LETTER

Thread: Towards fine-grained precision reconfiguration in variable-precision neural network accelerator

Shichang Zhang^{1,2}, Ying Wang¹, Xiaoming Chen¹, Yinhe Han^{1,2a}, Yujie Wang¹, and Xiaowei Li^{1,2}

Abstract In this work, we propose a neural network accelerator that supports finer-grained precision tuning for both activations and weights. To the best of our knowledge, this is the first neural network accelerator supporting arbitrary bit widths within 8 bits for both neural weights and activations. Our accelerator combines the features of bit-parallel and bit-serial design methods so that we can accomplish high arithmetic-unit utilization and precision tuning flexibility simultaneously. According to the cycle accurate simulation and the synthesized result, the proposed accelerator achieves up to 1.77x energy-efficiency improvement over the bit-serial design of *Stripes* for the evaluated workloads.

Keywords: DNN accelerator, variable bit-precision, bit-serial, bit-parallel, fine-grained precision

Classification: Integrated circuits

1. Introduction

Deep Neural Networks (DNNs) are playing a more important role in our daily life, such as in the tasks of image recognition, speech recognition, video detection, etc. DNNs are both computation- and memory-intensive algorithms and require highly-efficient architectures for inference acceleration. While conventional graphics processing unit (GPU) and central processing unit (CPU) solutions are thought energy inefficient in dealing with DNNs, many dedicated hardware architectures based on field programmable gate arrays (FPGAs) [1, 2, 3, 4] or application specific integrated circuits (ASICs) [5, 6, 7, 8, 9] have been proposed as alternative solutions to general-purpose processors.

To relieve the pressure of computations and memory accesses of DNNs, various approaches have been proposed, such as network pruning [10, 11, 12, 13, 14, 15] compression [16, 17, 18], weight quantization [19, 20], data representation optimization [21] and so on, along with dedicated accelerators [5]. Recently, some studies indicate that in some neural network models and datasets, lowering the bit width is effective almost without loss of accuracy [22, 23]. An extreme case is the binary neural network

(BNN) in which the activations and weights are both binarized, which tremendously reduces the amount of computations and memory accesses. However, BNNs work well only for some small-scale neural networks and less complicated datasets, such as MNIST, CIFAR-10 and SVHN [23]. When BNNs face large-scale neural networks and large datasets, the fully binarized neural networks lead to significant accuracy loss [22, 23].

Based on this observation, people have proposed variable-precision accelerators [24, 25, 26, 27] that could support both low and high bit-width neural models in a single hardware structure. Such architectures could self-adapt to different problem complexities and accuracy requirements, and work with neural models with different precisions, thereby always offering on-demand network capacity and increasing system energy efficiency in different scenarios. In theory, a larger precision tuning range and a finer tuning granularity mean that the accelerator could have sufficient operating options to choose and optimize when it faces a given deep learning problem.

Fig. 1. (a) Testing accuracy (%) of CIFAR-10 dataset with VGG-7 network [28]; (b) Top-1 accuracy (%) of ResNet-18 and AlexNet models on the ImageNet dataset [28]; (c) Object detection mAP on PASCAL VOC with R-FCN + ResNet-50/ResNet-101 [29]. AX and WX denote the bit widths for the quantization of activations and weights, respectively. Full-Prec. is 32-bit full precision.

According to the neural network training results by many other researchers' studies, we can find that different neural network models on different datasets have different bit widths to achieve a good accuracy, as shown in Fig. 1. From these observations and other potential possibilities, we can conclude that different networks for different tasks

¹State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China

²University of Chinese Academy of Sciences, Beijing 100190, China

a) yinhes@ict.ac.cn

DOI: 10.1587/ele.16.20190145

Received March 12, 2019

Accepted June 17, 2019

Publicized July 1, 2019

Copied July 25, 2019

need different bit widths for both activations and weights to achieve the goal of high efficiency. The existing variable-precision accelerators [24, 25, 26, 27] cannot fit in all the possible bit widths. They have to work in a higher-precision mode like 8-bit operations to replace 5-bit, 6-bit or others, which is a waste of computing and power resources. Thus, if accelerators can have a wide precision tuning range and a finer tuning granularity, we can always select the lowest possible bit width while meeting the accuracy and resource requirements such that the performance and energy efficiency can be both maximized. Therefore, it is beneficial to design a flexible neural network accelerator supporting continuously variable bit widths for both activations and weights.

