Non-Uniform Step Size Quantization for Accurate Post-Training Quantization

Sangyun Oh[1], Hyeonuk Sim[2], Jounghyun Kim[3], and Jongeun Lee[1,3†]

[1] Department of Electrical Engineering, UNIST, Ulsan, Korea
[2] Department of Computer Science and Engineering, UNIST, Ulsan, Korea
[3] Artificial Intelligence Graduate School, UNIST, Ulsan, Korea
{syoh, detective, maxedset, jlee}@unist.ac.kr

Abstract. Quantization is a very effective optimization technique to reduce hardware cost and memory footprint of deep neural network (DNN) accelerators. In particular, post-training quantization (PTQ) is often preferred as it does not require a full dataset or costly retraining. However, performance of PTQ lags significantly behind that of quantization-aware training especially for low-precision networks (<= 4-bit). In this paper we propose a novel PTQ scheme[1] to bridge the gap, with minimal impact on hardware cost. The main idea of our scheme is to increase arithmetic precision while retaining the same representational precision. The excess arithmetic precision enables us to better match the input data distribution while also presenting a new optimization problem, to which we propose a novel search-based solution. Our scheme is based on logarithmic-scale quantization, which can help reduce hardware cost through the use of shifters instead of multipliers. Our evaluation results using various DNN models on challenging computer vision tasks (image classification, object detection, semantic segmentation) show superior accuracy compared with the state-of-the-art PTQ methods at various low-bit precisions.

Keywords: Deep neural networks, Logarithmic-scale quantization, Post-training quantization, Subset quantization

1 Introduction

As deep learning becomes the highest performing method for many machine learning tasks, there is a growing interest in hardware DNN (Deep Neural Network) accelerators. DNNs of computer vision tasks such as image enhancement and super-resolution applications [2,20] often have very high compute and memory requirement. To reduce the hardware cost and memory footprint of such DNN accelerators, quantization can be a very effective approach [13,14,6]. In particular, post-training quantization (PTQ) is preferred for DNN deployment as it requires no costly retraining or large dataset. However, the performance of

† Corresponding author.
[1] Code will be publicly available at https://github.com/sogh5/SubsetQ

PTQ lags behind that of quantization-aware training especially for low-precision networks (e.g., < 4-bit).

In this paper we propose a novel PTQ scheme to bridge the gap, with minimal impact on the cost of a DNN hardware accelerator. Our scheme is based on logarithmic-scale quantization [16,17,18], which can help minimize hardware cost through the use of shifters instead of multipliers. The main idea of our scheme is to use an arithmetic precision that is higher than the representational precision. In linear (i.e., uniform step size) quantization, the two are the same. For instance, 3-bit quantized neural network would use 3-bit multipliers. However, for logarithmic-scale quantization [16], they are not always the same. For instance, 3-bit log-quantized data can have the same arithmetic precision as 4-bit linear-quantized data. Moreover, recent advanced logarithmic-scale quantization methods [17,8,18,25] often represent input data as a sum of two log-quantized words, which further disconnects arithmetic precision from representational precision.

Thus there is a new optimization problem: given a representational precision (P_r) and an arithmetic precision (P_a) where $P_a > P_r$, how to design a quantizer function such that it can maximize quantization performance for a given data distribution. Quantization performance can be defined in terms of quantization error or inference accuracy of a quantized DNN. We study this problem in the context of two-word log-scale quantization [17,8,18,25], for which we find the set of all possible *quantization points*, with each subset of the quantization points defining a new quantizer function. In other words, a quantizer may be specified as a *subset* of quantization points instead of an arithmetic function, and this extended view leads to a new category of quantizers that can make a more efficient use of the limited arithmetic precision than is possible otherwise. We call our quantization scheme *subset quantization*, which is more formally defined in Defining Quantizer for Subset Quantization Section.

In this paper we make the following contributions.

- We propose subset quantization (SQ), a novel quantization scheme for PTQ.
- We present a method to find the best subset and scale factor for subset quantization, for a given data distribution.
- We evaluate our method on image classification using ResNet models showing that our method outperforms state-of-the-art PTQ methods in most cases.
- We evaluate our method on another vision tasks such as object detection and semantic segmentation, demonstrating our method can achieve close to FP32 accuracy at ultra-low-bit weight precision.

2 Related Work

2.1 Uniform vs. Non-Uniform Quantization

A quantizer is defined as a function from the set of real numbers $\mathcal{R}$ to the set of integers. But in the context of DNN quantization, we are often interested in a *simulated quantizer*, which is a function that simulates the effect of quantization by applying a quantizer and a de-quantizer in a row [15]:

$$Q : \mathcal{R} \to \mathcal{R} = dequantizer \circ quantizer. \tag{1}$$

where $\circ$ represents function composition, and the input and output values have the same domain and the same scale. In this case, the essential role of a quantizer is to approximate a real value to one of a finite set of values, which we call *quantization points*.[2]

Linear quantization [13,14,6,9] uses quantization points that are spaced linearly. An example linear quantizer is:

$$q = Q_u(x) = \text{clip}(\left\lfloor \frac{x}{s} \right\rceil, L, U). \tag{2}$$

where s is step size, L, U are the lower and upper limits of the quantized values, and $\text{clip}(x, a, b) = \min(\max(x, a), b)$. In the above quantizer, the step size, or the distance from one quantization point to the next, is constant for all quantization points; hence, it is *uniform step size* quantization. By using clip we can focus on the more interesting range of input, i.e., $[sL, sU]$, which can result in a more efficient allocation of the limited number of quantization points as in ACIQ-Mix [1].

