products can be written as:

$$\sum_i y_i = \sum_i c(y_i) + \sum_i sign(y_i) \tag{22}$$

which means we can generate only the $c(y_i)$ by the multiplier to simplify the design, and add $\sum sign(y_i)$ later in the accumulation unit which is actually the total amount of negative y. This processing flow is depicted in Fig 7. Meanwhile, from Eq 21 we learn that $c(y)$ keeps y in original form if it is positive, and converts negative y by inverting each bit of its absolute value. So we can set each bit of $c(y)$ as:

$$\begin{aligned} \forall y[i] \in y, \quad c(y)[i] &= (|y|)[i] \oplus sign(y) \\ &= \begin{cases} \sim sign(y) & \text{if}(|y|)[i] = 1 \\ sign(y) & \text{if}(|y|)[i] = 0 \end{cases} \end{aligned} \tag{23}$$

As *sign(y)=sign(w)*, we can set *sign(w)* or $\sim$*sign(w)* to each bit according to the corresponding bit of $|y|$. The value of $|y|$ can be derived from Eq 19 as:

$$|y| = 2^{\epsilon_0^w+\epsilon_0^x} + 2^{\epsilon_1^w+\epsilon_0^x} + 2^{\epsilon_0^w+\epsilon_1^x} + 2^{\epsilon_1^w+\epsilon_1^x} \tag{24}$$

Therefore, we can simply set $\sim$*sign(w)* at bits pointed by the four exponents and assign *sign(w)* to the other bits, as shown in Fig 7. To handle the situation when multiple exponents are equal, we assume always $\epsilon_0^w \geq \epsilon_1^w$ and $\epsilon_0^x \geq \epsilon_1^x$, and forcibly turn $\{\epsilon_0^w, \epsilon_1^w\}$ into $\{\epsilon_0^w + 1, \oslash\}$ if $\epsilon_0^w = \epsilon_1^w$, because:

$$\begin{aligned} \text{if } \epsilon_0^w = \epsilon_1^w\text{: } |y| &= 2 \times 2^{\epsilon_0^w+\epsilon_0^x} + 2 \times 2^{\epsilon_0^w+\epsilon_1^x} \\ &= 2^{(\epsilon_0^w+1)+\epsilon_0^x} + 2^{\oslash} + 2^{(\epsilon_0^w+1)+\epsilon_1^x} + 2^{\oslash} \end{aligned} \tag{25}$$

Similarly, equal ϵ_0^x and ϵ_1^x are turned into $\{\epsilon_0^x + 1, \oslash\}$. In this way, we guarantee $\epsilon_0^w > \epsilon_1^w$ and $\epsilon_0^x > \epsilon_1^x$ so that there are at most two identical exponents in Eq 25.[3] In other words:

$$\epsilon_0^w + \epsilon_0^x > \epsilon_0^w + \epsilon_1^x > \epsilon_1^w + \epsilon_1^x \tag{26}$$

which indicates the only possible case of equal exponents is $\epsilon_0^w + \epsilon_1^x = \epsilon_1^w + \epsilon_0^x$. Therefore we can calculate the result using the following equation:

$$\begin{aligned} |y| = \big(1 \ll (\epsilon_0^x + \epsilon_0^w)\big) \big| \big(1 \ll (\epsilon_0^x + \epsilon_1^w)\big) \big| \big(1 \ll (\epsilon_1^x + \epsilon_1^w)\big) \\ + \big(1 \ll (\epsilon_1^x + \epsilon_0^w)\big) \end{aligned} \tag{27}$$

which is implemented with simple one-hot vector element-wise AND and ADD operations. The multiplier is followed by accumulate unit, where the generated $c(y_i)$ values are added in a standard integer (e.g. INT32) adder. According to Eq 22, the unit also counts the sign bits of weights and adds the sum at the end of accumulation as in Fig 7.

The proposed multiplier has good compatibility with different quantization bitwidth. Specifically, besides of 4/3/2-bit, it can also compute >4-bit quantization values. For example, assume a weight value is quantized into 6-bit with three 2-bit segments, i.e. $w = \pm(2^{\epsilon_0^w} + 2^{\epsilon_1^w} + 2^{\epsilon_2^w})$, its multiplication with 4-bit activation $x = (2^{\epsilon_0^x} + 2^{\epsilon_1^x})$ can be calculated as:

$$|y| = 2^{\epsilon_0^w+\epsilon_0^x} + 2^{\epsilon_0^w+\epsilon_1^x} + 2^{\epsilon_1^w+\epsilon_0^x} + 2^{\epsilon_1^w+\epsilon_1^x} + 2^{\epsilon_2^w+\epsilon_0^x} + 2^{\epsilon_2^w+\epsilon_1^x} \tag{28}$$

which equals to the addition of 6 power-of-two terms. This can be performed with the proposed multiplier in two cycles: input $\{(\epsilon_0^w, \epsilon_1^w), (\epsilon_0^x, \epsilon_1^x)\}$ on the first cycle, then input $\{(\epsilon_2^w), (\epsilon_0^x, \epsilon_1^x)\}$ on the second cycle. The similar process scheme can be used to process even higher quantization bitwidth such as 8-bit.

Besides, our design offers good compatibility to efficiently process other power-of-two formats. As mentioned at Section III-A, our multiplier unit can support APOT by configuring LUTs with fixed APOT exponents, and can support Log2 via bypassing LUTs and directly adding exponents.

B. Re-Quantization Pipeline

The result values of convolution are usually high-bit integers (INT32 or INT40), and then undergo regular biasing and activation. In order to compute them in the next layer, the accelerator uses re-quantization pipeline to convert the output feature map values into UPoT low-bit representation, e.g. 4-bit encoding. The pipeline fulfills Eq 15 with two units: the *Re-scale* to convert high-bit values to lower bits (8-bit in our design), and the *Encode* to generate the quantization format. In our design, the re-scale factor $\frac{s_x \times s_w}{s_y}$ is approximated as $\frac{\alpha}{2^\beta}$ so that the scaling can be performed with simple multiply-and-shift circuits to multiply α and right-shift β bits. As depicted in Fig 8, the final re-scaled number y_s is an 8-bit integer. Then y_s is quantized to UPoT format by finding the closest quantization level in Q^{up}. To fast find the result, we hold all values of $Q^{up} = \{q_0, q_1 \ldots\}$ and their 4-bit encoding, which are placed in the local buffer as ascending order, so that $q_j < q_{j+i}$. Fig 8 depicts the process of *Encode*. First the re-scaled INT8 number is compared with all Q^{up} values in parallel. By summing up the 1-bit comparison results, we can easily find the two closest values in Q^{up}: $q_j < y_s$ and $q_{j+1} > y_s$. Then these two values are compared with y_s to select the closest one, and the selected value's 4-bit encoding is then output as the final quantization result for y_s. As shown in Fig 8, at each cycle, the re-quantization pipeline takes a new activation value and re-quantizes it. Moreover, the pipeline has low hardware cost as the *Re-scale* and *Encode* units are optimized to minimize the number and complexity of hardware operations. It should

[3]If any one of the adder operands is $\oslash$ then the sum is $\oslash$, meaning a zero value. In other words, $\oslash + (\cdot) = \oslash$ and $2^{\oslash} = 0$.

