# Centered Symmetric Quantization for Hardware-Efficient Low-Bit Neural Networks

Faaiz Asim*[1]
faaizasm@unist.ac.kr
Jaewoo Park*[2]
hecate64@unist.ac.kr
Azat Azamat[3]
azatkariuly@unist.ac.kr
Jongeun Lee†[1]
jlee@unist.ac.kr

Dept. of Electrical Engineering[1]
Dept. of Physics[2]
Dept. of CSE[3]
Ulsan National Institute of Science and Technology (UNIST)
Ulsan, Korea

**Abstract**

In extremely low-precision quantization, the use of unequal numbers of positive and negative quantization levels seems sub-optimal when dealing with symmetrical data distribution such as weight parameters of a neural network. In this paper, based on an observation that a significant amount of quantization error can be caused by a quantizer with unequal vs. equal numbers of quantization levels, we propose a quantizer that has perfectly zero-centered quantization levels for weight quantization, dubbed *Centered Symmetric Quantization (CSQ)*, with an analysis and empirical quantification of why and how much performance gain CSQ can provide over conventional linear quantization (CLQ). Moreover, noting that it is tricky to implement $n$-bit CSQ using just $n$-bit arithmetic hardware, we also propose efficient methods of implementing CSQ using (i) standard multiplication hardware and (ii) bit-wise binarized neural-net hardware. Our experimental results using state-of-the-art quantization-aware training methods on ResNets and MobileNet-v2 show that using CSQ for weight in place of CLQ does offer significant performance advantage at extremely low-bit precision (2$\sim$3 bits) without any considerable overhead.

## 1 Introduction

Recently the performance of computer vision and image processing methods has been dramatically improved by deep neural networks, which however has high computational and memory requirements. To address this challenge, quantized neural networks (QNNs) [8, 10, 14, 15, 29] are widely used. While QNNs can reduce both computational complexity and memory requirement quite effectively over a full-precision (i.e., floating-point) version, the use of limited precision can degrade the performance of a QNN. One way to ameliorate

*equal contribution
†Jongeun Lee is the corresponding author (E-mail: jlee@unist.ac.kr).

the performance degradation of a QNN is to use quantization-aware training (QAT), which is centered around a *quantizer* function, a function from real numbers to integers (or fixed-point numbers). Quantizers often have parameters, and how to parameterize a quantizer function (*e.g.* using range [14] or scale [10]) and how to determine parameters (*e.g.* statistically [7, 30] vs. by training [14]) as well as the choice of a quantizer function itself (*e.g.* linear [10, 29] vs. nonlinear [6, 18, 19, 23, 24]) are all very critical, affecting the result of QAT. In particular, designing a good quantizer function that is both well-performing and hardware-efficient is of central importance for successful deployment of QNNs. In this paper we are concerned with the problem of designing a good quantizer function especially for weights, as opposed to other aspects such as training methods.

The most commonly used type of quantizer is a uniform step-size (or linear) quantizer, which places quantization levels at a uniform interval. Linear quantizers are very popular, in part due to their compatibility with conventional integer MAC (multiply and accumulate) hardware [29] as well as the availability of many sophisticated training methods developed for linear quantizers [8, 10, 14, 17, 20, 29].

At extremely low precision, however, a linear quantizer becomes severely unbalanced in terms of representing positive vs. negative data. For instance, a 2-bit quantizer has only one quantization level ($+1$) on the positive side vs. two ($-2, -1$) on the negative side, which can impact performance negatively (see Section 3.1 for analysis). This severe imbalance in the number of quantization levels at extremely low precision is what motivated the use of perfectly balanced quantization levels such as $\{-3, -1, 1, 3\}$ in recent low-precision QNN training works [4, 5, 7, 11, 17, 30]. We call the use of such a perfectly balanced set of quantization levels *Centered Symmetric Quantization (CSQ)* (a more precise definition is provided in Section 3.2). However, despite the increasing use of CSQ in extreme low-precision weight quantization, it is unknown whether and how much performance improvement can be made by the use of a CSQ quantizer instead of a conventional linear quantizer (CLQ). In fact, none of the previous work using CSQ has explicitly proposed CSQ or even provide a mathematical definition of a CSQ quantizer; rather, they focus on *training methods* and the use of a CSQ quantizer seems incidental.

In addition, efficient hardware realization of CSQ is not straightforward, which, however, is overlooked by all of the previous work using CSQ [4, 5, 11, 17]. For instance, while 2-bit CSQ values can be easily stored in 2-bit memory, performing multiplication with a 2-bit CSQ value using a 2-bit multiplier is problematic—it requires a 3-bit multiplier to handle large operands such as 3 and $-3$, which defeats the purpose of using 2-bit quantization. Despite the apparent difficulty of how to handle larger-than-$n$-bit values (*e.g.*, 3 and $-3$ when $n$=2) using an $n$-bit multiplier, most previous work using CSQ does not explain how their chosen quantization levels can be mapped efficiently on hardware.

