Squeezing Accumulators in Binary Neural Networks for Extremely Resource-Constrained Applications

Azat Azamat
Department of CSE, UNIST
Ulsan, South Korea
azatkariuly@unist.ac.kr

Jaewoo Park
Department of Physics, UNIST
Ulsan, South Korea
hecate64@unist.ac.kr

Jongeun Lee*
Department of EE, UNIST
Ulsan, South Korea
jlee@unist.ac.kr

ABSTRACT

The cost and power consumption of BNN (Binarized Neural Network) hardware is dominated by additions. In particular, accumulators account for a large fraction of hardware overhead, which could be effectively reduced by using reduced-width accumulators. However, it is not straightforward to find the optimal accumulator width due to the complex interplay between width, scale, and the effect of training. In this paper we present algorithmic and hardware-level methods to find the optimal accumulator size for BNN hardware with minimal impact on the quality of result. First, we present *partial sum scaling*, a top-down approach to minimize the BNN accumulator size based on advanced quantization techniques. We also present an efficient, zero-overhead hardware design for partial sum scaling. Second, we evaluate a bottom-up approach that is to use saturating accumulator, which is more robust against overflows. Our experimental results using CIFAR-10 dataset demonstrate that our partial sum scaling along with our optimized accumulator architecture can reduce the area and power consumption of datapath by 15.50% and 27.03%, respectively, with little impact on inference performance (less than 2%), compared to using 16-bit accumulator.

KEYWORDS

Neural network accelerator, quantization, binarized neural network, adder tree, accumulator, saturating arithmetic

ACM Reference Format:
Azat Azamat, Jaewoo Park, and Jongeun Lee. 2022. Squeezing Accumulators in Binary Neural Networks for Extremely Resource-Constrained Applications. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD '22), October 30-November 3, 2022, San Diego, CA, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3508352.3549418

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

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. 1711080972, 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).

ICCAD '22, October 30-November 3, 2022, San Diego, CA, USA

ACM ISBN 978-1-4503-9217-4/22/10…$15.00
https://doi.org/10.1145/3508352.3549418

1 INTRODUCTION

Binarized neural networks (BNNs) [3] are arguably the most hardware-friendly class of neural networks, and are a good match for extremely low-power and low-cost applications (e.g., inference on IoT devices). BNN replaces multiplications with simple XNOR operations [3], and some even adopt a streaming architecture (e.g., [17]), eliminating almost all offchip data transfers. These architectural choices shift the source of hardware overhead to addition operations. In particular, accumulators account for a major fraction of hardware overhead in terms of cost and power consumption (see Section 2.2), which could be effectively reduced by using reduced-width accumulators. However, reducing accumulator width increases the likelihood of overflows, which are detrimental to BNN performance.

To avoid overflows, one can add a scaling operation before accumulation, which creates a complex optimization problem to determine the best combination of scale factor and accumulator width that can maximize the quality of result while minimizing hardware cost. We target the layer-wise architecture as opposed to the streaming architecture, which means that the accumulator size is the same across all layers. However, a network may have many scale factors depending on granularity (e.g., per-layer or per-channel). Further if we consider network training and/or hardware features such as saturating accumulator (which may be less sensitive to overflows), the problem becomes even more complicated.

In this paper we first present a general methodology, called *Partial Sum Scaling (PSS)*, to optimize the precision as well as the network weight and scale parameters. However, since the general solution uses floating-point or fixed-point multipliers, which are expensive and may defeat the purpose of reducing the accumulator size, we also propose a simple yet effective version (*HW-friendly PSS*). The simplification is based on the observation that the scale factor is most heavily influenced by the architectural parameter called tile size, which must be fixed across all layers. Thus by using a constant scale factor, and further limiting it to a power of two, we can eliminate nearly all hardware overhead of the general version.

Our experimental results using BinaryNet [3] on CIFAR-10 and Bi-Real Net [10] on ImageNet demonstrate that our HW-friendly version can generate architectures that achieve similar performance as the general version, but with essentially zero hardware overhead, and that our HW-friendly version can reduce the accumulator precision to 7-bit with little impact on inference performance (<1%), which corresponds to about 21.48% reduction in power consumption in the BNN datapath compared with using 16-bit accumulators. Finally, combining our method with a different kind of accumulator, i.e., saturating accumulator, shows acceptable network accuracy within 2% from the baseline at extremely low width of 4-bit.