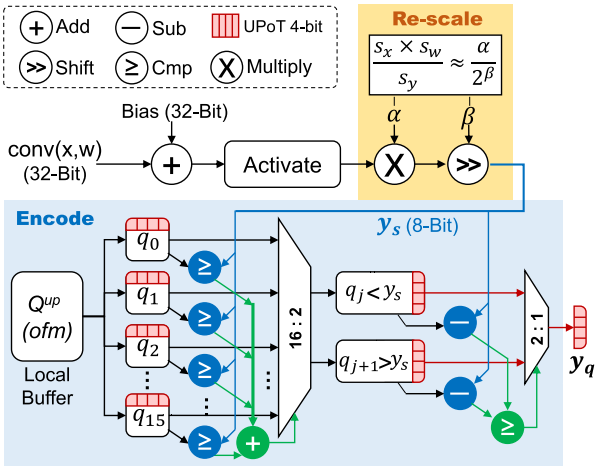

Fig. 8. Design of re-quantization pipeline to convert the generated high-bit output feature map values into 4-bit UPoT encoding using simple hardware components. The unit includes two parts: the *Re-scale* to transfer data into INT8 format, and *Encode* to select the closest value in Q^{up} set and output the corresponding 4-bit data.

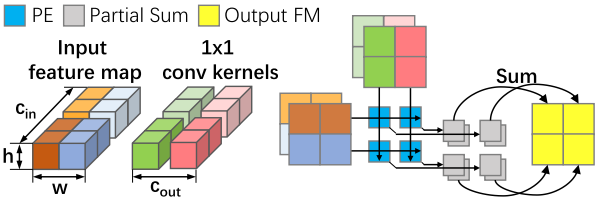

Fig. 9. Example of 1 × 1 convolution layer that is converted to the accumulation of several sub-matrix multiplications.

be noted that at the beginning of each layer, the values of Q^{up} in local buffer should be updated with corresponding Q^{up} set.

C. Overall Architecture

As convolution layer can be represented as matrix multiplication between model weights and input feature maps, many modern CNN accelerators use process engine (PE) array to efficiently perform matrix multiplication, such as the designs of NVDLA [6] and Eyeriss [7]. For convolution shapes that exceed PE array size, the computation is divided into sub-matrix multiplications to be directly mapped to hardware, while the accumulation of multiplication results generate the final result, as depicted in Fig 9. It should also be noted that the batch normalization can be fused into convolution layers and performed within the matrix multiplication process [27].

UPoT architecture applies PE array design, while the most critical computation modules, i.e. multiplier, accumulator and re-quantization, are specifically optimized to support UPoT formats. Fig 10 depicts the overall scheme, where 64 PEs are arranged in an 8 × 8 array. For each PE, total number of 16 power-of-two multipliers are placed to compute partial sum values in parallel. These results are summed up in the accumulate unit that uses a reduction tree to add 16 partial sums, and an extra adder is used to accumulate partial sums of multiple cycles. The extra bits of weight signs are also added, according to Eq 22. At the end of each PE computation, a final result is generated. Therefore, for each cycle, the PE array is able to compute multiplication between 8 × 16 and 16 × 8 matrices and generate 8 × 8 results. In case that convolution is divided into several sub-matrices, the calculation of output feature map

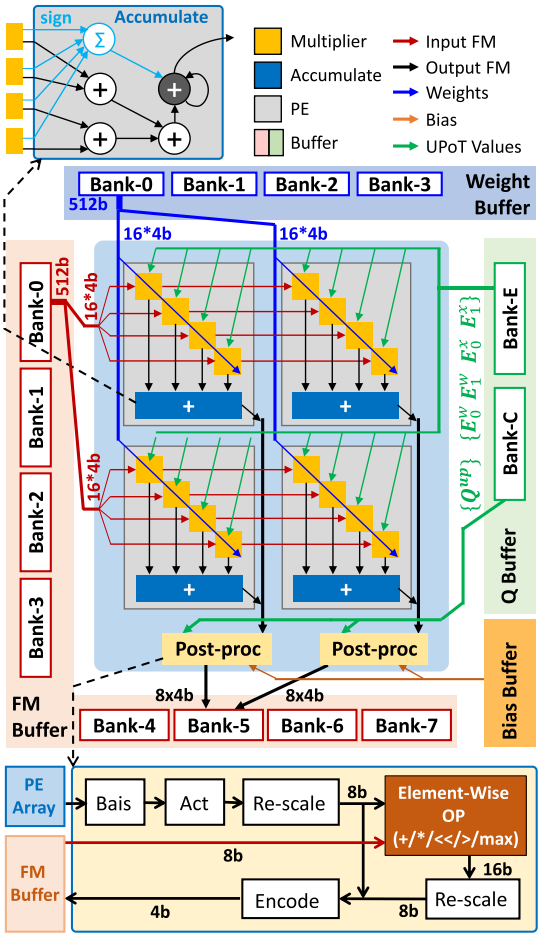

Fig. 10. Overall architecture of UPoT accelerator with 2-D 8 × 8 PE array. Each PE has 16 power-of-two multipliers and one accumulate unit. For each column of PEs, one post-process is placed, which include adding bias, activation, re-scale, element-wise operation(op) and encode modules. The element-wise operation module is responsible for element-wise/pooling. Dedicated local buffers are used to hold low-bit data, bias and UPoT quantization parameters.

value is performed inside each PE. As depicted in Fig 10, the PE array output values are sent into the post-processing unit which includes the bias-activate-encode pipeline as described in Fig 8. Besides, it also includes an element-wise operation module to support the element-wise layers and pooling layers in CNN models.