In this paper, we first provide a definition of CSQ along with a straightforward training scheme. We then empirically compare CSQ with CLQ in terms of network performance on CIFAR-10 and ImageNet datasets, which shows that replacing a CLQ quantizer with a CSQ quantizer can indeed generate modest performance gain. Although performance improvement by CSQ may appear small (CSQ is found to be 0.26$\sim$0.47% better than CLQ at 2-bit, and less at higher precision), at extremely low precision (2$\sim$3-bit) even this amount of improvement is often considered significant (see *e.g.* [17]). Also considering the very limited design space of nonlinear as well as linear quantizers at 2-bit, CSQ does offer a valuable alternative. Second, we propose efficient realization methods of CSQ using, (i) standard multipliers and (ii) bitwise Binarized Neural Network (BNN) hardware. Note that though CSQ can be most efficiently implemented in custom hardware, our proposed method can benefit

software implementations as well, so long as the efficiency of a software implementation is inversely proportional to the bitwidth of arithmetic operations (*e.g.* Dorefa-net-style [30] XNOR-popcount based implementation on CPU or GPU). Our QNN mapping experiments demonstrate that our proposed methods can map CSQ to digital hardware as efficiently as CLQ. Combined, our results show the superiority of CSQ over CLQ at very low precision.

# 2 Background and Previous Work

**Quantization Primer:** A uniform step-size quantizer can be formulated as

$$\tilde{x} = \text{clip}(\left\lfloor \frac{x}{s} \right\rceil, L, U) \cdot s \tag{1}$$

where $L$ and $U$ (the lower and upper bound) are the minimum and maximum integer values that $x_Q$ can take, $\lfloor \cdot \rceil$ is the round operation, and $\text{clip}(x,a,b) = \min(\max(x,a),b)$.

In the case of symmetric quantizer (for signed input):

$$L = -2^{b-1}, \quad U = 2^{b-1} - 1 \tag{2}$$

In the case of asymmetric quantizer (for un-signed input):

$$L = 0, \quad U = 2^b - 1. \tag{3}$$

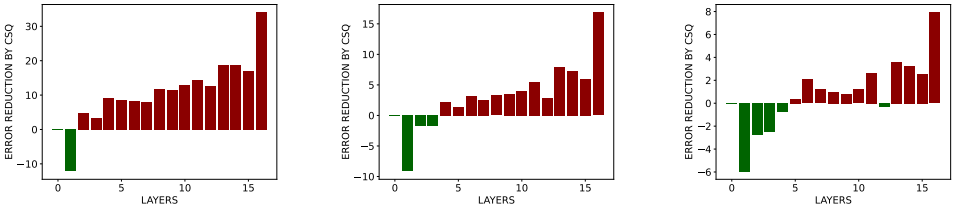
From now on we will refer to the quantization method defined by (1) and (2) as the Conventional Linear Quantizer (CLQ). We do not consider non-uniform quantization for the scope of this paper.

**Previous Work Using CSQ:** The idea of using a zero-centered quantizer is also seen in [4, 5, 7, 10, 17, 30]. However, none of these works address the problem of realizing zero-centered quantization on hardware. Instead they focus on novel training methods for their quantizers. One exception is [9], whose hardware implementation is presented in [10], which, however, ends up using *non-uniform* quantization levels (i.e., $\{-4,-1,1,4\}$ for 2-bit quantization).

# 3 CSQ: Motivation, Definition and Analysis

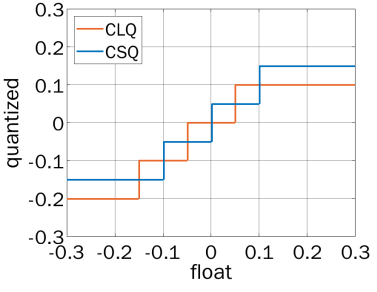
## 3.1 Impact of Unbalanced Quantization Levels

To motivate the use of perfectly balanced quantization levels, Figure 1 shows the impact of using unbalanced quantization levels (i.e., CLQ) on weight quantization error in each layer of ResNet-18 trained for ImageNet classification. The $y$-axis shows the percent increase in weight quantization error of using CLQ vs. prefectly balanced quantization levels (i.e., CSQ): error_increase$(\%) = (E_{CLQ} - E_{CSQ})/E_{CSQ} \times 100$, where $E_{CLQ}$ and $E_{CSQ}$ are the quantization errors defined as $\langle (w - w_q)^2 \rangle$, $w$ is the full precision weight, and $w_q$ is the quantized weight using either CLQ or CSQ. This experiment is done in a QAT setting where the layer-wise step size parameter is optimized by an exhaustive search for each case (CLQ or CSQ). One can see that the imbalance in weight quantization levels can increase weight quantization errors by about 10$\sim$20% in middle layers at 2-bit precision albeit less at higher precision. Also it is interesting to note that CSQ is more beneficial in later layers.