In this paper we make the following contributions:

- We present a general methodology to find minimal accumulator size based on existing quantization frameworks.
- We propose a hardware-friendly version, which relies on bit-selection only, thus requiring no additional logic, as well as a method to find optimal parameters such as partial sum precision.
- We propose a novel accumulator architecture for BNNs, which consists of either a modulo or saturating accumulator, zero-overhead rounding, and bit-selection circuitry, which, when used with our partial sum scaling, can reduce accumulator precision quite significantly.

2 BACKGROUND AND RELATED WORK

2.1 DNN Accelerator Architecture

The main computation of a BNN accelerator (e.g., FINN [17]) is performed by XNOR-popcount circuit. Architecturally the XNOR-popcount circuit is very similar to an array of multipliers followed by an adder tree commonly found in (multi-bit) neural network accelerators. The output of XNOR-popcount is called *partial sum* (psum), as illustrated in Figure 1. Partial sums are accumulated either temporally or spatially to produce the final sum, which is the result of a tensor operation such as matrix-vector multiplication (MVM) and convolution. Note that accumulator is necessary to handle a tensor operation of an *arbitrary size* on *fixed-sized* hardware. To increase parallelism, multiple instances of XNOR-popcount circuit can be employed, which may generate multiple partial sums, exploiting the output channel parallelism.

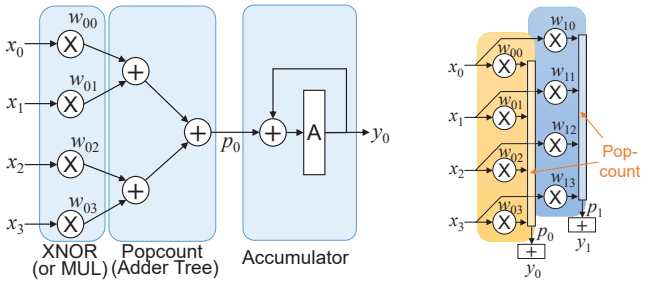
2.2 Is Accumulator Size Worth Optimizing?

Figure 2 shows the area and power breakdown of BNN datapath, which consist of XNOR-popcount logic and accumulators as illustrated in Figure 1. We have varied the tile size (i.e., the number of input elements that can be handled simultaneously by the hardware), which affects the number of XNORs and the size/height of the adder tree for the popcount function. Accumulator size is set to 16-bit. The results in Figure 2 show that the area and power consumption due to accumulators can be quite significant, at about 20% area and 40% power with the tile size of 64. While accumulator's significance decreases as tile size increases, in applications with extremely constrained resources (e.g., IoT devices), the tile size would be small and being able to reduce some of the accumulator power consumption would be very desirable.

2.3 Related Work

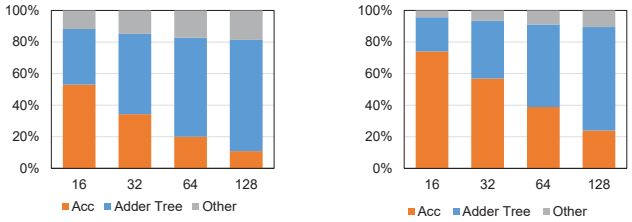
Accumulator size reduction has not received much attention in multi-bit NPUs (Neural Processing Units), where the contribution of accumulators to area and power consumption is very limited. As a result, NPUs are often generous with accumulator sizing. For instance, VTA [11], which is a configurable soft-core NPU for FPGAs, allows power-of-two accumulator sizes only (8-bit, 16-bit, 32-bit, etc.).

FINN [17] is a well-known BNN generation framework, which is based on the streaming architecture (i.e., all layers are implemented with dedicated hardware), with very limited scalability



(a) Single output (b) Multi-output

Figure 1: MAC array of a (binary) NPU architecture.



(a) Area breakdown (b) Power breakdown

Figure 2: Breakdown of BNN MAC arrays (x-axis: tile size).

for large networks such as those used in our experiments. FINN used 32-bit accumulators in earlier versions, but the most recent version includes accumulator size optimization, which determines the worst-case accumulator size *for each layer* computed from layer hyperparameters. Our work targets the layer-wise architecture, which is scalable and has a single datapath shared across all layers. There has been much interest in BNNs from the hardware design community [7, 13], but there is very little work on systematic exploration or optimization of accumulator size.