We use multi-bank buffers to store 4-bit weights and feature maps. Specifically, the weight buffer includes multiple banks and each bank has a 512-bit single port. At each cycle, the buffer offers a matrix of total 8 × 16 4-bit weights to the PE array, while every 16 values (as one matrix column) is broadcast to all 8 PEs in a column, as depicted in Fig 10. Similarly, the feature map buffer banks can provide a 16 × 8 matrix whose rows are broadcast to each PE row. The computation results are written back to another feature map bank. There are total 8 feature map banks and 8 weight banks in order to perform ping-pong in data prefetch. Additionally, bias data is stored in a single-bank bias buffer as depicted in Fig 10. We also provide a special buffer to store UPoT quantization parameters, which includes two banks to separately store every layer's exponents of $\{\mathbf{E}_i\}$ and Q^{up} values with associated 4-bit encoding, as shown in Fig 10. we use one buffer to

TABLE I

SUMMARY OF MAIN COMPUTATION RESOURCES USED IN UPOT DESIGN WITH 8 × 8 PE ARRAY

Module	Num	Compute Unit	Op Type	Num
Multiply	1024	Adder	INT3	4096
Accumulate	64	Counter	Bool	64
		Adder	INT14	512
		Adder	INT15	256
		Adder	INT16	128
		Adder	INT17	64
		Adder	INT32	64
Bias	8	Adder	INT32	16
Re-scale	8	Multiply	INT8	16
		Shifter	INT8	16
Element-Wise	8	Multiply	INT8	8
		Compare	INT8	8
		Adder	INT8	8
		Shifter	INT8	8
Encode	8	Compare	INT8	128
		Adder	Bool	8
		Sub	INT8	16

TABLE II

SUMMARY OF ON-CHIP BUFFER SCALES IN UPOT DESIGN WITH 8 × 8 PE ARRAY. ALL BUFFERS ARE SINGLE-PORT

Module	Name	Num	Width	Depth	Size(B)
Multiply	LUT($\mathbf{E}^x_{0,1}$)	2048	3b	4	3K
	LUT($\mathbf{E}^w_0$)	1024	4b	4	2K
	LUT($\mathbf{E}^w_1$)	1024	4b	2	1K
Re-quant	Q^{up} Buf	1	12b	16	24
WT Buf	WT Bank	8	512b	128	64K
FM Buf	FM Bank	8	512b	128	64K
Bias Buf	Bias Bank	1	64b	256	2K
Q Buf	Bank-E	1	3b	1024	387
	Bank-C	1	12b	1024	1536
Total					138K

store the Q^{up} values and encoding which is shared by all encode units. During the inference, at the beginning of each layer the corresponding parameters are read from the buffer to update the local buffers in multipliers and encode units.

UPoT supports coarse-grained program of CNN networks that allows user to claim the input sequence of desired CNN operation (e.g. convolution, depth-wise convolution, fully-connected, etc) with tensor shape and parameters (e.g. stride, dilation, padding, etc). The central control unit of UPoT parses these commands and schedules data load/store and computation. Therefore, our architecture supports the complete computation of entire CNN model inference.

V. HARDWARE EVALUATION

A. Implementation

Table I and Table II list the summary of computation units and on-chip memory used our design. The UPoT accelerator includes an 8×8 PE array while each PE has 16 multipliers and an accumulate tree. At each cycle, total number of 1024 power-of-two multiply-and-add operations can be fulfilled in pipeline, which is a typical scale in current CNN accelerators such as

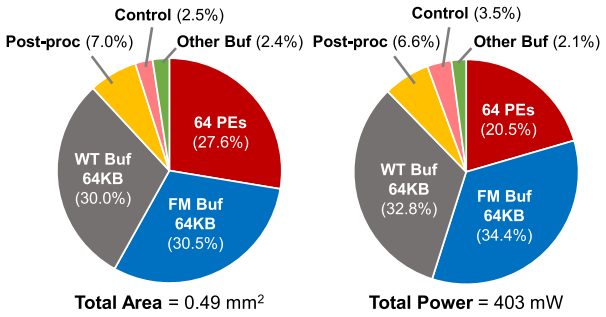

Fig. 11. Breakdown of UPoT accelerator's area and power.

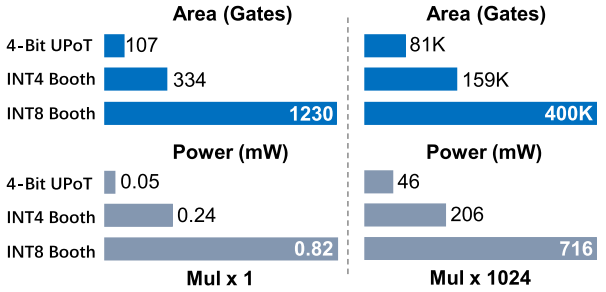

Fig. 12. Area (gates) and power (mW) comparison of 4-bit UPoT multiplier with Booth multipliers of INT4 and INT8. Different scales of ×1 and ×1024 multipliers are listed.

NVDLA [6]. Weight and feature map data are separately stored in two 8-bank buffers of 64KB size, which is smaller than mainstream on-chip SRAM size of 128–256KB used in most INT16/INT8 CNN accelerators [28] because UPoT uses 4-bit format to save the data size. Several small buffers are used to store bias and quantization parameters of Q^{up} and $\{\mathbf{E}_i\}$. The total size of on-chip SRAM is 138KB.

We design UPoT accelerator with SystemVerilog and implement it as ASIC circuit. To estimate its area and power consumption, we synthesis the design with Synopsys Design Compiler and Prime-Time PX tools using the TSMC 28nm library while the circuit frequency is set as 800MHz. Fig 11 depicts the detailed area proportions and power breakdown. We can observe that the total circuit area is 0.49 mm^2 and the power for computing ResNet50 model is 493 mW, while most circuit area (>88%) and energy cost (>87%) are contributed by weight/feature map buffers and PEs. By proposing low-bit quantization with optimized computation circuit, UPoT greatly reduces both area and energy costs to achieve high efficiency.

B. Power-of-Two Multiplier Evaluation

We compare the proposed power-of-two multiplier unit with traditional integer multipliers. Fig 12 lists the area and power overheads of a 4-bit UPoT multiplier. For comparison, we also evaluate standard INT4/INT8 Booth multiplier provided by the Design Compiler Designware IP Library [29]. All multipliers are synthesized with 800MHz TSMC 28nm technology with one-cycle latency. As depicted, a single UPoT multiplier is significantly smaller than others, with only 30% area and 20% power of one Booth INT4 multiplier. When compared with the INT8 Booth multiplier that is commonly used in current CNN accelerators, UPoT consumes only <8% area and <6% power. We also test the total area and power of 1024 multipliers in an 8 × 8×16 array, which is identical with architecture, and the results show that our UPoT multiplier circuit still have

TABLE III

COMPARISON WITH SEVERAL STATE-OF-THE-ART CNN ACCELERATORS INCLUDING NVDLA, EYERISS, PERMCNN AND MSQ IN TERMS OF DESIGN SCALES, HARDWARE OVERHEADS AND PERFORMANCE ON MAINSTREAM CNN MODELS INCLUDING RESNET50, VGG, MOBILENET_V2 AND EFFICIENTNET-B0. THE METRICS OF ENERGY EFFICIENCY (EE, FRAME/J) AND AREA EFFICIENCY (AE, FRAME/S/MM^2) ARE LISTED