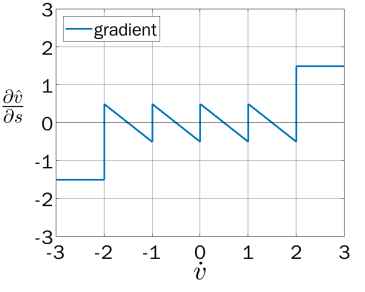



(a) 2-bit precision (b) 3-bit precision (c) 4-bit precision

Figure 1: Percent error reduction using CSQ (for ResNet-18 trained on ImageNet).




(a) Quantization of 2-bit signed data using conventional linear quantization (CLQ) vs. CSQ (ours).




(b) The gradient of the CSQ quantization output w.r.t. step size.

Figure 2: CSQ quantizer and gradient.

## 3.2 Quantizer Function for CSQ

We define CSQ quantizer as a quantizer function that has both *uniform step size* and *perfect symmetry* between the positive and negative quantization levels (*e.g.* $\{-1.5,-0.5,0.5,1.5\}$ for 2-bit). Mathematically, a CSQ quantizer (and a simulated quantizer for the purpose of training) can be formulated as follows.

$$\dot{v} = \left\lfloor \frac{v}{s} + 0.5 \right\rfloor - 0.5 \tag{4}$$

$$\bar{v} = \text{clip}\left(\dot{v}, -Q, Q\right) \tag{5}$$

$$\hat{v} = \bar{v} \times s \tag{6}$$

where $v$ is any input value, $s$ is the step size, and $Q = 2^{b-1} - 0.5$ with $b$ being the quantization precision (i.e., the number of bits). Here $\bar{v}$ is not an integer. Nevertheless, it is an exact value that can be represented by the $b$-bit CSQ format. Moreover, our proposed CSQ format permits efficient hardware and software realizations (see Section 4.2). Therefore $\bar{v}$ represents the value that is computed by $b$-bit hardware. Finally, $\hat{v}$ is the scaled-back version of $\bar{v}$, defined and used for the purpose of training. The proposed quantizer provides equal representation for the positive and negative sides of the input distribution. Figure 2a shows the 2-bit quantizer functions for conventional linear quantization and our CSQ. Figure 2b shows the gradient w.r.t. the step size, which is a quantization parameter.

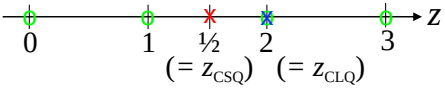



Figure 3: Zero-points for CLQ and CSQ at 2-bit (marked as "x"). Affine quantizer allows integer zero-point only (shown as green circles) whereas in relaxed affine quantizer, zero-point can take any real value.

### 3.3 Analysis of Uniform Quantizer via Affine Quantization

In this section we present an analytical comparison between CSQ and CLQ using a more general framework of affine quantization, which is defined as follows [15]:

$$x_{int} = \left\lfloor \frac{x}{s} \right\rceil + z, \quad \bar{x} = \text{clip}(x_{int}, 0, 2^b - 1), \quad \hat{x} = s(\bar{x} - z) \tag{7}$$

where $z$ is called *zero-point*, and must be an integer. Given the integer constraint on zero-point, an affine quantizer cannot represent CSQ (see Figure 3). However, if we relax the zero-point to be any real value, CLQ and CSQ can both be considered as a special case of the *relaxed* affine quantizer. Then the zero-point for CLQ ($z_{\text{CLQ}}$) and CSQ ($z_{\text{CSQ}}$) can be defined as follows:

$$z_{\text{CLQ}} = 2^{b-1}, \quad z_{\text{CSQ}} = 2^{b-1} - 0.5 \tag{8}$$

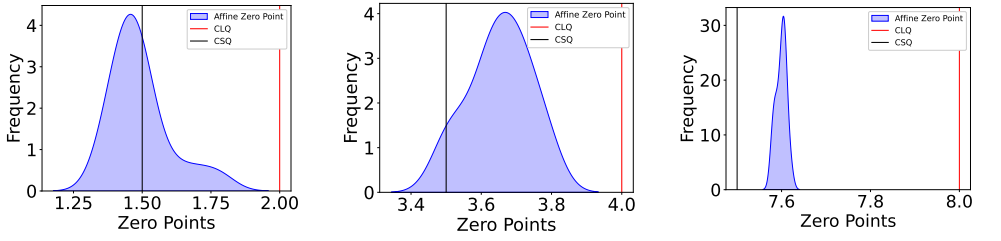
For instance, at $b = 2$, the term $\left\lfloor \frac{x}{s} \right\rceil$ has the range of $\{-2, -1, 0, 1\}$ in (signed) CLQ. Thus to satisfy (7), we have $z_{\text{CLQ}} = 2$.