Table I shows the state-of-the-art accelerators that support variable precisions. A-P represents the activation precision and W-P represents the weight precision. *BRein* [24] simply supports binary and ternary weights. *Stripes* [25] and *UNPU* [26] support one kind of variability for only activations or weights, but the other uses a fixed precision. *Bit-Fusion* [27] supports both variabilities but its bit width must be a power of two. It is a long jump from 4 to 8 bits for *Bit-Fusion* and it is even worse from 8 to 16 bits when using 9 bits, 10 bits or so is sufficient. To better adapt to different precision requirements of neural network models, this work presents a finer-grained variable-precision DNN accelerator named *Thread*. *Thread* supports changeable operators from 1-bit to 8-bit precisions for both activations and weights. Since 8-bit precision is enough to achieve reasonable accuracy [30], we only consider 8-bit as the highest bit width in our current implementation. However, this is not a limitation of our work. It can easily be extended to 16 bits or more if needed.

Table I. Comparisons of variable bit-precision accelerators

	A-P	W-P	Strategy
<i>BRein</i> [24]	1	1, 2	-
<i>Stripes</i> [25]	1–16	16	Bit-serial
<i>UNPU</i> [26]	16	1–16	Bit-serial
<i>Bit-Fusion</i> [27]	1, 2, 4, 8, 16	1, 2, 4, 8, 16	Fusion & Decomposition
This work	1–8*	1–8*	Bit-serial & Bit-parallel

*The proposed architecture has the ability to extend to more bits if needed.

To implement the architecture with continuously changeable precisions, we design a highly efficient data storage architecture and a highly efficient mixed computation architecture including bit-parallel activation and bit-serial weight multiplying. The main contributions of this work are as follows.

(1) We design a DNN accelerator that supports variable bit widths for both activations and weights continuously from 1 bit to 8 bits which can achieve up to 1.77x energy-efficiency improvement over the bit-serial design of *Stripes* for the evaluated workloads.

(2) We design a highly-efficient data storage architecture to store the variable-precision activations, while

weights are stored by the simple column-wise bit-serial storage method.

(3) We design a highly-efficient mixed computation architecture including bit-parallel activation and bit-serial weight multiplying.

The rest of this paper is organized as follows. Section 2 reviews the related work. Section 3 presents the proposed accelerator. Section 4 discusses the evaluation results. Finally Section 5 concludes the paper.

2. Related work

There have been several DNN accelerators that support variable bit widths of activations and/or weights. The most related works, *Stripes* [25], *UNPU* [26], and *Bit-Fusion* [27] will be discussed.

Stripes uses bit-serial inner-product units to exploit the precision variability. The variable bit width is applied to the activations but the weights are represented by a fixed bit width. Thus, its computation time scales with the activation precision proportionally. The fixed bit width of weights leads to very weak adaptability to various weight precisions, which wastes lots of computing and storage resources. In addition, due to *Stripes'* weight reuse, it cannot efficiently accelerate fully connected layers which widely appear in LSTM and other DNNs. Like *Stripes*, *UNPU* also uses the bit-serial inner-product computation manner. Different from *Stripes*, *UNPU* adopts a different data reuse dataflow, activation reuse. It supports variable bit widths for only weights but not for activations. It has the same problem with *Stripes* that the fixed bit width of activations has low adaptability for various activations. They both choose bit-serial operations with different kinds of data types, activations or weights. Different from *Stripes* and *UNPU*, *Bit-Fusion* has computational components that are dynamically composable and decomposable to support precision tuning. The bit widths of activations and weights can both be variable, but it only supports power-of-2 bit widths. Thus, *Bit-Fusion* cannot support fine-tuning of the bit width, especially when the bit width is greater than 4.