Recent DNN quantization works focus on training methods [5, 15] or novel quantizer functions [8], or both [9, 14], but they rarely consider hardware evaluation or hardware details such as accumulator. AQD [2] presents a method to get rid of all floating-point operations, but does not attempt to reduce accumulator precision below 32-bit. Some other work [4, 18] shows that their approach with 16-bit accumulators has no more than 1% degradation on CIFAR-10 and ImageNet image classification tasks. Authors in [16] use 16-bit *floating-point* accumulators for both training and inference. Authors in [12] have proposed to add an additional activation layer after each convolution/fully-connected layer to prevent overflows in low-precision accumulator. They achieve comparable results with the baseline while using only 8-bit accumulator. However, they do not optimize hardware architecture nor target BNNs. Finally, our proposed method has some similarity with our previous work [1] in that both rely on neural network quantization, but the latter targets analog ReRAM crossbar arrays with different constraints.

3 OUR PROPOSED METHODOLOGY

3.1 Overall Flow

Figure 3 illustrates our optimization methodology. First we construct an back-annotated computation graph (BCG), which is based

on a neural network graph but is a concrete computation graph with hardware architecture details added, such as tile size and partial sums. Second we define a quantizer function and add it to the BCG. The quantizer function is evaluated in terms of hardware implementation cost as well as expected network performance (e.g., classification accuracy), which is the most time-consuming step. This process is repeated varying the precision and quantizer function until desired performance and efficiency results are achieved.

3.2 Back-annotated Computation Graph

A back-annotated computation graph (BCG) is a neural network graph (or a computation graph) reconstructed from a hardware design of a neural network. A BCD essentially performs the same computation as the neural network from which it is derived, but there can be many differences such as in the order and precision of computation.

One such difference is partial sum. Since XNOR-popcount circuit (or a multiplier-and-adder array) is limited in size, it cannot perform an arbitrarily large matrix multiplication directly. As an example, consider the output of a fully-connected layer, which is computed as the dot product between two N-dimensional vectors, weight w and input x.

$$y = \sum_{i=1}^{N} w_i \cdot x_i = 2 \sum_{i=1}^{N} \text{popcount}(\text{XNOR}(w_i, x_i)) - N \qquad (1)$$

If w_i and x_i take $\{-1, 1\}$ binary values, the multiply-addition operation on the left can be replaced with the XNOR-popcount operation on the right.

Now given the hardware tile size of T (i.e., hardware performs T XNORs and T-bit popcount), we can rewrite the above as follows:

$$y = 2 \sum_{j=1}^{\lceil N/T \rceil} \underbrace{\sum_{i=1}^{T} \text{popcount}(\text{XNOR}(w_i, x_i))}_{psum} - N \quad = 2 \sum_{j=1}^{\lceil N/T \rceil} p_j - N$$

The inner summation for the partial sum is done by XNOR-popcount circuit as illustrated in Figure 1, but partial sum values should be added using an accumulator to produce the final output. Thus after explicitly identifying accumulators in the input neural network graph, we can proceed to reduce their precision by quantizing partials sums before accumulation.

3.3 Quantizer Function

Since the underlying neural network is already binarized, the input to the quantizer function, partial sum, is an integer with a finite range. That is, $0 \le p \le T$, where p is a partial sum and T is the tile size.

An example quantizer is given below:

$$\overline{p} = \text{clip}(\left\lfloor \frac{p}{\Delta} \right\rceil; 0, 2^a - 1) \qquad (2)$$

$$\hat{p} = \overline{p} \cdot \Delta \qquad (3)$$

where $\text{clip}(x; a, b) = \min(\max(x, a), b)$, $\lfloor \cdot \rceil$ is the round operation, Δ is a quantizer parameter called *scale factor*, and a is the accumulator's precision.

Note that $\overline{p}$ is the integer value that is given to the accumulator as input. Therefore only integer accumulators are required, and

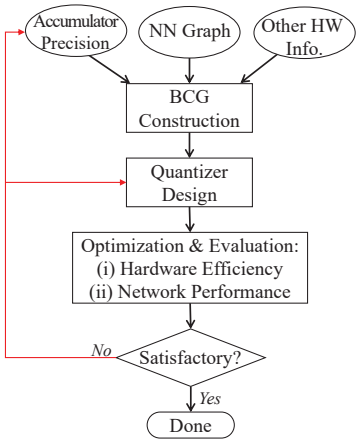


Figure 3: Our optimization methodology.

the scaling-back operation, (3), is performed after accumulation. However the real-valued output, $\hat{p}$, is used to capture the full effect of quantization. This version of quantizer (i.e., $\hat{p}$), called *simulated quantizer*, makes it easy to compute gradient and commonly used in DNN quantization.