The above view provides us with a methodology to somewhat objectively compare CSQ and CLQ by the distance of the optimal real-valued zero-point ($z^*$) to $z_{\text{CSQ}}$ vs. $z_{\text{CLQ}}$. Figure 4 shows the distribution of optimal zero-points for ResNet-20 for 2, 3 and 4-bit precision trained on CIFAR-10. The training settings are the same as explained in Section 5.1. These analytical results lead to the following key insights:

- The optimal zero-point is typically closer to CSQ than CLQ which indicates the importance of perfectly symmetric quantization levels (over exact zero-representation).
- As the precision increases, the optimal zero-point tends to move away from CSQ towards CLQ. A similar observation could be deduced from (8), since the difference between $z_{\text{CLQ}}$ and $z_{\text{CSQ}}$, relative to the entire quantization range, diminishes exponentially as $b$ increases (see Appendix I).

## 4 Efficient Realization of CSQ Multiplication

Here we present our methods to efficiently realize multiplications involving CSQ values, one using standard multipliers and one using bitwise BNN hardware. Since the activations are usually unsigned due to ReLU activation, in this section we conisder unsigned activations quantized by CLQ and weights quantized by CSQ. However, our methods can be extended to CSQ-CSQ multiplication as well.




(a) 2-bit precision (b) 3-bit precision (c) 4-bit precision

Figure 4: Distribution of optimal zero-point values of relaxed affine quantizer, across layers. The graphs also show the zero-points for CSQ and CLQ, represented as vertical lines.

## 4.1 Realization of CSQ via Affine Quantizer

In Section 3.3 we showed that CSQ can be considered as a special case of relaxed affine quantizer with a specific zero-point value. Based on this, we propose an efficient method to realize CSQ on standard multipliers (such as on CPUs; see Appendix E for derivation).

$$\hat{w}\hat{x} = ((\bar{w} - 0.5) \times s_w)(\bar{x} \times s_x) = \bar{w}\bar{x}s_w s_x - \underbrace{0.5 \times s_w s_x \bar{x}}_{\text{overhead}} \tag{9}$$

where $\bar{w}$ is an integer and $\bar{w} - 0.5$ is an exact CSQ value (i.e., 0.5 is the zero-point). Note that activations do not have a zero-point as they are quantized using unsigned CLQ. So our method is to use the integer $\bar{w}$ instead of its CSQ value during a tensor operation (*e.g.*, convolution), whose result is later corrected by the overhead term $0.5 \times s_w s_x \bar{x}$, which involves only the input activation and step size parameters, and has negligible operational overhead *on CPUs and GPUs* compared to the tensor operation. This method is very flexible, and can be applied to any software or hardware implementation.

A CSQ quantizer realized using (9) is more efficient than a relaxed affine quantizer as follows:

- Whereas CSQ's overhead term can be implemented using a simple bit-shift operation (multiplying 0.5 is equivalent to one-bit shift right), the overhead term of a relaxed affine quantizer would require an actual multiplication, which is much more costly.
- CSQ's zero-point value (i.e., 0.5) is fixed and therefore can be hardwired, but that of a relaxed affine quantizer needs to be retrieved from the memory, requiring additional memory access.

However, our CSQ realization method based on (9) can still result in small overhead *on custom hardware platforms* such as FPGAs (see results in Table 4) due to the extra hardware for the overhead term. Therefore, we also propose our bitwise-style CSQ realization method in Section 4.2 for the most efficient implementation of CSQ in hardware.

## 4.2 Realization of CSQ on Bitwise BNN Hardware

This method targets bitwise BNN hardware, which has an important advantage. Not only can it support QNNs using the method in [30] but it can also change precision at runtime without runtime overhead.

Table 1: Comparison of number representations: CLQ vs. CSQ (2-bit example)

| 2-bit binary | $\text{CLQ}_s$ | $\text{CLQ}_u$ | CSQ |
|---|---|---|---|
| 00 | 0 | 0 | $-3$ |
| 01 | 1 | 1 | $-1$ |
| 10 | $-2$ | 2 | 1 |
| 11 | $-1$ | 3 | 3 |

The core of bitwise BNN hardware is an AND-popcount or XNOR-popcount engine [27], which performs an inner-product computation between two $N$-dimensional bit vectors. Let $\mathbf{x}$ and $\mathbf{v}$ be an unsigned CLQ vector and a CSQ vector, respectively, each of which is $N$-dimensional and $n$-bit.