The proposed accelerator *Thread* is different from prior works as it supports continuously variable bit widths for both activations and weights. *Thread* aims to maximize the flexibility of the neural network accelerator to fit in the variable bit widths of the existing or future neural networks models. It combines both spatial and temporal design methods by exploiting the features of bit-parallel and bit-serial circuit designs to accomplish high arithmetic unit utilization and precision tuning flexibility simultaneously.

3. Thread architecture

Thread is a highly flexible energy-efficient neural network accelerator. We design the architecture from the aspects of data storage and data computation to achieve the goal of high efficiency when the data types are both continuously variable.

3.1 Overall architecture

The overall architecture of *Thread* is shown in Fig. 2. It is

mainly composed of the following components: processing slices (PSes), a pooling unit, two input/output buffers (IOBUFs), a weight buffer (WBUF), an input data format converter (IDFC) and an output data format converter (ODFC). There are 32 PSes that have 2048 processing elements (PEs) in total, which means that it can process 2048 inputs per cycle. The activations from IOBUF are shared by all the PSes, while the weights from WBUF are distributed to all the PSes. Each PS can produce one computation result after some clock cycles depending on the neural network parameters. Due to the bit-serial multiplication method that exploits shifting and addition to implement the multiplication, the PS eliminates traditional multipliers and only keeps an adder tree and an accumulator instead. Moreover, the fusible adder tree (FAT) design is the key in *Thread* to implement the variable precision operations of activations besides the bit-serial multipliers for weights. We will describe the FAT design in Section 3.3. Unlike *Stripes*, we design an independent pooling module that supports the max-pooling operation. As shown in Fig. 2, an IDFC and an ODFC are introduced to deal with data encoding and decoding. The microarchitecture including data storage, data computation and data-flow, will be discussed below.

Fig. 2. Overall architecture

3.2 Variable data storage architecture

Different from *Stripes*, we choose the reuse of input features. The reuse of input features is more efficient when using low bit-width weight or processing recurrent layers and fully-connected layers [26]. Depending on the data computation manner, the data bits of each weight are stored vertically in each column of WBUF while the activations are stored horizontally in the rows of IOBUF.

Fig. 3. IOBUF architecture for variable bit-precisions

1) *Activation storage*. We design the IOBUF by using two buffer groups and both are organized as shown in Fig. 3. As shown in Fig. 3(a) we use 12 small SRAM

units to compose a whole. The reason why we design a buffer group with 12 blocks is that 12 is the least common multiple of 1, 2, 3 and 4. This means that using 12 SRAM units can satisfy arbitrary bit-width data storage from 1 bit to 4 bits without redundant padding or memory space waste by reshaping the organization of the buffer group as illustrated in Fig. 3(b)–(e). By using two buffer groups, we can make it possible to store arbitrary bit-width data from 1 bit to 8 bits.

The row width of the SRAMs in the buffer group, is selected according to the input scale. For example, if the input is 32 operands per cycle, then the row width should be 32 bits. To fit in the storage architecture, activations are rearranged. The bits in the same bit position of different original data are stored in the same SRAM unit. The number of SRAM units in one row depends on the bit width of the activations. We give a simple example to explain the data storage format. Supposing the row width of the SRAMs is 4 bits, the bit width is 3 and we are given 4 activations, $a = 101$, $b = 011$, $c = 110$, and $d = 010$. First, we select the left-most bit from a , b , c and d in sequence, and we get ‘1010’. Next, we select the next bit in the same way until the last bit (i.e., the right-most bit). Finally, we will get ‘1010,0111,1100’, and it can be stored. When we fetch the data from the buffer group, it is an inverse operation to decode the data.

2) *Weight storage*. Rather than the activation storage approach, weight storage is relatively simple. Because the weight is stored in a bit-serial manner, each weight is stored bit-serially in a column of conventional buffer. We can serial-out one bit of the weight per cycle.