Our partial sum quantization needs to implement four operations: scaling (division by Δ), round, clip, and scaling-back (multiplication with Δ), which requires additional logic such as floating-point multipliers, incurring hardware overhead. We present a simpler version that eliminates such overhead in Section 4.

3.4 Network Performance Evaluation

After deciding the quantizer function, along with the range of the quantization parameter(s), network performance evaluation can begin. This step is similar to DNN quantization and can be done using existing methods (e.g., [5]). As in DNN quantization, it can be done as post-training quantization (PTQ) or quantization-aware training (QAT). This step takes long because it involves optimization of free variables, such as scale factor Δ, and weight in the case of QAT, to obtain the best possible network performance.

4 HW-FRIENDLY PARTIAL SUM SCALING

4.1 Simplification

Since our general methodology in Figure 3 requires floating-point or fixed-point multipliers to implement scale factor(s), we propose a simpler version that restricts the range of the scale factor(s). First we restrict the scale factor to a power of two, which allows for replacing multiplications with bit-shift operations. Second, we propose to use the same scale factor for all layers. While there are a number of factors determining the optimal scale factor value, one of the most important is the tile size, which is fixed in any given hardware. By hard-coding the scale factor, we can eliminate even the bit-shift operation, requiring bit-selection only, which does not require any logic gates.

Let a and b denote accumulator precision and effective psum precision, respectively. We call b *effective* psum precision, because technically the psum precision should be equal to the accumulator precision (see Figure 4). However, since the effect of a power-of-two scale factor is to essentially shed off a few, say c, bits from

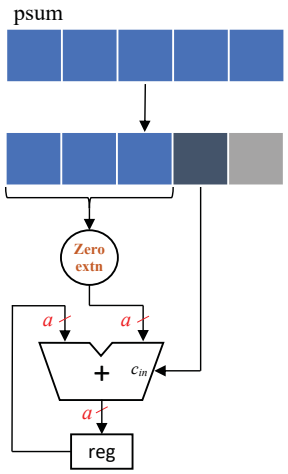


Figure 4: Proposed scaling accumulator with round operation.

a psum value, $b = a - c$ can be regarded as the effective psum precision. The question of how to determine the optimal value of c (or b) is considered in the next section. Once b is determined, the corresponding scale factor can be computed easily using the following formula:

$$\Delta = \frac{T}{2^b - 1} \approx \frac{T}{2^b} \tag{4}$$

Note that tile size (T) is usually chosen as a power of two (e.g., 32, 64 or 128), making the scale factor Δ above also a power of two, thus requiring no multiplier.

4.2 Finding Optimal Psum Precision

In our original partial sum scaling methodology, the scale factor Δ is the primary parameter and can be easily optimized, e.g., via gradient descent. Now that b (effective precision of psum) is the primary parameter and Δ is computed from b, it makes our optimization problem more difficult because b is a discrete variable. Although one could determine the value of b via gradient descent, doing so would require discretization and gradient estimate of b, which is at best approximate and cannot guarantee optimality. Instead, we employ an exhaustive search strategy varying a and b in all possible combinations, which generates only a few dozens of combinations while certainly being able to find the optimal precisions.

Another advantage of using an exhaustive search is that it can be used in the PTQ setting. For QAT, we first determine the psum precision (and consequently scale factor) using PTQ via exhaustive search, and then apply QAT with the psum precision fixed to the value found in PTQ. The time complexity of the exhaustive search for PTQ is quadratic to the number of precision values considered (see, for instance, Table 1, of which each entry takes one simple inference to generate).

4.3 Efficient Hardware Implementation

Zero-Overhead Round Operation. The round operation in quantization equation (as in (2)) usually requires an adder that will increase the overhead of the hardware. However, it can be avoided by simply truncating the input and feed the most significant bit of the truncated bits as the carry in (cin) for the accumulator. Figure 4 shows an example with 5-bit psum and scale parameter of 4, which means truncating the last two bits, one of which is fed to the accumulator. This way we can implement round operation without using extra adder.[1]

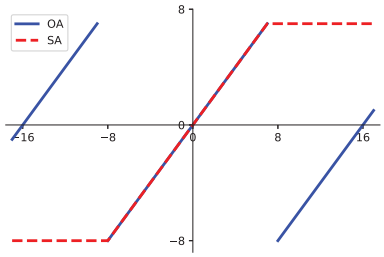


Figure 5: Actual result (y-axis) vs. ideal result (x-axis) of ordinary (OA) and saturating adder (SA). 4-bit example.