An $n$-bit binary number $a_{n-1}a_{n-2}...a_0$ ($n \geq 2$) represents the following value: $A_b = \sum_{i=0}^{n-1} a_i 2^i$. We define the CSQ number format as exemplified in Table 1, which leads to the following equalities (the second equality is not obvious, but is correct).

$$A_{csq} = \sum_{i=0}^{n-1} a_i 2^i - (2^n - 1)/2 = \sum_{i=0}^{n-1} (-1)^{a_i+1} 2^{i-1} \tag{10}$$

To avoid dealing with fractional numbers, let us use the $2\times$ scaled version of CSQ (*e.g.* $\{-3,-1,1,3\}$ for 2-bit precision). Then,

$$A_{csq2} = \sum_{i=0}^{n-1} (-1)^{a_i+1} 2^i, \tag{11}$$

which has a very similar mathematical structure as a CLQ number, allowing us to use the same trick of changing the order of precision ($n$) and vector dimension ($N$) as in the inner-product computation of two CLQ numbers. Finally, we arrive at the following inner-product computation method:

$$\mathbf{v} \cdot \mathbf{x} = (\mathbf{v}_H \cdot \mathbf{x}_H << 2) + (\mathbf{v}_H \cdot \mathbf{x}_L << 1) + (\mathbf{v}_L \cdot \mathbf{x}_H << 1) + \mathbf{v}_L \cdot \mathbf{x}_L \tag{12}$$

which is shown for the 2-bit case ($n = 2$). $\mathbf{x}_H$ and $\mathbf{x}_L$ (similarly for $\mathbf{v}_H$ and $\mathbf{v}_L$) are the $N$-dimensional bit-vectors of $\mathbf{x}$ containing only the higher and lower bits, respectively, and $<<$ is the bitwise shift-left operation. Each product on the right-hand side of (12) can be computed on BNN hardware in a single cycle. Thus $\mathbf{v} \cdot \mathbf{x}$ can be computed in four cycles, using an additional adder/accumulator. It is worth mentioning that the same method as illustrated in (12) is also used when computing the inner-product of two CLQ vectors.

Now for the inner-product of two bit-vectors (*e.g.* $\mathbf{v}_H \cdot \mathbf{x}_H$) we can use the same structure of a bitwise operation followed by popcount, with a slight variation.

$$\mathbf{v}_{csq2} \cdot \mathbf{x}_{clq_u} = 2 \cdot \text{popcount}(\text{AND}(\mathbf{v}_{csq2}, \mathbf{x}_{clq_u})) - \text{popcount}(\mathbf{x}_{clq_u}) \tag{13}$$

We subtract popcount($\mathbf{x}_{clq_u}$), since the bits of the CSQ bit-vector corresponding to the zero elements of the CLQ bit-vector must be ignored. Note that BNN hardware, *e.g.* [27], relies on the same method for the inner-product computation of two bit-vectors, except that we use AND instead of XNOR and to subtract a popcount value, we need an additional popcount operation. However, accelerating QNNs with the CLQ format also requires AND operations

in the exact same manner, and the additional popcount operation does not increase the hardware cost. Thus the bit-vector-level complexity of using CSQ is nearly the same as that of using CLQ. Since there is no distinction at the vector level, the inner-product operation with CSQ can be implemented as efficiently as with CLQ.

# 5 Experiments

## 5.1 Experimental Setup

We conduct experiments on CIFAR-10 [16] and ImageNet [25] using ResNet-18, ResNet-20, ResNet-34 [12] and MobileNet-v2 [26]. All quantized models on ImageNet are initialized with the weights of pretrained full-precision model of the same network. The first and last layers are kept at 8-bit precision. Other than convolution and fully connected layers, all the other layers, *e.g*. batch norm, are kept in full precision.

For ImageNet experiments we use the training recipe of [10]. We use stochastic gradient descent (SGD) optimizer, with 0.9 momentum, cosine learning rate decay [21] without restarts, and the initial learning rate of 0.01. Weight decay is $0.25 \times 10^{-4}$ for 2-bit, $0.5 \times 10^{-4}$ for 3-bit, and $10^{-4}$ for 4-bit quantization. The quantized models are fine-tuned for 90 epochs. For ResNet-18 ImageNet experiments, we train the full-precision model ourselves from scratch.