3.3 Flexible processing architecture

1) *Bit-parallel Processing*. Once the variable bit-width data layout becomes ready, it needs a coherent and highly-efficient processing architecture to do the computations. We design the fusible adder (FA) which includes two adders and a fusion unit (FU) as illustrated in Fig. 4(a). When the activation bit width is not higher than 4 bits, it will be fitted in one adder. When the activation bit width is higher and between 4 bits and 8 bits, it will be fitted in two adders and be fused through the FU. The FU’s function is to add the most significant bit (MSB) and the carry-in from the least significant bit (LSB) and splice the result and LSB as shown in Eq. (1). In the equation, A is MSB, c is the carry-in from LSB, and B is LSB. There is another problem that the operands are signed 2’s complements. An 8-bit word cannot simply be divided into two 4-bit words. To solve this problem we do a simple data transformation. Supposing an 8-bit word is $N[7:0]$, we extend the word from 8 bits to 10 bits, and we get $M[9:0]$. The Eq. (2) illustrates the details about the transformation, and it is expressed in Verilog description. We can see that the Most Significant 4 Bits have a sign extension and a zero bit is inserted between the third bit and the fourth bit.

$$F(A, c, B) = \{A + c, B\} \quad (1)$$

$$M[9:0] = \{N[7], N[7:4], 1'b0, N[3:0]\} \quad (2)$$

We design a FAT as shown in Fig. 4(b) to implement variable bit-width operations in a bit-parallel way. The FU

Fig. 4. (a) Fusible adder design; (b) Fusible adder tree design

Fig. 5. (a) Bit-parallel multiply; (b) Bit-serial multiply

is placed after the adder tree. It is more efficient than inserting it in the middle because it reduces the number of fusion units to one.

2) *Bit-serial Processing*. In our design the weights are processed in a bit-serial way. The bit-serial method can be taken simply as multiplying a data bit-serially and being shifting added. Compared with the bit-parallel multiplier, the bit-serial operation needs more cycles according to the bit width of the data. The conventional multiplication is to use a bit-parallel multiplier as shown in Fig. 5(a). The bit-serial operation includes addition and shifting operations as shown in Fig. 5(b). The architecture in Fig. 5(b) does not consider the bit-serial data's sign bit. When the bit-serial data is expressed in the signed 2's complement format, the adder should subtract the other data that is not bit-serial first if the bit-serial data is negative.

3) *Combination of Bit-parallel Fusion and Bit-serial Fusion*. The multiply and accumulate unit (MAC) design is the key of computations in the accelerator. In the design, we consider both bit-parallel fusion and bit-serial fusion techniques and the combination of them makes the MAC operations support arbitrary bit widths continuously for both activations and weights. The MAC design is shown in Fig. 6 and we call it a PS. Compared with traditional MAC arrays, our design removes all the multipliers and adds two accumulators and a fusion unit innovatively. We regard the PE as a replacement for the multiplier. The function of PE is to prepare the data needed by the adder according to the activation and weight bit. When the weight bit is its sign bit, the result is the negative number of the activation if the sign bit is 1. Otherwise, the result is 0. If it is not the sign bit, the result is the activation AND the weight bit. We select the output data from accumulator 1 and accumulator 2 or accumulator 0 according to the activations' bit width. If the activations' bit width is not higher than 4, it selects the results from accumulator 1 and accumulator 2; otherwise, it selects the results from accumulator 0.

3.4 Data flow

As *Thread* adopts input feature reuse, the input feature is shared among all the PSes. However, there is a potential problem when exploiting only one kind of data reuse in

Fig. 6. PS design.

Thread. The composable FA design offer two operational modes. One processes a single word whose precision is from 5 to 8 bits, and the other processes two words at a time whose precisions are from 1 bit to 4 bits. When *Thread* processes two pairs of operators in an FA, it has to double the weight provision if we only use input feature reuse. To prevent the access bandwidth of weights increases, we also exploit weight reuse. Thanks to our double buffer groups design, we split the input feature into two halves and store them into two buffer groups when the processing needs double data. The two data sharing the same weight can be fetched simultaneously to support weight reuse.