Scaling Operation. The scale factor is a power of two and the same across all layers. Therefore both scaling and scaling-back operations can be implemented without consuming any logic gates if we target one specific network, as is the case with FPGA accelerators (e.g., FINN [17]). Otherwise, we need a bit-shifter if we want to support different network models with possibly different optimal psum precision values. In either case, scaling-back operation can be merged with the following layer's input quantization, and requires no extra hardware.

Clipping Operation. In (2) we need a clipping operation to ensure that we do not use more bits than we are allowed (note the scale factor can take any value during training). During inference, we do not need clipping operation if the expression inside the clipping is known to be within accumulator precision's range. In the partial sum scaling, the scale factor is dictated by psum precision, and is always greater than 1. Thus the clipping operation can be safely removed.

4.4 Example: Saturating Adder

One example to demonstrate the usefulness of our proposed methodology is the evaluation of saturating adder (SA). Saturating adders are less sensitive to overflow than ordinary adders, which may help achieve more robust performance across many precision settings or networks. Figure 5 compares 4-bit ordinary and saturating adders.

Ordinary adder is a modulo adder defined as $y \leftarrow (y+p) \bmod 2^a$, or $y = (\sum_i p_i) \bmod 2^a$. Note that modulo and addition are associative, therefore we can change the order of operations freely, resulting in the more easier-to-compute formula. Saturating adder is defined as $y \leftarrow \text{clip}(y + p, 0, 2^a - 1)$. However, it can *not* be simplified to $\text{clip}(\sum_i p_i, 0, 2^a - 1)$, because clip and addition are not associative. This means that in order to evaluate the effect of using saturating adders accurately, we must use a more hardware-accurate framework such as Figure 3 that models tile size and partial sum explicitly.

[1]According to our synthesis result, our zero-overhead rounding circuit using the carry-in input for the cheap "add 1" functionality has a very small overhead of one half-adder, compared with not having such rounding feature. Though not completely zero, the overhead is very small nonetheless.

Table 1: PTQ with Binary ResNet-18 on CIFAR-10 (P: partial sum, A: accumulator)

P \ A	9-bit	8-bit	7-bit	6-bit	5-bit	4-bit	3-bit	2-bit
9-bit	9.92	-	-	-	-	-	-	-
8-bit	10.29	10.23	-	-	-	-	-	-
7-bit	50.26	10.33	10.40	-	-	-	-	-
6-bit	90.04	49.79	9.87	10.41	-	-	-	-
5-bit	**90.11**	89.62	55.54	10.76	9.98	-	-	-
4-bit	89.78	**89.78**	**89.56**	59.66	11.09	10.20	-	-
3-bit	86.86	85.86	85.86	**85.76**	**58.97**	**11.15**	9.93	-
2-bit	9.92	9.92	9.92	9.92	9.92	10.05	**10.20**	**10.01**
General	90.47	90.12	89.81	88.34	65.25	11.79	10.90	10.56

Table 2: PTQ with Bi-Real Net 18 on ImageNet (P: partial sum, A: accumulator)

P \ A	9-bit	8-bit	7-bit	6-bit	5-bit	4-bit	3-bit	2-bit
9-bit	0.09	-	-	-	-	-	-	-
8-bit	0.14	0.11	-	-	-	-	-	-
7-bit	0.48	0.14	0.09	-	-	-	-	-
6-bit	48.42	0.50	0.14	0.10	-	-	-	-
5-bit	**54.93**	50.55	0.82	0.14	0.09	-	-	-
4-bit	53.25	**53.25**	**48.80**	0.84	0.14	0.10	-	-
3-bit	38.40	38.40	38.78	**36.45**	**0.80**	**0.14**	0.08	-
2-bit	0.16	0.16	0.15	0.15	0.17	0.10	**0.10**	**0.09**

5 EXPERIMENTS

5.1 Experimental Setup

To evaluate the effectiveness of our proposed method, we use two BNNs, BinaryNet on CIFAR-10 and Bi-Real Net 18 on ImageNet. The CIFAR-10 BinaryNet [3] is a binarized version of ResNet-18 [6] and widely used in the literature. The Bi-Real Net [10] is a recent, state-of-the-art BNN model, showing very competitive accuracy on ImageNet. The network structure and learning conditions of the BinaryNet and Bi-Real Net are the same as in the original papers [3] and [10], respectively. We fix the tile size as $T = 64$ in all our experiments. We do not apply PSS to the first and last layer in a network. We compare the proposed method with a baseline design that uses 16-bit accumulators, as NPUs often support power-of-two accumulator sizes only (e.g., VTA soft-core NPU for FPGAs [11]). For hardware evaluation we design a BNN datapath in Verilog, which is synthesized with Synopsys Design Compiler using Samsung 65 nm technology.