For CIFAR-10 experiments we conduct experiments using LSQ[10] and NICE[2] for training. LSQ quantized models are trained from scratch. We use the same setting as with ImageNet experiments except the following: the initial learning rate is 0.1, weight decay is $10^{-4}$, and each model is trained for 300 epochs. Experiments are conducted using ResNet-20 network. NICE quantized models are trained using the official code[1] provided by authors. The quantized models are initialized with full precision model and fine-tuned for 120 epochs. We used 0.01 initial learning rate, 0.9 momentum, 0.04 weight decay and multi-step learning rate decay. Experiments are conducted using ResNet-18 network.

## 5.2 Experimental Results

For CIFAR-10 experiments we compare our method with CLQ using a state-of-the-art QAT method by [10] and [2]. [2] uses Reduced Symmetric Quantization (RSQ), which is explained in Appendix B. Each case is repeated five times; mean $\pm$ std. dev. is reported for each case. CIFAR-10 results are summarized in Table 2.

For ImageNet experiments, we compare CLQ vs. CSQ using ResNet and MobileNet. We also compare our method with PACT [8], LQ-Nets [28] and QIL [14]. Since we use the training method by LSQ [10], the CLQ case also represents the previous work (LSQ). However, LSQ use pre-activation ResNet [12] which has higher performance than the standard ResNet architecture, and their trained models are not available. Therefore for a fair comparison, we have implemented LSQ ourselves, and use it as the baseline. [17] is another recent work that presents a training method for QNNs, that outperforms LSQ. However, they already use a quantizer that results in zero-centered quantization levels, similar to the one proposed in this paper. Therefore, we do not compare our results with them. The ImageNet results are summarized in Table 3 which shows that CSQ outperforms CLQ for all cases. However,

[1] https://github.com/Lancer555/NICE

Table 2: Comparison of RSQ (see Appendix B) and CLQ vs. CSQ (ours) on ResNet-20 and ResNet-18 for CIFAR-10.

| | | Top-1 Accuracy @ Precision | | |
|---|---|---|---|---|
| **Precision (W/A) :** | | 2/2 | 3/3 | 4/4 |
| | | ResNet-18 | | |
| **Training Method** | **Quantizer** | *Full Precision: 93.20* | | |
| NICE | RSQ | 92.07±0.06 | 93.00±0.13 | **93.48±0.16** |
| NICE | CSQ | **92.58±0.10** | **93.24±0.12** | 93.45±0.16 |
| | | ResNet-20 | | |
| | | *Full Precision: 91.22* | | |
| LSQ | CLQ | 89.09±0.11 | 90.70±0.18 | **91.07±0.13** |
| LSQ | CSQ | **89.46±0.23** | **90.80±0.12** | 90.96±0.25 |

Table 3: Comparison of CLQ vs. CSQ (ours), along with previous QAT results on ImageNet. The CLQ case also represents LSQ [10]. MobileNet-v2 2-bit case did not converge.

| | Top-1 Accuracy @ Precision | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **Network** | ResNet-18 | | | ResNet-34 | | | MobileNet-v2 | | |
| | *Full Precision: 70.58* | | | *Full Precision: 73.31* | | | *Full Precision: 71.88* | | |
| **Precision (W/A)** | 2/2 | 3/3 | 4/4 | 2/2 | 3/3 | 4/4 | 2/2 | 3/3 | 4/4 |
| PACT | 64.40 | 68.10 | 69.20 | - | - | - | - | - | - |
| LQ-Nets | 64.90 | 68.20 | 69.30 | 69.80 | 71.90 | - | - | - | - |
| QIL | 65.70 | 69.20 | 70.10 | 70.60 | 73.10 | 73.70 | - | - | - |
| CLQ (LSQ) | 66.59 | 69.38 | 70.52 | 70.56 | 73.21 | 73.82 | - | 60.41 | 66.82 |
| CSQ (LSQ) | **66.92** | **69.48** | **70.63** | **70.82** | **73.29** | **74.01** | - | **60.89** | **66.98** |

the performance gain by CSQ diminishes as precision increases which is consistent with the insights from our analysis in Section 3.3. [2]

## 5.3 FPGA Synthesis Results

In Table 4 we compare hardware synthesis results for five cases: (i) CLQ realized using standard multiplication, (ii) CLQ on bitwise BNN hardware, (iii) CSQ on bitwise BNN hardware0, (iv) (relaxed) affine quantizer using a 8-bit fixed-point zero-point, and (v) affine quantizer with a hardwired zero-point for CSQ. The hardware is a 16x16 two-dimensional array of PEs (processing elements) performing an MVM (matrix-vector multiplication) operation. For synthesis we use Xilinx Vivado, targeting Zynq ZCU104 FPGA, and the FPGA resource utilization and the minimum clock cycle time (latency) are reported. ADP (area-delay product) is calculated as the product of latency and the geometric mean of LUT and FF utilization, with the case of CLQ being 100%.[3]