4. Evaluation

4.1 Experimental methodology

We implement *Thread* in Verilog, and synthesized it with Synopsys Design Compiler (DC) in SMIC 55 nm technology. The proposed architecture design is simulated with Synopsys Verilog Compile Simulator (VCS), and its power is analyzed with Synopsys PrimeTime (PT) running at 400 MHz. The main parameters of *Thread* are as follows. The computation part of *Thread* architecture consists of 32 PSes which include 2048 PEs and 32 max-pooling units. The storage part consists of two 48 KB IOBUFs and one 32 KB WBUF. The baseline for comparison is the classic bit-serial design of *Stripes*. We loyally implement its bit-serial computation and storage method.

4.2 Experimental setup

Thread and *Stripes* both process the data by broadcasting 256 bits to each tile or slice, and distributing 1024 bits across the 32 tiles or slices. Both of them adopt the same data processing dataflow and have the same ability to handle different neural network models. Without loss of generality, we choose one convolutional layer to evaluate both systems because convolutional operations take most of the inference time in deep CNNs. We set a convolutional layer as the benchmark and its parameters are listed in Table II. In our experiment, the weights of both systems are processed in the bit-serial way. Here we set the bit width of the weights to 8. We change the bit width of the activations from 1 to 8. For *Stripes* that only supports one kind of variable precisions, the bit width of the activations is fixed at 8.

Table II. Parameters of the benchmark

parameters	value	parameters	value	parameters	value
Kernel	3×3	Activation-bit	1–8	Y_length	13
Stride	1	Weight-bit	8	Inputlayers	384
Pad	1	X_length	13	Outputlayers	384

Fig. 7. Energy-efficiency comparison with Stripes

4.3 Experimental results

We have implemented the core architecture of *Stripes* for comparison. The parameters and configurations of *Stripes* like the number of PEs, and the size of buffers are the same with those of *Thread* for a fair comparison. Fig. 7 shows the energy-efficiency comparison results. In the figure, *Stripes* is shown as ‘byte’ and our variable-precision is shown as ‘bit-x’. From Fig. 7 we can find that our design is more energy efficient than *Stripes* when the bit width is not higher than 4 bits. We can achieve up to 1.77x energy-efficiency improvement than *Stripes* when the bit width is 1 bit. On average from 1-bit to 4-bit precisions we can achieve 1.61x energy-efficiency improvement than *Stripes*. Even taking into account the high-precision operations that do not have the advantage over *Stripes*, we also can get an average of 1.19x energy-efficiency improvement.

Fig. 8 presents the overall power consumption and area overhead. Because the proposed architecture design can process two words when the bit width is not greater than 4 bits, we convert the power and area of two-word processing results to a corresponding single-word processing result, as we see that the area of bit-1 to bit-4 is half of the whole design. The power consumption mainly consists of the clock network, the registers, the combinational logic and the memory. Moreover, the main power consumption comes from the clock network and the memory. Compared with the ‘byte’ storage energy, it consumes less energy when the bit width is less than 5 bits due to the proposed buffer design. It reduces 52.5% energy when the bit width is 1. For the bit widths from 1 to 4, it reduces an average 28.7% energy consumption of storage. It consumes a little more memory power when the bit width is from 5 to 8 because the multi-bank buffer design is limited by the memory library. For the computing part, the bit widths in bit-x share the same computational logic, so their power consumptions are almost the same and they share the same area. When processing the bit widths from 1 to 4, our FAT design cuts down 46.8% energy consumption over the MAC design of *Stripes*.

Regarding the area overhead, we also have the advantage over *Stripes*. On average from 1-bit to 4-bit precisions *Thread* can achieve 1.7x area improvement than *Stripes*. Considering the all bit widths from 1 to 8, *Thread* achieves 1.29x area improvement on average.