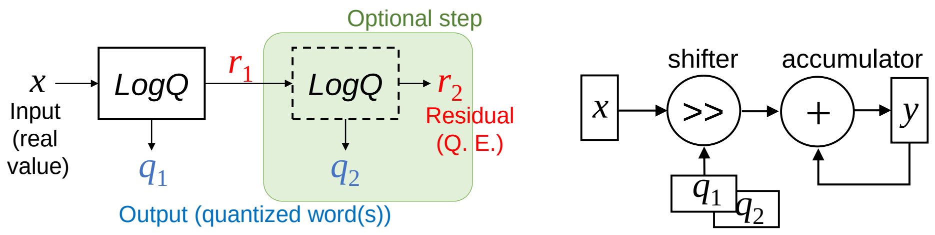


Fig. 1: Non-uniform step-size quantization example: selective two-word log-scale quantization [17,8,25]. Its processing element hardware (right) uses shifters instead of multipliers as in log-scale quantization.

There are a few quantization schemes that have non-uniform step size. Logarithmic-scale (*log-scale* for short) quantization [16] is an example, in which quantization points are spaced geometrically. Usually the base of exponent is set to 2, which enables very efficient arithmetic hardware using shifters instead of costly multipliers. Log-scale quantization is also motivated by the bell-shaped distribution of weight/activation values [16,18], which can, in certain cases, lead

[2] Quantization points are similar to quantization levels but there are some differences. Whereas quantization levels are often integers and may have a different scale than quantization thresholds, quantization points have the same scale as quantization thresholds and can be used as a substitute for them.

to lower expected quantization error than in linear quantization, since the former allocates more quantization points for common, low-magnitude values.

On the other hand, log-scale quantization may allocate too many quantization points near zero, which is especially true at high precision. Selective two-word log-scale quantization (STLQ) schemes [17,8,25] address this problem by employing another round of quantization step for those values that have high quantization error in the first round. In other words, each input value may be quantized with one or two quantized words, depending on the residual in the first round (see Figure 1). APoT [18] is another solution, which is based on a similar idea but always uses two shifters. The APoT quantizer maps an input value to the sum of two terms, each of which is a power-of-two. Our subset quantization, which may be seen as an evolution of log-based quantization schemes, further extends the set of quantization points while not affecting representational precision.

2.2 Determining Quantization Parameters

The main problem of quantization is that of determining quantization parameters such as step size s and lower and upper bounds L, U in the case of the linear quantizer in (2). Quantization parameters may be determined through training or other means (e.g., statistics). State-of-the-art performances in DNN quantization are often obtained by determining quantization parameters through training. This kind of training usually involves training of both weight and quantization parameters, hence called *joint-training*. However, even if quantization parameters are not trained, quantization-aware training, which trains weight parameters only (in this case, quantization parameters may be fixed or determined by other means, e.g., by statistics) can still outperform post-training quantization [15]. Post-training quantization (PTQ) is any method that determines quantization parameters for a given set of (fixed) weight parameters. Despite its lower performance, PTQ is often preferred for deployment as it does not require retraining or a full dataset.

2.3 Post-Training Quantization Methods

Here we briefly review recent PTQ methods. For image classification tasks, recent state-of-the-art PTQ methods have shown accuracy close to that of FP32 (32-bit floating-point precision) at low bits such as 4-bit for both weight and activation [28,23,12,10,19].

BitSplit [28] splits the n-bit binary sequence (e.g., power-of-two value of weight including sign bit) into $n-1$ ternary bits, then minimizes quantization error via calibration set and alpha scaling factor. It shows highly competitive results at various precision settings including 3-bit. AdaRound [23] subdivides the round operation of quantization process into floor and ceiling, and finds the favorable selection in terms of quantization error. It proposes an approximated per-layer quadratic task loss for the relaxation of selections, and shows result close to FP32 at 4-bit precision. AdaQuant [12] proposes a joint optimization method for weight and activation including layer-wise integer programming that searches bit combinations, followed by parameter tuning for batch normalization and bias. It shows performance close to FP32 at 4-bit for image classification.

PWLQ [10] divides bell-shaped distribution into two symmetric regions by using a breakpoint, and equally divides quantization points into each region, so that step size can be determined according to data density. It can search multiple breakpoints (up to 3) and shows close to FP32 performance at 4-bit precision. BRECQ [19] minimizes quantization error in various granularities such as model, stage, block, and layer. It uses a sigmoid-like trainable parameter for the round operation, and searches bit combinations using the genetic algorithm.

3 Our Proposed Method

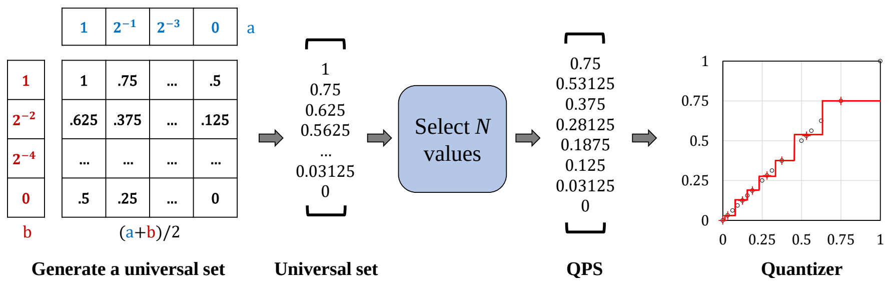


Fig. 2: Overview of our proposed quantization scheme ($N = 8$, 4-bit quantization including the sign bit). In the quantizer graph, black circles and red plus signs represent a universal set and a quantization point set (QPS), respectively.