Design	**UPoT**(ours)					**NVDLA**	**Eyeriss**		**PermCNN**	**MSQ**	
Weight Format	UPoT 4-bit					INT8	INT16		INT16	MSQ 4-bit	
Act Format	UPoT 4-bit					INT8	INT16		INT16	INT4	
MAC Number	1024					1024	168		128	900	
On-Chip Buffer(KB)	138					256†	181		288†	1024	
CMOS Technology	28nm					28nm	28nm‡		28nm	28nm∓	
Frequency (MHz)	800					1000	464‡		800	463∓	
Area (mm^2)	0.49					3.0	2.27‡		3.44	8.46∓	
CNN Model	Yolov3	Res50	VGG	Mobv2	EffB0	Res50	VGG	Mobv2	VGG	Yolov3	Mobv2
DRAM Access(MB)	54.5	25.2	16.4	8.6	139.0	50.5	107.0	16.5	33.9	-	-
Core Power(mW)	403.0	403.0	403.0	403.0	403.0	536.0	236.0	236.0	442.0	-	-
Total Power(mW)⋆	762.0	1236.6	612.9	2900.7	2628.4	1790.6	249.0	249.0	689.0	-	-
Processing Time(ms)	25.5	5.1	13.2	0.6	10.5	6.5	619.0	83.8	23.1	45.5	0.4
Throughput(frame/s)	39.2	196.6	76.0	1721.5	95.3	153.0	1.6	11.9	43.3	22.0	2664.1
Core EE(frame/J)	97.2	487.8	188.6	4271.8	236.5	285.5	6.9	575.2	98.0	-	-
Total EE(frame/J)	51.4	159.0	124.0	593.5	36.3	85.9	6.6	545.2	62.9	-	-
AE(frame/s/mm2)	80.8	405.3	156.7	3549.5	196.5	51.0	0.7	5.3	12.6	2.1	249.4

⋆ Core Power reports only on-chip circuit and buffer read/write energy, while Total Power also includes the DDR access.
† The PermCNN buffer size only includes feature map and weight buffers.
‡ Implementation details of 28nm Eyeriss is provided by [18], which was estimated based on 65nm Eyeriss implementation.
∓ Original implementation is on Xilinx XC7045 FPGA [22]. The 28nm ASIC area, frequency and throughput are estimated according to [30] and [31]. Power is not provided in the original paper and therefore not estimated for ASIC.

much lower overheads. It should be noted that though UPoT multiplier includes several LUTs, the hardware cost is quite low as LUTs are indexing circuits can be efficiently built using a few gates since our LUTs are only 4-depth or 2-depth scales. In the implementation of Design Compiler, the total LUTs and indexing circuits in each multiplier takes the area of <50 NAND gates. Furthermore, the simple calculation pattern in Eq 27 also reduces the circuit complexity.

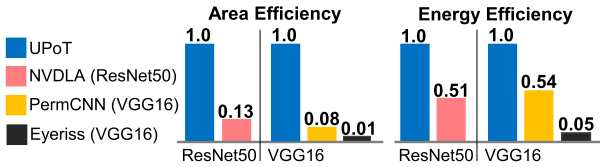

Fig. 13. Normalized comparison of area and energy efficiency for multiple CNN accelerators.

C. Overall Efficiency

To evaluate the overall efficiency, we use a cycle-accurate software model to simulate the performance of UPoT for classical VGG and ResNet50 models, as well as more advanced CNN models including MobileNet_v2 [3], EfficientNet-B0 [17] and YOLOv3 [32]. We compare with several representative CNN accelerators: Eyeriss [7] which uses the systolic array for matrix multiplication; NVDLA [6] which provides validated open-source design for FPGA and ASIC; PermCNN [18] which creates and leverages sparsity in CNN models. We also compare with a state-of-the-art logarithmic quantization accelerator MSQ [22], which uses 4-bit power-of-two to represent model weights. For each accelerator, we report its implementation details as well as its latency and power for CNN model inference. The circuit and performance results of the compared accelerators are from their original papers except the 28nm Eyeriss results that are obtained from the research of [18]. PermCNN also reports its overall power including both computation and off-chip DDR accesses. For NVDLA and UPoT, we follow the similar estimation method to summarize the amount of DDR data transfer including weight and feature maps, and calculate the energy as 21pJ/bit, which was typical for LPDDR3 model [33]. Meanwhile, in order to guarantee the model accuracy, low-bit quantization techniques usually keep the first and last layers in higher precision (e.g. 32-bit floating-point or INT8). In our approach, each model's first and last layer weights are quantized into UPoT 8-bit format, which are computed with the power-of-two multipliers following the scheme of Eq 28. It should be noted that as 8-bit layers only take up small proportions of the total computation (i.e. 0.5% of VGG16, 0.9% of ResNet50), the computation efficiency of these layers have limited impact on the overall performance.

Table III lists the comparison results of all the accelerators. We can observe that UPoT has very low circuit area which is much less than other accelerators as it optimizes computation units and reduces buffers while reaching equivalent and even better throughput. Meanwhile, UPoT achieves the lowest overall power on both tested CNN models due to the simplified circuit and reduced data transfer. Considering the PermCNN have reached high sparsity of ≈70%, this result indicates that low-bit quantization could be more effective than sparsity in reducing the system energy costs. Additionally, we introduce the metrics of energy and area efficiency in Table III, which can provide more fair and comprehensive evaluations. As depicted, UPoT achieves 1.7×–2.5× energy efficiency and 8.5×–9.2× area efficiency when being compared with NVDLA and PermCNN, respectively. When compared with Eyeriss, UPoT reaches 14× and 158× energy and area efficiency respectively, as Eyeriss requires more computation and buffer resources to process INT16 CNN models. More importantly, in comparison with the state-of-the-art logarithmic accelerator MSQ, UPoT achieves superior area efficiency with great margins, indicating our design is effective to fully exploit

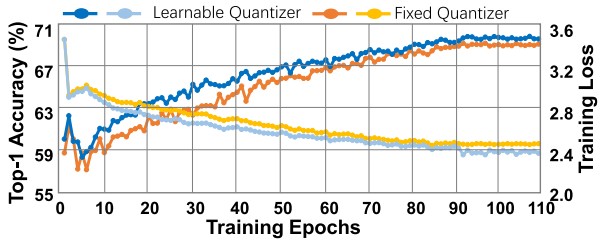

Fig. 14. Top-1 accuracy and training loss of MobileNet_V2 using fixed and learnable non-uniform quantizer respectively.

the hardware advantage of power-of-two formats. Note that the UPoT accelerator also supports 3-bit and 2-bit quantization, which will further improves the energy efficiency as the data transfer will be effectively reduced. Furthermore, when UPoT accelerator runs APOT and Log2, the energy efficiency would also increase because the LUTs are fixed or bypassed so that the energy of LUT reconfiguration is saved.