5.2 Impact of Overflow on Network Accuracy

Table 3a and Table 4a show the inference performance of CIFAR-10 BinaryNet (based on ResNet-18) and ImageNet Bi-Real Net 18 for different accumulator precision without applying our partial sum scaling. For both networks, accuracy drops significantly when accumulator is 9-bit or lower. Even after retraining, 9-bit accumulator's performance is not able to come within 1% degradation from the baseline accuracy.

5.3 PTQ Result & Finding the Best Psum Precision

As discussed in Section 4.2, we perform an exhaustive search for all combinations of psum/accumulator precision in the post-training quantization (PTQ) setting. Note that psum precision cannot exceed accumulator precision. Scale factor is computed by (4) with b as the partial sum precision. Results for CIFAR-10 and ImageNet are summarized in Table 1 and Table 2, respectively. The highest network performance determines the optimal combination of psum-accumulator. For instance, 9-bit accumulator works best with 5-bit psum, etc.

Table 3: Binary ResNet-18 on CIFAR-10 with different accumulator precision (baseline is 90.72%)

(a) w/o psum scaler

Accumulator	Inference	Retraining
32-bit	90.72	–
11-bit	90.72	–
10-bit	**90.51**	**90.68**
9-bit	49.16	89.34
8-bit	9.91	47.52
7-bit	9.82	11.18

(b) w/ psum scaler

Accumulator	psum	OA	SA
9-bit	5-bit	90.59	90.66
8-bit	4-bit	90.52	90.58
7-bit	4-bit	**90.33**	**90.17**
6-bit	3-bit	88.72	89.04
5-bit	3-bit	88.67	88.90
4-bit	3-bit	81.81	88.84
3-bit	2-bit	21.34	88.37
2-bit	2-bit	10.56	80.43

Table 4: Bi-Real Net 18 on ImageNet with different accumulator precision (baseline is 56.39%)

(a) w/o psum scaler

Accumulator	Inference	Retraining
32-bit	56.39	–
11-bit	**56.39**	–
10-bit	51.72	**55.95**
9-bit	0.56	52.01
8-bit	0.16	34.38
7-bit	0.10	12.44

(b) w/ psum scaler

Accumulator	psum	OA	SA
7-bit	4-bit	**55.32**	**55.68**
6-bit	3-bit	52.59	53.51
5-bit	3-bit	49.99	53.04
4-bit	3-bit	38.01	52.95
3-bit	2-bit	25.47	48.73

To see how close our HW-friendly version's network performance is, compared to our general version's where each layer has its own floating-point scale factor, we perform a PTQ experiment using the general version on the CIFAR-10 network, which finds optimal scale factors via gradient descent (weights are frozen, but only scale factors are trained using [5]). Figure 6 shows the result, where the y-axis shows Δ as computed by (4) (HW-friendly PSS) or found by gradient descent (general PSS). In the graph, the blue dots representing floating-point scale factors for all 16 layers (except the first and last layers) are so close to each other that they appear as one dot, which confirms that there are very little differences among layers. Also the floating-point scale factors found by gradient descent are very similar to the ones found by exhaustive search, and in fact, if we limit the floating-point scale factors to power-of-two values only, they all coincide with integer values. In terms of the network performance, the differences are very small

as shown in the last row of Table 1, with the differences being due to the floating-point vs. power-of-two value differences.

5.4 QAT Result

After determining the optimal values of b, effective partial sum precision, we retrain our networks using the QAT method similar to [5]. Results are summarized in Table 3b and Table 4b. For BinaryNet on CIFAR-10, 7-bit accumulator with both ordinary and saturating adders shows comparable results within 0.5% degradation from the baseline. It is clear that ordinary adder cannot work well on extremely-low accumulators (i.e., 3-bit and 2-bit). However, saturating adder is more effective in this case and can improve the performance significantly. For instance, 3-bit saturating adder can achieve 88.37%, whereas 3-bit ordinary adder is only 21.34%. On the other hand, saturating adder has additional overhead on hardware due to an extra clipping operation (see synthesis result in Table 5).

The same trend can be observed for the ImageNet result where Bi-Real Net 18 is able to achieve near baseline result at 7-bit accumulator with either ordinary or saturating adders.