3.1 Overview

Figure 2 illustrates our proposed quantization scheme. First we design a universal set S_U such that the set is rich enough to represent any given input distribution but each element in the set can be efficiently generated by hardware. We present one such set in Section 3.4. Due to the encoding restriction (i.e., limited weight precision) we cannot use all the elements in S_U at least simultaneously. Instead we select N values, where N is determined by the quantization precision (i.e., representational precision).

For n-bit quantization (including the sign bit), we select $N = 2^{n-1}$ values on the non-negative side, which may or may not include zero. (If zero is included, it means that we are wasting one code word, since the negation of zero is also zero, but we see that this rarely happens.) For instance, if $n = 4$, then $N = 8$, and if $n = 3$, $N = 4$. There are many ways to choose N elements out of S_U, and this flexibility affords us a degree of freedom by which we can better approximate any input data distribution.

Once we select the set of N points, which we call *quantization point set* (QPS), the negative values are defined simply as the negation of the non-negative values. The symmetricity in the quantizer reduces QPS exploration time (see Optimization for Subset Quantization Section) as well as hardware complexity. Finally, from the chosen QPS directly follows a quantizer/dequantizer definition, in which we include a scale factor α, which plays a similar role as in previous work [18,28]. Note that scale factors can be implemented as part of the succeeding layer, where it is combined with input quantization computation, thus having negligible cost [13,30]. We next present a generalized quantizer framework based on QPS, followed by our quantization parameter determination algorithm and universal set design.

3.2 Defining Quantizer for Subset Quantization

Quantization Point Set: Quantization point set (QPS) of a quantizer is the *range* of its simulated quantizer function. Since simulated quantizer [15] is a function that applies a quantizer and a de-quantizer in a row (see (1)), its input domain has the same scale as the output domain. Therefore, QPS is also the set of points in the input domain that can be quantized with no quantization error.

Then one can view all quantization schemes as approximating an input to the nearest[3] element in a QPS, with the only difference among different quantization schemes being the definition of QPS. Accordingly, we can write a fairly general quantizer definition that takes as parameter a QPS (S_Q) in addition to an input value (x). Its simulated quantizer is as follows:

$$Q(x, S_Q) = \arg\min_{p \in S_Q} |x - p|. \tag{3}$$

Examples: The QPS for linear quantization such as (2) can be defined as follows, where α is a step size and $2N$ is the number of quantization points; that is, for k-bit quantization, $N = 2^{k-1}$. For brevity we assume that input is symmetric around zero.

$$S_Q^{lin}(\alpha) = \{\alpha \cdot i \,|\, i = -N, -N+1, \cdots, N-1\}. \tag{4}$$

Note that the above is the QPS of (2) if we set $\alpha = s$, $L = -sN$, $U = s(N-1)$. Similarly, the QPS for log-scale quantization can be defined as follows, where α is a scale factor.

$$\begin{aligned} S_Q^{log}(\alpha) = \{-\alpha\, 2^{-i} \,|\, i = 0, 1, \cdots, N-1\} \cup \{0\} \\ \cup \{\alpha\, 2^{-i} \,|\, i = 0, 1, \cdots, N-2\}. \end{aligned} \tag{5}$$

[3] One may design a quantizer to output a non-nearest element, which is suboptimal but may be motivated by computational efficiency. An example is log-scale quantization, which was defined [16] as doing a round operation in the logarithmic domain, which is not necessarily the nearest one in the linear domain.

One way to enhance the accuracy of log-scale quantization is to use two words [17,18], the QPS for which can be defined as follows.

$$S_Q^{2log}(\alpha) = \{q_1 + q_2 \,|\, q_1 \in S_Q^{log}(\alpha),\, q_2 \in S_Q^{log}(\alpha)\}. \tag{6}$$

Subset Quantization: The idea of subset quantization (SQ) is to define S_Q not as a fixed set of numbers or variables but as an arbitrary subset (with a size limit of $2N$) of a larger set. The larger set is called *universal set*, denoted by S_U.

$$S_Q^{sq}(\alpha) = \{\text{any } q_i \in S_U(\alpha) \,|\, i = 1, \cdots, 2N\}. \tag{7}$$

which says S_Q^{sq} is any subset of S_U with $2N$ or fewer elements. Note that quantization precision restricts only the size of a QPS, but not that of a universal set. While (7) and (3) define the simulated quantizer for SQ, the definition is undeterministic. Exactly which subset to choose is left for optimization (see next section), much like determining the value of α. A quantization scheme may choose a different subset for each layer (per-layer quantization) or for each channel (per-channel quantization).

While any set may be used as the universal set, in this paper we use one that is similar to the QPS of the two-word log-scale quantization scheme as our universal set (i.e., S_Q^{2log}) due to its low hardware complexity and rich expressiveness (see Section 3.4).

3.3 Optimization for Subset Quantization

Minimizing Quantization Error Optimization for SQ needs to determine (i) the value of α and (ii) which subset of S_U to choose as QPS. For PTQ we are given all weight values ($\{w_i\}$), and the objective is to minimize L2 quantization error.