VI. Performance Evaluation

A. Experiment Settings

1) Dataset: We evaluate ImageNet (ILSVRC-2012), a standard public image classification benchmark. We also use a common data augmentation method to randomly crop and resize the training images to 224 × 224 pixels, and flip them with probability of 50%. The validation images are center-cropped to 224 × 224 pixels without any data augmentation.

2) Model Selection: We use several state-of-the-art CNN models for the image classification task, including ResNet18/50, MobileNet_v2 [3] and EfficientNet-B0 [17]. For reproducibility and fair evaluation, our baseline takes the official release of floating-point models provided by the public library TorchVison[4] without modifying the model structure.

3) Quantization Setting: We quantize all layers of the tested CNN models into UPoT format. We evaluate the performance of different quantization bit-width including 4/3/2-bit, except that the first and last layers are converted to 8-bit as mentioned in Section-V-C. For EfficientNet-B0, negative-padding [34] method is used to deal with *Swish* activation function.

4) Training Setting: The platform for training and inference is PyTorch version 1.51 atop Intel XEON 6146 CPU, 8 NVIDIA RTX8000 cards with CUDA 10.2. We use the SGD optimizer with initial learning rate of $1e^{-2}$, $2e^{-2}$, $4e^{-2}$ for ResNet18/50, MobileNet_V2 and EfficientNet-B0 respectively. We take the cosine annealing strategy [34] to schedule the learning rate and always scale the learning rate of step size by 0.1. We enable weight decay of $1e^{-5}$ only on the EfficientNet-B0 model. All models are trained for 90 epochs with the batch size of 512. For every 30 epochs, the latest weights and scale factors are used to update the Q^{up} set as shown in Fig 6.

B. Training Efficiency

UPoT can be efficiently trained due to the adaptive learnable quantizer which dynamically changes the scale factor and periodic updates the Q^{up} set. Fig 14 shows the learning curve of our method on 4-bit quantization and compare it with a fixed Q^{up} set and fixed scale factor that is frozen as $\frac{\partial V_{dq}}{\partial s}$=0. We can observe the proposed quantizer quickly surpasses the fixed quantizer after several warm-up epochs, and eventually achieves higher accuracy and lower loss within a faster convergence process. To further evaluate the training efficiency, we re-implement two state-of-the-art methods on PyTorch, including uniform quantization LSQ [2] and logarithmic quantization APOT [8]. All methods perform 4-bit quantization. We report their training time and memory footprint on our GPU server and top-1 accuracy on ResNet18 and MobileNet_V2 in Table IV. LSQ requires the least training time and memory costs as it uses integer formats that are easier to compute on GPU devices. Compared with LSQ, APOT requires much more training time and memory usage because its inefficient quantizer design. This is because APOT applies progressive quantization (5-bit→4-bit→3-bit→2-bit) technique to mitigate accuracy loss, but have to spend much more training time to obtain the final low-bit quantized models. Meanwhile, though APOT's accuracy is better than LSQ on ResNet18 due to there are more single-peak distributions, its performance on MobileNet_V2 is poor as the lightweight model has more irregular multi-peak distributions which are contrary to the assumption of APOT, as depicted in Fig 2. On the other hand, UPoT achieves the best accuracy on both models while its training time and memory usage are much less than APOT and are only slightly higher than LSQ. Therefore, UPoT can efficiently quantize CNN models to power-of-two formats with low hardware costs but higher accuracy.

TABLE IV

Training Time and GPU Memory Usage for Training 4-Bit Weight/Activation Quantization With LSQ, APOT and UPoT Methods Using 8 NVIDIA RTX8000 Cards. Batch Size Is 64/GPU. The Model's Top-1 Accuracy (%) After Training Is Listed for ResNet18 (RN) and MobileNet_V2 (MN)

Method	Epoch	Time/Ep	Total Time	Mem	RN (%)	MN (%)
LSQ	80	9.6 min	12.8 h	34GB	70.4	69.6
APOT	240	10.1 min	40.5 h	69GB	70.5	69.2
UPoT	90	10.2 min	15.5 h	38GB	71.1	70.3

C. Performance Evaluation

We compare UPoT with several state-of-the-art methods of 3 categories: the *logarithmic* with power-of-two format, the *uniform* with integer format and the *non-uniform* that uses customized format neither integer nor power-of-two. Results from their original papers are used. For APOT and LSQ works, we evaluate their performance with our re-implemented versions on settings which are not included in their papers. Since different baselines are used in these works, in order to fair evaluation, we also report each method's accuracy gain (+%) or loss (−%) compared to its own baseline.

Table V lists the Top-1 accuracy of ResNet18 model using 4/3/2-bit quantization for both weight and activation values. It can be observed that most of state-of-the-art methods can achieve accuracy gain at 4-bit setting and lose accuracy at 3-bit and 2-bit settings. Among all methods, UPoT achieves the best accuracy with the highest gain or lowest loss. In addition, we further evaluate ResNet50 performance in Table VI, where UPoT still achieves the best accuracy with the best results on

[4]https://pytorch.org/vision/stable/models.html

TABLE V

RESNET18 PERFORMANCE (TOP-1 ACCURACY) ON IMAGENET DATASET USING MULTIPLE QUANTIZATION METHODS. EACH METHOD IS COMPARED WHEN WEIGHT/ACTIVATION VALUES (W/A) ARE IN 4/3/2-BITWIDTH. FOR EACH SETTING, THE ACCURACY CHANGE AGAINST BASELINE (+/−) AND MODEL SIZE (MB) ARE LISTED. SPECIAL PRECISION SETTING REQUIRED IN EACH METHOD IS ALSO LISTED. THE (−) MARK DENOTES THE ABSENCE OF DATA IN ORIGINAL PAPERS