5.5 Hardware Synthesis Results

To evaluate the hardware efficiency of the architectures generated using our proposed method, we have designed 64×64 array binary datapaths (having 64-input and 64-output, thus 4,096 XNORs and 64 adder trees, along with accumulators), which are then synthesized to obtain area and power estimates. Results are summarized in Table 5. The baseline is the 16-bit OA case.

Our architecture with 7-bit OA with PSS (partial sum scaling) consumes 21.48% less power and 11.22% less area in the datapath than the baseline. Moreover, the architecture with SA and PSS is so robust that in the case of CIFAR-10, accumulator precision can be squeezed down to 4-bit, reducing the power consumption by 27.03% and area by 15.50% with an acceptable network accuracy loss of less than 2% compared to the 16-bit baseline.

5.6 Accuray-Efficiency Tradeoff

Figure 7 shows the accuracy-area tradeoff and accuracy-power tradeoff of three different methods: OA without PSS, OA with PSS, and SA with PSS. These results are for the Binary ResNet-18 on CIFAR-10. While 7-bit OA with PSS has the highest accuracy with no hardware overhead, the graphs also show that on the wider range of accumulator precision, SA with PSS makes the best tradeoff among the three methods, followed by OS with PSS, which confirms the superiority of our proposed method.

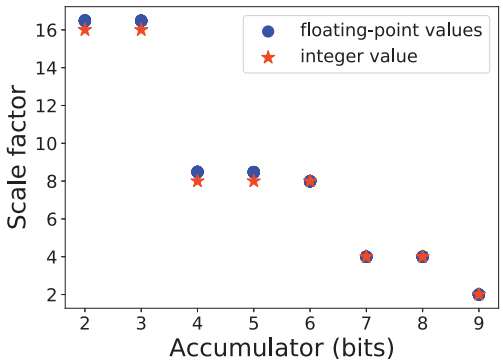
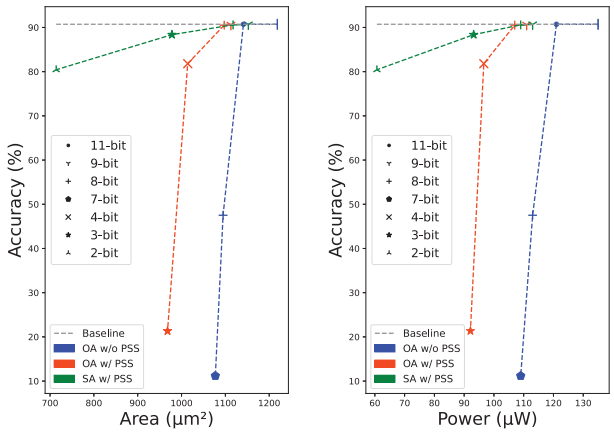


Figure 6: Float-point and integer scale factor values for different accumulator precisions.

Table 5: Synthesis result (64×64 array)

Case	Area (μm^2)	Power (μW)
16-bit OA	1218.60	135
10-bit OA	1125.72 (−7.62%)	118 (−12.59%)
7-bit OA + PSS	1081.80 (−11.22%)	106 (−21.48%)
5-bit OA + PSS	1048.68 (−13.94%)	101 (−25.18%)
7-bit SA + PSS	1099.80 (−9.74%)	108 (−20.00%)
4-bit SA + PSS	1029.60 (−15.50%)	98.5 (−27.03%)



(a) Accuracy vs. area **(b) Accuracy vs. power**

Figure 7: Accuracy vs. area and power for CIFAR-10 Binary ResNet-18.

6 CONCLUSION

BNNs fit well with applications with extremely constrained resources, but they still have relatively high accumulator overhead, especially when the tile size is on the small side. To provide the most optimized architecture, we have proposed a general methodology to squeeze even accumulators, PSS, which can help navigate the complex optimization process involving accumulator precision, psum precision, and scale factors. Our hardware-friendly version of PSS eliminates nearly all hardware overhead of the general PSS while achieving very similar performance. The exploration results demonstrate that our methodology can reduce the accumulator width to 7-bit with little degradation compared to 16-bit accumulator. Moreover, combining our method with a different kind of accumulator, i.e., saturating accumulator, shows acceptable network accuracy within 2% from the baseline at extremely low width of 4-bit. In this paper we assume the layer-wise architecture, but the streaming architecture presents an even more interesting optimization opportunity to use different accumulator precisions for different layers, which is left for future work.