Finding Optimal Scale Factor Given a QPS, the optimal scale factor can be found efficiently. Let $\{q_j\}$ be the given QPS (before applying a scale factor). After applying scale factor α, the final QPS can be written as $S_Q = \{\alpha q_j\}$. To calculate quantization loss between $\{w_i\}$ and S_Q, let αq_i be the nearest quantization point in S_Q for w_i. Then,

$$\forall i, \quad \alpha q_i = \arg\min_{p \in S_Q} |p - w_i|. \tag{8}$$

$$\mathcal{L} = \sum_i (w_i - \alpha q_i)^2. \tag{9}$$

To find α that minimizes $\mathcal{L}$, we set the derivative of $\mathcal{L}$ w.r.t. α to zero, which gives

$$\alpha^* = \frac{\sum_i w_i \cdot q_i}{\sum_i {q_i}^2}. \tag{10}$$

Since finding αq_i (the nearest quantization point for w_i) in (8) depends on the current value of α, the above procedure (8)$\sim$(10) is repeated until α converges. We initialize α to 1. We find empirically that this iterative method converges usually fast, at the average within 17 iterations (when tolerance for α is 1e-5).

Proof of Convergence To find the optimal scale factor α for a given quantization point set $S(\alpha) = \{\alpha q_j\}$ that minimizes L2 quantization error, we use this iterative procedure, as explained in previous section.

- First, initialize α to 1.
- Second, update α using the (10) until α does not change any more.

Note that q_i on the right hand side of the (10) is a short-hand notation of a function $q(w_i, \alpha)$ defined below:

$$q_i := q(w_i, \alpha) = \frac{1}{\alpha} \cdot \arg\min_{p \in S(\alpha)} |p - w_i|. \tag{11}$$

We argue that updating α using (10) converges. Recall (9) which is the L2 quantization error. When α is updated according to (10), $q_i = q(w_i, \alpha)$ may or may not be updated. If no q_i is updated (i.e., $\forall i,\ q(w_i, \alpha) = q(w_i, \alpha^*)$), then (9) is a simple 2nd order function on α, and the optimal value of α can be found in one step by (10), after which α remains unchanged.

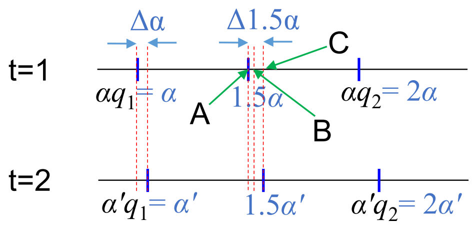


Fig. 3: The nearest quantization point may change when α is increased by $\Delta\alpha = \alpha' - \alpha$. In this example, $q_1 = 1$, $q_2 = 2$.

If some q_i are updated (i.e., $\exists i,\ q(w_i, \alpha) \neq q(w_i, \alpha^*)$), it leads to either (a) a decreased or the same value of $\mathcal{L}$ or (b) temporarily increased value of $\mathcal{L}$ but afterwards $\mathcal{L}$ is reduced. To see this, let us consider an example illustrated in Figure 3. In this example, we only consider one weight value w, which is quantized in the first iteration (t=1) using $S(\alpha) = \{\alpha q_1, \alpha q_2\}$ as the quantization point set. The first iteration updates α to α' (assume $\alpha' > \alpha$). Then in the second iteration (t=2), w is quantized using $S(\alpha') = \{\alpha' q_1, \alpha' q_2\}$. To further simplify, let us assume that the two quantization points q_1, q_2 are given as follows: $q_1 = 1, q_2 = 2$. In the figure, $A = 1.5\alpha$, $C = 1.5\alpha'$, and $B = A + 0.5\Delta\alpha$, where $\Delta\alpha = \alpha' - \alpha$. There are three cases as follow.

Case 1: All the points between α' and 1.5α (= A) will be quantized to αq_1 (at t=1) or $\alpha' q_1$ (at t=2). In this case, $q(w, \alpha)$ is not updated.

Case 2: Similarly, all the points between $1.5\alpha'$ (= C) and 2α will be quantized to αq_2 (at t=1) or $\alpha' q_2$ (at t=2). In this case also, $q(w, \alpha)$ is not updated.

Case 3: If w is between points A and C, $q(w, \alpha)$ is updated from q_2 to q_1. There are two sub-cases. If w is greater than B, then the quantization error at t=2 is *reduced* compared with that of t=1. However, if w is less than B, the quantization error is *increased* compared with that of t=1. Because of this last case, we cannot say that (10) monotonically decreases $\mathcal{L}$. Now, in order for our procedure to oscillate and not converge, the update rule (10) should decrease α, so that we can go back to the situation of t=1, which however cannot happen. This is because when w is less than B, the quantization error is $w - \alpha'$, which is minimized when α *increases*. Therefore, there cannot be an indefinite oscillation between two α values, and $\mathcal{L}$ is only temporarily increased, but eventually converges to a minimum value.

The above is for the case of $\alpha' > \alpha$, but the convergence of the other case, $\alpha' < \alpha$, can be shown in a similar way.