Method	Type	Special Precision	Baseline(%)	W/A=4-bit		W/A=3-bit		W/A=2-bit	
				Top-1(%)	Size	Top-1(%)	Size	Top-1(%)	Size
UPoT	*Logarithmic*	8-bit	70.2	**71.2 (+1.0)**	**5.9MB**	**70.2 (+0.0)**	**4.6MB**	**67.5 (-2.7)**	**3.2MB**
APOT [8]		INT8	70.2	70.7 (+0.5)	5.9MB	69.9 (-0.3)	4.6MB	67.3 (-2.9)	3.2MB
MSQ [22]		INT8	69.8	70.3 (+0.5)	5.9MB	-	-	-	-
LSQ [2]	*Uniform*	INT8	70.1	70.7 (+0.6)	5.9MB	69.4 (-0.7)	4.6MB	66.7 (-3.4)	3.2MB
LSQ+ [19]		INT8	70.1	70.8 (+0.7)	5.9MB	69.3 (-0.8)	4.6MB	66.8 (-3.3)	3.2MB
EWGS [21]		FP32	69.9	70.6 (+0.7)	7.6MB	69.7 (-0.2)	6.3MB	67.0 (-2.9)	4.9MB
DAQ [20]		FP32	69.9	70.5 (+0.6)	7.6MB	69.6 (-0.3)	6.3MB	66.9 (-3.0)	4.9MB
QIL [10]	*Non-Uniform*	FP32	70.2	70.1 (-0.1)	7.6MB	69.2 (-1.0)	6.3MB	65.7 (-4.5)	4.9MB
LQ-Net [25]		FP32	70.3	69.3 (-1.0)	7.6MB	68.2 (-2.1)	6.3MB	64.9 (-5.4)	4.9MB
DDQ [9]		Mixed	70.5	71.2 (+0.7)	5.8MB	-	-	-	-
DMBQ [24]		FP32+Mixed	70.3	-	-	70.0 (-0.3)	6.3MB	-	-

TABLE VI

RESNET50 PERFORMANCE (TOP-1 ACCURACY) ON IMAGENET DATASET USING MULTIPLE QUANTIZATION METHODS

Method	Type	Special Precision	Baseline(%)	W/A=4-bit		W/A=3-bit		W/A=2-bit	
				Top-1(%)	Size	Top-1(%)	Size	Top-1(%)	Size
UPoT	*Logarithmic*	8-bit	76.1	**76.8 (+0.7)**	**13.6MB**	**76.4 (+0.3)**	**10.8MB**	**74.2 (-1.9)**	**7.9MB**
APOT [8]		INT8	76.4	76.6 (+0.2)	13.6MB	75.8 (-0.6)	10.8MB	73.4 (-3.0)	7.9MB
LSQ [2]	*Uniform*	INT8	76.9	76.7 (-0.2)	13.6MB	75.8 (-1.1)	10.8MB	73.7 (-3.2)	7.9MB
PACT [35]		FP32	76.9	76.5 (-0.4)	19.4MB	75.3 (-1.6)	16.6MB	72.2 (-4.7)	13.8MB
DoReFa [36]		FP32	79.9	71.4 (-5.5)	19.4MB	69.9 (-7.0)	16.6MB	67.1 (-9.8)	13.8MB
LQ-Net [25]	*Non-Uniform*	FP32	76.4	75.1 (-1.3)	19.4MB	74.2 (-2.2)	16.6MB	71.5 (-4.9)	13.8MB

all settings of 4/3/2-bitwidth. Meanwhile, to maintain accuracy, most low-bit quantization methods use *special precision* for some critical layers, which are usually in higher precision than the low-bit formats. As shown in Table V and Table VI, EWGS [21], DAQ [20], QIL [10], LQ-Net [25], DMBQ [24], PACT [35] and DoReFa [36] don't quantize the first and last layers, so expensive floating-point computations are still used in their inference. This also inflates their model size, making their size almost 1.6×–1.8× larger than UPoT, which undermines their advantage in model compression. Moreover, DDQ [9] and DMBQ use mixed precision technique where each layer is quantized into different bitwidth, which adds to the hardware complexity as the accelerator must consider the worst-case quantization format. On the contrast, UPoT provides outstanding accuracy using only simple representation formats, indicating its efficiency in hardware acceleration.

Table VII lists the quantization results (Top-1 accuracy) for the MobileNet_V2 model [3], which is a compact lightweight model widely used in edge devices. Most 4-bit quantization results are from the original papers except QIL, APOT, PACT and LSQ whose results are from MQBench [37]. We also run experiments of 3-bit quantization on APOT and LSQ using our re-implemented versions, while 2-bit results are not reported as their accuracy degradation are unacceptable. As shown in Table VII, unlike ResNet18/50, MobileNet_V2 is more sensitive to quantization, resulting in accuracy degradation for all the tested methods at 4-bit setting. The reported accuracy losses are quite different, varying from -7.1% to -0.4%. Among them, DDQ [9] achieves the best performance on 4-bit setting as it is capable to represent more various distributions using its specialized formats with mixed bitwidth. However, its complex and mixed-bitwidth quantization scheme is hard to be deployed in hardware accelerator as it requires complicated computations. Meanwhile, UPoT has the second best accuracy in 4-bit setting, which is slightly lower than DDQ but outperforms QIL, APOT, DAQ, PACT, LSQ and DSQ by significant margins (0.3% - 5.5%). UPoT is also the best in 3-bit setting. Moreover, Table VIII lists the results of EfficientNet-B0 [17] which is a state-of-the-art lightweight CNN model generated by AutoML. UPoT achieves the best Top-1 accuracy for both 4/3-bit settings, and outperforms APOT and LSQ as they fail to match the irregular distributions. We also notice UPoT has slightly higher accuracy loss than LSQ+ [19], which enhances LSQ by using asymmetric quantization. However, such scheme adds its complexity of inference as each value has a dynamic

TABLE VII

MOBILENET_V2 PERFORMANCE (TOP-1 ACCURACY) ON IMAGENET DATASET USING MULTIPLE QUANTIZATION METHODS WHEN WEIGHT/ACTIVATION (W/A) ARE IN 4/3-BITWIDTH

W/A	Method	Type	SP	Base	Top-1(%)	Size (MB)
4-bit	**UPoT**	*Log-*	8-bit	71.9	**70.4 (-1.5)**	**2.4**
	APOT [37]	*Log-*	INT8	72.6	68.6 (-4.0)	2.4
	MSQ [22]	*Log-*	INT8	71.9	65.6 (-6.2)	2.4
	LSQ [37]	*Uni-*	INT8	71.9	69.9 (-2.0)	2.4
	EWGS [21]	*Uni-*	FP32	71.9	70.3 (-1.6)	2.4
	DAQ [20]	*Uni-*	FP32	71.9	70.0 (-1.9)	2.4
	PACT [37]	*Uni-*	FP32	72.6	70.7 (-1.9)	6.2
	DSQ [38]	*Uni-*	FP32	71.9	64.8 (-7.1)	6.2
	QIL [37]	*Non-*	FP32	72.6	70.3 (-2.3)	6.2
	DDQ [9]	*Non-*	Mixed	71.9	71.3 (-0.6)	2.4
3-bit	**UPoT**	*Log-*	8-bit	71.9	**66.3 (-5.6)**	**2.1**
	APOT‡	*Log-*	INT8	71.9	64.2 (-7.7)	2.1
	LSQ‡	*Uni-*	INT8	71.9	65.3 (-6.6)	2.1

‡ results from our re-implemented version.