Fig. 8. Detail comparison with Stripes

5. Conclusions

Neural network models could produce comparable accuracy result to that of the original floating-point models with low bit-precision representations. Leveraging this property of neural networks, we develop *Thread*, a neural network accelerator, which combines spatial and temporal design methods by exploiting the features of bit-parallel and bit-serial circuit designs to accomplish high arithmetic unit utilization and precision tuning flexibility simultaneously. The experiment result shows that *Thread* achieves a higher energy efficiency than *Stripes*, which is up to 1.77x in low bit widths. *Thread* can be used in many scenarios due its high flexibility of bit width tuning.

Acknowledgments

The work was supported in part by National Natural Science Foundation of China (NSFC) under grant No. (61834006, 61804155, 61874124), and Beijing Municipal Science & Technology Commission (Z171100000117019, Z181100008918006), Strategic Priority Research Program of the Chinese Academy of Sciences (XDPB12), the Innovative Project of ICT, CAS (5120186140), the Youth Innovation Promotion Association CAS, and the Young Elite Scientists Sponsorship Program by CAST (2018QNRC001, 2016QNRC001).

References

- [1] Y. Ma, *et al.*: “Optimizing loop operation and dataflow in FPGA acceleration of deep convolutional neural networks,” *FPGA* (2017) 45 (DOI: 10.1145/3020078.3021736).
- [2] X. Han, *et al.*: “CNN-MERP: An FPGA-based memory-efficient reconfigurable processor for forward and backward propagation of convolutional neural networks,” *ICCD* (2016) 320 (DOI: 10.1109/ICCD.2016.7753296).
- [3] C. Zhang, *et al.*: “Optimizing FPGA-based accelerator design for deep convolutional neural networks,” *FPGA* (2015) 161 (DOI: 10.1145/2684746.2689060).
- [4] H. Sharma, *et al.*: “From high-level deep neural models to FPGAs,” *MICRO* (2016) 1 (DOI: 10.1109/MICRO.2016.7783720).
- [5] S. Han, *et al.*: “EIE: Efficient inference engine on compressed deep neural network,” *ISCA* (2016) 243 (DOI: 10.1109/ISCA.2016.30).
- [6] Y. Chen, *et al.*: “Dadiannao: A machine-learning supercomputer,” *MICRO* (2014) 609 (DOI: 10.1109/MICRO.2014.58).
- [7] X. Zhou, *et al.*: “Cambricon-S: Addressing irregularity in sparse neural networks through a cooperative software/hardware approach,” *MICRO* (2018) 15 (DOI: 10.1109/MICRO.2018.00011).
- [8] C. Deng, *et al.*: “PermDNN: Efficient compressed DNN archi-