Algorithm 1: FindBestQPS

```
   Input: $w$: pretrained weight, $S_U$: universal set, $N$: $2^{k-1}$, loss($w, S$): loss
          function
   Result: $S_Q$: the QPS that minimizes loss
1  $l_{\min} \leftarrow \infty$
2  for $S$ in each $N$-element subset of $S_U$ do
3      $\alpha \leftarrow$ FindScaleFactor($S, w$)
4      $l_{\text{curr}} \leftarrow$ loss($w$, $\alpha S$)
5      if $l_{\text{curr}} < l_{\min}$ then
6          $S_Q \leftarrow \alpha S$
7          $l_{\min} \leftarrow l_{\text{curr}}$
8      end
9  end
10 return $S_Q$
```

Finding the Best Subset There are only $\binom{|S_U|}{|S_Q|}$ number of subsets to consider. Thus we can search exhaustively all cases for the one that gives the minimum quantization error (see Algorithm 1). We do this for each layer or each output channel, depending on the quantization granularity.

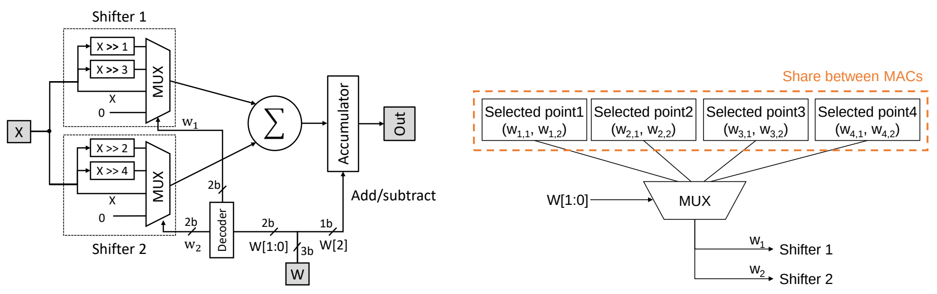
3.4 Designing a Universal Set

Our initial candidate for S_U is S_Q^{2log}, which is rich and simple enough to implement in hardware, requiring just two shifters and one adder for the multiplication. But in order to further optimize the multiplication hardware design, we explore the following design options. They all have the form of $a + b$, where a and b are either zero or 2^k as listed in Table 1. Note that thanks to a scale factor the sets are equivalent to their scaled versions, thus we assume that they are scaled such that the maximum value is one.

Table 1: Exploring universal set design, $S_U = \{a + b \,|\, a \in A, b \in B\}$.

Option	A	B
1	$\{2^{-1}, 2^{-2}, 2^{-3}, 0\}$	$\{2^{-1}, 2^{-2}, 2^{-4}, 0\}$
2	$\{1, 2^{-2}, 2^{-4}, 0\}$	$\{2^{-1}, 2^{-3}, 2^{-5}, 0\}$
3	$\{1, 2^{-1}, 2^{-2}, 2^{-3}, 2^{-4}, 0\}$	$\{1, 2^{-1}, 2^{-2}, 2^{-3}, 2^{-4}, 0\}$
4	$\{1, 2^{-1}, 2^{-2}, 2^{-3}, 2^{-4}, 2^{-5}, 0\}$	$\{1, 2^{-1}, 2^{-2}, 2^{-3}, 2^{-4}, 2^{-5}, 0\}$
5 (chosen)	$\{1, 2^{-1}, 2^{-3}, 0\}$	$\{1, 2^{-2}, 2^{-4}, 0\}$

The universal set designs differ in terms of hardware complexity as well as expressiveness. For instance, Option 4 generates a universal set with 23 elements whereas Option 3 and Option 5 have only 17 and 16 elements, respectively, but Option 4 also has the highest complexity. After a careful comparison (see Supplementary Material), we have chosen Option 5, which is rich enough yet has the lowest hardware complexity among the options.



(a) MAC (b) Decoder

Fig. 4: MAC design for the proposed subset quantization (3-bit case including sign-bit).

3.5 Hardware Design

Figure 4 illustrates the MAC (multiply-and-accumulate) hardware design for our quantizer. We assume that activation is linear-quantized and weight is quantized with our subset quantization. From the definition of our universal set (S_U) (i.e., sum of two power-of-two's), one straightforward implementation of multiplication is to use two shifters and one adder, in which shifters are variable-amount shifters (called *barrel shifters*). This naïve design, which would work for any universal set definition, can be optimized as shown in Figure 4 exploiting our universal set definition. In our optimized design shown in Figure 4a, each barrel shifter is replaced with a MUX and two constant-amount shifters. Note that a MUX is much cheaper than a barrel shifter and constant-amount shifters are just

Table 2: Image classification results for ***weight-only*** PTQ methods. * indicates results with the same model but different baseline. For quantized cases, we report *performance degradation* with absolute performance numbers in parentheses (same in later tables).

Network	W-bits/A-bits	32/32 (FP)	4/32	3/32	2/32
ResNet-18	OMSE+opt [7]	69.64	2.52 (67.12)	-	-
	AdaRound [23]	69.68/71.08*	0.97 (68.71)	3.01 (68.07*)	15.12 (55.96*)
	AdaQuant [12]	71.08	2.26 (68.82)	12.96 (58.12)	70.78 (0.30)
	BitSplit [28]	69.76	0.65 (69.11)	3.01 (66.75)	-
	BRECQ [19]	71.08	0.38 (70.70)	1.27 (69.81)	4.78 (66.30)
	SQ (Ours)	69.76	**0.30 (69.46)**	**1.00 (68.76)**	**4.14 (65.62)**
ResNet-50	OMSE+opt [7]	76.01	1.34 (74.67)	-	-
	AdaRound [23]	76.07/77.00*	0.84 (75.23)	3.58 (73.42*)	29.05 (47.95*)
	AdaQuant [12]	77.20/77.00*	3.50 (73.70)	12.39 (67.61*)	76.51 (0.49*)
	BitSplit [28]	76.13	0.55 (75.58)	2.89 (73.24)	-
	BRECQ [19]	77.00	0.71 (76.29)	1.39 (75.61)	4.60 (72.40)
	SQ (Ours)	76.13	**0.38 (75.75)**	**0.99 (75.14)**	**3.86 (72.27)**
InceptionV3	OMSE+opt [7]	77.40	3.74 (73.66)	-	-
	AdaRound [23]	77.40	1.64 (75.76)	-	-
	OCS [29]	77.40	72.60 (4.80)	-	-
	SQ (Ours)	77.24	**0.58 (76.66)**	3.61 (73.63)	11.36 (65.87)