TABLE VIII

EFFICIENTNET-B0 PERFORMANCE (TOP-1 ACCURACY) ON IMAGENET DATASET USING MULTIPLE QUANTIZATION METHODS WHEN WEIGHT/ACTIVATION (W/A) ARE IN 4/3-BITWIDTH

W/A	Method	Type	SP	Base	Top-1(%)	Size (MB)
4-bit	**UPoT**	*Log-*	8-bit	77.7	**74.8 (-2.9)**	**3.3**
	APOT‡	*Log-*	INT8	77.7	73.7 (-4.0)	3.3
	LSQ [19]	*Uni-*	INT8	76.1	71.9 (-4.2)	3.3
	LSQ+ [19]	*Uni-*	INT8	76.1	73.8 (-2.3)	3.3
3-bit	**UPoT**	*Log-*	8-bit	77.7	**70.4 (-7.3)**	**2.8**
	APOT‡	*Log-*	INT8	77.7	69.6 (-8.1)	2.8
	LSQ [19]	*Uni-*	INT8	76.1	67.5 (-8.5)	2.8
	LSQ+ [19]	*Uni-*	INT8	76.1	69.3 (-6.8)	2.8

‡ results from our re-implemented version.

TABLE IX

OBJECT DETECTION MODEL RETINANET USES 4/3/2-BIT UPOT QUANTIZATION AND INSTANCE SEGMENTATION MODEL SOLO USES 4-BIT UPOT QUANTIZATION ON COCO DATASET. METRICS OF PIXEL-LEVEL 50-TARGET PRECISION (AP_{50}) AND 100-TARGET RECALL (AR_{100}) ARE LISTED

	RetinaNet (Detection)				SOLO (Segmentation)	
Metric	FP32	W4A4	W3A3	W2A2	FP32	W4A4
AP_{50}	0.460	0.463	0.451	0.403	0.470	0.445
AR_{100}	0.471	0.470	0.460	0.425	0.416	0.400

zero point which needs to be specially handled in hardware. Besides, compared with LSQ+, UPoT's baseline is better trained, which makes it more sensitive to quantization effects and leads to higher accuracy loss.

Moreover, to test UPoT's generality, we run tasks of object detection and instance segmentation which are more difficult for quantization technique. For object detection, we use UPoT to quantize RetinaNet [39] in 4/3/2-bit formats. For segmentation, the SOLO [40] model is quantized in 4-bit formats. Both tasks use COCO dataset [41]. As shown Table IX, when using 4-bit quantizaion, UPoT can achieve higher accuracy than FP32 precision baseline on the object detection task, and only slightly lower accuracy on the instance segmentation task, even though the instance segmentation is usually quite sensitive to quantizaion errors. The extended evaluations prove the robustness of our approach in different CNN tasks.

VII. CONCLUSION

In the paper, we propose Universal Powers-of-Two (UPoT), a novel low-bit quantization method with accelerator design. UPoT adaptively builds the low-bit data to provide flexible representation values, which is especially useful for prevalent irregular distributions in compact models. Our method achieve outstanding accuracy on ImageNet dataset using several up-to-date CNN models and outperforms multiple state-of-the-art low-bit quantization methods. Moreover, it can significantly reduce hardware resources and power consumption compared with several state-of-the-art CNN accelerators.

REFERENCES

[1] Y. Gong et al., "Quality driven systematic approximation for binary-weight neural network deployment," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 69, no. 7, pp. 2928–2940, Jul. 2022.

[2] S. K. Esser, J. L. McKinstry, D. Bablani, R. Appuswamy, and D. S. Modha, "Learned step size quantization," 2019, *arXiv:1902.08153*.

[3] 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.

[4] M. Ahmadi, S. Vakili, and J. M. P. Langlois, "CARLA: A convolution accelerator with a reconfigurable and low-energy architecture," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 68, no. 8, pp. 3184–3196, Aug. 2021.

[5] X. Wu, Y. Ma, M. Wang, and Z. Wang, "A flexible and efficient FPGA accelerator for various large-scale and lightweight CNNs," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 69, no. 3, pp. 1185–1198, Mar. 2022.

[6] Nvidia. (2018). *Nvidia Deep Learning Accelerator*. [Online]. Available: http://nvdla.org/primer.htm

[7] Y. H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE J. Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, May 2017.

[8] Y. Li, X. Dong, and W. Wang, "Additive powers-of-two quantization: An efficient non-uniform discretization for neural networks," 2019, *arXiv:1909.13144*.

[9] Z. Zhang, W. Shao, J. Gu, X. Wang, and P. Luo, "Differentiable dynamic quantization with mixed precision and adaptive resolution," in *Proc. Int. Conf. Mach. Learn.*, 2021, pp. 12546–12556.

[10] S. Jung et al., "Learning to quantize deep networks by optimizing quantization intervals with task loss," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2019, pp. 4350–4359.

[11] K. Yamamoto, "Learnable companding quantization for accurate low-bit neural networks," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2021, pp. 5029–5038.

[12] B. Liu et al., "A 22 nm, 10.8μW/15.1μW dual computing modes high power-performance-area efficiency domained background noise aware keyword-spotting processor," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 67, no. 12, pp. 4733–4746, Dec. 2020.

[13] B. Zhao et al., "REMAP: A spatiotemporal CNN accelerator optimization methodology and toolkit thereof," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, early access, Sep. 16, 2022, doi: 10.1109/TCAD.2022.3207320.

[14] B. Liu et al., "More is less: Domain-specific speech recognition microprocessor using one-dimensional convolutional recurrent neural network," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 69, no. 4, pp. 1571–1582, Apr. 2022.

[15] T. Yuan, W. Liu, J. Han, and F. Lombardi, "High performance CNN accelerators based on hardware and algorithm co-optimization," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 68, no. 1, pp. 250–263, Jan. 2021.

[16] D. Miyashita, E. H. Lee, and B. Murmann, "Convolutional neural networks using logarithmic data representation," 2016, *arXiv:1603.01025*.

[17] M. Tan and Q. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 6105–6114.

[18] C. Deng, S. Liao, and B. Yuan, "PermCNN: Energy-efficient convolutional neural network hardware architecture with permuted diagonal structure," *IEEE Trans. Comput.*, vol. 70, no. 2, pp. 163–173, Feb. 2021.

[19] Y. Bhalgat, J. Lee, M. Nagel, T. Blankevoort, and N. Kwak, "LSQ+: Improving low-bit quantization through learnable offsets and better initialization," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW)*, Jun. 2020, pp. 696–697.