The table indicates that at 2-bit precision, bitwise BNN hardware is about 10% more efficient in ADP than a multiplier-based design. Between the bitwise BNN hardware cases, CLQ is slightly more efficient than CSQ, but the difference is very marginal. Overall, CSQ implemented on BNN hardware can reduce ADP over multiplier-based CLQ by 8.6% at 2-bit, though this improvement becomes less at higher precision ($\geq$ 3-bit). Our results also indicate that the relaxed affine quantizer has a considerable hardware overhead. On the other

[2] In Appendix F we provide additional experiments for CSQ vs. CLQ using Post-Training Quantization (PTQ) and QAT with knowledge distillation.

[3] ADP is a commonly used metric for hardware efficiency, and we use FPGA as opposed to ASIC, since (i) FPGA is frequently used for deep neural network acceleration and (ii) FPGA synthesis result is easily reproducible and more complete.

Table 4: 16x16 MVM hardware synthesis results on Xilinx FPGA

| Precision | 2-bit | | | | 3-bit | | | |
|---|---|---|---|---|---|---|---|---|
| | Utilization | | Latency | ADP | Utilization | | Latency | ADP |
| | LUT | FF | (ns) | (%) | LUT | FF | (ns) | (%) |
| CLQ w/ Mult. | 897 | 671 | 2.07 | 100 | 1171 | 1009 | 2.11 | 100 |
| CLQ w/ BNN HW | 673 | 624 | 2.20 | 88.8 | 992 | 926 | 2.31 | 96.5 |
| CSQ w/ BNN HW | 681 | 630 | 2.24 | 91.4 | 1001 | 944 | 2.35 | 99.6 |
| (Relaxed) Affine Quantizer | 1002 | 849 | 2.81 | 161 | 1362 | 1277 | 2.89 | 166 |
| CSQ w/ Affine HW | 913 | 680 | 2.33 | 114 | 1207 | 1015 | 2.42 | 117 |

hand, when restricted to CSQ, the overhead of an affine quantizer becomes greatly reduced thanks to our method of using constant zero-point, though it still has a significant overhead compared with our proposed BNN hardware-based implementation. These results show that CSQ is as efficient as CLQ and much more efficient than relaxed affine quantizer.

# 6 Conclusion

In this paper we provided an analysis of CSQ for extreme low-bit quantization, which is both completely symmetric around zero and trainable using existing linear QAT methods. Our analyses and experimental results using state-of-the-art QAT methods with CIFAR-10 and ImageNet datasets show that a simple change of quantization levels can result in significant performance improvement for extremely low-bit quantized neural networks ($\leq$ 3-bit). We also showed that CSQ can be realized efficiently on standard multipliers and BNN hardware by our methods. Considering there are very few previous works targeting 2-bit network performance, CSQ can be a very useful tool for optimizing extreme low-precision neural networks for deployment.

# 7 Acknowledgement

This work was supported by the Samsung Advanced Institute of Technology, Samsung Electronics Co., Ltd., by IITP grant (No. 2020-0-01336, Artificial Intelligence Graduate School Program (UNIST)) and NRF grant (No. 2020R1A2C2015066) funded by MSIT of Korea, and by Free Innovative Research Fund of UNIST (1.170067.01).

# References

[1] Ankur Agrawal, Sae Kyu Lee, Joel Silberman, Matthew Ziegler, Mingu Kang, Swagath Venkataramani, Nianzheng Cao, Bruce Fleischer, Michael Guillorn, Matthew Cohen, et al. A 7nm 4-core ai chip with 25.6 tflops hybrid fp8 training, 102.4 tops int4 inference and workload-aware throttling. In *2021 IEEE International Solid-State Circuits Conference (ISSCC)*, volume 64, pages 144–146. IEEE, 2021.

[2] Chaim Baskin, Evgenii Zheltonozhkii, Tal Rozen, Natan Liss, Yoav Chai, Eli Schwartz, Raja Giryes, Alexander M Bronstein, and Avi Mendelson. Nice: Noise injection and clamping estimation for neural network quantization. *Mathematics*, 9(17):2144, 2021.

[3] Yash Bhalgat, Jinwon Lee, Markus Nagel, Tijmen Blankevoort, and Nojun Kwak. Lsq+: Improving low-bit quantization through learnable offsets and better initialization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 696–697, 2020.

[4] Yoonho Boo, Sungho Shin, Jungwook Choi, and Wonyong Sung. Stochastic precision ensemble: self-knowledge distillation for quantized deep neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 6794–6802, 2021.