wires, taking no logic gates. This is possible because of the way our universal set is constructed—each term has only four cases including zero.

Each MUX can select inputs independently, resulting in 16 different combinations, out of which only four cases are actually used due to the limited width of quantized weight value (which is 2 except the sign-bit in the figure). Thus the role of QPS is to select the four cases that will be actually used, and is implemented as the decoder. The decoder consists of a 16-bit register (= 4 × 4-bit) and a 4-bit 4-to-1 MUX. The 16-bit register stores the QPS chosen by Algorithm 1 (actually their 2-bit logarithm values), and can be shared among all the MACs in the same layer or channel, depending on the quantization granularity, thus having negligible area. Then a decoder is practically reduced to a simple MUX, but even this MUX can often be shared among a number of MACs within a MAC array, depending on the dataflow of the MAC array [5]. In summary, the hardware cost of our optimized MAC is very small: two 4-to-1 MUXes, one adder, and an accumulator, plus a 4-to-1 MUX, which could be shared among a number of MACs depending on the hardware dataflow.

4 Experiments

4.1 Experimental Setup

For evaluation we use three applications: image classification using ImageNet dataset [11,26], object detection [21], and semantic segmentation [4]. We perform

Table 3: Image classification results for ***fully-quantized*** PTQ methods. * indicates results with the same model but different baseline.

Network	W-bits/A-bits	32/32 (FP)	4/8	4/4	3/3	2/4
ResNet-18	AdaQuant [12]	71.97/71.08*	-	4.57 (67.40)	-	71.08 (0.21*)
	Seq. AdaQuant [12]	71.97	-	2.57 (69.40)	-	-
	AdaRound [23]	69.68	1.13 (68.55)	-	-	-
	ACIQ-Mix [1]	69.70	-	2.70 (67.00)	-	-
	LAPQ [24]	69.76/71.08*	-	9.46 (60.30)	-	70.90 (0.18*)
	BitSplit [28]	69.76	0.66 (69.10)	2.20 (67.56)	8.46 (61.30)	-
	BRECQ [19]	71.08	0.50 (70.58)	1.48 (69.60)	5.05 (66.03)	6.28 (64.80)
	SQ (Ours)	69.76	**0.37 (69.39)**	**1.21 (68.55)**	**4.62 (65.14)**	**5.06 (64.70)**
ResNet-50	AdaQuant [12]	77.20/77.00*	-	3.50 (73.70)	-	76.88 (0.12*)
	Seq. AdaQuant [12]	77.20	-	2.10 (75.10)	-	-
	AdaRound [23]	76.07	1.06 (75.01)	-	-	-
	OMSE+opt [7]	76.01	1.03 (74.98)	3.41 (72.60)	-	-
	ACIQ-Mix [1]	76.10	0.80 (75.30)	2.30 (73.80)	-	-
	LAPQ [24]	76.10/77.00*	-	6.10 (70.00)	-	76.86 (0.14*)
	BitSplit [28]	76.13	-	2.42 (73.71)	9.91 (66.22)	-
	PWLQ [10]	76.13	0.51 (75.62)	**1.28 (74.85)**	-	-
	BRECQ [19]	77.00	-	1.95 (75.05)	8.04 (68.96)	6.71 (70.29)
	SQ (Ours)	76.13	**0.46 (75.67)**	1.48 (74.65)	**6.80 (69.33)**	**5.43 (70.70)**
InceptionV3	AdaRound [23]	77.40	1.68 (75.72)	-	-	-
	ACIQ-Mix [1]	77.20	9.00 (68.20)	-	-	-
	OMSE+opt [7]	76.23	1.44 (74.79)	-	-	-
	PWLQ [10]	77.49	1.04 (76.45)	-	-	-
	SQ (Ours)	77.24	**0.66 (76.58)**	-	-	-

PTQ using pretrained weights; we have not modified a network or performed retraining before PTQ in any way. We apply SQ to weight only; activation is quantized using BRECQ [19] unless noted otherwise. We have used the framework, models, and pretrained weights mainly from the official PyTorch 1.6.0 and used Nvidia RTX Titan GPUs for experiments (CUDA 9.2, cuDNN 7.6.5).

4.2 Comprehensive Results

For all the results, we have repeated the PTQ experiments five times for each case by changing the random seed value, and the first and last layers are set to 8-bit linear quantization. In all the cases, the granularity of SQ is per-channel but we only use one QPS per layer. In other words, we use a scale factor for each channel in our PTQ process, but we map all weight values with one QPS for each layer. The per-layer QPS not only shows superior performance, but also allows for simpler hardware implementation.