[20] D. Kim, J. Lee, and B. Ham, "Distance-aware quantization," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2021, pp. 5271–5280.

[21] J. Lee, D. Kim, and B. Ham, "Network quantization with element-wise gradient scaling," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2021, pp. 6448–6457.

[22] S.-E. Chang et al., "Mix and match: A novel FPGA-centric deep neural network quantization framework," in *Proc. IEEE Int. Symp. High-Performance Comput. Archit. (HPCA)*, Feb. 2021, pp. 208–220.

[23] M. S. Ansari, B. F. Cockburn, and J. Han, "An improved logarithmic multiplier for energy-efficient neural computing," *IEEE Trans. Comput.*, vol. 70, no. 4, pp. 614–625, Apr. 2021.

[24] S. Zhao, T. Yue, and X. Hu, "Distribution-aware adaptive multi-bit quantization," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2021, pp. 9281–9290.

[25] D. Zhang, J. Yang, D. Ye, and G. Hua, "LQ-Nets: Learned quantization for highly accurate and compact deep neural networks," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2018, pp. 365–382.

[26] Z. Song et al., "DRQ: Dynamic region-based quantization for deep neural network acceleration," in *Proc. ACM/IEEE 47th Annu. Int. Symp. Comput. Archit. (ISCA)*, May 2020, pp. 1010–1021.
[27] Pytorch. (2019). *Batch Normalization Fusion*. [Online]. Available: https://learnml.today/speeding-up-model-with-fusing-batch-normalization-and-convolution-3
[28] D. Moolchandani, A. Kumar, and S. R. Sarangi, "Accelerating CNN inference on ASICs: A survey," *J. Syst. Archit.*, vol. 113, Feb. 2021, Art. no. 101887.
[29] *Designware IP Family Reference Guide of Synopsys*, Synopsys, Mountain View, CA, USA, 2005.
[30] A. Boutros, S. Yazdanshenas, and V. Betz, "You cannot improve what you do not measure: FPGA vs. ASIC efficiency gaps for convolutional neural network inference," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 11, no. 3, pp. 1–23, Sep. 2018.
[31] L. Shannon, V. Cojocaru, C. N. Dao, and P. H. W. Leong, "Technology scaling in FPGAs: Trends in applications and architectures," in *Proc. IEEE 23rd Annu. Int. Symp. Field-Program. Custom Comput. Mach.*, May 2015, pp. 1–8.
[32] J. Redmon and A. Farhadi, "YOLOv3: An incremental improvement," 2018, *arXiv:1804.02767*.
[33] M. Schaffner, F. K. Gürkaynak, A. Smolic, and L. Benini, "DRAM or no-DRAM? Exploring linear solver architectures for image domain warping in 28 nm CMOS," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, 2015, pp. 707–712.
[34] E. Park and S. Yoo, "PROFIT: A novel training method for sub-4-bit MobileNet models," in *Proc. Eur. Conf. Comput. Vis.* Berlin, Germany: Springer, 2020, pp. 430–446.
[35] J. Choi, Z. Wang, S. Venkataramani, P. I.-J. Chuang, V. Srinivasan, and K. Gopalakrishnan, "PACT: Parameterized clipping activation for quantized neural networks," 2018, *arXiv:1805.06085*.
[36] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, "DoReFa-Net: Training low bitwidth convolutional neural networks with low bitwidth gradients," 2016, *arXiv:1606.06160*.
[37] Y. Li et al., "MQBench: Towards reproducible and deployable model quantization benchmark," 2021, *arXiv:2111.03759*.
[38] R. Gong et al., "Differentiable soft quantization: Bridging full-precision and low-bit neural networks," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, Oct. 2019, pp. 4852–4861.
[39] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 2980–2988.
[40] X. Wang, T. Kong, C. Shen, Y. Jiang, and L. Li, "SOLO: Segmenting objects by locations," in *Proc. Eur. Conf. Comput. Vis.* Berlin, Germany: Springer, 2020, pp. 649–665.
[41] T.-Y. Lin et al., "Microsoft COCO: Common objects in context," in *Proc. Eur. Conf. Comput. Vis.* Berlin, Germany: Springer, 2014, pp. 740–755.

Tian Xia (Member, IEEE) received the Ph.D. degree from the National Institute of Applied Sciences (INSA), France, in 2016. He has been an Assistant Professor with the College of Artificial Intelligence, Xi'an Jiaotong University, since 2019. His current research domains include the cloud-computing virtualization, innovative parallel computation architecture, and reinforcement learning algorithms and applications.

Boran Zhao received the bachelor's degree in Internet of Things from Jiangnan University in 2015 and the master's degree in electronic and communication engineering from Xidian University in 2018. He is currently pursuing the Ph.D. degree with the College of Artificial Intelligence, Xi'an Jiaotong University. His current research interests include DNN accelerator for autonomous agents.

Jian Ma is currently pursuing the degree in artificial intelligence with Xi'an Jiaotong University. He participates in research on DNN accelerator designing with the College of Artificial Intelligence, Xi'an Jiaotong University. His research interests include DNN model quantization.

Gelin Fu received the bachelor's degree in control science and engineering from Chongqing University, Chongqing, China, in 2019. He is currently pursuing the Ph.D. degree with the College of Artificial Intelligence, Xi'an Jiaotong University, Xian, China. His research interests include system modeling and optimization and computer architecture.

Wenzhe Zhao received the bachelor's, master's, and Ph.D. degrees from Xi'an Jiaotong University in 2005, 2008, and 2014, respectively. He is currently an Assistant Professor with the College of Artificial Intelligence, Xi'an Jiaotong University. During 2011 to 2013, he was a Visiting Student with the Rensselaer Polytechnic Institute (RPI). His current research directions are computer architecture, DNN-based computing architecture, and VLSI design.

Nanning Zheng (Fellow, IEEE) received the Graduate degree from the Department of Electrical Engineering in 1975, the M.E. degree in information and control engineering from Xi'an Jiaotong University in 1981, and the Ph.D. degree in electrical engineering from Keio University in 1985. He is currently a Professor and the Director of the Institute of Artificial Intelligence and Robotics, Xi'an Jiaotong University. He became a member of the Chinese Academy Engineering in 1999.

Pengju Ren (Member, IEEE) received the bachelor's and Ph.D. degrees from Xi'an Jiaotong University in 2004 and 2012, respectively. He is currently a Professor with the College of Artificial Intelligence, Xi'an Jiaotong University. During 2009 to 2011, he was a Visiting Ph.D. Student with the Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology (MIT). His current research interests include on-chip networks and computer architecture.