REFERENCES

[1] Azat Azamat, Faaiz Asim, and Jongeun Lee. 2021. Quarry: Quantization-based ADC Reduction for ReRAM-based Deep Neural Network Accelerators. In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 1–7.

[2] Peng Chen, Jing Liu, Bohan Zhuang, Mingkui Tan, and Chunhua Shen. 2021. AQD: Towards Accurate Quantized Object Detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 104–113.

[3] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2016. Binarized Neural Networks: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1. *arXiv preprint arXiv:1602.02830* (2016).

[4] Barry de Bruin, Zoran Zivkovic, and Henk Corporaal. 2020. Quantization of Deep Neural Networks for Accumulator-constrained Processors. *Microprocessors and Microsystems* 72 (2020), 102872.

[5] Steven K Esser, Jeffrey L McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S Modha. 2019. Learned Step Size Quantization. *arXiv preprint arXiv:1902.08153* (2019).

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 770–778.

[7] Chun-Hsian Huang. 2021. An FPGA-based Hardware/Software Design Using Binarized Neural Networks for Agricultural Applications: A Case Study. *IEEE Access* 9 (2021), 26523–26531.

[8] Sugil Lee, Hyeonuk Sim, Jooyeon Choi, and Jongeun Lee. 2019. Successive Log Quantization for Cost-Efficient Neural Networks Using Stochastic Computing. In *2019 56th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[9] Yuhang Li, Xin Dong, and Wei Wang. 2019. Additive Powers-of-Two Quantization: An Efficient Non-uniform Discretization for Neural Networks. *arXiv preprint arXiv:1909.13144* (2019).

[10] Zechun Liu, Baoyuan Wu, Wenhan Luo, Xin Yang, Wei Liu, and Kwang-Ting Cheng. 2018. Bi-Real Net: Enhancing the Performance of 1-bit CNNs with Improved Representational Capability and Advanced Training Algorithm. In *Proceedings of the European conference on computer vision (ECCV)*. 722–737.

[11] Thierry Moreau, Tianqi Chen, Luis Vega, Jared Roesch, Eddie Yan, Lianmin Zheng, Josh Fromm, Ziheng Jiang, Luis Ceze, Carlos Guestrin, et al. 2019. A Hardware–Software Blueprint for Flexible Deep Learning Specialization. *IEEE Micro* 39, 5 (2019), 8–16.

[12] Renkun Ni, Hong-min Chu, Oscar Castañeda Fernández, Ping-yeh Chiang, Christoph Studer, and Tom Goldstein. 2021. WrapNet: Neural Net Inference with Ultra-Low-Precision Arithmetic. In *International Conference on Learning Representations ICLR 2021*. OpenReview.

[13] Eriko Nurvitadhi, David Sheffield, Jaewoong Sim, Asit Mishra, Ganesh Venkatesh, and Debbie Marr. 2016. Accelerating Binarized Neural Networks: Comparison of FPGA, CPU, GPU, and ASIC. In *2016 International Conference on Field-Programmable Technology (FPT)*. IEEE, 77–84.

[14] Sangyun Oh, Hyeonuk Sim, Jounghyun Kim, and Jongeun Lee. 2022. Non-Uniform Step Size Quantization for Accurate Post-Training Quantization. In *Proceedings of the 17th European Conference on Computer Vision (ECCV)*. Springer International Publishing.

[15] Sangyun Oh, Hyeonuk Sim, Sugil Lee, and Jongeun Lee. 2021. Automated Log-Scale Quantization for Low-Cost Deep Neural Networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 742–751.

[16] Charbel Sakr, Naigang Wang, Chia-Yu Chen, Jungwook Choi, Ankur Agrawal, Naresh Shanbhag, and Kailash Gopalakrishnan. 2019. Accumulation Bit-Width Scaling For Ultra-Low Precision Training Of Deep Networks. *arXiv preprint arXiv:1901.06588* (2019).

[17] Yaman Umuroglu, Nicholas J Fraser, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees Vissers. 2017. FINN: A Framework for Fast, Scalable Binarized Neural Network Inference. In *Proceedings of the 2017 ACM/SIGDA international symposium on field-programmable gate arrays*. 65–74.

[18] Xiandong Zhao, Ying Wang, Xuyi Cai, Cheng Liu, and Lei Zhang. 2020. Linear Symmetric Quantization of Neural Networks for Low-precision Integer Hardware. In *International Conference on Learning Representations ICLR 2020*. OpenReview.