Image Classification For image classification, we use the ImageNet dataset and present the results for ResNet series [11] and InceptionV3 [27]. We compare with various state-of-the-art PTQ methods [23,12,28,10,19], and our SQ, as applied to weight quantization. We have mainly referred to BRECQ[4] for our code implementation.

[4] https://github.com/yhhhli/BRECQ

Table 2 and 3 show the results. When only weight is quantized and activation is floating-point, our SQ shows consistently and significantly better performance than other PTQ methods and we see a similar trend when activation is quantized as well.[5] In particular, our SQ shows superior results in the ultra-low-bit condition for both 3-bit and 2-bit.

Semantic Segmentation For semantic segmentation task, we have applied our SQ method to the official DeepLabV3+ [4] source code[6] and compared it with other PTQ methods. We use MobileNetV2 [26] as the encoder backbone, and use ASPP [3] as the decoder. SQ is applied to all layers with weight parameters, and activation is quantized to 8-bit using linear quantization. Table 4 shows the results. We have conducted experiments on low-bit cases less than or equal 4-bit in consideration of performance and search time efficiency of SQ. The 4-bit result clearly shows the superior performance of our method compared with the previous PTQ method. The previous PTQ methods do not report the result of 3-bit or lower, but our 3-bit SQ result shows relatively small performance degradation, and is even better than that of 4-bit uniform quantization.

Table 4: PTQ methods on semantic segmentation with the MobileNetV2 [26] backbone.

Network	W-bits/A-bits	32/32 (FP)	4/8	3/8
DeepLabV3+ (mIoU%)	Uniform	70.81	20.76 (50.05)	-
	PWLQ [10]	70.81	3.15 (67.66)	-
	SQ (Ours)	70.81	**1.85 (68.96)**	**8.87 (61.94)**

Object Detection We have evaluated our SQ method using object detection task, SSD-Lite [21]. We have modified the author's official PyTorch source code[7], and used MobileNetV2 [26] as the backbone network. Again we have applied SQ to all layers with weight parameters and 8-bit linear quantization is applied to activation. Table 5 shows the results. In the case of 4-bit, SQ shows a similar level of performance as that of the current state-of-the-art. In the 3-bit case, the performance trend is similar to that of the semantic segmentation results, but the performance degradation is smaller and again our 3-bit SQ result shows lower performance degradation than 4-bit uniform quantization.

[5] For InceptionV3 4-bit in Table 3, we only present the result with 8-bit linear quantization because our implementation for low-bit activations [19] did not work properly in this case.

[6] https://github.com/jfzhang95/pytorch-deeplab-xception

[7] https://github.com/qfgaohao/pytorch-ssd

Table 5: PTQ methods on object detection with the MobileNetV2 [26] backbone.

Network	W-bits/A-bits	32/32 (FP)	4/8	3/8
SSD-Lite (mAP%)	Uniform	68.70	3.91 (64.79)	-
	DFQ [22]	68.47	0.56 (67.91)	-
	PWLQ [10]	68.70	**0.38 (68.32)**	-
	SQ (Ours)	68.59	**0.38 (68.21)**	**3.87 (64.72)**

5 Conclusion

We presented a novel non-uniform quantization method called subset quantization (SQ), which is a high-performing PTQ methods at extreme low-precision while remaining hardware-friendly. Subset quantization adds a new dimension to quantizer definition by allowing quantization points to be defined as a subset of a larger pool called universal set. This view allows a quantization point set to be defined in a more flexible fashion so that it can adapt to diverse input statistics, which is very useful for deep neural networks. Our experimental results with challenging vision tasks demonstrate that our SQ results for ultra low-bit weight quantization outperform state-of-the-art quantizers of the same precision.

Acknowledgements: This work was supported by the Samsung Advanced Institute of Technology, Samsung Electronics Co., Ltd., by IITP grants (No. 2020-0-01336, Artificial Intelligence Graduate School Program (UNIST), and No. 1711 080972, Neuromorphic Computing Software Platform for Artificial Intelligence Systems) and NRF grant (No. 2020R1A2C2015066) funded by MSIT of Korea, and by Free Innovative Research Fund of UNIST (1.170067.01).