[5] Peng Chen, Jing Liu, Bohan Zhuang, Mingkui Tan, and Chunhua Shen. Aqd: Towards accurate quantized object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 104–113, 2021.

[6] Jooyeon Choi, Hyeonuk Sim, Sangyun Oh, Sugil Lee, and Jongeun Lee. MLogNet: A logarithmic quantization-based accelerator for depthwise separable convolution. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*. doi: 10.1109/TCAD.2022.3150249.

[7] Jungwook Choi, Pierce I-Jen Chuang, Zhuo Wang, Swagath Venkataramani, Vijayalakshmi Srinivasan, and Kailash Gopalakrishnan. Bridging the accuracy gap for 2-bit quantized neural networks (qnn). *arXiv preprint arXiv:1807.06964*, 2018.

[8] Jungwook Choi, Zhuo Wang, Swagath Venkataramani, Pierce I-Jen Chuang, Vijayalakshmi Srinivasan, and Kailash Gopalakrishnan. Pact: Parameterized clipping activation for quantized neural networks. *arXiv preprint arXiv:1805.06085*, 2018.

[9] Jungwook Choi, Swagath Venkataramani, Vijayalakshmi Srinivasan, Kailash Gopalakrishnan, Zhuo Wang, and Pierce Chuang. Accurate and efficient 2-bit quantized neural networks. In *Proceedings of the 2nd SysML Conference*, 2019.

[10] Steven K Esser, Jeffrey L McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S Modha. Learned step size quantization. In *International Conference on Learning Representations*, 2019.

[11] Ruihao Gong, Xianglong Liu, Shenghu Jiang, Tianxiang Li, Peng Hu, Jiazhen Lin, Fengwei Yu, and Junjie Yan. Differentiable soft quantization: Bridging full-precision and low-bit neural networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4852–4861, 2019.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[13] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[14] Sangil Jung, Changyong Son, Seohyung Lee, Jinwoo Son, Jae-Joon Han, Youngjun Kwak, Sung Ju Hwang, and Changkyu Choi. Learning to quantize deep networks by optimizing quantization intervals with task loss. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4350–4359, 2019.

[15] Raghuraman Krishnamoorthi. Quantizing deep convolutional networks for efficient inference: A whitepaper. *arXiv preprint arXiv:1806.08342*, 2018.

[16] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[17] Junghyup Lee, Dohyung Kim, and Bumsub Ham. Network quantization with element-wise gradient scaling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6448–6457, 2021.

[18] Sugil Lee, Hyeonuk Sim, Jooyeon Choi, and Jongeun Lee. Successive log quantization for cost-efficient neural networks using stochastic computing. In *Proceedings of the 56th Annual ACM/IEEE Design Automation Conference (DAC)*, pages 7:1–7:6, June 2019.

[19] Yuhang Li, Xin Dong, and Wei Wang. Additive powers-of-two quantization: An efficient non-uniform discretization for neural networks. In *International Conference on Learning Representations*, 2019.

[20] Yuhang Li, Ruihao Gong, Xu Tan, Yang Yang, Peng Hu, Qi Zhang, Fengwei Yu, Wei Wang, and Shi Gu. Brecq: Pushing the limit of post-training quantization by block reconstruction. *arXiv preprint arXiv:2102.05426*, 2021.

[21] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

[22] Joel Max. Quantizing for minimum distortion. *IRE Transactions on Information Theory*, 6(1):7–12, 1960.

[23] Sangyun Oh, Hyeonuk Sim, Sugil Lee, and Jongeun Lee. Automated log-scale quantization for low-cost deep neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 742–751, 2021.

[24] Sangyun Oh, Hyeonuk Sim, Jounghyun Kim, and Jongeun Lee. Non-uniform step size quantization for accurate post-training quantization. In *Proceedings of the 17th European Conference on Computer Vision (ECCV)*. Springer International Publishing, 2022.

[25] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet. *International journal of computer vision*, 115(3):211–252, 2015.

[26] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[27] Yaman Umuroglu, Nicholas J Fraser, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees Vissers. Finn: A framework for fast, scalable binarized neural network inference. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 65–74, 2017.

[28] Dongqing Zhang, Jiaolong Yang, Dongqiangzi Ye, and Gang Hua. Lq-nets: Learned quantization for highly accurate and compact deep neural networks. In *Proceedings of the European conference on computer vision (ECCV)*, pages 365–382, 2018.

[29] Xiandong Zhao, Ying Wang, Xuyi Cai, Cheng Liu, and Lei Zhang. Linear symmetric quantization of neural networks for low-precision integer hardware. In *International Conference on Learning Representations*, 2019.

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