- texture with permuted diagonal matrices,” MICRO (2018) 189 (DOI: [10.1109/MICRO.2018.00024](https://doi.org/10.1109/MICRO.2018.00024)).
- [9] Z. Wang, *et al.*: “EERA-DNN: An energy-efficient reconfigurable architecture for DNNs with hybrid bit-width and logarithmic multiplier,” IEICE Electron. Express **15** (2018) 20180212 (DOI: [10.1587/elex.15.20180212](https://doi.org/10.1587/elex.15.20180212)).
 - [10] P. Molchanov, *et al.*: “Pruning convolutional neural networks for resource efficient inference,” arXiv preprint arXiv:1611.06440 (2016) <https://arxiv.org/abs/1611.06440>.
 - [11] J.-H. Luo, *et al.*: “Thinet: A filter level pruning method for deep neural network compression,” ICCV (2017) 5058 (DOI: [10.1109/ICCV.2017.541](https://doi.org/10.1109/ICCV.2017.541)).
 - [12] Y. He, *et al.*: “Channel pruning for accelerating very deep neural networks,” ICCV (2017) 1398 (DOI: [10.1109/ICCV.2017.155](https://doi.org/10.1109/ICCV.2017.155)).
 - [13] R. Yu, *et al.*: “NISP: Pruning networks using neuron importance score propagation,” CVPR (2018) 9194 (DOI: [10.1109/CVPR.2018.00958](https://doi.org/10.1109/CVPR.2018.00958)).
 - [14] S. Anwar, *et al.*: “Structured pruning of deep convolutional neural networks,” JETC **13** (2017) 32 (DOI: [10.1145/3005348](https://doi.org/10.1145/3005348)).
 - [15] H. Li, *et al.*: “Pruning filters for efficient convnets,” arXiv preprint arXiv:1608.08710 (2016) <https://arxiv.org/abs/1608.08710>.
 - [16] A. Dubey, *et al.*: “Coreset-based neural network compression,” ECCV (2018) 454.
 - [17] W. Chen, *et al.*: “Compressing neural networks with the hashing trick,” ICML (2015) 2285.
 - [18] F. Manessi, *et al.*: “Automated pruning for deep neural network compression,” ICPR (2018) 657 (DOI: [10.1109/ICPR.2018.8546129](https://doi.org/10.1109/ICPR.2018.8546129)).
 - [19] J. Wu, *et al.*: “Quantized convolutional neural networks for mobile devices,” CVPR (2016) 4820 (DOI: [10.1109/CVPR.2016.521](https://doi.org/10.1109/CVPR.2016.521)).
 - [20] D. Lin, *et al.*: “Fixed point quantization of deep convolutional networks,” ICML (2016) 2849.
 - [21] H.-J. Kang: “Short floating-point representation for convolutional neural network inference,” IEICE Electron. Express **16** (2019) 20180909 (DOI: [10.1587/elex.15.20180909](https://doi.org/10.1587/elex.15.20180909)).
 - [22] S. Zhou, *et al.*: “Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients,” arXiv preprint arXiv:1606.06160 (2016) <https://arxiv.org/abs/1606.06160>.
 - [23] I. Hubara, *et al.*: “Quantized neural networks: Training neural networks with low precision weights and activations,” J. Mach. Learn. Res. **18** (2018) 6869.
 - [24] K. Ando, *et al.*: “BRein memory: A 13-layer 4.2 K neuron/0.8 M synapse binary/ternary reconfigurable in-memory deep neural network accelerator in 65 nm CMOS,” IEEE Symp. VLSI Circuits (2017) 24 (DOI: [10.23919/VLSIC.2017.8008533](https://doi.org/10.23919/VLSIC.2017.8008533)).
 - [25] P. Judd, *et al.*: “Stripes: Bit-serial deep neural network computing,” MICRO (2016) 1 (DOI: [10.1109/MICRO.2016.7783722](https://doi.org/10.1109/MICRO.2016.7783722)).
 - [26] J. Lee, *et al.*: “UNPU: A 50.6TOPS/W unified deep neural network accelerator with 1 b-to-16 b fully-variable weight bit-precision,” ISSCC (2018) 218 (DOI: [10.1109/ISSCC.2018.8310262](https://doi.org/10.1109/ISSCC.2018.8310262)).
 - [27] H. Sharma, *et al.*: “Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural network,” ISCA (2018) 764 (DOI: [10.1109/ISCA.2018.00069](https://doi.org/10.1109/ISCA.2018.00069)).
 - [28] S. Jung, *et al.*: “Joint training of low-precision neural network with quantization interval parameters,” arXiv preprint arXiv:1808.05779 (2018) <https://arxiv.org/abs/1808.05779v1>.
 - [29] P. Yin, *et al.*: “Quantization and training of low bit-width convolutional neural networks for object detection,” arXiv preprint arXiv:1612.06052v2 (2017) <https://arxiv.org/abs/1612.06052v2>.
 - [30] S. Han, *et al.*: “Deep compression: Compressing deep neural network with pruning, trained quantization and Huffman coding,” arXiv preprint arXiv:1510.00149v5 (2016) <https://arxiv.org/abs/1510.00149v5>.