References

1. Banner, R., Nahshan, Y., Soudry, D.: Post training 4-bit quantization of convolutional networks for rapid-deployment. In: Advances in Neural Information Processing Systems. vol. 32. Curran Associates, Inc. (2019), `https://proceedings.neurips.cc/paper/2019/file/c0a62e133894cdce435bcb4a5df1db2d-Paper.pdf`
2. Chen, C., Chen, Q., Xu, J., Koltun, V.: Learning to see in the dark. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 3291–3300 (2018)
3. Chen, L., Papandreou, G., Kokkinos, I., Murphy, K., Yuille, A.L.: Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. CoRR **abs/1606.00915** (2016), `http://arxiv.org/abs/1606.00915`
4. Chen, L.C., Zhu, Y., Papandreou, G., Schroff, F., Adam, H.: Encoder-decoder with atrous separable convolution for semantic image segmentation. Eur. Conf. Comput. Vis. (ECCV) (2018)
5. Chen, Y., Krishna, T., Emer, J.S., Sze, V.: Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. IEEE Journal of Solid-State Circuits **52**(1), 127–138 (2017). https://doi.org/10.1109/JSSC.2016.2616357
6. Choi, J., Wang, Z., Venkataramani, S., Chuang, P.I.J., Srinivasan, V., Gopalakrishnan, K.: Pact: Parameterized clipping activation for quantized neural networks. arXiv preprint arXiv:1805.06085 (2018)
7. Choukroun, Y., Kravchik, E., Yang, F., Kisilev, P.: Low-bit quantization of neural networks for efficient inference. In: 2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW). pp. 3009–3018 (2019). https://doi.org/10.1109/ICCVW.2019.00363
8. Ding, R., Liu, Z., Chin, T.W., Marculescu, D., Blanton, R.D.S.: Flightnns: Lightweight quantized deep neural networks for fast and accurate inference. In: Proceedings of the 56th Annual Design Automation Conference 2019. DAC '19, Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3316781.3317828, `https://doi.org/10.1145/3316781.3317828`
9. Esser, S.K., McKinstry, J.L., Bablani, D., Appuswamy, R., Modha, D.S.: Learned step size quantization. In: International Conference on Learning Representations (2019)
10. Fang, J., Shafiee, A., Abdel-Aziz, H., Thorsley, D., Georgiadis, G., Hassoun, J.: Post-training piecewise linear quantization for deep neural networks. In: European Computer Vision Association (2020)
11. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: IEEE Conference on Computer Vision and Pattern Recognition (2016)
12. Hubara, I., Nahshan, Y., Hanani, Y., Banner, R., Soudry, D.: Improving post training neural quantization: Layer-wise calibration and integer programming. CoRR **abs/2006.10518** (2020), `https://arxiv.org/abs/2006.10518`
13. Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H., Kalenichenko, D.: Quantization and training of neural networks for efficient integer-arithmetic-only inference. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 2704–2713 (2018)
14. Jung, S., Son, C., Lee, S., Son, J., Han, J.J., Kwak, Y., Hwang, S.J., Choi, C.: Learning to quantize deep networks by optimizing quantization intervals with task loss. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. pp. 4350–4359 (2019)

15. Krishnamoorthi, R.: Quantizing deep convolutional networks for efficient inference: A whitepaper. arXiv preprint arXiv:1806.08342 (2018)
16. Lee, E.H., Miyashita, D., Chai, E., Murmann, B., Wong, S.S.: Lognet: Energy-efficient neural networks using logarithmic computation. In: 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). pp. 5900–5904 (2017). https://doi.org/10.1109/ICASSP.2017.7953288
17. Lee, S., Sim, H., Choi, J., Lee, J.: Successive log quantization for cost-efficient neural networks using stochastic computing. In: Proceedings of the 56th Annual Design Automation Conference 2019. DAC '19, Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3316781.3317916, `https://doi.org/10.1145/3316781.3317916`
18. Li, Y., Dong, X., Wang, W.: Additive powers-of-two quantization: An efficient non-uniform discretization for neural networks. In: International Conference on Learning Representations (2020)
19. Li, Y., Gong, R., Tan, X., Yang, Y., Hu, P., Zhang, Q., Yu, F., Wang, W., Gu, S.: Brecq: Pushing the limit of post-training quantization by block reconstruction. In: International Conference on Learning Representations (2021)
20. Lim, B., Son, S., Kim, H., Nah, S., Mu Lee, K.: Enhanced deep residual networks for single image super-resolution. In: Proceedings of the IEEE conference on computer vision and pattern recognition workshops. pp. 136–144 (2017)
21. Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S.E., Fu, C., Berg, A.C.: Ssd: Single shot multibox detector. In: Eur. Conf. Comput. Vis. (ECCV) (2016)
22. Markus Nagel, Mart van Baalen, T.B., Welling, M.: Data-free quantization through weight equalization and bias correction. In: IEEE/CVF International Conference on Computer Vision (2019)
23. Nagel, M., Amjad, R.A., van Baalen, M., Louizos, C., Blankevoort, T.: Up or down? adaptive rounding for post-training quantization. CoRR **abs/2004.10568** (2020), `https://arxiv.org/abs/2004.10568`
24. Nahshan, Y., Chmiel, B., Baskin, C., Zheltonozhskii, E., Banner, R., Bronstein, A.M., Mendelson, A.: Loss aware post-training quantization. CoRR **abs/1911.07190** (2019), `http://arxiv.org/abs/1911.07190`
25. Oh, S., Sim, H., Lee, S., Lee, J.: Automated log-scale quantization for low-cost deep neural networks. in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR) pp. 742–751 (2021)
26. Sandler, M., Howard, A.G., Zhu, M., Zhmoginov, A., Chen, L.: Inverted residuals and linear bottlenecks: Mobile networks for classification, detection and segmentation. In: IEEE Conference on Computer Vision and Pattern Recognition (2018)
27. Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z.: Rethinking the inception architecture for computer vision. CoRR **abs/1512.00567** (2015), `http://arxiv.org/abs/1512.00567`
28. Wang, P., Chen, Q., He, X., Cheng, J.: Towards accurate post-training network quantization via bit-split and stitching. In: International Conference on Machine Learning (2020)
29. Zhao, R., Hu, Y., Dotzel, J., Sa, C.D., Zhang, Z.: Improving neural network quantization without retraining using outlier channel splitting. In: International Conference on Machine Learning (2019)
30. Zhao, X., Wang, Y., Cai, X., Liu, C., Zhang, L.: Linear symmetric quantization of neural networks for low-precision integer hardware. In: International Conference on Learning Representations (2020), `https://openreview.net/forum?id=H1lBj2VFPS`